NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

IMPLEMENTATION OF AUTONOMOUS NAVIGATION AND MAPPING USING A LASER LINE SCANNER ON A TACTICAL UNMANNED AERIAL VEHICLE

by

Mejdi Ben Ardhaoui

December 2011

Thesis Advisor:	Timothy H. Chung
Second Reader:	Duane Davis

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
16–12–2011	Master's Thesis	

4. TITLE AND SUBTITLE

Implementation of Autonomous Navigation and Mapping using a Laser Line Scanner on a Tactical Unmanned Aerial Vehicle

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Mejdi Ben Ardhaoui

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of the Navy

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

13. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number. NA.

14. ABSTRACT

The objective of this thesis is to investigate greater levels of autonomy in unmanned vehicles. This is accomplished by reviewing past literature about the developing of components of software architecture that are necessary for unmanned systems to achieve greater autonomy.

The thesis presents implementation studies of existing sensor-based robotic navigation and mapping algorithms in both software and hardware, including a laser range finder, on a quadrotor unmanned aerial vehicle platform for real-time obstacle detection and avoidance.

This effort is intended to lay the groundwork to begin critical evaluation of the strengths and weaknesses of the MOOS-IVP autonomy architecture and provide insight into what is necessary to achieve greater levels of intelligent autonomy in current and future unmanned systems.

15. SUBJECT TERMS

Artificial Intelligence, Obstacle Avoidance, Potential Field, Occupancy Grid, Quadrotor, MOOS-IvP

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
Unclassified	Unclassified	Unclassified	UU	85	

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

IMPLEMENTATION OF AUTONOMOUS NAVIGATION AND MAPPING USING A LASER LINE SCANNER ON A TACTICAL UNMANNED AERIAL VEHICLE

Mejdi Ben Ardhaoui
Captain, Tunisian Army
B.S. Management, Tunisian Military Academy, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 2011

Author: Mejdi Ben Ardhaoui

Approved by: Timothy H. Chung
Thesis Advisor

Duane Davis
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The objective of this thesis is to investigate greater levels of autonomy in unmanned vehicles. This is accomplished by reviewing past literature about the developing of components of software architecture that are necessary for unmanned systems to achieve greater autonomy.
The thesis presents implementation studies of existing sensor-based robotic navigation and mapping algorithms in both software and hardware, including a laser range finder, on a quadrotor unmanned aerial vehicle platform for real-time obstacle detection and avoidance.
This effort is intended to lay the groundwork to begin critical evaluation of the strengths and weaknesses of the MOOS-IVP autonomy architecture and provide insight into what is necessary to achieve greater levels of intelligent autonomy in current and future unmanned systems.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

Acknowledgements

I would like to thank my advisor Timothy Chung for his time and patience as I worked through this thesis. His guidance and insights were invaluable. I would also like to thank Duane Davis who greatly helped me with fixing my code.

Also I would like to express my sincere appreciation to Aurelio Monarrez who helped me working with the quadrotor platform.

Most of all I need to thank my Daughters, Sayma and Eya, whose love and support made this possible.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1:
Introduction

1.1 Introduction

In this thesis work we are interested in implementing and demonstrating appropriate path-planning and obstacle-avoidance algorithms using a unmanned aerial vehicle (UAV) platform in order to accomplish a simple military-type scenario. The UAV is tasked with a mission to autonomously transit from a known launch site to a given target location. Onboard sensors play the most important role in providing mission-environment information to avoid obstacles encountered during the mission.

The platform of interest is the quadrotor UAV, which is a rotorcraft equipped with four powered rotors laid up symmetrically around its center. Quadrotors are considered an ideal solution for many tactical applications. Because of the low platform footprint and weight, relatively simple mechanical design and control, hover capability and slow speeds, this platform became very popular to fit several applications needed for both military and civilian missions, especially for surveillance and indoor navigation.

UAVs in general have gained popularity for surveillance and reconnaissance missions to provide situational awareness to ground-based military units and to use onboard sensors to identify or intercept objects which represent a threat to the safety of military and civilian personnel. The majority of UAVs are controlled by human operators from ground control stations (GCSs) using radio frequency (RF). Moreover, UAVs also have the capability to provide precise and real-time information to an on-scene operator with high resolution images from cameras and other capable sensors. This can help inform a commander in making appropriate safety decisions for his unit.

1.1.1 Motivation

During the last several years, the Tunisian Army has identified border security as a high priority. The diverse terrain features of the western border separating Tunisia and Algeria, for example, significantly increase the difficulty of conducting patrolling missions using conventional approaches. Nowadays, borders are physically protected by mounted and dismounted border patrol agents, which presents both a strain on limited resources as well as a risk to personnel.

Despite the large presence of national agents, we still fear that terrorists, drug smugglers, and arms dealers can exploit existing gaps and vulnerabilities of the borders. For example, in 2007, Tunisia announced that it had killed twelve and captured fifteen Islamic extremists, six of whom had arrived from Algeria [1].

The desire to use UAVs is fueled by the need to increase situational awareness and remote sensing capabilities for such border protection missions. UAVs have attracted major defense and security research interest during the last decade. When there is a need to acquire aerial surveillance and intelligence in challenging environments, such as dense forests, tunnels or caves, or when the situation makes using a manned aircraft impractical, UAVs presently are the best solution for keeping human life and expensive aircraft out of risk. UAVs, equipped with a camera and additional sensors for navigation, can be hastily deployed to gather information, especially where signals from satellites are either unavailable (e.g., area is not covered or cost is prohibitive) or unusable due to low signal strength.

In this thesis, we hope to develop a better understanding of the new techniques used in surveillance and reconnaissance. UAVs offer major advantages over traditional methods, not only in terms of performance, but also increasingly because of their lower relative cost. UAVs have been used by many countries, and it has been shown that they can provide more precise information than other stand-off intelligence sources. In war, most decisions are made in a short frame of time. The accuracy of information gathered is very important in decision making. No one wants to kill innocent people because of lack of information, while on the other hand, no one wants to take the risk of bypassing an opportunity to stop an enemy.

1.1.2 Research Questions

As can be seen from various reported experiments, onboard sensors represent the quadrotor's source for information gathered from the environment. The quadrotor needs this information to navigate and perform its mission. The first question one must ask is: what are the minimum quality requirements that one must impose on sensors to obtain an accurate navigation system?

We must also consider cost. If UAVs are believed to be useful for border security, what are the potential cost implications of UAVs for future Ministry of Defense budgets?

The last question concerns the feasibility of using different autonomy software architectures. This research evaluates the MOOS-IvP architecture, and attempts to determine if this architecture provides the best solution for developing and implementing the algorithms needed for

navigation and control.

Questions 1 and 2 will be answered in Chapter 3. Question 3 is addressed in the experiments with results presented in Chapter 4, and the answer will be discussed in Chapter 5.

1.2 Background

1.2.1 Related Work

Signficant work has been done developing various control methodologies for the quadrotor aerial robotic platform. A research group at Stanford University, the Stanford Testbed of Autonomous Rotorcraft for Multi Agent Control (STARMAC), is using a quadrotor to study new multi-agent algorithms to avoid collision and obstacles. Others, including Hanford et al. [2], have tried to build low-cost experimental testbeds that use quadrotors for a variety of research interests.

Numerous efforts have investigated control of quadrotor platforms, ranging from simple trajectories to dynamically challenging acrobatic maneuvers, using various control methodologies. For example, Bouabdallah et al. [3] compared the stability performance of quadrotors for proportional-integral-derivative (PID) and linear quadratic (LQ) techniques. Chen et al. [4] derived combined model-based predictive controllers for quadrotors, one based on a linearization feedback controller concept and the other based on the back-stepping controller. Johnson and DeBitetto [5] present a guidance system which generates position, heading, and velocity commands based on the current state of the UAV, as reported by the navigation system and a waypoint list. The guidance system commands a straight-line course between waypoints.

Higher level autonomy for unmanned systems is also an active area of research for aerial platforms, although much of the previous works focus on helicopter-based platforms. Koo and his colleagues [6, 7] developed a hierarchical hybrid control structure to govern a UAV in its mission to search for, investigate, and locate objects in an unknown environment. The hierarchical architecture consists of four layers: the strategic, tactical, and trajectory planners, as well as the regulation layer. The strategic planner coordinates missions with cooperating UAVs and creates a sequence of waypoints for the tactical planner. The tactical planner can land, search an area, approach a waypoint, avoid collision, and inspect objects on the ground. Additionally, it overrules the strategic planner for safety reasons. The trajectory planner decomposes commanded behavior into a sequence of primitive maneuvers, guarantees safe and smooth transitions, and assigns an appropriate sequence of height commands to the regulation layer to execute.

Kottmann [8] designed a trajectory generator which converts flight maneuvers on an abstract level into waypoints that must be contained in the trajectory. In a further step of refinement, the generator splits the routes between waypoints into several segments of constant velocity or constant acceleration. After this, the ground station uploads the segments to the helicopter onboard system for execution.

Lai et al. [9] present a hierarchical flight control system containing three tiers: the navigation, path, and stabilizing layers. The navigation manager controls the mission and commands a series of locations to the path controller. When the path controller reaches a location, it switches its destination to the next location. The stabilizing controller applies the nominal values from the path controller for both attitude and height to the helicopter.

Kim et al. [10, 11] designed a hierarchical flight control system which allows a group of UAVs and Unmanned Ground Vehicles (UGVs) to cooperate. The flight management system employs three strategy planners to solve specific missions. The first strategy planner implements the simple waypoint navigation of a predefined course. The second strategy planner operates a pursuit-evasion game where pursuers capture evaders in a given grid field. The third strategy planner executes flight-speed position tracking, where an UAV pursues a moving ground vehicle.

Williams [12] developed his own open source project called *Vicacopter*, incorporating two ways of autonomous control: waypoint following and programmable missions. For waypoint following, the ground station uploads a sequence of waypoints to the helicopter. After the helicopter reaches a location, it heads for the next location. For programmable missions, the ground station uploads a series of control commands to the helicopter for execution.

The MOOS-IVP portion of this work expands previous work done in Unmanned Underwater Vehicles (UUVs). In their seminal paper entitled *Nested Autonomy with MOOS-IvP for Interactive Ocean Observatories*, Michael Benjamin and his co-authors describe the relation between the MOOSDB (Mission Oriented Operating Suite-Database) and the behavior-based autonomy system which implemented by the IVP Helm (Interval Programming) [13].

1.2.2 Different Type of Quadrotors

The first quadrotor vehicle was built in 1907 by two French scientists [14]. The challenge was to make the machine lift itself off the ground using its own power and pilot itself.

In 1920, Eienne Oemichen built a quadrotor machine with eight additional rotors for control and propulsion. To offset the overweight, he used a hydrogen balloon to provide additional lift and stability. In 1924 he made a successful flight without the balloon [14].

Conventional UAVs

Conventional UAVs are the most used in military missions, some of which are illustrated in Figure 1.1. These unmanned systems have been used in many missions lately, and have demonstrated that they can be very reliable and more efficient than other manned aircraft.

Figure 1.1: Conventional UAV

Non-Conventional UAVs

Even though non-conventional UAV systems are not used in current military contexts, they still have a role to play in many current scientific research efforts. Their low speed compared to conventional UAV represent an handicap for any military mission, however, nano-UAV's have the potential to be useful in spying missions.

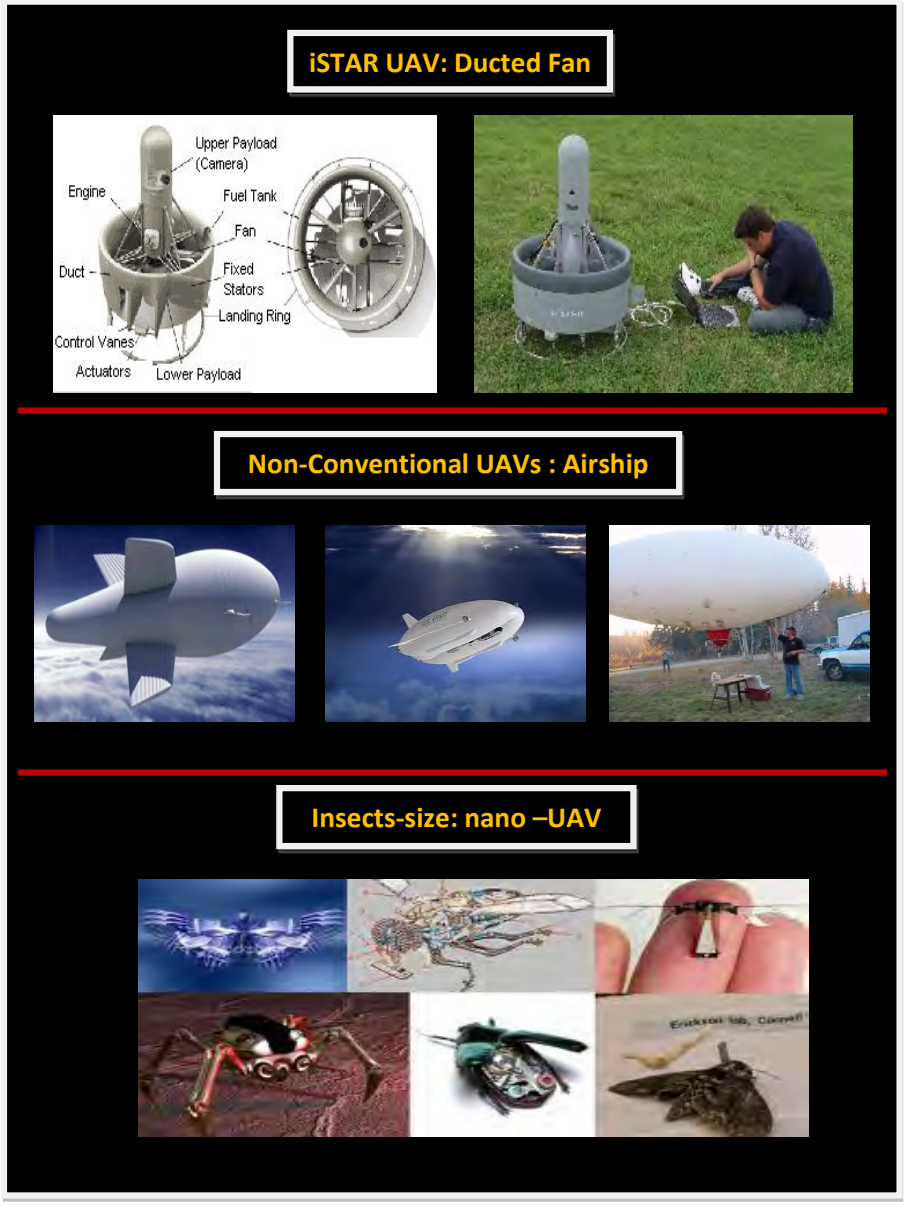

Figure 1.2: Non-Conventional UAV

1.2.3 Why The Quadrotor?

As previously mentioned, the usefulness of unmanned vehicles stems from the ability to avoid placing human life in dangerous situations. Modern UAVs are controlled by both autopilots and human controllers in ground stations. These characteristics allow them to fly either under their own control or to be remotely operated. Rotor-wing aircraft grant some major advantages over fixed-wing aircraft because of their ability to be launched without requiring a runway and also their ability to hover in a given place. A recent result reported in [15] is a 13.6 cm helicopter capable of hovering for three minutes, demonstrating the miniaturization of unmanned aerial vehicles. Compared to fixed-wing aircraft, the quadrotor can be commanded to go anywhere and examine targets in unlimited detail. That is why this kind of UAV is considered ideal for many information gathering applications in military contexts.

