

ABSTRACT

Title of Document: AUTONOMOUS TARGET RECOGNITION
AND LOCALIZATION FOR MANIPULATOR
SAMPLING TASKS

Michael Pearson Naylor, M.S., 2006

Directed By: Assistant Professor Ella Atkins
Department of Aerospace Engineering

Future exploration missions will require autonomous robotic operations to minimize overhead on human operators. Autonomous manipulation in unknown environments requires target identification and tracking from initial discovery through grasp and stow sequences. Even with a supervisor in the loop, automating target identification and localization processes significantly lowers operator workload and data throughput requirements.

This thesis introduces the Autonomous Vision Application for Target Acquisition and Ranging (AVATAR), a software system capable of recognizing appropriate targets and determining their locations for manipulator retrieval tasks. AVATAR utilizes an RGB color filter to segment possible sampling or tracking targets, applies geometric-based matching constraints, and performs stereo triangulation to determine absolute 3-D target position.

Neutral buoyancy and 1-G tests verify AVATAR capabilities over a diverse matrix of targets and visual environments as well as camera and manipulator configurations. AVATAR repeatably and reliably recognizes targets and provides real-time position data sufficiently accurate for autonomous sampling.

AUTONOMOUS TARGET RECOGNITION AND LOCALIZATION FOR
MANIPULATOR SAMPLING TASKS

By

Michael Pearson Naylor

Thesis 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
Master of Science
2006

Advisory Committee:
Assistant Professor Ella Atkins, Chair / Advisor
Associate Professor David Akin
Assistant Professor Sean Humbert

© Copyright by
Michael Pearson Naylor
University of Maryland at College Park
2006

Acknowledgements

First I would like to thank all of the faculty, staff and other students at the Space Systems Laboratory that have provided their help and support over the last few years. Thanks goes to my advisor, Dr. Ella Atkins, for her help and guidance during my years as both an undergraduate and a graduate student. To Dr David Akin and Dr. Sean Humbert for providing great feedback on my research as well as possible avenues for future application of the software.

Many thanks to Stephen Roderick, without whom more than half of this thesis would have been impossible. He patiently provided countless hours of assistance whenever necessary.

To all the graduate students at SSL – thanks! Emily, you have had to put up with me almost continuously for the last 4 years. Nick, lets raise a cheer to the now infamous Ducky Video. And to the other students, past and present, thanks for the discussions, assistance and side projects throughout the years. To the number 32!

A special thanks goes to Dr. Hanumant Singh from Woods Hole Oceanographic Institution for not only providing me with underwater imagery from his SeaBED AUV, but also giving me the opportunity to work at WHOI for a summer as well as take part in a SeaBED cruise in Puerto Rico.

Finally, I want to recognize all my family and friends who have supported me through over 20 years of being a student, without them all of this would have been impossible.

Thanks everybody!

Table of Contents

<i>Acknowledgements</i>	<i>ii</i>
<i>Table of Contents</i>	<i>iii</i>
<i>List of Figures</i>	<i>vi</i>
<i>List of Tables</i>	<i>viii</i>
<i>List of Abbreviations</i>	<i>ix</i>
CHAPTER 1 INTRODUCTION	1
1.1 Motivation.....	1
1.2 Problem Statement.....	5
1.3 Approach	6
1.4 Contributions.....	9
1.5 Thesis Structure	10
CHAPTER 2 COMPUTER VISION BACKGROUND	11
2.1 Camera Calibration and Camera Model.....	12
2.2 Lighting Correction	15
2.3 Feature Extraction	17
2.3.1 Edge Detection.....	18
2.3.2 Intensity Based Segmentation.....	20
2.4 Stereo Correspondence.....	24
2.4.1 Point and Feature Correlation	24
2.4.2 Epipolar Geometry	27
2.4.3 Image Rectification.....	29
2.4.4 3-D Reconstruction	31
CHAPTER 3 VISION ALGORITHMS	32
3.1 Calibration.....	32
3.1.1 Camera Intrinsic and Extrinsic Calibration.....	32
3.1.2 Camera-Manipulator Registration	33
3.2 Lighting Correction	36
3.2.1 Frame-Averaging	37
3.2.2 Lighting Pattern Estimation	38
3.3 Feature Extraction	42
3.3.1 Filter Creation	43
3.3.2 Application of the Color Filter.....	45
3.3.3 Feature Extraction.....	47

3.4	Stereo Correspondence.....	53
3.4.1	Feature Matching	53
3.4.2	3-D Reconstruction for Target Position	58
3.5	Visual Servoing.....	60
3.5.1	Visual Servo Algorithm	60
3.5.2	Minor Visual Servo Functions	63
3.6	Management of anomalies, occlusions, and poor visibility conditions. 65	
3.6.1	Poor Visibility	65
3.6.2	Occluded Targets	66
CHAPTER 4 SOFTWARE DESIGN AND IMPLEMENTATION		68
4.1	System Architecture.....	68
4.1.1	DMU Sub-Architecture.....	69
4.1.2	Vision System Modules	72
4.2	AVATAR	73
4.2.1	Common.....	74
4.2.2	Acquire.....	75
4.2.3	Analyze	79
4.2.4	Config	82
4.3	TAU	84
4.3.1	VisionInterface.....	84
4.3.2	TAUNet.....	85
4.3.3	TAUUnit	87
4.3.4	TAUGUI	87
4.4	Visual Servo Controller	88
4.4.1	Visual Servo Software Integration.....	89
4.4.2	Visual Servo Control Law	91
4.5	Software Utilities.....	94
4.5.1	Custom Utilities	94
4.5.2	Software Engineering Tools	98
CHAPTER 5 EXPERIMENTAL PLATFORM AND TEST PLAN		106
5.1	Vision Hardware	106
5.1.1	Stereo Cameras	106
5.1.2	Camera Placement	108
5.1.3	Housings and Mountings	109
5.1.4	Transition to Deep-Sea Configurations	111
5.2	Ranger Manipulator System.....	113
5.2.1	Manipulator Configuration	113
5.2.2	Ranger Computer Architecture and Interface.....	117
5.2.3	Observed end effector positioning accuracy and resolution	118
5.3	Test Sequence	119

5.3.1	Camera and Manipulator Calibrations	119
5.3.2	System-Level Testing with a Visually Distinct Object.....	121
5.3.3	Evolution to Repeatable, Accurate Target Identification and Tracking	122
5.3.4	Visual Servo Testing.....	124
CHAPTER 6 TEST RESULTS		125
6.1	Ranger Tests with Low Resolution Cameras	125
6.1.1	Calibration Parameters.....	125
6.1.2	Vision System Accuracy.....	126
6.1.3	Vision System Precision	127
6.1.4	Overall System Behavior	130
6.2	1-G Ranger Tests with High-Resolution Cameras.....	131
6.2.1	Calibration Data	132
6.2.2	Automated Registration	132
6.2.3	Visual Servo Experiments.....	135
6.3	Increased Target Realism.....	138
6.3.1	Laboratory Tests	138
6.3.2	Underwater Imagery	144
CHAPTER 7 CONCLUSIONS AND FUTURE WORK		147
7.1	Conclusions.....	147
7.1.1	Vision Algorithms.....	147
7.1.2	Software Structure	148
7.1.3	Sampling Tasks.....	148
7.2	Future Work.....	149
7.2.1	Computer Vision.....	150
7.2.2	Software Architecture	151
7.2.3	ASTEP Manipulator.....	151
<i>Appendix A Relationship Between Essential and Fundamental Matrices with Camera Calibration Parameters</i>		<i>153</i>
<i>Appendix B Derivation of the Registration Algorithm</i>		<i>154</i>
<i>Appendix C Full Algorithm for Extracting Feature Geometric Data.....</i>		<i>158</i>
<i>Appendix D CAD Drawings for the Internal Camera Mounts</i>		<i>161</i>
<i>Appendix E Results from IEEM Tracking</i>		<i>164</i>
<i>Bibliography</i>		<i>166</i>

List of Figures

Figure 1-1: Map of the east Arctic and Gakkel Ridge (Picture from [1]).....	3
Figure 1-2: CAD model of JAGUAR from I-DEAS	4
Figure 1-3: SAMURAI Robotic Manipulator.....	5
Figure 1-4: System Component Diagram	6
Figure 1-5: System CAD Model	7
Figure 2-1: Low-light image corrected by WHOI lighting algorithm [12]	16
Figure 2-2: BG Bayer Pattern	17
Figure 2-3: Sample edge detection performed using MATLAB	19
Figure 2-4: Original grayscale and B/W thresholded images with highlighted histogram.....	21
Figure 2-5: Result of different texture-based segmentation algorithms [23].....	22
Figure 2-6: Output from OpenCV corner detection algorithm	27
Figure 2-7: Epipolar geometry of a stereo camera system	28
Figure 2-8: Geometric representation of rectified images	29
Figure 2-9: Rectified images from a calibrated camera pair.....	30
Figure 3-1: Algorithm to determine camera-manipulator registration	36
Figure 3-2: Algorithm to determine frame-average correction ratios.....	38
Figure 3-3: Algorithm for extracting lighting correction data	40
Figure 3-4: Algorithm for creating a lighting pattern template image.....	41
Figure 3-5: Side-by-side comparison of an image corrected for the lighting pattern.	42
Figure 3-6: Algorithm for extracting feature ratio values.....	44
Figure 3-7: Sample plots showing Red/Blue ratio data for rubber ducky target (left) and uncorrected sand dollar (right)	45
Figure 3-8: Algorithm for RGB ratio color filter.....	46
Figure 3-9: RGB ratio filtering process	47
Figure 3-10: Algorithms used to extract feature raw data	49
Figure 3-11: Perimeter points of feature shown in sample feature match	50
Figure 3-12: Ineffective filter of Ranger's IEEM and end effector	51
Figure 3-13: Algorithm for extracting feature geometric properties	53
Figure 3-14: Algorithm to calculate feature shape vectors	54
Figure 3-15: Algorithm to match features by shape estimates	55
Figure 3-16: Algorithm to create a geometric based possible match list.....	57
Figure 3-17: Sample correlated images used for geometric match testing.....	58
Figure 3-18: Algorithm to calculate 3-D target position via stereo triangulation.....	59
Figure 3-19: Algorithm for arm motion through visual servo	63
Figure 4-1: Overview of AUV system architecture.....	68
Figure 4-2: DMU Software Architecture UML Diagram	71
Figure 4-3: Vision System Overview	73
Figure 4-4: UML Class Diagram of the AVATAR Acquire Module.....	75
Figure 4-5: UML Class Diagram for AVATAR Analyze Module	80
Figure 4-6: TAUTUI Interface to AVATAR.....	88
Figure 4-7: Visual Servo Controller State Machine	90
Figure 4-8: Ranger Control Loop	92

Figure 4-9: Open-Loop Visual Servo Diagram	93
Figure 4-10: filtergui program in the process of creating a target filter	98
Figure 4-11: Colored output of gcov results for each specific file	102
Figure 5-1: Cameras used for AVATAR testing	107
Figure 5-2: DeepSea SSC-5000 Camera Housing	110
Figure 5-3: Internal Camera Mount Assembled and Disassembled	111
Figure 5-4: SSL Manipulator Test Frame Attached to WHOI Sled	112
Figure 5-5: D-H Parameters and Frame Assignments of the Ranger Manipulator... 115	
Figure 5-6: Inverse kinematics flowchart for the Ranger manipulator [53].	116
Figure 5-7: Ranger configurations in 1-G and neutral buoyancy	122
Figure 5-8: Images of Deep Water Sampling Target Fields	123
Figure 5-9: Ranger and AVATAR configuration for visual servo testing.....	124
Figure 6-1: Difference in Z reconstruction due to single pixel error	129
Figure 6-2: Ranger successfully grabbing a target in both testing environments.....	131
Figure 6-3: Upper Arm Joint Angles During Visual Servo Simulation.....	136
Figure 6-4: Wrist Joint Angles During Visual Servo Simulation	137
Figure 6-5: Cartesian position and orientation during visual servo simulation	137
Figure 6-6: Blue vs. Green ratio data for light laboratory targets.....	139
Figure 6-7: Red vs. Blue ratio data for light laboratory targets.....	140
Figure 6-8: Red vs. Green ratio data for light laboratory targets.....	140
Figure 6-9: Realistic targets easily segmented in lighted laboratory environment..	141
Figure 6-10: Blue vs. Green ratio data for dark laboratory targets.....	142
Figure 6-11: Red vs. Blue ratio data for dark laboratory targets.	142
Figure 6-12: Red vs. Green ratio data for dark laboratory targets.....	143
Figure 6-13: Some targets extracted in darkened laboratory environment.....	144
Figure 6-14: Red vs. Blue ratio data for sand dollar images.	145
Figure 6-15: Red vs. Green ratio data for sand dollar images.	145
Figure 6-16: Sand dollar targets extracted from color attenuated image.....	146

List of Tables

Table 3-1: List of Feature Shape Points.....	50
Table 6-1: Camera calibration data for Sony camera testing.....	125
Table 6-2: Target Acquisition System Accuracy Data	127
Table 6-3: Vision System Precision Data	128
Table 6-4: Calibration data for testing with Scorpion cameras	132
Table 6-5: Vision System Precision with High Resolution Cameras	134
Table 6-6: Hand-Eye Registration Precision Results.....	134
Table 6-7: Difference of Arm Telemetry and Transformed Vision Coordinates	135

List of Abbreviations

ASTEP – Astrobiology Science and Technology for Exploring Planets
AUV – Autonomous Underwater Vehicle
AVATAR – Autonomous Vision Application for Target Acquisition and Ranging
CAD – Computer Aided Design
CCD – Charge-Coupled Device
DMU – Data Management Unit
DOF – Degree of Freedom
FOV - Field of View
GUI – Graphical User Interface
HSV(I) – Hue-Saturation-Value (or Intensity)
IEEM – Interchangeable End Effector Mechanism
JAGUAR – Just Another Great Underwater Autonomous Robot
LED – Light Emitting Diode
NASA – National Air and Space Administration
OpenCV – Open Computer Vision Library
PD – Proportional-Derivative
RGB – Red-Green-Blue
ROV – Remotely Operated Vehicle
SAMURAI – Sub-sea Arctic Manipulator for Underwater Retrieval and Autonomous Interventions
SEW – Shoulder-Elbow-Wrist
SIP – Stereo Image Pair
SSL – Space System Laboratory
STL – Standard Template Library
TAU – Target Acquisition Unit
TUI – Text User Interface
UML – Unified Modeling Language
XML – eXtensible Markup Language
WHOI – Woods Hole Oceanographic Institution

Chapter 1 Introduction

Increased levels of robotic system autonomy will enable scientific exploration in previously unreachable destinations. This thesis focuses on the acquisition and tracking of sampling targets for a dexterous robotic manipulator using a computer vision system. The system developed for this thesis is designed for an autonomous underwater vehicle (AUV) operating at a depth of 5000m where untethered human teleoperation is impossible. Although targeted for an AUV, the system is generalizable to other operations underwater, in space, or on planetary surfaces.

1.1 Motivation

Reliable and capable autonomous robotic systems are in great demand for exploration in harsh, inaccessible environments. Development of such systems will allow for greater scientific return on missions where ground support, communications, and operator workload are prohibitive in terms of cost and factors such as time delay or communication bandwidth constraints. Enhanced robotic perception of the environment is a key enabler to reduced human interaction. Tasks utilizing robotic manipulators are notorious for the strain placed on human operators, both mentally and physically. Lack of sufficient camera views during teleoperation, hand strain from long-term use of hand controllers, and mental stress associated with difficult teleoperation tasks are all challenges that can be mitigated through effective automation.

A robotic manipulator, SAMURAI, the Sub-sea Arctic Manipulator for

Underwater Retrieval and Autonomous Interventions, is being developed at the University of Maryland's Space Systems Laboratory (SSL) as a combined research effort with Woods Hole Oceanographic Institution (WHOI) for NASA's Astrobiology Science and Technology for Exploring Planets (ASTEP) program. The manipulator will be attached to an AUV under development at WHOI and used for autonomous sampling missions at a depth of 5000m in the Arctic Ocean. This mission must be conducted with full autonomy due to shifting ice sheets that make a continuous high-speed communications tether infeasible. Because of their versatility and presence as a primary science instrument, mission organizers chose a stereovision system as the perceptive means for locating sampling targets.

The culmination of the ASTEP project is a field expedition to the Gakkel Ridge in the ice-covered Eastern Arctic Basin shown in Figure 1-1. Located in this area is evidence of hydrothermal activity found during the joint US-German AMORE 2001 icebreaker expedition. This environment is one of the last truly unexplored regions on Earth due to its inaccessibility under the icecap. The unknown and harsh Gakkel characteristics provide a realistic and productive terrestrial environment with analogue to space exploration missions. Most closely matched is a proposed mission to explore the oceanic aspects of Europa, requiring fully autonomous under-ice operation made even more difficult than Gakkel by the absence of a capable manned surface ship.



Figure 1-1: Map of the east Arctic and Gakkel Ridge (Picture from [1])

WHOI's JAGUAR AUV, shown through a CAD depiction in Figure 1-2, will be fitted with the SAMURAI manipulator, shown in Figure 1-3, to perform the desired undersea sampling tasks. Undersea manipulation has in the past exclusively been performed via teleoperation, except for simple low degree of freedom (DOF) grappling activities. From the science perspective, exploration of Gakkel has the potential to identify new life forms and vastly improve our understanding of undersea geology. From the engineering perspective, this mission will deploy the first fully autonomous undersea dexterous (6-DOF) manipulator, along with the first real-time undersea visual sampling target recognition system, the product from research described in this thesis. The culmination of the project will be the manipulator-AUV system, completely untethered, operating at 5000m depths.

The major requirements and constraints dictating system development are:

- Necessity of full autonomy due to depth and shifting ice sheets on the surface
- Near real-time target tracking to account for small target motion and/or AUV perturbations with frequency limited by the strobe light recharge cycle
- Simplicity of software architecture and vision algorithms to facilitate future research regardless of software familiarity or computer vision background
- Mission success defined by successfully sampling biological or geological target(s)

To perform the autonomous sampling tasks, an accurate sensory system must be present on the vehicle. A calibrated stereo camera pair will be affixed to the AUV to locate and extract sampling targets, representing them in an AUV body frame from which the manipulator end effector has known offset and attitude. This thesis develops and tests the stereovision system known as AVATAR, the Autonomous Vision Application for Target Acquisition and Ranging.

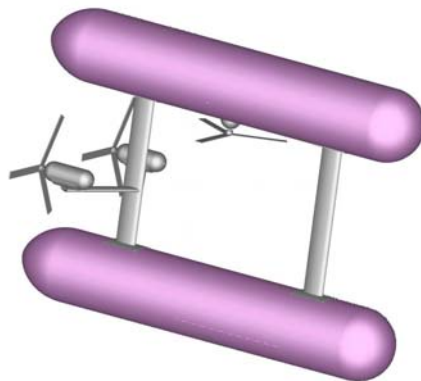


Figure 1-2: CAD model of JAGUAR from I-DEAS



Figure 1-3: SAMURAI Robotic Manipulator

1.2 Problem Statement

The goal of this thesis is to develop and verify in a hardware-based test environment a computer vision system capable of autonomously determining accurate real-world positions of targets for manipulator sampling tasks. This task is made more challenging than traditional factory automation tasks by a variety of factors. First, poor lighting conditions in the deep-sea environment require substantial image pre-processing and filtering. Also, targets may be mobile and their precise visual characteristics and locations are unknown a priori. A stereovision system must be calibrated and rigorously tested within an evolving software architecture able to satisfy all goals associated with the autonomous sampling tasks for the ASTEP mission. The software should be sufficiently flexible to function with different

camera setups, in vastly different environments, and with a wide array of targets. Additionally, the vision system should be independent of manipulator kinematic configuration so it can provide valid data when combined with any robotic arm.

1.3 Approach

The goal of successful autonomous undersea sampling relies heavily upon economical use of available computing power via efficient and correct algorithms for perception, planning, and control. Three computers will be present on the JAGUAR-SAMURAI vehicle: the WHOI AUV computer, the SAMURAI Data Management Unit (DMU), and the vision computer where AVATAR resides. Each of these computers has specific real-time constraints in which it must perform critical tasks. Figure 1-4 shows a UML component diagram of the relevant system hardware, and Figure 1-5 is a CAD representation of the AUV with attached manipulator and cameras.

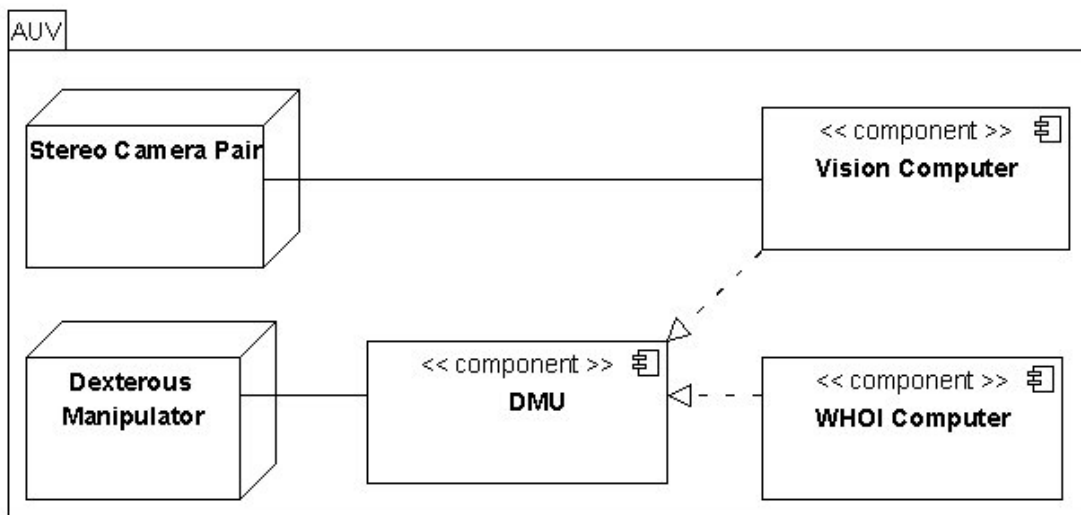


Figure 1-4: System Component Diagram

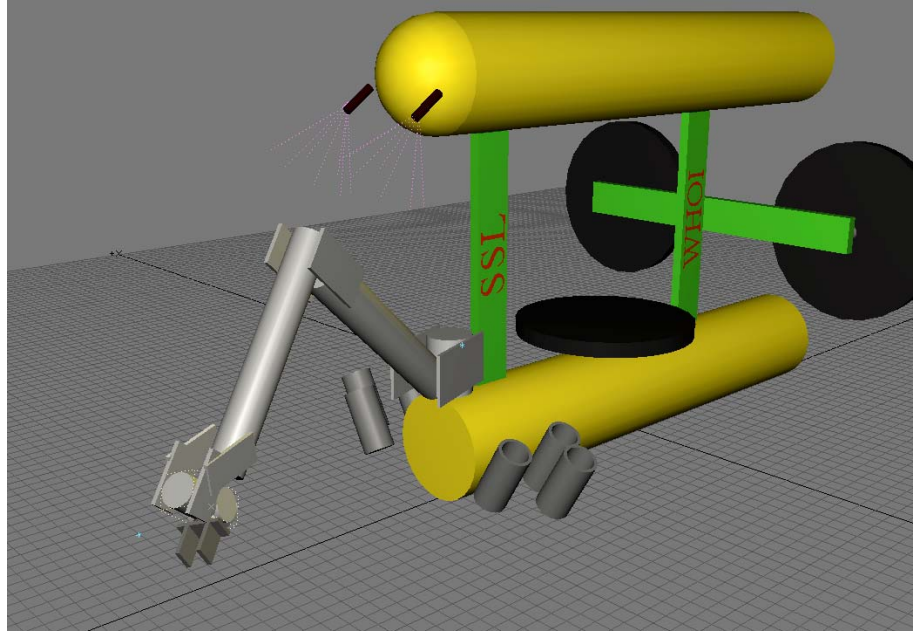


Figure 1-5: System CAD Model

Vision Computer

The vision computer contains two primary software modules. The first is the vision system software (AVATAR) that performs image acquisition, target filtering, and target extraction and 3-D localization tasks in sequence. The second is an interface to AVATAR known as the Target Acquisition Unit, or TAU. AVATAR applies a series of algorithms to a pair of raw images to extract 3-D positions of the targets:

- Lighting and color correction, dependent on environment
- Color based pixel-by-pixel filter dependent on sampling target type
- Feature data extraction, also dependent on target type
- Matching of features for stereo correspondence
- Triangulation of 3-D target coordinates

Given data acquisition and processing overhead, AVATAR has a frequency of 1Hz in a laboratory environment with ample lighting. During AUV operations, image acquisition frequency will be limited by the recharge cycle of the strobe, about 2.5 seconds, so lighting becomes the limiting factor for target updates. TAU is designed to remove all external interface considerations from AVATAR and ensure that changes in the rest of the system do not propagate through AVATAR, and vice-versa. The different implementations of TAU provide both front-end and back-end interfaces with AVATAR, as well as simulation modes that require no vision-related hardware. Communication between the DMU and the vision computer is achieved through the back-end TAU interface located on the vision computer.

Data Management Unit (DMU)

The DMU houses all software for real-time control of SAMURAI as well as the task and motion planners that select from the suite of identified sampling targets and maneuver the vehicle and manipulator to collect each sample. At predetermined points in the ASTEP mission, while the AUV is located on the ocean floor, the DMU takes control from the WHOI computer. Based on data received from the vision computer, through a local TAU front-end interface, as well as prior knowledge of possible sampling sites, the DMU will drive the AUV to an appropriate position to attempt target sampling tasks. The DMU operates at a frequency of 125Hz, which sets the update rates for all trajectories and relevant data loggers.

WHOI Computer

JAGUAR is based on an earlier version of a similar WHOI AUV called SeaBED [2],

the software for which will be ported to JAGUAR. The WHOI AUV computer consists of two main parts, a mission execution script, written in Perl, and the main AUV control code and device drivers written in C, which communicate via sockets. The mission script vs. controller split was implemented to help separate the low-level data acquisition and control tasks from the higher-level mission planning and goal tasks [2]. The vehicle control loop runs at 10Hz, while all core navigation data is logged at 5Hz. Due to limitation with the strobe charging time, the maximum rate images can be acquired and written to the hard drive is 2.5s.

1.4 Contributions

The applied computer vision contributions from this thesis of direct relevance to the ASTEP mission are:

- Development of computer vision algorithms to extract 3-D positions of desired sampling or tracking targets
- Design of software to perform autonomous target extraction and provide robust public interfaces for laboratory testing in addition to autonomous operations
- Validation of software through comprehensive use of unit testing and integration within a system utilizing daily builds with continuous integration
- Rigorous laboratory testing of stereo vision algorithms and software with the Ranger manipulator in 1-G and underwater environments to prove target localization capabilities and software stability
- Implementation of a visual servo controller that tracks both manipulator and

sampling target to reduce sensitivity to camera calibration errors and external disturbances

- Additional testing with low-light and color attenuated images with realistic targets in preparation for the transition to AUV operations

1.5 Thesis Structure

This thesis overviews the research efforts of the SSL-WHOI team to develop the ASTEP AUV system, then focuses on the computer vision and software design specifics that comprise the research component of this thesis. To provide initial background, Chapter two reviews key aspects of computer vision related to this research, as well as providing insight into other contemporary computer vision systems. Chapter three describes the computer vision algorithms integrated into the final AVATAR system, ranging from calibration and target recognition algorithms to visual servoing. Chapter four focuses on the software design and implementation of the entire vision system, also describing the tools and evaluation methods used to validate the software. Chapter five discusses the test design and implementation for AVATAR, including the hardware, testing environments, assumptions and limitations associated with both the vision-related and manipulator hardware. The test results are discussed in Chapter six, and finally conclusions and future work are presented in Chapter seven.

Chapter 2 Computer Vision Background

State of the art stereovision systems are capable of accurately characterizing three-dimensional environments given accurate calibration and image processing algorithms tuned to the environment and task to be accomplished. Many elegant domain-specific solutions have been developed, yet creation of a fully-functional, generalized stereo system is still far from realization. Most stereo systems require four major modules: calibration, feature extraction, stereo correspondence, and 3-D reconstruction. The complexity of each module depends on a priori knowledge of physical parameters of the system, uniqueness of targets in the field of view, object motion and variance over time, and lighting conditions. Accepted approaches to each of these problems will be discussed in this chapter to provide background and motivation for the vision algorithms selected in this work.

Many of the algorithms discussed throughout this chapter come from existing software libraries. Software implemented in MATLAB makes heavy use of both the Camera Calibration Toolbox [3] as well as the built-in Image Processing Toolkit. Algorithms implemented in C and C++ make use of the Open Computer Vision Library [4] (OpenCV) for image handling as well as other core tasks. Lighting correction background research is courtesy of Dr. Hanumant Singh at WHOI who also provided access to a MATLAB lighting correction algorithm and a library of uncorrected, raw images from various SeaBED cruises.

2.1 Camera Calibration and Camera Model

The primary factor that determines the overall accuracy of a stereo system is system calibration accuracy. Full calibration of a stereo system requires precise characterization of two parameter sets: intrinsic and extrinsic. Intrinsic calibration refers to determination of internal properties of each camera, including focal length and piercing point, while extrinsic calibration refers to the physical relationship of the left camera to the right camera, a rotation matrix ${}^R_L R$ that rotationally aligns the left camera with respect to the right camera, and a translation vector ${}^R_L \mathbf{t}$ that describes the Cartesian difference between the cameras.

Knowledge of both intrinsic and extrinsic parameters enables unambiguous calculation of 3-D coordinates for any matched points. The calibration procedure and camera model utilized for this research are from the Camera Calibration Toolbox for MATLAB [3], which is based on the work of Heikkilä and Silvén [5]. The first step of the calibration procedure is acquiring synchronized sets of “checkerboard images” (i.e. images with a fully visible checkerboard pattern). After the user provides data on the size and dimensions of the pattern, the software attempts to extract all corner points of the pattern in each image, with the user supplying the outside corners as well as an initial guess for distortion. Once the checkerboard patterns have been recorded, the main calibration algorithm runs a non-linear least squares gradient descent algorithm to optimize all of the parameters. After this intrinsic calibration has been applied to both cameras, the extrinsic parameters can be determined via the correspondence between checkerboard patterns in the synchronized images.

The final aspect of calibration related to this research is the determination of the camera-manipulator registration, also known as hand-eye calibration [6][7], between the manipulator and the vision system. For a successful grasp, the manipulator controller requires object position data to be provided in a known frame of reference, which an arbitrarily positioned vision system does not provide. For the calculated 3-D target positions to be useful, they must first be transformed into the manipulator frame of reference.

Intrinsic Calibration

The intrinsic parameters used in [3] consist of:

- $\mathbf{f}_c = \begin{bmatrix} f_x \\ f_y \end{bmatrix}$, the two-dimensional focal length in millimeters
- $\mathbf{c}_c = \begin{bmatrix} c_x \\ c_y \end{bmatrix}$, the principal point (center of image) expressed in pixels
- α_c , the skew coefficient (angle between image x-axis and y-axis)
- \mathbf{k}_c , a vector of five numbers describing both radial and tangential distortion

The focal distance \mathbf{f}_c is a vector with two elements f_x and f_y that represent a unique value in *mm* expressed in units of horizontal and vertical pixels. If the camera has square pixels in the CCD array, these two values should be very close. On the other hand, if the pixel elements on the CCD are rectangular, the ratio of f_x to f_y will not be close to 1 – this is referred to as the “aspect ratio”. Since this model takes into account the variation in horizontal and vertical pixel size, it can handle the general case of non-square pixels. The principal point \mathbf{c}_c defines the position of the camera

center in pixels, which is used when performing transformations between image plane and real world coordinates. The skew coefficient α_c is the angle between the x and y pixel axes, and by including this value in the camera model, the case where an image has non-rectangular pixels can be handled. The vector of distortion coefficients \mathbf{k}_c contains the coefficients used in a 6th order non-linear distortion model known as the “Plumb Bob” model developed by Brown in 1966 [8].

Extrinsic Calibration

The extrinsic parameters of a stereo system describe the relative position and orientation of the cameras – for this research the extrinsic calibration is defined as the translation and rotation of the right camera with respect to the left camera. Once again using the Camera Calibration Toolbox for MATLAB, these values can be computed with knowledge of the intrinsic parameters and corresponding images of checkerboard patterns. With previous knowledge of the intrinsic parameters combined with the entire set of point matches of corners of checkerboards, the MATLAB toolbox will output the set of all calculated translations and rotations while determining the overall translation vector ${}^R\mathbf{t}_L$ and rotation vector ω , as defined by the Rodrigues Rotation Formula [9]. The rotation matrix ${}^R_L R$ can then be calculated using the `rodrigues` function provided in the toolbox.

Camera-Manipulator Registration

Many methods exist to uniquely determine the transformation between a manipulator frame of reference and the camera system frame of reference [6][7]. Without an

accurate transformation, the data acquired from the vision system will be useless, although implementing a visual servo controller can help alleviate problems with inaccuracies [10][11]. As with all transformations in 3-D space, the camera-manipulator relationship is defined by a rotation and translation of one coordinate system onto another.

Most literature focuses on placement of the camera system at the wrist of the manipulator as this allows the cameras to move with the same degrees of freedom as the manipulator [6][7]. In this research, the situation is slightly different as it is desirable for the cameras to sense the full manipulator workspace at all times, rather than strictly view the area immediately ahead of the manipulator end effector. Since the cameras are fixed relative to the robot base, the desired transformation is thus between camera frame and manipulator base frame. This transformation is calculated through a process of tracking corresponding points known in both camera frame and robot base frame, and performing an algorithm on a set of corresponding points to extract the rotation and translation. These corresponding points are obtained by having the vision system track a distinct object on the manipulator, while recording the vision frame coordinates from the stereo analysis and robot base frame coordinates determined from encoder telemetry. The algorithm implemented for this research is discussed in Section 3.1.2.

2.2 Lighting Correction

One of the major concerns when dealing with high-depth imagery is the lack of ambient light. Given tetherless AUV power restrictions, intense floodlights cannot be

used to illuminate the entire scene. Instead, low-power strobes or LED arrays are the sole source of illumination. In such cases, lighting correction algorithms can be applied to make a dark image with high color attenuation appear as it would in ample light. An example is illustrated in Figure 2-1, a WHOI SeaBED image of the ocean floor populated by sea urchins.

