ABSTRACT

Title of Dissertation: ACTIVE VISION BASED EMBODIED-AI DESIGN FOR NANO-UAV AUTONOMY

Nitin Jagannatha Sanket
Doctor of Philosophy, 2021

Dissertation Directed by: Professor Yiannis Aloimonos
Department of Computer Science

The human fascination to mimic ultra-efficient flying beings like birds and bees has led to a rapid rise in aerial robots in the recent decade. These aerial robots now posses a market share of over 10 Billion US Dollars. The future for aerial robots or Unmanned Aerial Vehicles (UAVs) which are commonly called drones is very bright because of their utility in a myriad of applications. I envision drones delivering packages to our homes, finding survivors in collapsed buildings, pollinating flowers, inspecting bridges, performing surveillance of cities, in sports and even as pets. In particular, quadrotors have become the go to platform for aerial robotics due to simplicity in their mechanical design, their vertical takeoff and landing capabilities and agility characteristics.

Our eternal pursuit to improve drone safety and improve power efficiency has given rise to the research and development of smaller yet smarter drones. Furthermore, smaller drones are more agile and task-distributable as swarms. Embodied Artificial Intelligence (AI) has been a big fuel to push this area further. Classically, the approach to designing such nano-drones possesses a strict distinction between perception, planning and control

and relies on a 3D map of the scene that are used to plan paths that are executed by a control algorithm.

On the contrary, nature's never-ending quest to improve the efficiency of flying agents through genetic evolution led to birds developing amazing eyes and brains tailored for agile flight in complex environments as a software and hardware co-design solution. In contrast, smaller flying agents such as insects that are at the other end of the size and computation spectrum adapted an ingenious approach – to utilize movement to gather more information. Early pioneers of robotics remarked at this observation and coined the concept of "Active Perception" which proposed that one can move in an exploratory way to gather more information to compensate for lack of computation and sensing. Such a controlled movement imposes additional constraints on the data being perceived to make the perception problem simpler.

Inspired by this concept, in this thesis, I present a novel approach for algorithmic design on nano aerial robots (flying robots the size of a hummingbird) based on active perception by tightly coupling and combining perception, planning and control into sensorimotor loops *using only on-board sensing and computation*. This is done by re-imagining each aerial robot as a series of hierarchical sensorimotor loops where the higher ones require the inner ones such that resources and computation can be efficiently re-used. Activeness is presented and utilized in four different forms to enable large-scale autonomy at tight Size, Weight, Area and Power (SWAP) constraints not heard of before. The four forms of activeness are: 1. By moving the agent itself, 2. By employing an active sensor, 3. By moving a part of the agent's body, 4. By hallucinating active movements. Next, to make this work practically applicable I show how hardware and software co-design can

be performed to optimize the form of active perception to be used. Finally, I present the world's first prototype of a RoboBeeHive that shows how to integrate multiple competences centered around active vision in all it's glory. Following is a list of contributions of this thesis:

- The world's first functional prototype of a RoboBeeHive that can artificially pollinate flowers.
- The first method that allows a quadrotor to fly through gaps of unknown shape, location and size using a single monocular camera with only on-board sensing and computation.
- The first method to dodge dynamic obstacles of unknown shape, size and location on a quadrotor using a monocular event camera. Our series of shallow neural networks are trained in simulation and transfers to the real world without any fine-tuning or re-training.
- The first method to detect unmarked drones by detecting propellers. Our neural network is trained in simulation and transfers to the real world without any fine-tuning or re-training.
- A method to adaptively change the baseline of a stereo camera system for quadrotor navigation.
- The first method to introduce the usage of saliency to select features in a direct visual odometry pipeline.

- A comprehensive benchmark of software and hardware for embodied AI which would serve as a blueprint for researchers and practitioners alike.

ACTIVE VISION BASED EMBODIED-AI DESIGN FOR NANO-UAV AUTONOMY

by

Nitin Jagannatha Sanket

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Advisory Committee:
Professor Yiannis Aloimonos, Chair/Advisor
Dr. Cornelia Fermüller
Professor Davide Scaramuzza
Professor Dinesh Manocha
Professor Inderjit Chopra, Dean's Representative

Dedication

To my family.

Acknowledgments

I owe my gratitude to all the people who have made this thesis possible and because of whom my doctoral study experience has been one that I will cherish forever.

I still remember to this day I first met Prof. Yiannis Aloimonos in his room to discuss about how perception leads to cognition. I am eternally grateful to him for inspiring me and giving me an opportunity to join his lab as a doctoral student. I am also eternally grateful to Dr. Cornelia Fermüller for introducing me to the field of neuromorphic cameras and perception. In particular, my experience during the Telluride Neromorphic workshop was one of a kind that I will cherish forever. Both Yiannis and Cornelia have treated me like their own child, giving their time whenever I needed them, mentoring me and helping me hone my skills. One of the most remarkable things was the amount of freedom they gave me during my doctoral studies, they funded me to setup the aerial robotics lab without hesitating even once. They gave me freedom to mentor students and teach courses all while pursuing my research – these was the best years of my academic life.

I still remember the day, Prof. Yiannis came into the room and asked the question "What is the minimal amount of information you would need to solve a task?", further he added "You could do this to fly through gaps of unknown shape and location!". He then said "Do all this with only a single camera!". I was thinking in the paradigm of building

a map when he said “Use Active vision: Move to make your problem easier”. This was the monumental moment of my Ph.D., where I adopted the active philosophy to solve problems on extremely computation and sensor starved aerial robots – nano-quadrotors. Further down the line, Cornelia said, “Why don’t we use event cameras to solve problems, they are energy efficient and bio-inspired”. This led to tackling the problem of dynamic obstacle dodging using event cameras.

Both Yiannis and Cornelia were a powerhouse of inspiration and they motivated and nurtured my wild ideas and also helping translate the math into code and high quality publications to be made open-source. I am elated to have worked with the both of you.

I am forever indebted to Chahat Deep Singh for his help with all my papers. He has been the best friend, housemate, lab-mate, roadtrip partner and astrophotography expeditions during the last five years of my life. I would not have been able to fix all those linux issues without your help and would not have enjoyed discovering the math used in the papers without our intense discussions. I still remember getting yelled at by other professors for being loud at discussions. Chahat was one of the heros that helped setup the aerial robotics lab including helping me in sweeping the floors and getting yelled at for this. Chahat, I will forever try to maintain this friendship and collaboration.

I would also specifically like to thank Chethan Mysore Parameshwara for all his help with ROS and event cameras. Chethan, thanks for inspiring me to work on neuromorphic algorithms and cameras with you. It has been a pleasure.

I would also like to thank all my other labmates of Perception and Robotics Group (PRG, which was formerly called Computer Vision Lab or CVL) duing my past five years of Ph.D.: Chahat Deep Singh, Chethan Mysore Parameshwara, Kanishka Ganguly,

Huai-Jen Liang, Aleksandrs Ecins, Konstantinos Zampogiannis, Dr. Francisco Barranco, Levi Burner, Snehesh Shreshta, Michael Maynord, Chinmaya Devaraj, Matthew Evanusa, Xiaomin Lin, Peter Sutor, Siqin Li, Dr. Behzad Sadrfaridpour, Dr. Krishna Kidambi and Anton Mitrokhin. PRG has always been a second home to me.

I am also indebted to all the Masters students that helped me with experiments in my research: Prateek Arora, Ashwin V. Kuruttukulam, Abhinav Modi, Kartik Madhira, Miguel Maestre Trueba, Varun Asthana, Saumil Shah and Akash Guha. Without you guys, it would not have been possible to conduct these hard and amazing experiments.

I am thrilled to have worked with brilliant professors such as Prof. Davide Scaramuzza and Prof. Guido de Croon and I am indebted for the opportunity for collaboration. You have made me a better writer, a better experimentalist and an overall better researcher.

I would also like to thank the unsung heros which have contributed immensely in the completion of this thesis: University of Maryland Institute for Advanced Computer Studies staff in particular Janice Perrone for handling a multitude of order requests with patience and Tom Ventsias for handling our media outreach. Ivan Penskiy and Kimberly Edwards have been of immense help for setting up the lab, procuring materials for experiments and I would like to wholeheartedly thank them. I also thank the patient housekeeping staff for keeping our lab clean during messy experiments.

My housemates have helped in balancing workload with unlimited doses of pure unadulterated fun and dad jokes. I thank Chahat Deep Singh, Sunaina Prabhu, Vinayak Bendale, Ankita Tondwalkar, Priyal Gala, Anoorag Sunkari, Prateek Arora, Harshvardhan Uppaluru, Kedar Gaitonde, Pranay Kanagat, Shankar Ramesh, Kunal Mehta, Meghavi Prashnani and Arpit Agarwal. Stella (Anoorag's pet), you have helped alleviate stress

during my Ph.D. with your purest soul and that cute smile.

I owe my deepest thanks to my family - my mother R. S. Shubha and my father K. Jagannatha, you have always stood by me and guided me through my career, and have pulled me through against impossible odds at times. I owe everything I am today to both of you. Words cannot express the gratitude I owe you. Thanks Amma and Daddy for believing in me and for being patient for the last five years with giving me absolute freedom to pursue my interests. You have never let the large physical distance between us bother me by always being there at any time of the day or night when I was tensed.

I would like to thank my dearest uncle Chinmaya, aunt Suma and my cousins Nikhil and Aditya who have been so welcoming as they were the only family away from my mother and father. They never let me feel alone by constantly checking for my well being.

Finally, I want to thank Dr. Vikram Hrishikeshavan, Dr. Derrick Yeo for all the help regarding aerospace things, lending tools when needed along with help with hardware. I am grateful to Prof. Inderjit Chopra for giving me an opportunity to teach ENAE788M which I will cherish forever for meeting one of the best graduate students and challenging them with hands-on projects.

I would like to acknowledge financial support from the Office of Naval Research (ONR), Brin Family Foundation, Northroup Gumman Corporation, Samsung Electronics, National Science Foundation (NSF), NVIDIA and Intel for all the research work discussed herein.

I would also like to thank the amazing open-source and open-hardware communities of Ubuntu, TensorFlow, ArduPilot, RaspberryPi and PX4 without whose work this thesis

would not have been possible.

It is impossible to remember all, and I sincerely apologize to those I have inadvertently left out from the bottom of my heart.

Lastly, thank you all and thank you God!

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Robots have always been imagined as intelligent agents that can solve any problem faster, cheaper and better than human beings. Such an imagination has been prevalent since the 1800s. Despite major advances in technology, autonomy in robots fall significantly short of predictions made in the past along with the expectations of most people. Most successful autonomous robots today are generally big, bulky and targeted towards a particular set of task such as cleaning floors, assembling cars and so on. To build a general purpose robot such as those shown in TV shows which posses capabilities similar to humans, we need to utilize concepts used by living agents for both hardware and software co-design.

The capabilities of such a general purpose robot in the wild can be categorized as: (a) Navigation, (b) Human robot interaction and finally (c) Physical interaction or Manipulation.

Navigation involves moving around in space without running into static or dynamic obstacles such as trees and humans. Human robot interaction involves the ability to understand humans, reason about their intent by interacting with them though a natural language to disambiguate hard to parse queries. Finally, physical interaction or manipulation involves the ability to alter the world by picking, moving and nudging objects.

This thesis focuses on the first capability: Navigation, specifically tailored towards aerial robots or Unmanned Aerial Vehicles (UAVs). However, they can be easily adapted to ground robots with minimal effort. This fundamentally involves answering the following three questions: *Where am I? and What is a hazard when I am moving? Where should I go next?*

This capability has been traditionally achieved using computer vision algorithms with the aim of building a representation of general applicability: a 3D reconstruction of the scene. Using this representation, planning tasks are constructed and accomplished to allow the quadrotor to demonstrate autonomous behavior. Note that I will use the words aerial robot(s), Unmanned Aerial Vehicle(s) (UAV(s)), quadrotor(s) or drone(s) interchangeably and they refer to the same entity unless specified otherwise.

1.1 Active Agents

Although aerial robots, are inherently active agents, their perceptual capabilities in literature so far have been mostly passive in nature. Researchers and practitioners today use traditional computer vision algorithms with the aim of building a representation of general applicability: a 3D reconstruction of the scene. Using this representation, planning tasks are constructed and accomplished to allow the robot to demonstrate autonomous behavior. However, this is in stark contrast to the methodology used by living agents such as birds and bees that have been solving these problems from agent with relative ease and extreme efficiency. These living beings utilize their activeness (the ability to control movement of their bodies or a part of it) to simplify perception problems by

building specific task driven sensorimotor loops (combination of perception, planning and control). This thesis is built upon this philosophy: the agent can control it's own movement or movement of a part of it's body to make it's perception problem simpler. This is due to additional constraints introduced by moving in particular ways. Such a movement to manipulate the perception forms a sensorimotor loop: a perception, planning and control loop to solve the task at hand. Note that solving a real world task at hand can utilize multiple of such sensorimotor loops.

1.2 Active vs Passive Approaches to Perception

Different set of tasks or competences of an aerial robot have been traditionally achieved with passive perception (or vision) which is based on the human or primate vision and involves sensing the world in 3D. This philosophy revolves around obtaining a 3D map first and then utilizing it for various tasks. However, a lot of tasks rarely require a full 3D map of the scene to be accomplished and are hence not minimalist (not utilizing less power, computation or number of sensors) by the virtue of their design. The major advantage of such a system is that it is agnostic to the morphology of the robot and can be almost directly adapted to different shapes, sizes and kinds of robots given that they posses the required computation, sensing and power on-board.

On the contrary, active perception (or vision) adapts the design philosophy based on the current operating constraints of computation, sensing and power. Such an approach though is not directly transferable to different agent morphologies, it is generally more power-efficient for the set of tasks it is designed for. Such an active design method is task

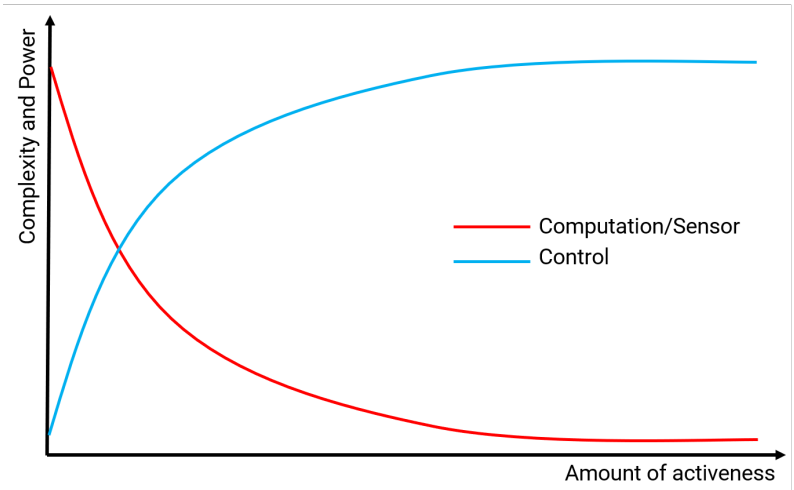


Figure 1.1: Sensing, Control and Computation variation with respect to the amount of activeness used by the agent.

driven and utilizes the minimal amount of information, computation and power required for the task. It can inherently handle risk of a sensor failure by the virtue it's design utilizing exploration to gather more information and is generally more robust albeit it might take more time as compared to it's passive counterpart (See Fig. 1.1). Table A.1 shows a comparison of different behaviours of an UAV using both active and passive design philosophies. Notice that the integration of different behaviours is harder in the active approach. To make this problem easier and more tractable, we propose to re-use multiple of these competences by conceptualizing each agent as a set of a hierarchical sensorimotor loops (or competences or behaviours). This makes it easier to adapt the agent to related but different sets of problems just by changing the checking condition (condition to see if a sensorimotor loops needs to be terminated). For eg., in the function of flower pollination, one would check for the flower and this could easily be changed to check for survivors in a search and rescue task.

Table 1.1: Minimalist design of autonomous UAV behaviours.

Competence	Passive Approach	Active and Task-based Approach
Kinetic stabilization	Optimization of optical flow fields	Sensor fusion between optical flow and IMU measurements
Obstacle avoidance	Obtain 3D model and plan accordingly	Obtain flow fields and extract relevant information from them
Segmentation of independently moving objects	Optimization of flow fields	Fixation and tracking allows detection
Homing	Application of SLAM	Learn paths to home from many locations
Landing	Reconstruct 3D model and plan accordingly	Perform servoing of landing area and plan appropriate policy
Pursuit and Avoidance	Reconstruct 3D model and plan accordingly	Track while in motion
Integration: Switching between behaviors	Easy: The planner interacts with the 3D model	Hard: An attention mechanism on ideas switching between behaviors

1.3 Forms of Activeness on a UAV

Activeness on a UAV or any robot in general can be accomplished in multiple ways. 1. By moving the agent itself, 2. By employing an active sensor, 3. By moving a part of the agent's body, 4. By hallucinating active movements.

In the first approach, the entire agent moves such that the perception problem becomes simpler. Such an approach is generally used by smaller robots where moving the entire agent is not very power hungry as compared to adding another sensor by increasing it's weight and computation.

The second approach utilizes an active sensor, i.e., a sensor which only works when movement is present. A class of sensors that are inspired by the animal eyes are called event cameras. Such cameras only collect the asynchronous intensity changes in light rather than traditional image frames. They have a higher dynamic range and lower latency compared to classical cameras. However, such a sensor lacks the ability to perform recognition from single timestep due to the lack of dense data when small or no movement is present. However, these sensors excel at tasks that involve movement, which is the core concept of activeness. This approach is useful when recognition of static objects is not required or high-speed and severe illumination changes might be encountered.

The third approach brings the change of robot's body morphology to enable simpler

perception. Such an approach can be used to make the robot smaller as required while utilizing a bigger set of sensors. Such an approach is desired when moving the robot is less power efficient than adding additional components to enable movement of the sensor suite. Such a method can also simplify certain perception problems by directly estimating depth.

Finally, the last approach entails utilizing a method which hallucinates an active observer. For e.g., one could hallucinate a heatmap of microsaccades a human being might exhibit by looking at an image. Such a method is computationally more expensive but can be utilized when power used by computation is far lesser than moving the agent or a part of it.

1.4 Hardware and Software Co-design

Keen readers would have realized that the active approach to designing algorithms is dictated by the amount of sensors, computation and power supply on-board which are commonly called Size, Weight, Area and Power (SWAP) constraints. This is in unison with what Alan Kay, a pioneer computer scientist quoted in the 1980s – *"People who are really serious about software should make their own hardware."* Thus, understanding the hierarchy of autonomous systems and applying it in engineering, becomes a problem of synergistic hardware and software co-design. This multidimensional optimization problem across different strata (hardware – integrated chips, sensors, effectors and software – the set of programs running on the system) is a new research area that has the potential to lead to a disruptive technology in this field. We call this field "Embodied AI". This is

directly inspired by how nature has taken ages to solve the optimization problem between hardware (sensing using eyes and other sensors) and software (neural architecture) and it is still being refined by a method we call 'Genetic Evolution' [2].

This multidimensional problem is intractable if we try to study all combinations of sensor placements, computation and algorithms. Hence, to make the problem tractable, we limit all our algorithms to be implemented using deep learning modules which obey SWAP constraints and to the morphology of a quadrotor. However, the sensor suite is allowed to change between a monocular traditional or event camera and a stereo camera.

1.5 When is Active design useful?

The next obvious question that comes to mind is when such an active design philosophy is useful. An active philosphy using the agent's movement or the movement of a part of it's body is useful when the agent cannot carry passive sensors that can sense the required quantity (such as depth for mapping) due to SWAP constraints. However, that being said, an active sensor can still be used despite the size of the robot for low-latency applications. To add further, an active mechanism on a large robot (with enough SWAP constraints for a myriad of sensors) can act as a failure mechanism when one or more of the sensors fail. This could avoid the UAV from crashing by landing it safely. An active philosophy is also useful when uncertainty is expected in either the environment or sensing. In stark contrast to computer vision approaches, the agent would move to explore the world to gather more information for a confident prediction.

Fig. 1.2 shows how the ability of a robot is severly affected as the SWAP constraints

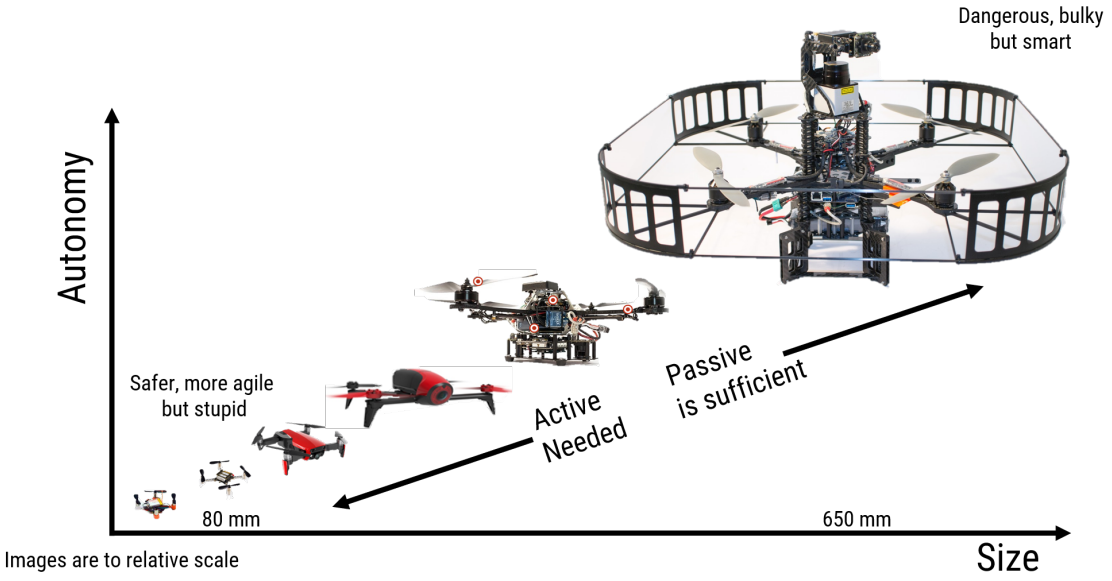


Figure 1.2: Algorithmic design philosophies for different sized robots along with their capabilities.

change. This shows that we need to use the active design philosophy as SWAP constraints become tighter (UAV becomes smaller). This is important as smaller UAVs are safer, more agile and are scalable to be used as swarms. But, today's passive algorithmic design philosophy has bounded their abilities severely and the level of autonomy of these smaller UAVs are very far from those of their larger counterparts. This shows how tightly coupled the design philosophy is with the SWAP constraints.

However, when we bring living beings into the same plot as above we see from Fig. 1.3 that the different between autonomy levels is not as significant even though living being size changes drastically. This is because of the adaptation of sensing, neural architectures and the methodology used by these agents that scale well with their SWAP constraints. Also, notice that the level of autonomy possessed by living agents are far

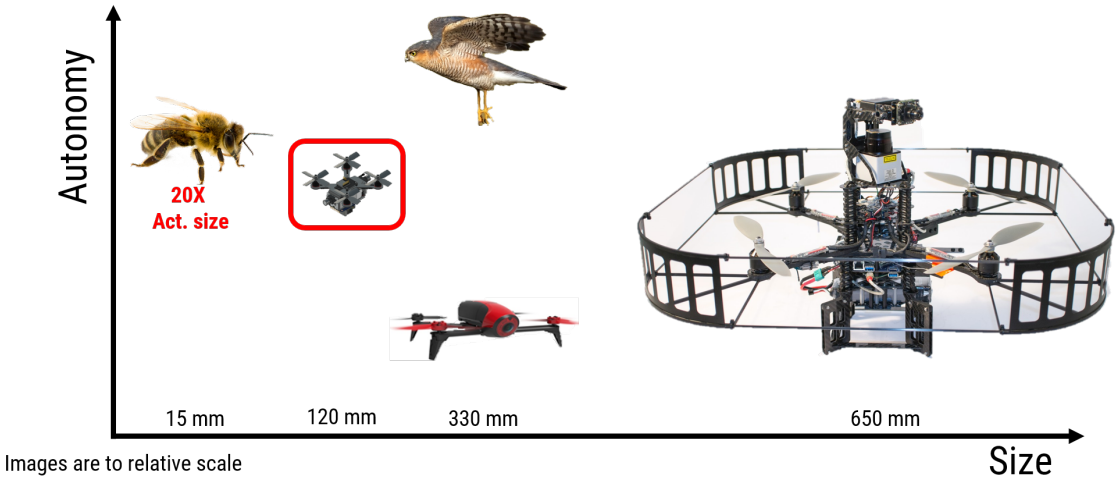


Figure 1.3: Amount of autonomous capabilities for different sized robots and living beings with respect to size. The red box shows where this thesis aims to take the autonomy of a nano-quadrotor.

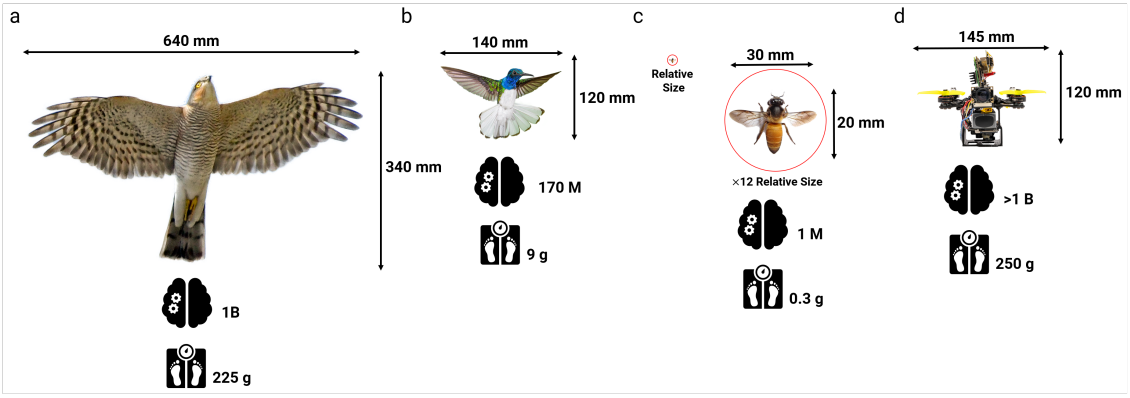


Figure 1.4: Comparison of our proposed "bee" nano-quadrotor with birds and bees. (a) Sparrowhawk, (b) White Necked Jacobin Hummingbird, (c) Giant Honeybee, and (d) Our proposed "bee" nano-quadrotor. The number next to the brain and scale icon shows the number of neurons and the weight respectively. *Note that the images are to relative size.*

higher than our man-made autonomous UAVs. A detailed comparison between a sparrowhawk, a hummingbird, a bee and our "bee" nano-quadrotor is shown in Fig. 1.4. *One can observe that our nano-quadrotor has more computation than the sparrowhawk, weights almost the same as the sparrowhawk but is around the size of the hummingbird but has the capabilities similar to that of a bee.* This is because of our technology and design methodology being not as efficient as that of nature and this area has a huge scope for further research. This thesis is one of the few works that has taken baby steps in this area of on-board nano-UAV autonomy using an active approach.

Next, I will describe some applications of a nano-UAV (specifically a quadrotor that weighs $<$ 250g and maximum motor to motor diagonal size of 120 mm) which is based on the active design philosophy.

1.6 Applications of an Active Nano-Quadrotor(s)

Quadrotors out of all possible aerial robots have gained massive popularity due to the simplicity in their mechanical design: a frame with four motors where diagonally opposite motors spin the same direction. Such a design is inherently stable due to counteracting torques and can be directly controlled by changing motor speeds since it is a differentially flat system. A quadrotor has the following advantages over a traditional fixed wing aircraft: can hover in place, generally has a higher payload for the same size and can fly faster for the same size. They also don't need a runway or a catapult mechanism to take off and can be easily deployed fully autonomously in the field.

In particular, nano-quadrotors are defined in this thesis as a quadrotor of maximum

side of 85 mm (motor to motor diagonal dimension of 120 mm) and a maximum All Up Weight (AUW) of 250 g. These vehicles are safe, agile and can carry enough payload to be autonomous with off-the-shelf components making them easy to repair and distribute for further research even in developing countries. A few applications of such a nano-quadrotor (one or many) are described next.

Fast exploration and mapping

Nano-quadrotors can be used to map large areas fast using a centralized or decentralized mapping server where a fast map could be generated on-board and can be further optimized when they are downloaded to a large server. Such a method can be very useful when one can deploy drones with different sensor suites for gather data from different spectra. This would be useful for mapping bridges and pipes where gaps are small and large drones cannot traverse. Such nano-drones can also be used for sports to obtain angles that were not possible before.

Search and Rescue

Nano-UAVs are specially suited for search and rescue since they can be readily deployed with minimal effort in the field and can traverse gaps of unknown shapes, sizes and locations. Such a swarm with different sensor suites can be used to find survivors (using a thermal camera). Nano-UAVs can traverse terrain that might not be possible for ground robots or larger UAVs due to their size, agility and full 3D maneuverability. Such small nano-UAVs because of their cost-effectiveness can be used as 'disposable information capture devices' for areas such as nuclear plants. Finally, they can also provide a bird's

eye view of the scene along with 3D reconstructions for necessary evaluation and to collaborate with other ground robots.

E-sports and Hobbying

Drones have become popular due to a sport called *drone racing* and have carved out a new space that blends reality with computer games. These drone races are manually piloted and recently AlphaPilot competition from Lockheed Martin posed to do the same in a completely autonomous manner with on-board sensing and computation which was won by Prof. Guido de Croon's team from TUDelft using a monocular and active approach[1]. These features could be built into drones of smaller sizes to make them safer to learn and for hobbying purposes so they do not hurt other birds in flight. One such step has been taken by DJI in the form of DJI FPV drone (that weights about 1 Kg), but it still is far from being a nano-quadrotor.

Co-operative delivery

Although nano-UAVs cannot carry larger payloads, they can be combined together (either using rigid links or tethered cables) to carry larger payloads. Such an approach would be desirable during search and rescue as smaller drones are easier to deploy and are faster in exploration but can be combined together to lift larger payloads.

Inspection

Drones are a popular tool for inspecting large structures where access is limited to ground robots or using human labor is dangerous such as under bridge structures [3] or historical

[1] https://mavlab.tudelft.nl/mavlab-wins-the-alpha-pilot-challenge/

monuments [4] or in radio active areas [5]. Recently , Skydio announced an active approach called 3D Scanhttps://www.skydio.com/3d-scan to map and inspect any structure in a GPS denied environment using only on-board computation and sensing. In addition, a research area could be to use a gimbal camera with zoom that can also use active perception to reduce overall control effort of moving the entire vehicle to obtain a higher resolution image for mapping. Such an approach would make it safer when tight areas have to be mapped with a high degree of accuracy.

1.7 Research Objectives

Since I started my doctoral studies in 2016, there has been an astounding progress in the field of quadrotors and specially autonomous ones. One of the major factors that influenced this growth was the Drone Racing League (DRL) which was launched publicly in Jan, 2016. Since then, there has been a rapid development and availability of hobby grade parts such as sensors, motors and on-board flight controllers that are on-par with their commercially available counterparts and those used in the top of the line research. This trend has also set off a cycle where Ph.D.'s are working with hobbyists to build autonomous drones (different multirotors and fixed wing planes) using hobby grade parts so that such research is accessible to the general public. To fuel this growth further, there has been a rapid advancement in the field of on-board algorithms involving visual perception (visual inertial odometry), better sensor fusion, motion planning, control schemes and decision making. The recent deep learning trend has also shown to improve robustness over classical methods if the training data is generated appropriately. These networks

can also be trained in simulation and transferred to the real world with minimal effort if the problems are chosen carefully. To add further, these networks can be hardware accelerated to obtain speeds not possible before with classical methods without extensive reprogramming to fit the required computer architecture.

Although, electronics and avionics exist for building efficient nano-quadrotors (we define this as a maximum diagonal motor to motor size of 120 mm and AUW of 250 g), they rarely posses autonomous features. This is because of a large number of open challenges to scale down autonomy with limited sensing and computation. With the advent of special deep learning accelerator chips, open-source simulation tools for data generation, new generation sensors and new FAA rules for small drones, now is the perfect time to dive into make these nano-quadrotors autonomous. This research direction also caught attention of many top notch researchers across the world due to the DARPA's Fast Light Autonomy project where the goal was to build an autonomous quadrotor that can fly upto 20ms^{-1} with on-board sensing and computation.

The myriad of challenges to make nano-drones autonomous span across multiple domains and multiple Ph.D. theses and will take an effort from researchers all across the world to be addressed. My doctoral work focuses on a few key components which revolve around the central philosophy of active perception. Each of my works has a philosophical ideology of active perception with a practical application of immediate impact. I particularly showcase four forms of active perception (combining perception, planning and control into a single sensorimotor loop to make the problem simpler to solve): 1. By moving the agent itself, 2. By employing an active sensor, 3. By moving a part of the agent's body, 4. By hallucinating active movements. Next, to make this work practically

applicable I show how hardware and software co-design can be performed to optimize the form of active perception to be used. Finally, I present the world's first prototype of a RoboBeeHive that shows how to integrate multiple competences centered around active vision in all it's glory.

Today's drone autonomy in indoor (sometimes outdoor) spaces has been traditionally achieved using motion capture setup which are expensive, have a large setup and calibration time, and strongly limit the flying area. The key advantages of such a system is their *sub-millimeter* accuracy, constant and *low* latency of about 10ms and high update rate of up to 200Hz. However, such a system is not applicable when drones need to be deployed in the wild with wind, changing terrain, varied visual features and illumination, and other external obstacles. This has necessitated the need for on-board perception (sensing) and computation. Such an on-board system also has the added advantage of preserving privacy and security (by being harder to hack into). In this thesis, I focus on on-board sensing and perception to solve fundamental problems for aerial robot autonomy: static and dynamic obstacle avoidance, flying through gaps and homing (where do I find home). All these problems are tackled by *controlling the agent's movement in a way to simplify perception problems* (this is called a sensorimotor loop) using the active perception philosophy.

1.8 State of the Art

I will next describe the state of the art in various related areas that can potentially advance the area of autonomy on nano-UAVs.

Active Perception by moving the agent

Active perception is the concept where one would control their movements (control and planning) in order to simplify the perception problem. This is in stark contrast to passive perception where a 3D map is first constructed and then motion planning is performed which is then used to control the robot's movement. Such a passive approach relies on the fact that the perception information obtained is accurate and robust which is rarely the case in the wild. Such an approach has spurred from the fact that it is the most evident way to setup a mathematical optimization problem which can be sub-divided into multiple fields so that fast progress could be achieved. Now that the field has matured to the point where we have full-fledged products based on this philosophy, academia needs to think about the next conceptualization of a robot. Revisiting what experts of the field put forth about three decades ago: we need to combine perception, planning and control into a single entity called sensorimotor loops. This has been called by different names: Active vision, Animat vision, or perception-aware planning. This literature has gained a lot of momentum in the last five years due to active vision's robustness, built-in contingency planning and minimalism. The literature spans across multiple robot platforms such as quadrotors, ground robots and humanoids to name a few and is dominant where robots can carry a limited payload in-terms of sensors and computation.

Active view planning

Another way planning and perception have been tightly coupled is through an area called active view planning. Here, the planner takes into account the current or accumulated perception information to plan a path on the fly to obtain a better view for the desired

task. For e.g., if one is mapping a bridge [6], generally spots under the bridge are dark and require closer data and hence the view planner would take that into account. Such a planner is accomplished by various statistical measures such as entropy or estimated coverage amount for the map built. Another flavor of active view planning comes in the form of keeping some points of interest in the field of view while executing a desired trajectory. Such a problem is tackled by formulating minimum time trajectories for quadrotors with a limited field of view camera to ensure that the points of interest are always in the field of view. Alternative formulations have been presented to control yaw for autonomous aerial cinematography [7]. To add further, a few approaches also consider obstacle avoidance when planning such perception-aware trajectories [8].

Active Sensing using event cameras

Over the last decade, there has been an enormous advancement in sensor technology, specially imaging sensors or cameras. Even after these significant advancements, the dynamic range and latency of these cameras is nowhere comparable to those possessed by living agents, specially ones that can fly such as birds and bees. This observation motivated the neuromorphic engineers to develop a new class of imaging sensors called event cameras. These event cameras have 'smart pixels', wherein each pixel is asynchronous and collects the changes in light rather than a traditional intensity measurement to create classical image frames. Such a sensor outputs an event cloud rather than an image frame. where each event contains the location and polarity. The polarity takes values from the set $\{+1, -1, 0\}$, where a +1(-1) indicates a intensity increase (decrease) and 0 indicates the event did not trigger. Since, only the intensity changes are recorded, huge bandwidth sav-

ings of upto orders of magnitude is obtained. Such a sensor also has a ultra high dynamic range of upto **100dB** which is much higher than a traditional camera [9]. The latency of such a camera can be on the order of a few microseconds which is two to three orders of magnitude lower than that of the comparable classical camera.

Since event sensor data is tightly coupled to it's movement and the scene, they are also called active sensors in the essence that one has to move (or the scene has to move) to obtain data. To reiterate, the event camera has a high dynamic range, low latency, low bandwidth and is particularly suited for an active agent. As one would think intuitively, the larger the perception latency, the slower the robot can respond to abrupt or dynamic changes in the environment/scene. However, robots in the wild often experience dynamic obstacles such as humans, birds, insects along with unstructured obstacles in collapsed buildings like falling rocks. Although in theory, one could recognize the objects in the scene to obtain dynamic obstacles, they are not robust to motion blur and drastic illumination changes which classical cameras suffer from in dynamic and wild scenarios. Event cameras by the virtue of their design are perfectly suited for the task of dynamic obstacle detection. In literature, event camera based Independently Moving Objects (IMO or dynamic obstacles) has been performed in two major ways: by motion compensating the event volume (or frame as a projection of events) [10, 11] or by learning to predict the segmentation masks directly[12, 13]. In the first method, the algorithms construct an Image of Warped Events (IWE) or event frames then use a measure of IWE's contrast or sharpness as a metric to warp the event cloud to obtain a *sharp* IWE or event frame. The blurry parts of this resultant image are the regions that do not comply with the motion model of the 'background' (the parts of the scene that are not IMOs) are the 'foreground'

or IMOs. Such a way is robust but generally slow due to processing of 3D spatio-temporal event data, depending on amount of motion and scene this can use a lot of memory and be slower than realtime (defined as 50Hz). In the second method, a network is trained on these input event frames (without sharpening) to predict directly the pixel locations that belong to IMOs. Such a method is faster and robust if enough data is given to learn.

Event cameras due to their myriad of advantages over classical cameras have also gained attention of other roboticists to build visual odometry [14, 15, 16] and Simultaneous Localization And Mapping (SLAM) algorithms[17]. Event cameras have also been utilized on humanoid robots [18] and other aerial robots to track objects of interest.

Active Perception by moving a part of the agent

One can also achieve activeness by moving a part of the agent rather than the entire agent itself (Such an example in nature can be seen when the owlets bob their head around in circles, see Fig; 1.5). Though morphable designs of robots in it's early days [19, 20, 21, 22, 22, 23, 24] were to showcase that one could make such a design functional and were targeted towards mechanical and control design of these systems. In the last decade, gimbal systems on drone have become common to the point that even a cheap hobby drone generally is equipped with one. One of the major driving forces for such a design is aerial videography to get a smooth footage with little to no post-processing. This also inspired roboticists to build rigs with pan, tilt and zoom cameras [25] on robots so that they could track objects faster by moving the camera rather than the entire robot itself. Similarly, stereo systems were built on the same principle to enable better tracking and depth estimation by changing baseline.

Figure 1.5: A stack of images showing an owlet bobbing its head (see red highlight) to make perception easier. This is an example of an agent moving a part of it's body to exhibit activeness. For original video see https://vimeo.com/152347964.

Hallucinated Activeness

Activeness as discussed before comes in various forms and involves some amount of physical movement. However, similar to the difference in the way the older agents (or humans) approach problems as compared to younger ones, activeness changes its form as more data or information about something is acquired. Foe e.g., a bird does not have to explore the area around it's nest since it already knows how it looks by creating a mental picture or it. In such a case, activeness becomes closer to passiveness with one single difference: the activeness is hallucinated or is in the imagination of the agent. Hallucinated activeness in the most simplified form could be captured in visual saliency: the amount of time an agent's gaze rests on different parts of the image (See Fig. 1.6 for an example of how this map looks). Such a saliency method depends on the context and is generally a top to down approach. For e.g., if one is looking for a yellow ball, the saliency heatmap would light-up at spots of the image which are yellow. Saliency could also be in the form of a rudimentary motion segmentation method, such an approach would be trivial if an event camera is employed. Also, note that most living agents have evolved eyes to sense in the range to find the most salient objects from their perspective. For e.g., bees and butterflies can 'see' in the ultraviolet range to find the salient flowers where they can drink nectar from (See Fig. 1.7). In literature, saliency has been used for navigation [26], human-robot interaction [27, 28] and to make robots have gazes similar to that of humans[29].

Nano/Pico-quadrotor design

Smaller UAVs are inherently safer, more agile and their ability to be deployed as large

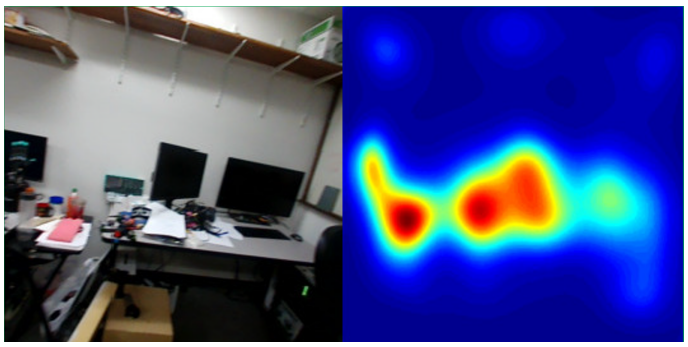


Figure 1.6: Left to right: Color image of the scene, corresponding saliency map output by SalGAN [1]. The hotness of the saliency color corresponds to the value being higher.

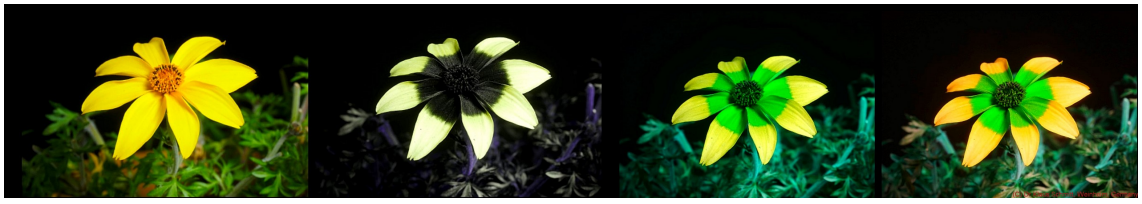


Figure 1.7: Left to right: Bidens ferulifolia flower as seen by Human vision, reflected UV, butterfly vision and bee vision. Note that altough images shown here for simulated butterfly and bee vision are at the same resolution as those seen human eyes, the real resolution of the eyes on these flying agents is much smaller. Photo credits and ©: Dr. Klaus Schmitt.

swarms [30]. In literature, pico-quadrotors have been developed with a myriad of capabilities such as walking and flying but rarely have enough on-board computation to be of practical value. Recently, custom chips based on the RISC-V architecture have been developed to aid autonomy features using deep learning to these tiny pico-quadrotors [31], however they require a mastery on hardware design along with writing lower level driver code to make them functional and autonomous: thereby limiting further research. Moreover, cameras and other sensors that can be used on these sized drones are either custom made and expensive. To add further, the battery life on these tiny drones are less than 2 minutes due to the limited payload capabilities. A similar issue is observed with flapping bird drones which are inherently safer[32]. On the other end of the spectrum, research groups all over the world have also worked on integrating larger sensors and computation modules into smaller units to run a full SLAM stack along with path planner and controls enabled by the latest generation of Graphics Processing Units (GPUs) [33]. Although massive advances have been made in this regard and have been made open-source and open-hardware, they still use expensive sensors to achieve a good SLAM accuracy. To this end, we propose to utilize activeness in our design and build nano-quadrotors which fight right in-between the two aforementioned areas to enable autonomous features at scales not possible before due to the higher payload and computation as compared to pico-quadrotors and improved agility, safety and size decrease as compared to micro-quadrotors. We also bring the first step in studying the co-design of hardware and software for nano-quadrotors using embodied AI.

1.9 Summary

In this chapter, I discussed what active agents are, how an active agent differs from a passive agent. This was then extended to why and how one should use activeness of an UAV to make perception problems simpler to enable autonomy at scales that were not possible before. I also talk about how activeness is a hardware and software co-design problem. Then, I formally present the research objectives of this thesis followed by different applications of nano-quadrotors. In the literature review, I summarized the state of the art on different forms of active perception, combining perception and planning, and nano/pico-quadrotor design.

Chapter 2: Contributions

In this chapter, I summarize the key contributions of the papers re-printed in the appendix. In particular, this chapter highlights the individual results, refers to the related video, open-source and open-hardware tutorials. I also put this thesis' research in-context to the state of the art literature in this chapter. In total, this research has been published in three peer-reviewed journals and six peer-reviewed conference publications. One further paper is under preparation for *AAAS Science Robotics* journal. Paper A was successfully demonstrated live to multiple audience, most notable to Sam Brin (brother of Sergey Brin) which was awarded USD 200K from the Brin Family foundation to our lab to advance machine perception on drones along with a grant to the University of Maryland of USD 2M for building the Brin Family Aerial robotics lab which is currently used by various departments. Paper B was presented by Profs. John Baras and Yiannis Aloimonos to the Office of Naval Research and obtained a grant of USD 2.2M to advance the field of Intelligent and Learning Autonomous Systems: Composability and Correctness. The works in papers A and B also won the runner up in the Brin Family prize in 2016. Papers B and C have helped cultivate relationships with one of the best labs for aerial robotics research in the world: Robotics and Perception Group at the University of Zurich headed by Prof. Davide Scaramuzza and Micro Air Vehicle Laboratory at the Delft University

of Technology headed by Prof. Guido de Croon. The works from this thesis was used to create the world's first prototype of the RoboBeeHive which is described in detail in Chapter 3. Finally, a lot of the research in this thesis has led to the creation of two fully open-source and open-hardware courses with video lectures and slides: ENAE788M: Hands-on Autonomous Aerial Robotics and CMSC828T: Vision, Planning and Control in Aerial Robotics.

2.1 Active Perception by moving the agent

In this section, I present the work on the textbook definition of active perception: by controlling the agent's movement to simplify the perception problem. Such a work is also called perception-aware planning in the classic literature.

2.1.1 Paper A: GapFlyt

(P1) **Nitin J. Sanket***, Chahat Deep Singh*, Kanishka Ganguly, Cornelia Fermüller, Yiannis Aloimonos "GapFlyt: Active Vision Based Minimalist Structure-Less Gap Detection For Quadrotor Flight", *IEEE Robotics and Automation Letters (RA-L)*, (2018) Vol. 3, No. 43847–3854. DOI: http://dx.doi.org/10.1109/LRA.2018.2843445.

2.1.1.1 Brief Description

In this work, we address one of the biggest challenges for autonomous operation of a UAV in complex environments: navigating through narrow gaps of unknown shape,

size and location. To enable such a competence, our quadrotor (which could be replaced by any multirotor UAV) is equipped with a front facing monocular camera, an inertial measurement unit, a laser altimeter and an on-board computer. All our perception, control and planning calculations are done using on-board sensing and computation. Instead of relying on the 3D structure of the scene using a mapping or odometry approach, we utilize the activeness of the quadrotor to simplify the problem of gap detection. A gap is defined as the contour where there is an abrupt discontinuity of depth. This is inherently captured in the optical flow measurements when the quadrotor explores the scene (See Fig. 2.1a). Since the optical flow is also a function of the rotation, we average multiple measurements such that the dominant average flow values are from translation which in-turn have inverse depth embedded in them. So by simply moving the quadrotor to compensate for lack of computation and sensing, we can obtain the location of the gap on the image plane without reconstruction or explicit computation of depth. We then track the gap indirectly by tracking the foreground (outside the gap) and background (inside the gap) regions separately since it is unstable to track a depth discontinuity directly. Finally, our quadrotor aligns the center of the gap with the image center to fly through the gap (See Fig. 2.1b).

2.1.1.2 Comparison to the state of the art

Current state of the art for flying through narrow gaps has focused on either planning and controls for high speed flight or for on-board perception and sensing with known shape and size gaps. The common ground to all the state of the art literature is that they

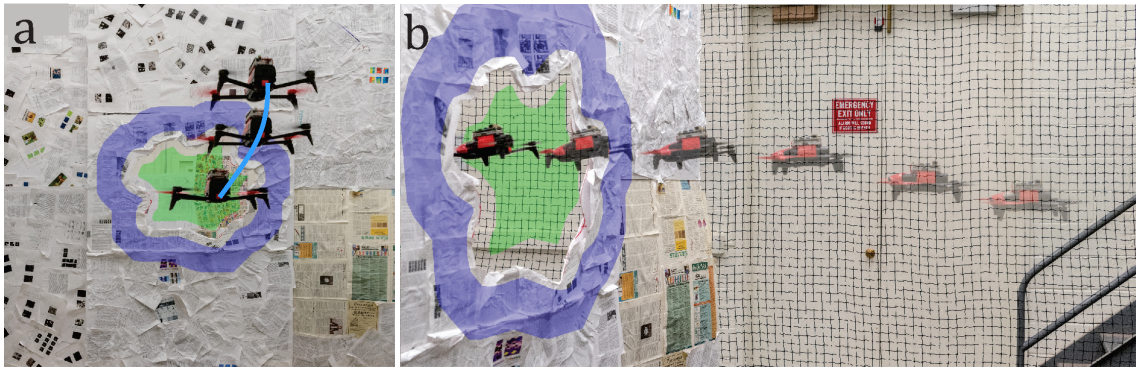


Figure 2.1: Different parts of the GapFlyt framework: (a) Detection of the unknown gap using active vision and TS^{2}P algorithm (cyan highlight shows the path followed for obtaining multiple images for detection), (b) Sequence of quadrotor passing through the unknown gap using visual servoing based control. The blue and green highlights represent the tracked foreground and background regions respectively.

are focused towards controls and planning since the perceptual part is either given or simple because of known assumptions. In contrast, our work's focus was centered around perception and focused on solving the problem with minimal information and sensing. With just a monocular camera, we achieved a remarkable success rate of upto 85% at 2.5ms^{-1} even with a minimum tolerance of just 5cm which is comparable or in some cases better than the methods that use either depth or a motion capture system or a known window shape and size.

2.1.1.3 Related Videos

(V1) https://youtu.be/FSSqB7ag04w

2.1.1.4 Related Software and Hardware

(S1) https://github.com/prgumd/GapFlyt

2.1.1.5 Related Media Publicity

IEEE-Spectrum, TechCrunch, NVIDIA, Technology News Update, Drone Below, Japan Tech Crunch and many more.

2.2 Active sensing using event cameras

In this section, I present the work on using activeness in the sensor design: sensor only outputs data when the agent moves. In particular, the sensor we use is called an event camera and the output data is a combination of the camera's movement along with the structure of the scene.

2.2.1 Paper B: EVDodgeNet

(P2) **Nitin J. Sanket***, Chethan M. Parameshwara*, Chahat Deep Singh, Ashwin V. Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, Yiannis Aloimonos "EVDodgeNet: Deep Dynamic Obstacle Dodging with Event Cameras", *IEEE International Conference on Robotics and Automation (ICRA)*, (2020). DOI: https://doi.org/10.1109/ICRA40945.2020.9196877.

2.2.1.1 Brief Description

In this work, we address one of the most common problems quadrotors (UAVs in general) encounter when navigating in the wild: obstacle avoidance of dynamic or moving obstacles. These obstacles could be in the form of other living agents such as human beings, birds, insects or non-living things like other drones, rocks being thrown by other people etc. The fundamental challenge to dodging dynamic obstacles is in the perception, which involves distinguishing one's own motion to that of other moving objects. To enable this competence, our quadrotor is equipped with a front and down facing event cameras, an inertial measurement unit, a laser altimeter and an on-board computer. All our perception, control and planning calculations are done using on-board sensing and computation to dodge dynamic obstacles of unknown shape, size and location without explicit depth information. Contrary to the state of the art we utilize event cameras, which are active sensors and are tailor made for the purpose of detecting independently moving objects/obstacles. These event cameras contain 'smart pixels', where each pixel operates asynchronously and is triggered by brightness changes. We present a solution to dodge dynamic obstacles by utilizing three shallow neural networks. The first network *deblurs* the input event frame, the second network predicts the motion between two event frames and finally the third network predicts the segmentation flow: optical flow only at moving objects. We then filter these measurements using temporal information which is then used to obtain a safe direction in which the quadrotor moves to avoid collision (See Fig. 2.2). All our networks are trained in simulation and generalize to the real-world without any fine-tuning or re-training.

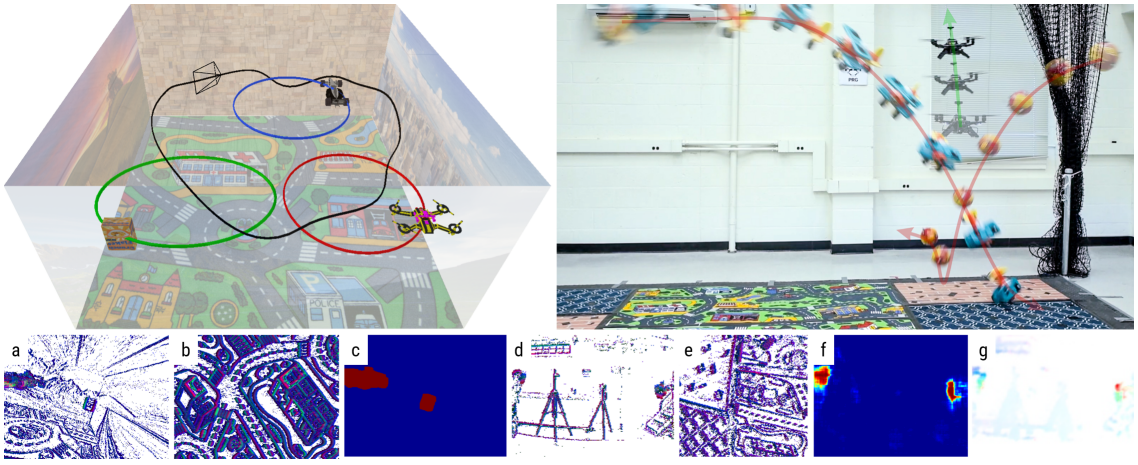


Figure 2.2: A real quadrotor running *EVDodgeNet* to dodge two obstacles thrown at it simultaneously. From left to right in bottom row: (a) Raw event frame as seen from the front event camera. (b) Segmentation output. (c) Segmentation flow output which includes both segmentation and optical flow. (d) Simulation environment where EVDodgeNet was trained. (e) Segmentation ground truth. (f) Simulated front facing event frame.

2.2.1.2 Comparison to the state of the art

Current state of the art for dodging dynamic obstacles that utilize classical cameras either recognize the obstacles such as humans or other animals, or assume information about the shape of the object such as a spherical ball. Other methods that work for different shapes either use a stereo camera or an RGB-D sensor to obtain depth. Recently, approaches have been presented to detect dynamic obstacles using either stereo or monocular event cameras. Monocular event camera based methods are generally slow as they typically work on the entire event cloud. To this end, our work currently holds the state of the art for monocular event based dodging on a quadrotor using event cameras with an accuracy of 70% including objects of unknown shape, size, location and low light

scenarios.

2.2.1.3 Related Videos

(V2) https://youtu.be/k1uzsiDI4hM

2.2.1.4 Related Software and Hardware

(S2) https://github.com/prgumd/EVDodgeNet

2.2.1.5 Related Media Publicity

Mashable, Futurism and many more.

2.2.2 Paper C: EVPropNet

(P3) **Nitin J. Sanket**, Chahat Deep Singh, Chethan M. Parameshwara, Cornelia Fermüller, Guido C.H.E. de Croon, Yiannis Aloimonos "EVPropNet: Detecting Drones By Finding Propellers For Mid-Air Landing And Following", *Robotics: Science and Systems (RSS)*, 2021.

2.2.2.1 Brief Description

In this work, we address a specific variant of the problem described in *EVDodgeNet* which is to detect other drones (specifically multirotor UAVs). Such a detection is important either for detection of malicious drones for security purposes or for swarming

where we would like to detect and estimate relative pose to other drones in the swarm. Instead of relying on special structures, markings or fiducials on drones, we detect the most ubiquitous part of a drone: it's propellers. Since, the area occupied by the non-occluded propeller is far greater than that of the largest fiducial that can be placed on the drone it makes the argument compelling to detect drones by finding propellers. However, classical cameras do not posses the shutter speeds to capture the rotation of a propeller. To this end, we detect propellers by using an event camera for its high temporal resolution, low latency and high dynamic range. Our quadrotor can detect other drones using a single event camera with on-board computation and sensing. In particular, we model the geometry of a propeller to generate synthetic data which is then used to detect propellers in the real world without any fine-tuning or re-training. Our *EVPropNet* network runs on a power budget of just 2W at 35Hz making it perfectly suited for real-world applications. We show two applications of our work: (a) tracking and following an unmarked drone and (b) landing on a near-hover drone (See Fig. 2.3).

2.2.2.2 Comparison to the state of the art

Current state of the art for detection of drones rely on either appearance based neural network methods which often fail if the background is not sky. The other approach is to use an active or a passive fiducial marker on a drone which can be detected fairly robustly. Active markers are generally more robust and have a farther sensing range as compared to the passive ones. However, for detection of unmarked drones, only the appearance based methods are applicable. Other methods which RADAR to detect drones

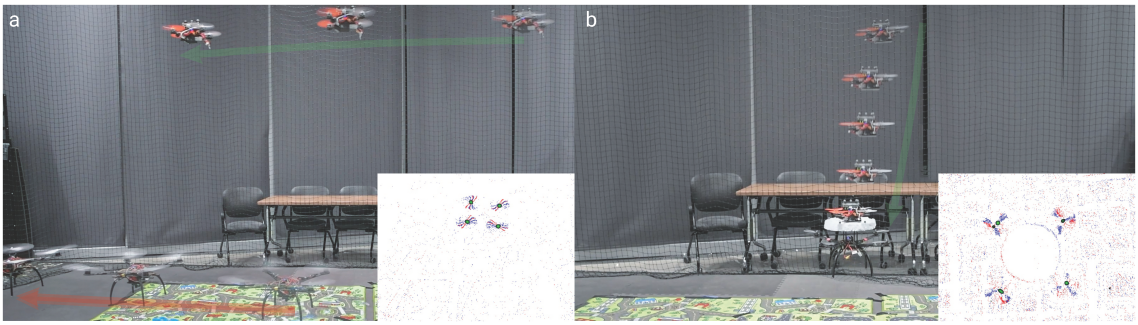


Figure 2.3: Applications presented in this work using the proposed propeller detection method for finding multi-rotors. (a) Tracking and following an unmarked quadrotor, (b) Landing/Docking on a flying quadrotor. Red and green arrows indicates the movement of the larger and smaller quadrotors respectively. Time progression is shown as quadrotor opacity. The insets show the event frames $\mathcal{E}$ from the smaller quadrotor used for detecting the propellers of the bigger quadrotor using the proposed *EVPropNet*. Red and blue color in the event frames indicate positive and negative events respectively. Green color indicates the network prediction.

are generally bulky to be carried on-board a drone. In contrast, out method can detect both marked/unmarked drones since propellers act as the fiducial for our method. Our network can detect propellers at a rate of 85.1% even when 60% of the propeller is occluded and can run at upto 35Hz on a 2W power budget. Our applications also show an impressive success rate of 92% and 90% for the tracking and landing tasks respectively.

2.2.2.3 Related Videos

(V3) https://ter.ps/EVPropNet

2.2.2.4 Related Software and Hardware

(S3) https://github.com/prgumd/EVPropNet

2.3 Active Perception by moving a part of the agent

In this section, I present the work on a modification to the textbook definition of activeness. Here instead of moving the entire agent to obtain more information, we move a part of the body, similar to head movements birds or humans make to obtain more information. Such a method of activeness is useful as it is generally faster and applicable when the agent's movement costs a lot more energy rather than just moving a part of it.

2.3.1 Paper D: MorphEyes

(P4) **Nitin J. Sanket**, Chahat Deep Singh, Varun Asthana, Cornelia Fermüller, Yiannis Aloimonos "MorphEyes: Variable Baseline Stereo For Quadrotor Navigation", *IEEE International Conference on Robotics and Automation (ICRA)*, (2021).

2.3.1.1 Brief Description

In this work, we address a long-standing problem with stereo cameras on a UAV: to adapt the depth sensing range of the stereo vision system on the fly to obtain better depth quality. Our quadrotor is equipped with a stereo camera mounted on two linear servo motors and on-board computer on which all the perception, planning and control algorithms are run with on-board sensing and computation. Since the depth quality of a stereo camera depends on the distance between two cameras (baseline) and the structure

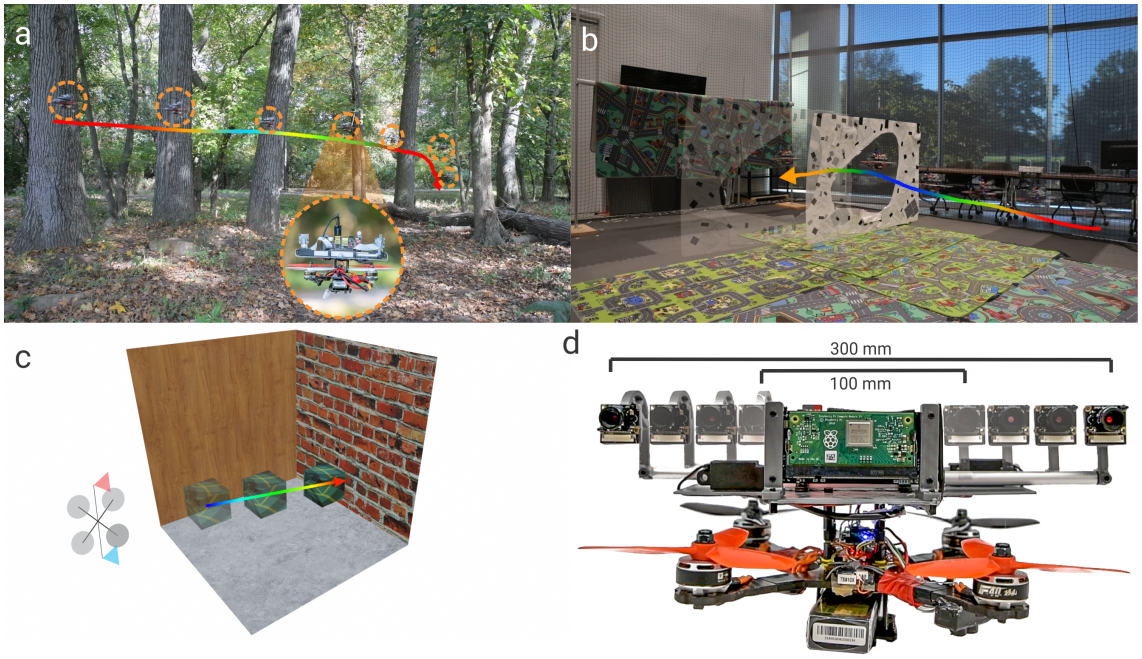


Figure 2.4: Three applications of a variable baseline stereo system were explored in this work. (a) Flying through a forest, (b) Flying through an unknown shape and location dynamic gap, (c) Detecting an Independently Moving Object. In all the cases, the baseline of the stereo system is changing and is colored coded as jet (blue to red indicates 100 mm to 300 mm baseline). The opacity of the quadrotor/object shows positive progression of time. (d) Variation of baseline from 100 mm to 300 mm. Notice that the stereo system is bigger than the quadrotor at the largest baseline.

(depth) of the scene, we adapt our baseline on the fly based on the region of interest. In order to obtain extrinsic calibration during the baseline change, we pre-calibrate our camera at fixed baseline pairs and interpolate between the extrinsics on the fly using a dual quaternion formulation due to its speed and robustness. Finally, we demonstrate three approaches of such a variable baseline stereo system: (a) flying through a forest, (b) flying through an unknown shaped/location static/dynamic gap, and (c) accurate 3D pose detection of an independently moving object (See Fig. 2.4).

2.3.1.2 Comparison to the state of the art

Related state of the art comes in two different domains: (a) morphable designs and (b) variable baseline stereo systems. In the first domain, control and planning algorithms have been developed for morphing aerial robots: aerial robots that can adapt their shape in air. However, such a morphing mechanism is rarely driven by perception and is generally used to showcase the capability to be able to change shape in a controllable manner. In the second domain, various slider based and servo based variable baseline stereo systems have been developed for tracking purposes. In our work, we combine these two philosophies and present a variable baseline morphing system based on perceptual information. Using our method, we can navigate through a forest at speeds of upto $5\text{ms}^{-}1$ with a stereo camera setup and on-board sensing and computation. We improve on speed of gap detection by $80\times$ (when using same computer for processing) when compared to [34] as our proposed approach does not require any active quadrotor maneuvers for gap detection. We also improve on speed of gap traversal by 20% (3 ms^{-1} over 2 ms^{-1}) and smallest gap clearance by 20% (4 cm over 5 cm) when compared to the previous state of the art [34].

2.3.1.3 Related Videos

(V4) https://youtu.be/0G5iIF0pAlg

2.3.1.4 Related Software and Hardware

(S4) https://github.com/prgumd/MorphEyes

2.4 Hallucinated Activeness

In this section, I present the work on a non-obvious form of activeness: activeness by imagination or hallucinated activeness. Such a method is useful when a look-up map for activeness can be constructed, for e.g., a bird does not have to explore the around it's nest since it already knows how it looks which is imbibed in a mental picture. In this work, hallucinated activeness is manifested as visual saliency: the amount of time a human gaze would rest on different parts of the image.

2.4.1 Paper E: SalientDSO

(P5) **Nitin J. Sanket***, Huai-Jen Liang*, Cornelia Fermüller, Yiannis Aloimonos "SalientDSO: Bringing Attention to Direct Sparse Odometry", *IEEE Transactions on Automation Science and Engineering (T-ASE)*, (2019) Vol. 16, No. 4, pp. 1619-1626. DOI: http://dx.doi.org/10.1109/TASE.2019.2900980.

2.4.1.1 Brief Description

One of the most important competences required by an agent is the ability to navigate towards home. This is called the *homing problem* and is solved by using different approaches across the animal kingdom. While the smaller agents rely on direction vectors such as magnetic field and the sense of smell, the larger agents rely on perceptual information in the form of high level contextual cues. The larger agents like humans build a semantic map using a series of fixations (a heatmap of fixations is called visual saliency).