The primary concession of quadrotors is their relatively short endurance. Operational time is restricted to less than half an hour and, because of payload limitations, the option to mount a spare power supply is not available. Although the simplicity of the mechanical design is considered to be an advantage, especially for maintenance, it has a drawback in terms of on-station time and mission duration.

1.2.4 Contribution

This thesis adds to work done previously in developing obstacle avoidance and map-building algorithms using MOOS-IvP modules. MOOS-IvP is an open source software architecture for autonomous systems that has been used by many students conducting research with unmanned underwater vehicles [13]. Development of algorithms for application to different UAV platforms leveraging existing controller and/or driver software will enhance the usability of these unmanned systems.

1.3 Outline of Thesis

The thesis consists of five chapters. The first chapter covers the main idea behind this thesis, the purpose of the study, and background information about UAVs and their use. In the second chapter, a description of the goal is given, and then an overview of obstacle avoidance and how to model map building using grid occupancy is also discussed. In Chapter III, an overview of the hardware used for this project is described in detail, and an introduction to MOOS-IvP and its tools is presented.

The fourth chapter covers details on design of the experiments, the results obtained from both

simulation and real-world implementations, and techniques used to perform the task. An algorithm for obstacles avoidance and map building is proposed. The last chapter summarizes the results, pointing out possible future work.

CHAPTER 2:
Formulation

2.1 Mission Description

There are several well known approaches for obstacle avoidance and path planning. Most of the proposed techniques are suitable for outdoor operation, while only a few of them have been designed for indoor environments. For our project, we are more interested in developing algorithms for autonomously flying a quadrotor inside the Naval Postgraduate School's Center for Autonomous Systems (Figure 2.1). This will be done by adapting techniques which have been successfully tested on Underwater Unmanned Vehicles.

Figure 2.1: NPS Center for Autonomous Vehicle Research (CAVR)

Obstacle avoidance, which refers to the methodologies of shaping the robot's path to overcome unexpected obstacles, is one of the most challenging issues to the successful application of robot systems.

To avoid obstacles during a given mission, we first need to know where these obstacles are located, and we then avoid them by combining sensorial data and motion control strategies. A

common strategy used for such problems is to build a map of the robot's environment using an Occupancy Grid algorithm, and then apply a potential field technique to drive the quadrotor from a starting point to the goal. These two approaches will be discussed in the following sections.

2.2 Mapping Using Occupancy Grids

2.2.1 Basic Sensor Model

The occupancy grid known as Evidence Grid-Based Map building was introduced by Moravec and Elfes [16], [17]. A two dimensional grid is so far the most used in the field of mobile robotics, and each square of the grid is called a cell. Each individual cell represents the status of the working space, which can either be empty or occupied by the obstacle. Such maps are very useful for robotic applications. Robots need to know their environment in order to perform a specific task, such as navigation, path planning, localization and collision avoidance [18].

For a given range-finding sensor, such as a laser line scanner, with maximum range R and sensor beam half-width β, the model can be decomposed into a number of sectors labeled I-IV, as illustrated in Figure 2.2 and presented by [19]. .

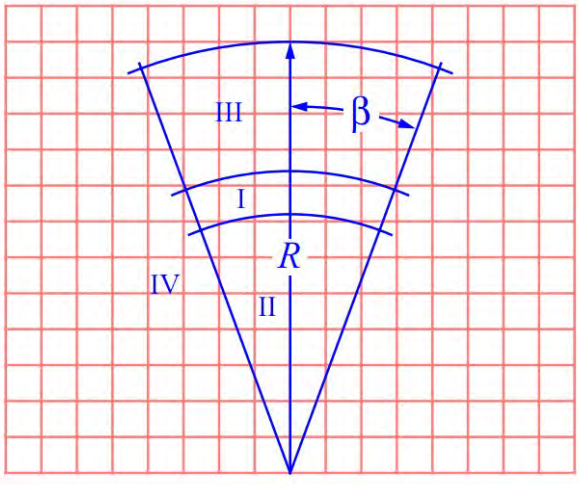

Figure 2.2: The regions of space observed by an ultrasonic sensor. From: [19]

A model for the occupancy of cells in a grid is Region I is the region associated with the laser reading. Region II represents an empty area where nothing is detected by the range reading. Region III is the area covered by the laser beam, but remain unknown whether it is occupied

or not because of the occlusion described in chapter IV. Finally, Region IV is outside the beam area and does not represent any interest.

For a given measurement reading a range of s, the given sensor model provides probabilistic expressions for reasoning about the occupancy of each cell in the occupancy map. As illustrated in Figure 2.3, the relevant parameters for reasoning about the highlighted (black outlined) cell include r, which is the distance of the grid element from the sensor location, and α, the angle of the grid element relative to the sensor center beam.

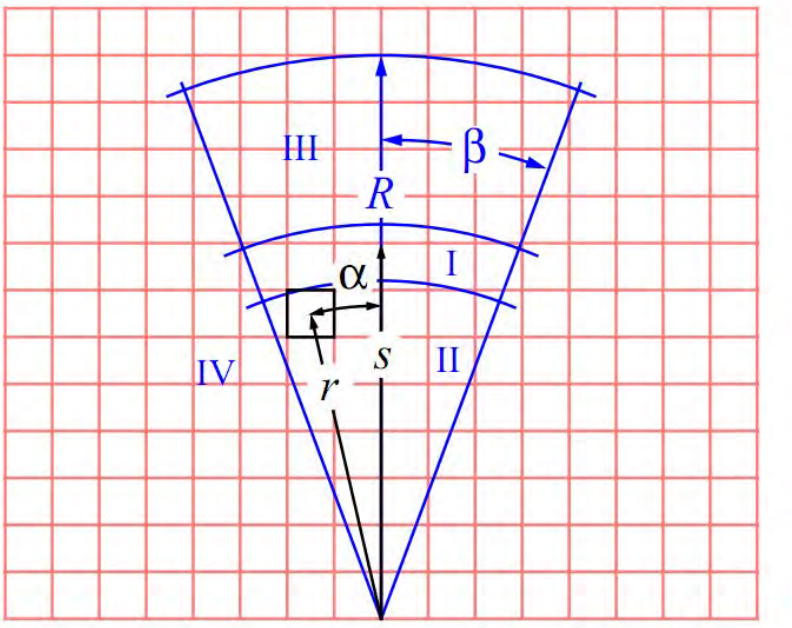

Figure 2.3: Obstacle Represented in a Grid

For a measurement s that fall in region I, the likelihood that the measurement was due to an actual obstacle present at that range and its complement can be computed by:

$$P(s|\text{Occupied}) = \frac{(\frac{R-r}{R}) + (\frac{\beta-\alpha}{\beta})\beta}{2} * Max_{\text{occupied}}$$
$$P(s|\text{Empty}) = 1 - P(s|\text{Occupied})$$

where Max_{occupied} is due to the assumption that a reading of "Occupied" is never completely reliable [19].

Similarly, for a range measurement in region II, the probabilities can be computed according to

the model:

$$P(s|\text{Empty}) = \frac{(\frac{R-r}{R}) + (\frac{\beta-\alpha}{\beta})\beta}{2}$$
$$P(s|\text{Occupied}) = 1 - P(s|\text{Empty})$$

Because of the occlusion, we assume that the probability of occupancy in region III is 50% occupied and 50% unoccupied. Region IV is not of interest.

2.2.2 Theoretical Review of Bayesian Rules

Let $H = \{\text{Occupied}, \text{Empty}\}$ be a binary random variable. The probability of H is $0 <= P(H) <= 1$. If $P(H)$ is known, its complement, $P(notH)$, also denoted $P(\neg H)$, can be computed simply by $P(\neg H) = 1 - P(H)$.

These probabilities only provide us with prior information, and they are assumed independent from the sensor reading. For the robot we need a function to compute the probability of elements in the map (i.e., at location $[i][j]$ in the grid) based on the sensor reading(s) and whether the state of this element is either "Occupied" or "Empty." Then we can compute $P(H|s)$, which represents the probability that the hypothesis (either "Occupied" or "Empty") given a particular sensor reading(s).

Note, however, the sensor model provides only the likelihood probability $P(s|H)$. To derive the posterior probability $P(H|s)$, we use Bayes' rule [20]. As a review, if we consider A and B as two events, such that $P(A) > 0$ and $P(B) > 0$, then $P(A|B)$ is the conditional probability of event A given event B, such that:

$$P(A|B) = \frac{P(A \bigcap B)}{P(B)} \tag{2.1}$$

where $P(A \bigcap B)$ is the probability of intersection set of A and B, also known as the joint probability of A and B.

From Equation 2.1 we can write:

$$P(A \bigcap B) = P(A|B)P(B) = P(B|A)P(A)$$

The law of total probability is given by :

$$P(A) = \sum_{i \in I} P(B_i)P(A|B_i) \tag{2.2}$$

where $P(B_i) > 0$, and $B_i \cap B_j = \emptyset$ for $(i \neq j, i, j \in I \subseteq \{1,2,3,...,n\})$, where n is the cardinality of set I.

Recalling the set union identity, $A = \bigcup_{i \in I}(A \cup B)$, the total probability (Equation 2.2) generalized from the equations above can be expressed as:

$$P(B_j|A) = \frac{P(A|B_j)P(B_j)}{\sum_{i \in I} P(B_i)P(A|B_i)} \tag{2.3}$$

where $P(B_i|A)$ is called the posterior probability, $P(A|B_j)$ is called the likelihood, and $P(B_j)$ is the prior probability. The denominator term, $\sum_{i \in I} P(B_i)P(A|B_i)$ is also given a name, namely the evidence.

Thus, it is now possible to compute the posterior probability, that is, $P(H|s)$, using the sensor likelihood probability, $P(s|H)$, via Bayes' rule:

$$P(H|s) = \frac{P(s|H)P(H)}{P(H)P(s|H) + P(\neg H)P(s|\neg H)}$$

For example, if the hypothesis H in question is "Occupied," the above expression becomes:

$$P(\text{Occupied}|s) = \frac{P(s|\text{Occupied})P(\text{Occupied})}{P(\text{Occupied})P(s|\text{Occupied}) + P(\text{Empty})P(s|\text{Empty})}.$$

2.2.3 Updating the Occupancy Map with Bayes' Rule

Initially, all the grid map cells have to be initialized with a probability of 0.5 to reflect uncertainty in occupancy of the cells. When we get an observation from the sensor, the new probabilities are computed according to Bayes' Rule, and the prior probability $P(H)$ is replaced with the new posterior probability value. Values above 0.5 indicate that cells are probably occupied; otherwise cells are probably empty. Every time the sensor platform moves, the grid cells within the occupancy map are continuously updated with values computed using the new measurements from the sensor.

Even though the Bayesian update approach is considered to be more accurate than perhaps its more naïve binary sensor models, the computational cost for map building is very high, which is worsened by a higher frequency of updating [21]. To avoid such problem, it is recommended not to do the computation on the onboard PC.

2.3 Potential Field Navigation

2.3.1 General structure

The artificial potential fields for a robot were first proposed by Khatib [22] and has been studied extensively since its inception. In this formulation where the mobile robot seeks to navigate to a goal location, the robot is considered a particle in the configuration space which is subjected to an artificial potential field $U(\boldsymbol{q})$, where $\boldsymbol{q}$ is the (vector) state of the robot (typically, for a mobile point robot in the plane, $\boldsymbol{q} = (x,y)^T$). At each iteration, the artificial force $F(\boldsymbol{q})$ induced by the potential field indicates the most promising direction. The following equations follow the notation and construction presented in [23].

The potential field is defined as the sum of an attractive potential U_{att} pushing the robot toward the goal and a repulsive potential U_{rep} pushing the robot away from obstacles.

$$U(\boldsymbol{q}) = U_{att}(\boldsymbol{q}) + U_{rep}(\boldsymbol{q}). \tag{2.4}$$

Consider the force vector experienced by the particle to be the negative gradient (i.e., direction of steepest descent) of the potential field.

$$\begin{aligned} \boldsymbol{F}(q) = -\boldsymbol{\nabla} U(q) \quad &= \quad -\boldsymbol{\nabla} U_{att} - \boldsymbol{\nabla} U_{rep} \\ &= \left(\frac{\partial U}{\partial x}, \frac{\partial U}{\partial y} \right)^T \Bigg|_{\boldsymbol{q}}, \quad \text{for } \mathbf{q} \in \mathbb{R}^2, \end{aligned}$$

where $\boldsymbol{\nabla} U$is the gradient vector of U evaluated at the robot position. Equivalently, the force is defined as the sum of the two attractive and repulsive force vectors, F_{att} and F_{rep}, respectively, i.e.,

$$\boldsymbol{F}(\boldsymbol{q}) = \boldsymbol{F}_{att}(\boldsymbol{q}) + \boldsymbol{F}_{rep}(\boldsymbol{q}). \tag{2.5}$$

2.3.2 Attractive Potential Field

The attractive field may be simply defined as a parabolic form:

$$U_{att}(q) = \frac{1}{2}\zeta\rho^2 goal(q) \tag{2.6}$$

with ζ a positive scalar and ρ goal($\boldsymbol{q}$) is the distance from $\boldsymbol{q}$ to goal.