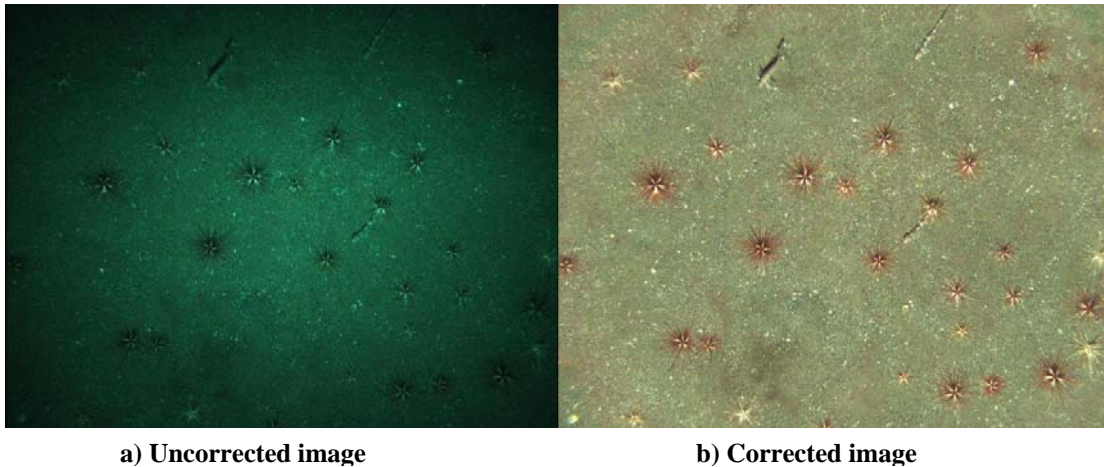


Figure 2-1: Low-light image corrected by WHOI lighting algorithm [12]

Application of such algorithms is absolutely necessary when human scientists analyze the images if the true color of targets is of importance. However, when performing computerized analysis to extract color-based features while using high bit-depth cameras, the benefit of lighting correction algorithms is not as clear. The first correction algorithm described was developed at WHOI and utilizes extensive knowledge of the cameras and water chemistry to extremely accurately correct for light attenuation [12]. Unfortunately, this algorithm operates with a calculation time of approximately 10s in MATLAB, which renders it useless for near real-time applications. Other algorithms exist that attempt to correct for color attenuation with faster methods, but the quality of results tend to decrease rapidly as the algorithm

becomes simpler and quicker.

Other important factors for color correction are the process by which an image is recorded and how the CCD array is constructed. Most digital cameras contain a Bayer pattern mosaic of photosensors to allow a single chip to record true color images. A Bayer pattern refers to the layout of photosensors on the CCD chip. For many scientific cameras, such as the Point Grey Scorpions used in this research, the raw imagery must first be converted from the grayscale Bayer image into an RGB image using one of a variety of algorithms. OpenCV implements many different methods of Bayer pattern correction for all different patterns of sensors. Figure 2-2 below shows the common BG pattern, as shown in the on-line documentation for OpenCV [13].

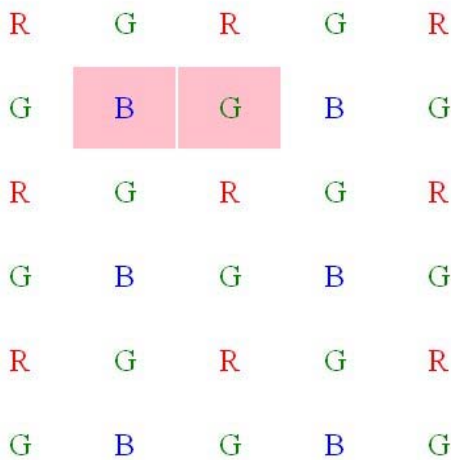


Figure 2-2: BG Bayer Pattern

2.3 Feature Extraction

The next major step in a stereo system is to extract distinguishing features from each image. Edge, shape, texture and color are factors that can be used to extract features

from a raw image. In many cases, a combination of methods is required to accurately segment each feature. For instance, an edge detector might be applied to the image before a shape detection algorithm is used to determine which remaining features correspond to a desired shape. In other algorithms, such as texture matching, an initial color filter can substantially limit the feature search space to decrease processing time. Four methods are discussed below in greater detail: edge detection, Hough transforms, color based segmentation, and Eigenspace identification. The method used for this research is color based segmentation due to its intuitive nature, simplicity of implementation, and immediate successes during initial testing.

2.3.1 Edge Detection

Edge detection algorithms have been developed and refined over the past 30 years, providing a mature toolset for image feature extraction [15]. There are two main approaches to edge detection: template matching and differential gradient. All routines, however, calculate a local intensity gradient and, based on the magnitude of that calculation, determine whether or not a specific pixel is part of an edge. The Canny edge detector is one of the most widely applied algorithms, but there are many others including the Sobel, Roberts and Prewitt methods [16]. The base behind all edge detection algorithms is the application of convolution masks – anywhere from just two masks, x and y , up to 12 for more complicated template matching detectors [15].

Most simple algorithms apply convolution masks that detect edges of a specific orientation, while more complicated strategies apply larger or multiple masks

to determine multi-orientation edges. The more robust edge detection algorithms are correspondingly more computationally intensive. For the general case where targets have unknown shape or orientation, many operators are required to segment full targets. In applications similar to this research, where near real-time execution is the goal, anything but the most simplistic algorithm is infeasible [15]. Figure 2-3 shows sample output from one of MATLAB's edge detection routines.

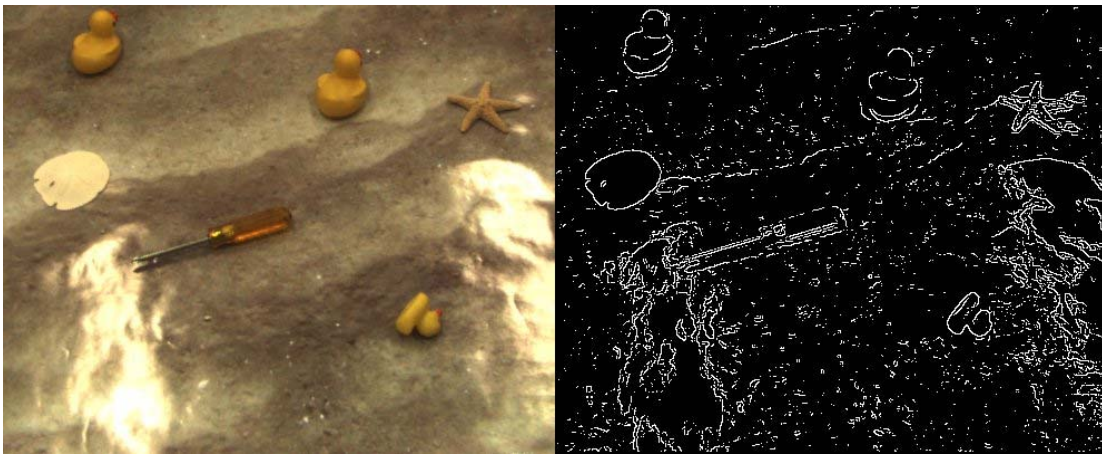


Figure 2-3: Sample edge detection performed using MATLAB

Many problems are immediately recognizable from this output. First, only the white sand dollar target has a completely formed boundary, and all the other targets would need to have additional processing to complete the shape. Also, reflections caused by lighting and sand texture show up as boundaries. Further processing is required to complete broken edges, remove linear edges and calculate all position data about the detected features. Algorithms exist to perform these necessary operations, such as variations on the Hough transform [17] used to locate different shapes in images, or the Euler spiral [18], but these only complicate the software.

2.3.2 Intensity Based Segmentation

Similar to edge detection, color based segmentation algorithms range from simple to complex. A simple algorithm, such as the one used for this research, examines the color properties at a single pixel location, while more complicated algorithms may take into account neighboring pixels, patterns, and textures. Similarly to edge detection methods, all but the simplest algorithms are orders of magnitude more complex than what can be used for real-time or near real-time applications. First, basic histogram segmentation will be detailed, followed by a more complicated texture-based method and finally the Eigenspace identification method.

Histogram Segmentation

Especially when applied to binary or grayscale images, histogram segmentation is an extremely simple, yet effective, tool for extracting features. In such images, foreground objects tend to lie in a different section of the image histogram from the background, thus selection of an optimal threshold value is fairly straightforward [17]. Figure 2-4 shows an example of a grayscale image with a threshold applied between the two peaks in the histogram. The highlighted portion of the histogram represents foreground values. This method works well when the grayscale values of foreground features and the background are sufficiently different to be segmented in this manner, but when dealing with color images containing many different foreground and background entities, more complicated methods must be used to accurately extract desired features.

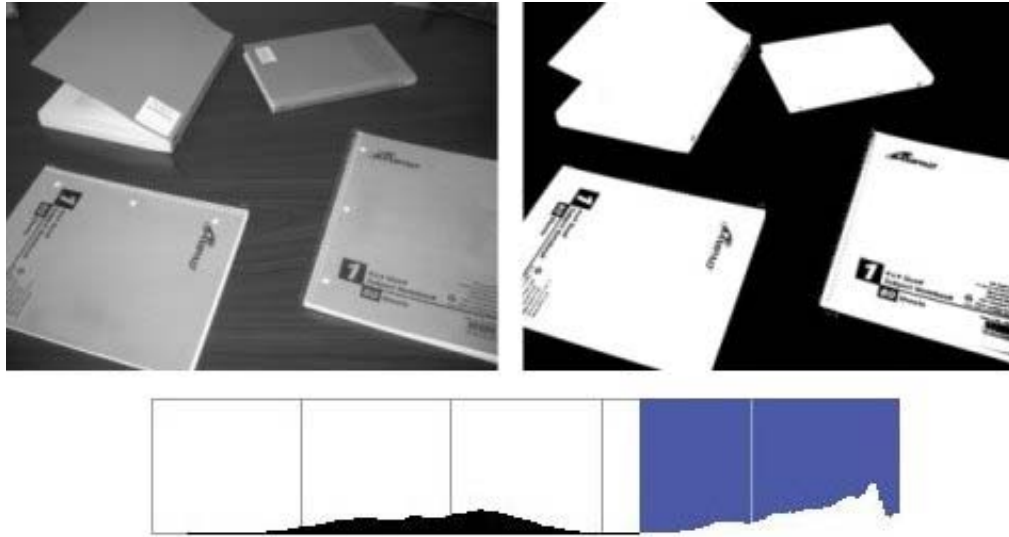


Figure 2-4: Original grayscale and B/W thresholded images with highlighted histogram

Multi-channel histogram segmentation is the extension of grayscale histogram segmentation and can achieve much better results in complex color images. Thresholding in this manner can be applied in either RGB or hue, saturation and value (or intensity) (HSV(I)) color-spaces. Depending on the visual properties of objects and the image background, working in one space or the other, or possible both, has benefits. RGB histograms tend to have little easily accessible data other than the fact that, as overall magnitude of a specific pixel increases, the RGB values also tend to increase. On the other hand, hue-oriented algorithms create much greater dispersion of peaks and separation of image regions while there is also the possibility of using a single value, hue, for segmentation. However, using a hue-oriented color scheme requires that each pixel be converted to another color-space, which can cause a large number of calculations when performed each iteration, although algorithms do exist to facilitate this conversion [19][20].

Texture-based Segmentation

When the single-pixel color data present in an image is not sufficient to effectively segment features of interest, the inclusion of texture-based segmentation methods can greatly increase the ability of an algorithm to distinguish desired objects. Texture segmentation algorithms can look at a wide array of information such as brightness ranges, spatial frequencies, and orientations [19]. Each texture type has many algorithms designed for extracting and labeling regions within an image, from simple thresholding through involved frequency domain analyses. Accurately segmenting a complex image into different regions quickly becomes a complicated algorithm, either in terms of mathematical understanding or computational complexity.

Despite the complexity issues, there are many successful applications of texture-based segmentation methods. Figure 2-5 shows example output from three different texture-based segmentation algorithms, as discussed in [21], with results also from [22][23].

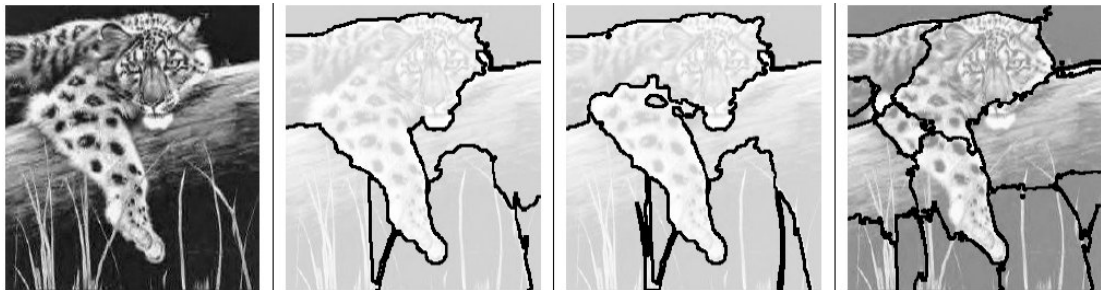


Figure 2-5: Result of different texture-based segmentation algorithms [23]

Eigenspace Identification

One final algorithm is used to ensure that, once a feature has been successfully segmented, the feature is actually what the algorithm is supposed to be locating,

regardless of size or orientation. There are many appearance-based algorithms, and one of the foremost is the Eigenspace Identification method [16][24]. This method consists of a learning algorithm applied over a set of existing images prior to an identification algorithm applied to new images.

The learning algorithm must be applied to an initial set of images containing the object in all desired recognizable poses, where the object is easily segmented from the background. If the object translates between successive images, or if lighting conditions change, the initial learning stage can become quite complicated, requiring an abundant amount of computing resources to analyze and store the learned data.

Each image must be represented as a vector, formed by scanning the image top to bottom and left to right and placing the results in a vector of length N^2 . By transforming an image into this representation, vector math can be utilized for more complicated image-based calculations, such as using the dot product for image correlation. Once all images have been converted to vectors, the learning algorithm continues by finding the average vector between all the images, creating a covariance matrix, computing the Eigenvalues and associated Eigenvectors and then finally calculating Eigenspace points for each image and storing the discrete Eigenspace curve as the representation of the segmented object.

Application of this data set to new images to recognize a desired object first requires all possible objects to be accurately segmented from the background before application of similar vector and Eigenspace calculations are applied to each object and a search performed within the library of data. All these steps must be performed after feature segmentation, and a search through a comprehensive library of possible

objects can be quite computationally complex. This project requires real-time object identification, so object recognition, particularly given uncertainty in object shape, etc., benefits more from compact and efficient strategies than from more comprehensive and elegant approaches such as Eigenspace Identification that require a prohibitive search over a database that may or may not accurately depict the objects to be sampled.

2.4 Stereo Correspondence

Use of a stereo camera system rather than a single camera requires identification of common features in the two cameras' image planes. Once features have been segmented within each corresponding image, they must be matched, initially on an overall feature level, and then through detailed corresponding points assigned to each feature that enable accurate 3-D object reconstruction. An overview of stereo correlation strategies is provided below, followed by a discussion of strategies to further increase accuracy and effectiveness of the algorithms, specifically use of Epipolar geometric constraints and the rectification process.

2.4.1 Point and Feature Correlation

The correlation process is composed of two main steps – feature correlation and point correlation. Feature correlation matches overall features between images while point correlation operates at the sub-feature level to accurately determine which points within a feature correspond to points in the other image's matched feature.

Feature Correlation

Feature-based correlation restricts the correspondence problem to a select few features extracted from an image. Unconstrained matching algorithms can become quite complicated, involving algorithms such as Eigenspace Identification described above, or any other weighting function designed to calculate a numerical value that represents each different feature. For example, a weighting function might include pixel area, various moments of the feature, and color measurements.

There are two main types of constraints that can be applied to sets of features to assist in the matching process: geometric and analytical [16]. Geometric constraints, such as Epipolar geometry, discussed below, greatly limit possible feature matches based solely on geometric knowledge from the camera setup. Analytical constraints are logic-based constraints such as the uniqueness constraint stating that each feature can only have one match and the continuity constraint that disparity must vary continuously throughout the image, barring odd scene geometry and occluded features.

If constraints are not placed on the matching algorithm, problems can arise when attempting 3-D reconstruction. Most notably is a scene with multiple objects with the exact same size, color and geometry. A weighting function could return values that result in incorrect matches since the features “look” exactly the same. However, by placing appropriate constraints on the system, many of these results can be eliminated from the start.

Point Correlation

A similar problem to that of feature matching is extracting points of interest within these features or in the general case an entire image, then successfully matching these points with one another. Without correct matches between these points, accurate matching of features is meaningless for further 3-D analysis. Most existing algorithms match points by determining the brightness pattern of the target pixel and its neighborhood with the other image in an attempt to find a similar pattern [19]. The initial points can be selected from a variety of algorithms that find points of interest based on available data, such as edges or corner features.

The most common points extracted from images are corner points. These points are located throughout complex scenes, either as geometric corners, such as on a building, or simple corners in patterns of intensities. Fortunately, these patterns remain visible in subsequent images, thus act as good choices for object tracking [16]. Based on image data of the neighborhood surrounding these corner points, similar algorithms can be used to match these points across corresponding images, or on a smaller scale, between already matched features. Figure 2-6 shows an example image after being run through one of OpenCV's corner detection algorithms. A small circle marks each detected corner.



Figure 2-6: Output from OpenCV corner detection algorithm

In the general case where numerous points are matched between images, a general depth map can be generated showing approximate distances to any point in the overlapping camera fields of view. Ideally when working with a single target feature, extraction and matching of points encompassing the entire feature provides data relating size and distance of the object to whatever degree necessary. More details on 3-D reconstruction are presented below in Section 2.4.4.

2.4.2 Epipolar Geometry

The geometry of stereo is known as epipolar geometry. Once calculated, the epipolar geometry of a system will map a point in one image to a line in the corresponding image. The outcome is essentially that the search for corresponding points and/or features is limited to a search along a known line rather than through the entire image. Figure 2-7 shows the epipolar geometry of a stereo system.

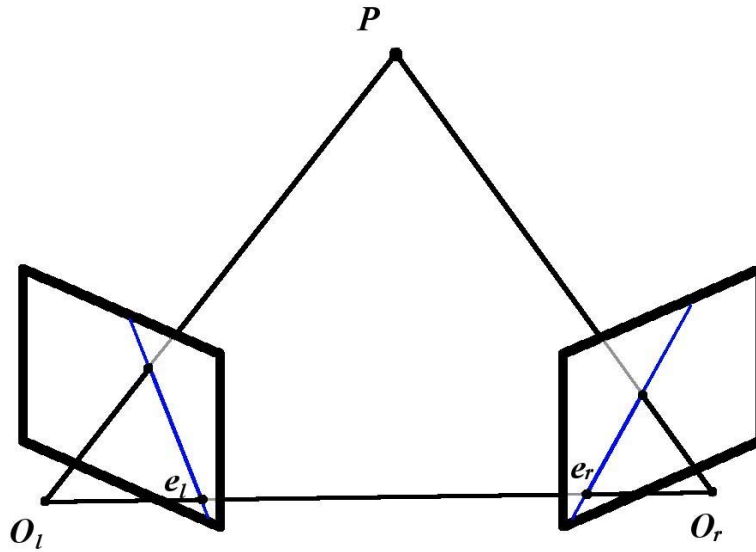


Figure 2-7: Epipolar geometry of a stereo camera system

Point P , the target point in each image, along with the origins of each camera, O_l and O_r , form the epipolar plane. The intersection of this plane with each of the image planes form a line known as an epipolar line, shown in blue. Each point in an image only has a single epipolar line traveling through it, except for the point known as the epipole, at which all epipolar lines intersect. The epipoles are denoted by e_l and e_r in Figure 2-7. As point P moves along the vector leading to O_r it stays in the same epipolar plane and the image coordinates in the right image remain constant, but in the left image the point slides along the epipolar line.

With knowledge of the intrinsic and extrinsic parameters of the stereo system from calibration, the Fundamental and Essential matrices of the system can be calculated to help determine geometric correspondence between the two images. The

Fundamental matrix relates pixel coordinate points between images and the Essential matrix relates camera coordinate points. As such, the Essential matrix is based solely on the extrinsic parameters of the system, while the Fundamental matrix is based on both extrinsic and intrinsic parameters, and is related to the Essential matrix by multiplications of the intrinsic parameter matrices of each camera. Appendix A shows the equations relating calibration parameters to the Essential and Fundamental matrices as well as the epipolar geometry.

2.4.3 Image Rectification

To perform general 3-D reconstruction of an entire scene it is useful to have all the epipolar lines of an image be collinear – in other words, the point correspondence problem is reduced even further to a simple search along a single scanline, as show in Figure 2-8. The gray boxes represent the rectified images, with the blue segments as the transformed epipolar lines parallel to the baseline.

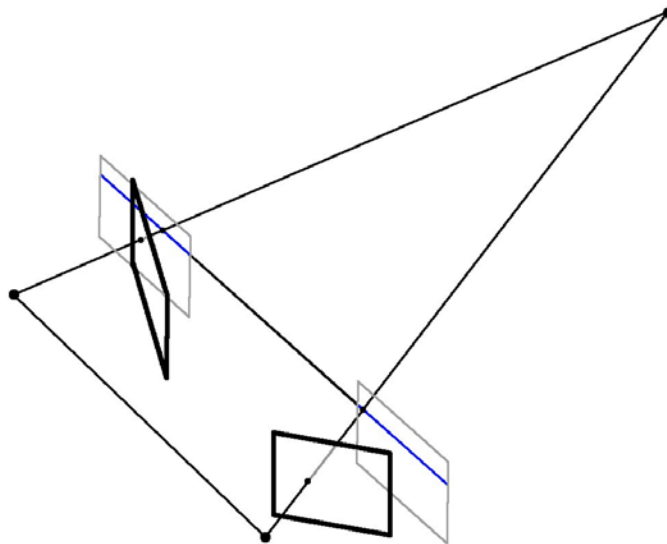


Figure 2-8: Geometric representation of rectified images

This effect is achieved by rotating the left camera so that the epipole goes to infinity along the horizontal axis, followed by rotating the right camera by the same amount (to recover original geometry) and then finally rotating the right camera again by the rotation matrix ${}^L_R R$ associated with the extrinsic parameters of the system. Once this has been completed, it may be necessary to adjust the scale in both camera reference frames.

Image rectification is possible regardless of the initial orientation and relative position of the cameras, assuming they do, in fact, share significant image overlap. Figure 2-9 shows example rectified images. Marked on the image are example epipolar lines showing how points anywhere in the scene of the left image lie upon the same scanline in the right image. By applying these geometric constraints to the pair of images, the matching problem becomes much easier, although the preparation becomes much more complicated.

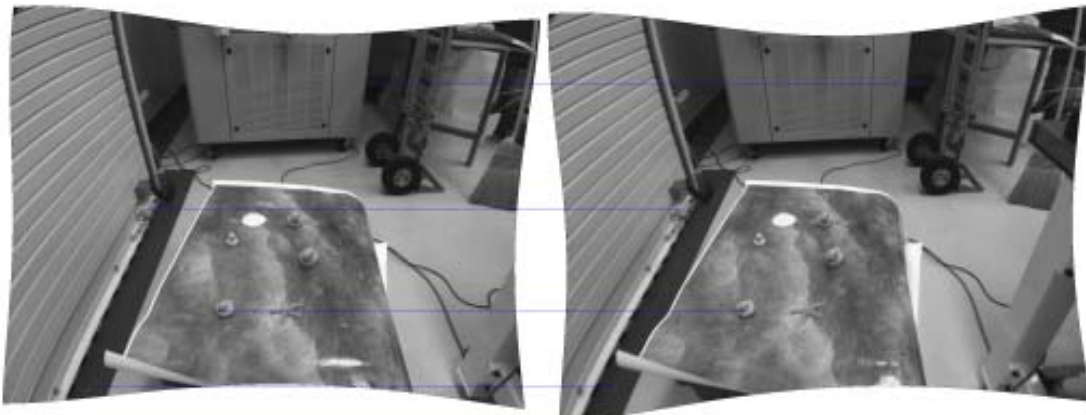


Figure 2-9: Rectified images from a calibrated camera pair

2.4.4 3-D Reconstruction

The ultimate goal of any stereoscopic system is some form of 3-D reconstruction, whether it is of the entire scene or localization of a single object. The ability to uniquely determine a 3-D position of image points is dependent on the knowledge of the intrinsic and extrinsic parameters of the system. There are three applicable cases of 3-D reconstruction. First, if both intrinsic and extrinsic parameters are known, any point in the scene can be determined unambiguously in all three dimensions by triangulation. Second, if only the intrinsic parameters are known, then reconstruction is possible up to an unknown scaling factor. The final case is if only the pixel correspondences are known, leading to reconstruction up to an unknown, global projective transformation [16]. The focus of this thesis is on the ability to absolutely determine three-dimensional coordinates in the first case, where all calibration parameters are presumed known.

Once an algorithm designed to calculate the 3-D position of a point or scene reaches the final triangulation step, the rest of the process is straightforward. The first step is to convert pixel coordinates into camera coordinates using the intrinsic parameters of the system. Once both left and right camera points are known in camera frame, the extrinsic parameters are applied to determine actual 3-D target position.

Chapter 3 Vision Algorithms

This chapter will describe and illustrate the algorithms that were implemented within the AVATAR software package in the order that they are used within the system. The first section will describe calibration and the camera-manipulator registration, followed by a discussion of feature segmentation and data extraction. Stereo correspondence procedures are described that enable feature matching between synchronized images with stereo triangulation for 3-D reconstruction. A visual servo controller is described next, followed by a section on management of anomalies and other problems that may arise.

3.1 Calibration

One of the most essential aspects of a computer vision system is to maintain an accurate set of calibration parameters. As described generally in Chapter 2, three main phases of calibration must be completed to accurately identify and sample targets from stereo camera feedback. First is the intrinsic calibration of each camera. The second step is determining the geometry of the camera system (extrinsic calibration). Finally, registration between the vision system and robotic manipulator must be performed to determine the transformation between vision system and manipulator base frame.

3.1.1 Camera Intrinsic and Extrinsic Calibration

The first calibration step is to mathematically estimate intrinsic camera parameters based on the correlation of unique features between 2-D image plane coordinates and

known 3-D world coordinates. As is standard practice in the vision community, a planar checkerboard pattern is used to provide a matrix of readily distinguished corner features. The checkerboard is presented to each camera at a series of different orientations and positions to provide a three-dimensional set of points for calibration. For accurate calibration, the size of each checkerboard box must be known and must be consistent across the calibration pattern. To identify the best intrinsic parameter set for each camera, a least squares gradient descent search minimizes re-projection error from 2-D to 3-D coordinates. It has been shown that the intrinsic parameter set can be accurately estimated with a minimum of five checkerboard images. For more information on this algorithm see [3][5], and for implementation details see [14][25][26]. Using the Matlab Camera Calibration Toolbox [3] and the results from intrinsic calibration, the extrinsic parameters of the stereo system can be determined based on the correlation of the calibration points between the two cameras. For this extrinsic calibration, synchronized stereo images with the same visible checkerboard pattern are required.

3.1.2 Camera-Manipulator Registration

The final calibration step is to determine the 3-D coordinate transformation between the vision system and manipulator base frame [14]. The algorithm implemented in this work operates on corresponding lists of n points from the vision (camera) and manipulator base frames, ${}^V\mathbf{P}$ and ${}^M\mathbf{P}$ respectively. Figure 3-1 shows the algorithm utilized to compute the transformation matrix M_VT that translates 3-D coordinates from stereo camera frame V to manipulator base frame M . First, the point cloud

center positions ${}^V C$ and ${}^M C$ are calculated by averaging all points. Next, each point list is normalized by the center to align both point clouds about the same center point. The rotation matrix ${}^M_v R$ is assigned to the identity matrix then computed through iteration.

Within the iterative loop, a series of rotations about single axes are applied to align the two point clouds. In each iteration, sequential rotations about the x-axis, y-axis, and then the z-axis are applied. Intuitively, this algorithm is iteratively applying rotations to “reverse” the point set rotation so that the “unrotated” point sets are as close to coincident as possible. The formulation presented here is based on multiple Z-Y-X Euler angle rotations [27]. First, the algorithm determines the angle γ in the Y-Z plane by which each point was rotated to reach its current position. The derivation for each case is shown in Appendix B. Assuming the magnitudes of corresponding points are the same and using $\sin^2 \gamma + \cos^2 \gamma = 1$, the results from these equations can be used to determine the sum of the squares of the magnitudes. Finally, by dividing these two results, the cosine and sine values can be calculated, and an initial rotation matrix can be formed. This rotation is then applied to the set of points from the vision frame, and the algorithm continues to the next rotation.

Once the iterative loop finishes, a final ${}^M_v R$ rotation matrix is now available to use in calculating the translation between the two frames, ${}^M \mathbf{t}_v$. This rotation is applied to the list of points in the vision frame, and then the average displacement between all corresponding points between the manipulator base frame and rotated vision frame is calculated and used for the translation. Results from implementation

with the Ranger manipulator are provided in Section 6.2.2.

Initialize point clouds ${}^V \mathbf{P}[n], {}^M \mathbf{P}[n]$

$$\text{Compute point cloud centers: } {}^V C = \frac{\sum_{i=1}^n {}^V \mathbf{P}[i]}{n}, {}^M C = \frac{\sum_{i=1}^n {}^M \mathbf{P}[i]}{n}$$

Compute normalized point cloud coordinates: ${}^V \bar{\mathbf{P}} = {}^V \mathbf{P} - {}^V C, {}^M \bar{\mathbf{P}} = {}^M \mathbf{P} - {}^M C$

$$\text{Initialize rotation matrix: } {}^M_V R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

do $i = 1$ to maxIterations

$$\bar{m}_v^2 \cos \gamma = \sum_{j=1}^n {}^V \bar{\mathbf{P}}[j].y {}^M \bar{\mathbf{P}}[j].y + {}^V \bar{\mathbf{P}}[j].z {}^M \bar{\mathbf{P}}[j].z$$

$$\bar{m}_v^2 \sin \gamma = \sum_{j=1}^n - \left({}^V \bar{\mathbf{P}}[j].z {}^M \bar{\mathbf{P}}[j].y \right) + {}^V \bar{\mathbf{P}}[j].y {}^M \bar{\mathbf{P}}[j].z$$

$$\bar{m}_v^2 = \sqrt{\left(\bar{m}_v^2 \cos \gamma \right)^2 + \left(\bar{m}_v^2 \sin \gamma \right)^2}$$

$$\cos \gamma = \frac{\bar{m}_v^2 \cos \gamma}{\bar{m}_v^2}, \sin \gamma = \frac{\bar{m}_v^2 \sin \gamma}{\bar{m}_v^2}$$

$$R_\gamma = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{bmatrix}$$

$${}^M_V R = R_\gamma {}^M_V R$$

$${}^V \bar{\mathbf{P}} = R_\gamma {}^V \bar{\mathbf{P}}$$

$$\bar{m}_v^2 \cos \beta = \sum_{j=1}^n {}^V \bar{\mathbf{P}}[j].z {}^M \bar{\mathbf{P}}[j].z + {}^V \bar{\mathbf{P}}[j].x {}^M \bar{\mathbf{P}}[j].x$$

$$\bar{m}_v^2 \sin \beta = \sum_{j=1}^n - \left({}^V \bar{\mathbf{P}}[j].x {}^M \bar{\mathbf{P}}[j].z \right) + {}^V \bar{\mathbf{P}}[j].z {}^M \bar{\mathbf{P}}[j].x$$

$$\bar{m}_v^2 = \sqrt{\left(\bar{m}_v^2 \cos \beta \right)^2 + \left(\bar{m}_v^2 \sin \beta \right)^2}$$

$$\cos \beta = \frac{\bar{m}_v^2 \cos \beta}{m_v^2}, \sin \beta = \frac{\bar{m}_v^2 \sin \beta}{m_v^2}$$

$$R_\beta = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix}$$

$${}^M_v R = R_\beta {}^M_v R$$

$${}^v \bar{\mathbf{P}} = R_\beta {}^v \bar{\mathbf{P}}$$

$$\bar{m}_v^2 \cos \alpha = \sum_{j=1}^n {}^v \bar{\mathbf{P}}[j].x {}^M \bar{\mathbf{P}}[j].x + {}^v \bar{\mathbf{P}}[j].y {}^M \bar{\mathbf{P}}[j].y$$

$$\bar{m}_v^2 \sin \alpha = \sum_{j=1}^n -\left({}^v \bar{\mathbf{P}}[j].y {}^M \bar{\mathbf{P}}[j].x \right) + {}^v \bar{\mathbf{P}}[j].x {}^M \bar{\mathbf{P}}[j].y$$

$$\bar{m}_v^2 = \sqrt{\left(\bar{m}_v^2 \cos \alpha \right)^2 + \left(\bar{m}_v^2 \sin \alpha \right)^2}$$

$$\cos \alpha = \frac{\bar{m}_v^2 \cos \alpha}{m_v^2}, \sin \alpha = \frac{\bar{m}_v^2 \sin \alpha}{m_v^2}$$

$$R_\alpha = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$${}^M_v R = R_\alpha {}^M_v R$$

$${}^v \bar{\mathbf{P}} = R_\alpha {}^v \bar{\mathbf{P}}$$

end do

$${}^M \mathbf{t}_v = \frac{\sum_{i=1}^n {}^M \mathbf{P}[i] - {}^M_v \mathbf{R} {}^v \mathbf{P}[i]}{n}$$

$${}^M_v T = \begin{bmatrix} {}^M_v R & {}^M \mathbf{t}_v \\ 0 & 1 \end{bmatrix}$$

Figure 3-1: Algorithm to determine camera-manipulator registration

3.2 Lighting Correction

Once an image has been acquired, it may need to be corrected for adverse lighting

conditions. At depth, color data is significantly lost when the scene is lit with a low intensity strobe or LED array. When color is important for analysis, images can be restored to their true color as described in Chapter 2, but depending on the application this may be unnecessary. Non-uniform lighting is also an issue. Especially with strobe lights, the location at which the light is focused is much more illuminated than surrounding areas. The light magnitude decrease radially outward from the focus location is nonlinear, requiring an algorithm to correct for the intensity pattern associated with a specific light source.

All lighting correction algorithms discussed below, aside from the WHOI algorithm, consist of an offline data extraction process, which determines scaling factors and exponential coefficients that are then applied during system operation. This procedure is valid under the assumption that lighting conditions will remain invariant between dives given consistent cameras, lights, and mounting configuration.

3.2.1 Frame-Averaging

The lighting correction algorithm developed in this work is a simple frame-averaging strategy that creates linear correction factors that can be applied in real-time. This solution is motivated by a more accurate but computationally-intensive algorithm developed and validated by WHOI [12]. This algorithm is performed offline and requires a set of images where both raw imagery and imagery corrected by the WHOI algorithm is available.

Figure 3-2 shows the process for calculating the frame average correction coefficients. Two sets of images are input: a set of uncorrected raw images and the

corresponding set of images that have already been corrected by the WHOI algorithm. First, pixel intensities $\mathbf{I}(i,j)$ for each RGB channel are summed across each image. Next, the ratio for each channel of corrected to uncorrected value is computed and stored in variables R_{fix} , G_{fix} and B_{fix} . These ratios are applied to each channel at every pixel in a new, uncorrected image to get a color-corrected image. An example undersea image correction is shown in Figure 3-5 on page 42. This correction strategy requires only one multiplication per pixel, minimizing real-time computational overhead.

```

do  $i = 1$  to  $\mathbf{I}_{width}$ 
  do  $j = 1$  to  $\mathbf{I}_{height}$ 
     $R_{raw} = R_{raw} + \mathbf{I}_{raw}(i, j).red$ 
     $G_{raw} = G_{raw} + \mathbf{I}_{raw}(i, j).green$ 
     $B_{raw} = B_{raw} + \mathbf{I}_{raw}(i, j).blue$ 
     $R_{corrected} = R_{corrected} + \mathbf{I}_{corrected}(i, j).red$ 
     $G_{corrected} = G_{corrected} + \mathbf{I}_{corrected}(i, j).green$ 
     $B_{corrected} = B_{corrected} + \mathbf{I}_{corrected}(i, j).blue$ 
  end do
end do
 $R_{fix} = \frac{R_{cor}}{R_{raw}}, G_{fix} = \frac{G_{cor}}{G_{raw}}, B_{fix} = \frac{B_{cor}}{B_{raw}}$ 

```

Figure 3-2: Algorithm to determine frame-average correction ratios

3.2.2 Lighting Pattern Estimation

To increase the quality of baseline results from the frame-averaging algorithm, this strategy was augmented to account for the non-uniform lighting pattern generated by the strobe. This correction provides greater contrast in target features relative to their

background thus facilitates cleaner target extraction.

