Trinity College
Trinity College Digital Repository

Senior Theses and Projects | Student Works

4-1-2012

# Implementation of a Simultaneous Localization and Mapping Algorithm in an Autonomous Robot

Adam T. Norton
*Trinity College*, adam.norton@trincoll.edu

Anson R. McCook
*Trinity College*, anson.mccook@trincoll.edu

Follow this and additional works at: http://digitalrepository.trincoll.edu/theses

Recommended Citation
Norton, Adam T. and McCook, Anson R., "Implementation of a Simultaneous Localization and Mapping Algorithm in an Autonomous Robot". Senior Theses, Trinity College, Hartford, CT 2012.
Trinity College Digital Repository, http://digitalrepository.trincoll.edu/theses/212

# Implementation of a Simultaneous Localization and Mapping Algorithm in an Autonomous Robot

Adam Norton '12
Anson McCook '12

Faculty Advisor: Dr. David Ahlgren

May 7, 2012

**Abstract**

A robot was built and programmed to implement a Simultaneous Localization and Mapping (SLAM) Algorithm. Traditional robotic mapping suffers from compounding sensor error, thus resulting in maps that become highly erroneous over time. SLAM combats this problem by taking a probabilistic approach to mapping. By combining odometry data with sensor measurements of surrounding landmarks through a Kalman Filter, the robot was able to accurately map its surrounding environment, and localize itself within that environment.

# Contents

**1 Introduction** **1**
- 1.1 Motivation . . . . . 1
- 1.2 Project Goals . . . . . 1

**2 Theory** **2**
- 2.1 VFH . . . . . 2
- 2.2 Kinect . . . . . 2
- 2.3 SLAM . . . . . 3
  - 2.3.1 General Principles . . . . . 3
  - 2.3.2 Kalman Filter Derivation [2] . . . . . 6
  - 2.3.3 Kalman Conclusion . . . . . 13

**3 Implementation** **14**
- 3.1 iRobot Create . . . . . 14
- 3.2 Kinect . . . . . 15
  - 3.2.1 OpenNI Interface . . . . . 16
  - 3.2.2 Landmark Extraction . . . . . 16
- 3.3 SLAM . . . . . 17
  - 3.3.1 Initialization . . . . . 18
  - 3.3.2 Input Vector . . . . . 18
  - 3.3.3 State Prediction . . . . . 20
  - 3.3.4 Measurement Vector . . . . . 20
  - 3.3.5 Calculate Kalman Gain . . . . . 20
  - 3.3.6 State Update . . . . . 21
  - 3.3.7 Landmark/System Update . . . . . 21

**4 Results** **22**
- 4.1 VFH . . . . . 22
- 4.2 Filter Simulation . . . . . 23
- 4.3 SLAM Simulation . . . . . 23
- 4.4 SLAM Experiments . . . . . 24
  - 4.4.1 Single Landmark . . . . . 24
  - 4.4.2 Three Landmarks . . . . . 25

**5 Conclusion** **25**

# 1 Introduction

## 1.1 Motivation

Situations arise where it is desirable to have exploratory robots navigate environments that may be previously uncharted, too dangerous for living beings, or for which GPS data is unavailable. There are many ways to program robotic navigation, some better than others. Common obstacle avoidance algorithms, such as wall-following or Vector Field Histogram (VFH) algorithms, concentrate on immediate navigation but do not attempt to learn more about the environment. One algorithm that looks to tackle this problem is the Simultaneous Localization and Mapping (SLAM) algorithm. SLAM is a probabilistic method in which a robot maps an environment while simultaneously localizing itself within the map. Once a map has been generated, navigation becomes much easier.

One of the biggest hurdles in robotics is being able to accurately analyze sensor data. Some mapping robots look to track odometry data as a means to determine location, and to create a map. However odometry data is imperfect. The fact that there is always some degree of sensor error needs to be considered. As time elapses, sensor error compounds, drastically distorting any map that has been created. Figure 1 shows an example of odometry based mapping. In this case, error from the odometery quickly accumulates and while any individual error may be small, the sum quickly becomes significant.