The function U_{att} is assumed nonnegative and reaches its minimum at $\boldsymbol{q}_{goal}$ where $\boldsymbol{\nabla} U_{att}(\boldsymbol{q}_{goal}) = 0$. Since $\boldsymbol{F}_{att}$ is differentiable everywhere in the configuration space, the attractive force can be defined as the following:

$$\boldsymbol{F}_{att}(\boldsymbol{q}) = -\boldsymbol{\nabla} U_{att(q)} = -\zeta\rho goal(q)\boldsymbol{\nabla}\rho goal(q) = -\zeta(\boldsymbol{q} - \boldsymbol{q}_{goal})$$

Another possible form of U_{att} is conical:

$$U_{att}(q) = \zeta\rho goal(q) \tag{2.7}$$

In this case, we will have then a force of type:

$$U_{att}(q) = -\zeta \bigtriangledown \rho goal(q) = -\zeta\frac{q - qgoal}{\|q - qgoal\|} \tag{2.8}$$

The advantage of the conical shape is that the force is constant on the space configuration. Moreover, it does not tend to infinity when the robot moves away from the goal. However, it is not null at the goal location, $\boldsymbol{q}_{goal}$.

The most commonly used form of potential field function proposed by Khatib, which is known by the gradient form, is defined as the following:

$$U_{att}(\boldsymbol{q}) = \frac{1}{2}\zeta d^2 \tag{2.9}$$

where $d = |\boldsymbol{q} - \boldsymbol{q}_{att}|$. Recall that $\boldsymbol{q}$ is the current position of the robot, $\boldsymbol{q}_{att}$ is the position of an attraction point (e.g., the goal), and ζ is an adjustable scaling constant. The corresponding force function is :

$$\boldsymbol{F}_{att}(\boldsymbol{q}) = -\boldsymbol{\nabla} U_{att} = -\zeta(\boldsymbol{q} - \boldsymbol{q}_0) \tag{2.10}$$

2.3.3 Repulsive Potential Field

The repulsive potential is used to create a potential boundary around obstacles that cannot be crossed by the robot. In addition, we do not want this potential to affect the movement of the robot when it is sufficiently far from the obstacles. We denote $d = |\boldsymbol{q} - \boldsymbol{q}_0|$ to represent the distance between the robot and the obstacle in question. The formula proposed by Khatib is given:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{d} - \frac{1}{d_0})^2 & \text{if } d < d_0, \\ 0 & \text{if } d > d_0, \end{cases}$$

where $\boldsymbol{q}$ is the robot position and $\boldsymbol{q}_0$ is the obstacle position. Recall that d_0 is the influence distance of the force, and η is an adjustable constant.

So then we have the corresponding repulsive force function:

$$\boldsymbol{F}_{rep}(\boldsymbol{q}) = \begin{cases} \eta\left(\frac{1}{d} - \frac{1}{d_0}\right)\frac{(\boldsymbol{q}-\boldsymbol{q}_0)}{d^3} & \text{if } d < d_0, \\ 0 & \text{if } d > d_0, \end{cases}$$

2.3.4 Path Planning

After defining the potential forces, it is sufficient to perform gradient descent to reach the goal. The gradient descent consists simply of following the direction indicated by the force F, by moving in this direction with one step of lenght δ_i. For example, with q=(x,y), we have:

$$x(q_{i+1}) = x(q_i) + \delta_i \frac{\partial U}{\partial x(x,y)} \tag{2.11}$$

$$y(q_{i+1}) = y(q_i) + \delta_i \frac{\partial U}{\partial y(x,y)} \tag{2.12}$$

If we have a path to the goal, when we reach a distance less than δ_i from the q_{goal} , the algorithm should stop and return "goal reached."

CHAPTER 3:
Hardware Experiments

3.1 Equipment

3.1.1 Quadrotor

The following picture shows the prototype UAV control system developed in this thesis.

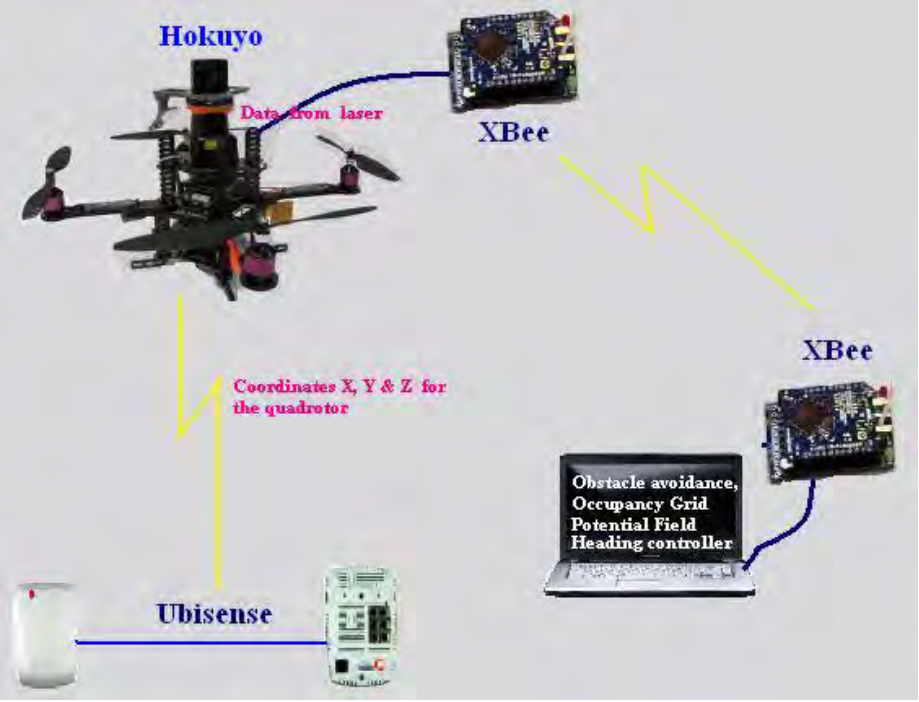

Figure 3.1: Quadrotor Architecture

In this project, we use the commercially available Ascending Technology Pelican quadrotor. The Pelican consists of a well-modeled platform with four rotors. In order to eliminate the need for a tail rotor, like in standard helicopter structure, the rotors were designed to rotate in a cross configuration. In other words, the front and rear rotors rotate counter-clockwise, while the left and the right ones rotate clockwise, as shown in Figure 3.2. Moreover, to give more stability to the quadrotor, the rotors have fixed-pitch blades and their air flows are oriented downwards. The quadrotor was chosen because it is considerably easier to manage than conventional helicopters. Another advantage of using quadrotor design is that they are typically motorized by batteries

instead of a gas engine, which is convenient for certain missions where noise caused by the engine can decrease the chances of surprising the enemy. However, as a drawback, batteries allow shorter flying time. Nevertheless, the quadrotors are petite in size, can be used for indoors experiments, and are a valuable research and potentially tactical platform.

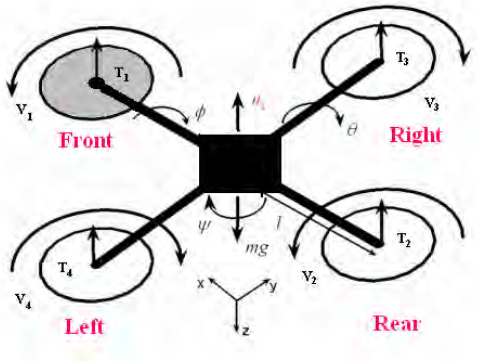

Figure 3.2: Quadrotor's motions description. From: [24]

The following paragraph describes how to compute the variables roll, pitch and yaw.

Let V_i and T_i be the torque and thrust for i^{th} rotor, and l denotes the distance of the rotor from the center of mass. Then the total thrust, denoted u_1, is given by:

$$u_1 = (T_1 + T_2 + T_3 + T_4) \tag{3.1}$$

In addition to the total thrust, the rotation angles of roll, pitch, and yaw can also be controlled by inputs u_2, u_3, and u_4, respectively, using differential thrust:

- The roll angle, ϕ, is achieved by increasing or decreasing the left and right rotor speed via control $u_2 = l(T_4 - T_3)$
- The pitch angle, θ, is provided by increasing or decreasing the front and back rotor speeds via control $u_3 = l(T_1 - T_2)$
- The yaw angle, ψ, is obtained from the torque resulting from the subtraction of the counter-clockwise (front and back) from the clockwise (left and right) speeds, i.e., with control input $u_4 = (u_3 + u_4 - u_1 - u_2)$.

Table 3.1 summarizes some global characteristics of the Pelican platform.

Flight Performance		General and Technical Data	
Cruise speed	36 km/h	Sensor board (platform)	AutoPilot V2
Max. speed	50 km/h	Dimensions	50cm×50cm×20cm
Max. wind speed	36 km/h	Weight of vehicle (ready to fly)	750 g
Usual operating altitude	50 m	Number of propellers	4
Max. payload(g)	500g	Propeller size	10 in
Max. flight time	25 min	Max flight time at max payload	12-15 min
Launch Type	VTOL	LiPo Battery voltage, capacity	11.1 V (6000 mAh)
GPS, Compass	Included	Programmable processor	Included
Safe landing mode	include	Serial port interface	Included

Table 3.1: Ascending Technologies *Pelican* quadrotor specifications [25]

3.1.2 Laser Scanner

Figure 3.3: Hokuyo URG-04LX. From: [26]

For path planning and obstacle detection within an unknown working space, laser scanners are a commonly used solution for mobile robotic applications. Most previous work in the robots field has been done with laser scanners, including those manufactured by Hokuyo. The Hokuyo lasers are relatively expensive compared to other sensors, but given their weight, power, resolution, and integration benefits, these small sensors are significantly easier to connect and to mount on the quadrotor platform.

For this work, a Hokuyo URG-04LX unit is used for area scanning. The light source of the sensor is an infrared laser of wavelength 785nm with a Class 1 laser safety rating. The scan area is a 240^o field of view with maximum sensing range of 4m. The resolution of the scanner is $360^o/1024$ ticks which is approximately 0.36^o such that the total number of scan points is 683. Therefore, if we multiply the number of intervals (683-1) by the angular resolution (0.36^o) we get 240^o. Table 3.2 summarizes some general characteristics of the URG-04LX.

Specifications	
Voltage	5.0 V $\pm$ 5 %
Current	0.5 A (Rush current 0.8 A)
Detection Range	0.02 m to approximately 4 m
Laser wavelength	785 nm, Class 1
Scan angle	240^o
Scan time	100 ms/scan (10.0 Hz)
Resolution	1 mm
Weight	141 gm (5.0 oz)
Interface	USB 2.0, RS232
Angular Resolution	0.36^o

Table 3.2: Hokuyo URG-04LX Lase specifications [26]

The URG-04LX laser scanner comes with a library of software written in C and C++ that provide the ability to communicate and capture data from the sensor. For this thesis, the source code was modified to suit the logging and analysis needs of this project, and can be found in Appendix A.

The following picture shows sample data that was captured in the Autonomous Systems Lab.

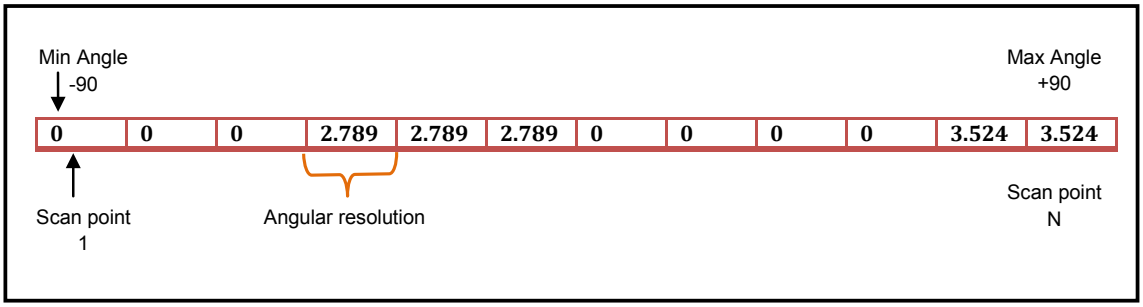

Figure 3.4: Sample Data Representation

Caption:

- 0 is max range.
- non zero (e.g., 2.789) means obstacle at range in meters.
- N is total number of scan points.

The data returned from the laser range finding sensor consists of an array of tuples whose elements are beam intensity values and range values for a single scan. These scans occur at a

frequency of 10 Hz. Also, for a given beam, if no obstacle is detected within the maximum sensor range, that scan point is given a value of zero, which can be used for filtering the data during processing. During each of these scans, the UAV position must also be recorded to recover and register the location of obstacles in the global coordinate frame versus the local (body-fixed) relative coordinate frame. With a known UAV pose (i.e., position and orientation) and the laser scan data, a digital surface model (DSM) or terrain map can also be obtained, e.g., see Figure 3.5, which has been studied in previous works. Looking at the figure above, the laser data can be transformed to the global frame using the following equations:

$$
\begin{aligned}
x &= x_r + \rho * \cos(\theta) \\
y &= y_r \\
z &= z_r - \rho * \sin(\theta)
\end{aligned}
$$

where ρ is the distance measured by the laser scanner and θ is the angle. The robot, or UAV, position is denoted by x_r, y_r, and z_r. Using these relationships, a plot showing the DSM can be obtained.

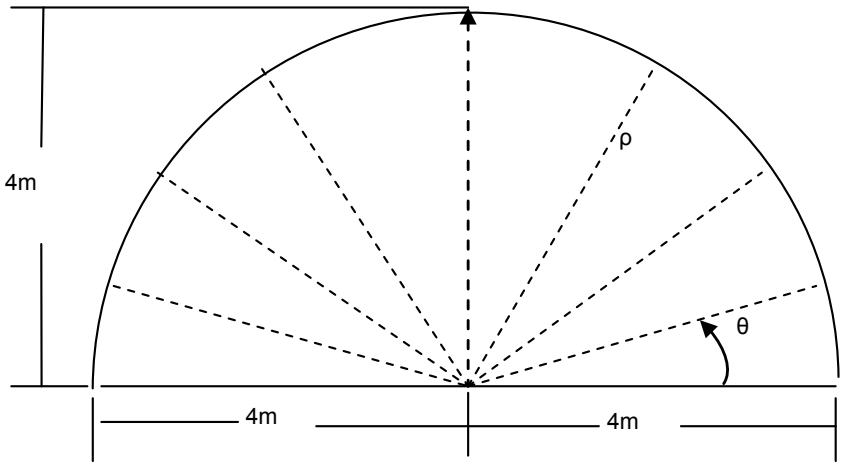

Figure 3.5: How to read Data from Hokuyo

3.1.3 Digi XBee Wireless RF Modules

XBee and XBee-PRO 802.15.4 OEM RF modules, illustrated in Figure 3.6 are embedded solutions providing wireless end-point connectivity to devices. These modules use the IEEE 802.15.4 networking protocol for fast point-to-multipoint or peer-to-peer networking. They are designed for high-throughput applications requiring low latency and predictable communication timing [27].

Figure 3.6: The Xbee RF module used in this project [27].