The first step of this algorithm is to compare pixel magnitudes near the center of the projected lighting pattern with the magnitude of a pixel of similar visual properties in a more distant portion of the image. This process provides data relating changes in pixel intensity, presumed from light attenuation, with distance from the center of the lighting pattern. Images used in this research tend to have the lighting pattern centered on the image center, but the correction algorithm is based solely on pixel distance from a predetermined point, which could be located anywhere in the image depending on the light source. A MATLAB function was written to facilitate point extraction and lighting change calculation, as shown in Figure 3-3 below. The function first loads a raw image in a Bayer pattern, converts the image into a 3-channel RGB format, then displays the result and requests user-input for point matches. For this work, it is assumed that despite poor lighting conditions, the user can still identify similarly-colored light and dark objects, although prior to correction most vision algorithms would be unable to autonomously provide such data.

```

rgbImage = BayerCorrect(rawImage)
display(rgbImage)
User selects  $n$  point matches:  $\mathbf{p}_{light}$  and  $\mathbf{p}_{dark}$ 
do  $i = 1$  to  $n$ 

$$\mathbf{R}[i] = \frac{\mathbf{p}_{light}[i].red}{\mathbf{p}_{dark}[i].red}$$


$$\mathbf{G}[i] = \frac{\mathbf{p}_{light}[i].green}{\mathbf{p}_{dark}[i].green}$$


$$\mathbf{B}[i] = \frac{\mathbf{p}_{light}[i].blue}{\mathbf{p}_{dark}[i].blue}$$


$$\mathbf{d}[i] = \sqrt{(\mathbf{p}_{dark}[i].x - o_x)^2 + (\mathbf{p}_{dark}[i].y - o_y)^2}$$

end do
end GetLightingData

```

Figure 3-3: Algorithm for extracting lighting correction data

In Figure 3-3, the \mathbf{p} objects represent sets of data relevant to a single pixel – red, blue and green value in addition to x and y pixel location. The local \mathbf{R} , \mathbf{G} , and \mathbf{B} vectors have length equal to the number of points selected by the user and store the ratio of light to dark value for each color. Finally, the \mathbf{d} object stores pixel distance from the center of the image, o_x and o_y , except for the case where the center of the image is not the center of the lighting pattern and a different set of values would be used.

Once this set of data has been calculated, an exponential curve is fit to the data using Microsoft Excel. Data from each color channel is separately analyzed, and a coefficient and exponent are determined for each. Each curve fit equation is based solely on pixel distance $\mathbf{d}[i]$ from the center of the light source projection, so a

template image is created with the exponential calculations already performed to reduce the number of calculations during execution time.

Figure 3-4 below shows the process for creating the image containing the lighting correction template. \mathbf{I} represents the size of images to be processed, d is the distance measurement calculated at each position with o being the light source focus, \mathbf{R} , \mathbf{G} and \mathbf{B} are matrices that hold the correction ratios. Combining \mathbf{R} , \mathbf{G} and \mathbf{B} into a single 3-channel image provides the correction pattern. Parameters \mathbf{c} and \mathbf{e} used in the exponential calculations are output from the exponential curve fit based on the data from Figure 3-3.

```

do  $i = 1$  to  $\mathbf{I}_{width}$ 
  do  $j = 1$  to  $\mathbf{I}_{height}$ 
     $d_{(i,j)} = \sqrt{(i - o_x)^2 + (j - o_y)^2}$ 
     $\mathbf{R}_{(i,j)} = \mathbf{c}_{red} e^{\mathbf{e}_{red} d_{(i,j)}}$ 
     $\mathbf{G}_{(i,j)} = \mathbf{c}_{green} e^{\mathbf{e}_{green} d_{(i,j)}}$ 
     $\mathbf{B}_{(i,j)} = \mathbf{c}_{blue} e^{\mathbf{e}_{blue} d_{(i,j)}}$ 
  end do
end do

```

Figure 3-4: Algorithm for creating a lighting pattern template image

To apply the result of Figure 3-4, the real-time software must simply perform a multiplication for each channels (RGB) of each pixel to adjust the uncorrected image with the lighting correction pattern. Figure 3-5 shows the results from application of the color and lighting correction algorithms described in this section. The WHOI

algorithm image is considered as “truth” but as mentioned earlier, the execution time to perform the correction is prohibitive to real-time operation. For AVATAR, simply using the lighting pattern correction is sufficient to provide consistent data across the entire image, which is essential to target segmentation. The frame averaging correction makes the images more visually appealing to an observer, but a linear change in RGB intensity does nothing to increase effectiveness of the color filter described in the following section.

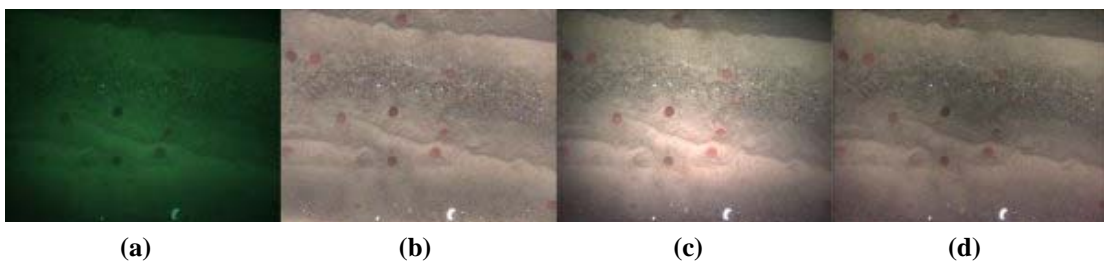


Figure 3-5: Side-by-side comparison of an image corrected for the lighting pattern.

(a) Uncorrected, (b) WHOI algorithm [12], (c) frame averaging, (d) frame averaging with lighting pattern estimation

3.3 Feature Extraction

As discussed in Section 2.3, numerous methods have been developed to extract features from an image. For this research, a basic RGB based color filter is applied to segment targets of a specific color. This is most closely related to the histogram filters, yet by using a combination of all three channels, choosing specific ranges on the image histograms may not produce consistent results across channels. The main issues to address with this algorithm are inconsistent lighting due to shadows or uncorrected lighting variations, and identification of targets with a color that closely matches the background.

For initial development, only the first problem was handled – creation of an

algorithm that deals with lighting changes. The developed filtering algorithm distinguishes targets from ratios of red, green and blue values for each pixel in addition to the magnitude of each. Intuitively, this approach removes the effects of overall brightness disparity through the use of ratios -- while the brightness of all colors may be changing, the relative amount of red vs. blue, blue vs. green, or green vs. red will remain relatively constant. A similar effect could be achieved by converting each pixel into a hue-based format, a potential future improvement, as would be the use of multiple methods [17][20].

3.3.1 Filter Creation

Prior to application of a filter, the base values for a desired target must be determined. A MATLAB function was created to simplify the filter creation process. An ideal filter would separate the image into two groups: pixels that are part of a desired target and pixels that are either background or part of an undesired target. This program allows the user to select sets of points from both groups and then plot the appropriate filter values. Through the plots of the different filter values, appropriate ranges can be extracted to successfully perform target extraction on other images. Figure 3-6 shows pseudocode for this function.

```
Display RGB image
User selects  $n$  image points:  $\mathbf{p}[1, \dots, n]$ 
do  $i = 1$  to  $n$ 
     $\mathbf{M}[i] = \mathbf{p}[i].red + \mathbf{p}[i].green + \mathbf{p}[i].blue$ 
     $\mathbf{RvG}[i] = \frac{\mathbf{p}[i].red}{\mathbf{p}[i].green}$ 
     $\mathbf{RvB}[i] = \frac{\mathbf{p}[i].red}{\mathbf{p}[i].blue}$ 
     $\mathbf{BvG}[i] = \frac{\mathbf{p}[i].blue}{\mathbf{p}[i].green}$ 
end do
```

Figure 3-6: Algorithm for extracting feature ratio values

This function is applied to both the desired sampling target as well as any competing targets with similar visual properties. By plotting the output ratio data from both sets, stored in vectors **RvG**, **RvB**, and **BvG**, with respect to the overall magnitude at each point, **M**, appropriate filter values are easily extracted. Figure 3-7 shows an example plot from this program with the range of values for sampling target and background clearly separated for a chromatically-distinctive target, a yellow rubber ducky, and a less clear distinction for a sand dollar from an image similar to Figure 3-5 on page 42. Use of the magnitude data is only necessary when there are similarities in color data corresponding with variations in magnitude.

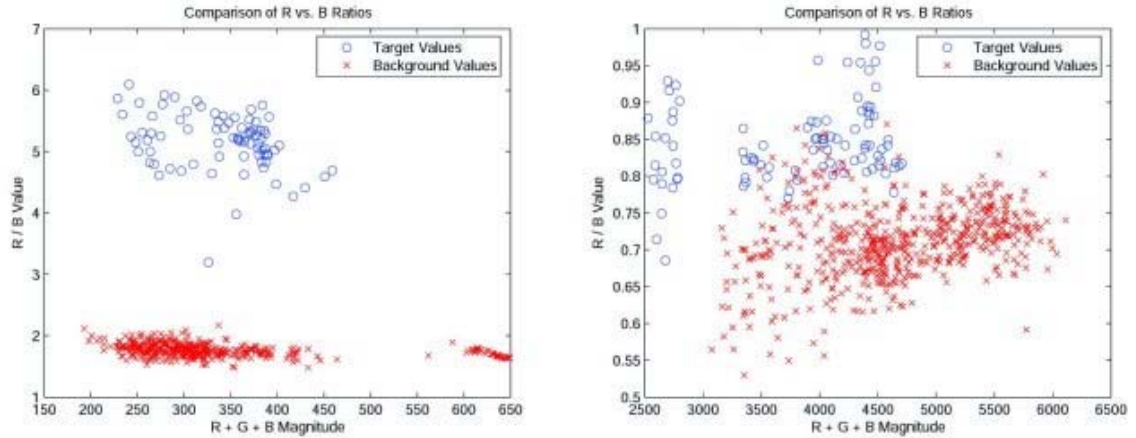


Figure 3-7: Sample plots showing Red/Blue ratio data for rubber ducky target (left) and uncorrected sand dollar (right)

3.3.2 Application of the Color Filter

Once ranges of values have been selected, application of the filtering algorithm is straightforward. The algorithm compares pixel RGB ratio data with the values set for the filter, and if they fit within the target range they are unchanged, otherwise they are set to 0. Figure 3-8 shows this process. Increased accuracy, at the risk of increased complexity, can be achieved by including the magnitude measurement either as a maximum/minimum similar to the ratio ranges, or by providing multiple ratio maximum/minimum values depending on the overall magnitude.


```

do  $i = 1$  to  $\mathbf{I}_{width}$ 
  do  $j = 1$  to  $\mathbf{I}_{height}$ 
    
$$RvG_{pixel} = \frac{\mathbf{I}[i, j].red}{\mathbf{I}[i, j].green}$$

    
$$RvB_{pixel} = \frac{\mathbf{I}[i, j].red}{\mathbf{I}[i, j].blue}$$

    
$$BvG_{pixel} = \frac{\mathbf{I}[i, j].blue}{\mathbf{I}[i, j].green}$$

    if  $RvG_{min} \leq RvG_{pixel} \leq RvG_{max}$  AND
        $RvB_{min} \leq RvB_{pixel} \leq RvB_{max}$  AND
        $BvG_{min} \leq BvG_{pixel} \leq BvG_{max}$  then
      
$$\mathbf{I}[i, j] = \mathbf{I}[i, j]$$

    else
      
$$\mathbf{I}[i, j] = 0$$

    end if
  end do
end do

```

Figure 3-8: Algorithm for RGB ratio color filter

Upon successful completion of the color-based portion of the feature extraction algorithm, the resulting image contains a thresholded image where only “good” target points remain. After the entire image is thresholded, an erode operation [13][19] is applied to remove stray noise pixels and incomplete features. Since this process reduces the quality of remaining features of interest, a Feature-AND operation [19] is then applied to restore these features to their full quality. The Feature-AND process performs a basic AND operation between two images, but restores any connecting pixels that may only appear in one image. Thus, small features will be removed

through the erosion process and larger features, which remain after erosion, will be restored to their original state.

Although this erosion and restoration process is optional, it allows the color filter to be significantly more liberal in the filter maximum and minimum values, with the positive result that features of interest are more complete, but the negative result that more false positives and noise remain in the image. By implementing the erode-and-restore algorithm, the features of interest will remain at higher quality and unwanted features will be removed. Unfortunately, as with the other optional algorithms, this adds complexity and computations to the algorithm, reducing the overall frequency with which the vision analysis can operate. Figure 3-9 shows images at each stage of the filtering process.

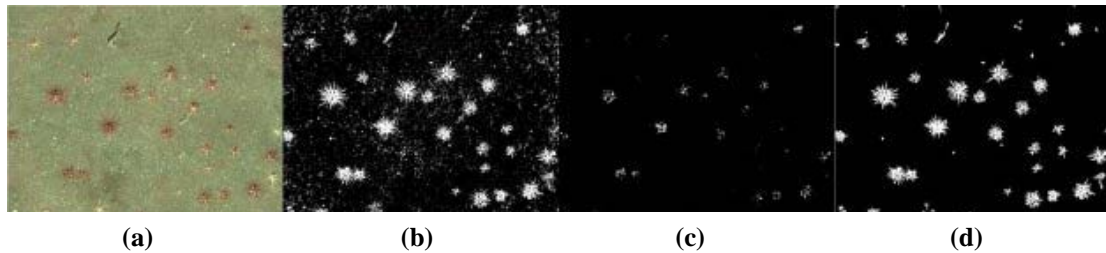


Figure 3-9: RGB ratio filtering process

(a) start with lighting-corrected RGB image, (b) filter based on RGB ratio values, (c) erode to remove noise and (d) restore with feature AND operation (Original image courtesy WHOI)

3.3.3 Feature Extraction

The next portion of the algorithm is designed to operate on the lighting and color filtered images to determine location and size of all remaining features. The algorithm for extracting feature size and location from a filtered image consists of a recursive search process that calculates any desired target data useful for later algorithms. The search process begins at the top-left corner of the filtered image and

searches from left to right, top to bottom for pixels that remained “on” after the filtering algorithm is executed. Upon reaching a first “on” pixel, the local minimum X value associated with the global minimum Y value is recorded, and the algorithm initiates a search to record the locations of all connecting pixels. If the pixel area of the remaining features fit within a specified threshold, the recorded list of pixel values representing each entire feature is used for further calculations including centroid, area and aspect ratio. Figure 3-10 shows the two basic functions associated with the feature information extraction algorithm.

```

Function DolmageFeatureSearch
  numFeatures = 0
  do i = 1 to Iwidth
    do j = 1 to Iheight
      if I[i, j] ≠ (0,0,0) then
        numFeatures = numFeatures + 1
        F[numFeatures] = CheckConnectedPixels(i, j)
      end if
    end do
  end do
end Function

Function CheckConnectedPixels
  if I[i, j] ≠ (0,0,0) then
    f .add(i, j)
    I[i, j] = (0,0,0)
    f .add(CheckConnectedPixels(i + 1, j))
    f .add(CheckConnectedPixels(i + 1, j - 1))
    f .add(CheckConnectedPixels(i, j - 1))
  end if
end Function

```

```

    f .add(CheckConnectedPixels( $i - 1, j - 1$ )
    f .add(CheckConnectedPixels( $i - 1, j$ )
    f .add(CheckConnectedPixels( $i - 1, j + 1$ )
    f .add(CheckConnectedPixels( $i, j + 1$ )
    f .add(CheckConnectedPixels( $i + 1, j + 1$ )
end if
return f
end Function

```

Figure 3-10: Algorithms used to extract feature raw data

The first function, *DoImageFeatureSearch* searches the image until finding a non-black pixel in RGB format (i.e., $\mathbf{I}[i, j] \neq (0,0,0)$). After an “on” pixel is found, the recursive procedure described in the *CheckConnectedPixels* function begins. This function checks the value of an input pixel and if non-zero adds the pixel location to a list. This operation is repeated on all eight neighboring (adjacent) pixels. By setting all three channels to zero after recording each identified location with non-zero initial value, the function ensures that no pixel will be counted twice. On return, the *f* vector contains ordered pairs for all pixel locations within the current feature. *DoImageFeatureSearch* retains a list of all possible features within *F*.

Implementation of these feature extraction functions is realized in C++ using the Standard Template Library, or STL [28]. By storing the pixel locations as an STL vector, inherent STL functions can be used to operate on the data. For instance, the feature area can be calculated by calling the “.size()” member function. STL iterators are used to iterate through each list of feature points with functions to determine aspect ratio and other relevant information.

The data set for each feature is produced from a few simple calculations. First, a single loop through all the data points is performed to identify boundary points around the image edges. Eight edge points of the feature are recorded, which consist of all perturbations of local and global maximum and minimum X and Y values. Table 3-1 shows a list of the eight points and Figure 3-11 visually shows each point on a sample set of matched targets, from the feature matching algorithm detailed in Section 3.4.1.

Table 3-1: List of Feature Shape Points

Point Number	Local/Global X	Max/Min X	Local/Global Y	Max/Min Y	Notation in Figure 3-13
1	Local	Minimum	Global	Minimum	$(X_{l,Yrop}, Y_{top})$
2	Local	Maximum	Global	Minimum	$(X_{r,Yrop}, Y_{top})$
3	Global	Maximum	Local	Minimum	(X_r, Y_{top}, X_r)
4	Global	Maximum	Local	Maximum	(X_r, Y_{bot}, X_r)
5	Local	Maximum	Global	Maximum	$(X_{r,Ybot}, Y_{bot})$
6	Local	Minimum	Global	Maximum	$(X_{l,Ybot}, Y_{bot})$
7	Global	Minimum	Local	Maximum	(X_l, Y_{bot}, X_l)
8	Global	Minimum	Local	Minimum	(X_l, Y_{top}, X_l)

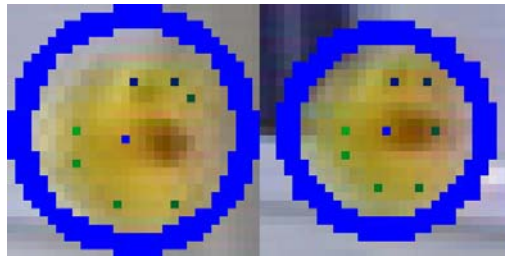


Figure 3-11: Perimeter points of feature shown in sample feature match

Once the boundary points have been determined, aspect ratio is calculated based on

the global bounding box of the feature. The ratio of feature area to bounding box area is also calculated, and henceforth referred to as the area ratio. The area ratio gives the software an idea of feature “density”. For instance, a feature generated by connected noise pixels can have a significant area, but will usually be far from solid. Figure 3-12 is an example using Ranger’s parallel jaw end effector. An ineffective filter for the Ranger Interchangeable End Effector Mechanism (IEEM) has left much of the jaw still visible. Although the remaining pixels do not form a solid feature, they are still identified as a feature due to connectivity.



Figure 3-12: Ineffective filter of Ranger’s IEEM and end effector

After compiling feature data, another filter is applied based on the feature’s image plane geometric properties. Feature area, aspect ratio, and area ratio are used to determine whether or not the recently-discovered feature is a correct match for the target the vision system is attempting to identify. Figure 3-13 shows the computation of 2-D feature geometric properties. Definitions for each of the X and Y variables and related subscripts are present in Table 3-1. The $totalX$ and $totalY$ variables are sums of all point locations used in the calculation of the centroid, (C_x, C_y) . The total number of pixels is denoted by $size(\mathbf{f})$, where \mathbf{f} is the list of feature point pixel

locations determined from the algorithm in Figure 3-10.

```

do  $i = 1$  to  $\text{size}(\mathbf{f})$ 
     $\text{totalX} = \text{totalX} + \mathbf{f}[i].x$ 
     $\text{totalY} = \text{totalY} + \mathbf{f}[i].y$ 
    if  $i == 1$  then
         $X_r = \mathbf{f}[i].x$ 
         $X_l = \mathbf{f}[i].x$ 
         $Y_{bot} = \mathbf{f}[i].y$ 
         $Y_{top} = \mathbf{f}[i].y$ 
    else
        if  $\mathbf{f}[i].x > X_r$  then
             $X_r = \mathbf{f}[i].x$ 
             $Y_{bot, X_r} = \mathbf{f}[i].y$ 
             $Y_{top, X_r} = \mathbf{f}[i].y$ 
        else if  $\mathbf{f}[i].x == X_r$  then
            if  $\mathbf{f}[i].y > Y_{bot, X_r}$  then
                 $Y_{bot, X_r} = \mathbf{f}[i].y$ 
            end if
            if  $\mathbf{f}[i].y < Y_{top, X_r}$  then
                 $Y_{top, X_r} = \mathbf{f}[i].y$ 
            end if
        end if
    end if
/* Similar calculations for other parameters, full algorithm in Appendix C */
end do

 $C_x = \frac{\text{totalX}}{\text{size}(\mathbf{f})}$ ,  $C_y = \frac{\text{totalY}}{\text{size}(\mathbf{f})}$ 

 $\text{aspectRatio} = \frac{X_r - X_l}{Y_{bot} - Y_{top}}$ 

 $\text{boxArea} = (X_r - X_l)(Y_{bot} - Y_{top})$ 

```

$$areaRatio = \frac{size(\mathbf{f})}{boxArea}$$

Figure 3-13: Algorithm for extracting feature geometric properties

3.4 Stereo Correspondence

Once a list of extracted features is available, the next step is to match features to establish proper correspondence between stereo images. This will eventually enable 3-D position determination for targets within the camera frame of reference.

3.4.1 Feature Matching

As described in Chapter 2, many algorithms exist for autonomously matching features and points between corresponding images [29][30][31][32]. When practical, most algorithms require human interaction to improve accuracy [33]. For this application, human interaction is not feasible except for offline algorithm tuning. Different matching algorithms comparing size, shape and geometric properties of the system were applied in different phases of this research. The first algorithm is based solely upon feature properties, only eliminating geometric impossibilities from the list of possible matches. The second algorithm inverts this process, initially creating a list of geometric possibilities, based upon Epipolar constraints, and then uses feature properties as a sanity check to ensure the features are visually similar.

The shape-based algorithm begins by using the nine interest points from each feature – eight points around the edge of the features, as shown in Figure 3-11, and the centroid. By determining the unit vector and magnitude from each of the nine points to every other point, a mathematical “shape” is calculated which can be used to

compare feature sets. Figure 3-14 shows the algorithm used to calculate the set of shape vectors. Although this procedure performs redundant calculations, this procedure retains matching code simplicity.

```

do  $i = 1$  to 9
  do  $j = 1$  to 9
    if  $i \neq j$  then
      
$$\mathbf{M}[i][j] = \sqrt{(\mathbf{p}[i].x - \mathbf{p}[j].x)^2 + (\mathbf{p}[i].y - \mathbf{p}[j].y)^2}$$

      
$$\mathbf{S}[i][j] = \left( \frac{\mathbf{p}[i].x - \mathbf{p}[j].x}{\mathbf{M}[i][j]}, \frac{\mathbf{p}[i].y - \mathbf{p}[j].y}{\mathbf{M}[i][j]} \right)$$

    else
       $\mathbf{M}[i][j] = 0$ 
       $\mathbf{S}[i][j] = (0,0)$ 
    end if
  end do
end do

```

Figure 3-14: Algorithm to calculate feature shape vectors

The \mathbf{p} variables represent the sets of nine points associated with a single feature. \mathbf{M} and \mathbf{S} store the calculated data for magnitude and shape, respectively. Each image will have a set of \mathbf{p} vectors with size equal to the number of discovered features in that image. By minimizing the differences between feature descriptors from corresponding images, targets with the same shape and orientation will be matched correctly between paired images with this comparison of relative position between external points and the centroid. It is important to note that relying on this orientation data requires the assumption that the camera image planes are nearly parallel. Figure 3-15 shows the basic process for the shape-based feature matching.

```

Pleft = {pl[1], ..., pl[nleft]}
Pright = {pr[1], ..., pr[nright]}
do i = 1 to nleft
    BM[i] = FindBestMatch(Pleft[i], Pright)
end do
HandleRepeatMatches(BM)

```

Figure 3-15: Algorithm to match features by shape estimates

The **P**_{left} and **P**_{right} variables contain the sets of feature points and related shape and magnitude values, **M** and **S**, calculated in Figure 3-14. **BM** stores the minimum error value match between a feature from the left image and the set of features from the right image, as determined by the *FindBestMatch* function. The *FindBestMatch* function determines the best match by minimizing differences in the respective **M** and **S** data. In addition to using shape data, the algorithm applies a single geometric constraint – the X value of the centroid in the left image must be greater than the X value in the right image. In other words, the feature must be in front of the cameras. Even after applying minimization of error data, if identical targets are placed within the same field of view, mismatches can occur. In the case that multiple features from the left image are matched with the same feature in the right image, the *HandleRepeatMatches* routine determines which has a lower error and voids the other matches.

To simplify the feature matching process and eliminate impossible geometric matches, a new algorithm was developed that primarily utilizes geometric parameters to create a list of possible matches. Feature shape properties are then required only to “break ties” in cases where multiple possible matches are available. Due to the

reliance on the more selective epipolar constraints, the complexity of the matching algorithm significantly decreases in addition to becoming more accurate. In target fields where all targets have significant pixel area and there is minimal overlap, it is almost impossible to have multiple match cases after making some intelligent assumptions, thus increasing the constraints already imposed by epipolar geometry.

Epipolar geometry constrains features to match along a single line within a corresponding image in a calibrated stereo system. By making assumptions on minimum and maximum distance from the cameras, this line through the full image can be reduced to a short segment. Distance assumptions are valid since target area will be too large or small if too close or far from the cameras, respectively. These constraints are met by translating feature coordinates in an original image to a limited search box in the corresponding image. The parameters of the box are defined such that a realistic segment of the epipolar line is contained within the box, but also, to account for calibration errors, a region around the line is also searched for possible matches. This algorithm represents a significant simplification to the research presented in [30]. Figure 3-17 shows a populated target field with all targets correctly, matched using this set of assumptions. Figure 3-16 summarizes this matching process.

```

do  $i = 1$  to  $n_{left}$ 
  do  $j = 1$  to  $n_{right}$ 
     $dx = \mathbf{P}_{left}[i].x - \mathbf{P}_{right}[j].x$ 
     $dy = \mathbf{P}_{left}[i].y - \mathbf{P}_{right}[j].y$ 
    if  $dx_{min} < dx < dx_{max}$  &&
       $dy_{min} < dy < dy_{max}$  then
       $\mathbf{PM}[i].add(\{\mathbf{P}_{left}[i], \mathbf{P}_{right}[j]\})$ 
    end if
  end do
  if  $size(\mathbf{PM}[i]) > 1$  then
     $HandleRepeatMatches(\mathbf{PM}[i])$ 
  end if
end do

```

Figure 3-16: Algorithm to create a geometric based possible match list

The output of this routine is the list of possible matches $\mathbf{PM}[i, j]$. If a possible match between left feature i and right feature j exists, then that value of the \mathbf{PM} matrix will be non-zero. The four values of dx_{min} , dx_{max} , dy_{min} and dy_{max} form the box around possible target positions based on geometry. Usually the \mathbf{PM} data will not contain multiple matches for each feature. However, if repeat matches do occur, the shape-based error minimization routine can be applied to determine the “better” match based on shape.



Figure 3-17: Sample correlated images used for geometric match testing

3.4.2 3-D Reconstruction for Target Position

The final step of the stereo correspondence process is to triangulate target position. Similar to the calibration process, a stereo triangulation algorithm from the MATLAB calibration toolbox [3] was converted into C code for seamless integration with the rest of the target acquisition system. Figure 3-18 shows the stereo triangulation process, where initial pixel coordinates are represented by x_{left} and x_{right} while output vision-frame coordinates are designated by XL and XR for left and right camera locations, respectively.

$$\begin{aligned}
& \text{normalize}(x_{left}, x_{right}) \\
& X_l = \begin{bmatrix} x_{left} \\ 1 \end{bmatrix}, X_r = \begin{bmatrix} x_{right} \\ 1 \end{bmatrix} \\
& u = {}^R_L R \cdot X_l \\
& DD = (X_l \cdot X_l)(X_r \cdot X_r) - (X_r \cdot u)^2 \\
& NN1 = (X_r \cdot u)(X_r \cdot {}^R \mathbf{t}_L) - (X_r \cdot X_r)(u \cdot {}^R \mathbf{t}_L) \\
& NN2 = (X_l \cdot X_l)(X_r \cdot {}^R \mathbf{t}_L) - (u \cdot {}^R \mathbf{t}_L)(X_r \cdot u) \\
& Z_l = \frac{NN1}{DD}, Z_r = \frac{NN2}{DD} \\
& X1 = X_l Z_l \\
& X2 = {}^L_R R (X_r Z_r - {}^R \mathbf{t}_L) \\
& XL = \frac{1}{2}(X1 + X2) \\
& XR = {}^R_L R (XL) + {}^R \mathbf{t}_L
\end{aligned}$$

Figure 3-18: Algorithm to calculate 3-D target position via stereo triangulation

The first step of the algorithm is the *normalize* function [3]. This function applies the intrinsic camera calibration parameters to the initial image plane coordinates. The next step is to build a homogenous coordinate vector by adding a third dimension, the 1. The remainder of the algorithm triangulates the initial rays, X_l and X_r , in 3D space. The extrinsic parameters of the system also appear in this algorithm, shown as ${}^R_L R$, or the transpose ${}^L_R R$, and the translation vector ${}^R \mathbf{t}_L$.

Implementation of this algorithm exists in two forms. The first method calculates a 3-D position for all of the externally matched points determined in the shape calculation. The second method determines target position solely from the pixel values calculated for the centroid of the feature in both images. Using all points

will reduce error from a single point, but in the case where the image planes of the cameras are not aligned, these external points will not match. By using only the centroid, testing has shown that accurate localization is still possible, but the position is much more sensitive to single pixel error. Both methods were implemented during tests of the vision system with Ranger.

3.5 Visual Servoing

Uncertainties in calibration due to even slight camera misalignment can result in poor target sampling success. Research has been done that shows inclusion of visual target data within a simple control loop, often referred to as visual servoing, can overcome errors that arise from calibration uncertainties or errors, even if the initial calibration is extremely inaccurate [10][11]. To successfully implement visual servoing, the vision system must be able to identify and track both the manipulator end effector and the desired sampling target.

3.5.1 Visual Servo Algorithm

The goal of the visual servo algorithm is to alleviate inaccuracies in manipulator-camera relative positions due to calibration errors. Instead of triangulating the position of just the sampling target, the vision system must now also recognize a target on the manipulator. To be of use to the manipulator controller, the coordinates must be transformed into a frame of reference that the manipulator recognizes. For this algorithm it is assumed that the procedure discussed in Section 3.1.2, the camera-manipulator registration, has already been performed and these values are known. The vision system must now “track” both targets through subsequent analyses to

make sure all data is consistent. By commanding the manipulator to move towards the sampling target, in terms of what the cameras are seeing, and repeating until the distance is zero, inaccuracies within the stereo system calibration and camera-manipulator registration can be ignored and successful sampling can still occur.

The visual servoing algorithm is shown in Figure 3-19. The first step of the algorithm is to acquire positions of possible manipulator and sampling targets from the vision system. Once the targets have been acquired and their positions calculated, the software must choose the correct targets if multiple candidates exist. After the targets have been selected and verified, the third step of Figure 3-19 is reached. Subsequently, all \mathbf{p} variables represent 3-D locations of targets. The leading superscript denotes the coordinate frame – ν for vision, θ for manipulator base and T for manipulator tool frame, while the trailing subscript defines target type. By default all measurements are made by the vision system, but if that is not the case a second subscript indicates the measurement device, such as *telem* for arm telemetry. The rotation matrix ${}^{\nu}_v \mathbf{R}$ and translation vector ${}^{\theta} \mathbf{t}_{vision}$ are the result of the hand-eye calibration (camera-manipulator registration). However, the manipulator controller provides the rotation matrix ${}^{\theta}_T \mathbf{R}$ so that the vision system does not require knowledge of manipulator pose in addition to tool position. The tool vector ${}^T \mathbf{p}_{tool}$ is the translation from where the vision system measures the arm position and the actual tool tip. During tests with Ranger, ${}^T \mathbf{p}_{tool}$ was the vector from the IEEM to the tip of the parallel jaw grippers.

Together this data enables calculation of tool tip position in the manipulator

base frame based on the coordinates originally measured by the vision system. First, the rotation from vision to base frame is applied to the vision frame coordinates of the arm target. Next, the translation from the location of the vision frame origin to manipulator base frame origin must be added. The final step is to transform the last offset of the end-effector, from vision target to tip of tool, into base frame coordinates and add that to the previous result. Once this step has been accomplished, arm telemetry is used to ensure the visual estimate is reasonable.

Once a consistent arm position is verified, the base frame coordinates of the sampling target are calculated in the same manner as the manipulator vision target except with no additional tool offset. Finally, a base frame motion vector is calculated to drive the manipulator toward its target grasp state.

This motion vector is scaled down to match a maximum move distance for safe operation of the manipulator. If the calculated motion vector is less than the maximum move value, the following arm motion is the final move. If at any point the vision system loses track of either target, the system will stop moving until either the target is recognized once again, a timeout is reached, or an operator kills the process. As the manipulator approaches the target, it is likely that partial or total occlusion of the target will occur. Without full knowledge of end effector and manipulator design, purposely omitted from the vision software to ensure portability, it would be impossible to account for occlusion from these sources. If the target is completely stationary, its position could be assumed constant if lost, but this may not be valid. A more in-depth analysis of target occlusion is located in Section 3.6.2.

```

GetNewVisionData( ${}^V \mathbf{p}_{arm}, {}^V \mathbf{p}_{sample}$ )

CheckForValidVisonData( ${}^V \mathbf{p}_{arm}, {}^V \mathbf{p}_{sample}$ )

 ${}^0 \mathbf{p}_{tool} = {}^0 \mathbf{R}^V \mathbf{p}_{arm} + {}^0 \mathbf{t}_{camera} + {}^0 \mathbf{R}^T \mathbf{p}_{tool}$ 

CheckVisionMeasurement( ${}^0 \mathbf{p}_{tool}, {}^0 \mathbf{p}_{tool,telem}$ )

 ${}^0 \mathbf{p}_{sample} = {}^0 \mathbf{R}^V \mathbf{p}_{sample} + {}^0 \mathbf{t}_{camera}$ 

 ${}^0 \mathbf{e}_{sample} = {}^0 \mathbf{p}_{sample} - {}^0 \mathbf{p}_{tool}$ 

if  $\|{}^0 \mathbf{p}_{motion}\| > d_{last}$  then
    end VisualServo
else if  $\|{}^0 \mathbf{p}_{motion}\| > d_{max}$  then
     $K_s = \frac{\|{}^0 \mathbf{p}_{motion}\|}{d_{max}}$ 
else
     $K_s = 1$ 
end if

 ${}^0 \mathbf{e}_{motion} = \frac{{}^0 \mathbf{e}_{sample}}{K_s}$ 

MoveManipulator( ${}^0 \mathbf{e}_{motion}$ )

```

Figure 3-19: Algorithm for arm motion through visual servo

3.5.2 Minor Visual Servo Functions

Target Identification

For the visual servo algorithm to function properly, the system must be capable of recognizing multiple target types. Target identification in this manner is simple – multiple target filters are performed on the initial images to extract both types of visual targets, sampling vs. manipulator. With correct target identification, this

procedure is sufficient. In Figure 3-19 this is handled by the *GetNewVisionData* function. This performs the appropriate feature extraction and matching algorithms to populate the respective **p** variables with data describing the sampling and manipulator targets. At this stage of development, no calculations are performed to select which targets are correct, thus the system relies on unique identification of one sampling and one manipulator target.