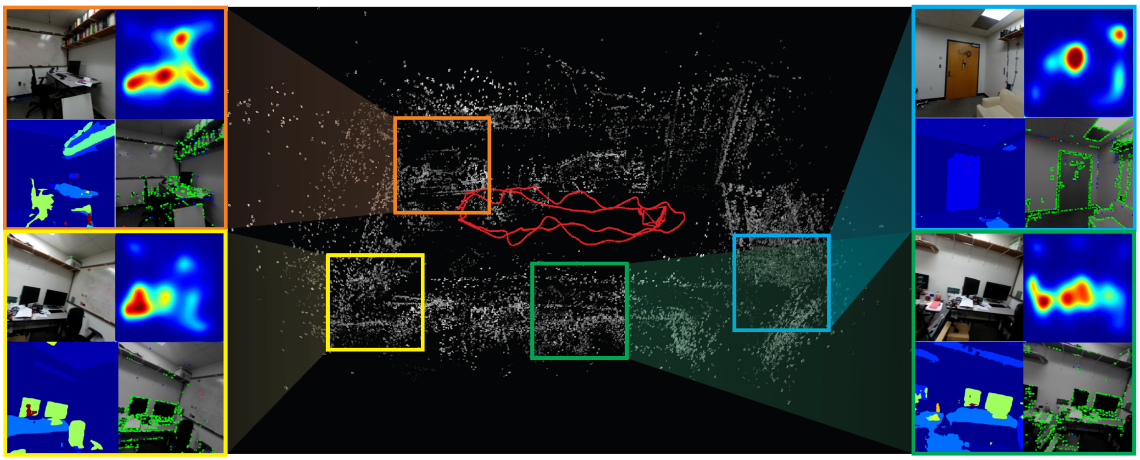


Figure 2.5: Sample point-cloud output of SalientDSO which does not have loop closure or global bundle adjustment. Each inset (color-coded to suite the respective location on the map) in clockwise direction from top left show the corresponding image, saliency, scene parsing outputs and active features. Observe that features from non-informative regions are almost removed approaching object-centric odometry.

Although, extensive studies have shown that fixations play a key role in human perception, our robots rarely use it for odometry/map building. This is partially because where to fixate is a cognitive problem and is generally achieved through a top down task planner which is generally very naive on a robot. Although, there have been works to segment the scene using a series of fixations, these fixations are not automatic. Recently, deep learning methods for obtaining high quality saliency maps have been constructed and form the basis of our work. We address the homing problem by increasing the robustness and accuracy of the visual odometry algorithm using visual saliency and scene parsing.

2.4.1.2 Comparison to the state of the art

The current state of the art in visual or visual inertial odometry deals with either improving the quality of features so that long term tracking of these features can be maintained despite changes to illumination, viewpoint and rotation. Lately, direct approaches with better optimization formulations have shown to be more accurate. To add further, event cameras have been used to enable odometry estimation with similar mathematical formulations at high speeds and high dynamic range scenes. In the last five years, deep learning based approaches for odometry estimation have struck the right balance between speed and accuracy but suffer from one severe limitation: they do not generalize to unseen data. To this end, our method combines best of both worlds, we use saliency and scene parsing from a neural network to select features for a classical direct method of odometry. Our method achieves about $5\times$ improvement in Root Mean Square error even with as low as 40 features.

2.4.1.3 Related Videos

(V5) https://youtu.be/EOLA1hSYaXU

2.4.1.4 Related Software and Hardware

(S5) https://github.com/prgumd/SalientDSO

2.5 Hardware and Software Co-design

In this section, I present the work on studying hardware and software which is essential since the active design philosophy is dictated by the amount of SWAP constraints on-board along with the agent's morphology

2.5.1 Paper F: PRGFlow

(P6) **Nitin J. Sanket**, Chahat Deep Singh, Cornelia Fermüller, Yiannis Aloimonos "PRGFlow: Unified SWAP-Aware Deep Global Optical Flow for Aerial Robot Navigation", *IET Electronics Letters*, 2021.

2.5.1.1 Brief Description

Since an agent has a multitude of competences with interlinks between them, it is intractable to analyse all the hardware and software combinations to achieve the same set of tasks with different methods. To make this problem tractable, we constrain some of the parameters such as the morphology of the robot, sensor suite and the task at hand. In particular, we analyse a quadrotor with on-board sensing and computation using a monocular camera, an inertial measurement unit and an altimeter source for the task of estimating ego-motion/odometry. We vary the software (neural network architectures) and hardware computational modules with different SWAP constraints. We analyse and report our findings on effects of variation in neural network architecture and hardware on metrics of speed, power and accuracy. We also address the question when should one use deep learning for such an approach. We believe this will be a stepping stone for

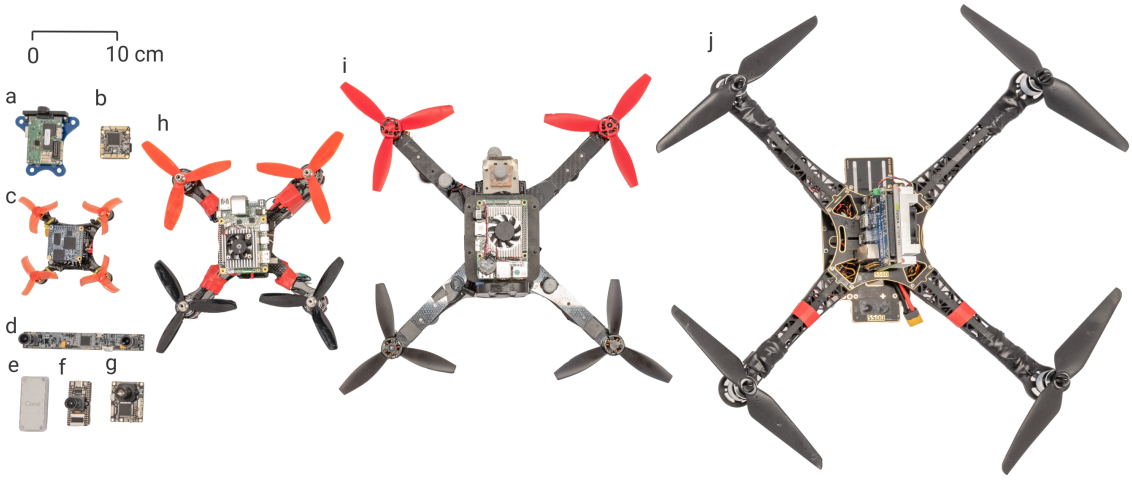


Figure 2.6: Size comparison of various components used on quadrotors. (a) Snapdragon Flight, (b) PixFalcon, (c) 120 mm quadrotor platform with NanoPi Neo Core 2, (d) MYNT EYE stereo camera, (e) Google Coral USB accelerator, (f) Sipeed Maix Bit, (g) PX4Flow, (h) 210 mm quadrotor platform with Coral Dev board, (i) 360 mm quadrotor platform with Intel® Up board, (j) 500 mm quadrotor platform with NVIDIA® Jetson™ TX2. Note that all components shown are to relative scale.

researchers and practitioners alike for studying hardware and software co-design.

2.5.1.2 Comparison to the state of the art

The current state of the art in visual or visual inertial odometry using deep learning do not generalize to novel scenes with good accuracy. Our work utilizes compositional networks to iteratively warp the image to regress the final warping parameters to maintain good accuracy when generalizing from training on simulation to the real world without fine-tuning or re-training. We observe that even simple dead-reakoning we obtain an RMSE of less than 3% of the trajectory length highlighting the robustness of the proposed *PRGFlow* approach. Our method can easily adapt to different SWAP constraints by using

network compression mechanisms with minimal effort. Our method is more robust and faster as compared to a traditional method when noise is present. Such a method is essential when using a low quality camera with varying/dynamic conditions. TO add further, our method can easily be adapted to work on event cameras as well by constructing event frames which posses high dynamic range and low motion blur.

2.5.1.3 Related Software and Hardware

(S6) https://github.com/prgumd/PRGFlow

2.6 Unrelated Contributions

During my Ph.D., I also co-authored three other papers (out of which two are under-review). These papers deal with the problem of segmentation of independently moving objects using event cameras and interactive perception (which takes active perception to the next level in hierarchy).

(U1) Chethan Mysore Parameshwara, **Nitin J. Sanket**, Chahat Deep Singh, Cornelia Fermüller, Yiannis Aloimonos "0-MMS: Zero-Shot Multi-Motion Segmentation With A Monocular Event Camera", *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

(U2) Chethan Mysore Parameshwara*, Simin Li*, Cornelia Fermüller, **Nitin J. Sanket**, Matthew S. Evanusa, Yiannis Aloimonos "SpikeMS: Deep Spiking Neural Network for Motion Segmentation", *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2021.

(U3) **Nitin J. Sanket***, Chahat Deep Singh*, Cornelia Fermüller, and Yiannis Aloimonos, "NudgeSeg: Zero-Shot Object Segmentation by Repeated Physical Interaction", *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2021.

Chapter 3: RoboBeeHive: Integration of Active Competences

3.1 Hierarchy of competences

We re-imagine each agent (UAV) as a series of competences which is based on the set of tasks it needs to achieve. One such example is shown in Fig. 3.1, where both single agent competences (blue) and multi-agent competences (green) are shown. The inner competences are required for outer ones and the speed at which these competences have to run decreases as their hierarchical level increases in a non-linear manner.

At the bottom of the hierarchy is the task of kinetic stabilization (ego-motion/odometry estimation) that deals with maintaining a stable pose of the UAV. This involves estimating the 6 DOF pose (position and orientation) by combining information from all the sensors on-board. The caveat here is that this has to be performed at a fast rate and has to be reasonably robust to changes in environmental conditions. Note that the rate at which this competency has to be performed increases with the decrease in UAV size [30].

Once the UAV is stable, it needs to analyze the scene for potential dynamic and static obstacles or interactions for which the scene needs to be segmented into safe and unsafe regions. In particular, we segment independently moving objects by "looking at" anomalies apropos to our own movement [35, 36] and static obstacles by looking at stacked optical flow magnitudes [34].

Next comes the ability to avoid static and/or dynamic obstacles. If a collision is unavoidable, *how should the UAV move or orient itself to minimize the damage?* is also a question covered by this competence. Interestingly, with decreased size (less than 80 mm) and weight (less than 50 g), the effect of the impact of a collision is minimal owing to a small change in momentum [30, 37], thereby making them safer. However, for aerial robots at high-speed, the velocity is large. Thereby making the change in momentum large even if the weight is small – this necessitates the need for high-speed robust obstacle avoidance.

The next competence is the ability to perform homing, i.e., to find a specific location in an environment. This involves maintaining a qualitative representation of a map so that the UAVs can return home. In the most basic form, one could just maintain a vector that points towards the home (homing vector). It is well studied that ants and some insects employ this strategy [38, 39, 40].

Next in the hierarchy comes the ability to land (on a static or a dynamic surface). This is required to dock onto a platform either for charging or safe landing [41, 42, 43, 44]. Landing on a dynamic surface could involve perching [45] onto a co-operative or adversarial agent which would involve different prediction and control strategies.

Further, the agent or the agent swarm has to pursue or escape from other agent(s). This involves the prediction of movements of the other agent(s) and online reactive control. [46, 47, 48]

Lastly, collaborative manipulation of objects is required to complete the navigation and interaction flight stack. [49, 50, 51]

Each of the aforementioned competences involves solving some form of an iterative

optimization problem which is slow and complex. However, in embodied AI, we use the 'active' nature of the quadrotor to bring in constraints to the optimization problem which were not present before. This makes the problem tractable to solve on the limited computing and memory budget available on the nano-quadrotor. Here, the activeness of a quadrotor means that it can perform maneuvers and control the image acquisition process, thus, introducing new constraints that did not exist before – this is called 'active' vision [52, 53, 54, 55, 56].

3.2 Motivation and Conceptualization of the RoboBeeHive

It is well known that the bees are responsible for one-third of our 'bites' because of their contribution to the pollination of flowers. However, the population of bees has been declining worldwide. While the experts on this topic work hard to find a solution and protect the bees, the possibility that the bees disappear in the near future cannot be discounted. Such an event would threaten humanity by considerably reducing the food supply. This contingency motivates our proposal. To this end and as a societal application of our research, we propose to design, develop and test the RoboBeeHive which will be a fully autonomous system of a swarm of nano-quadrotors to pollinate flowers. *Such a similar system can be easily adapted for finding survivors in a disaster scenario.* The RoboBeeHive (An artistic conceptualization is shown in Fig. 3.2), consists of a 'beehive charging station' housing of a large number of autonomous nano-quadrotors (pollination and explofration quadrotors). Each quadrotor is very small – still larger than a bee but small enough to emulate the bee behavior with regard to pollination.

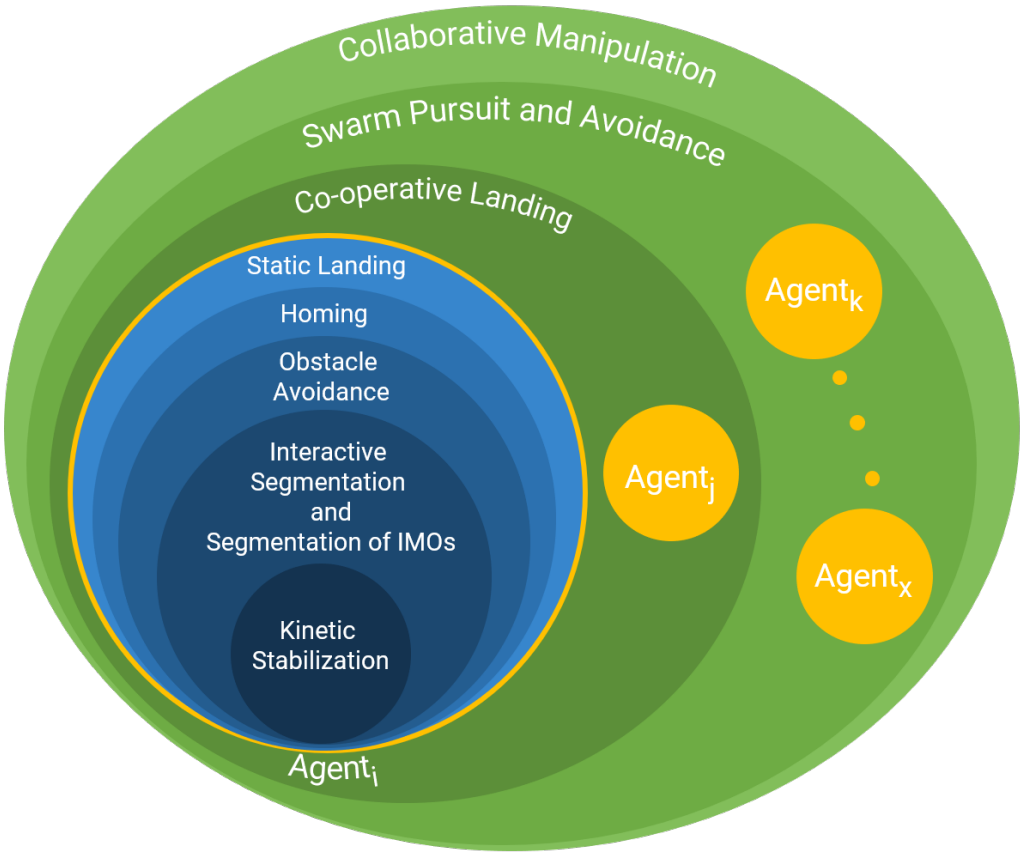


Figure 3.1: Proposed hierarchy of competences with the exterior ones needing the ones inside them, blue color indicates competences related to individual agents, green color indicates competences related to multiple agents and yellow bubbles show multiple agents.

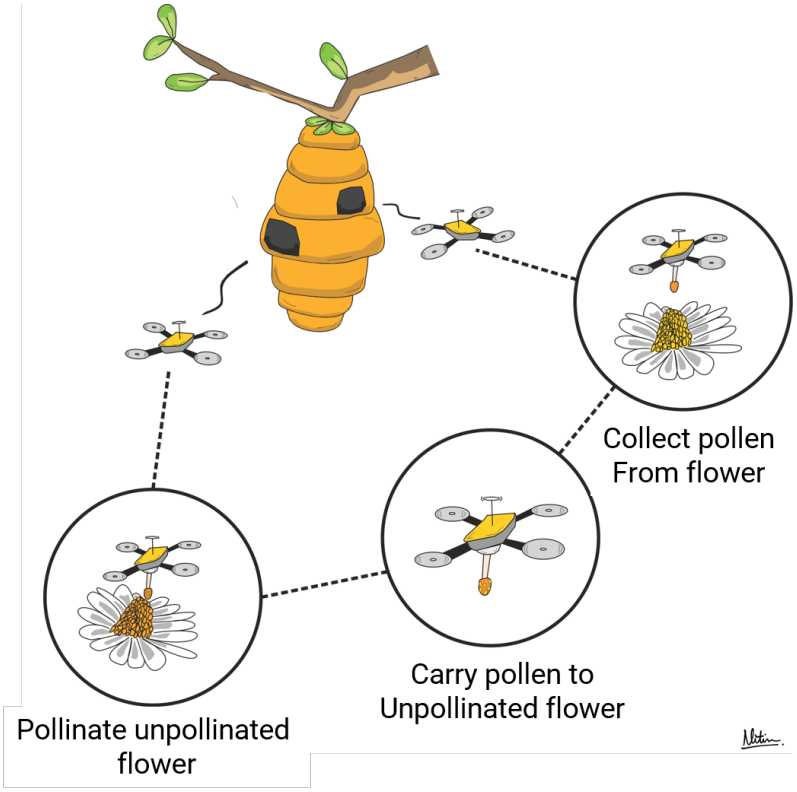


Figure 3.2: Illustration of the RoboBeeHive.

The ‘beehive charging station’ is itself a big UAV with a bee-hive like structure attached to it on the bottom (with all the pollination and exploration quadrotors inside). This ‘beehive charging station’ will navigate to the desired location using GPS. Then it will detect and perch onto a tree before deployment. After which the exploration quadrotors explore the farm and feed the data back to the bee-hive when they dock. During the charging phase, a homing vector algorithm is trained offline. This data is then fed to the pollination quadrotors along with the target sector or the area to pollinate which is coordinated by the bee-hive charging station. During the ‘pollination phase’, the pollination quadrotors identify and recognize each individual flower in its sector. Then they perform a soft landing procedure on the flower thereby ‘touching’ them and they repeat the process thus carrying pollen from flower to flower. This is performed whilst avoiding collision with other objects, humans, and animals. This application also demonstrates the final competence of Integration or switching between behaviors. It also demonstrates complex swarm behaviors.

3.3 Implementation Details

The RoboBeeHive prototype demonstration has ten steps.

1. The BeeHive drone finds and perches onto a tree branch (mimicked here by a horizontal PVC pipe).

2. The BeeHive drone switches off its motors to conserve energy and flower like mechanism holding the bee drones open-up.

3. The Bee drones take off one-by-one.

4. The Bee drones navigate to a target sector.

5. While navigating, they also avoid dynamic and static obstacles. The dynamic obstacle is mimicked by a GoGo Bird ornithocopter flying at the drone.

6. The bee drones find flowers.

7. They perform a soft-landing on a flower, thereby, "touching" them to gather pollen. They then navigate to a different flower and "touch" them to transfer the pollen.

8. The bee drones navigate back to the hive drone once they are done using a homing vector.

9. The bee drones land on the Hive drone.

10. The flower like mechanism closes and BeeHive drone starts its motors and undocks from the tree to navigate to the next location.

Each of the steps is described in context of which of the quadrotors (Bee or the BeeHive) perform the competence and are described next along with hardware details.

3.3.1 Bee Nano-quadrotors

The bee nano-quadrotors have an All Up Weight (AUW) of about 250 g and measures $150 \times 150 \times 120\times$mm (See Fig. 3.3) (To put this size in perspective, the world smallest 2D Lidar – Intel Realsense L515 weighs 100g and measures $61 \times 61 \times 26$mm which cannot be used on a drone this small due to weight restrictions and lack of data bandwith on computers of the size that would fit. Our bee nano-quadrotor is smaller

than the Samsung Note 20 Ultra smartphone in it's longest dimension and has a similar weight). It has a maximum thrust of about 1Kg, giving a thrust to weight ratio of 4:1. It has a RaspberryPi CM4 mated to a StereoPi v2 motherboard for computation along with a Coral TPU for running the neural networks. Finally, the sensor suite contains a downfacing TFMini Lidar for altitude measurements and a downfacing optical flow sensor that are used for holding position using the Flywoo Goku F745 AIO flight controller (See Fig. 3.3). The quadrotor also contains a frontfacing and a downfacing RGB camera for obstacle avoidance and flower detection. We are also currently working on a custom carrier and interface board for the CM4 which is much smaller than the StereoPiv2 and is shown in Fig. 3.5.

The different versions of the Bee nano-quadrotor are shown in Fig. 3.4. The first iteration used used 3 in. propellers mated to a T-Motor F20 II Motors along with a separate flight controller and ESC and a PX4Flow optical flow sensor and had an AUW of 350g. The next version used 2.5 in. propellers mated to T-Motor F1404 motors and same other avionics as before and had an AUW of 305g. The third version replaced the carbon fiber frame with a 3D printed one and saved another 10g for an AUW of 295g. The fourth version moved back to the carbon fiber frame for better rigidity and noise performance but replaced the flight controller and ESC to an all in one board for a total AUW of 265g. The current version replaced the Coral USB accelerator with a gutted version and a custom heatsink for a final AUW of 250g.

The flower detection is performed using a color segmentation using a Gaussian Mixture Model on the HSV color space along with morphological operations (Fig. 3.6). The control policy for flower landing involves aligning the flower center to the center of

the image and then decreasing altitude until a “touch” is detected by a change in IMU values.

The obstacle avoidance is performed by clustering deep optical flow to detect non-rigid parts of the scene (Fig. 3.7) that have a Focus of Expansion on the image plane which indicate a crash would happen. The control policy is similar to that in [57].

The homing vector is maintained as an integration of the optical flow values to home, which would be replaced by a compass outdoors.

Landing is performed using an April Tag fiducial marker on the Hive drone.

3.3.2 BeeHive Quadrotor

The hive quadrotor is built on a Catalyst Machineworks Tasmanian V2 frame which has a diagonal motor to motor distance of 950mm and has an AUW of about 6Kg with 4 bee nano-quadrotors. It has a maximum thrust of 22.4Kg, giving a thrust to weight ratio of 3.7:1. It has an NVIDIA TX2 for on-board computation and the same optical flow and laser altimeter as the bee nano-quadrotors that are used for holding position using the Pixhawk Cube Orange flight controller. An upfacing Intel Realsense D435i is used to detect and perch onto a pipe which mimics a tree branch. The bee quadrotors are housed in a flower petal like mechanism controlled by 4 servo motors (see Fig. 3.8). See Fig. 3.9 for a different versions of the BeeHive drone.

The pipe (tree branch analog) is detected using a simple depth clustering operation followed by a cylinder fitting to the RGB-D data obtained by the Intel RealSense (Fig. 3.10).

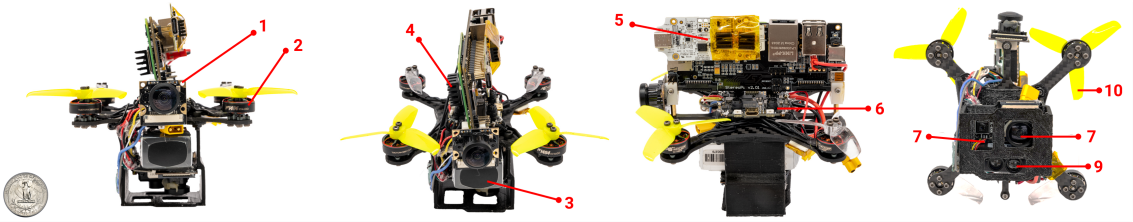


Figure 3.3: Different parts of the Bee nano-quadrotor. 1. Front facing RGB camera, 2. T-Motor F1404 Motors, 3. Tattu R-Line 3S 750mAh LiPo battery, 4. Raspberry Pi CM4 mated to a StereoPi v2 motherboard, 5. Gutted Google Coral USB Accelerator with custom heatsink, 6. Flywoo Goku F745 AIO Flight Controller and 4 in 1 ESC, 7. Downfacing RGB camera, 8. Optical flow sensor, 9. TFMini Lidar, 10. Gemfan 2540×3 propeller. Bottom left of the image shows a standard US quarter for scale reference.

The RoboBeeHive is reconfigurable to perform various tasks by changing the checking conditions in the sensorimotor loops with the same hierarchical software architecture.

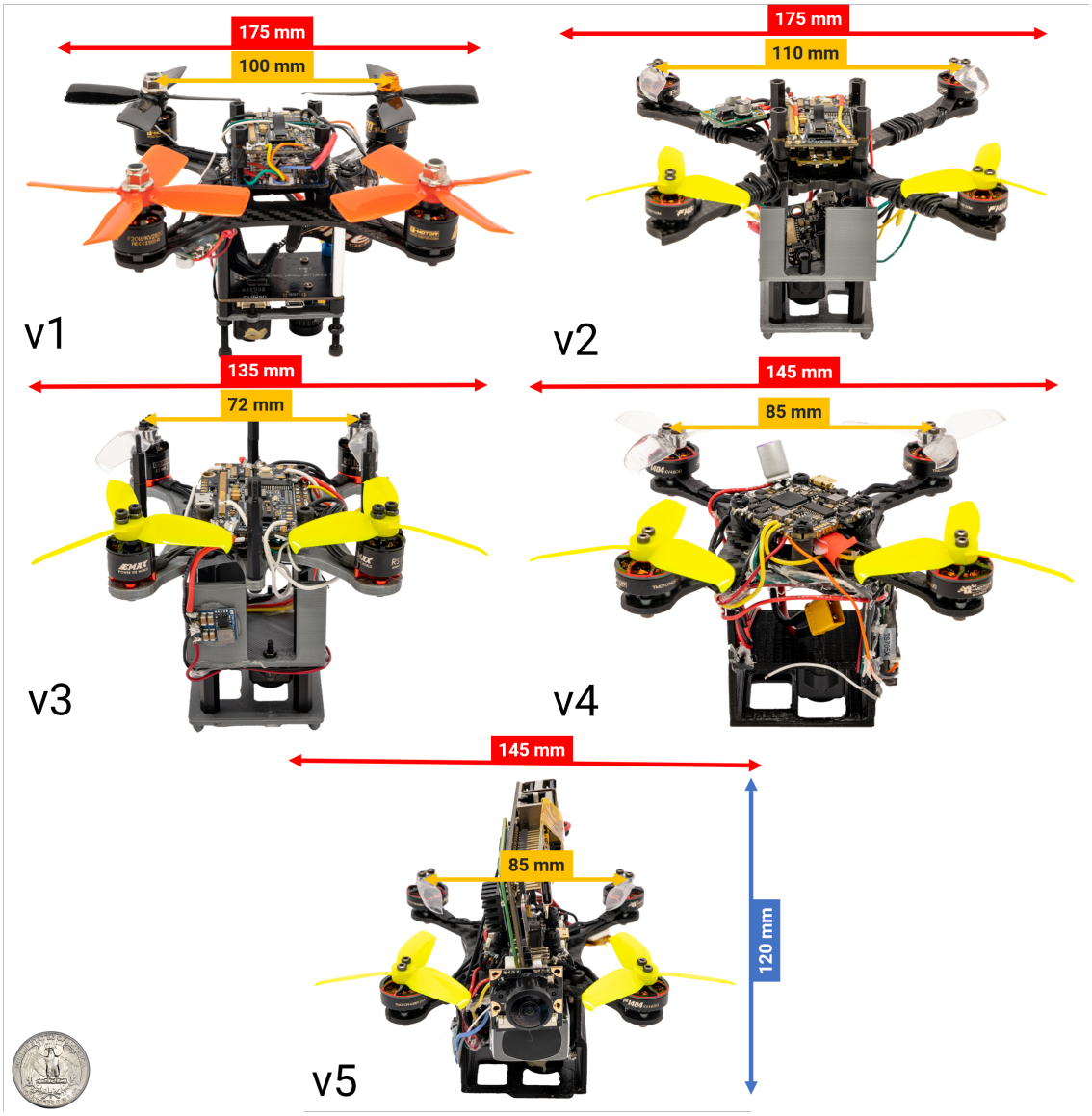


Figure 3.4: Different iterations of the Bee nano-quadrotor (number at the bottom left of each quadrotor shows the version number). Bottom left of the image shows a standard US quarter for scale reference.

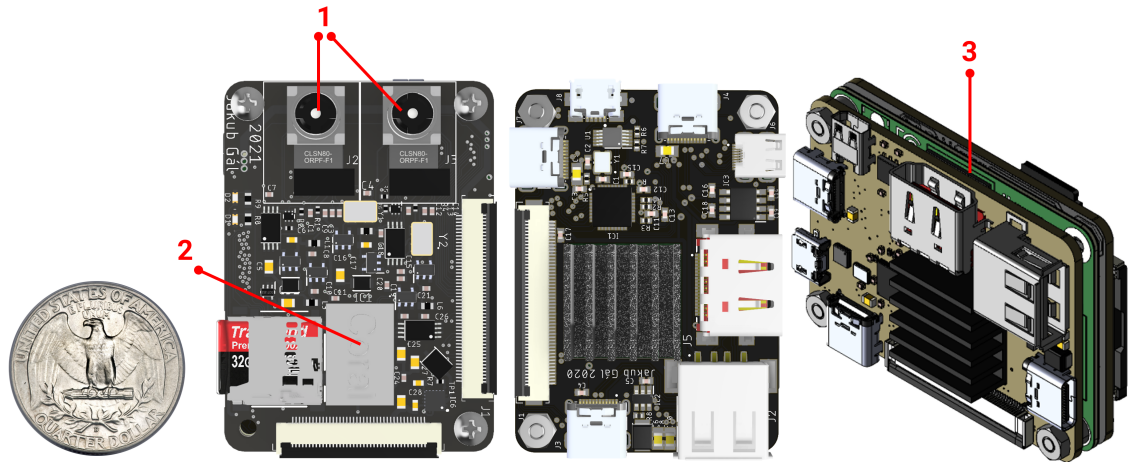


Figure 3.5: Left to right: Camera board, Interface board and RaspberryPi CM4. 1. Cameras, 2. Coral USB Accelerator attached to the PCIe port on the CM4, and 3. CM4 board sandwitched between the camera and interface board. Design based on https://grabcad.com/library/tpu-cam-with-cm4-1. Bottom left of the image shows a standard US quarter for scale reference.

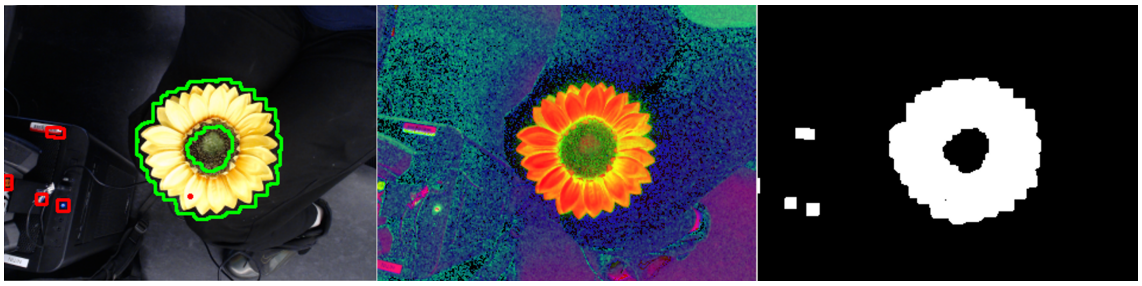


Figure 3.6: Left to right: Raw RGB image (green overlay shows the detected flower and red overlay shows removed false positives based on geometry), HSV representation of the RGB image , yellow color thresholded using the Gaussian Mixture Model.

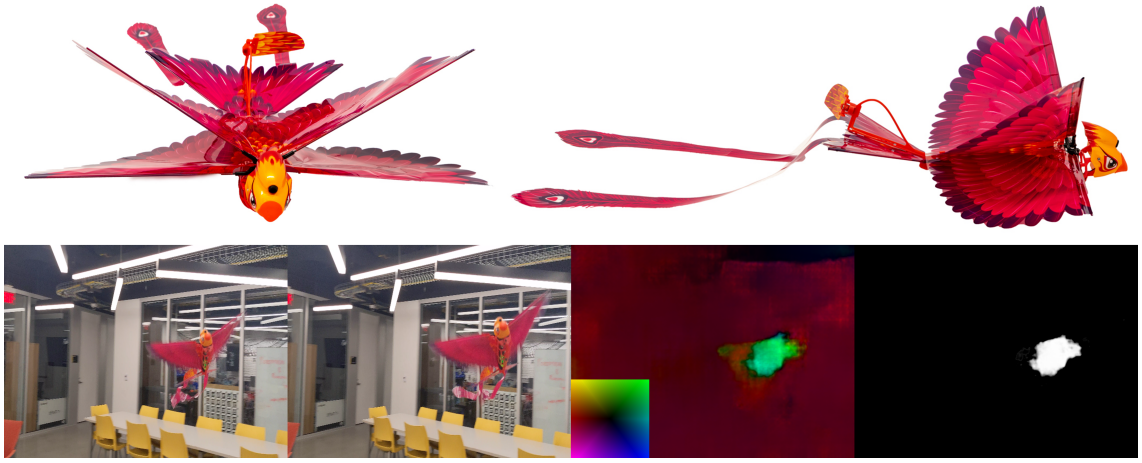


Figure 3.7: Top Row: Different views of the GoGoBird ornithocopter used as a dynamic obstacle. Bottom row (left to right): Consecutive RGB images taken from the front camera on the bee nano-quadrotor , optical flow color map and detected dynamic obstacle (Inset shows the color representation used, the hue of the color represents the direction and the saturation represents the magnitude). Notice how the optical flow colors of the dynamic obstacle and the background regions are different and are easily clustered.

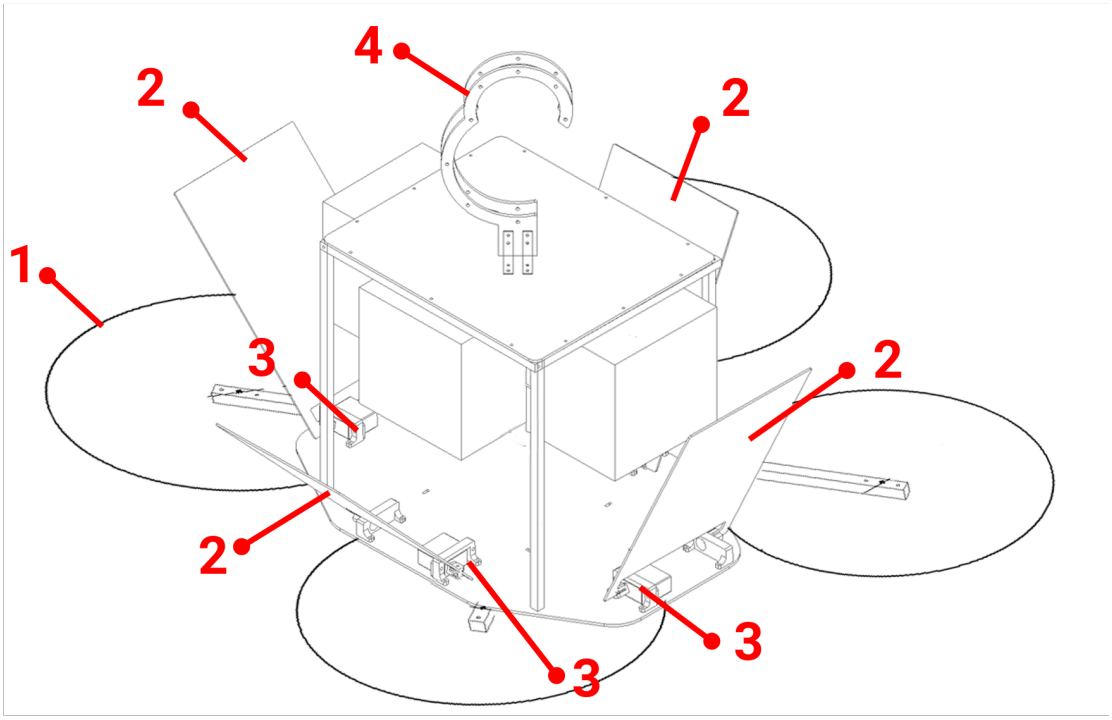


Figure 3.8: CAD Model of the BeeHive drone. 1. Propeller, 2. Flower petal flaps, 3. Flower petal servo motors, 4. Hook for perching.

Figure 3.9: Different iterations of the Hive drone (left to right show progression of versions). *Note that this image only shows the drone without the perching and bee holding flower mechanism which is under construction and was delayed due to COVID-19 causing machine shop closures and shipping delays.* Bottom left of the image shows a standard US quarter for scale reference.

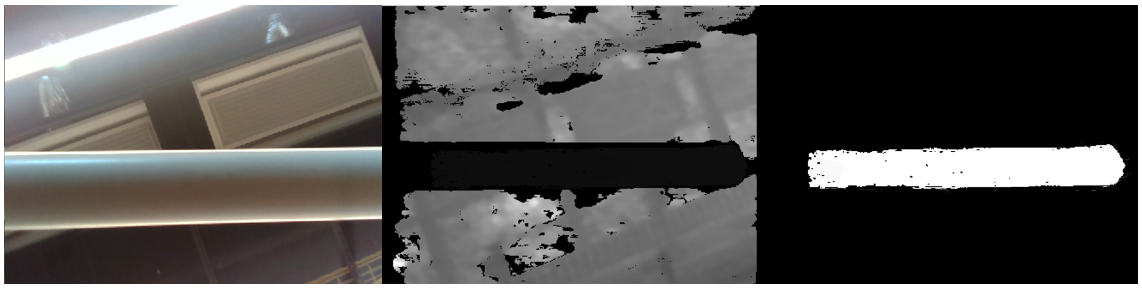


Figure 3.10: Left to right: RGB Image of the pole used for perching, Depth Image of the pole (brighter is farther), Mask of the segmented pole.

Chapter 4: Future Directions

When my doctoral work stated in 2016, drones or quadrotors were mostly seen as research platforms that needed extensive expertise and a large team to operate. The drones sold were limited in capabilities, mostly did not have safety features and were limited to manual line-of-sight operation. Since then, a large number of companies have made drones for photography or videography accessible at a low enough cost that they have become consumer commodities which can be readily bought at any electronics store just like a television. Also, hobbyist parts have grown in quality standards to those of major drone manufacturers. This coupled with the latest First Person View (FPV) drone racing motivated hobbyists all over the world to collaborate with researchers to build amazing open-source software and hardware for multiple flying platforms such as ArduPilot or PX4. This was also further made possible by hackable products to enable autonomy by simply adding sensors and computation on-board drones such as the Parrot® Bebop 2. Such drones were even used in the IROS 2018 competition[1]. Impressively, in the last two years we have also seen fully autonomous drones that use omni-directional cameras and on-board computation to track subjects while avoiding obstacles such as the Skydio™ 2. Recently, Skydio™ also announced Skydio 3D Scan™ which is a completely au-

[1] https://rise.skku.edu/iros2018racing/

tonomous solution to mapping a structure using active view planning where all the sensing and computation is done on-board. Furthermore, a myriad of open-source aerial robotics courses from across the best universities of the world have lowered the barrier of entry to into the field of aerial robotics research. Although, looking at the impressive live demonstrations from Skydio^{TM}, it might seem that the scope of research in aerial autonomy is limited, but there still exist a lot of open challenges that academia could address. One such example is to build autonomy is scales that are either too small or too large. The smaller drones suffer from lack of sensors and computation while the larger drones suffer from lack of agility. Moreover, homogeneous or heterogeneous swarm operation in unstructured environments is still an big open-challenge.

In the next section, I will discuss the main limitations of the approaches proposed in this thesis and further in the subsequent section, I will also provide a few possible future directions to address these problems.

4.1 Limitations of Proposed Approaches

GapFlyt

In GapFlyt, we could navigate through static gaps of unknown shape, size and location. However, it relied on a deep learning based optical flow for the detection of the gap which inherently relies on features in the scene and the training data. This could be improved by generating training data in a way that the network is almost dataset agnostic to enable operation in the wild. GapFlyt also could not estimate the size of the gap, thereby giving no guarantees in regard to the quadrotor being able to fit through the gap.

This could be solved by either fusing information from the inertial sensor or using a single beam lidar or utilizing a model of the quadrotor's movement to obtain metric scale. Finally, GapFlyt could not deal with dynamic gaps because the entire agent had to move to obtain more information. This could be solved by just moving a part of the agent.

EVDodgeNet

In EVDodgeNet, we could dodge obstacles of unknown shape and location at a hover or near-hover situation using event cameras. However, when the motion was larger, our method struggled and this could be trackled by estimating a full 3D pose instead of homography in the network. There is room for research in how to make such a 3D pose network generalize to novel scenes trained in simulation without fine-tuning or re-training. EVDodgeNet also did not guarantee no collision for an unknown sized object, this could be solved by utilizing a stereo camera as described by [58].

EVPropNet

In EVPropNet, we detected unmarked drones by finding propellers and showed applications of mid-air docking and following. The fundamental assumption we made was that the propeller speeds are much larger than that of the scene, which is generally true but would fail when flying speeds more than 10m/s^{-1} on drone propellers larger than 10 inches (as they spin slower). During such a scenario, a combination of the method used in EVDodgeNet and EVPropNet would make the system more robust. We also observed that the accuracy of detection of the propeller varied with the propeller size on the image

with accuracy dropping for smaller sizes due to lack of data and a large number of false positives for larger sizes. To tackle this problem, one can utilize an active pan-tilt and zoom gimbal to control the camera to obtain optimal accuracy.

MorphEyes

MorphEyes presented the first work on utilizing a variable baseline stereo system on a quadrotor for enabling lower error in depth estimation depending on the task. However, such a system could also be used to gather more information to further simplify perception problems. For e.g., one could move the cameras normal to the scene in question to obtain boundaries. This could be achieved by utilizing a stereo gimbal system with pan, tilt and zoom. A tighter coupling between perception and planning would also enable perception at even lower SWAP constraints. Furthermore, information using depth of field could enable even faster computation.

SalientDSO

In SalientDSO, we demonstrated that utilizing saliency to obtain features improved the accuracy and reliability (lower number of optimization failures) over standard sampling methods in a direct odometry framework. To add further, even with lower number of features SalientDSO performed better than the state of the art at the time. However, SalientDSO only computed saliency at keyframes and did not propagate information. It also did not combine saliency and scene parsing information from multiple frames. Such an approach would drastically increase the accuracy further.

PRGFlow

PRGFlow presented a method to obtain odometry using a monocular downfacing camera, an altimeter source and an inertial measurement unit using a deep neural network. The main idea of the work is to present a method to evaluate software and hardware co-design and to serve as a blueprint for researchers and practitioners alike for designing robots. However, this work did not talk about modelling problems that reuse either software or hardware to achieve a set of tasks. This would be a new emerging area where a lot of work needs to be done utilizing the expertise of multiple domains such as computer science, computer architecture and electrical engineering.

4.2 Future Work

Formalism on Active Perception

Although, there has been a significant work on active perception, they are mostly scattered with the theme only being at the philosophical level: to combine perception, planning and control into a single entity. There is a fundamental mathematical formulation that is missing. Such a formulation would not only accelerate research in this area, it would also make it easier for the newcomers in the field to 'see' the advantages and limitations in a formal manner. One could start by modelling the agent's movements, constrain it with kinodynamic constraints of the agent which should give rise to conditions where perception would be easier and how the agent should be controlled to do so. This also entails modelling the robot's perception engine utilizing the advances in both computer

vision and computer graphics fields. Such a philosophy would also enable building *smart sensors* that could perform near-sensor processing for the required tasks. Finally, this might give rise to a 'silver bullet' representation of visual data to perform a set of related tasks. Another interesting topic is how one could *conceal the active vision exploratory movements in the movements used for non-verbal ques for human-robot interaction*. Such a system would lead to an organic experience for the user, which is more robust, highly bio-inspired and would make *robots more approachable and understandable*. For e.g., if a robot is unsure, it can nod its head to show this uncertainty while also collecting more data samples to decrease this uncertainty.

Interactive Perception

In a lot of cases simply moving one's own body or a part of the body to obtain more information is not enough to solve the task at hand. For e.g., it might not be possible to segment a cluttered scene without decluttering the scene. In such a scenario, one needs to physically interact with the scene either by poking, nudging or grasping to gather more information to simplify the perception problem at hand. Interactive perception or perception by physical interaction is the logical evolution to active perception and is still at a nascent stage. Again, we would need a formalism to help researchers build algorithms on the theoretical foundations of this powerful concept.

Hardware and Software Co-design

Although there have been several attempts to build hardware and software to enable autonomy at extremely limited SWAP constraints but the works are generally spread across multiple domains of electrical engineering and computer science. This makes it hard for people to adapt such research ideas into their own projects without significant learning curve or budget or prior experience. One of the major deterrents for the development of the field of hardware and software co-design is the lack of tools to study combined complexity of such a system. Although, this thesis proposed a simple idea to evaluate such tradeoffs for one single problem, it is far from sufficient to perform such an analysis for a large agent with multiple compute modules and a multitude of sensors. The problem becomes even more interesting when the agent morphology can change along with utilizing its active and interactive abilities.

Formalism on Agent Morphology

Since both active and interactive perception depend on the morphology of the agent, a nascent research area would be to define optimization problems which involve not only the task at hand but also the morphology of the agent. In particular, complaint designs which are inherently robust to collisions can also be used in an active or interactive manner either to gather more information to re-use parts of the agent's body to perform various tasks. Such a design method would make the robots lighter and more efficient by reducing the number of parts required to build them. To add further, task planners which involve changes to morphology on-the-fly needs to be developed. Morphing designs would open

doors to novel applications which were not possible from fixed designs and they would in-general be more robust. Finally, fault-tolerant control methods need to be developed for such mechanisms.

Role of Memory in Activeness

Since, passiveness takes over activeness as the amount of prior information increases, it is unclear how memory would interplay with the activeness of the agent. Some of the questions are: Does it depend on the SWAP constraints? Is it better to learn a specialized representation if the task has to be performed repeatedly? How should one decide to store something in memory rather than attempt the task actively? It is also unclear how to bring more constraints to active vision with prior information from memory.

Active and Interactive Perception with multiple agents

Another interesting area to be explored is how activeness can be used to make a homogeneous/heterogeneous swarm smarter. One could imagine a swarm of robots with different sensor suites performing maneuvers to acquire information and share amongst each other to solve a complex set of tasks. Interaction with the scene or amongst agents is also an interesting area with minimal amount of exploration.

Appendix A: GapFlyt

Nitin J. Sanket*, Chahat Deep Singh*, Kanishka Ganguly, Cornelia Fermüller, Yiannis Aloimonos "GapFlyt: Active Vision Based Minimalist Structure-Less Gap Detection For Quadrotor Flight", *IEEE Robotics and Automation Letters (RA-L)*, (2018) Vol. 3, No. 43847–3854. DOI: 10.1109/LRA.2018.2843445.

GapFlyt: Active Vision Based Minimalist Structure-less Gap Detection For Quadrotor Flight

Nitin J. Sanket*, Chahat Deep Singh*, Kanishka Ganguly,

Cornelia Fermüller, Yiannis Aloimonos

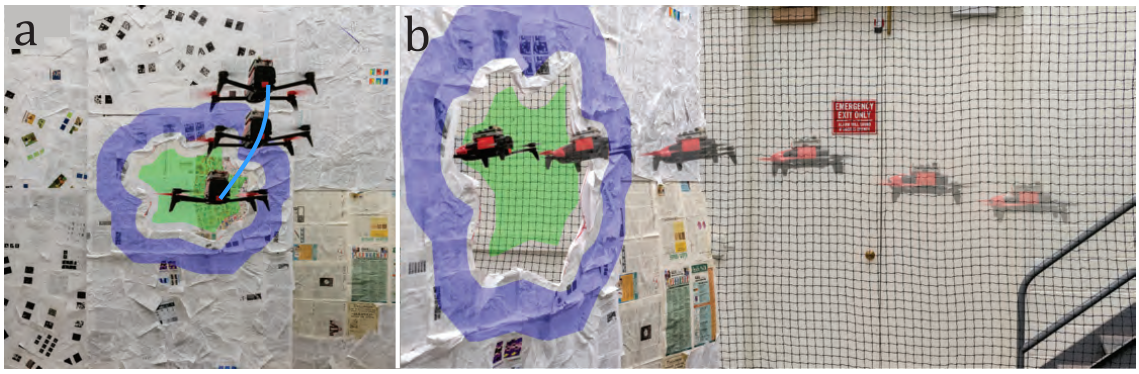


Figure A.1: Different parts of the pipeline: (a) Detection of the unknown gap using active vision and TS^{2}P algorithm (cyan highlight shows the path followed for obtaining multiple images for detection), (b) Sequence of quadrotor passing through the unknown gap using visual servoing based control. The blue and green highlights represent the tracked foreground and background regions respectively. Best viewed in color.

Abstract — Although quadrotors, and aerial robots in general, are inherently active agents, their perceptual capabilities in literature so far have been mostly passive in nature. Researchers and practitioners today use traditional computer vision algorithms with the aim of building a representation of general applicability: a 3D reconstruction of the scene. Using this representation, planning tasks are constructed and accomplished to allow the quadrotor to demonstrate autonomous behavior. These methods are inefficient as they are not task driven and such methodologies are not utilized by flying insects and birds. Such

agents have been solving the problem of navigation and complex control for ages without the need to build a 3D map and are highly task driven.

In this paper, we propose this framework of bio-inspired perceptual design for quadrotors. We use this philosophy to design a minimalist sensori-motor framework for a quadrotor to fly though unknown gaps without an explicit 3D reconstruction of the scene using only a monocular camera and onboard sensing. We successfully evaluate and demonstrate the proposed approach in many real-world experiments with different settings and window shapes, achieving a success rate of 85% at 2.5ms^{-1} even with a minimum tolerance of just 5cm. To our knowledge, this is the first paper which addresses the problem of gap detection of an unknown shape and location with a monocular camera and onboard sensing.

A.1 Supplementary Material

The supplementary hardware tutorial, appendix, code and video are available at `prg.cs.umd.edu/GapFlyt.html`.

A.2 Introduction and Philosophy

The quest to develop intelligent and autonomous quadrotors [59, 60, 61] has taken a center stage in the recent years due to their usefulness in aiding safety and intuitive control. To achieve any form of autonomy, quadrotors need to have perceptual capabilities in order to sense the world and react accordingly. A generic and fairly common solution to providing perceptual capabilities is to acquire a 3D model of its environment. Creating

Table A.1: Minimalist design of autonomous quadrotor (drone) behaviours.

Competence	Passive Approach	Active and Task-based Approach
Kinetic stabilization	Optimization of optical flow fields	Sensor fusion between optical flow and IMU measurements
Obstacle avoidance	Obtain 3D model and plan accordingly	Obtain flow fields and extract relevant information from them
Segmentation of independently moving objects	Optimization of flow fields	Fixation and tracking allows detection
Homing	Application of SLAM	Learn paths to home from many locations
Landing	Reconstruct 3D model and plan accordingly	Perform servoing of landing area and plan appropriate policy
Pursuit and Avoidance	Reconstruct 3D model and plan accordingly	Track while in motion
Integration: Switching between behaviors	Easy: The planner interacts with the 3D model	Hard: An attention mechanism on ideas switching between behaviors

such a model is very useful because of its general applicability – one could accomplish many tasks using the same model. The process of obtaining a 3D model of the environment using onboard sensing and a myriad of different sensors has gained momentum in the last few years [62]. Sensors like the LIDAR, RGB-D and stereo camera cannot be used on a small quadrotor due to their size, weight, and power limitations. This constrains us to a monocular camera along with the already present onboard inertial sensor (IMU) and many algorithms have been developed based on the same principle [63][64].

Instead of attempting to recover a 3D model of the scene, we want to recover a "minimal" amount of information that is sufficient to complete the task under consideration. We conceptualize an autonomous quadrotor as a collection of processes that allow the agent to perform a number of behaviors (or tasks) (Table A.1). At the bottom of the hierarchy is the task of kinetic stabilization (or egomotion). Next comes the ability for obstacle avoidance, where the obstacles could be static or dynamic, followed by ability to perform homing, i.e., to find a specific location in an environment. Last in the hierarchy comes the ability to land (on a static or a dynamic surface) and the ability to pursue, or escape from, another agent. This hierarchy of competences, with each competence needing the one before it, constitutes the sensorimotor system of the quadrotor agent. These behaviors can be accomplished without an explicit 3D reconstruction because of the "active" nature of the quadrotor. The quadrotor can perform maneuvers and control the image

acquisition process, thus introducing new constraints that were not there before - this is called "active" vision [52, 53, 54]. This methodology was inspired by the fruit fly [65]. Prior research has shown that fruit flies, and other insects [66] [67], do not perceive depth directly. It is achieved by a balance of active and exploratory procedures. In this paper, we focus on the second competence of obstacle avoidance. Specifically, the question this paper deals with is: *"Can a quadrotor manage to go through an arbitrarily shaped gap without building an explicit 3D model of a scene, using only a monocular camera?"* We develop the visual mathematics of the solution, evaluate and demonstrate the proposed approach in many real experiments with different settings and window shapes.

Traditional computer vision based approaches such as sparse or dense reconstruction [68, 69, 70, 71] have been used to obtain a 3D structure of the scene over which sophisticated path planners have been used to plan collision free paths. Lately, deep-learning driven approaches have taken a center stage in solving the problem of fast collision avoidance and safe planning on quadrotors [72]. Most of these neural network approaches compute Fast Fourier Transforms (FFTs) for large filter sizes [73]. Such approaches can be processed on a Field-Programmable Gate Array (FPGA), rather than a Graphical Processing Unit (GPU) to drastically improve the computation performance and power efficiency [74][75].

To our knowledge, this is the first paper which addresses the problem of gap detection with a monocular camera and onboard sensing. However, the problem of going through gaps has fascinated researchers from many years and some of the recent works which present algorithms for planning and control can be found in [76] [77]. Some of the works which paved way to the bio-inspired approach used in this paper can be found in

[78, 79, 80, 81].

The key contributions of this paper are given below:

- Active vision based structure-less minimalist gap detection algorithm – Temporally Stacked Spatial Parallax or TS^{2}P (Fig. A.1a).
- Visual servoing on a contour for the quadrotor to fly through unknown gaps (Fig. A.1b).

A.2.1 Organization of the paper:

Sec. A.3 presents the detection algorithm of a gap in an arbitrary shaped window using Deep Learning based optical flow *and the role of active vision in such algorithms.* Sec. A.4 describes the algorithm used for tracking the gap/safe point and the quadrotor control policy. Sec. A.5 illustrates the experimental setup and provides error analyses and comparisons with traditional approaches. We finally conclude the paper in Sec. A.6 with parting thoughts on future work.

A.2.2 Problem Formulation and Proposed Solutions

A quadrotor is present in a static scene (Fig. A.2), where the absolute depth of each point as 'seen' from the camera can be modelled as an univariate bimodal gaussian distribution. The location at which there is a maximum spatial depth discrepancy between pixels (projected point) is defined as the Contour ($\mathcal{C}$) of the opening. In this paper, the words boundary, gap, window or opening refer to the same entity and will be used interchangeably. Any pixel close to the mode corresponding to the lower depth value is

defined as the Foreground ($\mathcal{F}$) and similarly that corresponding to the higher depth value is defined as the Background ($\mathcal{B}$). A simple way of solving the problem of finding the gap for a near fronto-parallel pose is to find the depth discrepancy which is a trivial clustering problem when the depth is known. The depth can be known if multiple calibrated cameras are used or the entire scene is reconstructed in 3D. These methods are computationally expensive [62]. In this paper, we propose a 'minimalist' solution to find any arbitrary shaped gap for the quadrotor to go through using Temporally Stacked Spatial Parallax (TS^{2}P) algorithm. A control policy based on contour visual servoing is used to fly through unknown gaps.

A.3 Gap Detection using TS^{2}P

Before we explain the procedure for gap detection, we need to formally define the notation used. $({}^aX_b, {}^aY_b, {}^aZ_b)$ denotes the coordinate frame of b represented in the ref-

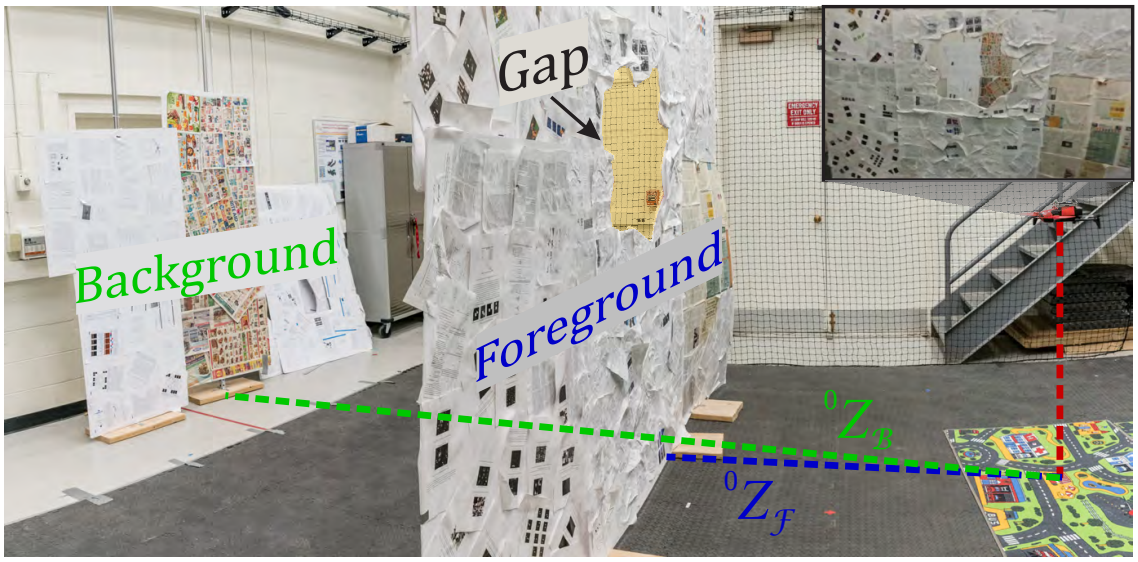


Figure A.2: Components of the environment. On-set Image: Quadrotor view of the scene.

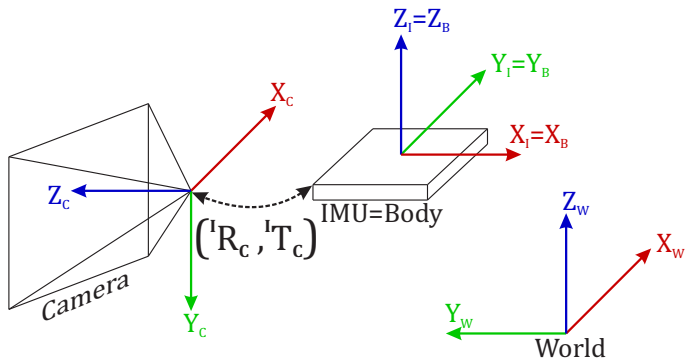


Figure A.3: Representation of co-ordinate frames.

erence of a. The letters I, C, B and W are used as sub/superscript to denote quantities related to Inertial Measurement Unit (IMU), Camera, Body and World respectively. If a sub/superscript is omitted, the quantity can be assumed to be in W. C and I are assumed to be rigidly attached to each other with known intrinsics and extrinsics. B is defined to be aligned with I (Fig. A.3). A pinhole camera model is used for the formation of the image. The world point $\mathbf{X}$ gets projected onto the image plane point $\mathbf{x}$.

The camera captures images/frames at a certain frequency. Let the frame at time t^i be denoted as $\mathbb{F}^i$ and is called the i^{th} frame. Optical flow [82] between i^{th} and j^{th} frames is denoted by ${}^{j}_{i}\dot{p}_{\mathbf{x}}$ which is a sum of both translational $\left({}^{j}_{i}\dot{p}_{\mathbf{x},T}\right)$ and rotational $\left({}^{j}_{i}\dot{p}_{\mathbf{x},R}\right)$ components and are given by:

$$ {}^{j}_{i}\dot{p}_{\mathbf{x},T} = \frac{1}{Z_{\mathbf{x}}} \begin{bmatrix} xV_z - V_x \\ yV_z - V_y \end{bmatrix} \; ; \; {}^{j}_{i}\dot{p}_{\mathbf{x},R} = \begin{bmatrix} xy & -(1+x^2) & y \\ (1+y^2) & -xy & -x \end{bmatrix} \Omega $$

where $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^T$ denotes the pixel coordinates in the image plane, $Z_{\mathbf{x}}$ is the

depth of $\mathbf{X}$ (world point corresponding to pixel $\mathbf{x}$), $V = \begin{bmatrix} V_x & V_y & V_z \end{bmatrix}^T$ and Ω are the linear and angular velocities of the C in W between t^i and t^j respectively.

The otherwise complex depth estimation problem can be trivialized by moving in a certain way[52]. These "active" vision principles dictate us to control the quadrotor's movements so as to make the interpretation of optical flow easier. Since the quadrotor is a differentially flat system [83], the rotation about (X_B, Y_B, Z_B) or roll, pitch and yaw can be decoupled. As an implication, the movements can be controlled in a way to simplify the depth $(Z_{\mathbf{x}})$ estimation problem. The quadrotor is controlled in such a way that $\Omega \approx 0$, $V_z \ll V_x$ and $V_z \ll V_y$, then the optical flow can be modelled as:

$$ {}^j_i\dot{p}_{\mathbf{x}} = {}^j_i Z_{\mathbf{x}}{}^{-1} \begin{bmatrix} -{}^j_i V_x & -{}^j_i V_y \end{bmatrix}^T + \eta $$

where η is the approximation error. This shows that using the aforementioned "active" control strategy, we obtain an implicit 3D structure of the environment in the optical flow. The inverse depth in this "controlled" case manifests as a linear function of the optical flow.

The optical flow equation can be written for both foreground $(\mathcal{F})$ and background $(\mathcal{B})$ pixels independently. The magnitude of optical flow for $\mathcal{F}$ is given by,

$$ \left\| {}^j_i\dot{p}_{\mathbf{x},\mathcal{F}} \right\|_2 = {}^j_i Z_{\mathbf{x},\mathcal{F}}{}^{-1} \sqrt{{}^j_i V_x^2 + {}^j_i V_y^2} + \nu $$

where $\nu \sim \mathcal{N}(0, \sigma)$ is assumed to be an additive white Gaussian noise and is independent of the scene structure or the amount of camera movement between two frames (V, Ω).

For such assumptions to be valid, the optical flow algorithm needs to work well for a wide range of camera movements in a variety of scene structures. Using fast traditional optical flow formulations based on methods like [84] or [85] voids such assumptions. This motivated us to use deep-learning based flow algorithms which excel at this task while maintaining a reasonable speed when running on a GPU. In this paper, FlowNet2 [86] is used to compute optical flow unless stated otherwise.

A simple method to reduce noise is to compute the mean of the flow magnitudes across a few frames. Let $\xi = \{\mathbb{F}^j, ..., \mathbb{F}^k\}$ be a set of N frames from which the optical flow is computed with respect to some reference frame $\mathbb{F}^i$ where the complete gap is assumed to be visible. Here, $N = \bar{\bar{\xi}}$ is the cardinality of the set ξ. The mean flow magnitude at $\mathbf{x}$ for $\mathcal{F}\left(\left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{F}}\right\|_2\right)$ and $\mathcal{B}\left(\left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{B}}\right\|_2\right)$ is given by,

$$\left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{F}/\mathcal{B}}\right\|_2 = \left(NZ_{\mathbf{x},\mathcal{F}/\mathcal{B}}\right)^{-1}\sum_{r=j}^{k}{}_i^{r}V + \nu'$$

where $\nu' \sim \mathcal{N}\left(0, N^{-0.5}\sigma\right)$ and $V = \sqrt{V_x^2 + V_y^2}$. Clearly, the noise varies inversely with N.

Since $Z_{\mathbf{x},\mathcal{F}} < Z_{\mathbf{x},\mathcal{B}}$, we can say that $\left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{F}}\right\|_2 > \left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{B}}\right\|_2$. Now, $\left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{F}}\right\|_2 - \left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{B}}\right\|_2 \geq \tau$ can be used as a criterion for finding possible boundary regions. It was found experimentally that using inverse flow differences $\left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{B}}\right\|_2^{-1} - \left\|{}_i^{\xi}\dot{p}_{\mathbf{x},\mathcal{F}}\right\|_2^{-1} \geq \tau'$ gave better numerical stability and better noise performance due to the scale compression by the inverse function. This is inherently the spatial derivative of inverse average (stacked) flow magnitudes and it can be written as $\Xi = \nabla \cdot \left\|{}_i^{\xi}\dot{p}_{\mathbf{x}}\right\|_2^{-1}$, where, $\nabla = \begin{bmatrix}\partial/\partial x & \partial/\partial y\end{bmatrix}^T$. Note that this is the same as solving the edge detection problem in computer vision and

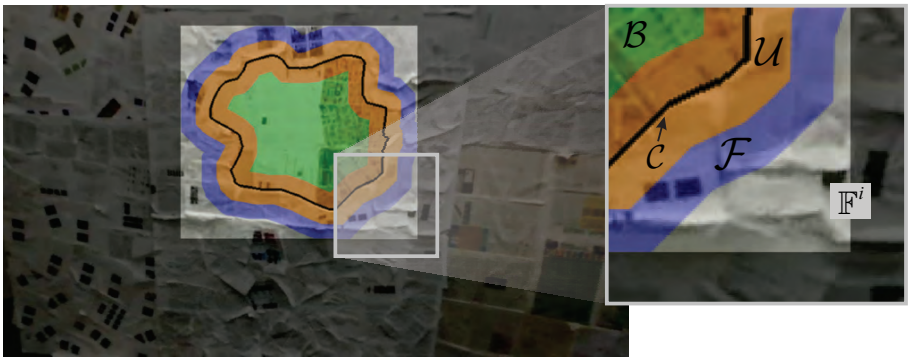


Figure A.4: Label sets used in tracking. *(blue: foreground region, green: background region, orange: uncertainty region, black line: contour, brighter part of frame: active region and darker part of frame: inactive region.)*

any kernel or method like the Sobel operator or the Canny edge detection algorithm can be used.

A.4 High Speed Gap Tracking For Visual Servoing Based Control

This section presents a targeted solution for tracking a contour using label sets propagated using Focus of Expansion (FOE) constraints. A pixel at location $\mathbf{x}$ is associated with a score $\chi(\mathbf{x}) \in [-1, 1]$ which denotes its score as foreground or background. The foreground and background pixel locations are defined by $\mathcal{F} = \{\mathbf{x}|\chi(\mathbf{x}) = +1\}$ and $\mathcal{B} = \{\mathbf{x}|\chi(\mathbf{x}) = -1\}$ respectively.

We define the opening $\mathcal{O}$ on the image plane as $\mathcal{O} = \{\mathbf{x}|\chi(\mathbf{x}) < 0\}$. The pixel locations which cannot be perfectly classified as foreground or background belong to the uncertainty zone and are defined as $\mathcal{U} = \{\mathbf{x}|\chi(\mathbf{x}) \in (-1, +1)\}$. The contour location is defined as $\mathcal{C} = \{\mathbf{x}|\chi(\mathbf{x}) = 0\}$. Fig. A.4 gives a visual representation of the different sets on a sample image.

The problem statement dictates the tracking of contour location $\mathcal{C}$ across time. This

Algorithm 1: Label Set Propagation using FOE constraints.