XBee are very small in size and can be used for any project where we need to interconnect and communicate between components via wireless network. The XBee models are known to have limited input and output pins. Moreover, the XBee units does not support analog output or allow access to other integrated capabilities. For example, if we want to control the speed of a motor, we need to have additional electronic components, e.g., a pulse-width modulation breakout board.

Though this limitation prevents standalone operations for embedded systems, for the purpose of this study, the required capability is to be able to communicate between the onboard sensor on the quadrotor and the laptop which is running most of the code.

Specifications	
Indoor/Urban Range	Up to 300 ft. (90 m), up to 200 ft (60 m) int'l variant
Outdoor RF line-of-sight Range	Up to 1 mile (1600 m), up to 2500 ft (750 m) int'l variant
Supported Network Topologies	Point-to-point, Point-to-multipoint, Peer-to-peer, and Mesh
Operating Frequency Band	ISM 2.4 GHz

Table 3.3: XBee-PRO Serial RF (802.15.4) module – Technical Specifications [27]

3.1.4 Ubisense Location Solution

Where GPS does not work, such as inside structures or tunnel environments, alternate methods for obtaining robot positioning is necessary, as knowledge of the robot location (and orientation) are essential to constructing maps and navigation within them. One technology available in the Unmanned Systems Laboratory (see Figure 2.1) is the Ubisense system, which uses radio frequency beacons and sensors to provide position information. Characterization of the Ubisense position estimates and tracking performance is beyond the scope of this thesis, but the reader is referred to ongoing efforts to evaluate this system for use with quadrotor and other mobile robot platforms.

Figure 3.7: Ubisense. From [28]

Ubisense uses a Combined Angle-of-Arrival and Time-Difference-of-Arrival approach, implemented in proprietary software. In order to have a higher accuracy, Ubisense gathers more readings from each sensor (depicted in Figure 3.7) and allows for a graphical display and the simulation of the effects of configuration changes [28]. Though not explicitly utilized in this thesis, the Ubisense hardware may be a useful tool for providing robot localization information in future experiments.

3.2 Software

The simulation of both the hardware and the algorithms necessary for navigation and mapping was developed in MATLAB (as described further in Chapter 4), with the intention of making use of the iMatlab interface [30] to MOOS. The implementation of the models described in this thesis are included in the appendices[1]. The remainder of this section provides an overview of the MOOS-IvP autonomy software architecture for eventual integration with the presented models.

3.2.1 Mission Oriented Operating Suite

The Mission Oriented Operating Suite (MOOS) [29] was developed at MIT by Paul Newman in 2001. It comprises open source C++ modules that provides autonomy for a number of robotic platforms. Originally, MOOS was developed to support operations for autonomous marine vehicles, and its first use was for the Bluen Odyssey III vehicle owned by MIT.

The following schematic presents a typical MOOS communication configuration proposed by Newman [29].

Figure 3.8: A Typical MOOS Communication Setup proposed by Paul Newman [30]

[1]The associated software can also be found online at `http://faculty.nps.edu/thchung`.

Note the module naming convention, where a module preceded by a “p” means it is a process, and one preceded by an “i” represents that it is associated with an instrument. Because most of the modules shown in this graph were developed to work with unmanned underwater vehicles, we will not use them for this project.

For our project, however, MOOS is utilized to implement a set of low level functions required for a quadrotor to fly. One advantage of MOOS is its subscription-based database system. In other words, users can run many processes with only one message interface. Additionally, MOOS is a modular architecture which makes it easier for programming and updating the project by adding other pieces of code. For a multi-group project, members can work independently without even knowing what the others are working on. Instead, additional modules relevant to the project include software to interface with the quadrotor and laser sensor as well as the algorithms for potential field navigation and occupancy grid-based mapping. Figure 3.9 illustrates the required modules for our navigation and mapping project.

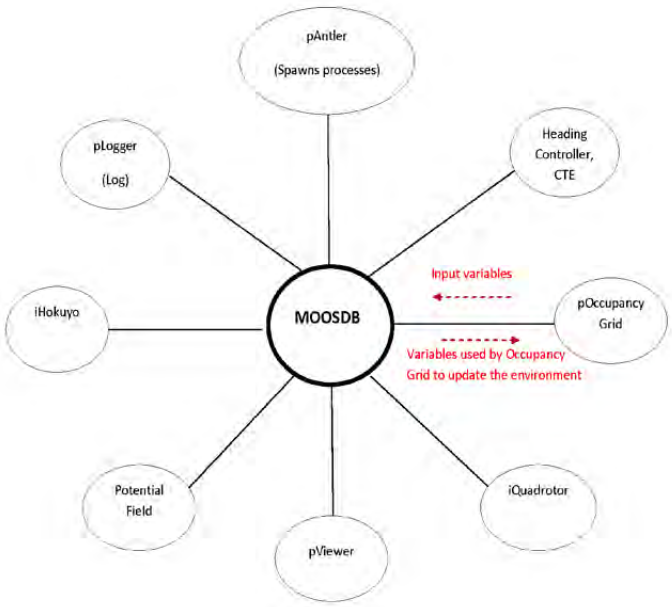

Figure 3.9: MOOS Communication Setup Needed for the Project

Another advantage of MOOS is that the software comes with a set of mission parameters in a standard format. This enables users to just change out settings according to their projects without needing to rewrite every function. After we have made all the necessary changes and

added the new modules, MOOS can be rebuilt simply by executing the `build-moos.sh` script.

3.2.2 Interval Programming, IvP

Interval Programming (IvP) is a mathematical programming model for multi-objective optimization [31]. In the IvP model, each objective function is a piecewise linear function. The IvP model and algorithms are included in the *IvP Helm* software as the method for representing and reconciling the output of helm behaviors. The term interval programming was inspired by the mathematical programming models of linear programming (LP) and integer programming (IP). The pseudo-acronym IvP was chosen simply in this spirit (and to avoid acronym clashing). For more information on how to use MOOS-IvP, the reader is referred to the MOOS-IvP documentation [31].

CHAPTER 4:
Design, Analysis, and Results

4.1 Methods

The broad scope of this project involved both hardware and software development, and this thesis represents initial efforts to implement and integration the various system components.

4.1.1 Architecture Design

Because of the rapid development time frame, it was easier to prototype the implementation of the hardware using MATLAB as a simulator and interface for the experiments instead of using simulation from the MOOS-IvP software. MATLAB also contains common linear algebra and image-processing functions which helped to develop test cases and evaluate results with much less effort. With MATLAB we can quickly generate plots and visualize the simulations to help us understand the behavior of the quadrotor in a given scenario. Using MATLAB exclusively throughout the project, may not be recommended for operational settings, as it is a higher level scripting language which has potential performance limitations, e.g., slow runtime speeds, and potential hardware interface challenges.

Figure 4.1describes the designed hardware and software architecture, for which this work has prepared the foundation. The color-coded links between boxes represents how hardware components are going to communicate with each other and with the software modules.

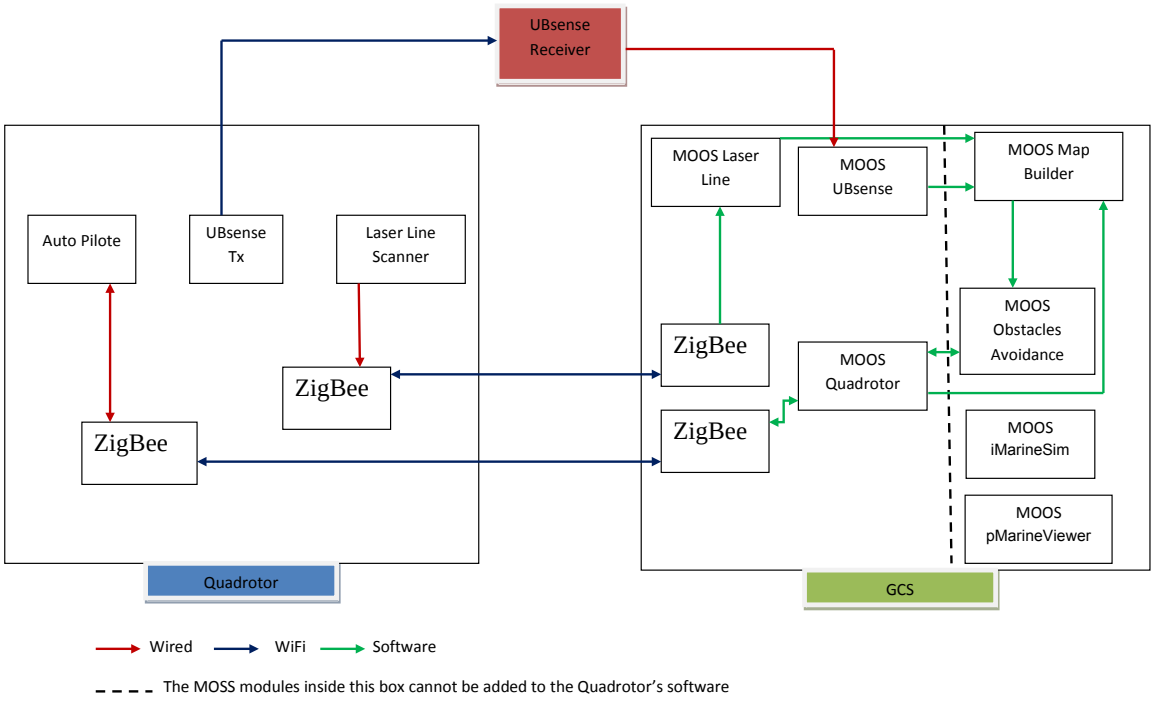

Figure 4.1: Proposed hardware and software architecture for navigation and mapping using the laser line scanner and the quadrotor UAV.

4.1.2 Wireless Communication Integration Tests

In order to ensure that the XBee wireless modules are working, a simple test is to send a chunk of data from both sides of the wireless channel. If the transmitted and received data are the same, with no extra letters or digits added, then a successful connection between the two XBee modules has been established.

If developing in the UNIX-based environment, it is recommended that you use `cutecom` software[1] to communicate with the devices, as it is very straightforward and easy to use. The only required information is the name of the device. To do that, one can simply execute the command `dmesg` in a terminal window to display the name of the device to which it was attached. The

[1] Available at `http://cutecom.sourceforge.net/`

following illustrative message gives an idea of how to access the name of the serial port:

```
[###] usb 6-2:  FTDI USB Serial Device converter now attached to ttyUSB0
```

In this case, in `cutecom` we should enter the device name as `/dev/ttyUSB0`.

If you are using Windows, we recommend using HyperTerminal. As we did in UNIX, we need to know to which serial communications port the XBee was connected. Under "My computer," select "Manage" and then "Device Manager." You should see under Ports something like "USB Serial Port(COM4)." This means that your device was attached to port COM4. So when you use HyperTerminal, you should choose COM4 to open the connection with the device. After that, we need to have the same setup on both sides (i.e., Baud Rate and Parity). Before you can send data, you must make sure that the connection between the XBees is working perfectly. We first need to configure the hardware. Usually, the Xbees come with a software called X-CTU for configuration. Suppose you send "VVVVVV," but you receive "VxobVxobVxob." This means that there is a problem in your setup or maybe a conflict with USB's version. After you solve this problem, you can start sending your data to the XBee. A simple shell script (provided in Appendix A) can be used to redirect the data from the laser to the XBee.

4.2 Simulation Results

4.2.1 Data Collection

To have a better understanding of the sensor's data, it was necessary to gather large amounts of sensor data from various situations. According to the manual for the sensor, the maximum range that can be reached is about 4 meters, but the experiments proved that 5 meters are reachable, especially when we use the sensors indoors. Moreover, windows represent a trap for sensors. From a distance of three meters, sensors do not get any returns from a window. Even though this error can be corrected at a closer distance, the results from the potential field computations will be wrong and could result in a waste of time in navigating from a given start point to the goal.

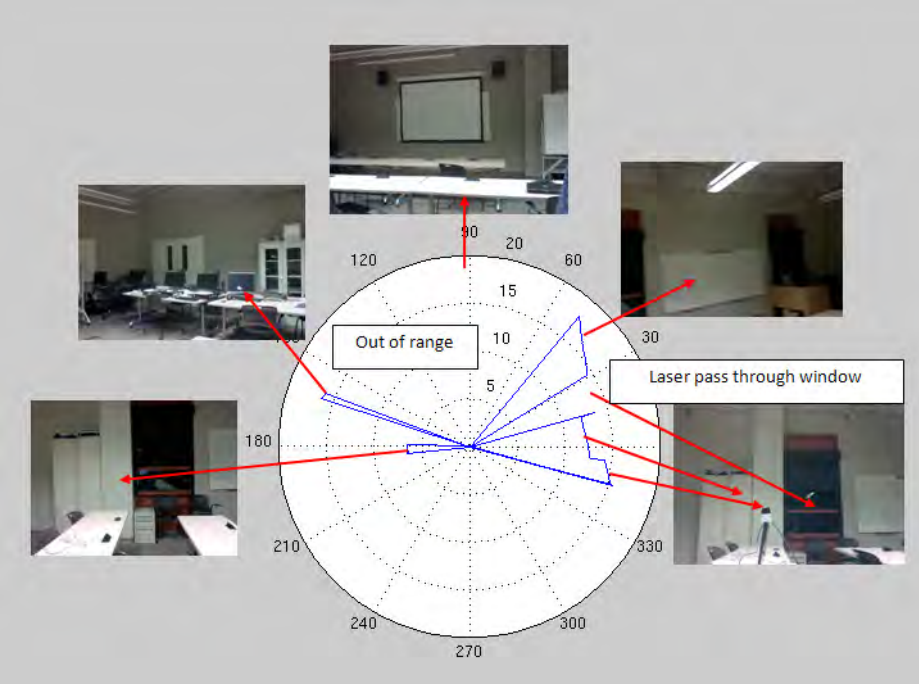

Figure 4.2: Real Data From Sensors

4.2.2 Occlusion

Occlusion is defined by the shape of the obstacle and the position of the sensor facing the obstacle at that time. Occlusion can be a problem if the mission consists of not just finding goals, but also building a real map for the environment's mission. To solve this problem, the sensor has to face the obstacle from all sides. Figure 4.3 gives an idea how the shapes can be interpreted differently.