Target Tracking

Implementation of the visual servo controller introduces the requirement that the selected sampling and arm targets recognized over a sequence of images are equivalent. In Figure 3-19 this is denoted the *CheckForValidVisonData* function. This function validates that observed motion between frames is consistent with expectations. In the case of a stationary target, there should be no significant motion, and, in the case of a slowly moving target, such as the arm, the perceived motion should match projected estimates. All our targets were presumed stationary during test sequences. Also, due to the slow update rate of the vision system, constrained both by AUV electrical power and processing considerations to approximately 1 Hz, it is impossible to track fast moving targets. However, in future work it may be possible to add a Kalman filter to propagate arm motion at intermediate time points.

If only single targets for each of the sampling and manipulator targets are found, they are trivially matched over an image sequence. With multiple targets, the algorithm selects the target with position closest to the previously used target.

3.6 Management of anomalies, occlusions, and poor visibility conditions

When operating in a fully autonomous setting, a system must be capable of handling off-nominal situations that may arise. To date, the AVATAR vision system has focused on baseline implementation and validation, but some problematic anomaly scenarios have been enumerated for which we suggest potential strategies for robust autonomous management. For the underwater sampling mission, likely challenges will take the form of poor visibility caused by silt or hydrothermal vent fluid, target occlusion, and misinterpretation of the image data resulting in target recognition or localization anomalies.

3.6.1 Poor Visibility

Although deep-sea visibility is typically excellent, poor visibility conditions could be encountered due to two major sources: the vehicle or manipulator agitating the ocean floor and causing silt to rise, or from the black smoke emitted by the hydrothermal vents. To handle the case where the AUV or manipulator causes silt to rise due to impact with the ocean floor, the system need only wait a short period of time for the silt to settle. Videos recorded during dives of the WHOI ROV Jason II show that it is a common occurrence for the manipulator, while being teleoperated, to collide with the soil and cause temporary visibility problems. However, after only a few seconds of remaining motionless, the agitated silt settles and visibility returns to normal. In terms of the vision algorithm, this means that it must be able to recognize when the quality of the image has decreased due to visibility degradation, possibly also through feedback from the manipulator, and initiate a wait sequence until image quality has

been restored.

The second case is where visibility is reduced due to occlusion from hydrothermal vent fluid. Hydrothermal vents or “black smokers” excrete high temperature fluids that will cloud camera views. The interaction of ocean floor currents with vent fluid is perhaps the most likely long-term poor visibility scenario that could compromise sampling efforts. These currents can spread the fluid while it is still rising, and downstream of the vent visibility can be quite poor. This problem can be managed by moving upstream of the vent to another possible sampling area. Although such maneuvers are dictated by autonomy software outside the scope of this thesis, the vision system must be able to recognize and alert this autonomy software when the camera visibility becomes poor, ideally also classifying the poor visibility conditions as due to fluid or silt.

3.6.2 Occluded Targets

The other major problem that must be handled is target occlusion due to either the manipulator blocking key portions of the camera field of view, or sampling site topography causing occlusion of desired targets. Having full view of the desired sampling target in both cameras is vital to accurate 3-D localization and successful target retrieval. Prior to field trials with the final system, these issues must be dealt with to ensure simple target occlusion does not cause mission failure.

There are currently two methods to deal with target occlusion as a result of the manipulator entering the camera field of view. Depending on overall accuracy of the system calibration and confidence that the target and AUV base are truly stationary, a

single snapshot of the visual target data with the manipulator stowed out of the camera views is sufficient to fully specify the target location open-loop. If this “dead-reckoning” operational paradigm is sufficient, then manipulator occlusion during the sampling process can be ignored.

Alternatively, if regular data updates are required, either for a visual servo system or because of a slow moving target or drifting AUV, then active steps must be taken to prevent occlusion by the manipulator. Research is underway at SSL for autonomous obstacle avoidance with a manipulator [34]. By creating virtual obstacles for manipulator poses that would block visibility for the cameras, and then avoiding these poses, it may be ensured that target occlusion will not occur.

If the manipulator is not blocking the camera view yet targets are still visible in only a single camera view, then it is likely that something within the sampling site is occluding targets for one of the cameras. Then, if there are no other targets can be successfully retrieved while in this position, the AUV must move to another sampling location where target occlusion does not occur.

Chapter 4 Software Design and Implementation

4.1 System Architecture

The ASTEP system consists of the AUV and its computer, the manipulator and its computer (DMU), and the vision system cameras plus computer. These systems are supported by the sensors, actuators, and auxiliary equipment (e.g., strobe lights, batteries, etc.) to enable robust autonomous deep-sea operation. Figure 4-1 shows the layout of the key computers and related hardware on the AUV. The DMU computer interfaces with both the vision computer that connects with the cameras, and the WHOI computer that controls the AUV.

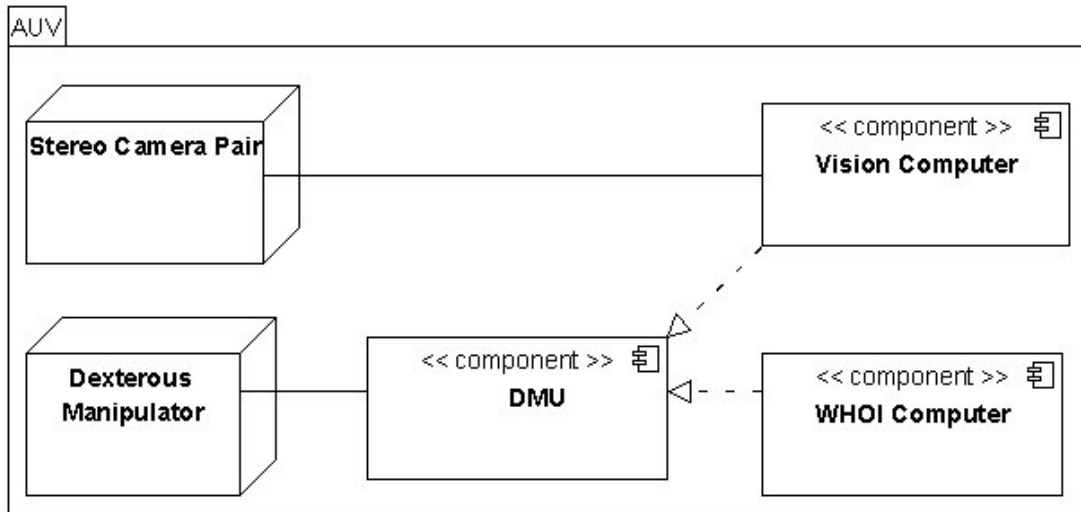


Figure 4-1: Overview of AUV system architecture

First, the sub-architecture of the DMU computer will be analyzed to develop the relationship between all three systems (vision, manipulator and AUV). Next, the modules of the vision computer and its protocols will be broken down into greater

detail

Two different architectural components are of major importance to this research. First is the structure internal to the vision system. This module (AVATAR) includes both vision analysis methods/software as well as all the interfaces to control and retrieve data from the cameras. The second modules (TAU) developed to isolate AUV and manipulator specifics from vision algorithms are the software and data structures that interface the vision system with the external vehicle. Two such system-specific interfaces are discussed: a lab-based implementation for the Ranger manipulator and an implementation that will enable future field trials with the SAMURAI-AUV system.

4.1.1 DMU Sub-Architecture

For testing and field trials, the vision-based target acquisition system will exist as one of many modules within a larger vehicle-wide software system. For this thesis, testing of AVATAR takes place within the scope of the Ranger manipulator system architecture. When the NASA-ASTEP mission comes to fruition, the vision system will be implemented as part of the larger-scale SSL-WHOI SAMURAI-AUV software system.

Ranger Software Architecture

When tested with the Ranger manipulator, the vision system is interfaced through the Data Management Unit, or DMU. The DMU is the primary computer for Ranger, executing all safety, control and interface algorithms necessary for operating the manipulator. Traditionally, Ranger has been teleoperated through a control station

that receives commands from hand controllers. To operate Ranger autonomously, a trajectory file must be loaded into the Ranger control station. For vision-based sampling tasks, a special trajectory software module capable of communicating with TAU through a DMU object was implemented to perform as a visual servo controller.

As shown in Figure 4-2, the system controller is the top-level DMU module. The system controller can manage multiple arm controllers, where each arm controller manages a single manipulator. For the Ranger implementation of the DMU, the note inside the system controller can be ignored, as it is only relevant for the ASTEP mission with SAMURAI. Within the arm controller are three methods of moving the manipulator hardware: runtime trajectories, incremental joint-by-joint commands, and a resolved rate controller that utilizes hand controllers. Each of these types of controllers is based on implementations of the manipulator's inverse and forward kinematics for calculating joint angle changes based either on desired Cartesian position and orientation or iterative joint trajectories. The runtime trajectories are different from the other two methods, as they run based on a file that may contain a series of waypoints in either Cartesian or joint space. At 125Hz the specified controller will calculate the new joint angles and apply them to the manipulator.

The vision system interfaces through a special trajectory item communicated through TAU during the system control loop. The visual servo trajectory implementation uses other, more traditional trajectory controllers (joint-by-joint or Cartesian) within itself to determine incremental joint angles for the arm based on the desired motion calculated by the vision system. By using Ranger's communication

protocols within TAU, implementation of TAU within the Ranger DMU structure was straightforward.

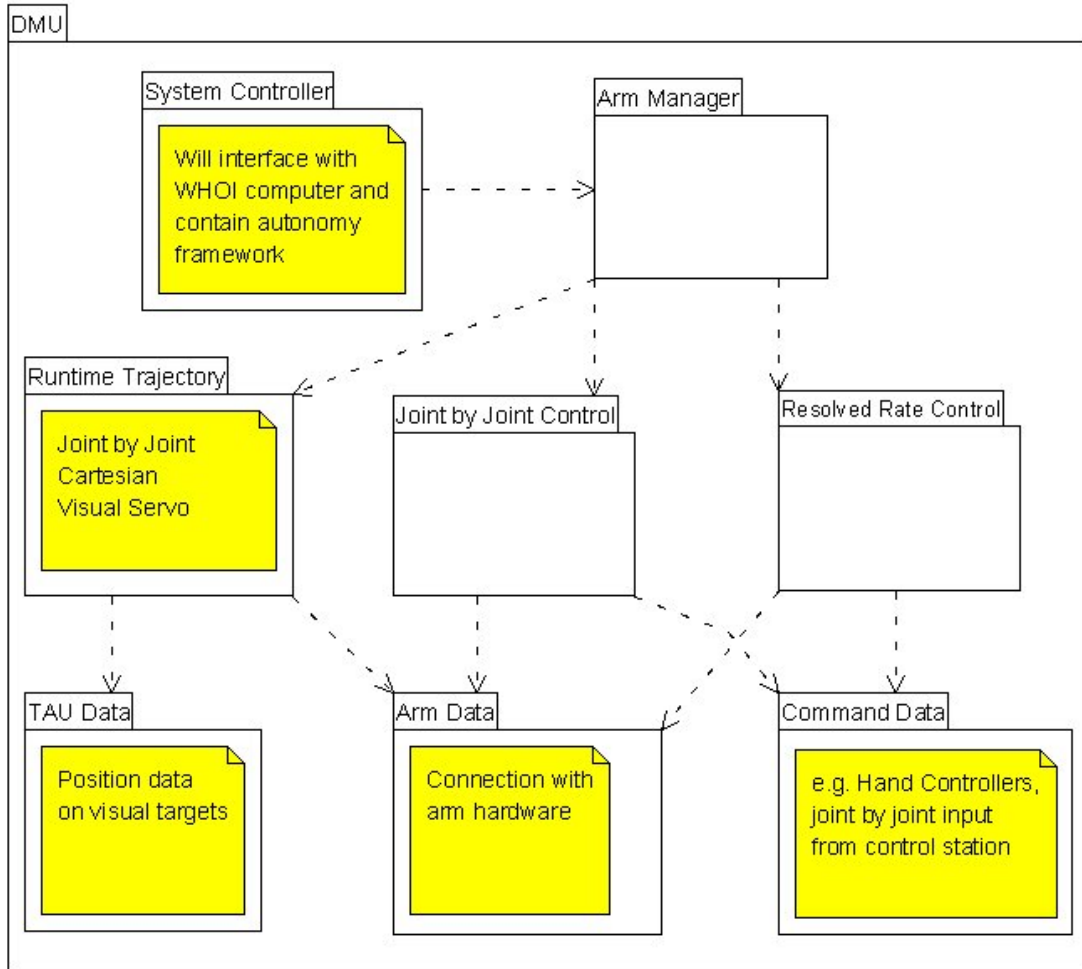


Figure 4-2: DMU Software Architecture UML Diagram

ASTEP Software Architecture

Due to the modularity of the TAU interface, future missions with the SAMURAI manipulator will be conducted with essentially the same vision software as with Ranger, with the additional capabilities present in the system controller note in Figure 4-2. In addition to manipulator kinematic and hardware specifics, the main difference

is the presence of an autonomy engine on top of the DMU. This module must issue the supervisory directives a human operator initiates with the Ranger manipulator based on knowledge of manipulator and vision system data, such as deciding upon a specific target in the manipulator workspace or realizing that no targets are reachable and thus AUV motion is necessary.

4.1.2 Vision System Modules

The operational version of the target acquisition software is split into two main modules, the primary AVATAR system responsible for converting raw images to target coordinates and TAU, the software that provides the link between AVATAR and other systems including the manipulator, AUV, and human user for lab-based tests. The AVATAR/TAU separation of functionality was created to isolate changes in the computer vision and image processing from perpetuating outside of AVATAR, and vice versa assure that the operation of AVATAR is not affected by external changes in system architecture, communication protocols, and AUV/manipulator hardware and software systems. Figure 4-3 shows a high level diagram of the vision system implemented in this thesis. The arrows imply knowledge; for instance, the Analyze module has knowledge of Common, but not vice versa. A typical target acquisition cycle consists of the DMU requesting an analysis via TAUUnit, the implementation of TAU on the DMU. TAUNet, the TAU implementation on the vision computer, handles this request by invoking AVATAR through an instance of VisionInterface, a class that implements and initializes AVATAR specifically for use through TAU. Once AVATAR has completed the analysis, the data is relayed

back to the DMU through VisionInterface and TAUNet and finally to TAUUnit. Section 4.2 discusses each of the modules from Figure 4-3 in greater detail.

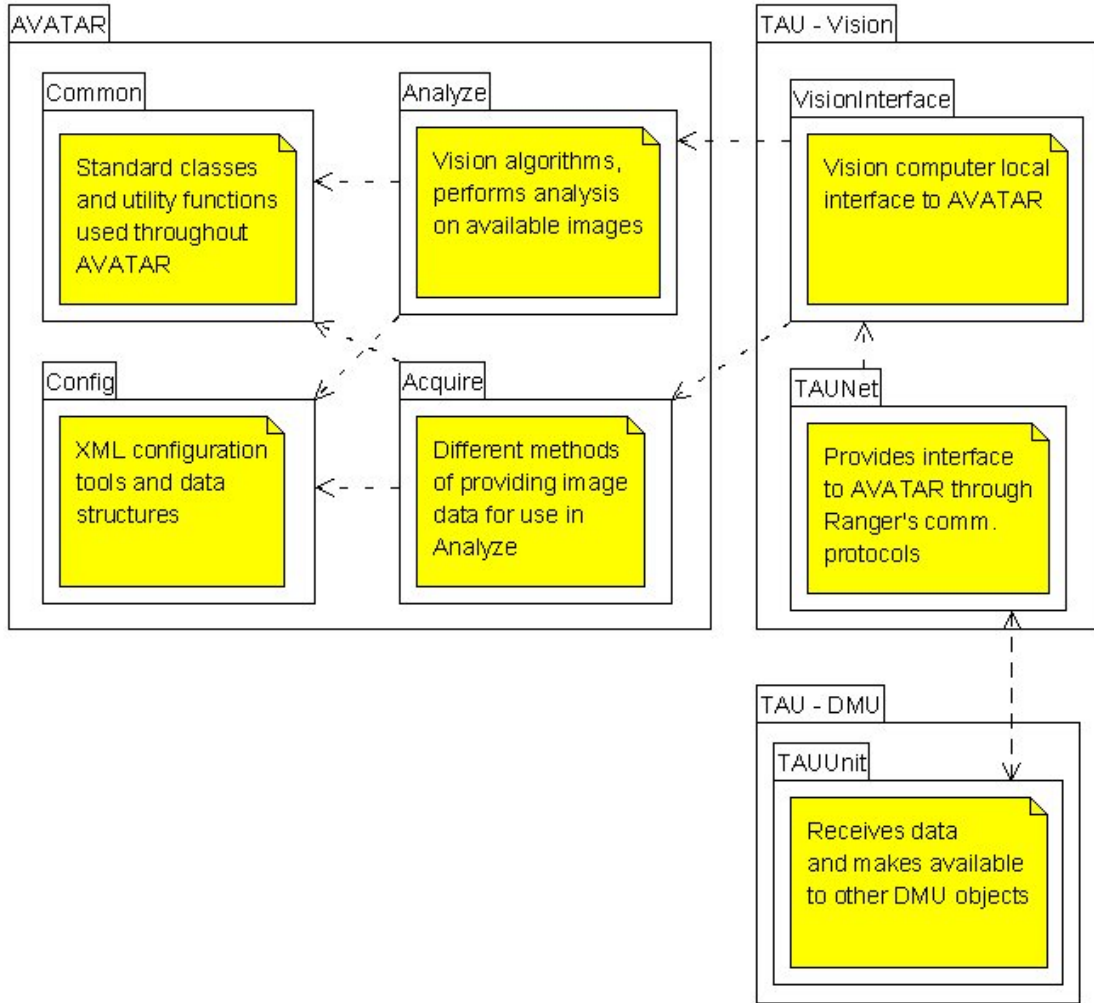


Figure 4-3: Vision System Overview

4.2 AVATAR

AVATAR is an object-oriented software product designed to be modular and reliable for fully autonomous operation. Many open-source software tools were utilized to aid in this effort, as described below in Section 4.5.2. The final AVATAR release

contains four major modules – `Acquire`, `Analyze`, `Common`, and `Config`.

The `Common` and `Config` modules hold support classes that are used throughout the rest of AVATAR. `Common` handles the storage and general manipulation of images, while `Config` reads system configuration, making sure the proper values are set and storing modified configurations as needed. `Acquire` handles the acquisition of images from the cameras and stores images in memory for future use. `Analyze` encapsulates all image processing and computer vision algorithms necessary to accurately identify and calculate desired target positions.

4.2.1 Common

The `Common` module contains the low-level building blocks for AVATAR, encapsulated in the `StereoImagePair` class. Each `StereoImagePair`, or SIP, may contain a pair of OpenCV images or a pair of virtual “images” mapped to raw memory associated with acquiring images over a camera bus. The SIP class allows a corresponding pair of images to be transmitted anywhere in the system without requiring additional overhead to keep track of left vs. right image. In addition to the SIP class, the `Common` module also houses the `StereoImagePairFileIO` class used to read and write images from a hard disk.

Apart from providing the framework for storing the images, `Common` also houses utility functions used throughout AVATAR. The image-related functions handle swapping endian values for 16-bit images in cases where this is necessary. Also included is the function for converting a Bayer pattern image to RGB format. The last utility functions create and manage the system-wide logger, `log4cpp`.

4.2.2 Acquire

The `Acquire` module is split into five classes, four related to acquiring a SIP and one containing the custom driver for firewire cameras. All classes that perform image acquisition are derived from the `AcquireStereoImagePair` class. `AcquireStereoImagePair` contains the basic functions required for the spectrum of acquire methods, while the derived classes, `AcquireSIPFirewire`, `AcquireSIPFileLoader` and `RawDataAcquire` implement the specific algorithms for acquiring images from different sources. The final class is the `AvatarCameras` class, providing access to camera hardware over a firewire bus. Figure 4-4 shows a UML diagram of the classes in the `Acquire` module describing important functions and objects.

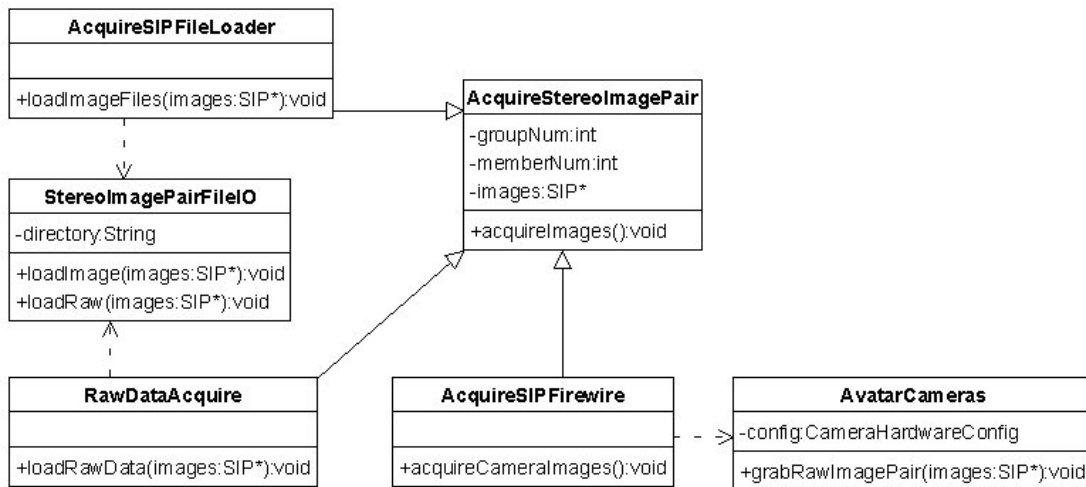


Figure 4-4: UML Class Diagram of the AVATAR Acquire Module

The `AcquireStereoImagePair` class functions enable unique identification of each set of acquired images in addition to storing the association to the SIP that receives acquired images and performs all initializations general to each

of the derived classes. Two values, group number and member number, identify each subsequent set of acquired images. Member number is incremented every cycle. Although group size can be set, group number is incremented only when a system configuration change occurs. Thus, when either the member number reaches the group size, or a configuration change occurs, the group number is incremented and the member number is reset. The default group size is 99,999 for file naming purposes; five digits are reserved in the current naming scheme.

`AcquireSIPFileLoader` is a legacy class that reads data from stored 8-bit image files, e.g., jpg or bmp, rather than acquiring real-time image data. This is a relatively simple class that interfaces with Common's `StereoImagePairFileIO` class to load a corresponding pair of images at each iteration. Since current system hardware includes 16-bit cameras and 16-bit images, this class remains unused except when performing past test cases, although implementation with an 8-bit camera system would require this class.

The `RawDataAcquire` class replaces the `AcquireSIPFileLoader` class in the new system. This class can handle raw images of any bit-depth as specified by the configuration file. However, it only supports raw images, which in this case are binary files containing only the image data. The current file naming scheme for raw files contain all information required to reconstruct the correct image size and bit-depth from the raw memory block. This acquire method is used for recreating a previous test from raw data and the system log file. See Section 4.3 for a detailed explanation on how the recreation is performed.

Once the raw data is in memory, further manipulation is required based on the

raw data format. Currently there are two possible transformations that must occur, both of which are contained in the `Common` utility functions. The first is the endian swap, necessary only if the system to which the cameras are connected has a different byte order than the firewire cameras. The second is the Bayer pattern correction used if the cameras store images in a grayscale Bayer pattern rather than a three channel RGB image (recall the definition of Bayer pattern from Section 2.2).

`AcquireSIPFirewire` is the final derived class for acquiring images. It contains an `AvatarCameras` object that handles image acquisition from the hardware. This is the most significant difference from the `RawDataAcquire` class. The other major difference to `AcquireSIPFirewire` is that it allows the user to dynamically adjust camera physical properties, e.g. exposure or white balance, to account for changing light conditions, environment changes, etc.

The `AvatarCameras` class is most complex module within the `Acquire` module, as it has to interface with the kernel modules, camera hardware, and the rest of the `Acquire` framework. The idea behind this class was to create a wrapper driver for `libdc1394`, described in Section 4.5, to make the library more suitable for autonomous operations while keeping the `AcquireSIPFirewire` class as simple as possible. To accomplish this, the `AvatarCameras` class must be able to perform two main tasks – provide a configuration interface to the cameras and store corresponding images from the camera pair into memory.

Each firewire camera has different features the user can customize, so the current feature set, enumerated in the `CameraHardwareConfig` structure, is the set of features the Point Grey Scorpion cameras use for hardware testing. Although

this diminishes software portability to a certain degree, the ASTEP target mission will only use the Scorpion cameras. libdc1394 can handle a much larger set of camera features, so in future work the programmer must add additional features to the configuration set as required and make the corresponding changes to the software.

The implementation of libdc1394 for initializing and taking images with the cameras is the most complicated portion of the `Acquire` module. The first step of the initialization process is to locate all cameras on the bus, then attempt to match the camera hardware ID values with those specified in the configuration file. It is critical that the left and right cameras are properly identified or else the stereo calculations will be incorrect. Once the cameras are correctly identified, they are initialized for the type of images recorded – a DMA format 7 capture.

There are a few pitfalls that must be avoided when locating the cameras. The two major problems come from the firewire bus after hot-plugging the cameras or restarting the computer to which the cameras are attached. If the initialization procedure attempts to locate the cameras before the firewire bus has settled after a camera hot-plug then an error will occur, but this is easily avoided by reattempting camera location after a small pause. The greater problem occurs only on machines running Timesys6, the real-time operating system to be used for the ASTEP vision computer and DMU. For an unknown reason, frequently the kernel improperly recognizes the cameras. The current fix for this problem is to reset the bus a given number of times then assume there actually are no cameras. To date, this method has worked flawlessly for finding cameras given that they are properly attached to the computer.

Once the initialization of the cameras is successful, the remainder of the libdc1394 interface is straightforward. The `grabRawImagePair` function used to perform the image grab must ping the cameras then copy the appropriate memory buffer for use in the rest of the target acquisition system. If both cameras record images, expected at this point in the process, error checking can be safely ignored and execution continues uninterrupted. However, if one or both of the cameras fails to acquire an image, the reason must be discovered before the process can continue. The only such problem encountered thus far has involved intermittent (loose cable) firewire connection between cameras and computer after initialization. Such a problem cannot be fixed via software necessitating reliable connections for ASTEP.

4.2.3 Analyze

The `Analyze` module contains all classes that perform image processing actions. A single class, `AnalyzeStereoImagePair`, performs the target acquisition analysis. `AnalyzeStereoImagePair` also coordinates each subsequent processing step, keeps track of all discovered features, performs feature matching between images, calculates the final target coordinates, and stores the data for retrieval from a TAU process. The system is set up such that an RGB SIP is necessary for an analysis to occur and targets to be located. Figure 4-5 shows the `Analyze` module class diagram.

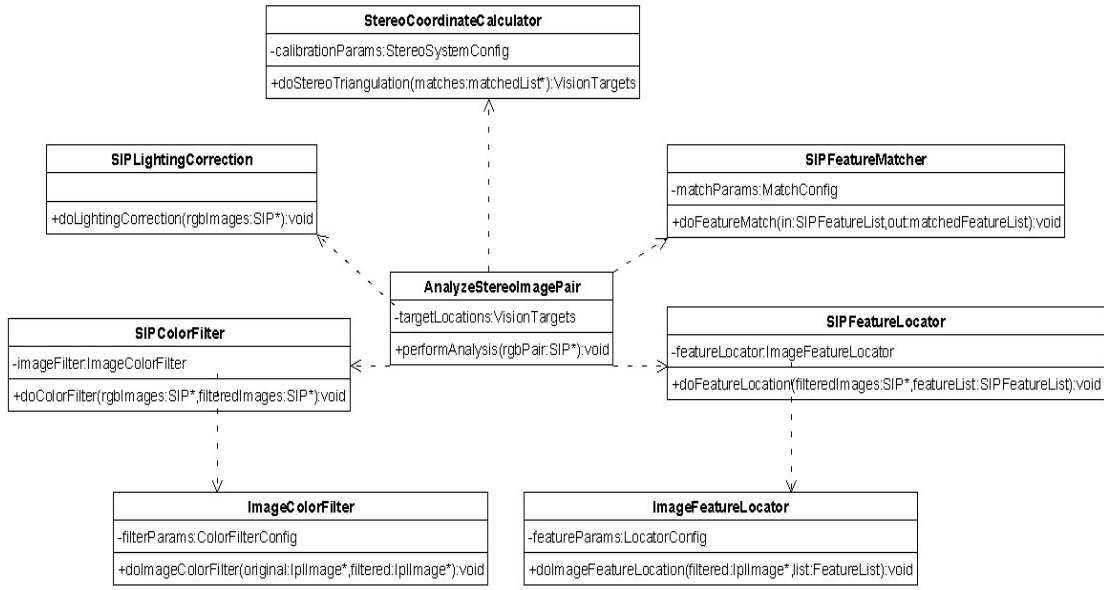


Figure 4-5: UML Class Diagram for AVATAR Analyze Module

The Analyze structure originated as a set series of image processing algorithms, implemented in classes derived from the ProcessStereoImagePair class, which involves receiving a SIP as input and providing another SIP as output. However, to decrease execution time by reducing number of calculations, certain processing algorithms were not implemented depending on environment lighting and the visual uniqueness of target from background. For instance, during laboratory testing with the rubber ducky target, the order of processing start with the color filter implemented in the SIPColorFilter and ImageColorFilter classes, followed by feature extraction implemented in SIPFeatureLocator and ImageFeatureLocator classes. This operational environment does not necessitate the use of the color correction algorithm used for modifying images acquired in low-light environments with a distinct lighting pattern, or the erosion with feature-AND restoration required with visually nondescript target fields. In *O-*

notation [35] all the lighting correction and color filtering algorithms have complexity $O(mxn)$, where m is image height and n is image width, thus by leaving out unnecessary algorithms, the overall complexity is substantially decreased. Testing performed on an Intel Core-Duo Mac Mini running at 1.8GHz with 1GB RAM had an average execution time of 1.22s for the `Analyze` routine without any form of lighting correction. The addition of both lighting correction routines increased this execution time to 1.36s. Not shown in Figure 4-5 is the flexibility to insert additional image processing classes, such as a lighting correction class. The architecture is designed to facilitate insertion, removal and modification of the `ProcessStereoImagePair` derived classes, given careful analysis since additional complexity may significantly increase execution time.

The `SIPColorFilter` class, after being passed the appropriate SIP to filter, sequentially performs the `ImageColorFilter` processing necessary for each image, with separate filtering functions to handle 8-bit and 16-bit images. Recall that the color filtering process was described above in Section 3.3.

Following color filtering, the remaining image features must be extracted. The `SIPFeatureLocator` class contains `ImageFeatureLocator` objects for each of the right and left images, which can then extract the required data for the feature matching process. Output from the `SIPFeatureLocator` class takes the form of the `SIPFeatureList` class, STL vectors of required data for continued analysis.

The next step of the analysis is to correctly match extracted features between images based on values from the feature locator process. Relative position matching

works through a recursive procedure that matches each feature in one image with another feature in the second image. This matching process is handled within the `SIPFeatureMatcher` class, capable of implementing a variety of matching algorithms, but focusing on the geometric constraint algorithm detailed in Section 3.4.1. Output from the `SIPFeatureMatcher` class takes the form of a `matchedFeatureList`, an STL container holding all relevant data.

The final image processing task takes place in the `StereoCoordinateCalculator` class, which calculates target coordinates based on the intrinsic and extrinsic calibration parameters of the stereo vision system. This procedure is based directly on the `stereo_triangulation` method from [3] but is converted into C++. Details of this algorithm were provided in Section 3.4.2.

Completion of the `StereoCoordinateCalculator` process populates a `VisionTargets` object, part of the `VisionInterface` library, with all target data needed external to AVATAR, such as 3-D position, centroid coordinates, and object area. By providing sole access to AVATAR data through this `VisionTargets` object, AVATAR is isolated from all other interfaces.

4.2.4 Config

Both `Acquire` and `Analyze` modules contain numerous parameters that must be modified to reflect changes in the operating environment, target properties, and camera systems. The `Config` module facilitates changing these parameters via an external public interface. An XML configuration file contains all relative information

to completely fill the necessary values for the AVATAR system. The handling of the XML files is mainly done using an SSL created XML Config tool based on TinyXml [36], a small, easily configurable XML parsing tool, and Boost [37], a set of open-source C++ libraries.

The four configuration structures are `CameraHardwareConfig`, `TargetConfig`, `MatchConfig` and `StereoSystemConfig`. `CameraHardwareConfig` contains data about the properties of acquired images (height and width in pixels, bit depth, channels, Bayer pattern flag, and endian swap flag), camera hardware IDs, and initial values for the camera feature set including flags for the use of automatic hardware modifications where applicable. `TargetConfig` contains configuration values for the color filters and feature locators as well as search window specifics, separated into multiple sections for different target types. A future modification will implement multiple window capabilities to enable focus of attention on multiple regions of interest. The `MatchConfig` structure contains target geometric information and epipolar constraints used during the feature matching process. `StereoSystemConfig` contains all calibration values, intrinsic and extrinsic, necessary for the stereo triangulation calculations. The specific values were discussed in Section 2.1.

In addition to header files for configuration data structures, the `Config` module also includes utility functions. These functions provide an interface for reading and writing configuration files from either a file on disk or a *string* variable. Due to conflicts with `libdc1394` all XML and Boost code must be in separate libraries from modules containing `libdc1394`-dependent code. The configuration utility

implementation is contained within the TAU interfaces, but it was also used widely throughout AVATAR unit tests.

4.3 TAU

The Target Acquisition Unit, TAU, describes the collection of software used to interface manipulator systems (and in the future the WHOI AUV computer) to AVATAR. There are four levels of TAU software. First is the wrapping of AVATAR into a single public interface, the `VisionInterface` class. The second layer is the `TAUNet` application that provides network access to AVATAR through the `VisionInterface` class. Nominal execution of `TAUNet` does not provide public access to the vision system – another interface must be present to send the appropriate messages to `TAUNet`. These interface programs are based on the `TAUUnit` class that provides basic interface messages to `TAUNet`, useful for non-human control, while derived classes can be outfitted with greater functionality, such as displaying images in the `TAUGUI` implementation or simply saving images to the disk in the `TAUTUI` implementation.

4.3.1 VisionInterface

The `VisionInterface` class consists of a generic header file with multiple implementations, each for a different version of AVATAR's `Acquire` methods. By providing this standard interface, the higher levels of TAU can use any method seamlessly. The two currently implemented versions of `VisionInterface` are `TAU_1394` and `TAU_Raw`, corresponding to `AcquireSIPFirewire` and `RawDataAcquire`, respectively. This implementation with multiple source files

for the same class allows changing the object file used during linking rather than modifying which class is used in the source and recompiling the entire class.

The main differences between `TAU_1394` and `TAU_Raw` stem from the fact that `TAU_1394` runs new analyses each cycle, whereas `TAU_Raw` must parse previous tests' log files, read their images, and reanalyze these images. Each method has inherent difficulties. The initialization of the firewire cameras in `TAU_1394` is achieved as described above. On the other hand, `TAU_Raw` must parse large log files and requires a great deal of preparation to ensure all images, log files and configuration files exist and are in the correct directories prior to execution.

