ABSTRACT

Title of Document: DESIGN AND IMPLEMENTATION OF A COLLABORATIVE MODEL FOR ASTRONAUT - ROVER EXPLORATION TEAMS

Maxime Franck Ransan
Master Of Science,2006

Directed By: Assistant Professor Ella M. Atkins
Department Of Aerospace Engineering

Astronauts and robots must collaborate effectively to safely and efficiently explore harsh extraterrestrial environments. This research describes a planner/scheduler system to manage a hybrid rover-astronaut team, focusing on real-time contingency response. Collaborative exploration scenarios studied include observation, sampling, and rescue tasks. A search strategy with branching factor control trades planning time with search space completeness. The planner reacts quickly to anomalies when required but also can build optimal plans that minimize deployment time while maximizing science return. A six-wheeled rover was constructed to perform imaging tasks and carry payloads as needed. The rover is equipped with vision-based navigation, control, and path planning algorithms with obstacle avoidance for navigation in unmapped environments. A human astronaut interacts with its autonomous companions and the planner through a laptop GUI. A series of simulation and hardware-based tests evaluate planner performance and astronaut-rover collaboration scenarios.

DESIGN AND IMPLEMENTATION OF A COLLABORATIVE MODEL FOR ASTRONAUT - ROVER EXPLORATION TEAMS

By
Maxime Franck Ransan

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 M. Atkins, Chair
Associate Professor David L Akin
Associate Professor Robert M. Sanner

Dedication

I dedicate this thesis to my family for their continuous support.

Acknowledgements

I would like to thank my advisor Dr Ella Atkins for her support during this research. As my teacher, she was the one who raised my interest for the field of Artificial Intelligence; as my advisor she gave me the opportunity to work on a great project. Thank you for all the time spent correcting my English, and the patience she took to have me as a student.

Many thanks to the people in the lab who contributed to make the space system lab a wonderful place to work. I am particularly thankful to my graduate friends Leon A., Tim W., Shane J. Catherine M., Mike N. and Nick S., for their useful advice. I will always be grateful to my friend Joseph Easley, for his help and his interest in this project.

I want to thank my best friend and roommate Arnaud Zemmour. His friendship largely contributed to make those two years in the USA a wonderful experience.

Finally I would like to thank my parents Jean Paul and Marie Christine as well as my sister Sonia, who gave me love and support during those 2 years away from home.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

In future exploration missions astronauts will work with robots to accomplish ambitious scientific and engineering tasks (Fong and Nourbakhsh 2005). Planetary surface rovers must be capable of autonomous navigation and manipulation, and they also must appropriately interact with their human companions. A variety of mission scenarios will involve navigation to a set of locations where tasks such as site inspection, sampling, or construction must be completed. Cost and efficiency factors may favor specialist rovers that team to accomplish a broad range of tasks. With such a team, rovers must be assigned tasks based on their capabilities as well as expected traversal cost between worksites.

Rather than directing activities from a remote control station, astronauts will move in the same environment as their robot companions. Each astronaut will be assigned an initial set of tasks to accomplish, some of which may be completed alone and others will require the assistance of one or more rovers. Rovers may support astronauts by providing tools, storing samples, or relaying sensor data to a mission control station. Any rover tasked with directly supporting an astronaut activity will follow the lead of the astronaut, understanding their high-level role but requiring real-time characterization of astronaut motions to effectively operate at the same worksite. Even for missions in which rovers and astronauts operate at distinct worksites, rovers may be constrained to execute only nearby tasks so that they can quickly reach the

astronaut to provide support when requested. Rovers may be equipped with life support supplies as well as tools and sensors, providing improved safety for the astronauts as well as increased operational efficiency.

Activity planning and execution for a rover team has previously been investigated with systems such as CLEaR (Closed-Loop Execution and Recovery) (Estlin et al 2005a/b). CLEaR relies on the ASPEN planner, which incorporates an iterative repair algorithm to facilitate dynamic response. This system has been successfully demonstrated but has not yet been extended to reason about direct collaboration with humans. Other researchers (e.g., (Fong Thorpe and Baur 2003)) have specifically focused on the interaction between humans and robots, but primarily with the humans as "supervisors" or "reasoning experts" rather than as "mobile agents" operating in the same environment as the rovers.

This research focuses on activity planning and execution for a collaborative astronaut-robot team. Figure 1 illustrates a mission in which multiple tasks must be completed by a team of two astronauts and three rovers.

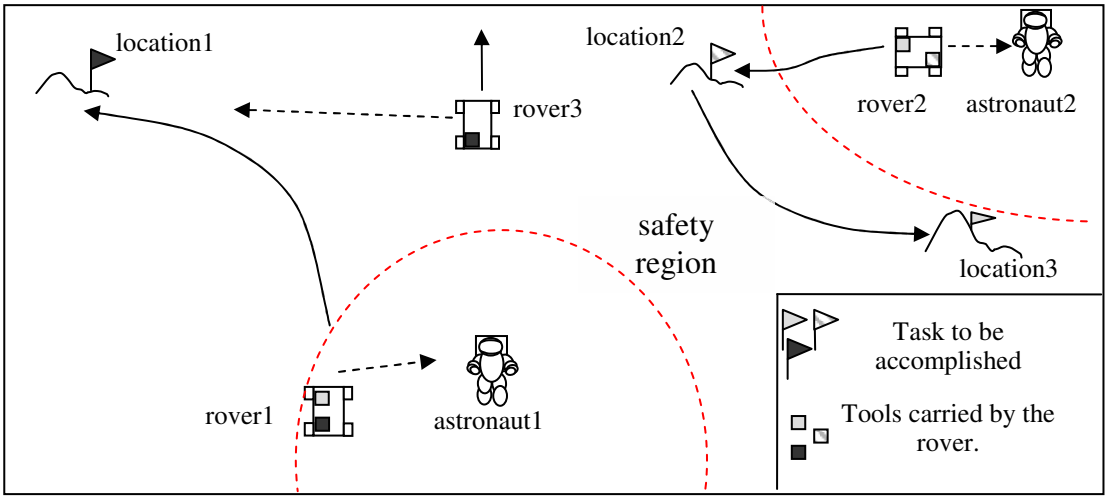


Figure 1.1: Astronaut-Rover Team Task Execution.

Figure 1.1 flags represent tasks to be performed at each location (e.g., dark gray for rock sampling, striped for construction, light grey for image capture). The rovers are also marked with squares representing tools they can use to accomplish specific tasks (e.g., a grasp/scoop end effector to retrieve a rock sample). As illustrated, `rover2` is capable of performing construction and image capture tasks but cannot execute the rock extraction task shown at `location1`. Astronauts are presumed to always be capable of undertaking a task, but in certain cases task completion by an astronaut requires the assistance of a rover.

Provided all marked worksite targets are accurately mapped and task completion requirements are known in advance, the task allocation/planning problem is straightforward, and task schedules can be optimized based on traversal distances and

task execution capability and cost constraints. However, extraterrestrial worksites are likely not well-mapped, at least initially, and static plans cannot capitalize on the acute sensing and real-time decision-making capabilities of astronaut team members. In reality, as an SEVA (surface extra-vehicular activity) mission progresses, astronauts can be expected to frequently alter the "default" plan to explore new targets of opportunity or to adapt when unanticipated situations arise. The rovers, of course, can also plan for contingencies and search for opportunities, but such behaviors would likely be quite limited in scope thus more predictable.

Astronaut safety is of paramount importance in all SEVA activities. Drawing analogy to the "buddy system" used by hikers and divers, the Figure 1.1 scenario calls for each astronaut to be accompanied by a rover that need not directly participate in the astronaut's task, but that must not stray further from the astronaut than some maximum distance. Unless assistance is requested or a cooperative task is scheduled, the "buddy" rover is free to complete its own tasks and periodically verifies the astronaut is safe through sensors (e.g., visual) or direct communication. In Figure 1.1, `rover1` is the "buddy" that must remain within the safety region of `astronaut1`. This rover cannot reach `location1` to accomplish the rock sampling task, so `rover3` must be assigned instead of `rover1`. Should the astronaut require assistance, `rover1` can quickly provide life support or other backup equipment without extended traversal.

Robust fully-cooperative mission planning and execution require substantial research in areas ranging from dynamic, multi-agent task allocation to cooperative human-robot motion planning. The key contribution of this research is the design and implementation of an architecture that is able to compute optimal plans for a team of astronauts and rovers, while dealing with contingency events in a safe and real - time manner.

1.2 Research Objectives

With the overall goal of managing in real-time a team of rovers and astronauts, the research in this thesis may be classified within three categories:

1 - *Implementation of a centralized planning tool* to plan, schedule, and coordinate task-level actions for each robotic and human agent in the field. The implemented forward search algorithm prioritizes astronaut safety and enables design-to-time [5] adjustment to trade off planner execution time and schedule optimality.

2 - *Design of a multi-agent coordination executive* to distribute actions to exploration team members and to retrieve information from each agent, including sensor data feedback along with astronaut feedback and intentional plan deviations. This coordination executive also must identify situations in which plans/schedules must be modified. Both planner and executive must react in real-time to unexpected events to maintain system safety, an overriding goal previously identified in architectures such as CIRCA [6].

3 – *Design and implementation on a real rover* to test the planning/coordination architecture with the less-predictable performance of a deployed hardware system. To achieve this objective, a six-wheel rover was constructed and onboard avionics were

assembled and programmed, including an inertial and vision-based navigation system capable of obstacle avoidance, environment mapping, path planning, and dead reckoning vehicle actuation.

1.3 Outline

This thesis describes the design, implementation, and testing of the aforementioned planning and coordination architecture for teams of astronauts and rovers. Chapter 2 reviews related work in areas ranging from mobile robot navigation to human-robot interaction (HRI) and task planning/scheduling algorithms. Chapter 3 focuses on the details of our centralized planning tool, followed by Chapter 4 that describes multi-agent simulation results obtained to characterize the performance of this planner/scheduler. Chapter 5 describes the rover hardware while Chapter 6 focuses on the onboard navigation and control algorithms and their performance. Chapter 7 presents a suite of experiments that demonstrate and characterize the performance of the planning/coordination architecture given a group of simulated rovers plus the "real" rover and an astronaut-in-the-loop. Chapter 8 concludes the thesis and identifies directions for future work.

Chapter 2: Previous Work

This thesis has three main objectives that span a wide range of research areas. The main contribution is the implementation of our design-to-time planner; consequently Section 2.1 describes action planning techniques. This research also requires the design and implementation of an onboard rover navigation system which covers areas such as machine vision, mapping, obstacle detection and path planning. Successful techniques as well systems are presented in Section 2.2. Section 2.3 introduces the field of Human Robot Interaction, overviewing work and continued challenges from the literature.

2.1 Task Planning and Scheduling

Action or decision planning requires finding a set of actions that transforms the world from an initial state to a desired goal state or states. Due to the large number of possible states, this problem is usually NP-hard, requiring time exponential in the number of actions and features to compute a solution. Nevertheless, algorithms have been implemented to reduce the search space thus improve worst and average-case execution times. This section will introduce existing planning/search techniques, with more detail available in [7].

2.1.1 Classical Planning

Classical planning techniques deal with restricted-state transition problem which are defined as being deterministic (each action results in a unique new state), static (if no actions is applied the world remains the same), finite (the system has a finite number of states), and fully observable (the state of the world is fully accessible). Classical planning is also referred as STRIPS planning, as the STRIPS planner was one of the earliest attempt to solve restricted-state transition systems [8].

Perhaps the most fundamental planning techniques rely on *forward search* where states are expanded until the desired state is found. Forward search includes algorithms like A-star, best-first-search, breadth-first-search, and uniform cost search. *Backward search* is the inverse of forward search: the search starts from the goal state and tries to regress to the initial state. The famous *STRIPS* algorithm [8] is a reduced version of backward search that identifies a satisfying solution relatively efficiently but that may not always find an existing solution due to an incomplete search strategy (i.e., as with the famed Sussman anomaly).

Partial Order Planning (POP) was introduced soon thereafter, maintaining computational efficiency similar to STRIPS but enabling solution to problems such as the Sussman Anomaly through least-commitment action ordering. The planner UCPOP [9] was perhaps the most common POP planner and is still frequently used in the classroom and beyond. Plan-space planning is a subsequent evolution of the original POP algorithm. In a *PSP procedure* (Plan Space Planning procedure), the initial plan is composed of two actions a_0 and a_∞ which set up the problem. The

procedure refines this initial plan by solving open goals (Actions with preconditions that are not supported by another action) and threats (conflicts between two actions).

2.1.2 Modern Search Techniques

Because classical planning techniques do not scale to large problems, new techniques were devised more recently. With strategies such as Simple Temporal Logic and Hierarchical Task Network, the main advancement was the implementation of domain expertise in the planner. It enables the user of such techniques to implement domain-specific rules in order to control and constrain the search enabling faster computation.

Recent *planning-graph techniques* develop a solution through a procedure that bridges state space planning and plan space planning. The solution is specified in the form of a sequence of action sets:

$$\Pi = \{\{a1, a2, a3\}, \{a4, a5\}, \{a6\}, \{a7, a8\}\}$$

With this representation, actions *a1,a2* and *a3*, in this example, can be executed in parallel or in any order, but must execute before *a4* and *a5* which must execute before *a6*, etc. *Graphplan* techniques were first introduced in [10] and are generally considered among the greatest achievements in the AI planning community.

The development of control rules in Simple Temporal Logic (STL) has significantly improved the performance of traditional forward search algorithms. STL is a logical

formalism that enables the search algorithm to prune nodes during the search. STL was first introduced by [11]; this implementation remains among the most efficient.

A practical planning technique applicable to a diverse set of complex problems utilizes the Hierarchical Task Network (HTN). The input of an HTN planning problem includes a set of methods that can be viewed as domain dependent "recipes". The planning process consists of recursively decomposing non-primitive tasks until primitive tasks are reached. More details can be found in [7].

2.1.3 Scheduling

As planning is the problem of finding a valid set of actions to achieve a goal, scheduling focuses on time and resource allocation for this set of actions. As described in [7], historically, these two problems have been treated separately; the set of actions to perform was already known in most of the scheduling problems (for instance organizing jobs in a factory), while planning problems did not pay attention to resource allocation. Separating the two problems is certainly too restrictive but can help speeding up the search for a solution. Scheduling can be seen as an optimization problem where different criteria are minimized depending on the application. Here is a non exhaustive list of some of the cost criteria minimized by schedulers:

- Makespan = maximum ending time of the schedule.
- The total number of late actions.
- The peak resource usage.

- The total quantity of resources allocated (weighted uniformly or diversely with multiple resources).

A very well studied scheduling problem is machine scheduling where several jobs must be accomplished on several machines. This NP-hard problem is generally not solvable but approximation techniques have been implemented to scale large problems (see [12] for a survey of those methods). Reference [25] presents an integrated approach to planning and scheduling. This method is based on chronicles which is the set of timelines for every state variable of the problem. The planning/scheduling procedure is based on threats and open goals; more details can be found in [7].

In 1999 the successful test of the *Remote Agent* architecture on the Deep Space One (DS-1) Mission ([13] and [14]) proved the feasibility of a robust planning/ scheduling tool. This system was able to command autonomously a spacecraft in order to meet exploration goals and system safety requirements during a deep space mission. The *Remote Agent* is composed of 3 modules:

- A *Planner/Scheduler*: computes detailed temporal plans of activities that must be executed in order to accomplish mission goals.
- *Smart Executives*: execute the activities computed by the planner.
- *Mode Identification and Recovery*: Monitors the state of the spacecraft; identifies failures and tries to correct them.

The *Planner/Scheduler* component is based on an *Iterative Refinement Search* search engine. Every open goal, threat, or uninstantiated temporal subgoal is repaired incrementally using a known method until the plan is consistent. It can be viewed as a best first search in the repair method space. A set of repair methods is applicable in each node, and when one of them leads to an inconsistent plan the search engine backtracks and tries another repair method.

Since March 2003, the EO-1 Autonomous Sciencecraft Experiment [15] has been tested and used on the Earth Orbiting 1 satellite in order to maximize data scientific data collection. This system includes:

- *Onboard Science Algorithms* which allow the spacecraft to autonomously detect interesting science events. Those algorithms are based on machine learning techniques.
- *Execution Software* which enables event-driven processing and low-level autonomy.
- *The Continuous Activity Scheduling Planning Execution and Replanning (CASPER)[16]*

The CASPER planner is based on an iterative repair algorithm that detects threats in a plan and searches for methods to repair them. The spacecraft succeeds in improving the science return data by directing the satellite to regions of interest and selecting only interesting data to download to earth.

2.1.4 Planning under uncertainty

Classical planning assumes that every action produces a unique state but there are numerous real-world situations where this is not the case. When dealing with hardware, failures or off-nominal events can occur, implying that several states can be reached with non-negligible probability when executing an action. This class of problem is called planning under uncertainty. Two solutions approaches are commonly used to address such problems:

- Avoid the complexity, by detecting the unexpected state and computing a new plan starting from this new state. This is a reasonable solution if unexpected states occur with sufficiently low probability and if planning time is not important.
- Plan using Markov Decision Process (MDP) or Model Checking approaches. These planners build solutions that handle all the possible states that can be reached.

Planning based on Markov decision processes represents the planning problem as an optimization problem where the goal is to find the action policy that has the highest utility function. Several "output" states are defined for every action, and probabilities are associated with each of these child states. An MDP policy is a function from (reachable) state space to the space of planned actions. For each reachable state, the policy indicates which action to perform. The planning algorithm searches for the policy that has the highest overall utility, a function of both the value of being in each state and the likelihood of actually reaching that state.

Planning based on Model Checking is used when no probability of the different outcomes are available. Those planning algorithm uses Model Checking algorithms, more details are provided in [7].

2.1.5 Multi Agent Planning

Multi - agent planning has received significant attention since many systems are composed of individual entities (agents) that are able to plan and perform tasks autonomously and communicate their intentions and knowledge with other agents. The goal of multi-agent planners is the global achievement of a common goal (set). Several techniques have been implemented, the best choice of which depends highly on the specific agent properties and objectives. Two main approaches have been developed:

- *Centralized planners:* each agent receives orders from a central entity and feeds back status and state information [3]. This architecture enables a global view of the entire system and allows effective coordination through common knowledge. However centralized systems are only possible when all the agents can communicate with the centralized systems; this is not the case when communication is restricted to close neighbors. The major drawback of a centralized architecture is also the fact that failure in the centralized system implies failure of the entire system; this safety issue can be critical. However, this issue can be mitigated if

the centralized system has redundancies or is more protected than the other agents. Both would be the case for a centralized "mission control" or "base station" managing a group of planetary rover and astronaut agents.

- Distributed systems are more difficult to robustly design and maintain since no global view of the world is available to any agent. Decentralized architectures are still a great challenge since very limited (or no) communication is available. The main advantage is that the achievement of the global goal does not depend on a unique entity ensuring great safety through redundancy. In [17] the authors use a Market-oriented technique to allocate resources to multiple agents. Based on their current status each agent sends a bid or creates an auction through central authority which decide on the resource allocation.
- In [18] the multi-agent system is modeled using a Markov Decision processes method. This system is composed of a set of states, a set of actions, the probability distribution of the outcome of an action and the probability distribution of the agent observation of a state. Each agent chooses an action that maximizes a utility function. [18] proposes to include knowledge of other agent(s) in the agent definition to improve the overall performance.

2.2 Mobile Robot Navigation

Robot navigation has been studied extensively but challenges remain unsolved, particularly in extraterrestrial environments that are unmapped, laden with obstacles, and that do not have GPS coverage. Additionally, embedded resources on small autonomous robots are limited requiring simplification of even existing computationally-complex mapping and navigation algorithms. To introduce the technologies adapted for the rover system designed and deployed in this work, relevant machine vision techniques used for mapping and obstacle detection are introduced, followed by a more detailed characterization of the mapping and localization problems. The last section presents path planning algorithms of relevance to any autonomous mobile agent with focus on planetary surface rover implementations.

2.2.1 Machine Vision

Cameras, monocular or more commonly a stereo pair, have now become to the standard for obstacle detection. Machine vision is an active field of research with a wide variety of applications ranging from inertial navigation to surveillance and relative navigation applications such as docking and formation/station-keeping. The choice of vision algorithm is highly dependent on the environment, categorized at a high-level as:

- indoor
- unstructured outdoor (e.g., Mars field)
- structured outdoor (e.g., on-road navigation)

As described in [19] one of the great successes for navigation in outdoor structured environments is NAVLAB [20] project and its various incarnations. This vision system processes each image based on color and texture information and classify cells as either *road* or *not road*. A neural network [21] is then used to determine the correct steering angle based on aggregate cell classifications. NAVLAB has been demonstrated capable of driving a car at 55 miles per hour on a structured highway.

For unstructured outdoor navigation, the primary challenge is obstacle detection and terrain characterization (i.e., traversability) in an unknown environment. As previous knowledge of the terrain is often not available, general techniques must be developed. With stereo vision, typical techniques match corresponding pixels in the two camera images, then the distance can be computed by:

$$d = \frac{f * B}{\Delta_{pixel}} \qquad (2.1)$$

where d = distance from the camera, f = focal length, B=distance between cameras, and Δ_{pixel} = difference between the 2 pixels. This equation assumes undistorted image data and that the cameras have the same orientation.

For high-speed mobile robot navigation, researchers have focused on improving the efficiency of the classical stereo algorithm rather than enhancing the fundamental algorithms used to select corresponding points. Reference [22] describes a successful implementation where a pair of cameras is used to compute a Digital Elevation Map of the terrain. In tests, the rover was able to precisely estimate its future configuration while navigating rough terrain.

For indoor navigation, vision algorithms were mainly used to detect and track distinctive features for self localization (see Section 2.2.2). Features are unique groups of pixels typically detected by processing a grayscale image with one of the following:

- *Edge detection:* [23] the edge of walls of or distinct objects are usually good features (especially corners where several edges intersect)
- *Difference of Gaussians*:[24] By applying two gaussian convolutions to the image (with two standard deviations) and by subtracting them, distinctive areas of the image are highlighted.
- *Laplacian of Gaussian*:[25] By applying a Laplacian of Gaussian kernel to the image, distinctive areas of the image are highlighted.

2.2.2 Localization and Mapping

The mapping problem is highly correlated with the localization problem [26]. Mapping requires spatial knowledge of a physical environment, thus each new token in an image must be situated relative to the current robot position. Error in localization results in mapping error. Utilizing the same image data to discern both mapping and localization information is known as SLAM (Simultaneous Localization and Mapping).

Historically mapping has been divided into metric and topological approaches. The metric approach consists of storing the geometric properties of the world while the

topological approach stores the world with items (objects/features) and connections between them. One of the most famous metric maps is the occupancy grid [27] in which the world is represented by fine-grained grids containing the probability of occupation likelihood. A topological substitute was designed in [28], using sets of polyhedra to represent the geometry of the world.

As described in [26], several factors complicate mapping and localization processes:

- Measurement error is cumulative. Acquisition of multiple measurements and use of filters can improve local state estimates but cannot cancel global mapping/localization errors.
- Depending on the map structure, the dimensionality of the problem can vary greatly. For instance, mapping geographic features requires tens to hundreds of elements, but mapping a detailed surface of an outdoor environment requires millions of values.
- The correspondence problem requires identifying unique item(s) from different images. The complexity of the correspondence problem can again grow exponentially with the number and visual characteristics of the items to be identified.
- Additional complexity arises when an environment is dynamically changing. Localization in this case will be time dependent requiring additional map storage and image processing resources.

Mobile Robot SLAM

One of the most popular estimation algorithms adapted to mobile robot localization and mapping is the Extended Kalman Filter (EKF), capable of use in static and dynamic environments. EKF-based systems typically employ maps with landmark features. Algorithm speed is a function of number of landmarks to be tracked. This technique has been successfully applied in [29] where the position of a single camera as well as a set of features was accurately computed in real-time. To avoid drift error over an image sequence, the camera must always reacquire a sufficient set of features from earlier images to maintain consistency in estimates despite dead reckoning errors. The main advantage of the EKF is its capacity to be implemented in real-time, but real-time performance is only possible with a small number of landmarks. In typical outdoor environments, a robot must keep track of numerous features (e.g., stored as a digital elevation map or occupancy grid), creating a high-dimensional map structure inappropriate for an EKF to manage in real-time.

Another popular SLAM technique is based on the Expectation Minimization algorithm (EM) developed by [30]. This algorithm has been proven most efficient for the correspondence problem. It tries to find the most likely map by performing a hill climbing search over the space of all maps. This process cannot be done incrementally for each new sensor reading, however, making this algorithm inapplicable in real-time situations.

Mapping Algorithms

Algorithms that map but do not localize rover position require the position of the rover at each map update, which makes the problem simpler but does not solve it. Indeed, map error will highly depend on cumulative dead reckoning error. The most popular mapping algorithm is the grid occupancy algorithm introduced in [27]. This algorithm can generate a map with noisy or incomplete data and has been widely used for sonar sensors or laser range finders as their probabilistic noise signatures are well-known. With this algorithm, each time a new measurement is made, the cells are updated with the following formula which is derived from Bayes' rule:

$$\frac{p\left(m_{x,y} \mid z^t, s^t\right)}{1-p\left(m_{x,y} \mid z^t, s^t\right)} = \frac{p\left(m_{x,y} \mid z_t, s_t\right)}{1-p\left(m_{x,y} \mid z_t, s_t\right)} \times \frac{1-p\left(m_{x,y}\right)}{p\left(m_{x,y}\right)} \times \frac{p\left(m_{x,y} \mid z^{t-1}, s^{t-1}\right)}{1-p\left(m_{x,y} \mid z^{t-1}, s^{t-1}\right)} \qquad (2.2)$$

where $z^t = \{z_1, z_2, z_3 \ldots, z_t\}$ represents the sensor measurements at times t=1, … n, $s^t = \{s_1, s_2, s_3 \ldots, s_t\}$, represents the pose estimations at times t=1, … n, $p\left(m_{x,y} \mid z^t, s^t\right)$ is the probability that cell (x,y) contains an obstacle at time t given z^t and s^t, $p\left(m_{x,y} \mid z_t, s_t\right)$ is the probability that cell (x,y) is occupied based on a single sensor measurement at time t taken when the robot is at pose s_t (also called inverse sensor model), $p\left(m_{x,y}\right)$ is the prior probability for cell (x,y), and $p\left(m_{x,y} \mid z^{t-1}, s^{t-1}\right)$: probability that there is an obstacle in cell (x,y) at time t-1. Note that if there is no previous knowledge then $p\left(m_{x,y}\right)$ is set to 0.5 and this term has no effect on the above equation. Other similar algorithms have been designed, such as that in [26]. Generally, these stochastic techniques are derived from Bayes' rule.

2.2.4 Path Planning

For a robot to navigate safely, a trajectory must be computed and followed that avoids obstacles while meeting kinematic and dynamic constraints. Path planning has been extensively studied and adapted to a variety of domain-specific conditions. Environments can be mapped or unknown, static or dynamic, safe or unsafe (requiring real-time response), open or laden with obstacles. The kinematic and dynamic properties of the robot that will follow the path also play an important role in the path planning process, particularly for a nonholonomic system with limited actuator authority.

The majority of deployed mobile robot navigation systems are based on search graphs [31]. Let a configuration be defined as a unique set of positions/angles for each degree of freedom of the robot, but not including derivative information (speeds). Since the goal is to find a feasible path between two configurations, the discrete search solution is to search the space of intermediate reachable configurations until a feasible or optimal solution is identified. Search algorithms employed by graph-based path planners include *forward search* and its derivates *backward-search*, *bi-directional search*, *graph-search*, *A-star* or *best-first-search*.

The most distinctive feature between various path planner implementations is the manner in which new configurations are generated. The configuration generation process depends on the type of robot (rover, manipulator, aircraft, spacecraft) , the

map structure, and knowledge of the environment (certain/uncertain, complete/incomplete, static/dynamic). New configurations can be randomly generated like in [32] where a planner has been implemented for a vehicle pulling a trailer, or maps can be decomposed in connectivity cells as shown in [31].

As described in [33], when the map structure is topological, a *Voronoi diagram* can also be used to generate viable configuration sequences. *Visibility maps* can also be used, in which case algorithms with polynomial complexity can be used to determine a correct path. *Potential fields* have been frequently used, assigning a repulsive force to every obstacle while an attractive force is assigned to the goal position. By summing the forces and moving in the negative gradient direction, the robot is above to locally avoid obstacles while driving to the goal. Unfortunately *potential fields* can lead to local minima and the robot can become stuck.

When the map is grid based, each cell has associated cost that depends on its position relative to the goal and the probability that there is an obstacle. The search algorithm looks for the goal using forward search techniques and selects cells with the best score. Depending on the complexity of the environment and the detail of the map, this method can be computationally expensive.

2.3 Human Robot Interaction

Human Robot Interaction (HRI) is receiving increased attention due to recent progress in artificial intelligence, computer studies, speech simulation, and better interaction models. Historically, Human Computer Interaction (HCI) was the focus of attention, concentrating on the development and evaluation of algorithms and interfaces that promote efficient and intuitive interaction between human and computer. The expansion to HRI requires additional focus on physical attributes and actions of both human and robotic agents as they move about and interact within a potentially common environment. Human Robot Interaction can be separated into two categories, although there is some overlap:

- *Task-Oriented Interaction* [34]*:* These HRI systems take advantage of unique robot and human skills and mix these skills to effectively execute tasks. Optimality with respect to cost or utility value(s) is the primary performance evaluation criterion, and robots are clearly considered more as a collaborative partner rather than a social agent. Communication in such systems is highly engineered and task oriented. The human usually needs to learn the interaction protocol for the team to properly interact. This category includes all form of teleoperation where the human drives the robot and receive feedback from it. As we will see later, the HRI scheme presented in this thesis is primarily classified as task-oriented but also shares some aspects with the second category.
- *Robot as Social Partner* (see Section 2.3.1). This category of HRI researches strategies to promote "natural" interaction between with

a human collaborator/partner. Research in natural language processing and understanding as well as gesture-based communication all fall within this category. Because of the complexity in achieving the breadth possible during human-human interaction, most of the research in this category has focused on limited social behavior. Therefore those social robots are for now restricted to roles of task assistant that requires explicit directives, immobile companions (e.g., Kismet [35]), or pets (e.g., the Aibo).

2.3.1 Challenges in Human Robot Interaction

As described in [36], all task-oriented robotic systems must possess a certain set of capabilities:

- Cognition (path planning, decision making, task planning)
- Perception (environment sensing)
- Action (mobility and/or manipulation)
- Human Robot Interaction (data communication, feedback display)

Robots that interact with human(s) have additional design requirements:

- ***Human perception***: The robot must be aware of human behavior and intent potentially affecting the robot's actions and interactions. This design issue is critical for correct task execution. Fields ranging from teleoperation to video surveillance have developed techniques to perceive human behavior and commands. Humans

communicate through a variety of perceptual media, not all of which are effectively interpreted by today's robotic systems. Nonetheless, to aptly communicate with their human companion(s), the robot must be aware of human intention without a dedicated interface that increases workload for the human astronaut. Speech and gesture recognition researchers are improving the ability of robots to communicate with humans in their "native languages". For instance in [4], researchers have implemented a visual gesture-based driving system. Equipped with a pair of camera, the robot is able to identify human body parts and track them so as to determine the desired speed and orientation commands. In [27], the system is able to fit a 3-D temporal motion model of a human to generate synchronized sequences of human poses and positions. In [38], the identification system has demonstrated the ability to recognize human behavior (one or multiple person) from video surveillance, enabling detection of urban violence.

- ***Natural human-robot interaction:*** A social robot must act socially in order to interact properly with the human. Several researchers have focused their work on the appropriate behavior that the robot must have in order to approach the human. Their work ranges from human-aware trajectory computation to the implementation of social convention and norms. In [49], the path planning algorithm

takes into consideration the position and pose of the human and computes a trajectory that respects social rules. For instance, the robot will not approach the human from behind and surprised him. Instead the robot will favor a trajectory, perhaps suboptimal with respect to traversal distance, that makes the robot visible to the human as soon as possible.

- ***Readable Social Cues:*** For natural communication with a human a robot must indicate its internal state as well as communicate its intention in a transparent manner. Research has focused on expression of emotion through facial expression, body gesturing, and vocalization. In [35], the platform Kismet is introduced. Kismet is a robot head equipped with fifteen actuators capable of conveying intentionality through facial expression. The robot has a "motivation" system composed of:
    - **drives** which are basic needs stimulated by objects or fatigue (generated over time). For instance Kismet can be configured to be attracted by bright colored objects.
    - **Emotions** which depends on its affective state, computed by summing contributions from the drive system as well as the behaviors.

  This system has proven to successfully interact with a human in an infant-like manner.

- ***Real-time performance:*** In order to properly interact with a human, the robot must perceive human intentions, make plausible decisions, and show its intention in real-time. Besides appearing decidedly "non-human", extended delay would add uncertainty and misunderstanding to the interaction.

The following section describes some of the research in different topic related to human-robot interaction. There is no complete system that solves all the design issues we discussed. Instead, researchers and engineers have worked in specialized systems deployed with specific interaction protocols.

2.3.2 Teleoperation

In hazardous or inaccessible environments such as space, robotic systems will be controlled from a remote station rather than in close proximity. Teleoperation is a form of interaction between a human and a robot that has received a lot of attention from the research community. As described in [40], its major challenges are:

- *Time Delays.* The usual time delay for an Earth-orbiting space robotic application would be approximately 6 seconds. This delay forces the users to adopt the "move and wait" strategy, that usually slow the procedure and causes instability.
- *Limited data throughput.* Spacecraft can communicate through a very limited set of ground assets. Feedback is therefore restricted

implying greater difficulty in communicating detailed information at high bandwidth.

Researchers have mainly worked on providing better feedback and information to the user despite the time delays. For instance, in [40], a system was designed to provide better and faster teleoperation through model-based teleoperation and haptic feedback and was demonstrated on a peg-in-hole task with a 6s time delay. In [41] a predictive display has been implemented in order to improve the teleoperation of underwater free-flying vehicle under time delays. Recent research [34] has also focused on classifying manipulator trajectories in order to determine the type of task the user is trying to perform. This is a step toward supervisory control in which the user can effectively specify commands through motions rather than keystrokes.

2.3.3 Human-Robot Collaboration

Hybrid teams of rovers and humans are difficult to manage with human or synthetic intelligence. Researchers in [42] designed an architecture where astronauts manage a potentially large team of rovers. However, the workload required to manage the rovers reduces the astronaut's productivity in other tasks, whether in close physical proximity (EVA, extra-vehicular activity) or at a remote worksite (e.g., a base station). Because of management overhead, this architecture can be adapted to a wide range of tasks/goals and can span a spectrum of so-called sliding autonomy levels (from "pure" teleoperation to full autonomy) [43]. This system is able to prioritize unexpected requests from the agents so as to minimize idle time in which rovers wait for instructions or repair. Initial tests show promising results but the project is still ongoing.

Research in [44] focuses on in situ collaboration between astronauts and rovers. A scenario has been demonstrated with two rovers (K9 and Gromit) and two humans: one astronaut traversing with the rovers and one crewmember in a habCom overseeing the scene and controlling the rovers. The astronaut explicitly asks for a specific rover to take a picture of the astronaut pointing at the rock, then the astronaut orders to the other rover to investigate this rock before the end of the EVA. Even if the test was successful, the previous scenario required substantial human oversight because a rover operator intervened every time new goals were requested for the K9 rover. Nevertheless this test has demonstrated the kind of scenarios and interactions NASA is planning for future missions.

Chapter 3: Planning For Astronaut - Rover Teams

Before designing a planning algorithm, the goals, task attributes, and challenges specific to human-robot team planning must be understood. This chapter begins by identifying what we believe are realistic scenarios for future planetary exploration by a team of astronauts and rovers. Next, the planning architecture is outlined with specific algorithms defined for the centralized team planner and coordination executive responsible for distributing tasks and data among planner and mobile (human and robotic) exploration agents.

3.1 Astronaut-Rover Team Mission Scenarios and Challenges

The initial goal of the team is to accomplish a set of exploration tasks potentially at different locations. Each agent (rover or astronaut) has a limited set of competences that constrains the individual agent's abilities but that enables the aggregate team to perform a wide range of tasks. For example, every astronaut is able to pick up a rock but only "geologist" astronauts are able to identify which rocks are of scientific interest. Typical rovers would have a suite of sensors and more sample storage capacity than an astronaut but would be more limited in their ability to select targets (e.g., rocks) of interest and rapidly collect desirable samples.

Multiple competences can be required to perform even a simple task. For instance, a rock sampling task might require not only a geologist to identify and pick up the rock, but also a rover to store the sample and return it to the base. For the task to be executed the planning tool must decide which agents must perform the task and when they will execute the task together.

Scenarios ranging from sampling to habitat construction may require a specific task ordering (precedence). For example, before collecting a rock sample in an unknown area, it would be advantageous to first take a high resolution picture of the surroundings, select site(s) of interest, then organize astronaut/rover teams with appropriate competences to explore sites of interest.

Although rovers may encounter unexpected environmental conditions and improbable internal states, a robust robotic system will follow the specified plan to the best of its abilities. The very creativity and adaptability that make an astronaut a valuable explorer challenge a hybrid robot-astronaut planner. Although trained astronauts will do their best to accomplish assigned tasks, they are much more likely to deviate from a planned task schedule. Additionally, astronaut safety must be a top priority, even at the expense of goal achievement. Future rovers and astronauts will likely carry backup life support equipment for emergency situations (e.g., astronaut running out of oxygen or unable to move due to injury). As team manager, the planner and coordination executive should therefore be able to detect the emergency situation and to respond to it in a quick and appropriate manner.

Another scenario where astronauts might behave unexpectedly is opportunistic exploration, situations in which they choose to deviate from the current plan to explore a site/perform a task they perceive as higher-priority. Since astronauts are indisputably the superior "creative" intellect given current technology, the planner must support such deviations. Because the astronaut chooses not to accomplish the original planned task set, the plan is no longer valid. Both emergency and voluntary plan deviations require rapid detection and replanning/plan repair capabilities, primarily for safety in the former case and for efficiency, especially given cooperative tasks, in the latter case.

Unexpected cooperation is yet another scenario that could arise. An astronaut could require assistance from a rover to complete a task, or a rover could lose a capability (e.g., broken sensor) and require assistance from an astronaut or rover to continue its task or return to base. As an intuitive example, a "geologist" astronaut may opportunistically encounter a rock outcropping of potential interest but requires specific sensors or tools only possessed by a rover teammate to analyze, extract, or stow the geological sample. Although such situations must be dealt with expediently to prevent inefficiency for the team and frustration for the astronaut(s), safety is always the priority since no rescuing agent will be used to provide such assistance to the astronaut.