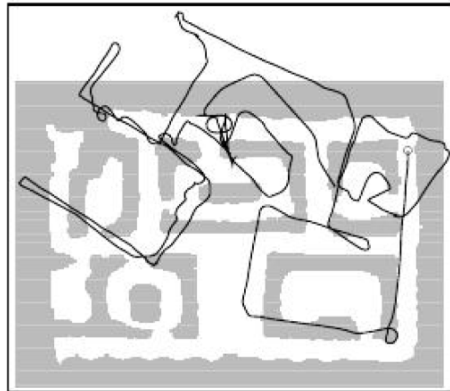

**Figure 1:** Odometry Error

Alternatively, some robots have access to GPS. While GPS is convenient, it is oftentimes not available, or not accurate enough for use indoors or in confined environments.

The solution to this problem is Simultanoues Localiztion and Mapping. The SLAM algorithm creates a map of the environment by statistically merging odometry data with sensor measurements of the environment. This forms a feedback loop by which a robot can make a much more accurate estimate of the state of its environment as well as the robot's location within that environment.

## 1.2 Project Goals

1) The main goal of this project is to study and understand the complex SLAM algorithm
2) To create a robot that is capable of implementing a SLAM algorithm in order to demonstrate understanding of the algorithm
3) To document the SLAM process in such a way as to provide future students with an easy to understand manual for implementing their own SLAM algorithms

## 2 Theory

### 2.1 VFH

Vector Field Histogram is an algorithm that uses a sensor scan of the environment in front of the robot and determines passable regions. It then picks the most suitable region and drives through it. An example is shown in the following figure. The first image shows the robot, the obstacles in its environment, and the robots sensor scan. The second image shows the data returned by the senor scan, as well as a short range passable region. The third image shows long range passable regions.

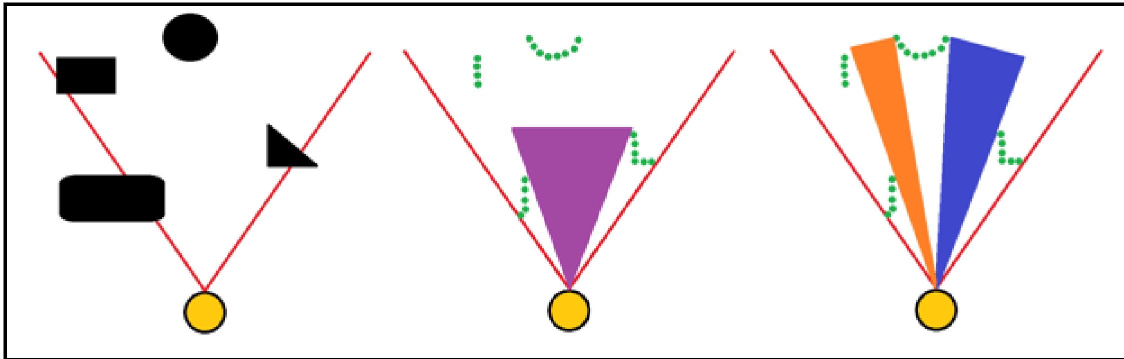

**Figure 2:** VFH

### 2.2 Kinect

The Xbox Kinect is a vision sensor made by Microsoft for use with the Xbox 360 gaming console. The Kinect is revolutionary in that it is capable of 3D image sensing. From Figure 3 below, it can be seen that the middle sensor is an RGB camera, and the other two are used in 3D depth sensing. The Kinect works by projecting a large number of IR dots around the area in front of the device. An example of this can be seen below in Figure 4. The depth device to the left in Figure 3 projects the IR dots around the room. The sensor to the right then observes the dot pattern. The Kinect is able to calculate the depth to every point in sight based on the dispersion pattern. For example, dots that are close together indicate objects that are closer, while dots that are more spread out indicate objects that are farther away. [13]