Other than the initialization, which only changes the method of acquiring images, there are limited differences between the two classes, such as the method to change the configuration. To make changes to both versions simultaneously, a single source file contains code shared between both implementations, while the specific `TAU_xxx.` file contains the functions that differ.

4.3.2 TAUNet

The `TAUNet` application provides a single program that executes continuously on the vision computer, waiting to receive commands over the network to start/stop target acquisition. `TAUNet` is the main program for executing the desired Vision Interface, thus providing a back-end public interface to AVATAR over a network. To provide access to the vision system, an implementation of the `TAUUnit` class that communicates with `TAUNet` is discussed later in this section.

`TAUNet` utilizes Ranger's communication protocol for all message handling

activities. A separate channel and set of messages were developed specific to TAU to facilitate operation of the vision system. The actions required for nominal vision system operation consist of starting or stopping a continuous search for targets, performing a single “snapshot” of the current view to determine target locations, and retrieving the target coordinates from the most recent analysis. If the designated task is based on a single system configuration, these operations are all that are necessary for the entire mission profile.

The remainder of the tasks performed by TAUNet provide capabilities for more complicated sampling tasks. Still necessary for autonomous operations is the ability to send a new configuration to the system. This provides the interface for changing any aspect of the system – camera parameters if lighting is different than expected, the image filter parameters to search for a different target type(s), or modifications to the camera or stereo system properties, among others. In addition to changing configuration is the ability to retrieve the current configuration, most useful during supervised autonomy so the operator can determine what changes to make.

Finally, TAUNet can retrieve any desired set of images from AVATAR, such as the original images, filtered images, or images marked with located and matched features. To ease network load associated with transmitting large data streams, images are converted to .jpg format since the human user views them rather than precisely analyzed with vision algorithms. This compression step reduces image size from 3MB for an 8-bit .bmp or close to 8MB for the 16-bit raw 3-channel image to approximately 50kB for an 8-bit .jpg. OpenCV’s file saving only handles 8-bit images at this time, so compressed 16-bit images are not available.

Although the current capabilities of TAUNet are limited to those discussed here, the system is designed to facilitate extension. Further testing may identify the need for immediate access to vision system internals, in which case real-time access by the DMU or human user to more features will be an indispensable tool.

4.3.3 TAUUnit

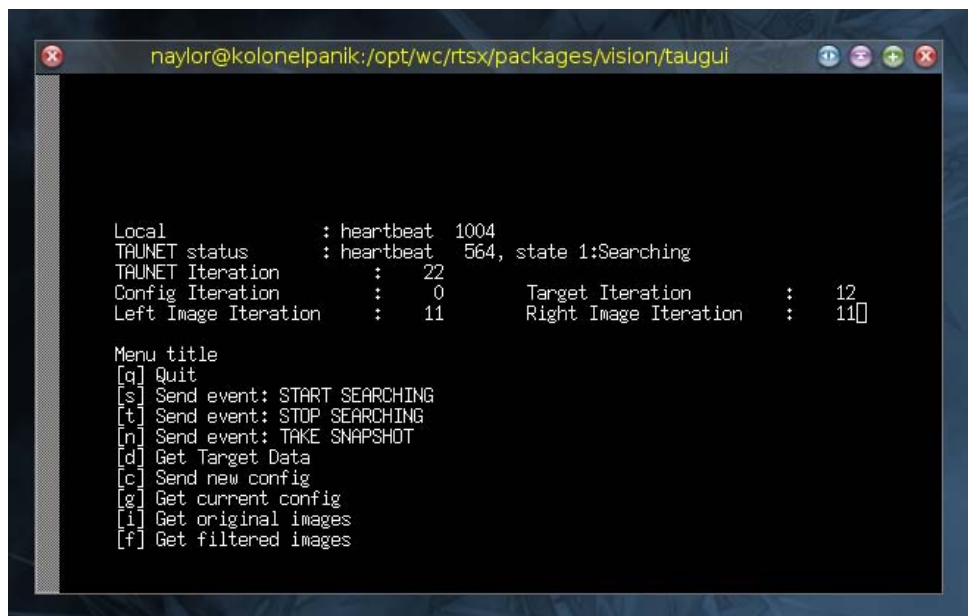
The next level of interface to the vision system is the TAUUnit class. This class provides message transmission (TX) and receiving (RX) capabilities to connect with TAUNet. Where the TAUNet application must be executed locally on the computer with either the cameras connected or previously recorded images, a TAUUnit class can be created in any executable, such as the DMU, then invoked to send commands or retrieve data from the vision system.

In addition to providing the public interface to TAUNet, the TAUUnit class also maintains a log of action execution timing information. For instance, if a continuous search for targets is underway the system must ensure the target data stored in the TAUUnit object is the most recent data. To allow this check, TAUUnit stores the current iteration number of the vision system, the iteration of the vision system when target data was last retrieved, as well as iterations for when the configuration and images were last retrieved.

4.3.4 TAUGUI

The highest level of interface to the vision system is a subclass of TAUUnit, the TAUUnit_GUI. There are two executable programs that instantiate this class: TAUTUI, providing a text-based interface shown in Figure 4-6, and TAUGUI,

providing a wx-widgets GUI interface. The foremost difference between `TAUUnit_GUI` and its base class is the re-implementation of the function for handling receipt of an image from `TAUNet`. This subclass can be populated with additional capabilities deemed necessary for human control of the vision system, thus keeping the `TAUUnit` base class bereft of information superfluous for autonomous control. The final implementation of the GUI is still under development, so all testing was performed with a fully-functional `TAUTUI`.



```
naylor@kolonelpanik:/opt/wc/rtsx/packages/vision/taugui

Local          : heartbeat 1004
TAUNET status  : heartbeat 564, state 1:Searching
TAUNET Iteration : 22
Config Iteration : 0          Target Iteration : 12
Left Image Iteration : 11      Right Image Iteration : 11

Menu title
[q] Quit
[s] Send event: START SEARCHING
[t] Send event: STOP SEARCHING
[n] Send event: TAKE SNAPSHOT
[d] Get Target Data
[c] Send new config
[g] Get current config
[i] Get original images
[f] Get filtered images
```

Figure 4-6: TAUTUI Interface to AVATAR

4.4 Visual Servo Controller

Initial tests with the Ranger manipulator were performed “open loop” with respect to the vision system: a target was visually located with respect to the manipulator base frame, then the manipulator maneuvered its end effector to this position for sample target acquisition. Such a procedure presumes highly accurate camera-manipulator

registration and minimal motion of the target relative to the AUV. In practice, disturbances (e.g., currents) can induce motion of the AUV and/or target, and registration parameters may not be sufficiently precise for reliable sampling.

To mitigate the effects of these error sources, AVATAR and the DMU were augmented to support a visual servo controller for true “closed-loop” target acquisition. In this paradigm, the vision system computes both the sampling target position and a visually-distinct region near the manipulator wrist, thereby enabling computation of the relative offset between the end effector and target in the same (camera) reference frame. By using this offset as an error to be minimized over time, the end effector can be collocated with the target in a truly closed-loop manner that does not require accurate camera-manipulator registration.

4.4.1 Visual Servo Software Integration

Successful integration of the visual servo process requires substantial augmentation of the vision system. Filtering for multiple target types requires target filters be specific to a single target type to differentiate between similar features. Initial selection of the primary targets for sampling and arm tracking is now required – the arm as detected by AVATAR must correspond with telemetry data internal to the manipulator system, while still allowing for errors introduced by the camera calibration and registration. Previously temporal tracking was irrelevant as only a single snapshot of target data from the vision system was utilized. Figure 4-7 shows the state machine for the visual servo controller.

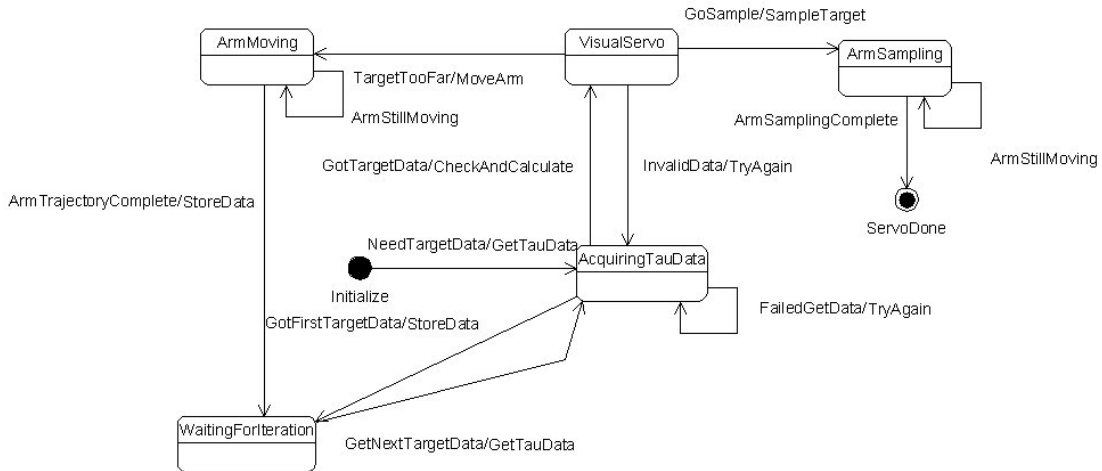


Figure 4-7: Visual Servo Controller State Machine

The visual servo controller begins in an initialize state waiting for its first set of data from TAU. This data set must contain the location of both a sampling target and arm target. The main execution cycle requests vision data, waits for it, calculates an iterative move based on the target position relative to the manipulator, initiates this move, and repeats until the error is below a specified threshold. This implementation is not a fully closed-loop visual servo controller where the vision system is actively tracking the arm and providing feedback at a sufficient frequency. Rather, it is more of an iterative series of open-loop moves. For example, the estimate of arm and sample target position takes approximately 1 second to accomplish, which will result in calculation of a motion vector for the manipulator tool position with a given magnitude (5cm during initial testing). This motion occurs over a specified period, which happened to be 5 seconds during testing. Once this motion has stopped another visual position estimate is performed. This entire cycle takes over 6 seconds to complete. For the final AUV system, limited by the strobe refresh rate, the entire

process could be scaled to fit exactly within the given time constraints – vision estimate plus manipulator motion time equals strobe recharge. A system with faster frequency would provide no overall benefit.

Two states exist for waiting until manipulator motion is complete. `ArmMoving` describes the state when the arm is moving towards the target, but the target is still too far from the manipulator to sample. `ArmSampling` is the state when the calculated manipulator position is within a specified distance from the sampling target and must now execute motion required to physically capture the target. In the case that errant target data is provided from TAU to enter the `VisualServo` state (i.e. arm position estimate too far off or sampling target moved too much) then a new set of data will be requested from TAU.

4.4.2 Visual Servo Control Law

Closed-Loop Control of Ranger Manipulator

A block diagram of Ranger's inner control loop is shown in Figure 4-8. Initially, a desired motion, $\mathbf{r}_{desired}$, either a joint-space or Cartesian position or velocity vector, is fed into the DMU. The DMU control loop operates at 125Hz, with each cycle calculating a new set of desired joint angles, $\mathbf{q}_{desired}$. These desired joint angles are fed into each LPU, which generates commanded torque values, $\tau_{commanded}$, for each joint at 750Hz. The LPUs use a PD (proportional-derivative) control law based on position and velocity data from the encoders. These commanded torques are sent to motor controller boards to generate motor current values, i_{motor} .

Motion of the manipulator is converted into joint angles by optical position encoders. The actual joint angle values, \mathbf{q}_{actual} , are utilized by both the LPU and DMU to close the control loop. While the LPU requires the joint feedback at each iteration, the DMU controller only uses the actual joint angles to generate the initial set of desired joint positions through inverse kinematics. The DMU then bases all subsequent desired joint position calculations on the previous iteration's desired joint position.

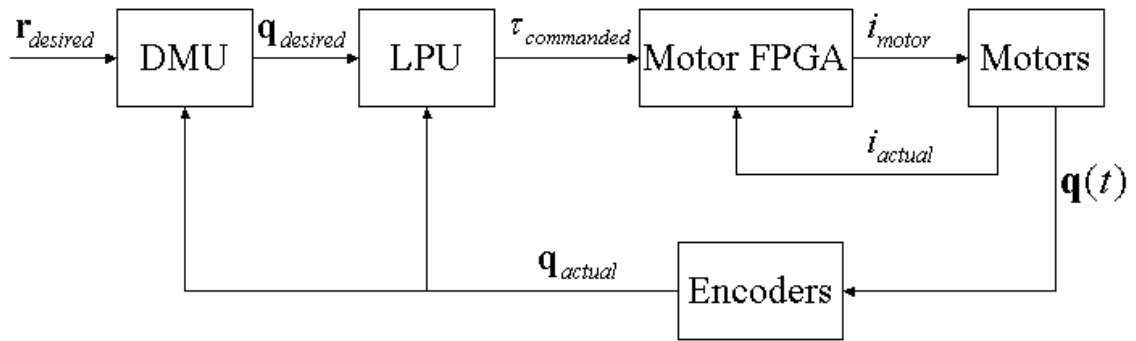


Figure 4-8: Ranger Control Loop

Integration of Visual Servo Data with Ranger Control System

Figure 4-9 shows how vision frame Cartesian position estimates of the manipulator and sampling target, ${}^v\mathbf{p}_{arm}$ and ${}^v\mathbf{p}_{sample}$, respectively, are combined with Ranger's telemetry data to determine a new arm position, ${}^0\mathbf{p}_{servo}$. There are a few key integration items of note for the system to function properly. The first is the translation from ${}^0\mathbf{p}_{arm}$ to ${}^0\mathbf{p}_{tool}$ -- changing the manipulator reference point from the visually distinct target to the tip of the end effector. The rotation matrix ${}^0_T R$ is provided by the manipulator forward kinematics present within the DMU and is based

on joint positions provided from the motor encoders. Knowledge of arm telemetry is also required for the final calculation that transforms desired arm motion ${}^0\mathbf{e}_{motion}$ from an iterative move into the Cartesian position ${}^0\mathbf{p}_{servo}$, the input into Ranger's control system. Once ${}^0\mathbf{p}_{servo}$ has been commanded as a desired position, the visual servo system pauses until the arm motion has completed.

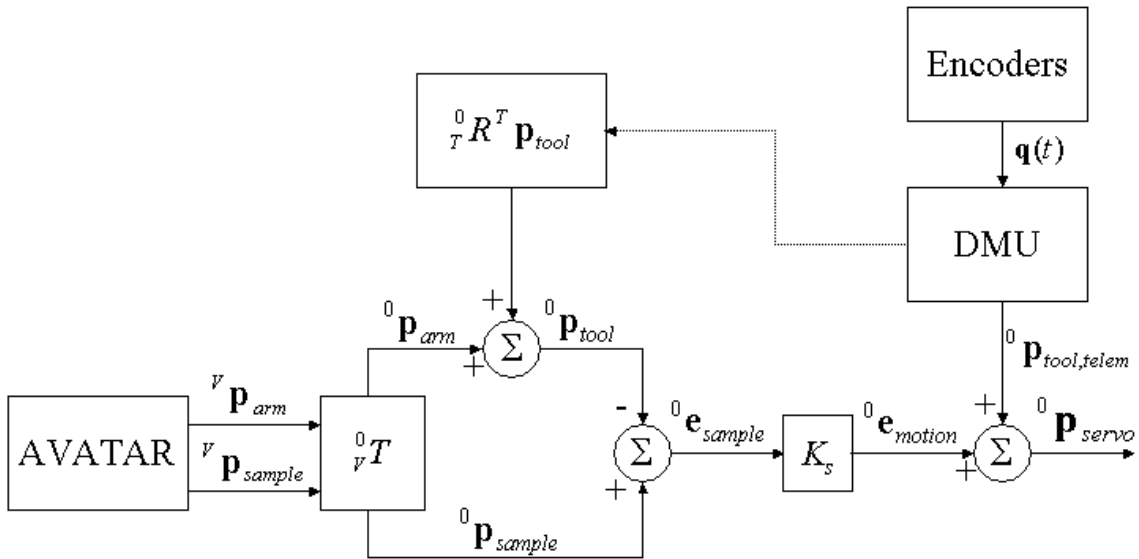


Figure 4-9: Open-Loop Visual Servo Diagram

The goal of the visual servo system is to drive the value ${}^0\mathbf{e}_{sample}$ to 0, which would mean that ${}^0\mathbf{p}_{tool} = {}^0\mathbf{p}_{sample}$ and the sampling task could be completed. The value K_s in Figure 4-9 is a scaling parameter to adjust the magnitude of arm motion based on several factors. If K_s is small then more iterations of the visual servo routine are applied, which will more closely resemble a fully closed-loop controller, while $K_s = 1$ results in a simple dead-reckoning attempt at sampling. To get the most

benefit out of the vision data while maximizing periods of arm motion, K_s should be tuned so that the calculated motion can be safely executed prior to the next strobe cycle. For example, with the manipulator stationary, the vision system acquires images when the strobe flashes. Approximately 1s later position data is provided to the manipulator, so motion should be scaled such that it is completed within the next 1.5s (assuming a 2.5s strobe recharge). Further testing of the system will dictate appropriate values for K_s in a fully-lit environment. The final value of ${}^0\mathbf{p}_{servo}$ is a Cartesian position of the end effector tool tip, with orientation remaining constant from previous telemetry. This value is then used as input to the DMU control loop, recall $\mathbf{r}_{desired}$ from Figure 4-8.

4.5 Software Utilities

Two sets of software utilities provide a human interface to the vision system as well as management and validation of the entire software suite. The first set is a collection of custom utilities created within the AVATAR/TAU framework to allow for easy integration with the firewire cameras, image acquisition during calibration, filter creation, and other similar tasks. The second set of software engineering tools is used for version control, integration of unit tests, defect tracking and other tasks to help manage and validate the software as it is being developed.

4.5.1 Custom Utilities

A set of utility programs provide the ability to generate developmental data not required by the vision system for deployed operation. These tools generate

configuration data that remains constant during application of the vision system to a specific target area, or provide knowledge that enables the system to choose between predefined configuration values. The rest of the tools provide the ability to test specific aspects of the system without requiring the entire code-base to run unit tests, such as testing that the firewire cameras are properly attached and the kernel driver is functioning.

libdc1394 and Firewire Camera Custom Driver

This system utilizes the open-source libdc1394 library [38] to communicate with the firewire-based Scorpion cameras. An interface driver was written to streamline the libdc1394 functions with the specific Scorpion cameras used in the research, as well as with the data storage methods. The implementation of this driver is within the `AvatarCameras` class.

The `AvatarCameras` driver allows certain configuration parameters to be set through XML files, but all parameter values that will remain constant throughout AVATAR tests are hard-coded in to minimize complexity. For instance, the resolution and acquire methods, important for switching between different cameras, or for using cameras with many pre-defined states, are set to constant values. Other camera parameters, such as exposure and white-balance are set through configuration variables. The underlying idea behind this driver-on-a-driver is to decouple the involved aspects of the camera initialization process from the configuration of the target acquisition system by the user.

Camera Interface Utilities

The Point Grey Scorpion cameras have the ability to automatically determine the proper “feature set” to apply while recording images. This feature set consists of values related to camera exposure, shutter time, white balance, etc. However, since AVATAR heavily depends on consistent values and auto-adjust does not provide sufficiently fast or repeatable value sets, applying a known feature set will ensure repeatability between consecutive analyses. Problems can arise if the auto-adjust is constantly altering values, such as during the light-dark pattern seen with a strobe light; in some cases the white balance is mistuned such that the entire image is discolored. To handle the selection of a camera feature set, a program was created to record images until steady state values are reached by the automatic adjustment ability, and then if the images are of good quality the feature set is stored for use with the vision system.

Another utility facilitates proper testing and initialization of the vision system through a set of programs used to acquire images directly from the cameras and save them to disk. Both GUI and text-based programs provide critical functionality for vision system development. A text-based image acquisition utility has been developed to ensure that the cameras are functioning correctly with respect to the rest of the computer hardware, permissions are set correctly on the devices, and the kernel driver has successfully recognized and is communicating with the cameras. Since this program is independent of AVATAR, TAU, and the rest of the AUV/DMU software system, it can be run without the overhead associated with the full system.

Similarly, a standalone GUI-based utility serves as the main method for acquiring calibration images. Since ensuring the entire checkerboard image is present within the field of view of both cameras is necessary to the calibration procedure, this program facilitates that process through real-time presentation of acquired images to the user. Also, the GUI update of this program is useful in camera placement, as the user can immediately ensure the cameras are placed in the desired position. Similar to the text-based program, this program is independent from the remaining software architecture, so it can execute on any computer given the appropriate Linux software.

Filter Creation Utility

To enable quick creation of color filters, a program was created to display an image and allow the filter parameters to be adjusted in real-time. This `filtergui` program is the counterpart to the MATLAB-based filter creation procedure discussed in Section 3.3.1, but without the requirement of executing MATLAB. Via the OpenCV Highgui interface, a previously recorded image is displayed with a set of slider bars to adjust filter values. Through this interface, a user is able to immediately examine output of the image filter when applied to a specified configuration. However, when dealing with multiple similar target types or targets with similar properties to the background the MATLAB analysis is a much more reliable tool since it allows the user to extract exact values rather than continuously modify the slider bars until the output appears correct. Figure 4-10 shows a screenshot of the `filtergui` program in action.

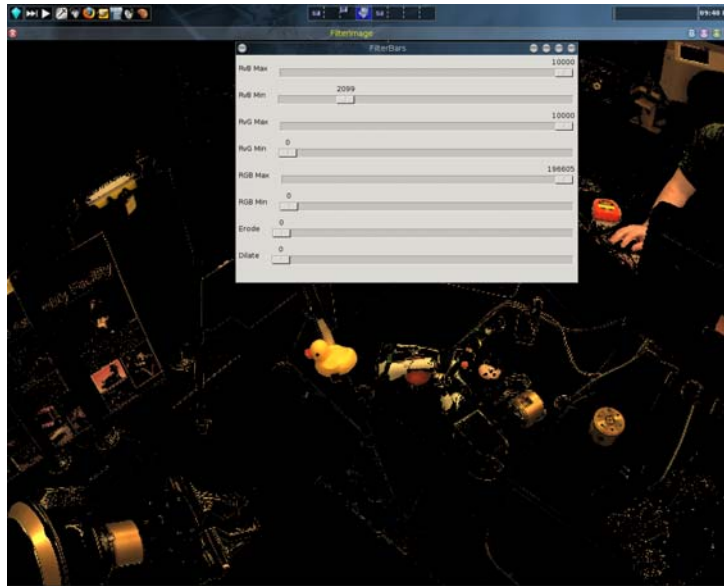


Figure 4-10: filtergui program in the process of creating a target filter

4.5.2 Software Engineering Tools

To ensure proper software functionality, steps were taken to minimize novice programming mistakes. Attention was given to const-correctness when dealing with references (allowing values to be modified only if they should be modified), ensuring zero memory leaks, proper use of inheritance and polymorphism of classes, and many other common C++ issues where problems can easily arise [39][40]. Although attempts were made to follow proper programming tactics within the software itself, the use of external analytical tools can greatly increase the ability of a programmer to create reliable code. Comprehensive documentation of the system, logging of programmatic state and internal data during execution, frequent system-wide unit testing and coverage analysis in addition to continuous integration are all built-in to the SSL software system to further validate and accurately profile all software.

Documentation and Logging

For a software system to be useful to anyone besides the original programmer, comprehensive documentation of the code is a must. In addition, the ability to track bugs and other defects in a central database eases collaboration in a multi-programmer environment. Another important aspect of documentation is data logging during program execution, as this allows a programmer to track down bugs as well as re-create a previous execution or state.

Writing software is hard. Writing software with sufficient, understandable documentation thus allowing other programmers to easily interface with your software is more difficult. Doxygen [41] is a tool that parses software to automatically create legible, comprehensible documentation in the form of hyper-linked html pages for on-line viewing or LaTeX files for off-line use. By commenting the software in a specific manner, the programmer can almost effortlessly oversee the creation of the Doxygen documentation. Doxygen is used to provide explanations for files, classes, variables, functions, etc. as necessary throughout the software system, in addition to providing longer, more detailed pages describing in greater depth how to use different software modules, known problems associated with the system, test results and anything else necessary for the continued use and portability of the software.

In concert with Doxygen, which creates its own UML diagrams, Gentleware's Poseidon for UML tool [42] is frequently used to visually plan and describe object-oriented software. Tracking down unwanted dependencies and other unforeseen

architecture mistakes is much easier with UML diagrams. Poseidon is compliant with UML 2.0 and was used to create the diagrams used throughout this thesis.

Creating useful execution logs assists experimenters in continued development of products in addition to providing a useful tool for debugging and fixing problems. Log4Cpp [43] is a tool that provides an easy-to-use interface with valuable capabilities. It is also reputed to be efficient so it will not greatly affect execution time. The system is set up on a priority basis allowing superfluous debug messages to be ignored during actual runs, leaving only the most important messages to be logged, or, during development periods, leaving all messages to be logged. A simple flag set at the beginning of program execution determines the logging priority.

System-wide Unit Tests and Memory Profiling

One important consideration for large-scale software creation in a multi-programmer, experimental laboratory setting is ensuring that the software will always perform as desired. Providing a framework to allow unit tests at compile time, during a daily/nightly build or any other time is necessary so that bugs or other unexpected behaviors are caught before an actual test is run with hardware. Ensuring a large percentage of code coverage as well as testing for memory leaks within the unit tests further validates software for implementation.

CxxTest [44] is one of many unit test frameworks, and is the current tool used system-wide at SSL. The first major use of unit testing is an assurance that after changes have been made to a specific code section, the program will continue to behave correctly, or, if it was incorrect before, it will exhibit correct behavior in the

future. By using the CxxTest selection of “assertions” the programmer can make sure functions return correct mathematical values from a complex calculation or another form of successful operation, in addition to testing whether the code behaved properly when provided inaccurate, incompatible, or otherwise erroneous input. In essence, a wide set of test cases better guarantees proper software execution during critical test periods. When used in conjunction with `gcov`, described below, the programmer can be confident that the software is thoroughly tested prior to final execution.

Test coverage programs are widely used throughout the software industry to analyze programs. The two most important reasons to apply a test coverage analyzer are to ensure not only that the majority of the software is validated in unit tests, but also to provide output that gives the programmer the ability to focus on optimizing the sections of code executed with the highest frequency. With the knowledge gained from such a utility, a programmer can have much more confidence in their software.

The software created for AVATAR, SAMURAI and other SSL projects now utilizes the `gcov` [45] utility, which works in concert with GNU CC. After integration into the build system, `gcov` automatically analyzes all software to provide the coverage output mentioned above. However, a programmer cannot easily parse the output provided by `gcov`, thus a shell script is used to sift through the `gcov` output and create html output as part of the Doxygen documentation.

A `gcov` results page is created that links all software modules that currently have active results from testing. For each file, a results page similar to Figure 4-11 is created to visually convey concise results of the coverage testing. The color-coded results quickly display to the viewer the extent to which each source file is tested by

the unit test system, along with the test time and date and the architecture/OS of the computer that ran the tests. A red highlight means less than 33% of the code is tested, yellow means 33% - 66% while green means greater than 66% is tested. The names of the source files in this list are links to a text file showing the source file with an execution count at the beginning of each line. Through this output, a programmer can ensure that all possible execution paths have been tested prior to deployment of the system.

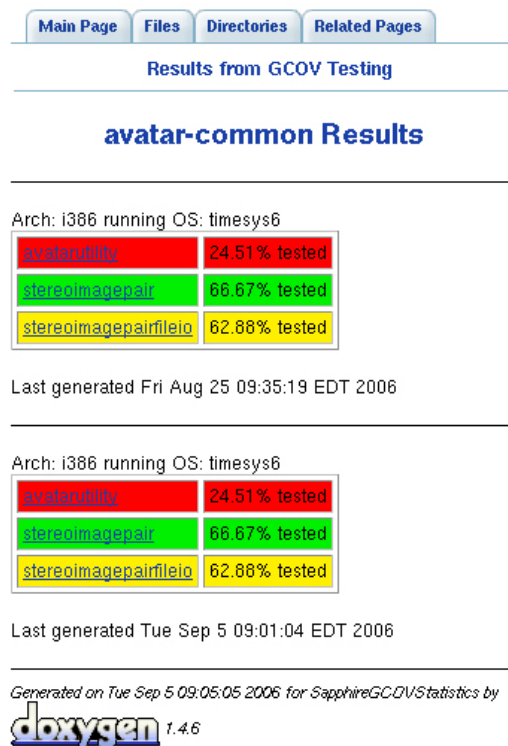


Figure 4-11: Colored output of gcov results for each specific file

A memory leak of any magnitude can have drastic consequences during program execution that must last for hours, continuously performing operations while slowly (or quickly) exhausting system memory. Large memory leaks will often have more immediate consequences as they tend to cause program termination much

sooner. Detection of all memory leaks within a software system can be a daunting task, especially if a programmer only has the source code available. Fortunately, there are tools that can run on top of a program to alert the programmer of possible memory leaks and provide line numbers and source files to help locate the offending code. The Valgrind [46] memory profiler was used for the software associated with this thesis.

Continuous Integration and Defect Tracking

Amassing these software validation tools, but only infrequently using them, destroys many of the benefits they provide. Another set of tools are applied at SSL to ensure that all software is continuously integrated and validated after every change, while supplying version control and correlated defect tracking.

The open-source Subversion [47] tool is used to handle version control. Subversion provides many benefits in a multi-programmer environment, such as allowing each user to have a separate working copy to make individual changes, and allowing a programmer to easily revert to a previous version of software or view differences between two versions. With good server backups, Subversion's code repository also provides reliable software backup.

Subversion can be coupled with Trac, a web-based software management and defect tracking system [48]. Trac allows users to view a timeline of recent software revisions to the Subversion repository, in addition to creating "tickets" to notify another programmer, or themselves, of possible bugs in the system due to a recent change that was made or a possible enhancement that would benefit the overall group.

Also coupled with Subversion and Trac is the CruiseControl continuous integration tool [49]. CruiseControl continuously performs entire system builds, with the corresponding unit test cases, every time a programmer commits a software revision to the Subversion repository. If no revisions were made, a mandatory daily build is still performed to ensure system stability. This provides immediate feedback on any problems that were caused by recent changes. Provided comprehensive system-wide unit testing with a good percentage of coverage, successful builds throughout the day can help ensure proper execution during tests, demonstrations, and field-trials. This idea of, at minimum, a full daily build with test cases is a widely accepted concept that provides immense benefits to system development [50].

Example Benefits of Software Engineering Tools

By looking at output logs from the build system, dating back seven months, the benefits of the various tools implemented within the system are immediately evident. Within the logs, 484 separate full-system builds and tests were performed in that time period, although the total number is higher due to sparse periods when the CruiseControl (continuous integration) software was offline. These logs show that 45 builds failed due to problems specifically with the AVATAR/TAU code, while nearly 4 out of 5 builds has some problem associated with it.

Out of these 45 failed builds due to the vision software there are only 7 actual errors, but they were not fixed before the next build occurred. The trend of the build errors is in the earlier dates the problems were associated more with problems relating to cross-platform compiles or missing/undefined references while linking. The later

errors are due to the unit tests failing. This was happening because configuration files and other parameters were being tuned during laboratory tests, which was causing changes in software behavior that, in turn, caused changes in unit test system output. The system output is checked with a reference file during the unit tests, so when the output changes, the tests fail until an updated reference file is provided. In these cases the tests were still performing as they should, but the programmer is alerted because the output has changed, which could be a sign of larger problems.

An instance when Valgrind, the memory profiling tool, was especially useful was during the transition from 8-bit to 16-bit images. At the time the software was under development OpenCV did not handle saving and loading of 16-bit images to the hard disk, so custom software was developed to facilitate these operations. These changes were widespread throughout the system where any specific image handling operations were occurring. Rather than OpenCV providing memory handling, it had to be accomplished manually. A single omission of freeing dynamically allocated memory caused system-wide failure during unit tests and would have been quite difficult to immediately track down without the assistance of the tool. Similar problems occurred during the transition from dynamically allocated arrays to STL variables for storing data within the system. These problems caused significantly smaller memory leaks (on the order of bytes rather than mega-bytes) and were not as evident as the 8MB memory losses for each 16-bit RGB image. Once again, the use of the memory profiler allowed the problems to be tracked down much more quickly, as well as pointing them out prior to a runtime crash.

Chapter 5 Experimental Platform and Test Plan

Due to the extensive suite of software and hardware required to fully test and validate the vision system, development of a proper test matrix is critical for success. This chapter will focus on the major considerations taken into account during vision system testing, including the vision hardware and manipulator systems, then summarizes a test sequence aimed at comprehensive system evaluation and validation within practical availability and functionality constraints.

5.1 Vision Hardware

The quality and performance of the vision system hardware is directly related to successful system operation. Below, the two sets of cameras used during different phases of testing will be discussed, including discussion of factors that influenced specific camera choices. The next issue that arises during testing is where to place the cameras to ensure maximum coverage of the target sampling area while taking into consideration manipulator occlusion and stereo baseline factors. The final subsection will describe the housings and mountings used for securing the cameras during operation, including details regarding the transition to a deep-sea configuration.

5.1.1 Stereo Cameras

Two sets of cameras were used during AVATAR testing. Initial tests were performed using analog Sony XC-999 cameras. Due to poor image quality issues related to these cameras, higher resolution cameras were purchased for the next phase of testing

– Point Grey Scorpion cameras that run on a firewire bus. Figure 5-1 shows the two camera types.



Figure 5-1: Cameras used for AVATAR testing

Left: Sony XC-999 Right: Point Grey Scorpion

The first phase of testing used Sony XC-999 cameras because they were readily available. While the availability of the cameras made their use convenient, many issues caused problems. First, since the cameras are analog, they be routed through a frame grabber board to be digitized for use within AVATAR. This involves using proprietary drivers for the legacy frame grabber boards, thus being limited to the Windows operating system. Also, the resolution for these cameras is limited to 640x480 with 8-bit depth per channel. Finally, the age of the cameras has resulted in significant degradation of image quality in many of the available cameras.

Once the first phase of testing was completed with the Sony cameras, design criteria were developed to ensure similar issues would not plague the next test phase. The first criterion is that the cameras use a form of digital output, to avoid the need for extra frame grabber hardware, and similarly have open-source drivers readily available. Next, the cameras need to have 16-bit depth per channel to maximize the data available in each image along with a dramatic increase in resolution. The cameras must also be compact to fit within deep-sea-rated housings. Finally, perhaps

the most important constraint was on cost given the limited project budget.

Based on these criteria, the decision was made to purchase Point Gray Scorpion 14SO cameras [51]. These cameras receive power and transmit data over a standard firewire cable, thus are capable of utilizing standard software tools on different operating systems for image acquisition. These cameras have 1280x960 resolution with 16-bits per channel operating up to 19 frames per second (FPS). Each image is stored in a Bayer pattern, meaning the CCD is organized with alternating elements sensitive to different wavelengths of light (Section 2.2). The camera itself is a compact 50x40x50mm in size but requires an attached external lens to focus light onto the CCD.

Computar model H3Z4512 lenses are attached to the Scorpion cameras. The H3Z4512 are vari-focal cs-ir 4.5-12.5mm F1.2 TV lenses, and were recommended for operation at ranges from 0.5 m up to 10 m. They must be manually adjusted for focus and zoom. Figure 5-1 shows the Computar lens attached to the Scorpion camera.

