Implementation of a Probabalistic Technique for Robotic Mapping

Adam A. Wright '10
Nathan M. Swaim '10
Orko Momin '10

Faculty Advisor: Dr. David J. Ahlgren

May 17, 2010

Abstract

Exploration of unmapped terrain like Mars or other planets requires autonomous robots to be able to steer around obstacles and to know their position with respect to key landmarks. Reliance on measures of displacement from an initial position to build a map suffers from cumulative inaccuracies that can render such a map useless. This project applies a technique called Simultaneous Localization and Mapping (SLAM) to dramatically reduce cumulative error. Key to building this map and the resulting location history is the use of a camera and laser rangefinder to detect and measure obstacle position relative to the robot. The high accuracy of multiple distance scans combined with inaccurate displacement measurements and a probabilistic mathematical method, Kalman filtering, yield a very accurate map. This map is sufficiently accurate that it may be used to make navigation decisions. The primary project objective is to develop a robot capable of performing SLAM. The second key objective in this project is to package the SLAM code and algorithms into an undergraduate tutorial, so subsequent undergraduates will be able to use this work as a building block, and to incorporate SLAM into robotics projects more easily, advancing the average level of robotic intelligence.

Contents

1 Introduction

1.1 Motivation

Many robots are reactive, meaning they perceive their environment at the current time, and react based only on that information. There is no memory of what the robot has already done. An example of a reactive robot is the iRobot Roomba. The Roomba drives around your living room, vacuuming while it moves, until it hits a wall. At that point it turns around at a random angle, and moves in that direction until it hits another wall. The robot only makes decisions based on the front bump sensor. It is not concerned with keeping track of where it has already vacuumed, since it relies on random turn angles to vacuum the whole room. This kind of reactive algorithm is good enough for vacuuming your living room, but when a robot must navigate more complex environments, more advanced perception is required.

1.2 Problem Statement

One type of advanced perception is mapping. For the purposes of this project, it is assumed that the robot does not have any previous knowledge of its environment and can only build up a map using its on board sensors. This assumption is made so that the results of this project may be applied to robots that do not benefit from human interaction. A "map" that the robot could use would be in the form of coordinates of every obstacle in the environment and the robot's coordinates. A robot with such a map could plan an efficient path through the environment to some goal (A goal might be GPS coordinates).

1.3 Objectives

To solve this problem, a robot will be built that is capable of performing Simultaneous Localization And Mapping (SLAM). SLAM is a probabilistic technique that allows a robot to build a map of its environment as it operates. A secondary objective is to produce a tutorial for undergraduate engineering students explaining how to use SLAM for their own projects. This subject has been inaccessible due to lack of detailed understanding of important mathematical concepts such as probability and stochastic linear systems. This tutorial will contain a preliminary section explaining the mathematics that students may not have encountered. The rest of the tutorial will be devoted to explaining the concepts behind SLAM and giving readers a walk-through of some of the experiments completed in this project. By completing the tutorial, students will be able to achieve the same end result as this project and gain an understanding of how to incorporate SLAM into their projects.

2 Simultaneous Localization And Mapping (SLAM)

2.1 Localization and Mapping

Before getting into SLAM, localization and mapping must be defined. **Mapping** is a robotics term that refers to constructing a list of objects and their locations in the environment relative to a reference frame [6, p152]. In SLAM, these objects are called landmarks. **Localization** is determining the robot's position in a reference coordinate frame when already given a map. The robot must use its sensors to detect features and to associate them with features from the map [6, p5].

As its name suggests, SLAM does both of these tasks simultaneously. Before SLAM, it was thought that doing these tasks separately would yield better results as localization and mapping

were considered to be mathematically unrelated. It has since been found that the opposite is actually true, these two tasks are related, and SLAM takes advantage of this relationship [7].

2.2 Odometer Based Mapping

Before discussing SLAM, we will first introduce a technique for mapping that has a substantial issue due to cumulative error, called odometer based mapping. It is assumed that

1. The robot can make measurements of landmarks and distinguish one landmark from another and
2. The robot can compute its change in position after moving using an on-board odometer.

Figure 1 shows a diagram of this process. The robot at time k takes a measurement of the relative distance to landamrks 1 and 3. From these measurements, the robot can construct a map with the coordinates of two of the landmarks. Then, the robot moves from the location at time k to the one

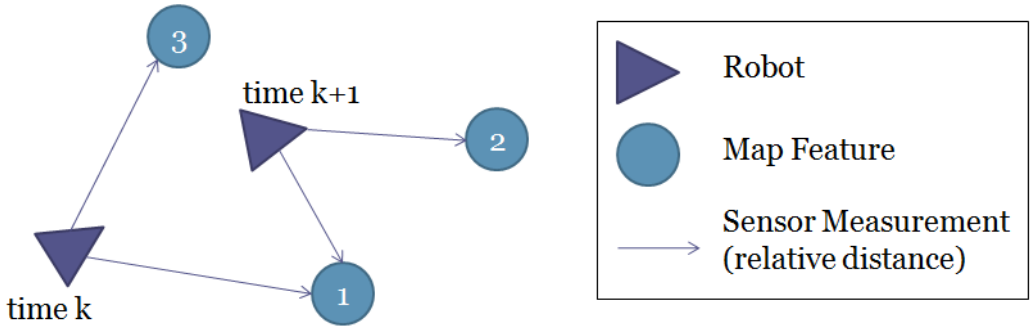

Figure 1: Diagram showing the steps in odometer based mapping.

at time $k+1$ and computes the change in position. However, there is error in the computed change in position, so the estimated position of the robot at time $k+1$ also has some error associated with it. When the robot gets to the location shown at time $k+1$, the robot takes a measurement of landmark 2, in addition to the already observed landmark 1. The position at time k and the change in position can be used to compute the positions of the two landmarks with respect to the reference coordinate frame. Now the robot has two different position measurements of landmark

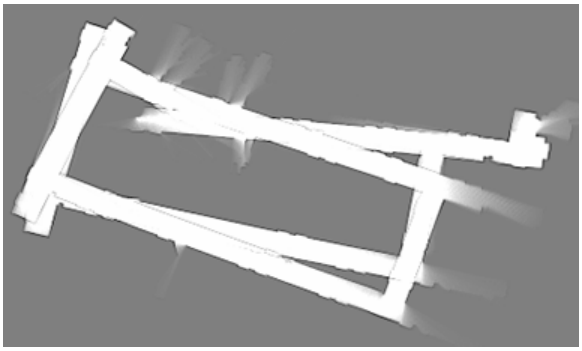

Figure 2: Shows a map of a rectangular building constructed using odometer based mapping.

1 with respect to the reference coordinate frame. The second measurement is polluted by error

from the robot's computation of its change in position. In an attempt to intelligently estimate the position of landmark 1, the robot averages the two measurements of the landmark's positions. This noise from the change in position builds up over time, causing the estimated position of landmarks to be skewed. This effect is demonstrated in Figure 2. Since the map in Figure 2 was generated using odometer based mapping, the cumulative error associated with measuring change in position caused the map to become bent.

2.3 The Solution: Kalman Filter SLAM

SLAM solves this problem by statistically combining the robot position data and the landmark measurement data. SLAM has two main steps:

1. Landmark Extraction, and
2. Data Fusion.

Landmark extraction is the process of using sensors to measure the locations of landmarks in the environment. Part of this process, called data association, is is the processes of recognizing the same landmark from different locations. Various techniques for landmark extraction are discussed in [8] [9], and the specific algorithms used for this project will be discussed in detail.

The data fusion step requires a statistical approach. Different applications of SLAM use different statistical techniques. The most commonly used technique (and the one used in this project) is called the Kalman filter. The Kalman filter assumes that the map and robot pose (position and heading) can be represented using one system state in a state-space model.

The same assumptions are made as the previous section on odometer based mapping. They are re-stated here:

1. The robot can make measurements of landmarks and distinguish one landmark from another and
2. The robot can compute its change in position after moving using an on-board odometer.

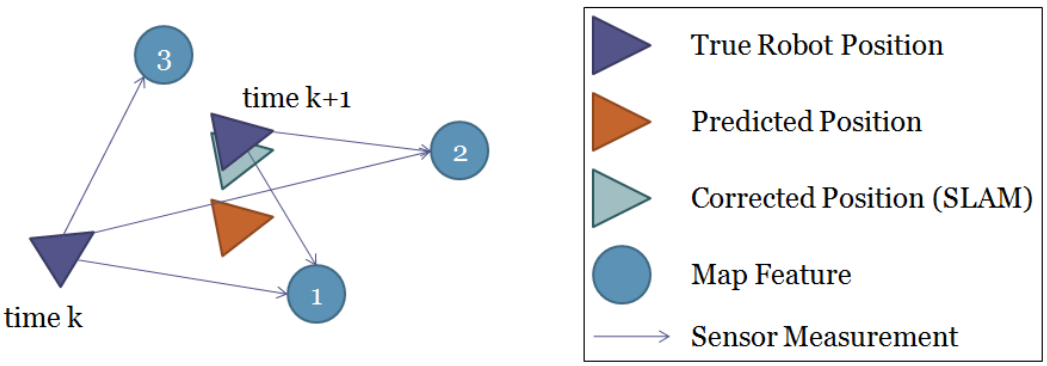

Figure 3: Diagram of the predict-update process.

Kalman filter data fusion has two steps: **predict** and **update**. Figure 3 shows the predict-update structure of Kalman filter based SLAM. The robot starts at time k with some estimate of its current position. Before moving to the next position, it takes measurements of landmarks 1, 2, and 3. Then the robot moves, arriving at the second purple triangle at time $k+1$. Odometer data is used to **predict** the robot's new position. However, there will be some error as discussed in section 2.2,

so instead of the robot being at the predicted location, it will be at the true position, shown by the purple triangle. Next the robot takes measurements of all landmarks that are in view. The robot uses the Kalman filter to incorporate sensor measurements at time $k+1$. This step is called the **update** step. In addition to using these new measurements of landmarks 1 and 2 to compute a better estimate of their positions, the Kalman filter also uses this information to update the predicted robot position. Thus, the predicted position is corrected to a more accurate estimate of its position, shown by the light blue triangle. In this way, SLAM combats the cumulative error in the odometer measurements.

3 Statistical Estimation: Kalman Filter

3.1 Kalman Filter Overview

In order to use the Kalman filter, the system must be described using a stochastic linear state-space model. The model used has some error terms in the state and measurement equations to model the cumulative error in the odometer and the uncertainty in the sensor measurements. Each of the vectors in this state-space model are considered to be random vectors so that uncertainties about each vector may be defined. The consideration of uncertainty will be most important for the noise terms in the system. The state equation is

$$\vec{x}(k+1) = F(k)\vec{x}(k) + G(k)\vec{u}(k) + \vec{v}(k),$$

where $\vec{x}(k)$ is the state vector at time k, $\vec{u}(k)$ is the input vector at time k, $\vec{v}(k)$ is the process noise at time k, $F(k)$ describes the system dynamics at time k, and $G(k)$ describes how the input vector affects the transition to the next state. The process noise is assumed to be white and Gaussian with mean 0 and covariance matrix $V(k)$. Whiteness means that the process noise at any time j is not correlated with the process noise at any other time i.