Data: $\mathbb{C}^i_{\mathcal{F}}, \mathbb{C}^i_{\mathcal{B}}, \mathbb{F}^i, \mathbb{F}^j, \mathcal{F}^i, \mathcal{B}^i$
Result: $\mathbb{C}^j_{\mathcal{F}}, \mathbb{C}^j_{\mathcal{B}}, \mathcal{F}^j, \mathcal{B}^j$

1 $\mathbb{C}^j_{\mathcal{F}}$ = FeatureTracker$(\mathbb{C}^i_{\mathcal{F}}, \mathbb{F}^i, \mathbb{F}^j)$;

2 $A = \begin{bmatrix} \mathbb{C}^i_{\mathcal{F},x} & -\mathbb{1} \end{bmatrix}$;

3 $B = \begin{bmatrix} \mathbb{C}^j_{\mathcal{F},x} - \mathbb{C}^i_{\mathcal{F},x} \end{bmatrix}$;

4 $\begin{bmatrix} \alpha \\ \beta \end{bmatrix} = A^{\dagger}B$;

5 $x_{0,\mathcal{F}} = {}^{\beta}/_{\alpha}; \qquad y_{0,\mathcal{F}} = \left\langle \mathbb{C}^j_{\mathcal{F},y} \frac{1}{\alpha} \left(\mathbb{C}^j_{\mathcal{F},y} - \mathbb{C}^i_{\mathcal{F},y} \right) \right\rangle$;

6 $\mathcal{F}^j = \alpha \begin{bmatrix} \mathcal{F}^i_x - x_{0,\mathcal{F}} \\ \mathcal{F}^i_y - y_{0,\mathcal{F}} \end{bmatrix} + \begin{bmatrix} \mathcal{F}^i_x \\ \mathcal{F}^i_y \end{bmatrix}$;

7 Repeat steps 1 through 6 for $\mathcal{B}$ in parallel;

problem is hard as the contour tracking relies on updating $\chi(\mathbf{x}) \; \forall \, \mathbf{x} \in \mathcal{U}$ over time which is non-trivial and computationally expensive. To simplify the problem, we use the dual formulation of the problem which is to track the pixel locations which belong to the set defined by $\{\mathbf{x} \notin \mathcal{U}\} = \mathcal{F} \cup \mathcal{B}$. This enables us to track the contour indirectly at high speeds as corner tracking is comparatively faster [87]. The trade-off in using the dual formulation is that we don't obtain the actual contour location across time - which might be needed for aggressive maneuvers, but this is not dealt in the scope of this paper. The label set propagation algorithm is described in Algorithm 1. Here, $\mathbb{C}^i_{\mathcal{F}}$ and $\mathbb{C}^i_{\mathcal{B}}$ represents a set of corner/feature points for the foreground and background pixels respectively in $\mathbb{F}^i$. $A^{\dagger}$ denotes the pseudo inverse of the matrix A and $\begin{bmatrix} x_{0,\mathcal{F}} & y_{0,\mathcal{F}} \end{bmatrix}^T$ is the FOE for the foreground. Intuitively, we solve the linear equations of the horizontal flow field to obtain the divergence/time-to-contact α. The divergence in x-direction is used to also predict the y-coordinates.

A.4.1 Safe Point Computation and Tracking

We can assume that the real-world points corresponding to the background $\mathcal{B} \cup \mathbb{U}_B$ are far enough that we can approximate them to lie on a plane. The foreground points under consideration $\mathcal{F} \cup \mathbb{U}_F$ occupy a small area around the contour which can be assumed to be planar. Here, $\mathbb{U}_F \subset \mathcal{U}$ and $\mathbb{U}_B \subset \mathcal{U}$ are the sets which actually belong to the foreground and background respectively.

Now, the quadrotor can be represented as an ellipsoid with semi-axes of lengths a, b and c. As an implication of the above assumptions, the projection of the quadrotor on the window at any instant is an ellipse. Let us define the projection of the quadrotor on the image as $\mathcal{Q}$. The largest $\mathcal{Q}$ can be written in terms of matrix equation of the ellipse centered at $[h, k]^T$ defined as $Q(h, k, R_\theta) = 0$.

Here, R_θ is a two-dimensional rotation matrix and θ is the angle the largest semi-axis of the ellipse makes with the X_C axis. The projection of the quadrotor on the image is given by $\mathcal{Q} = \{\mathbf{x} | Q(\mathbf{x}) \leq 0\}$. The safe region $\mathcal{S}$ can be computed as

$$\mathcal{S} = \bigcup_{\forall \theta} \mathcal{O} \ominus \mathcal{Q}$$

where $\ominus$ denotes the Minkowski difference of sets. Now, we define the 'safest point' ($\mathbf{x}_s$) as the barycentric mean of $\mathcal{S}$.

Remark. *The above optimization problem can only be solved using convex optimization with a guaranteed global solution when both $\mathcal{O}$ and $\mathcal{Q}$ are convex sets. A conservative solution to the above problem is fitting the largest scaled version of $\mathcal{Q}$ inside $\mathcal{O}$ when $\mathcal{O}$*

is a non-convex set and $\mathcal{Q}$ is a convex set.

Note that as $\mathcal{Q}$ is a chosen model, it can always be chosen to be a convex set, i.e., convex hull of the non-convex set. Also, from the above remark, the 'safest point' ($\mathbf{x}_s$) can be defined as the center of the largest ellipse which can be fit inside $\mathcal{S}$ whilst maintaining the eccentricity equal to that defined by $\mathcal{Q}$. The optimization problem becomes,

$$\underset{a,\theta}{\operatorname{argmax}} \mathcal{S} \cap \mathcal{Q} \text{ s.t. } \mathcal{Q} \subseteq \mathcal{S} \text{ and } |\theta| \leq \theta_{max}$$

This problem can be solved using the procedure described in [88]. However, a desirable property of the safe region is that it should favor less-aggressive maneuvers. This can be modelled as a regularization penalty in the above formulation,

$$\underset{a,\theta}{\operatorname{argmax}} \mathcal{S} \cap \mathcal{Q} + \lambda\theta \text{ s.t. } \mathcal{Q} \subseteq \mathcal{S} \text{ and } |\theta| \leq \theta_{max}$$

Solving the above optimization problem is computationally intensive and not-possible without the prior knowledge of the scale/depth. For obtaining the minimalist solution, we assume that *the gap is large enough for the quadrotor to pass through* and replace the above optimization problem by an empirically chosen approximation. A simple and efficient safe point computation can be performed as the median of the convex set $\mathcal{O}$ and is given by $\mathbf{x}_s \approx \operatorname{argmin}_{\mathbf{x}} \sum_{\forall o \in \mathcal{O}} \|o - \mathbf{x}\|_2$.

Remark. *If the above approximation is used when $\mathcal{O}$ is non-convex, the amount of deviation from the 'actual' safe point is a function of $\overline{\overline{Conv\,(\mathcal{O})}}/\overline{\overline{\mathcal{O}}}$. Here $\overline{\overline{Conv\,(\mathcal{O})}}$ is the convex hull of $\mathcal{O}$.*

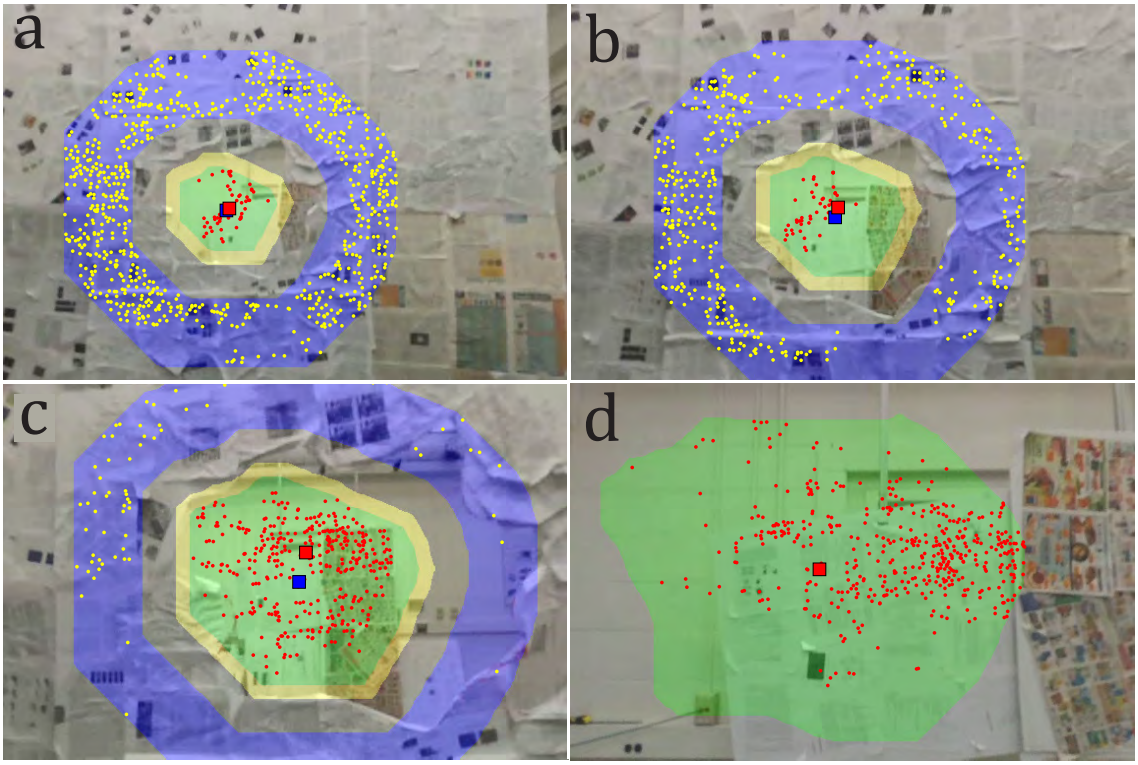


Figure A.5: Tracking of $\mathcal{F}$ and $\mathcal{B}$ across frames. (a) shows tracking when $\overline{\overline{\mathbb{C}^i_{\mathcal{F}}}} > k_{\mathcal{F}}$ and $\overline{\overline{\mathbb{C}^i_{\mathcal{B}}}} > k_{\mathcal{B}}$. (b) When $\overline{\overline{\mathbb{C}^i_{\mathcal{B}}}} \leq k_{\mathcal{B}}$, the tracking for $\mathcal{B}$ will be reset. (c) When $\overline{\overline{\mathbb{C}^i_{\mathcal{F}}}} \leq k_{\mathcal{F}}$, the tracking for $\mathcal{F}$ will be reset. (d) shows tracking only with $\mathcal{B}$, when $\mathcal{F} = \emptyset$. *(blue: $\mathcal{F}$, green: $\mathcal{B}$, yellow: $\mathcal{O}$, yellow dots: $\mathbb{C}^i_{\mathcal{F}}$, red dots: $\mathbb{C}^i_{\mathcal{B}}$, blue Square: $\mathbf{x}_{s,\mathcal{F}}$, red Square: $\mathbf{x}_{s,\mathcal{B}}$.)*

Keen readers would note that the formulation for the safe point is in-terms of $\mathcal{O}$ which we wanted to avoid tracking in the first place. Indeed this is true and a simple solution takes care of this. Because we are propagating $\mathcal{F}$ and $\mathcal{B}$ with FOE constraints, the cross-ratios of $\{\|o, f\|_2 \,|o \in \mathcal{O}, f \in \mathcal{F}\}$ and $\{\|o, b\|_2 \,|o \in \mathcal{O}, b \in \mathcal{B}\}$ are preserved. Here, $\mathcal{F}$ and $\mathcal{B}$ are computed from the detected opening $\mathcal{O}$ as follows: $\mathcal{F} = \{\mathcal{O} \oplus \epsilon_1 - \mathcal{O} \oplus \epsilon_2\}$ and $\mathcal{B} = \{\mathcal{O} \ominus \epsilon_3\}$. Here, ϵ_i is a user chosen kernel (circular in this paper).

The $\mathcal{F}$ and $\mathcal{B}$ are propagated from $\mathbb{F}^i$ onwards where the detection was performed.

The 'safest point' ($\mathbf{x}_s$) is computed as follows:

$$\mathbf{x}_s = \begin{cases} \mathbf{x}_{s,\mathcal{F}}, & \overline{\overline{\mathcal{F}}} \geq \overline{\overline{\mathcal{B}}} \\ \mathbf{x}_{s,\mathcal{B}}, & \text{otherwise} \end{cases}$$

where $\mathbf{x}_{s,\mathcal{F}}$ and $\mathbf{x}_{s,\mathcal{B}}$ are the safest points computed individually for $\mathcal{F}$ and $\mathcal{B}$ by taking their median respectively. Fig. A.5 shows the tracking of $\mathcal{F}$ and $\mathcal{B}$ across time and how the tracking algorithm actively switches between $\mathbf{x}_{s,\mathcal{F}}$ and $\mathbf{x}_{s,\mathcal{B}}$ for the safest point $\mathbf{x}_s$. When $\overline{\overline{\mathbb{C}^i_{\mathcal{F}}}} \leq k_{\mathcal{F}}$ or $\overline{\overline{\mathbb{C}^i_{\mathcal{B}}}} \leq k_{\mathcal{B}}$, the tracker/feature points are reset in the current $\mathcal{F}$ and $\mathcal{B}$ sets respectively. Here, $k_{\mathcal{F}}$ and $k_{\mathcal{B}}$ are empirically chosen thresholds. For all experiments $k_{\mathcal{F}} = 40$ and $k_{\mathcal{B}} = 20$. Due to resetting and active switching, $\mathbf{x}_s$ can jump around making the control hard, hence a simple Kalman filter with a forward motion model is used to smooth out the value of $\mathbf{x}_s$. From here on, safe point refers to the safest point $\mathbf{x}_s$.

A.4.2 Control Policy

We propose a control policy such that it follows the tracked $\mathbf{x}_s$. The quadrotor follows the dynamic model as given in [89]. The controller uses the traditional backstepping approach based on [89] and contains the following loops: Inner loop and outer loop controllers. Inner loop controls the attitude stability while the outer loop controller is responsible for the quadrotor position. It is important to note that frames are transformed from C to B.

Since the quadrotor is differentially flat, the altitude Z_B can be controlled independently from X_B and Y_B [83]. The control policy is to align the projection of the body

center on the image plane with $\mathbf{x}_s$. The difference between the two centers is called the error e. The x and y component of the error e can be minimized by varying roll (ϕ) and net thrust (u_1) respectively. A simple Proportional-Integral-Derivative (PID) controller on e is used. This control policy only deals with the alignment of B to $\mathbf{x}_s$ and does not deal with moving forward (Z_C). To move forward, the quadrotor pitch (ϕ) needs to be controlled. The rate of forward motion is controlled by the pitch angle θ_0 which is empirically chosen. The bigger the value of θ_0 the faster the quadrotor will fly towards the gap. It is important to note the implicit assumption made in this paper that the gap is large enough for the quadrotor to go through safely.

A.5 Experiments

A.5.1 Experimental Setup

The proposed framework was tested on a modified hobby quadrotor, Parrot® Bebop 2, for its cost effectiveness and ease of use. The Bebop 2 is equipped with a front facing camera, a 9-axis IMU and a downward facing optical flow sensor coupled with a sonar. The Parrot® Bebop 2 allows only high level controls in terms of body frame velocities using ROS. An NVIDIA Jetson TX2 GPU is mounted on the Bebop 2 as shown in Fig. A.6 and is used to run all the perception and control algorithms onboard. The TX2 and Bebop 2 communicate via a WiFi connection, where the images are received at 30Hz. The overall weight of the flight setup is 680g with the dimensions being $32.8 \times 38.2 \times 12$cm.

All the experiments were prototyped on a PC running Ubuntu 16.04 with an Intel® Core i7 6850K 3.6GHz CPU, an NVIDIA Titan-Xp GPU and 64GB of RAM in MATLAB

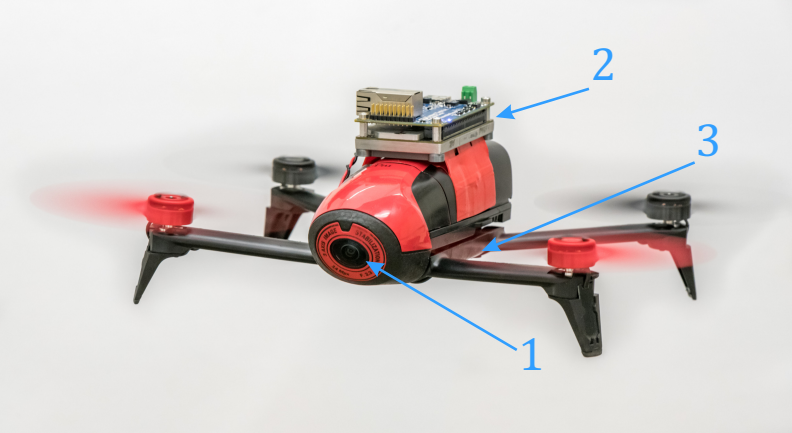


Figure A.6: The platform used for experiments. (1) The front facing camera, (2) NVIDIA TX2 CPU+GPU, (3) Downward facing optical flow sensor (camera+sonar) which is only used for position hold.

using the Robotics Toolbox. The deep learning based optical flow runs on Python with TensorFlow back-end. All the final versions of the software were ported to Python to run on the NVIDIA Jetson TX2 running Linux for Tegra® (L4T) 28.2. A step-by-step tutorial on using Bebop 2 as a research platform is available at `prg.cs.umd.edu/GapFlyt.html`.

The environmental setup for the experiments consists of a rigid scene which has two near-planar structures, one for the foreground and the other for the background. As shown in Fig. A.2, let us denote the initial perpendicular distance between the quadrotor body center and the foreground as ${}^0Z_{\mathcal{F}}$ and the background as ${}^0Z_{\mathcal{B}}$. The near-planar structures are made of foam-core with newspapers stuck on them to add texture to the scene. The gap is located near the center of the foreground and is of an arbitrary shape. For the detection of the window, the quadrotor executes a fixed diagonal straight line trajectory in the $X_W - Z_W$ plane as shown in Fig. A.7 while taking a number of images along its path. The number of images used for the detection stage is a parameter denoted by N.

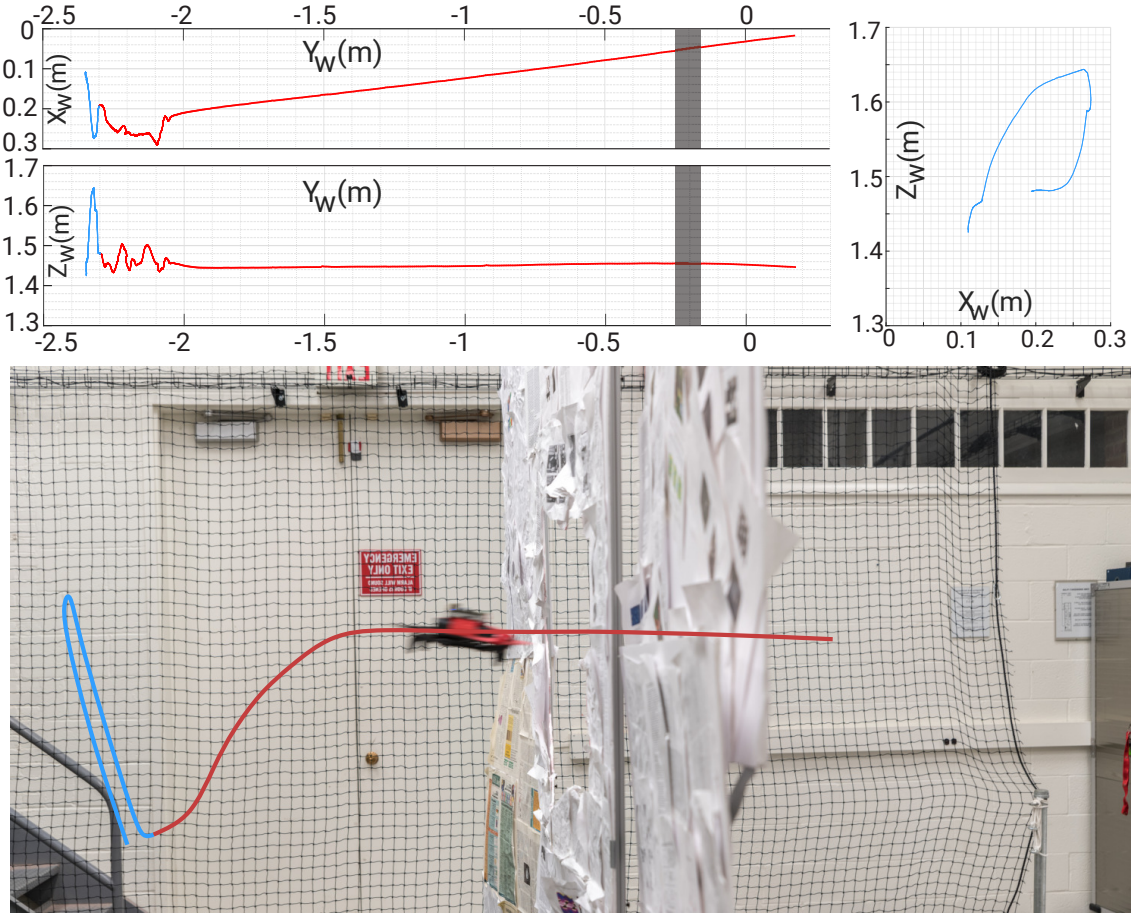


Figure A.7: First two rows: (X_W, Y_W), (Y_W, Z_W) and (X_W, Z_W) Vicon estimates of the trajectory executed by the quadrotor in different stages (gray bar indicates the gap). (X_W, Z_W) plot shows the diagonal scanning trajectory (the lines don't coincide due to drift). Last row: Photo of the quadrotor during gap traversal. *(cyan: detection stage, red: traversal stage.)*

Once the window is detected, $\mathcal{F}$ and $\mathcal{B}$ are tracked across time in order for quadrotor to go through the gap using visual servoing based control.

A.5.2 Experimental Results

The pipeline was evaluated on different variations of the environmental setup. In the first experiment, we test our pipeline on five different arbitrarily shaped gaps as shown in Fig. A.8. Unless otherwise stated ${}^0Z_{\mathcal{F}} \sim 2.6$m and ${}^0Z_{\mathcal{B}} \sim 5.7$m. The aim here is

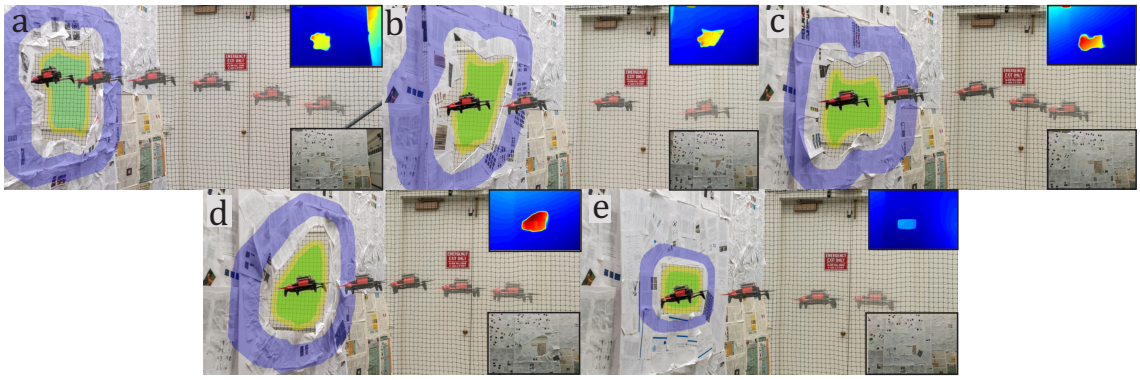


Figure A.8: Sequence of images of quadrotor going through different shaped gaps. Top on-set: Ξ outputs, bottom on-set: quadrotor view.

to find biases in the detection pipeline. The windows were chosen to have a diversity in the geometrical sharpness of the corners, convexity of the shape and the size. As stated earlier, the only constraint imposed on the gaps is that they are large enough to go through with the quadrotor pitch angle close to zero and near-convex. The outputs of TS^{2}P algorithm for different windows are shown in Fig. A.8 with $N = 4$ along with their inverse average (stacked) flow magnitudes Ξ, illustrating that our detection algorithm is independent of shape and size of the opening. A canny edge detector is run on Ξ followed by morphological operations to obtain $\mathcal{C}$.

The second experiment is designed to test the noise sensitivity of TS^{2}P. The intuition is that as ${}^0Z_{\mathcal{F}} \to {}^0Z_{\mathcal{B}}$, noisier the detection result . The outputs for different ${}^0Z_{\mathcal{F}}$ and ${}^0Z_{\mathcal{B}}$ are shown in Fig. A.9 when $N = 4$. This is because the fidelity of Ξ becomes less and is more prone to noise. By increasing N the noise gets averaged out across frames improving the fidelity of Ξ.

In the third experiment, we present detection outputs for different values of N, image baselines and image sizes. The effect of N has been already discussed previously. Having a very small baseline results in effectively dropping the value of N and vice-

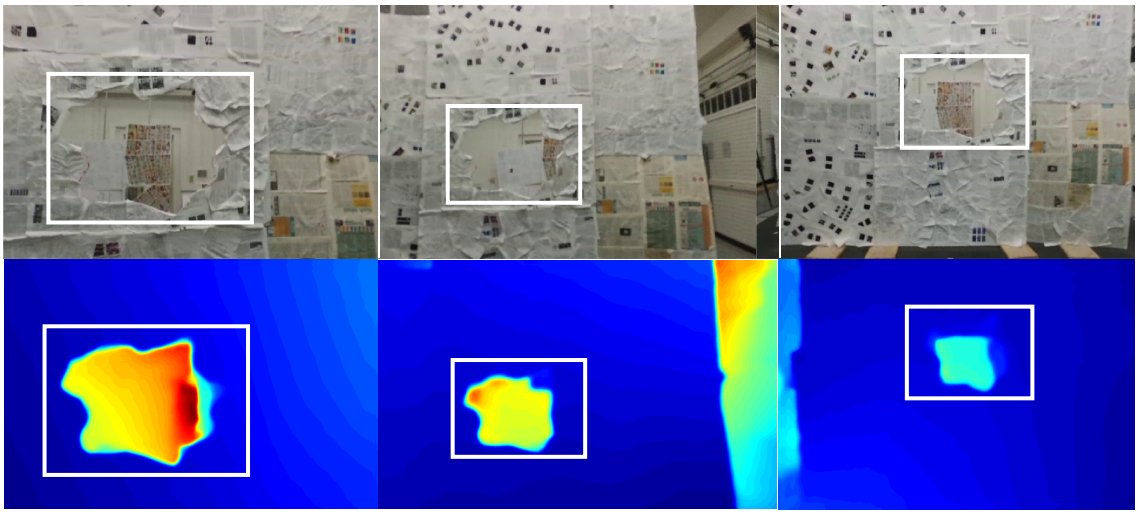


Figure A.9: Top Row (left to right): Quadrotor view at ${}^0Z_{\mathcal{F}} = 1.5, 2.6, 3$m respectively with ${}^0Z_{\mathcal{B}} = 5.7$m. Bottom Row: Respective Ξ outputs for $N = 4$. Observe how the fidelity of Ξ reduces as ${}^0Z_{\mathcal{F}} \rightarrow {}^0Z_{\mathcal{B}}$, making the detection more noisy. *(white boxes show the location of the gap in Figs. A.9 to A.13.)*

versa. The results from different sized images as illustrated in Fig. A.12 show that the detection algorithm can work even on a very small quadrotor which can only carry a very low-resolution camera (as low as 32×48 pixels). Our algorithm can also handle dynamic noises very well though being modelled as a gaussian for the discussion. However, one can notice that the results improve significantly with increase in N (Fig. A.13) demonstrating the advantage of TS^{2}P.

Gap detection using TS^{2}P almost always results in an eroded version of the true gap. This is good for safe maneuvers like the one considered in this paper. However, aggressive flight algorithms might suffer due to conservative results. This can be mitigated by tuning the values of image baselines, N and the trajectory of the quadrotor to obtain minimum erosion results. Tuning these parameters is easy when a prior about the scene is known or the flow algorithms are so fast that one can actively change the detection trajectory so as to maximize the coverage on the part of the contour least 'seen'. The dynamic choice of

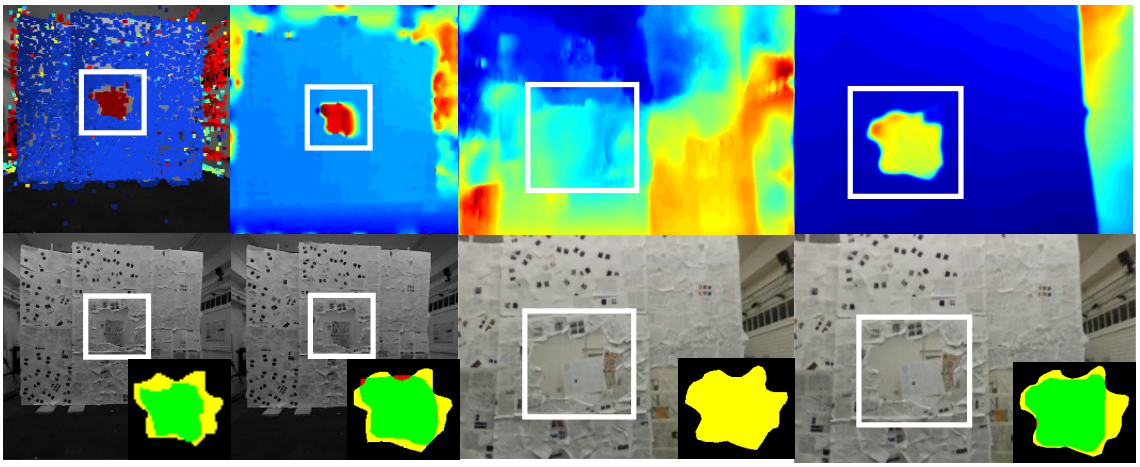


Figure A.10: Comparison of different philosophies to gap detection. Top row (left to right): DSO, Stereo Depth, MonoDepth, TS^{2}P. Bottom row shows the detected gap overlayed on the corresponding input image. *(green: $\mathcal{G} \cap \mathcal{O}$, yellow: false negative $\mathcal{G} \cap \mathcal{O}'$, red: false positive $\mathcal{G}' \cap \mathcal{O}$.)*

these parameters comes into the scope of our future work.

In the last experiment we present alternative approaches including state-of-the-art methods which can be used to find the gap. The methods can be subdivided into structure based approaches and stuctureless approaches. The structure based approaches can be defined as the set of approaches where a full 3D reconstruction of the scene is computed, whereas, stuctureless approaches do not. The structure based approaches presented are DSO [71] – Direct Sparse Odometry, depth from hardware stereo cameras [70] and Stereo SLAM – Simultaneous Localization and mapping using Stereo Cameras and IMU [70]. The data for the structured approaches were collected using a Parrot® SLAMDunk [70]. The structureless approaches presented are MonoDepth [90] – deep learning based monocular depth estimation and the proposed TS^{2}P on two different deep learning based dense optical flow algorithms, namely, FlowNet2 [86], SpyNet [91] and DIS [92]. Table A.2 shows the comparison of the stated methods averaged over 150 trials.

Fig. A.10 compares the results of DSO, stereo depth, MonoDepth and our method (TS^{2}P) with the ground truth. It can be inferred that the MonoDepth results are extremely noisy (even with different models) making it impossible to detect the gap as the images in our paper were never "seen" during training. Note that we don't retrain or finetune any of the deep learning models in this paper. Retraining MonoDepth and other deep learning based methods used in this paper on our dataset might lead to better results. Whereas, DSO and stereo depth results can used to detect the opening with some filtering. Stereo SLAM and DSO are slow in the map building stage (taking about 6s and 12s respectively), however, once the map is built and the algorithms are in the localization stage the depth (or scaled depth) are obtained at 20Hz. The Stereo SLAM and Stereo Depth were run on the SLAMDunk with an NVIDIA Jetson TK1 processor which is much slower than the NVIDIA Jetson TX2 processor used for running DSO and other methods.

Fig. A.11 compares different optical flow methods used for TS^{2}P. Though SpyNet and DIS optical flow are faster, FlowNet2 outputs significantly better results at the edges which is important for obtaining a good gap detection – this can be observed by looking at Ξ for each algorithm.

After the gap detection has been performed, $\mathcal{F}$ and $\mathcal{B}$ are computed from the detected gap $\mathcal{C}$. Fig. A.5 shows $\mathcal{F}$ and $\mathcal{B}$ being propagated across time as the quadrotor is in pursuit of going through the gap with the update of $\mathbf{x}_s$. A comparison of tracking using different methods are given in Table A.3. Clearly, KLT outperforms all other methods with a theoretical maximum quadrotor speed of 8 ms^{-1} in the given scene. The theoretical maximum speed is calculated for a global shutter camera in such a way that the motion parallax is constrained within one pixel for the scene with ${}^0Z_{\mathcal{F}} \sim 2.6$m and

Table A.2: Comparison of different methods used for gap detection.

Method	Sensor(s)	Run Time (Init. Time) in s	DR	AFN	AFP
DSO[71]	Monocular	0.05 (6)	0.88	0.20	**0.00**
Stereo Depth*[70]	Stereo	0.10	0.90	0.17	0.04
Stereo SLAM*[70]	Stereo + IMU	0.05 (12)	0.91	0.15	**0.00**
Mono Depth[90]	Monocular	0.97	0.00	–	–
TS^{2}P (FlowNet2[86])	Monocular	1.00	**0.93**	**0.14**	0.02
TS^{2}P (SpyNet[91])	Monocular	0.12	0.74	0.16	0.05
TS^{2}P (DIS[92])	Monocular	0.45	0.62	0.20	0.04

* indicates algorithm tested on NVIDIA TK1 otherwise NVIDIA TX2.

${}^0Z_{\mathcal{B}} \sim 5.7$m. The calculation assumes that none of the tracking/matching methods work when the motion blur is more than one pixel. However, most of the methods can work well upto some pixels of motion blur, this will in-turn increase the theoretical maximum speed by this factor. If a rolling shutter camera is used without rolling shutter compensation, the theoretical maximum speed value has to be divided by the factor of blur caused by rolling shutter. We achieved a practical maximum speed of 2.5ms^{-1} in our experiments. We were limited to this speed due to the acceleration constraints on the Bebop 2 and the rolling shutter camera.

We achieved a remarkable success rate of 85% over 150 trials for different arbitrary shaped windows under a wide range of conditions which includes a window with a minimum tolerance of just 5cm (Fig. A.14). Success is defined as window detection output $\mathcal{O}$ having at least 75% overlap with the ground truth and traversal through the gap without collision. Failure cases also include the optimization failures and/or feature tracking failures for structure based approaches. For TS^{2}P, we define Detection Rate (DR), Average False Negative (AFN) and Average False Positive (AFP) as follows (AFN and AFP are computed only for successful trails):

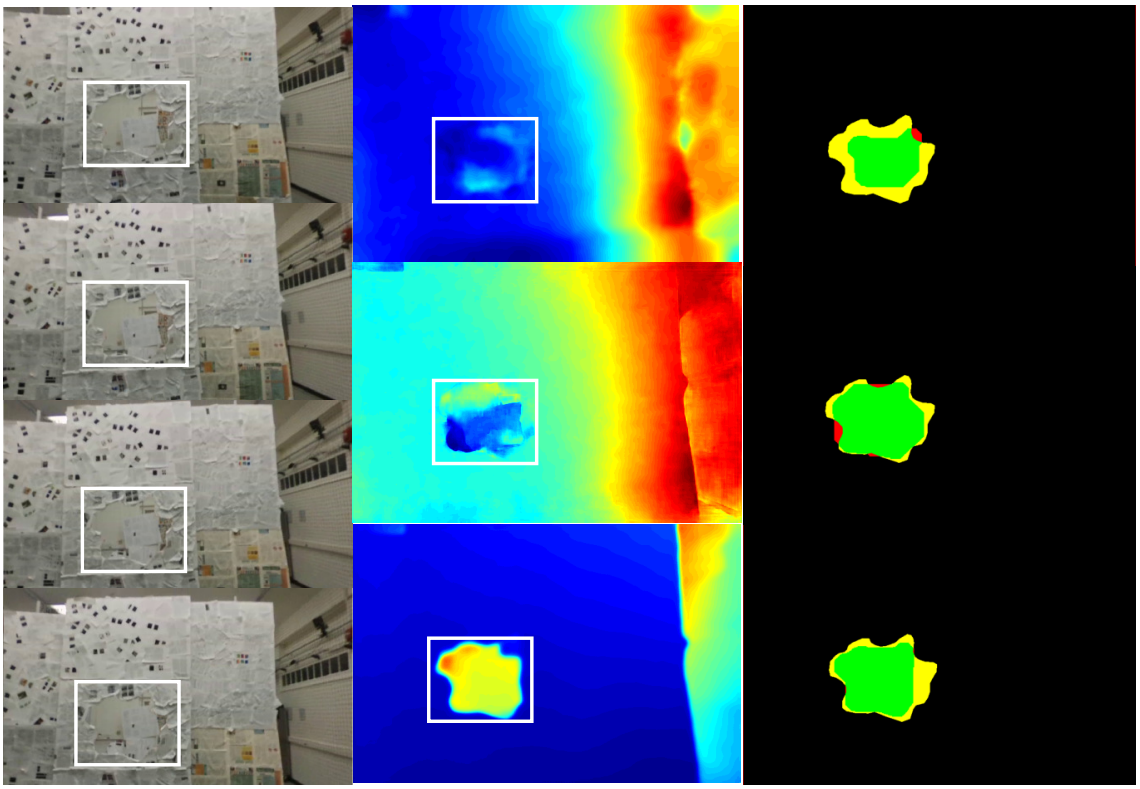


Figure A.11: Left Column: Images used to compute Ξ. Middle Column (top to bottom): Ξ outputs for DIS Flow, SpyNet and FlowNet2. Right Column: Gap Detection outputs. *(green: $\mathcal{G} \cap \mathcal{O}$, yellow: false negative $\mathcal{G} \cap \mathcal{O}'$, red: false positive $\mathcal{G}' \cap \mathcal{O}$.*

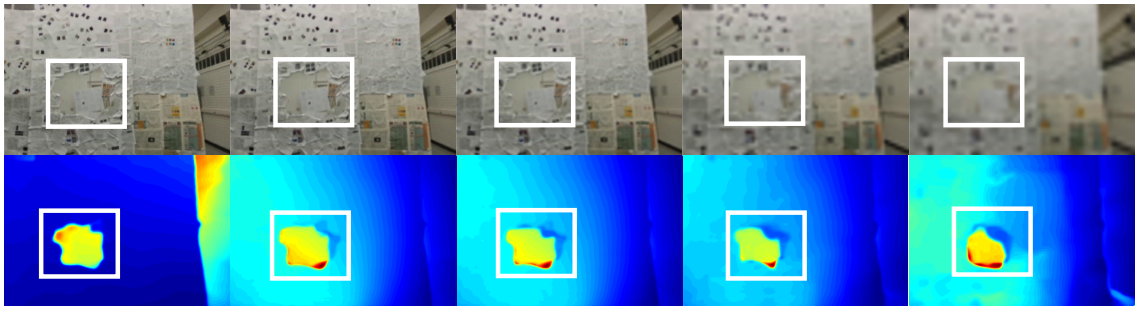


Figure A.12: Top row (left to right): Quadrotor view at image sizes of 384×576, 192×288, 96×144, 48×72, 32×48. Note all images are re-scaled to 384×576 for better viewing. Bottom row shows the respective Ξ outputs for $N = 4$.

$$\text{DR} = \frac{\sum_{k=1}^{\text{Num. Trails}} \left(\lambda_D^k\right)}{\text{Num. Trails}}; \lambda_D^k = \left(\frac{\overline{\overline{\mathcal{G} \cap \mathcal{O}}}}{\overline{\overline{\mathcal{G}}}}\right)^k \geq 0.75$$

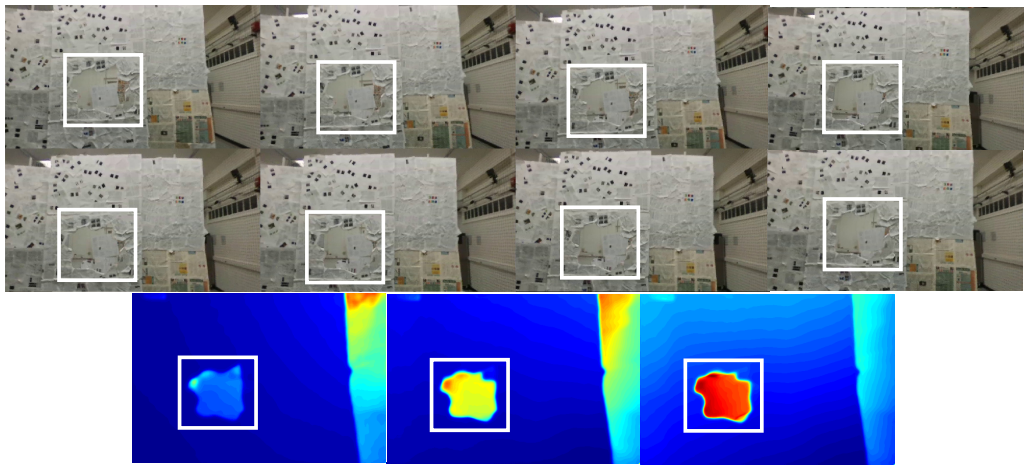


Figure A.13: Top two rows show the input images. The third row shows the Ξ outputs when only the first 2, 4 and all 8 images are used.

Figure A.14: Quadrotor traversing an unknown window with a minimum tolerance of just 5cm. *(red dashed line denotes $\mathcal{C}$.)*

$$\text{AFN} = \frac{\sum_{k=1}^{\text{Num. Succ. Trails}} \left(\lambda_N^k\right)}{\text{Num. Succ. Trails}}; \lambda_N^k = \left(\frac{\overline{\overline{\mathcal{G} \cap \mathcal{O}'}}}{\overline{\overline{\mathcal{G}}}}\right)^k$$

$$\text{AFP} = \frac{\sum_{k=1}^{\text{Num. Succ. Trails}} \left(\lambda_P^k\right)}{\text{Num. Succ. Trails}}; \lambda_P^k = \left(\frac{\overline{\overline{\mathcal{G}' \cap \mathcal{O}}}}{\overline{\overline{\mathcal{G}}}}\right)^k$$

Table A.3: Comparison of different methods used for tracking.

Method	Run Time (ms)	Theo. Max. Speed (ms^{-1})
GMS[93]	40	0.40
FAST[94] + RANSAC	8.3	1.92
Cuda-SIFT[95] + RANSAC	5	3.20
KLT[87]	2	8.00

where $\mathcal{A}'$ is the negation of the set $\mathcal{A}$.

In the next section, we also experiment with different textures to qualitatively evaluate the robustness of TS^{2}P.

A.5.3 Robustness of TS^{2}P against different textures

Optical flow algorithms are generally biased to specific textures, and there is a high correlation between highly textured surfaces and good optical flow computation. To demonstrate the robustness of our approach (TS^{2}P) we test the algorithm against ten additional setups. The various scenarios are tabulated in Table A.4. Each scenario is a combination of different textures from the following set: Bumpy, Leaves, Door, Newspaper, Wall, Low-Texture and Cloth. We now describe each of textures used.

Bumpy texture provides an uneven texture over the original newspaper scenario. These "bumps" are made of crumpled paper. The depth (protrusion) of the bumps are large and are about 25% of the distance to gap from the quadrotor's initial position. This scenario mimics the uneven texture on rock walls around a cave opening, for example.

Leaves texture mimics foliage. In this setup magnolia and .. leaves are glued onto foam-board. The two leaves used are of very different sizes and shapes. The leaves texture are also uneven with depth variation as large as 10% of the distance to the quadrotor's

initial position. We use both sides of the leaves. The front-side of the leaves are of a glossy texture with very high reflectance while the back-side are of matte texture with very low reflectance. Also, the leaves look similar to each other. This texture provides similar repeating patterns and large changes in reflectance.

Door texture is a wall with a door. Both these are white with very low texture.

Newspaper texture is the setup similar to the one used in the main paper. Newspaper is glued onto foam-board. This presents an artificial high-texture scenario.

Wall texture is foam-core with a small amount of logos. We consider this as a medium-texture scenario.

Low-Texture is white foam-core with a few scribbles near the window which are required for tracking. This is artificially crafted to mimic a minimal-textured plain wall with windows.

Cloth texture is created by the usage of wrinkled bed sheets. This scenario mimics hanging curtains, hanging paintings and hanging flags.

A combination of the aforementioned textures creates scenarios which test the bias of the TS^{2}P algorithm. In all the above cases, ${}^0Z_{\mathcal{F}} \sim 2.8$m and ${}^0Z_{\mathcal{B}} \sim 5.6$m and $N = 3$ frames are used for stacking/averaging in all the cases.

Our approach works in most of the scenarios as presented in Fig. A.15 since it uses deep learning based optical flow to estimate the position of the gap in the image plane. Even in the low-textured scenarios, the window detection output $\mathcal{O}$ still has at least 75% overlap with the ground truth as mentioned in our paper. TS^{2}P works even with no textures on one of the foreground or background. Though tracking the $\mathcal{F}$ and $\mathcal{B}$ without any textures is not possible.

Table A.4: Comparison of our approach with different setups

Scenario	Foreground	Background
1	Bumpy	Leaves
2	Bumpy	Door
3	Bumpy	Newspaper
4	Bumpy	Cloth
5	Leaves	Wall
6	Leaves	Newspaper
7	Cloth	Wall
8	Low-Texture	Leaves
9	Low-Texture	Leaves
10	Low-Texture	Leaves

A.6 Conclusions

We present a minimalist philosophy to mimic insect behaviour to solve complex problems with minimal sensing and active movement to simplify the problem in hand. This philosophy was used to develop a method to find an unknown gap and fly through it using only a monocular camera and onboard sensing. A comprehensive comparison and analysis is provided. To our knowledge, this is the first paper which addresses the problem of gap detection of an unknown shape and location with a monocular camera and onboard sensing. As a parting thought, IMU data can be coupled with the monocular camera to get a scale of the window and plan for aggressive maneuvers.

Acknowledgement

The authors would like to thank Konstantinos Zampogiannis for helpful discussions and feedback. This work was supported in part by the Brin Family Foundation, in part by the Northrop Grumman Corporation, and in part by the National Science Foundation under Grants SMA 1540917 and CNS 1544797, respectively.

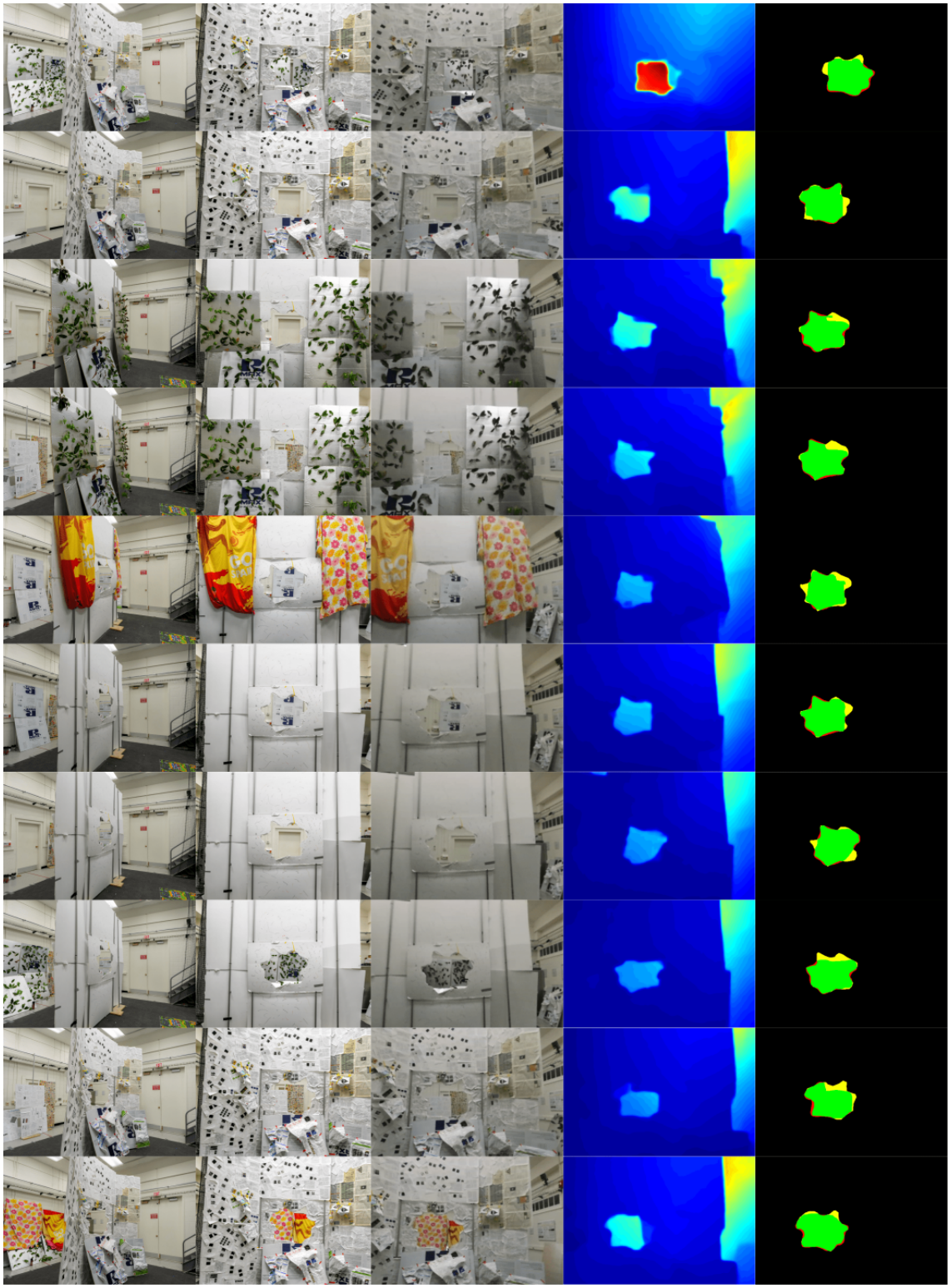


Figure A.15: Left to right columnwise: Side view of the setup, Front view of the setup, sample image frame used, Ξ output, Detection output - Yellow: Ground Truth, Green: Correctly detected region, Red: Incorrectly detected region. Rowwise: Cases in the order in Table A.4. Best viewed in color.

Appendix B: EVDodgeNet

Nitin J. Sanket*, Chethan M. Parameshwara*, Chahat Deep Singh, Ashwin V. Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, Yiannis Aloimonos "EVDodgeNet: Deep Dynamic Obstacle Dodging with Event Cameras", *IEEE International Conference on Robotics and Automation (ICRA)*, (2020). DOI: 10.1109/ICRA40945.2020.9196877.

EVDodgeNet: Deep Dynamic Obstacle Dodging with Event Cameras

Nitin J. Sanket*, Chethan M. Parameshwara*, Chahat Deep Singh,

Ashwin V. Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, Yiannis Aloimonos

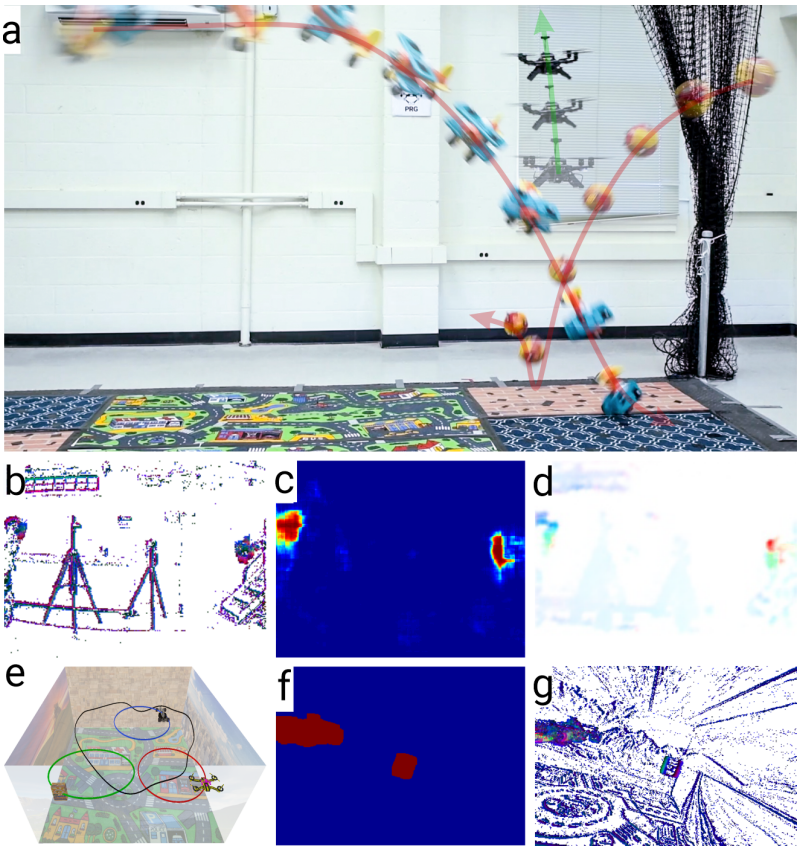


Figure B.1: (a) A real quadrotor running *EVDodgeNet* to dodge two obstacles thrown at it simultaneously. (b) Raw event frame as seen from the front event camera. (c) Segmentation output. (d) Segmentation flow output which includes both segmentation and optical flow. (e) Simulation environment where *EVDodgeNet* was trained. (f) Segmentation ground truth. (g) Simulated front facing event frame. *All the images in this paper are best viewed in color.*

Abstract — Dynamic obstacle avoidance on quadrotors requires low latency. A

class of sensors that are particularly suitable for such scenarios are event cameras. In this paper, we present a deep learning – based solution for dodging multiple dynamic obstacles on a quadrotor with a single event camera and on-board computation. Our approach uses a series of shallow neural networks for estimating both the ego-motion and the motion of independently moving objects. The networks are trained in simulation and directly transfer to the real world without any fine-tuning or retraining.

We successfully evaluate and demonstrate the proposed approach in many real-world experiments with obstacles of different shapes and sizes, achieving an overall success rate of 70% including objects of unknown shape and a low light testing scenario. To our knowledge, this is the first deep learning – based solution to the problem of dynamic obstacle avoidance using event cameras on a quadrotor. Finally, we also extend our work to the pursuit task by merely reversing the control policy, proving that our navigation stack can cater to different scenarios.

B.1 Supplementary Material

The supplementary hardware tutorial, appendix, code and video are available at prg.cs.umd.edu/EVDodgeNet.html.

B.2 Introduction and Philosophy

The never-ending quest to understand and mimic ultra-efficient flying agents like bees, flies, and birds has fueled the human fascination to create autonomous, agile and ultra-efficient small aerial robots. These robots are not only utilitarian but are much safer

to operate in static or dynamic environments and around other agents as compared to their larger counterparts. Need for creation of such small aerial robots has given rise to the development of numerous perception algorithms for low latency obstacle avoidance. Here, latency is defined as the time the robot takes to perceive, interpret and generate control commands [96].

Low latency static obstacle avoidance has been studied extensively in the last decade [97]. Recently, however, dynamic obstacle avoidance has gained popularity in the field of robotics due to the exponential growth of event cameras. These are bioinspired vision sensors that output per-pixel temporal intensity differences caused by relative motion with microsecond latency [98].

Event cameras have the potential to become the de-facto standard for visual motion estimation problems due to their inherent advantages of low latency, high temporal resolution, and high dynamic range [99]. These advantages make event cameras tailor made for dynamic obstacle avoidance.

In this paper, we present a framework to dodge multiple unknown dynamic obstacles on a quadrotor with event cameras using deep learning. Although dynamic obstacle detection using traditional cameras and deep learning has been extensively studied in the computer vision community under the umbrellas of object segmentation and detection, they are either of high latency, computationally expensive (not enough to be used on micro/nano-quadrotors) and/or do not generalize to novel objects without retraining or fine-tuning.

Our work is closely related to [96] with the key difference being that our approach uses deep learning and generalizes to unknown real objects after being trained only on

simulation.

B.2.1 Problem Formulation and Contributions

A quadrotor moves in a static scene with multiple Independently Moving dynamic Objects/obstacles (IMOs). The quadrotor is equipped with a front facing event camera, a downfacing lower resolution event camera coupled with sonar, for altitude measurements, and an IMU.

The problem we address is as follows: *Can we present an AI framework for the task of dodging/evading/avoiding these dynamic obstacles without any prior knowledge, using only on-board sensing and computation?*

We present various flavors of the dodging problem, such as hovering quadrotor dodging unknown obstacles, slow-moving quadrotor dodging unknown shaped obstacles given a bound on size, hovering and slow moving quadrotor dodging known objects (particularly targeted to spherical objects of known radii).

We also broaden the horizon of our approach by demonstrating pursuit/intercept of a known object using the same deep-learning framework. This showcases that our proposed framework can be used in a general navigation stack on a quadrotor and can be re-purposed for various related tasks.

A summary of our contributions is given below (Fig. B.1):

- We propose and implement a network (called EVDeBlurNet) that *deblurs* event frames, such that learning algorithms trained on simulated data can generalize to real scenes without retraining or fine-tuning.

- We design and implement a network (called EVSegFlowNet) that performs both segmentation and optical flow of IMOs to obtain both segmentation and optical flow in a single network.
- We propose a control policy based on estimated motion of multiple IMOs under various scenarios.
- We evaluate and demonstrate the proposed approach on a real quadrotor with on-board perception and computation.

B.2.2 Related Work

We subdivide the related work into three parts, i.e., ego-motion estimation, independent motion segmentation, and obstacle avoidance.

B.2.2.1 Independent Motion Detection and Ego-motion Estimation – Two sides of the same coin

Information from complementary sensors, such as standard cameras and Inertial Measurement Units (IMUs), has given rise to the field of Visual Inertial Odometry (VIO) [64, 63]. Low latency VIO algorithms based on event cameras have been presented in [15, 99], which use classical feature tracking inspired methods to estimate ego-motion. Other works, instead, try to add semantic information to enhance the quality of odometry by adding strong priors about the scene [100, 101]. Most works in the literature focus on ego-motion estimation in static scenes which are seldom encountered in the real world. To account for moving objects, these algorithms implement a set of outlier rejection schemes

to detect IMOs. We would like to point out that by carefully modelling these "outliers" one can estimate both ego-motion and IMO motion [102].

B.2.2.2 Image stabilization as a key to independent motion segmentation

Keen readers might have contrived that by performing the process of image stabilization IMOs would "stand-out". Indeed, this was the approach most robust algorithms used in the last two decades. A similar concept was adapted in some recent works on event-based cameras for detecting IMOs [103, 10, 104]. Recently a deep learning based approach was presented for IMO detection using a structure from motion inspired approach [12].

B.2.2.3 Obstacle avoidance on aerial robots

The works presented in the above two subsections have aided the advancement of obstacle avoidance on aerial robots. [105, 34] presented approaches for high speed static obstacle avoidance by estimating depth maps and visual servoing using a monocular imaging camera respectively. [106] provides a detailed collation of the prior work on static obstacle avoidance using stereo cameras. A hardware and software architecture was proposed in [107, 108] for high speed quadrotor navigation by mapping the cluttered environment using a lidar. Using event cameras for high speed dodging is not new and the first work was presented in [109] where an approach was presented to avoid a dynamic spherical obstacle using stereo event cameras. Very recently, [96] presented a detailed analysis of perception latency for dodging a dynamic obstacle.

B.2.3 Organization of the paper

The paper is structured into perception and control modules. The perception module (Refer to Fig. B.2) is further divided into three segments.

1. The input to the perception system are event frames (Sec. B.3.3). Such a projection of event data to generate *event frames* suffers from misalignment [11] unless motion compensation is performed. We call this misalignment or loss of contrast/sharpness as *blur* due to its visual resemblance to classical image motion blur. To perform motion compensation and denoising, we present a neural network called *EVDeBlurNet* in Sec. B.3.3.

2. Sec. B.3.4 presents how ego-motion is obtained using *EVHomographyNet*.

3. Sec. B.3.5 describes how segmentation and optical flow of IMOs are obtained using the novel *EVSegFlowNet*.

Sec. B.5 presents the control scheme for dodging given the outputs from the perception module. We also bring the generality of our perception stack into limelight in Sec. B.5.4 by adapting our approach to the problem of pursuit. Sec. B.6 illustrates the experimental setup and provides error analyses of the approaches presented along with detailed ablation studies. We finally conclude the paper in Sec. B.7 with parting thoughts on future work.

B.3 Deep Learning Based Navigation Stack For Dodging Dynamic Obstacles

An overview of our proposed approach is illustrated in Fig. B.2.

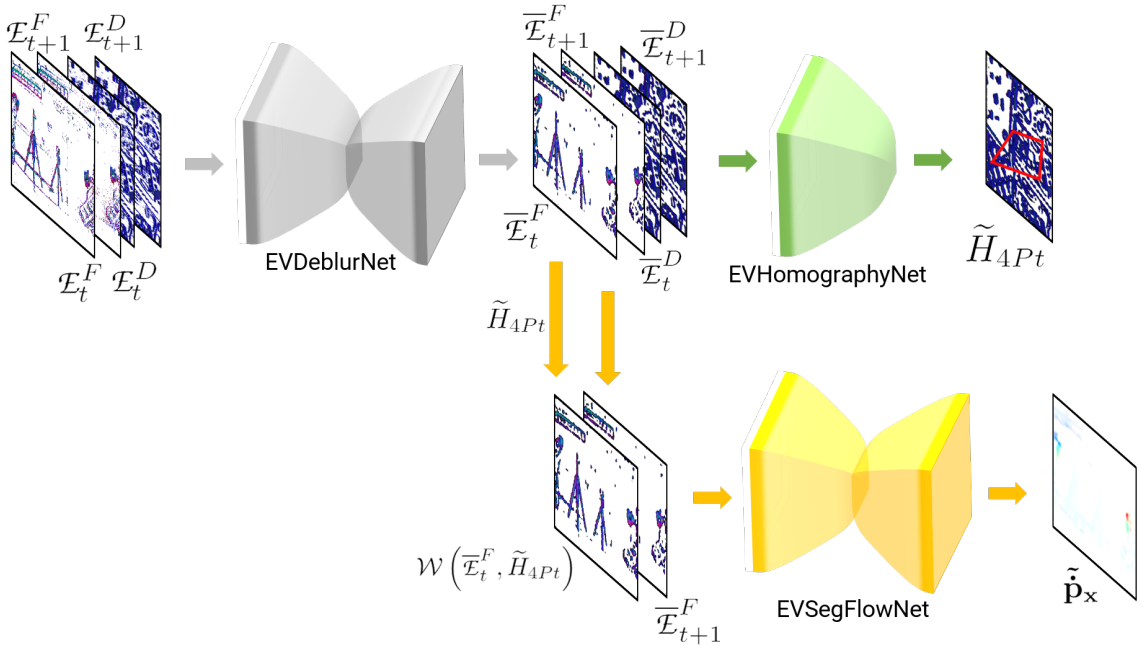


Figure B.2: Overview of the proposed neural network based navigation stack for the purpose of dodging.

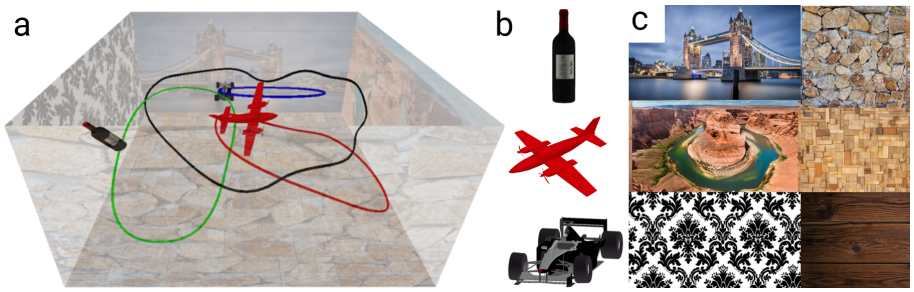


Figure B.3: (a) A sample simulation scene used for training our networks, (b) Sample objects used in (a), (c) sample scene textures used in (a).

B.3.1 Definitions Of Coordinate Frames Used

The letters I, E^F, E^D, S and W denote coordinate frames on the Inertial Measurement Unit (IMU), front facing event camera, down facing event camera, down facing sonar and the world respectively (Fig. B.16). All the sensors are assumed to be rigidly attached with the intrinsic and extrinsic calibration between them known. A pinhole camera model is used for the formation of the image. The world point $\mathbf{X}$ gets projected onto

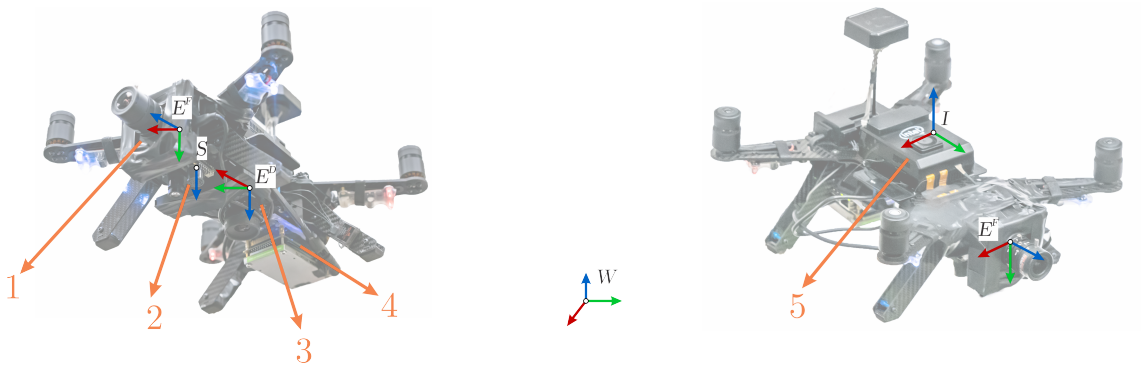


Figure B.4: Representation of coordinate frames on the hardware platform used. (1) Front facing DAVIS 240C, (2) down facing sonar on PX4Flow, (3) down facing DAVIS 240B, (4) NVIDIA TX2 CPU+GPU, (5) Intel® Aero Compute board.

the image plane point $\mathbf{x}$. Unless otherwise stated, the points on the image plane are used after undistortion.

B.3.2 Event Frame $\mathcal{E}$

A traditional grayscale (global-shutter) camera records frames at a fixed frame rate by integrating the number of photons for the chosen shutter time. This is done for all pixels synchronously. In contrast, an event camera only records the polarity of logarithmic brightness changes *asynchronously at each pixel*. If the brightness at time t of a pixel at location $\mathbf{x}$ is given by $I_{t,\mathbf{x}}$, an event is triggered when:

$$\| \log\left(I_{t+1,\mathbf{x}}\right) - \log\left(I_{t,\mathbf{x}}\right) \|_1 \geq \tau$$

Here τ is a threshold which will determine if an event is triggered or not. τ is set at the driver level as a combination of multiple parameters. Each triggered event outputs the following data:

$$\mathbf{e} = \{\mathbf{x}, t, p\}$$

where $p = \pm 1$ denotes the sign of the brightness change. The event data unlike an image can vary in data rate and are generally output as a vector of four numbers per triggered event. The data rate is small when the amount of motion and/or scene contrast are small and large when either the motion or the scene contrast is large. This can be beneficial for asynchronous operation, however on a low power digital processor like the one on-board a micro-quadrotor, this can be appalling. To maintain a near constant computational bottle-neck for event processing, we create event frames denoted as $\mathcal{E}$. An event frame is essentially a collection events triggered in a spatio-temporal window starting at t_0 and a temporal depth of δt. The event frame $\mathcal{E}$ is formed as follows.

$$\mathcal{E}\left(\mathbf{x}, \delta t\right)_{+} = \sum_{t=t_0}^{t_0+\delta t} \mathbb{1}\left(\mathbf{x}, t, p = +1\right)$$
$$\mathcal{E}\left(\mathbf{x}, \delta t\right)_{-} = \sum_{t=t_0}^{t_0+\delta t} \mathbb{1}\left(\mathbf{x}, t, p = -1\right)$$
$$\mathcal{E}\left(\mathbf{x}, \delta t\right)_{\tau} = \left(\sum_{t=t_0}^{t_0+\delta t} \mathbb{1}\left(\mathbf{x}, t, p = \pm 1\right)\right)^{-1} \mathbb{E}\left(t - t_0\right)$$

Here $\mathbb{1}$ is an indicator function which has a value of 1 for an event triggered with polarity of p. For $\mathcal{E}_{+}$ the value of p is +1. Here $\mathbb{E}$ is the expectation/averaging operator. Finally $\mathcal{E}_{+}$, $\mathcal{E}_{-}$ and $\mathcal{E}_{\tau}$ are normalized such that minimum and maximum values are scaled between $[0, 1]$. Essentially $\mathcal{E}_{+}$/$\mathcal{E}_{-}$ captures the per-pixel average number of positive/negative event triggers in the spatio-temportal window spanned between t_0 and $t_0 + \delta t$. $\mathcal{E}_{\tau}$ captures the average trigger time per pixel. This event frame representation is inspired by previous works [10][110]. The event frame $\mathcal{E}$ is composed by depthwise stacking $\mathcal{E}_{+}$, $\mathcal{E}_{-}$ and $\mathcal{E}_{\tau}$, i.e., $\mathcal{E} = \{\mathcal{E}_{+}, \mathcal{E}_{-}, \mathcal{E}_{\tau}\}$. Using event frames has some pragmatic advantages as compared

to processing event by event on the raw event stream.