5.1.2 Camera Placement

Camera placement covers two major issues – placement of the stereo pair of cameras with respect to the manipulator and placement of the cameras relative to each other. Placement with respect to the manipulator mainly concerns the ability to see the target sampling area while limiting occlusion by the manipulator. Placement of the two cameras relative to each other encompasses a multitude of issues related to specific capabilities of the vision system.

The camera/lens combination was measured to have approximately a 75-

degree field of view (FOV) in air, but when placed underwater in a housing, the FOV drops to approximately 55 degrees. This affects the range of possible baselines for the stereo system. If an overlap of 75% is desired at a distance of 1m, a good range for use with the Ranger dexterous manipulator, then the maximum baseline between the cameras is limited to 0.26m. A large overlap is necessary to maximize the possible sampling area, so keeping the cameras close is important.

After acquiring test images with the cameras placed as close as possible while inside the deep-water housings, it is evident that the high resolution of the Scorpion cameras provides sufficient pixel disparity to accurately locate features, even with the shorter baseline. This result is somewhat surprising because earlier tests with the Sony cameras indicated short camera baselines compromised localization accuracy. A single pixel offset with the Sony camera test setup resulted in a nearly 10cm shift in calculated target position during neutral buoyancy tests. With the minimum baseline of 10cm, constrained by the underwater housings, a single pixel offset with a target located at a distance of 1.2m is only 2cm. All testing was performed with this baseline, as extending it will only increase accuracy of the system, until there is insufficient image overlap.

5.1.3 Housings and Mountings

For underwater tests of the vision/manipulator system, waterproof housings are necessary for the cameras and required computers. Due to the configuration of Ranger during initial tests with the Sony cameras, additional housings were unnecessary, as a Sony XC-999 pair was pre-mounted within the pressurized Ranger

vehicle with data communication lines to surface computers. However, for the next phase of testing, housings needed to be procured for both cameras and computers. The camera housings for the Scorpion cameras consist of two parts – a deep-water pressure housing and the internal mount to secure the camera.

Deep-Sea Pressure Housings

Operating at great undersea depths requires high-strength pressure housings to operate cameras at one atmosphere (surface) pressure. For the scope of this testing, housings were purchased to withstand 3000m of pressure. This fulfilled requirements for an initial deep-sea test phase that was not completed prior to the writing of this thesis. The commercial-off-the-shelf (COTS) housings are model number SSC-5000 from DeepSea Power and Light [52], manufactured for one of their proprietary analog cameras, but modified in-house to work with the firewire cabling and connectors required for operations at depth. These housings consist of aluminum housings with sapphire lenses, as shown in Figure 5-2. Models are available with depth ratings up to 6000m, where titanium is used in lieu of aluminum.



Figure 5-2: DeepSea SSC-5000 Camera Housing

Internal Camera Mounts

Since the DeepSea camera housings are designed for use with a DeepSea camera, custom internal mounts were constructed to secure the Scorpion cameras to the housings. Delrin was used as the material for this mount to reduce electrical interference with the cameras. A design was first made in I-DEAS after measuring the external housings and cameras. Once a working CAD drawing was accepted, the internal mounts were manufactured on a Bridgeport mill and Hardinge lathe. Figure 5-3 shows the internal camera mount assembly components, and Appendix D contains the set of CAD drawings.



Figure 5-3: Internal Camera Mount Assembled and Disassembled

5.1.4 Transition to Deep-Sea Configurations

Plans were originally made for implementing the entire vision system on two deep-sea platforms for open-water testing. The first platform was designed as a large test-bed for both manipulator and vision systems. A frame around the test area was planned for testing positions of the cameras with respect to the manipulator to determine the most effective camera positions. The sampling/storage area resides on either side of the manipulator rest position. These two areas were the main usable

workspace in this configuration. Figure 5-4 shows a picture of the manipulator test frame attached to the WHOI sled.



Figure 5-4: SSL Manipulator Test Frame Attached to WHOI Sled

The second underwater configuration is on WHOI's JAGUAR AUV. Interfacing a manipulator with an AUV is a daunting task, and placing cameras to provide vision data in the manipulator workspace makes that task more difficult. Working on an AUV requires meeting stringent weight and moment constraints as well as ensuring that all cables or protrusions, such as camera housings, are located and secured to avoid becoming entangled with the environment. Also, manipulator occlusion of the camera field of view must be minimized and characterized prior to deployment, particularly for visual servo operations where the manipulator and target must both be visible throughout the motion sequence.

Due to the large set of constraints on manipulator and camera position, there is a small set of possible configurations when interfaced on the AUV. As neither the AUV nor a mockup is currently available, simulated environments were created for determining appropriate positions for the cameras with respect to the manipulator. Initial data provided by CAD models and basic laboratory mockups provide a basis

for positioning the hardware when the AUV is available.

5.2 Ranger Manipulator System

The SSL's Ranger manipulator was utilized for hardware testing in this research. Ranger is a 10DOF manipulator with eight revolute joints and two torque-driven tool drives. Kinematically, Ranger is segmented at the wrist into two four degree of freedom sections for mathematical simplification.

5.2.1 Manipulator Configuration

Since Ranger has eight degrees of freedom rather than the traditional six, it has a relatively complex mechanical and kinematic design, but it also possesses more capabilities due to the redundant degrees of freedom. Much research has been performed to analyze and characterize the additional manipulator capabilities [53][54]. While the dexterous workspace of the manipulator increases substantially given the extra degrees of freedom, singularities are more frequent but are also more easily avoided. One of the key benefits is that 8DOFs provide an infinite number of configurations for a given tool pose. The extra DOFs also allow the manipulator to move while the tool position remains constant and enable smooth planar motions. These additional degrees of freedom allow motions necessary for robust obstacle avoidance, although obstacle avoidance was not emphasized in this work.

The redundancy of the manipulator is controlled in two segments – a four DOF upper arm segment and the four DOF wrist. The upper arm segment control is in the form of the roll angle of the shoulder-elbow-wrist (SEW) plane. Through the use of the SEW angle, the upper arm joint angles can be computed independent of the

wrist joint angles, while the arm still possesses an additional DOF for avoiding the wrist singularity [53]. A more in-depth look at both singularity considerations and manipulator kinematics is provided below.

Workspace

Fully extended, the Ranger manipulator has a reach of approximately 1.3 meters. However, singularities exist in fully extended joint configurations. Additionally, large, sometimes prohibitive, torques are required to hold the arm straight in 1-G given its native neutral buoyancy design environment, further limiting the dexterous manipulator workspace, defined as the volume of space the robot end effector can reach in all orientations [27].

Due to the redundant degrees of freedom, and the resulting capabilities to avoid singularities, the dexterous workspace is almost as large as the reachable workspace. By correctly orientating the SEW angle and prudent use of the skew angle wrist, there are configurations that avoid nearly all singularities throughout the reachable workspace. To-date, a full analysis of the dexterous workspace has not been performed, so more quantitative workspace characterization is not possible.

Kinematics

The serial manipulator literature reveals that, except in research environments, revolute manipulators with greater than six degrees of freedom have usually been avoided due to increased complexity of hardware and the dramatic increase in required kinematic analysis. Despite these facts, Ranger required extra DOFs to avoid obstacles and singularities.

As with all serial manipulators, the forward kinematics are straightforward to derive. Previous research on SEW kinematics with a seven DOF manipulator was used to develop the forward kinematics of Ranger [55]. By building on this research, and through the use of modified Denavit-Hartenberg (DH) notation [27], the forward kinematics are fully developed in [53]. Figure 5-5 illustrates the D-H parameters and link frame assignments for the 8DOF Ranger manipulator. This figure was provided by Dr. Craig Carignan, which shows corrected values from the similar figure in [53].

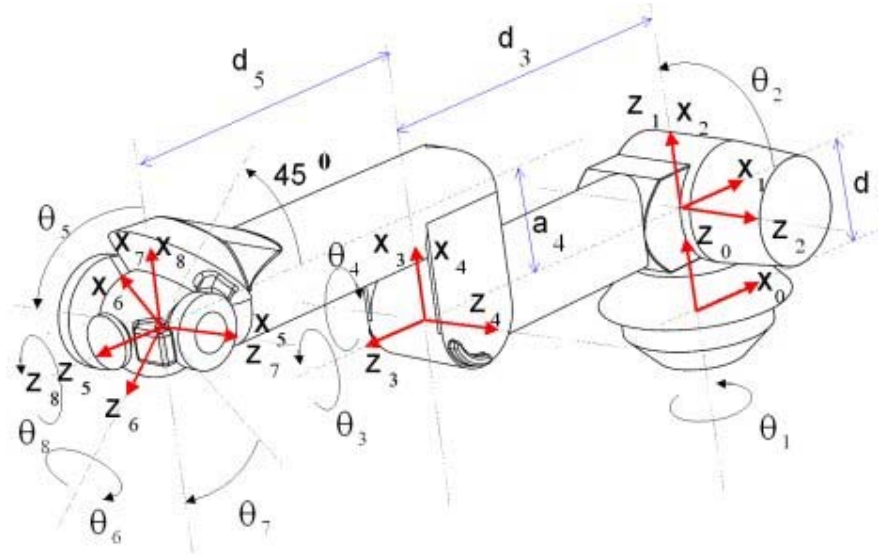


Figure 5-5: D-H Parameters and Frame Assignments of the Ranger Manipulator

Inverse kinematic solutions are more difficult to compute. To simplify the analysis, the manipulator is broken into two segments, joints 1-4 in the upper arm, and joints 5-8 in the wrist. A different method is used to solve the inverse kinematics of each section. Joints 1-4 use the Extended Jacobian Method based on the wrist location and SEW angle. Joints 5-8 use the General Inverse Method that finds a locally optimal solution for joint velocities specific wrist orientations and additional constraints

imposed by tool and forearm orientations [53][54]. The kinematic redundancies in the skew wrist design cause additional singularities that prohibit the use of the Extended Jacobian Method. Figure 5-6 shows a flow chart of how the 8-DOF inverse kinematics of Ranger are solved, from [53].

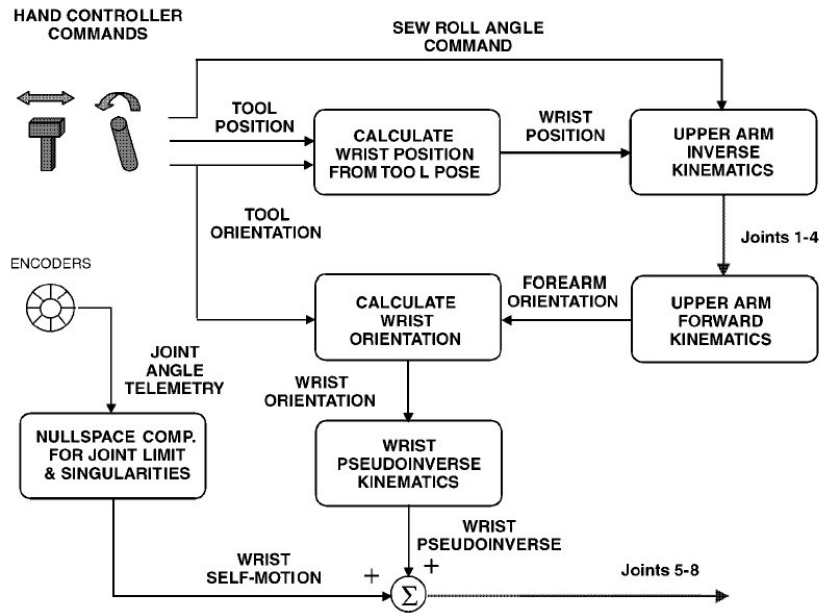


Figure 5-6: Inverse kinematics flowchart for the Ranger manipulator [53].

Singularity Considerations

Similar to the kinematics problem, the singularities are decomposed into two separate areas: upper arm and wrist. The traditional arm singularities that arise at workspace boundaries can easily be avoided by preventing arm motion to these boundaries. These singularities are known as external singularities. On the other hand, the internal singularities, or singularities that arise within the usable workspace of the manipulator due to loss of rank in Jacobian matrices during inverse kinematics, are an important problem since not all of these singularities are intuitive.

Based on the design of the arm, most of the upper arm internal singularities lie within regions outside of the normal workspace. One occurs when the wrist lies along the base frame z-axis, which happens when the arm is extended straight to the side, a configuration that would not be commanded during nominal operation. The only singularity that must be handled within the inverse kinematics solver is when the shoulder pitch angle is zero, but by holding the shoulder roll angle fixed this problem can be avoided.

Most wrist singularities result in loss of a single degree of freedom, which only has the result of removing redundancy. However, another type of singularity exists that causes loss of two degrees of freedom. An approach to such a singularity is internally recognized by abnormally large commanded joint velocities and can only be recognized in this manner [53]. The system is designed to automatically ignore joint velocities above a specified threshold so the manipulator will halt. In this case, a human teleoperator is needed to assist, or, under full autonomous control, an algorithm would need to be developed to back out and re-plan the motion to avoid the singularity.

5.2.2 Ranger Computer Architecture and Interface

Fortunately, the internal kinematic and control calculations are decoupled from the vision system. The software architecture from Section 4.1.1 enumerates the available methods to provide desired position data, which is then translated to Ranger motion. Aside from hand controllers and direct joint-by-joint control through a user interface, trajectory items can be created and used flexibly. For the vision system, this requires

a visual servo object to compute desired overall end effector motion then relies on other trajectory planning functions based on inverse kinematics to calculate the desired intermediate positions for smooth arm motions that avoid singularities [34].

5.2.3 Observed end effector positioning accuracy and resolution

Previous testing of Ranger was performed to characterize both static and dynamic performance characteristics of the manipulator [56][57]. These tests were performed in compliance with ANSI standards ANSI/RIA R15.05-1-1990 (R1999) for point-to-point static performance characteristics and ANSI/RIA R15.05-2-1990 (R1999) for path-related and dynamic performance characteristics. Results show Ranger is statically accurate to about 2 cm, while having a static repeatability of about 0.5 mm and a static compliance no worse than 0.4 mm/kg of applied force at maximum reach. For path following, Ranger has an average Cartesian accuracy of 1 mm, a Cartesian repeatability of 1 mm and a Cartesian path cornering radius of about 1 cm.

In terms of this research, the most important number is the static accuracy limited to 2 cm. Although gravity is one of the key causes of this error when operating in 1-G, there are also significant errors in directions not parallel to the gravity vector. This error adds uncertainty to the Ranger telemetry data, which is used as a truth-value during the automated registration procedure. This additional uncertainty will cause even more problems with a single snapshot, dead-reckoning approach to target sampling. Implementation of the visual servo controller can alleviate many of these issues by working directly with vision system data.

5.3 Test Sequence

Comprehensive testing of the fully integrated AVATAR system can be broken into several distinct steps. As mentioned in Section 3.1, the first step is to develop a set of calibration parameters for the stereo camera setup. Once this has been accomplished, the camera-manipulator registration must be determined to provide the transformation parameters between the vision reference frame and manipulator base frame. The next step is to test the sampling capabilities of the entire system by placing a visually distinct object in the camera FOV and manipulator workspace then performing a sampling task. Hardware and software upgrades evolve the system toward higher accuracy and more repeatable test results, enabling extension to more realistic targets. The final step of testing is using the vision system to not only track the sampling target, but also to track a distinct feature on the manipulator itself, enabling the visual servo algorithm to specify manipulator motion.

5.3.1 Camera and Manipulator Calibrations

Although some vision systems operate with an uncalibrated system, performing a full calibration can drastically reduce computational complexity while ensuring accurate target localization up to calibration accuracy limits. Throughout testing, the procedure for determining intrinsic and extrinsic parameters of the vision system remained fairly constant, although slight modifications were made when migrating from MATLAB to a C++ algorithm. The manipulator registration procedure evolved over time, but was always mathematically based on an algorithm designed to iteratively align two point clouds via estimation of a Z-Y-X Euler angle rotation

matrix (recall the algorithm from Section 3.1.2).

Vision System Calibration

Camera calibration throughout testing process has remained primarily a manual process. Every time the relative position of cameras changed, a new extrinsic calibration must be performed, but to be conservative an entire calibration was always performed. The use of a checkerboard in numerous poses throughout the field of view of both cameras requires human interaction both moving the checkerboard and validating images. However, once the images have been acquired and validated, the remainder of the calibration procedure is straightforward and can be handled automatically. For some of the camera setups, other software developed at the SSL for autonomously calibrating cameras [25] was used to determine the intrinsic parameters of each camera. This software utilizes OpenCV functions that perform the same tasks as described in [3]. However, the software does not currently perform the extrinsic calibration procedure that to-date must be performed manually.

Camera-Manipulator Registration

Multiple procedures were developed to determine an accurate camera-manipulator registration between the manipulator and camera system. Small errors, especially with the rotation matrix, can cause large errors when applied to points at the extremities of the manipulator workspace. Initially, the transformation was measured with inaccurate measurement devices (rulers) and rotation assumed at approximate angles. As testing commenced, this was immediately deemed insufficient.

An initial procedure manually tracked points in both the vision frame and

manipulator frame and developed a transformation between the two using an early version of the registration algorithm from Section 3.1.2. Concurrently, registration was attained using a Faro Arm, a portable coordinate-measuring machine (CMM). Since the CMM registration process was highly accurate, the software-based registration system was temporarily abandoned.

Given that the cameras might be perturbed relative to the manipulator frame during deployment where no CMM is available, the method of determining points in both vision and manipulator frames was subsequently revisited. With the higher resolution Scorpion cameras and the improved algorithm described in Chapter 3, a hand-eye registration was obtained autonomously for use in the final phases of testing through use of Ranger's trajectory generation system.

5.3.2 System-Level Testing with a Visually Distinct Object

Preliminary testing of an early version of the AVATAR software, coupled with the Sony XC-999 cameras, was performed using a visually distinct sampling target, such as the rubber duckys seen throughout this thesis. A series of tests were conducted in both 1-G laboratory and neutral buoyancy underwater environments. Initial tests performed in 1-G were designed to characterize the accuracy and precision of the vision system with respect to the Ranger manipulator in its original configuration. Sampling tests were performed with great success.

Transitioning to underwater testing introduces a necessity for multiple SCUBA divers and deck crew, in addition to the manipulator operators, which greatly increases the inherent overhead. Due to these factors, the scope of testing in the

neutral buoyancy tank was highly limited. Despite these issues, similar precision tests were performed, in addition to the overall sampling task operational tests. In the underwater tests, Ranger was operated in its extended wide-body configuration due to concurrent testing that required the extended configuration. Figure 5-7 shows Ranger's configuration in both environments.

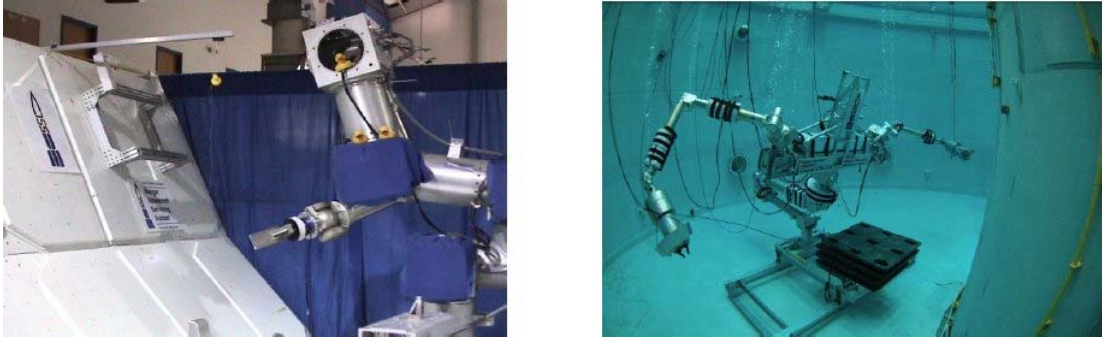


Figure 5-7: Ranger configurations in 1-G and neutral buoyancy

5.3.3 Evolution to Repeatable, Accurate Target Identification and Tracking

To increase accuracy and reliability of the vision system data, the transition was made to the higher resolution Scorpion cameras. Also, by the start of these tests, significant upgrades had been made to the AVATAR software. The major goals associated with this phase of testing were to quantify the accuracy and precision of the vision system, in addition to the ability to work with less distinct, more realistic sampling targets. Figure 5-8 shows sample target fields taken with WHOI's SeaBED AUV, processed along with other target fields engineered for capture by the AVATAR system.



Figure 5-8: Images of Deep Water Sampling Target Fields

Due to the lack of availability of hardware for underwater operations, testing with the Scorpion cameras was performed only in a 1-G laboratory with the original Ranger configuration. In addition to increased accuracy of the AVATAR data, software upgrades enabled operation with the fully functional AVATAR software described in Chapter 4. As described in Section 5.3.1, an automated hand-eye registration process was also implemented, further narrowing the gap between current capability and fully-autonomous operation. The automated registration procedure computes the transformation between the manipulator base frame and the camera frame of reference. Accuracy of this process determines manipulator accuracy when sampling a target via the dead reckoning process. To facilitate transition to the visual servo experiment, where a target on the arm must be tracked, the testing of the automated registration also uses a target on the Ranger arm – the IEEM. The IEEM is an easily recognizable gold object that lies after all degrees of freedom on the wrist.

Aside from the visual servo tests discussed in Section 5.3.4, tests focused on identification and tracking of difficult targets rather than sampling. A series of future tests will be needed to verify the integrated system once SAMURAI is operational.

5.3.4 Visual Servo Testing

The culmination of AVATAR testing is implementation of the visual servo procedure to demonstrate the capability to sample a stationary or slowly moving target. Since the design of a sampling end effector is outside of the scope of this research, these visual servo tests are designed to validate the sampling procedure and software. To this effect, sampling targets are not be “sampled” during these tests, instead the location of the target as determined by the visual servo process will be demonstrated by placement of a pointer end effector. Since successful sampling of a target requires “sufficient” accuracy given a compliant end effector (with some tolerance for error), placement of the pointer is potentially a more difficult task. Figure 5-9 shows the test setup for visual servo demonstrations.

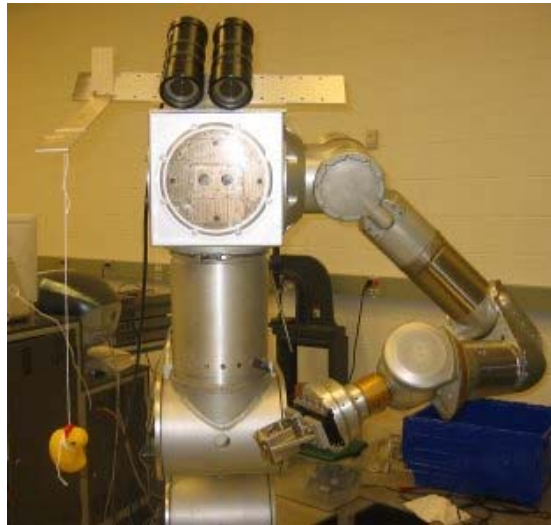


Figure 5-9: Ranger and AVATAR configuration for visual servo testing

Chapter 6 Test Results

6.1 Ranger Tests with Low Resolution Cameras

The first series of tests were performed with the system consisting of Sony XC-999 cameras in both 1-G and underwater environments and with two configurations of the Ranger manipulator. Testing in both environments was performed to determine whether the system was capable of retrieving a target with unique color properties from the background. In both environments data was recorded on the precision of the vision system. During 1-G testing, accuracy data, with respect to Ranger's internal telemetry, was also recorded.

6.1.1 Calibration Parameters

Table 6-1: Camera calibration data for Sony camera testing

Intrinsic	f_x (mm)	+/-	f_y (mm)	+/-	c_x (px)	+/-	c_y (px)	+/-
1-G Left	358.46	0.53	358.52	0.52	311.06	0.88	240.13	0.85
1-G Right	358.00	0.52	357.55	0.52	311.78	0.87	230.42	0.84
Underwater Left	477.42	0.36	478.57	0.35	315.16	0.75	244.41	0.72
Underwater Right	478.57	0.36	476.91	0.35	305.49	0.75	236.31	0.72
Extrinsic (underwater)								
Rotation (rad)	ω_x	ω_y	ω_z					
	-0.010	0.026	0.026					
Translation (mm)	t_x	t_y	t_z					
	-49.29	-0.41	2.38					

Table 6-1 shows calibration data for this phase of testing, as calculated using [3]. The intrinsic parameters are shown for both test setups, the 1-G testing with head-mounted cameras and underwater testing using Ranger's boresight cameras.

Extrinsic calibration is only provided for the underwater tests, as the other data was not properly saved. The uncertainty values for the extrinsic parameters are also unknown.

6.1.2 Vision System Accuracy

During 1-G testing, an effort was made to produce an initial accuracy characterization of the target acquisition system. The concept was to use the manipulator as a measurement tool since its positioning accuracy was expected to be at least an order of magnitude greater than that of the vision system. A target was fixed to the wrist of the manipulator and moved to 11 different static locations within the workspace of the manipulator and within the FOV of the vision system. Position data for the target from the target acquisition system was collected as well as manipulator pose data at each of the 11 locations. Target position data was then derived from the manipulator data, based on registration between target and manipulator, and compared with the data from the target acquisition system.

Table 6-2 summarizes the results. Of the eleven trials, errors ranged from 3.8-8.0cm with an average error of 5.3cm. Note that the negative X-axis for the coordinate frame used for this data corresponds to distance (range) from the camera.

Table 6-2: Target Acquisition System Accuracy Data

Test Number	Target Acquisition System			Vehicle Telemetry			Difference(cm)
	X(m)	Y(m)	Z(m)	X(m)	Y(m)	Z(m)	
1	-0.605	0.108	-0.190	-0.614	0.123	-0.143	5.1
2	-0.501	-0.006	0.007	-0.487	0.000	0.049	4.5
3	-0.481	0.014	-0.382	-0.497	0.019	-0.348	3.8
4	-0.647	-0.022	-0.286	-0.674	0.009	-0.230	7.0
5	-0.628	0.014	-0.072	-0.625	0.030	-0.019	5.6
6	-0.717	-0.001	0.054	-0.704	0.022	0.129	8.0
7	-0.732	0.015	-0.113	-0.728	0.035	-0.040	7.5
8	-0.626	-0.003	-0.173	-0.637	0.004	-0.130	4.5
9	-0.518	-0.088	-0.171	-0.540	-0.078	-0.136	4.3
10	-0.558	-0.186	0.021	-0.555	-0.177	0.062	4.2
11	-0.450	-0.143	-0.234	-0.481	-0.147	-0.209	4.0

The minimum error is 3.8cm, maximum error is 8.0cm, and average error is 5.3cm with a standard deviation of 1.5cm. There are at least two significant sources of error. One source is the poor registration between the vision system and the manipulator coordinate frames for the 1-G testing. Even small rotational errors cause significant positioning error at extended distances from the cameras, which is evident in tests six and seven. Additionally, as discussed in Section 5.2.3, Ranger has a static positioning accuracy of 2cm [56] that impacts accuracy as well.

6.1.3 Vision System Precision

Without an external measurement system it is difficult to obtain the accuracy data presented previously, however, precision of the vision system is easily measured. In both testing environments a large set of images were acquired with a stationary target (500 in 1-G testing and 80 in neutral buoyancy). As shown in Table 6-3, results from 1-G testing showed a much more precise system due to better camera quality.

Table 6-3: Vision System Precision Data

Environment	# Points	σ_x (cm)	σ_y (cm)	σ_z (cm)	magnitude	
1-G	9	0.27	0.28	0.57	0.69	
Underwater	9	0.58	0.36	3.91	3.97	
Underwater	1	0.71	0.37	4.79	4.86	(Centroid)
Underwater	1	0.48	0.31	0.28	0.64	(Selective)

Figure 6-1 shows the neutral buoyancy data. These plots show that in cases where only the centroid was used, the averages jump consistently between three levels, marked as black lines. This is because the triangulation algorithm is, for rotationally aligned cameras, based mainly on horizontal shift in pixel location of matching points. As the amount of horizontal shift for the centroid between left and right images varies discretely, so does the calculated depth. However, on each of these levels the calculated positions are proximate. The final row in Table 6-3 shows that only selecting a single level for the centroid-only data results in similar standard deviation as the 1-G case. Conversely, when all nine points are used, the standard deviation drops significantly, yet the values are spread further apart than on the discrete levels associated with the centroid-only calculation.

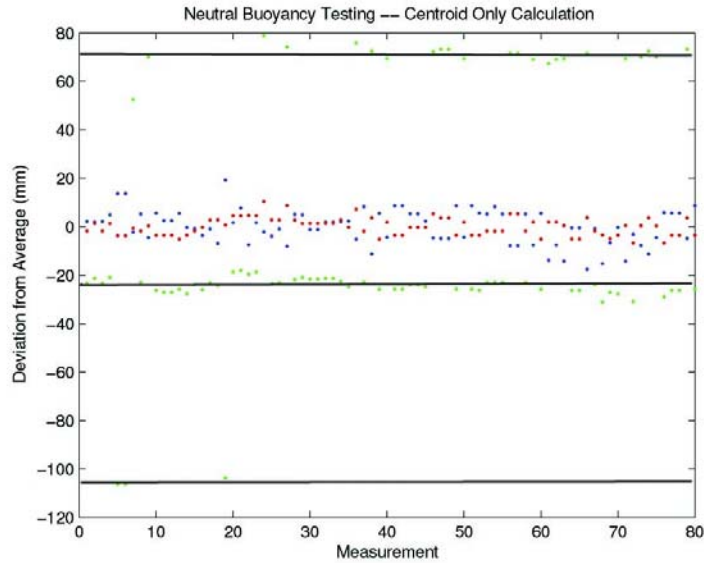


Figure 6-1: Difference in Z reconstruction due to single pixel error

During testing, the centroid-only method was utilized, but all values were monitored so that only results from the correct level were sent to the manipulator. Removing all data points from the incorrect levels, the standard deviation reduces to 2.80mm in the Z direction.

The calculated target position jumped between those discrete levels during neutral buoyancy testing because of the dramatic increase in manipulator size. Due to workspace limitations, the target was placed approximately three times as far from the vision system: 1450mm in neutral buoyancy and only 500mm in 1-G. This caused the target pixel area to reduce significantly, thus the fixed-size boundary blending between target and background occupied a much larger percentage of feature size. In neutral buoyancy, the target filled an area of only 12x12 pixels while in 1-G it filled an area of 25x25 pixels. The boundary blending adds 2-3 pixels around the border on all sides of the target – a much more substantial percentage of the more

distant neutral buoyancy target. This results in less predictable filter output, which causes the triangulated location to vary more significantly. When the target was placed at approximately the same distances in 1-G and underwater environments the results were similarly promising; this indicates the underwater environment itself did not impact vision system precision.

6.1.4 Overall System Behavior

Despite vision system inaccuracies, autonomous sampling sequences were quite effective in practice. Out of approximately 30 test runs in the 1-G environment, only two were unsuccessful in grasping the target. The first failure occurred when running Ranger in a different mode such that the tool offset from the end-effector was ignored. This caused Ranger to attempt a grab 37cm away from the duck target. The second failure resulted in improper target acquisition by the vision system. In an attempt to acquire live footage, the video feed from the left camera was split into a video recording system. Unfortunately this reduced the quality of the feed and made the left image much darker than that recorded by the right camera. The difference in brightness between the two images caused the filters to incorrectly select and match corresponding points, resulting in an incorrect object position estimate. Other tests in a variety of lighting conditions were successful so long as both left and right images had similar brightness.

Neutral buoyancy testing was successful considering the dramatic increase in manipulator size and increase in target distance from the vision system. After performing the accurate registration process with the portable CMM and constraining

the target localization to the appropriate level of calculation, successful target retrieval occurred for one out of seven tries. In each case where the target was missed, the manipulator was systematically too close in the depth dimension hitting the target but not grasping it with the end-effector. Small errors in the rotation aspect of the vision system to manipulator registration are suspected to have caused this error. Figure 6-2 shows successful tests in both environments, the 1-G view from the vision system itself, and the neutral buoyancy view from an external camera on the left Ranger arm (the right arm is grabbing the target).



Figure 6-2: Ranger successfully grabbing a target in both testing environments

6.2 1-G Ranger Tests with High-Resolution Cameras

The first set of results will focus on the automated hand-eye registration process developed as a step toward fully autonomous operation. The registration procedure is performed multiple times with different sets of original points to demonstrate the precision of the algorithm. After the registration parameters have been chosen, they are used to transform a set of independently recorded points from vision frame into manipulator frame to evaluate accuracy between the two sets.

The second set of results describes the outcome of visual servo experiments.

This compares the difference between attempting a target grab based on dead-reckoning from an initial snapshot from the hand-eye transformation with results from the implemented visual servo controller.

6.2.1 Calibration Data

Table 6-4 shows the calibration data used for this phase of testing. The intrinsic parameters of focal length and principle point are shown in the upper rows, while the extrinsic parameters are shown below.

Table 6-4: Calibration data for testing with Scorpion cameras

Intrinsic	f_x (mm)	+/-	f_y (mm)	+/-	c_x (px)	+/-	c_y (px)	+/-
Left Camera	1049.13	0.67	1050.38	0.68	712.28	1.13	564.45	1.18
Right Camera	1047.20	0.67	1048.66	0.69	702.74	1.22	561.13	1.16
Extrinsic								
Rotation (rad)	ω_x	+/-	ω_y	+/-	ω_z	+/-		
	0.002	0.001	0.003	0.001	0.047	0.000		
Translation (mm)	t_x	+/-	t_y	+/-	t_z	+/-		
	-106.53	0.08	-2.58	0.07	0.82	0.29		

6.2.2 Automated Registration

As with the testing with the lower resolution Sony cameras, vision system precision and accuracy were evaluated to characterize the overall system. Since the Ranger IEEM is used as the target during the automated registration process, the same target is used throughout the rest of the vision system tests.

Precision evaluation is performed with two procedures. First, the precision of the vision system is calculated based on numerous sets of static images to ensure

consistent results. Second, the automated registration procedure is repeated three times with two different sets of points to illustrate the precision of the overall algorithm. The accuracy measurements are based on data relating perceived arm position with instantaneous arm telemetry. By taking a random set of points within the manipulator workspace and applying the hand-eye registration transformation on the vision data, the difference between this result and the arm telemetry will portray relative accuracy of the system.

Precision

During automated registration testing, data was recorded for seven random points within the Ranger workspace at distance magnitudes ranging from 0.65m to 0.95m from the cameras. A 3-D reconstruction from fresh images was performed at each point five distinct times then analyzed for repeatability. Table 6-5 shows the results from this experiment in the form of standard deviation for all three dimensions and the corresponding magnitude. The substantial improvement from the earlier system is evident, as the maximum standard deviation magnitude over all three dimensions is less than 4mm. The seventh point is not listed as all five analyses resulted in the same 3-D reconstruction. Another test was performed with two rubber duck targets placed approximately 2.5m from the cameras. Thirty tests were performed, and in every case the pixel centroids were measured at precisely the same value, thus there was zero change in reconstructed position.

Table 6-5: Vision System Precision with High Resolution Cameras

σ_x (cm)	σ_y (cm)	σ_z (cm)	magnitude
0.13	0.14	0.27	0.33
0.10	0.05	0.36	0.38
0.03	0.13	0.26	0.29
0.01	0.09	0.24	0.26
0.01	0.07	0.29	0.30
0.00	0.04	0.01	0.05

Based on this data we can be confident that AVATAR itself will provide extremely consistent results. The next task is to demonstrate that when combined with Ranger to determine hand-eye registration, results are once again consistent. Two different sets of points within the workspace were chosen for this experiment. For each set, three iterations of the registration procedure are performed. Test results summarized in Table 6-6 show excellent precision for this process. Each of the Euler angles has error less than 0.005 radians, while the overall magnitude of the standard deviation for the translation vector is slightly more than 5mm. This indicates that vision system precision is maintained through the registration process.