Due to power and life support limitations as well as the inherent dangers with EVA, maximizing exploration productivity is a top priority. In planning terms, given a set of tasks to accomplish, it is important to identify and successfully execute a plan that maximizes the number of goal-oriented tasks while maintaining safety and minimizing overall execution time. Given traversal requirements, this problem can be related to the Multi Traveling Salesman Problem [45] but has significantly more constraints. In our situation the salesmen are not allowed to go everywhere, and "cities" are ordered, which makes traditional heuristics impossible to apply. Even if capability constraints reduce the search space, the problem of finding the optimal task allocation remains NP-hard, implying exponential time to compute the solution. Obviously, for large-scale, long-duration missions, exponential computational time renders full plan/schedule optimization an unrealistic requirement.

Reliable and efficient contingency response is an important challenge to address as well. Unexpected events that occur during EVA will generally require a quick and appropriate response. The challenge is then to provide a planning tool that is able to compute, within a time constraint, a valid plan that takes into consideration the specifics of the "unexpected" emergency situation. The coordination executive must also be guaranteed to detect at least safety-critical emergencies and to efficiently relay updated response plans to the team.

Acting as a mission control or base station, our centralized planner must be able to plan actions that coordinate activities among all members of the astronaut-rover team.

Planning parallel activities can also require extra computation and dedicated data structures, especially when multiple agents must cooperatively execute one or more tasks.

3.2 Description of the Architecture

Analogous to a mission control or base station directing and monitoring a human/robot EVA team, a centralized planning and execution system has been implemented, as illustrated in Figure 3.1. The planner directs the activities of each mobile "agent" based on its goals and feedback from the agents (e.g., locations, available energy, capabilities). The planner also responds to directives issued by the astronaut team members. Such directives range from "I need help" to "I'm doing another task" to "Tell a rover to execute a new task" (e.g., defined by the astronaut). With the centralized planning structure, communication between astronauts and rovers is limited to data associated with the accomplishment of common tasks, with all other messages relayed through the planning/execution system.

Our application is composed of three interacting components: a team planner, a coordination executive, and the mobile agents deployed in the field. The role of each module is outlined below followed in Section 3.3 by a detailed description of the implemented algorithms.

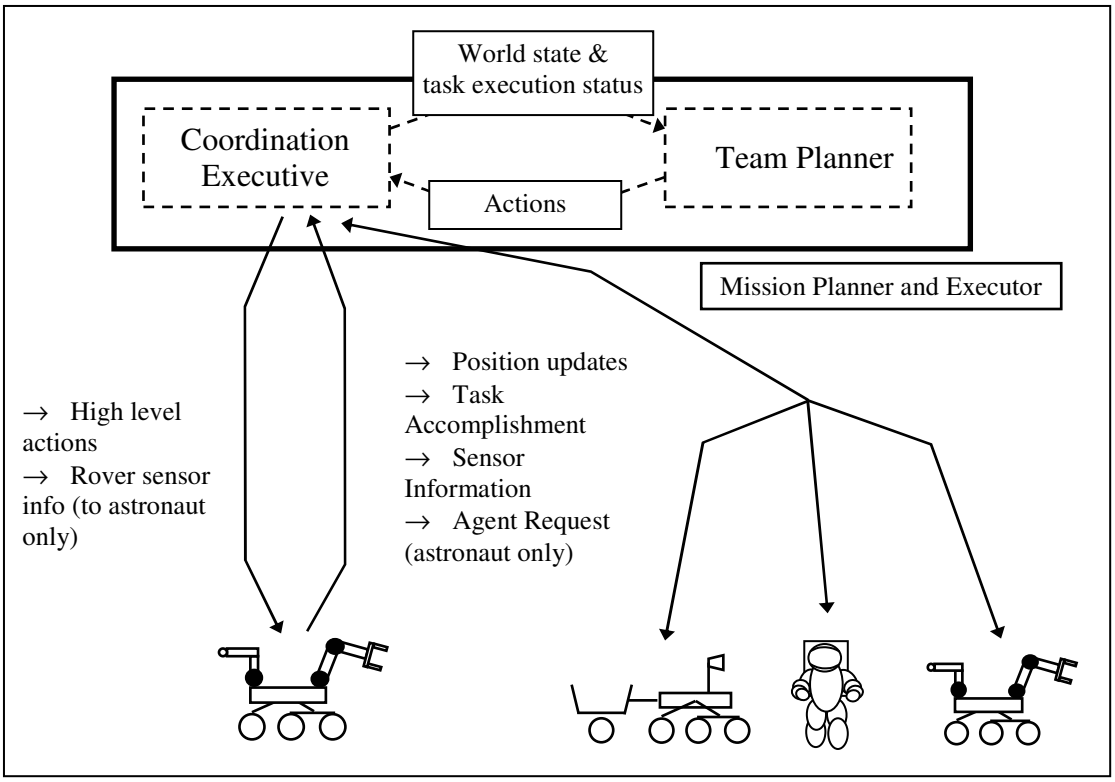


Figure 3.1: Centralized planning and coordination of a multi-agent human-astronaut team.

3.2.1 Team Planner

The centralized planning module takes the current state of the world and computes a *plan*, a scheduled set of high-level tasks or actions, for all mobile agents so that the maximum number of high-priority tasks can be accomplished by the team in a minimum amount of time. The implemented design-to-time algorithm can be configured to react quickly and possibly suboptimally or to consume the additional time needed for plan optimization.

3.2.2 Coordination Executive

The execution module controls the flow of information between the central planner and the distributed set of rovers and astronauts, dispatching actions in real-time in accordance with the nominal policy and any dynamic updates. The execution agent also gathers and processes available state information from the rovers/astronauts to enable detection of off-nominal events. The executive is responsible for managing the reaction to off-nominal occurrences, either by adjusting the policy (e.g., task execution timings) or by notifying the planner that a new/modified plan must be created. For instance, when an astronaut requires help, the execution module will receive the information, detect the event, notify the rovers, and call for the planner to compute an emergency plan as quickly as possible.

3.2.3 Mobile Agents

Each rover and astronaut is either a physical mobile entity or is simulated by a unique process that communicates with the execution module with the same protocol as a real agent would utilize. Rovers are assumed to understand high-level directives (task descriptions) and either to correctly execute them or return an annotated error message indicating the task was not successfully accomplished. The software is configured such that the simple simulated agents can be replaced with “real” robotic and astronaut agents. More details regarding the mobile agent simulation software and the rover constructed as part of this research will be provided in Chapters 4 and 7.

3.3 Mission Task Planner

The mission planner computes a valid plan for the team of rovers and astronauts. Below, we specify a world model for astronaut-rover exploration, then we focus on the planning techniques that were implemented to enable design-to-time planning .

3.3.1 World Model

Based on anticipated planetary surface missions, a set of state features and operators are defined to characterize the task-level goals, agents, and associated resource requirements/availability. Three state variable types were defined and uniquely indexed (i): tasks ($task_i$), rovers ($rover_i$), and astronauts ($astronaut_i$).

Each task ($task_i$) is defined by the attributes listed below in Table 3.1. Locations are discrete instantiated symbols defined by x and y coordinates (e.g., *loc1 is situated at (45.2 , 17.2)).* The tasks represent goals that the rovers and astronauts must complete, after which *done* is set to *true*. Tasks are defined as a certain *type*, implying a required set of competences needed for the task to be accomplished. Preconditions (previous actions) are also provided for situations in which execution of a certain task requires the agent to have already executed another set of tasks. In the current implementation, each task has a constant (approximate) energy cost, representing the level of effort (resource expenditure) expected to accomplish this task. For the rover, this cost would represent power and fuel required, while for the astronaut, it would represent physical exertion level and associated oxygen use. Constant expected duration (execution time) values are also specified for each task.

Table 3.1: Task State Parameters

Attribute	Value
location	pointer to a location object *(x,y)*
type	{ROCK_SAMPLING, IMAGE_CAPTURE, WATER_ANALYSIS, DRILLING, …}
required competences	{STORAGE, MANIPULATOR, ROCK_RECOGNITION, … }
previous actions	Task precedence: list of pointers to tasks that must be completed prior to executing this task
done	{true or false}
energy	$\in \Re$, expected amount of energy required to accomplish the task.
duration	$\in \Re$, expected time needed for the task to be executed.
required_approval	{true or false}

Attributes for the rover state variables are listed in Table 3.2. Similar to the task state variable, each rover has an *(x, y)* location and a set of competences. For the planning problems presented in this research, rovers are presumed to be fully able to navigate between task locations and to typically be successful with the tasks they are equipped to handle. Rovers can also help an astronaut in an emergency (e.g., by providing oxygen or power when the astronaut runs low) or when the astronaut requires a specific tool onboard the rover. The *status* variable indicates the current

state of the rover, including nominal and exceptional situations. IDLE indicates the rover is waiting for its next task, while NOMINAL shows the rover is following its planned set of traversal and task execution actions. DISABLE is set when the rover is unable to continue its operations, and BUDDY/RESCUER is set when the rover is supporting an astronaut. Rovers have a limited amount of onboard energy, a value diminished by each traversal and task execution.

Table 3.2: Rover State Parameters

Attribute	**Value**
location	pointer to a location
set of competences	{STORAGE, MANIPULATOR, ROCK_RECOGNITION, WATER_SENSOR…}
status	{ IDLE or NOMINAL or DISABLE or BUDDY/RESCUER }
energy	$\in \Re$, current level of energy

Astronauts are capable of accomplishing many of the same tasks the rovers can complete, thus they are included as "mobile agent" resources within the planner. However, they also may elect not to execute the planned activities assigned to them. One of the reasons to have humans in the field is to capitalize on their superior

sensing and reasoning abilities. While astronauts are trained to accept instructions, they also will not prefer to be strictly "pawns" of a computerized planning agent. In practice, the astronaut might decide to select a different task based on his preference or observations. In this work, we assume the astronaut will inform the planner of such deviations.

Table 3.3 lists the attributes of the astronaut state variable. As with the rover, the *status* variable indicates the current activity execution state, including a BUDDY/RESCUER status set when an astronaut is assisting another "lead" astronaut with a task or during an emergency. The astronaut state is augmented with two unique flags: *need-help* and *supported*. The first (*need-help*) is set when assistance is requested, prompting the planner to redirect a rover (or other astronaut) as an assistant. The second (*supported*) is set to "*true*" once the assistance arrives. In an emergency, for example, any rover or astronaut that has life support equipment can be tasked as an assistant. For non-emergency task execution, a rover can be requested to provide a specific tool through the same protocol.

Table 3.3: Astronaut State Parameters.

Attribute	Value
location	pointer to a location
set of competences	{STORAGE, MANIPULATOR, ROCK_RECOGNITION, WATER_SENSOR…}
status	{ IDLE or NOMINAL or DISABLE or BUDDY/RESCUER }
need_help	{true / false}
supported	{true / false}
energy	$\in \Re$, current level of energy

Rovers and astronauts can perform four different actions that change world state:

- Navigate between 2 locations (e.g., rover3 traverses from loc1 to loc2).

```
Goto (rover3, loc1, loc2)
```

- Perform a task (e.g., astronaut2 executes task3)

```
Execute (astronaut2, task3)
```

- Rescue an astronaut (e.g., rover5 rescues astronaut2)

```
Rescue (rover5, astronaut2)
```

- Cooperatively perform a task (e.g., rover1 and astronaut2 cooperatively execute task4)

```
Cooperate({rover1,astronaut2}, task4)
```

In order for actions to be applicable a coherent list of preconditions must be true in the current state. Those preconditions not only include intuitive statements such as "task and rover must be at the same location before rover can execute the task", but also some safety checks in order to ensure the security of the astronaut and the consistency of the plan. For instance, a rover that is currently rescuing the astronaut will never be assigned a "perform", a "goto", or a "cooperate" action.

3.3.2 Planning Techniques

For this work, we implemented a recursive, time-controlled best first search algorithm to plan the set of actions. Below, we first describe the best first search algorithm, followed by a description of its application to our problem. Next, we explain how we modified this algorithm to obtain a time/optimality-configurable planning tool via a branching factor control strategy.

Best First Search

Best first search is typically implemented as a forward search. When problems can be represented in a set of states (the state space), forward search techniques can be applied to find a solution which contains a set of action that starts at the initial state and transform the world into a desired state. Each node in the search graph corresponds to a state of the world, each arc corresponds to an action applied to the state, and a plan corresponds to a set of actions, potentially with precedence constraints. Forward search algorithms typically maintain two lists: an *open-list* that contains all the nodes in the graph that have been deemed reachable from the initial

(root) node but have not been expanded, and the *closed-list* that contains all the expanded nodes. Every node in the search graph maintains 3 values:

- $g(n)$ = the cost of getting to the node from the initial state;
- $h(n)$ = the estimate, according to the heuristic function of the cost of getting to the goal from the current node.
- $f(n) = g(n) + h(n)$. This is the estimate of total path cost when traversing to the goal along the path through node n.

The primary difference between the different forward search techniques is the manner in which new nodes are inserted in the open list:

- *Breadth first search*: child nodes are put at the end of the queue
- *Depth first search:* child nodes are put at the front of the queue
- *Uniform cost search*: the queue is sorted in increasing $g(n)$ order.
- *Greedy search*: the queue is sorted in increasing $h(n)$ order.
- *A* search*: the queue is sorted in increasing $f(n)$ order.
- *Best first Search*: Child nodes are grouped at the front of the queue but inserted in increasing $f(n)$ order.

Figure 3.2 illustrates an example search tree along with open and closed lists created with the best first search algorithm.

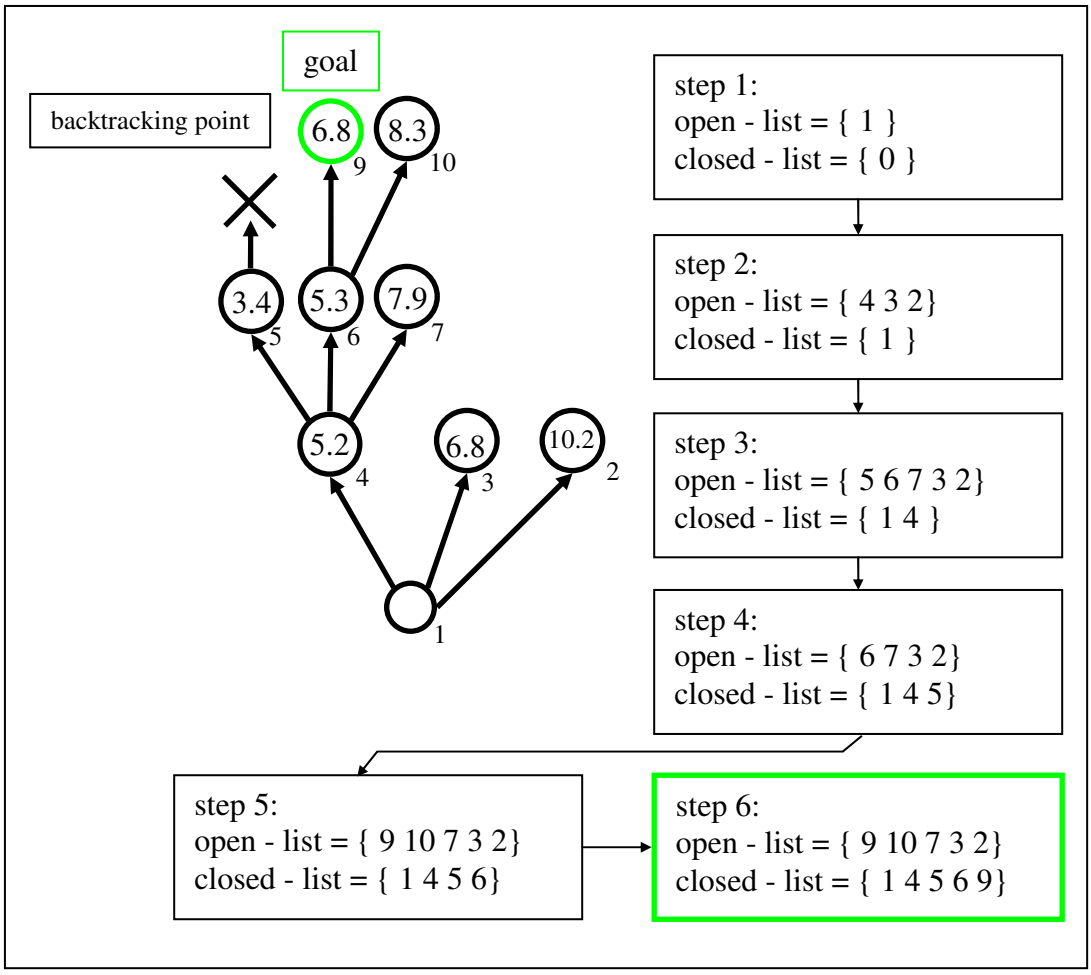


Figure 3.2: Best First Search Illustration

Best first search is usually chosen when a coherent solution must be found quickly. Unlike A*, it is not guaranteed to find an optimal solution, but it typically identifies a solution more quickly due to its bias to explore search tree depth rather than breadth.

Adaptation to Planning for Astronaut - Rover Exploration Teams

One of the challenges defined above was planning actions that execute in parallel (by the different team members). A plan is typically a list of ordered actions with no indication of whether they can be executed in parallel or not. In order to deal with

this issue we added two variables to each action: *start* and *end* time. Start time is the time when the action becomes applicable and end time is when the task is expected to end, deviation from which is dynamically fed back to the executor upon task completion.

Each time a new action is added to the plan, the previous set of planned actions is evaluated to find the appropriate start time. There are two cases:

- *Agent dependency:* For each agent that will be executing this action, this function finds the time at which previously-scheduled actions will be completed. The latest end time over all collaborating agents determines the start time of the new action. For instance, before performing a sampling task the agent must complete the *goto* action that brought the agent to the location of the task.
- *Task dependency:* As defined previously tasks can be ordered to satisfy precedence constraints. For instance, suppose a `rock sample` at `location 2` must be completed before a `water test` at `location 5`. Each time an *Execute* or *cooperate* action is added to the plan, this function will record the time the execution of the previous tasks (if any) ended and make sure the new action starts at or after this last task completion time.

Because this work does not yet model other quantities such as power or fuel, time is the sole measure of cost. The cost function *g(n)* is set to the largest end time over all mobile agent timelines for the set of actions from initial state to node *n*. As described above, the expected time required for task execution is tabulated for "sampling" actions and is a function of traversal distance and expected speed of the agent for "goto" tasks. The heuristic function *h(n)* is not implemented in this work, therefore *f(n) = g(n)* for our planning problems.

Design-to-Time Planner Implementation

Our planning problem is distinct from the classical formulation where the objective is to reach a goal state. Rather than arriving at a desired state the intent is to complete the maximum number of tasks in a minimum amount of time.[1] Indeed if it is possible to accomplish the entire set of tasks then our goal statement can be simplified to the state where all the tasks are done. Unfortunately, in many situations not all the tasks can be completed, particularly with total time or energy constraints. Another case where it might be unfeasible to accomplish some tasks is when no agent possesses the required competences.

Since limited time can be spent doing EVA, plan optimality is a major concern, suggesting offline optimization will be highly useful to plan exploration activities. However, since optimal planning may require substantial (exponential) time, a more

[1] Currently all tasks are presumed equally useful. In the future, tasks could be assigned priority/utility values such that the planner maximizes total task completion value rather than simply number of tasks.

timely (real-time) search tool must also be available for use when needed. Our anytime implementation spawns different instances of best first search and stores results in a partial plan format. When a time limit is reached, the search process is interrupted and the best partial plan is returned. The procedure calls two different functions: *general_search* (see Figure 3.3)*, a* recursive function that performs the search and stores all partial plans, and *get_best_plan,* a function that finds the best partial plan stored. The "best" plan achieves the greatest number of task within a minimum amount of time. For instance, a plan accomplishing 20 tasks will always be preferred over one accomplishing 19 tasks. If two plans succeed in executing the same number of tasks then the plan ending the soonest will be chosen by the planner.

```
general_search (list of nodes : openList, list of nodes : closedList, list of nodes
: partialPlans )
        {
        if ( is_goal_reacheable_from(top_node) = =  true) {
                add top_node to partialPlans;
        }
        top_node = random_selection(openList);
        if ( current_time > time_limit) {
                add top_node to partialPlans;
                return NULL;
        }
        children = expand (top_node);
        if (children = = NULL) add top_node to partialPlans
        else {
                add children to openList ; //based on the f(n) values
        }
        return general_search(openList, closedList, partialPlans);
```

Figure 3.3: General Search Algorithm for the Anytime Planner.

To find different plans in each best first search instantiation, we randomly select the node to expand rather than select the next one in the cost-ordered open list. This strategy is effective despite the loss of cost ordering because all our tasks have equal value in our current implementation, and best-first search priorities high-depth solutions due to its design. The graph shown below illustrates this modified best first search algorithm. This method is analogous to optimization techniques that elect to explore different regions of the input space to avoid local minima.

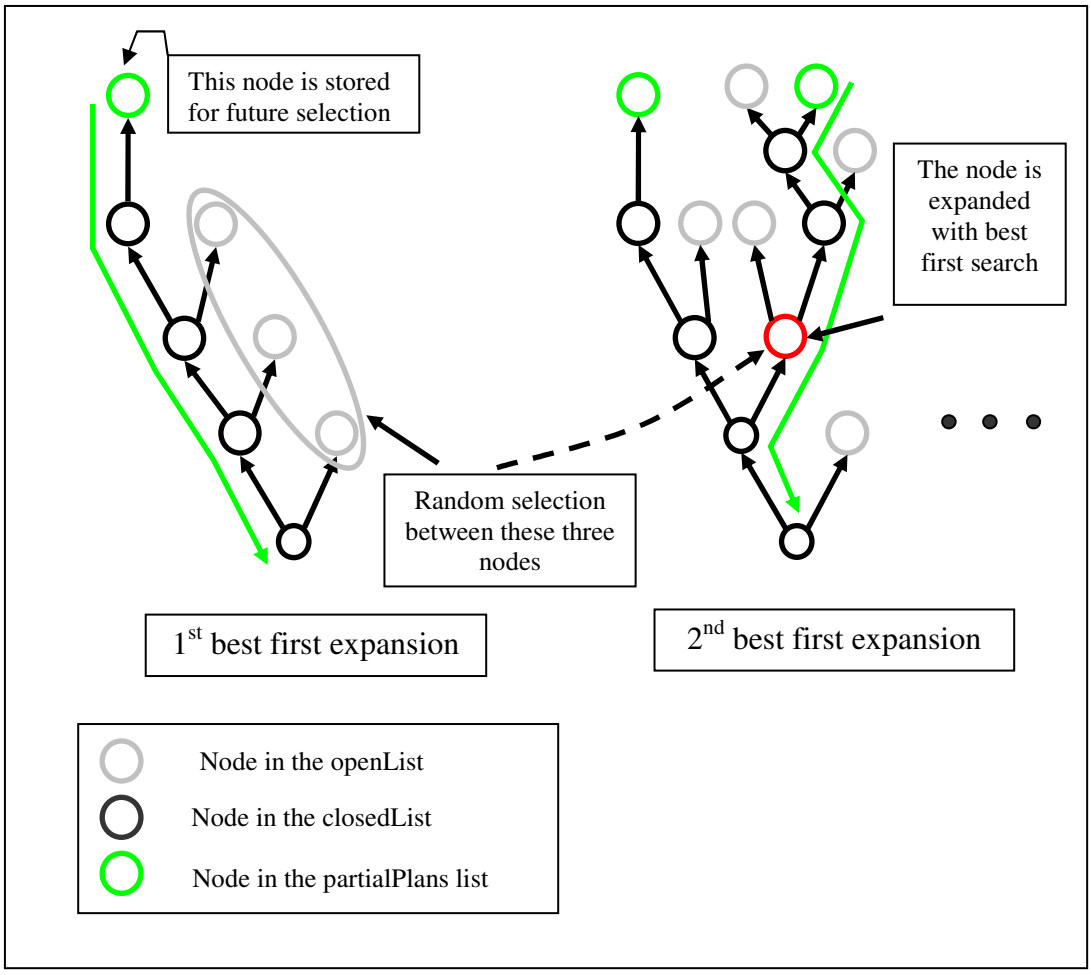


Figure 3.4: Modified Best First Search Algorithm

Branching Control Strategy

The planning *time_limit* value also enables the planner to be tuned appropriately prior to execution, effecting a design-to-time strategy (branching factor control) in concert with the anytime algorithm described above. Managing computation time is a critical issue, particularly when the plan must be adapted in real-time. Controlling the width of the search graph also enables control of optimality versus planning time. The selection among candidate actions for each node is made with the distance-based traversal cost function. For instance, when the branching factor is two, only the two

best (closest) applicable tasks for that agent will be selected. This timeliness feature compromises optimality; however, since fewer partial plans are explored during the search the computation of the plan is much faster. This trade off is inevitable and considered acceptable for our application given the obvious need to rapidly replan, particularly when an astronaut requests emergency assistance.

The setup of the branching factor is tightly coupled to the time allowed to compute the plan. If a long period of time is allocated for the calculation of the plan then it is more advantageous to have a large branching factor. When developing a rescue plan, a small branching factor is better able to limit the search space with our closest-distance heuristic.

The planning process can be interrupted for two reasons. First, the time limit may have been reached so that the planner must return a valid plan immediately. The second case is when all possible nodes have been expanded. This could happen if the branching factor is too small or if the search space has actually been searched exhaustively, yielding an optimal solution. Early termination with reduced branching factor is not desirable because that time could have been used to explore more possible solutions. Although not implemented in this work, this situation could be mitigated by employing an iterative deepening algorithm that quickly provided a solution with a low branching factor but then restarted with progressively higher branching factors as time permits.

3.3 Coordination Executive

Once the centralized planner develops or updates a team activity plan, the *Coordination Executive* must communicate this plan to the mobile agents (human and robotic) and supervise/monitor its execution throughout. Coordination Executive design goals were the following:

- **Transmit action commands to each agent**: Since the planner output is a single list of actions, the coordination executive must dispatch each action to the right agent at the right time.
- **Real-time monitoring**: Task execution must be monitored to facilitate timely task dispatch and to enable coordinated activity synchronization among agents given uncertainty in task completion times. Unexpected events such as emergency calls must also be recognized within a certain amount of time to ensure correct and timely response.
- **Appropriate Decision Making for Unexpected Events:** Once recognized, unexpected events must be handled appropriately either through pre-planned contingency response or real-time plan updates. Once an appropriate response is prepared, it must be sent to the team in a coordinated and timely manner.

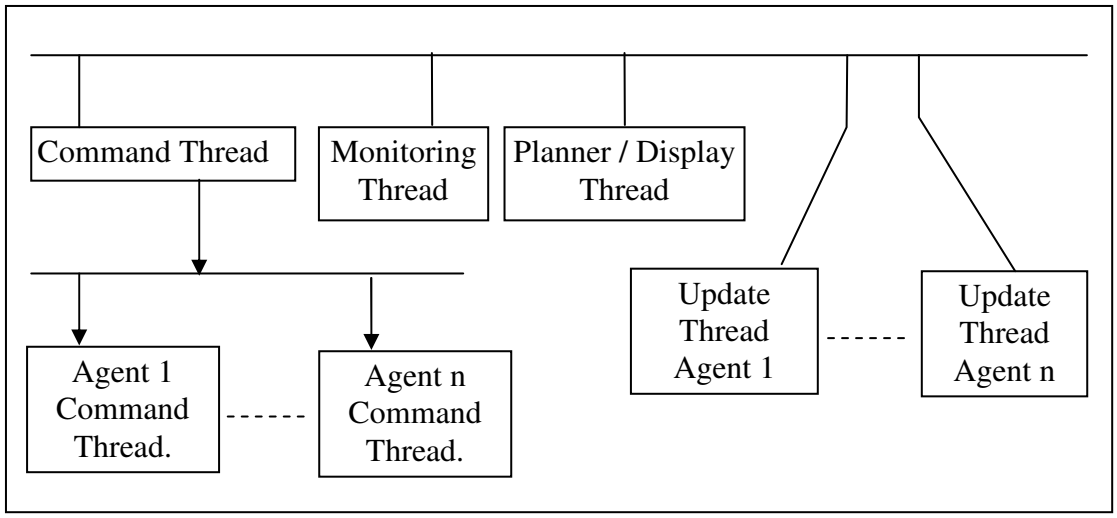


Fig 3.5: Coordination Executive.

The *Coordination Executive* is a multi-threaded program as shown in Figure 3.5. The *Command thread* separates itself into as many sub-threads as there are agents in the field. Each of these threads sends the appropriate command based on the list of actions computed by the planner. Each new command is sent only when the action is applicable, in other words the command is only sent when the preconditions required for action execution are all true. When an unexpected event is detected related threads can be stopped via a flag in the agent definition (see explanation in Section 3.3.1). When a new plan is found the set of command threads are created again, taking into account the new set of actions.

Table 3.4 summarizes the role of each Coordination Executive thread including frequency and shared data. The *Monitoring thread* periodically checks the status of every agent and reports nominal task execution status including completion times as well as detecting unexpected events that compromise the agent's ability to complete

the task or require assistance by one or more other agents. This thread alerts every other thread when an anomaly occurs and a new plan needs to be computed. In the current version this *Monitoring thread* checks for:

- *Agent task execution speed (rate).* Currently, this thread monitors task execution rate but does not halt activities or prompt replanning when significant deviation from expected is noted. In the current implementation, this thread monitors instantaneous speed of the agents, thereby generating false alerts with a real rover that may frequently stop and recompute new paths in cluttered environments. Execution time must be supervised, but in future implementations a more complex check procedure is needed (e.g., speed averaged over a time window or normalized by approximate percentage of the traversal completed relative to expected completion percentage).
- *Agent status.* This thread identifies execution events based on agent feedback data. First the thread checks for assistance requests, such as a call for emergency help or a lower-priority request for a tool or competence currently unavailable at the agent's worksite. Finally it checks for cancellation of a call for assistance, for example when an astronaut no longer requires the requested tool or competence. This thread is used to detect unexpected events; it does not find an appropriate response to them but alerts the system that a new plan must be computed.

The *Planner / Display thread* is in charge of first computing a new plan using the planning tool described earlier and then of displaying execution information at the screen. In this software implementation illustrated in Figure 3.5, the Planner is an "agent" the Coordination Executive can invoke as needed. Upon startup, the Executive first calls the Planner with the specified set of tasks to accomplish. Once the planner returns the initial plan/schedule, the Executor begins directing all mobile agents in accordance with this plan, calling the Planner again as required to prepare contingency responses. In this implementation, the planner never directs or invokes the Coordination Executive except through the Executive's interpretation of the planned task schedules.

The *update Agent i* threads are the communication threads that will be described in more detail below (see Section 4.1.3). They collect data from the mobile agents using socket communications, and update the Executive's knowledge of agent state (position, orientation, energy, etc).

Table 3.4: Coordination Executive Thread Specifities

Thread	**Shared Variables**	**Frequency**
Monitoring Thread	All Agent States	10 Hz
Agent *i* Thread	Agent *i* State	30 Hz
Planner / Display Thread	All Agent States Plan (set of actions)	10 Hz
Command Thread	All Agent States Plan (set of actions)	Not Applicable (aperiodic)
Agent *i* Command Thread	Agent i State Actions involving agent *i*	Not Applicable (aperiodic)

Chapter 4: Multi - Agent Simulation

This chapter presents the astronaut-rover simulation environment designed and implemented to support this research, as well as simulation-only results that demonstrate and validate the planner and executive described in Chapter 3.

4.1 Software Implementation

Simulation results were a necessary first step to test and prove that the architecture as well as the algorithms we had designed were interacting correctly with the agents.

The centralized planner/executive architecture was conceived and coded such that it functions the same either in simulation or in real-life experiments. Consequently the central planner, which contains the planning and execution tool (i.e Figure 3.1), is a single process that connects with the mobile agents only via TCP/IP socket. This multi-process organization promotes interchangeability of hardware and simulation-based mobile agents. Below we first detail how the agents were simulated followed by an overview of the communication protocol between the mobile agents and the central planner/executor.

4.1.1 Rover Navigation Simulation

Each rover is simulated with a single process composed of four different threads:

- *Navigation*: This thread simulates the rover navigating between waypoints.
- *Execute*: This thread simulates the execution of a task.

- *Process Message*: This thread analyses the message received from the central planner. For example, when a message containing a *goto* action is received, this thread processes the specified waypoint information and makes it available for the *Navigate* thread.
- *Send Updates*: This thread sends updates of the rover internal parameters. It enables the central planner to be aware of the status of agent position, remaining energy, velocity, etc.

More thread implementation details are provided below.

Navigation

The movement of the rover is simulated using a straight line between waypoints. In order to simulate the roughness of the terrain, the speed of the rover is randomly perturbed. At randomly-selected times during a traverse, the value of the rover speed is changed using a Gaussian distribution centered around the expected speed. The user can specify the standard deviation in order to simulate a wide range of behaviors.

Task Execution

Like the planner, the rover has a database containing the expected energy and duration of each task it has the competences to perform. In order to simulate the execution of the task, the time and energy requirements for task execution are randomly generated using a Gaussian distribution centered around the expected value. Energy consumption is based on the time required to accomplish the task. If the task execution is longer (or shorter) than expected then the energy consumed to execute

this task will also be larger (or smaller). The relation between energy consumption and execution time is presumed linear. For instance if the astronaut takes 20% more time to execute the task its energy consumption will be 20% greater during task accomplishment.

The task execution result is also randomly generated. In order to simulate execution failure a random number is generated. If it is greater than a user-defined threshold, an error indicating task execution failure is returned to the central planner.

Process Message

This thread processes the raw data received from the coordination executive. The communication protocol (see Section 4.1.3) between agents and the central planner has been implemented. The aim of this message thread is to translate the message received from the central planner into pieces of data that the rest of the process can understand. Currently, the coordination executive can send three types of commands to the rover: *Goto* actions with the desired location, *Execute* with the type of the task and *Stop* with no additional information.

Send Updates

Since one of the requirements of our system was event management in real time, it was mandatory for each rover in the field to send updates at regular time intervals. This update protocol requires the rover to send information about its position (x , y and heading angle), speed, and level of energy. This thread retrieves status

information, computes the appropriate message according to the protocol described below, and sends it to the coordination executive. Table 4.1 summarizes the properties of each thread.

Table 4.1 Rover Thread Summary.

Thread	Variables	Frequency
Navigation	position ; speed ; energy ; goal_position ; status	20 Hz
Perform	task_type ; task_energy ; task_duration ; status	20 Hz
Process Message	status ; goal_position ; task_type	30 Hz
Send Updates	position ; speed ; energy	30 Hz

4.1.2 Astronaut Simulation

The implementation of the astronaut simulation process is similar to the rover simulator. Threads are again created for navigation, task execution, message processing, and data transmission. The primary augmentation for the astronaut was simulation and communication of astronaut-specific (unexpected) events. As a consequence the set of data transmitted to the coordination executive is augmented with the following:

- *require_help:* a command that allows the astronaut to request assistance for task completion or an emergency.

- *type_of_help:* emergency or type of competence that the astronaut requires.
- *disable:* indicates whether or not the astronaut is following the plan or is leaving it to pursue a goal of its own. This command would be significantly expanded with a real astronaut, since the astronaut (unlike the simulator) would typically share the nature of their plan deviation with the coordination executive (mission control) so all would understand their decision and the rationale behind it.

An additional thread was implemented to trigger unexpected behavior from the astronaut. This thread randomly determines the time as well as the type of unexpected situation that arises

4.1.3 Communication between Agents and the Coordination Executive

The coordination executive and the agents are single processes that communicate via TCP/IP. The Coordination Executive manages all communications, with a link to the central planner and to each mobile agent. Each process runs a server and a client. Figure 4.1 illustrates the communication structure.

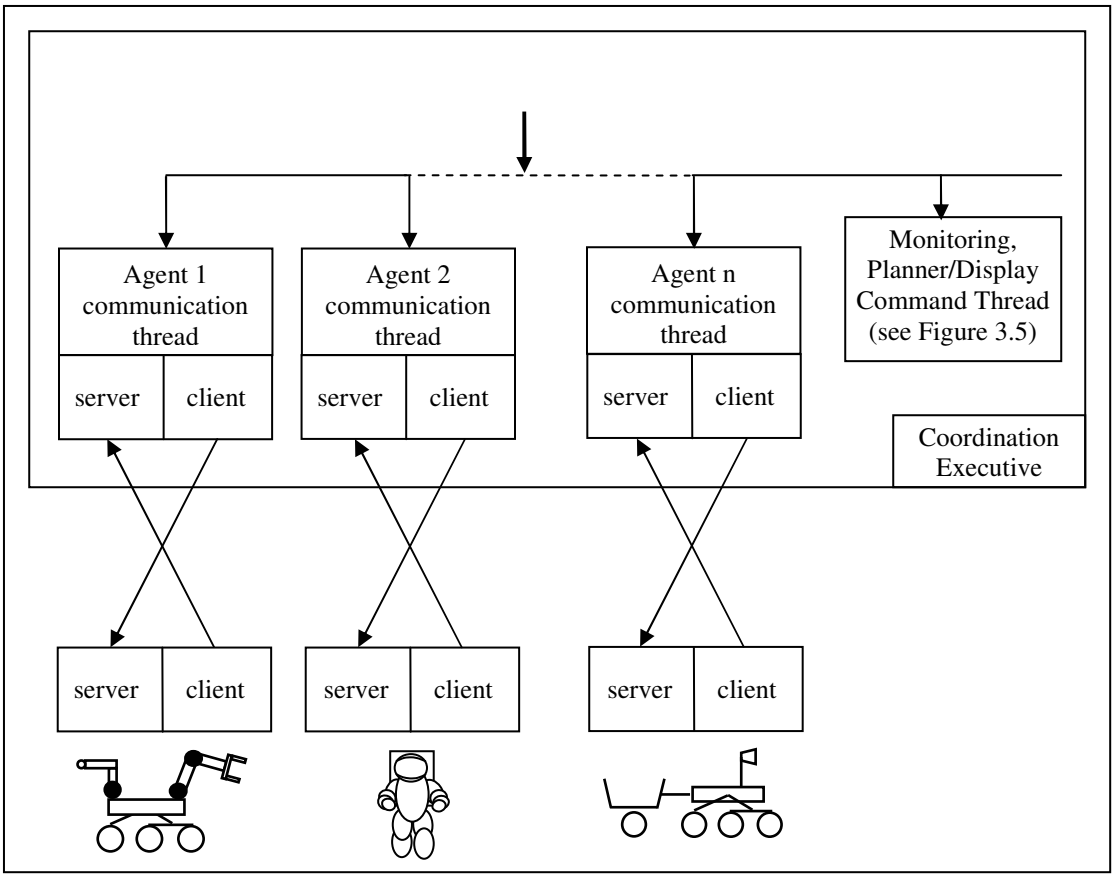


Figure 4.1: Socket Communication Architecture.