The measurement equation is

$$\vec{y}(k) = H(k)\vec{x}(k) + \vec{w}(k),$$

where $\vec{y}(k)$ is the measurement at time k, $\vec{w}(k)$ is the measurement noise at time k, and $H(k)$ describes the relationship between the system state at time k and the measurement at time k. Here, $\vec{w}$ describes the uncertainty in the measurement and is also assumed to be white and Gaussian with mean 0 and covariance matrix $W(k)$. H is assumed to be full row rank for any k.

All of these assumptions may become valid or invalid depending on the specific application. The Kalman filter is commonly re-derived with different assumptions depending on the application [5]. For SLAM, the assumption of linearity is usually too restrictive. The most common form of the Kalman filter used for SLAM is called the Extended Kalman Filter (EKF). The EKF works by linearizing the system and then applying the linear Kalman filter.

The estimate of the system state at time k is called $\hat{x}(k)$. In addition to the state estimate, the filter also provides an estimate of the error covariance matrix, called $P(k)$. This matrix is the covariance of the error vector $\vec{x}(k) - \hat{x}(k)$. It can be thought of as a measure of how good or bad the estimate is at time k. Given the assumptions discussed above, the Kalman filter provides recursive equations for computing $\hat{x}(k+1)$ and $P(k+1)$, given

1. The previous estimate of the system state, $\hat{x}(k)$,
2. The previous estimate of the error covariance, $P(k)$,

3. The system input at time k, $\vec{u}(k)$, and

4. The system measurement at time $k+1$, $\vec{y}(k+1)$.

A flowchart for the algorithm is shown in Figure 4. The equations are given in two steps: prediction

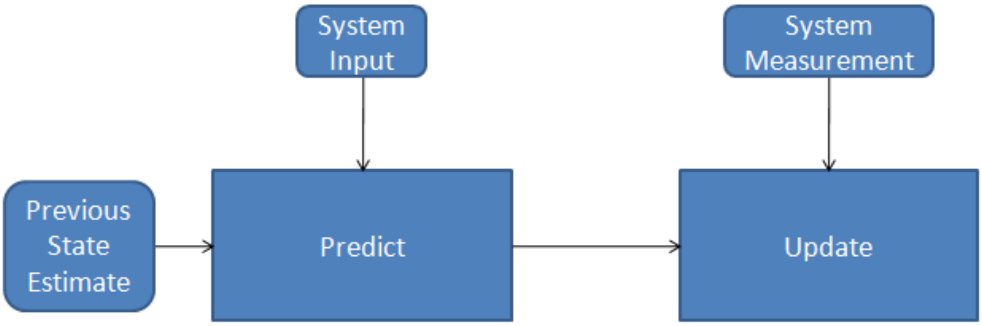

Figure 4: Flowchart showing the steps in the Kalman filter algorithm.

and update. The prediction equations are

$$\hat{x}(k+1|k) = F(k)\hat{x}(k) + G(k)u(k)$$
$$P(k+1|k) = F(k)P(k)F(k)^T + V(k)$$

This is an intermediate prediction of the system state based only on the input, $\vec{u}(k)$. The following equations are the update equations

$$\hat{x}(k+1) = \hat{x}(k+1|k) + Rv$$
$$P(k+1|k) = P(k+1|k) - RH(k+1)P(k+1|k),$$

where

$$v = \vec{y}(k+1) - H(k+1)\vec{x}(k+1|k)$$
$$S = H(k+1)P(k+1|k)H(k+1)^T + W(k+1)$$
$$r = P(k+1|k)H(k+1)^T P(k+1|k)S^{-1}[5,\ \text{p285}].$$

These equations incorporate the information from the newest measurement $\vec{y}(k+1)$. Once the system is described using the state and measurement equations above, these prediction and update equations will form the core of the SLAM algorithm [5, p273].

3.2 Kalman Filter SLAM

In this section, the system model for SLAM will be constructed. This model is a slightly expanded version of the one presented in [5, p295]. The state vector will be a $(3+2n) \times 1$ vector, where n is

the number of landmarks in the system. The vector is defined as

$$\vec{x}(k) = \begin{bmatrix} x_r(k) \\ y_r(k) \\ \theta_r(k) \\ x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix}$$

where $x_r(k)$, $y_r(k)$, and $\theta_r(k)$ are the robot x position, y position, and heading, respectively, with respect to the reference coordinate frame, and x_i and y_i are the positions of the landmarks. Figure

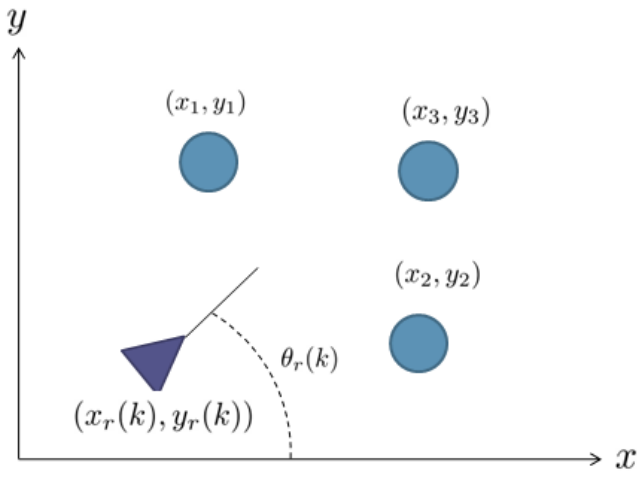

Figure 5: Visualization of the quantities included in the state vector for $n = 3$.

5 visually shows these quantities for a system with 3 landmarks ($n = 3$). The first three elements make up the robot pose and each (x_i, y_i) pair after that is the position of the ith landmark.

The input vector contains the change in robot pose from time k to time $k+1$. This information is read from the odometer. The vector is defined as

$$\vec{u}(k) = \begin{bmatrix} \Delta x_r(k) \\ \Delta y_r(k) \\ \Delta \theta_r(k) \end{bmatrix}$$

where $\Delta x_r(k)$, $\Delta y_r(k)$, and $\Delta \theta_r(k)$ are the change in x position, y position, and robot heading, respectively. Figure 6 shows a visualization of the input vector.

Now $F(k)$ and $G(k)$ will be defined so that the state equation advances the robot position without changing the position of each landmark. $F(k)$ is defined as the $(3+2n) \times (3+2n)$ identity

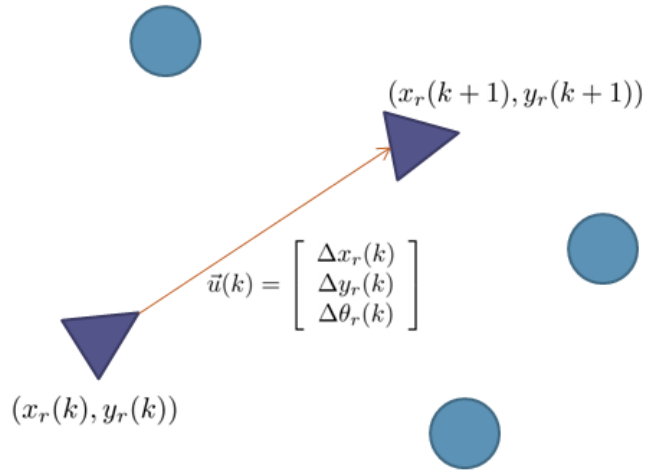

Figure 6: Visualization of the input vector for $n = 3$.

matrix, so that $F\vec{x}(k) = \vec{x}(k)$. $G(k)$ is the $(3 + 2n) \times 3$ matrix defined by

$$G(k) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}.$$

By defining $G(k)$ in this way, the new state vector is computed as the old state vector plus the change in robot pose. So the robot is advanced to its next position as desired, and the landmarks are not affected because of the $2n$ rows of zeros at the end of the vector $G\vec{u}(k)$.

The process noise $\vec{v}(k)$ is never known directly, since it is in the state equation for the purpose of modeling noise associated with the system advancing from time k to time $k + 1$. However, in order to give the Kalman filter some information about this noise, the covariance matrix of $\vec{v}(k)$, called $V(k)$, will be given as an initial condition. The covariance of $\vec{v}$ is a $(3n+2) \times (3n+2)$ matrix defined by

$$V(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}.$$

The 1's in the first three diagonals describe the amount of uncertainty in the first three elements in the state vector due to any noise in the state equation. One example of a source of noise is wheel slippage. If the wheels of the robot slip, the odometer reading will be erroneous. The rest of the matrix is filled with 0's because there is no uncertainty in the transition of landmark position from time k to $k + 1$. The landmarks are assumed to be stationary objects, and thus there is no error associated with their location.

Next, we move on to the measurement vector. It is defined as

$$\vec{y}(k) = \begin{bmatrix} x_1 - x_r(k) \\ y_1 - y_r(k) \\ x_2 - x_r(k) \\ y_2 - y_r(k) \\ \vdots \\ x_n - x_r(k) \\ y_n - y_r(k) \end{bmatrix}$$

since the measurement in SLAM is a measurement of the position of each landmark that is in view relative to the robot's position. Figure 7 shows the quantities included in the measurement vector visually. The matrix H relates the measurement vector to the state vector. First, we define H as

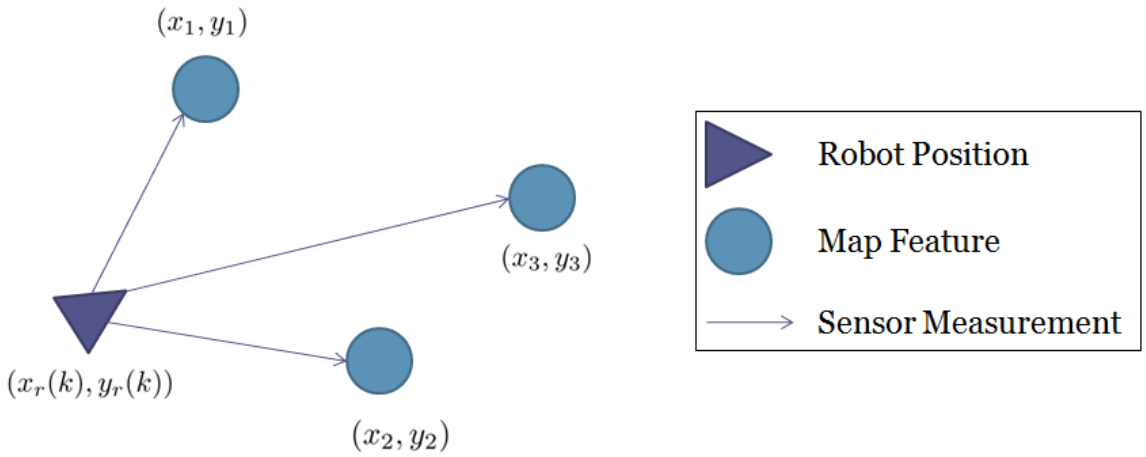

Figure 7: Visualization of the measurement vector for $n = 3$.

a block matrix as follows

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{bmatrix}.$$

Each H_i relates the ith landmark measurement to the system state. So H_i is defined by

$$H_i = \begin{bmatrix} -1 & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 & 1 & \cdots & 0 \end{bmatrix}.$$

As an example, let the number of landmarks, n, be 3. Assume that the robot has just used its sensors to detect all landmarks that are currently visible, and it has detected landmark 1 and landmark 3. Then, $\vec{y}(k)$ would be

$$\vec{y}(k) = \begin{bmatrix} x_1 - x_r(k) \\ y_1 - y_r(k) \\ x_3 - x_r(k) \\ y_3 - y_r(k) \end{bmatrix}.$$

So, we want to build H using H_1 and H_3. H_1 and H_3 would be

$$H_1 = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$H_3 = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and the measurement equation would be

$$\vec{y}(k) = H\vec{x}(k) + \vec{w}(k)$$

$$\begin{bmatrix} x_1 - x_r(k) \\ y_1 - y_r(k) \\ x_3 - x_r(k) \\ y_3 - y_r(k) \end{bmatrix} = \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_r(k) \\ y_r(k) \\ \theta_r(k) \\ x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \end{bmatrix} + \vec{w}(k)$$

$$= \begin{bmatrix} x_1 - x_r(k) \\ y_1 - y_r(k) \\ x_3 - x_r(k) \\ y_3 - y_r(k) \end{bmatrix} + \vec{w}(k)$$

Since the expected value of the noise term $\vec{w}(k)$ is 0, the left and right sides match up properly.