- The control command can be produced within a constant time bound as event frames are produced at a near constant rate.
- The spatial relationships between event triggers are preserved along with polarity and timing information which is exploited by convolutional neural networks employed in this paper.
- Event frames can be produced in *linear time* in the number of event triggers.

B.3.3 EVDeBlurNet

The event frame $\mathcal{E}$ consists of three channels. The first and second channels contain the per-pixel average count of positive and negative events. The third channel contains the per-pixel average time between events (refer to Section S.I for a mathematical formulation). Though event representation offers many advantages regarding computational complexity and providing tight time bounds on operation, there is a hitch. Event frames can be "blurry" (projection of misaligned events) based on a combination of the integration time δt (observe in Fig. B.15 how sharpness of the image decreases as integration time δt increases), apparent scene movement on the image plane (which depends on the amount of camera movement and depth of the scene) and scene contrast (contrast of the latent image pixels). Here, we define blur on the event frame $\mathcal{E}$ as the misalignment of the events in a small integration time δt.

An event is triggered when the relative contrast (on the latent image I) exceeds a

threshold τ and is mathematically modelled as

$$\| \log (I) \|_1 \approx \| \langle \nabla_{\mathbf{x}} \log (I) , \dot{\mathbf{x}} \Delta t \rangle \|_1 \geq \tau \tag{B.1}$$

Here, $\langle \cdot, \cdot \rangle$ denotes the inner/dot product between two vectors, $\nabla_{\mathbf{x}}$ is the spatial gradient, $\dot{\mathbf{x}}$ is the motion field on the image and Δt is the time since the previous event at the same location. The above equation elucidates how the latent image contrast, motion and depth are coupled to event frames. Note that, $\dot{\mathbf{x}}$ depends on the 3D camera motion and the scene depth. We refer the reader to [111] for more details.

This "blurriness" of the event frame can adversely affect the performance of algorithms built on them. To alleviate this problem, we need to deblur the event images. This is fairly easy if we directly use the spatio-temporal event cloud and follow the approach described in [11]. Essentially the problem deals with finding point trajectories along the spatio-temporal point cloud to maximize a heuristically chosen contrast function. Mathematically, we want to solve the following problem.

$$\underset{\theta}{\operatorname{argmax}} \ \mathcal{C} \left(\mathcal{W} \left(\mathcal{E}, \theta \right) \right) \tag{B.2}$$

where $\mathcal{C}$ is a heuristic contrast function and θ are the parameters of point trajectories in the spatio-temporal point cloud according to which the events are warped and $\mathcal{W} \left(\mathcal{E}, \theta \right)$ represents the event image formed by the warped events. In our scenario, we want to model the deblurring problem in 2D, i.e., working on $\mathcal{E}$ directly without the use of a spatio-temporal point cloud so that the problem can be solved efficiently using a 2D Con-

volutional Neural Network (CNN). Such a deblurring problem using a single image has been studied extensively for traditional cameras for rectifying motion blurred photos. Our modified problem in 2D can be formulated as:

$$\underset{\mathcal{K}}{\operatorname{argmax}} \ \mathcal{C}\left(\mathcal{K} \circledast \mathcal{E}\right) \tag{B.3}$$

Here $\mathcal{K}$ is the heterogeneous deblur kernel and $\circledast$ is the convolution operator. However, estimating $\mathcal{K}$ directly is not constrained enough to be learned in an unsupervised manner. Instead, we formulate the deblurring problem inspired by Total Variation (TV) denoising to give us the final optimization problem as follows:

$$\underset{\overline{\mathcal{E}}}{\operatorname{argmax}} \ \mathcal{C}\left(\overline{\mathcal{E}}\right) + \lambda \underset{\overline{\mathcal{E}}}{\operatorname{argmin}} \mathcal{D}\left(\mathcal{E}, \overline{\mathcal{E}}\right) \tag{B.4}$$

where $\overline{\mathcal{E}}$ represents the deblurred event frame, λ is a regularization penalty and $\mathcal{D}$ represents a distance function to measure similarity between two event frames. Note that directly solving $\operatorname{argmax}_{\overline{\mathcal{E}}} \ \mathcal{C}\left(\overline{\mathcal{E}}\right)$ yields trivial solutions of high frequency noise.

To learn the function using a neural network we convert the argmax operator into an argmin operator as follows:

$$\underset{\overline{\mathcal{E}}}{\operatorname{argmin}} \ -\mathcal{C}\left(\overline{\mathcal{E}}\right) + \lambda \mathcal{D}\left(\mathcal{E}, \overline{\mathcal{E}}\right) \tag{B.5}$$

Intuitively, the higher the value of the contrast, the lower the value of the loss function, but going away too far from the input will penalize the loss function striking a balance between high contrast and similarity to the input image. We call our CNN which generates

the deblurred event images *EVDeBlurNet*. It takes as input $\mathcal{E}$ and outputs deblurred $\overline{\mathcal{E}}$. The network architecture is a simple encoder-decoder with four convolutional and four deconvolutional layers with batch normalization (Sec. B.3.6). Another benefit of the encoder decoder's lossy reconstruction is that it removes stray events (which are generally noise) and retains events corresponding to contours, thereby greatly increasing the signal to noise ratio.

Recently, [112] also presented a method for deblurring event frames to improve optical flow estimation via a coupling between predicted optical flow and sharpness in the event frame in the loss function. In contrast, our work presents a problem-independent deblurring network without the supervision from optical flow.

B.3.4 EVHomographyNet

A simple and computationally inexpensive way to obtain odometry on a quadrotor is to use a downfacing camera looking at a planar surface. This approximation coupled with data from an IMU and a distance sensor enables high speed "cheap" odometry for navigation. Recently, deep learning approaches have shown more robust homography estimation in traditional images [113, 114]. Inspired by this, we propose the first deep learning based solution to the problem of homography estimation using event cameras which can be run on an embedded computer at reasonably high speeds and good accuracy. Also, the added benefit of using a deep network for homography is that the tradeoff between speed and accuracy could be altered easily (by changing number of parameters). Let us mathematically formulate our problem statement. Let $\mathcal{E}_t$ and $\mathcal{E}_{t+1}$ be the event

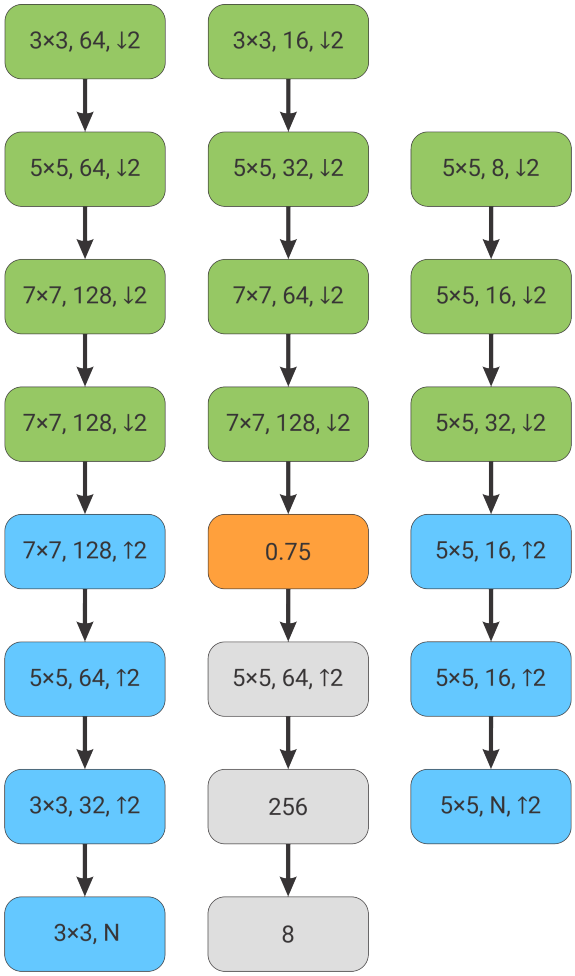


Figure B.5: Network Architectures used in the proposed pipeline. Left: EVDeblurNet, Middle: EVHomographyNet and Right: EVSegFlowNet. Green blocks show the convolutional layer with batch normalization and ReLU activation, cyan blocks show deconvolutional layer with batch normalization and ReLU activation and orange blocks show dropout layers. The numbers inside convolutional and deconvolutional layers show kernel size, number of filters and stride factor. The number inside dropout layer shows the dropout fraction. N is 3 and 6 respectively for EVDeblurNet when using losses $\mathcal{D}_1/\mathcal{D}_2$ and $\mathcal{D}_3$. N is 2 and 5 respectively for EVSegFlowNet when using losses $\mathcal{D}_1/\mathcal{D}_2$ and $\mathcal{D}_3$.

frames captured at times t and $t+1$, respectively, and $\delta t \ll \Delta t$ where Δt is the time difference between the start times of event frame accumulation. In the scenario presented before, the transformation between the two events frames is a homography. This can be written as $\mathbf{x}_{t+1} = \mathbf{H}_t^{t+1}\mathbf{x}_t$, where $\mathbf{x}_{t+1}, \mathbf{x}_t$ represent the homogeneous point correspondences in the two event frames and $\mathbf{H}_t^{t+1}$ is the resulting non-singular 3×3 homography

matrix between the two frames. We adapt the previous works on deep learning based homography estimation [113] [114] for both supervised and unsupervised flavors of deep learning based homography estimation. For the supervised flavor of the algorithm, we generate synthetic homography warped event frames and train them using the following loss function.

$$\underset{\widetilde{H}_{4Pt}}{\operatorname{argmin}} \ \mathbb{E}\left(\|\widetilde{H}_{4Pt} - \hat{H}_{4Pt}\|_2\right) \tag{B.6}$$

Here, $\widetilde{H}_{4Pt}$ and $\hat{H}_{4Pt}$ are the predicted and ground truth 4-point homographies. We refer the readers to [113] for more details.

For the unsupervised version, we adapt the mathematical formulation [114] for TensorDLT and the Spatial Transformer Network (STN) using bilinear interpolation. The final loss function is given as:

$$\underset{\widetilde{H}_{4Pt}}{\operatorname{argmin}} \ \mathbb{E}\left(\mathcal{D}\left(\mathcal{W}\left(\mathcal{I}_t, \widetilde{H}_{4Pt}\right), \mathcal{I}_{t+1}\right)\right) \tag{B.7}$$

where $\mathcal{W}$ is a generic differentiable warp function and can take on different mathematical formulations based on it's second argument (model parameters). In this case, $\mathcal{W}$ contains both the TensorDLT and the STN. As before, $\mathcal{D}$ represents a distance measuring image similarity between two event frames (Refer to the Sec. B.3.7) for the mathematical formulations of $\mathcal{D}$).

B.3.5 EVSegFlowNet

The end goal of this work is to detect/segment Independently Moving Objects (IMOs) and to dodge them. One could fragment this problem into two major parts, detecting IMOs, and subsequently estimating their motion to issue a control command to move away from the IMO in a safe manner. Let's start by discussing each fragment. Firstly, we want to segment the object using consecutive event frames $\mathcal{E}_t$ and $\mathcal{E}_{t+1}$. A simple way to accomplish this is by generating simulated data with known segmentation for each frame and then training a CNN to predict the foreground (IMO)/background segmentation. Such a CNN can be trained using a simple cross-entropy loss function as shown below.

$$\underset{p_f}{\operatorname{argmin}} \; -\mathbb{E}\left(\mathbb{1}_f \log\left(p_f\right) + \mathbb{1}_b \log\left(p_b\right)\right) \tag{B.8}$$

Here, $\mathbb{1}_f, \mathbb{1}_b$ are the indicator variables denoting if a pixel belongs to foreground or background. They are mutually exclusive, i.e., $\mathbb{1}_f = \neg\mathbb{1}_b$ and p_f, p_b represent the foreground and background predicted probabilities where $p_f + p_b = 1$. Note that each operation in the above equation is performed per pixel, and then an average over all pixels is computed. In the second step we want to estimate the IMO motion. Without any prior knowledge about the IMO it is impossible to estimate the 3D motion of the IMO from a monocular camera (event based or traditional). To make this problem tractable, we assume a prior about the object. More details can be found in Section B.5.

Once we have a prior about the object, we can estimate the 3D IMO motion using optical flow of the pixels corresponding to the IMO on the image plane. A simple way to

obtain optical flow is to train a CNN in a supervised manner. However, recent research has shown that these do not generalize well to new scenes/objects [115]. A better way is to use a self-supervised or completely unsupervised loss function.

$$\underset{\dot{\mathbf{x}}}{\operatorname{argmin}} \ \mathbb{E}\left(\mathcal{D}\left(\mathcal{W}\left(\mathcal{E}_t, \dot{\mathbf{x}}\right), \mathcal{E}_{t+1}\right)\right) \tag{B.9}$$

Here $\dot{\mathbf{x}}$ is the estimated optical flow between $\mathcal{E}_t \mapsto \mathcal{E}_{t+1}$ and $\mathcal{W}$ is a differentiable warp function based on optical flow and bilinear interpolation implemented using an STN. The self-supervised flavor of this algorithm [116] utilizes corresponding image frames instead of event frames for the loss function but the input is still the stack of event frames. One could utilize the two networks we talked about previously and solve the problem of dodging, however, one would need to run two neural networks for this purpose. Furthermore, this method suffers from a major problem: any unsupervised or self-supervised method can estimate rigid optical flow (optical flow corresponding to the background regions $\mathcal{B}$) accurately but the non-rigid optical flow (optical flow corresponding to the foreground regions $\mathcal{F}$) is not very accurate. This is an artifact because of the number of pixels corresponding to the foreground is often far less than that corresponding to the background, i.e., $\overline{\overline{\mathcal{F}}} \ll \overline{\overline{\mathcal{B}}}$. One would have to train for a lot of iterations to obtain accurate optical flow results on these foreground pixels which runs into the risk of overfitting to the dataset. This defeats the promise of self-supervised or unsupervised formulations.

To solve both the problems of complexity and accuracy, we formulate the problem using a semi-supervised approach to learn segmentation and optical flow at the same time, which we call *EVSegFlowNet*. We call the output of the network *segmentation flow*

denoted by $\tilde{\dot{\mathbf{p}}}$ which is defined as follows.

$$\tilde{\dot{\mathbf{p}}}_{\mathbf{x}} = \begin{cases} \dot{\mathbf{x}}, & \text{if } \mathbb{1}_f(\mathbf{x}) = 1 \\ \mathbf{0}, & \text{if } \mathbb{1}_b(\mathbf{x}) = 1 \end{cases} \tag{B.10}$$

One could intuit that we can obtain a noisy segmentation for free by simple thresholding on the magnitude of $\tilde{\dot{\mathbf{p}}}_{\mathbf{x}}$. To utilize the network to maximum capacity the input to the network is the ego-motion/odometry based warped event frame such that the background pixels in the two input event frames are almost aligned and the only misalignment comes from the IMOs. This ensures that the network's capacity can be utilized fully for learning sub-pixel accurate optical flow for IMO regions. The input to the EVSegFlowNet is $\mathcal{W}\left(\mathcal{E}_t, \widetilde{H}_{4Pt}\right)$ and $\mathcal{E}_{t+1}$. Here, $\widetilde{H}_{4Pt}$ is transformed to E^F before warping.

A complexity analysis of EVSegFlowNet is given in Sec. B.3.8. The success of our approach can be seen from the experimental results. The loss function for learning $\tilde{\dot{\mathbf{p}}}_{\mathbf{x}}$ is given below.

$$\begin{aligned} \underset{\tilde{\dot{\mathbf{p}}}_{\mathbf{x}}}{\operatorname{argmin}} \; & \mathbb{E}\left(\mathcal{D}\left(\mathcal{W}\left(\mathcal{E}'_t, \tilde{\dot{\mathbf{p}}}_{\mathbf{x}}\right) \circ \mathbb{1}_f, \mathcal{E}_{t+1} \circ \mathbb{1}_f\right)\right) + \\ & \lambda_1 \mathbb{E}\left(\|\tilde{\dot{\mathbf{p}}}_{\mathbf{x}} \circ \mathbb{1}_b\|_1\right) + \lambda_2 \mathbb{E}\left(\|\tilde{\dot{\mathbf{p}}}_{\mathbf{x}} \circ \mathbb{1}_b\|_2^2\right) \end{aligned} \tag{B.11}$$

Here, λ_1 and λ_2 are regularization parameters. This loss function is essentially the image difference with elastic net like regularization penalty. This penalty makes the network make background flow zero fairly quickly as compared to simple l_1 or quadratic

penalty whilst being robust to outliers (errors in segmentation mask creation). The loss functions are given in the next section.

Note that all our networks were trained in simulation and directly transfer to the real world without any re-training/fine-tuning. We call our dataset *Moving Object Dataset (MOD)*. Detailed information about the dataset can be found in Sec. B.4. Brief details about the dataset are presented next.

MOD contains data from multiple simulated scenes in Blender and provides ground truth for camera poses, IMO segmentation, object poses and depth. A sample image of a scene from the dataset is shown in Fig. B.3.

B.3.6 Network Details

In this Section, we will present the information on network architecture and training details.

The network architecture is shown in Fig. SB.5. Notice the simplicity in our network owing our performance to the approach of stacking multiple shallow networks to obtain good performance. *It must be noted that using advanced architectures might lead to better performance.* We leave this as an avenue for future work.

EVDeblurNet was trained for 200 epochs with a learning rate of 10^{-3} for 200 epochs with a batch size of 256 for losses using $\mathcal{D}_1$ and $\mathcal{D}_2$ and with a batch size of 32 for losses using $\mathcal{D}_3$. Also, the loss part associated with the contrast is scaled by a factor of 2.0 and the loss part associated with the distance is scaled by a factor of 1.0. This is equivalent to setting $\lambda = 0.33$.

EVSegNet, EVFlowNet, and EVSegFlowNet were trained for 50 epochs with a learning rate of 10^{-4} and a batch size of 256. EVHomographyNet was trained for 200 epochs with learning rate 10^{-4} and a batch size of 256.

For all the networks, the event frames $\mathcal{E}$ were normalized by dividing each pixel value by 255 and then subtracting by 0.5 and finally scaling by 2.0 to bound all values between $[-1, 1]$.

For networks using $\mathcal{D}_3$ loss, the values of α are output by the networks last n channels. Here n denotes the number of input channels per image (3 in our case of event frame). $N = 2n$ when using $\mathcal{D}_3$ loss.

B.3.7 Loss Functions

In this Section, we present the mathematical formulations for variants of the loss functions used in this work.

The two flavors of the heuristic Contrast function $\mathcal{C}$ used in Section II-A of the main paper is inspired by [11] are given below (denoted by $\mathcal{C}_1$ and $\mathcal{C}_2$).

$$\mathcal{C}_1\left(\mathcal{E}\right) = \mathbb{E}\left(\|\text{Var}\left(\nabla\mathcal{E}\right)\|_1\right)$$

$$\text{Var}\left(\mathcal{E}\right) = \mathbb{E}\left(\mathcal{E}\left(\mathbf{x}\right) - \mathbb{E}\left(\mathcal{E}\right)\right)$$

$$\mathcal{C}_2\left(\mathcal{E}\right) = \mathbb{E}\left(\|\nabla\mathcal{E}\|_1\right)$$

where $\nabla = \begin{bmatrix}\nabla_x & \nabla_y\end{bmatrix}^T$ is the 2D gradient operator (sobel in our case), Var is the variance operator and $\mathbf{x}$ denotes the pixel location. *The key difference from [11] is that we use the variance operator on the gradients instead on raw values as empirically this gave us*

better results and was more stable during training.

Mathematical formulations of the different variants of the distance function $\mathcal{D}$ used in Sections II-A, II-B and II-C of the main paper which measures the similarity between two event frames are given below (denoted as $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$).

$$\mathcal{D}_1\left(\mathcal{E}_1, \mathcal{E}_2\right) = \mathbb{E}\left(\|\mathcal{E}_1 - \mathcal{E}_2\|_1\right)$$

$$\mathcal{D}_2\left(\mathcal{E}_1, \mathcal{E}_2 | \, \alpha, \epsilon\right) = \mathbb{E}\left(\left(\left(\mathcal{E}_1 - \mathcal{E}_2\right)^2 + \epsilon^2\right)^{\alpha}\right)$$

$$\mathcal{D}_3\left(\mathcal{E}_1, \mathcal{E}_2 | \, \alpha, \epsilon, c\right) = \frac{b}{d}\left(\left(\frac{\left({}^{\left(\mathcal{E}_1 - \mathcal{E}_2\right)}/_{c}\right)^2}{b} + 1\right)^{{}^{d}/_{2}} - 1\right)$$

$$b = \|2 - \hat{\alpha}\|_1 + \epsilon; \qquad d = \begin{cases} \hat{\alpha} + \epsilon & \text{if } \hat{\alpha} \geq 0 \\ \hat{\alpha} - \epsilon & \text{if } \hat{\alpha} < 0 \end{cases}$$

$$\hat{\alpha}_i = \left(2 - 2\epsilon_\alpha\right)\frac{e^{\alpha_i}}{e^{\alpha_i} + 1} \quad \forall i$$

Here, $\mathcal{D}_1$ is the generic l_1 photometric loss [117] commonly used for traditional images, $\mathcal{D}_2$ is the Chabonnier loss [118] commonly used for optical flow estimation for traditional images and $\mathcal{D}_3$ is the robust loss function presented in [119]. In $\mathcal{D}_3$, the value of α is output from the network (Refer to B.3.6 for architecture details).

B.3.8 Compression Achieved by using EVSegFlowNet

Now, let's analyze the complexity of EVSegFlowNet as compared to a combination of separate segmentation and flow networks we call EVSegNet and EVFlowNet respectively. Let $\mathcal{O}$ denote the complexity measure as the minimum number of neu-

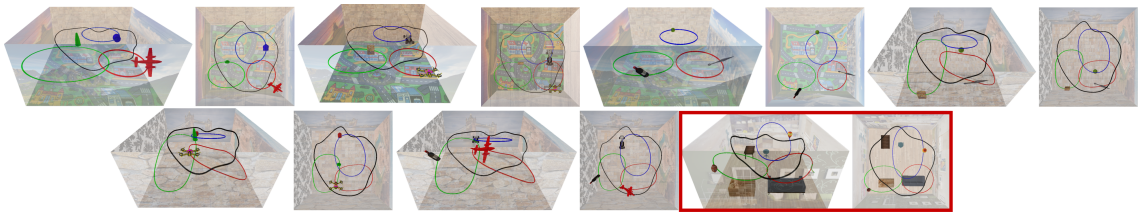


Figure B.6: Various Scene setups used for generating data. Red box indicates the scene used for generating out of dataset testing data to evaluate generalization to novel scenes.

Figure B.7: Moving objects used in our simulation environment. Left to right: ball, cereal box, tower, cone, car, drone, kunai, wine bottle and airplane. Notice the variation in texture, color and shape. *Note that the objects are not presented to scale for visual clarity.*

rons to obtain a satisfactory generalization performance on a specific task. We also assume that for smaller and shallow networks complexity scales with number of neurons almost linearly. The complexity for a combination of EVSegNet and EVFlowNet is given by $\mathcal{O}(S) + \mathcal{O}(F)$. Let the complexity of segmentation and flow obtained by EVSegFlowNet be $\mathcal{O}(\widetilde{S})$, $\mathcal{O}(\widetilde{F})$ respectively. A compression/speedup is achieved when $\frac{\mathcal{O}(\widetilde{S}) + \mathcal{O}(\widetilde{F})}{\mathcal{O}(S) + \mathcal{O}(F)} < 1$. Now, because we are only estimating flow for foreground pixels in EVSegFlowNet we have $\frac{\mathcal{O}(\widetilde{F})}{\mathcal{O}(F)} \approx \frac{\overline{\overline{\mathcal{F}}}}{\overline{\overline{\mathcal{B}}}}$. Also, as we mentioned before we get segmentation for free from EVSegFlowNet hence $\mathcal{O}(\widetilde{S}) \approx 0 \ll \mathcal{O}(S)$. This also implies that we achieved good compression/speedup by our formulation as $\frac{\mathcal{O}(\widetilde{S}) + \mathcal{O}(\widetilde{F})}{\mathcal{O}(S) + \mathcal{O}(F)} \ll 1$.

B.4 Multi Moving Object Event Dataset

Extensive and growing research on visual (inertial) odometry or SLAM have lead to the development of a large number of datasets. Recent adaption of deep learning

Figure B.8: Random textures used in our simulation environment

to solve these aforementioned problems have fostered the development of large scale datasets (large amount of data). However, most of these datasets are built with the fundamental assumption of static scenes in mind and as a manifestation of which moving or dynamic objects are often not included in these datasets [120, 121, 122].

To this end, we propose to use synthetic scenes for generating "unlimited" amount of training data with one or more moving objects in the scene. We accomplish this by adapting and proliferating the simulator presented in [121]. To incubate generalization to novel scenes and to utilize the algorithm trained on simulation directly in the real world, we create synthetic moving objects which vary significantly in their texture, shape and trajectory. We also choose random textures for the walls of the 3D room in which objects will move about.

To generate data, we randomize wall textures, objects and object/camera trajectories to obtain seven unique configurations out of which one is exclusively used for test of generalization on more complex structures. Each configuration has a room with three objects moving as shown in Fig. SB.6. Images are rendered at 1000 frames per second at a resolution of 346×260 and a field of view of 90° for each configuration. Using these images, events are generated following the approach described in [121]. Later event frames $\mathcal{E}$ are generated with three different integration times δt of $\{1, 5, 10\}$ ms. Details

Figure B.9: Different textured carpets laid on the ground during real experiments to aid robust homography estimation from EVHomographyNet.

about the room, lighting and objects are given next.

B.4.1 3D room and moving objects

Each room is of size $10 \times 10 \times 5$ m and has random textures on all the walls. These random textures consist of different patterns, colors and shapes. These textures mimic those which occur in real-world indoor and outdoor environments such as skyscrapers, flowers, landscape, bricks, wood, stone and carpet. Each room contains seven light sources inside it for uniform illumination.

The camera is moved inside the 3D room on trajectories such that almost all possi-

ble combinations of rotation and translation are obtained. This is aimed at replicating the movement which could be encountered on a real quadrotor.

We have three Independently Moving Objects (IMOs) in each room. Each object is unique in color, shape, texture and size. The objects are chosen to range from simple shapes and textures to complex ones. The objects chosen are ball, cereal box, tower, cone, car, drone, kunai, wine bottle and airplane. The trajectories of the objects are chosen such that many different combinations of relative pose between the camera and the objects are encountered. Also, the objects are moving ten times faster than the camera simulating objects being thrown at a hovering or a slow moving (drifting) quadrotor. The wall textures and moving objects are shown in Figs. SB.7 and SB.8 respectively.

B.4.2 Dataset for EVDeblurNet

To learn a simple deblur function, we obtaian data from a down facing camera looking at a planar texture and such that no moving objects appear in the frame. The textures for the floor are chosen to replicate the common floor patterns such as wooden flooring, stone, kid's play carpet, and tiles. A total of 15K event frames corresponding to five different textures and integration times δt of $\{1, 5, 10\}$ ms are obtained. Random crops of 128×128 are used to train the network.

B.4.3 Dataset for EVSegNet, EVFlowNet and EVSegFlowNet

In this dataset, the camera follows the same trajectory given in Section B.4.2 to capture the moving objects in the 3D environment. The camera is moving approximately

at 0.005 m per frame and moving objects are ranging from 0.05 to 0.06 m per frame. There are some instances where moving objects collide with the camera. We specifically included these scenarios in the dataset so that the learning approaches could learn utilizing both small and large changes in object appearances between consecutive event frames. Average of two objects per frame is captured from the camera (minimum of 0 objects to maximum of 3 objects). Using the six scenarios, 70K event frames are obtained (including data from integration times of $\{1, 5, 10\}$ ms). Event frames from two random integration times per scenario are chosen for training. We obtain 60K images for training and 10K images for testing (we only use 1 ms data not used for training for testing due to mask alignment errors at higher integration times). During training, we use frame skips of one to four to faciliate variable baseline learning of flow and segmentation. We call this test set as "in dataset" testing because the test set though differs significantly in appearance due to integration times still contains the same objects and textures as the training set.

For measuring the amount of generalization of our approach, we created a more complex and completely different scenario for testing. This scenario contains immovable 3D objects such as table, chair and a box to 3D room with different textures (different per wall and different from training set textures). The textures on the wall are more realistic depicting a real indoor environment (Refer to the scenario in the red box in Fig. B.6). We particularly designed such a scene to highlight that our network is mostly learning from contours and motion information of the objects which is agnostic to scene appearance. Here, we use integration times δt of $\{1, 2\}$ ms. We obtain 6K frames for "out of dataset" testing.

B.4.4 Dataset for EVHomographyNet

To train EVHomographyNet, we use the same training set from EVSegFlowNet with the major difference being that here only one frame is used at a time. A random patch of size 128×128 is obtained from the 346×260 frame. Then a random perturbation between $\pm\gamma$ is applied to each of the corners. This is used to obtain the homography warped event frame. This approach is exactly the same as given in [113, 114]. For testing "in dataset" the center crops of all the images used for training are chosen and random perturbations of different $\pm\gamma$ are applied. (Refer to Table I of the main paper).

For "out of dataset" testing a similar treatment is given to the out of dataset used for evaluating EVSegFlowNet.

B.5 Control Policy for Dodging Dynamic Obstacles

In this section, we present a solution for evading multiple known and/or unknown IMOs.

Let us consider three different flavors of IMOs: (a) Sphere with known radius r, (b) Unknown shaped objects with known bound on the size and (c) Unknown objects with no prior knowledge. We tackle each of these cases differently. Knowing the prior information about the geometric nature helps us achieve much more robust results and fine-grain control.

We define $\mathcal{F}$ as the projection of all the IMOs on the image plane such that $\mathcal{F} = \bigcup_{\forall i} \mathcal{F}_i$, where $\mathcal{F}_i$ denotes the i^{th} IMO's image plane projection. Now, let's discuss each flavor of the problem separately in the following subsections.

B.5.1 Sphere with known radius r

Let us first begin with the simplest case, i.e., *a single spherical IMO with known radius* r. On an undistorted image the projection of a sphere on the image plane is an ellipse [123]. Evading such an object under no gravitational influence has been tackled and well analyzed by [96]. For spherical objects under the gravitational influence, we estimate the initial 3D position using the known radius information and then we track the object over a few $\mathcal{E}$ to obtain the initial 3D velocity. Here, the tracking is done by detection (segmentation) on every frame.

Assuming a classical physics model, we predict the future trajectory $\mathbf{X}_i^{\text{IMO}}$ of the sphere when it is only under the influence of gravity. Now, we define the point $\mathbf{X}_{i,p}^{\text{IMO}}$ as the intersection of the trajectory $\mathbf{X}_i^{\text{IMO}}$ and the image plane. For the case of a single spherical IMO, we compute the distance between $\mathbf{X}_{i,p}^{\text{IMO}}$ and the initial position of the quadrotor O, denoted by vector $\mathbf{x}_{\text{min}} \in \mathbb{R}^{2\times 1}$. The "safe" direction is represented as $\mathbf{x}_s = -\mathbf{x}_{\text{min}}$. A simple Proportional-Integral-Derivative (PID) controller based on the differential flatness model of the quadrotor is used with high proportional gain for a quick response to move in the direction $\mathbf{x}_s$. The minimum amount of movement is equal to the extended obstacle size (the size of the quadrotor is added to the object size).

Now, let's extend to the evasion of *multiple spherical IMOs*. We assume that while objects are detected, there is no occlusion among different IMOs in the front event camera frame. Then, each object $\mathcal{F}_i$ is clustered using mean shift clustering. For each object $\mathcal{F}_i$, the 3D position and velocity are estimated as before. It is important to note that since all the objects were targeted at the quadrotor, they are bound to intercept the image plane,

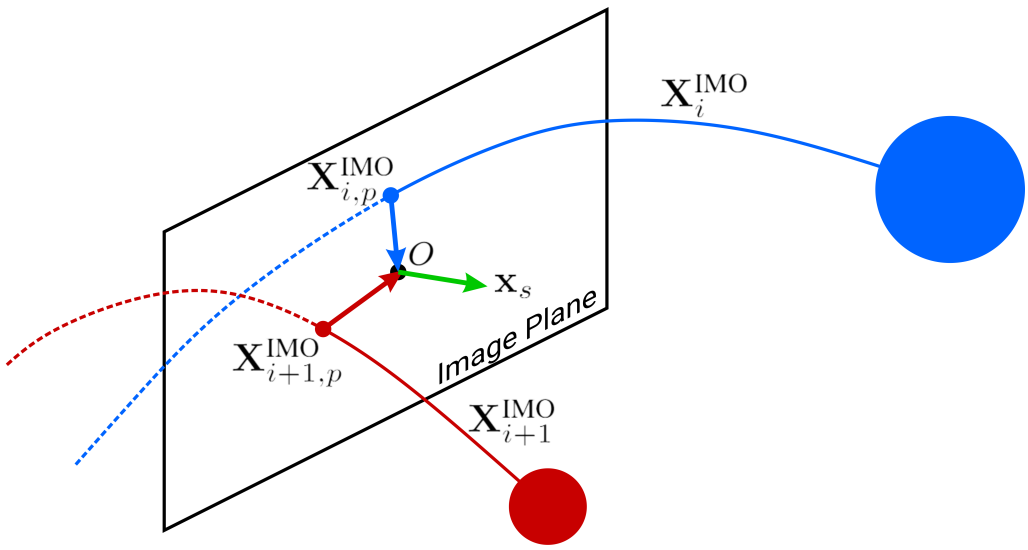


Figure B.10: Vectors $\mathbf{X}_{i,p}^{\text{IMO}}$ and $\mathbf{X}_{i+1,p}^{\text{IMO}}$ represent the intersection of the trajectory and the image plane. $\mathbf{x}_s$ is the direction of the "safe" trajectory. All the vectors are defined with respect to the center of the quadrotor projected on the image plane, O. Both of the spheres are of known radii.

say at point $\mathbf{X}_{i,p}^{\text{IMO}}$ (Fig. B.10). For evasion from multiple objects, we adapt the following approach. First, we find the two objects m and $m+1$ from a consecutive cyclic pair of vectors such that (here $\hat{(\cdot)}$ represents a unit vector):

$$\underset{\mathbf{X}_{i,p}^{\text{IMO}}, \mathbf{X}_{i+1,p}^{\text{IMO}}}{\operatorname{argmin}} \left\langle \hat{\mathbf{X}}_{i,p}^{\text{IMO}}, \hat{\mathbf{X}}_{i+1,p}^{\text{IMO}} \right\rangle \tag{B.12}$$

In other words, the objects m and $m+1$ forms the largest angle among all the consecutive cyclic pairs. So we deploy a strategy to move the quadrotor in $\mathbf{x}_s$ direction in the image plane such that

$$\mathbf{x}_s = \begin{cases} -\hat{\mathbf{X}}_\beta, & \text{if } \max_{\forall i} \langle \hat{\mathbf{X}}_\beta, \mathbf{X}_i \rangle < \max_{\forall i} \langle -\hat{\mathbf{X}}_\beta, \hat{\mathbf{X}}_i \rangle \\ \hat{\mathbf{X}}_\beta, & \text{otherwise} \end{cases} \tag{B.13}$$

where $\mathbf{X}_\beta = \hat{\mathbf{X}}_{m,p}^{\text{IMO}} + \hat{\mathbf{X}}_{m+1,p}^{\text{IMO}}$

B.5.2 Unknown shaped objects with bound on size

Now, consider the case of evading an IMO of an arbitrary shape $\mathcal{S}$. As the projection of $\mathcal{S}$ on the image plane can be either convex or non-convex, we first obtain the convex hull of $\mathcal{S}$ denoted by $\mathcal{H}$. Clearly, an evasive maneuver performed using $\mathcal{H}$ guarantees evasion from the object when the rotation of the IMO with respect to the camera is small.

Next, we find the principal axes of the projection of $\mathcal{H}$ on the image plane. Because we have a bound on size, i.e., we have a bound on the length of the maximum principle axis in 3D, we can evade this object assuming it to be a sphere of this diameter. Note that this method is more conservative than the previous approach constraining the sensing range and latency based on how close the bound is to actual object size.

B.5.3 Unknown objects with no prior knowledge

Without any prior knowledge about the object, it is geometrically infeasible to obtain the 3D velocity of an IMO using a monocular camera. However, we can predict a possible safe trajectory $\mathbf{x}_s$ depending on the velocity direction of the IMOs on the image plane. We compute the unit vector $\mathbf{v}_i^{\text{IMO}}$ in which the IMO is moving by tracking the segmentation mask of the IMO or by computing the mean optical flow direction of the region of interest. For a single unknown IMO, a heuristic is chosen such that the quadrotor moves in the direction perpendicular to the velocity of the IMO on the image plane, i.e., a safe direction for the quadrotor motion which satisfies $\langle \mathbf{x}_s, \mathbf{v}_i^{\text{IMO}} \rangle = 0$.

For evasion from multiple objects, we adapt a similar approach as in Section B.5.1. First, we find the two objects m and $m+1$ from a consecutive cyclic pair of velocity

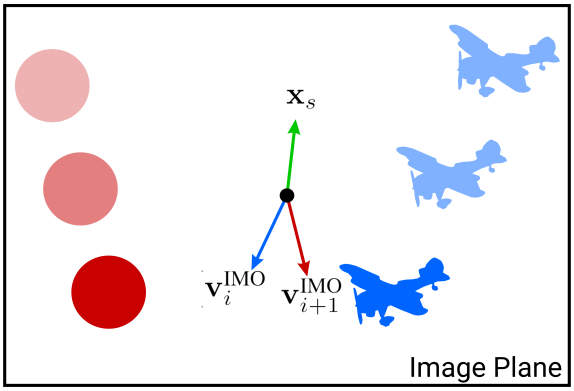


Figure B.11: Representation of velocity direction of multiple unknown IMOs. The vector $\mathbf{v}_i^{\text{IMO}}$ and $\mathbf{v}_{i+1}^{\text{IMO}}$ represent velocities of the corresponding objects. $\mathbf{x}_s$ denotes the "safe" direction for the quadrotor.

vectors such that (Fig. B.11):

$$\underset{\mathbf{v}_{i,p}^{\text{IMO}}, \mathbf{v}_{i+1,p}^{\text{IMO}}}{\operatorname{argmin}} \left\langle \hat{\mathbf{v}}_{i,p}^{\text{IMO}}, \hat{\mathbf{v}}_{i+1,p}^{\text{IMO}} \right\rangle \tag{B.14}$$

We deploy a strategy to move the quadrotor in $\mathbf{x}_s$ direction in the image plane such that

$$\mathbf{x}_s = \begin{cases} -\hat{\mathbf{v}}_\beta, & \text{if } \max_{\forall i} \langle -\hat{\mathbf{v}}_\beta, \hat{\mathbf{v}}_i \rangle < \max_{\forall i} \langle \hat{\mathbf{v}}_\beta, \hat{\mathbf{v}}_i \rangle \\ \hat{\mathbf{v}}_\beta, & \text{otherwise} \end{cases} \tag{B.15}$$

where $\mathbf{v}_\beta = \hat{\mathbf{v}}_{m,p}^{\text{IMO}} + \hat{\mathbf{v}}_{m+1,p}^{\text{IMO}}$

B.5.4 Pursuit: A reversal of evasion?

The generality of our perception stack for navigation is demonstrated by showing that pursuit can be accomplished using a simple reversal of the control policy for the cases presented in B.5.1 and B.5.2.

Figure B.12: Objects used in experiments. Left to right: Airplane, car, spherical ball and Bebop 2.

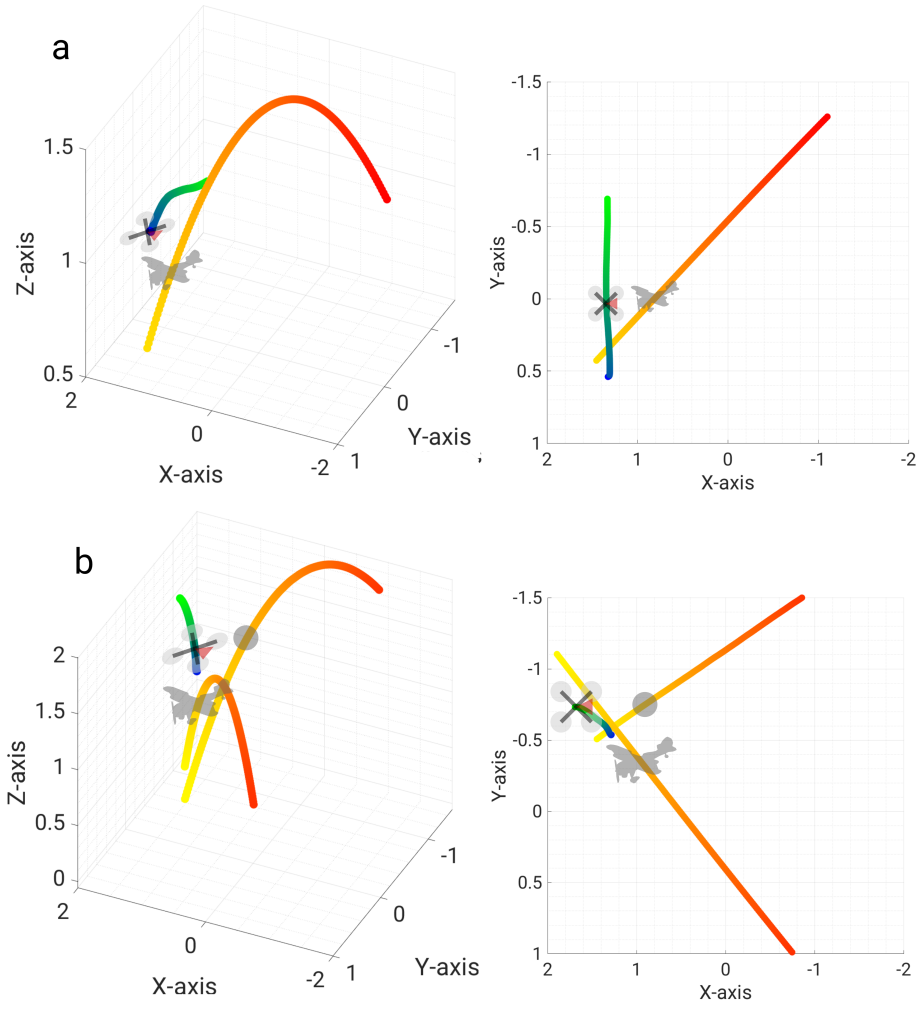


Figure B.13: Vicon estimates for the trajectories of the objects and quadrotor. (a) Perspective and top view for single unknown object case, (b) perspective and top view for multiple object case. Object and quadrotor silhouettes are shown to scale. Time progression is shown from red to yellow for objects and blue to green for the quadrotor.

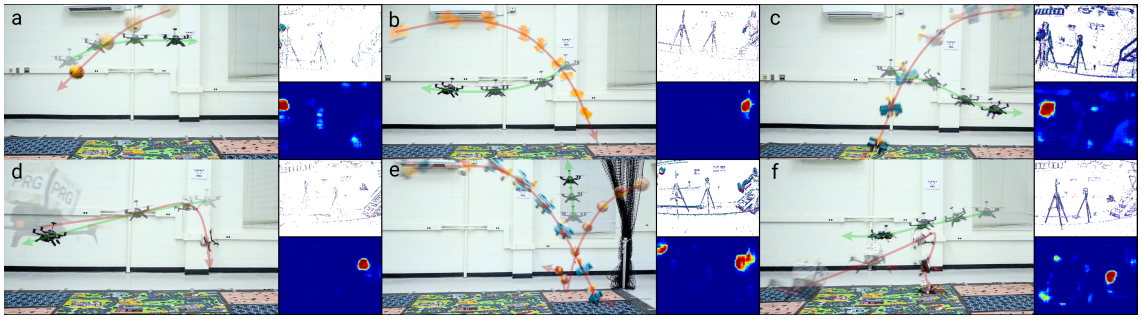


Figure B.14: Sequence of images of quadrotor dodging or pursuing of objects. (a)-(d): Dodging a spherical ball, car, airplane and Bebop 2 respectively. (e): Dodging multiple objects simultaneously. (f): Pursuit of Bebop 2 by reversing control policy. Object and quadrotor transparency show progression of time. Red and green arrows indicate object and quadrotor directions respectively. On-set images show front facing event frame (top) and respective segmentation obtained from our network (down).

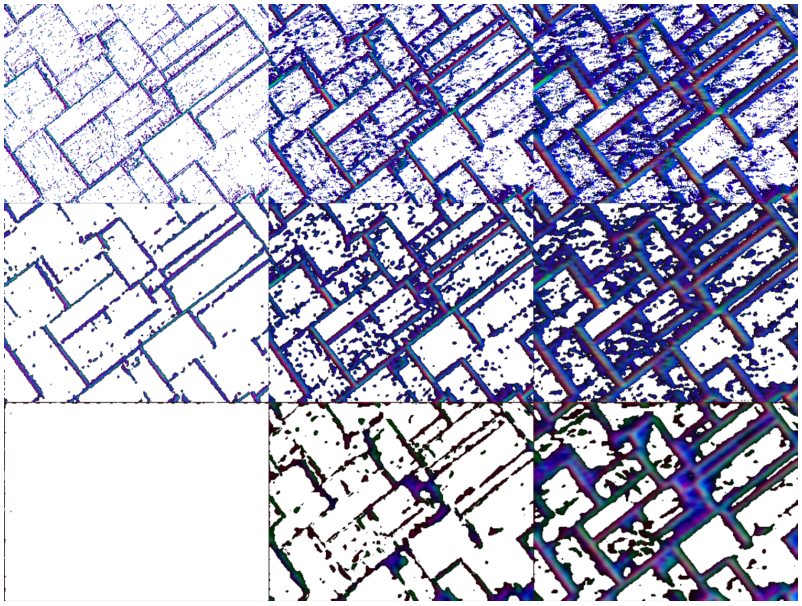


Figure B.15: Output of EVDeBlurNet for different integration time and loss functions. Top row: raw event frames, middle row: deblurred event frames with $\mathcal{D}_2$ and bottom row: deblurred event frames with $\mathcal{D}_3$ with δt. Left to right: δt of 1 ms, 5 ms and 10 ms. Notice that only the major contours are preserved and blurred contours are thinned in deblurred outputs.

Additionally, for an IMO which is self-propelled like a quadrotor, one can perform both pursuit and evade tasks by assuming a linear motion model. Note that here no concept of the agent's intent is used but it can be introduced with an additional neural network

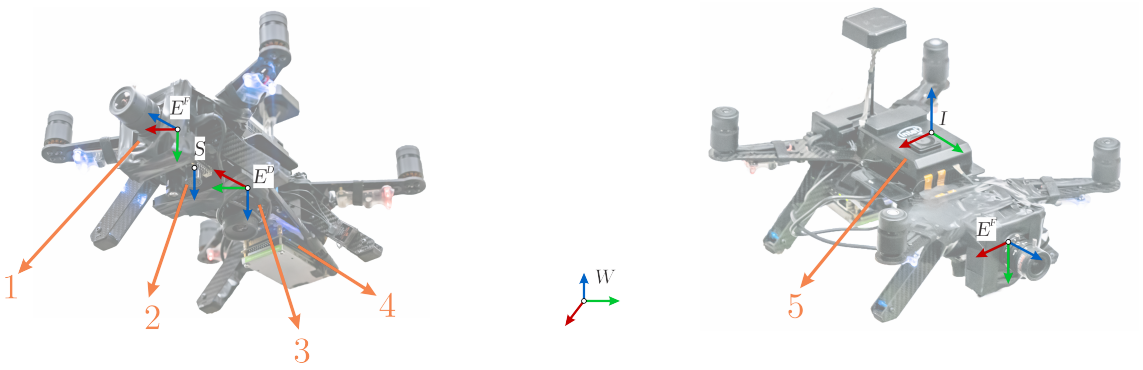


Figure B.16: Representation of coordinate frames on the hardware platform used. (1) Front facing DAVIS 240C, (2) down facing sonar on PX4Flow, (3) down facing DAVIS 240B, (4) NVIDIA TX2 CPU+GPU, (5) Intel® Aero Compute board.

for predicting the motion model of the agent (intent) [124]. We leave this for future work.

In the next section, we provide a detailed experimental analysis and present our results for all the aforementioned cases.

B.6 Experiments

B.6.1 Experimental Setup

The proposed framework was tested on a modified Intel® Aero Ready to Fly Drone. The Aero platform was selected for its rugged carbon fiber chassis and integrated flight controller running the PX4 flight stack.

For our experiments, we mounted a front facing DAVIS 240C event camera mated to a 3.3 - 10.5 mm $f/1.4$ lens set at 3.3 mm giving us a diagonal Field Of View (FOV) of 84.5°, a downfacing DAVIS 240B event camera mated to a 4.5 mm $f/1.4$ lens giving us a diagonal FOV of 67.4° and a down facing PX4Flow sensor for altitude measurements. Additionally, we obtain inertial measurments from the IMU on the flight controller. We

also mounted an NVIDIA Jetson TX2 GPU to run all the perception and control algorithms on-board (Fig. SB.16). All the communications happen over serial port or USB. The takeoff weight of the flight setup including the battery is 1400 g with dimensions being $330 \times 290 \times 230$ mm. This gives us a maximum thrust to weight ratio of 1.35.

All the neural networks were prototyped on a PC running Ubuntu 16.04 with and Intel® Core i7 6850K 3.6GHz CPU, an NVIDIA Titan-Xp GPU and 64GB of RAM in Python 2.7 using TensorFlow 1.12. The final code runs on-board the NVIDIA Jetson TX2 running Linux for Tegra® (L4T) 28.1. All the drivers for creating event frames and sensor fusion are written in C++ for efficiency and all the neural network codes run on the TX2's GPU in Python 2.7. We obtain a flight time of about 3 mins.

To enable robust homography estimation, we laid down carpets of different textures on the ground to obtain strong contours in event frames (Refer to Fig. SB.9).

The experiments were conducted in the Autonomy Robotics and Cognition (ARC) lab's indoor flying space at the University of Maryland, College Park. The total flying volume is about $6 \times 5.5 \times 3.5$ m^3. A Vicon motion capture system with 8 vantage V8 cameras are used to obtain ground truth at 100 Hz. The objects were either thrown or flown (in-case of the bebop experiment) at the quadrotor during hover or slow flight (simulating slow drift) at speeds ranging from 4.4 ms^{-1} to 6.8 ms^{-1} from a distance ranging from 3.6 m to 5.2 m. To enable robust homography estimation, we laid down carpets of different textures on the ground to obtain strong contours in event frames (Refer to supplementary Fig. S6).

We used four different objects in our experiments, (a) a spherical ball of diameter 140 mm, (b) a car of size $185 \times 95 \times 45$ mm (here a bound of 240 mm is used), (c) an

airplane of size $270 \times 250 \times 160$ mm (size information not used in experiments), (d) a Bebop 2 of size $330 \times 380 \times 200$ mm. Also, we used an integration time δt of 30 ms for all our experiments.

B.6.2 Experimental Results and Discussion

In this paper, we considered the case of navigating through different sets of multiple dynamic obstacles. We dealt with six different evading combinations and one pursuit experiment: (a) Spherical ball with a known radius of 140 mm, (b) car with a bound on the maximum dimension size of 240 mm (with maximum error of $\sim 20\%$ from the original size), (c) airplane with no prior information, (d) Bebop 2 flying at a constant velocity, (e) multiple unknown objects, (f) pursuit of Bebop 2 and (g) low-light dodging experiment. For each evasion case, the objects are directly thrown towards the Aero quadrotor such that a collision would definitely occur if the Aero holds its initial position. The objects used in the experiments are shown in Fig. B.12. For each case, a total of 30 trials were conducted. The Vicon plots for cases (c) and (e) are shown in Fig. B.13. Notice that the objects would have hit the quadrotor if it had not moved from its initial position. We achieved a remarkable success rate of 86% in cases (a) and (b), 76% in case (c). Both Parrot Bebop 2 experiments (case (d), (f)) resulted in 83% success rate. Case (e) was carefully performed with synchronized throws between the two objects and resulted about 76% success rate. For the low-light experiment (case (g)), we achieved a success rate of 70%. Here success is defined as both a successful detection and evasion for the evade experiments and both a successful detection and collision for the pursuit task. Fig.

B.14 shows sequences of images for cases (a)-(f) along with sample front facing event frame and segmentation outputs.

Before the IMO is thrown at the quadrotor, the quadrotor maintains its position (hover) using the differential X^W and Y^W estimates from the EVHomographyNet and Z^W estimates from the sonar.

When the IMO is thrown at the quadrotor, the IMO is detected for five consecutive frames to estimate either the trajectory or image plane velocity and to remove outliers using simple morphological operations. This gives a perception response lag of 60 ms (each consecutive frame pair takes 10 ms for the neural network computation and 2 ms for the post-processing). Finally, the quadrotor moves using the simple PID controller presented before.

Note that, we talked about obtaining both segmentation and optical flow from EVSegFlowNet. This was based on the conceptualization of optical flow being used for other tasks as well. However, if only the dodging task is to be performed, a smaller segmentation network can be used without much loss of accuracy.

Table B.1: Quantitative evaluation of different methods for Homography estimation.

	RMSE$_i$ in px.					RMSE$_o$ in px.				
Method (Loss)	$\gamma = \pm[0,5]$	$\gamma = \pm[6,10]$	$\gamma = \pm[11,15]$	$\gamma = \pm[16,20]$	$\gamma = \pm[21,25]$	$\gamma = \pm[0,5]$	$\gamma = \pm[6,10]$	$\gamma = \pm[11,15]$	$\gamma = \pm[16,20]$	$\gamma = \pm[21,25]$
Identity	3.92 ± 0.83	11.40 ± 0.70	18.43 ± 0.70	25.50 ± 0.70	32.55 ± 0.71	3.92 ± 0.84	11.40 ± 0.70	18.44 ± 0.71	25.49 ± 0.70	32.55 ± 0.71
S	3.23 ± 1.13	3.90 ± 1.34	5.31 ± 2.05	9.63 ± 4.57	17.65 ± 7.00	4.15 ± 1.78	5.05 ± 2.19	6.99 ± 3.11	11.21 ± 4.84	18.37 ± 6.61
US* ($\mathcal{D}_1$)	2.97 ± 1.22	3.84 ± 1.61	5.99 ± 2.78	11.64 ± 5.69	20.36 ± 7.68	3.92 ± 1.53	5.31 ± 2.43	8.14 ± 3.86	13.63 ± 5.87	21.22 ± 7.35
US* ($\mathcal{D}_2$)	2.48 ± 0.93	3.53 ± 1.43	5.89 ± 2.70	11.74 ± 5.69	20.51 ± 0.70	3.19 ± 1.26	4.86 ± 2.31	7.92 ± 3.73	13.47 ± 5.71	21.22 ± 7.08
DB + S	2.73 ± 1.01	3.16 ± 1.23	$\mathbf{4.00 \pm 1.79}$	$\mathbf{6.50 \pm 3.54}$	$\mathbf{12.22 \pm 6.58}$	3.69 ± 1.51	$\mathbf{4.49 \pm 2.10}$	$\mathbf{5.91 \pm 3.16}$	$\mathbf{9.04 \pm 4.90}$	$\mathbf{14.60 \pm 6.95}$
DB + US ($\mathcal{D}_1$)	$\mathbf{2.19 \pm 0.88}$	$\mathbf{3.04 \pm 1.57}$	4.99 ± 2.75	10.16 ± 5.54	18.62 ± 7.85	$\mathbf{3.08 \pm 1.37}$	4.63 ± 2.68	7.57 ± 4.30	13.16 ± 6.25	21.08 ± 7.49
DB + US ($\mathcal{D}_2$)	2.41 ± 1.06	3.30 ± 1.77	5.36 ± 3.02	10.39 ± 5.78	18.77 ± 8.07	3.35 ± 1.76	5.05 ± 3.03	8.11 ± 4.65	13.46 ± 6.48	21.08 ± 7.81

* Trained for 100 epochs on supervised and then fine-tuned on unsupervised for 100 more epochs. γ denotes the perturbation range in px. for evaluation.

Fig. B.15 shows the input and output of EVDeBlurNet for losses $\mathcal{D}_2$ and $\mathcal{D}_3$ under $\delta t = \{1, 5, 10\}$ ms. Observe the amount of noise (stray events not associated with strong contours) in the raw images (top row of Fig. B.15). The second row shows the output

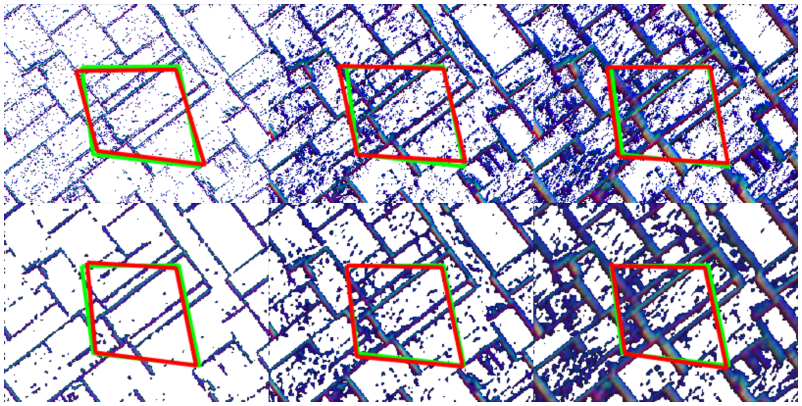


Figure B.17: Output of EVHomographyNet for raw and deblurred event frames at different integration times. Green and red color denotes ground truth and predicted $\widetilde{H}_{4Pt}$ respectively. Top row: raw events frames and bottom row: deblurred event frames. Left to right: δt of 1 ms, 5 ms and 10 ms. Notice that the deblurred homography outputs are almost not affected by integration time.

of EVDeBlurNet for $\mathcal{D}_2$ loss. Observe that this works well for smaller integration times but for larger integration times, the amount of denoising and deblurring performance deteriorates. However, $\mathcal{D}_3$ loss which is aimed at outlier rejection is more suppressive of weak contours and hence one can observe that the frame has almost no output for smaller integration times. This has the effect of working well for larger integration times.

Fig. B.17 shows the output of EVHomographyNet using the supervised loss function on both raw (top row) and deblurred frames (bottom row). Observe that the deblurred homography estimation is more robust to changes in different integration times. Table B.1 (extended table available in Supplementary section S.VII.) shows the quantitative evaluation of different methods used for training EVHomographyNet. Here, DB represents deblurring using the combination of $\mathcal{D}_2$ and $\mathcal{C}_2$ loss, S and US refer to supervised and unsupervised losses respectively. RMSE_i and RMSE_o denote the average root mean square error [114] in the testing dataset with textures similar to that of the training set, and completely novel textures respectively. RMSE_o quantifies how well the network

can generalize to unseen samples. Notice that the supervised flavors of the algorithm work better (lower RMSE_i and RMSE_o) than their respective unsupervised counterparts. We speculate that this is because of the sparsity in data and that the simple image based similarity metrics not being well suited for event frames. We leave crafting a novel loss function for event frames as a potential avenue for future work. Also, notice how deblur variants of the algorithms almost always work better than their respective non-deblurred counterparts highlighting the utility of EVDeblurNet.

Table B.2: Quantitative evaluation of different methods for Segmentation of IMO.

Method (Loss)	DR_i in %	DR_o in %	Run Time in ms	FLOPs in M	Num. Params in M
SegNet	49.0	40.4	1.5	222	0.03
DB + SegNet	81.5	68.7	7.5	4900	2.33
DB + H + SegNet	83.2	69.1	10	5150	3.63
SegFlowNet	88.3	81.9	1.5	222	0.03
DB + SegFlowNet	93.3	90.1	7.5	4900	2.33
DB + H + SegFlowNet	93.4	90.7	10	5150	3.63

Table B.2 shows the quantitative results of different variants of segmentation networks trained using the $\mathcal{D}_2$ loss for SegFlowNet. Also, H denotes the stack of warped $\mathcal{E}_t$ and $\mathcal{E}_{t+1}$ using the outputs of the network DB + S in Table B.1. Here DR denotes the detection rate and is defined as:

$$\text{DR} = \mathbb{E}\left(\frac{\overline{\overline{(\mathcal{D} \cap \mathcal{G}) \circ \mathbb{1}_{\mathcal{E}}}}}{\overline{\overline{(\mathcal{G} \circ \mathbb{1}_{\mathcal{E}})}}} \geq \tau\right) \times 100\% \tag{B.16}$$

where $\mathcal{G}$ and $\mathcal{D}$ denote the ground truth and predicted masks respectively, and $\mathbb{1}_{\mathcal{E}}$ denotes the presence of an event in either of the input event frames. For our evaluation, we choose $\tau = 0.5$. Notice that using both deblur and homography warping helps improve

the results as anticipated. Again, DR_i and DR_o denote testing on trained objects and completely novel objects. As before, deblurring helps generalize much better to novel objects and deblurring with homography warping gives better results showing that the network's learning capacity is utilized better. Also, notice that the improvement in segmentation by warping using homography (last row) is marginal due to the 3D structure of the scene. The network architectures and training details are provided in the Section S.III of the supplementary material.

B.7 Conclusions

We presented an AI-based algorithmic design for micro/nano quadrotors, taking into account the knowledge of the system's computation and latency requirements using deep learning. The central conception of our approach is to contrive AI building blocks using shallow neural networks which can be re-purposed. This philosophy was used to develop a method to dodge dynamic obstacles using only a monocular event camera and on-board sensing. To our knowledge, this is the first deep learning based solution to the problem of dynamic obstacle avoidance using event cameras on a quadrotor. Moreover, our networks are trained in simulation and directly transfer to the real world without fine-tuning or retraining. We also show the generalizability of our navigation stack by extending our work to the pursuit task. As a parting thought, a better similarity scoring metric between event frames or a more robust construction of event frames can improve our results.

Acknowledgement

This work was partly funded by the Brin Family Foundation, National Science Foundation under grant BCS 1824198, ONR under grant N00014-17-1-2622, the Northrop Grumman Mission Systems University Research Program. The authors would like to thank NVIDIA for the grant of an NVIDIA Titan-Xp GPU and Intel for the grant of the Intel Aero Platform.

Appendix C: EVPropNet

Nitin J. Sanket, Chahat Deep Singh, Chethan M. Parameshwara, Cornelia Fermüller, Guido C.H.E. de Croon, Yiannis Aloimonos "EVPropNet: Detecting Drones By Finding Propellers For Mid-Air Landing And Following", *Robotics: Science and Systems (RSS)*, (2021).

EVPropNet: Detecting Drones By Finding Propellers For Mid-Air Landing And Following

Nitin J. Sanket, Chahat Deep Singh, Chethan M. Parameshwara,

Cornelia Fermüller, Guido C.H.E. de Croon, Yiannis Aloimonos

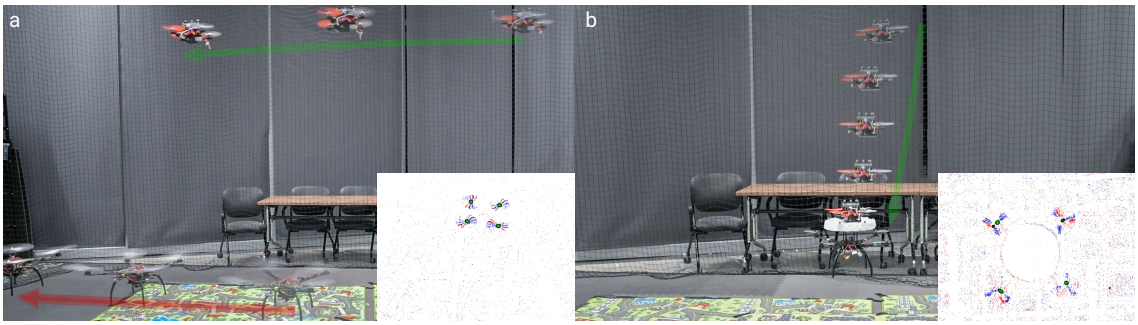


Figure C.1: Applications presented in this work using the proposed propeller detection method for finding multi-rotors. (a) Tracking and following an unmarked quadrotor, (b) Landing/Docking on a flying quadrotor. Red and green arrows indicates the movement of the larger and smaller quadrotors respectively. Time progression is shown as quadrotor opacity. The insets show the event frames $\mathcal{E}$ from the smaller quadrotor used for detecting the propellers of the bigger quadrotor using the proposed *EVPropNet*. Red and blue color in the event frames indicate positive and negative events respectively. Green color indicates the network prediction. All the event images in this paper follow the same color scheme. Vicon estimates are shown in corresponding sub-figures of Fig. C.8. *All the images in this paper are best viewed in color on a computer screen at a zoom of 200%.*

Abstract — The rapid rise of accessibility of unmanned aerial vehicles or drones pose a threat to general security and confidentiality. Most of the commercially available or custom-built drones are multi-rotors and are comprised of multiple propellers. Since these propellers rotate at a high-speed, they are generally the fastest moving parts of an image and cannot be directly "seen" by a classical camera without severe motion blur.

We utilize a class of sensors that are particularly suitable for such scenarios called event cameras, which have a high temporal resolution, low-latency, and high dynamic range.

In this paper, we model the geometry of a propeller and use it to generate simulated events which are used to train a deep neural network called *EVPropNet* to detect propellers from the data of an event camera. *EVPropNet* directly transfers to the real world without any fine-tuning or retraining. We present two applications of our network: (a) tracking and following an unmarked drone and (b) landing on a near-hover drone. We successfully evaluate and demonstrate the proposed approach in many real-world experiments with different propeller shapes and sizes. Our network can detect propellers at a rate of 85.1% even when 60% of the propeller is occluded and can run at upto 35Hz on a 2W power budget. To our knowledge, this is the first deep learning-based solution for detecting propellers (to detect drones). Finally, our applications also show an impressive success rate of 92% and 90% for the tracking and landing tasks respectively.

C.1 Supplementary Material

The accompanying video and code are available at `http://prg.cs.umd.edu/EVPropNet`.