Each message is composed of two strings:

- a header that contains information about the data
- the data itself

As illustrated in Figure 4.2, the header is a 10-byte string. The first 6 specify the number of data bytes to follow, the 7th byte is a space character; the 8th and 9th bytes indicates the type of message, while the remaining byte is an "end of line" character. Tables 4.2 and 4.3 summarize the message types and data associated with those messages. Table 4.2 details messages sent from the *Coordination Executive* to each agent. The first action is *Goto* in which the central planner indicates the final location

to reach. The second message is for *Execute Task* which contains the type as well as the identification number of the specific task to execute (see Table 3.1 for task examples). Finally the last message is *Stop* where the *Coordination Executive* indicates the agent should stop executing its current task. Since the *Rescue* action is currently similar to a *Goto* action in its execution (agent reaching a desired location), a GoTo action is actually sent to the rescuer agent.

Table 4.3 details messages sent from each agent to the central planner. The first message is used to transmit regular agent state updates. Position, orientation, speed, and energy state information are all sent to the Coordination Executive. Agents can also send task execution status as well as sensor feedback. Currently only data from the rover's camera can be transmitted, but more types of messages can be easily implemented once the capabilities exist on the physical rover platform(s).

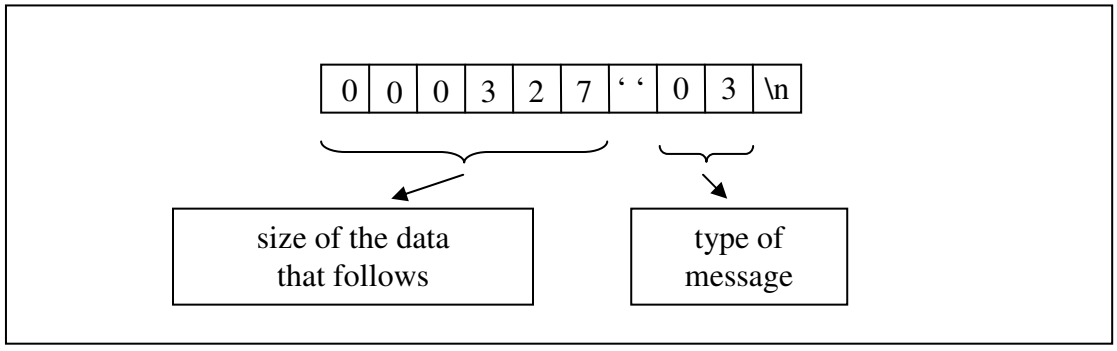


Figure 4.2: Message Header Structure.

Table 4.2: Message Types from Coordination Executive to Mobile Agents

message ID #	**meaning**	**data**
0	Goto action	*x, y* coordinates of the goal position
1	Execute action	task type ; task ID number
3	Stop	∅

Table 4.3: Message Types from Mobile Agents to Coordination Executive

message ID #	**meaning**	**data**
0	System Update Information	x, y coordinates ; heading angle ; speed ; energy + {disable ; require_help ; help_type}*
1	Task completion	task type ; task ID number ; task success/failure status
4	Image	Image Data (pixels)
5*	Astronaut Yes or No answer	1 ⇔ yes 0 ⇔ no
* additional information sent when the agent is an astronaut		

4.2 Simulation Results

A suite of simulations were run to evaluate the performance of our planner/executor system when managing a group of simulated mobile agents. A simple case to illustrate the different outputs of the simulation is presented first, followed by more complex examples that evaluate planner scalability. The tradeoff between timely response and optimality will be analyzed followed by a demonstration of planner/executor behavior when faced with unexpected events.

In the following examples the expected speeds of the agents were set to 3 $m.s^{-1}$ and the expected times of task execution vary from 2 to 7 seconds depending on the task type (*Drilling*, *Rock Sampling* etc…). While not physically realistic, these numbers were chosen to shorten execution time. Since the author had to wait for the real-time simulation to collect data (energy consumption, task execution time, etc…), realistic speeds would have been inappropriate. Nevertheless ratio between task execution and traversal time were respected. During execution, agent speeds were randomly generated using a Gaussian distribution centered on 3 $m.s^{-1}$ and a standard deviation set to 0.75 $m.s^{-1}$. In Chapter 7, where a real rover and astronaut are involved, realistic speeds based on field experience were modeled within the planner's domain knowledge.

4.2.1 Base Cases

4.2.1.1 Scenario 1

Consider the following scenario:

- 6 tasks (1 drilling, 2 image acquisition with approval from the astronaut, 1 chemical test (check for water), 1 image acquisition without approval, 1 rock sampling)
- 10 locations
- 4 agents (1 astronaut and 3 rovers).

Table 4.3 describes the competences of each agent involved in the test, while Table 4.4 describes the possible tasks to be accomplished. Those competences were chosen with realistic knowledge of the capacities of the robotic and human agents. *Rock_Recognition* is a shared competence between human and robots since numerous machine learning algorithms are able to identify region of interest.[15]

Table 4.4 Agent competences for the first scenario

Agent	**Competences**
Astronaut (ID = 0)	{ GRABBING ; ROCK_RECOGNITION}
Rover (ID = 1)	{TAKING_PICTURE , STORAGE , WATER_ANALYSIS}
Rover (ID = 2)	{DRILLING, STORAGE, ROCK_RECOGNITION}

Rover (ID = 3)	{DRILLING , TAKING_PICTURE , WATER_ANALYSIS}

Table 4.5: Task definitions.

Task	**Competences Required**
TAKE_PICTURE	{TAKING_PICTURE}
TAKE_PICTURE_APPROVAL	{TAKING_PICTURE}
ROCK_SAMPLING	{ROCK_RECOGNITION, GRABBING, STORAGE}
WATER_TEST	{ROCK_RECOGNITION, WATER_ANALYSIS}
DRILL_ROCK	{DRILLING}

Each simulation was randomly initialized. The user defines how many locations and tasks are desired, and the software randomly generates location (*x,y* coordinates) and tasks (type and location).

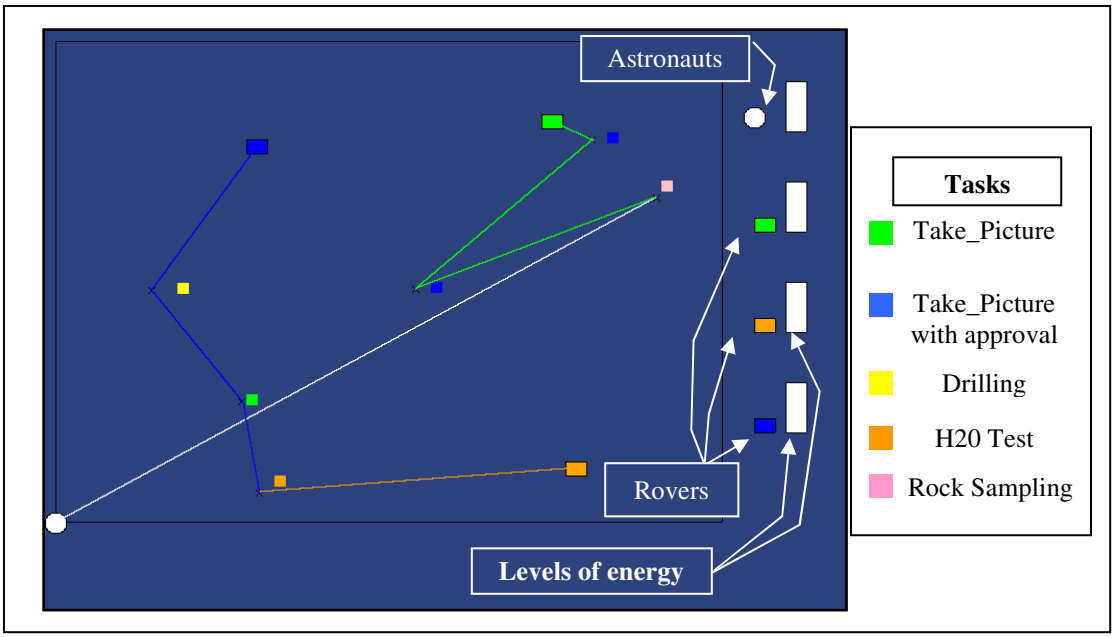


Figure 4.3: Base Case Scenario.

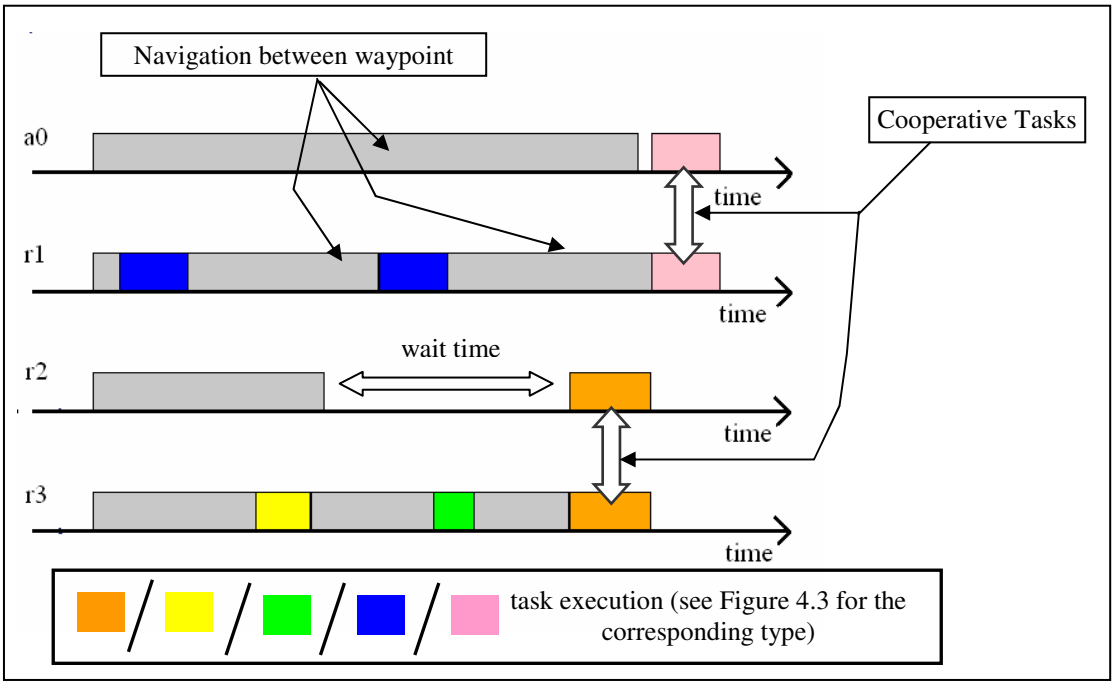


Figure 4.4: Base Case Scenario – Planned schedule and expected energy consumption

Figure 4.3 shows an overhead snapshot of a typical plan. The locations of tasks and agents are indicated, and the computed plan (navigation actions) is spatially represented by traversal lines. Figure 4.4 shows the temporal representation of the plan with tasks inserted along a timeline based on their expected execution times. For this example, the planner's time limit was set to 10 seconds during which 2684 partial plans were stored using a branching factor of 4. The planner successfully finds a plan that accomplishes every task. This test as well as all following cases were all completed on a Dell Laptop with a 1Ghz Pentium 3 processor and 512 Mb of RAM; the operating system was the Fedora Core 3 Linux distribution. Figure 4.5 shows the execution of the plan by the simulated agents. We can see that the execution times are slightly different that the ones expected; this is due to the fact that in simulation task duration is randomly generated with a Gaussian centered on the expected value.

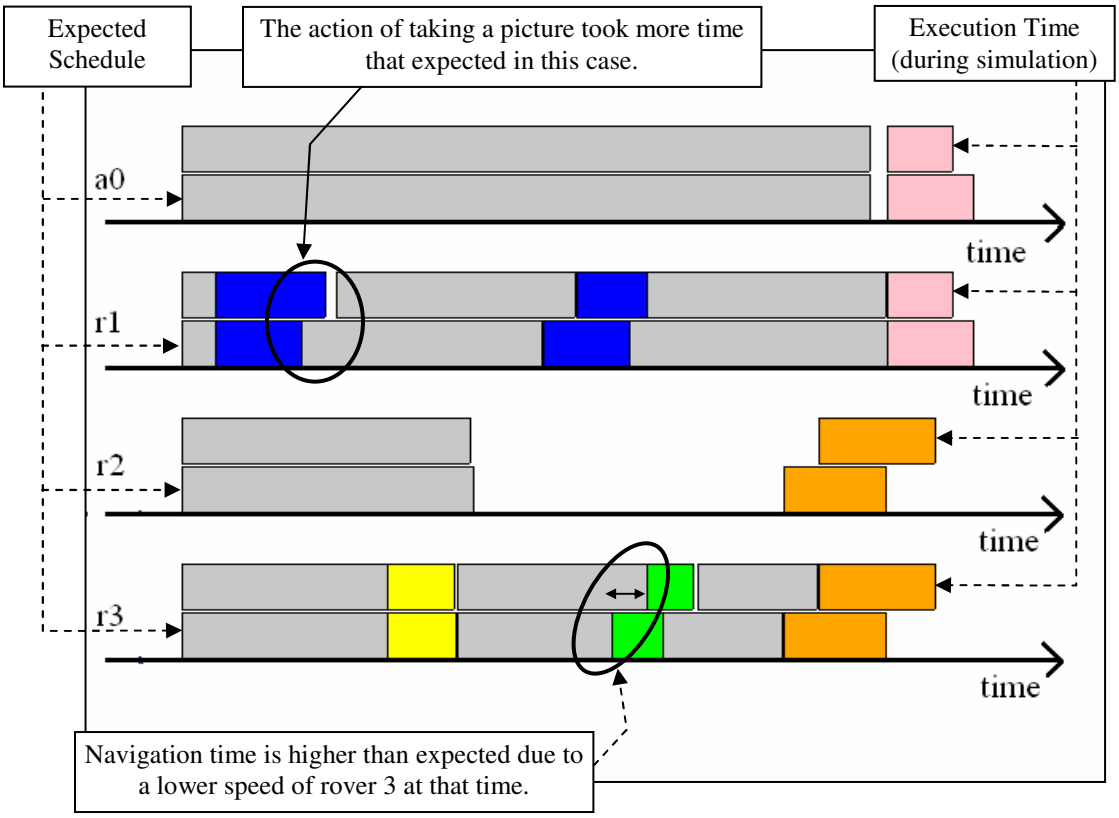


Figure 4.5: Planned and actual execution times for the base case plan.

Figure 4.6 shows the energy consumption. The green curvee is the expected energy consumption based on the task and agent definition. The red curve is the simulated energy consumption received by the central planner during the simulation. We can see that the simulated energy curve utilizations differ slightly from their expected values; again the explanation derives from the fact that each action execution (time and energy consumption) is simulated using a Gaussian distribution centered on expected values. The standard deviation governs how much the simulation data will vary from the expected values.

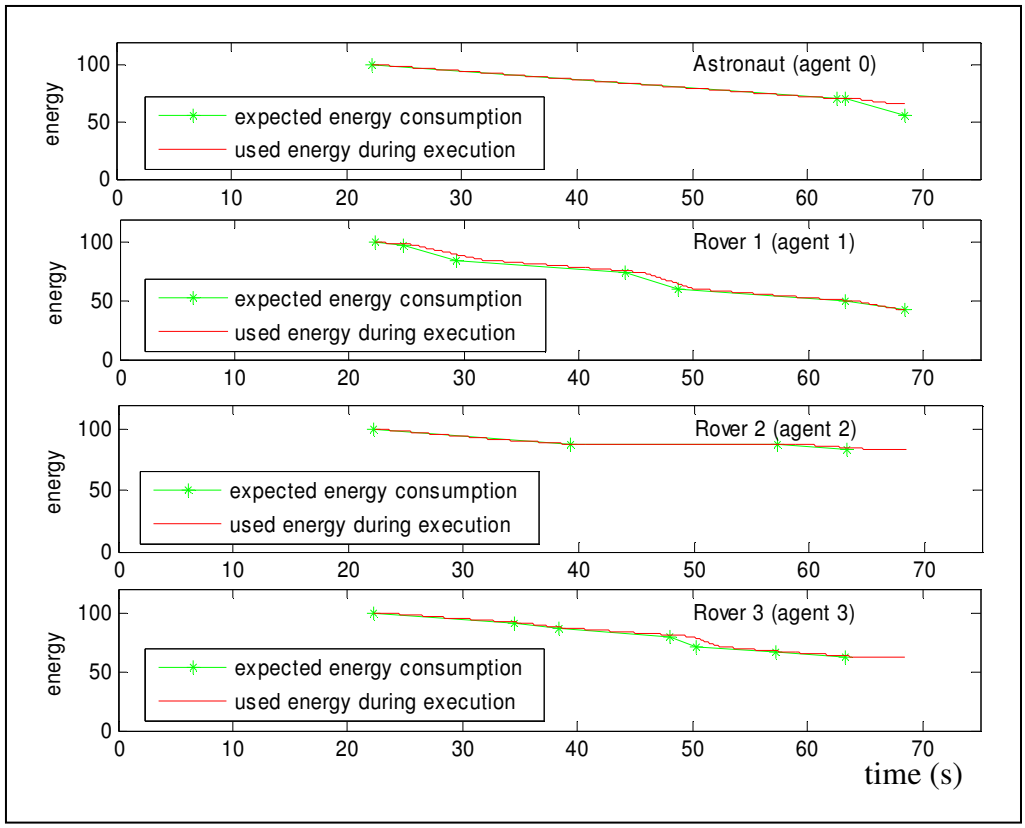


Figure 4.6: Expected and simulated energy consumption.

4.2.1.2 Scenario 2

This second scenario is very simple but illustrates the importance of the branching factor setting as well as the behavior of the planner in a simple situation. Two robotic agents are involved in the scenario and they are both capable of independently performing the 3 *drilling* tasks to be performed at 3 different locations. Two branching factors were tested. First the branching factor was set to 1 which implies that the planner will only perform a classic best-first search without looking at other possible plans. During a second test the branching factor was changed to 2 enlarging

the search space and allowing the analysis of other plans. As shown in Figure 4.7, two of the three tasks are close to the two rovers while the remaining one is far away.

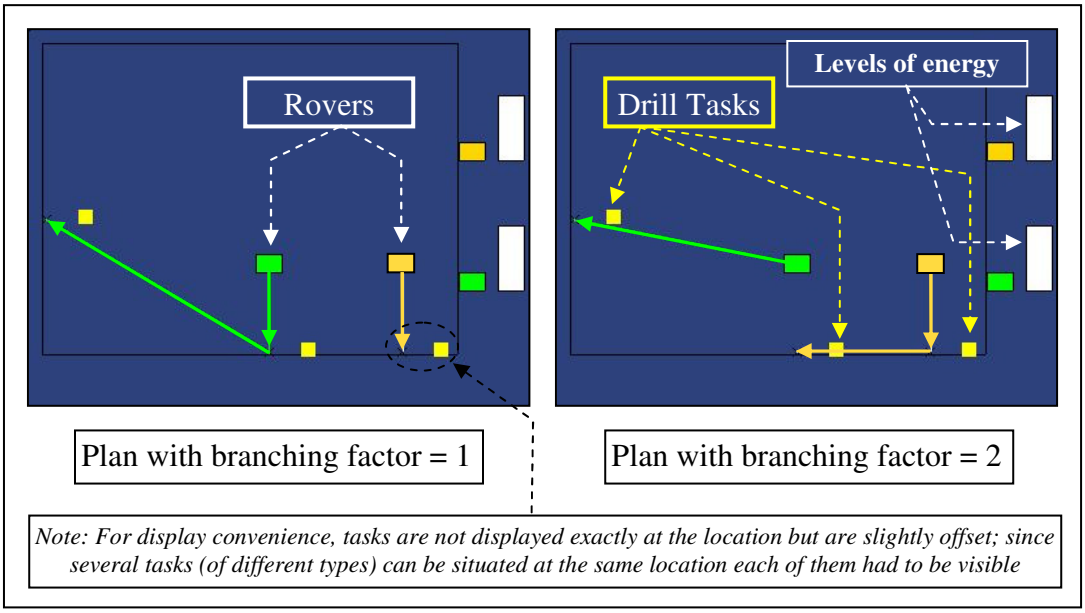


Figure 4.7: Plan computed with different branching factor

This example illustrates the planner search process. With a branching factor of 1, the planner selects the locations that are closest according to the heuristic described in Section 3.3.2. Accordingly the two rovers are sent to the 2 closest tasks while the remaining task is scheduled at the next level and executed by the closest rover given their positions after the first task for each has been completed. We can clearly see that this is not the best option possible; indeed the best plan is the one that takes advantage of the fact that 2 of the 3 tasks are very close and can be executed by the same agent. This optimized solution is found when the branching factor is set to 2, enabling the search algorithm to find the alternate solution that sends one of the agents directly to the most distant task. Figure 4.8 shows the schedules computed for both branching

factors; it is clear that the branching factor of 2 enables a much better solution, minimizing total execution time

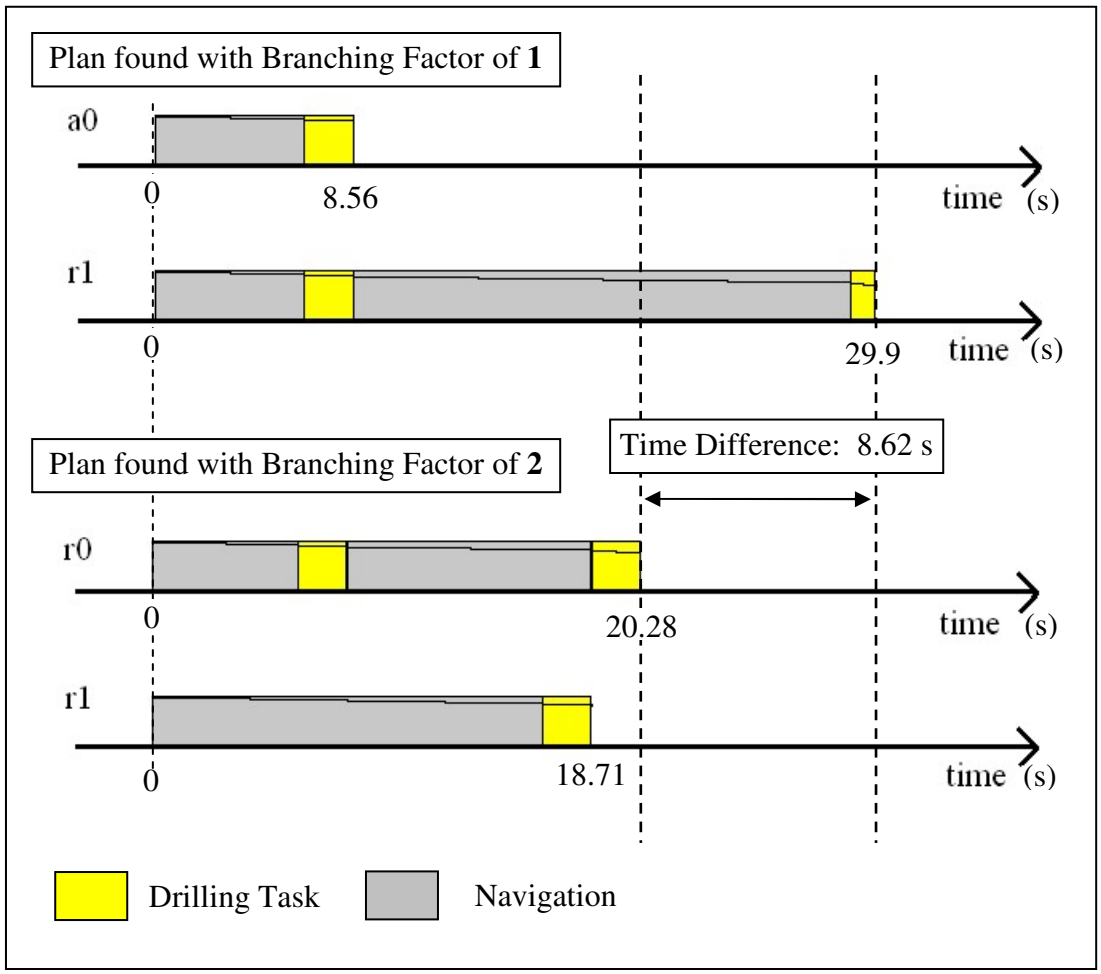


Figure 4.8 a): Schedule computed for two different branching factor.

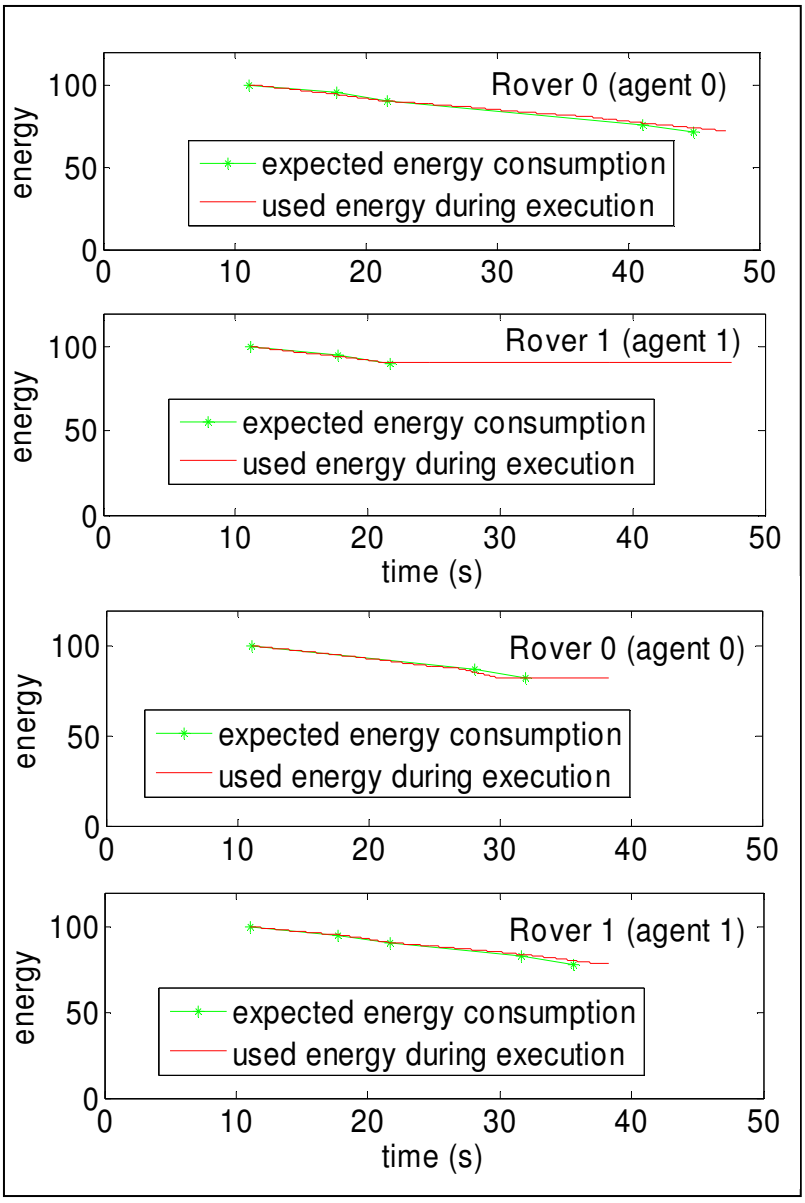
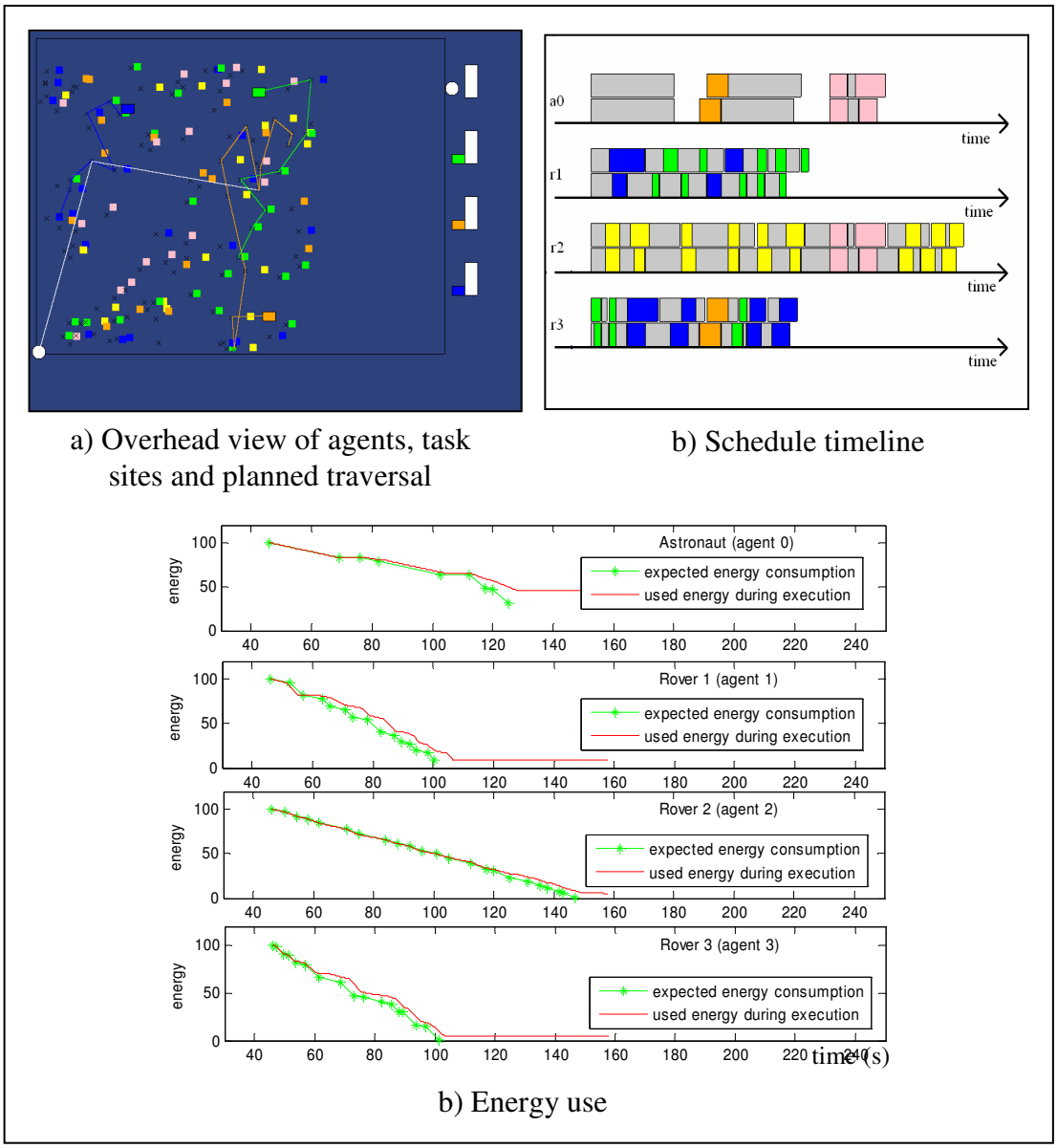


Figure 4.8 b): Expected and simulated energy consumption for two different branching factor.

4.2.2 Expanded Scenario

In the first test, the search-space was relatively simple since only 10 tasks had to be allocated among 4 agents. The purpose of this larger-scale test is to evaluate and validate planner behavior when the search-space becomes sufficiently large to necessitate truncation for real-time planning. This scenario has the same agents and task types as the example 4.2.1.1 but with 100 tasks to be completed. In fact, due to energy constraints not all tasks can be accomplished, nevertheless the goal is to execute a maximum number of tasks. Because of the limited amount of energy only 26 tasks were executed.

Figure 4.9 illustrates the planned traversal and task schedule for each agent after 30 seconds of computation.



a) Overhead view of agents, task sites and planned traversal

b) Schedule timeline

b) Energy use

Figure 4.9: 100-task scenario: Partial plan and schedule after 30 planning seconds.

4.2.3 Time vs Optimality

One of the main goals in our planning implementation was to trade off planning time constraints with solution optimality. By setting the planning *time_limit* value and the branching factor to high values, more plans will be explored thereby identifying a

more optimal solution. To examine the trade off between optimality and computational time we set up an example of intermediate complexity with 24 tasks. Not all of these tasks could be executed due to energy constraints, but depending on the ordering of the task execution, plans with 16 to 20 accomplished tasks could be found. Figure 4.10 shows the results obtained for different computation times. Each point in the graph is an average of 5 experiments and the standard deviation was taken into account when fitting the curve. The inverse of the standard deviation was used to computes weights in the error term as shown in the following equation:

$$error = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left(y_i - f\left(x_i\right) \right)^2 \tag{4.1}$$

where y_i is the value of the ith experimental point, $f(x_i)$ is the fitted value of the ith experimental point, σ_i is the standard deviation of the ith experimental point.

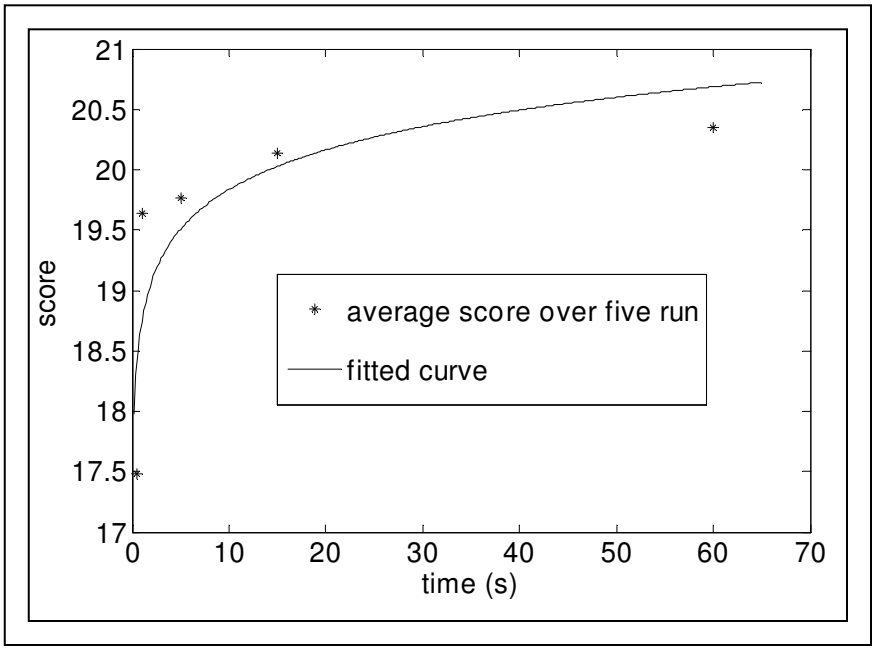


Figure 4.10: Optimality vs Computation Time. for a branching factor of 4.

In order to compare the results of the different runs of the planner for the same problem we created a post-process plan scoring function. This function is designed as a quantitative measure of plan quality, first ranking plans in order of number of tasks completed and secondarily ranking by execution time for plan sets that complete the same number of tasks:

$$score(plan) = \#\_of\_tasks + \frac{(execution\_time)}{upper\_bound - lower\_bound} \quad (4.2)$$

where upper_bound = the maximum amount of time the execution of *i* tasks could require and lower_bound = the minimum amount of time the execution of *i* tasks could require.

As an example, suppose that after running the planner several times we find that the execution of 19 tasks ranges from 120 to 200 seconds then a plan that accomplish 19 tasks in 130 seconds will have a score of:

$$score = 19 + \frac{200 - 130}{80} = 19.875 \quad (4.3)$$

This grading system ensures that a plan with a greater number of tasks will always have a larger score. It also enables us to make the difference between 2 plans that have a same number of task but different execution time. Again, note that this function was only used to compare different solutions identified by the planner, not to guide the search process itself.

Figure 4.10 shows the tradeoff between optimality and computation time for this 24-task example. As expected, solution quality improves with planning time. For instance, 10 seconds are required to plan for 20 actions. The graph approaches an asymptote of (apparently) 21 tasks which clearly indicates the maximum score possible with a branching factor of 4 is not the "perfect" score of 24. This suboptimal situation exists for two reasons:

- Not all nodes are exhaustively searched in the graph with branching factor 4, so the asymptote is not reached even with the higher time limits graphed in Figure 4.9.
- The search graph is limited to a branching factor of 4, which prunes the majority of the full exhaustive search space over all combinations of task orderings and agent assignments.

Although in this problem 20 out of 24 tasks are reliably accomplished even with modest branching factor and execution time, it is still the case that branching factor value has a great influence on plan quality. For the 24-task scenario, planner statistics were collected with branching factors of 2, 4, and 6. Figure 4.11 only shows the curve fits computed from experimental results for branching factors of 2, 4, and 6.

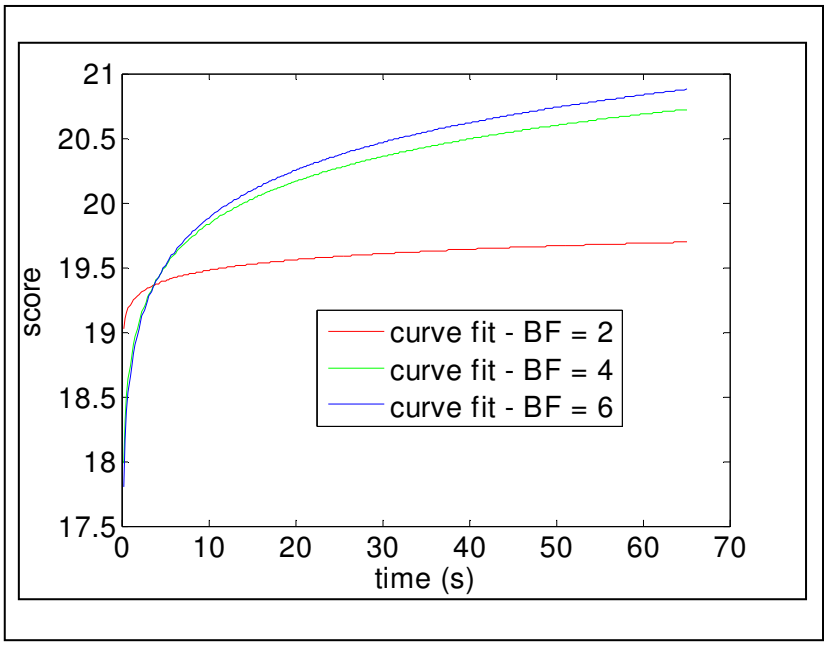


Figure 4.10: Influence of branching factor (B.F.) on plan quality.