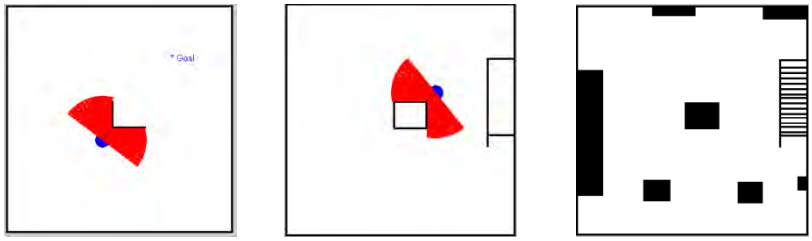

Figure 4.3: Real Data From Sensors

4.2.3 Occupancy Grid

The Occupancy Grid algorithm as described in Chapter II, consists of updating the probability of occupancy every time we observe a hit from the sensor. The snapshots depicted in Figure 4.4 and Figure 4.5, taken from simulated data, show how an obstacle can be detected. This simulation was performed in a known environment where the obstacles were plotted with values equal to one. The MATLAB code can be found in Annex B.

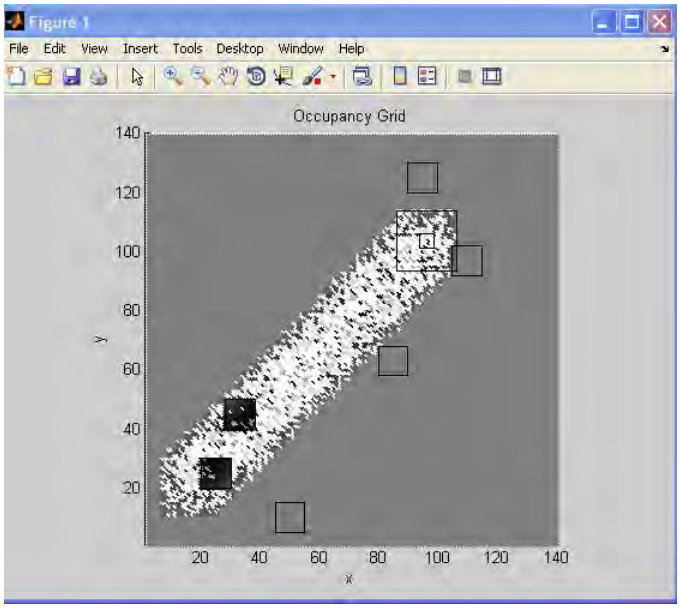

Figure 4.4: Obstacle Detection

As we can see from Figure 4.4, an area of the grid darkens when an an obstacle is detected and lightens when the observation returns a low probability of occupancy. After several runs, the map should look like Figure 4.5.

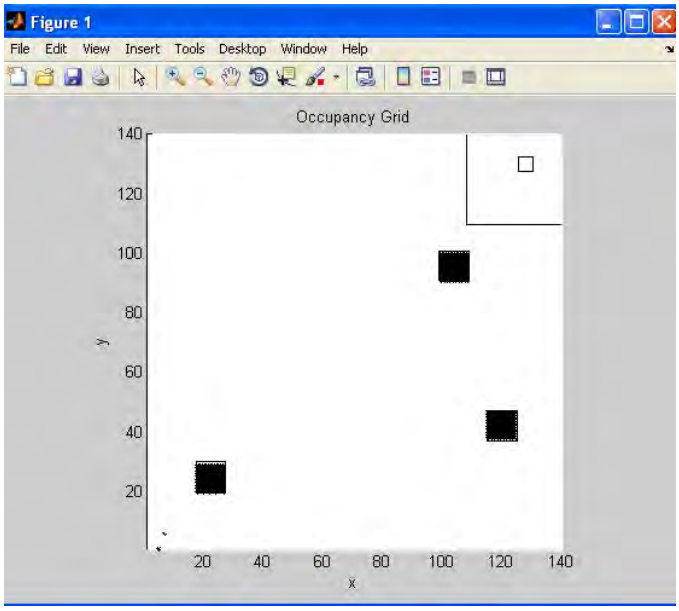

Figure 4.5: Updating Grid Probability

4.2.4 Potential Field

The MATLAB picture shown in Figure 4.6 displays a potential field plot indicating, first, how a quadrotor can react to repulsion forces generated by the obstacles and, second, how the attraction forces push the quadrotor toward the goal even as it avoids obstacles encountered in its path.

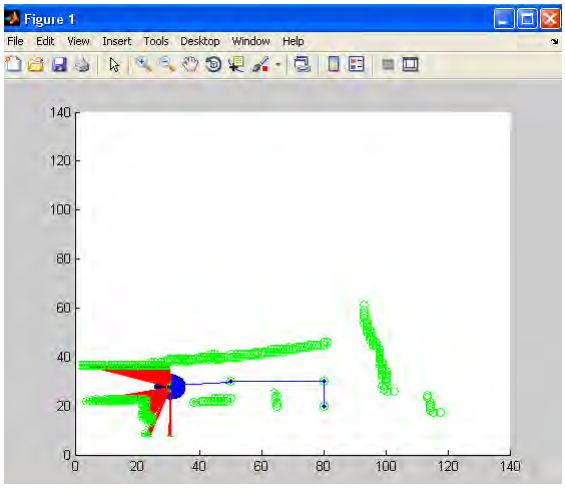

Figure 4.6: Path and Map Generated by the Potential Field Functions

4.3 Physical Robot Tests

Most of the experiments have been done in the Autonomous System Lab. The first experiment was conducted by driving the quadrotor using a cart. This step was very important in evaluating the results from the map-building process. This section highlights a number of initial component efforts in order to prepare for integrated testing.

4.3.1 Converting between Local and Global Coordinate Frames

Preliminary laser data processing tasks includes conversion from relative local coordinates (due to range and bearing measurements) of the obstacles to global coordinates in order to ensure an appropriately rectified map reflecting the real world.

Coordinate transformations in the plane require a rotation and translation of the robot frame to/from the global reference frame. Consider first the case where the quadrotor is located at $(x_R, y_R) = (0,0)$, i.e., both the local and global coordinate frames overlap at the origin, but with some heading θ, as shown in Figure 4.7 below.

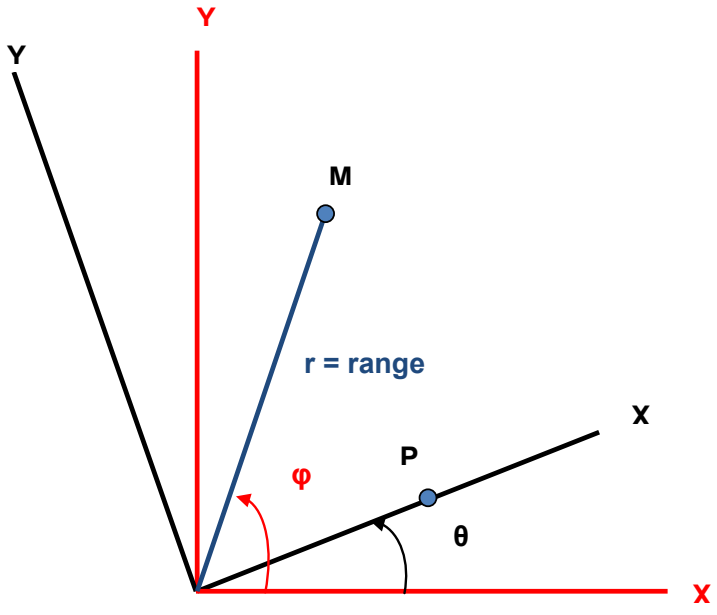

Figure 4.7: Definition of parameters for transformations between rotated coordinate frames.

Given the local robot coordinate system, denoted by axes x, y (as opposed to the global frame with axes x', y'), a laser range finder provides a relative range and bearing measurement, namely

ρ and ϕ, respectively. The scan point (labeled point M in Figure 4.7) represents a possible obstacle at that location.

Then, recalling the rotation, θ, between frames, the coordinates of the scan point can be represented in both local and global reference frames as:

$$\begin{aligned} x = \rho\ \cos(\phi) \quad \text{(local)} \qquad &\Longleftrightarrow \qquad x' = \rho\ \cos(\phi - \theta) \quad \text{(global)} \\ y = \rho\ \sin(\phi) \quad \text{(local)} \qquad &\Longleftrightarrow \qquad y' = \rho \sin(\phi - \theta) \quad \text{(global)} \end{aligned}$$

As mentioned previously, for simplicity in modeling, assume the sensor scan spans 180^o starting at $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. Then the angle to the minimum angle, i.e., the first scan point, requires an offset of $-\frac{\pi}{2}$. The following examples show the results for different heading angles as special cases.

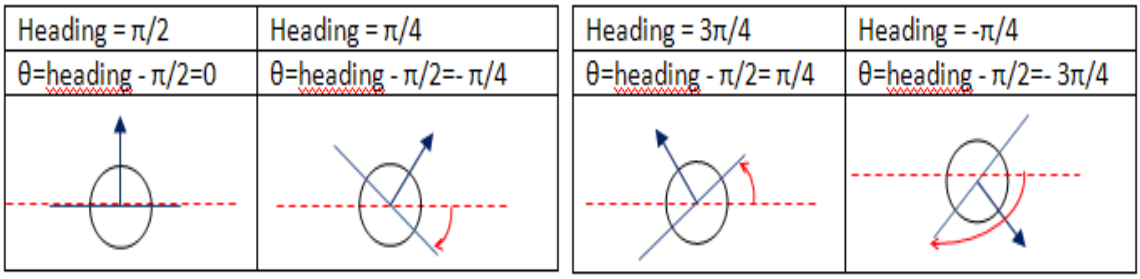

Figure 4.8: Coordinate transformation of laser scan points for different robot heading angles.

In the more general case where the quadrotor is located at a global position x_R, y_R with orientation θ, the conversion between local and global coordinates must account for this translation when computing the location of the scan point in question. In other words, the following expressions represent the appropriate transformation:

$$\begin{aligned} x = \rho\ \cos(\phi) \quad \text{(local)} \qquad &\Longleftrightarrow \qquad x' = x_R + \rho\ \cos(\phi - \theta) \quad \text{(global)} \\ y = \rho\ \sin(\phi) \quad \text{(local)} \qquad &\Longleftrightarrow \qquad y' = y_R + \rho \sin(\phi - \theta) \quad \text{(global)} \end{aligned}$$

Figure 4.9 illustrates an example case with a heading of $\theta = \frac{\pi}{2}$, where again the range and bearing to the scan point M are denoted ρ and ϕ.

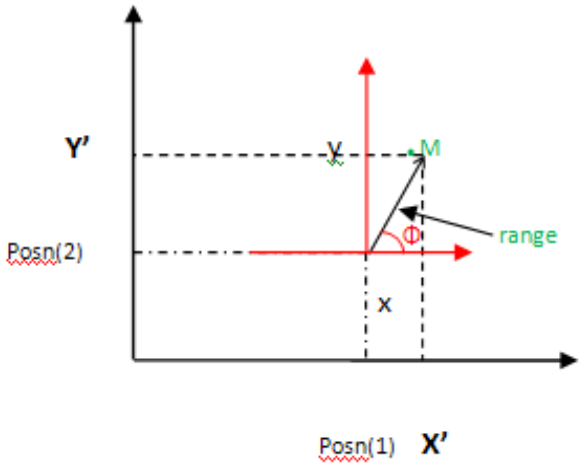

Figure 4.9: Coordinate transformation from local to global coordinates for the special case of a robot located at $(x_R, y_R) = (\text{posn}(1), \text{posn}(2))$ with heading $\theta = \frac{\pi}{2}$.

4.3.2 Map Building and Potential Field Process Flowchart

The graph below shows how the MATLAB scripts are executed and explains which script is called and the order in which the scripts are called. The order of execution is very important because, for example, if GETDATA does not work for some reason, then the whole process will fail to execute. The scripts were not designed to catch errors and return a detailed message to help users. We will leave this for a future work.

4.3.3 How to Call MATLAB Scripts from MOOS

This section describes the procedure for integrating algorithms, e.g., those developed in MATLAB, into the MOOS autonomy software architecture.

Installation of MOOS-IvP software

The MOOS-IvP autonomy software is available at `http://www.moos-ivp.org`, the source for which which can be downloaded from a version-controlled software server. Two steps are indispensable to building MOOS-IvP, and they have to be executed in order. First, the MOOS tree must be built by executing `build-moos.sh` script. Then, by executing the script `build-ivp.sh`, the IvP directory will be built automatically and the project should be ready to run [31]. The main processes that must we are concerned with are: `On Startup()`, `Connect to Server()`, `Iterate()`, and `On New Mail()`.

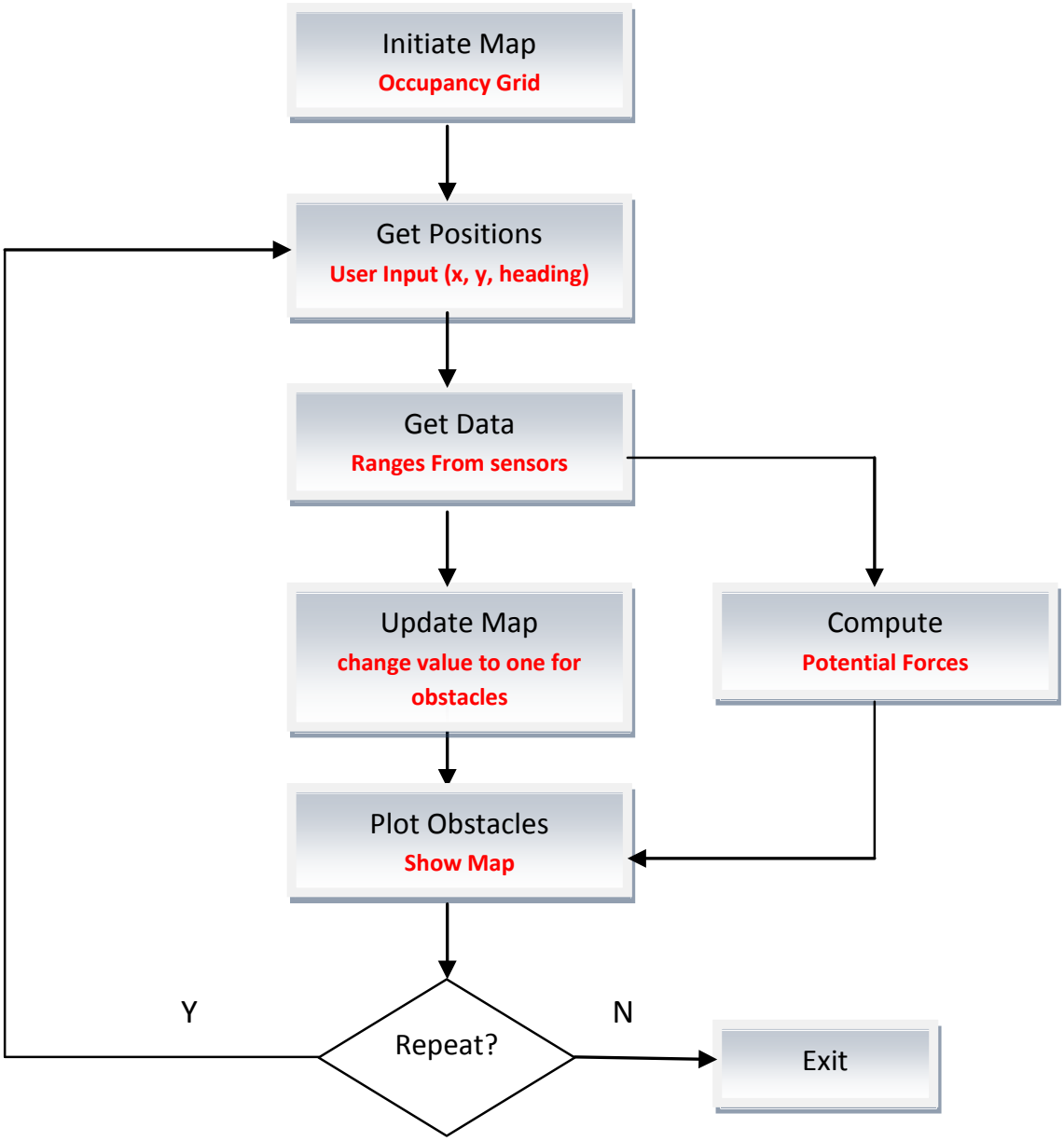

Figure 4.10: Map Building and Potential Field Process Flowchart

Running MATLAB with MOOSDB

MOOS code can be called from inside MATLAB using the open-source software project, iMatlab [30]. iMatlab allows MATLAB's script to connect the MOOSDB either to receive or to send variables needed for the project. To send data from the matlab script, we have to use the

following iMatlab syntax:

$$>> \texttt{iMatlab}(\texttt{MOOS MAIL TX}, \mathit{VARNAME}, \mathit{VARVAL}),$$

where, for example, *VARNAME*=pitch, *VARVAL*=15.