The covariance matrix for $\vec{w}(k)$, called $W(k)$, is the $2n \times 2n$ identity matrix

$$W(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix}.$$

Defining $W(k)$ this way tells the Kalman filter that each element of the measurement vector has some uncertainty associated with it.

4 Project Goals

The goals for this project are centered around producing a usable software framework in National Instruments LabVIEW that can be used for SLAM. In order to make the project tractable, the focus will be on the data fusion part of SLAM. For landmark extraction, the following assumptions are made:

1. Landmarks are colored cylinders, and there are only three. The colors are red, green, and blue, and

2. At least one landmark is always in view of the robot's sensors.

The main goal of the project is to produce a robot capable of performing SLAM. In this goal the following tasks must be completed:

1. Landmark extraction,
2. Kalman filter research/Simulation,
3. Kalman filter SLAM with one landmark,
4. Multi-Landmark Kalman filter SLAM, and
5. SLAM-driven navigation.

The second goal is to write a tutorial explaining how to achieve the results obtained at the end of this project. The tutorial will focus on explaining the math needed to understand the Kalman filter and will provide documented LabVIEW source code for use in other projects.

5 Division of Labor

This section describes how the workload was divided up among the three group members, Adam Wright, Orko Momin, and Nathan Swaim. In the first semester, Adam Wright researched the Kalman filter and developed a simulation based on the Kalman filter. Adam Wright and Orko Momin did the mechanical additions to the iRobot Create so that the sensors and processor could be mounted on the robot. Orko Momin and Nathan Swaim did research into landmark extraction techniques. This included analysis of how easy each one would be to implement and use for SLAM. Nathan Swaim and Adam Wright developed LabVIEW VI's that control the iRobot Create and gather data from its shaft encoders. Nathan and Adam also calibrated the laser rangefinder and the webcam. In the second semester, Nathan Swaim developed the image processing algorithm and code. Adam Wright developed the program architecture based on the requirements of the Kalman filter. All team members tested the results for the various experiments conducted, all of which will be discussed in detail later in this report. Nathan Swaim and Adam Wright troubleshot the code for multiple landmark extraction and the SLAM code for multiple landmarks. Finally, Orko Momin developed a navigation algorithm that used SLAM and wrote the program for it. All of this work will be discussed in detail throughout this report.

6 Our Robot

6.1 Robot Configuration

An iRobot Create acts as the base of the robot. The Create is equipped with two differential drive, DC motors and shaft encoders that provide information regarding the motion of the robot. In addition to the Create's shaft encoders, the system uses a Hokuyo laser rangefinder and a USB web-camera as its sensors. The system is controlled using a netbook running LabVIEW 2009. To make space for the various sensors and the netbook, a mechanical platform was placed on the iRobot Create. The platform is made out of four pieces of angled iron. The angled iron pieces act as risers that are bolted to the base of the Create. Two Plexiglas sheets were fitted between the risers, providing space for the netbook. The remainder of this section is dedicated to explaining the various components of the design.

Figure 8: Diagram of all components of the robot and how they are connected.

6.1.1 Sony VAIO Netbook With LabVIEW

The robot's software is run on a Sony VAIO netbook. As illustrated in Figure 8, it is the central unit of the robot and all sensors and the iRobot Create interface with it. The Netbook is a compact PC which is ideal in size for the purpose of this project. The Netbook interfaces with the Create using serial communication which is discussed in detail in later sections. The webcam and the Hokuyo Laser Rangefinder are connected to the netbook using USB ports. The netbook is equipped with National Instrument's LabVIEW Robotics 2009. LabVIEW is a visual programming environment and platform that can be used for data acquisition, instrument control and automation. LabVIEW is particularly useful in performing the various computations and autonomous control that this project requires and was also the primary choice given the level of experience the members of the team have had with working in the LabVIEW environment. LabVIEW allows users to create programming blocks called Virtual Instruments (VI). Built-in VIs can be used to run programming loops, perform mathematical and boolean processes, communication and data manipulation amoung other tasks and allows users to make their own VIs. Using these built-in VIs, customized VIs were created to build the system architecture and all other system software including the SLAM algorithm, Kalman filter and navigation algorithm.

6.1.2 iRobot Create

As mentioned previously, the base of the robot is an iRobot Create. It is a robotics development platform based on the iRobot Roomba vacuum cleaner. The Create is modified to contain a cargo bay with a 25 pin port for digital and analog input and output. It is additionally equipped with a

Figure 9: Picture showing the front of the robot.

serial port that can be used to communicate with the Create. The netbook uses RS 232 protocol to communicate with the Create Open Interface (OI). The Create's serial port is configured to receive 8 bit opcodes at 57600 baud, the default rate. The serial port is then used to send initialization code, actuation commands and receive distance and angle information from the shaft encoders. These tasks are explained below and are done through specific VIs programmed in LabVIEW.

1. Initialization: Sending an op code value of 128_{10} starts the Create OI. The next step is to choose the operating mode. The OI can be operated in four modes: Off, Passive, Safe and Full. Sending an opcode value of 131_{10} puts the OI in safe mode enabling standard safety features, meaning that if it is next to the edge of a table and the sensors detect the edge of the table, the robot will shut down.

2. Actuator Control: Sending an op code of value 145_{10} allows the netbook to control the Create's forward and backward motion. The op code is followed by four data bytes that set the wheel speeds in millimeters per second. The first two bytes specify the right wheel speed while the next bytes specify the left wheel speed. A positive velocity makes a wheel drive forward while a negative velocity makes the wheel spin backward.

3. Encoder Information: In addition to sending motor commands to the Create, the RS 232 port can be used to obtain distance and angle information. This information is used in the SLAM process. Sending an op code of 142_{10} requests the OI to send a packet of sensor data bytes. This op code is followed by the Packet ID for the packet that needs to be retrieved. Packet 19 returns the distance the Create has travelled in millimeters since the distance was last requested. Positive values reflect forward motion while negative values indicate backward motion. Similarly, packet 20 contains the angle in degrees the Create turned since the last time this information was requested. Counter-clockwise angles are positive while clockwise angle values are negative. The values are stored in a counter, so care must be taken to read them frequently enough so that they do not roll over to 0 and start counting over [1].

6.2 Sensor Setup

A Hokuyo laser rangefinder coupled with a standard Logitech USB webcam are used for the purpose of landmark extraction. Recall that landmark extraction is an integral part of SLAM and refers to the process of extracting various features in the environment that can be recognized by the robot repeatedly to help the mapping and localization processes. A careful study of existing literature shows that there are multiple different approaches that have been taken to landmark extraction. These include the use of a camera rig to obtain panoramic images [2], the use of laser cameras and the technique of segment matching [3], and the use of a single camera to obtain quadrangular images[4]. After conducting a survey of the various landmark extraction techniques, it was realized that most techniques were suited to specific environments and sensor setups. The team then decided to specify the environment that would be used to develop the SLAM algorithm and make sensor choices based on it. The landmarks used would be cylinders of various colors. The final setup contains red, blue, and green landmarks. For these specific landmarks, using a laser rangefinder and a webcam is simple and appropriate. The webcam is used to detect and distinguish between objects in the environment using their color, while the laser rangefinder is used to get distance information for each detected landmark. Once the team decided on using a laser rangefinder and a camera, a survey was performed to decide on the best possible combination of specific sensors to be purchased. Table 1 below highlights the different options that were discussed. Taking into consideration the

Sensor	**Positives**	**Negatives**
Laser Rangefinder	Accurate and precise data on distance of objects from robot	Complicated algorithms to differentiate between environment and landmarks
Stereo-vision Camera	Able to detect landmarks and get distance information	Not easy to implement - would have to develop a driver
Camera	Easy to detect landmarks by color	More difficult to get distance information

Table 1: Original sensor possibilities

budget allotted to the team and the quality of the scan obtained, the team decided upon a Hokuyo URG-04LX-UG01 laser rangefinder. It has a sensing range of 5.6 meters and the measurement accuracy is known to be within three percent tolerance of the current reading. Additionally the scanning rate is 100 ms across a 240^o field of view, which is sufficient for the purpose of this project. Figure 10 shows a single measurement made by the laser rangefinder. Although only one measurement is shown here, the laser rangefinder makes over 700 of these measurements in each scan, evenly distributed over the 240^o field of view. The black arrow is a vector representing the measurement. The distance of the measurement, r, is read in mm, and the angle is read as the angle of the vector $\vec{m}$ from the x axis, ranging from -120^o to 120^o. A publicly available LabVIEW VI performs the serial communication with the Hokuyo laser rangefinder in order to extract this data. The data returned is in the form of two arrays: one array contains all the distance measurements, and the other array contains all of the angles corresponding to each distance.

The Logitech USB camera was chosen for its easy plug and play feature and ease of image extraction through LabVIEW. The image data is returned from the camera as a special LabVIEW type, and built in image processing functions are used to make requried computations.

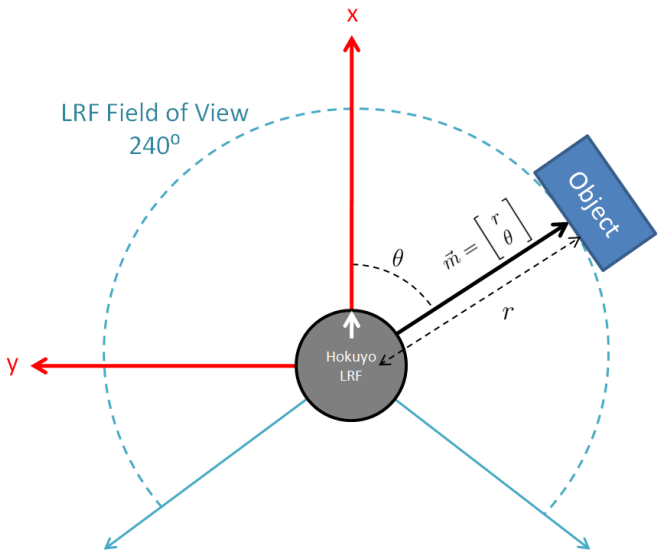

Figure 10: Diagram of the laser rangefinder showing one measurement in detail. The solid black arrow is the (range, bearing) measurement and the blue box is the object being measured.