The figure above shows two interesting dependencies of plan quality on branching factor. As expected, we observe that when the branching factor is larger, better solutions can be found. A branching factor of 6 yields much better solutions than branching factor 2 solutions and marginally better solutions than possible with branching factor 4 with the same execution time constraints. The second observed property is observed by focusing on the beginning of the graph where planning time is significantly restricted. In this case, small branching factors enable better planning results, illustrating that our distance-based heuristic used to define the reduced search-space is useful. With larger branching factor, the plans explored are more likely to be exploring costly nodes from a distance perspective, even though more nodes are eventually explored with sufficient time (thereby ultimately yielding an equal or higher-quality solution). Since the search is limited to a few plans due to the time

constraint, the solution will be very random. On the contrary when the branching factor is small then even if a few plans are explored they will all be using node close to the heuristic. As the results are better with the latter solution, we can conclude that this strategy is efficient and helps the search especially in emergency situation.

In conclusion, a search tree with large breadth is useful only if we have time to explore it; otherwise it can lead to very random solutions. When computational time is limited pruning the search space with our heuristic improves the obtained solution, although branching factor should be increased to escape local minima with more time. If the branching factor is too small then solution will clearly be less optimal as they could be.

4.2.4 Unexpected Events

In addition to temporal constraints, an important capability of the planning system is to deal with unexpected events resulting from emergencies, opportunities, or simply poorly modeled task execution timings or effects. In this work, focus is placed on emergency response, which is the most critical situation to handle during rover-astronaut exploration activities. Consider the emergency situation where the astronaut calls for help. Although more generally specifics of the emergency would be required to ensure appropriate response, in this work the response to such an event is to redirect an agent with life support equipment to the astronaut, with preference to redirect the closest rover (the first-level response automatically selected by the planner given branching factor of 1). The previously scenario with 24 tasks and 4

mobile agents (three rovers, one astronaut) was utilized. All mobile agents possessed life support equipment. The planner was allowed 30 for the original plan computation and 0.5 seconds to plan the emergency response. This 30s value was chosen based on Figure 4.10 in which we can see that a setting of 30 seconds with a branching factor of 4 gives good results while not being overly long. Additionally tests in Figure 4.10 were made with the same number of tasks and agents, a requirement to make the comparison possible. The 0.5 second response time was arbitrarily chosen by the author but could be dynamically adjusted depending on the level of urgency associated with the particularly emergency that has arisen. At a random time the [simulated] astronaut calls for help. Figure 4.12 illustrates the different events and actions that occur during plan execution, including the emergency response. Figure 4.13 shows the energy consumption during plan execution.

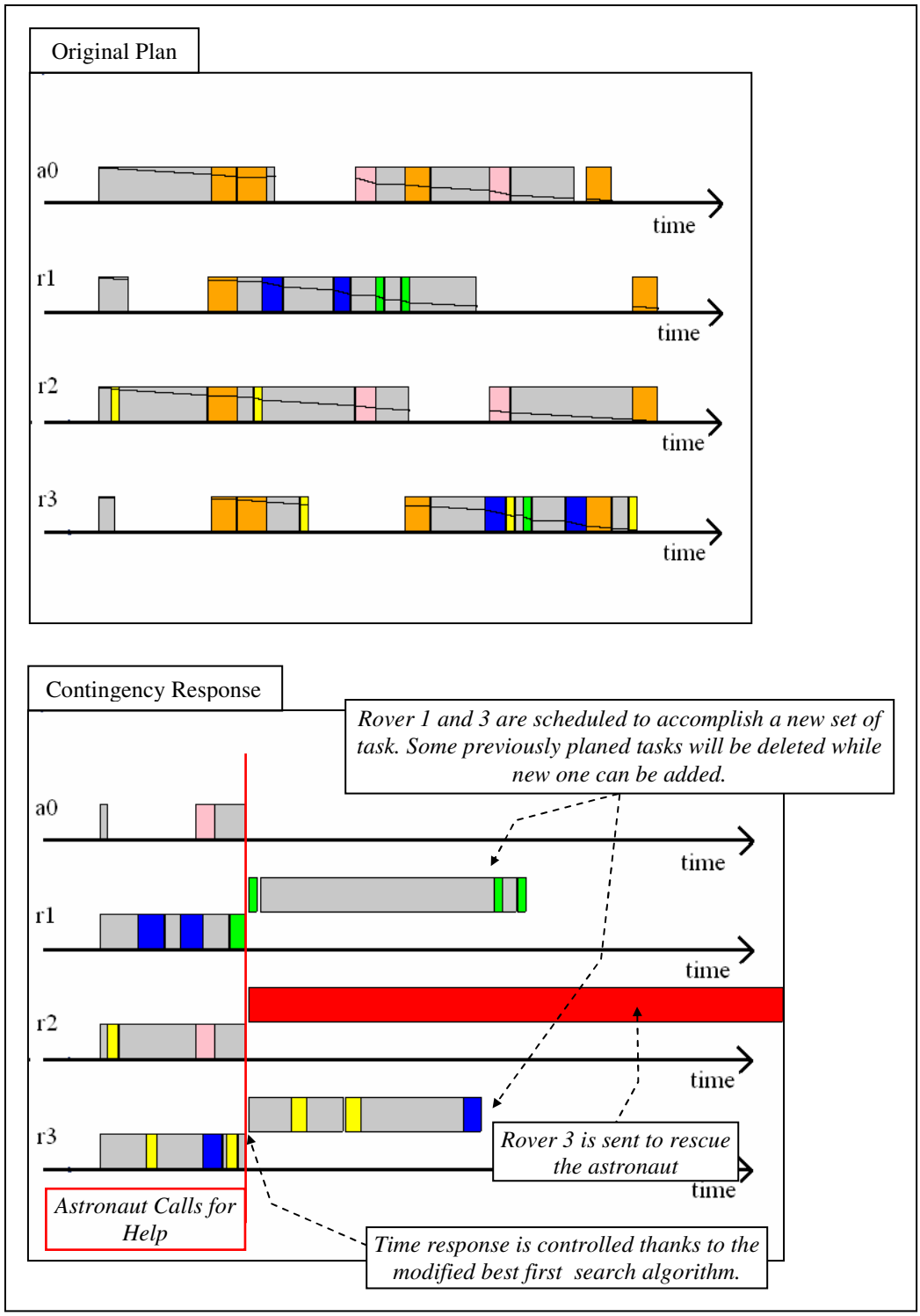


Figure 4.12: Contingency response to astronaut request for emergency assistance.

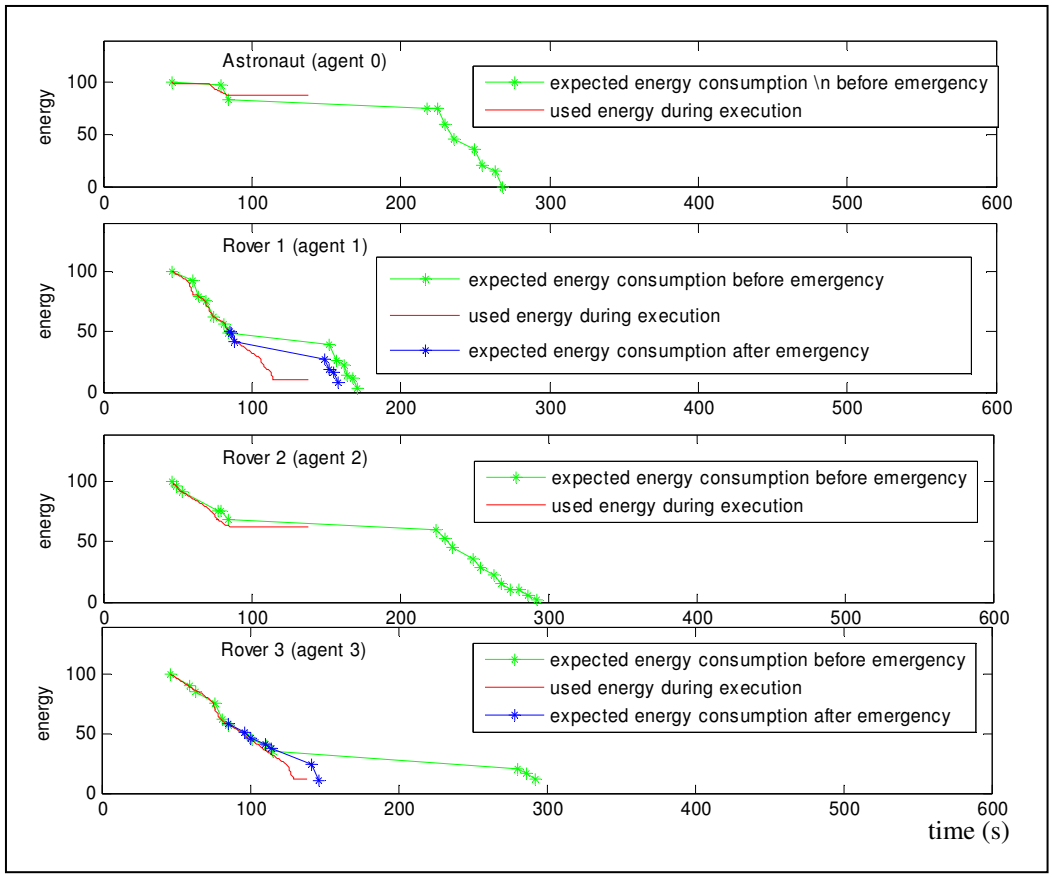


Figure 4.13: Energy consumption during contingency response for emergency assistance.

In the next scenario, we demonstrate the case in which the astronaut decides to deviate from the plan (e.g., to opportunistically explore a scientific target). The computation of a new plan is mandatory, since the pending tasks previously allocated to the astronaut will no longer be completed. A new plan is therefore required in real-time to reallocate the tasks among the remaining team members. Figure 4.14 illustrates such response. The setup of this example is similar to the previous one; the "leave plan" message from the astronaut is randomly generated during execution. Figure 4.13 does not show the initial plan initially computed but only the execution of

it as well as the second plan computed. The recomputation of the plan was constrained to 10s, not as restricted as the time limit for emergency response but still sufficiently small to minimize idle time or time spent pursuing other tasks that will be interrupted once the executor initiates a shift to the new plan.

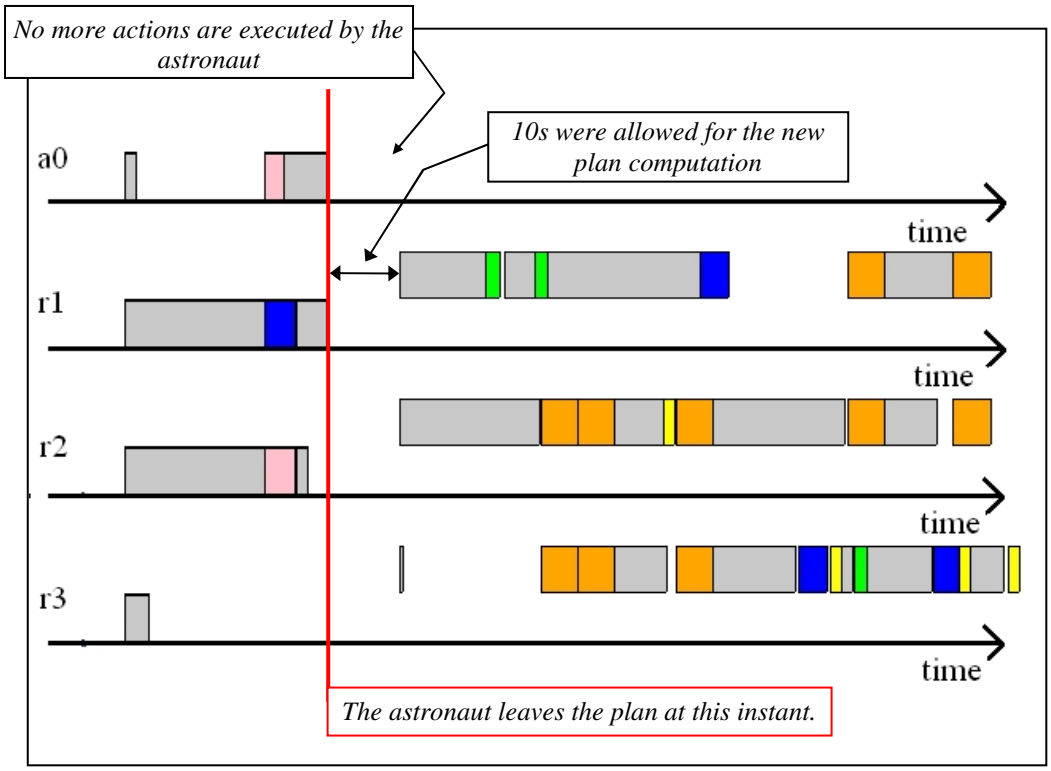


Figure 4.14: Contingency response when the astronaut leaves the plan - the remaining rovers accomplish 17 tasks out of 24.

Chapter 5: Rover System Design

This chapter describes the rover that was constructed, programmed, and integrated as a mobile agent for this research. A description of the rover's mechanical, sensor, and computer/software systems is provided below.

5.1 Mechanical Systems[2]

The rover is based on a six-wheel rocker bogie suspension system first developed and implemented at the Jet Propulsion Lab for the Mars rover Sojourner. The sides and the rear part of the body are connected to the suspension system with pivots which enable the chassis to stay at half the angle between the two sides, improving stability. The wheels are evenly spaced, separated by approximately twice their diameter. This enables the rover to traverse obstacles of approximately 1.5 times the wheel diameter. Figures 5.2 – 5.6 illustrate the primary mechanical subsystems of the rover.

---

[2] As a second-generation rover in the Space Systems Lab, the rover frame and mechanisms (except the pan-tilt unit) were designed and machined by Vegard Hamso, a mechanical engineering undergraduate in the University of Maryland's Space Systems Lab. The author of this thesis received the rover components but was fully responsible for assembly and test of the "basic" mechanical rover system under remote control. This thesis serves as the only documentation to-date of the rover hardware.

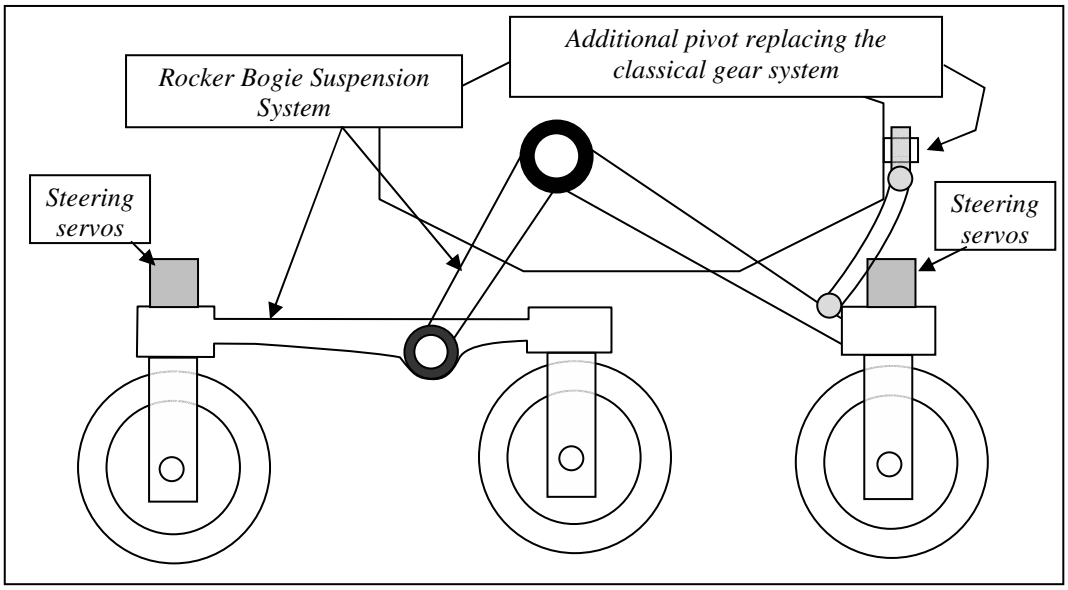


Figure 5.1a: Rover suspension and steering design. (Side View)

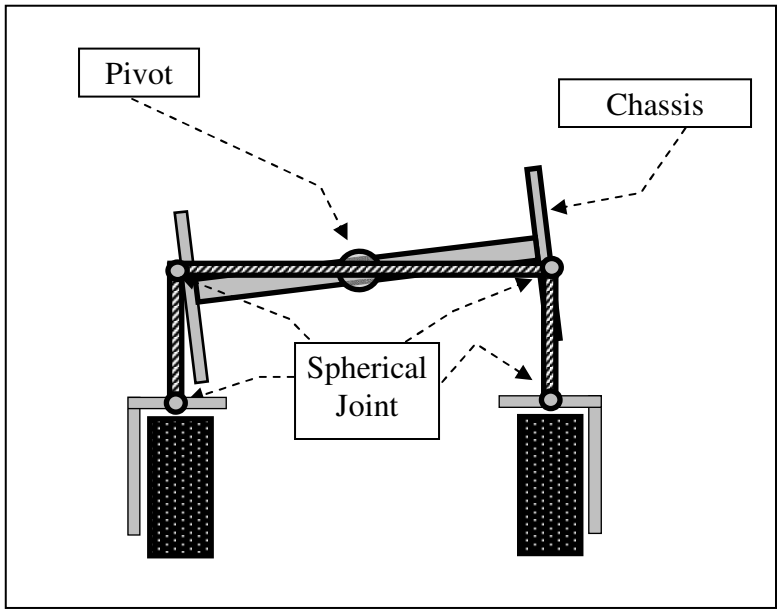


Figure 5.1b: Rover suspension and steering design. (Rear view)

We can see from the above figures that the rover is composed of a chassis (used to support the onboard electronics, batteries, etc.) and a suspension system composed of

two symetrical assemblies (one for each side). The chassis is linked to the suspension system via three pivot joints that are sufficient to stabilize it. Each suspension system is composed of two parts: one bar linkage holding the front and center wheel and one "V" component linking the front assembly to the chassis and the rear wheel. Most parts were fabricated with a Computer Numerically Controlled (CNC) milling machine due to their complex shapes.

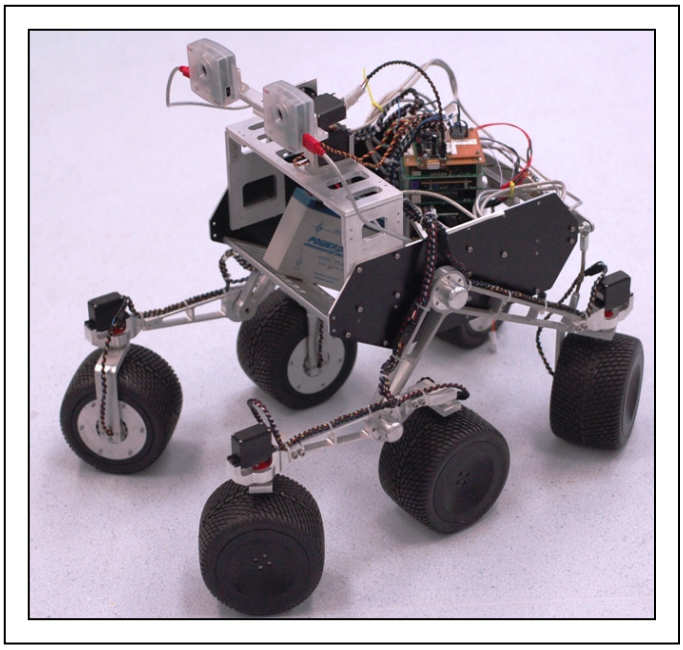


Figure 5.2: Assembled view of the rover.

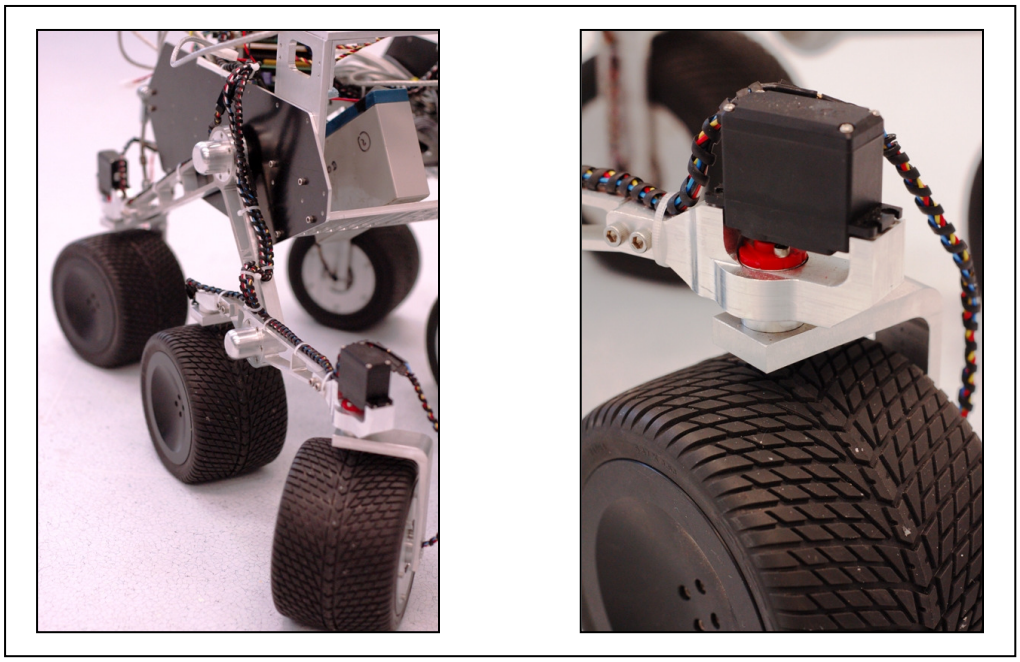


Figure 5.3: Rocker Bogie suspension system and steering system details.

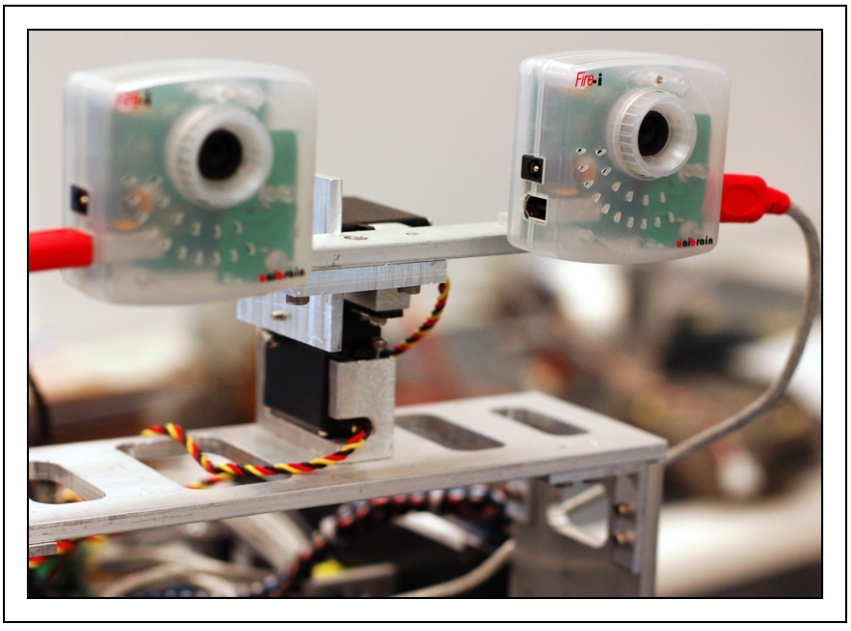


Figure 5.4: Camera Pair mounted on a pan tilt unit.

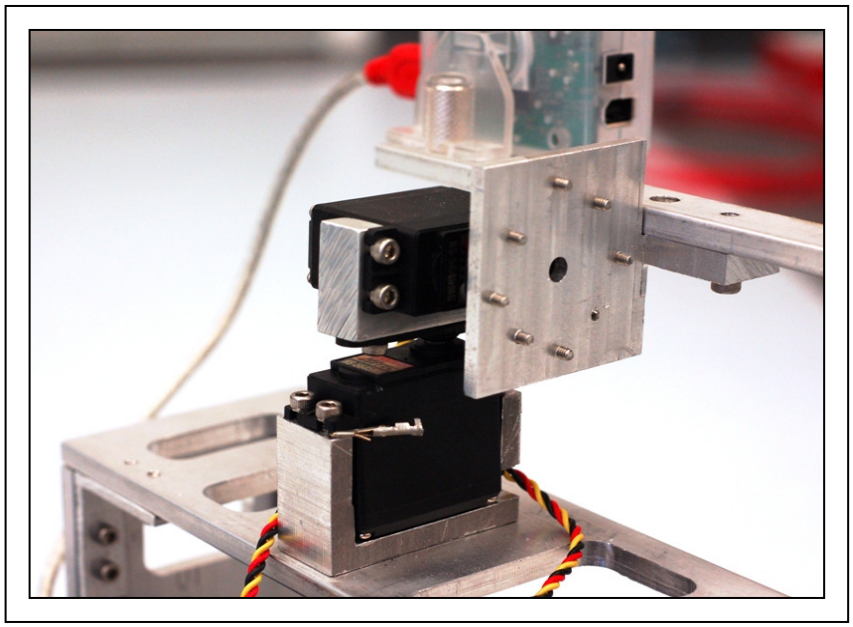


Figure 5.5a) Details of the pan - tilt unit. (picture)

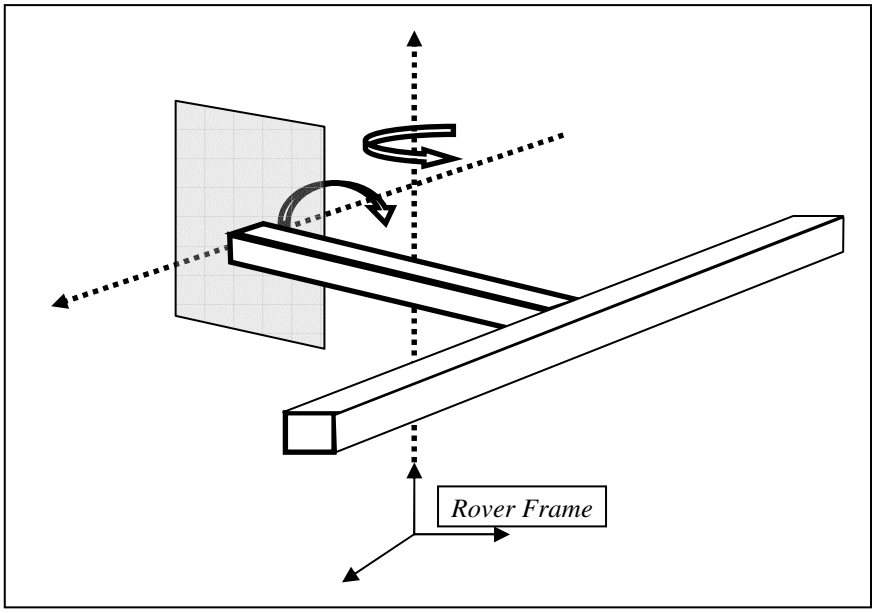


Figure 5.5b) Details of the pan - tilt unit. (graphic)

As illustrated in Figure 5.5, the pan-tilt unit has three principal components:

- *Primary (pan) servo holder.* This "U shape" aluminum part is linked to the rover frame and is used to hold the pan servo.
- *Secondary (tilt) servo holder.* Similar to the first holder, this part that holds the tilt servo has a "U shape" and is attached to the rotational axis of the pan servo.
- *Camera holder (mount).* This part is attached to the tilt servo and holds the pair of camera.

The pan tilt unit was machined using the manual milling machine since part shapes are quite simple.

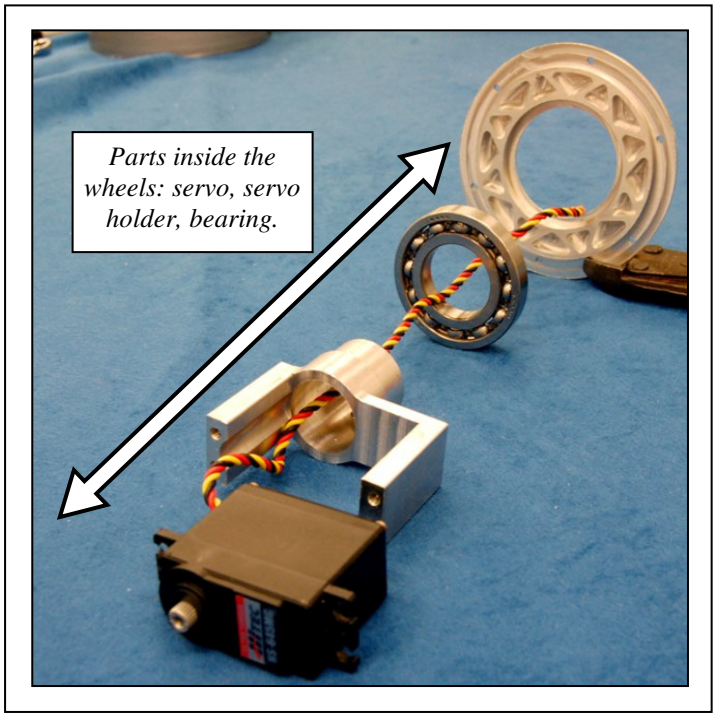


Figure 5.6: Mechanical parts inside each wheel.

5.2 Sensors and Servos

Autonomous navigation without GPS was a baseline design requirement for our rover. This requires the rover to sense its position and orientation (heading for traversing a near-flat surface) and to accurately command its servos to follow specified traversal directives. Below, we first describe the servo interface, followed by a description of modifications to the COTS (commercial off-the-shelf) servos to enable extraction of an encoder signal for dead reckoning navigation. Next the Microstrain Inertial Measurement Unit (IMU) and stereo camera pair will be described.

5.2.1 Servo - Serial Interface

Based on previous experience, the rover was designed to be driven by 12 COTS Hitec HS-645MG high-torque servos traditionally used by hobbyists. The main advantage of such servos is their low cost and availability. When driven by a servo control board (PONTECH SV203 Servo Motor Controller), message passing between computer and servos is handled via a traditional RS/232 serial communication protocol. Figure 5.7 illustrates the connection between the different servos and shows an example of a data sent through the serial port.

In the rover a total of 12 servos were used to a) steer the vehicle by changing the orientation of the 4 corner wheels (4 servos), b) pan and tilt the stereo camera pair (2 servos), and c) drive the vehicle (servos were positioned inside each wheel as described above) (6 servos). Two servo controller boards were necessary as each can handle a maximum of 8 servos.

Originally the COTS servos were not capable of continuous rotation but are designed with hard stops to reach discrete positions within a range of approximately 180

degrees. In order to use servo for wheel rotation the following modifications were required:

- Disconnect the potentiometer from the rotation axis.
- Cut the security block (hard stop) to allow full rotation.

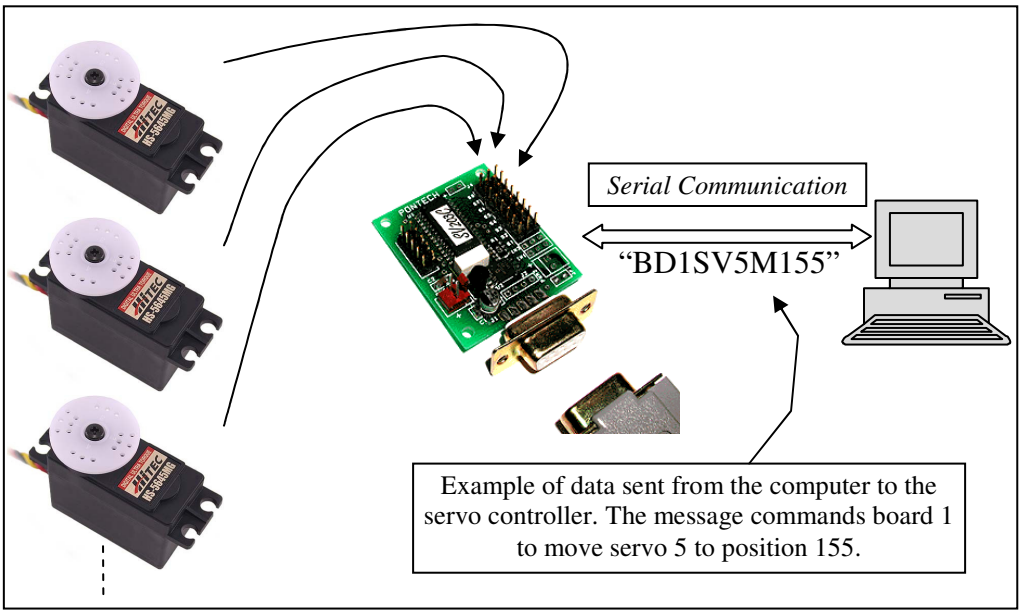


Figure 5.7 Connection between servos, control boards, and computer.

Despite their numerous advantages, servos are not the ideal drive system especially when modified for full rotation. First, when testing the servos after the full rotation modification we realized that only ten speeds were available in each direction. The control resolution is therefore very low, an artifact managed by the controller to be described in the next chapter. Also, although serial communication is very easy to setup and use, it remains very slow. We are currently using the servo board at a rate of 9600 baud which in practice limits the control loop to approximately 10Hz given message dropout and other processing and data I/O requirements.

A third drawback, which we will see in the following section, is the difficulty to built high resolution encoders within these servos, given that no external encoders were integrated both to reduce cost and fit all components within the constraints of the small rover wheel cavity. Based on experience, it is the recommendation of this researcher that the encoder/servo system was the weakest link in the hardware – upgrades will likely provide better navigation results. Accuracy deterioration in control as well as in position determination is in large part from this choice of rover actuation and encoder system.

5.2.2 Custom Encoders[3]

In order to estimate the position of the rover, odometry sensors are required to measure wheel rotation speed. We used custom encoders manually embedded within the COTS servos. The idea was to measure rotation of the first gear (the gear connected to the motor of the servo) to get better resolution. The encoder system is based on the P5587 photoreflector from Hamamatsu. This reflector embeds an infrared LED (for light emission) and a photoIC (for light reception) which includes a photodiode, an amplifier, a Schmidt trigger circuit, and an output phototransistor. The figure below shows a diagram taken from the P5587 manual that details the output voltage based on the surface that is in front of the sensor.

---

[3] The custom encoders described in this section were designed and primarily constructed by Joseph Easley, an undergraduate student in the University of Maryland Space Systems Lab.

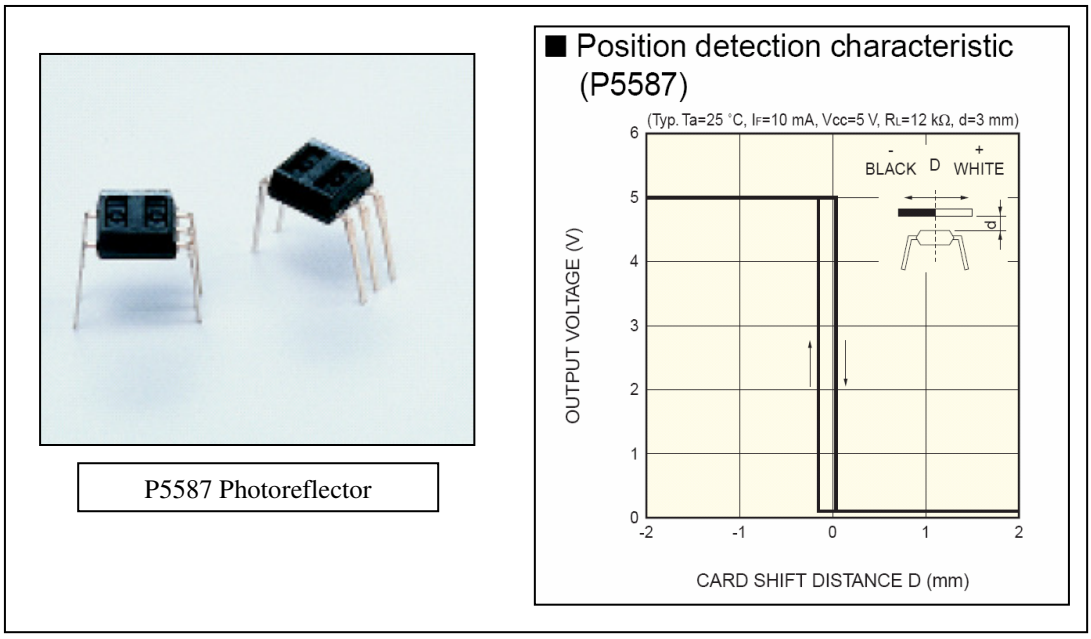


Figure 5.8: Position Detection with the photoreflector P5587 from Hamamatsu.

This photoreflector has to be integrated in a circuit that the vendor provides. The next step was to design a circuit card small enough to fit inside the encoder. Figure 5.9 shows the circuit for the photoreflector as well as the board design.

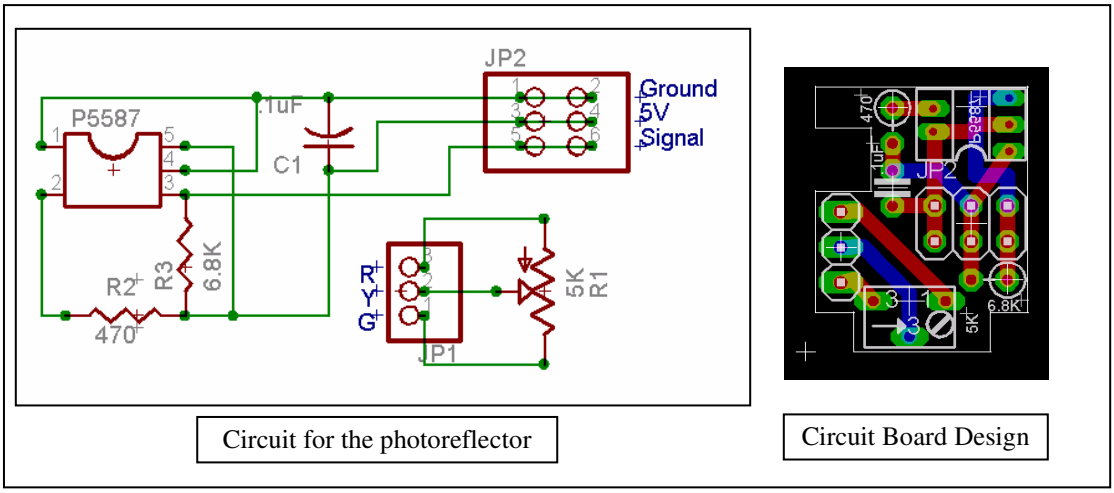


Figure 5.9 Circuit and circuit board for the photoreflector (J. Easley).