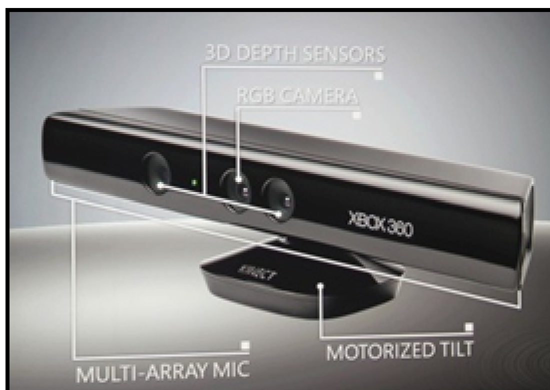


**Figure 3:** Kinect [11]

**Figure 4:** IR Dots [12]

## 2.3 SLAM

### 2.3.1 General Principles

As described in Section 1, robotic mapping suffers from compounding error in both sensor measurments (measurment error) and odometry (process error). The solution is a probabalistic approach called Simultaneous Localization and Mapping that makes use of a Kalman Filter. The Kalman filter is a probabalistic method for filtering out error, and is particularly applicable to mapping. The Kalman Filter works according to a Predict-Update scheme as shown in Figure 5.

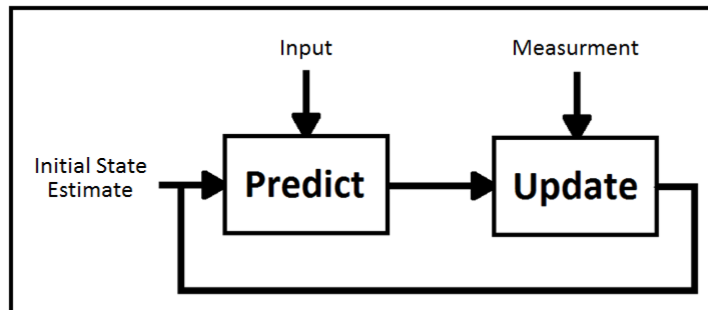


**Figure 5:** Kalman Filter Process

There are five main steps to the SLAM process. These are: State Estimation, Landmark Extraction, Data Association, State Update, and Landmark Update.

**State Estimation:** At any given time step, the robot knows the actions that it will apply to its motors in the next time step. The robot can also be programmed to know how those actions will change the robot position in its environment. Therefore, in the state estimation step, the robot uses this information to make a prediction of where it will be located in the next time step.

**Landmark Extraction:** After making a prediction for the next time step, the robot carries out some action on its motors and moves to the time step for which it has made a prediction. The robot then takes a sensor measurement of the environment and extracts any distinguishable landmarks. The landmarks that the robot chooses to extract must be such that they can be correctly recognized again at a future time step. Any error in landmark extraction can have catastrophic effects on the robot's estimation of the map.

**Data Association:** After extracting all recognizable landmarks in its environment, the robot must compare each observed landmark to the landmarks that have been observed at previous time steps. The robot matches up the landmarks that correspond and compares the observed landmark locations to a prediction of where the landmarks would have been had the state estimation step been completely accurate. This creates an error factor representing this difference, and is known as the innovation error.

**State Update:** The robot uses the innovation error to modify the prediction of its state generated in the state estimation step. This is done my multiplying the innovation error by a statistically optimal factor called the Kalman Gain, and adding it to the predicted state. The robot has now formed its best estimate of its state for that time step.

**Landmark Update:** The robot then takes any new landmarks that it has observed and adds them to the state so that they may be reobserved at a later time step.

In order to implement the Kalman Filter in the SLAM process, we have to describe the system. Therefore, it will be assumed that the robot moves in a linear fashion, and that the system can be described by the following equations:

$$x_{k+1} = F_k x_k + G_k u_k + v_k \tag{1}$$

$$y_k = H_k x_k + w_k \tag{2}$$

**Figure 6:** SLAM Process

Equation (1) is the state equation and indicates that the robot state at the next time step is a linear function of the current state, $x_k$, the input vector, $u_k$, and an error term, $v_k$.
Equation (2) is the output equation and indicates that the measurment vector taken from the sensors is a linear function of the robot state, $x_k$, and an error term, $w_k$

The variables are as follows (Assuming $n$ landmarks have been observed):
$\underline{x_k}$ : Robot state. This is a $(3 + 2n) \times 1$ vector containing the $x, y$ estimate of the robot's position as well as the estimated angle, $\theta$, that the robot is pointed. The state also contains the estimated $x, y$ coordinates of every previously observed landmark in the environment.

$$x_k = \begin{bmatrix} x_r \\ y_r \\ \theta_r \\ x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix}$$