C.2 Introduction

Aerial robots or drones have become ubiquitous in the last decade due to their utility in various fields such as exploration [59, 125, 126, 127], inspection [60], mapping [128], search and rescue [61, 34], and transport [129]. The low-cost and wide availability

of commercial drones for photography, agriculture and hobbying has skyrocketed drone sales [130]. This has also given rise to a series of malicious drones which threaten the general security and confidentiality. This necessitates the detection of drones. To make this problem hard, drones come in various shapes and sizes and generally do not carry any distinct visual on them to make them easy for visual detection. To this end, we propose to detect an unmarked drone by detecting the most ubiquitous part of a drone – the propeller. It is serendipitous that most of the common drones are multi-rotors and have more than one propeller, making their detection using the proposed approach easier. Detecting propellers is a daunting task for classical imaging cameras since it would require an extremely short shutter time and high sensitivity which make such sensors expensive and bulky. A class of sensors designed by drawing inspiration from nature that excel at the task of low-latency and high-temporal resolution data are called event cameras [98, 131]. Recent advances in sensor technologies have increased the spatial resolution of these sensors by about $10\times$ in the last 5 years [132]. These event cameras output per-pixel temporal intensity differences caused by relative motion with microsecond latency instead of traditional images frames. We utilize the fact that propellers are moving much faster than any other part of the scene. The problem formulation and our contributions are described next.

C.2.1 Problem Formulation and Contributions

An event camera (moving or stationary) is looking at a flying drone with at-least one spinning propeller and our goal is to locate the propeller's center on the image plane.

A summary of our contributions are:

- We simplify the geometric model of a propeller for the projection on the image plane which is used to generate event data.
- A deep neural network called *EVPropNet* trained on the simulated data which generalizes to the real-world without any fine-tuning or re-training for different propellers.
- Two specific applications of our *EVPropNet*: (a) Tracking and following an unmarked moving quadrotor (Fig. C.1a), (b) Landing on a near-hover quadrotor (Fig. C.1b) evaluated with on-board computation and sensing.
- Finally, we make our network `EdgeTPU` optimized so that it can run at 35Hz with a power budget of just 2W enabling deployment on a small drone.

C.2.2 Related Work

We subdivide the related work into three parts: detection of an unmarked drone based on appearance (on a classical camera), detection of a marked collaborative drone and detection of moving segments using event cameras.

C.2.2.1 Appearance based drone detection:

Classical RGB image based drone detection is an instance of object detection and has been accomplished by methods like Haar cascade detectors, with the newer deep learning based detectors such as YoLo [133] topping the accuracy charts [134, 135, 136].

One can clearly see that these methods work well when the drone is large in the frame and against a bright sky, thereby detecting the contour of the drone from its silhouette. Pawełczyk *et al.* [134] show extensive results on how the state-of-the-art appearance based drone detectors fail when the drones are against a non-sky background (such as trees which are very common).

C.2.2.2 Marked collaborative drone detection:

Marked drones are detected using a set of fiducial markers either for leader-follower configurations [137], swarming behaviors [138] or for docking [139]. Most commonly, a visual fiducial marker based on April Tags [140] or CC-Tags [141] is used for these tasks due to their robustness and near-invariance to angles. Moreover, they also provide the ability to distinguish between different tags which are particularly useful for tracking multiple drones. Li *et al.* [139] designed a custom tag similar to the CC-Tag and showed that it can be used for precise control for docking. On the contrary, Walter *et al.* [137] demonstrated the usage of Ultra-Violet (UV) spectrum which is robust to changing environmental conditions such as changing illumination and the presence of undesirable light sources and their reflection.

C.2.2.3 Moving Object Segmentation Using Event Cameras:

Event cameras, as described earlier are tailor made for detecting the parts of the image which have motion different to that of the camera (this task is commonly called motion segmentation). Mitrokhin *et al.* [142] developed one of the first motion segmen-

tation frameworks using event cameras for challenging lighting scenes highlighting the efficacy of event cameras to work at high-dynamic range scenes for fast moving objects. Stoffregen *et al.* [104] introduced an Expectation-Maximization scheme for segmenting the motion of the scene into various parts which was further improved in-terms of speed and accuracy by Parameshwara *et al.* [143] by proposing a motion propagation method based on cluster keyslices. The concept of motion segmentation has also been deployed on quadrotors for detection of other moving objects (including other unmarked drones) with a monocular [57] and a stereo event camera [58].

C.2.3 Organization of the paper

We first describe a geometric model of the propeller and then derive a simplified model of it's image projection in Sec. C.3. The geometric model is then used generate event data to train the proposed *EVPropNet* as described in Sec. C.4. We then present two applications of our network: (a) Tracking and following an unmarked drone and (b) Landing on a near-hover drone in Sec. C.5. We then present extensive quantitative evaluation of our network and applications along with qualitative results on different real world propellers in Sec C.6. Finally, we conclude the paper in Sec. C.7 with parting thoughts on future work.

C.3 Geometric Modelling of a Propeller

We first discuss a geometric model of a propeller [144] and then describe how it's projection can be approximated with a set of cubic basis splines. A propeller's spine is

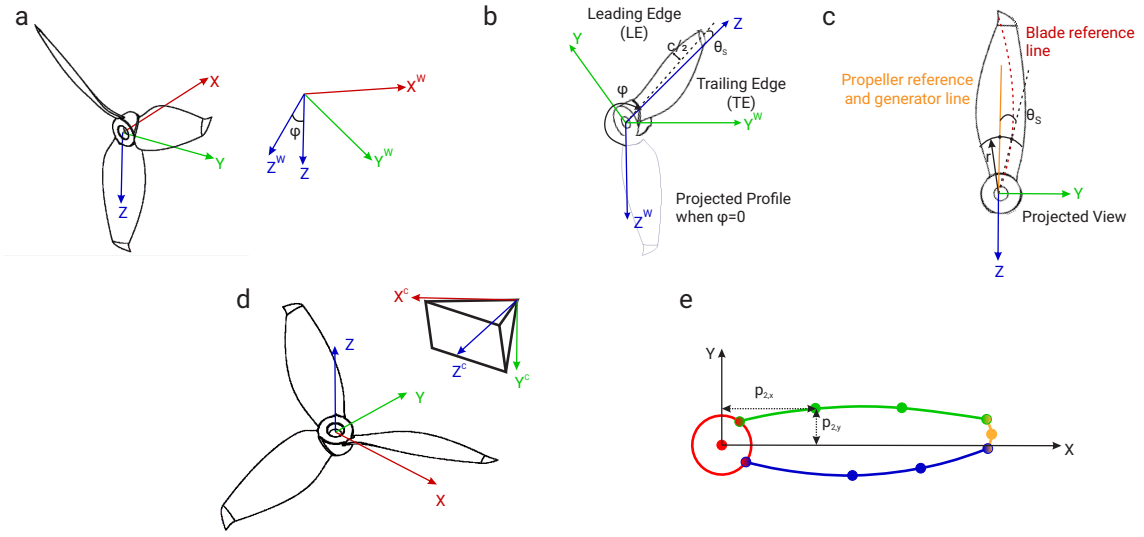


Figure C.2: (a) Coordinate frames used for the geometric modelling of a propeller, (b) Blade coordinate definition, (c) Skew definition, (d) Coordinate axes for propeller projection on camera, and (e) Simplified model of the projection of the propeller blade; Each color represents a single spline and points with same color denote knots used to fit the cubic spline. Bi-color points are used as knots for both the splines of respective color. See Table C.1 for a tabulation of the variables used in this figure.

constructed by rotating a straight line on a helicoidal surface. The coordinates of a point $\mathbf{x}$ on a surface formed by a straight line rotating about the X axis and concurrently moving along this axis is given by

$$\mathbf{x} = \begin{bmatrix} \frac{p\phi}{2\pi} & r\sin\phi & r\cos\phi \end{bmatrix}^T \tag{C.1}$$

Where p is the pitch of the propeller, r is the radius and ϕ is the angle of rotation in YZ plane of the radius arm relative to the Z^W axis (Fig. C.2a, also see Table C.1 for a tabulation of the parameters used in this derivation). Note that, p here refers to the nose-tail pitch as it is the most common definition used by manufacturers. Now, the locus of the mid-chord points of a rotating right handed propeller blade with $\phi = 0$ initially is given by

$$\mathbf{x}_{c/2} = \begin{bmatrix} -\left(i_G + \dfrac{p\theta_S}{2\pi}\right) & -r\sin\left(\phi - \theta_S\right) & r\cos\left(\phi - \theta_S\right) \end{bmatrix}^T \tag{C.2}$$

Here, θ_S (Figs. C.2b and C.2c) denotes the skew angle and is defined as the angle between the line normal to the shaft axis (called *directrix* or *propeller reference line*) and the line drawn through the shaft center line and the mid-chord point on the projected image of the propeller looking normally along the shaft center line and i_G denotes the generator line rake (distance that is parallel to the X-axis, from the directrix to the point where the helix of the section at radius r cuts the $X - Z$ plane). Extending Eq. C.2 for the leading and trailing edges of the blade (Fig. C.2b) gives us

$$\mathbf{x}_{LE/TE} = \begin{bmatrix} -\left(i_G + \dfrac{p\theta_S}{2\pi} + \dfrac{c}{2}\sin\theta\right) \\ -r\sin\left(\phi - \theta_S \pm \dfrac{90c\cos\theta}{\pi r}\right) \\ r\cos\left(\phi - \theta_S \pm \dfrac{90c\cos\theta}{\pi r}\right) \end{bmatrix} \tag{C.3}$$

Here $\theta = \tan^{-1}\left(\frac{p}{2\pi r}\right)$ is the pitch angle, c is the chord length at a certain radius and π and θ_S are in $^\circ$.

The above set of equations define how the propeller's *nose-tail line* can be generated in 3D. However, the blade section geometry is an aerofoil with a top and a bottom surface. A point on the top and bottom surfaces are given by

$$\mathbf{x}_{T/B} = \begin{bmatrix} x_c \mp y_t\sin\psi & y_c \pm y_t\cos\psi \end{bmatrix}^T \tag{C.4}$$

Table C.1: Parameters used in geometric model of the propeller.

Parameter Notation	Parameter Description
p	Propeller Pitch (nose-tail)
r	Radius
ϕ	Angle of rotation of radius arm relative to Z^W in YZ plane
i_G	Generator line rake
θ_S	Skew angle
c	Chord length
ψ	Chamber line slope at x_c
Subscripts T and B	Top and Bottom surfaces of blade
Subscripts LE and TE	Top and Bottom edges of blade
K	Camera intrinsic matrix
f	Camera focal length
c_x and c_y	Camera principal points
$\mathbf{p_i}$	Spline control points
$N_{i,k}(t)$	Spline basis function

where y_c is the y offset from the chord line, y_t is the ordinate of the point in question and ψ is the slope of the chamber line at the non-dimensional chordal position x_c. Now, if we consider the chord's mid point as the local origin, then a point's coordinates $\mathbf{x}$ are given by

$$\mathbf{x} = \begin{bmatrix} -\left(i_G + \dfrac{p\theta_S}{2\pi}\right)(0.5c - x_c)\sin\theta + y_{u,B}\cos\theta \\ r\sin\left(\theta_S - \dfrac{180\left((0.5c - x_c)\cos\theta - y_{u,B}\sin\theta\right)}{\pi r}\right) \\ r\cos\left(\theta_S - \dfrac{180\left((0.5c - x_c)\cos\theta - y_{u,B}\sin\theta\right)}{\pi r}\right) \end{bmatrix} \tag{C.5}$$

Where $y_u = y_c \pm y_t\cos\psi$ (Eq. C.4). To convert x into global coordinates $\mathbf{x}^W$,

$$\mathbf{x}^W = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \mathbf{x} \tag{C.6}$$

where ϕ is the angle between two adjacent blades. Although the propeller thickness varies along its chord line, we can neglect this value since we are concerned with the

projection of the propeller on the image plane assuming that distance from the camera is $\gg$ propeller thickness. We want to simulate how a propeller would "look" when imaged from a camera (which would be later converted into an event stream). We assume that the image is captured from a calibrated camera formulated using the pinhole model given by

$$\mathbf{x} = K \begin{bmatrix} R, & T \end{bmatrix} \mathbf{X}; \quad K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{C.7}$$

where K is the camera calibration matrix, f is the focal length and c_x, c_y denotes the principle point, $\begin{bmatrix} R, & T \end{bmatrix}$ denotes the pose of the camera, $\mathbf{X}$ is the world point being imaged and $\mathbf{x}$ is the location of the point on the image plane (with reuse of variables). $\mathbf{X}$ in Eq. C.7 (see Fig. C.2d for definition of coordinate frames) is given by $\mathbf{x}^W$ from Eq. C.6. Although, this method is the most generative way to model a propeller, i.e., generate 3D points of the propeller and then project onto the image plane, it would be computationally very expensive for a high fidelity image, hence we approximate the projection of a propeller blade with a set of cubic basis splines [145, 146] described next. Let the $n+1$ control points be $\mathbf{p_0}, \cdots, \mathbf{p}_n$ and $m+1$ knot vectors be $\{t_0, \cdots, t_m\}$, the spline curve $s\,(t)$ of degree k is given by

$$s\,(t) = \sum_{i=0}^{n} \mathbf{p}_i N_{i,k}(t) \tag{C.8}$$

Here, $N_{i,k}(t)$ is the basis function of degree k and is computed recursively as

$$N_{i,0}(t) = \begin{cases} 1 & \text{if } t_i \leq t \leq t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{C.9}$$

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(t) \tag{C.10}$$

In particular, $m = n + k + 1$ and we utilize the uniform B-spline, i.e., all the knots are uniformly distributed and are evaluated using the procedure described in [147]. We model each propeller blade using 4 cubic B-splines: one for the hub, one for the top part of the blade, one for the bottom part of the blade and one for the tip of the blade (Fig. C.2e). Each blade is replicated at a uniform angular spacing for the required number of blades (i.e., for a 3 bladed propeller, the blades would have an angle of 120° between them).

C.4 EVPropNet

We will now discuss how the geometric model of the propeller is used to generate event data. Then we describe the network architecture and loss function used to train *EVPropNet*.

C.4.1 Event Generation

As explained earlier, we now have a single image of a propeller with the required number of blades at the required high resolution. We overlay this propeller image on

top of a random real image background from the MS-COCO dataset [148] at a random starting angle θ_{HB} (angle the propeller reference line makes with the propeller Y axis), we denote this as image $\mathcal{I}_t$. We then perform the same procedure for a $\theta_{HB} + \delta\theta$ angle (with the same background) to generate the image $\mathcal{I}_{t+\delta t}$. Here, $\delta\theta = \omega\delta t$ is the angle the blade would rotate depending on the rotational speed of the propeller ω and the event frame integration time δt. We use a simple model for the event camera and events are triggered at a location $\mathbf{x}$ when

$$\| \log\left(\mathcal{I}_t\left(\mathbf{x}\right)\right) - \log\left(\mathcal{I}_{t+\delta t}\left(\mathbf{x}\right)\right) \|_1 \geq \tau \tag{C.11}$$

The event stream/cloud $\mathcal{E}$ is represented by

$$\mathcal{E} = \{ \begin{bmatrix} \mathbf{x} & t & \left(\log\left(\mathcal{I}_t\left(\mathbf{x}\right)\right) - \log\left(\mathcal{I}_{t+\delta t}\left(\mathbf{x}\right)\right)\right) \end{bmatrix}^T \} \tag{C.12}$$

Where τ is a user defined threshold and $\mathbf{x}$ is the pixel location. $\mathcal{E}$ is called the event cloud which is used to create the so-called *Event-frame* $\mathcal{E}$ (Fig. C.3 shows how $\mathcal{E}$ and $\mathcal{E}$ look) which is used as the input to the network.

$$\mathcal{E} = \left(\mathbb{E}_t\left(\text{Pol}\left(\mathcal{E}\left(t, t+\delta t\right)\right)\right)\right) \tag{C.13}$$

Here, $\mathbb{E}_t$ denotes the averaging operator only in time axis, Pol denotes the polarity values are extracted per pixel (last row of each element of $\mathcal{E}$).

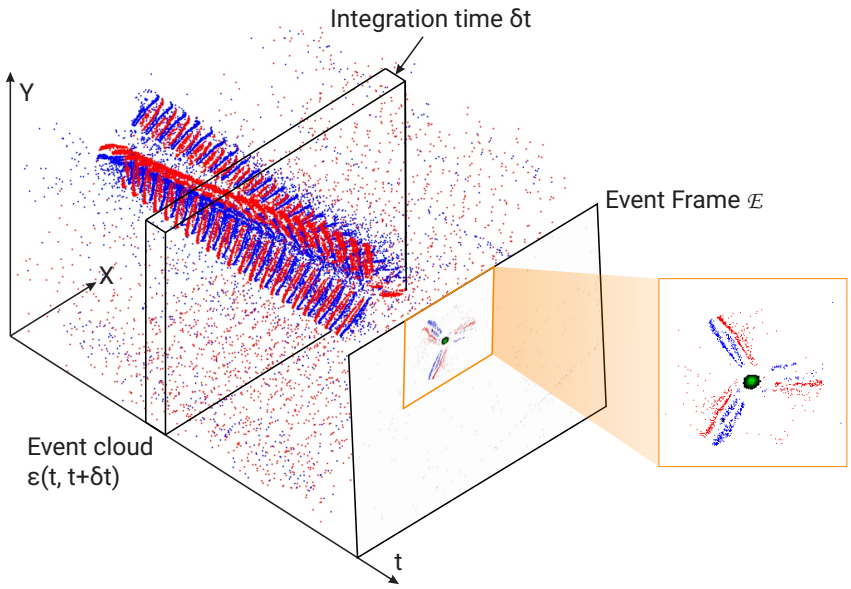


Figure C.3: Spatio-temporal event cloud $\mathcal{E}$ and Event frame $\mathcal{E}$. The cloud shows that the propeller creates a helix in the spatio-temporal domain. The zoomed in view shows the propeller with positive events colored red and negative events colored blue along with network prediction as green with the color saturation indicating confidence.

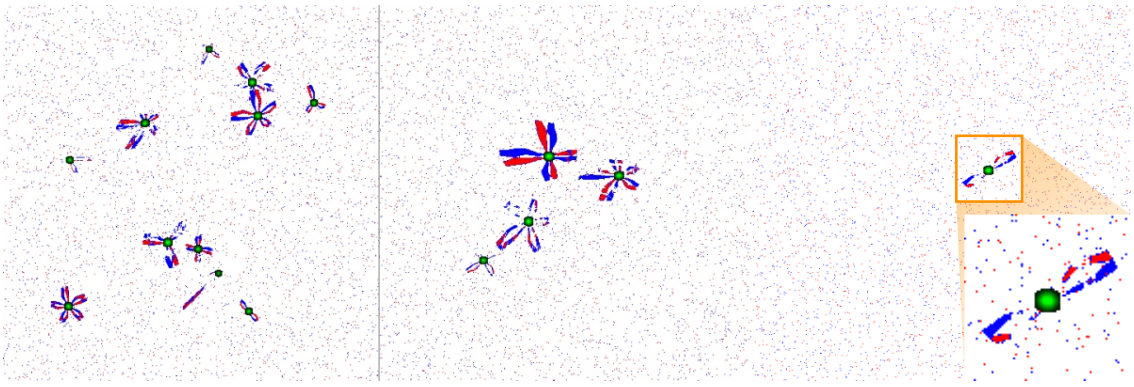


Figure C.4: Sample event images $\mathcal{E}$ from the generated synthetic dataset used to train *EVPropNet*. Here red and blue colors show positive and negative events respectively. Green color indicates our ground truth label with the color saturation indicating confidence as defined by Eq. C.14.

C.4.2 Data Generation

We generate 10K event frames $\mathcal{E}$ for training our network (See Fig. C.4 for sample images with labels overlaid). Each event frame contains upto N propellers (set to 12 in

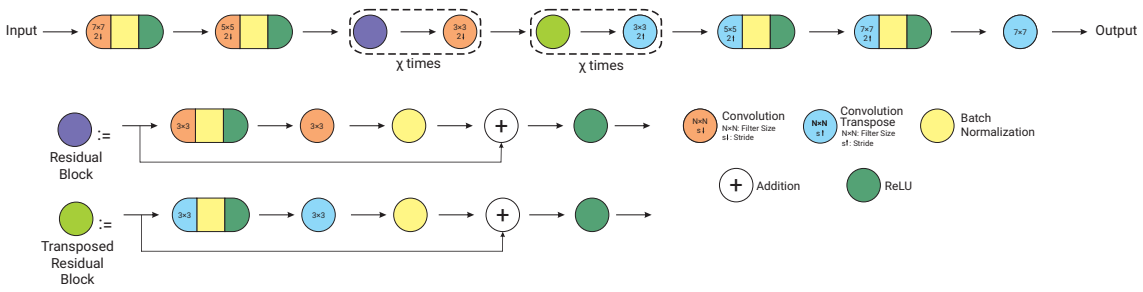


Figure C.5: Network architecture for *EVPropNet* (χ is a hyperparameter along with expansion rate – rate at which the number of neurons grow after each block). If no down/up-sampling rate is shown, it is taken to be 1. *This image is best viewed on the computer screen at a zoom of 200%.*

our case) with number of blades per propeller randomized from 2 to 6. The events for each propeller are obtained by varying τ as a gaussian random variable to provide some randomness in data generation along with randomization of the color of the propeller in $\mathcal{I}_t$ (same color is used for $\mathcal{I}_{t+\delta t}$) along with varying ω (rotational speed of the propeller, this is equivalent to varying the integration time δt). We also vary the background image for every propeller from the MS-COCO dataset. Each propeller is also warped using a random homography matrix to account for different camera angles along with scaling them (setting the pixel size of the propeller in the event image) to account for distance variation from the camera. Finally, we also vary the shape of each propeller by varying the basis spline parameters (to include bullnose and normal type propellers as well). See Fig. C.4 for some sample images from the dataset used to train *EVPropNet*. Note that, we do not use an event simulator like ESIM[149] to generate events since we only require $\mathcal{I}_t$ and $\mathcal{I}_{t+\delta t}$ which are directly constructed, hence this process is multiple orders of magnitude faster than real-time and parallelized.

C.4.3 Network Architecture and Loss Function

We choose an encoder-decoder architecture based on the ResNet [150] backbone (Fig. C.5) as it has the best accuracy and speed tradeoff [151, 152] with 2.7M parameters and 10MB model size. We train our network using simple mean square loss $\mathcal{L} = \mathbb{E}\left((\hat{p} - \tilde{p})^2\right)$ between the ground truth $\hat{p}$ and prediction $\tilde{p}$. $\hat{p}$ is obtained by Gaussian smoothing the perfect label $\hat{p_0}$ (binary mask) as given by Eq. C.14 (σ is the variance) to account for small distortion introduced by approximation of propeller shape. This approach is similar to the one introduced in [153].

$$\hat{p} = \frac{1}{2\pi\sigma^2} e^{-\left(\|\hat{p_0}\|_2 / 2\sigma^2\right)} \tag{C.14}$$

We choose the number of residual and transposed residual blocks χ as 2 and expansion factor as 2 (factor with which number of neurons grow after every block in Fig. C.5).

Finally, *EVPropNet* was trained with a learning rate of 1e-4 using ADAM optimizer with a batch size of 32 for 50 epochs.

C.5 Applications

We describe two applications of our propeller detection, i.e., following an unmarked moving quadrotor and landing on a near-hover quadrotor.

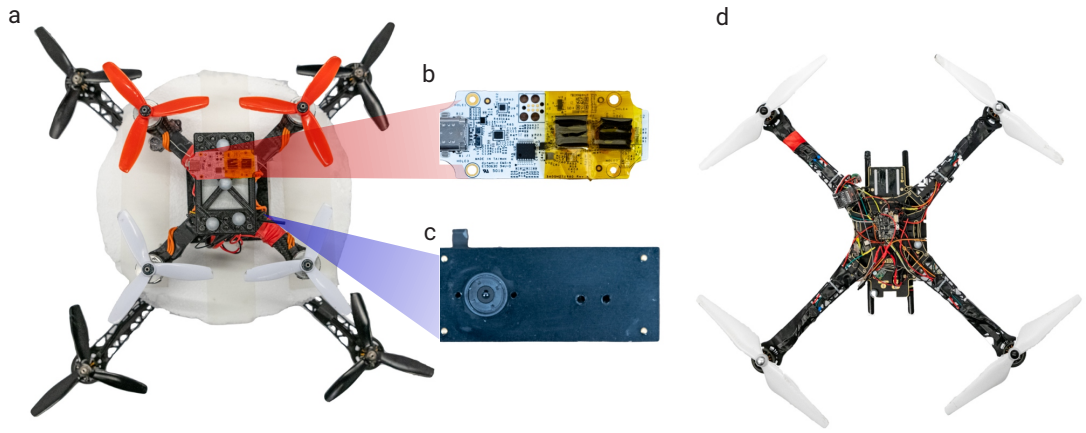


Figure C.6: (a) Smaller quadrotor on the bigger quadrotor used for landing experiments (Sec. C.5.1), (b) Gutted Coral USB Accelerator with custom heat sink used to run the neural networks, (c) Samsung Gen 3 DVS sensor used for experiments, (d) Bigger quadrotor used in the following experiments (Sec. C.5.2).

C.5.1 Following

In this application, the goal is to track and follow a quadrotor (or a multirotor in general) either for swarming or reconnaissance purposes. We detect the quadrotor as the centroid of filtered propeller detections as described in Sec. C.5.3. The control policy for altitude is to maintain the area of the quadrilateral joining the propeller points constant and the control policy for roll and pitch is to maintain the centroid of the quadrilateral in the center of the image and are given by:

$$\mathbf{u}_{\phi}(t) = K_{p,\phi} e_x(t) + K_{i,\phi} \int_0^{\tau} e_x(\tau) d\tau + K_{d,\phi} \frac{de_x(t)}{dt} \tag{C.15}$$

$$\mathbf{u}_{\theta}(t) = K_{p,\theta} e_y(t) + K_{i,\theta} \int_0^{\tau} e_y(\tau) d\tau + K_{d,\theta} \frac{de_y(t)}{dt} \tag{C.16}$$

$$\mathbf{u}_{T}(t) = K_{p,T} e_A(t) + K_{i,T} \int_0^{\tau} e_A(\tau) d\tau + K_{d,T} \frac{de_A(t)}{dt} \tag{C.17}$$

Where e_x and e_y denote the difference between detected quadrilateral center on the

image plane and the center of the image and e_A denotes the difference between area to maintain and current area.

C.5.2 Landing

For the second application, the goal is to land on a near-hover quadrotor either for in-air battery switching or infiltration of a hostile drone. We utilize the following key observations from [154]:

- The quadrotor flying above experiences negligible aerodynamic disturbances from mutual interaction.
- Forces in direction normal to the downwash are negligible and those in the downwash direction are significant.
- The aerodynamic torques disturb the bottom quadrotor so that it aligns vertically with the top quadrotor.

The smaller quadrotor (which will land) explores the area for any other quadrotor (or any multirotor in general), once it detects a multirotor, it switches to the align maneuver, where we perform the following control policy for roll ϕ and pitch θ axes (X and Y respectively) for aligning the center of the detected quadrotor (centroid of filtered propeller detections as described in Sec. C.5.3) with the center of the image:

$$\mathbf{u}_\phi(t) = K_{p,\phi} e_x(t) + K_{i,\phi} \int_0^\tau e_x(\tau) d\tau + K_{d,\phi} \frac{de_x(t)}{dt} \tag{C.18}$$

$$\mathbf{u}_\theta(t) = K_{p,\theta} e_y(t) + K_{i,\theta} \int_0^\tau e_y(\tau) d\tau + K_{d,\theta} \frac{de_y(t)}{dt} \tag{C.19}$$

Where e_x and e_y denote the difference distance between detected quadrotor center on the image plane and the center of the image. Once the errors e_x and e_y are lower than a threshold, we decrease altitude at a constant rate, checking for x and y alignment at every control loop and re-aligning as necessary. Once we are close to the big quadrotor (on which the smaller quadrotor will land), we initiate the land command.

C.5.3 Quadrotor Location from Detected Propellers and Filtering

We filter each propeller location on the image plane using a linear Kalman filter [155]. The motion model is a constant optical flow model. Once we obtain detections with a confidence above a certain threshold, the filtered propeller locations are used to compute the centroid of the quadrilateral (for the quadrotor case, polygon in general) which is used for control (to compute e_x and e_y), along with the area for altitude control.

C.6 Experimental Results and Discussion

C.6.1 Quadrotor Setup

All our experiments are performed with quadrotors for their minimal hardware complexity and cost-effectiveness but they can directly be adapted to any multirotor vehicle. Our smaller quadrotor is a custom built platform on a QAV-X 210mm sized (motor center to motor center diagonal distance) racing frame. The motors used are T-Motor F40III KV2400 mated to 5040×3 propellers (Fig. C.6a). The lower level controller and position hold is handled by ArduCopter 4.0.6 firmware running on the Holybro Kakute F7 flight controller mated to an optical flow sensor and TFMini LIDAR as altimeter source. All

the higher level navigational commands are sent by the companion computer (NVIDIA Jetson TX2 running Linux for Tegra®) using RC-Override to the flight controller running in Loiter mode using MAVROS. The event camera used is a Samsung Gen-3 Dynamic Vision Sensor [132] with a resolution of 640×480 px. (Fig. C.6c). and is mounted facing forward tilted down by 45° for the following experiment and facing down for the landing experiment. *All the computations and sensing are done on-board with no use of an external motion capture system.* Our neural network runs on a gutted Google Coral USB Accelerator with a custom heatsink attached to the TX2. The quadrotor take-off weight including the battery is 680 g and has a thrust to weight ratio of 5:1. Our network runs at 35Hz on the Coral accelerator (See Fig. C.6b. Implementation details are given in Sec. C.6.4) and our planning and control algorithms run at 15 Hz on the TX2.

The larger quadrotor used in the following experiment is built on a S500 frame with DJI F2312 960KV motors mated to white colored 9450×2 propellers (Fig. C.6d). Same avionics components are used as the smaller quadrotor.

The larger quadrotor used in the landing experiment is built on a S500 frame with T-Motor F80 Pro KV2500 motors mated to black colored 6040×3 propellers (Fig. C.6a). Same avionics components are used as the smaller quadrotor and ArduCopter firmware holds the position in Loiter mode during experiments with all the sensor fusion, control and planning handled by the flight controller. *The area where the smaller quadrotor can land is of radius 135mm, which gives a tolerance of just 30mm on each side.*

C.6.2 Experimental Results And Observations

C.6.2.1 Quantitative Evaluation of *EVPropNet*

In the first case study we discuss quantitative evaluation results of our propeller detection results for varying resolution of propeller blade r_{px} (the bounding box size of the propeller would be $2r_{\text{px}}$ and is directly correlated with real-world propeller size r), number of blades N_{blades}, noise probability p_n, data miss probability p_b, different camera roll and pitch angles (ϕ and θ respectively). Formally, p_n denotes the probability with which a pixel can have error (equally likely to be either a positive or negative event) and p_b denotes the probability with which the pixel where the propeller data exists did not fire either due to a dead-pixel or camouflage with the background. We use the following metric to denote a successful detection of a propeller.

$$\text{Success} := {}^{\mathcal{G}\cap\mathcal{D}}/_{\mathcal{G}\cup\mathcal{D}} \geq 0.5; \mathcal{G} : \text{Ground Truth}, \mathcal{D} : \text{Detection} \tag{C.20}$$

Detection Rate DR is given by $\text{DR} = \mathbb{E}\left(\text{Success}\right)$, where $\mathbb{E}$ is the expectation/averaging operator. The results are presented in Table C.2. When not specified, the values for the parameters are given as follows: $r_{\text{px}} = \{20, 30, 40, 50, 60\}$, $N_{\text{px}} = \{2, 3, 4, 5, 6\}$, RPM $= \{$5K,10K,20K,30K,40K$\}$, $p_n = \{0, 0.01, 0.02\}$, $p_b = \{0, 0.15, 0.3, 0.45, 0.6\}$, $\phi = \{0°, 10°, 20°, 30°, 60°\}$ and $\theta = \{0°, 10°, 20°, 30°, 60°\}$.

We see from Table C.2a that DR increases with propeller size and then decreases, this is because as the amount of data increases, the results improve and but when the propeller is large ($r_{\text{px}} = 60$), we observe an increase in false detections near the edges of

Table C.2: Detection Rate (%) ↑ of *EVPropNet* for variation in parameters.

(a) r_{px} (px.) for $\phi = \theta = 0°$

20	30	40	50	60
78.9	90.4	94.4	97.6	93.9

(b) N_{blades} for $\phi = \theta = 0°$

2	3	4	5	6
77.9	94.1	96.3	92.8	94.1

(c) RPM (min^{-1}) for $\phi = \theta = 0°$

5K	10K	20K	30K	40K
71.5	91.7	97.1	98.1	96.8

(d) p_n for $\phi = \theta = 0°$

0	0.01	0.02
92.6	91.4	89.1

(e) p_b for $\phi = \theta = 0°$

0	0.15	0.3	0.45	0.6
97.3	94.1	95.5	88.8	79.5

(f) ϕ (°) for $p_n = p_b = 0$

0	10	20	30	60
97.6	94.7	94.1	94.1	89.1

(g) θ (°) for $p_n = p_b = 0$

0	10	20	30	60
97.6	96.5	93.4	93.6	86.7

the propeller blades, dropping the DR slightly (Table C.2a). A similar trend is observed with N_{blades} and RPM with DR peaking for a 4 bladed propeller and at 10K RPM (Tables C.2b and C.2c). We also observe that with increase in p_n (Table C.2d), the detection results are not affected significantly highlighting the robustness of our network. Even when 60% of the propeller is camouflaged with the busy background, we obtain a DR of above 79% (Table C.2e) with the DR decreasing with increase in camouflage amount as expected. From Tables C.2f and C.2g, we also observe that even with camera angles (ϕ and θ) = $60°$, we obtain a DR of above 85%. Finally, we obtain an overall DR of 85.1% for variations in all parameters and 90.9% when no data is corrupted.

Also, if we define success for drone detection as detecting atleast η propellers of the drone, we obtain the drone detection rate DR_D as follows $\text{DR}_\text{D} = (1 - (1 - \text{DR})^\eta)$, where DR is the detection rate of a single propeller. For example, we would obtain a drone detection rate DR_D of 97.7%, 99.6% and 99.9% for a quadrotor, hexacopter and

Table C.3: Detection Rate (%) ↑ of AprilTags 3 for amount of tag blocked.

p_b				
0	0.15	0.3	0.45	0.6
100	91.5	73.3	40.5	4.0

octocopter respectively even when only 50% of the propellers are detected.

C.6.2.2 Quantitative Evaluation of April Tags 3

In the second case study, we evaluate how a custom designed passive fiducial marker would perform the task of detecting a drone (note that this is only applicable to a collaborative drone). In particular, we evaluate one of the most ubiquitous and robust passive fiducial markers April Tag 3 [140] (36h11 family) inspired from [156]. The parameters are the same as the first case study. From Table C.3, we observe that when the data is not missing (occluded or not correctly exposed), the April Tag detects the tags with an impressive DR (tag ID correctness is not considered) of 100%, but the accuracy falls significantly to 61.9% when data is missing (which is common in real-world due to high dynamic range scenarios and motion blur). It is also important to note the following reasons when a drone detection based on event camera based propeller detection will be better than a passive fiducial marker based detection.

- Detection of a non-collaborative drone for reconnaissance purposes
- High dynamic range and adverse lighting scenarios including fast movement
- Area occupied by non-occluded propellers is generally $\gg$ area occupied by the fiducial marker in the center (Refer to Sec. C.6.3 for a detailed analysis)
- When a major part of the fiducial marker would be generally occluded

C.6.2.3 Quantitative Evaluation of Appearance based drone detectors

In the third case study, we compare drone detection using classical appearance based detectors from [134]. We see that the Haar Cascade detectors have a DR of 55.2% and the MobileNet deep learning based detector has a detection rate of 69.4% which are far lower than those of the fiducial detector and our propeller detection method.

C.6.2.4 Performance Measure on different compute platforms

In the fourth case study, we present speed and timing results for *EVPropNet* on various commonly used computational platforms. We refer the readers to [152] for a detailed description of the compute modules used in this case study. *EVPropNet* has 2.7M parameters, a model size of ∼10MB and utilizes 17GOPs for a single forward pass. We can see from Table C.4 that running *EVPropNet* to detect a drone by detecting propellers on the Google Coral Accelerator attached to the NVIDIA Jetson TX2 has the best speed and detection performance per unit power.

Table C.4: Performance Metrics On Different Compute Modules.

Method	(Ours) *EVPropNet* (Ours)					AprilTags 3 (36h11)	AprilTags 3 (16h5)
Computing Platform	PC (i9)	PC (TitanXp)	TX2†	NCS2*	Coral*	TX2	TX2
Speed ↑ (Frames per second)	8.6	133.4	10.5	4.5	35.2	7.0	41.3
Weight (g) ↓	–	–	130	138	136	130	130
Peak Power (W) ↓	250	250	15	17	17	15	15
Speed/Unit ↑ Power (FPS/W)	0.03	0.53	0.7	0.27	2.07	0.47	2.75
Detection Rate (%) ↑	85.1	85.1	85.1	83.4	81.9	61.9	53.4
Speed×DR/Unit ↑ Power (FPS%/W)	2.55	45.10	59.57	22.52	**169.53**	29.09	146.85

† Active heatsink removed. * Attached to TX2, outer casing removed and custom heatsink.

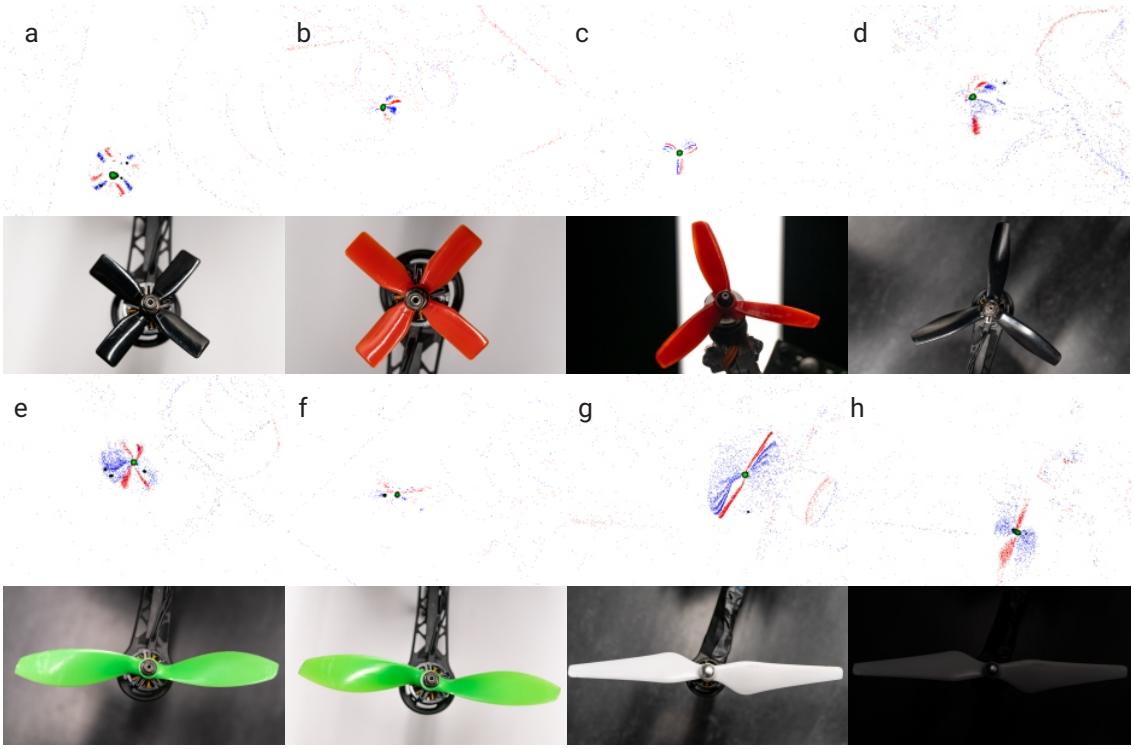


Figure C.7: Top rows: Input event frame $\mathcal{E}$ where red and blue colors show positive and negative events respectively. Green color indicates *EVPropNet* prediction with the color saturation indicating confidence. Bottom rows: reference images of the propeller taken with a Nikon D850 DSLR (32dB dynamic range). Scenarios (a) to (h) are explained in Table C.5.

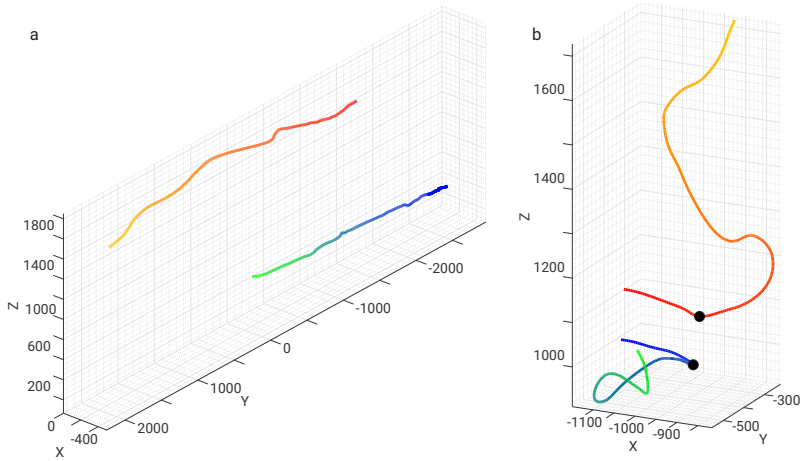


Figure C.8: Vicon estimates for the trajectories of the smaller and larger quadrotor in the application experiments shown in Fig. C.1. (a) Tracking and following, (b) Mid-air landing. Time progression is shown from yellow to red for the smaller quadrotor and and green to blue for the bigger quadrotor. The black dots in (b) show the moment in time where the touchdown occured.

Table C.5: Different Propeller Configurations Used for Qualitative Evaluation in Fig. C.7.

Scenario	Ref. Fig.	Prop. Color	Background Color	Prop. Radius (mm)	Background Light Intensity (lx)	Propeller Area / Motor Area
(a)	C.7a	Black	White	50.8	240	2.3
(b)	C.7b	Red	White	50.8	240	2.3
(c)*	C.7c	Red	White and Black	63.5	564 and 2	3.6
(d)	C.7d	Black	Black	76.2	240	5.2
(e)	C.7e	Green	Black	88.9	240	7.1
(f)	C.7f	Green	White	88.9	240	7.1
(g)	C.7g	White	Black	119.4	24	12.8

*Case (c)'s light intensity shows High Dynamic Range scenario with illumination of the light part being 564lx and dark part being 2lx (See Fig. C.7c).

C.6.2.5 Qualitative Evaluation on different real-world propellers

In the final case study, we present qualitative results of *EVPropNet* on different lighting scenarios, propeller sizes, propeller and background colors, N_{blades}, r and angles. Fig. C.7 shows the qualitative results where the description of the scene is given in Table C.5. Notice how *EVPropNet* can handle different real-world variations along with high dynamic range (Fig. C.7c, even a high-end DSLR cannot capture both shadows and highlights with it's 32dB of dynamic range but the event camera with it's 80dB can handle such a scene with ease), low contrast (Fig. C.7d) and low light with average intensity of 24lx (Fig. C.7g).

C.6.2.6 Quantitative Evaluation of applications

We now present the results for both our applications and we call them experiment 1 for tracking and following and experiment 2 for landing. We define success as being able detect the quadrotor and to not completely loosing track for experiment 1, and for the quadrotor to be able to detect the other quadrotor and land on it successfully without

collision for experiment 2. We average our results over 50 trails for each experiment and obtain a success rate of 92% for experiment 1 and 90% for experiment 2 (Vicon estimates for the trial shown in Fig. C.1 are shown in corresponding sub-figures of Fig. C.8). Commonly, the failure cases in experiment 1 happen when the larger quadrotor has a huge jerk that it moves outside the field of view of the camera. The failure cases in experiment 2 happen due to the aerodynamic interference between the two quadrotors which makes the bottom quadrotor drift at the last moment.

C.6.3 Analysis

We present analysis of three questions: 1. What is the ratio of visible area of the propellers to that of the largest square fiducial marker in the center? (Fig. C.9) 2. How does DR vary with focal length f, real-world propeller radius r and camera angle ϕ (angle around X axis)? (Fig. C.2d). 3. What makes *EVPropNet* generalize to the real-world without any fine-tuning or re-training?

To analyse the answer to the first question, we present a simplified geometric model of a multirotor (quadrotor shown in Fig. C.9a) where we are given a constraint on the drone's size S (diagonal motor to motor length) and number of propellers on the drone N_{prop}. Let us say that the largest fiducial marker that can fit in the center of the drone is inscribed in the circle of radius r_c. Also, we assume that the propeller does not generate events in the area in which is occluded by the arm and the motor. The motor radius is given by r_m and the arm width is given by $2r_m$ (Fig. C.9b). The area of one non-occluded propeller $\mathcal{A}_{\text{prop}}$ (gray highlighted area in Fig. C.9b) is given by

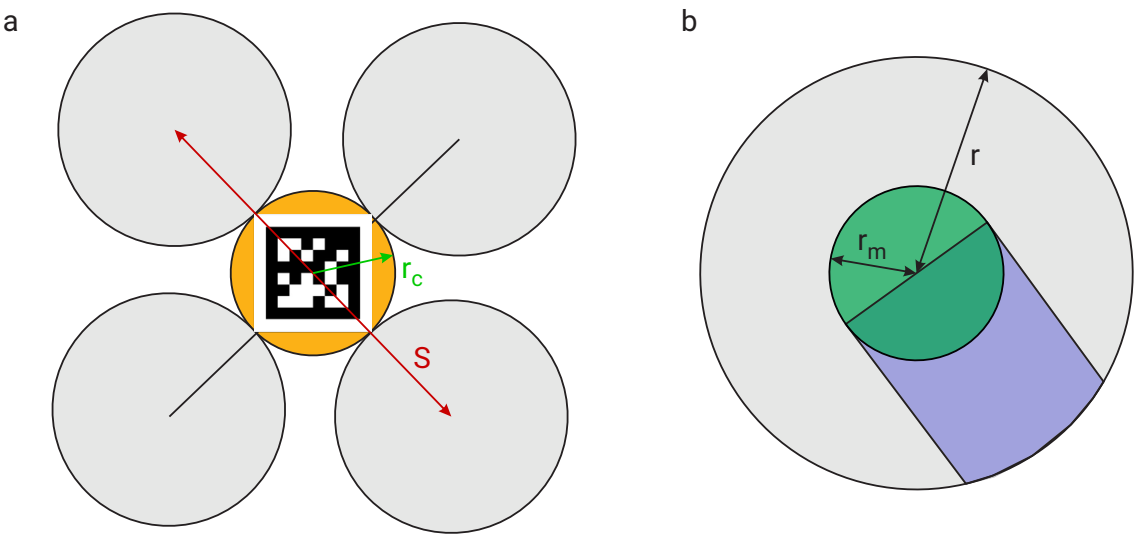


Figure C.9: (a) Simplified model of a quadrotor used to calculate area ratios of the propellers to that of the biggest square fiducial marker that can be fit in the center without obstruction, (b) Simplified arm and motor projection to compute amount of propeller occluded from generating events – gray areas show where the propeller is visible and generates events, green area is occluded by the motor and blue area is occluded by the arm.

Table C.6: $\mathcal{A}_{\text{ratio}}$ FOR SOME COMMON COMMERCIAL DRONES.

Name	S (mm)	N_{prop}	r (mm)	r_m (mm)	$\mathcal{A}_{\text{ratio}}$
DJI Phantom 4	350	4	119.4	12	109.8
QAV 210 X	210	4	63.5	14.0	51.2
DJI Inspire 2	603	4	190	18.5	69.2

$$\mathcal{A}_{\text{prop}} = r^2\left(\pi - \frac{\gamma}{2}\right) - \frac{\pi r_m^2}{2} - r_m r\cos\left(\frac{\gamma}{2}\right); \gamma = 2\sin^{-1}\left(\frac{r_m}{r}\right) \tag{C.21}$$

Hence, the ratio for the area of the largest visible fiducial marker to that of a N_{prop} propeller drone will be

$$\mathcal{A}_{\text{ratio}} = \frac{4N_{\text{prop}}\left(r^2\left(2\pi - \gamma\right) - \pi r_m^2 - 2r_m r\cos\left(\frac{\gamma}{2}\right)\right)}{\left(S - 2r\right)^2} \tag{C.22}$$

The value of $\mathcal{A}_{\text{ratio}}$ for some common commercially available drones are given in Table C.6 (Recall, N_{prop} is the number of propellers on the drone, r is the propeller radius, r_m is the motor radius, γ is defined in Eq. C.21 and S is the drone's diagonal motor to motor length). We clearly see that the probability of observing at-least one propeller (directly related to $\mathcal{A}_{\text{ratio}}$) is much higher than that of observing a fiducial marker in the middle, thereby reinforcing the motivation of our approach.

For the analysis of the second question, refer to Fig. C.10. We see that the DR of the propeller increases with an increase in real world propeller radius r until it reaches a maximum and then decreases (Fig. C.10a). This trend is observed since smaller propellers (small r) generate a small number of events leading to a low DR and increases with increase in number of events (directly correlated with r). However, with larger propellers the DR decreases as the number of false detections increase near the tip of the propeller. With a larger focal length (larger f), the curvature of the curve is larger since the relative projection area change (on the image plane) is more drastic. We can also observe a similar trend in Fig. C.10b with change in angle ϕ (angle around X^C axis in Fig. C.2d). Notice that the change in focal length affects the accuracy more significantly than the change in angle. Note that pitch θ has the similar effect to that of the roll ϕ.

Finally, we speculate why our *EVPropNet* generalizes to the real-world without any fine-tuning or re-training for different propellers.

- The data's visual quality from simulation is similar to those obtained from recently developed event cameras both in-terms of noise and data-rate. (This is not simple with data from classical cameras due to the lack of photo-realism.)

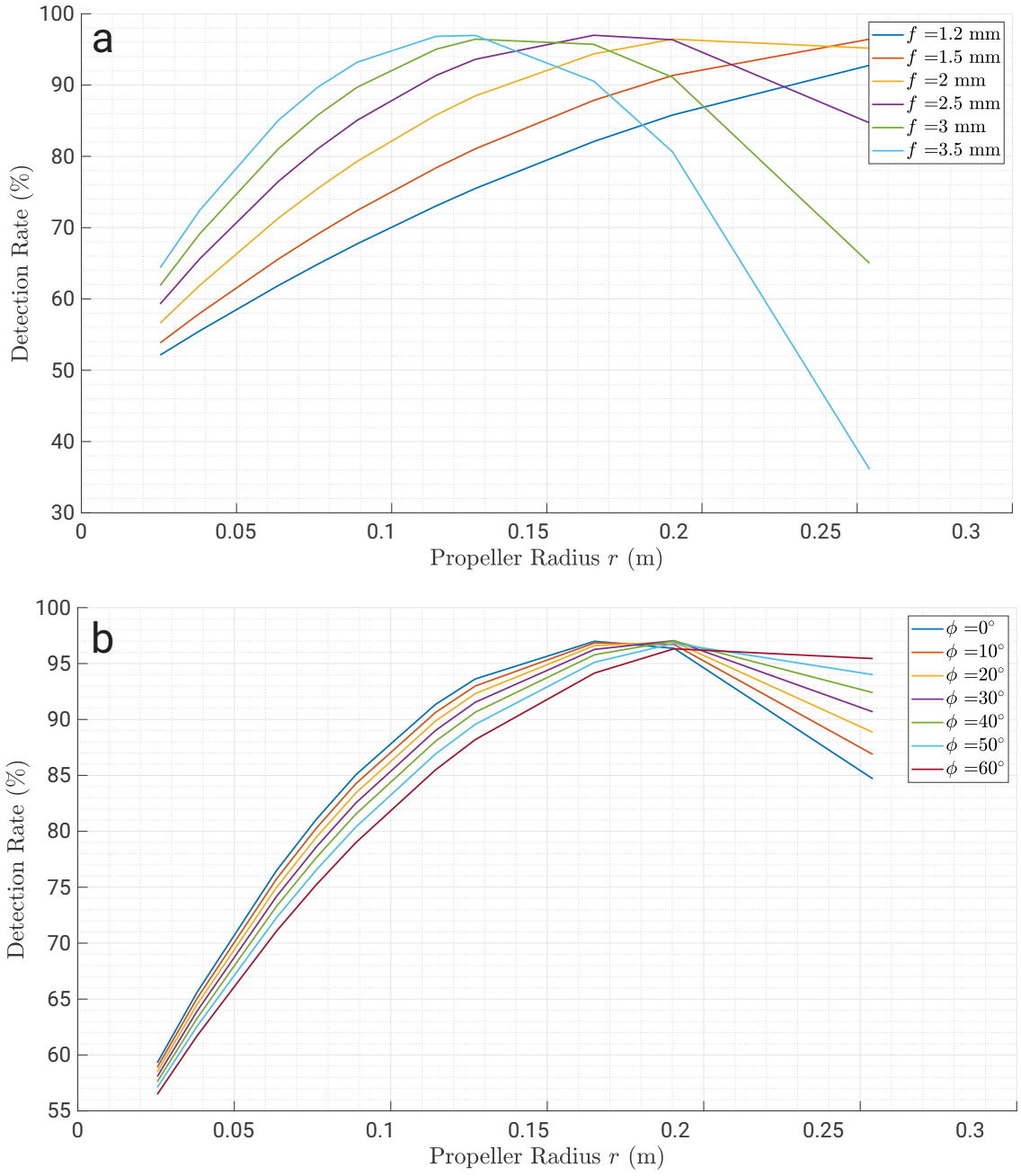


Figure C.10: Variation of Detection Rate with variation in real-world propeller radius r for different (a) Focal lengths f with $\phi = 0°$, and (b) Camera Roll ϕ with $f = 2.5$mm.

- The errors in simulation (as compared to the real-world) are lower when the integration time for creation of event frames are smaller (around 20ms maximum) as demonstrated by [57].

C.6.4 Implementation Considerations

To speed-up the computation of our network when deployed on an aerial robot, we quantize our network to `Int8` and compile our network using `EdgeTPU` optimizations for deployment on the Google Coral USB Accelerator. To enable smooth compilation and high accuracy retention, we make our inputs take only valid `Int8` values as given below

$$\mathcal{E}_{\texttt{EdgeTPU}} = (\mathcal{E} \times 255 + 127 | 0, 255) \tag{C.23}$$

$$(x|a, b) := \max(b, \min(x, a)) \tag{C.24}$$

The labels $\hat{p}$ are modified as $\hat{p}_{\texttt{EdgeTPU}} = \lfloor \hat{p} \times 255 - 0.5 \rfloor$ and take integer values in $[0, 255]$.

Finally, when using an event camera with a high resolution at a high temporal sampling rate, the bottleneck of the system is the transfer speed between the event sensor and the compute module which are dictated by the combined throughput of processor, cache, transfer speeds of the primary and secondary memory. Such a bottleneck can cause data loss and data lag in the buffer. We mitigate this issue by using the NVIDIA TX2 which has a throughput of $\sim$440MBs^{-1}.

C.7 Conclusions

We presented a method to detect unmarked drones (multi-rotors) by detecting a ubiquitous part of their design – the propellers. To enable detection of the propellers, we utilize the following fact: propellers rotate at high-speed and hence are generally the

fastest moving parts on an image. We model the geometry of the propeller and use it to simulate the data from an event camera whose qualities of high temporal resolution, low latency and high dynamic range make it perfectly suited for detecting propellers. We then train our *EVPropNet* deep network on this simulated data which generalizes directly to the real-world without any fine-tuning or re-training. We present two applications of detecting propellers on an unmarked drone: (a) tracking and following an unmarked drone and (b) landing on a near-hover drone. As a parting thought, an active zoom camera would increase the distance range from where the drones could be detected and would make our method a viable for deployment in the wild.

Acknowledgement

The support of the National Science Foundation under grants BCS 1824198 and OISE 2020624, the support of the Office of Naval Research under grant award N00014-17-1-2622, the Northrop Grumman Corporation and the Brin Family foundation are gratefully acknowledged. We also would like to thank Samsung for providing us with the event-based vision sensor used in this research.

Appendix D: MorphEyes

Nitin J. Sanket, Chahat Deep Singh, Varun Asthana, Cornelia Fermüller, Yiannis Aloimonos "MorphEyes: Variable Baseline Stereo For Quadrotor Navigation", *IEEE International Conference on Robotics and Automation (ICRA)*, (2021).

MorphEyes: Variable Baseline Stereo For Quadrotor Navigation

Nitin J. Sanket, Chahat Deep Singh, Varun Asthana,

Cornelia Fermüller, Yiannis Aloimonos

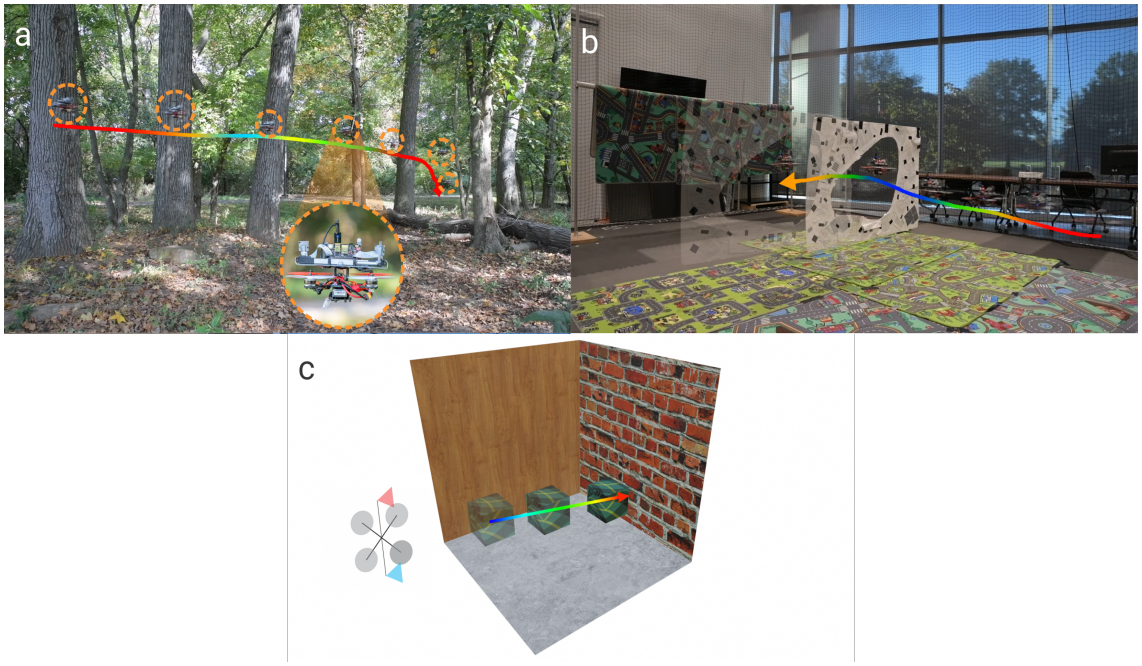


Figure D.1: Three applications of a variable baseline stereo system were explored in this work. (a) Flying through a forest, (b) Flying through an unknown shape and location dynamic gap, (c) Detecting an Independently Moving Object. In all the cases, the baseline of the stereo system is changing and is colored coded as jet (blue to red indicates 100 mm to 300 mm baseline). The opacity of the quadrotor/object shows positive progression of time. *All the images in this paper are best viewed in color on a computer.*

Abstract — Morphable design and depth-based visual control are two upcoming trends leading to advancements in the field of quadrotor autonomy. Stereo-cameras have struck the perfect balance of weight and accuracy of depth estimation but suffer from the

problem of depth range being limited and dictated by the baseline chosen at design time. In this paper, we present a framework for quadrotor navigation based on a stereo camera system whose baseline can be adapted on-the-fly. We present a method to calibrate the system at a small number of discrete baselines and interpolate the parameters for the entire baseline range. We present an extensive theoretical analysis of calibration and synchronization errors. We showcase three different applications of such a system for quadrotor navigation: (a) flying through a forest, (b) flying through an unknown shaped/location static/dynamic gap, and (c) accurate 3D pose detection of an independently moving object. We show that our variable baseline system is more accurate and robust in all three scenarios. To our knowledge, this is the first work that applies the concept of morphable design to achieve a variable baseline stereo vision system on a quadrotor.

D.1 Supplementary Material

The supplementary hardware tutorial and video are available at `prg.cs.umd.edu/MorphEyes.html`.

D.2 Introduction

The rapid rise of autonomous quadrotors in the recent decade has been due to the extensive research in the area of Simultaneous Localization and Mapping (SLAM)[157, 69] and Visual-Inertial Odometry (VIO)[64, 63, 158, **?**] which provide metric depth which can be directly used to perform path planning and thereby achieve navigation. Stereo camera-based solutions strike the perfect balance in accuracy, computation power and

weight to obtain a depth map when compared to LIDAR or RGB-D systems. However, they suffer from a major drawback that the error grows quadratically with depth, implying that depth in the near-range far exceeds the far-range. Depending on the scenario one might require high accuracy depth in the far range for which the near range resolution has to be increased which results in exorbitant computational cost. To this end, the experts of the field suggested that one could design a stereo system where the baseline could be varied[159, 25].

Another trend which is disrupting the quadrotor industry is morphable design which is commonly observed in birds[19]. Such a design philosophy has been applied by works in [19, 20, 21, 22, 22, 23, 24] to design a quadrotor with morphable frames to traverse narrow gaps or grasp objects. The aforementioned works have systematically modelled the dynamics and developed stable controllers during morph.

These two upcoming trends, namely, SLAM/Depth estimation and morphable designs have been mutually exclusive and can benefit greatly by being combined. We fill this void by designing a variable baseline stereo system extending the work in [159, 25] so that we can adapt the baseline for both perceptive and mechanical purposes, which is the first work of such a kind to our knowledge.

An added benefit of such a design is the usage of the concept of *Active Vision*: one can perform maneuvers and control the image acquisition process, thus introducing new constraints that were not there before [52, 53, 54, 34]. With our system one doesn't have to move the entire quadrotor which is often expensive in-terms of battery and/or time and can simply move one or more of the cameras to obtain such constraints.

D.2.1 Contributions

We develop a variable baseline stereo system which estimates disparity and a simple method to obtain extrinsic calibration between the cameras using forward kinematics of a linear actuator. We present an extensive theoretical analysis of our system and three applications for such a system, namely: (a) Navigating through a forest where the baseline changes with scene's minimum depth to keep depth error low at near-obstacles (Fig. D.1a), (b) Flying through an arbitrary shaped small static/dynamic gap by adjusting baseline for better depth estimation and reducing quadrotor footprint (Fig. D.1b) and (c) Improving the accuracy of estimating the 3D trajectory of an Independently Moving Object (IMO) by adjusting baseline (Fig. D.1c).

D.2.2 Organization of the paper

We present an extensive theoretical analysis of errors in a variable baseline stereo system in Sec. D.3. We talk about the software and hardware design of our system in Sec. D.4. Three different applications of the proposed system are discussed in Sec. D.5 and we conclude the paper in Sec. D.6.

D.3 Theoretical Analysis

Let ϵ_z, ϵ_d, b, f and z denote the depth error, disparity error, baseline, focal length and depth respectively. They are related by[159]

$$\epsilon_z = \frac{z^2}{bf}\epsilon_d$$

In the above expression, we assume that b is obtained after both camera feeds are distortion corrected, stereo-rectified and perfectly time synchronized. These corrections rely on both intrinsic and extrinsic camera calibration which can affect the disparity error ϵ_d and in-turn affect the depth error ϵ_z. We discuss this next.

Let K denote the intrinsic camera calibration matrix [160]:

$$K = \begin{bmatrix} f_x & \alpha f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Here, f_x, f_y, $c = \begin{bmatrix} c_x & c_y \end{bmatrix}^T$ and α denote the focal length in x, y, principle point and skew respectively. We'll use a subscript L or R to denote left and right cameras (assuming horizontal stereo system) respectively whenever required. We also denote the extrinsics between the left and right cameras (left camera being origin) as $\{R, T\}$.

The first step in a stereo pipeline is to undistort/distrotion rectify both feeds. Commonly, a radial-tangential model is used for distortion correction [161, 162, 163] which is given as follows

$$\mathbf{x}_r = \left(1 + k_1 r^2 + k_2 r^4 + k_5 r^6\right) \mathbf{x} + \begin{bmatrix} 2k_3 xy + k_4 \left(r^2 + 2x^2\right) \\ k_3 \left(r^2 + 2y^2\right) + 2k_4 xy \end{bmatrix} \tag{D.1}$$

$$\mathbf{x}_t = \begin{bmatrix} f_x \left(x_r + \alpha y_r\right) + c_x \\ f_y y_r + c_y \end{bmatrix}; \quad r = \sqrt{x^2 + y^2} = \|\mathbf{x}\|_2 \tag{D.2}$$

Where $\mathbf{x}_t = \begin{bmatrix} x_t & y_t \end{bmatrix}^T$, $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^T$, $k_1, \cdots, k_5$ denote the corrected co-ordinates, original co-ordinates and distortion parameters respectively.

Once this distortion correction has been performed, next step is stereo-rectification, i.e., transformation to make epipolar lines parallel in a stereo camera. This is done by relative rotation removal as follows [160]

$$\tilde{\mathbf{x}}_L = K_L R_{\text{rect}} K_L^{-1} \mathbf{x}_{t,L} \tag{D.3}$$

$$\tilde{\mathbf{x}}_R = K_R R_{\text{rect}} R^T K_R^{-1} \mathbf{x}_{t,R} \tag{D.4}$$

where, $K_L R_{\text{rect}} K_L^{-1}$ and $K_R R_{\text{rect}} R^T K_R^{-1}$ are generally denoted as H_L and H_R respectively denoting the pure rotation homographies for stereo-rectification. R_{rect} is given as follows

$$R_{\text{rect}} = \begin{bmatrix} r_x^T & r_y^T & r_z^T \end{bmatrix}^T ; \; \tilde{r}_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{D.5}$$

$$r_x = \frac{T}{\|T\|}; \; r_z = \frac{\tilde{r}_z - (\tilde{r}_z \cdot r_x)\, r_x}{\|\tilde{r}_z - (\tilde{r}_z \cdot r_x)\, r_x\|_2}; \; r_y = r_z \times r_x \tag{D.6}$$

In practice, each of the estimated quantities $\{R, T\}, k_{1,L/R}, \cdots, k_{5,L/R}, \alpha_{L/R}, f_{x/y,L/R}$ have errors. Let us denote percentage additive error in a quantity $\hat{a}$ as a_e, such that the estimate $\tilde{a} = \hat{a}(1 + (a_e/100))$ (for rotation matrix R, if R_e is an error rotation matrix, the estimate is defined as $\tilde{R} = \hat{R}R_e$). Also, due to imperfect undistortion or stereo-rectification a pixel which should ideally be at $\hat{\mathbf{x}}$ falls at $\tilde{\mathbf{x}}$ giving rise to error in both x and y directions which we denote as $e_x = \|\hat{x} - \tilde{x}\|_1$ and $e_y = \|\hat{y} - \tilde{y}\|_1$ respectively.

The interaction of each estimated quantity is complex to analyse by merely looking at the equations. Hence, to provide intuition we visualize the variation of each parameter and how it affects the error. Figs. D.2 and D.3 show the variation of e_x and e_y with

Table D.1: Relationship between e_x and e_y for errors in different parameters.