Figure 4.11 below shows a typical configuration for iMatlab.

```
ProcessConfig = iMatlab
{
    AppTick       = 10
    CommsTick     = 10
    Port          = COM6
    BaudRate      = 4800
    Verbose       = false
    Streaming     = false
    MOOSComms = true
    SerialComms = false
    SERIAL_TIMEOUT = 10.0
    SUBSCRIBE = DB_TIME @ 0.0
}
```

Figure 4.11: A typical configuration block for iMatlab

Integrating Potential Field and Occupancy Grid Mapping Algorithms into MOOSDB

The concept of how the potential field navigation and occupancy grid mapping processes should be connected to MOOSDB is illustrated in Figure 4.12. Taking advantage of iMatlab, the MATLAB-based implementations presented in this thesis can easily and transparently receive and send data with other processes running within the MOOS project.

When the sensor detects an obstacle in its range, this data will be used by the occupancy-grid process to build the map and also by the potential-field process to compute the new heading. The new heading will be translated to either yaw, pitch or roll. When this new value gets published, the iAutopilot process will give an order to the quadrotor to change direction. Actually, the check-mail process is in charge to listen and detect any changes occurring during the mission.

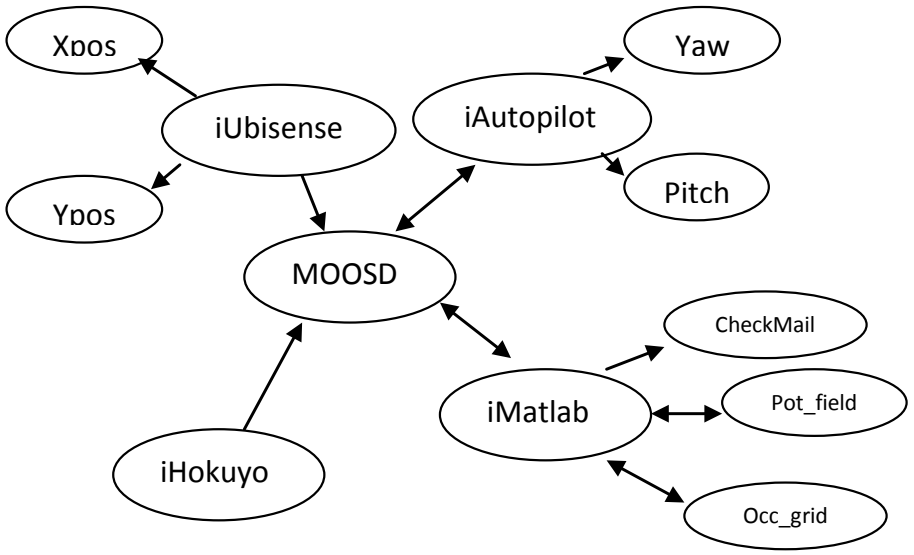

Figure 4.12: Proposed configuration for MOOSDB integrating navigation and mapping algorithms via iMatlab and hardware, e.g., laser scanner (Hokuyo), aerial platform (quadrotor), and localization system (UBisense).

4.3.4 Results

Despite successful preliminary investigations in simulation, our first attempt to build a map was not successful. A number of error phenomena were observed. For example, upon further analysis, there were repeatedly relatively large offsets between the real location of obstacles and their location as represented in the occupancy grid-based map.

Several factors could have caused these errors. The main factor was the grid size and resolution. To reduce the computational load, we fixed the size of the grid to a 140×140 square arena (i.e., 19,600 cells), such that each cell represented a 10cm$\times$10cm spatial block. Since the data collected from the sensor were on the order of millimeters due to the precision of the laser sensor, the shape of the obstacle within the map was limited by the coarse resolution. This resulted in a representation that was somewhat different than the shape of the real obstacle.

Another contributing factor to errors in the map was the imperfect knowledge of the vehicle position and orientation at the time we captured the image. Since no positioning system, such as the Ubisense, was available, rough pose estimates were measured by hand and aligned approximately with laser scan data.

A remaining problem is related to well-known limitations of the potential field algorithm, which, due to its gradient-based approach, can suffer from getting trapped in local minima of the potential field. An illustration of such a trapping scenario for potential field-based navigation is provided in Figure 4.13, showing how a vehicle can get trapped and fail to escape from the local minimum.

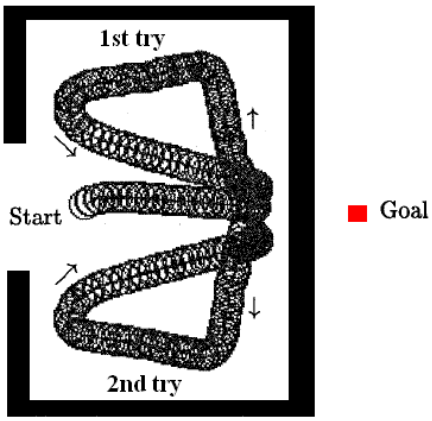

Figure 4.13: Illustration taken from [32] showing the pitfalls of local minima in potential field-based algorithms for mobile robot navigation.

The first problem is trivial and can be addressed by changing the grid size or by using software other than Matlab. This will allow implementation of a larger grid without being penalized in runtime cost.

Since we had to simulate the coordinates for the quadrator, instead of using the real position from the Ubisense (because the lack of time), we cannot be sure that the error we had will cause a real problem when we use GPS or Ubisense to navigate. We will leave this for future work to be tested and fixed. The last problem is very trivial and can be addressed easily. There are several approaches proposed to avoid this problem. A very simple implementation is to set up a series of repulsive forces with zero radius positioned at newly visited positions. As time progresses, the repulsive force decreases and the recently visited locations become more repulsive than positions visited a longer time ago. [33]

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5:
Conclusions

5.1 Discussions

This thesis presented an implementation study of sensor-based robot algorithms for probabilistic mapping of an indoor environment and collision-free navigation to a goal within that environment. The former task was addressed by generating an occupancy grid-based map using models of a laser line scanner sensor to detect obstacles, and both binary or Bayesian sensor updates to the map were investigated. Discretization of the given area provided cells in space which reflected the presence or absence of an obstacle in each cell. These types of probabilistic maps possess several advantages over static counterparts, including the ability to refine the maps in the presence of uncertainty. Additionally, if obstacles are dynamically moving in the workspace, the use of sensor-based methods such as presented in this thesis allow for appropriately interpreting newly obstructed locations while clearing previously occupied ones.

The field of robot motion planning is diverse and can be categorized coarsely into a few general approaches, including roadmap, cell decomposition, sampling, and potential field algorithms [34]. For this thesis' initial implementation, we selected Artificial Potential Field algorithms. A potential function is assigned to each obstacle detected using a model of the laser rangefinding sensor. Using the computed force to guide the motion of the robot, this approach allows us to derive a collision-free path to the goal. Though mathematically elegant, a number of implementation challenges were faced due to the need to address practical considerations. On the other hand, it will ultimately be easier to implement this algorithm within the MOOS-IvP architecture since it straightforwardly outputs a new desired heading which can be translated easily to pitch, roll and/or yaw controls to be sent to the quadrotor platform. If the new desired heading does not deviate significantly from the previous one, then the robot may just keep going forward without changing direction. If a new heading is computed, the nature of the quadrotor platform allows for easy translation of these deviations to motor commands.

In summary, this thesis can be considered an initial effort to contribute to a larger systems project involving the quadrotor unmanned aerial vehicle. The primary contributions are to explore and implement common robot algorithms for building a probabilistic map using an occupancy grid approach and also for avoiding obstacles using a potential field technique. Parallel

efforts by other researchers have developed flight control software, also in MATLAB, capable of sending and receiving information from the quadrotor's autopilot. This code can control the pitch, yaw, roll and thrust of the quadrotor executing a waypoint following mission. This thesis provides complementary and higher-level capabilities, given the autonomy necessary to compute new headings based on the navigation and mapping algorithms. Though beyond the scope of this investigation, the ultimate objective is to provide a robust integration of these parallel capabilities.

5.2 Future Work

There are numerous issues to be addressed and various extensions to be pursued, but this thesis can serve as both an initial proof of concept and a foundation for the implementation efforts necessary.

Recall that the main goal of this thesis was to provide the quadrotor a method to navigate in a two-dimensional environment. However, given the intrinsic three-dimensional nature of aerial robot platforms, restriction to the plane can be too limiting. For example, the shortest path to the goal may merit having the quadrotor fly above and over an obstacle instead of driving around it. Such tradeoffs cannot be explored nor accomplished if we just limit our computation to a two-dimensional grid. My recommendation is to convert this code to a three-dimensional structure, coupled with sensor fusion algorithms [35], to improve the fidelity of perception by increasing the observation opportunities and operational realism of the navigation mission.

Another avenue for future work is to improve the occupancy grid-based mapping algorithm to reduce the computational burden, so as to be executable on embedded systems onboard the quadrotor. Enhancements with new data structures and software engineering can leverage the lessons learned in this thesis to realize this recommendation.

REFERENCES

[1] C. S. Smith, "Tunisia is feared to be a new base for Islamists - Africa & Middle East - International Herald Tribune," 2007.

[2] S. D. Hanford, L. N. Long, and J. F. Horn, "A Small Semi-Autonomous Rotary-Wing Unmanned Air Vehicle (UAV)," *2003 AIAA Atmospheric Flight Mechanics Conference Exhibit*, no. September, pp. 1–10, 2005.

[3] S. Bouabdallah, A. Noth, and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," *2004 IEEERSJ International Conference on Intelligent Robots and Systems IROS IEEE Cat No04CH37566*, vol. 3, pp. 2451–2456, 2004.

[4] M. Chen and M. Huzmezan, "A combined MBPC/2 dof H controller for a quad rotor UAV," in *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, 2003.

[5] E. Johnson and P. DeBitetto, "Modeling and simulation for small autonomous helicopter development," *AIAA Modeling and Simulation Technologies*, pp. 1–11, 1997.

[6] F. Hoffmann, T. J. Koo, and O. Shakernia, "Evolutionary Design of a Helicopter Autopilot," in *3rd OnLine World Conference on Soft Computing WSC 3*, 1998, pp. 1–18. [Online]. Available: http://eprints.kfupm.edu.sa/38404/

[7] T. J. Koo, F. Ho, H. Shim, B. Sinopoli, and S. Sastry, "Hybrid Control Of An Autonomous Helicopter," *Control*, pp. 285–290, 1998.

[8] M. Kottmann, "Software for Model Helicopter Flight Control Technical Report Nr 314," *Language*, no. March, 1999.

[9] G. Lai, K. Fregene, and D. Wang, "A control structure for autonomous model helicopter navigation," V. . In canadian Conference on Electrical and Computer Engineering, Ed., Portoroz,Slowenien, pp. 103–107.

[10] H. J. Kim, D. H. Shim, and S. Sastry, "Flying robots: modeling, control and decision making," *Proceedings 2002 IEEE International Conference on Robotics and Automation Cat No02CH37292*, no. May, pp. 66–71, 2002.

[11] H. Kim, "A flight control system for aerial robots: algorithms and experiments," *Control Engineering Practice*, vol. 11, no. 12, pp. 1389–1400, 2003.

[12] A. Williams, "Vicacopter autopilot." [Online]. Available: http://coptershyna.sourceforge.net/

[13] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard, "Nested Autonomy for Unmanned Marine Vehicles with MOOS-IvP," *Journal of Field Robotics*, 2010.

[14] J. G. Leishman, "A History of Helicopter Flight," 2000. [Online]. Available: http://terpconnect.umd.edu/~leishman/Aero/history.html

[15] W. Wang and G. Song, "Autonomous Control for Micro-Flying Robot," *Design*, pp. 2906–2911, 2006.

[16] M. C. Martin and H. Moravec, "Robot Evidence Grids," 1996.

[17] A. Elfes, "Occupancy Grids: A Stochastic Spatial Representation for Active Robot Perception," in *Autonomous Mobile Robots*, S. S. Iyengar and A. Elfes, Eds. IEEE Computer Society Press; . Los Alamitos, California, 1991, pp. 60–70.

[18] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 3, pp. 278–288, 1991.

[19] R. R. Murphy, *Introduction to AI Robotics*. MIT Press, 2000, vol. 401.

[20] Eugene Lukacs, *Probability And Mathematical Statistics*. New York: Academic Press, 1972.

[21] John X. Liu, *New Developments in Robotics Research*. Nova Publishers, 2005.

[22] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[23] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005, vol. 12, no. 3.

[24] I. D. Cowling, O. A. Yakimenko, J. F. Whidborne, and A. K. Cooke, "A Prototype of an Autonomous Controller for a Quadrotor UAV," in *European Control Conference*, 2007, pp. 1–8.