$\underline{u_k}$ : System input. This is a $3 \times 1$ vector containing the predicted change in robot $x, y, \theta$ to the next time step.

$$u_k = \begin{bmatrix} \Delta x_r \\ \Delta y_r \\ \Delta \theta_r \end{bmatrix}$$

$\underline{y_k}$ : System measurement. This is a $2n \times 1$ vector containing observed, sensor measurements of the $x, y$ coordinates for each landmark with respect to the robot's coordinate frame.

$$y_k = \begin{bmatrix} x_1 - x_r \\ y_1 - x_r \\ \vdots \\ x_n - x_r \\ y_n - x_r \end{bmatrix}$$

$\underline{v_k}$ : Process noise. This is a $(3 + 2n) \times 1$ vector containing the error in the system e.g. slipping

wheels. The error is assumed to be Gaussian with zero mean.

$$v_k = \begin{bmatrix} \epsilon_{x_r} \\ \epsilon_{y_r} \\ \epsilon_{\theta_r} \\ \epsilon_{x_1} \\ \epsilon_{y_1} \\ \vdots \\ \epsilon_{x_n} \\ \epsilon_{y_n} \end{bmatrix}$$

$\underline{w_k}$ : Measurement noise. This is a $2n \times 1$ vector containing the error in the sensor measurements. The error is assumed to be Gaussian with zero mean.

$$w_k = \begin{bmatrix} \epsilon_{x_1 - x_r} \\ \epsilon_{y_1 - y_r} \\ \vdots \\ \epsilon_{x_n - x_r} \\ \epsilon_{y_n - y_r} \end{bmatrix}$$

$\underline{V_k}$ : Covariance matrix of the process noise vector. This is a $(3+2n) \times (3+2n)$ matrix

$$V_k = \begin{bmatrix} Var(\epsilon_{x_r}) & Cov(\epsilon_{x_r}, \epsilon_{y_r}) & Cov(\epsilon_{x_r}, \epsilon_{\theta_r}) & \dots & Cov(\epsilon_{x_r}, \epsilon_{y_n}) \\ Cov(\epsilon_{y_r}, \epsilon_{x_r}) & Var(\epsilon_{y_r}) & Cov(\epsilon_{y_r}, \epsilon_{\theta_r}) & \dots & Cov(\epsilon_{y_r}, \epsilon_{y_n}) \\ Cov(\epsilon_{\theta_r}, \epsilon_{x_r}) & Cov(\epsilon_{\theta_r}, \epsilon_{y_r}) & Var(\epsilon_{\theta_r}) & \dots & Cov(\epsilon_{\theta_r}, \epsilon_{y_n}) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ Cov(\epsilon_{y_n}, \epsilon_{x_r}) & Cov(\epsilon_{y_n}, \epsilon_{y_r}) & Cov(\epsilon_{y_n}, \epsilon_{\theta_r}) & \dots & Var(\epsilon_{y_n}) \end{bmatrix}$$

$\underline{W_k}$ : Covariance matrix of the measurement noise vector. This is a $2n \times 2n$ matrix

$$W_k = \begin{bmatrix} Var(\epsilon_{x_1 - x_r}) & Cov(\epsilon_{x_1 - x_r}, \epsilon_{y_1 - y_r}) & \dots & Cov(\epsilon_{x_1 - x_r}, \epsilon_{y_n - y_r}) \\ Cov(\epsilon_{y_1 - y_r}, \epsilon_{x_1 - x_r}) & Var(\epsilon_{y_1 - y_r}) & \dots & Cov(\epsilon_{y_1 - y_r}, \epsilon_{y_n - y_r}) \\ \vdots & \vdots & \ddots & \vdots \\ Cov(\epsilon_{y_n - y_r}, \epsilon_{x_1 - x_r}) & Cov(\epsilon_{y_n - y_r}, \epsilon_{y_1 - y_r}) & \dots & Var(\epsilon_{y_n - y_r}) \end{bmatrix}$$

$\underline{F_k}$ : State Transition Matrix. This matrix maps the previous state to the next state. For the purposes of this project, this will just be the $(3+2n) \times (3+2n)$ identity matrix since we want there to be a direct mapping.