Table 6-6: Hand-Eye Registration Precision Results

Euler Angles		
σ_α (rad)	σ_β (rad)	σ_γ (rad)
0.0049	0.0042	0.0024

Translation Vector			
σ_x (cm)	σ_y (cm)	σ_z (cm)	magnitude
0.33	0.23	0.33	0.52

Accuracy

With automated hand-eye registration, the transformation can be applied to the random points used to determine vision system precision. When each of these points

was recorded, the arm telemetry at that point was also logged. By transforming the vision data into the arm frame of reference, the relative accuracy between transformed data and arm telemetry data can be calculated. Table 6-7 shows a summary of standard deviation and maximum offset results from these transformations. The complete data sets from these tests is presented in Appendix E.

Table 6-7: Difference of Arm Telemetry and Transformed Vision Coordinates

Standard Deviation of Offsets			
σ_x (cm)	σ_y (cm)	σ_z (cm)	magnitude
0.28	0.92	0.34	1.02
Max Offsets			
x(cm)	y(cm)	z(cm)	magnitude
1.02	2.87	1.37	3.34

These results show that the updated system performs substantially better than the earlier system, but there is still potential for improvement. Multiple sources of error factor into these calculations, thus the relative accuracy achieved here is encouraging. However, this error is still appreciable, motivating use of the visual servo algorithm.

6.2.3 Visual Servo Experiments

In addition to automating some of the more difficult processes, the ability to deal with unforeseen inaccuracies in camera calibration and hand-eye registration parameters is an important feature of a fully autonomous system. To mitigate such errors through feedback, a visual servo system was implemented to track both the manipulator and target. Aside from showing successful completion of a sampling task, results from the visual servo test illustrate the difference between what visual servoing and dead reckoning with a single data snapshot, in the “open loop” procedure used for Section

6.1 tests.

Initial tests to validate the software implementation were performed in simulation mode, where arm telemetry is purely mathematical and a user provides vision data. The data recorded from these tests show smooth trajectories with each step moving closer to the target. In these tests, the system exhibits the ability to handle small-magnitude changes in manipulator or sampling target position. Figure 6-3, Figure 6-4, and Figure 6-5 show plots of upper arm and wrist joint angles as well as Cartesian position and orientation for a simulated sampling task.

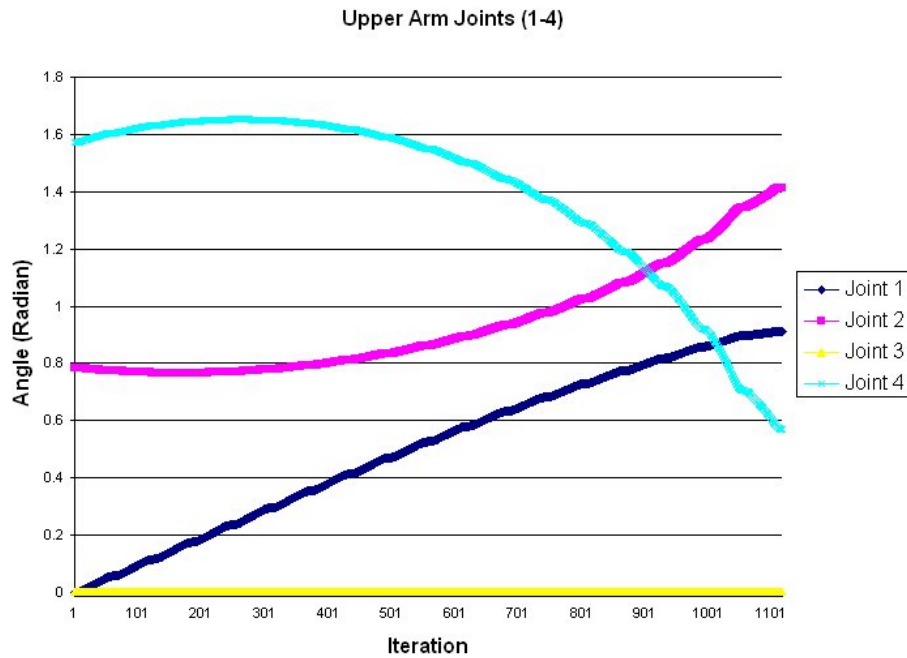


Figure 6-3: Upper Arm Joint Angles During Visual Servo Simulation

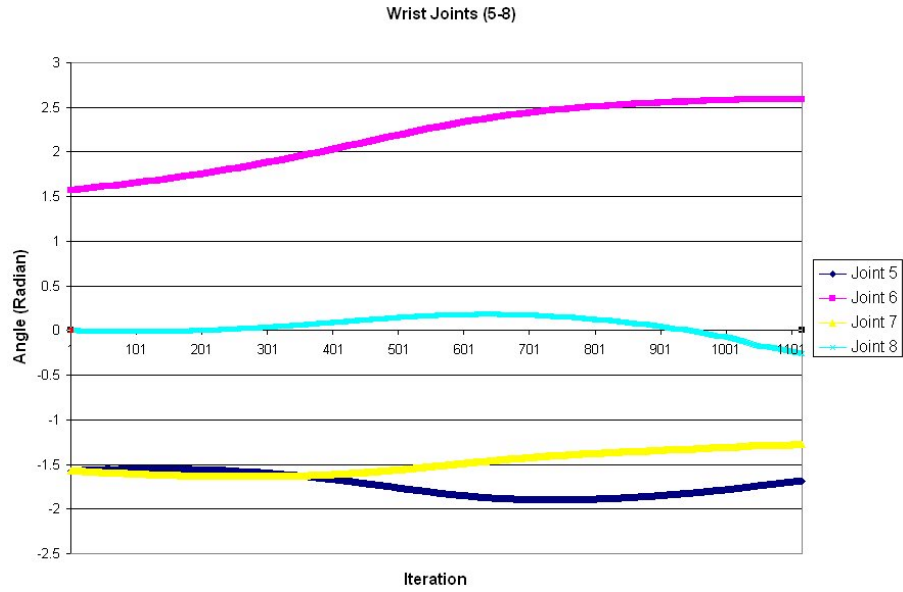


Figure 6-4: Wrist Joint Angles During Visual Servo Simulation

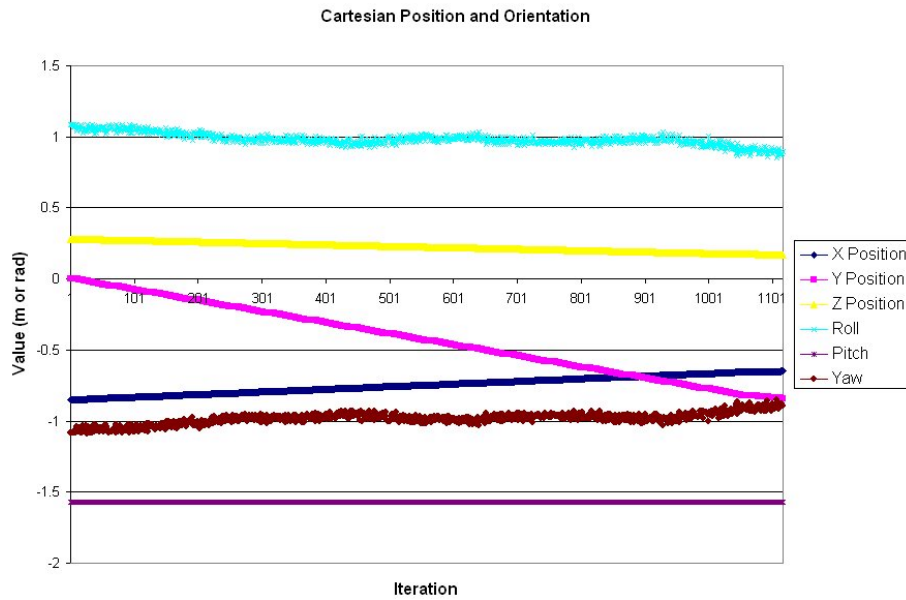


Figure 6-5: Cartesian position and orientation during visual servo simulation

Although the code appears to function properly in simulation, there have been many issues in getting it to work with the Ranger manipulator. Initial testing was successful – the manipulator would approach the position of the target and stop at the

desired position. However, on one test an error arose where the arm drew too much current when approaching a singularity and the power supply turned off. Further modifications to the software appeared to take care of this issue, but another test on the manipulator ran into the same power supply issue. Until this can be resolved, it will be impossible to perform more testing with Ranger.

With only a few initial test runs utilizing the visual servo system, the manipulator appeared to be within the same accuracy as the dead-reckoning tests with the earlier system. Until the final system is fully debugged and exhaustive testing can be performed, a decision on whether or not the visual servo routine provides clear benefits to the sampling capabilities cannot be made.

6.3 Increased Target Realism

The final set of tests was aimed to evaluate AVATAR performance when transitioned from the laboratory to a real-world environment where targets cannot be designed specifically for the task at hand. Based on both deep-water color attenuated imagery from WHOI, as well as cluttered target fields created specifically to stress the AVATAR algorithms, output from the initial stages of the vision system is shown with desired sampling targets extracted for further processing.

6.3.1 Laboratory Tests

A sample target field was created within view of the Scorpion camera pair. The targets in the simulated sampling field consist of different rocks, as well as a starfish and sand dollar. Feature extraction tests are performed with both lights on and lights off to simulate a dark environment. Using a combination of the MATLAB filter

creation process and the filtergui program, separate filters were created for both cases. In the lights on case, the focus was on extracting the rock targets, while in the lights off case, an attempt was made to extract whatever targets were distinguishable.

Output from the MATLAB filter creation program is displayed in Figure 6-6, Figure 6-7 and Figure 6-8. The target range of data versus unwanted data has a clearly distinguishable separation, which translates to clear image filtering parameters. As shown in Figure 6-9, the filtering process easily segments desired targets for further analysis. No further image processing is necessary for successful localization in these tests.

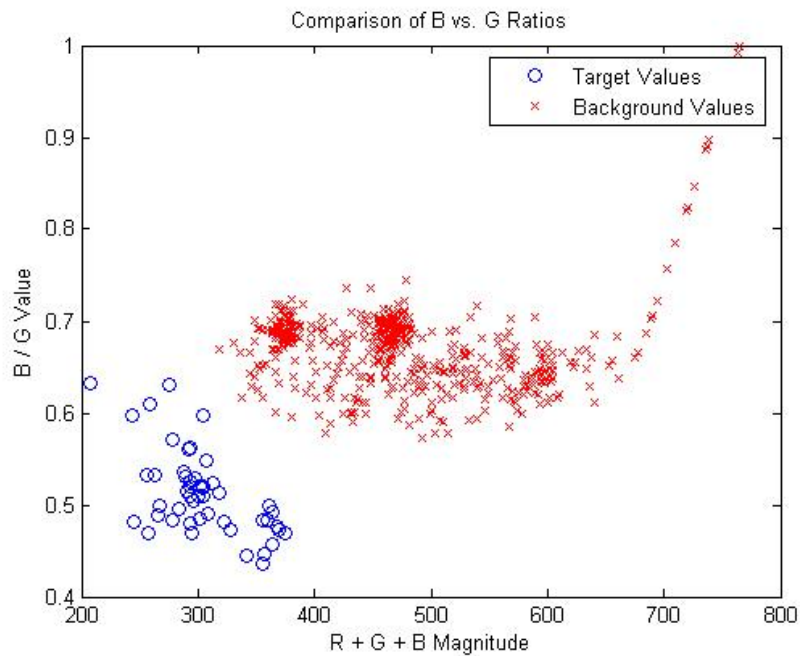


Figure 6-6: Blue vs. Green ratio data for light laboratory targets.

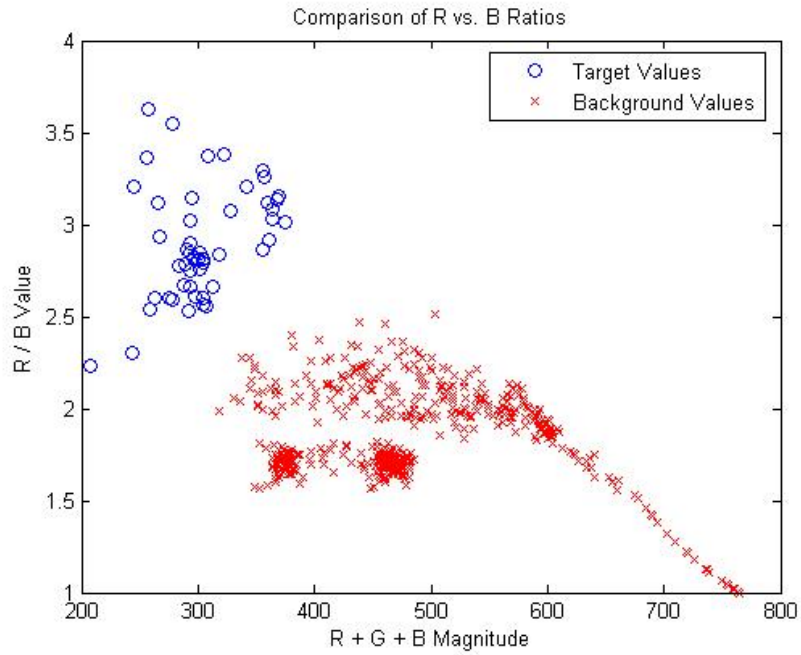


Figure 6-7: Red vs. Blue ratio data for light laboratory targets.

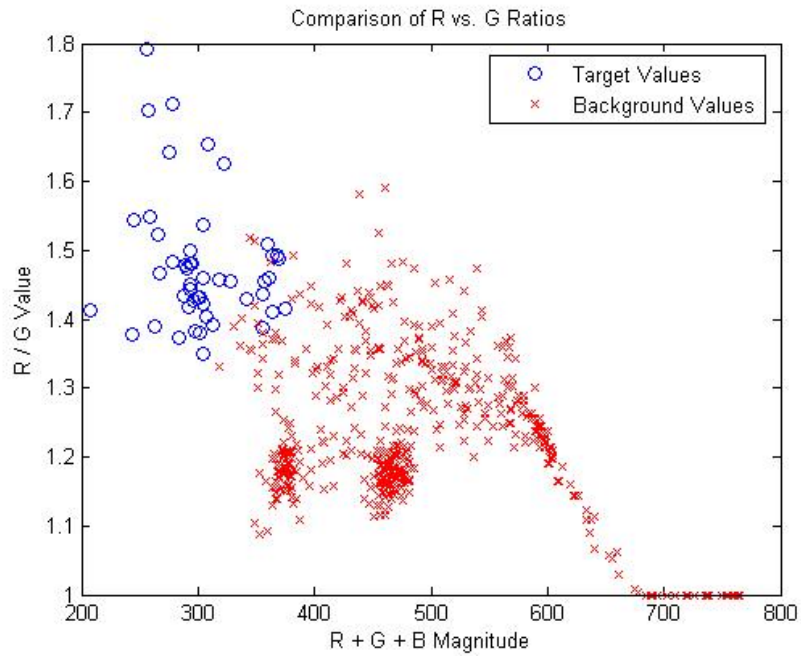


Figure 6-8: Red vs. Green ratio data for light laboratory targets.

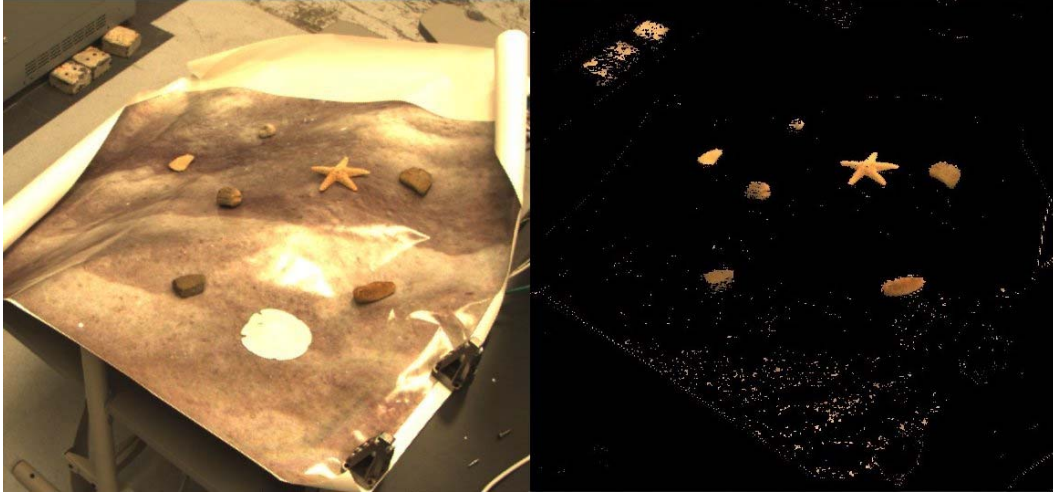


Figure 6-9: Realistic targets easily segmented in lighted laboratory environment.

Once the lights are turned off, the difficulty of the filtering process greatly increases. The MATLAB data from these tests, shown in Figure 6-10, Figure 6-11 and Figure 6-12, is still somewhat separated into two distinct clusters, although the targets areas now overlap significantly. The data is now separated more by a diagonal line rather than a distinct horizontal line, which requires a more complicated filter that accounts for varying ratio data versus RGB magnitude.

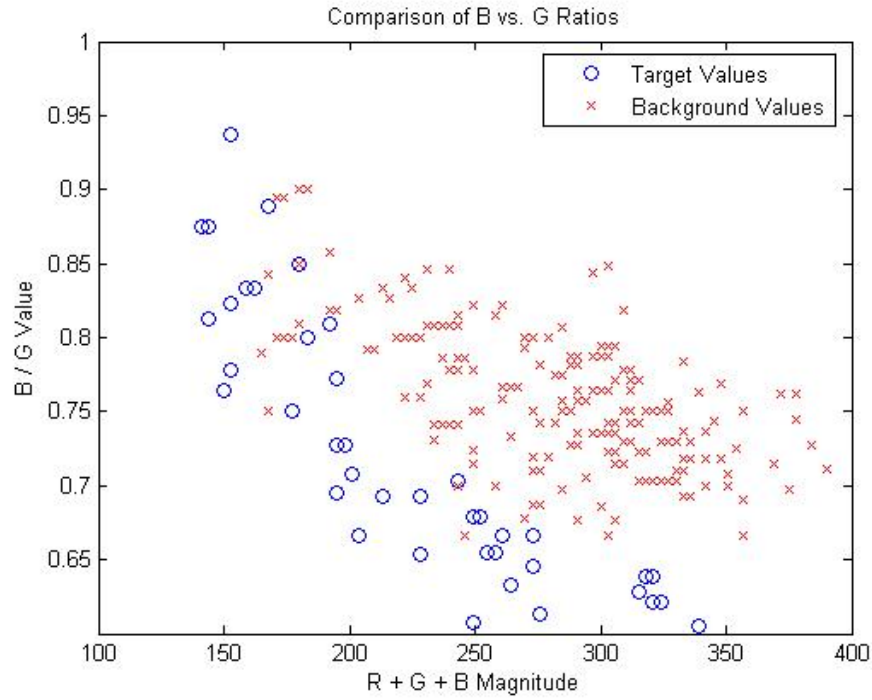


Figure 6-10: Blue vs. Green ratio data for dark laboratory targets.

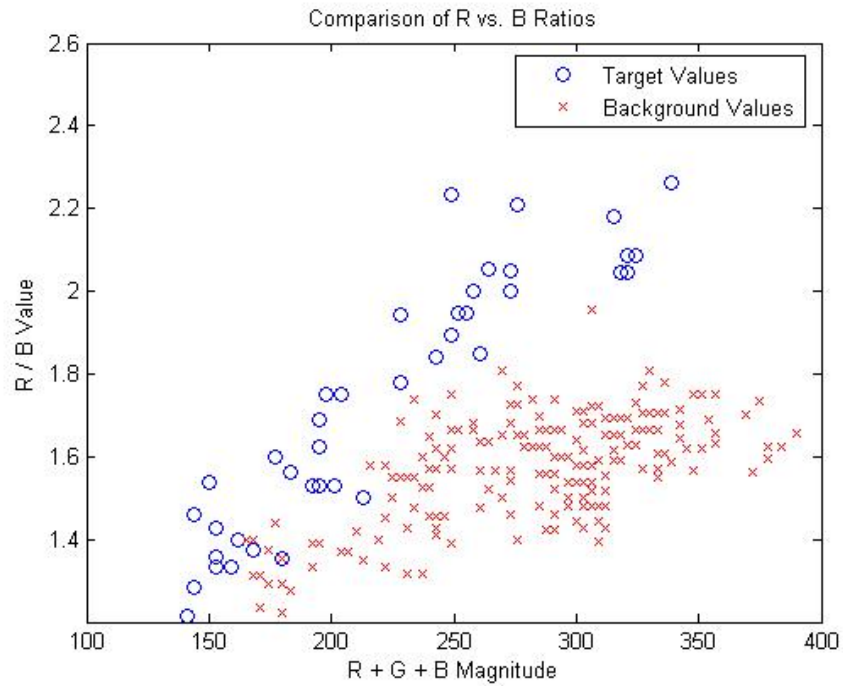


Figure 6-11: Red vs. Blue ratio data for dark laboratory targets.

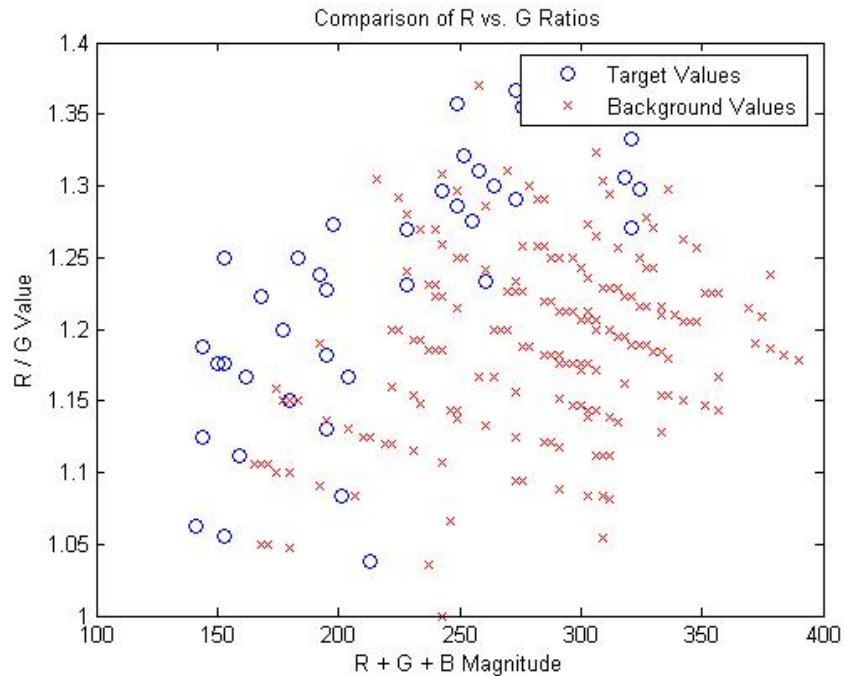


Figure 6-12: Red vs. Green ratio data for dark laboratory targets.

To achieve adequate results, a feature AND operator was applied to eroded versions of the original image to restore degraded features. Without the extra steps to erode and restore the image, it would have been impossible to extract usable feature data as much of the background remained in the processed image. Figure 6-13 shows the results from this test – the original image is dark, so the results were changed to binary values to clearly delineate the segmented features. In this test, the starfish, a single rock, and the sand dollar were all extracted with sufficient quality for further processing. Edges of the background image used to simulate sand also appear, but would be ignored through aspect ratio constraints. Also, note that the lead weights in the background also appear due to their similar color properties.



Figure 6-13: Some targets extracted in darkened laboratory environment.

6.3.2 Underwater Imagery

With full color correction applied to the raw WHOI imagery, any clearly distinct objects can be easily segmented. The focus of this test is to show that without full color correction, or even application of the simple frame averaging algorithm, targets can be cleanly extracted from the color attenuated images. At first, the feature extraction did not function suitably, as the difference in lighting from the center of the image to the edges drastically changed the RGB values at each pixel, due to the already attenuated data. By applying the lighting correction algorithm, discussed in Section 3.2.2, a more homogenous image is created that provides much more useful results from the filtering process.

Once again, the MATLAB filter creation algorithm was applied to the raw imagery to estimate good values for the filter process. Figure 6-14 and Figure 6-15 shows these results; due to the color attenuation, the Blue vs. Green chart is useless.

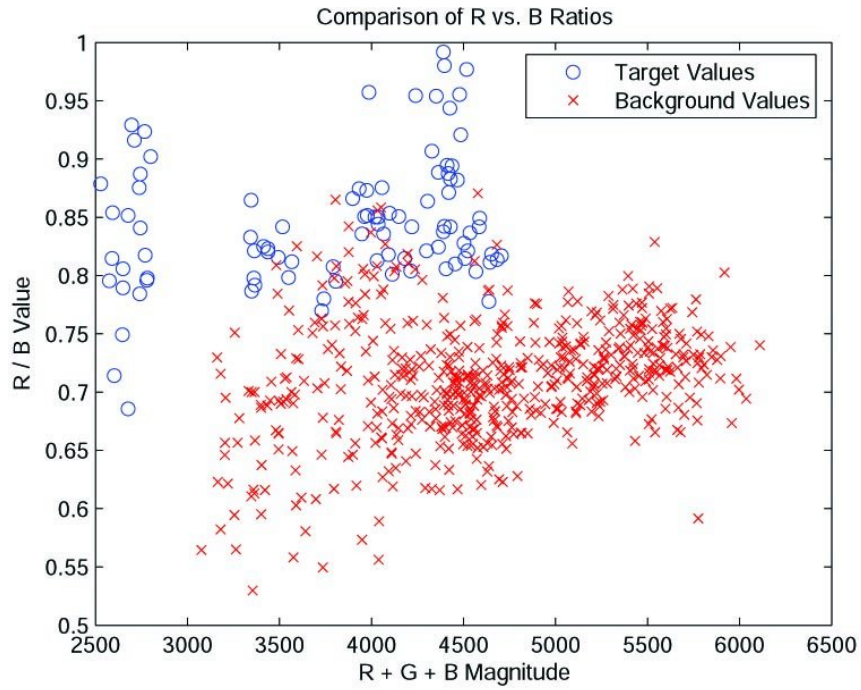


Figure 6-14: Red vs. Blue ratio data for sand dollar images.

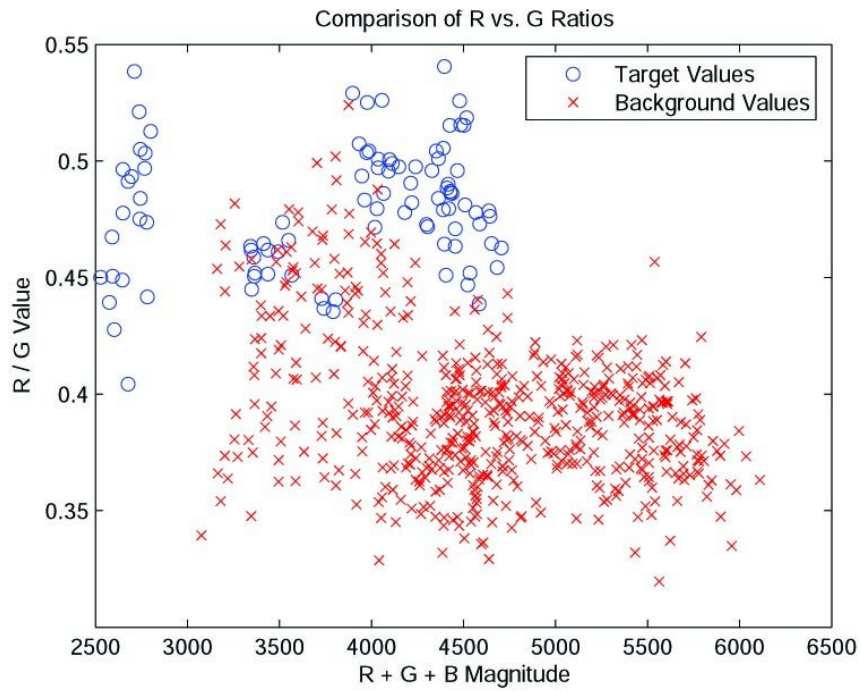


Figure 6-15: Red vs. Green ratio data for sand dollar images.

These plots are similar to the low-light results from the lab tests, but the lighting correction creates sufficient distinction to extract the majority of the sand dollars. Figure 6-16 shows the lighting corrected image and extracted features side by side.

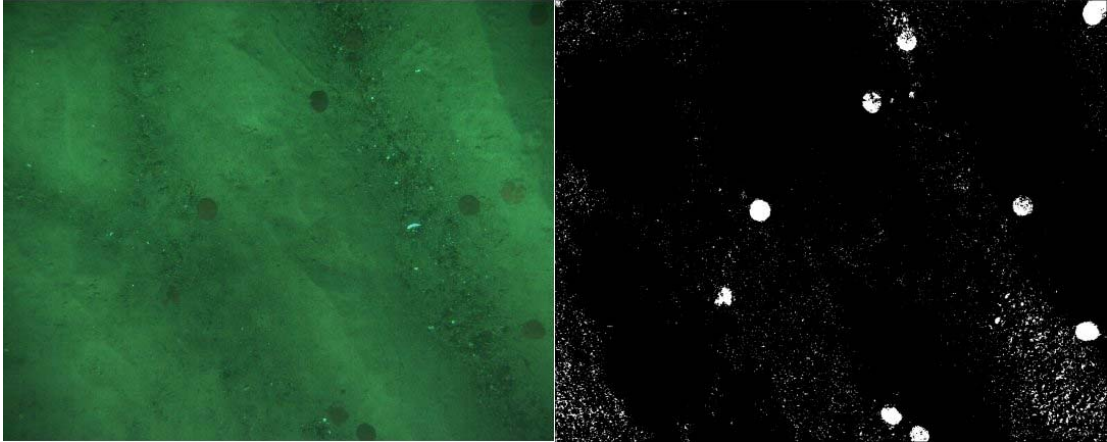


Figure 6-16: Sand dollar targets extracted from color attenuated image

Chapter 7 Conclusions and Future Work

7.1 Conclusions

This thesis describes the development and implementation of a fully autonomous vision system used to provide 3-D localization of sampling targets for a robotic manipulator. Three major focus areas are pursued: the development of the vision algorithms to perform feature segmentation and 3-D reconstruction, design of a logical, modular software structure, and finally hardware integration with a robotic manipulator and subsequent sampling tests. The overall system is capable of visually tracking both sampling targets and the manipulator, providing position data in the correct frame of reference to allow the manipulator to accurately approach a sampling target.

7.1.1 Vision Algorithms

The current set of vision algorithms provide the necessary capabilities to sample targets with sufficiently distinct color properties. Although the algorithms remain quite simple from a mathematical standpoint, this simplicity minimizes computational complexity and facilitates intuitive understanding, which is important in an environment where not everyone is fluent in the most recent computer vision algorithms.

The initial feature filter algorithm is performed by calculating the ratios of RGB pixel channels with one another and comparing with a desired range of values. This is essentially mapping a specific section of the RGB histogram to a feature of

interest. Any feature remaining in the image is extracted by a recursive procedure to locate and record all connected neighbor pixels, then geometric properties about the feature are evaluated to ensure compliance with the desired target. Once features have been extracted from a corresponding pair of images, they are matched with an algorithm that capitalizes on epipolar constraint and assumes a realistic and practical field of view. The 3-D target position can then be calculated through knowledge of the intrinsic and extrinsic parameters of the camera pair.

7.1.2 Software Structure

The software is split into two main modules: AVATAR, the computer vision system, and TAU, the public interface used to access AVATAR. This packaging allows a programmer to make local changes without propagation through other software systems. This modular structure is used down to the lowest level to ensure only required coupling occurs.

The use of external software management and validation tools allows the programmer to have greater confidence in the reliability of the software. Continuous integration and comprehensive unit testing quickly alerts the programmer to anomalies that may not immediately be recognized otherwise. Use of a memory profiling tool helps track down and reduce unforeseen memory leaks that can cause major problems hours into operation. Although 100% reliability is tough to guarantee, the use of additional tools can enable tested software to approach this goal.

7.1.3 Sampling Tasks

The success of sampling trials shows that the current system is capable of

autonomously sampling a desired target from within the manipulator's workspace. Testing with an earlier version of the system demonstrates that a less-capable version was able to successfully retrieve targets in both 1-G and underwater environments. With an increase in camera quality as well as software functionality and reliability, subsequent accuracy and precision data show much improved operational dependability of the overall system. Many of the tasks, aside from the initial calibration of the cameras and creation of the target filters, are completely autonomous. Finally, the implementation of the visual servo controller shows that even with little care taken in creating an extremely accurate camera calibration, and a fully autonomous procedure used for hand-eye calibration, the manipulator will successfully sample targets within its dexterous workspace.

7.2 Future Work

While the AVATAR system provides the capabilities necessary to autonomously sample a target, there are numerous avenues for future work. The modular nature of the software design allows quick and easy integration of new algorithms or other modifications, facilitating extension. Three main areas of possible future work will be discussed. First, implementing more complex, capable, or accurate computer vision algorithms could improve performance and flexibility. The second avenues for future upgrade are enhancements to the overall software architecture and ways to make the software run smoother and become more user-friendly. The final section will examine necessary changes for the transition from the 8DOF Ranger manipulator to the 6DOF SAMURAI manipulator.

7.2.1 Computer Vision

The major vision-related area that will benefit from future work is feature segmentation. There are many algorithms capable of performing quick and accurate feature extraction. By implementing multiple methods that complement the current color-based scheme, a user would be able to select the most capable algorithm for each target class.

Two approaches to this goal can be pursued. Complicated vision algorithms with the most accuracy, determining target type, in addition to pose, target motion and other quantities, are far from being used for real-time operations. On the other hand, through simplifying existing algorithm or researching faster algorithms potentially inspired by existing complex approaches, one might reduce execution time down to a reasonable level for real-time control.

Another useful enhancement would be researching methods of matching features extracted from one image, based on color data, with grayscale features in the other image. Most real-time computer vision applications use grayscale imagery since the one data channel can be analyzed more quickly than can multi-color data. Also, many underwater systems have one color camera and one grayscale camera, as color provides visually attractive pictures while the grayscale cameras have higher bit depth thus provide more data of use to scientists.

For moving targets and manipulator tracking, implementation of Kalman filters would provide a much more capable framework for ensuring desired outcomes. The current method assumes the most recent values are truth, providing no noise rejection capability.

Another possible avenue for research would be creating a 3-D map of the visible scene. This would enable obstacle detection and avoidance by the manipulator, an important capability for an autonomous manipulator sampling system operating in a complex 3-D environment (e.g., a hydrothermal vent mound with multiple chimney deposits).

7.2.2 Software Architecture

Although the software has been tested thoroughly, one issue that was never approached is profiling the software to determine bottlenecks to target for increased code and/or algorithmic efficiency, in turn yielding increased execution frequency. By apply a profiling tool, such as gprof [58], such areas could easily be discovered and possibly alleviated of extraneous code.