[25] Asctec, "solutions for EDUCATION - RESEARCH - UNIVERSITIES," *City*, vol. 49, no. April 2011, 2012.

[26] Hokuyo, "Hokuyo URG-04LX Laser," pp. 80 301–80 301, 2011.

[27] "Xbee." [Online]. Available: http://www.digi.com/

[28] "UBisense." [Online]. Available: http://www.ubisense.net/en/rtls-solutions/

[29] Paul M. Newman, "MOOS-IvP."

[30] P. Newman, "MOOS Meets Matlab iMatlab," pp. 1–4, 2009.

[31] M. R. Benjamin, H. Schmidt, J. J. Leonard, H. Release, and P. Newman, "Computer Science and Artificial Intelligence Laboratory Technical Report An Overview of MOOS-IvP and a Users Guide to the IvP Helm - Release 4 . 2 . 1 An Overview of MOOS-IvP and a Users Guide to," *Artificial Intelligence*, 2011.

[32] X. Yun and K.-c. Tan, "A wall-following method for escaping local minima in potential field based motion planning," *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR'97*, pp. 421–426, 1997.

[33] M. A. Goodrich, "Potential Fields Tutorial," pp. 1–9.

[34] G. Varadhan, "A Simple Algorithm for Complete Motion Planning of Translating Polyhedral Robots," *The International Journal of Robotics Research*, vol. 24, no. 11, pp. 983–995, Nov. 2005.

[35] S. B. Lazarus, P. Silson, A. Tsourdos, R. Zbikowski, and B. A. White, "Multiple sensor fusion for 3D navigation for unmanned autonomous vehicles," *Control Automation MED 2010 18th Mediterranean Conference on*, pp. 1182–1187, 2010.

[36] L. Hokuyo Automatic Co., "Driver for Hokuyo," 2008. [Online]. Available: http://www.hokuyo-aut.jp/02sensor/07scanner/download/index.html

[37] Nps, "Nps Wiki." [Online]. Available: https://wiki.nps.edu/display/~thchung/Home

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A:
C++ CODE for the LASER

A.1 Modified C++ code for the Hokuyo LIDAR Sensor

The driver for Hokuyo (URG) can be downloaded from here [36].

```
/*!
  \example mdCaptureSample.cpp

  \brief Sample to get data using MD command

  \author Satofumi KAMIMURA

  \modified by Mejdi Ben Ardhaoui

  $Id: mdCaptureSample.cpp 1683 2010-02-10 10:28:05Z satofumi $
*/

#include "UrgCtrl.h"
#include "delay.h"
#include "ticks.h"
#include <cstdlib>
#include <cstdio>

using namespace qrk;
using namespace std;


//! main
int main(int argc, char *argv[])
{
#ifdef WINDOWS_OS
  const char device[] = "COM3";
#else
  const char device[] = "/dev/ttyACM0";
#endif

  UrgCtrl urg;
  if (! urg.connect(device)) {
    printf("UrgCtrl::connect: %s\n", urg.what());
    exit(1);
  }

#if 1
  // Set to MD mode to acquire data
  urg.setCaptureMode(AutoCapture);
```

```
#else
  // Mode to get distance data and intensity data
  urg.setCaptureMode(IntensityCapture);
  urg.setCaptureSkipLines(2);
#endif
  int scan_msec = urg.scanMsec();

#if 0
  // Set range of acquisition from the center to left 90 degree.
  // Set range of acquistion from center to right to 90 degree.
  // So In total it will be 180 degree.
  const double rad90 = 90.0 * M_PI / 180.0;
  urg.setCaptureRange(urg.rad2index(-rad90), urg.rad2index(rad90));
#endif

  int pre_timestamp = ticks();

  // Data is acquired continuously using MD command
  // but outputs data of specified number of times.
  enum { CaptureTimes = 1};
  urg.setCaptureTimes(CaptureTimes);
  for (int i = 0; i < CaptureTimes;) {
    long timestamp = 0;
    vector<long> data;

    // Get data
    int n = urg.capture(data, &timestamp);
    if (n <= 0) {
      delay(scan_msec);
      continue;
    }
#if 1

    for (int j = 90; j < n-90; ++j) {

                printf("%4.3f\n", data[j]/1000.0);
                // We divide by 1000 to get results in meters

    }

#endif
    ++i;
  }

#ifdef MSC
  getchar();
#endif

  return 0;
}
```

A.2 Script for Capturing Laser Data

The following script allows to send the data captured to the Xbee instead of sending it to the standard output.

```
#!/bin/sh
stty -F /dev/ttyUSB0 57600
while [ 1 ]
do
        ./mdCaptureSample > /dev/ttyUSB0
        # If we want to keep a copy of data captured we
        # have to add the following line which save the
        # last Laser reading.
        # cp laserData.txt ../../../Laser/laserData.txt
done
```

Some notes:

- The $>$ is used to redirect the data captured from the laser to the XBee attached to `ttyUSB0`
- The argument 57600 means we communicate in both transmitting and receiving across the serial port with a baud rate of 57600 bits per second
- `mydatacapture` is the name of the modified laser driver code which calls other programs to extract data from the laser

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B:
MATLAB Code (Occupancy Grid)

The goal of this code is to represent a map of the environment as an array of cells. Each cell holds a probability value which indicates if that cell is occupied or not. All source files, including this MATLAB code, is available online[1].

```
% Created By Michael A. Goodrich
% Modified By Mejdi Ben Ardhaoui
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                          Global Variables                               %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

LabDim = 140;                % World Dimension (Square)
OBS_SIZE  = 10;              % Obstacles Size
Quadrotor = OBS_SIZE/4;      % Quadrotor Size
Sensor_Range = OBS_SIZE;     % Sensor Range
NObst = 6;                   % Number of Obstacles in this World
vel = 4;                     % Velocity in units per sample.
dt = 0.1;                    % How often the Quadrotor's position is updated.
ri = 10; ci = 30;            % Intial ROW and COLUMN on the world grid
                             % matrix, where the path is to begin
rf = 130; cf = 120;          % Final ROW and COLUMN on the world grid
                             % matrix, where the path is to end
theta = atan2(rf-ri,cf-ci);  % Initial direction of the Quadrotor
Del = 10;                    % Tolerance distance from the Goal

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                        CREATE PROBABILITY GRID                          %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

% Initialize the probability to 0.50
p = .50*ones(LabDim,LabDim);

% Plot the probability that a grid location is occupied by an obstacle as
% a pseudo-intensity -- darker colors indicate that the grid location is
% more likely to be occupied.

GridHandle = zeros(LabDim,LabDim);
clf; close all;
hold on;
axis([0 LabDim 0 LabDim]);

disp('Click to select starting position');
[ci,ri] = ginput(1);
% TODO check if the start point is within the obstacle boundaries
```

[1] All source code is available online at `http://faculty.nps.edu/thchung` under `Resources, Software`.

```
text(ci,ri,'Start', 'FontSize', 10);
disp('Click to select goal position');
[cf,rf] = ginput(1);
% TODO check if the start point is within the obstacle boundaries
text(cf,rf,'Goal', 'FontSize', 10);

% Re-compute the initial theta
theta = atan2(rf-ri,cf-ci);  % Initial direction of the Quadrotor

for i=1:LabDim
    for j=1:LabDim
        GridHandle(i,j) = plot(i,j,'.');
        set(GridHandle(i,j),'color',[1-p(i,j),1-p(i,j),1-p(i,j)],...
        'MarkerSize',5,'EraseMode','normal');
    end;
end;
hold off;
set(gca,'DataAspectRatio',[1,1,1]);
axis([0,140,0,140]);

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                        Likelihoods AND Definitions                      %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
% The two possible states for each cell: occupied and unoccupied. For each%
% cell, there are three possible observations: hit, no-hit, and no        %
% observation. We will only use the hit or miss observations.             %
% p(O=hit|S=occupied)       = "TrueHit"                                   %
% p(O=hit|S=unoccupied)     = "FalseAlarm"                                %
% p(O=no-hit|S=occupied)    = "MissedDetection" = 1-TrueHit               %
% p(O=no-hit|S=unoccupied)  = "TrueMiss" = 1-FalseAlarm                   %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

%The probability that an occupied cell is detected as a hit by the observer.
TrueHit = 0.98;

%The probability that an unoccupied cell is detected as a hit.
FalseAlarm = 0.06;

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                     PLACE OBSTACLES For KOWN WORLD                      %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
Obst1 = [20,20];
Obst2 = [45,5];
Obst3 = [28,40];
Obst4 = [80,58];
Obst5 = [105,92];
Obst6 = [90,120];

%=======================
% Plot the obstacles
%=======================
xvertices =[Obst1(1,1),Obst1(1,1),Obst1(1,1)+OBS_SIZE,Obst1(1,1)+OBS_SIZE];
```

```
yvertices =[Obst1(1,2),Obst1(1,2)+OBS_SIZE,Obst1(1,2)+OBS_SIZE,Obst1(1,2)];
patch(xvertices,yvertices,'r');
%-------------------------------------------------------------------------



% Find the center of obst1
x1=round((Obst1(1,1)+Obst1(1,1)+OBS_SIZE)/2);
y1=round((Obst1(1,2)+OBS_SIZE+Obst1(1,2))/2);
%-------------------------------------------------------------------------
xvertices =[Obst2(1,1),Obst2(1,1),Obst2(1,1)+OBS_SIZE,Obst2(1,1)+OBS_SIZE];
yvertices =[Obst2(1,2),Obst2(1,2)+OBS_SIZE,Obst2(1,2)+OBS_SIZE,Obst2(1,2)];
patch(xvertices,yvertices,'r');
%-------------------------------------------------------------------------
% Find the center of obst2
x2=round((Obst2(1,1)+Obst2(1,1)+OBS_SIZE)/2);
y2=round((Obst2(1,2)+OBS_SIZE+Obst2(1,2))/2);
%-------------------------------------------------------------------------
xvertices =[Obst3(1,1),Obst3(1,1),Obst3(1,1)+OBS_SIZE,Obst3(1,1)+OBS_SIZE];
yvertices =[Obst3(1,2),Obst3(1,2)+OBS_SIZE,Obst3(1,2)+OBS_SIZE,Obst3(1,2)];
patch(xvertices,yvertices,'r');
%-------------------------------------------------------------------------
% Find the center of obst3
x3=round((Obst3(1,1)+Obst3(1,1)+OBS_SIZE)/2);
y3=round((Obst3(1,2)+OBS_SIZE+Obst3(1,2))/2);
%-------------------------------------------------------------------------
xvertices =[Obst4(1,1),Obst4(1,1),Obst4(1,1)+OBS_SIZE,Obst4(1,1)+OBS_SIZE];
yvertices =[Obst4(1,2),Obst4(1,2)+OBS_SIZE,Obst4(1,2)+OBS_SIZE,Obst4(1,2)];
patch(xvertices,yvertices,'r');
%-------------------------------------------------------------------------
% Find the center of obst4
x4=round((Obst4(1,1)+Obst4(1,1)+OBS_SIZE)/2);
y4=round((Obst4(1,2)+OBS_SIZE+Obst4(1,2))/2);
%-------------------------------------------------------------------------
xvertices =[Obst5(1,1),Obst5(1,1),Obst5(1,1)+OBS_SIZE,Obst5(1,1)+OBS_SIZE];
yvertices =[Obst5(1,2),Obst5(1,2)+OBS_SIZE,Obst5(1,2)+OBS_SIZE,Obst5(1,2)];
patch(xvertices,yvertices,'r');
%-------------------------------------------------------------------------
% Find the center of obst5
x5=round((Obst5(1,1)+Obst5(1,1)+OBS_SIZE)/2);
y5=round((Obst5(1,2)+OBS_SIZE+Obst5(1,2))/2);
%-------------------------------------------------------------------------
xvertices =[Obst6(1,1),Obst6(1,1),Obst6(1,1)+OBS_SIZE,Obst6(1,1)+OBS_SIZE];
yvertices =[Obst6(1,2),Obst6(1,2)+OBS_SIZE,Obst6(1,2)+OBS_SIZE,Obst6(1,2)];
patch(xvertices,yvertices,'r');
%-------------------------------------------------------------------------
% Find the center of obst6
x6=round((Obst6(1,1)+Obst6(1,1)+OBS_SIZE)/2);
y6=round((Obst6(1,2)+OBS_SIZE+Obst6(1,2))/2);
%-------------------------------------------------------------------------
set(gca,'DataAspectRatio',[1,1,1]);
axis([1,140,1,140]);
```

```
title('Occupancy Grid');
xlabel('x');ylabel('y');
drawnow;
figure(1);

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%              CREATE GROUND TRUTH OCCUPANCY GRID FOR KNOWN WORLD            %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

Occupied = zeros(LabDim,LabDim);
for i=1:LabDim
    for j=1:LabDim
        if ((i>=Obst1(1,1) & i<=Obst1(1,1)+OBS_SIZE) & (j>=Obst1(1,2) ...
            & j<=Obst1(1,2)+OBS_SIZE))
            Occupied(i,j)=1;
        end;
        if ((i>=Obst2(1,1) & i<=Obst2(1,1)+OBS_SIZE) & (j>=Obst2(1,2) ...
            & j<=Obst2(1,2)+OBS_SIZE))
            Occupied(i,j)=1;
        end;
        if ((i>=Obst3(1,1) & i<=Obst3(1,1)+OBS_SIZE) & (j>=Obst3(1,2) ...
            & j<=Obst3(1,2)+OBS_SIZE))
            Occupied(i,j)=1;
        end;
        if ((i>=Obst4(1,1) & i<=Obst4(1,1)+OBS_SIZE) & (j>=Obst4(1,2) ...
            & j<=Obst4(1,2)+OBS_SIZE))
            Occupied(i,j)=1;
        end;
        if ((i>=Obst5(1,1) & i<=Obst5(1,1)+OBS_SIZE) & (j>=Obst5(1,2) ...
            & j<=Obst5(1,2)+OBS_SIZE))
            Occupied(i,j)=1;
        end;
        if ((i>=Obst6(1,1) & i<=Obst6(1,1)+OBS_SIZE) & (j>=Obst6(1,2) ...
            & j<=Obst6(1,2)+OBS_SIZE))
            Occupied(i,j)=1;
        end;
    end;
end;

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                              CREATE A Quadrotor                            %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

xvertices = [-Sensor_Range+ci,Sensor_Range+ci,Sensor_Range+ci,...
            -Sensor_Range+ci];
yvertices = [-Sensor_Range+ri,-Sensor_Range+ri,Sensor_Range+ri,...
             Sensor_Range+ri];
sensor = patch(xvertices,yvertices,[.9,.9,.9]);
xvertices = [-Quadrotor+ci,Quadrotor+ci,Quadrotor+ci,-Quadrotor+ci];
yvertices = [-Quadrotor+ri,-Quadrotor+ri,Quadrotor+ri,Quadrotor+ri];
robot = patch(xvertices,yvertices,'ms');
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
```

```
%                                   SIMULATION                                  %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

for SimLen=0:5000 % For a simulation length of 5000 samples or Goal reach

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                 OBSERVATION MODEL                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Generate a series of readings for the robot based on the locations of
    % obstacles. The location of the center of the robot, and therefore the
    % center of the sensor, is given by RobotX and RobotY.

    RobotX = round((max(xvertices)+min(xvertices))/2);
    RobotY = round((max(yvertices)+min(yvertices))/2);
    % TODO change the sensor model
    % A zero means nothing detected
    Observation = zeros(2*Sensor_Range,2*Sensor_Range);
    for i=1:2*Sensor_Range
        for j=1:2*Sensor_Range
            SensorX=i+RobotX-Sensor_Range;
            SensorY=j+RobotY-Sensor_Range;
            if (SensorX<=0) | (SensorY<=0) | (SensorX>LabDim) | ...
               (SensorY>LabDim)
                continue;
            end;
            if (Occupied(SensorX,SensorY)==1)
            if(rand(1,1)<=TrueHit)
                Observation(i,j) = 1;
                end;
            end;
            if (Occupied(SensorX,SensorY)==0)
                if(rand(1,1) <= FalseAlarm)
                   Observation(i,j) = 1;
                end;
            end;

%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                     UPDATE PROBABILITIES USING BAYES RULE                     %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%           For each cell in the grid, update the probability of it be        %
%             it be occupied using Bayes rule given the observation           %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

        % If we observe a hit ===>
        if Observation(i,j) == 1
                Cell_Occ = TrueHit * p(SensorX,SensorY);
            % p(SensorX,SensorY) is the probability that a cell is occupied

            Cell_Unocc = FalseAlarm * (1-p(SensorX,SensorY));
            % 1-p(SensorX,SensorY) is the probability that
            % a cell is unoccupied
```

```
            % Normalize
            p(SensorX,SensorY) = Cell_Occ / (Cell_Occ + Cell_Unocc);

        % If do not observe a hit ===>
        else
                Cell_Occ = (1-TrueHit) * p(SensorX,SensorY);
            % p(SensorX,SensorY) is the probability that a cell is occupied
                    % Recall that (1-TrueHit) is the MissedDetection likelihood

            Cell_Unocc = (1-FalseAlarm) * (1-p(SensorX,SensorY));
            %  1-p(SensorX,SensorY) is the prob that a cell is unoccupied
                        % (1-FalseAlarm) is the TrueMiss likelihood

            % Normalize
            p(SensorX,SensorY) = Cell_Occ / (Cell_Occ + Cell_Unocc);

        end;

    % Update intensity to reflect probability that a cell is occupied.
        % Darker colors mean higher probabilities
            set(GridHandle(SensorX,SensorY),'color',...
         [1-p(SensorX,SensorY),1-p(SensorX,SensorY),1-p(SensorX,SensorY)]);

        end;
    end;


%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                ROBOT MOVEMENT MODEL                         %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

%================================================
% Retrieve the coordinates that define the robot
%================================================

    xvertices=get(robot,'XData');
    yvertices=get(robot,'YData');
    sensorxvertices = get(sensor,'XData');
    sensoryvertices = get(sensor,'YData');

    %============================
    % If within world boundaries
    %============================

    if ((min(xvertices + vel*cos(theta))>1) & (max(xvertices + ...
        vel*cos(theta))<LabDim-Sensor_Range))
        xvertices = xvertices + vel*cos(theta);
        sensorxvertices = sensorxvertices + vel*cos(theta);
    else theta=pi/2;
    end;

    if ((min(yvertices + vel*sin(theta))>1) & (max(yvertices + ...
```

```
        vel*sin(theta))<LabDim-Sensor_Range))
        yvertices = yvertices + vel*sin(theta);
        sensoryvertices = sensoryvertices + vel*sin(theta);
    else theta=pi/2;
    end;


    %=========================================================
    % Stop robot if it is within Dealta tolerance box of goal
    %=========================================================
    if ((abs (RobotX - cf) <= Del) && (abs (RobotY - rf) <= Del))
        break;
    end;
    set(robot,'XData',xvertices,'YData',yvertices);
    set(sensor,'XData',sensorxvertices,'YData',sensoryvertices);
    drawnow;

end;
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C:
MATLAB Code (Potential Field)

The following script[1] was created to capture data from the XBee wireless module and inserted into an array data structure.