7 Experiments

The following sections describe in detail the experimental setup, software, and results for each experiment performed as part of this project. Each experiment uses the Kalman filter to estimate the system state for a specific system. As a better understanding of the Kalman filter was gained, the experiments were made closer to the eventual goal of the project: navigation driven by multi-landmark SLAM.

7.1 Simulator

7.1.1 Experiment Description

The system modeled in the simulator is that of a car driving in a straight line with constant acceleration, attempting to estimate its velocity and position (in one dimension) at time k. A measurement of the car's speed is used to reinforce the estimate of the car's position and velocity. As such, the state vector is defined as

$$\vec{x}(k) = \begin{bmatrix} d(k) \\ v(k) \end{bmatrix}$$

where $d(k)$ is the car's position at time k and $v(k)$ is the car's velocity at time k. Similarly, the input vector is defined as

$$\vec{u}(k) = \begin{bmatrix} a(k) \end{bmatrix},$$

where $a(k)$ is the acceleration due to the driver's pressure on the accelerator pedal at time k. For simulator, the acceleration between time k and $k+1$ is controlled by the user as the simulation runs.

Now, $F(k)$, $G(k)$, and $H(k)$ must be defined. To determine $F(k)$, note that $d(k+1) = d(k)+v(k)$

and $v(k+1) = v(k) + a(k)$, so $F(k)$ is defined as

$$F(k) = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

and $G(k)$ is defined by

$$G(k) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The measurement vector for the car system $\vec{y}(k)$,

$$\vec{y}(k) = \begin{bmatrix} s(k) \end{bmatrix},$$

where $s(k)$ is the speedometer reading. Since it is a direct measurement of the second element in the system state, the matrix $H(k)$ is defined as

$$H(k) = \begin{bmatrix} 0 & 1 \end{bmatrix}.$$

Plugging these function definitions in to the state equation and the measurement equation, we get the following system model:

$$\begin{bmatrix} d(k+1) \\ v(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} d(k) \\ v(k) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} [a(k)]$$

$$[s(k)] = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} d(k) \\ v(k) \end{bmatrix}$$

These equations are used to model the system in the simulator.

7.1.2 Software Architecture

Overview First, the simulation engine is initialized with the system starting state and the description of the system model. Then the program runs in a loop where each iteration has two steps: simulation, and Kalman filtering. Both aspects of the software are discussed later in this section. The simulation output is shown in Figure 11. The actual position and the estimated position are shown in the far left graph over time. The middle graph shows the velocity estimate compared to the actual velocity over time. The far right graph is a plot of the position error over time. "Current Covariance Estimate" output shows the $P(k)$ for the current k, or the current error covariance estimate computed by the Kalman filter. The car's acceleration may be changed by entering a value in the "Current Acceleration" control. The program may be stopped our paused by pressing the two middle buttons. Error is displayed in the bottom right two outputs. "Current Error" shows $\vec{x}(k) - \hat{x}(k)$ and "Current position error relative to actual state" shows the weighted position error, $Er(k)$, computed by

$$Er(k) = \frac{\left|\left| d(k) - \hat{d}(k) \right|\right|}{||d(k)||}, \tag{1}$$

where $\hat{d}(k)$ is the current estimate of car's position, $d(k)$. In all of these computations, the true system state is read directly from the simulation engine.

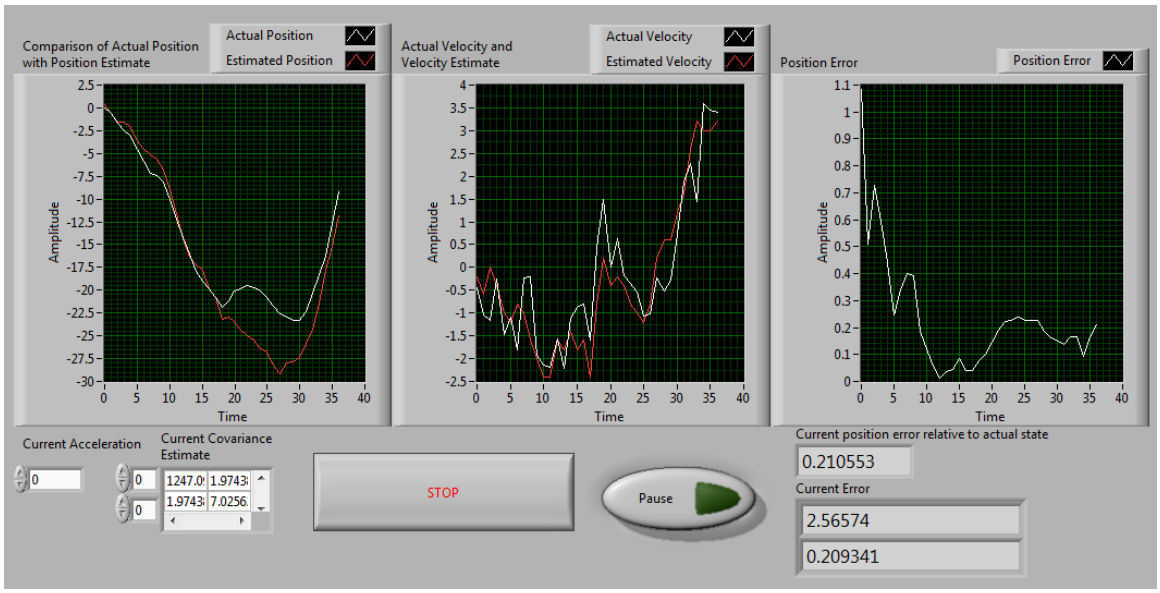

Figure 11: Simulation Program User Interface.

Simulation Engine A data type called `LinearDiscreteSystemSim` was written to drive this simulation engine. This is a data type that uses the state-space model state and measurement equations to advance the system state and generate measurements, respectively. This class stores $\vec{x}(k)$, $\vec{u}(k)$, $\vec{y}(k)$, $F(k)$, $G(k)$, and $H(k)$ for the current time k. In this way, a variable of type `LinearDiscreteSystemSim` contains all of the current information about the linear system being simulated.

To advance the state from time k to time $k+1$, there is a "State Transition" function. The inputs of this function are a variable of type `LinearDiscreteSystemSim`, the input vector at time k, $\vec{u}(k)$, and the process noise at time k, $\vec{v}(k)$. The output of this function is a variable of type `LinearDiscreteSystemSim`. The output contains the system model at time $k+1$. The next system state, $\vec{x}(k+1)$, which is written to the `LinearDiscreteSystemSim` output is computed using

$$\vec{x}(k+1) = F(k)\vec{x}(k) + G(k)\vec{u}(k) + \vec{v}(k)$$

where $F(k)$ and $G(k)$ are taken from the `LinearDiscreteSystemSim` input.

To make a measurement, there is a "Perform Measurement" function. The inputs are a variable of type LinearDiscreteSystemSim and a vector containing the measurement noise at time k, $\vec{w}(k)$. The outputs are a variable of type `LinearDiscreteSystemSim` and a vector containing $\vec{y}(k)$. The vector $\vec{y}(k)$ is computed by

$$\vec{y}(k) = H(k)\vec{x}(k) + \vec{w}(k),$$

where $H(k)$ is obtained from the `LinearDiscreteSystemSim` input.

So, the user can make measurements by using the "Perform Measurement" function, and can advance the system from time k to time $k+1$ using the "State Transition" function.

Uncertainty The Kalman filter assumes that the noise terms in the control system are Gaussian, so a Gaussian random number generator was developed. LabVIEW has a function for generating uniform random numbers between 0 and 1. An array of bins was constructed so that more bins contained values around the mean. The number of bins and the maximum and minimum value

associated with each bin is set using some parameters. The range of the random number generated is 0 to 1, and this number is then scaled as desired. Since there are more bins representing values closer to the mean, the uniform random number generator can be used to select one of these bins, thereby selecting a Gaussian random number.

Kalman Filter The Kalman filter code for this program went through three different stages, based on [5]. In this reference, three different sets of Kalman filter equations are provided. Each of these versions of the Kalman filter were coded in order to obtain proof of concept. All Kalman filter equations were taken from [5, p277-285].

Common to all three stages of the Kalman filter code was the `KalmanFilter` data type. This data type contains all of the matrices that describe the linear system model, the current system state estimate $\hat{x}(k)$, and the current error covariance $P(k)$.

The first set of equations assume that there is no process noise and no measurement noise. The simulator was configured to also produce no noise for this stage of the experiment. While this is not a very useful version of the Kalman filter, being able to use it and get good results was an important step before moving on to SLAM. The Kalman filter equations for this stage are

$$\hat{x}(k+1|k) = F(k)\hat{x}(k) + G(k)\vec{u}(k)$$

for prediction, and

$$\hat{x}(k+1) = \hat{x}(k+1|k) + H^T(HH^T)^{-1}(\vec{y}(k+1) - H\hat{x}(k+1|k))$$

for update.

Next was a model that only considered process noise. Again, the uncertainty portion of the code was modified so that process noise was generated, but measurement noise was not. The Kalman filter equations for this stage are

$$\begin{aligned}\hat{x}(k+1|k) &= F(k)\hat{x}(k) + G(k)\vec{u}(k)\\ P(k+1|k) &= F(k)P(k)F(k)^T + V(k)\end{aligned}$$

for prediction, and

$$\begin{aligned}\hat{x}(k+1) &= \hat{x}(k+1|k) + Rv\\ P(k+1) &= P(k+1|k) - RH(k)P(k+1|k)\end{aligned}$$

for update, where

$$\begin{aligned}v &= \vec{y}(k+1|k) - H(k)\hat{x}(k+1|k)\\ R &= P(k+1|k)H(k)^T\left(H(k)P(k+1|k)H(k)^T\right)^{-1}.\end{aligned}$$

The final stage was to use the full Kalman filter equations as presented in section 3.1.

7.1.3 Results

The simulator results can be seen in Figure 11. The leftmost graph shows the actual position of the robot and the robots position estimate. The middle graph shows the actual speed of the robot and the robot's speed estimate. The leftmost graph shows the error in the position estimate. When the simulation was first run, the error calculated was just the difference between the actual position

of the robot and the estimated position of the robot. Since the distance was always increasing, the error when calculated in this way also always increased. At first, it seemed like the simulator was working incorrectly. However, after extensive troubleshooting, it was found that there were no issues with the simulator. The issue was with the way the error was being calculated. A better way to calculate the error was developed, and is shown by Equation 1. With this change, the error did in fact tend to zero with time. This is the desired result of the SLAM algorithm, so the simulation can be considered a success. This demonstrated that an adequate understanding of the Kalman filter was gained, and experiments applying the SLAM algorithm could be developed.