Intrinsic parameters						
	f_e	α_e	k_{1e}	k_{2e}	k^*_{3e}	k_{5e}
e_y/e_x	≈ 1	0	≈ 1	≈ 1	≈ 2	≈ 1
Extrinsic parameters						
	$T^\dagger_{xe}$	ϕ_e	θ_e	ψ_e		
$e_{x,L}/e_{x,R}$	1	0	0	0		
$e_{y,L}/e_{y,R}$	1	0	0	0		

* k_{4e} and k_{3e} are related, i.e., e_x for k_{4e} = e_y for k_{3e} and vice-versa. † Errors for T_{ye} and T_{ze} are same as T_{xe}.

different intrinsic and extrinsic quantities respectively. *Notice that the X and Y scales for each of the plots is different though trend may seem similar.*

For each plot we assume the error is only due to one parameter. For the aforementioned analysis an image size of 128×128 px. is used with a sensor size of 4.24 mm (1/3" square sensor commonly used for robotics) and focal length of [1.5, 2, 3, 4, 8, 16] mm.

In practice, the errors of all the factors are combined. The relationship between x and y errors for variation of each parameter is summarized in Table D.1.

A plot of target versus achieved baseline is shown in Fig. D.4 where the mean of 10 samples for each position is plotted as a dark line with 10σ value shown as highlight. Notice that we achieve almost a perfect straight line of slope 1. The maximum calibration error of 1.83 mm was observed which is about 0.7% of the baseline.

The error in the axis parallel to the baseline of the stereo-camera (image X-axis for a horizontal stereo system and image Y-axis for a vertical stereo system) due to errors in intrinsics (including lens distortion) and extrinsics (stereo-rectification) is important only when the error magnitude is significantly larger than the search window (in-case of classical block matching). In such a case this error is directly related since the disparity error ϵ_d as it would lead to false matches (assuming zero matching error or perfect matching). The error in the axis normal to the baseline axis has a more complex pattern and depends

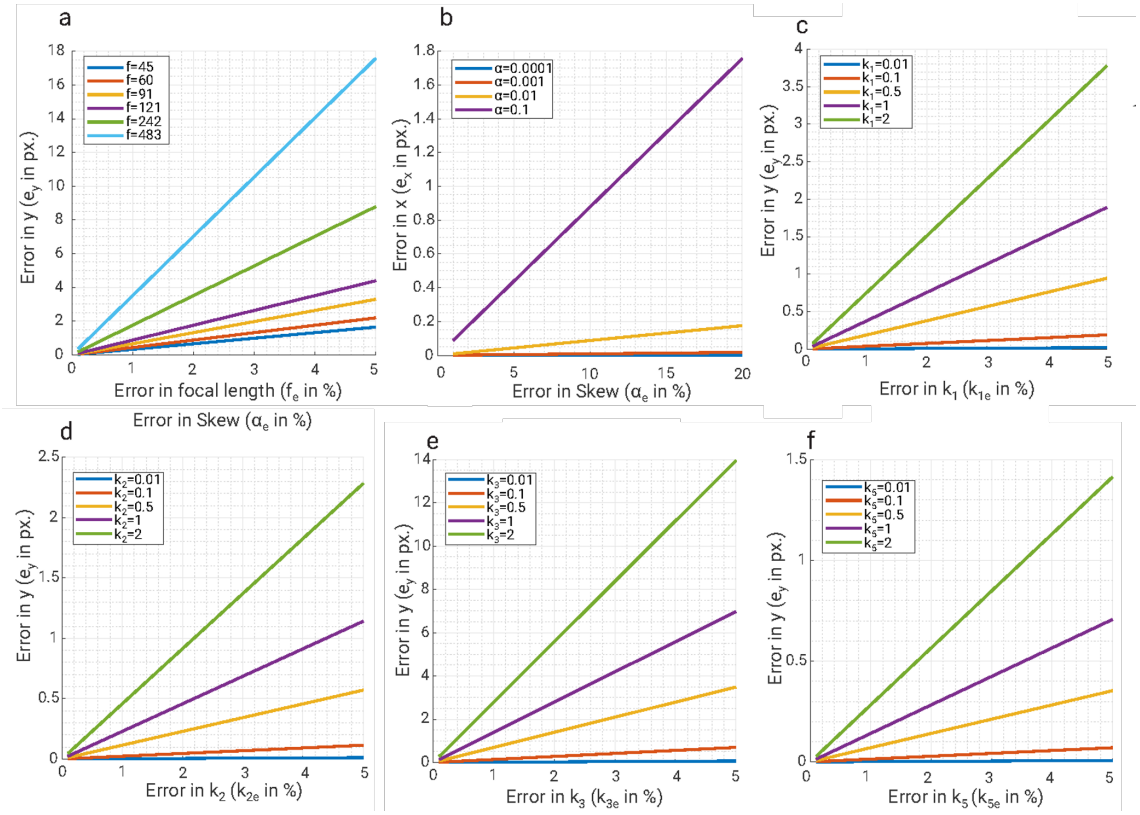


Figure D.2: Error in pixel location due to error in various estimated intrinsic parameters. (a) e_y vs. f_e, (b) e_x vs. α_e, (c) e_x vs. k_{1e}, (d) e_y vs. k_{2e}, (e) e_y vs. k_{3e}, (f) e_y vs. k_{5e}. *Notice that the X and Y scales for each of the plots is different though trend may seem similar.*

on the structure of the scene, i.e., if a pixel is shifted up/down (in-case of horizontal stereo system), the pixel could land at the edge of object boundaries giving a huge disparity error ϵ_d (since it is estimated by matching along the baseline axis) [164, 165].

Off-synchronization between the stereo cameras can also produce false disparity. The criterion for a disparity error (first order Taylor series approximation) of k px. when the camera is moving with a velocity of v ms^{-1} (acceleration of zero) and an synchronization error time of δt for a scene at depth Z is given by

$$v\delta t \geq \frac{kZ^2}{bf - kZ}$$

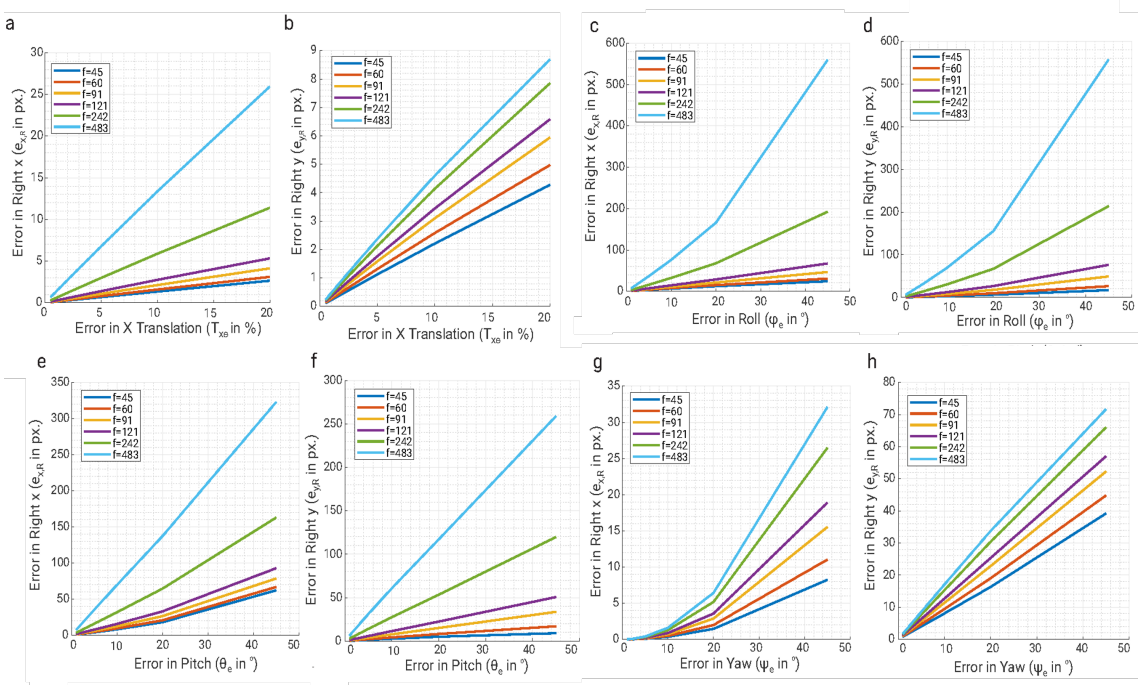


Figure D.3: Error in pixel location in right camera due to error in various estimated extrinsic parameters. (a) $e_{x,R}$ vs. T_{xe}, (b) $e_{y,R}$ vs. T_{xe}, (c) $e_{x,R}$ vs. ϕ_e, (d) $e_{y,R}$ vs. ϕ_e, (e) $e_{x,R}$ vs. θ_e, (f) $e_{y,R}$ vs. θ_e, (g) $e_{x,R}$ vs. ψ_e, (h) $e_{y,R}$ vs. ψ_e. *Notice that the X and Y scales for each of the plots is different though trend may seem similar.*

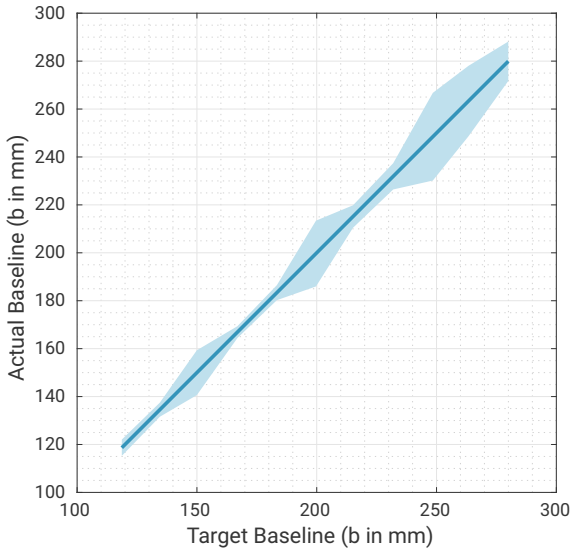


Figure D.4: Target vs. achieved baseline. *The highlight shows the 10σ value.*

Fig. D.5 shows the maximum achievable velocity to keep disparity error under $k = 1$ px. for $Z = 1$ m. Note that, in our work we do not vary the resolution of the sensor and only

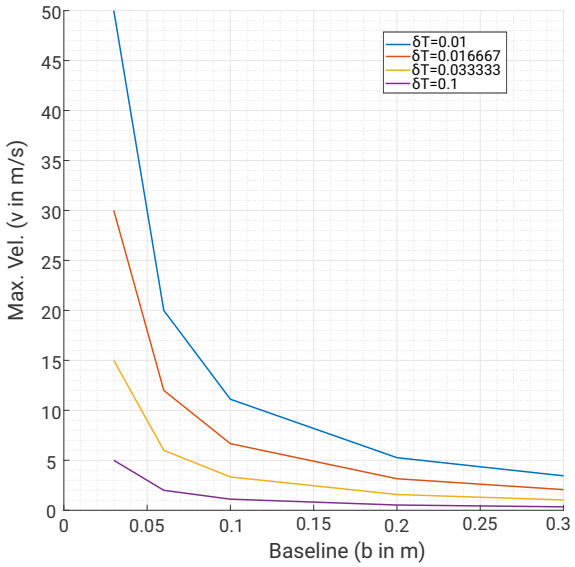


Figure D.5: Max. velocity to have a disparity error lower than k px. vs. baseline for different time synchronization errors (δt).

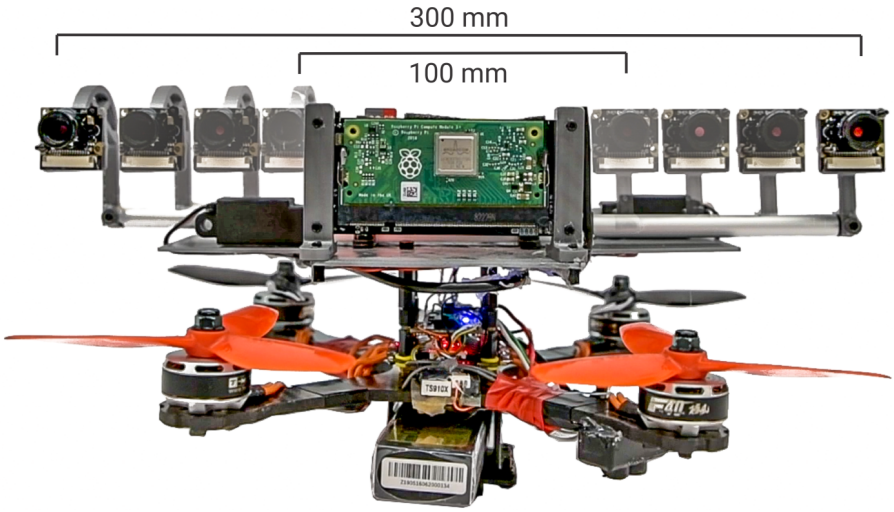


Figure D.6: Variation of baseline from 100 mm to 300 mm. Notice that the stereo system is bigger than the quadrotor at the largest baseline.

vary the baseline to increase far-range depth accuracy with the loss of near-range depth accuracy. A detailed analysis of this can be found in the fixed-resolution stereo section of [159].

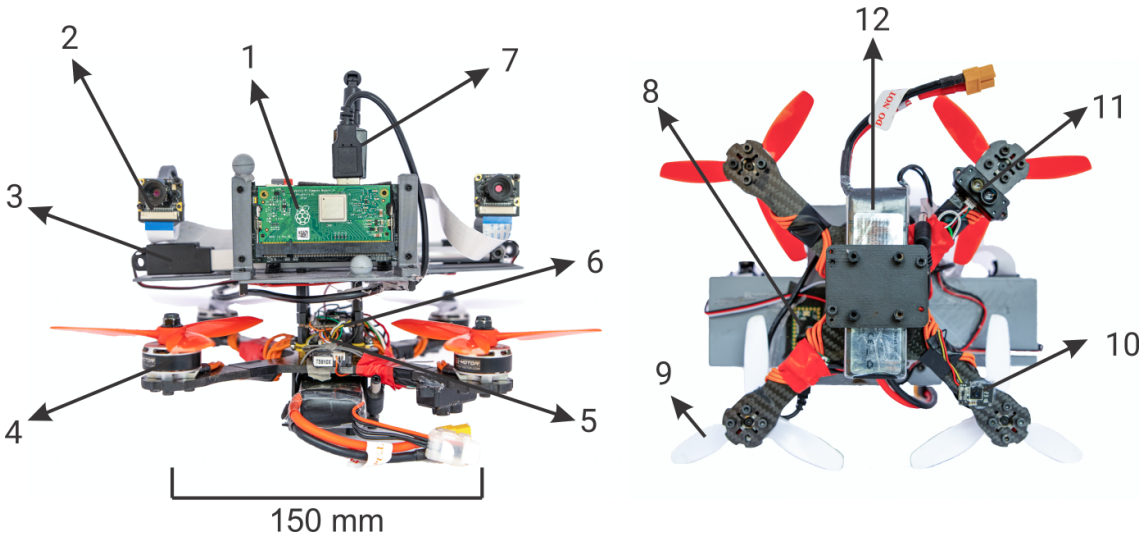


Figure D.7: Quadrotor platform used for experiments. (1) RaspberryPi 3B+ compute module, (2) Stereo camera, (3) Actuonix linear servo, (4) T-Motor F40 III Motors, (5) T-Motor F55A 4-in-1 ESC, (6) Holybro Kakute F7 flight controller, (7) WiFi module, (8) Teensy 3.2 microcontroller, (9) 5045×3 propeller, (10) Optical Flow module, (11) TFMini lidar, (12) 3S LiPo battery.

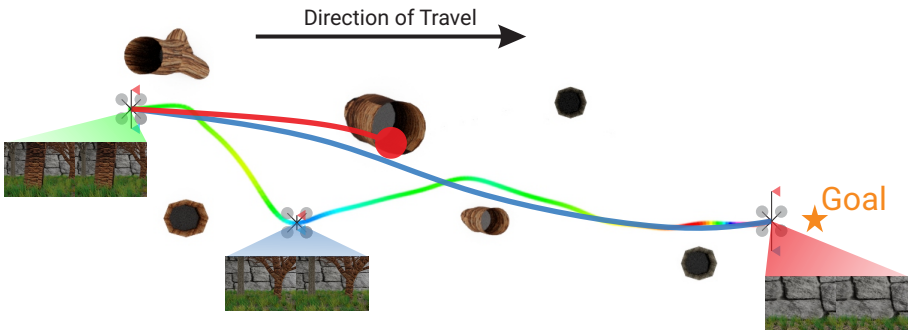


Figure D.8: Variable baseline stereo performance in simulated forest flight when compared to small and large baselines. Note that the large baseline system crashes (red curve) and small baseline system (blue curve) can traverse the scene but is about $4\times$ slower than the variable baseline system. The baseline for the variable baseline case is color-coded as jet (blue to red indicates small to large baseline).

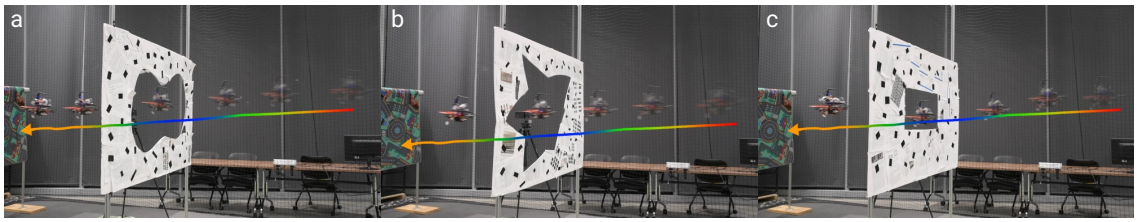


Figure D.9: Sequence of images of quadrotor going through different shaped gaps: (a) Infinity, (b) Goku, (c) Rectangle. In all the cases, the baseline of the stereo system is changing and is colored coded as jet (blue to red indicates 100 mm to 300 mm baseline).

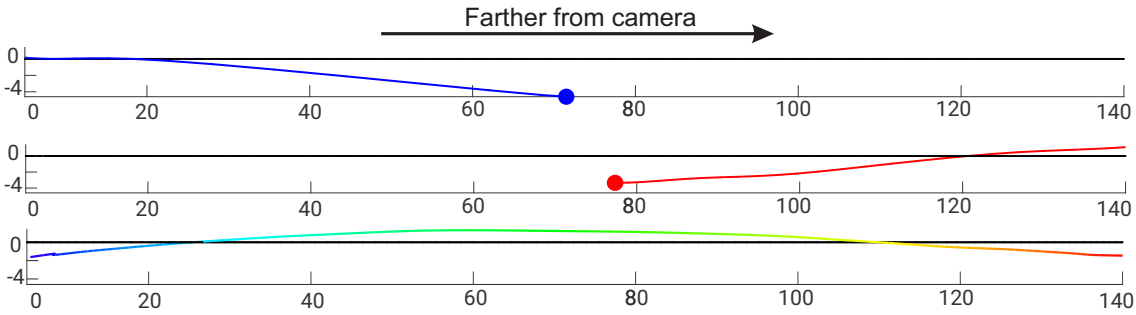


Figure D.10: Variable baseline stereo performance in simulated 3D IMO detection when compared to small and large baselines. Note that both the large baseline system (red curve) and small baseline system (blue curve) lose detection of the IMO at different parts of the scene (dots). The baseline for the variable baseline case is color-coded as jet (blue to red indicates small to large baseline). Black curve (horizontal line at zero vertical axis) represents the ground truth trajectory of the object.

D.4 Hardware and Software Design

D.4.1 Hardware Setup

Our variable baseline stereo system is built using a StereoPi[1] and Actuonix L12-R linear servos[2]. The StereoPi is a stereo camera system based on the RaspberryPi 3B+ coupled to two waveshare cameras with a Omnivision OV5647 sensor mated to a 3.96 mm f/2.6 lens with a diagonal field of view of 56.8°. The Actunonix linear servo used has a

[1] https://stereopi.com/

[2] https://www.actuonix.com/L12-R-Linear-Servo-For-Radio-Control-p/l12-r.htm

stroke length of 100 mm and a gear ratio of 50:1 (Fig. D.6 shows the system with different baselines from 100 mm to 300 mm). These components were chosen for their ease of use and cost-effectiveness for the performance obtained. They can easily be replaced with counterparts from other manufacturers.

D.4.2 Software Stack

Our software stack includes three sub-modules: (a) Camera drivers and disparity estimation, (b) Camera calibration prediction, and (c) Control of camera baseline. Each sub-module is discussed next.

The camera drivers are implemented in C++ (camera feeds are software synchronized with maximum synchronization error of 1ms) which are Robot Operating System (ROS) compatible. Disparity is obtained using a simplified version of [166].

The cameras were calibrated using the MATLAB's[3] camera calibration toolbox using images of the checkerboard on ten equally spaced baselines between 100 and 300 mm. The extrinsics are represented as a dual-quaternion [167, 168]. We interpolate between extrinsics on-the-fly (for intermediate values) using the screw linear interpolation (ScLERP) [169] for its speed and accuracy since the motion of the cameras are linear.

Finally, we control the camera baseline using ROS communicating with a Teensy 3.2 microcontroller which in-turn controls the servos. The camera baseline control methodology depends on the application and is explained in Secs. D.5.3, D.5.4 and D.5.5.

[3] https://www.mathworks.com/products/matlab.html

D.5 Experiments: Applications

D.5.1 Quadrotor Platform

The quadrotor used in the experiments is a custom-built platform called PRGCorgi210α[4] (Fig. D.7). The platform is built on a X-shaped 210 mm sized (motor to motor dimension) racing frame. The motors used are T-Motor F40III KV2400 mated to 5040×3 propellers. The lower level controller and position hold is handled by ArduCopter 4.0.4dev firmware running on the Holybro Kakute F7 flight controller mated to an optical flow sensor and TFMini LIDAR as altimeter source. All the higher level navigational commands are sent by the companion computer (RaspberryPi) using RC-Override to the flight controller running in Loiter mode using MAVROS. The RaspberryPi runs all the vision and planning algorithms on-board at 1 Hz. The quadrotor take-off weight including the battery is 780 g and has a thrust to weight ratio of 2:1.

D.5.2 Simulation Environment

We rendered stereo camera images with variable baseline using Blender®[5] 3D creation software. We modelled two scenes for different experiments: (a) Stereo Camera Navigating through a Forest (Fig. D.8) with a variety of trees and grass, and (b) a room with a cube moving away from the stereo camera (Fig. D.1c). The camera baseline changes based on the depth of the scene which is estimated using [166] (Secs. D.5.3, D.5.4 and D.5.5). The images are rendered in 640 × 480 px. resolution using Cycles

[4] https://github.com/prgumd/PRGFlyt/wiki/PRGCorgi
[5] https://www.blender.org/

render engine.

As mentioned before, we present three applications of our a variable baseline stereo system for quadrotor navigation which are explained in the next three sub-sections.

D.5.3 Forest Navigation

In this scenario, we are tasked with the problem of navigating through a forest. The aim is to navigate to the goal location without any collisions. We compute per-pixel depth from the disparity obtained using $Z = \frac{bf}{d}$. The control policy is used to change the current heading direction (velocity vector) using a simple Proportional-Integrative-Derivative (PID) controller to reach the goal whilst avoiding collisions. The current desired direction vector $\tilde{\mathbf{v}}_g$ is given by a weighted sum of the goal direction $\mathbf{v}_g$ and free path direction $\mathbf{v}_{\text{free}}$). The goal direction can be treated as a global path planner and the free path direction vector can be treated as a local path planner. The intuition is that when there is free space the global planner takes a larger weight and vice-versa to avoid collisions whilst still going towards the goal. Note that this strategy is not necessarily optimal and other strategies [170, 171] can be used to achieve optimality which require building a map which is generally expensive.

The free path direction $\mathbf{v}_{\text{free}}$ is obtained as follows: Consider a small neighborhood $\mathcal{N}$ on the image plane centered around where the goal direction vector intersects the image plane (this could be outside the visible region in which case the edge of the frame closest to the goal direction is chosen). $\mathbf{v}_{\text{free}}$ is chosen as the center of the largest free space in the neighborhood $\mathcal{N}$.

Now, let Z_{close} denote the closest depth value in this neighborhood. The neighborhood $\mathcal{N}$ is chosen to minimize the control effort and to avoid large changes in velocity (accelerations) and can be thought of as the projection of the quadrotor with safety margin onto the depth image. The final control equations are given by

$$\tilde{\mathbf{v}}_g = (1-w)\mathbf{v}_g + w\mathbf{v}_{\text{free}};\ \ w \in [0,1] \tag{D.7}$$

$$\mathbf{e}(t) = \tilde{\mathbf{v}}_g(t) - \tilde{\mathbf{v}}_{\text{curr}}(t) \tag{D.8}$$

$$\mathbf{u}(t) = K_p\mathbf{e}(t) + K_i\int_0^{\tau}\mathbf{e}(\tau)d\tau + K_d\frac{d\mathbf{e}(t)}{dt} \tag{D.9}$$

$$w = \frac{1}{1+e^{\frac{-1}{Z_{\text{close}}}}};\ \ Z_{\text{close}} = \min Z(x,y)\ \forall\ \{x,y\} \subset \mathcal{N} \tag{D.10}$$

where $\tilde{\mathbf{v}}_{\text{curr}}(t)$ is the estimated current heading direction. Notice how the current/local goal vector gets dominated by $\mathbf{v}_{\text{free}}$) when we get close to a collision ($w \to 1$) and by $\mathbf{v}_g$ when no collision is detected ($w \to 0$). As a safety mechanism if we detect $Z_{\text{close}} \leq \tau_{\text{safe}}$, where τ_{safe} is a safe braking corridor, stop the quadrotor in its position to avoid collision and then move towards the free space.

Finally, we adjust the baseline with respect to Z_{close}, i.e., $b = KZ_{\text{close}}$ (where $K > 0$ is a tunable gain parameter). For implementation we use a low-pass filtered Z_{close} to avoid matching artifacts.

The above method was tested both in simulated and real environments (Figs. D.8 and D.1a). We can observe in Fig. D.8 that the quadrotor with large baseline crashes since the first tree is not visible and the quadrotor with a smaller baseline can navigate to

the goal but is $4\times$ slower than the variable baseline quadrotor.

D.5.4 Flying through a static/dynamic unknown shaped gap

This scenario is inspired by the problem presented in GapFlyt[34]. A quadrotor is present in a scene equipped with a stereo camera setup whose baseline can be changed. The per-pixel depth 'seen' by the camera can be modelled by a univariate bimodal Gaussian distribution. The contour $\mathcal{C}$ of the gap/opening (on the image plane) is defined as the pixels which have the maximum spatial depth disparity. Since we have depth from disparity, this becomes a simple clustering problem. All the points which belong to the 'far' cluster (average higher Z value) are called background $\mathcal{B}$ and points in the 'close' cluster are called foreground $\mathcal{F}$. Similar to [34], we track both $\mathcal{F}$ and $\mathcal{B}$ separately (since its computationally cheaper and faster than running detection at every step) to infer $\mathcal{C}$. Contrary to [34], when we lose track or for a dynamic gap, we re-detect the gap by clustering on the depth image. Our control policy is to align the center of the image with the safest point $\mathbf{x_s}$ defined as follows

$$\mathbf{x}_s = \begin{cases} \mathbb{M}\left(\mathcal{F}\right), & \overline{\overline{\mathcal{F}}} \geq \overline{\overline{\mathcal{B}}} \\ \mathbb{M}\left(\mathcal{B}\right), & \text{otherwise} \end{cases}$$

where $\mathbb{M}$ denotes the median operator.

Also, note that since we do not move the quadrotor to detect the gap, we can fly through dynamic gaps by invoking depth based gap detection (Fig. D.1b). Again, we use a PID controller to align the center of the image with $\mathbf{x_s}$.

Finally, we adjust the baseline with respect to our closeness to the gap, i.e., $b = K\mathbb{M}(Z_{\mathcal{C}})$. Here, $Z_{\mathcal{C}}$ denotes the set of depth of all contour points and $K > 0$ is a tunable gain parameter. Such adjustment of baseline makes depth estimation more accurate when the gap is near and also makes the quadrotor smaller making it easier to traverse the gap.

We improve on speed of gap detection by $80\times$ (when using same computer for processing) when compared to [34] as our proposed approach does not require any active quadrotor maneuvers for gap detection. We also improve on speed of gap traversal by 20% (3 ms^{-1} over 2 ms^{-1}) and smallest gap clearance by 20% (4 cm over 5 cm) when compared to [34]. We show our gap traversal sequences for both static and dynamic window in Figs. D.9 and D.1b respectively.

Note that we can replace the proposed variable baseline solution with a fixed multiple camera setup [172] to estimate depth at different baselines. *Although, such a solution would not be morphable, thus hindering the quadrotor to go through narrow gaps.*

D.5.5 Accurate IMO Detection

Consider a scenario where we want to accurately detect the IMO in 3D either to dodge or pursue the target [35, 173]. The first step is to detect the IMO then to track it. To make the problem simple, we assume that only one IMO is visible in the field of view. The IMO is first detected by clustering the tracklets' motion [174] along with depth. Then the 3D trajectory of the IMO is tracked by computing depth from disparity. To improve the accuracy of the estimated trajectory (especially in situations when the object is very close or very far from the camera) we adjust the baseline with respect to the median depth

of the object, i.e., $b = K\mathbb{M}\left(Z_{\text{IMO}}\right)$. Here, Z_{IMO} denotes the set of depth of all IMO points and $K > 0$ is a tunable gain parameter. We show that using a variable baseline system we can track the object in a depth range larger than that just by using small/large baseline and with higher accuracy (Fig. D.10). Fig. D.10 shows that the detection of the IMO fails in both small and large baseline cases when the object is very far or near to the camera but the variable baseline system can detect the object through the entire range.

D.6 Conclusions

We applied the philosophy of morphable design to a stereo system to increase the depth sensing range. We discuss the hardware and software design of such a system for deployment on a mini-quadrotor. We present a simple calibration method and interpolate between the calibration values for the entire range using simple forward kinematics of the system. A thorough analysis of errors due to miscalibration is provided and we hope this will serve as a blueprint for researchers and practitioners alike. Finally, we demonstrate the utility of such a design philosophy in three different applications: (a) flying through a forest, (b) flying through a static/dynamic gap of unknown shape and location, and (c) accurate 3D pose detection of an independently moving object. We show that our variable baseline system is more accurate and robust in all the three scenarios when compared against a fixed baseline system. We feel that this would serve as a motivator for researchers and practitioners to incorporate sensors on controllable moving parts on robots to enable better perception capabilities.

Appendix E: SalientDSO

Huai-Jen Liang*, Nitin J. Sanket*, Cornelia Fermüller, Yiannis Aloimonos "SalientDSO: Bringing Attention to Direct Sparse Odometry", *IEEE Transactions on Automation Science and Engineering (T-ASE)*, (2019) Vol. 16, No. 4, pp. 1619-1626.

DOI: 10.1109/TASE.2019.2900980.

SalientDSO: Bringing Attention to Direct Sparse Odometry

Huai-Jen Liang*, Nitin J. Sanket*, Cornelia Fermüller, Yiannis Aloimonos

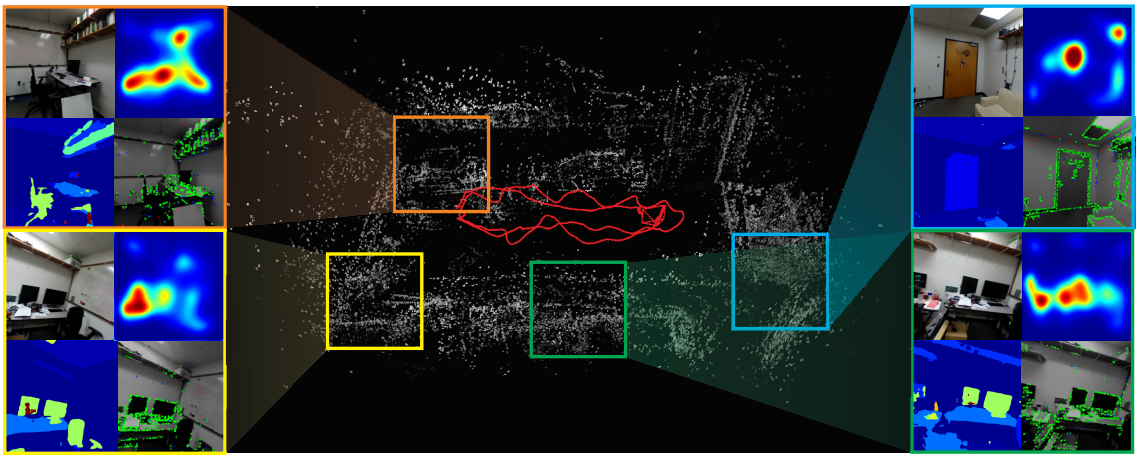


Figure E.1: Sample point-cloud output of SalientDSO which does not have loop closure or global bundle adjustment. Each inset (color-coded to suite the respective location on the map) in clockwise direction from top left show the corresponding image, saliency, scene parsing outputs and active features. Observe that features from non-informative regions are almost removed approaching object-centric odometry.

Abstract — Although cluttered indoor scenes have a lot of useful high-level semantic information which can be used for mapping and localization, most Visual Odometry (VO) algorithms rely on the usage of geometric features such as points, lines and planes. Lately, driven by this idea, the joint optimization of semantic labels and estimating odometry has gained popularity in the robotics community. This joint optimization method is accurate but is generally very slow. At the same time, in the vision community, direct and sparse approaches for VO have stricken the right balance between speed and accuracy.

We merge the successes of these two communities and present a pre-processing method to incorporate semantic information in the form of visual saliency to Direct Sparse Odometry – a highly successful direct sparse VO algorithm. We also present a framework to filter the visual saliency based on scene parsing. Our framework *SalientDSO*, relies on the widely successful deep learning based approaches for visual saliency and scene parsing which drives the feature selection for obtaining highly-accurate and robust VO even in the presence of as few as 40 point features per frame. We provide extensive quantitative evaluation of SalientDSO on the ICL-NUIM and TUM monoVO datasets and show that we outperform DSO and ORB-SLAM – two very popular state-of-the-art approaches in literature. We also collect and publicly release a CVL-UMD dataset which contains two indoor cluttered sequences on which we show qualitative evaluations. To our knowledge this is the first work to use visual saliency and scene parsing to drive the feature selection in direct VO.

Note to Practitioners — The algorithm of estimating the camera motion from a set of moving camera frames/images is commonly called Visual Odometry (VO). This problem has many applications ranging from building a 3D map of the scene for the robot to navigate, grasp and so on. Any VO algorithm must be fast, robust and with low drift (low accumulation in error). These desired functions are generally obtained by selecting "good" features in an image which in the computer vision sense turn out to be "corners". However, when we constrain the setting to an indoor scene with a lot of clutter, we have a lot of objects which can be used to obtain "good" features from both a computer vision sense and a conceptual sense. We use this philosophy and present a pre-processing

method to select better features as compared to a traditional VO pipeline using only geometric features and improve the robustness of the state-of-the-art VO method: Direct Sparse Odometry, obtaining more accurate and robust results even with lesser number of features. We evaluate our methods on three different datasets: ICL-NUIM, TUM monoVO and CVL-UMD. The CVL-UMD dataset was collected by the authors of this paper to demonstrate the robustness of our approach, namely, SalientDSO in cluttered indoor scenes.

E.1 Supplementary Material

The accompanying video, dataset and open-source code is available at prg.cs.umd.edu/SalientDSO.html.

E.2 Introduction and Philosophy

Simultaneous Localization and Mapping (SLAM) and Visual Odometry (VO) algorithms have taken center stage in the recent years due to their wide-spread usage. They play a prominent part in the perception and planning pipelines of self-driving cars, autonomous quadrotors, augmented and virtual reality. The never ending quest to come up with real-time solutions for these methods whilst being as accurate as their offline counterparts has led to alternative problem formulations in terms of constraints and optimization methods [175, 176, 177, 178].

Not so long ago, the field was dominated by indirect methods [175, 176, 157, 179] which rely on feature matching and foundations of multi-view geometry coupled with

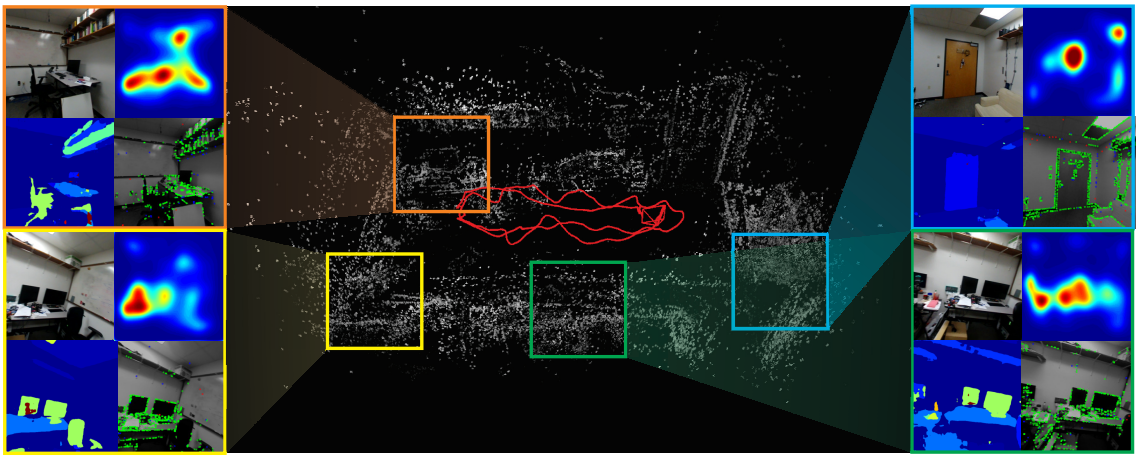


Figure E.2: Sample point-cloud output of SalientDSO which does not have loop closure or global bundle adjustment. Each inset (color-coded to suite the respective location on the map) in clockwise direction from top left show the corresponding image, saliency, scene parsing outputs and active features. Observe that features from non-informative regions are almost removed approaching object-centric odometry.

windowed optimization to build a map of the scene and obtain accurate poses. These approaches are based on the low-level geometric features and do not work very well with environments with repeating structures and/or texture-less surfaces. Some works have improved upon the previous approaches in-terms of speed and accuracy by incorporating prior knowledge such as the dynamics of the system and/or data from more sensors such as inertial measurement units [64], time-of-flight sensors [69] etc. However, minimalism is a trend forward, i.e., trying to achieve the same tasks with a minimal number of sensors [34]. In the scope of this paper, we focus on a monocular VO solution. The current state-of-the-art in monocular approaches which have the best compromise of speed and accuracy are direct sparse approaches such as Direct Sparse Odometry (DSO)[71].

However, object centric SLAM approaches are more robust by nature due to the high level semantics used in their formulation. Lately, joint optimization of 3D poses, structure and labelled object locations has improved the state-of-the-art significantly [180].

These frameworks rely on the widely successful deep learning based object recognition engine and pose graph optimization frameworks, combining both low-level geometric features and the high-level semantics.

However, humans perform the task of mapping very differently. The human visual system interprets the scene for various tasks like recognition [181], segmentation [182], tracking [183] and navigation by making a series of fixations [184]. This is called the Active approach [52, 53, 54, 34], whilst the traditional approach is called the Passive approach (See Table E.1). These fixations lie in the proto-segmentation of the salient objects/locations in the scene. The word proto-segmentation refers to the fact that a segmentation around the fixation point may lead to partial/complete segmentation of an object, which depends on the scenario. Solving the problem of recognition and tracking along with segmentation is like a chicken-egg problem. One would need a good segmentation for recognition and tracking and vice-versa. An Expectation-Maximization (EM) type of scheme, where one would jointly/alternatively optimize for the segmentation and recognition/tracking has gained popularity in literature lately, due to the advancement of fast and accurate optimization frameworks.

At the same time, very recently, this philosophy of fixation and attention has started to gain popularity in the robot navigation community [185, 180, 186, 187]. This is based on the fact that humans perform the task of mapping very differently from how it has been done in the robotics literature. They build "semantic/toplogical" maps to traverse the scene. This work combines the concepts used by humans and robotics literature to present a framework of indoor VO in which the features are selected based on a visual saliency map that is obtained by human eye tracking data. Our work is consistent with

Table E.1: Active vs Passive approach for computer vision tasks.

Task	Passive approach	Active approach
Segmentation	Graph cut or super-pixel based methods.	Fixation based region segmentation and recognition in a feedback loop.
Recognition	Sliding window of filter banks with a classification algorithm for final prediction.	Saliency/fixation based segmentation/clustering followed by selection of attributes and sliding window of filters with a simple classification algorithm.
Tracking and Failure recovery	Making an online dictionary for robustness against changes and use detection for failure recovery.	Tightly couple saliency into the tracking filter to reduce search space and use salient regions for failure recovery. By doing so, we introduce high level semantics into the low level processes (feedback).
Navigation and Mapping	Map based on features selected on image gradients.	Map only using salient region features or objects obtained using fixation based segmentation. Take advantage of the semantic relationships between differently labeled regions.

major theoretical works in the field of Active Perception [54, 188]. In [54], it is suggested that attention mechanisms precede any visual computation and in [188], that solutions to active perception problems may come from multi-modal fusion. Indeed our approach can be generalized to different inputs (LIDAR, radar, tactile, audio etc.).

This work aims to mimic the qualitative human vision in the framework of direct VO. The key contributions of this paper are:

- We present a framework of indoor visual odometry in which the features are selected based on a visual saliency map (Sample output is shown in Fig. E.2).
- We present a method to filter saliency map based on scene parsing.
- We provide experimental results on various simulated and real indoor environments

to demonstrate the improved performance of the proposed approach with comparisons to the state-of-the-art.

- We also make our CVL-UMD dataset and the source code open-source to facilitate future research.

The rest of the paper is organized as follows: Sec. E.3 presents the different parts of the proposed SalientDSO framework along with the preliminaries required. Sec. E.4 describes the visual saliency and scene parsing driven point selection algorithm used in SalientDSO. Detailed experiments along with quantitative and qualitative results are given in Sec. E.5. We finally conclude the paper in Sec. E.6 with parting thoughts on future work.

E.3 SalientDSO Framework

SalientDSO's framework is composed of a pre-processing step and a VO backbone. The VO backbone is responsible for initializing and tracking camera pose and optimizing all model parameters. The pre-processing step involves the saliency prediction and scene parsing using deep Convolutional Neural Networks (CNNs) and later using these outputs to select features/points. Fig. E.3 shows the algorithmic overview of SalientDSO, where blue parts of the figure show our contributions (which constitute the pre-processing step). Each component of SalientDSO is discussed in brief next.

We adopt DSO [71] as the backbone VO in SalientDSO. In brief, DSO[71] proposed a direct sparse model to jointly optimize all parameters (camera intrinsics, camera extrinsics, and inverse-depth values for feature points) and perform windowed bundle

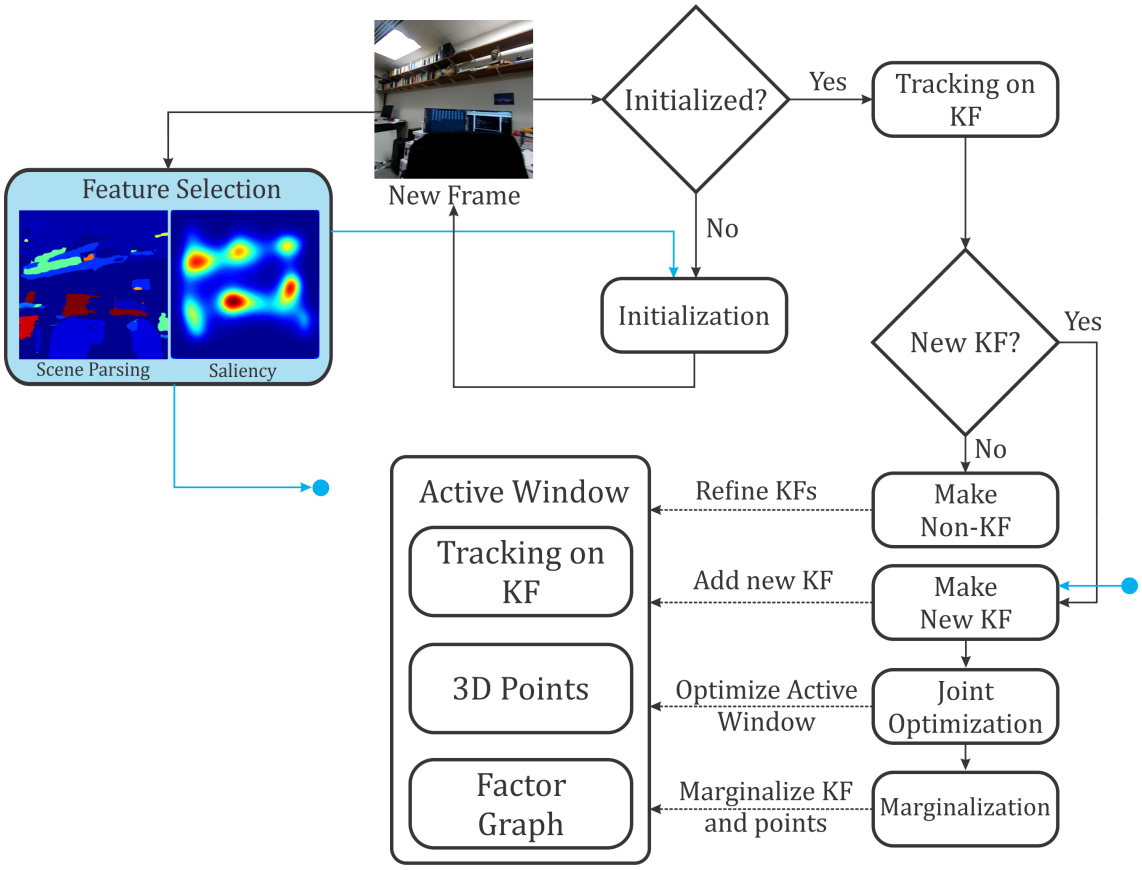


Figure E.3: Algorithmic overview of SalientDSO, blue parts show our contributions. Here KF is the abbreviation for Key Frame.

adjustment. It contains a front end and a back end detailed next.

Front-end: The front-end part of algorithm handles tracking, keyframe creation, outlier rejection, parameters initialization, candidate point activation, and marginalization as described in [71]. The candidate point selection in DSO is replaced by our novel approach described in Sec.E.4.3.

Back-end: The back end contains a factor graph which performs continuous windowed optimization using the approach given in [189]. It optimizes E_{photo} using Gaussian-Newton algorithm in a sliding window manner. The error functions are defined as in [71].

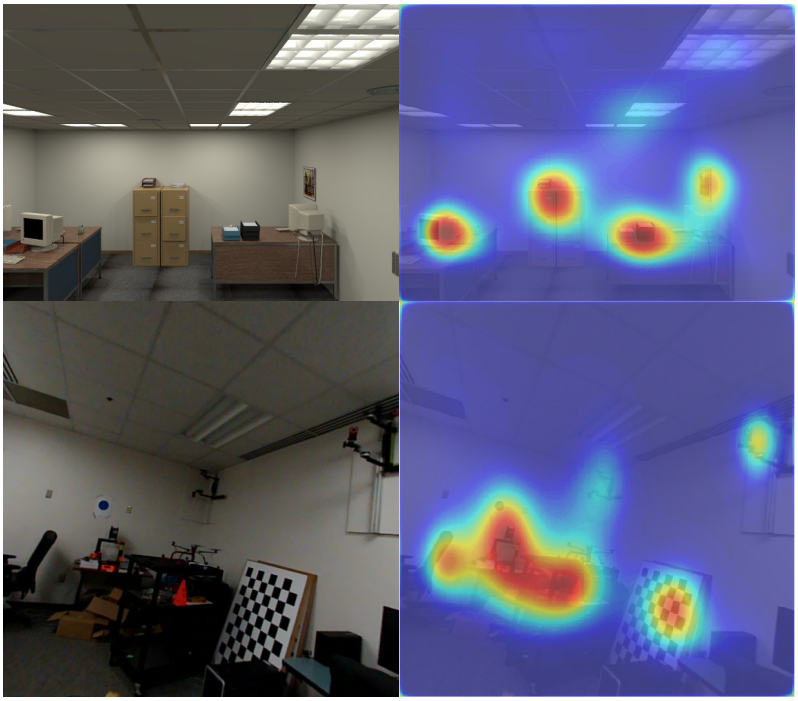


Figure E.4: Left column: Input image, Right column: Saliency overlayed on input image.

E.4 Point selection based on visual saliency and scene parsing

E.4.1 Visual Saliency Prediction

Visual saliency is defined as the amount of attention a human would give to each pixel in an image. This is quantitatively measured as the average time a person's gaze rests on each pixel in the image. Prediction of saliency is a hard problem and data driven approaches have lately excelled at this task. We adopt SalGAN [1] for saliency prediction in SalientDSO. Some sample results are shown in Fig. E.4. One can clearly notice that walls, floors, and ceilings have lower probability of being fixated on, which is the main idea of the proposed framework.

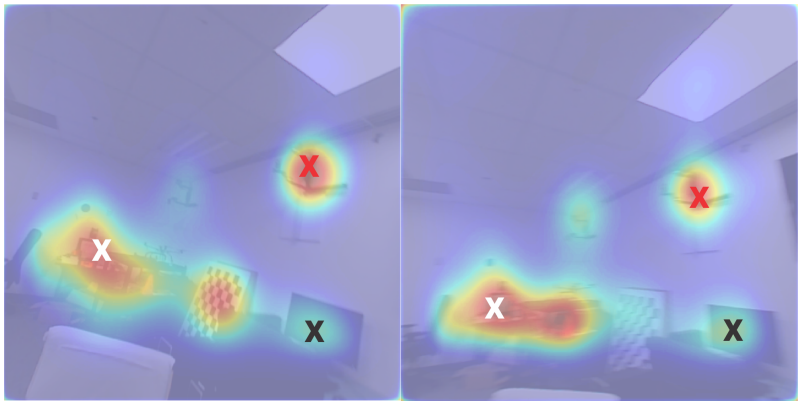


Figure E.5: Variation of Saliency Map due to changes in illumination and viewpoint. Notice that the fixation still remains inside the same object but the saliency map varies. The crosses of respective color highlight the fixation in the respective images.

E.4.2 Filtering saliency using semantic information

The saliency produced by SalGAN is concentrated around a fixation point inside the object and is fuzzy. The saliency map is robust with respect to the fixations remaining inside the object but the same fixation point is generally not obtained as the viewpoint and illumination changes [190, 191]. Notice in Fig. E.5 how the fixation point location is not robust but the proto-segmentation obtained by the fixation is robust (the fixation point remains inside the same object) with respect to changing illumination and viewpoint conditions. In this subsection, we utilize semantic information to filter the saliency so as to use features from proto-segmentation of an object which is more robust to viewpoint and illumination changes. The idea is to weigh down the saliency of uninformative regions, such as walls, ceilings and floors. These regions are uninformative from the computer vision sense of them being weak corners. Semantically objects/regions with unique color or size or shape are more informative than regions such as walls, ceilings and floors which have non-unique and non-discriminative features.

Algorithm 2: Saliency prediction and filtering.

Data: Input image I, Pre-defined weights w_C
Result: Predicted final saliency $\hat{S}^{\text{final}}$

$\hat{S}$ = SalGAN(I);
C = PSPNet(I);
for $\forall \{x_j, y_j\} \in I$ **do**
    $\hat{S}_j^{\text{weighted}} = w_C(C_j)\hat{S}_j$;
end
for $\forall \{x_j, y_j\} \in I$ **do**
    $\hat{S}_j^{\text{final}} = \text{median}\left\{\hat{S}_i^{\text{weighted}}, \forall i \in C_j\right\}$;
end

To obtain semantic information from a scene, we adopt Pyramid Scene Parsing [192] for retrieving semantic labels of every pixel in an image.

The predicted saliency map $\hat{S}$ is filtered using the PSPNet's per-pixel semantic output S:

$$\hat{S}_j^{\text{weighted}} = w_C(C_j)\hat{S}_j \tag{E.1}$$

Here, w_C are the pre-defined weights obtained empirically for different classes. To smoothen and maintain a consistent saliency map for each class, each pixel is replaced by the median of saliency for its respective class:

$$\hat{S}_j^{\text{final}} = \text{median}\left\{\hat{S}_i^{\text{weighted}}, \forall i \in C_j\right\} \tag{E.2}$$

All steps to generate $\hat{S}^{\text{final}}$ are summarized in Algorithm 2.

Algorithm 3: Saliency based points selection.

Data: Desired number of points N_{des}, s_{smooth}, $\hat{S}^{\text{final}}$
Result: Selected points
Initialize selected point set as $\{\emptyset\}$, $N_{\text{sel}} = 0$;
while $N_{sel} < N_{des}$ **do**
    Randomly select a patch M from distribution P_S;
    Split M into $d \times d$ blocks;
    for *each* $4d \times 4d$ *block* **do**
        for *each* $2d \times 2d$ *block* **do**
            for *each* $d \times d$ *block* **do**
                Select a point with the highest gradient which surpass the gradient threshold;
            end
            if *no selected point in this block* **then**
                Select a point with the highest gradient which surpass the weaker gradient threshold;
            end
        end
        if *no selected point in this block* **then**
            Select a point with the highest gradient which surpass the much weaker gradient threshold;
        end
    end
    $N_{\text{sel}} = N_{\text{sel}} +$ the number of selected points;
end

E.4.3 Features/Points selection

Instead of uniformly selecting candidate points from an image as in DSO, we select points based on saliency. This is very helpful where the scene has a lot of objects or clutter which can be found generally in indoor scenes.

First, we split an image into $K \times K$ patches. For a patch M_i, we not only compute the median of gradient as a region-adaptive threshold, but also compute the median of saliency as a region-adaptive sampling weight sw_i. Therefore, for each patch, the sam-

pling weight sw_i is computed as:

$$sw_i = \text{median}\left\{\hat{S}_j^{\text{final}}, \forall j \in M_i\right\} + s_{\text{smooth}} \tag{E.3}$$

where s_{smooth} is a laplacian smoothing term used to control the bias on a salient region and the probability of a patch M_i being sampled is:

$$\boldsymbol{P}_S(M_i) = \frac{sw_i}{\sum_{m \in M} sw_m} \tag{E.4}$$

Secondly, once a patch M_i has been selected, we further split M_i into $d \times d$ blocks. For each block, we select the pixel with the highest gradient only if it surpasses the region-adaptive threshold. With this strategy, we can select points which are well distributed in this salient region. In order to extract information from where no high-gradient pixels are present, we follow the same approach as DSO and run two more passes to select pixels with weaker gradient in a larger sub-region with a lower gradient threshold and an increased d. A summary of the whole selection method is given in Algorithm 3.

Fig. E.6 shows the selected points for some example scenes. We compare our selection based on saliency to the uniform selection adopted by DSO. One can easily notice that texture-less and mostly identical parts, such as walls, floors and ceilings, are down weighted in our pipeline. As demonstrated in Section E.5, this helps us trade the weak features on the floors and ceilings for weak features on objects where the saliency is generally higher - thus, in-turn, making the feature selection more robust and object-centric.

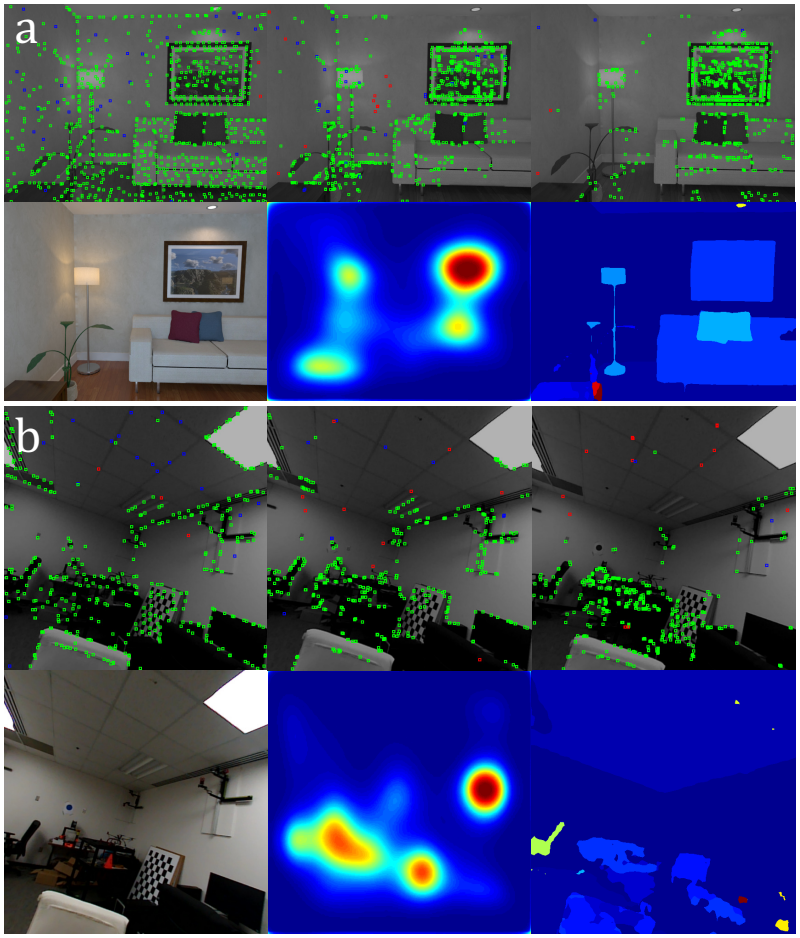


Figure E.6: Point selection using different schemes. Top rows in (a) and (b), left to right: features selected using DSO's scheme, saliency only, saliency+scene parsing. Bottom rows in (a) and (b), left to right: input image, saliency, scene parsing output. Notice how using saliency+scene parsing removed all non-informative features. (a) and (b) show images from ICL-NUIM and CVL-UMD datasets respectively.

E.5 Experimental Results and Analysis

In this section, we comprehensively evaluate SalientDSO on various datasets.

- **ICL-NUIM dataset**[193]: This dataset provides two scenes and four different trajectories for each scene which are obtained by running Kintinuous on real image data and finally used in a synthetic framework for obtaining ground-truth.

- **TUM monoVO dataset**[194]: This dataset provides 50 sequences comprising over 100 minutes videos. It ranges from indoor corridors to wide outdoor scenes. In our experiments, we only evaluate all methods on indoor sequences $\{\text{sequence\_}(1-18, 26, 28, 35-38,$ Only the indoor sequences are chosen because the usage of saliency obtained by human gaze is meaningful for indoor cluttered scenes.

- **CVL-UMD dataset**: This dataset was collected by the authors of this paper is available at `prg.cs.umd.edu/SalientDSO.html`. The data was collected using a Parrot® SLAMDunk [70] sensor suite. The data from the left camera is used in the experiments.

Different parameters used for running the experiments are shown in Table. E.2. For ICL-NUIM dataset, photometric correction is not required. To comprehensively evaluate the proposed method, we run each sequence in both forward and backward direction ten times.

Also, note that the Neural Networks employed in this work are adapted directly from their respective papers [1, 192] and were *not fine-tuned* as the concept of scene parsing and saliency are generic. The SalGAN [1] was trained on Salicon dataset [195]. The Salicon dataset collects the saliency data as a probability of visual attention by aggregating mouse trajectories and contains 10000 training images, 5000 validation images and 500 testing images. The PSPNet [192] was trained on the ADE20K dataset [196]. The ADE20K dataset contains annotations of 150 object classes all annotated by a single annotator for consistency. It contains 20210 training images, 2000 validation images and 3000 testing images.

Table E.2: Parameter settings for different datasets.

	TUM	ICL-NUIM	CVL-UMD
Num of active keyframes N_f	7	7	7
Num of active points N_p	2000	2000	1200
Global gradient constant g_{th}	7	3	7
Patch size K	8	8	8
Photometric correction	Yes	Not required	Not available

E.5.1 Quantitative Evaluation

Fig. E.7 shows the absolute trajectory Root Mean Square Error (RMSE_{ate}) on ICL-NUIM dataset (each rectangle shows a different run). Using visual saliency driven features, SalientDSO performs better in accuracy as compared to DSO. We also report alignment error e_{align} on TUM monoVO dataset in Fig. E.8. We disable the semantic filtering when we evaluate the proposed method on the TUM monoVO dataset, since this dataset provides only grayscale images and outputs from PSPNet are inaccurate and noisy for grayscale images. In Tables E.3 and E.4, we compare our method to DSO and ORB-SLAM [157] on the ICL-NUIM and TUM monoVO datasets. We urge the readers to refer to [197, 194] for a detailed description of the error metrics used in Tables E.3 and E.4 respectively. DSO and ORB-SLAM [157] are the current state-of-the-art direct and feature-based monocular VO methods. The results for DSO and ORB-SLAM are taken from [71]. ORB-SLAM is a full-fledged SLAM framework with loop closure and global alignment, while DSO and SalientDSO are merely odometry frameworks. To make the comparison fair, loop-closure detection and re-localization have been turned off for ORM-SLAM. The missing values in the table represent tracking failures. We achieve similar or

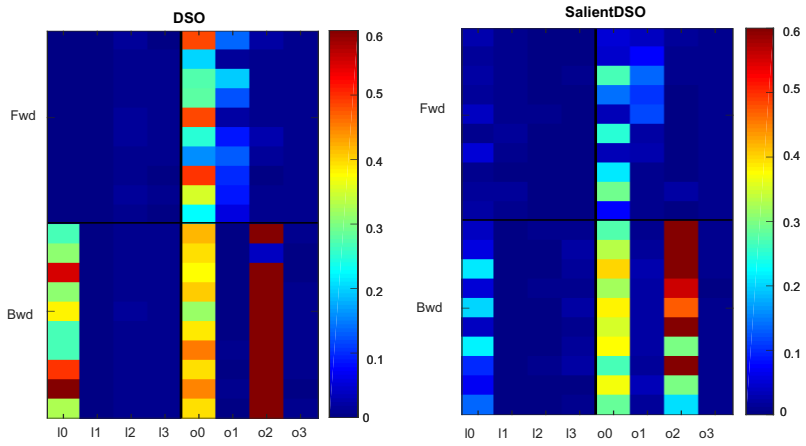


Figure E.7: Comparison of evaluation results for ICL-NIUM dataset. Left: DSO, Right: SalientDSO. Each square correspondes to a color coded error. Note that Salient DSO almost always has lower error than it's DSO counterpart.

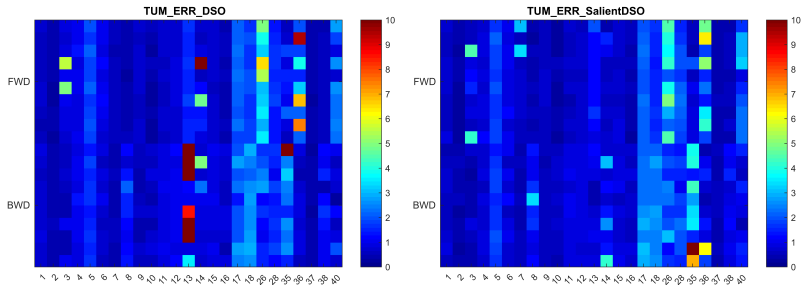


Figure E.8: Comparison of evaluation results for TUM dataset. Left: DSO, Right: SalientDSO. Note that Salient DSO almost always has lower error than it's DSO counterpart. Note that, for the TUM dataset scene parsing was turned off as TUM dataset only provides grayscale images and scene parsing outputs are very noisy for grayscale images.

better performance on most sequences. The improvement is not significant on the TUM monoVO dataset because most of the sequences involve a traversal through a hallway where there are no local salient objects or features for saliency prediction to work well. This makes SalientDSO's performance close to that of traditional DSO.

The claim in the paper is that the usage of visual saliency should result in more robust features than just using image gradient based features as in DSO. The intuition behind this claim is that visual saliency includes high level semantics which inherently make the features more robust. To support this claim, we anticipate that SalientDSO

Table E.3: RMSE_{ate} on ICL-NIUM dataset in m.

	Forward			Backward		
Sequence	ORB	DSO	SalientDSO	ORB	DSO	SalientDSO
ICL_l0	0.01	**0.003**	0.022	**0.01**	-	0.112
ICL_l1	0.02	**0.004**	0.009	0.04	0.003	**0.003**
ICL_l2	0.06	0.012	**0.004**	0.19	0.010	**0.005**
ICL_l3	0.03	0.006	**0.004**	0.05	**0.008**	0.013
ICL_o0	0.21	0.320	**0.140**	0.41	0.399	**0.336**
ICL_o1	0.83	0.094	**0.055**	0.68	**0.006**	0.020
ICL_o2	0.37	0.012	**0.008**	0.32	0.582	**0.512**
ICL_o3	0.65	**0.007**	0.009	0.06	**0.006**	0.008
Overall Avg.	0.271	0.057	**0.031**	0.218	0.144*	**0.126**

* indicates average taken only on sequences which completed.

Table E.4: e_{align} on TUM monoVO dataset in m.

	Forward			Backward		
Sequence	ORB	DSO	SalientDSO	ORB	DSO	SalientDSO
seq_01	3.02	**0.59**	0.60	1.73	0.72	**0.60**
seq_02	16.12	0.36	**0.33**	3.23	**0.43**	0.44
seq_03	3.42	1.75	**1.55**	1.42	0.59	**0.50**
seq_04	9.95	0.98	**0.82**	5.95	1.00	**0.76**
seq_05	-	1.86	**1.77**	-	**1.55**	1.66
seq_06	-	0.97	**0.93**	1.25	**0.73**	0.81
seq_07	1.69	**0.55**	1.14	2.02	**0.44**	0.48
seq_08	436.00	**0.36**	0.44	2.63	**1.28**	1.47
seq_09	2.04	0.65	**0.58**	0.67	**0.52**	0.53
seq_10	2.52	0.35	**0.34**	1.43	0.61	**0.61**
seq_11	7.20	0.62	**0.58**	2.99	**0.87**	0.89
seq_12	2.98	0.75	**0.67**	3.10	1.01	**0.84**
seq_13	5.13	1.54	**1.27**	2.59	8.96	**0.81**
seq_14	13.27	2.89	**0.71**	2.10	**1.35**	1.69
seq_15	2.90	0.71	**0.71**	1.90	0.88	**0.81**
seq_16	2.40	0.47	**0.45**	1.58	0.72	**0.67**
seq_17	12.29	2.10	**2.10**	**1.50**	2.13	2.50
seq_18	14.64	1.77	**1.52**	-	2.62	**2.47**
seq_26	28.46	3.98	**3.60**	4.62	**1.66**	1.89
seq_28	19.17	**1.48**	1.88	3.57	**1.47**	1.65
seq_35	14.09	1.10	**0.84**	16.81	**5.48**	9.97
seq_36	1.81	4.01	**3.25**	1.69	**0.70**	1.46
seq_37	0.60	**0.35**	0.40	1.30	**0.37**	0.46
seq_38	-	0.55	**0.50**	24.77	1.10	**1.03**
seq_40	-	**2.04**	2.16	18.93	**0.87**	1.04
Overall Avg.	28.55*	1.31	**1.17**	4.69*	1.52	**1.44**

* indicates average taken only on sequences which completed.

Table E.5: Comparison of success rate between DSO and SalientDSO on CVL-UMD dataset.