```
% Created by Mejdi Ben Ardhaoui
% Modified by Professor Timothy Chung

%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                   GetData.m                                  %
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%

%close all;
% Create Serial Port
s = serial('COM4', 'BaudRate', 57600, 'DataBits', 8, 'Parity', 'none', ...
    'StopBits', 1, 'FlowControl', 'none');
disp('Opening Connection to Quadrotor')
% Open port to start data transfer
fopen(s);

% Create an array ( Buffer)
a=[];

% Total number of beams used
N=546;

% loop through the beams and store data
for i=1:1:546% (N~=0)
a=[a ; fgetl(s)];
end;

% Convert string array to number array
ranges = str2num(a);

fclose(s);
delete(s);
clear s
```

[1] All source code is available online at `http://faculty.nps.edu/thchung` under `Resources, Software`.

The following script is used to compute the angle between a given beam and an obstacle detected by this beam.

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                Compute_angle.m                              %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
function angle_rep=compute_angles(posn)
% Resolution: Number of beams divided by 180 degrees
t=0.00576;
% Initialize theta in global coordinates
theta=posn(3)-pi/2;
% Compute angles between every obstacles and the center of the Robot
for i=1:546
    angle_rep(i)=theta+(i-1)*t;
end

return;
```

Every time we get a range return different from zero, we update the map by changing the value at this position from zero to one. One will be interpreted as an obstacle in the other scripts.

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                UpdateMap.m                                  %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
function map=updateMap(posn,ranges,mylab,map)
t=0.00576;
theta=posn(3)-pi/2;
ranges=round(10*ranges);
for i=1:1:546
        X(i)=round(posn(1) + ranges(i)*cos((i*t)-theta));
        Y(i)=round(posn(2) + ranges(i)*sin((i*t)-theta));

    if ranges(i)~=0
        % Test Borders limits
        if (X(i)<=0) || (Y(i)<=0) || (X(i)>mylab) || (Y(i)>mylab)
            break;
        end; %Don't do anything outside the world boundaries
        map(X,Y)=1;
    end
end
return;
```

This script will change coordinates from Local to Global coordinates.

```
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                  ComputePosn.m                               %
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
function [X,Y]=ComputePosn(posn,ranges,mylab)
t=0.00576;
theta=posn(3)-pi/2;
ranges=10*ranges;
for i=1:546
    if ranges(i)~=0
        X(i)=posn(1) + ranges(i)*cos(((i)*t)-theta);
        Y(i)=posn(2) + ranges(i)*sin(((i)*t)-theta);

        % Test Borders limits
        if (X(i)<=0) || (Y(i)<=0) || (X(i)>mylab) || (Y(i)>mylab)
            break;
        end
    else
       X(i)=posn(1);
       Y(i)=posn(2);
    end

end
return;
```

The following script will draw and visualize the Robot and the sensor.

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                        rob.m                                %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
function [obs_ptsX,obs_ptsY] = rob(posn,ranges)
rad=5;
clf;
hold off;
mylab=140;
map=zeros(mylab,mylab);
axis([0 mylab 0 mylab]);
axis xy
hold on;

%draw a little circle for the robot
angs = 0:pi/10:2*pi;
x = posn(1) + rad*cos(angs);
y = posn(2) + rad*sin(angs);
fill (x,y,'b');
%posn(3)=deg2rad(posn(3));

%Draw line in direction of heading
% line from center of robot to heading direction
h_line = line([posn(1), posn(1) + (rad)*cos(posn(3))], [posn(2),...
         posn(2) + (rad)*sin(posn(3))]);
set(h_line, 'Color', 'black','LineWidth',2)
ranges=round(10*ranges);
ranges( (ranges==0) ) = max(ranges);

t=0.00576;
theta=posn(3)-pi/2;
for i=1:1:546
    X(i)=round(posn(1) + ranges(i)*cos(theta+i*t));
    Y(i)=round(posn(2) + ranges(i)*sin(theta+i*t));
    %plot(X,Y,'g.');
    if ranges(i)~=0
        % Test Borders limits
        if X(i)<=0
            X(i)=0;
        end
        if Y(i)<=0
            Y(i)=0;
        end

        if X(i)>mylab
            X(i)=mylab;
        end
        if Y(i)>mylab
            Y(i)=mylab;
        end
%             break;
```

```
%          end; %Don't do anything outside the world boundaries
          plot(X,Y,'g.');
      end
      h_line = line([posn(1), posn(1) + ranges(i)*cos(theta+i*t)], ...
                [posn(2), posn(2) + ranges(i)*sin(theta+i*t)]);
      set(h_line, 'Color', 'red')

%Draw line in direction of heading
% line from center of robot to heading direction
h_line = line([posn(1), posn(1) + (rad)*cos(posn(3))], [posn(2), ...
          posn(2) + (rad)*sin(posn(3))]);
set(h_line, 'Color', 'black','LineWidth',2)
%TODO  draw an arrow showing the resultant force
end


% Determine coordinate locations for where scan points hit obstacles
% (less than 90% of max range)
obs_ptsX = X( ranges< 0.90*max(ranges) );
obs_ptsY = Y( ranges< 0.90*max(ranges) );
```

In order to determine the new heading, we first need to compute the repulsive and attractive forces. The resultant of these forces represent the new heading for the robot at this position.

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                compute_force.m                          %
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
function [SumFrepX,SumFrepY,SumFattX,SumFattY]=compute_force(Xgoal,Ygoal...
                                                    ,posn,ranges,X,Y,angle_rep)
% ranges(i) : distance from the robot position to the obstacle(i)
% d0        : the influence distance of the force
% n         : adjustable constant for the repulsive forces.
% Frep(i)   : Repulsive force from obstacle(i)
% posn(1)   : X postion of the robot
% posn(2)   : Y postion of the robot
% X(i)      : X postion of an obstacle(i)
% Y(i)      : Y postion of an obstacle(i)
% theta(i)  : angle from robot position to obstacle(i)
% Xgoal     : Goal X coordinate
% Ygoal     : Goal Y coordinate
% k         : Adjustable constant for the attraction force
% angle_att : angle between Goal and current position of the robot
%+++++++++++++++++++++++++++++++++++++++++++++
% Initialize variables
%+++++++++++++++++++++++++++++++++++++++++++++
k=3;
n=0.05;
d0=1;
%+++++++++++++++++++++++++++++++++++++++++++++
%compute angle to goal
%+++++++++++++++++++++++++++++++++++++++++++++
deltaX=Xgoal-posn(1);
deltaY=Ygoal-posn(2);
Dist2Goal=sqrt((deltaX)^2+(deltaY)^2);
angle_att = atan2(deltaY, deltaX);
%+++++++++++++++++++++++++++++++++++++++++++++
Frep=[0;0];
Fatt=[0;0];
for i=1:546
    if ((ranges(i)~=0) && (ranges(i)<=d0))

        %Repulsive Forces

        FrepX(i)=n*cos(angle_rep(i));
        FrepY(i)=n*sin(angle_rep(i));

        %Attractive Forces

        FattX(i)= k*cos(angle_att);
        FattY(i)= k*sin(angle_att);
    end
end
SumFrepX=-sum(FrepX);
```

```
SumFrepY=-sum(FrepY);
SumFattX=sum(FattX);
SumFattY=sum(FattY);
end
```

This is the main code for the project. This script will call all the scripts above in order to:

- draw the map and plot obstacles.

- drive the robot from a given start point to the goal

```
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
%                                    mapBuildNew.m                             %
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++%
clc; close all; clear;

% Define robot trajectory data structure
posn_hist = [];
obs_pts_hist = [];

% INITIALIZE MAP
hold off;
mylab=140;
map=zeros(mylab,mylab);
axis([0 mylab 0 mylab]);
axis xy
hold on;
Xgoal = input('Input Goal X position: ');
Ygoal = input('Input Goal Y position: ');
% LOOP UNTIL EXITED
while (true)


    % ASK FOR USER INPUT
    % Either robot position
        posn_x = input('Input robot X position: ');
        posn_y = input('Input robot Y position: ');
        posn_th = input('Input robot THETA heading in Radians: ');
        posn=[posn_x,posn_y,posn_th];

    % Update history of robot position
        posn_hist = [posn_hist; posn];

    % GET LASER DATA
    % Returns variable: ranges ([546x1] vector of range data)
        GetData

    % Compute X position and Y position for obstacles
        [X,Y]=ComputePosn(posn,ranges,mylab);

    % POPULATE AND UPDATE MAP
    % Updates variable 'map' with 1/0 occupancy locations
        map=updateMap(posn,ranges,mylab,map);

    % PLOT MAP, ROBOT
       [obs_ptsX, obs_ptsY] = rob(posn,ranges);
```

```
        obs_pts_hist = [obs_pts_hist; obs_ptsX', obs_ptsY'];

    % Compute Angles to Obstacles
        angle_rep=compute_angles(posn);

    % Compute Forces
        [SumFrepX,SumFrepY,SumFattX,SumFattY]=compute_force(Xgoal,Ygoal,...
                                                posn,ranges,X,Y,angle_rep);
        Fx=SumFrepX+SumFattX;
        Fy=SumFrepY+SumFattY;

    %Navigation
       posn(1) = posn(1)+Fx*.005;
       posn(2)= posn(2)+Fy*.005;
       posn(3) = atan2(Ygoal-posn(2),Xgoal-posn(1));
       figure(1)

    % CHECK IF EXIT
    % Check if complete
         decision = input('Continue with next robot location? Y/N: ', 's');
         if (strcmp(decision, 'N'))
             break;
         end

end      % end while(true)

% Output robot trajectory
posn_hist
hold on;

% Plot all obstacle hits
plot(obs_pts_hist(:,1), obs_pts_hist(:,2), 'go');

% Plot robot trajectory on figure
plot(posn_hist(:,1), posn_hist(:,2), 'b.-');
```

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia
2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California
3. Dr. Timothy H. Chung, Ph.D.
   Naval Postgraduate School
   Monterey, California
4. CDR Duane Davis, USN, Ph.D.
   Naval Postgraduate School
   Monterey, California