7.2 Stationary Ship

7.2.1 Experiment Description

The first experiment was developed in order to test the Kalman filter in a system less complicated than SLAM. In this scenario, there is only one landmark, and both the robot and the landmark are stationary. The following describes the various elements of the system. The state vector $\vec{x}(k) = \begin{bmatrix} r(k) \end{bmatrix}$ where $r(k)$ is the Euclidean distance between the robot and the red landmark. Since the robot and the landmark remain stationary, $F(k)$ is the one-dimensional unit vector. The stationary system has no input vector and so $\vec{u}(k)$ and $G(k)$ are zero. Since the system is stationary, the process noise, $\vec{v}(k)$, is assumed to be zero. The state equation then boils down to:

$$r(k+1) = r(k).$$

The measurement vector $\vec{y}(k)$ is the distance from the robot at every iteration as measured by the laser rangefinder, so $\vec{y}(k) = \begin{bmatrix} r'(k) \end{bmatrix}$. The measurement vector is used to update the state vector. The matrix $H(k)$ in this case is the identity matrix of dimension one since the state vector and the measurement vector represent the same quantity. Hence, the measurement equation is:

$$r'(k) = r(k) + w(k),$$

where $w(k)$ is the measurement noise due to noise in the laser rangefinder and webcam measurements.

7.2.2 Software Architecture

Overview The software for this experiment has two parts: landmark extraction and the Kalman filter. The landmark extraction algorithm determines the distance of the landmark from the robot using the camera and laser rangefinder. The Kalman filter is used to combine all the measurements into one estimate of the landmark's distance from the robot.

Sensor Calibration The webcam and the laser rangefinder were tested and calibrated so that a general relationship between the horizontal location of the landmark in the captured image and its corresponding distance measurement from the laser rangefinder scan was found. The laser rangefinder data is received as an array, and each data point in the array corresponds to a specific angle of the scan. The reading from one specific angle will always be in the exact same location of the array. This information was used to calibrate the webcam and the laser rangefinder. To perform the calibration, objects were put at the very edge of the webcam image and the array location of the distance reading that corresponded to the object was recorded. To sum up, each

data point consisted of the x coordinate in the image (in pixels) and the location in the array of the distance information for that pixel.

(Pixel location, Index of distance data in array)

Originally, many data points at the edge of the image were taken, but it was decided that a good enough approximation could be calculated by simply taking two data points, one at the far left corner of the image and one at the far right corner. Figure 12 illustrates the set-up used to calibrate the webcam and the laser rangefinder. From these two data points a linear relationship between

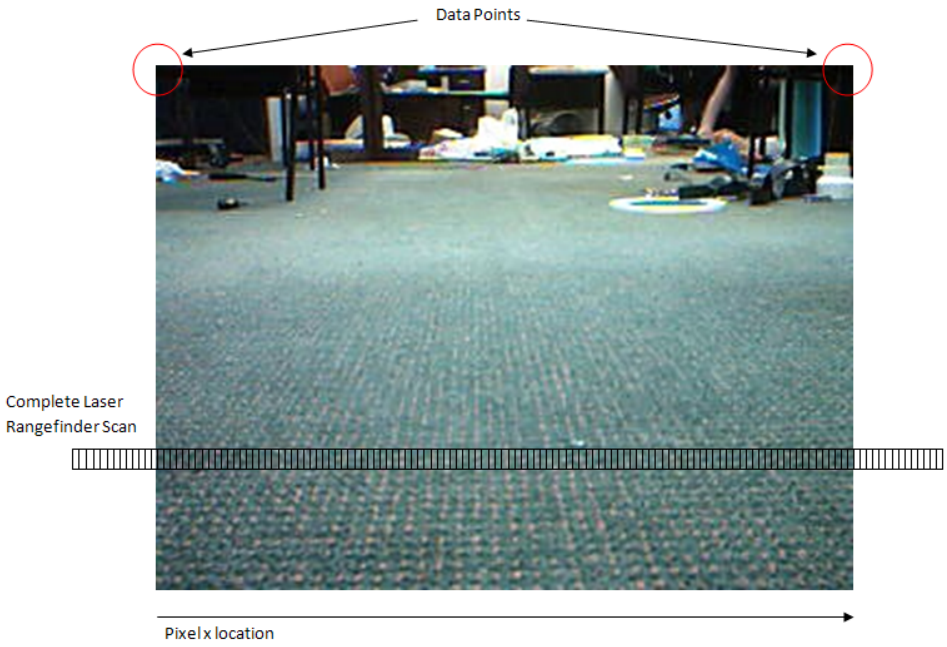

Figure 12: Diagram showing the calibration of the webcam and laser rangefinder

the horizontal pixel location in the image and the index of the corresponding data in the array was derived. The linear relationship is shown by the following equation:

$$i_{lrf} = 0.8x_{img} + 280, \tag{2}$$

where i_{lrf} is the index of the data in the laser rangefinder array and x_{img} is the horizontal pixel location of the landmark in the image.

Landmark Extraction As previously discussed, to properly perform SLAM a robot must be able find easily identifiable landmarks in its environment and gather information about their position relative to the robot. This is called landmark extraction. Since the focus of this project was to develop a working SLAM algorithm, it was decided that simple, easy to detect landmarks would be used. The webcam can produce an image that can be processed to detect colors fairly easily. Because of this, the landmark used in this experiment was a red cylinder, shown in Figure 13.

The algorithm for detecting the landmark is as follows. The webcam and the laser rangefinder point in the same direction. The webcam takes a picture and the laser rangefinder takes a scan at the same time. The image from the webcam is then sent through a series of image processing operations which detect the red from the landmark and determine its center of mass. The center

Figure 13: Red landmark used for stationary ship and straight line experiments.

of mass, x_{img} is then put into Equation 2. This results in an estimate of where in the array the corresponding distance information is located. When this method was tested, there were several problems with it. First of all, the equation did not always yield the exact location of the correct distance information in the scan. Secondly, it was not very robust in that when the robot turned, it would lose its spot in the laser rangefinder scan very quickly. Because of these issues, a final step was added to the process. A 40 element sub-array was taken from the complete laser rangefinder scan array. This sub-array was centered at the result of equation 2, i_{lrf}. This made it so that even when the equation did not give the exact location of the distance measurement, the distance measurement was always located in the sub-array. Figure 14 shows how the 40 element sub-array is taken from the complete laser rangefinder scan. Once the 40 element sub-array is extracted from

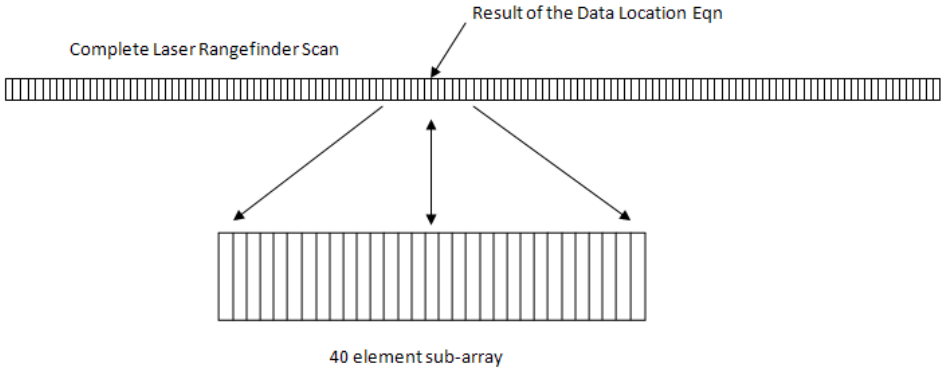

Figure 14: This diagram that shows how sub-array is taken from laser rangefinder scan.

the laser rangefinder scan, it is analyzed element by element looking for a change in distance of over 50 mm. If there is a change in distance of over 50 mm, then there is a landmark present and the distance reading is taken and saved. This distance reading along with the angle at which the measurement was taken are used to calculate the (x, y) position of the landmark relative to the position of the robot as shown in Figure 15. We will call this the $(\hat{x}, \hat{y})$ position for the rest of

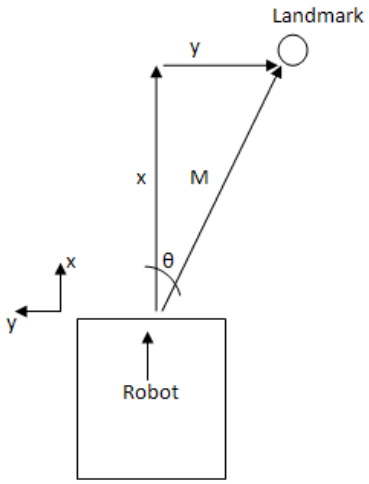

Figure 15: This diagram shows how the (x, y) position a landmark relative to the robot is calculated.

this discussion. Once the $(\hat{x}, \hat{y})$ position is determined, it must be translated so that it is an (x, y) position with regard to the original coordinate system. We will call this (x_i, y_i). In order to do this, the current (x_r, y_r) position of the robot will be taken from the robot state and added to the landmark $(\hat{x}, \hat{y})$. The result of this is the position of the landmark with respect to the original coordinate system, (x_i, y_i). The final result of the landmark extraction algorithm is this (x_i, y_i) for each visible landmark. It is input into the Kalman filter and used in the update section of the Kalman filter.

Kalman Filter The Kalman filer aspect of the program was exactly the same as in the final version of the simulator described in section 7.1.2.

7.2.3 Results

The state estimate $\hat{x}(k)$ converged to a real value close to the actual distance between the robot and the landmark while the error covariance converged to zero. These results were as predicted, so a more complex experiment, the straight line SLAM experiment, could be conducted.

7.3 Straight Line SLAM

7.3.1 Experiment Description

In the straight line experiment, the red landmark remains stationary while the robot moves towards it in a straight line. The robot moves using dead reckoning, so the Create's wheels are sent constant speed values. The state vector now has the robot's x, y coordinates and heading $(\theta(k))$ and the single landmark's x and y coordinates. The vector then takes the following form:

$$\vec{x}(k) = \begin{bmatrix} x_r(k) \\ y_r(k) \\ \theta_r(k) \\ x_1 \\ y_1 \end{bmatrix}$$

Since the system does not have any additional variables, $F(k)$ is the 5×5 identity matrix. The input vector, $\vec{u}(k)$, reflects the change that is caused by the forward motion of robot in the x and y directions and its heading. Hence $\vec{u}(k)$ is as follows:

$$\vec{u}(k) = \begin{bmatrix} \Delta x_r(k) \\ \Delta y_r(k) \\ \Delta \theta_r(k) \end{bmatrix}$$

$G(k)$, which defines the effect of the input vector on the control vector is as follows:

$$G(k) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

This form ensures that the robot's x, y and θ values are updated by the input vector without changing the one landmark's position.