The circuit board is installed inside the servo housing, enabling the photoreflector to face the first gear. After many tries, the best solution was determined to be gluing a piece of paper to the servo with the following design: 

This enables us to have 3 periods (and therefore 3 counts) per gear revolution. As the gearing ratio is equal to 50, we could obtain 150 counts per wheel revolution.

In order to determine the equivalent distance of one count we decided to make the rover drive a known distance while counting the encoder values. We performed this test repeatedly and selected the average value of 1.866 mm per encoder count.

5.2.3 Microstrain Inertial Measurement Unit

Although this work operates the rover only on near-level surfaces, position computation requires heading knowledge. Our solution was to use the 3DM-GX1™ Inertial Measurement Unit (IMU) manufactured by Microstrain®, chosen due to its ease of integration and availability from another project. This IMU combines three angular rate gyros with three orthogonal DC accelerometers, three orthogonal magnetometers, a multiplexer, 16 bit A/D converter, and embedded microcontroller. It outputs full three-dimensional orientation and angular velocity vectors in dynamic and static environments. Figure 5.10 shows a picture of the 3DM-GX1™ IMU.

Figure 5.10: Microstrain® 3DM-GX1™ IMU.

The inside micro controller computes in real time the orientation of the unit using a filter algorithm that fuses the 9 sensor measurements with an Extended Kalman Filter (EKF). The IMU data is accessed via RS/232 serial port at 115 kilobits per second. Available data includes:

- Raw sensor information: (integer value where 0 represents 0v and 65 535 represent 5 volts)
- Sensor information: raw data scaled to physical units.
- Stabilized sensor information: filtered data values.
- M: rotation matrix from the earth fixed coordinate system to the IMU coordinate system.
- stabM: Similar to M but with stabilized sensor information.
- Q: quaternion vector of the rotation from the earth fixed coordinate system to the IMU coordinate system.
- stabQ: Similar to Q but with stabilized sensor information.

- Euler: Euler angle of the rotation from the earth fixed coordinate system to the IMU coordinate system.
- stabEuler: Similar to Euler but with stabilized sensor information.
- Temp: Temperature.

Microstrain® provides a C API to interface with the IMU.

5.2.4 Cameras - FireWire Interface

Our target recognition and obstacle detection system is based on stereo image information. The Fire-i™ Digital Camera from Unibrain® was selected for three reasons:

- The Firewire connection is fast and extensible in the future to more cameras.
- This camera has decent optics but low cost.
- This camera had been proven to work under the Linux environment we planned to use, thus integration overhead was minimal.

The figure below shows a picture of the Fire-i™ Digital Camera as well as its specification. Linux image acquisition drivers were downloaded from the web (http://www.linux1394.org/) and integrated into the open source machine vision library we used for stereo image processing (see Section 6.1).

sensor type	SONY® Wfine* ¼" CCD
Pixel shape	square: 5.6 x 5.6 µm
Optics	f=4.65
Horizontal View Angle	42°
Vertical View Angle	32°
Resolution & speed	640 x 480 in RGB mode 15 frames per sec
Power	0.4 watt (sleep mode) 0.9 watt transmitting

Figure 5.11: Fire-i™ Digital Camera Specification

5.3 Computer System

From the perspective of this research, the heart of the rover is its computer system. As would be expected in our overall architecture, the rover acted as a standalone "mobile agent" which required that all computations required for task execution are made onboard. Because algorithms required for image-based obstacle detection and path planning required an embedded processor with decent computational abilities, our computer stack was designed around the PC 104 plus standard form factor. Below we briefly describe the PC 104 plus architecture followed by a description of the different CPU and interface boards and their support software.

5.3.1 PC - 104 plus and the Rover CPU/Interface Boards

PC 104 plus is an embedded computer standard by the PC/104 Consortium that defines both a form factor and a computer bus. Each module has the same dimensions and stack on other boards. The PC 104 plus board combines 2 buses: ISA and PCI.

The first board in the stack is the motherboard or CPU board. We selected the Cool RoadRunner III™ from Lippert® due to a combination of speed, power, and cost considerations.

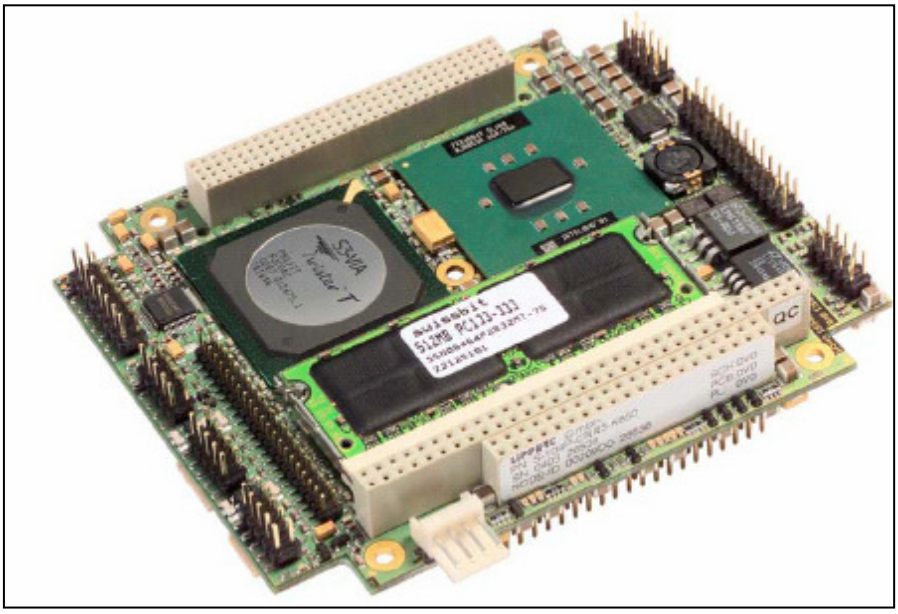


Figure 5.12: CPU board - Cool RoadRunner III™

Shown in Figure 5.12, this board features:

- 933 MHz Processor
- 512 Mbyte of RAM
- Graphic Card Savage4 AGP

- 2 RS232 serial Port
- 2 USB 1.1
- 10/100 Ethernet connection
- AC97 sound card
- Phoenix BIOS

We added a 2 GB compact flash card for onboard operating system and file storage.

To retrieve information from the stereo camera pair we needed a firewire interface in the stack. The card we used is the Ampro MiniModule 1394™. It provides two 400Mbps firewire ports and is illustrated in Figure 5.13.

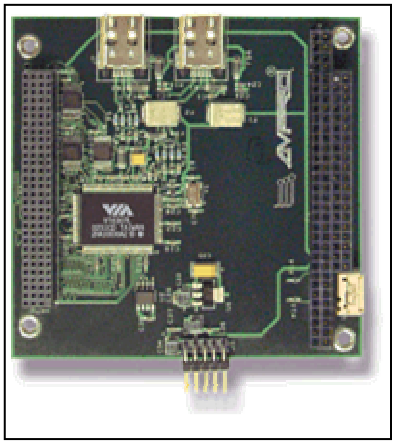


Figure 5.13: MiniModule 1394™ from Ampro - FireWire Adapter Board.

To measure wheel rotation, a counter board was needed to collect data from the custom encoders. Due to our previous experiences with the company, we chose the Diamond Systems® Quartz-MM™. This board provides 10 counters as well as digital inputs and outputs. For our application we ideally use six counters, one per wheel

encoder. In the data acquired for this thesis, only three counters were used as three of the six total wheel encoders were not functioning properly. Figure 5.14 shows a picture of the board and lists its main features, while Figure 5.15 shows the full PC/104+ stack mounted on the rover.

# of counters	10 (16 bit wide)
Maximum input frequency	7 Mhz
Low	-0.5 < < 0.8 volt
High	2.2 < <5 volt
Operating Temperature	0-60°C
Power	+5V ±10%@290mA

Figure 5.14 Quartz-MM™ Counter Board from Diamond Systems®.

5.3.3 Operating System

The operating system we selected is the RedHat Fedora Core 3™ distribution of Linux, a newer distribution (as of its integration within the rover) that is free and for which drivers are available for a substantial amount of hardware. This distribution is not a hard real-time operating system, but due to slow control loops and the inherent stability of the rover platform, hard real-time was not a driving system requirement.

The operating system was installed with the minimum size configuration which requires approximately 400Mb. Additional libraries included:

- `gcc, g++`: GNU C and C++ libraries

- `Libraw1394`: Firewire interface library
- `Tlib`: Machine vision library (see Section 6.1).

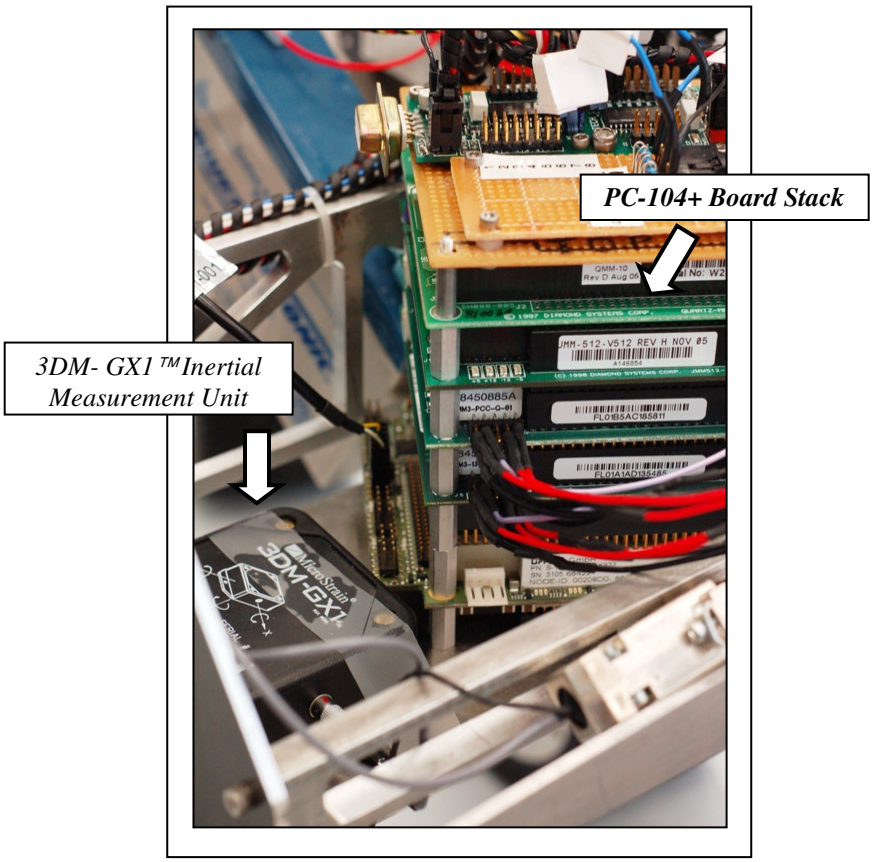


Figure 5.15: Illustration of sensor and computer system mountings.

Chapter 6: Rover Navigation and Control

Once the hardware was fully-functional, a substantial engineering effort was required to implement even a minimal subset of the rover competences (e.g., goto, take-picture) required for planetary surface exploration. This chapter describes the engineering behind the "goto" (navigation/control) task in an environment without GPS. Vision-based obstacle detection and mapping algorithms were implemented within the context of the open source TLIB machine vision library. As described in [46], TLIB was originally designed for Human-Robot Interaction research. We utilize TLIB data structures and its functions to access, convert, and store images. We first added a new firewire grabber for our cameras then added our own image processing functions for obstacle detection and mapping as will be described below (Sections 6.1-6.2).

As discussed in Chapter 5, our rover has limited ability to precisely actuate and measure its actual motions. The remainder of this chapter describes how we adapted and tuned path planning (Section 6.3), localization (Section 6.4), and control (Section 6.5) algorithms to perform as well as possible given our rover's performance characteristics. We conclude with a summary of the embedded software threads that together enable rover navigation in a lab or outdoor environment.

6.1 Obstacle Detection

Because the rover has no previous knowledge of its environment, it must be able to detect and avoid obstacles while navigating. Many techniques have been implemented to detect obstacles, but no one has proven reliable in every possible condition (indoor, outdoor, structured or natural environment). Since navigation was not the main topic of this research, the intent was to build a detection system that could handle structured and relatively open indoor and outdoor environments. Color is perhaps the simplest visual technique that can be used to identify objects/obstacles with known and distinct chromatic characteristics. For our tests, the environment was engineered with uniquely-colored objects that could be detected either as obstacles or waypoint targets.

The main complication when detecting objects with color is lighting. With dynamic lighting conditions, perceived object color can change dramatically, lowering the robustness of simple algorithms such as color filtering. Our solution was to use a machine learning algorithm trained with sample images taken in different lighting conditions. We implemented a neural network tool to identify uniquely-colored obstacles.

6.1.1 Neural Networks

Neural networks were inspired by biological observations of the brain. In their practical implementation neural networks are a simple mathematical model of a function $F : X \longrightarrow Y$, where X is the input (measurement) vector of any dimension

and Y is the output (classification) vector also of any dimension. Neural networks offer several capabilities:

- real-time computation of results.
- Ability to classify poorly-understood, complex, or imprecise data

Neural nets have been used in many applications and have proved their robustness in areas like pattern recognition, system monitoring, and adaptive control. The fundamental neural network is composed of several interconnected single processing units called perceptrons (Figure 6.1). The input of a perceptron can either be the input-vector of the neural network or the output of other perceptrons. It computes the weighted sum over its input which is then used by an activation function to output a single value.

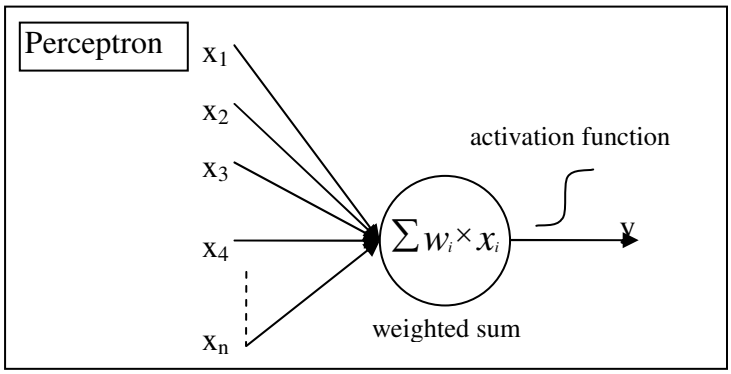


Figure 6.1: Description of a the perceptron

Perceptrons are connected to form a neural network, where each connection is defined with a weight that multiplies the activation level being transmitted. An important property of the neural network is that it was proven that any continuous function can be approximate to any accuracy by a three layer neural network illustrated in Figure 6.2 [47].

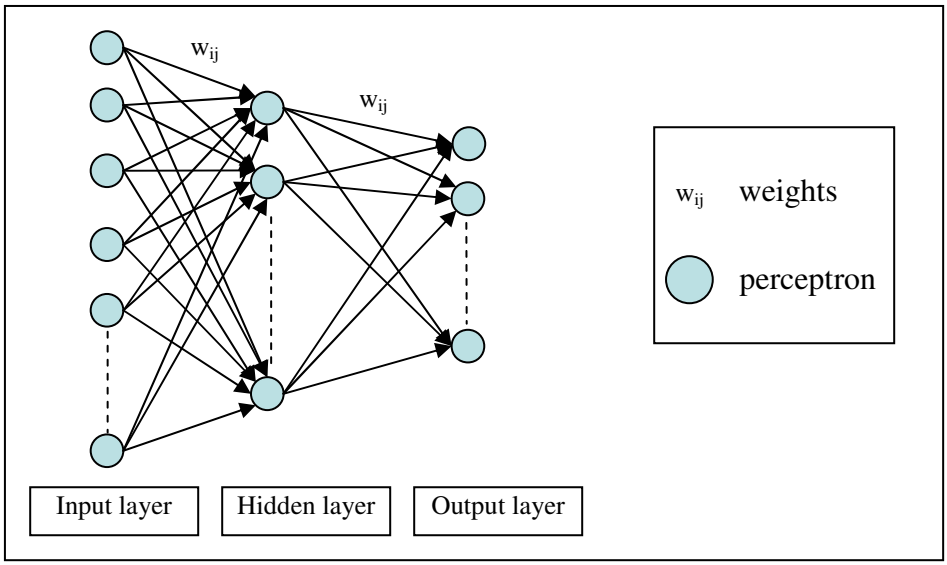


Figure 6.2: A three-layer neural network.

In order to function properly, the neural network needs to be trained. Although other alternative network structures and adaptation strategies have been proposed, the classical training process is the back-propagation algorithm (Figure 6.3) based on gradient descent.

initialize weights;

repeat

for each input vector e in training set

compute output vector;

y[i] = training output value

o[i] = computed output value

for each node *i* in output layer do:

$$error[i] = \frac{1}{2} \times \left(y[i] - o[i] \right)^2$$

$$\Delta_i = \left(y[i] - o[i] \right) \times o[i] \times \left(1 - o[i] \right)$$

for each node *h* in hidden layer do:

$$sum[h] = \sum_i \omega_{hi} \times \Delta_i$$

$$\Delta_h = sum[h] \times a[h] \times \left(1 - a[h] \right)$$

for each weight ω_{lm} in network do

if *m* is an output node $\omega_{lm} += \eta \times \Delta_{i=m} \times a_l$

else $\omega_{lm} += \eta \times \Delta_{h=m} \times a_l$

until stopping criteria satisfied

Figure 6.3: Back propagation algorithm for a three-layer neural network

6.1.2 Application to Obstacle Detection

Our application required processing images defined as 640x480 pixel arrays with 8-bit RGB (red-green-blue) color information for each pixel. The goal is to divide the image into a grid where each cell is assigned a probability indicating how likely it is that this grid is occupied by an obstacle. One of the main challenges in machine learning algorithms is definition of the input feature vector. The computation time as

well as the performance of the algorithm greatly depends on this vector; therefore domain expertise is needed to compute an efficient input array.

Inspired by pixel-level RGB intervals for object segmentation, we define our feature vector with six parameters:

- Red channel average in the cell
- Red channel standard deviation in the cell
- Green channel standard deviation in the cell
- Green channel standard deviation in the cell
- Blue channel average in the cell
- Blue channel standard deviation in the cell

The output vector is a single value between 0 and 1, defining the probability that the cell contains an obstacle. (1 = 100% probability the cell contains an obstacle, 0=0% probability the cell contains an obstacle). A similar cell decomposition method is described in [20].

During the training process the image is divided into cells from which the feature vector is computed. All cells are the same user-specified size. The user indicates if the cell corresponds to an obstacle or not. The neural network is then trained using multiple cells from multiple images. Figure 6.4 illustrates this training procedure.

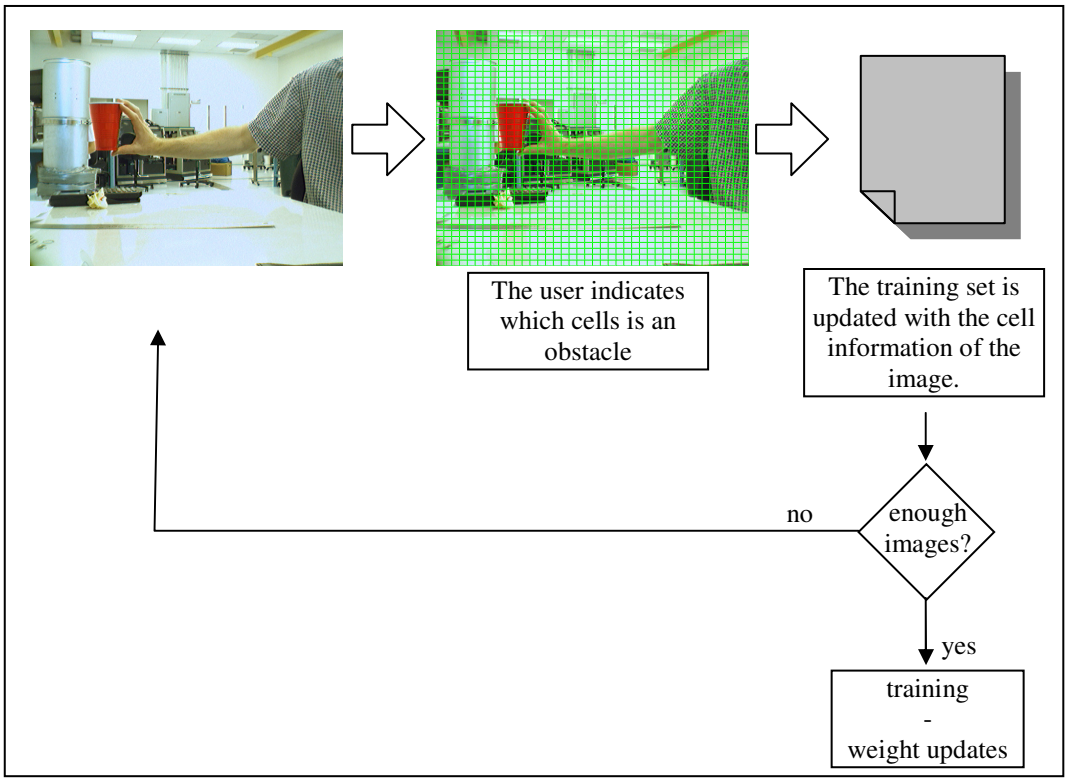


Figure 6.4: Color-based obstacle detection neural network training procedure

6.1.3 Results

When building the training set, 70% of the data is used for training and the remaining 30% is used to validate the neural network. The graphic in Figure 6.5 summarizes the results and the tests performed on the neural network used for indoor navigation.

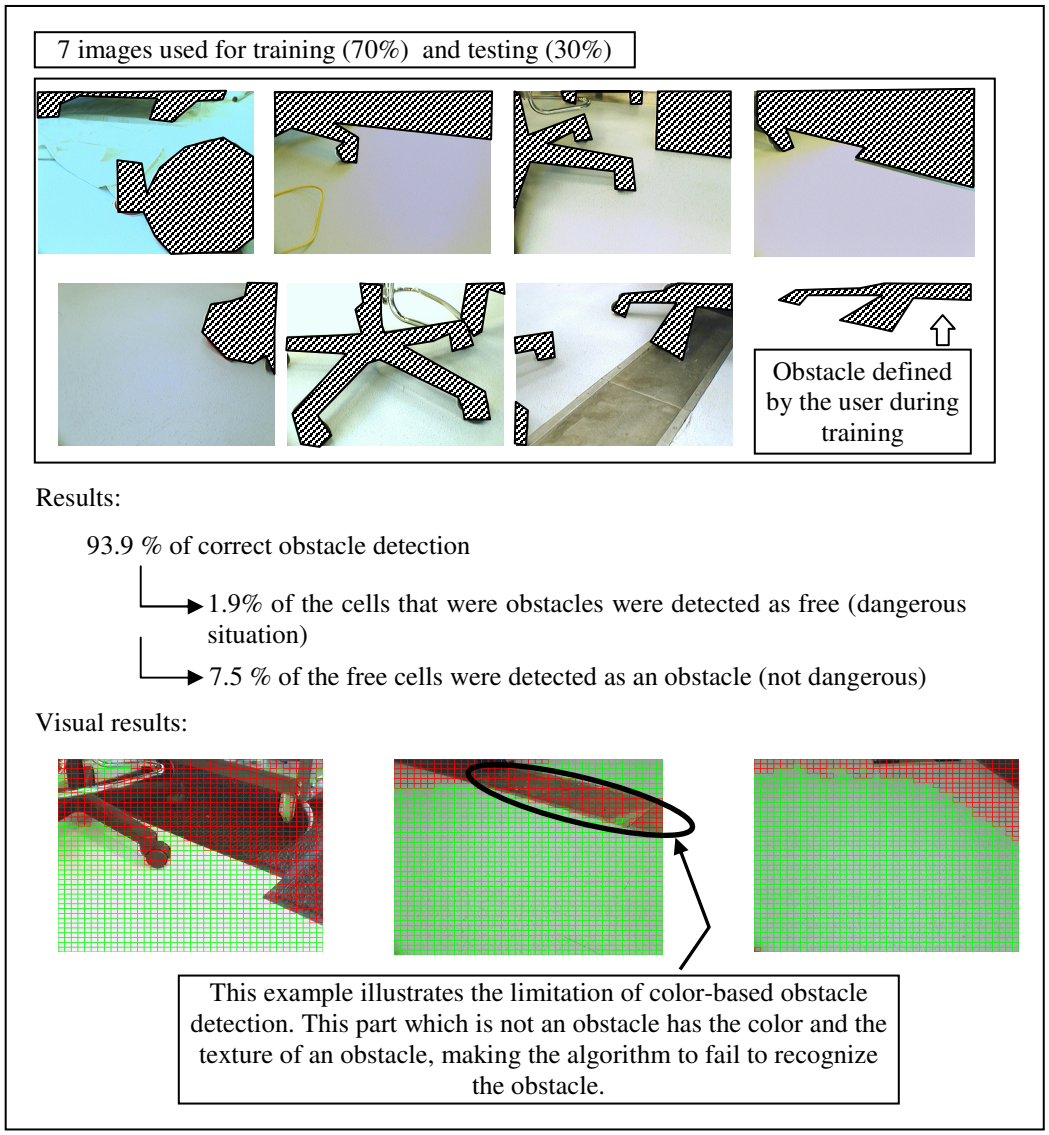


Figure 6.5: Performance of the neural network in an indoor lab environment.

6.2 Mapping

In order to properly navigate, the rover must be able to model the world and store useful terrain information such as obstacle locations. Once obstacles in an image are so classified, the next step is to translate this information into a map so that the navigation system of the rover can establish a correct path. Below, we describe the map structure that was implemented to store terrain information, and then we detail the process of transferring image information into the map structure.

6.2.1 Map Structure

We use a grid-based structure to build our map. This has the advantage of being easy to manipulate and provides fast access to the information. We created a recursive grid structure to facilitate memory allocation. When navigating pieces of the map are dynamically allocated depending on rover location.

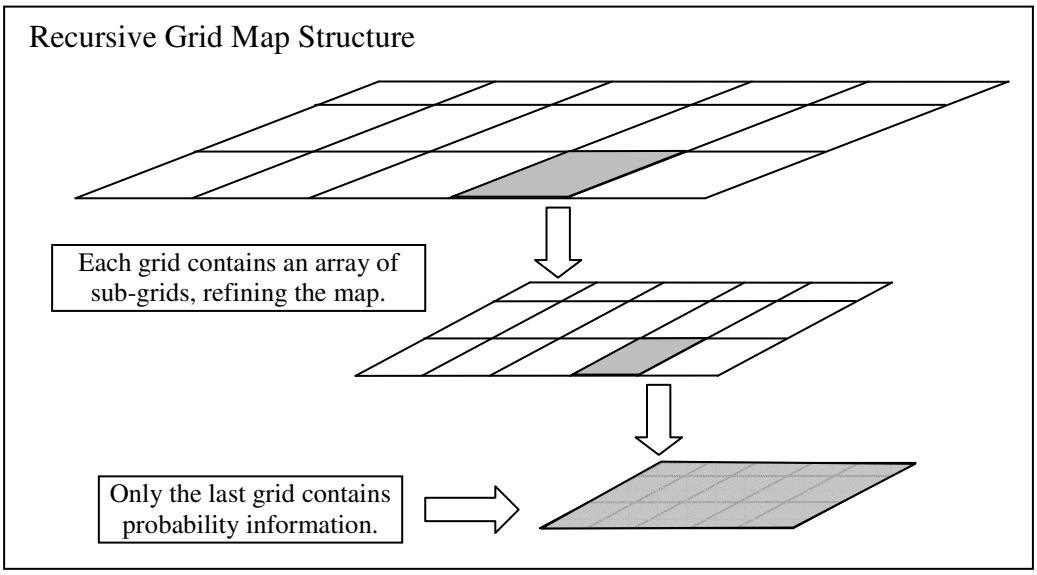


Figure 6.6: Recursive grid map structure

As we will see below, this map is used by the path planner to find a trajectory free of obstacles. The planner represents the rover as a point in the map, a strong assumption that could lead to dangerous paths. To compensate, the map structure enlarges obstacles by a certain number of cells. This "margin of safety" prevents the path planner from finding paths that are too close to the obstacles. Figure 6.6 illustrates the map structure. The bottom-level grid contains the detailed map information:

- Probability of Obstacle $\in$ [0, 1]
- isObstacle $\in$ {true ,false}. Indicates if the cell is part of an obstacle or not. Even if the probability is less than 0.5, the cell can be identified as an obstacle because of the margin of safety

6.2.2 Projecting Camera Images on the Ground

This section describes the geometrical formulas used to describe the transformation between image plane pixels and real spatial locations on the ground. In order to find the projection point of a pixel we need to find the equation of the line that goes through the focal point of the camera and the desired pixel. The projection point on the ground is defined by the intersection of this line and the plane z=0. We assume the position of the focal point in the global frame is known (this point is attached to the rover whose position and orientation is defined at all times). The first step is then to find the position of the pixel in the global (map) coordinate frame. Figure 6.7

shows the projection of an image from a side view. Figure 6.8a shows a top view while Figure 6.8b introduces the coordinate frames used below.

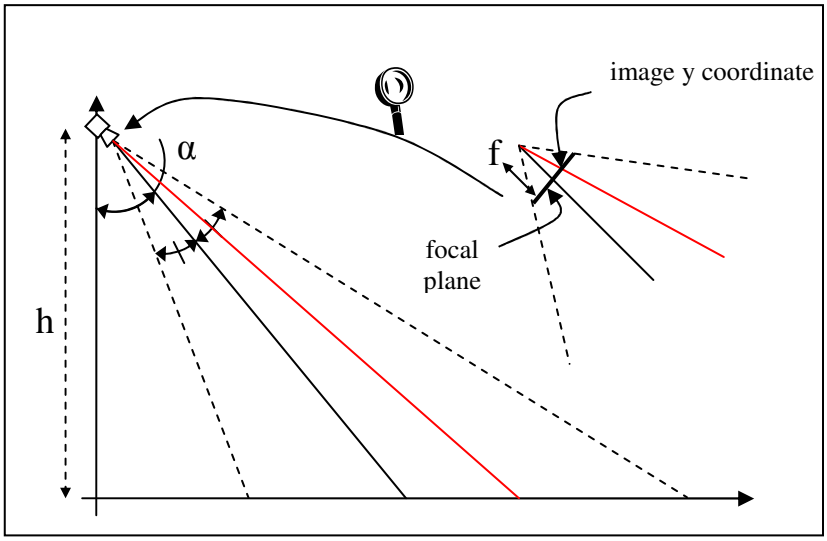
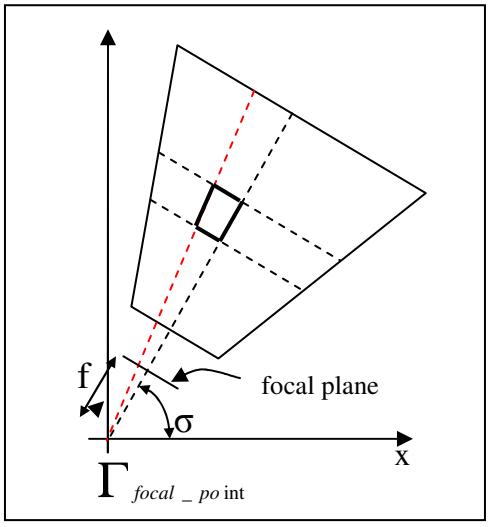
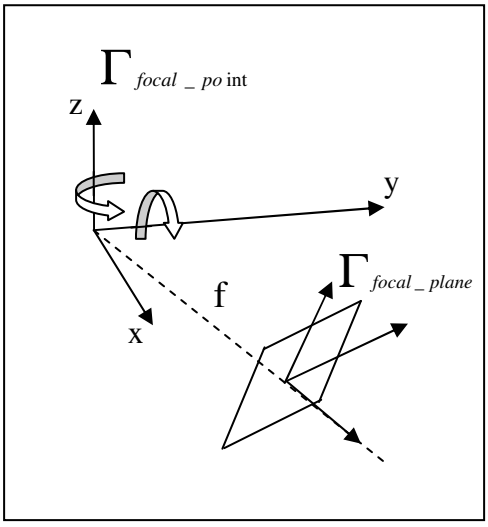


Figure 6.7: Side view of image projection to the ground



a) Top view of projection to the ground



b) Coordinate Frames

Figure 6.8: Image Projection.

Define a frame $\prod_{focal\_point}$ (see Figure 6.8) with the same orientation as the global frame but centered at the focal point. Now define a frame $\prod_{focal\_plane}$ (see Figure 6.8) at the center of the focal plane with the orientation of the camera. We wish to compute the location (*x_real , y_real*) in the global frame of the projection of a point in the image from the following input parameters:

- focal length: *f (px)*
- position of the point in the image: *(x_pixel , y_pixel)*
- height of the camera: *h(m)*
- tilt angle of the camera relative to the vertical axis: α
- Global pan angle of the camera about the vertical axis:

  σ = *pan angle of the camera + rover heading angle*
- position of the focal point in global coordinates: *(x_origin , y_origin)*

Two consecutive rotations as well as a translation are applied to transform $\prod_{focal\_point}$ into $\prod_{focal\_plane}$. The first rotation is the "pan" rotation about the Z axis, while the second is the "tilt" rotation about the intermediate y axis.

"pan" rotation:

$$R_{pan} = \begin{pmatrix} \cos(\sigma) & -\sin(\sigma) & 0 \\ -\sin(\sigma) & \cos(\sigma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.1}$$

"tilt" rotation:

$$R_{tilt} = \begin{pmatrix} \cos\left(\frac{\Pi}{2} - \alpha\right) & 0 & \sin\left(\frac{\Pi}{2} - \alpha\right) \\ 0 & 1 & 0 \\ -\sin\left(\frac{\Pi}{2} - \alpha\right) & & \cos\left(\frac{\Pi}{2} - \alpha\right) \end{pmatrix} \tag{6.2}$$

The coordinate at the center of the focal plane in Γ_{focal_point} is given by:

$$P_{center} = \begin{pmatrix} \cos\left(\frac{\Pi}{2} - \alpha\right) \times \cos(\sigma) \\ \cos\left(\frac{\Pi}{2} - \alpha\right) \times \sin(\sigma) \\ -\sin\left(\frac{\Pi}{2} - \alpha\right) \end{pmatrix} \tag{6.3}$$

One last rotation is necessary if we want to keep the original image coordinate (x along the width axis and y along the height axis)

$$R_{image} \begin{pmatrix} 0 & 0 & -1 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{6.4}$$

Now that we have defined every transformation, it is possible to find the equation of the line that goes trough the focal point and any point in the focal plane. ${}^{\Gamma_{focal\_point}}_{image}T$ is the transformation matrix from can map any point expressed in the image frame in the Γ_{focal_point} frame.

$${}^{\Gamma_{focal\_point}}_{image}T = \begin{pmatrix} R_{pan} \times R_{tilt} \times R_{image} & {}^{\Gamma_{focal\_point}}P_{center} \\ 0 \quad 0 \quad 0 & 1 \end{pmatrix} \tag{6.5}$$

This transformation matrix is used to find the unit vector of the line that goes through the focal point and any desired pixel in the image.

We then solve the projection of this point by computing the intersection of the line with the plane at z = 0. The unit vector of the projection line is given by:

$$\overrightarrow{u} = \begin{pmatrix} x\_pixel \times \sin(\sigma) + y\_image \times \sin\left(\frac{\Pi}{2} - \alpha\right) \times \cos(\sigma) + f \times \cos\left(\frac{\Pi}{2} - \alpha\right) \times \cos(\sigma) \\ -x\_pixel \times \cos(\sigma) + y\_image \times \sin\left(\frac{\Pi}{2} - \alpha\right) \times \sin(\sigma) + f \times \cos\left(\frac{\Pi}{2} - \alpha\right) \times \sin(\sigma) \\ y\_image \times \cos\left(\frac{\Pi}{2} - \alpha\right) - f \times \sin\left(\frac{\Pi}{2} - \alpha\right) \times \cos(\sigma) \end{pmatrix}$$

(6.6)

Having the unit vector as well as a point that belongs to the projection line, we can find its equation:

$$\begin{pmatrix} x = a_1 + k \times u_x \\ y = a_2 + k \times u_y \\ z = a_3 + k \times u_z \end{pmatrix} \quad \text{with} \quad \begin{pmatrix} a_1 = x\_origin - u_x \\ a_2 = y\_origin - u_y \\ a_3 = h - u_z \end{pmatrix} \tag{6.7}$$

Next we find *k* so that z=0 and then compute *x, y*, the coordinates in the global frame of the projection of the point.

6.2.3 Obstacle Probability Map Management

Because the camera is viewing the ground at a certain angle, it is not possible to distinguish the shape of the obstacle. As shown in Figure 6.9, two different obstacles are perceived identically by the vision system.

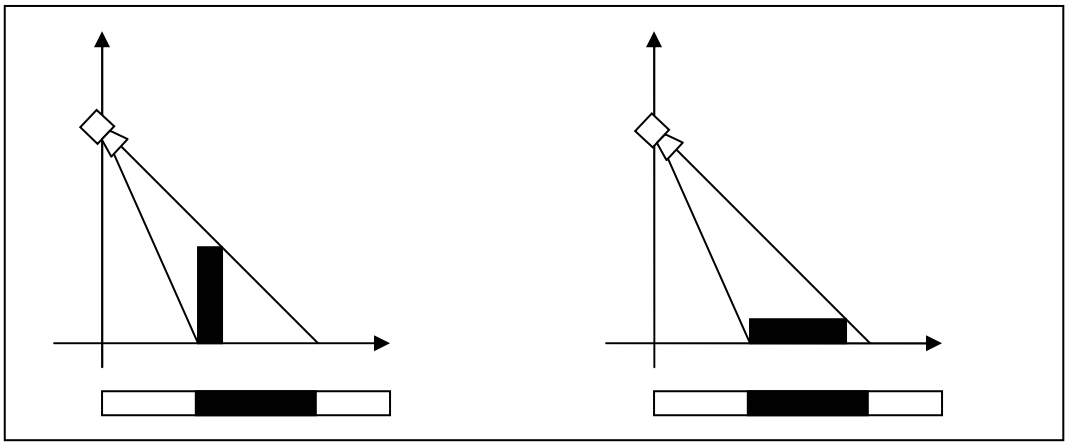


Figure 6.9: Uncertainty in obstacle shape due to projection.