Sequence	DSO	SalientDSO
CVL_01_Fwd	53%	**65%**
CVL_01_Bwd	59%	**92%**
CVL_02_Fwd	73%	**96%**
CVL_02_Bwd	71%	**91%**

should perform much better than DSO when the number of points is very low (as low as 40 points). To demonstrate this claim, we evaluate on each CVL-UMD sequence. We run each sequence in both forward and backward direction 100 times, with an extremely low point density of $N_p = 40$. The results are shown in Table. E.5. We define failure as either an optimization failure or tracking loss. Our proposed method is much more robust and predicts an accurate trajectory, while DSO has a much higher failure rate and its trajectory and projected point cloud shows significant drift in scale and position. An example of trajectory and projected point cloud is shown in Fig. E.9. This experiment highlights the robustness of features chosen in SalientDSO for cluttered indoor scenes and how this will be useful for robots with very low computation power due to the less computational and memory requirements when N_p is low.

E.5.2 Qualitative Evaluation

Examples of the reconstructed scenes of sequences CVL_01 and TUM sequence_01 are shown in Figs. E.10 and E.11 respectively. Although both reconstructed scenes look similar, one could observe that amount of drift in SalientDSO is much less compared to DSO (refer to the zoomed part of Fig. E.10), i.e., the checkerboard's features from different loops align better in our approach – ideally the features from different loops should

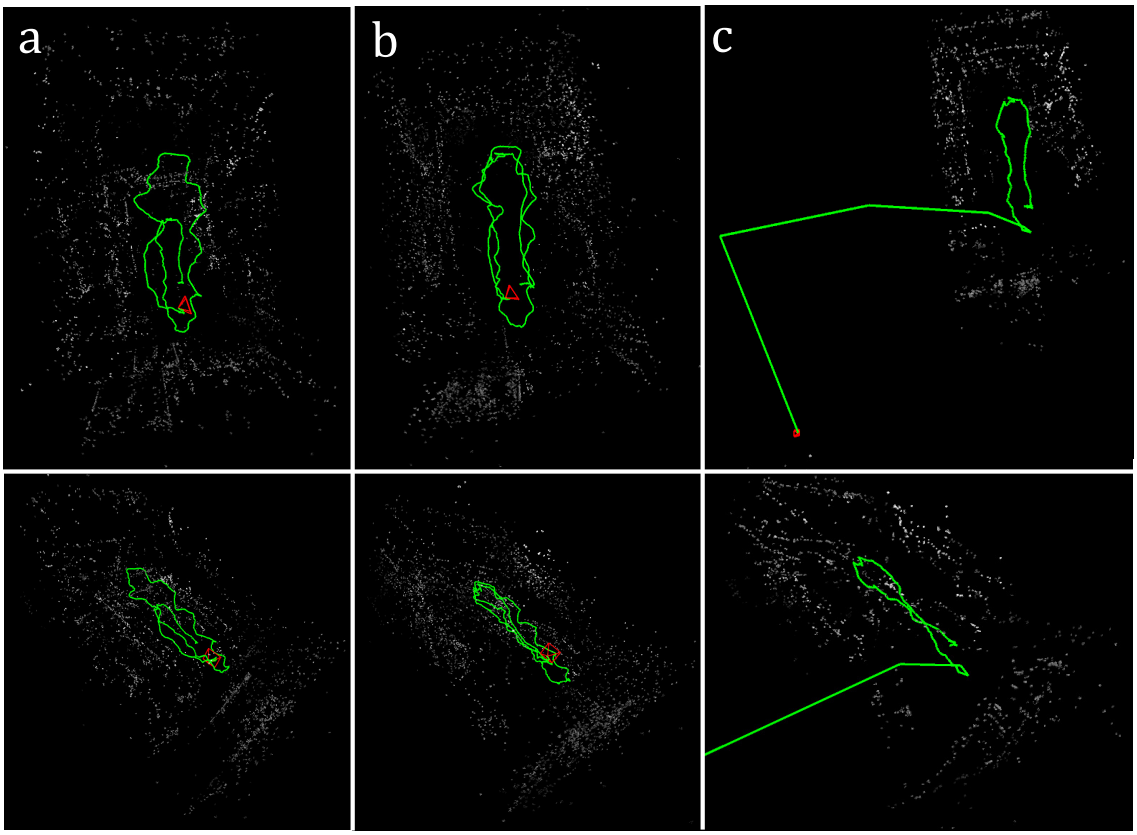


Figure E.9: Comparison of outputs for $N_p = 40$ – very few features. (a) Success case of DSO with a large amount of drift, (b) Success case for SalientDSO, (c) Failure case of DSO where the optimization diverges due to very few features. Notice that SalientDSO can perform very well in these extreme conditions showing the robustness of the features chosen.

align perfectly. However, due to drift the checkerboard appears as two different planes from two different loops, the spread between the planes of the checkerboard indicate drift. Notice that the spread between the planes of the checkerboard are much higher in DSO as compared to SalientDSO. Instead of sampling random high gradient points, sampling salient points which are considered to be more informative by the visual saliency model improves the robustness of VO.

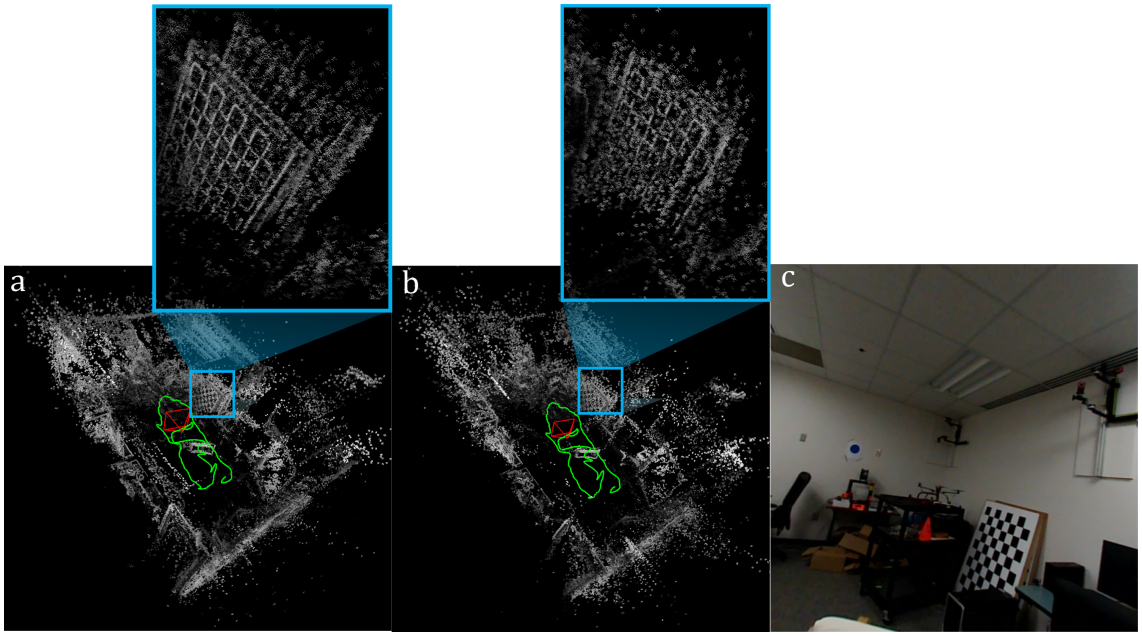


Figure E.10: Comparison of drift. (a) DSO's output, (b) SalientDSO's output, (c) Image corresponding to crop shown in the inset. Observe that SalientDSO's output has the checkerboard from different times more closely aligned as compared to DSO. Here $N_p = 1000$.

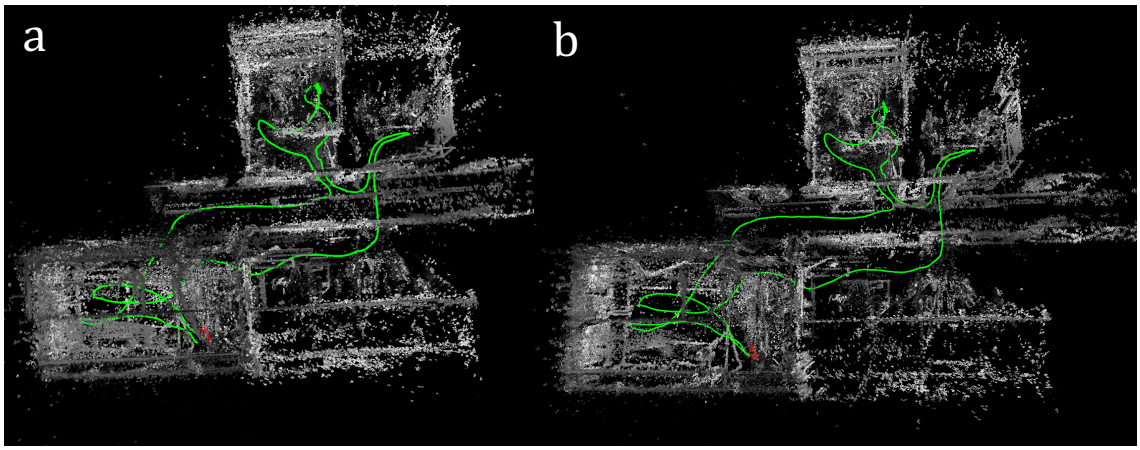


Figure E.11: Sample outputs for TUM sequence_1. (a) DSO, (b) SalientDSO. Here $N_p = 1000$.

E.6 Conclusions

We introduce the philosophy of attention and fixation to visual odometry. Based on this philosophy, we develop Salient Direct Sparse Odometry, which brings the concept

of attention and fixation based on visual saliency into Visual Odometry to achieve robust feature selection. We provide thorough quantitative and qualitative evaluations on ICL-NUIM and TUM monoVO dataset to demonstrate that using salient features improves the robustness and accuracy. We also collect and publicly release a new CVL-UMD dataset with cluttered scenes for mapping. We show the robustness of our features by very low drift visual odometry with as low as 40 features per frame. Our method takes about a second per keyframe (*not every frame*) for computation of saliency and scene parsing on an NVIDIA Titan-Xp GPU and the remaining computations run real-time at 30fps on an Intel® Core i7 6850K 3.6GHz CPU. In the near future, we plan to extend our method to outdoor environment. We also consider to implement our method on hardware to make the complete pipeline real-time. Finally, we also make our source code open-source for enabling future research.

Acknowledgement

This work was supported in part by ONR, NSF under grant awards N00014-17-1-2622 and 1824198 respectively, the Brin Family Foundation through a gift to the Perception and Robotics Group at the University of Maryland and the Northrop Grumman Corporation.

Appendix F: PRGFlow

Nitin J. Sanket, Chahat Deep Singh, Cornelia Fermüller, Yiannis Aloimonos "PRGFlow: Benchmarking SWAP-Aware Unified Deep Visual Inertial Odometry", *IET Electronics Letters*, (2021) (Accepted).

PRGFlow: Unified SWAP-Aware Deep Global Optical Flow for Aerial Robot Navigation

Nitin J. Sanket, Chahat Deep Singh, Cornelia Fermüller, Yiannis Aloimonos

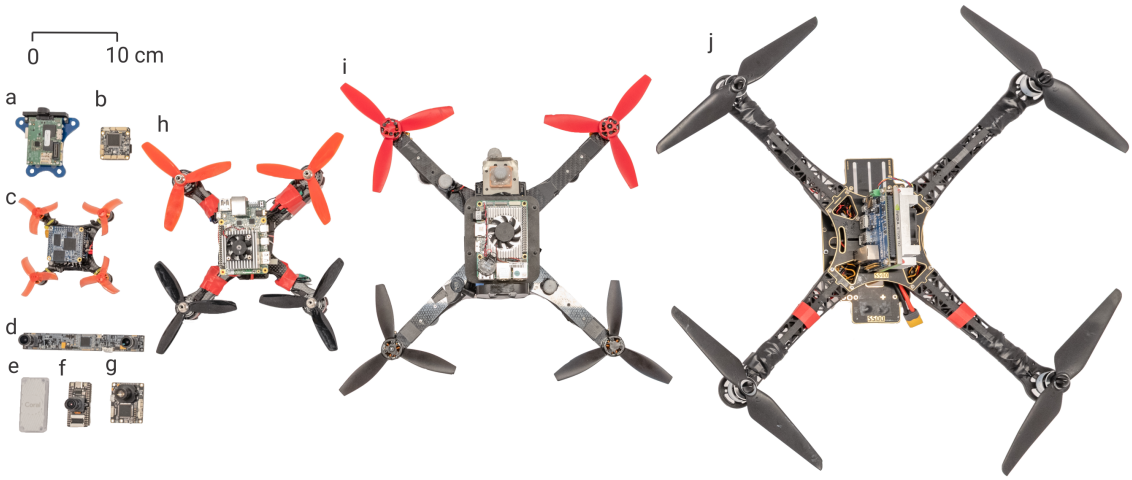


Figure F.1: Size comparison of various components used on quadrotors. (a) Snapdragon Flight, (b) PixFalcon, (c) 120 mm quadrotor platform with NanoPi Neo Core 2, (d) MYNT EYE stereo camera, (e) Google Coral USB accelerator, (f) Sipeed Maix Bit, (g) PX4Flow, (h) 210 mm quadrotor platform with Coral Dev board, (i) 360 mm quadrotor platform with Intel® Up board, (j) 500 mm quadrotor platform with NVIDIA® Jetson™ TX2. Note that all components shown are to relative scale. *All the images in this paper are best viewed in color.*

Abstract — Odometry on aerial robots has to be of low latency and high robustness whilst also respecting the Size, Weight, Area and Power (SWAP) constraints as demanded by the size of the robot. A combination of visual sensors coupled with Inertial Measurement Units (IMUs) has proven to be the best combination to obtain robust and low latency odometry on resource-constrained aerial robots. Recently, deep learning approaches for

Visual Inertial fusion have gained momentum due to their high accuracy and robustness. However, the remarkable advantages of these techniques are their inherent scalability (adaptation to different sized aerial robots) and unification (same method works on different sized aerial robots) by utilizing compression methods and hardware acceleration, which have been lacking from previous approaches.

To this end, we present a deep learning approach for visual translation estimation and loosely fuse it with an Inertial sensor for full 6DoF odometry estimation. We also present a detailed benchmark comparing different architectures, loss functions and compression methods to enable scalability. We evaluate our network on the MSCOCO dataset and evaluate the VI fusion on multiple real-flight trajectories.

F.1 Introduction

A fundamental competence of aerial robots [34] is to estimate ego-motion or odometry before any control strategy is employed [59, 61, 198, 126]. Different sensor combinations have been used previously to aid the odometry estimation with LIDAR based approaches topping the accuracy charts [199, 200]. However such approaches cannot be used on smaller aerial robots due to their size, weight and power constraints. Such small aerial robots are generally preferred due to safety, agility and usability as swarms [201, 202, 46]. In the last decade, imaging sensors have struck the right balance considering accuracy and general sensor utility [33]. However, visual data is dense and requires a lot of computation, which creates challenges for low-latency applications. To this end, sensor fusion experts proposed to use IMUs along with imaging sensors, because IMUs

are lightweight and are generally available on aerial robots [203]. Also, employing IMUs with even a monocular camera enables the estimation of metric depth which can be useful for many applications.

In the last decade, several VIO algorithms have been used in commercial products and also many algorithms have been made open-source by the research community. However, there is no trivial way of downscaling these algorithms for smaller aerial robots [35].

In the last five years, deep learning based approaches for visual and visual inertial odometry estimation have gained momentum. We classify as such algorithms all approaches which learn to predict odometry in an end-to-end fashion using one of the aforementioned sensors or which use deep learning as a part of the odometry estimation. The networks used in these approaches can be compressed to smaller size with generally a linear drop in accuracy to cater to SWAP constraints. The critical issue with deep networks for odometry estimation is that to have the same accuracy as classical approaches they are generally computationally heavy leading to larger latencies. However, leveraging hardware acceleration and better parallelizable architectures can mitigate this problem.

In this work, we present a method for visual inertial odometry estimation targeted towards a down-facing/up-facing camera coupled to an altimeter source such as a barometer (outdoor) or SONAR or single beam LIDAR (indoor). Our approach uses deep learning to obtain translation - shift and zoom-in/out and/or yaw. The inputs to the network are rotation compensated using Inertial estimates of attitude. Finally, we use the altimeter to scale the shifts to real-world velocities similar to [204]. We also benchmark different combinations of our approach and answer the following questions: *How many warping*

blocks to use? What network architecture to use? Which loss function to use? What is the best way to compress? Which common hardware is the best for a certain-sized aerial robot?

F.1.1 Problem Definition and Contributions

A quadrotor is equipped with a down/up-facing camera coupled to an altimeter and an IMU. The aim is to estimate ego-motion or odometry combining all sources of information. The presented approach has to be *scalable* and *unified* so that the same method can be used on different sized aerial robots catering to different SWAP constraints (Fig. F.1 shows examples of different sized quadrotors with different components which can be used on quadrotors).

A summary of our contributions is given below:

- A deep learning approach to estimate odometry using visual, inertial and altimeter data
- A comprehensive benchmark of different network architectures, hardware architectures and loss functions
- Real-flight experiments demonstrating robustness of the presented approach
- Notes to practitioners whenever applicable

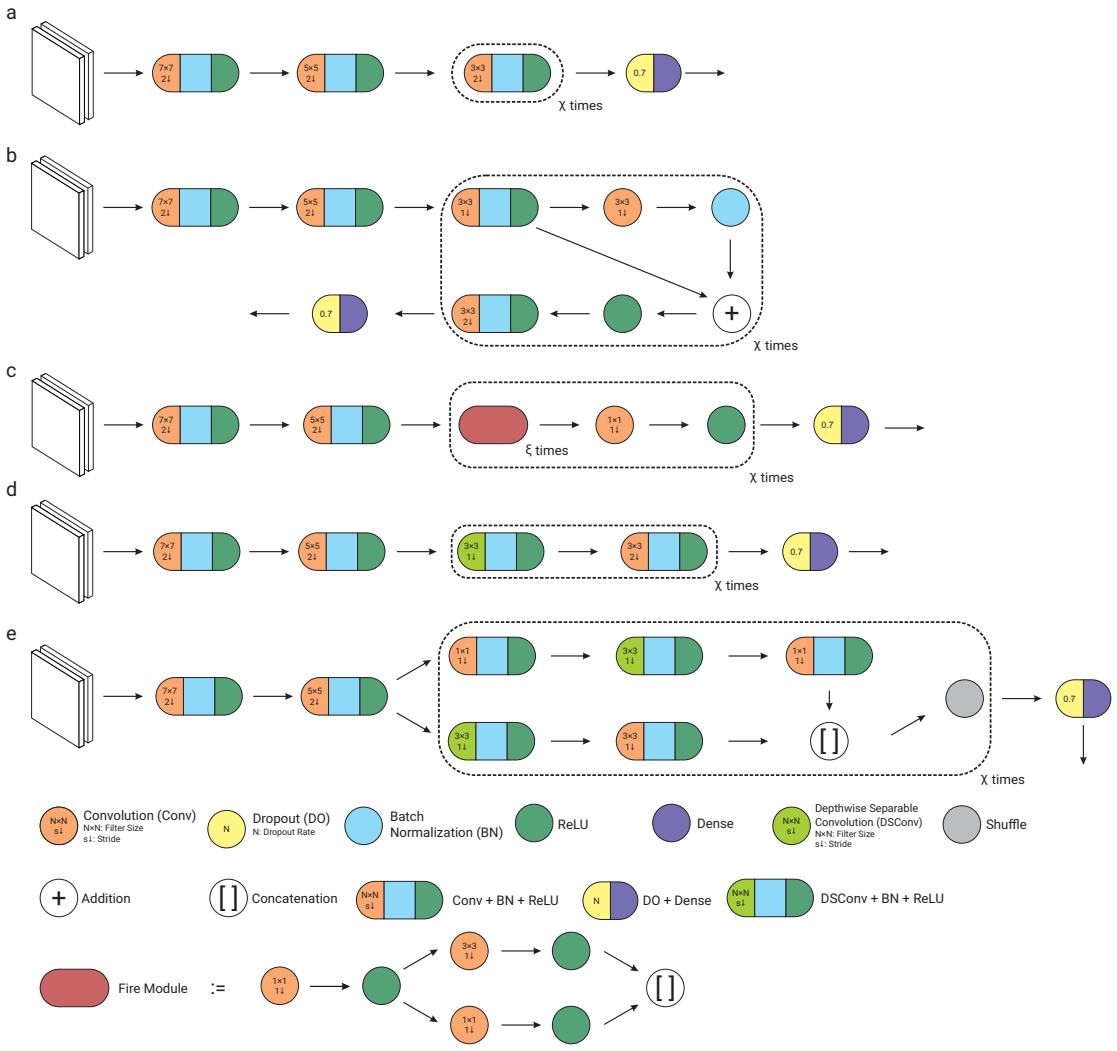


Figure F.2: Different network architectures. (a) VanillaNet, (b) ResNet, (c) SqueezeNet, (d) MobileNet and (e) ShuffleNet. (χ and ξ are hyperparameters). Each architecture block is repeated per warp parameter prediction. *This image is best viewed on the computer screen at a zoom of 200%.*

F.1.2 Related Work

There has been extensive progress in the field of Visual or Visual-Inertial Odometry (VI or VIO) using classical approaches, but adapting them to deep learning is still in a nascent stage. We categorize related work into the following three categories: VI/VIO using classical methods (non-deep learning), deep learning based VI/VIO, and deep learning

and odometry benchmarks. Also, note that we do not consider Simultaneous Localization And Mapping (SLAM) approaches [62] such as ORB-SLAM [157], LSD-SLAM [178], LOAM [199], V-LOAM [200] and probabilistic object centric slam [205]. We also exclude LIDAR and SONAR based odometry approaches from our discussion.

F.1.2.1 VI/VIO using classical methods

Following are the state-of-the-art approaches in chronological order.

- MSCKF [203] proposed an Extended Kalman Filter (EKF) for visual inertial odometry.
- OKVIS [206] proposed a stereo-keyframe based sliding window estimator to reduce landmark re-projection errors.
- ROVIO [64] also uses an EKF but included tracking 3D landmarks along with tracking of image patches.
- DSO [71] uses a direct approach using a photometric model coupled with a geometric model to achieve the best compromise of speed and accuracy.
- VINS-Mono [207] introduced a non-linear optimization based sliding window estimator with pre-integrated IMU factors.
- Salient-DSO [100] builds upon DSO to add visual saliency using deep learning for feature extraction. However, the optimization or regression of poses is performed classically.

F.1.2.2 Deep learning based VI/VIO

- PoseNet [208] uses a deep network to re-localize a camera in a pre-trained scene which brought the robustness and ease of use of deep networks for camera pose regression into limelight. Better loss functions for the same function were presented in [209].

- SfMLearner [210] took this one step further to regress camera poses and depth simultaneously from a sequence of video frames in a completely self-supervised (unsupervised) manner using geometric constraints.

- GeoNet [211] built upon SfMLearner to add additional geometric constraints and proposed a new training method along with a novel network architecture to predict pose, depth and optical flow in a completely self-supervised (unsupervised) manner.

- D3VO [212] tightly incorporates the predicted depth, pose and uncertainty into a direct visual odometry method to boost both the front-end tracking as well as the back-end non-linear optimization.

- VINet [213] proposed a supervised method to estimate odometry from a CNN + LSTM combination using both visual and inertial data. This approach, however, does not present results about its generalizability to novel scenes.

- DeepVIO [214] presents an approach to tightly integrate visual and inertial features using a CNN + LSTM to estimate odometry. This method also does not present results about its generalizability to novel scenes.

F.1.2.3 Deep learning and Odometry benchmarks

A myriad of datasets such as KITTI [215], EuRoC [216], TUMMonoVO [217], and PennCOSYVIO [218] have been proposed to evaluate the performance of VI/VIO approaches, but they do not contain enough images to train a neural network to generalize to novel datasets.

Though there exist several benchmarks for either classical VO/VIO approaches [219] and for deep learning for classification/regression tasks [220, 221, 222], there is a big void in benchmarks for deep learning based VO/VIO approaches, which is the focus of this work.

F.2 Pseudo-Similarity Estimation Using PRGFlow

Let us mathematically formulate our problem statement. Let $\mathcal{I}_t$ and $\mathcal{I}_{t+1}$ be the image frames captured at times t and $t+1$, respectively. Now, the transformation between the image frames can be expressed as $\mathbf{x}_{t+1} = \mathbf{H}_t^{t+1}\mathbf{x}_t$, where $\mathbf{x}_{t+1}, \mathbf{x}_t$ represent the homogeneous point correspondences in the two image frames and $\mathbf{H}_t^{t+1}$ is the non-singular 3×3 transformation matrix between the two frames.

While in general the transformation between views is a non-linear function of the 3D rotation matrix $\mathbf{R}_t^{t+1}$ and 3D translation vector $\mathbf{T}_t^{t+1}$, for certain scene structures, the transformation can simplify to a linear function. One such case is when the real world area is planar or can be approximated to be a plane, or the focal length is large. This scenario is also called a homography with $\mathbf{H}_t^{t+1}$ referred to as the *Homography matrix*.

From $\mathbf{H}_t^{t+1}$ we can recover a finite number of $\{\mathbf{R}_t^{t+1}, \mathbf{T}_t^{t+1}\}$ solutions. Furthermore,

$\mathbf{H}_t^{t+1}$ may also be decomposed into simpler transformations such as in-plane rotation or yaw, zoom-in/out or scale, translation and out-of-plane rotations or pitch + roll. It is difficult to accurately derive $\{\mathbf{R}_t^{t+1} and \mathbf{T}_t^{t+1}\}$ from $\mathbf{H}_t^{t+1}$, and the errors in the solutions of the two components are coupled, i.e., an error in the translation estimate would induce a complementary error in the rotation estimate; this is highly undesirable.

Complementary sensors help mitigate this problem, and IMU is such a sensor which is present on quadrotors and can provide accurate angle measurements within a small interval. Thus, the problem of estimating cheap ego-motion reduces to finding $\mathbf{T}_t^{t+1}$ (2D translation and zoom-in/out). Further, one can also obtain zoom-in/out using the altimeter on-board, but such an approach is noisy in a small interval. Hence we will estimate the 2D translation and zoom-in/out transformation which we refer to as "Pseudo-similarity" since it is one degree of freedom less than the similarity transformation (2D translation, zoom-in/out and yaw). We also call our network which estimates pseudo-similarity "PRGFlow". Note that, one can easily use our work to also estimate yaw without any added effort by changing the warping function. Mathematically, the pseudo-similarity transformation is given in Eq. F.1, where W and H depict the image width and height, respectively.

$$\mathbf{x}_{t+1} = \begin{bmatrix} \frac{W}{2} & 0 & \frac{W}{2} \\ 0 & \frac{H}{2} & \frac{H}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1+s & 0 & t_x \\ 0 & 1+s & t_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{x}_t \tag{F.1}$$

We utilize the Inverse Comompositional Spatial Transformer Networks (IC-STN) [223] for stacking multiple warping blocks for better performance. We extend the work in [223] to support pseduo-similarity and different warp types. A detailed study of which

warp type performs the best is given in Sec. F.5.1. We further divide the experiments into table-top experiments (Sec. F.3) and flight experiments (Sec. F.4).

F.3 Table-top Experiments and Evaluation

F.3.1 Data Setup, Training and Testing Details

We train and test all our networks on the MS-COCO dataset [148], using the `train2014` and `test2014` splits for training and testing. During training, we obtain a random crop of size 300 $\times$ 300 px. (denoted as $\mathcal{I}_t$ or $\mathcal{I}_1$), which is then warped using pseudo-similarity (2D translation + scale) to synthetically generate $\mathcal{I}_{t+1}$ or $\mathcal{I}_2$ obtained by using a random warp parameter in the range $\gamma_1 = \pm \begin{bmatrix} 0.25 & 0.20 & 0.20 \end{bmatrix}$ (unless specified otherwise). Then the center 128 $\times$ 128 px. patch are extracted (to avoid boundary effects) which are denoted as $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. A stack of $\mathcal{P}_1$ and $\mathcal{P}_2$ patches of size 128 $\times$ 128 $\times$ $2N_c$ (N_c is the number of channels in $\mathcal{P}_1$ and $\mathcal{P}_2$, i.e., 3 if RGB 1 if grayscale) is fed into the network to obtain the predicted warp parameters: $\widetilde{\mathbf{h}} = \begin{bmatrix} s & t_x & t_y \end{bmatrix}^T$.

Our networks were trained in Python™ 2.7 using TensorFlow[1] 1.14 on a Desktop computer (described in Sec. F.3.5) running Ubuntu 18.04. We used a mini-batch-size of 32 for all the networks with a learning rate of 10^{-4} without any decay. We trained all our networks using the ADAM optimizer for 100 epochs with early termination if we detect over-fitting on our validation set.

We tested our trained networks using two different configurations: (i) In-domain and (ii) Out-of-domain. In the in-domain testing (not to be confused with in-training

[1] https://www.tensorflow.org/

dataset), we warped images from the `test2014` split of the MS-COCO dataset using a random warp parameter in range $\gamma_1 = \pm \begin{bmatrix} 0.25 & 0.20 & 0.20 \end{bmatrix}$ (same warp range as training). This was used for evaluation as described in Sec. F.3.4. To commend on the generalization of the approach, we also tested it on out-of-domain warps, i.e., twice the warp range it was originally trained on, denoted by $\gamma_2 = \pm \begin{bmatrix} 0.50 & 0.40 & 0.40 \end{bmatrix}$. This test was constructed to highlight the generalizability vs. speciality of the network configuration. Next we discuss the loss network architectures.

F.3.2 Network Architectures

We use five network architectures inspired by previous works. We call the networks as follows: VanillaNet [224], ResNet [225], ShuffleNet [226], MobileNet [227] and SqueezeNet [228]. Each network is composed of various blocks. The output of each block is used as an incremental warp as proposed in [223]. We use for each of these blocks the same name as the network, for e.g., VanillaNet has VanillaNet blocks. We use the following shorthand to denote the architecture. VanillaNet$_a$ denotes VanillaNet with a VanillaNet blocks. We modify the networks from their original papers in the following manner: We exclude max-pooling blocks and replace them with stride in the previous convolutional block. Instead of varying sub-sampling rates (rates at which strides change with respect to depth of the network), we keep it fixed to the same value at every layer. All the architectures used are illustrated in Fig. F.2. The architectures shown in Fig. F.2 are repeated for every warping block used, where each block predicts the incremental warp as proposed in [223]. We exclude the number of filters for the purpose of clarity, but they

Figure F.3: PRG Husky-360γ platform used in flight experiments. (a) Top view, (b) front view, (c) down-facing leopard imaging camera.

can be found in the code provided with the supplementary material which will be released upon the acceptance for publication.

We also test two different sizes of networks, i.e., large (model size $\leq$ 8.3 MB) and small (Model size $\leq$ 0.83 MB). Here the model sizes are computed for storing `float32` values for each neuron weight. Due to file format packing there is generally a small overhead and the actual file sizes are slightly larger.

F.3.3 Loss Functions

The trivial way to learn the warp parameters directly is to use a supervised loss, since the labels are "free" as the data is synthetically generated on the fly. The loss function used in the supervised case is given as

$$\mathcal{L}_s = \underset{\widetilde{\mathbf{h}}}{\operatorname{argmin}} \ \mathbb{E}\left(\|\widetilde{\mathbf{h}} - \hat{\mathbf{h}}\|_2\right), \tag{F.2}$$

where $\widetilde{\mathbf{h}}$, $\hat{\mathbf{h}}$, $\mathbb{E}$ are the predicted parameters, ideal parameters and expectation/averaging operator respectively. We also study the question "Does using self-supervised loss give better out-of-domain generalization performance?" The unsupervised losses are gener-

ally complicated since they are under-constrained compared to the one in supervised approaches. Hence, we study different unsupervised loss functions. We can write the loss functions as:

$$\mathcal{L}_{us} = \underset{\widetilde{\mathbf{h}}}{\operatorname{argmin}} \ \mathbb{E}\left(\mathcal{D}\left(\mathcal{W}\left(\mathcal{P}_1, \widetilde{\mathbf{h}}\right), \mathcal{P}_2\right) + \lambda_i \mathcal{R}_i\right), \tag{F.3}$$

where $\mathcal{W}$ is a generic differentiable warp function, which can take on different mathematical formulations based on it's second argument (model parameters), and $\mathcal{D}$ represents a distance measuring image similarity between the image frames. Finally, $\lambda_i,\ \mathcal{R}_i$ represent the i^{th} Lagrange multiplier and it's corresponding regularization function. We experiment with different $\mathcal{D}$ and $\mathcal{R}$ functions described below.

$$\mathcal{D}_{\text{L1}}(A, B) = \mathbb{E}\left(\|A - B\|_1\right) \tag{F.4}$$

$$\mathcal{D}_{\text{Chab}}(A, B) = \left(\left((A - B)^2 + \epsilon^2\right)^{\alpha}\right) \tag{F.5}$$

$$\mathcal{D}_{\text{SSIM}}(A, B) = \mathbb{E}\left(\frac{1 - \textbf{SSIM}(A, B)}{2} + \alpha\left(\|A - B\|_1\right)\right) \tag{F.6}$$

$$\mathcal{D}_{\text{Robust}}(A, B) = \mathbb{E}\left(\frac{b}{d}\left(\left(\frac{\left({}^{(A-B)}/_{c}\right)^2}{b} + 1\right)^{d/2} - 1\right)\right) \tag{F.7}$$

$$b = \|2 - \hat{\alpha}\|_1 + \epsilon; \qquad d = \begin{cases} \hat{\alpha} + \epsilon & \text{if } \hat{\alpha} \geq 0 \\ \hat{\alpha} - \epsilon & \text{if } \hat{\alpha} < 0 \end{cases}$$

$$\hat{\alpha}_i = (2 - 2\epsilon_\alpha)\frac{e^{\alpha_i}}{e^{\alpha_i} + 1} \quad \forall i$$

Here, $\mathcal{D}_{\text{L1}}$ is the generic l_1 photometric loss [229] commonly used for traditional images, $\mathcal{D}_{\text{Chab}}$ is the Chabonnier loss [230] commonly used for optical flow estimation, $\mathcal{D}_{\text{SSIM}}$ is the loss based on Structure Similarity [231] commonly used for learning ego-motion and depth from traditional image sequences and $\mathcal{D}_{\text{Robust}}$ is the robust loss function presented in [232]. Also, note that α has different connotation in each loss function.

The functions given above can take any generic input such as the raw image or a function of the image. In our paper we experiment with different inputs such as the raw RGB image, grayscale image, high-pass filtered image and image cornerness score (denoted by $\mathcal{I}$, $\mathcal{G}$, $\mathcal{Z}(\mathcal{I})$ and $\mathcal{C}(\mathcal{I})$ respectively). The same set of functions can be used both as the metric function and the regularization function. We will denote the combination using a shorthand representation. Consider using the loss function with SSIM on raw images as the metric function and photometric L1 on high-pass filtered images as the regularization function with a Lagrange multiplier of 5.0, the shorthand for this function is given in Eq. F.8.

$$\mathcal{D}_{\text{SSIM}}\left(\mathcal{I}\right) + 5.0\mathcal{D}_{\text{L1}}\left(\mathcal{Z}(\mathcal{I})\right) \tag{F.8}$$

F.3.4 Evaluation Metrics

We use the following evaluation metrics to quantify the performance of each network. Let the predicted warp parameters be $\widetilde{\mathbf{h}} = \begin{bmatrix} \widetilde{s} & \widetilde{t_x} & \widetilde{t_y} \end{bmatrix}^T$ and the ideal warp parameters be $\hat{\mathbf{h}} = \begin{bmatrix} \hat{s} & \hat{t_x} & \hat{t_y} \end{bmatrix}^T$. W and H denote the image width and height respectively. Then the scale and translation error in pixel are given as:

Table F.1: Different Computers Used on Aerial Robots.

Name	Cost (USD) ↓	Size ($l \times w \times h$ mm) ↓	Weight (g) ↓	CPU Arch.	CPU Clock Speed × Threads (GHz) ↑	RAM (GB) ↑	GPU (GOPs) ↑	Max. Power* (W) ↓	Ease of use ↑
NanoPi Neo Core 2 LTS[2]	28	**40 × 40 × 3**	**7**	ARM	1.36 × 4	1.0	-	**10**	★★★
BananaPi M2-Zero[3]	**24**	65 × 30 × 5	15	ARM	1.00 × 4	1.0	-	**10**	★★★
Google Coral Dev Board[4]	150	88 × 60 × 22	136	ARM	1.50 × 4	1.0	4000	**10**	★★★½
Google Coral Accelerator[5]	75	65 × 30 × 8	20	-	-	-	4000	4.5	★★★★
NVIDIA® Jetson™ TX2[6]	600	87 × 50 × 48	200	ARM	2.00 × 6	8.0	1500	15	★★★
Intel® Up Board[7]	159	86 × 56 × 20	98	x86	1.92 × 4	1.0	-	**10**	★★★★½
Laptop[8]	1600	391 × 267 × 31	2200	x86	3.40 × 8	16.0	6463	180	★★★★★

*Power consumption is for board not for chip.

$$\mathcal{E}_{\text{scale}} = \mathbb{M}\left(\sqrt{\frac{W^2 + H^2}{2}}|\widetilde{s} - \hat{s}|\right) \tag{F.9}$$

$$\mathcal{E}_{\text{trans}} = \mathbb{M}\left(\frac{\sqrt{\left(W\left(\widetilde{t_x} - \hat{t_x}\right)\right)^2 + \left(H\left(\widetilde{t_y} - \hat{t_y}\right)\right)^2}}{2}\right) \tag{F.10}$$

Here, $\mathbb{M}$ denotes the median value (we choose the median value over the mean to reject outlier samples with low texture).

We also convert errors to accuracy percentage as follows:

$$\mathcal{A} = \left(1 - \frac{\mathcal{E}_{\text{scale}} + \mathcal{E}_{\text{trans}}}{\mathbb{I}_{\text{scale}} + \mathbb{I}_{\text{scale}}}\right) \times 100\% \tag{F.11}$$

Here, $\mathbb{I}_{\text{scale}}$ and $\mathbb{I}_{\text{trans}}$ denote the identity errors for scale and translation respectively (error when the prediction values are zero).

[2] http://nanopi.io/nanopi-neo-core2.html
[3] http://www.banana-pi.org/m2z.html
[4] https://coral.ai/products/dev-board/
[5] https://coral.ai/products/accelerator
[6] https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/
[7] https://up-board.org/
[8] https://www.asus.com/us/ROG-Republic-Of-Gamers/ROG-GL502VS/
[9] https://www.ibuypower.com/

F.3.5 Hardware Platforms

We tested multiple small factor computers and Single Board Computers (SBCs) (we'll refer to both as computers or computing devices or hardware) which are commonly used on aerial platforms. The platforms tested are summarized in Table F.1. Details not present in the table are explained next.

We overclocked the NanoPi Neo Core 2 LTS to 1.368 GHz from the base clock of 1.08 GHz to obtain better performance. This necessitated the use of an active cooling solution for long duration operation (more than 3 mins continuously). The weight of the cooling solution (4 g) and micro-SD card (0.5 g) are not included in Table F.1. Without the active cooling solution the computer goes into a thermal shutdown which will be harmful during in-flight operation. One can change the clock speed to certain available frequencies between 0.48 GHz and 1.368 GHz. To run larger models on the NanoPi's measly 1 GB of RAM we allocated 1 GB of SWAP memory on the Sandisk UHC-II class-10 microSD card which has much lower transfer speed as compared to RAM. The same cooling and SWAP solution were used for BananaPi M2-Zero as well. Note that, the Google Coral USB Accelerator is not officially supported with the NanoPi but we discovered a workaround which will be released in the accompanying supplementary material. Also, the Google Coral USB Accelerator is attached to a USB 2.0 port which limits its maximum performance when used with the NanoPi. The only reason why one would use BananaPi over the NanoPi is for the smaller width (30 mm versus 40 mm) which could be suited for a non X shaped quadrotor. However, the NanoPi is lighter, faster and has less area as compared to the BananaPi. Both NanoPi and BananaPi ran Ubuntu 16.04 LTS

core with TensorFlow 1.14.

A significant speedup of upto 576× were obtained on the Coral Dev and the Coral USB accelerator when the original TensorFlow model was converted into TensorFlow-Lite and optimized for Edge TPU compilation. Without the Edge-TPU optimization the models run on the CPUs of these computers are far slower than the Tensor cores. To use both Coral Dev and Coral USB accelerator TensorFlow-Lite-Runtime 2.1 is used. The Coral Dev board ran Mendel Linux 1.5.

The NVIDIA® Jetson™ TX2 in our setup is used with the Connect Tech's Orbitty carrier board (weighing 41 g and its weight is included in Table F.1). This carrier board allows for the most compact setup with the TX2. Note that the NVIDIA® Jetson™ TX2 can be used without the active heatsink for a short duration (less than 5 mins) reducing its weight by 59 g (a massive 30% reduction in weight without any loss of performance). However, extended operation without the active heatsink can result in thermal shutdown or permanent damage to the computer. During our experiments, we set the operation mode to fix the CPU and GPU frequencies to the maximum available value, and this in-turn maxes out the power consumption (the steps to achieve this will be released in the accompanying supplementary material). The TX2 ran Linux For Tegra (L4T) R28.2.1 installed using Jetpack 3.4 with TensorFlow 1.11.

We use neither the AVX (not supported on hardware) nor the SSE (not supported by the TensorFlow version supported on Up board) instruction set on the Intel® Up board. We speculate huge speed-ups if a future version of TensorFlow supports SSE on the Up board. The Up board ran Ubuntu 16.04 LTS with TensorFlow 1.11.

The laptop is an Asus ROG GL502VS with Intel® Core™ i7-6700HQ and GTX

1070 GPU and weighs about 2200 g which can be used on a large ($\geq$ 650 mm sized) quadrotor. Also, note that an Intel NUC coupled to an eGPU can also be used with similar specifications but this setup will still be heavier, arguably more expensive and probably less reliable than gutting out a gaming laptop. The laptop ran Ubuntu 18.04 LTS with TensorFlow 1.14.

The desktop PC is a custom built full tower PC from iBUYPOWER with an Intel® Core™ i9-9900KF and NVIDIA® Titan-Xp GPU which cannot be used on an aerial robot but is included to serve as a reference. The desktop PC ran Ubuntu 18.04 LTS with TensorFlow 1.14. Whenever possible we use the NEON SIMD instruction set for ARM computers and the SSE instructions for x86 systems.

F.4 Flight Experiments and Evaluation

This section presents real-world experiments flying various simple trajectories with odometry estimation using PRGFlow. We use the lowest avg. pixel error 8.3 MB (large) model for evaluation (ResNet_4 with T$\times$2, S$\times$2 warping configuration). The network outputs a $\mathbb{R}^{3\times1}$ vector denoted as $\widetilde{\mathbf{h}} = \begin{bmatrix} \widetilde{s} & \widetilde{t_x} & \widetilde{t_y} \end{bmatrix}^T$. The predicted translational pixel velocities (global optical flow) $\begin{bmatrix} \widetilde{t_x} & \widetilde{t_y} \end{bmatrix}^T$ are converted to real world velocities by scaling them using the focal length and the depth (adjusted altitude) [204]. A similar treatment is given to the Z-pixel velocity s. Also, we obtain attitude using a Madgwick filter [233] from a 9-DoF IMU, which is used to remove rotation between consecutive frames, and feed the values into the network. Finally, to obtain odometry, we simply perform dead-reckoning on the velocities obtained by our network. *Our networks were trained on MS-*

COCO as described in F.3.1, and are used in the flight experiments without any fine-tuning or re-training to highlight the generalizability of our approach. Note that the performance can be significantly boosted by multi-frame fusion of predictions, and we leave this for future work.

F.4.1 Experimental Setup

We tested our algorithm on the PRG Husky-360γ platform[9]: a modified version of the Parrot® Bebop 2 for its ease of use which was originally designed for pedagogical reasons. It is equipped with a down facing Leopard Imaging LI-USB30-M021 global shutter camera[10] with a 16 mm lens which gives a diagonal field of view of $\sim 22^\circ$ (Refer to Fig. F.3). The PRG Husky also contains an 9-DoF IMU and a down-facing sonar for attitude and altitude measurements respectively. Higher level control commands are given by an on-board companion computer: Intel® Up Board at 20 Hz. The images are recorded at 90 Hz at a resolution of 640×480 px. The overall takeoff weight of the flight setup is 730 g (which gives a thrust-to-weight ratio of ~ 1.5) with diagonal motor to motor dimension of 360 mm.

The flight experiments were conducted in the Autonomy Robotics and Cognition (ARC) lab's netted indoor flying space at the University of Maryland, College Park. The total flying volume is about $6 \times 5.5 \times 3.5$ m^3. A Vicon motion capture system with 12 vantage V8 cameras were used to obtain ground truth at 100 Hz.

The PRG Husky was tested on five trajectories: `circle`, `moon`, `line`, `figure8`

[9] https://github.com/prgumd/PRGFlyt/wiki/PRGHusky

[10] https://leopardimaging.com/product/usb30-cameras/usb30-camera-modules/li-usb30-m021/

Table F.2: Quantitative evaluation of different warping combination for Pseudo-similarity estimation.

Network (Warping)	$\mathcal{E}_{\text{scale}}$ (px.) ↓		$\mathcal{E}_{\text{trans}}$ (px.) ↓		FLOPs (G) ↓	Num. Params (M) ↓
	γ_1	γ_2	γ_1	γ_2		
Identity	11.4	22.8	10.3	20.4	–	–
VanillaNet$_1$ (PS×1)	2.4	15.0	**1.3**	12.5	**0.37**	**2.07**
VanillaNet$_1$ (PS×1) DA	4.1	17.7	2.3	14.2	**0.37**	**2.07**
VanillaNet$_2$ (PS×2)	2.2	9.9	1.4	12.4	0.42	2.17
VanillaNet$_2$ (S×1, T×1)	2.5	15.2	1.5	12.2	0.46	2.10
VanillaNet$_2$ (T×1, S×1)	2.5	15.1	1.5	12.5	0.42	2.15
VanillaNet$_4$ (PS×4)	2.3	11.9	1.5	14.9	0.42	2.15
VanillaNet$_4$ (S×2, T×2)	2.6	15.4	1.6	12.6	0.46	2.08
VanillaNet$_4$ (T×2, S×2)	**2.0**	8.5	1.5	12.5	0.46	2.08
VanillaNet$_4$ (T×2, S×2) γ_2	2.7	**2.8**	4.6	**7.2**	0.46	2.08

Table F.3: Quantitative evaluation of different network architectures for Pseudo-similarity estimation using T×2, S×2 warping block for large model ($\leq$8.3 MB).

Network	$\mathcal{E}_{\text{scale}}$ (px.) ↓		$\mathcal{E}_{\text{trans}}$ (px.) ↓		FLOPs (G) ↓	Num. Params (M) ↓
	γ_1	γ_2	γ_1	γ_2		
Identity	11.4	22.8	10.3	20.4	–	–
VanillaNet$_4$	1.9	6.4	1.5	12.4	0.46	2.08
ResNet$_4$	**1.7**	15.1	**0.9**	**10.1**	0.59	2.12
SqueezeNet$_4$	2.1	**5.7**	2.2	13.8	2.20	2.12
MobileNet$_4$	4.0	14.2	1.6	12.0	**0.41**	**2.04**
ShuffleNet$_4$	6.4	17.4	3.0	13.9	1.20	2.10

Table F.4: Quantitative evaluation of different network architectures for Pseudo-similarity estimation using T×2, S×2 warping block for small model ($\leq$0.83 MB).

Network	$\mathcal{E}_{\text{scale}}$ (px.) ↓		$\mathcal{E}_{\text{trans}}$ (px.) ↓		FLOPs (G) ↓	Num. Params (M) ↓
	γ_1	γ_2	γ_1	γ_2		
Identity	11.4	22.8	10.3	20.4	–	–
VanillaNet$_4$	3.3	8.9	3.1	14.0	0.18	0.21
ResNet$_4$	4.4	12.5	**2.4**	**12.1**	0.20	**0.20**
SqueezeNet$_4$	**2.4**	**5.6**	4.0	14.9	0.19	**0.20**
MobileNet$_4$	8.3	18.7	3.7	13.4	0.16	**0.20**
ShuffleNet$_4$	8.3	17.6	4.6	15.7	**0.13**	0.21

and `square` which involve change in both attitude and altitude with an average velocity of about 0.5 ms^{-1} and a maximum velocity of 1.5 ms^{-1}.

F.5 Discussion

F.5.1 Algorithmic Design

We answer the following questions in this section:

Table F.5: Quantitative evaluation of different network inputs for Pseudo-similarity estimation using T×2, S×2 warping block for large model (≤8.3 MB).

Testing Data (Training Data)	$\mathcal{E}_{\text{scale}}$ (px.) ↓		$\mathcal{E}_{\text{trans}}$ (px.) ↓	
	γ_1	γ_2	γ_1	γ_2
Identity	11.4	22.8	10.3	20.4
$\mathcal{I}$ ($\mathcal{I}$)	2.0	8.5	**1.5**	12.5
$\mathcal{G}$ ($\mathcal{I}$)	**1.8**	**6.3**	**1.5**	**12.3**
$\mathcal{G}$ ($\mathcal{G}$)	2.7	14.1	1.6	12.7
$\mathcal{Z}(\mathcal{I})$ ($\mathcal{Z}(\mathcal{I})$)	13.1	9.4	10.4	16.0
$\mathcal{G}$ ($\mathcal{Z}(\mathcal{I})$)	11.8	20.7	9.8	19.8
$\mathcal{I}$ ($\mathcal{Z}(\mathcal{I})$)	13.1	22.5	10.5	20.1
$\mathcal{Z}(\mathcal{I})$ ($\mathcal{G}$)	8.5	19.8	4.1	17.6
$\mathcal{Z}(\mathcal{I})$ ($\mathcal{I}$)	17.2	20.1	4.2	17.4

Table F.6: Quantitative evaluation of different loss functions for Pseudo-similarity estimation using PS×1 warping block for large model (≤8.3 MB).

Loss Function (Architecture)	$\mathcal{E}_{\text{scale}}$ (px.) ↓		$\mathcal{E}_{\text{trans}}$ (px.) ↓		FLOPs (G) ↓	Num. Params (M) ↓
	γ_1	γ_2	γ_1	γ_2		
Identity	11.4	22.8	10.3	20.4	–	–
Supervised $\mathcal{L}_s$ (VanillaNet$_1$)	2.4	15.0	1.3	12.5	0.37	2.07
$\mathcal{D}_{\text{Robust}}$ ($\mathcal{I}, \mathcal{C}(\mathcal{I})$) (ResSqueezeNet$_1$)	12.9	25.2	7.2	**11.7**	**1.01**	**2.18**
$\mathcal{D}_{\text{SSIM}}$ ($\mathcal{I}$) (ResSqueezeNet$_1$)	3.4	21.2	6.0	13.8	**1.01**	**2.18**
$\mathcal{D}_{\text{SSIM}}$ ($\mathcal{I}$) + $0.1\mathcal{D}_{\text{L1}}$ ($\mathcal{C}(\mathcal{I})$) (ResSqueezeNet$_1$)	**2.0**	**16.1**	6.2	14.6	**1.01**	**2.18**
$\mathcal{D}_{\text{SSIM}}$ ($\mathcal{I}$) + $0.1\mathcal{D}_{\text{L1}}$ ($\mathcal{Z}(\mathcal{I})$) (ResSqueezeNet$_1$)	2.7	16.6	6.4	13.6	**1.01**	**2.18**
$\mathcal{D}_{\text{L1}}$ (DB ($\mathcal{E}$)) [35]	5.4	17.7	3.7	16.5	4.92	3.6
$\mathcal{D}_{\text{Chab}}$ (DB ($\mathcal{E}$)) [35]	5.1	17.1	3.4	16.7	4.92	3.6
Supervised DB ($\mathcal{E}$) [35]	4.1	16.2	**3.3**	15.1	4.92	3.6

Table F.7: Quantitative evaluation of different compression methods for Pseudo-similarity estimation using PS×1 warping.

Method	$\mathcal{E}_{\text{scale}}$ (px.) ↓		$\mathcal{E}_{\text{trans}}$ (px.) ↓		$\mathcal{E}_{\text{scale}}$ (DA) (px.) ↓		$\mathcal{E}_{\text{trans}}$ (DA) (px.) ↓	
	γ_1	γ_2	γ_1	γ_2	γ_1	γ_2	γ_1	γ_2
Identity	11.4	22.8	10.3	20.4	11.4	22.8	10.3	20.4
Teacher from Scratch	2.4	15.0	1.3	12.5	4.3	17.2	2.5	14.1
Student from Scratch	**3.5**	**9.9**	2.8	13.2	**4.2**	**16.8**	4.2	16.0
Projection Loss Student [234]	3.7	10.9	2.8	**13.1**	8.0	17.7	4.2	**15.2**
Model Distillation Student [235]	3.8	12.3	**2.7**	13.4	7.2	17.7	**4.0**	**15.2**

1. What is the best warp combination?

2. What network architecture is the best? Does the best network architecture vary with respect to the number of parameters?

3. How does the input affect performance of a network?

Table F.8: Comparison of PRGFlow with different classical methods.

Method	$\mathcal{E}_{\text{scale}}$ (px.) ↓	$\mathcal{E}_{\text{trans}}$ ↓ (px.)	$\mathcal{E}_{\text{scale}}$ (DA) (px.) ↓	$\mathcal{E}_{\text{trans}}$ (DA) (px.) ↓	Time (ms) ↓
Identity	11.4	10.3	11.4	10.3	–
Supervised $\mathcal{L}_s$	1.9	1.5	**4.1**	2.3	**1.1**[*]
FFT Aligment [236]	**0.3**	**0.1**	13.4	6.0	35.5
SURF [237]	0.4	**0.1**	11.2	**1.3**	17.6
ORB [238]	0.6	**0.1**	11.5	1.4	12.9
FAST [239]	0.8	0.2	12.0	**1.3**	55.9
Brisk [240]	0.7	0.2	13.0	1.4	38.7
Harris [241]	0.4	**0.1**	10.9	1.4	60.2

[*] On all cores.

4. What is the best way to compress a model?

5. When is it advisable to choose deep learning for odometry estimation?

The performance of different IC-STN warp combinations for VanillaNet are given in Table F.2. The in-domain and out-of-domain test results have a headings of γ_1 and γ_2 respectively. All the networks were trained using supervised l_2 loss (Eq. F.2). Note that, the networks were constrained to be $\leq$8.3 MB. Under this condition, one can clearly observe the following trend when only using PS blocks for warping – as the number of warping blocks increases the performance reaches a maximum and then deteriorates. This is because the number of neurons per warp block directly affects the performance, and increasing the number of warp blocks without increasing model size hurts accuracy when the number of neurons per warp block become small. An interesting observation is that predicting translation before scale almost always results in better performance and also decoupling the predictions of scale and translation using S and T blocks generally results in better performance (lower pixel error) as compared to predicting them together using PS blocks. This is serendipitous. As drift in the X and Y position are generally higher, one could obtain these positions faster (since only a part of the network is used

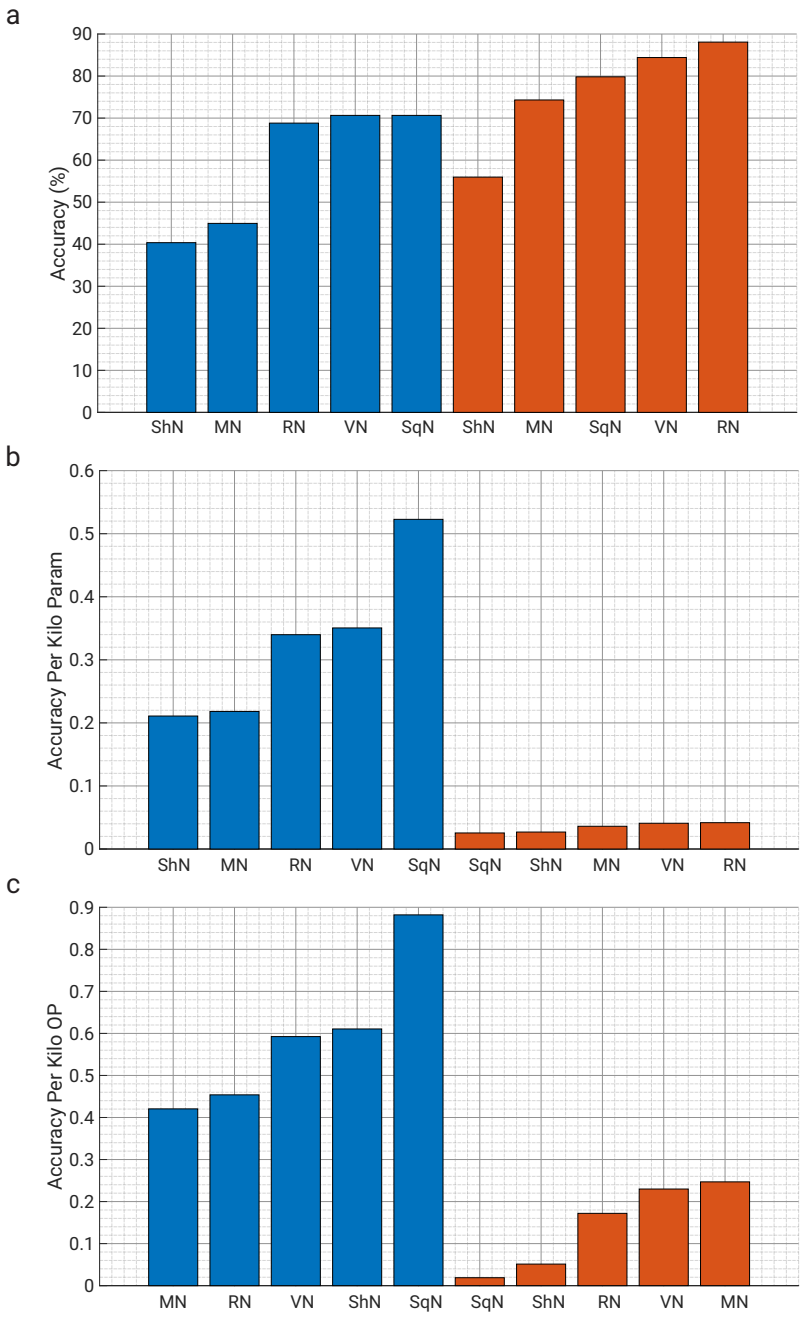


Figure F.4: (a) Accuracy, (b) Accuracy per Kilo param, (c) Accuracy per Kilo OP for different network architectures. Blue and orange histograms denote small ($\leq$0.83 MB) and large ($\leq$8.3 MB) networks respectively. Here the following shorthand is used for network names: VN: VanillaNet, RN: ResNet, SqN: SqueezeNet, MN: MobileNet and ShN: ShuffleNet. All networks use T$\times$2, S$\times$2 warping configuration.

for output) and at higher frequency than the Z position [242].

We also observe that training and testing with data augmentation of brightness, contrast, hue, saturation and Gaussian noise worsens the average performance by 73% (comparing rows 2 and 3 in Table F.2) since the network now has to learn to be agnostic to a myriad of image changes. Also, training on γ_2 (last two rows of Table F.2) decreases the average error by 62% when testing on γ_2 - we speculate it will further decrease with increasing model size. This shows that as the deviation range in training increases, one can require more parameters for the same average performance. *Overall the best performing warp configuration is T×2, S×2 when considering an average of both in-domain and out-of-domain tests.*

Next, let us study the performance with different network architectures. We use T×2, S×2 warps for all the networks, and they are trained using supervised l_2 loss (Eq. F.2). The results for networks constrained by model size $\leq$8.3 MB and $\leq$0.83 MB can be found in Tables F.3 and F.4 respectively. For ease of analysis, the results are also visually depicted in Fig. F.4. One can clearly observe that ResNet gives the best performance for both small and large networks (Fig. F.4a) and should be the network architecture of choice when designing for maximizing accuracy without any regard to the number of parameters or amount of OPs (operations). However, if one has to prioritize maximizing accuracy whilst minimizing number of parameters, then SqueezeNet and ResNet would be the choice for smaller and larger networks respectively (Fig. F.4b). Another trend one can observe is that one would need 10× more parameters for a 19% increase in accuracy. Lastly, if one has to prioritize in maximizing accuracy whilst minimizing the number of OPs, then SqueezeNet and MobileNet would be the choice for smaller and

larger networks respectively (Fig. F.4c). Clearly, the most optimal architecture in-terms of accuracy, number of parameters and OPs is SqueezeNet for smaller networks and ResNet for larger networks. *The effect of the choice of network architecture is fairly significant on the accuracy, number of parameters and OPs and needs to be carefully considered when designing a network for deployment on an aerial robot*. Also, note that sometimes using a classical approach to solve a small part of the problem can significantly simplify the learning problem for the network thereby maximizing accuracy and minimizing the number of parameters and OPs [34].

To gather more insight into what data representation is more important for odometry estimation, we explore training and testing on the following data representations: (a) RGB image, (b) Grayscale image, (c) High pass filtered image. We choose the T$\times$2, S$\times$2 warp configuration and the VanillaNet$_4$ architecture trained using supervised l_2 loss constrained by model size $\leq$8.3 MB. Table F.5 summarizes the results obtained. Surprisingly, training on RGB images and evaluating on RGB images gives worse performance in the testing both for in-domain (γ_1) and out-of-domain (γ_2) ranges than testing on grayscale data. We speculate that this is due to conflicting information in multiple channels. Another surprising observation is that training and/or testing on high-pass filtered images results in large errors, which is contrary to the classical approaches. We speculate that this is because conventional neural networks rely on "staticity" of the image pixels (image pixels change slowly and are generally smooth).

We also explore the state-of-the-art self-supervised/unsupervised loss functions to test for claims of better out-of-domain generalization. We choose half-sized ($\leq$4.15 MB) ResNet (to output scale) and SqueezeNet (to output translation) denoted as ResSqueezeNet

trained using PS$\times$1 blocks for this experiment since it empirically gave us the best results (we exclude other architecture results for the purpose of brevity). We also include results from a VanillaNet$_1$ trained using PS$\times$1 blocks using the supervised l_2 loss function to serve as a reference (Refer to Table F.6). We observe that scale error when using SSIM and cornerness (obtained using a heatmap as the output of [243]) in the loss approaches the performance of the supervised network, but the translation error is almost three times that of the supervised network. Surprisingly, the supervised network also performs better than most unsupervised networks on out-of-domain tests. *This hints that we need better loss functions for unsupervised methods and better network architectures to take advantage of these unsupervised losses.* From a practitioner's point of view, the supervised networks perform better and generalize better to out-of-domain. Another keen insight is that focusing on crafting better supervised loss functions may lead to a massive boost in performance.

Finally, we explore different strategies to compress the network. We specifically consider a setup where we compress a 8.3 MB model VanillaNet$_1$ PS$\times$1 to a 0.83 MB model VanillaNet$_1$ PS$\times$1. We test three different methods, (a) direct dropping of weights, (b) projection inspired loss and (c) model distillation. In the first method, we reduce the number of neurons and number of blocks to reduce the number of weights and train the smaller network from scratch. In the second method, we train both the larger and smaller networks together as given by Eq. F.12 [234]. Here, $\widetilde{h_T}, \widetilde{h_S}$ denote the predictions from the teacher and student network respectively. We choose $\lambda_1, \lambda_2, \lambda_3$ as 1.0, 1.0 and 0.1 respectively. In the third method, we use an already trained teacher network (large 8.3 MB model) and define the loss to learn the predictions of the teacher using the student

(small 0.83 MB model) as given by Eq. F.13 [235]. When no data augmentation is used, we observe that directly dropping weights gives the best performance (Refer to Table F.7). However, when the data augmentation is added the model distillation gives the best results, albeit only slightly better than directly dropping of weights. This observation is contrary to that observed with classification networks where massive boosts in performance are observed when using either Eq. F.12 or F.13. From a practitioner's point of view, the simplest method of directly dropping weights work the best for regression networks like the one used in this work and can provide up to $10\times$ savings in the number of parameters and OPs at the cost of $\sim$19% accuracy. The same effect is observed when training and testing with and without data augmentation.

$$\mathcal{L}_{\text{Proj}} = \lambda_1 \mathcal{L}_s \left(\hat{h}, \widetilde{h_T}\right) + \lambda_2 \mathcal{L}_s \left(\hat{h}, \widetilde{h_S}\right) + \lambda_3 \mathcal{L}_s \left(\widetilde{h_T}, \widetilde{h_S}\right) \tag{F.12}$$

$$\mathcal{L}_{\text{TS}} = \mathcal{L}_s \left(\widetilde{h_T}, \widetilde{h_S}\right) \tag{F.13}$$

To address the elephant in the room, we try to answer the following question: "When should one use deep learning over a traditional approach?" We compare the proposed deep learning approach PRGFlow to the common method of fast feature matching on aerial robots in Table F.8. Note that the runtime for traditional methods are given for MATLAB[11] implementations on one thread of the Desktop PC to standardize the libraries and optimizations used. Up to $5\times$ and $10\times$ speedup can be obtained using efficient C++ implementations on a multi-threaded CPU and GPU respectively (we don't explore this

[11] https://www.mathworks.com/products/matlab.html

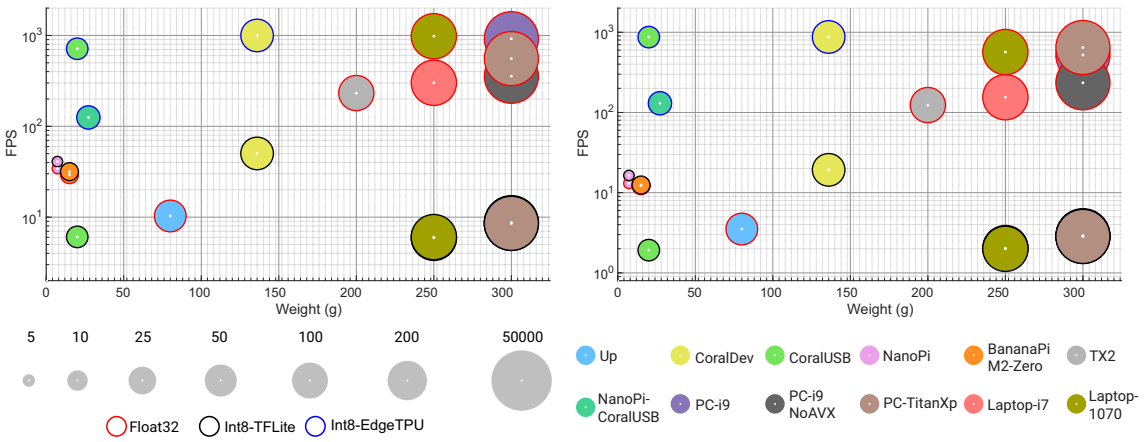


Figure F.5: Weight vs. FPS for VanillaNet$_4$ (T$\times$2, S$\times$2) on different hardware and software optimization combinations. Left: small ($\leq$0.83 MB) model, right: large ($\leq$8.3 MB) model. The radius of each circle is proportional to log of volume of each hardware (this is shown in the legend below the plots with volume indicated on top of each legend in cm^3). The outline on each sample indicates the configuration of quantization or optimization used (`Float32` (red outline) is the original TensorFlow model without any quantization or optimization, `Int8-TFLite` (black outline) is the TensorFlow-Lite model with 8-bit Integer quantization and `Int8-EdgeTPU` (blue outline) is the TensorFlow-Lite model with 8-bit Integer quantization and Edge-TPU optimization. The samples are color coded to indicate the computer it was run on (shown in the legend on the bottom). *Also note that, Laptop and PC (Deskop) weight and volume values are not to actual scale for visual clarity in all images. All the figures in this paper use the same legend and color coding for ease of readability.*

in our work). One can clearly observe that even with C++ implementations the traditional handcrafted features are slower than the deep learning methods which can utilize the parallel hardware accelerations on GPUs. Though on the surface it seems like the traditional methods give far superior performance in terms of accuracy (lower error), the efficacy of deep learning approaches are brought into limelight when we train and test with data augmentations. This simulates a bad quality camera common on smaller aerial robots. The drop in accuracy (from no-noise data to noisy data) in deep learning approaches is much less than that compared to traditional approaches, i.e., *deep learning approaches on an average fail more gracefully compared to traditional approaches.*

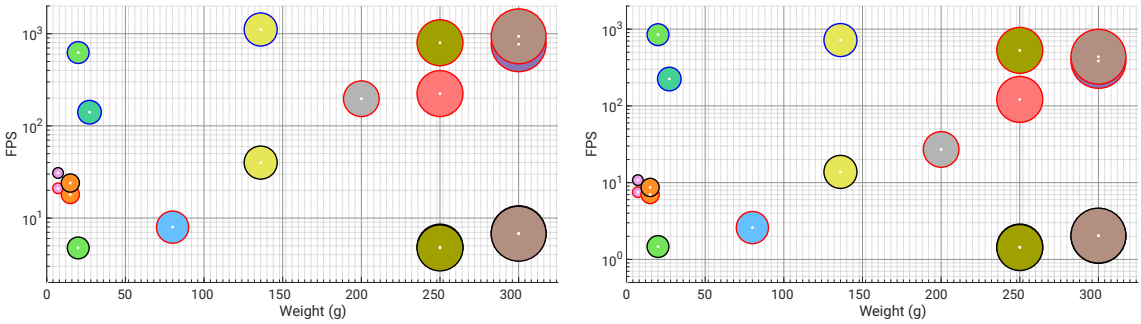


Figure F.6: Weight vs. FPS for ResNet$_4$ (T$\times$2, S$\times$2) on different hardware and software optimization combinations. Left: small ($\leq$0.83 MB) model, right: large ($\leq$8.3 MB) model. The radius of each circle is proportional to log of volume of each hardware.