The measurement vector $\vec{y}(k)$ corresponds to a displacement vector between the robot and the landmark, so it has x and y components. The matrix H is defined for the case where $n = 1$ as in section 3.2. While the noise terms cannot be modeled directly, the covariance matrices for each one must be provided. The process noise covariance, or the covariance of $\vec{v}(k)$ is

$$V(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

This says that there is uncertainty in the robot's change in position, but not in the actual position of the one landmark.The measurement noise covariance, or the covariance of $\vec{w}(k)$ is

$$W(k) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

This covariance matrix encodes the uncertainty involved in measuring the location of the landmark.

7.3.2 Input Vector

Since the input vector is the change in x, y, and θ with respect to the reference coordinate frame, some computation must be done on the data received from the Create to determine $\vec{u}(k)$. The Create's shaft encoders measure two quantities: average distance traveled by both wheels since the distance register was last polled, and the change in angle in the robot's heading since the heading register was last polled.

Figure 16 shows the quantities involved in this computation. The blue circles with arrows show the robot before (lower-right) and after (upper-left) making a turn. The arrows inscribed in the blue circles indicate the robot's heading. The Create returns $S(k)$, the arc length of this turn, and the change in heading. Using some simple geometry, it is clear that the change in heading is equal to $\theta(k)$. Before computing $\Delta x_r(k)$, $\Delta y_r(k)$, and $\Delta \theta_r(k)$, the change in position with respect to

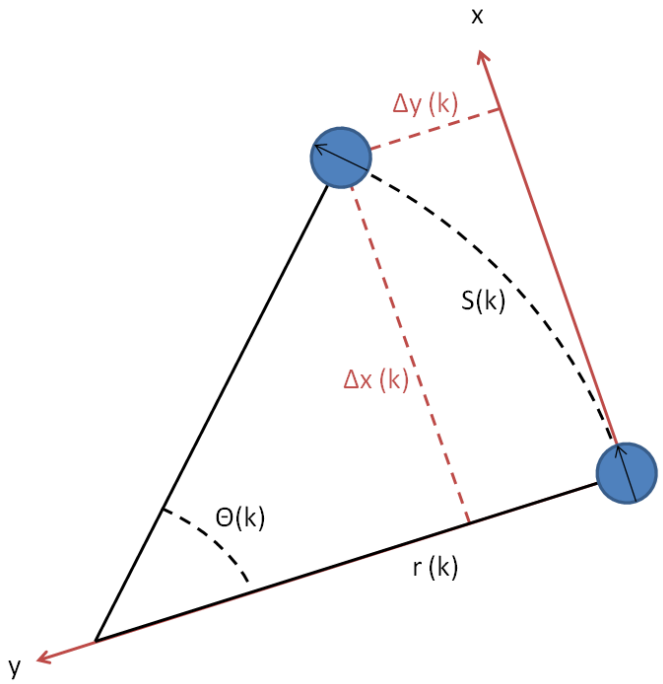

Figure 16: Diagram showing the quantities involved in computing the input vector. $\theta(k)$ and $S(k)$ are measured directly by the Create's shaft encoders.

the (x, y) coordinate frame in Figure 16, denoted $(\Delta x, \Delta y)$, must be computed. First, the turning radius is

$$r(k) = \frac{S(k)}{\theta(k)}.$$

Using this radius and simple trigonometry,

$$\Delta x(k) = r(k)\sin(\theta(k)).$$

Then, using similar logic,

$$\begin{aligned} r(k) - \Delta y(k) &= r(k)\cos(\theta(k) \\ r(k) - r(k)\cos(\theta(k)) &= \Delta y(k) \\ r(k)(1 - \cos(\theta(k)) &= \Delta y(k). \end{aligned}$$

So, to summarize,

$$\begin{bmatrix} \Delta x(k) \\ \Delta y(k) \end{bmatrix} = \begin{bmatrix} \Delta r(k)\sin(\theta(k)) \\ \Delta r(k)(1 - \cos(\theta(k)) \end{bmatrix},$$

where $r(k) = S(k)/\theta(k)$.

To compute $(\Delta x_r(k), \Delta y_r(k))$, $(\Delta x, \Delta y)$ must be converted to the reference coordinate frame. Since this vector is a displacement vector, it suffices to rotate $(\Delta x, \Delta y)$ by $\theta_r(k)$. This is accomplished using a 2×2 rotation matrix of angle $\theta_r(k)$, or

$$\begin{bmatrix} \cos(\theta_r(k)) & -\sin(\theta_r(k)) \\ \sin(\theta_r(k)) & \cos(\theta_r(k)) \end{bmatrix}.$$

So the final control vector is computed by

$$\vec{u}(k) = \begin{bmatrix} \Delta x_r(k) \\ \Delta y_r(k) \\ \Delta \theta_r(k) \end{bmatrix} = \begin{bmatrix} \cos(\theta_r(k)) & -\sin(\theta_r(k)) & 0 \\ \sin(\theta_r(k)) & \cos(\theta_r(k)) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \Delta x(k) \\ \Delta y(k) \\ \theta(k) \end{bmatrix}.$$

7.3.3 Software Architecture

Overview This experiment is the first one in which all the components of the robot were used. Figure 17 shows the flowchart for the program. The first step in the program is to construct the measurement and update vectors, $\vec{y}(k)$ and $\vec{u}(k)$. $\vec{y}(k)$ is built directly from the landmark extraction code. The vector $\vec{u}(k)$ is read from the iRobot Create as described in section 7.3.2 above. At this point, $\vec{u}(k)$ can be plugged into the Kalman predict step. The predicted state and $\vec{y}(k)$ can then be used in the update step to compute the final prediction for this iteration, $\hat{x}(k)$.

Landmark extraction is performed as in the stationary ship experiment, the Create is driven at a constant speed and the shaft encoders are read as described in section 6.1. The Kalman filter used in the stationary ship experiment was used directly in this experiment as well.

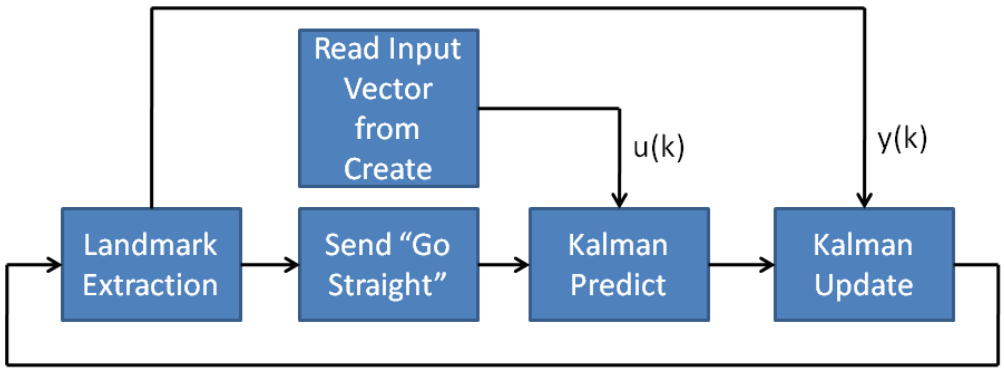

Figure 17: Flowchart showing straight line SLAM software architecture.

Landmark Extraction The landmark extraction algorithm for the straight line experiment is the same algorithm used in the stationary ship experiment.

Kalman Filter The Kalman filter code used in the straight line experiment was the same as for the stationary ship experiment.

Map Generation The map was generated from the Kalman estimate, $\hat{x}(k)$. At the end of each iteration of the main loop, the robot position, $(x_r(k), y_r(k))$, is saved to an array. This array of (x, y) points is plotted as a line using the LabVIEW waveform graph control. Another array containing the location of the landmark was also passed to this graph control and plotted as solid white dots. A sample map is shown below in figure 18.

7.3.4 Results

Initially, the results of the experiment were not as expected. The covariance matrix did not tend to 0 and the estimate was not reasonable at all. Because of this, the update and predict equations

as well as the covariance matrices were examined in greater detail. It was found that the initial covariance matrix was causing the experiment to fail. For the initial trials, the covariance matrix was a 5×5 identity matrix. After some research, it was found that covariance matrices that resulted in successful SLAM were not the identity matrix, rather they had zeros instead of ones in some places. The process noise covariance matrix used for the successful experiment is as follows:

$$V(k) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Using this covariance matrix, the experiment yielded results as expected with two constraints. The landmark always had to be in the field of view of the webcam and the laser rangefinder, and the measurement associated with the landmark had to be valid. The second constraint arose due to the fact that the Hokuyo laser rangefinder being used did not always give good measurement readings. When looking at a complete scan, there would be measurements of 0 throughout the scan when there should not have been any. This occurred most often when there were no objects within its range. This would sometimes cause the measurement of the landmark to be 0, which was incorrect. This problem was corrected in later experiments.

The robot was able to detect the landmark and get its (x, y) position, as in the stationary ship experiment. When the robot moved, it was able to keep track of the landmark through the iterations of the program. Also, the predict and update steps of the Kalman filter worked properly, resulting in the robot updating its estimate of its position accurately. All together, this resulted in the robot being able to move towards the landmark while updating its estimate of its position and the position of the landmark relative to a set coordinate system. Figure 18 shows the map generated from the straight line experiment. The red dot is the estimated position of the landmark. The landmark was not directly in front of the robot, as shown by its position. The white line shows the estimate of the path of the robot. The line is not straight because the noise in the landmark measurements changed the estimate of the path slightly. Despite this, the estimate is still very good.

7.4 Multi Landmark SLAM

7.4.1 Experiment Description

The multiple landmark experiment incorporates all three landmarks into the system. Because of this, the various components of the Kalman filter system needed to be updated to include all of the landmarks. The robot itself still used dead reckoning to move in a straight line the same way it did in the straight line experiment.

7.4.2 Software Architecture

Overview This experiment will be extremely similar to straight-line SLAM. Landmark extraction was modified to include blue and green in addition to red. A "Landmark Management" component was added to the software to handle cases in which landmarks were not visible. The first version of this code relied on a separate landmark database to store landmarks not observed in the current measurement instead of just using the system state, $\vec{x}(k)$. This method required a lot of complex matrix manipulation, and thus was fraught with bugs. Later on it was discovered that by simply manipulating the H matrix the same affect could be achieved with little computational overhead.