In order to take this issue into account we modified the probability of certain cells. When several consecutive cells are found to be obstacles their probability is lowered to compensate for the uncertainty. The graph below illustrates this modification:

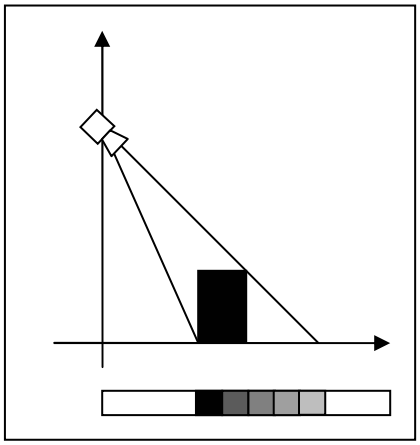


Figure 6.10: Probability adjustment illustration.

In practice, new image data quite often must be correlated with existing map data of the same region. When the rover is traveling, it will take several images of the area it expects to traverse, and updates the map accordingly.

The update formula we used is based on a low-pass filter. It computes the probability of the map cell using the new measurement as well as the previous one:

$$P\_cell_{t+1} = (1-\nu)\times P\_cell_{t} + (\nu)\times P\_measurement_{t+1} \quad (6.8)$$

The variable ν indicates how much we trust the new measured value relative to the existing map value. When ν is high (near 1) then the map will change with each new input value, while a map with low ν will evolve very slowly. This formula represents a low pass filter where ν represents the trade off between delay and noise elimination. We used this formula instead of the classical update formula described in the introduction (derived from Bayes rule) since it allows us more control over the update process. As we will see later, the position of the rover diverges over time due to dead reckoning error therefore it is preferable to favor new measurements.

6.2.4 Example

Since the rover uses two cameras the projection and mapping process is repeated twice by taking into account the different location of the two cameras. Despite the fact that we have two cameras, we are not performing any depth calculations. The pair of cameras is used as a redundant system which improves the accuracy to a limited extent, but stereo vision processing is not currently performed. The figure below describes the process of mapping the two images taken from the left and right cameras.

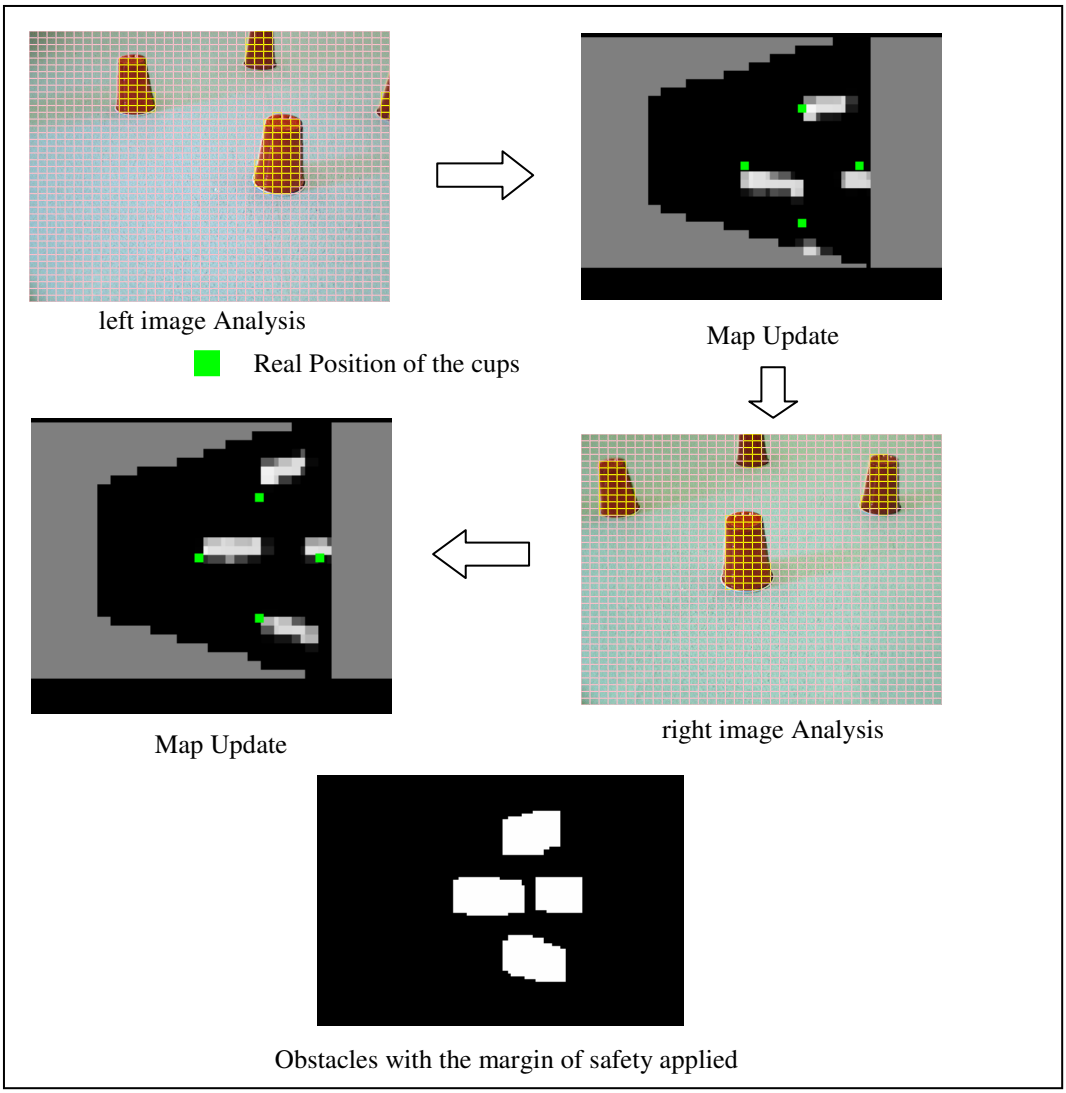


Figure 6.11: Mapping with two cameras and neural network trained to detect red cups.

6.3 Path Planning

Once the robot has mapped its surroundings it must be capable of computing a correct path to reach the goal. A path planning algorithm is required for the rover to find a set of waypoints that reach the goal while avoiding (mapped) obstacles. Much like task

planners, path planning algorithms typically are search-based. We utilize an A* algorithm as our path planner, with initial rover position and heading as the initial state and destination location/waypoint as the goal state. The shortest-distance (clear) route from start to goal state with acceptable turning characteristics is considered the optimal solution, as will be defined more precisely below. The only discrete action possible is *move* that transfers the rover from one position to another while avoiding obstacles and respecting the kinematic constraints of the rover. To define the path planning problem, we first define terminology. A search node is a 2D (**x,y**) position coordinate. Nodes are arranged in a graph where links (edges) between nodes represent direct traversal paths. As was discussed in Chapter 2, A* requires a cost function *g(n)* and heuristic function *h(n).* Each will be precisely defined for our path planning problem below.

While A* is domain independent, the process of expanding the search graph requires definition of valid edges/links and computation of *g(n)* along each path. In our algorithm, a point is considered reachable from another point (thereby defining a valid traversal graph edge) when no obstacles are detected along the path that links the two points. The process of generating new points in the search graph is decomposed into four steps:

- Defining and expanding a window around the initial state
- Finding reachable new points (defining possible graph edges)
- Checking if the path is free of obstacles (validating graph edges)
- Computing cost *g(n)* and heuristic *h(n)* values

The first step is to obtain map information around the point we want to expand (this area is called the window of search). Once this step is accomplished traversal points are detected with the method illustrated below in Figure 6.12. Each default cell in the window is checked; if it is an obstacle then additional neighboring cells are analyzed to see if they can instead be added to the search graph (see the graphic below for cell configuration). Each point is then tested to see if the path is free. This first test is a straight line test -- a series of points are evaluated along the straight line between the initial node and its child.

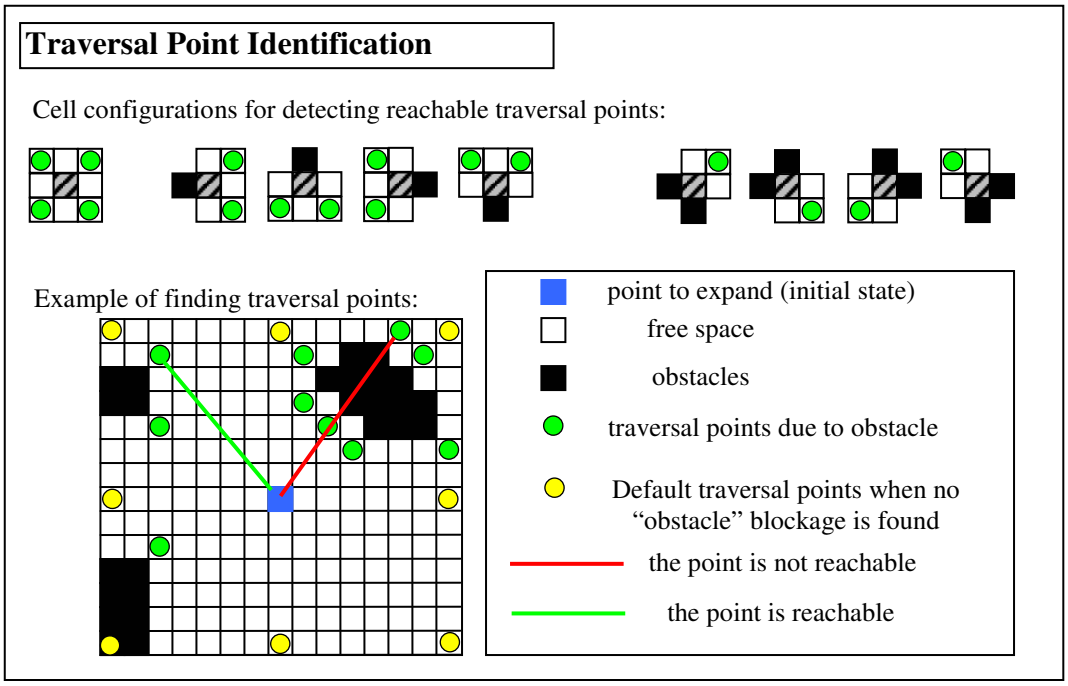


Figure 6.12: Description of the search for new traversal points.

Once a set of children are found to be "straight line" reachable, a second test is performed to see if the global path is correct. If the rover does not stop at each waypoint to turn, kinodynamic constraints dictate creation of a curved path. The smooth path we compute is a spline curve that goes through the waypoint sequence

found by the path planner. Therefore the first straight line test is not sufficient to determine if the path between waypoints is free of obstacles or not. The Figure 6.13 graphic illustrates this issue.

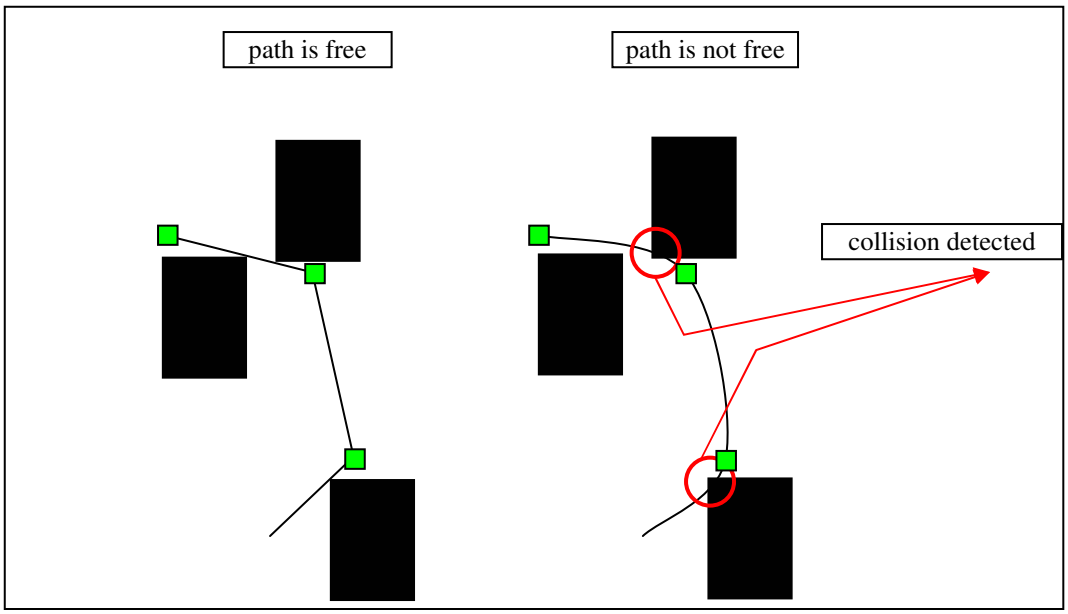


Figure 6.13: Illustration of conflict with curved paths.

In order to avoid this problem a second test is perform on the curved path that the rover would take by going through this new waypoint. A cubic spline is found and a series of intermediate point are tested in order to determine if the path is free. The cubic spline interpolation is described below.

6.3.3 Cubic Spline Interpolation

A cubic spline is a segmented curve where every variable of the spline is described by a cubic function between each point. In our case we have only 2 variables (*x, y*), and a cubic spline linking $n + 1$ points will have the following form:

$$\begin{aligned} S_x(t) &= \sum_{i=0}^{n} \Pi_{i,i+1}(t) \times \left(a_{xi} \times t^3 + b_{xi} \times t^2 + c_{xi} \times t \ + d_{xi}\right) \\ S_y(t) &= \sum_{i=1}^{n} \Pi_{i,i+1}(t) \times \left(a_{yi} \times t^3 + b_{yi} \times t^2 + c_{yi} \times t \ + d_{yi}\right) \end{aligned} \tag{6.9}$$

where $\Pi_{i,i+1}(t)$ is a door function [48]

The goal is then to find the correct $a_{xi}, b_{xi}, c_{xi}, d_{xi}, a_{yi}, b_{yi}, c_{yi}, d_{yi}$ for each segment. With n+1 points (n segments) we have $(n)\times 8$ unknowns. A traditional choice is to constrain the curve with the following set of equations. First the spline must be continuous therefore each start and end point of the n splines must correspond with the corresponding waypoints:

$$\begin{aligned} &S_x(i) = x_i \wedge S_x(i+1) = x_{i+1} (\forall i \in [0,n]) \\ &S_y(i) = x_i \wedge S_y(i+1) = y_{i+1} (\forall i \in [0,n]) \end{aligned} \Rightarrow 4n \text{ constraints} \tag{6.10}$$

Since it is usually convenient to make the interpolation as smooth as possible, we require that the first and second derivatives also be continuous:

$$\begin{aligned} S_x'(i)^- &= S_x'(i)^+ (\forall i \in [1, n-1]) \\ S_y'(i)^- &= S_y'(i)^+ (\forall i \in [1, n-1]) \\ S_x''(i)^- &= S_x''(i)^+ (\forall i \in [1, n-1]) \\ S_y''(i)^- &= S_y''(i)^+ (\forall i \in [1, n-1]) \end{aligned} \Rightarrow 4(n\text{-}1) \text{ constraints} \quad (6.11)$$

Finally for the last 4 constraints two choices are possible:

$$\begin{aligned} S_x''(0) &= 0 \wedge S_x''(n-1) = 0 \\ S_y''(0) &= 0 \wedge S_y''(n-1) = 0 \end{aligned} \text{ (“natural”)} \quad (6.12)$$

or

$$\begin{aligned} S_x'(0) &= f_x'(0) \wedge S_x'(n-1) = f_x'(n+1) \\ S_y'(0) &= f_y'(0) \wedge S_y'(n-1) = f_y'(n+1) \end{aligned} \text{ (“clamped”)} \quad (6.13)$$

This is the traditional way of computing the spline. However, we used a slightly different set of constraints. Since the initial orientation of the spline is provided as additional input, we have a different set of unknowns and equations than the traditional problem.

As for the traditional spline computation, our spline must be continuous therefore each start and end point of the *n* splines must correspond with the corresponding waypoints

$$\begin{aligned} S_x(i) &= x_i \wedge S_x(i+1) = x_{i+1} (\forall i \in [0, n]) \\ S_y(i) &= x_i \wedge S_y(i+1) = y_{i+1} (\forall i \in [0, n]) \end{aligned} \Rightarrow 4n \text{ constraints} \quad (6.14)$$

Then we impose a tangent at each point including the start and end point. This constrains the orientation of the rover at the different waypoints.

$$\frac{\partial y}{\partial x}^{-}(i) = \frac{\partial y}{\partial x}^{+}(i) = \alpha_i \forall i \in [1; n-1] \Rightarrow 2(n-1) constra\,\text{int}\, s$$
$$\frac{\partial y}{\partial x}(0) = \alpha_0 \wedge \frac{\partial y}{\partial x}(n+1) = \alpha_{n+1} \Rightarrow 2 constra\,\text{int}\, s$$
$$\Rightarrow 2n \text{ constraints} \quad (6.15)$$

Finally for the last constraints we impose what we called a coefficient of dilatation for each end point of each segment, providing 2*n* additional constraints. The advantage of our constraints is that each segment can be solved individually and with significant control over the spline properties. For each segment the following equations must be solved:

$$\begin{aligned}
&S_x(0) = x_{start_point} \\
&S_y(0) = y_{start_point} \\
&S_x'(0) = \cos(angle_0) \times coeff_dilatation_1 \\
&S_y'(0) = \sin(angle_0) \times coeff_dilatation_1 \\
&S_x(1) = x_{end_point} \\
&S_y(1) = y_{end_point} \\
&S_x'(1) = \cos(angle_1) \times coeff_dilatation_2 \\
&S_y'(1) = \sin(angle_1) \times coeff_dilatation_2
\end{aligned} \quad (6.16)$$

$angle_0$ and $angle_1$ correspond to $\frac{\partial y}{\partial x}(i)$ and $\frac{\partial y}{\partial x}(i+1)$ for each segment *i*. The coefficient of dilatation controls the shape of the curve; a large value will make the curve take wider turns while smaller values will produce sharper turns. In our practical implementation we found that setting the coefficient of dilatation to a value proportional to the distance between the 2 points generates reasonable curves.

By taking into consideration the fact that (generally) long segments tend to be less populated with obstacles, the tangent of the curve at the different end points (*angle*$_0$ and *angle*$_1$) is computed from a weighted average of the two segments' orientation. The Figure 6.14 graph details the process.

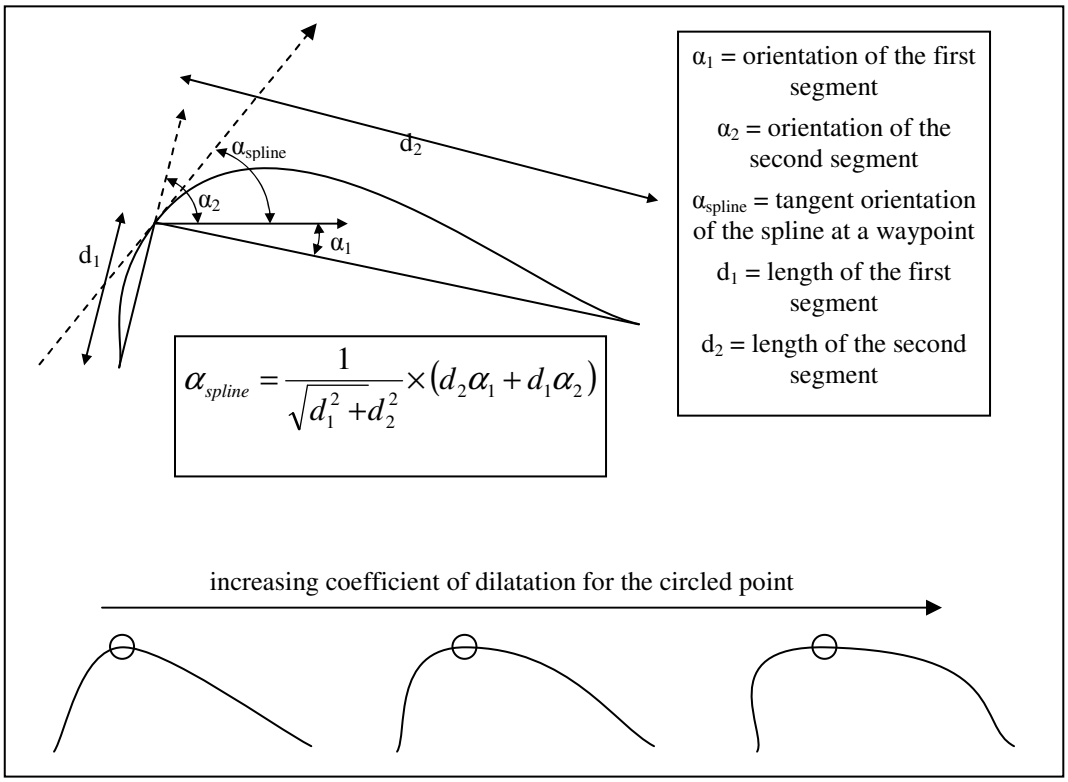


Figure 6.14: Details of the spline computation.

6.3.4 Speed, Acceleration and Radius of Turn

Using a spline it is now straightforward to derive the speed, the acceleration, and the radius of turn:

Speed:

$$V = \sqrt{V_x^{\,2} + V_y^{\,2}}$$
$$V_x = \frac{\partial S_x(t)}{\partial t}; V_y = \frac{\partial S_y(t)}{\partial t} \tag{6.17}$$

Acceleration:

$$A = \sqrt{A_x^{\,2} + A_y^{\,2}}$$
$$A_x = \frac{\partial^2 S_x(t)}{\partial t^2}; A_y = \frac{\partial^2 S_y(t)}{\partial t^2} \tag{6.18}$$

Radius of turn:

$$A_{normal} = \frac{1}{V}\begin{pmatrix} -Vy \\ Vx \end{pmatrix}.\begin{pmatrix} Ax \\ Ay \end{pmatrix}$$
$$R = \frac{V^2}{A_{normal}} \tag{6.19}$$

We have seen earlier that the rover's trajectories are constrained to be curves unless the rover stops before turning. Due to the kinematic constraints (steering), the rover can only make turns with radius greater than 45cm. Even if the path planner minimizes turns in its trajectory, it can detect but does not prevent turns with radius less than 45cm. This is due to algorithmic constraints: the path planning algorithms can only be sure of the value of the turning radius only when the final path is found. In the current implementation, the path planner does not provide warnings about

dangerous turns since the rover is not able to take them into account, but such capability would be straightforward to add in future work.

6.3.5 Heuristic and Cost functions

We have seen earlier that A* requires computation of three values: *h(n), g(n)* and *f(n)=g(n)+h(n).* Indeed, our goal is to obtain a path that is both short and that has the fewest (sharp) turns possible. Our cost function *g(n)* that represents the cost of traversing to node *n* from the start state thus includes two terms to minimize both the distance and the navigation penalty (e.g., accuracy) due to turning:

$$g(n) = Dist(path) + \alpha \times Turning\_Cost(path) \tag{6.20}$$

$$Turning\_Cost(path) = \frac{1}{Dist(path)} \times (\sum change\_of\_heading) \tag{6.21}$$

In this equation, α is a weight that enables the user to tune the influence of the turning term relative to the distance term. As is traditional, the heuristic *h(n)* value is estimated by the distance between the node and the goal position, a guaranteed admissible heuristic.

6.3.6 Examples

In order to test and improve the algorithm, initial tests were performed in a highly structured environment. We built this environment to test difficult cases such as that illustrated in Figure 6.16 and 6.17.

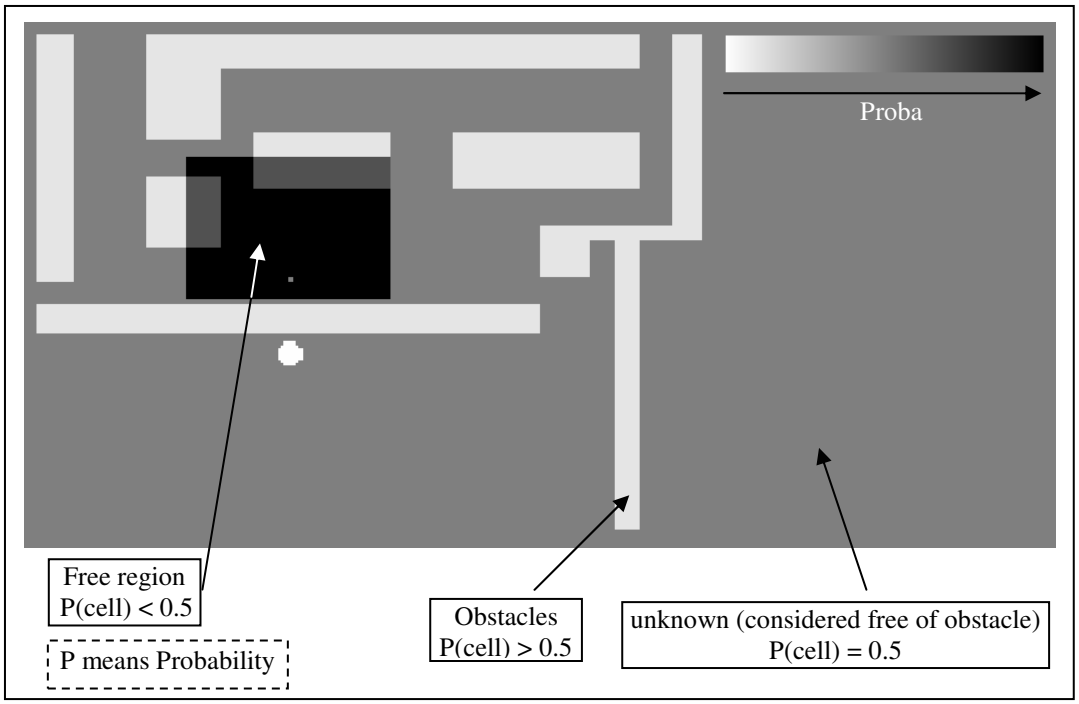


Figure 6.15: Simulation environment used to test the path planning algorithm

As seen in Figure 15 above, this environment is highly challenging to navigate due to narrow passages and a very hard to reach central region. The Figure 16 illustration shows a simple example where the path planner finds a path between two points. The color with which the path is drawn indicates the instantaneous radius of turn while the green line indicates the search graph.

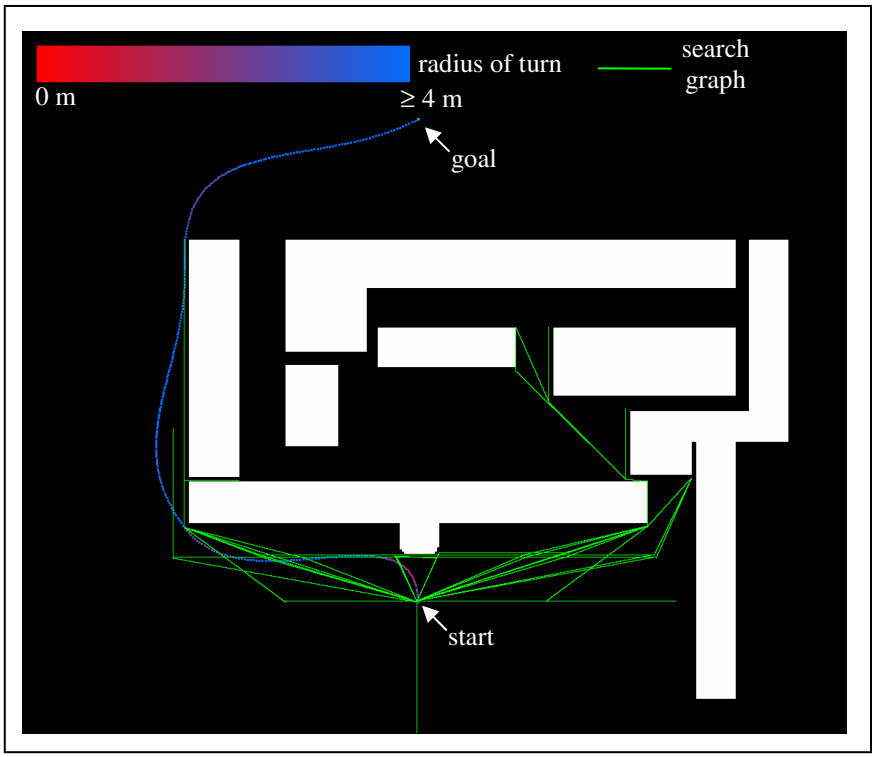
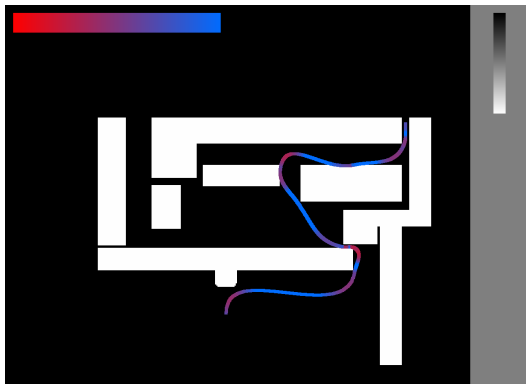
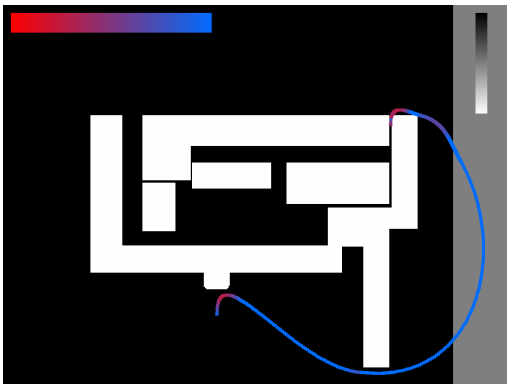


Figure 6.16: Example trajectory computation.

Figure 16 illustrates the behavior of the path planner when changing the margin of safety. The first graph is created with a margin of safety of 4 cells (⇔20 cm in this case). The path planner finds a solution that goes through the narrow passage. In the second graph, this passage is no longer free due to an increase in the margin of safety (6 cells ⇔ 30cm). In this case, the path planner was forced to find another trajectory to reach the goal.



graph 1 - margin of safety = 4 cells



graph 2 - margin of safety = 6 cells

Figure 6.17: Influence of the margin of safety in a cluttered environment.

6.4 Localization

In order to properly navigate rovers need to accurately localize themselves in the environment. As described earlier in Chapter 2, localization is a difficult problem and efficient relative positioning requires advanced algorithms. Our rover is equipped with wheel odometry sensors (counters/encoders) and an inertial measurement unit. In our indoor lab environment, the magnetometers cannot be used due to metallic structure inside the building, therefore only angular rates are available. For outdoor experiments, the magnetometer is used, providing a much more precise heading estimate. Since the rover is not equipped with GPS (a realistic condition for extraterrestrial exploration), it is impossible to locate the rover in an absolute manner indoors or outdoors (i.e. relative to an inertial Earth-centered coordinate system for example). It would be possible to produce good estimates of the rover relative to its environment (e.g., set of landmarks) but this requires advanced SLAM algorithms that are not the topic of this research. Therefore given our goal to utilize

straightforward localization techniques and given our limited sensor suite, rover position will only be determine with classical odometry sensors and basic dead reckoning algorithms that lead to increased error as the rover navigates. Below we describe heading computation in an indoor environment, and then we detail the computation of rover position via dead reckoning with heading as input.

6.4.1 Heading Computation

As magnetometers do not work inside a building, the orientation of the rover must be determined from angular rate measurements. During outdoor experiments, we are using the Microstrain IMU (with magnetometer) to directly provide the heading angle.

Indoors, the IMU heading estimate is incorrect (due to magnetometer corruption) but it does provide angular rates from its internal gyros. For tests on a level lab floor, we presume all angular motion is in yaw, an assumption that simplifies attitude computation without a significant loss of precision as long as the rover remains nearly upright. Because the rate data from the low-cost Microstrain IMU is noisy, we used a first order filter in order to smooth the angular rate:

$$\dot{\theta}(t+1) = \nu \times \dot{\theta}(t) + (1-\nu)\dot{\theta}_{measurement}(t) \tag{6.22}$$

To determine the heading of the rover, we applied a Runge-Kutta integration scheme. Since angular rate is not a function of actual heading angle, Runge-Kutta integration simplifies to integration that average over 3 measurements:

$$\theta^{'} = f(t)$$
$$\theta_{t+h} = \theta_t + \frac{h}{6} \times (k_1 + 4k_2 + k_3) \tag{6.23}$$

with

$$k_1 = f(t) \;\; k_2 = f(t + \frac{h}{2}) \;\; k_3 = f(t + h) \tag{6.24}$$

The Figure 18 graph shows an example of heading computation from angular rate (ν = 0.8 for the low pass filter). Note that due its mounting configuration the positive yaw axis of the rover is opposite from the IMU's positive yaw axis.

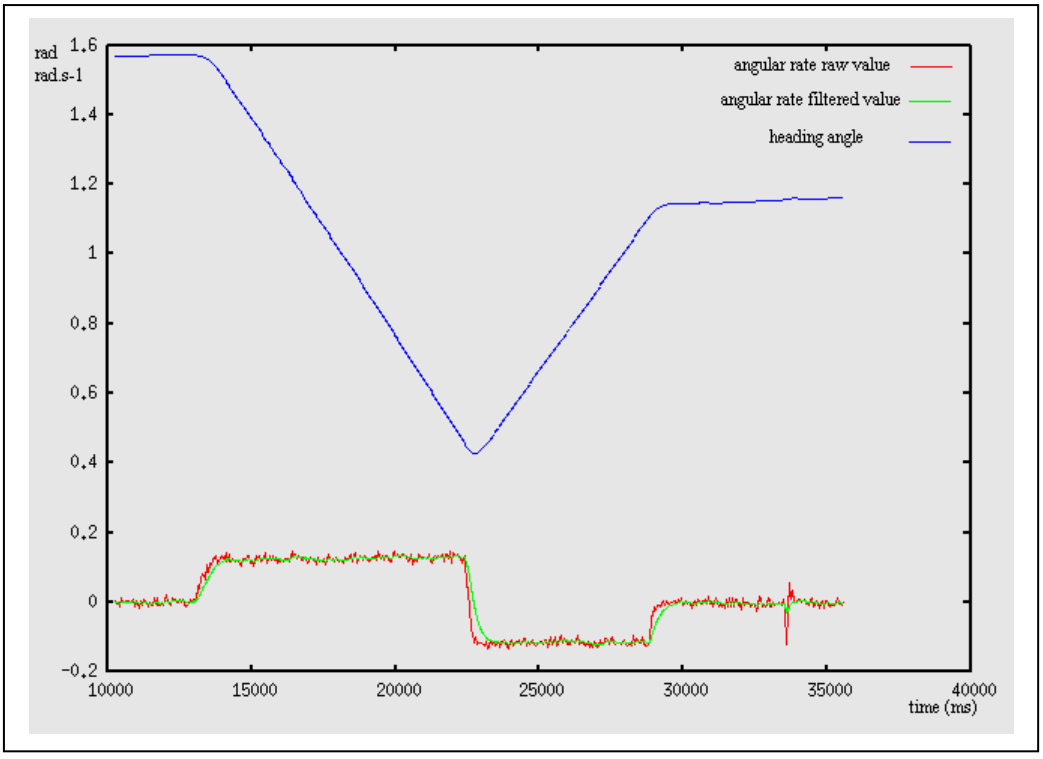


Figure 6.18: IMU angular rate information and heading computation.

6.4.2 Position Computation

To compute the position of the rover we use counter information to estimate the distance traveled between position updates. The x and y coordinates are then calculated from:

$$\begin{aligned} x(t+1) &= x(t) + d \times \cos\left(\frac{\theta(t+1)+\theta(t)}{2}\right) \\ y(t+1) &= y(t) + d \times \sin\left(\frac{\theta(t+1)+\theta(t)}{2}\right) \end{aligned} \tag{6.25}$$

where d = distance traveled between time t and t+1. This position computation algorithm is clearly not an optimal solution. A Kalman filter would likely provide better results given knowledge of rover dynamics, but this model provided sufficiently accurate results to test our task planning system, our primary objective for having a rover that can navigate in its environment. Position computation results and examples can be found in Section 6.5.2.

6.5 Control

For the rover to follow a desired trajectory, it must compute the proper inputs for every motor based on a desired reference command and its actual state. To simplify the problem we decouple our controller in two parts:

- speed control
- position control

The speed controller tries to maintain a desired wheel rotation speed while the position controller directs the rover toward the current waypoint with steering commands.

6.5.1 Position Controller

As described above, the path planner outputs a path that enables the rover to reach a goal position while avoiding obstacles. The path is a list of points, separated by approximately 15 cm. Those points are generated with the spline procedure defined above. The position controller takes as input the current position of the rover and the next waypoint it is supposed to attain. The output of the position controller is a radius of turn that will then be translated into steering commands.