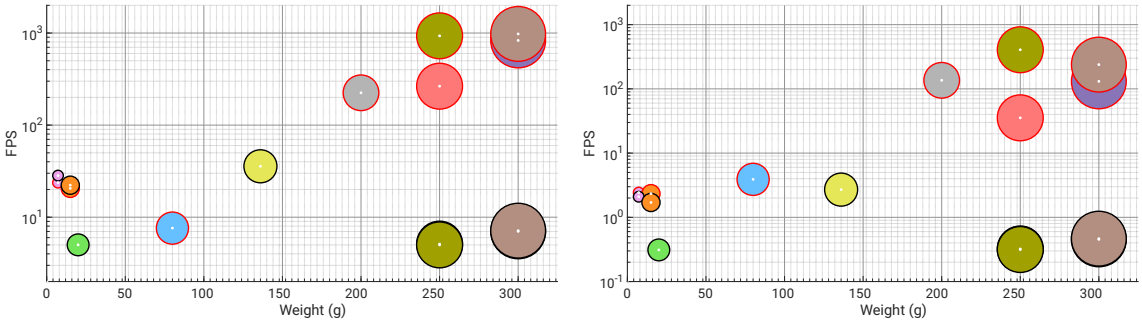


Figure F.7: Weight vs. FPS for SqueezeNet$_4$ (T$\times$2, S$\times$2) on different hardware and software optimization combinations. Left: small ($\leq$0.83 MB) model, right: large ($\leq$8.3 MB) model. The radius of each circle is proportional to log of volume of each hardware.

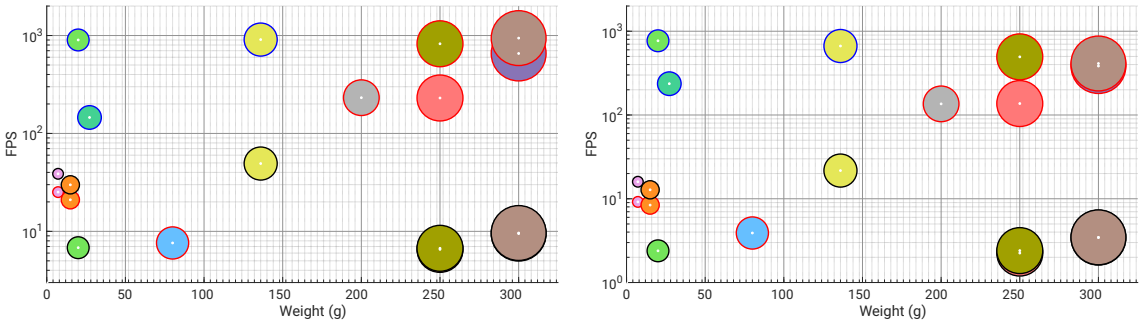


Figure F.8: Weight vs. FPS for MobileNet$_4$ (T$\times$2, S$\times$2) on different hardware and software optimization combinations. Left: small ($\leq$0.83 MB) model, right: large ($\leq$8.3 MB) model. The radius of each circle is proportional to log of volume of each hardware.

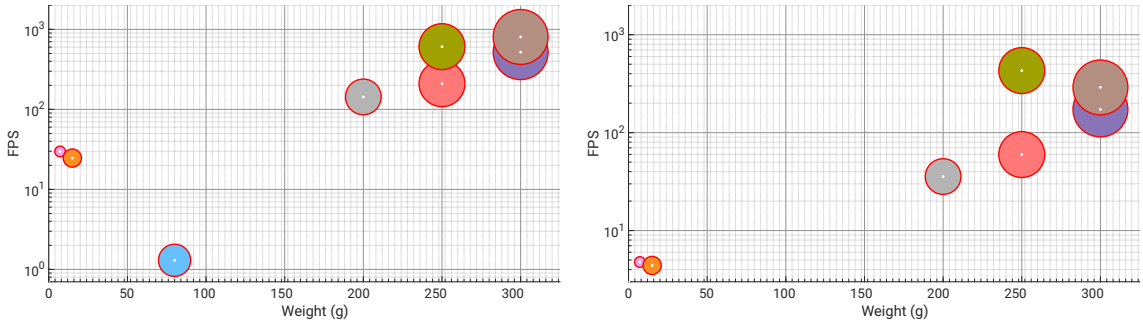


Figure F.9: Weight vs. FPS for ShuffleNet$_4$ (T×2, S×2) on different hardware and software optimization combinations. Left: small (≤0.83 MB) model, right: large (≤8.3 MB) model. The radius of each circle is proportional to log of volume of each hardware.

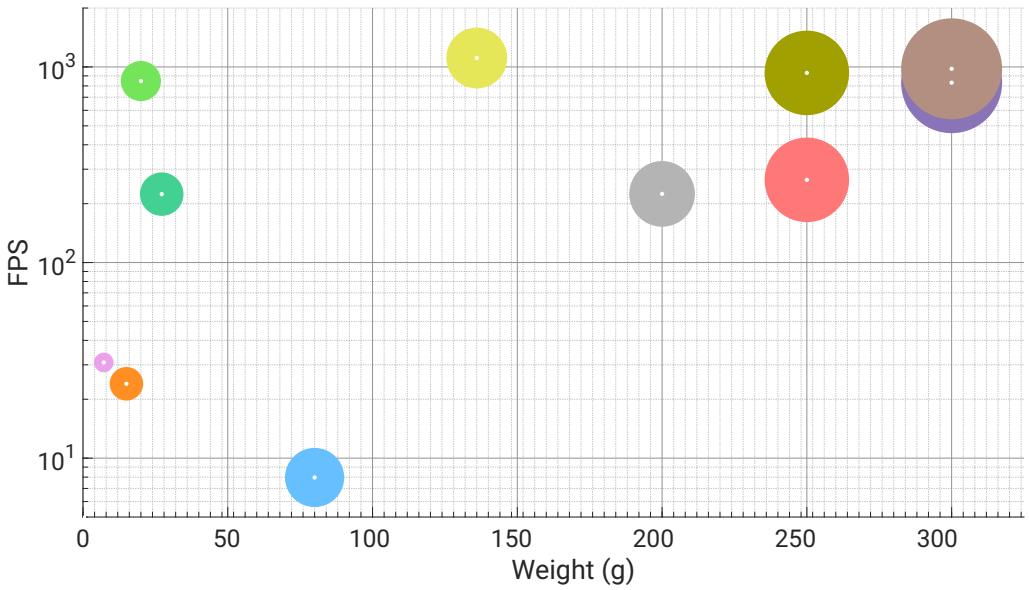
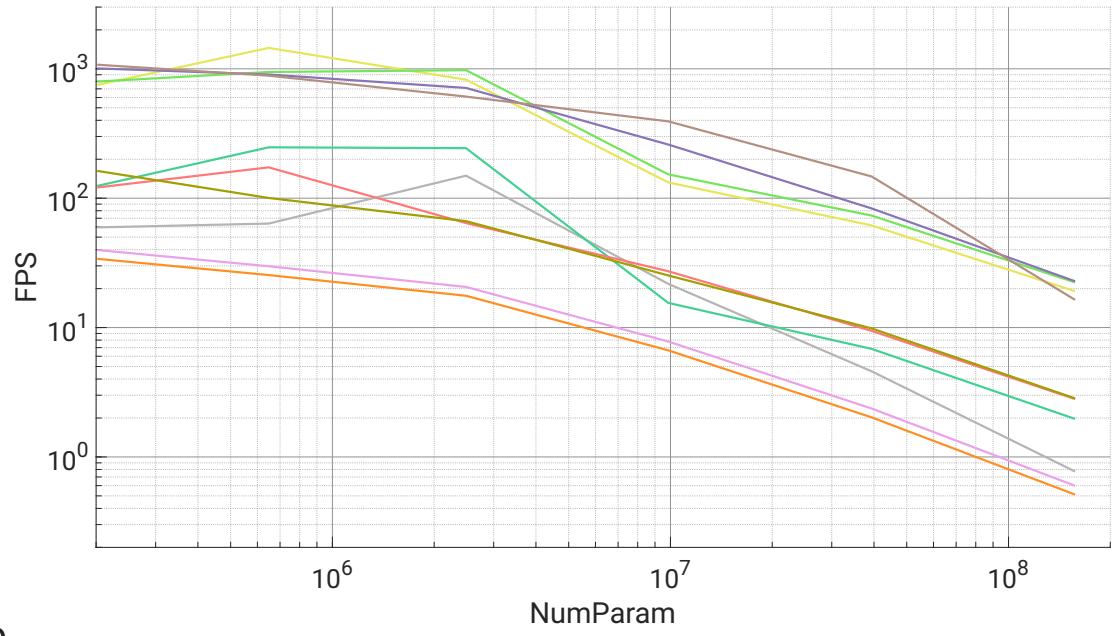
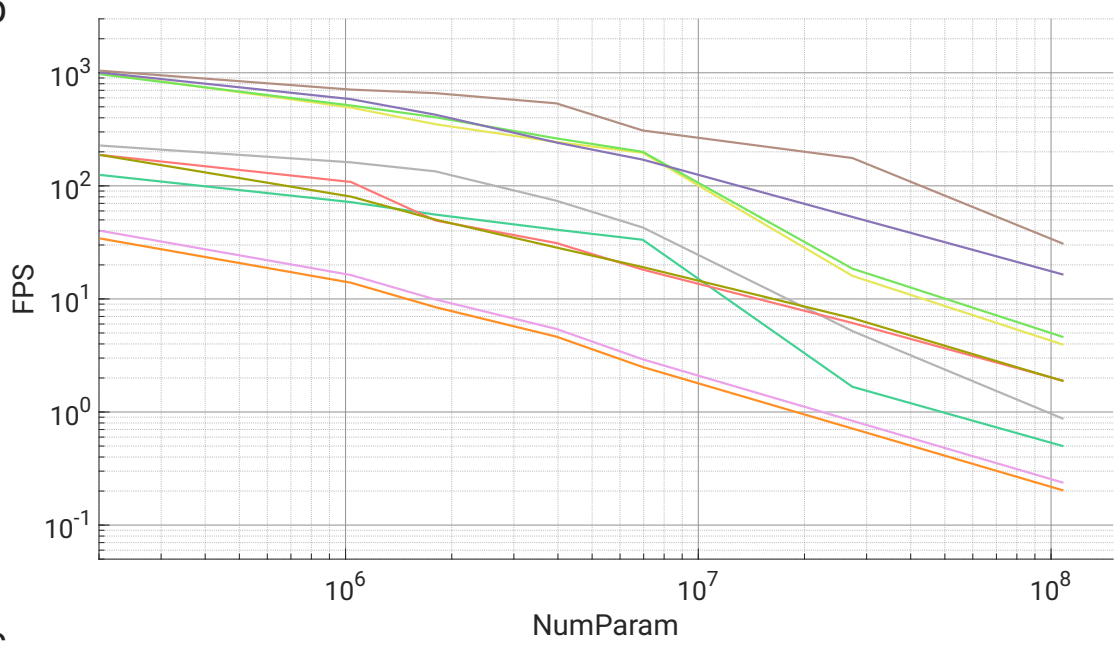


Figure F.10: Weight vs. FPS for the best model architecture on each hardware coupled to the best software optimization combination. The radius of each circle is proportional to log of volume of each hardware. The best model architecture and model optimization for each hardware are: Up: ResNet$_S$-`Float32`, CoralDev: ResNet$_S$-`Int8-EdgeTPU`, CoralUSB: ResNet$_S$-`Int8-EdgeTPU`, NanoPi: ResNet$_S$-`Int8`, BananaPiM2-Zero: ResNet$_S$-`Int8`, TX2: SqueezeNet$_S$-`Float32`, Laptop-i7: SqueezeNet$_S$-`Float32`, Laptop-1070: SqueezeNet$_S$-`Float32`, PC-i9: SqueezeNet$_S$-`Float32`, PC-TitanXp: SqueezeNet$_S$-`Float32`. All networks use T×2, S×2 configuration and S and L subscripts indicate small and large networks respectively. The best network for each hardware was chosen with the avg. error ≤ 2.5 px. and the configuration which gives the higest FPS.

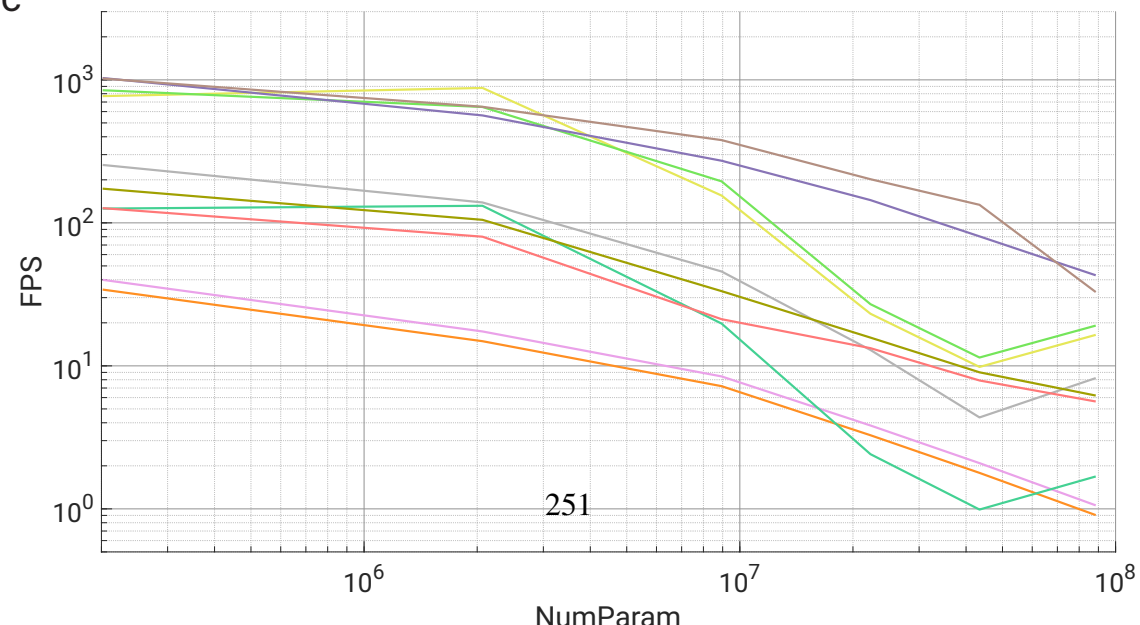
a



b



c



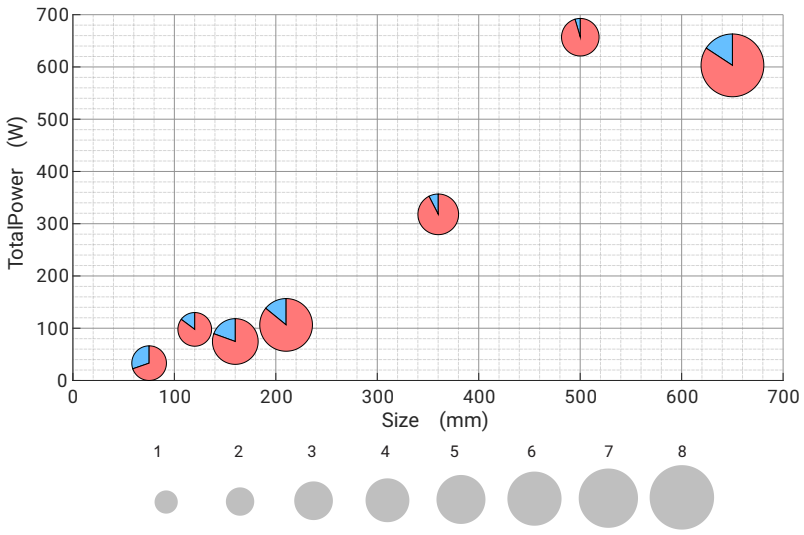


Figure F.12: Total Power vs. Quadrotor Size at hover. Each sample is a pie chart which shows the percentage of power consumed by the motors in red and compute and sensing power in blue. The radius of the pie chart is proportional to the power efficiency (in g/W and is given as the ratio of hover thrust to hover power). Refer to the legend on the bottom (gray circles) with the numbers on top indicating power efficiency in g/W.

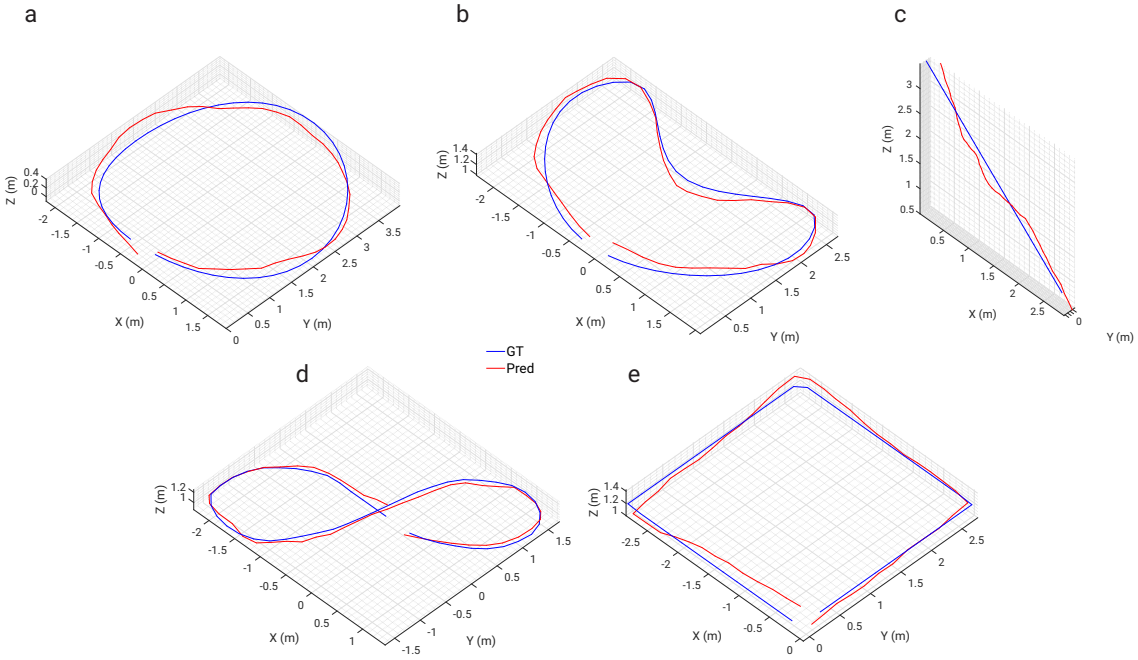


Figure F.13: Comparison of trajectory obtained by dead-reckoning (red) our estimates with respect to the ground truth (blue) for quadrotor flight in various trajectory shapes. (a) `Circle`, (b) `Moon`, (c) `Line`, (d) `Figure8` and (e) `Square`.

F.5.2 Hardware Aware Design

We answer the following questions in this section:

1. How fast do different network architectures run on a variety of computing platforms subject to weight and volume constraints?
2. What is the most efficient network architecture for a given hardware abiding the SWAP constraints?
3. How does varying network width and depth affect the speed of different network architectures on different computing platforms?
4. Which hardware setup is more power efficient?
5. How significant is power used for computing compared to power used by other quadrotor components?

Table F.9: Different-sized Quadrotor Configuration with respective computers.

Quadrotor Size (mm)	Propeller Size (mm)	Motor	Computing Board	Computer Weight (g) ↓	Total Weight (g) ↓	Auto. Thrust to Weight ↑	Auto. Hover Power (W) ↓	Hover Power (W) ↓	ρ ↓
75	40	Happymodel SE0706 15000KV	Nano Pi	7	62	1.43	33	17	1.92
120	63	T-Motor F15 Pro KV6000	Nano Pi + Coral USB	29	209	4.67	98	72	1.36
160	76	T-Motor F20II KV3750	Nano Pi + Coral USB	29	279	4.93	75	49	1.53
210	152	EMAX RS2306 KV2750	Coral Dev Board	136	536	**9.91**	106	64	1.67
360	178	T-Motor F80 Pro KV1900	NVIDIA® NVIDIA® Jetson™ TX2	200	1100	7.69	318	222	1.43
500	254	iFlight XING 2814 1100KV	NVIDIA® NVIDIA® Jetson™ Xavier AGX	600	2000	4.98	657	551	**1.19**
650	381	Tarot 4414 KV320	Intel® NUC + NVIDIA® Jetson™ Xavier AGX	1300	3900	2.72	603	405	1.49

In the wise words of Alan Kay, a pioneer in computer engineering *"People who are really serious about software should make their own hardware"* one would ideally want to design a hardware chip for a specific SWAP constraint dictated by the size and amount of features desired in the aerial robot. This can generally only be achieved by the elite drone manufacturing companies owing to the exorbitant non-recurring engineering cost,

Table F.10: Trajectory evaluation for flight experiemtnts of PRGFlow.

Trajectory	Error (m) ↓				Traj.
	X	Y	Z	RMSE	Length (m)
`Circle`	0.04	0.06	0.14	0.12	12.21
`Moon`	0.06	0.08	0.08	0.09	11.67
`Line`	0.06	0.06	0.10	0.09	3.84
`Figure8`	0.03	0.03	0.05	0.05	10.91
`Square`	0.04	0.05	0.10	0.08	10.77

thereby, putting research labs at a handicap. However, due to the rising Internet of Things (IoT) technologies and computers required to fit tight SWAP constraints researchers can repurpose these computers for efficient utilization on quadrotors (or aerial robots in general). In this spirit, we limit our analysis to commonly used computers designed for IoT purposes which are repurposed for use on aerial robots. The computers used in our experiments are summarized in Table F.1 and discussed in more detail in Subsec. F.3.5.

Refer to Figs. F.5, F.6, F.7, F.8 and F.9 for a plot of Weight vs. FPS (Frames Per Second) vs. volume of the computer for VanillaNet, ResNet, SqueezeNet, MobileNet and SqueezeNet respectively. All the networks include both small ($\leq$0.83 MB) and large ($\leq$8.3 MB) configurations training using supervised l_2 loss function and optimized using different post-quantization optimizations such as `Int8-TFLite` and `Int8-EdgeTPU`. We exclude results from `Float32-TFLite` due to inferior performance without any significant speedups as compared to the original `Float32` model. On can clearly observe that for all the networks the desktop and laptop give the best speed (highest FPS) but also have the largest weights. It would be advisable to gut out a gaming laptop and use it on a larger quadrotor ($\geq$650 mm) due to the availability of integrated NVIDIA® mobile GPUs with a large amount of CUDA® cores which can be used to accelerate both deep

learning and traditional computer vision tasks. An important factor to realize is that using `Int8-EdgeTPU` optimization to run on either the Coral Dev board or the Coral USB accelerator can provide significant speed-ups of up to 52× compared to `Int8-TFLite` without significant loss in accuracy. However, not all operations are supported in TensorFlow Lite and even less operations are supported for EdgeTPU optimization. Also, a drop in speed when going from a smaller model to a larger model is less significant in coral boards due to efficient TPU architecture. Hence, it is advisable to use the Coral Dev board or the Coral USB accelerator whenever possible. We also exclude Intel®'s Movidus Neural Compute sticks in our analysis since they provide inferior performance than Coral boards, are harder to use, weigh more and are larger.

A non-obvious observation is that the `Int8-TFLite` execution speed is much lower than the `Float32` model on laptops and desktops (both CPU and GPU). This is because of lack of optimized 8-bit integer instruction sets which are generally present in lower-end ARM computers such as the NanoPi and BananaPi. We can also observe a similar drop in performance of up to 2.6× when AVX and SSE optimized instruction sets are not used on the desktop (Fig. F.5, datapoint indicated as PC-i9 NoAVX). The lack of good performance of the Up board can also be pinned to the lack of AVX and SSE instruction sets and should be avoided for neural network tasks if not coupled to a neural network acclerator such as the Intel® Movidius compute stick or the Coral USB accelerator. We also observe speedups of upto 1.7× on the NanoPi and BananaPi by converting a `Float32` model to `Int8-TFLite` due to accelerated NEON instruction sets. Also, note that ShuffleNet models do not support `Int8-TFLite` and `Int8-EdgeTPU` optimizations due to unsupported layers. Another interesting observation is that, ResNet

(both smaller and larger) and smaller SqueezeNet models achieve almost the same speed on both CPUs and GPUs on the desktop computer.

We choose as the best network architecture and configuration (small versus large) for each hardware, the one which has an average error $\leq$2.5 px. and gives the maximum speed. These results are illustrated in Fig. F.10. For smaller boards like NanoPi and BananaPi it is recommended to use the smaller ResNet with `Int8-TFLite` optimization and for coral boards `Int8-TFLite` optimization should be used. For the medium sized board TX2 it is advisable to use the smaller SqueezeNet without any optimization (however, we did not explore TensorRT optimization which can result in a 5-20$\times$ speedup since we limit our analysis to the TensorFlow on Python^{TM} only environment for ease of use). Again, for larger computers such as a laptop (expect similar performance with a NUC + Jetson Xavier AGX) and a desktop, the smaller SqueezeNet model gives the best performance.

To gather insight into which network dimension (width versus depth) affects the speed the most and to observe the trend on different computers, we measured the speed by varying depth only (Fig. F.11a), width only (Fig. F.11b) and width + depth together (Fig. F.11c). All the networks used in this experiment were VanillaNet$_1$ (PS$\times$1) for consistency. One would expect that increasing depth and width should result in a drop in speed, but this is not always the case. The speeds peak for specific depth/width values (different for different computers) when the perfect balance of memory accesses, size of convolution filters and OPs is achieved. This is more prominent in smaller computers as compared to larger ones (laptop and desktop). This has to be carefully considered when designing networks for use on aerial robots. Also, note that the rate of drop in FPS is more

significant with increase in width. A similar trend is observed in Fig. F.11c. When the depth is increased and the width is decreased there is a sudden increase in FPS towards the end. Finally, performance using Coral with NanoPi can give higher FPS compared to the desktop or laptop when the models are smaller. Hence, using multiple NanoPi + Coral configurations can be more efficient (in-terms of weight and volume) on larger aerial robots as well.

We categorized quadrotors into six standard configurations based on their size – from 75 mm to 650 mm (pico to large sized), abiding the SWAP constraints. Refer to Table F.9 and Fig. F.12. Each quadrotor is configured with a suitable motor, propeller size and most powerful computer that fits the respective quadrotor frame. We define ρ as the ratio of hover power with and without computer. Most literature only talks about the amount of power the on-board computer uses, but this only highlights half of the story. Adding a computer to a quadrotor (to make it autonomous) not only consumes power from the battery but also adds weight which in-turn makes the motor power consumption higher at hover (one would need more thrust to keep the quadrotor flying). Generally, the power efficiency (thrust to power ratio in g/W) decreases with an increase in thrust aggravating the situation further. The essence of this is quantified by ρ which indicates how much more power the autonomous quadrotor uses compared to a manual one with the same configuration (excluding computer and sensors). A value of $\rho = 1.0$ is the theoretical best, and the larger the value (above 1.0), the more inefficient the setup. One could clearly observe that the amount of thrust directly increases with quadrotor size as it should but the thrust-to-weight ratio follows a parabolic curve (opening at the top) which achieves a maximum value at 210 mm size. This is due to efficient motor design perfected for racing

quadrotors. We also observe that for smaller quadrotors ($\leq$75 mm) the power overhead due to adding the computer is significant (as high as 92%). The power overhead decreases and then increases again as size increases due to the addition of multiple computers at 650 mm size. Also, note that for aggressively flying quadrotors the value of ρ will decrease significantly since we choose the most efficient hovering motors available on the market to maximize battery life. Fig. F.1 shows four different sized quadrotors, computers and other commonly used quadrotor electronics for a size comparison. We also show how small a hardware designed from the ground-up (Snapdragon flight) can be. We exclude this from our discussions since the smaller model of Snapdragon flight is currently phased out by Qualcomm. Finally, we also experimented with the Sipeed Maix Bit which is a low power neural network accelerator ($<$ 1 W of power) weighing only 20 g with a camera and which can be used on smaller sized quadrotors. However, due to lack of ease of use we exclude it from our discussion.

F.5.3 Trajectory Evaluation

The predictions $\widetilde{h}$ (VanillaNet$_4$ T$\times$2, S$\times$2 large model trained using supervised l_2 loss) are obtained every four frames and are integrated using dead-reckoning to obtain the final trajectory. The trajectory is aligned with the ground truth and evaluated using the approach given in [244]. The errors in inidividual axes (l_1 distance) and all axes (RMSE) are given for various trajectories in Table F.10 and are illustrated in Fig. F.13. We notice that even with simple dead-reakoning we obtain an RMSE of less than 3% of the trajectory length highlighting the robustness of the proposed PRGFlow.

F.6 Summary And Directions For Future Work

A summary of our observations are given below:

- The effect of the choice of network architecture is fairly significant on the accuracy, number of parameters and OPs and needs to be carefully considered when designing a network for deployment on an aerial robot.

- Although number of parameters is inversely correlated to speed, sometimes increasing the number of parameters (depth or width) can achieve a speedup due to a better balance in memory accesses, size of convolution filters and OPs which is more significant in smaller computers than larger ones.

- It is advisable to use supervised methods for odometry estimation since they are much easier to train and are generally more accurate than their unsupervised counterparts. The loss in accuracy from simulation training to real world is insignificant if enough variation in samples is used which have real world images.

- For odometry related regression problems, the simplest method of dropping weights works as well as more complex distillation methods.

- Accelerated instruction sets can provide huge speedups and should not be neglected for neural network based methods.

- Lastly, deep learning based odometry have two main advantages: they are generally faster due to hardware parallelization and fail more gracefully (work better in adverse scenarios) when compared to their classical counterparts.

- A combination of both classical and deep learning methods to solve a problem generally would yield better explainability and preformance gains.

Based on the observations and empirical analyses we hope the following directions for future work can further enhance ego-motion/odometry capabilities using deep learning.

- Crafting better supervised loss functions (probably on manifolds) should yield more accurate and robust results and probably will surpass advantages of un-supervised/self-supervised methods given enough domain variation.

- Formulating problems as nested loops can further simplify problems and provide better explainability than end-to-end deep learning. Also, this enables solving certain problems directly with simple mathematical formulations which can further decrease the number of neurons/OPs required, thereby reducing latency in most cases.

- Utilizing multi-frame constraints can significantly boost performance of deep odometry systems.

- Better architectures need to be designed to fully utilize the capabilities of un-supervised/self-supervised methods of deep odometry estimation.

F.7 Conclusions

We presented a simple way to estimate ego-motion/odometry on an aerial robot using deep learning combining commonly found sensors on-board: a up/down-facing cam-

era, an altimeter source and and IMU. By utilizing simple filtering methods to estimate attitude one can obtain "cheap" odometry using attitude compensated frames as the input to a network. We further provided a comprehensive analysis of warping combinations, network architectures and loss functions. All our approaches were benchmarked on different commonly used hardware with different SWAP constraints for speed and accuracy which we hope will serve as a reference manual for researchers and practitioners alike. We also show extensive real-flight odometry results highlighting the robustness of the approach without any fine-tuning or re-training. Finally, as a parting thought, utilizing deep learning when failure is often expected would most likely lead to more robust system.

Acknowledgement

The support of the Brin Family Foundation, the Northrop Grumman Mission Systems University Research Program, ONR under grant award N00014-17-1-2622, and the support of the National Science Foundation under grant BCS 1824198 are gratefully acknowledged. We would also like to thank NVIDIA for the grant of a Titan-Xp GPU used in the experiments.

Bibliography

[1] J. Pan, C. Canton Ferrer, K. McGuinness, N. E. O'Connor, J. Torres, E. Sayrol, and X. Giro-i-Nieto. SalGAN: Visual Saliency Prediction with Generative Adversarial Networks. *ArXiv e-prints*, January 2017.

[2] Ursula Jander and Rudolf Jander. Allometry and resolution of bee eyes (apoidea). *Arthropod Structure & Development*, 30(3):179–193, 2002.

[3] Prajwal Shanthakumar, Kevin Yu, Mandeep Singh, Jonah Orevillo, Eric Bianchi, Matthew Hebdon, and Pratap Tokekar. View planning and navigation algorithms for autonomous bridge inspection with uavs. In *International Symposium on Experimental Robotics*, pages 201–210. Springer, 2018.

[4] Martin Saska, Vít Krátkỳ, Vojtěch Spurnỳ, and Tomáš Báča. Documentation of dark areas of large historical buildings by a formation of unmanned aerial vehicles using model predictive control. In *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2017.

[5] Dinesh Thakur, Giuseppe Loianno, Wenxin Liu, and Vijay Kumar. Nuclear environments inspection with micro aerial vehicles: Algorithms and experiments. In *International Symposium on Experimental Robotics*, pages 191–200. Springer, 2018.

[6] Prajwal Shanthakumar, Kevin Yu, Mandeep Singh, Jonah Orevillo, Eric Bianchi, Matthew Hebdon, and Pratap Tokekar. View planning and navigation algorithms for autonomous bridge inspection with uavs. In Jing Xiao, Torsten Kröger, and Oussama Khatib, editors, *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 201–210, Cham, 2020. Springer International Publishing.

[7] Qianhao Wang, Yuman Gao, Jialin Ji, Chao Xu, and Fei Gao. Visibility-aware trajectory optimization with application to aerial tracking. *arXiv preprint arXiv:2103.06742*, 2021.

[8] Zhichao Han, Ruibin Zhang, Neng Pan, Chao Xu, and Fei Gao. Fast-tracker: A robust aerial system for tracking agile target in cluttered environments, 2020.

[9] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew Davison, Jörg Conradt, Kostas

Daniilidis, et al. Event-based vision: A survey. *arXiv preprint arXiv:1904.08405*, 2019.

[10] A. Mitrokhin et al. Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, Oct 2018.

[11] G. Gallego et al. A unifying contrast maximization framework for event cameras, with applications to motion, depth, and optical flow estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[12] A. Mitrokhin et al. Ev-imo: Motion segmentation dataset and learning pipeline for event cameras. *arXiv preprint arXiv:1903.07520*, 2019.

[13] Anton Mitrokhin, Zhiyuan Hua, Cornelia Fermuller, and Yiannis Aloimonos. Learning visual motion segmentation using event surfaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14414–14423, 2020.

[14] Andrea Censi and Davide Scaramuzza. Low-latency event-based visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 703–710. IEEE, 2014.

[15] A. Zhu et al. Event-based visual inertial odometry. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5816–5824. IEEE, 2017.

[16] Yi Zhou, Guillermo Gallego, and Shaojie Shen. Event-based stereo visual odometry. *IEEE Transactions on Robotics*, 2021.

[17] Henri Rebecq, Timo Horstschäfer, Guillermo Gallego, and Davide Scaramuzza. Evo: A geometric approach to event-based 6-dof parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2):593–600, 2016.

[18] Arren Glover and Chiara Bartolozzi. Robust visual tracking with a freely-moving event camera. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3769–3776. IEEE, 2017.

[19] Stefano Mintchev and Dario Floreano. Adaptive morphology: A design principle for multimodal and multifunctional robots. *IEEE Robotics & Automation Magazine*, 23(3):42–54, 2016.

[20] Moju Zhao, Koji Kawasaki, Xiangyu Chen, Shintaro Noda, Kei Okada, and Masayuki Inaba. Whole-body aerial manipulation by transformable multirotor with two-dimensional multilinks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5175–5182. IEEE, 2017.

[21] Moju Zhao, Tomoki Anzai, Fan Shi, Xiangyu Chen, Kei Okada, and Masayuki Inaba. Design, modeling, and control of an aerial robot dragon: A dual-rotor-embedded multilink robot with the ability of multi-degree-of-freedom aerial transformation. *IEEE Robotics and Automation Letters*, 3(2):1176–1183, 2018.

[22] A Desbiez, F Expert, M Boyron, J Diperi, S Viollet, and F Ruffier. X-morf: A crash-separable quadrotor that morfs its x-geometry in flight. In *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 222–227. IEEE, 2017.

[23] Na Zhao, Yudong Luo, Hongbin Deng, and Yantao Shen. The deformable quadrotor: Design, kinematics and dynamics characterization, and flight performance validation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2391–2396. IEEE, 2017.

[24] Davide Falanga, Kevin Kleber, Stefano Mintchev, Dario Floreano, and Davide Scaramuzza. The foldable drone: A morphing quadrotor that can squeeze and fly. *IEEE Robotics and Automation Letters*, 4(2):209–216, 2018.

[25] Y. Nakabo, T. Mukai, Y. Hattori, Y. Takeuchi, and N. Ohnishi. Variable baseline stereo tracking vision system using high-speed linear slider. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1567–1572, 2005.

[26] Stamatios Psarras, A Zarkali, S Hanna, et al. Visual saliency in navigation: Modelling navigational behaviour using saliency and depth analysis. International Space Syntax Symposium, 2019.

[27] Boris Schauerte and Gernot A Fink. Focusing computational visual attention in multi-modal human-robot interaction. In *International conference on multimodal interfaces and the workshop on machine learning for multimodal interaction*, pages 1–8, 2010.

[28] Boris Schauerte and Rainer Stiefelhagen. "look at this!" learning to guide visual saliency in human-robot interaction. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 995–1002. IEEE, 2014.

[29] Cynthia Breazeal, Aaron Edsinger, Paul Fitzpatrick, and Brian Scassellati. Active vision for sociable robots. *IEEE Transactions on systems, man, and cybernetics-part A: Systems and Humans*, 31(5):443–453, 2001.

[30] Yash Mulgaonkar and Vijay Kumar. Towards open-source, printable pico-quadrotors. In *Proc. Robot Makers Workshop, Robotics: Science Systems Conf*, 2014.

[31] Daniele Palossi, Antonio Loquercio, Francesco Conti, Eric Flamand, Davide Scaramuzza, and Luca Benini. Ultra low power deep-learning-powered autonomous nano drones. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2018)*. ETH Zurich, 2018.

[32] GCHE De Croon, M Perçin, B Remes, R Ruijsink, and C De Wagter. The delfly. *Dordrecht: Springer Netherlands. doi*, 10:978–94, 2016.

[33] Morgan Quigley, Kartik Mohta, Shreyas S Shivakumar, Michael Watterson, Yash Mulgaonkar, Mikael Arguedas, Ke Sun, Sikang Liu, Bernd Pfrommer, Vijay Kumar, et al. The open vision computer: An integrated sensing and compute system for mobile robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 1834–1840. IEEE, 2019.

[34] Nitin J Sanket, Chahat Deep Singh, Kanishka Ganguly, Cornelia Fermüller, and Yiannis Aloimonos. Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robotics and Automation Letters*, 3(4):2799–2806, 2018.

[35] Nitin J. Sanket, Chethan M. Parameshwara, Chahat Deep Singh, Ashwin V. Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, and Yiannis Aloimonos. Evdodgenet: Deep dynamic obstacle dodging with event cameras, 2019.

[36] Tonci Novkovic, Remi Pautrat, Fadri Furrer, Michel Breyer, Roland Siegwart, and Juan Nieto. Object finding in cluttered scenes using interactive perception. *arXiv preprint arXiv:1911.07482*, 2019.

[37] Y. Mulgaonkar, A. Makineni, L. Guerrero-Bonilla, and V. Kumar. Robust aerial robot swarms without collision avoidance. *IEEE Robotics and Automation Letters*, 3(1):596–603, Jan 2018.

[38] Matthew Collett, Thomas S Collett, Sonja Bisch, and Rüdiger Wehner. Local and global vectors in desert ant navigation. *Nature*, 394(6690):269, 1998.

[39] Randolf Menzel, Konstantin Lehmann, Gisela Manz, Jacqueline Fuchs, Miriam Koblofsky, and Uwe Greggers. Vector integration and novel shortcutting in honeybee navigation. *Apidologie*, 43(3):229–243, 2012.

[40] Mario Pahl, Hong Zhu, Jürgen Tautz, and Shaowu Zhang. Large scale homing in honeybees. *PLoS One*, 6(5):e19669, 2011.

[41] Karan P Jain and Mark W Mueller. Flying batteries: In-flight battery switching to increase multirotor flight time. *arXiv preprint arXiv:1909.10091*, 2019.

[42] Davide Falanga, Alessio Zanchettin, Alessandro Simovic, Jeffrey Delmerico, and Davide Scaramuzza. Vision-based autonomous quadrotor landing on a moving platform. In *2017 IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 200–207. IEEE, 2017.

[43] John A Dougherty and Taeyoung Lee. Monocular estimation of ground orientation for autonomous landing of a quadrotor. *Journal of Guidance, Control, and Dynamics*, (null):1407–1416, 2016.

[44] Bas J Pijnacker Hordijk, Kirk YW Scheper, and Guido CHE De Croon. Vertical landing for micro air vehicles using event-based optical flow. *Journal of Field Robotics*, 35(1):69–90, 2018.

[45] C. E. Doyle, J. J. Bird, T. A. Isom, J. C. Kallman, D. F. Bareiss, D. J. Dunlop, R. J. King, J. J. Abbott, and M. A. Minor. An avian-inspired passive mechanism for quadrotor perching. *IEEE/ASME Transactions on Mechatronics*, 18(2):506–517, April 2013.

[46] Daigo Shishika and Derek A Paley. Mosquito-inspired distributed swarming and pursuit for cooperative defense against fast intruders. *Autonomous Robots*, 43(7):1781–1799, 2019.

[47] Dingjiang Zhou, Zijian Wang, and Mac Schwager. Agile coordination and assistive collision avoidance for quadrotor swarms using virtual structures. *IEEE Transactions on Robotics*, 34(4):916–923, 2018.

[48] Senthil Hariharan Arul and Dinesh Manocha. Dcad: Decentralized collision avoidance with dynamics constraints for agile quadrotor swarms. *arXiv preprint arXiv:1909.03961*, 2019.

[49] Michael Gassner, Titus Cieslewski, and Davide Scaramuzza. Dynamic collaboration without communication: Vision-based cable-suspended load transport with two quadrotors. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5196–5202. IEEE, 2017.

[50] Farhad A Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric control of a quadrotor uav transporting a payload connected via flexible cable. *International Journal of Control, Automation and Systems*, 13(6):1486–1498, 2015.

[51] Guofan Wu and Koushil Sreenath. Geometric control of multiple quadrotors transporting a rigid-body load. In *53rd IEEE Conference on Decision and Control*, pages 6141–6148. IEEE, 2014.

[52] John Aloimonos et al. Active vision. *International journal of computer vision*, 1(4):333–356, 1988.

[53] Jeannette Bohg et al. Interactive perception: Leveraging action in perception and perception in action. *IEEE Transactions on Robotics*, 33(6):1273–1291, 2017.

[54] Ruzena Bajcsy et al. Revisiting active perception. *Autonomous Robots*, pages 1–20, 2017.

[55] Cornelia Fermüller, David Doermann, Daniel Dementhon, Liuqing Huang, Radu Jasinschi, Ehud Rivlin, Rajeev Sharma, David Shulman, Bradley Stewart, and Cheolwoo Yoo. Basic visual capabilities. 1993.

[56] Cornelia Fermüller and Yiannis Aloimonos. Vision and action. *Image and vision computing*, 13(10):725–744, 1995.

[57] Nitin J Sanket, Chethan M Parameshwara, Chahat Deep Singh, Ashwin V Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, and Yiannis Aloimonos. Evdodgenet: Deep dynamic obstacle dodging with event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10651–10657. IEEE, 2020.

[58] Davide Falanga, Kevin Kleber, and Davide Scaramuzza. Dynamic obstacle avoidance for quadrotors with event cameras. *Science Robotics*, 5(40), 2020.

[59] Teodor Tomic et al. Toward a fully autonomous uav: Research platform for indoor and outdoor urban search and rescue. *IEEE robotics & automation magazine*, 19(3):46–56, 2012.

[60] Tolga Özaslan et al. Inspection of penstocks and featureless tunnel-like environments using micro UAVs. In *Field and Service Robotics*, pages 123–136. Springer, 2015.

[61] Nathan Michael et al. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, 2012.

[62] Takafumi Taketomi et al. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, June 2017.

[63] T. Qin et al. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[64] M. Bloesch et al. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 298–304. IEEE, 2015.

[65] Thomas S Collett. Insect vision: controlling actions through optic flow. *Current Biology*, 12(18):R615–R617, 2002.

[66] Bruno Mantel et al. Exploratory movement generates higher-order information that is sufficient for accurate perception of scaled egocentric distance. *PloS one*, 10(4):e0120025, 2015.

[67] Karl Kral and Michael Poteser. Motion parallax as a source of distance information in locusts and mantids. *Journal of insect behavior*, 10(1):145–163, 1997.

[68] Geoslam. https://geoslam.com/, 2018.

[69] Felix Endres et al. 3-D mapping with an RGB-D camera. *IEEE Transactions on Robotics*, 30(1):177–187, 2014.

[70] Parrot slamdunk. http://developer.parrot.com/docs/slamdunk/, 2018.

[71] Jakob Engel et al. Direct sparse odometry. *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[72] N. Smolyanskiy et al. Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness. *arXiv preprint arXiv:1705.02550*, 2017.

[73] Paolo D'Alberto et al. Generating FPGA-accelerated DFT libraries. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pages 173–184. IEEE, 2007.

[74] Eriko Nurvitadhi et al. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 5–14. ACM, 2017.

[75] Song Han et al. ESE: Efficient speech recognition engine with sparse LSTM on FPGA. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84. ACM, 2017.

[76] G. Loianno et al. Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu. *IEEE Robotics and Automation Letters*, 2(2):404–411, 2017.

[77] D. Falanga et al. Aggressive quadrotor flight through narrow gaps with onboard sensing and computing using active vision. In *Robotics and Automation, 2017 IEEE International Conference on*, pages 5774–5781. IEEE, 2017.

[78] Nicolas Franceschini et al. From insect vision to robot vision. *Phil. Trans. R. Soc. Lond. B*, 337(1281):283–294, 1992.

[79] Mandyam V Srinivasan et al. Robot navigation inspired by principles of insect vision. *Robotics and Autonomous Systems*, 26(2-3):203–216, 1999.

[80] Julien R Serres and Franck Ruffier. Optic flow-based collision-free strategies: From insects to robots. *Arthropod structure & development*, 46(5):703–717, 2017.

[81] K. Scheper et al. Behavior trees for evolutionary robotics. *Artificial life*, 22(1):23–48, 2016.

[82] Longuet-Higgins et al. The interpretation of a moving retinal image. In *Proc. R. Soc. Lond. B*, volume 208, pages 385–397. The Royal Society, 1980.

[83] K. Sreenath et al. Trajectory generation and control of a quadrotor with a cable-suspended load-a differentially-flat hybrid system. In *Robotics and Automation, 2013 IEEE International Conference on*, pages 4888–4895. IEEE, 2013.

[84] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. 1981.

[85] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[86] E. Ilg et al. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2017.

[87] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. 1991.

[88] Olaf Hall-Holt et al. Finding large sticks and potatoes in polygons. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 474–483. Society for Industrial and Applied Mathematics, 2006.

[89] D. Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Robotics and Automation, 2011 IEEE International Conference on*, pages 2520–2525. IEEE, 2011.

[90] Clément G. et al. Unsupervised monocular depth estimation with left-right consistency. In *Computer Vision and Pattern Recognition, 2017 IEEE International Conference on*, 2017.

[91] A. Ranjan and Michael J. Black. Optical flow estimation using a spatial pyramid network. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[92] T. Kroeger et al. Fast optical flow using dense inverse search. In *European Conference on Computer Vision*, pages 471–488. Springer, 2016.

[93] JiaWang Bian et al. Gms: Grid-based motion statistics for fast, ultra-robust feature correspondence. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[94] Edward Rosten et al. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):105–119, 2010.

[95] Mårten Björkman et al. Detecting, segmenting and tracking unknown objects using multi-label mrf inference. *Computer Vision and Image Understanding*, 118:111–127, 2014.

[96] D. Falanga et al. How Fast Is Too Fast? The Role of Perception Latency in High-Speed Sense and Avoid. *IEEE Robotics and Automation Letters*, 4(2):1884–1891, April 2019.

[97] P. Sermanet et al. Speed-range dilemmas for vision-based navigation in unstructured terrain. *IFAC Proceedings Volumes*, 40(15):300–305, 2007.

[98] G. Gallego, , et al. Event-based vision: A survey. *arXiv preprint arXiv:1904.08405*, 2019.

[99] A. Vidal et al. Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018.

[100] Huai-Jen Liang et al. SalientDSO: Bringing attention to direct sparse odometry. *IEEE Transactions on Automation Science and Engineering*, 2019.

[101] S. Bowman et al. Probabilistic data association for semantic SLAM. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1722–1729. IEEE, 2017.

[102] R. Sabzevari and D. Scaramuzza. Multi-body motion estimation from monocular vehicle-mounted cameras. *IEEE Transactions on Robotics*, 2016.

[103] V. Vasco et al. Independent motion detection with event-driven cameras. In *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 530–536. IEEE, 2017.

[104] T. Stoffregen et al. Event-based motion segmentation by motion compensation. In *International Conference on Computer Vision (ICCV)*, 2019.

[105] H. Alvarez et al. Collision avoidance for quadrotors with a monocular camera. In *Experimental Robotics*, pages 195–209. Springer, 2016.

[106] A. Barry et al. High-speed autonomous obstacle avoidance with pushbroom stereo. *Journal of Field Robotics*, 35(1):52–68, 2018.

[107] K. Mohta, , et al. Fast, autonomous flight in gps-denied and cluttered environments. *Journal of Field Robotics*, 35(1):101–120, 2018.

[108] K. Mohta et al. Experiments in fast, autonomous, gps-denied quadrotor flight. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7832–7839. IEEE, 2018.

[109] E. Mueggler et al. Towards evasive maneuvers with quadrotors using dynamic vision sensors. In *2015 European Conference on Mobile Robots (ECMR)*, pages 1–8. IEEE, 2015.

[110] A. Maqueda et al. Event-based vision meets deep learning on steering prediction for self-driving cars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5419–5427, 2018.

[111] G. Gallego et al. Event-based camera pose tracking using a generative event model. *arXiv preprint arXiv:1510.01972*, 2015.

[112] A. Zhu et al. Unsupervised event-based learning of optical flow, depth, and egomotion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 989–997, 2019.

[113] D. DeTone et al. Method and system for performing convolutional image transformation estimation, November 23 2017. US Patent App. 15/600,545.

[114] Ty Nguyen et al. Unsupervised deep homography: A fast and robust homography estimation model. *IEEE Robotics and Automation Letters*, 3(3):2346–2353, 2018.

[115] S. Meister et al. Unflow: Unsupervised learning of optical flow with a bidirectional census loss. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[116] A. Zhu et al. Ev-flownet: self-supervised optical flow estimation for event-based cameras. *arXiv preprint arXiv:1802.06898*, 2018.

[117] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.

[118] D. Sun et al. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[119] J. Barron. A general and adaptive robust loss function. *CVPR*, 2019.

[120] W. Li et al. Interiornet: Mega-scale multi-sensor photo-realistic indoor scenes dataset. In *British Machine Vision Conference (BMVC)*, 2018.

[121] H. Rebecq et al. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, October 2018.

[122] A. Zhu et al. The multivehicle stereo event camera dataset: An event camera dataset for 3d perception. *IEEE Robotics and Automation Letters*, 3(3):2032–2039, July 2018.

[123] C. Wylie. *Introduction to projective geometry*. Courier Corporation, 2011.

[124] R. Spica et al. A real-time game theoretic planner for autonomous two-player drone racing. *arXiv preprint arXiv:1801.02302*, 2018.

[125] Kimberly McGuire, Mario Coppola, Christophe De Wagter, and Guido de Croon. Towards autonomous navigation of multiple pocket-drones in real-world environments. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 244–249. IEEE, 2017.

[126] KN McGuire, C De Wagter, K Tuyls, HJ Kappen, and GCHE de Croon. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, 4(35):eaaw9710, 2019.

[127] Nitin J Sanket, Chahat Deep Singh, Varun Asthana, Cornelia Fermüller, and Yiannis Aloimonos. Morpheyes: Variable baseline stereo for quadrotor navigation. *arXiv preprint arXiv:2011.03077*, 2020.

[128] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4557–4564. IEEE, 2012.

[129] Daniel Mellinger, Michael Shomin, Nathan Michael, and Vijay Kumar. *Cooperative Grasping and Transport Using Multiple Quadrotors*, pages 545–558. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[130] Zoran Valentak. Drone market share analysis, 2018.

[131] P. Lichtsteiner, C. Posch, and T. Delbruck. A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[132] B. Son, Y. Suh, S. Kim, H. Jung, J. Kim, C. Shin, K. Park, K. Lee, J. Park, J. Woo, Y. Roh, H. Lee, Y. Wang, I. Ovsiannikov, and H. Ryu. 4.1 a 640×480 dynamic vision sensor with a 9µm pixel and 300meps address-event representation. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, pages 66–67, 2017.

[133] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[134] Maciej Ł Pawełczyk and Marek Wojtyra. Real world object detection dataset for quadcopter unmanned aerial vehicle detection. *IEEE Access*, 8:174394–174409, 2020.

[135] Eren Unlu, Emmanuel Zenou, Nicolas Riviere, and Paul-Edouard Dupouy. Deep learning-based strategies for the detection and tracking of drones using several cameras. *IPSJ Transactions on Computer Vision and Applications*, 11(1):1–13, 2019.

[136] Fabian Schilling, Fabrizio Schiano, and Dario Floreano. Vision-based drone flocking in outdoor environments. *IEEE Robotics and Automation Letters*, 6(2):2954–2961, 2021.

[137] Viktor Walter, Nicolas Staub, Antonio Franchi, and Martin Saska. Uvdar system for visual relative localization with application to leader–follower formations of multirotor uavs. *IEEE Robotics and Automation Letters*, 4(3):2637–2644, 2019.

[138] Luis A. Mateos. Apriltags 3d: Dynamic fiducial markers for robust pose estimation in highly reflective environments and indirect communication in swarm robotics, 2020.

[139] Guanrui Li, Bruno Gabrich, David Saldana, Jnaneshwar Das, Vijay Kumar, and Mark Yim. Modquad-vi: A vision-based self-assembling modular quadrotor. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 346–352. IEEE, 2019.

[140] Maximilian Krogius, Acshi Haggenmiller, and Edwin Olson. Flexible layouts for fiducial tags. In *IROS*, pages 1898–1903, 2019.

[141] Lilian Calvet, Pierre Gurdjos, Carsten Griwodz, and Simone Gasparini. Detection and Accurate Localization of Circular Fiducials under Highly Challenging Conditions. In *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 562 – 570, Las Vegas, United States, June 2016.

[142] Anton Mitrokhin, Cornelia Fermüller, Chethan Parameshwara, and Yiannis Aloimonos. Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[143] Chethan M Parameshwara, Nitin J Sanket, Chahat Deep Singh, Cornelia Fermüller, and Yiannis Aloimonos. 0-mms: Zero-shot multi-motion segmentation with a monocular event camera.

[144] John Carlton. *Marine propellers and propulsion*. Butterworth-Heinemann, 2018.

[145] Wenchao Ding, Wenliang Gao, Kaixuan Wang, and Shaojie Shen. An efficient b-spline-based kinodynamic replanning framework for quadrotors. *IEEE Transactions on Robotics*, 35(6):1287–1306, 2019.

[146] Eric W. Weisstein. B-spline. From MathWorld—A Wolfram Web Resource.

[147] Kaihuai Qin. General matrix representations for b-splines. *The Visual Computer*, 16(3-4):177–186, 2000.

[148] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[149] Henri Rebecq, Daniel Gehrig, and Davide Scaramuzza. ESIM: an open event camera simulator. *Conf. on Robotics Learning (CoRL)*, October 2018.

[150] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[151] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[152] Nitin J Sanket, Chahat Deep Singh, Cornelia Fermüller, and Yiannis Aloimonos. Prgflow: Benchmarking swap-aware unified deep visual inertial odometry. *arXiv preprint arXiv:2006.06753*, 2020.

[153] Philipp Foehn, Dario Brescianini, Elia Kaufmann, Titus Cieslewski, Mathias Gehrig, Manasi Muglikar, and Davide Scaramuzza. AlphaPilot: Autonomous Drone Racing. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[154] Karan P Jain and Mark W Mueller. Flying batteries: In-flight battery switching to increase multirotor flight time. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3510–3516. IEEE, 2020.

[155] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[156] Bernd Pfrommer, Nitin Sanket, Kostas Daniilidis, and Jonas Cleveland. Penncosyvio: A challenging visual inertial odometry benchmark. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3847–3854. IEEE, 2017.

[157] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[158] Nitin J. Sanket, Chahat Deep Singh, Cornelia Fermüller, and Yiannis Aloimonos. PRGFlow: Benchmarking SWAP-Aware Unified Deep Visual Inertial Odometry, 2020.

[159] D. Gallup, J. Frahm, P. Mordohai, and M. Pollefeys. Variable baseline/resolution stereo. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[160] Jean-Yves Bouguet. Camera Calibration Toolbox for Matlab. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[161] Janne Heikkila and Olli Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of ieee computer society conference on computer vision and pattern recognition*, pages 1106–1112. IEEE, 1997.

[162] Zhongwei Tang, Rafael Grompone von Gioi, Pascal Monasse, and Jean-Michel Morel. A precision analysis of camera distortion models. *IEEE Transactions on Image Processing*, 26(6):2694–2704, 2017.

[163] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Proceedings of the seventh ieee international conference on computer vision*, volume 1, pages 666–673. Ieee, 1999.

[164] Cornelia Fermüller, LoongFah Cheong, and Yiannis Aloimonos. Visual space distortion. *Biological Cybernetics*, 77(5):323–337, 1997.

[165] LoongFah Cheong, Cornelia Fermüller, and Yiannis Aloimonos. Effects of errors in the viewing geometry on shape estimation. *Computer Vision and Image Understanding*, 71(3):356–372, 1998.

[166] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[167] Yan-Bin Jia. Dual quaternions. 2013.

[168] Konstantinos Daniilidis. Hand-eye calibration using dual quaternions. *The International Journal of Robotics Research*, 18(3):286–298, 1999.

[169] Ken Shoemake. Animating rotation with quaternion curves. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, 1985.

[170] Boyu Zhou, Fei Gao, Luqi Wang, Chuhao Liu, and Shaojie Shen. Robust and efficient quadrotor trajectory generation for fast autonomous flight. *IEEE Robotics and Automation Letters*, 4(4):3529–3536, 2019.

[171] Sikang Liu, Nikolay Atanasov, Kartik Mohta, and Vijay Kumar. Search-based motion planning for quadrotors using linear quadratic minimum time control. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2872–2879. IEEE, 2017.

[172] D. Honegger, T. Sattler, and M. Pollefeys. Embedded real-time multi-baseline stereo. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5245–5250, 2017.

[173] Kevin van Hecke, Guido de Croon, Laurens van der Maaten, Daniel Hennes, and Dario Izzo. Persistent self-supervised learning: From stereo to monocular vision for obstacle avoidance. *International Journal of Micro Air Vehicles*, 10(2):186–206, 2018.

[174] Chethan M Parameshwara, Nitin J Sanket, Chahat Deep Singh, Cornelia Fermüller, and Yiannis Aloimonos. 0-mms: Zero-shot multi-motion segmentation with a monocular event camera.

[175] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[176] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[177] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[178] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[179] Jan Stühmer, Stefan Gumhold, and Daniel Cremers. Real-time dense geometry from a handheld camera. In *Joint Pattern Recognition Symposium*, pages 11–20. Springer, 2010.

[180] Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 1722–1729. IEEE, 2017.

[181] Xiaodong Yu, Cornelia Fermüller, Ching Lik Teo, Yezhou Yang, and Yiannis Aloimonos. Active scene recognition with vision and language. In *2011 International Conference on Computer Vision*, pages 810–817. IEEE, 2011.

[182] Abhijit S Ogale, Cornelia Fermuller, and Yiannis Aloimonos. Motion segmentation using occlusions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):988–992, 2005.

[183] Cornelia Fermüller and Yiannis Aloimonos. Tracking facilitates 3-d motion estimation. *Biological Cybernetics*, 67(3):259–268, 1992.

[184] Ajay Mishra, Yiannis Aloimonos, and Cheong Loong Fah. Active segmentation with fixation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 468–475. IEEE, 2009.

[185] Javier Civera, Dorian Gálvez-López, Luis Riazuelo, Juan D Tardós, and JMM Montiel. Towards semantic slam using a monocular camera. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1277–1284. IEEE, 2011.

[186] Lifeng An, Xinyu Zhang, Hongbo Gao, and Yuchao Liu. Semantic segmentation–aided visual odometry for urban autonomous driving. *International Journal of Advanced Robotic Systems*, 14(5):1729881417735667, 2017.

[187] Kostas Alexis Tung Dang, Christos Papachristos. Visual saliency–aware receding horizon autonomous exploration with application to aerial robotics. In *Robotics and Automation (ICRA), 2018 IEEE International Conference on*. IEEE, 2018.

[188] Huaping Liu, Fuchun Sun, and Xinyu Zhang. Robotic material perception using active multi-modal fusion. *IEEE Transactions on Industrial Electronics*, 2018.

[189] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[190] Olivier Le Meur. Robustness and repeatability of saliency models subjected to visual degradations. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 3285–3288. IEEE, 2011.

[191] Chelhwon Kim and Peyman Milanfar. Finding saliency in noisy images. In *Computational Imaging X*, volume 8296, page 82960U. International Society for Optics and Photonics, 2012.

[192] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 00, pages 6230–6239, July 2017.

[193] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Hong Kong, China, May 2014.

[194] J. Engel, V. Usenko, and D. Cremers. A Photometrically Calibrated Benchmark For Monocular Visual Odometry. *ArXiv e-prints*, July 2016.

[195] Ming Jiang, Shengsheng Huang, Juanyong Duan, and Qi Zhao. Salicon: Saliency in context. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1072–1080, 2015.

[196] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, page 4. IEEE, 2017.

[197] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[198] Cornelia Fermüller. Navigational preliminaries. In Y. Aloimonos, editor, *Active Perception*. Lawrence Erlbaum Associates, 1993.

[199] Ji Zhang and Sanjiv Singh. Loam: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, volume 2, 2014.

[200] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2174–2181. IEEE, 2015.

[201] Fabio Morbidi, Randy A Freeman, and Kevin M Lynch. Estimation and control of uav swarms for distributed monitoring tasks. In *Proceedings of the 2011 American Control Conference*, pages 1069–1075. IEEE, 2011.

[202] Aaron Weinstein, A Cho, Giuseppe Loianno, and Vijay Kumar. Vio-swarm:: A swarm of 250g quadrotors. *IEEE RA-L Robotics and Automation Letters*, 2018.

[203] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572. IEEE, 2007.

[204] Dominik Honegger, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *2013 IEEE International Conference on Robotics and Automation*, pages 1736–1741. IEEE, 2013.

[205] Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 1722–1729. IEEE, 2017.

[206] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[207] Tong Qin et al. VINS-Mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[208] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[209] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5974–5983, 2017.

[210] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1851–1858, 2017.

[211] Zhichao Yin and Jianping Shi. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1983–1992, 2018.

[212] Nan Yang, Lukas von Stumberg, Rui Wang, and Daniel Cremers. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry, 2020.

[213] Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. Vinet: Visual-inertial odometry as a sequence-to-sequence learning problem. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[214] Liming Han, Yimin Lin, Guoguang Du, and Shiguo Lian. Deepvio: Self-supervised deep learning of monocular visual inertial odometry using 3d geometric constraints, 2019.

[215] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[216] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[217] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. *arXiv preprint arXiv:1607.02555*, 2016.

[218] Bernd Pfrommer, Nitin Sanket, Kostas Daniilidis, and Jonas Cleveland. Penncosyvio: A challenging visual inertial odometry benchmark. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3847–3854. IEEE, 2017.

[219] Jeffrey Delmerico and Davide Scaramuzza. A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2502–2509. IEEE, 2018.

[220] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[221] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[222] Kirk YW Scheper and Guido CHE de Croon. Evolution of robust high speed optical-flow-based landing for autonomous mavs. *Robotics and Autonomous Systems*, 124:103380, 2020.

[223] Chen-Hsuan Lin and Simon Lucey. Inverse compositional spatial transformer networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[224] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[225] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[226] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[227] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[228] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[229] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for image restoration with neural networks. *IEEE Transactions on computational imaging*, 3(1):47–57, 2016.

[230] D. Sun et al. A quantitative analysis of current practices in optical flow estimation and the principles behind them. *International Journal of Computer Vision*, 106(2):115–137, 2014.

[231] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[232] J. Barron. A general and adaptive robust loss function. *CVPR*, 2019.

[233] Sebastian OH Madgwick, Andrew JL Harrison, and Ravi Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *2011 IEEE international conference on rehabilitation robotics*, pages 1–7. IEEE, 2011.

[234] Sujith Ravi. Projectionnet: Learning efficient on-device deep networks using neural projections. *arXiv preprint arXiv:1708.00630*, 2017.

[235] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[236] B Srinivasa Reddy and Biswanath N Chatterji. An fft-based technique for translation, rotation, and scale-invariant image registration. *IEEE transactions on image processing*, 5(8):1266–1271, 1996.

[237] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.

[238] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[239] Deepak Geetha Viswanathan. Features from accelerated segment test (fast). *Homepages. Inf. Ed. Ac. Uk*, 2009.

[240] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *2011 International conference on computer vision*, pages 2548–2555. Ieee, 2011.

[241] Christopher G Harris, Mike Stephens, et al. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[242] Matthias Faessler, Davide Falanga, and Davide Scaramuzza. Thrust mixing, saturation, and body-rate control for accurate aggressive quadrotor flight. *IEEE Robotics and Automation Letters*, 2(2):476–482, 2016.

[243] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 224–236, 2018.

[244] Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7244–7251. IEEE, 2018.