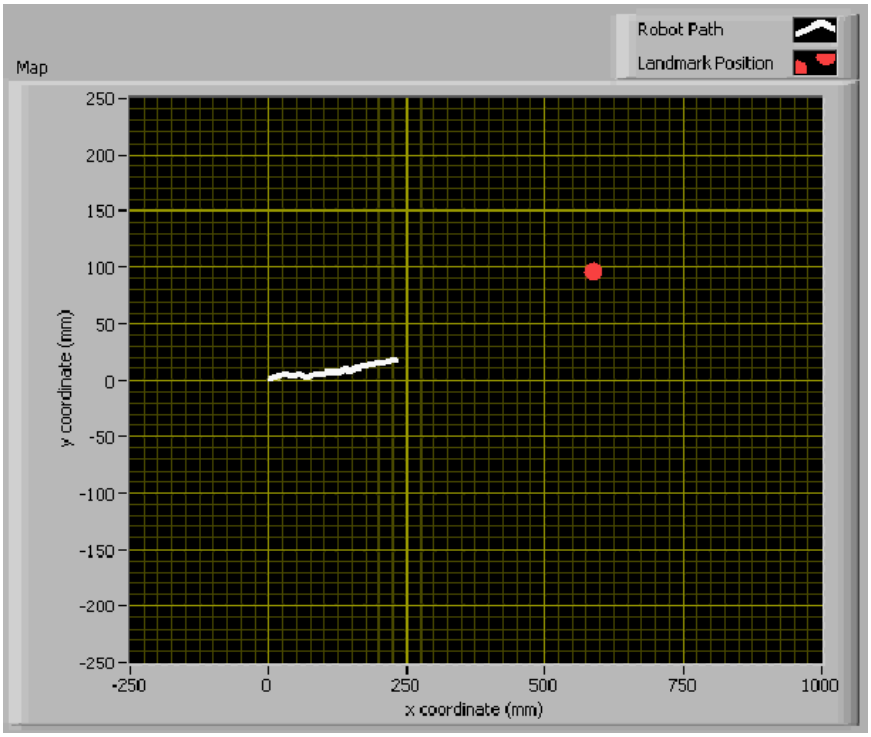

Figure 18: Map output of straight line experiment

Landmark Extraction For multiple landmark SLAM, the landmark extraction algorithm had to be updated to detect multiple landmarks. The basic algorithm, however, remained the same. Instead of having one red landmark, there were three landmarks of different colors, one red, one green, and one blue, shown in Figure 19.

Separate sub-vi's were written to detect the green landmark and then the blue landmark in the image from the webcam using the same process used to detect the red landmark. Then, when the image is taken and the laser rangefinder makes its scan, the image is processed three times, once looking for the red landmark, once looking for the blue landmark, and once looking for the green landmark. Each landmark detected is assigned an ID according to its color, and its center of mass is used to find the distance measurement from the laser rangefinder scan using the same process previously described. The assigning of the ID is very important because it is used to identify the same landmark for each iteration of the code. It is important that the red landmark is always assigned the same ID so that the program will not associate the distance measurement from the red landmark with the ID of the blue landmark. This is called data association. If it is not done properly, it will cause the program to think that the location of a landmark suddenly jumped to a different place, which can cause issues in the update step of the Kalman filter and result in errors in the state estimate.

Landmark Management After the landmark extraction step, a list of N observed landmarks and their ID's are passed along to the Kalman filter step. Before performing the update, the landmarks must be compiled into the measurement vector. For this scheme to work, there must be a correspondence between the indexes in the state vector with the current landmark position and the ID of the landmark as seen by the landmark extraction code. This correspondence is built up dynamically depending on the order in which the landmarks are first detected. The state estimate

Figure 19: The 3 different landmarks with different colors

will be just as explained in section 3.2:

$$\hat{x}(k) = \begin{bmatrix} x_r(k) \\ y_r(k) \\ \theta_r(k) \\ x_1(k) \\ y_1(k) \\ \vdots \\ x_n(k) \\ y_n(k) \end{bmatrix}$$

In addition to the state vector, there is an array that contains the correspondences between the landmark indexes shown in the state vector above and the landmark ID's. The array is defined by

$$\texttt{LM\_IDs} = \begin{bmatrix} \texttt{ID1} & \texttt{ID2} & \ldots & \texttt{IDn} \end{bmatrix},$$

where `ID1` corresponds to the 1st landmark in the system state, `ID2` corresponds to the 2nd landmark in the system state, etc..

Landmark extraction has completed and output its data to the Kalman update step. First, the H matrix must be built for each measurement. As discussed in section 3.2, the H matrix is composed of many different H_i matrices, one for each landmark. The landmark extraction code outputs an array of landmark data in which ea ch element is of the form

$$\texttt{LM\_Meas} = \left(\texttt{LM\_ID}, \begin{bmatrix} x \\ y \end{bmatrix}, \begin{bmatrix} Var(x) & Cov(x,y) \\ Cov(x,y) & Var(y) \end{bmatrix} \right),$$

where `LM_ID` is the ID of the landmark as determined by the landmark extraction code, $\begin{bmatrix} x \\ y \end{bmatrix}$ is the measured (x, y) coordinate of the landmark in the reference frame, and the matrix

$$\begin{bmatrix} Var(x) & Cov(x,y) \\ Cov(x,y) & Var(y) \end{bmatrix} \tag{3}$$

is the covariance matrix for $\begin{bmatrix} x \\ y \end{bmatrix}$. If the landmark is already in the state vector, the proper index is found by searching for `LM_ID` in the `LM_IDs` array. This index is the i index in H_i. So, the H_i matrix is built as discussed in section 3.2. The H_i matrix is re-stated here to refresh the reader's memory

$$H_i = \begin{bmatrix} -1 & 0 & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 & 1 & \cdots & 0 \end{bmatrix}.$$

The next index must also be looked up in `LM_IDs`. It may not be $i+1$, since it is not known which landmarks will be observed at each step. However, there does not need to be an H_i for all landmarks in the state vector. As long as each H_i is defined properly, the measurement equation will be satisfied. All of these H_i's are combined to make up a $2N \times (3n+2)$ H matrix.

The W matrix must also be built for this iteration. This matrix is built from the covariance matrix mentioned in equation (3). If $W_1, \ldots, W_N$ is are the covariance matrices, the complete $W(k)$ matrix is a block matrix defined by

$$W(k) = \begin{bmatrix} W_1 & 0 & \cdots & 0 \\ 0 & W_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & W_N \end{bmatrix}.$$

If this is the first time the landmark has been observed, it cannot be looked up in the state vector; it must be added. Landmarks that have never been observed before are added to the state estimate and do not take part in the Kalman update step. To add a landmark to the system, the state vector and error covariance must be expanded, `LM_IDs` must be updated, and the control system matrices must also be updated. To add the landmark to the state vector, the measured landmark position $\begin{bmatrix} x \\ y \end{bmatrix}$ is inserted just below the nth landmark position estimate in $\hat{x}(k)$ as $\begin{bmatrix} x_{n+1}(k) \\ y_{n+1}(k) \end{bmatrix}$:

$$\hat{x}(k) = \begin{bmatrix} x_r(k) \\ y_r(k) \\ \theta_r(k) \\ x_1(k) \\ y_1(k) \\ \vdots \\ x_{n+1}(k) \\ y_{n+1}(k) \end{bmatrix}.$$

The augmented $P(k)$, denoted $P'(k)$, is computed by inserting the covariance matrix from `LM_Meas` at the lower right diagonal of $P(k)$:

$$P'(k) = \begin{bmatrix} P(k) & 0 & 0 \\ 0 & Var(x) & Cov(x,y) \\ 0 & Cov(x,y) & Var(y) \end{bmatrix}.$$

Next the control system model must be modified. First, $F(k)$ is still the identity matrix, except now its size is increased to $(3+2(n+1)) \times (3+2(n+1))$ to match the state vector. Two rows of zeros are added to $G(k)$ to make it the correct size of $(3+2n) \times 3$. Two rows of zeros and two

columns of zeros are also added to $V(k)$ to make it the correct size. Unless a new landmark is added, these matrices all stay the same.

Figure 20 shows a flowchart of the landmark management portion of the code. The "Add LM to $\vec{y}(k)$" block also changes $H(k)$ and $W(k)$ as discussed above. Similarly, the "Add LM to state" block also augments $F(k)$, $P(k)$, $G(k)$, and $V(k)$ as discussed above.

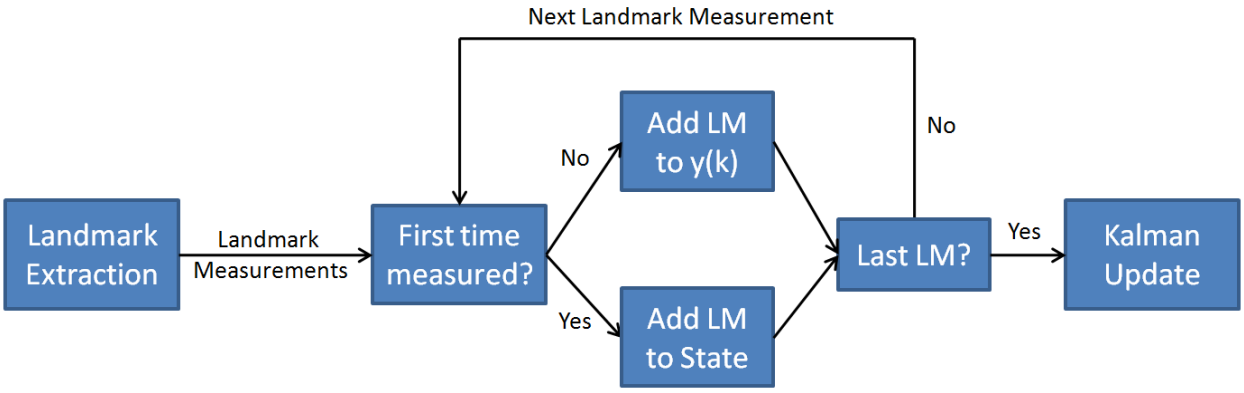

Figure 20: Flowchart for the landmark management code.

Kalman Filter For this experiment, the Kalman filter was streamlined for easier usability. Two new data types were created, called `KalmanEstimate` and `StochasticLinearSystem`. The `KalmanEstimate` type contains the current state estimate, $\hat{x}(k)$, and the current error covariance estimate $P(k)$. The `StochasticLinearSystem` type contains $F(k)$, $G(k)$, $H(k)$, $V(k)$, and $W(k)$.

The Kalman Predict and Update function use the same Kalman predict and update equations as stated in section 3.1, but the input now is both a variable of type `KalmanEstimate` and one of type `StochasticLinearSystem`. This change in convention has improved the organization of the code.

Map Generation Map generation is done exactly the same as for the straight line experiment, except the second array of landmark positions contains the position of the green and blue landmarks, in addition to the red one. Figure 21 shows a map generated after one run of the Multi-Landmark SLAM algorithm.