In order to find the turning radius, the controller fits a circle between the two points (current position and next waypoint) with the constraint that the circle is tangent to the current orientation of the rover. The system of equations that needs to be solved in order to determine the radius of turn is nonlinear; therefore to find a solution within a reasonable amount of time we implemented a bipartition technique that searches between turning radii of 0.45 and 4.00 m. The search is limited to ten iterations, which was found to be sufficient to obtain a good estimate of the radius. The graphic below illustrate the search technique for the first three iterations:

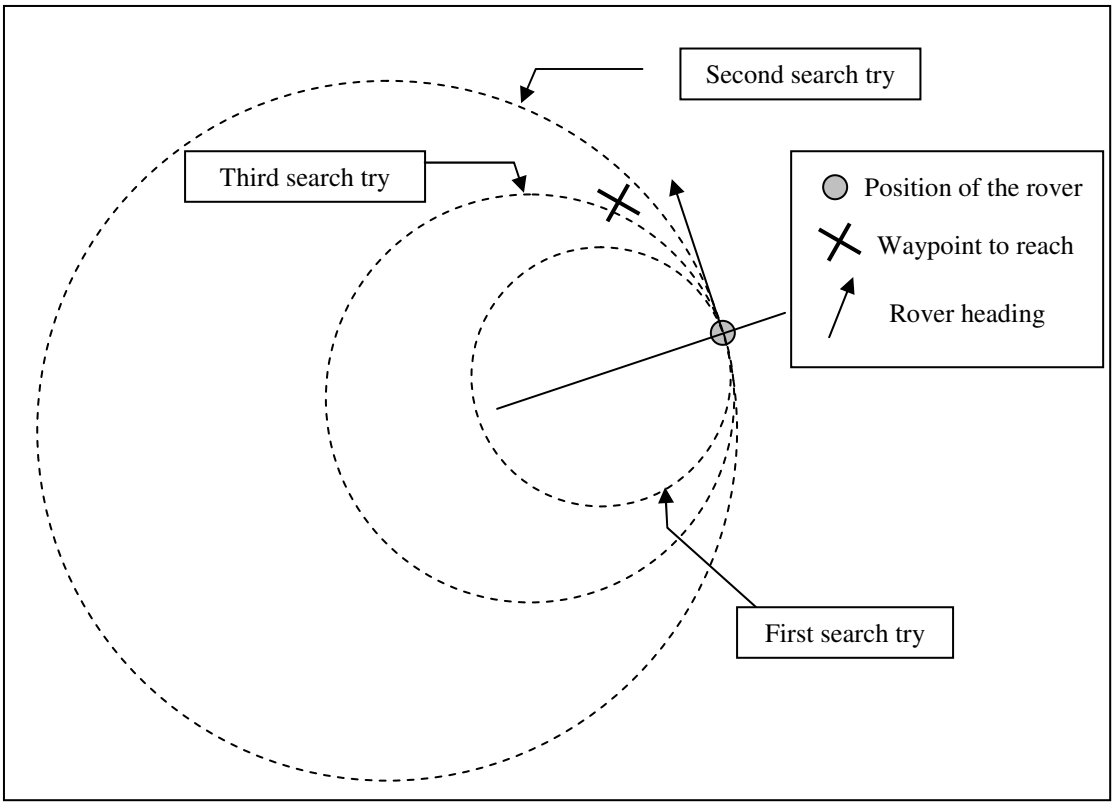


Figure 6.19: Search for optimal radius of turn using a bipartition algorithm.

As there is no sensor that measures wheel steering angles, no additional feedback control could be implemented to ensure the commanded radius of turn was correctly executed.

6.5.2 Position Control Results

Although a feedback-based speed controller is in place, position control is achieved through open loop dead reckoning since the real position of the rover is unknown. The Figure 6.20 graphs show three examples with different reference trajectories. Figure 6.20 a) illustrates an example where the rover followed a smooth trajectory. Since there are no tight turns, the rover is not slipping and the heading computation is

accurate resulting in a close match between commanded reference, estimated, and real (actual) positions. This example demonstrates our position control algorithm is functioning properly.

Figure 6.20 b) shows a case where the rover is directed to perform large changes in heading. We can clearly see that the estimate of the orientation become less accurate as the rover changes its orientation. Slippage due to tight turns adds substantial error in the position estimate, as indicated by the difference between estimated and real positions. The position control algorithm, however, is faithfully minimizing the error between the commanded reference and the navigation system's position estimate.

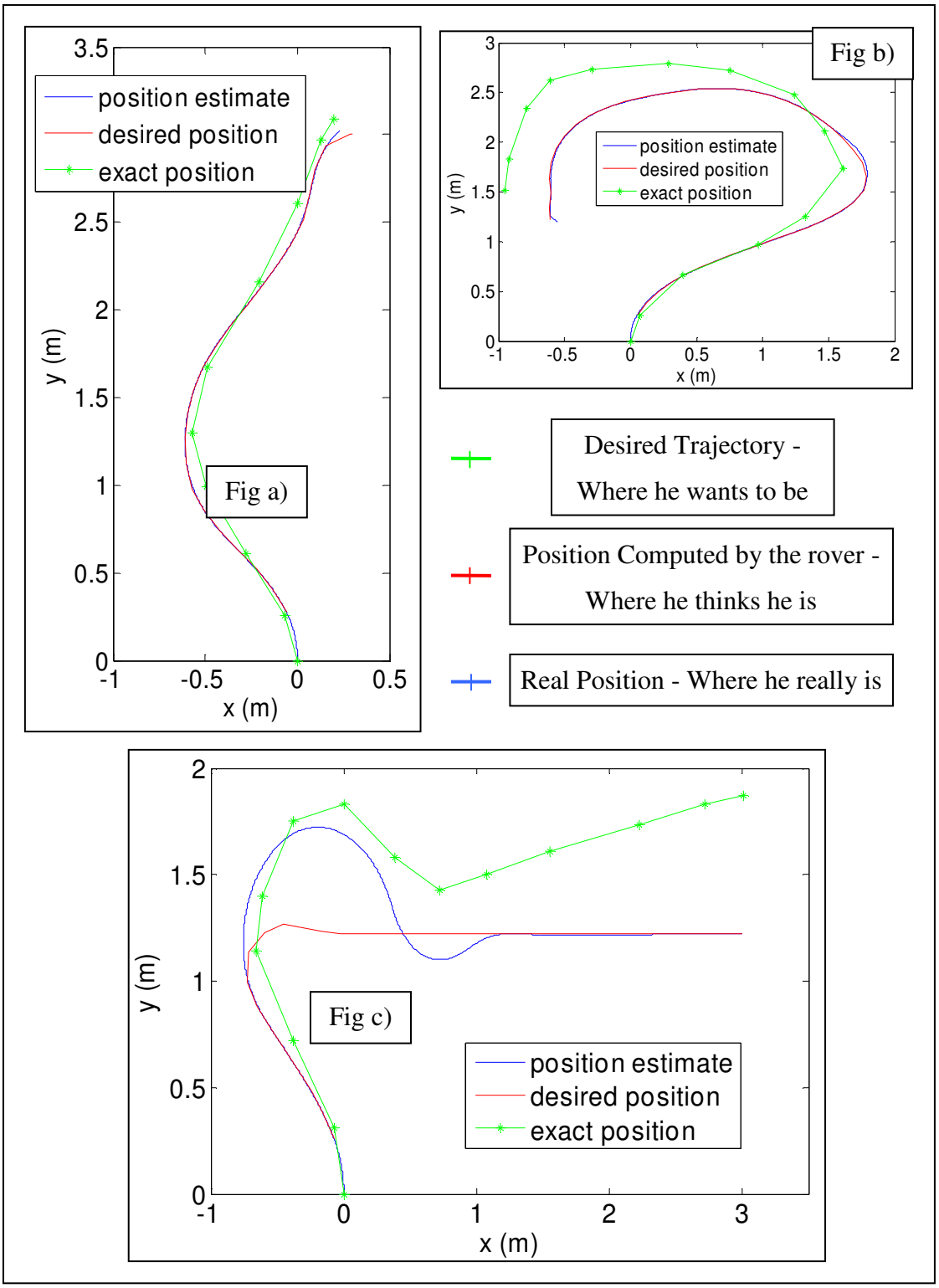


Figure 6.20: Position estimation and control

Figure 6.20 c) shows that kinematic constraints of the rover can prevent it from following a desired trajectory. As illustrated in the graph, the desired trajectory makes a very tight turn; the rover tries to follow it but its minimum radius of turn is 45 cm which is not sufficient to track the desired path. Nevertheless after one meter the position control algorithm succeeds in placing the rover on the desired trajectory. Again, however, we can observe that tight turns provoke slippage and errors in the orientation estimation given our indoor test environment where magnetometer data is unavailable but "real" position can be easily measured. Note that, given similar slippage properties, we expect better performance outdoors where the magnetometer is available.

6.5.2 Speed Controller

Each generated plan includes a speed schedule for every waypoint. Although extensible in future work, this speed schedule is currently a constant value (0.08 $m.s^{-1}$) for waypoints that are situated in a free region, and 0 $m.s^{-1}$ for waypoints that are in unknown areas. This prevents the rover from traversing into zones it has not yet mapped. For the rover to traverse at the commanded speed, a speed controller was implemented. While traversing, the rover can encounter rough terrain, therefore more power must be sent to the drive motors to compensate as different terrain conditions are encountered.

Because of the encoders embedded inside the wheels, we were able to implement a closed loop speed controller. Recall that the rover has the following characteristics:

- Due to the use of the RS/232 interface, the frequency of the control loop is practically limited to 10Hz. (7 bytes to command each servo, 10 servos to control, 9600 Baud speed, requirement for the CPU to devote resources to other tasks beyond RS/232).
- Because we are using commercial servos to drive the rover, the resolution is limited to 10 values and in practical implementation, only 6 of these are applicable (to avoid damage to the servos).

To summarize, we have a very slow control loop with limited servo resolution. Therefore, the design of the controller was simple, based on practical observation of the system rather than theoretical analysis. Currently, speed control is achieved with a proportional controller. The first step is to read each encoder and filter it with a first order low pass filter to eliminate noise:

$$\dot{V}(t+1) = \nu \times \dot{V}(t) + (1-\nu)\dot{V}_{measurement}(t) \text{ with } \nu = 0.85 \quad (6.27)$$

The Figure 6.16 graph shows an example of speed estimation using encoder information. The red graph is the estimate of speed based on the average of three encoder signal time histories. The green graph shows the output servo command and is not expressed in m/s. The purpose of this graph is to illustrate the problem of estimating the speed. ($\nu = 0.85$ for the low pass filter).

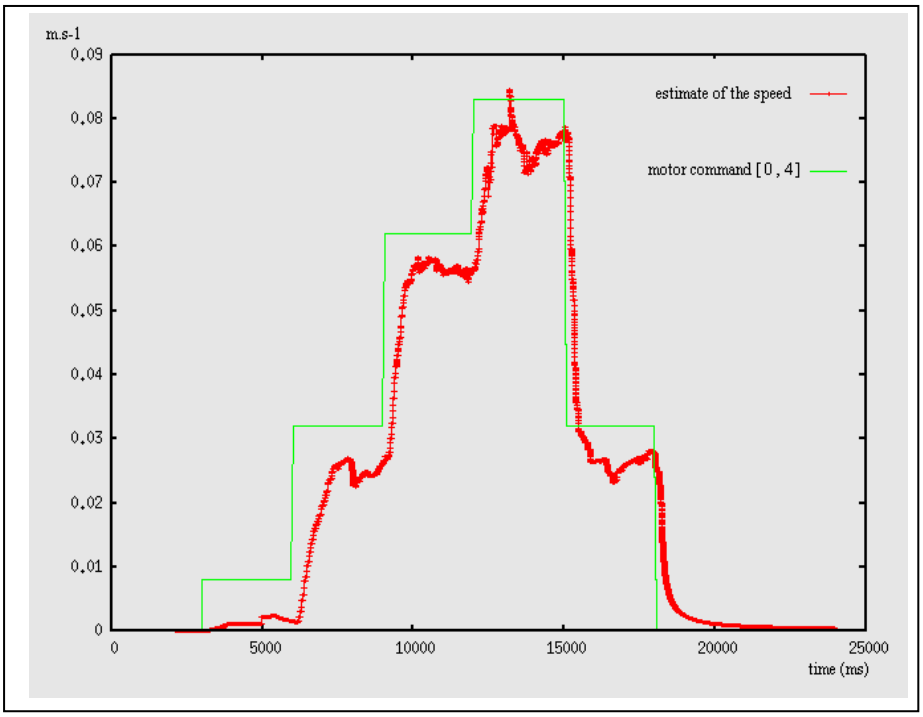


Figure 6.21 Speed estimation and control illustrating the effect of limited encoder resolution and servo control authority.

When implementing our speed controller, we add a lot of problem with noise. The noise is mainly due to our poor counter resolution (150 counts per wheel revolution); we try to reduce it using a low pass filter but that kind of filter introduces delays in the speed determination which deteriorate the performance of the control algorithm.

The next step is to compute the speed error and the appropriate servo command based on the proportional control law:

$$\begin{aligned} Ve &= Vdesired - V(t) \\ C+ &= K_p \times Ve \end{aligned} \qquad (6.28)$$

Note that we are adding the error term to the command because this is a speed controller (e.g., when we reach the desired speed we want to keep sending the same

command to the motor). This is different than position control where actuation is zeroed when the desired position is reached.

6.5.3 Results

Because of the very low power authority, the results are very bad and the behavior of the rover is not optimal. Indeed when the desired speed is setup to be in between 2 control levels, the control loop switches from the 2 values. The graph below illustrates this issue.

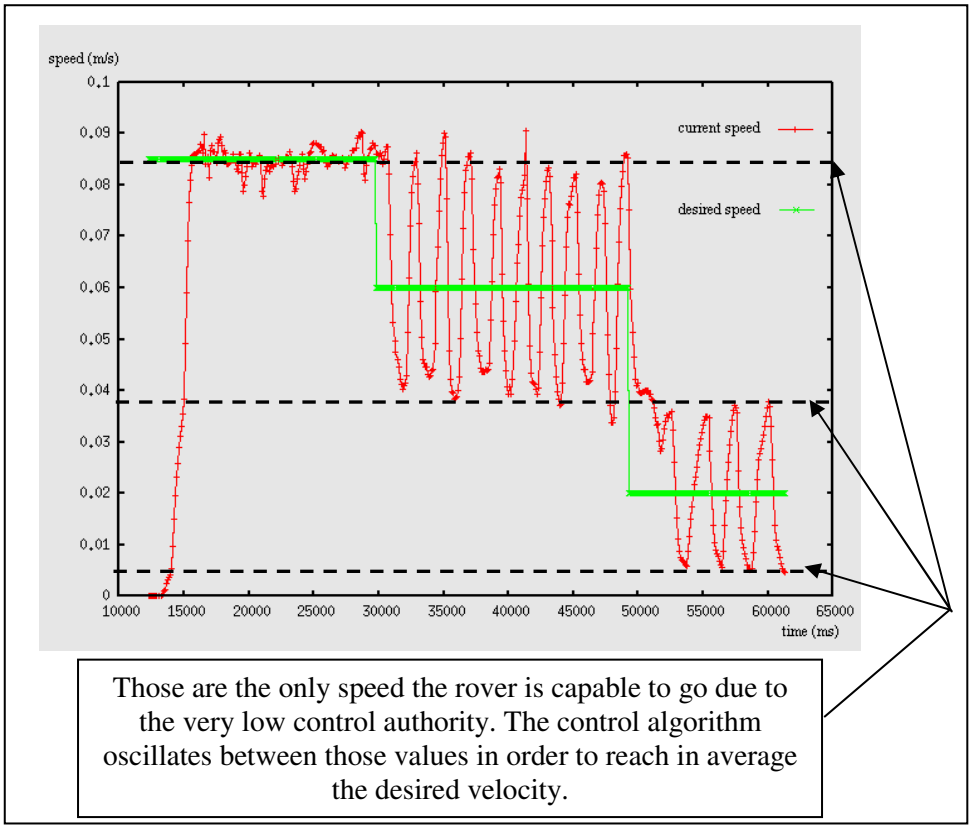


Figure 6.22: Control System Analysis.

When the desired speed is setup to be close to the one corresponding to a power level, the control system oscillates a lot less. The fig 6.17 illustrates this phenomenon; since

the first desired speed corresponds exactly to a command level of 4 the control system finds this value and keeps it as the error is very small. On the contrary the second and third speed are right in between 2 command level making the system to perform numerous and huge oscillations.

A solution to this issue is to simulate a new command level by switching very fast from the upper command level to the lower one. If this is perform at a very high frequency then it looks like there is a new command level. In our case the frequency control loop is limited to 10 Hz due to the slow serial communication making this solution impossible to implement.

6.7 Software Implementation

The navigation and control software described above was implemented and tested on our rover, with the goal of providing a test platform capable of executing exploration tasks as dictated by our task planner and coordination executive. The program that controls the rover is composed of 4 different threads:

- *navigation*: detects obstacle, updates the map, and computes new path plans as necessary (frequency $\in$ [2 5] Hz depending on the time required to compute the trajectory)
- *position (localization)*: computes the position of the rover by reading and processing sensor information (frequency = 20 Hz)
- *goal management*: In accordance with the specified plan and the actual rover position, this thread determines the next position that the navigation/control system must reach (frequency = 20 Hz)
- *control*: This thread controls the steering servos and speed of every wheel so that the rover traverses to the next waypoint (frequency = 10 Hz)

Table 6.1 summarizes the variables that are shared by the different threads.

Table 6.1: Shared Variables for Rover Navigation and Control Threads

Variable	Thread
Position (x, y, heading)	Position (read + write), Goal Management (read), Navigation (read), Control (read)
Plan (list of waypoints)	Navigation (read + write), Goal Management (read)
Next Waypoint	Goal Management (read + write), Control (read)
Desired Speed	Goal Management (read + write), Control (read)
Speed	Control (read + write)
Mode	Navigation (read +write), Goal Management (read + write), Control (read)

6.6.1 Navigation Thread

The navigation thread enables the rover to acquire knowledge of its surrounding environment as well as to make sure that it navigates safely. The first step is to acquire the image from the left camera. Once this is done the trained neural network is used to detect obstacles. The map structure is then updated to take into consideration this new image data. This same set of operations is repeated a second time for the image grabbed from the right camera.

The next step makes sure the previously computed plan does not intersect any obstacles, since map updates may have identified new obstacles that might block the path planned for the rover. A procedure is then executed to detect such dangerous cases and to update the speed schedule of the rover. This procedure guarantees the rover will not drive in unmapped areas thus will not hit obstacles. As described

previously, speeds of waypoints that are situated in unknown or obstacle cells of the map are set to 0 $m.s^{-1}$. In the case where the previous plan does not intersect any obstacles the thread becomes dormant until the next invocation. If the path is found to no longer be free of obstacles, then the planning tool is called in order to compute a new set of waypoints. Note that the rover may be stopped during this operation as the time needed to find a valid plan cannot be predicted (it highly depends on environmental complexity). The graphic below summarizes navigation thread activity:

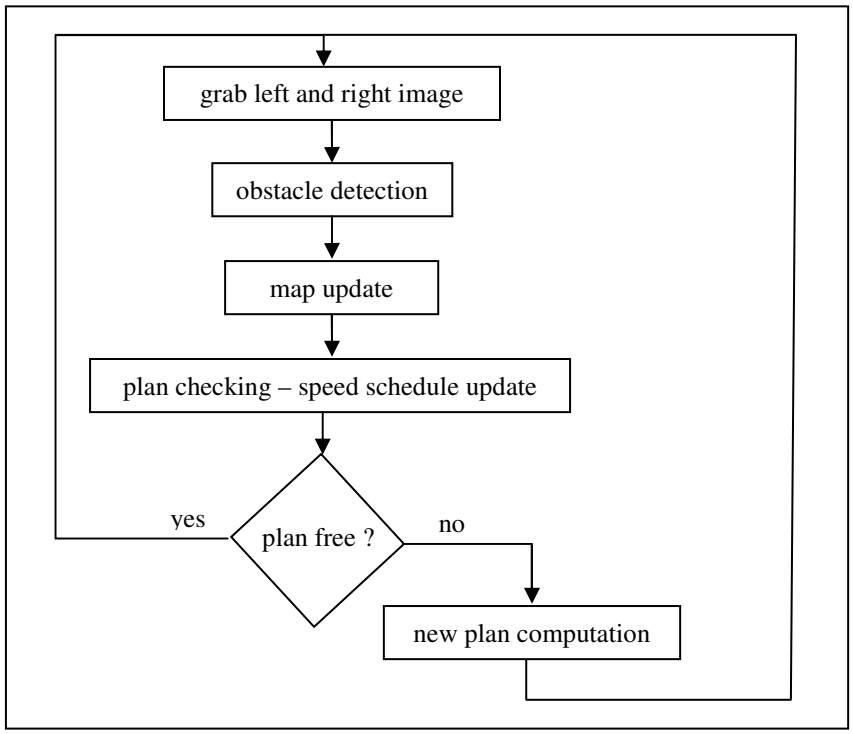


Figure 6.23: Navigation thread activities

6.6.2 Goal Management Thread

As discussed previously the rover has kinematic constraints that can prevent it from following the planned trajectory precisely. Note that the rover is mechanically capable of turning in place, but this function was not implemented; in future research such motion capacity can be implemented making every waypoint reachable. In several cases, despite the effort of the control thread, the rover will not exactly reach the desired waypoint. Figure 6.24 illustrates this issue.

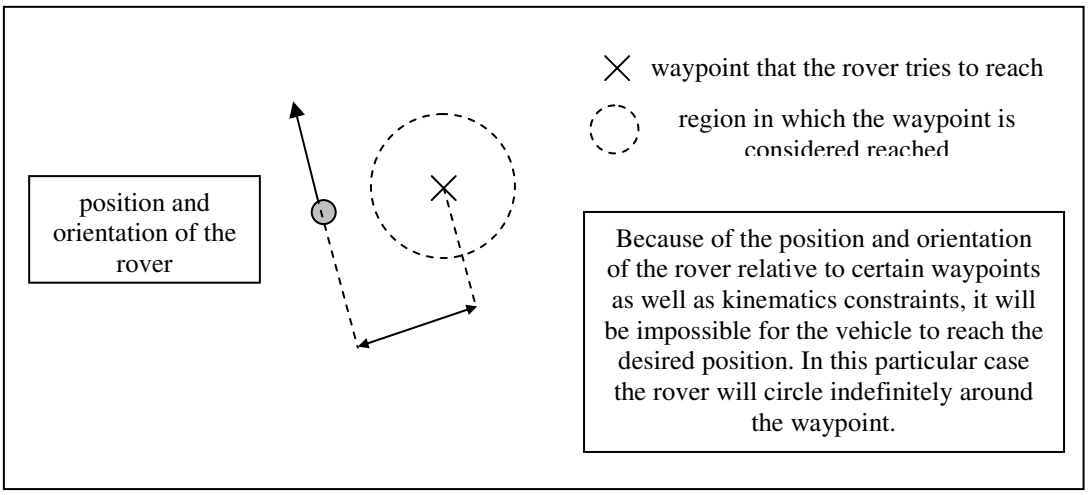


Figure 6.24: Situation where the next waypoint is not reachable

As described by the above graphic, the desired position of the rover can be unreachable from its current position and orientation; the purpose of the goal management thread is to control which waypoint should be reached next. The goal management thread consists in a series of test that first check if the rover has reached waypoints or if it is in a situation that requires modification of the waypoint to reach.

The first test checks if the current waypoint has been reached; a waypoint is considered reached when the rover is positioned within a pre-defined region (see graphic). If the waypoint is reached then it is replaced by the next waypoint in the plan. A second test is then performed to avoid dangerous situations. As explained earlier the rover can be in such position and orientation that it becomes impossible for it to get to the waypoint. This situation can be avoided or compensate for, by computing the distance from the rover to the next waypoint in the plan. When this distance is shorter than the one that separates the rover to its current waypoint or the length of the segment between the 2 waypoints, then goal management thread will replace the current waypoint by the next one in the list. This method has proven to improve greatly the behavior of the rover but it is not 100% reliable.

First, since the rover is not exactly following the trajectory computed by the path planner, it is not sure that it will not hit obstacles. While in a lot case the next waypoint will be easier to reach, it is not guaranteed. Some specific cases can put the rover in a more dangerous situation; but it is important to recall that the goal was not to obtain the best navigation system possible, but to have a system that behave nicely in easy to navigate environment.

Finally a third test is performed in order check if a new plan has been recomputed by the navigation thread. If so then the list of waypoint will be updated.

6.6.3 Position (Localization) and Control Threads

The position thread updates the position of the rover based on sensor information. It starts by reading data from the IMU and encoders then following the localization procedure defined above in which heading angle is updated first then *x, y* coordinates are estimated via dead reckoning. The control thread computes the necessary commands for the rover to maintain a desired speed and to reach a desired position (the current goal waypoint). As described previously, heading control is performed open loop, while speed control is performed with the proportional controller that bases commands on encoder feedback.

Chapter 7: System Test Results

7.1 Graphical Astronaut Interface

Because we have no way to instrument an astronaut, we have limited our human-in-the-loop testing to a simple computer interface that allows the "astronaut" test subject to enter requests and interact with the coordination executive. As described in [44], an astronaut is likely to communicate with simple and clear voice commands that a speech processing program could understand. Since the implementation of such a program was beyond the scope of this thesis, a simple graphical interface was deemed sufficient for our experiments.

As a support tool, the primary design requirement was that astronauts must be able to express actions and needs that can be understood by our planning system. Consequently the interface has the following components:

- *"Help" button:* the human can indicate he needs life support equipment.
- *Tool Assistance:* A "require rover" button enables the human to choose from a different set of tools (competences) that are found only on rover(s).
- *"Leave plan" button:* This button indicates the astronaut has elected to deviate from the plan to pursue a goal of their own.

- *Action Indication:* The actions to be executed by the astronaut are displayed with the expected time. A "done" button enables the astronaut to signify task completion.
- *Task dialogue:* When tasks results require human approval, the astronaut has the ability to answer with *yes* or *no*.

Figure 7.1 shows the interface presented when the astronaut requires a tool, and when he is requested to confirm a task result.

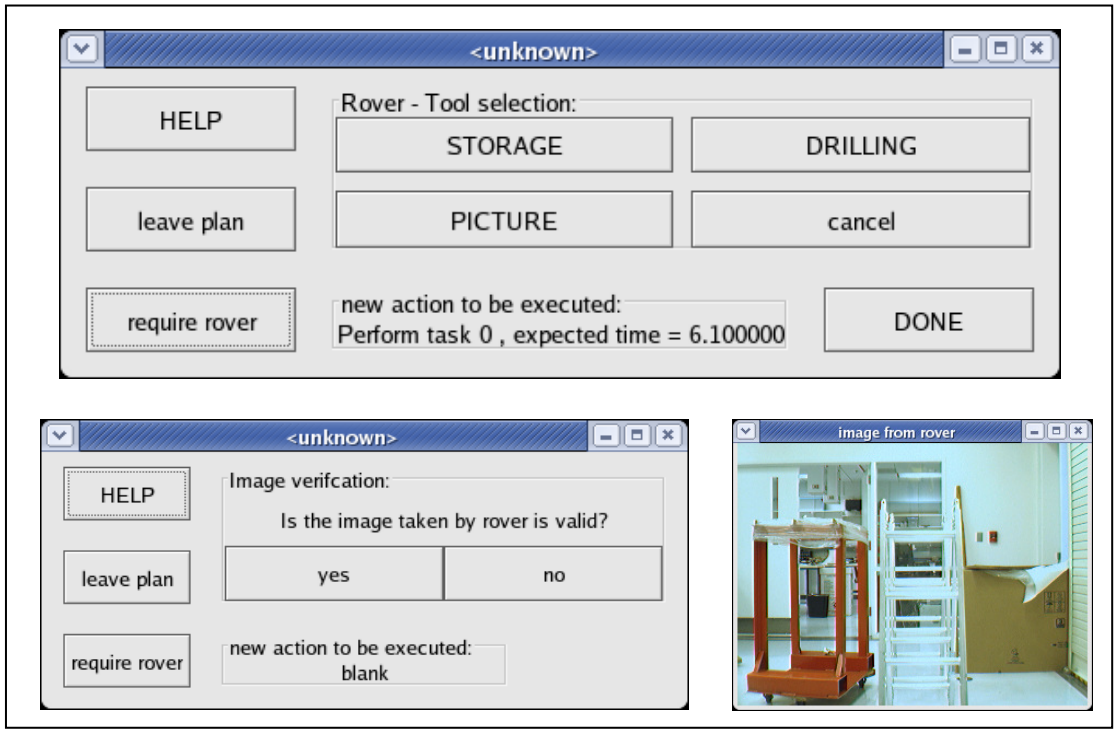


Figure 7.1: Graphical Astronaut Interface.

7.2 Simple Indoor Scenario

The first test scenario was designed to verify the planning system's integration with a real rover and a real human. The system was only composed of those 2 agents, and

two “Take a Picture with Approval” tasks were specified. Only the rover was equipped with a camera, therefore the planning problem was trivialized to the rover sequentially completing the two tasks but requiring approval from the astronaut (thus the astronaut had to focus attention on the “take picture” task at least briefly). The goal was to test the smoothness of the system as well as the correct flow of communication between the embodied agents and the planner/executor system. At the discretion of the astronaut, a call for help may also be issued during plan execution to test both the planner and the rover reactions. The “Take a Picture with Approval” task consists of a sequence of three sub-actions for the rover: 1) tilt the camera pair to the horizon, 2) take a picture, and 3) tilt the camera back to its default configuration.

During the execution, the coordination executive and task planner reacted as expected, effectively managing task execution and the flow of information. It also provided a correct response to the emergency call from the astronaut. Figure 7.2 shows the execution time line while Figure 7.3 shows the path followed by the rover during execution. We can clearly see that the rover is traveling more slowly than expected and that it does not follow the expected path. This is due to the fact that traversal time is computed using a straight line approximation between waypoints, without consideration of kinematic constraints. The closer the waypoints, the less accurate this approximation is, as the rover is taking a more circuitous path to account for required heading changes. As seen previously, dead reckoning results in increased position error over time, and orientation error also grows without magnetometer data (indoors). We can clearly see in Figure 7.3 that the rover, when returning to the its

initial position to rescue the astronaut, is approximately 1 meter away from where it believes it has traversed. During execution the rover performs a 180 degree change of orientation resulting in a significant heading error. As error in heading has been observed to have significant influence in position computation error, this result was expected.

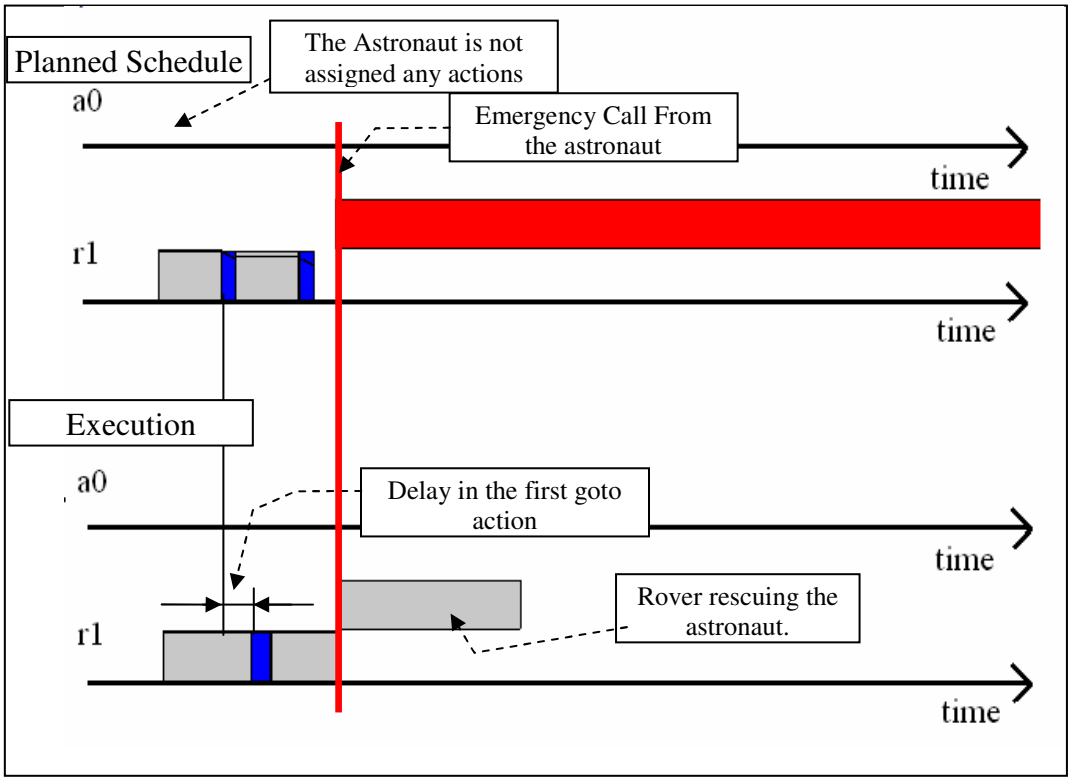


Figure 7.2: Plan and Execution Schedule.

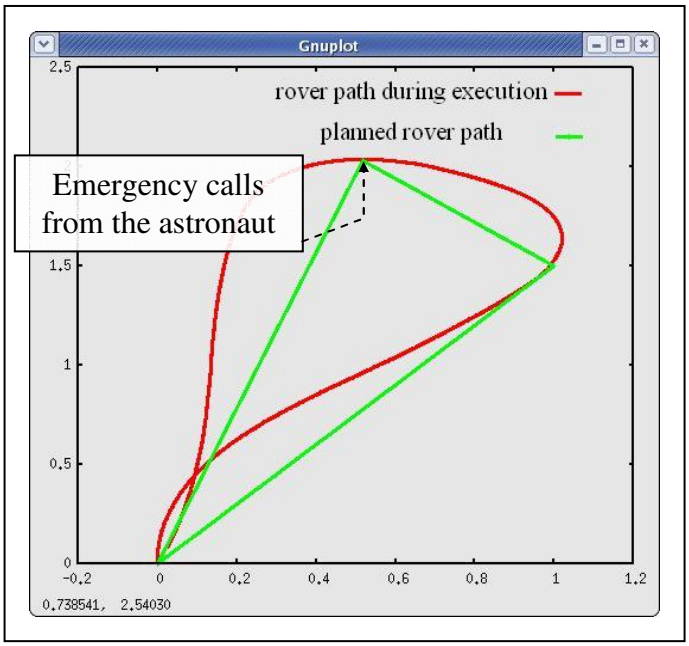
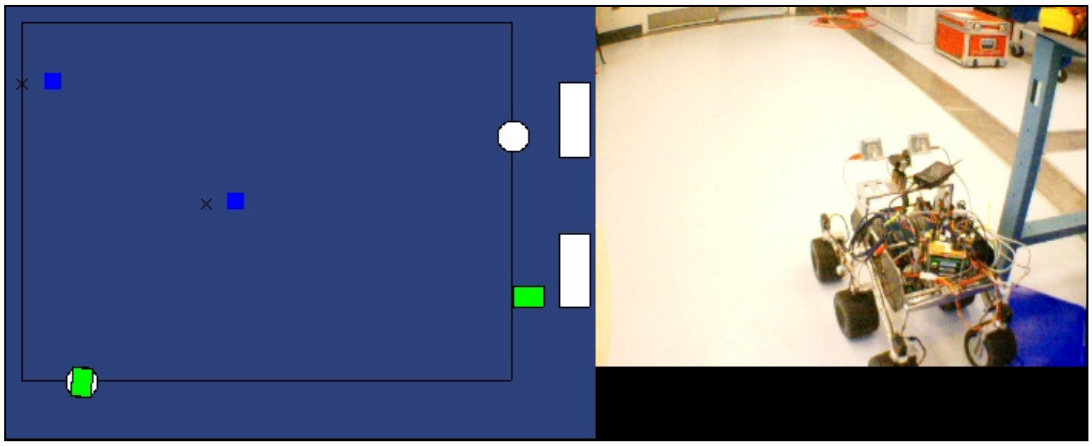
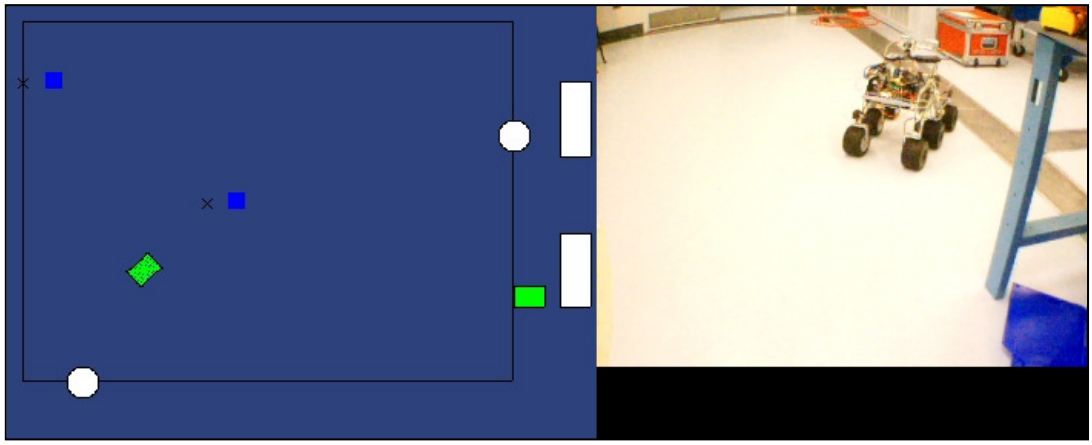


Figure 7.3: Rover path during execution compared to the planned path.

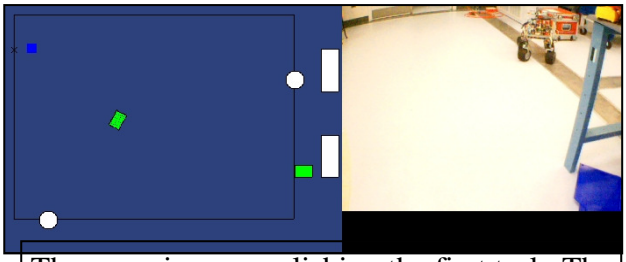
Figures 7.4-7.5 show information displayed by the planner/executive and the view from an external camera for the task execution and rescue operation. The two images are synchronized. The planner/executive process displays the position of the agents on the field based on the information it receives from them. Although the rover's position estimate may not be precise, the planner has no way to know the true position thus it displays position estimates as its best estimate or reality. This error is illustrated by the last image of the sequence. The coordination executive believes the planner has situated the rover in the same place as the astronaut, but by looking at the picture we can clearly see that the rover did not exactly return to its original position.



The rover is in its initial position and orientation; the astronaut, sitting at the table, is at the same location.

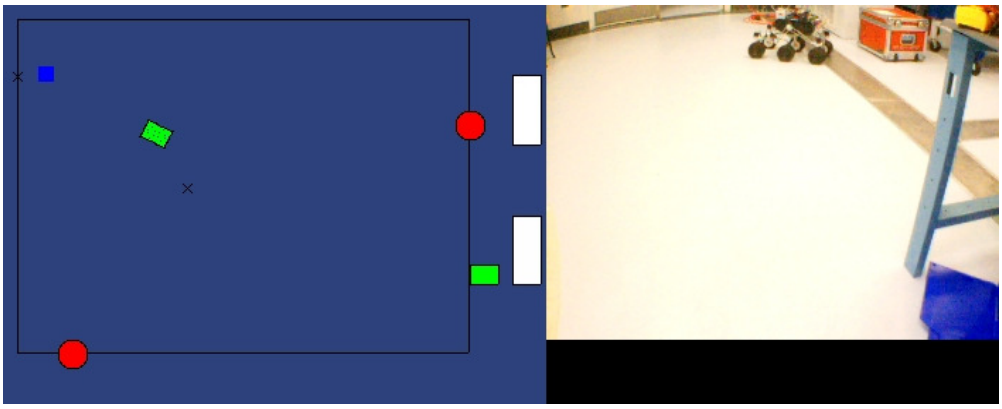


The rover is on its way to the first task.