Due to the parallel nature of processing two images simultaneously, moving to a multi-threaded process structure also has can increase execution efficiency, particularly with multi-core or multi-CPU processing capability. Although useful for future applications, these upgrades were not pursued for this research. AVATAR execution frequency is limited by the JAGUAR strobe (1-2.5 sec flash interval) more than code overhead.

7.2.3 ASTEP Manipulator

One of the most important aspects of future work that will arise when SAMURAI is completed is the integration of AVATAR and TAU with the new manipulator. Ranger's kinematic calculator automatically enables the arm to move smoothly and reliably throughout the workspace. Without the SEW and additional wrist degree of

freedom, SAMURAI will not be able to achieve smooth planar motion, instead constrained to a single configuration for a given tool position and pose. The absence of redundancy will make singularity avoidance a greater issue, as well as avoidance of obstacles within the immediate workspace. Due to these factors, a reliable system must be developed for robustly planning obstacle-free and singularity-free paths from the current manipulator position to the desired manipulator sampling position.

Another crucial issue that will arise once integration with SAMURAI and JAGUAR occurs is the problem of camera occlusion by the manipulator. Space is extremely limited on an AUV so there are few camera positions where they could obtain a complete, overlapping view of the manipulator workspace. With any of these mounting options the manipulator will block parts of the camera view during its transit to the sampling target, potentially motivating a hybrid control scheme that actively switches between dead reckoning and visual servoing modes based on occlusion constraints.

Appendix A Relationship Between Essential and Fundamental

Matrices with Camera Calibration Parameters

The Essential matrix E is defined as:

$E = \mathbf{t}_x^\Delta R$ where \mathbf{t}_x is skew-symmetric matrix of 3-D translation vector and R is the rotation matrix from extrinsic calibration

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

The Camera matrix K contains the calibration parameters in the form:

For homogenous image points \mathbf{p}_r and \mathbf{p}_l :

$\mathbf{p}_r^T E \mathbf{p}_l = 0$ is the defining equation for the essential matrix

is the camera matrix of intrinsic parameters

$$F = K_r^T E K_l^{-1}$$

To find epipolar lines for a point in the left image or for a point in the right image:

$$\mathbf{l}_r = F \mathbf{p}_l$$

$$\mathbf{l}_l = F^T \mathbf{p}_r$$

The epipoles (the intersection of all epipolar lines in an image) satisfy:

$$F \mathbf{e}_l = 0$$

$$\mathbf{e}_r^T F = 0$$

Appendix B Derivation of the Registration Algorithm

The equation for a general transformation is given by:

$${}^1\mathbf{p} = {}^1\mathbf{R}^0 \mathbf{p} + {}^1\mathbf{t}_0$$

Assuming pure planar rotation about x-axis this equation may be described by:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} {}^1t_x \\ {}^1t_y \\ {}^1t_z \end{bmatrix}$$

$$x_1 = x_0$$

$$y_1 = y_0 \cos \gamma - z_0 \sin \gamma$$

$$z_1 = y_0 \sin \gamma + z_0 \cos \gamma$$

Solving for $\cos \gamma$ and $\sin \gamma$ from the y_1 equation yields:

$$\cos \gamma = \frac{y_1 + z_0 \sin \gamma}{y_0}$$

$$\sin \gamma = \frac{y_0 \cos \gamma - y_1}{z_0}$$

Substituting the expression for $\cos \gamma$ into the equation for z_1 :

$$z_1 = y_0 \sin \gamma + z_0 \left(\frac{y_1 + z_0 \sin \gamma}{y_0} \right)$$

$$y_0^2 \sin \gamma + z_0 y_1 + z_0^2 \sin \gamma - z_1 y_0 = 0$$

$$(y_0^2 + z_0^2) \sin \gamma = -y_1 z_0 + y_0 z_1$$

Similarly substituting for $\sin \gamma$ gives:

$$z_1 = y_0 \frac{y_0 \cos \gamma - y_1}{z_0} + z_0 \cos \gamma$$

$$y_0^2 \cos \gamma - y_0 y_1 + z_0^2 \cos \gamma - z_0 z_1 = 0$$

$$(y_0^2 + z_0^2) \cos \gamma = y_0 y_1 + z_0 z_1$$

The values of $(y_0^2 + z_0^2)\sin \gamma$ and $(y_0^2 + z_0^2)\cos \gamma$ are computed by the registration algorithm, yielding the transformation due to x-axis rotation.

Next, assuming pure planar rotation about the y-axis, the transformation equation may be described by:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} {}^1t_x \\ {}^1t_y \\ {}^1t_z \end{bmatrix}$$

$$x_1 = x_0 \cos \beta + z_0 \sin \beta$$

$$y_1 = y_0$$

$$z_1 = -x_0 \sin \beta + z_0 \cos \beta$$

Solving for $\cos \beta$ and $\sin \beta$ from the x_1 equation yields:

$$\cos \beta = \frac{x_1 - z_0 \sin \beta}{x_0}$$

$$\sin \beta = \frac{x_1 - x_0 \cos \beta}{z_0}$$

Substituting this expression for $\cos \beta$ into the equation for z_1 :

$$z_1 = -x_0 \sin \beta + z_0 \left(\frac{x_1 - z_0 \sin \beta}{x_0} \right)$$

$$x_0 z_1 = -x_0^2 \sin \beta + z_0 x_1 - z_0^2 \sin \beta$$

$$(x_0^2 + z_0^2) \sin \beta = z_0 x_1 - x_0 z_1$$

Similarly substituting for $\sin \beta$ gives:

$$z_1 = -x_0 \frac{x_1 - x_0 \cos \beta}{z_0} + z_0 \cos \beta$$

$$z_0 z_1 = -x_0 x_1 + x_0^2 \cos \beta + z_0^2 \cos \beta$$

$$(x_0^2 + z_0^2) \cos \beta = z_0 z_1 + x_0 x_1$$

The values $(x_0^2 + z_0^2) \sin \beta$ and $(x_0^2 + z_0^2) \cos \beta$ then enable the registration algorithm to describe the transformation due to a y-axis rotation.

Finally, assuming pure planar rotation about the z-axis, the transformation equation may be described by:

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} + \begin{bmatrix} {}^1t_x \\ {}^1t_y \\ {}^1t_z \end{bmatrix}$$

$$x_1 = x_0 \cos \alpha - y_0 \sin \alpha$$

$$y_1 = x_0 \sin \alpha + y_0 \cos \alpha$$

$$z_1 = z_0$$

Solving for $\cos \alpha$ and $\sin \alpha$ from the x_1 equation yields:

$$\cos \alpha = \frac{x_1 + y_0 \sin \alpha}{x_0}$$

$$\sin \alpha = \frac{x_0 \cos \alpha - x_1}{y_0}$$

Substituting this expression for $\cos \alpha$ into the equation for y_1 :

$$y_1 = x_0 \sin \alpha + y_0 \left(\frac{x_1 + y_0 \sin \alpha}{x_0} \right)$$

$$x_0 y_1 = x_0^2 \sin \alpha + y_0 x_1 + y_0^2 \sin \alpha$$

$$(x_0^2 + y_0^2) \sin \alpha = x_0 y_1 - y_0 x_1$$

Similarly substituting for $\sin \alpha$:

$$y_1 = x_0 \frac{x_0 \cos \alpha - x_1}{y_0} + y_0 \cos \alpha$$

$$y_0 y_1 = -x_0 x_1 + x_0^2 \cos \alpha + y_0^2 \cos \alpha$$

$$(x_0^2 + y_0^2) \cos \alpha = x_0 x_1 + y_0 y_1$$

The values $(x_0^2 + y_0^2) \sin \alpha$ and $(x_0^2 + y_0^2) \cos \alpha$ then enable the registration algorithm to compute the transformation resulting from a z-axis rotation.

Appendix C Full Algorithm for Extracting Feature Geometric Data

```
do  $i = 1$  to size( $\mathbf{f}$ )
     $totalX = totalX + \mathbf{f}[i].x$ 
     $totalY = totalY + \mathbf{f}[i].y$ 
    if  $i == 1$  then
         $X_r = \mathbf{f}[i].x$ 
         $X_l = \mathbf{f}[i].x$ 
         $Y_{bot} = \mathbf{f}[i].y$ 
         $Y_{top} = \mathbf{f}[i].y$ 
    else
        if  $\mathbf{f}[i].x > X_r$  then
             $X_r = \mathbf{f}[i].x$ 
             $Y_{bot,Xr} = \mathbf{f}[i].y$ 
             $Y_{top,Xr} = \mathbf{f}[i].y$ 
        else if  $\mathbf{f}[i].x == X_r$  then
            if  $\mathbf{f}[i].y > Y_{bot,Xr}$  then
                 $Y_{bot,Xr} = \mathbf{f}[i].y$ 
            end if
            if  $\mathbf{f}[i].y < Y_{top,Xr}$  then
                 $Y_{top,Xr} = \mathbf{f}[i].y$ 
            end if
        end if
        if  $\mathbf{f}[i].x < X_l$  then
             $X_l = \mathbf{f}[i].x$ 
             $Y_{bot,Xl} = \mathbf{f}[i].y$ 
             $Y_{top,Xl} = \mathbf{f}[i].y$ 
        else if  $\mathbf{f}[i].x == X_l$  then
            if  $\mathbf{f}[i].y > Y_{bot,Xl}$  then
```

```

         $Y_{bot,Xl} = \mathbf{f}[i].y$ 
    end if
    if  $\mathbf{f}[i].y < Y_{top,Xl}$  then
         $Y_{top,Xl} = \mathbf{f}[i].y$ 
    end if
end if
if  $\mathbf{f}[i].y > Y_{bot}$  then
     $Y_{bot} = \mathbf{f}[i].y$ 
     $X_{r,Yb} = \mathbf{f}[i].x$ 
     $X_{l,Yb} = \mathbf{f}[i].x$ 
else if  $\mathbf{f}[i].y == Y_{bot}$  then
    if  $\mathbf{f}[i].x > X_{r,Yb}$  then
         $X_{r,Yb} = \mathbf{f}[i].x$ 
    end if
    if  $\mathbf{f}[i].x < X_{l,Yb}$  then
         $X_{l,Yb} = \mathbf{f}[i].x$ 
    end if
end if
if  $\mathbf{f}[i].y < Y_{top}$  then
     $Y_{top} = \mathbf{f}[i].y$ 
     $X_{r,Yt} = \mathbf{f}[i].x$ 
     $X_{l,Yt} = \mathbf{f}[i].x$ 
else if  $\mathbf{f}[i].y == Y_{top}$  then
    if  $\mathbf{f}[i].x > X_{r,Yt}$  then
         $X_{r,Yt} = \mathbf{f}[i].x$ 
    end if
    if  $\mathbf{f}[i].x < X_{l,Yt}$  then
         $X_{l,Yt} = \mathbf{f}[i].x$ 
    end if
end if

```

```
end if
end if
end if
end do

$$C_x = \frac{totalX}{size(\mathbf{f})}, C_y = \frac{totalY}{size(\mathbf{f})}$$

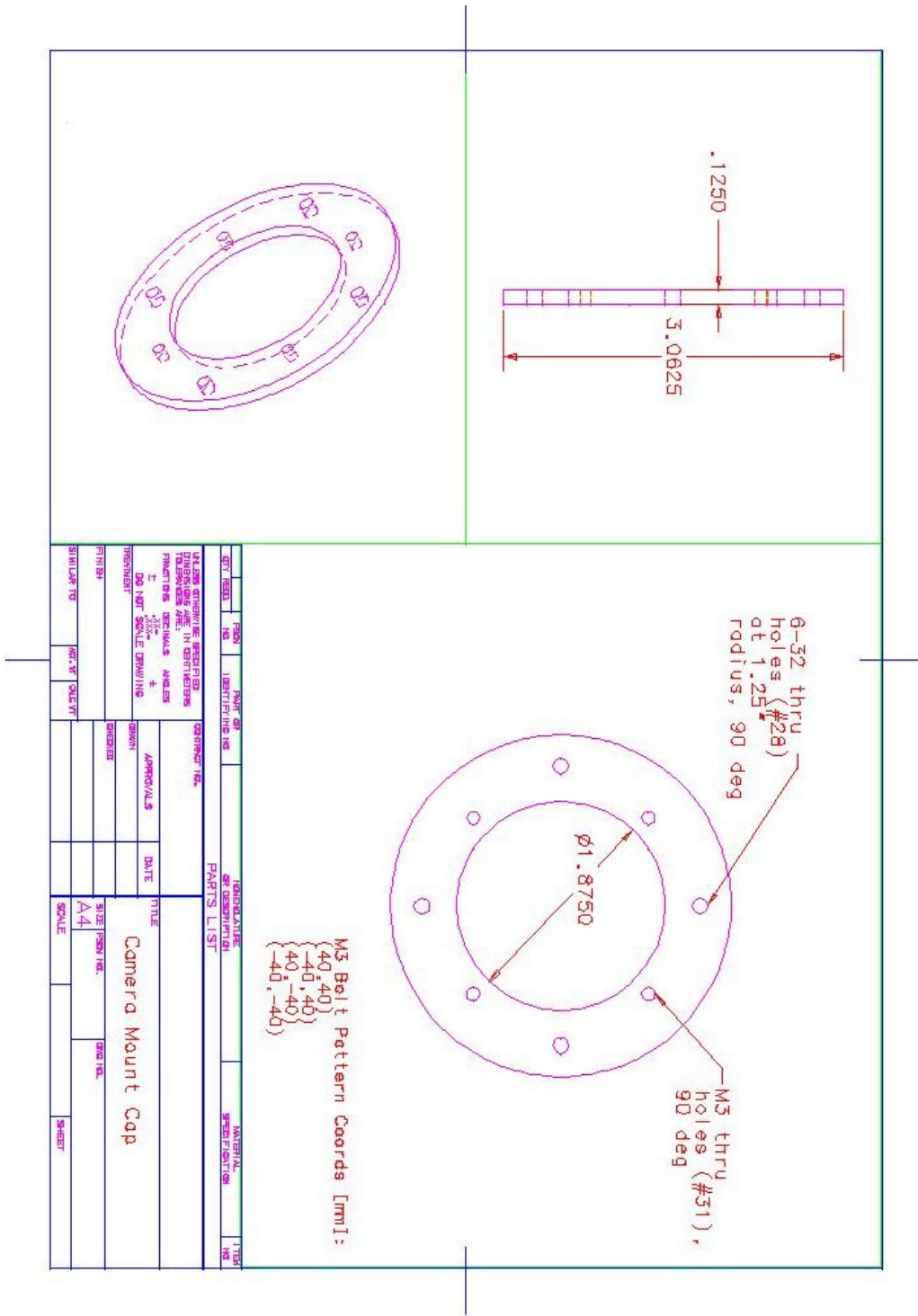

$$aspectRatio = \frac{X_r - X_l}{Y_{bot} - Y_{top}}$$

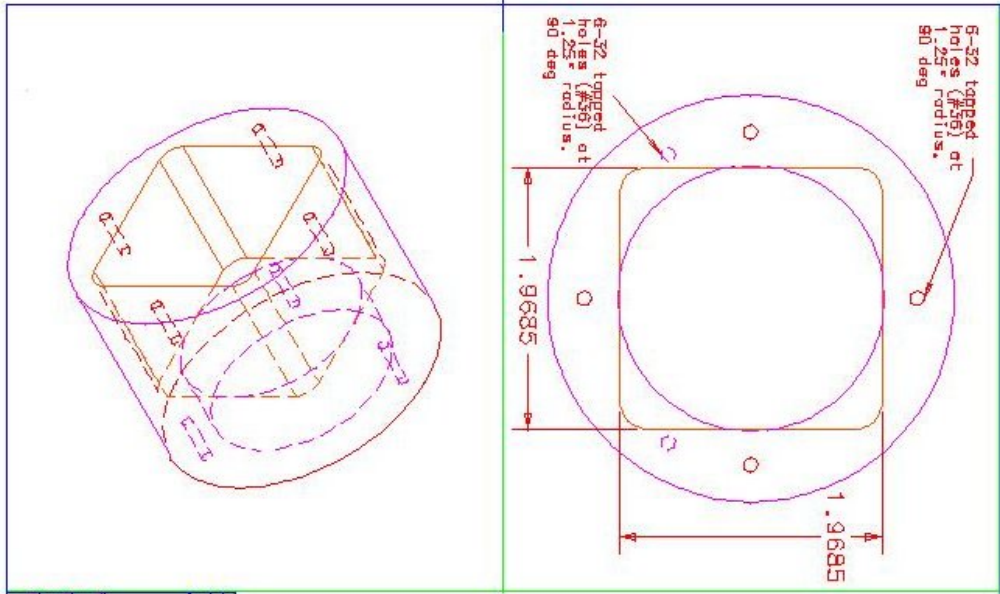

$$boxArea = (X_r - X_l)(Y_{bot} - Y_{top})$$


$$areaRatio = \frac{size(\mathbf{f})}{boxArea}$$

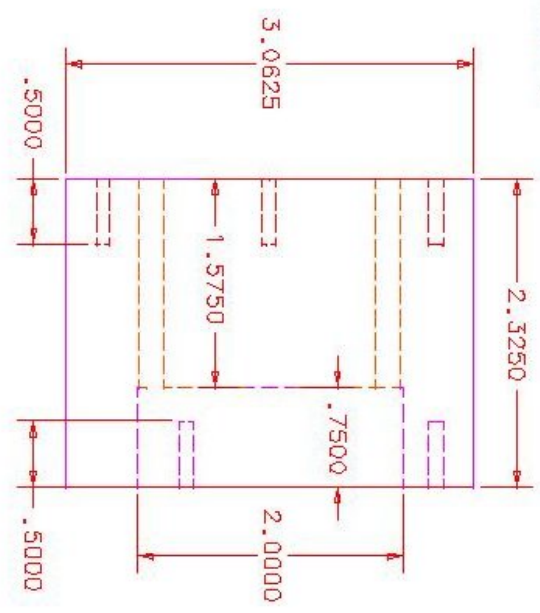
```

Appendix D CAD Drawings for the Internal Camera Mounts

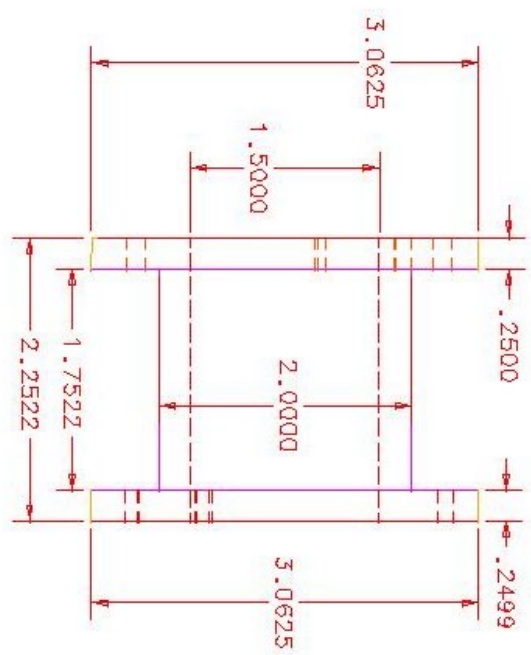
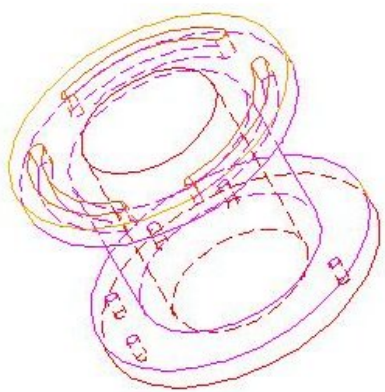
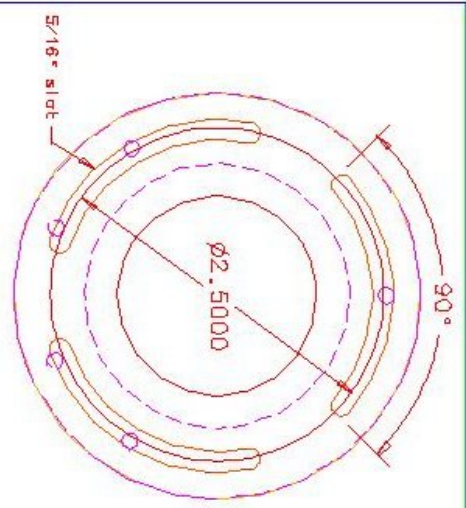




120deg Bolt Pattern Coords:
 (0,1.25)
 (1.083,-0.625)
 (-1.083,-0.625)



REV	ISS	NO	IDENTIFYING NO	REVISION	DATE	BY	CHK	APPROVALS	DATE	TITLE	SIZE	PROJ. NO.	DWG. NO.	SHEET
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN DECIMALS FRACTIONS, DECIMALS AND ANGLES ± DO NOT SCALE DRAWING											PARTS LIST INTERNAL SPECIFICATION		TITLE Internal Camera Mount	
DRAWN: _____ CHECKED: _____ APPROVED: _____ DATE: _____											SIZE: A4 SCALE: _____		SHEET: _____	



Bolt Pattern Coordinates (Back View):
 {0.12719}
 {1.1003,-0.6387}
 {0.5147,1.2125}
 {-1.0951,-0.6668}
 {-0.4830,-1.2202}

QTY	REQD	PERSON	NO	PART OR IDENTIFIER	NO	QUANTITY	DESCRIPTION	NATIONAL SPECIFICATION	UNIT
PARTS LIST									
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN DECIMALS FRACTIONS DECIMALS ANGLES ± DO NOT SCALE DRAWING					CONTRACT NO.				
DRAWN					APPROVALS				
CHECKED					DATE				
TITLE					Mount Interface				
FINISH					SIZE PERSON NO.				
SCALE					DWG NO.				
SHEET					SHEET				

Appendix E Results from IEEM Tracking

Transformed									Cartesian
Measured Vision			Measured Arm			Difference			Difference
x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)	x (cm)	y (cm)	z (cm)	(cm)
-51.75	-41.61	13.59	-51.63	-41.27	14.72	0.12	0.34	1.13	1.19
-51.82	-41.36	13.60	-51.63	-41.27	14.72	0.19	0.09	1.13	1.15
-51.82	-41.36	13.60	-51.63	-41.27	14.72	0.19	0.09	1.13	1.15
-51.60	-40.97	13.35	-51.63	-41.27	14.72	-0.03	-0.30	1.37	1.41
-51.60	-40.97	13.35	-51.63	-41.27	14.72	-0.03	-0.30	1.37	1.41
-52.30	-40.99	-5.22	-52.19	-41.64	-4.36	0.11	-0.65	0.86	1.09
-52.77	-41.65	-5.00	-52.19	-41.64	-4.36	0.58	0.01	0.64	0.87
-52.77	-41.65	-5.00	-52.19	-41.64	-4.36	0.58	0.01	0.64	0.87
-52.80	-41.64	-5.00	-52.19	-41.64	-4.36	0.61	0.00	0.64	0.89
-52.77	-41.65	-5.00	-52.19	-41.64	-4.36	0.58	0.01	0.64	0.87
-55.50	-52.54	-4.86	-55.22	-52.99	-4.47	0.27	-0.45	0.40	0.66
-55.30	-51.93	-4.93	-55.22	-52.99	-4.47	0.08	-1.06	0.46	1.16
-55.30	-51.93	-4.93	-55.22	-52.99	-4.47	0.08	-1.06	0.46	1.16
-55.30	-51.93	-4.93	-55.22	-52.99	-4.47	0.08	-1.06	0.46	1.16
-55.30	-51.93	-4.93	-55.22	-52.99	-4.47	0.08	-1.06	0.46	1.16
-55.11	-53.15	-16.43	-54.33	-55.49	-15.99	0.78	-2.35	0.44	2.51
-54.87	-52.62	-16.40	-54.33	-55.49	-15.99	0.54	-2.87	0.41	2.95
-54.87	-52.62	-16.40	-54.33	-55.49	-15.99	0.54	-2.87	0.41	2.95
-54.87	-52.62	-16.40	-54.33	-55.49	-15.99	0.54	-2.87	0.41	2.95
-54.87	-52.62	-16.40	-54.33	-55.49	-15.99	0.54	-2.87	0.41	2.95
-62.00	-57.97	-19.61	-60.98	-58.20	-19.53	1.02	-0.24	0.09	1.05
-61.65	-57.37	-19.64	-60.98	-58.20	-19.53	0.67	-0.83	0.11	1.07
-61.63	-57.43	-19.64	-60.98	-58.20	-19.53	0.65	-0.78	0.12	1.02
-61.65	-57.37	-19.64	-60.98	-58.20	-19.53	0.67	-0.83	0.11	1.07
-61.63	-57.43	-19.64	-60.98	-58.20	-19.53	0.65	-0.78	0.12	1.02
-66.63	-59.50	5.72	-66.35	-60.75	6.15	0.27	-1.26	0.43	1.35
-66.68	-59.41	5.71	-66.35	-60.75	6.15	0.33	-1.34	0.43	1.44
-66.68	-59.41	5.71	-66.35	-60.75	6.15	0.33	-1.34	0.43	1.44
-66.68	-59.41	5.71	-66.35	-60.75	6.15	0.33	-1.34	0.43	1.44
-66.68	-59.41	5.71	-66.35	-60.75	6.15	0.33	-1.34	0.43	1.44
-62.97	-45.85	5.12	-62.22	-47.50	5.96	0.75	-1.66	0.84	2.00
-62.97	-45.85	5.12	-62.22	-47.50	5.96	0.75	-1.66	0.84	2.00
-62.97	-45.85	5.12	-62.22	-47.50	5.96	0.75	-1.66	0.84	2.00
-62.97	-45.85	5.12	-62.22	-47.50	5.96	0.75	-1.66	0.84	2.00
-62.97	-45.85	5.12	-62.22	-47.50	5.96	0.75	-1.66	0.84	2.00

	Trial 1-1	Trial 1-2	Trial 1-3	Trial 2-1	Trial 2-2	Trial 2-3
α (rad)	-2.322	-2.336	-2.332	-2.331	-2.334	-2.333
β (rad)	1.577	1.575	1.573	1.566	1.568	1.570
γ (rad)	0.012	0.013	0.013	0.017	0.017	0.017
x (m)	-0.22861	-0.21913	-0.22217	-0.22265	-0.22061	-0.22068
y (m)	0.20837	0.20358	0.20511	0.203	0.20201	0.20307
z (m)	-0.1223	-0.12402	-0.12584	-0.13054	-0.12965	-0.12841

Bibliography

- [1] NOAA News Online (Story 1081),
<http://www.noaanews.noaa.gov/stories/s1081.htm> (version current 30 November 2006)
- [2] H. Singh, R. Armstrong, F. Gilbes, R. Eustice, C. Roman, O. Pizarro, J. Torres, *Imaging Coral I: Imaging Coral Habitats with the SeaBED AUV*, The Journal for Subsurface Sensing Technologies and Applications, pp. 25-42, vol 5, no 1, 2004.
- [3] *Camera Calibration Toolbox for MATLAB*,
http://www.vision.caltech.edu/bouguetj/calib_doc/ (version current 19 November, 2006)
- [4] Open Computer Vision Library, <http://sourceforge.net/projects/opencvlibrary/>
(version current 29 November 2006)
- [5] J. Heikkilä, O. Silvén. *A four-step camera calibration procedure with implicit image correction*. IEEE Computer Vision and Pattern Recognition Conference, San Juan, Puerto Rico, 1997.
- [6] Y. Shiu and S. Ahmad, *Calibration of Wrist-Mounted Robotic Sensors by Solving Homogeneous Transform Equations of the Form $AX = XB$* , IEEE Trans. On Robotics and Automation, Vol 5 No 1, Feb 1989.
- [7] Y. Motai and A. Kosaka, *SmartView: Hand-Eye Robotic Calibration for Active Viewpoint Generation and Object Grasping*, Proc. of IEEE Int. Conference on Robotics & Automation, May 2001.
- [8] D.C. Brown, *Decentering Distortion of Lenses*. Photometric Engineering, pp. 444-462, Vol. 32, No. 3, 1996.
- [9] Belongie, Serge, "Rodrigues' Rotation Formula," From *MathWorld* – A Wolfram Web Resources, created by Eric W. Weisstein.
<http://mathworld.wolfram.com/RodriguesRotationForumula.html> (version current 7 December 2006)
- [10] G. Hager, WC Chang and A.S. Morse, *Robot Feedback Control Based on Stereo Vision: Towards Calibration-Free Hand-Eye Coordination*. Proceedings of IEEE Intl. Conference on Robotics and Automation, San Diego, Ca, May 8-13, 1994.
- [11] G. Hager, *A Modular System for Robust Position Using Feedback from Stereo Vision*. IEEE Transactions on Robotics and Automation, August 1997.
- [12] H. Singh, A. Can, R. Eustice, S. Lerner, N. McPhee, O. Pizarro, C. Roman, *SeaBED AUV Offers New Platform for High-Resolution Imaging*, EOS, Transactions of the AGU, vol 85, no 31, pp 289,294-295, August 2004.
- [13] OpenCV Documentation,
<http://www.cse.iitb.ac.in/~sharat/current/cs687/opencv/> (version current 20 November 2006).
- [14] M. Naylor, N. Scott, E. Atkins and S. Roderick, *Toward Autonomous Sampling and Servicing with the Ranger Dexterous Manipulator*. AIAA Infotech@Aerospace Conference. Crystal City, VA, September 2005.
- [15] E.R. Davies, *Machine Vision: Theories, Algorithms, Practicalities 2nd Edition*.

- Academic Press, San Diego, 1997.
- [16] E. Trucco and A. Verri, *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, Upper Saddle River, NJ, 1998.
 - [17] M. Nixon and A. Aguado, *Feature Extraction & Image Processing*. Newnes, Oxford, Great Britain, 2002.
 - [18] B. Kimia, I. Frankel, A. Popescu, *Euler Spiral for Shape Completion*, International Journal of Computer Vision, Vol. 54, 2003.
 - [19] J. Russ. *The Image Processing Handbook*. CRC Press, Boca Raton, Florida, 1992.
 - [20] K. Plataniotis and A. Venetsanopoulos, *Color Image Processing and Applications*. Springer, New York, 2000.
 - [21] M. Galun, E. Sharon, R. Basri and A. Brandt. *Texture Segmentation by Multiscale Aggregation of Filter Responses and Shape Elements*. Proc. IEEE Intl. Conference on Computer Vision, 716-723, 2003.
 - [22] E. Sharon, A. Brandt and R. Basri, *Segmentation and boundary detection using multiscale intensity measurements*. CVPR, I:469-476, 2001.
 - [23] J. Malik, S. Belongie, T.K. Leung, and J. Shi. *Contour and texture analysis for image segmentation*. International Journal of Computer Vision, 43(1):7-27, 2001.
 - [24] S. Nayar, S. Nene and H. Murase, *Subspace Methods for Robot Vision*. IEEE Transactions on Robotics and Automation, October 1996.
 - [25] M. Stolen, *Rover Vision Localization System*. Space Systems Laboratory internal document #06-007, 2006.
 - [26] J. Smithanik, *Optimal Vision-Based Position Estimation of an Underwater Space Simulation Robot*. Masters Thesis, University of Maryland, College Park, 2004.
 - [27] J. Craig, *Introduction to Robotics Mechanics and Control, Third Edition*. Pearson Prentice Hall, Upper Saddle River, NJ. 2005.
 - [28] D. Musser, G. Derge and A. Saini, *STL Tutorial and Reference Guide, Second Edition*. Addison-Wesley, Boston. 2001.
 - [29] D. Scharstein and R. Szeliski, *A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms*. International Journal of Computer Vision, Vol. 47, April 2002.
 - [30] T. Kanade and M. Okutomi, *A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment*. IEEE Transactions on Pattern Analysis and Machine Intelligence, September 1994.
 - [31] A. Yezzi and S. Soatto, *Stereoscopic Segmentation*. International Journal of Computer Vision, Vol. 53, July 2003.
 - [32] S.B. Kang and R. Szeliski, *Extracting View-Dependent Depth Maps from a Collection of Images*. International Journal of Computer Vision, Vol. 58, July 2004.
 - [33] C. Sayers, *Remote Control Robotics*. Springer, New York, 1999.
 - [34] N. Scott, *Preventing Camera Occlusion for Visually-Guided Manipulators using a Line-Based Obstacle Avoidance Technique*. Master's Thesis, University of Maryland, College Park, In Progress.
 - [35] R. Dechter, *Constraint Processing*, Morgan Kaufmann Publishers, San

- Francisco, 2003.
- [36] TinyXml Main Page, <http://www.grinninglizard.com/tinyxml/> (version current 19 November 2006).
 - [37] Boost C++ Libraries, <http://boost.org/> (version current 19 November 2006).
 - [38] libdc1394: The Linux API for IEEE1394 / Firewire cameras, <http://damien.douxchamps.net/ieee1394/libdc1394/> (version current 19 November 2006).
 - [39] T. Cargill, *C++ Programming Style*. Addison-Wesley, Reading, Massachusetts.1992.
 - [40] S. Meyers, *More Effective C++*. Addison-Wesley, Boston. 1996.
 - [41] Doxygen, <http://www.stack.nl/~dimitri/doxygen/> (version current 19 November 2006).
 - [42] Gentleware – model to business: UML tools and services, <http://www.gentleware.com/> (version current 30 November 2006).
 - [43] Log for C++, <http://log4cpp.sourceforge.net/> (version current 30 November 2006).
 - [44] CxxTest, <http://cxxtest.sourceforge.net/> (version current 19 November 2006).
 - [45] Using and Porting the GNU Compiler Collection (GCC): Gcov, http://gcc.gnu.org/onlinedocs/gcc-3.0/gcc_8.html (version current 19 November 2006).
 - [46] Valgrind Home, <http://valgrind.org/> (version current 30 November 2006).
 - [47] Subversion, <http://subversion.tigris.org/> (version current 30 November 2006).
 - [48] The Trac Project, <http://trac.edgewall.org/> (version current 30 November 2006).
 - [49] CruiseControl Home, <http://cruisecontrol.sourceforge.net/> (version current 30 November 2006).
 - [50] S. McConnel, *Rapid Development: Taming Wild Software Schedules*. Microsoft Press, Redmond, WA, 1996.
 - [51] Point Grey Research Inc. – Home, <http://www.ptgrey.com/> (version current 19 November 2006).
 - [52] Underwater Lights and Cameras by DeepSea Power & Light, <http://www.deepsea.com/> (version current 19 November 2006).
 - [53] C. Carignan and R. Howard, *A Partitioned Redundancy Management Scheme for an Eight-Joint Revolute Manipulator*. *Journal of Robotic Systems*, 17(9):453-468, September 2000.
 - [54] C. Carignan and R. Howard, *A Skew-Axis Design for a 4-Joint Revolute Wrist*. *Proc. IEEE Int. Conf. on Robotics and Automation*, Washington, 3636-3642, May 2002.
 - [55] K. Kreutz-Delgado, M. Long and H. Seraji, *Kinematic Analysis of 7 DOF Manipulators*. *International Journal of Robotics Research*, Vol. 11, No. 5, 1992, pp. 469-481.
 - [56] S. Roderick or W. Smith, *DT21-0039 Test Report: Ranger Static Performance Measurements*, Space Systems Laboratory internal document #06-005.
 - [57] S. Roderick or W. Smith, *DT21-0041 Test Report: Ranger Dynamic Performance Measurements*, Space Systems Laboratory internal document #06-006.

[58] GNU gprof, <http://www.gnu.org/software/binutils/manual/gprof-2.9.1/gprof.html> (version current 20 November 2006)