7.4.3 Results

Initially, the experiment yielded mixed results when the robot was stationary and three landmarks were in view. The location of one of the landmarks was always correct, but the location of the other two landmarks was correct at times and incorrect at times. The cause of this was found to be the incorrect measurements of 0 present in the laser rangefinder scan previously discussed. In the sub-array of 40 elements, there were objects measured in the scan that were not landmarks, such as the wall. When there were several measurement readings of 0 it would register as a transition of greater than 50 mm and would then be used incorrectly as the landmark measurement. To account for this, a cut-off distance was implemented. If a measurement was greater than this distance, it would be set to 0 in the sub-array. This would then not allow it to register as a transition, and the correct landmark measurement could be found from the array. Once this correction was made, the measurements of all three of the landmarks were correct. From that point, a test was done

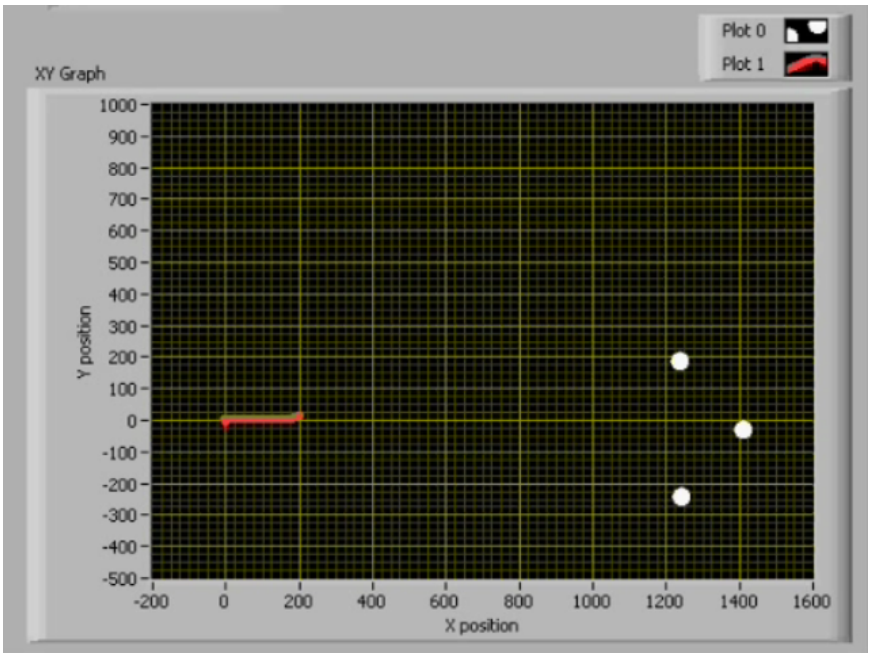

Figure 21: Map generated by the multi-landmark version of the SLAM algorithm

where one landmark was in the field of view of the robot, and the second was added after a period of time. This caused the program to crash due to non-conformability of Kalman filter matrices. Specifically, the dimensions of the error covariance P matrix did not change as they were supposed to with the addition and or removal of landmarks. Instead of re-building the entire system and measurement model every step, the system model, $F(k)$ and $G(k)$, was changed only when there is a new landmark. The list of ID's and landmark measurements are used to build the measurement model, the $H(k)$ matrix. Figure 21 shows the results of a run of the experiment. The landmarks are shown by the white dots, and the path of the robot is shown by the red line. Landmarks were able to be added and subtracted as they were seen, and the system state was correctly estimated.

7.5 SLAM-Driven Navigation

7.5.1 Experiment Description

As a demonstration of how the SLAM algorithm can be applied, a SLAM-based navigation process was developed that works based on the predicted coordinates of the robot and the landmarks from the state estimate. The red landmark is assigned to be the goal and the aim of the process is to reach the goal without hitting any object on the way. This algorithm has been developed in several steps.

7.5.2 Simple Navigation

The navigation algorithm was first developed for the simplest scenario where the robot only has the red landmark in view. The landmark was modeled as a point. When the robot is started up it forms a Cartesian coordinate system for it where the positive x axis corresponds to the direction that the robot is pointed in. Hence the robot starts off at the origin. The red landmark is immediately visible to the robot, so its coordinates are obtained from state estimate of the SLAM process. Based

on the coordinate information the robot calculates the angular displacement (heading) in degrees of both the robot and the red landmark relative to the origin. The robot then attempts to correct its heading so that it points toward the landmark. First, a displacement vector between the red landmark and the robot is computed, and then by the basic convention of counter-clockwise being the positive direction, the angle of that displacement vector is compared with the robot's heading. If the robot's heading is smaller than the displacement vector's angle, the landmark is to the left of the robot and conversely the landmark is to the right if the robot's heading is larger than the vector's angle. The robot adjusts itself accordingly to the left or right using the differential drive technique. To achieve differential drive, the left wheel driven slower than the right wheel if the robot needs to turn right and the reverse is done if the robot needs to turn left. At every iteration, the navigation algorithm receives coordinate information directly from the SLAM algorithm and repeats the process as explained. The robot stops once it is sufficiently close to the landmark. This is determined by calculating the Euclidean distance between the robot and the landmark and comparing it to a threshold value.

From repeated tests the algorithm was found to work consistently for various starting orientations of the robot as long as the landmark was in view and recognizable before the robot began moving. The next step in development was to incorporate multiple landmarks into the scenario.

7.5.3 Multiple Landmark Navigation

For the multiple landmark setting, the blue and green landmarks were initially placed in setup shown in Figure 22. In this setup all the landmarks are visible to the robot when it starts up. The

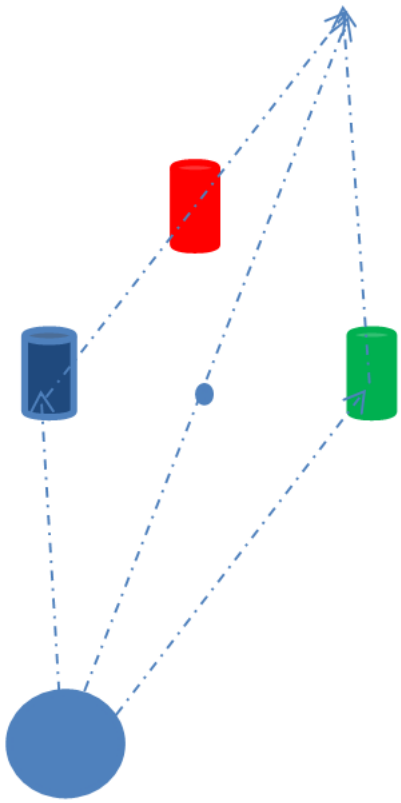

Figure 22: The 3 different landmarks with different colors

robot's goal is still to end up at the red landmark and must travel to it while avoiding the other two landmarks. A modified algorithm was developed to approach this situation. Based on the coordinate system initially developed, the robot obtains the coordinates of the landmarks, which are again modeled as single points in the setup, from the state estimate of the SLAM algorithm. Through simple geometry, the coordinates of the midpoint between the blue and the green landmark is calculated. This is illustrated in the diagram. For any arbitrary coordinate system, the vectors connecting the robot to the landmarks can be added up and the midpoint of the resulting vector will always be at the point in concern (denoted here by the blue dot). The robot then uses the simple navigation algorithm to first reach this point. The robot's own coordinates are updated from the SLAM algorithm continually and used to steer it towards the immediate goal. Additionally the distance between the robot's coordinates and the goal coordinates is calculated and when it is sufficiently small the algorithm moves to the next step. At the next step, the robot's goal coordinates are changed to the coordinates of the red landmark which is now in robot's view. Again, the simple navigation is followed to get the robot sufficiently close to the red landmark.

7.5.4 Further Development

The successful completion of these experiments demonstrates the robot's ability to perform navigation based solely on the estimated position of the robot itself and the different landmarks. However, the two experiments performed here are specific in nature to the placement of the landmarks. Based on the position data received from the SLAM algorithm, the experiment can be modified to make it generally applicable to other landmark placements. For instance the implementation of a VFH (Vector Field Histogram) based obstacle avoidance algorithm would be a plausible next step. In order to do so the system would have to simulate an array of LIDAR distance information based on the estimated coordinates of the different landmarks and modeling their shape and size. This array can then be used as an input to a VFH algorithm that would take into account the target heading (the red landmark) and the robot's current heading and determine the best immediate heading for the robot.

8 Discussion

The results of these experiments demonstrate that our robot was able to perform SLAM with simple, easy to detect landmarks. Furthermore, it was able to use the state vector generated while performing SLAM to do some simple navigation. In order to make this algorithm usable in real-world environments, changes would have to be made. The most significant change that would need to be made is to significantly upgrade the landmark extraction algorithm. The current landmark extraction algorithm was developed with the sole purpose of making it easy to develop the SLAM algorithm. If a robot using this algorithm were put into a room, it would most likely have great difficulty detecting and using landmarks. Since this is so important to any SLAM algorithm, the robot would then not be able to perform SLAM. Because of this, the next step would be to develop a better landmark extraction algorithm similar to an algorithm discussed earlier in Section 6. This may require an upgrade of the sensors used, since currently a cheap webcam and a slow laser rangefinder that has noise issues are being used. With a more advanced landmark extraction algorithm sending the same landmark information to the Kalman filter, the robot would then be able to function in a real-world environment. Then, it could be used on Q, the Robot Study Team's autonomous land vehicle. Q competes in the Intelligent Ground Vehicle Competition (IGVC). There are two challenges, the autonomous challenge and the navigation challenge. Q has issues with the autonomous challenge that SLAM would help solve. For the autonomous challenge, the robot has

to stay inside two white or yellow lines and go from the start point to the finish point. Within the lines are obstacles that the robot has to avoid such as barrels and mesh fences. In certain situations, Q does not navigate the obstacles properly. When faced with a decision where there the robot cannot continue straight and needs to make a decision to go left or right, sometimes the wrong decision is made resulting in Q going back toward the start point instead of toward the finish. If Q had a SLAM algorithm, it would be able to make a better decision because it would have a history of where it has been on the course and would then go away from where it has already been. This would eliminate the issue of Q going back towards the start point, and would most likely result in more success at the competition.

References

[1] "Create Manuals." IRobot Corporation: Home Page. Web. 03 Dec. 2009. <http://www.irobot.com/hrd_right_rail/create_rr/create_fam/createFam_rr_manuals.html>.

[2] Ranganathan, Ananth and Frank Dellaert. *Automatic Landmark Detection for Topological Mapping Using Bayesian Surprise (Technical Report No. GT-IC-08-04)*, College of Computing, Georgia Institute of Technology, 2008.

[3] Arana Arejolaleiba, N., F. Lerasle, M. Briot, C. Lemaire, and J. B. Hayet. "A Smart Sensor Based Visual Landmarks Detection for Indoor Robot". *ICPR02.* 848-51. Print.

[4] Hayet, J.B., F. Lerasle, M Devy, "Visual landmarks detection and recognition for mobile robot navigation," *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on* , vol.2, no., pp. II-313- II-318 vol.2, 18-20 June 2003 <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1211485&isnumber=27266>

[5] Choset, Howie, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents).* New York: The MIT, 2005. Print.

[6] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics.* Cambridge, Mass.: MIT, 2006. Print.

[7] Durrant-Whyte, Hugh, and Tim Bailey. "Simultaneous Localization and Mapping: Part 1." IEEE Robotics and Automation Magazine (June 2006): 99-108. Print.

[8] Madhavan, Raj, Hugh Durrant-Whyte, and Gamini Dissanayake. *Natural Landmark-based Autonomous Navigation using Curvature Scale Space.* Proc. of 2002 IEEE International Conference on Robotics & Automation, Washington, D.C. Print.

[9] Dailey, Matthew N., and Manukid Parnichkun. *Landmark-based Simultaneous Localization and Mapping with Stereovision. Tech.*

[10] "Simple Map Utilities: Simple Mapping Utilities (pmap)." USC Robotics Research Lab. Web. 31 Mar. 2010. <http://www-robotics.usc.edu/ ahoward/pmap/index.html>.

[11] Pitman, Jim. *Probability.* New York: Springer-Verlag, 1993. Print.