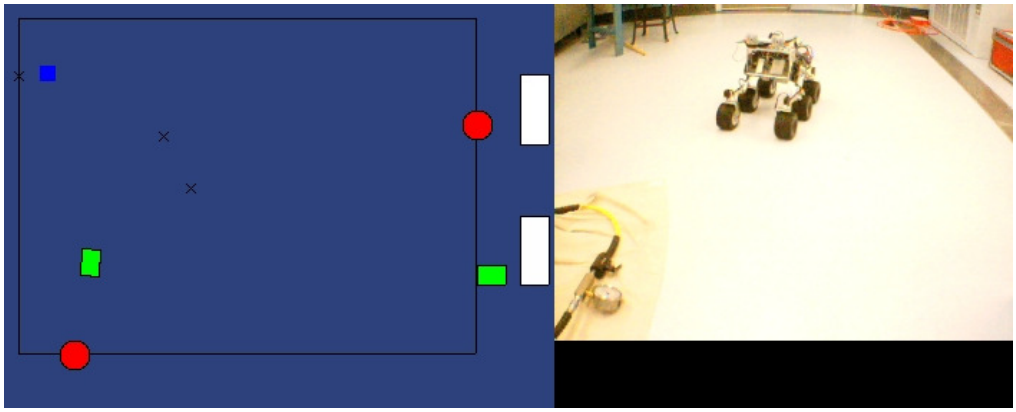


The rover is accomplishing the first task. The picture on the right was stored by the coordination executive and sent to the astronaut for approval.

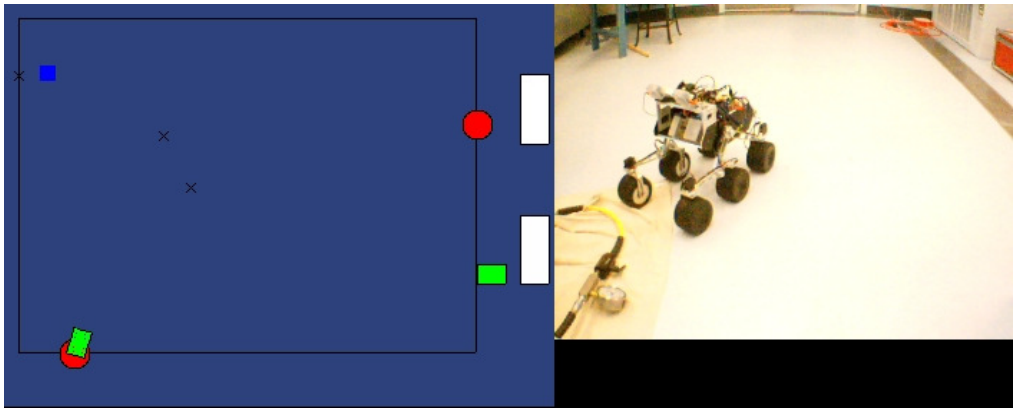
Figure 7.4: Execution snapshots from nominal plan execution.



While the rover is heading for the second task, the astronaut calls for help



The rover has received a rescue order and is now traveling to reach the astronaut



The rover has reached the astronaut's position for rescue support.

Figure 7.5: Execution snapshots for the emergency situation.

7.3 Outdoor Field Test

This section presents the results obtained during an outdoor field test. The setup of this test includes:

- Rough terrain: speed control was necessary to climb the terrain irregularities.
- Two sets of obstacles were placed on the path of the rover. The rover could see the first set of obstacles from the start position but the second set was out of its field of view.
- The goal position was set to be at 3.5 meters in front of the initial position.

Figure 7.6 presents the internal map of the rover at different times during the traverse. We can clearly the different obstacles in the last snapshot of the internal map. In the first image we see that the rover detects the four "obstacle" cups in front of it and chooses a path accordingly. A few seconds later, the rover perceives the second obstacle that appears to be intercepting the previously-planned trajectory. Consequently the rover re-computes a trajectory taking into account the terrain update.

Magnetometers can be used outdoors enabling more precise heading computation, but due to the terrain difference (mud, grass) more slippage can be expected. Nevertheless position computation was better in the outdoor environment.

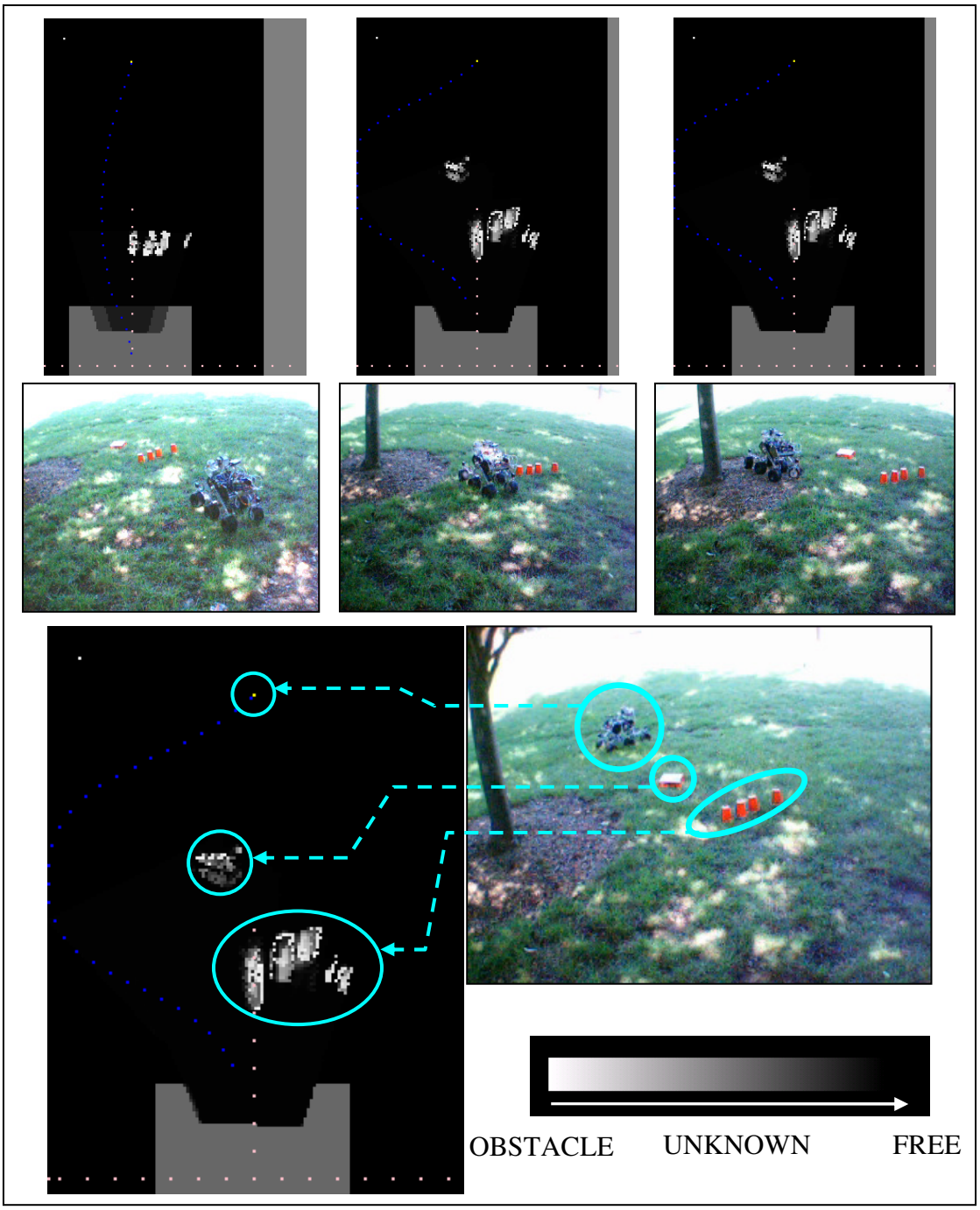


Figure 7.6: Outdoor field test: External view and internal terrain map.

7.3 Outdoor Scenario with the rover, astronaut, and one simulated collaborative tasks.

The last test was conducted to demonstrate the practical utility of this architecture as well as define its limitations. The test includes the lab rover which is capable of not only taking pictures but also storing rock samples. The astronaut is able to identify interesting rocks and grab them. Two additional simulated rovers are included during the test. Figure 7.7 show a picture displaying the experimental setup as well as the coordinate system.

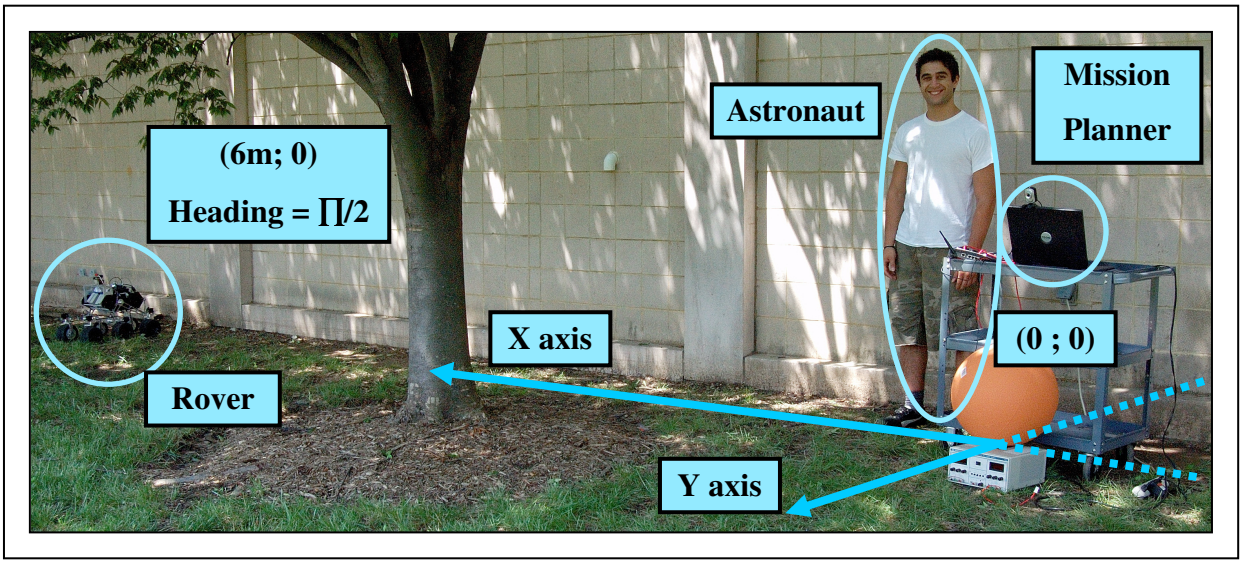


Figure 7.7: Experimental setup - initial position of the rover and the astronaut.

Seven tasks were required in this problem, including three "`Take Picture`", two "`Drill Rock`", one "`Water Test`", and one "`Rock Sampling`". The latter task is important because the competences of each agent were defined so that this task had to be performed collaboratively by the real astronaut and the real rover. Figure 7.8

shows the mission planner display including the computed plan. The planner was allowed 12 seconds for the plan computation. As the number of task was small, the branching factor was set to 8 so that no node was pruned during the search. Note the collaboration between the real astronaut and the real rover since they are synchronously sent to the same location to perform the rock sampling task.

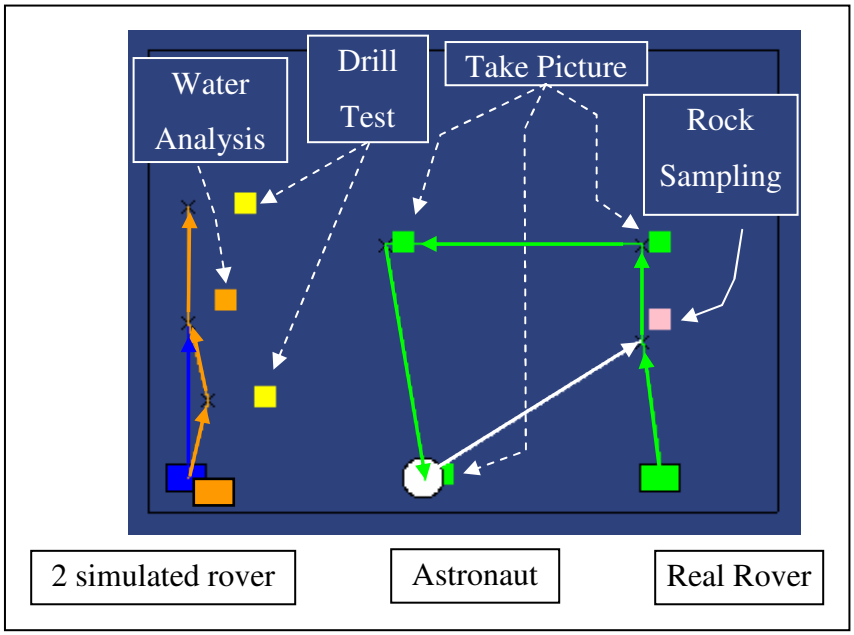


Figure 7.8: Collaborative Astronaut-Rover Plan.

As shown in the Figure above, simulated and real agents do not interfere during the plan. This is only due to our problem setup; consequently nothing would prevent the planner to plan collaboration action between a real and a simulated agent, although physical interaction might not be effective. For instance if the astronaut required a "`water sensor`" during execution of a task, the mission planner would have decided to send the simulated rover with this specific tool to the location of the

astronaut. The astronaut would then be alerted (through the GUI) that the rover was present.

For this specific test, the obstacle detection system was switch off and replaced with a beacon tracking tool. Since pose estimation error increases over a traversal, the real rover is more likely to arrive at an unexpected location after navigating for an extended period of time. As the goal in this test was to retrieve rock samples and bring them back to the base (the origin of the coordinate system), a visual target was used to guide the rover back to base. Using a color detection system similar to that described in Chapter 6 for obstacle detection we implemented a search and follow procedure that tracks an orange ball. In our tests, only the last waypoint (the base location) was marked with this colored beacon. When driving from the previous waypoint back to base the rover follows its planned route the first half of the traversal. It then stops, scans the environment by panning the cameras from left to right. The ball is then identified and tracked until the rover reaches the base waypoint (in practice, when the ball fills the rover's image plane).

The test was successful; the rover with the help of the human was able to sample rocks and accurately return to base. The Coordination Executive succeeds in monitoring the activity of both the simulated and the real rover. Figure 7.8 details the milestones achieved during plan execution.

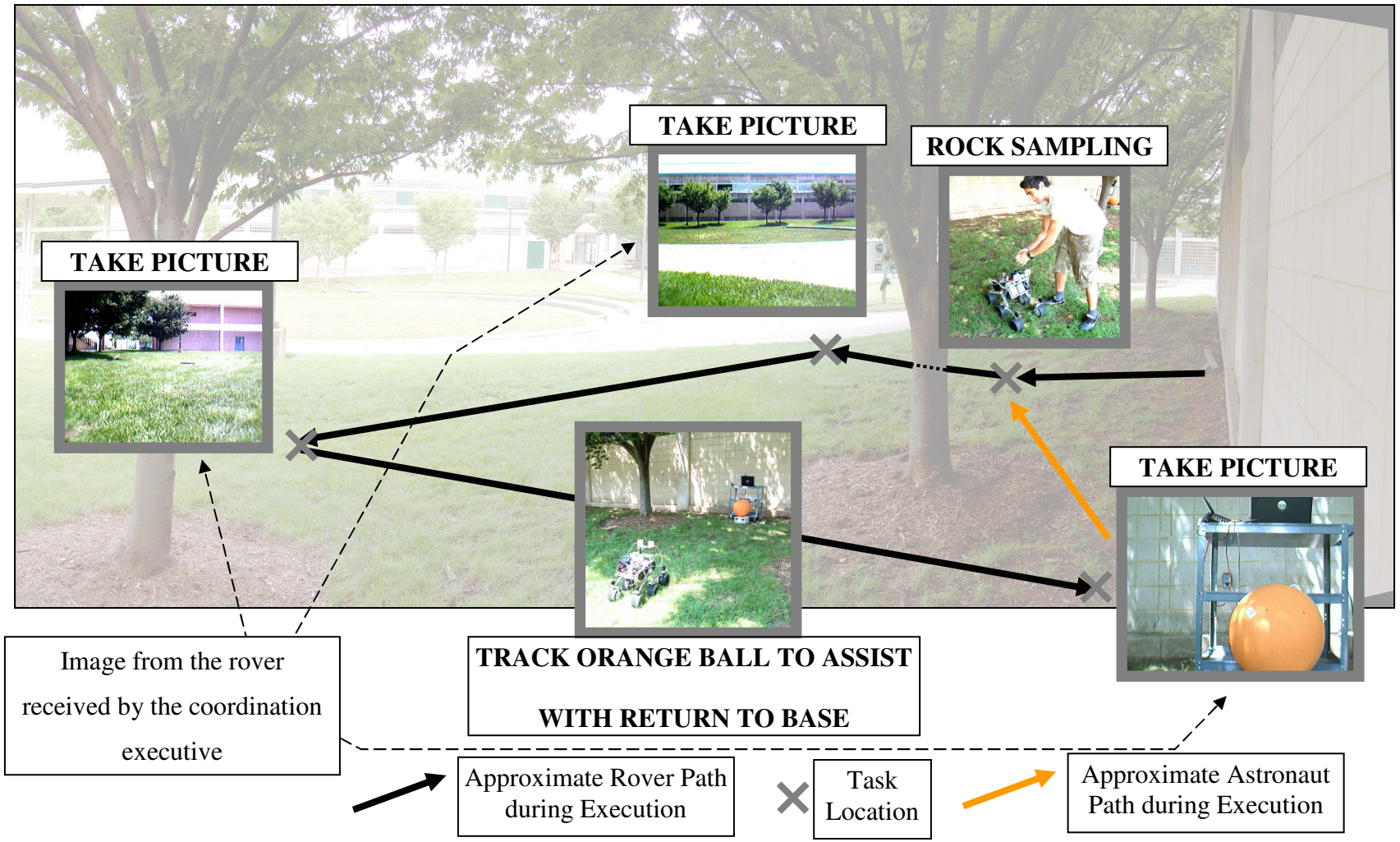


Figure 7.9: Execution Activities and Task Accomplishment Results

Figure 7.9 details the schedules of the four agents. As the human is not able to send position updates to the planner, its motion was simulated. Nevertheless the astronaut did perform the task and inform the planner when he finished placing rocks in the rover. The astronaut was expected to walk at 0.2 m/s by the mission planner but the simulated astronaut was "running" at 3m/s. (This was not intentional but a code mistake). Since the rover is significantly slower (≈0.08m/s), the astronaut has to wait for the rover, although this could be mitigated by the planner with numerous tasks and agents by making astronaut traversal/task executions much longer than those for the rovers. For the first three navigation actions, the real speed of the rover was close to the expected speed since the execution time almost matches the expected one. The last navigation action (return to base) was longer due to the stop command for visual detection of the base that was unmodeled in the planner.

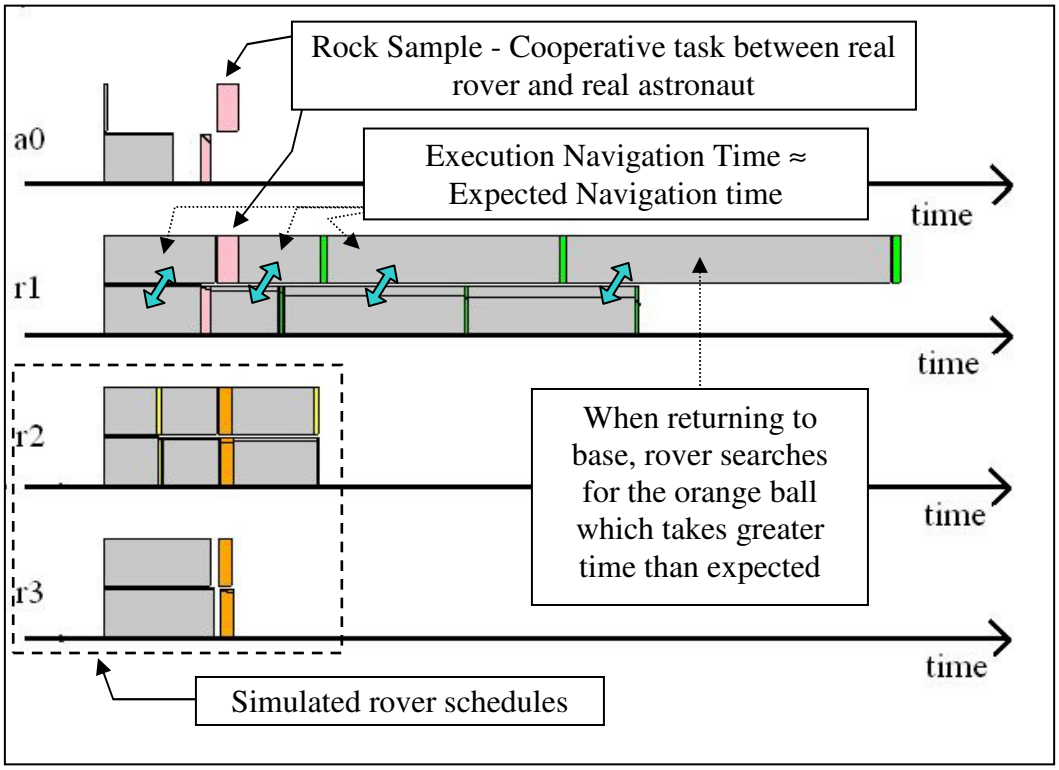


Figure 7.10: Computed schedule and actual execution times for tests with the real rover (r1) and astronaut (a0)

Chapter 8: Conclusion and Future Work

This thesis has described the design and implementation of a planning and execution system for astronaut-rover exploration teams tasked with collaboratively accomplishing planned tasks and maintaining safety and efficiency throughout. The three main research thrusts were the development of a flexible planning tool that can trade execution time for solution optimality when required, the implementation of a coordination executive to manage the multi-agent system, and the development and integration of rover hardware and onboard navigation software to enable real-world testing. The resulting system has been demonstrated capable of managing in real-time a team of astronauts and rover by scheduling optimized plans for each agent and by providing safe response to emergency or unexpected events generated by the environment or by directive from an astronaut.

8.1 Planning Tool

The implementation of the planning tool was the first step in this research. In particular it included the design of the agent and task model, as well as requiring adaptation for improved time response or better optimality (more tasks accomplished in a smaller amount of time). Simulation results have demonstrated the robustness of the planner over a wide range of situations and a large number of tasks. The

implementation enables new users to implement new tasks and new types of agent very easily. Computation time is controlled, ensuring not only quick response when needed but also deep search of an optimal solution. An additional branch control strategy using a closest-point strategy has also been included in order to restrain the search. Limiting the search space when little time is allowed for plan computation allows better plan quality while wide search space is used when a lot of time is available in order to explore more solution.

8.2 Coordination Executive

The second part of this research consists in the implementation of an execution module to control the agents' activity. This module is in charge of sending commands to the agents and managing status information as well as sensor data and emergency messages. Both simulation and field test results have proved that this module succeeds in managing the team of agents. Scenarios including emergency calls have been tested and consistently show appropriate and timely results.

8.3 Onboard Navigation Software

The design and implementation of the onboard navigation system was necessary for the field test. Because we were limited in time, basic algorithms were used in order to provide position determination, obstacle detection, mapping, and path planning capabilities. Despite limitations in the hardware and relatively simple navigation protocols, the rover is able to navigate in simple environments without impacting obstacles. The main source of error comes from position determination which practically limits our field tests to short traverses (from 0 to 20 meters).

8.4 Future Work

This work is requires significant extension to be applicable for real collaborative planetary surface exploration missions. This research has accomplished a first step by defining and implementing a model for astronaut-rover teams, but more work needs to be done to increase the adaptability of the system and the range of possible scenarios. The ultimate goal is to provide a system that can take full advantage of each agent's capabilities while minimizing unnecessary delays. We believe that humans and robots will soon be capable of accomplishing several tasks together and that such an architecture can improve the execution of large-scale scenarios beyond what is possible using strictly dynamically-computed (reactive) plans.

Future work should focus on the following issues:

- ***Execution monitoring***: Currently the planner/executor is able to collect information but does not react to off-nominal situations in an efficient manner. For instance, speeds of the agent are recorded but in the case where a rover is going slower than expected (as is frequently the case with our rover) no measure is taken, implying significant delay fpr cooperative activities during plan execution. More dynamic schedule adjustment should be implemented to automatically compensate when task execution speed is faster/slower than expected.
- Similarly, when an agent fails to execute a task the planner currently assumes the agent is no longer able to perform it and

computes a new plan. This model works when task execution failed due to a faulty sensor. In the general case, however, failure can be caused by other parameters and such drastic measures will not be desirable. Obviously, this part of future work is not limited to those two examples. Extensive field tests including human with different background, will lead to better knowledge of how the planner/executor system should best behave. Since this collaboration model is relatively new, no protocol has been adopted by the scientific community, implying that efforts must be done in order to first define and then implement a correct respond to those contingencies.

- ***Agent Model***: At this stage of the implementation, our model of an agent (for simulation purposes) is quite simple. More work needs to be done to expand and more realistically define agent capabilities and variance in the simulated execution of tasks. Parameters such as skills, knowledge, level of attention, willingness to be part of the team, are useful attributes to integrate within the planning problem. Human and rover internal parameters may evolve during the execution; consequently the previous "optimal" plan may become less efficient over time. A combination of improvements to agent models and planning rules/heuristics must be made for this architecture to appropriately model and

reason about the uniqueness of each human or robotic agent and his/her/its evolution during a mission.

- ***Implementation of a Task Execution Collaborative Dialogue*** is one of the biggest improvements needed for this architecture to work efficiently during larger-scale field trials. The current executive sends each agent involved in task accomplishment an "execution command". Even if the task is collaborative, no direct dialogue is now available between the agents since the goal in this first step was to efficiently place the correct set of agents at the right place at the right time. Given historical evidence with respect to NASA mission control, planetary surface exploration tasks will be well defined and rehearsed in advance. A protocol can be implemented for each type of activity and the different agents involved will be aware of what the others must accomplish. Communication between agents will make correct task accomplishment possible. For instance, when a rover has to store a rock during a "Rock Sample" task it must communicate with the astronaut to know if/when he succeeds in placing the rock in the sample container.
- ***Onboard Navigation Software***: As stated previously, the onboard navigation system of the rover was not the priority in this research. Simple algorithms were used which leaves opportunity to implement more sophisticated algorithms. Position determination is

obviously the first topic that requires attention since it influences greatly the efficiency and accuracy of the overall system. To mitigate position errors for at least outdoor testing, hardware such as GPS can be added to navigate accurately over long distances. SLAM algorithms can also be implemented in order to improve position determination, but map structure must be chosen carefully. Obstacle detection can be improved using stereo vision in order to map the terrain. By combining color and depth information more accurate results should be obtained. Path planning can also be improved by changing the way new configurations are generated during the search of the path. Consequently kinematic constraints can also be explicitly be taken into account during path planning, and new navigation modes can be implemented. For instance the rover currently drives like a car but it has the mechanical ability to drive using the "GO , STOP, TURN" navigation protocol (like current Mars Exploration Rovers). Combining those two modes of navigation will require work but will certainly improve rover navigation capabilities.

- ***Hardware Improvement***: The wheel drive system is the first hardware that needs upgrading. New motors with COTS encoders would substantially improve the controllability of the system, particularly since a motor intended as a drive motor will have more than four possible speeds.

Bibliography

[1] T.W. Fong and I. Nourbakhsh, "Interaction Challenges in Human-Robot Space Exploration," *Interactions*, Vol. 12, No. 2, March 2005, pp. 42-45.

[2] T. Estlin, D. Gaines, C. Chouinard, F. Fisher, R. Castano, M. Judd, R. Anderson, and I. Nesnas. "Enabling Autonomous Rover Science Through Dynamic Planning and Scheduling," *Proc. of IEEE Aerospace Conference*, Big Sky, Montana, March, 2005a (CL#05-0071).

[3] T. Estlin, D. Gaines, F. Fisher, and R. Castano. "Coordinating Multiple Rovers with Interdependent Science Objectives," *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems*, Utrecht, the Netherlands, July, 2005b (CL#03-1202).

[4] T. Fong, C. Thorpe, and C. Baur, "Advanced Interfaces for Vehicle Teleoperation: Collaborative Control, Sensor Fusion Displays, and Remote Driving Tools", *Autonomous Robots* 11(1), July 2001.

[5] Wagner, Thomas A., Garvey, Alan J. and Lesser, Victor R.. "Criteria Directed Task Scheduling". *Journal for Approximate Reasoning (Special Issue on Scheduling)*, Volume 19, Elsevier Science Inc., pp. 91-118. January 1998.

[6] D. J. Musliner, E. H. Durfee, and K. G. Shin,"World Modeling or the Dynamic Constructionof Real-Time Control Plans," *Artificial Intelligence*, 74(1) (1995) 83-127.

[7] M. Ghallab, D. Nau, and P. Traverso, "Automated Planning, theory and practice". *Morgan Kaufmann, 2004*. ISBN 1-55860-856-7

[8] R. Fikes and N.Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". *Artificial Intelligence*, 2(3-4): 189-208,1971

[9] J. Penberthy and D.S. Weld. "UCPOP: a sound, complete, partial order planner for ADL". In *Proceedings of the International Conference on Knowledge Representation and Reasoning (KR)*, pp 103-114,1992.

[10] A.L. Blum and M.L. Furst. "Fast Planning through planning graph analysis". *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI),* pp. 1636-1642, 1995.

[11] F. Bacchus and F. Kabanza, "Using temporal logics to express search control knowledge for planning," *Artificial Intelligence,*116:123-191, 2000.

[12] E. Anderson, C. Glass, and C.Potts. “Machine Scheduling”. In E.Aarts and J.Lenstra eds., *Local Search in Combinatorial Optimization*, pp. 361-414 Wiley, 1997.

[13] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian Williams. “Remote Agent: To Boldly Go Where No AI System Has Gone Before”, *Artificial Intelligence* 103(1-2):5-48, August 1998

[14] Douglas Bernard , Gregory Dorais , Edward Gamble, Bob Kanefsky, James Kurien, Guy K. Man, William Millar, Nicola Muscettola, Pandurang Nayak, Kanna Rajan, Nicolas Rouquette, Benjamin Smith, William Taylor, Yu-Wen Tung, “Spacecraft Autonomy Flight Experience: The DS1 Remote Agent Experiment”, *American Institute of Aeronautics and Astronautics Space Technology Conference and Exposition*, Albuquerque, NM, Sept. 28-30, 1999.

[15] Steve Chien, Rob Sherwood, Daniel Tran, Benjamin Cichy, Gregg Rabideau, Rebecca Castano, Ashley Davies, Rachel Lee, Dan Mandl, Stuart Frye, Bruce Trout, Jerry Hengemihle, Jeff D'Agostino, Seth Shulman, Stephen Ungar, Thomas Brakke, Darrell Boyer, Jim Van Gaasbeck, Ronald Greeley, Thomas Doggett, Victor Baker, James Dohm, Felipe Ip, "The EO-1 Autonomous Science Agent," *aamas*, pp. 420-427, *Third International Joint Conference on Autonomous Agents and Multiagent Systems* - Volume 1 (AAMAS'04), 2004.

[16] G. Rabideau, R. Knight, S. Chien, A. Fukunaga, A. Govindjee "Iterative Repair Planning for Spacecraft Operations in the ASPEN System", *International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS 1999).* Noordwijk, The Netherlands. June 1999

[17] F. Ygge, H. Akkermans, "Decentralized Markets versus Central Control: A Comparative Study", *Journal Of Artificial Intelligence Research* 11 (1999) 301-333.

[18] P. J. Gmytrasiewicz, P. Doshi, "A Framework for Sequential Planning Multi-Agent Settings", *Journal of Artificial Intelligence Research* 24, 2005 49-79.

[19] G. N. DeSouza and Avinash C.kak, "Vision For Mobile Robot Navigation: A survey". *IEEE Transactions On Pattern Analysis And Machine Intelligence, VOL.24, NO.2,* February 2002.

[20] C. Thorpe, M. Hebert, T. Kanade and S. Shafer, "Vision and Navigation for the Carnegie-Mellon University NavLab", Institute for Electrical and Electronics Engineers *Journal of Robotics and 44 Automation*, Vol 3 (2) March 1987.

[21] T.M. Jochem, D.A Pomerleau, and C.E Thorpe, "Vision-Based Neural Network Road and Intersection Detection and Traversal", In proceedings of the *Institute for Electrical and Electronics Engineers Conference on Intelligent Robots and Systems, pp 857-864,* Sept 1997.

[22] S.Lacroix, A. Mallet, D. Bonnafous, G. Bauzil, S. Fleury, M. Herrb, R. Chatila, "Autonomous Rover Navigation on Unknown Terrains : Function and Integration", *The International Journal of Robotics Research. Vol21, No10-11,* October-November 2002, pp 917-942. 2002 Sage Publication.

[23] Lindeberg, Tony "Edge detection and ridge detection with automatic scale selection", *International Journal of Computer Vision*, 30, 2, pp 117--154, 1998

[24] Kai-Ming Kiang, Richard Willgoss, "Distinctive Feature Analysis of Natural Landmarks as a Front end for SLAM Application", *2nd International Conference on Autonomous Robots and Agents*, Dec 13-15 2004, Palmerston North, New Zeland.

[25] S. Tabbone and D. Ziou. "On the Behavior of the Laplacian of Gaussian for Junction Models". *In Second Annual Joint Conference on Information Sciences*, pages 304--307, NC, USA, 1995.

[26] S. Thrun., "Robotic mapping: A survey". *In G. Lakemeyer and B. Nebel, editors, Exploring Artificial Intelligence in the New Millenium. Morgan Kaufmann,* 2002.

[27] A.Elfes. "Occupancy Grids: A probabilistic Framework for Robot Perception and Navigation". *PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University*, 1989.

[28] R.Chatila and J.-P Laumond. “Position referencing and consistent world modeling for mobile robots”. In *Proceeding of the 1985 IEEE International Conference on Robotics and Automation*, 1985.

[29] A. J. Davison, “Real-Time Simultaneous Localisation and Mapping with a Single Camera*"*. *In Proceedings of the ninth international Conference on Computer Vision ICCV'03*, Nice, France, October 2003. *Institute for Electrical and Electronics Engineers* Computer Society Press

[30] Sebastien Grange and Terrence W Fong and Charles Baur, “TLIB: A real-time computer vision library for HCI applications". *Digital Image Computing - Techniques and Applications Conference,* December 2003 publisher: IAPR/IEEE.

[31] Steven M.LaValle. “Planning algorithms”. Copyright 2006 Cambridge University Press ISBN-10: 0521862051 | ISBN-13: 9780521862059

[32] S. Sekhavat, F. Lamiraux, J.P. Laumond, G. Bauzil, A. Ferrand, “Motion planning and control for Hilare pulling a trailer: experimental issues” , *Institute for Electrical and Electronics Engineers. International Conference. on Robotics and Automation, (1997).*

[33] Andrew Moore, “Tutorial Slide On Robot Motion Planning”, *http://www.autonlab.org/tutorials/motion.html*

[34] Vytas SynSpiral, Mark B.Allan, Rodney Martin, Kevin R.Wheeler, “Modeling and Classifying Six-Dimensional Trajectories for Tele-operation Under Time Delay.” *American Association for Artificial Intelligence Spring Symposium 2006, To Boldly Go Where No Human-Robot Team Has Never Gone Before.*

[35] Breazeal, C. and Scassellati, B. (1999), "How to build robots that make friends and influence people". *To appear in IROS99, Kyonjiu, Korea*.

[36] T.W. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots," *Robotics and Autonomous Systems,* special issue on Socially Interactive Robots 42 (3-4), pp 143-166, 2003

[37] R. Urtasun and P. Fua, “Human Motion Models for Characterization and Recognition,” *In proceedings of the Automated Face and Gesture Recognition conference*, Seoul, Korea, May 2004. W*ork supported in part by the EU CogViSys project.*

[38].F. Cupillar and F. Bremond and M. Thonnat, “Tracking Group of People for Video Surveillance”. *In Proceedings of the 2nd European Workshop on Advanced Video-Based Surveillance System*. Universtity of Kingston , London, sept 2001.

[39] Rachid Alami, Aurélie Clodic, Vincent Montreuil, Emrah Akin Sibot, and Raja Chatilla, "Toward Human-Aware Robot Task Planning". *American Assoication for Artificial Intelligence. Sping Symposium 2006 "To Boldly Go Where No Human-Robot Team Has Gone Before*

[40] Y.Tsumaki, T. Goshozono, K. Abe, M. Uchiyama, R.Koeppe and G. Hirzinger. Verification of an Advanced Space Teleoperation System Using Internet", proceedings of the 2000 IEEE/RSJ International Conference on Robots and Systems.

[41] Sarah Hall, "Model Following Control Strategies and Human Iinterface Techniques for the Treatment of Time Delay During Teleoperation". *Phd Dissertion*, University Of Maryland, Space System Laboratories, 2004

[42] A. Elfes, J.M. Dolan, G.Podnar, S. Mau, M. Bergerman. "Safe and Efficient Robotic Space Exploration with Tele-Supervised Autonomous Robots", *American Association for Artificial Intelligence Spring Symposium 2006 To Boldly Go Where No Human-Robot Team Has Never Gone Before.*

[43] F. Heger, L. Hiatt, B.P. Sellner, R.Simmons, and S. Singh, *"Results in Sliding Autonomy for Multi-Robot Spatial Assembly." 8th International Symposium on*

Artificial Intelligence, Robotics and Automation in Space, Munich Germany, Sptember 2005.

[44] L. Pedersen, W.J. Clancey, Maarten Sierhuis, N.Muscettola, D.E. Smith, D.Lees, K.Rajan, S. Ramakrioshnan, P.Tompkins, A Vera, T. Dayton. "Field Demonstration of Surface Human-Robotic Exploration Activity". *American Association for Artificial Intelligence Spring Symposium 2006 To Boldly Go Where No Human-Robot Team Has Never Gone Before.*

[45] D. S. Johnson, L. A. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra, eds., pp. 215-310. London: John Wiley and Sons, 1997.

[46]A.P. Dempster, A.N. Laird, and D.B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". *Journal of the Royal Statistical Society, Series B, 39(1):1–38, 1977*.

[47] Funahashi, K., "On the Approximate Realization of Continuous Mappings by Neural Networks", *Neural Networks*, 2, 183-192, 1989.

[48] Bartels, R. H.; Beatty, J. C.; and Barsky, B. A. "Hermite and Cubic Spline Interpolation." Ch. 3 in *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*. San Francisco, CA: Morgan Kaufmann, pp. 9-17, 1998.