$$F_k = \begin{bmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

$\underline{G_k}$ : Input Transition Matrix. This matrix maps the input vector to the next state. For the purposes of this project, this will just be a $(3+2n) \times 3$ matrix where the top $3 \times 3$ matrix is the

identity matrix. This is the case since we want the $\Delta x, \Delta y, \Delta\theta$ from the input vector to get mapped to only the $x, y, \theta$ elements of the state vector.

$$G_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}$$

$\underline{H_k}$ : Measurment Transition Matrix.This matrix maps the state vector to the measurement vector. For the purposes of this project, this will just be a $2n \times (3+2n)$ matrix consisting of a $H_i$ submatrix for each observed landmark.

$$H_k = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_n \end{bmatrix} \qquad H_i = \begin{bmatrix} -1 & 0 & 0 & \dots & 1 & 0 & \dots & 0 \\ 0 & -1 & 0 & \dots & 0 & 1 & \dots & 0 \end{bmatrix}$$

Each $H_i$ corresponds to the $i^{th}$ landmark and is arranged with enough 0's in each row so that the $2 \times 2$ identity matrix lines up with the $i^{th}$ landmark in the state vector. This arrangement is required so that $H_k x_k$ takes the same form as the measurement vector:

$$H_k x_k = \begin{bmatrix} x_1 - x_r \\ y_1 - y_r \\ \vdots \\ x_n - x_r \\ y_n - y_r \end{bmatrix}$$

### 2.3.2 Kalman Filter Derivation [2]

The robot can never know its state with absolute certainty due to measurement and process error. Therefore, we will keep track of the robot state as the mean of a normal distribution. We know from basic probability that the normal distribution is given by:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{3}$$

Where $\mu$ and $\sigma^2$ denote the mean and variance respectively.
Now, since the robot state is a vector containing multiple variables, we must keep track of the mean of the joint distribution of these variables.
Now, we denote the joint normal distribution of $n$ variables as:

$$p(x) = \frac{1}{\sqrt{(2\pi)^n |P|}}\, e^{-\frac{1}{2}(x-\mu)^T P^{-1}(x-\mu)} \tag{4}$$

Here, $x$ denotes the vector of variables. $P$ denotes their covariance matrix, with $|P|$ being the determinant of the covariance matrix. And $\mu$ is the vector containing the means of the $n$ variables.

The Kalman Filter works by predicting the robot state at the next time step based on known information, and then correcting this prediction at the following time step based upon measured

data at that time.
Therefore, we will need to find an equation for the prediction of the robot state.
The robot state will be predicted by plugging the current robot state into the state equation. We also know that the process noise has mean 0, so we predict $v_k$ to be 0 and get:

$$\hat{x}_{k+1} = F_k \hat{x}_k + G_k u_k \tag{5}$$

Note here that the hat on $x$ denotes that it is an estimate rather than the actual state, and that the subscript denotes that it is an estimate of the state at time step $k+1$. It should also be noted that we will use a superscript $-$ (e.g. $\hat{x}_k^-$) to denote that the estimate is performed with information up to time $k-1$. This makes both the notation $\hat{x}_k^-$ and $\hat{x}_{k+1}$ predictions since they are estimates of a certain time step with information up to the previous time step.

While the important variable to keep track of is the robot state, it turns out that in order to do this, we need to also keep track of the covariance matrix as well. (We will see this later) Therefore, we must find a formula for the prediction of the covariance matrix.
Before we do this, we will define some notation.
The definition of covariance for a vector of random variables, $\mathbf{X} = X_1, X_2, ..., X_n$, is

$$Cov(\mathbf{X}) = E\big[(x-\mu)(x-\mu)^T\big] \tag{6}$$

We know that the means of the vectors $v_k$ and $w_k$ are zero. Therefore, we get

$$Cov\big(v_k\big) = E\Big[\big(v_k - 0\big)\big(v_k - 0\big)^T\Big] = E\Big[v_k v_k^T\Big] \triangleq V_k \tag{7}$$

$$Cov\big(w_k\big) = E\Big[\big(w_k - 0\big)\big(w_k - 0\big)^T\Big] = E\Big[w_k w_k^T\Big] \triangleq W_k \tag{8}$$

Also, we are considering the robot state to be the mean of a normal distribution, therefore, we may consider the state prediction to be the mean of the distribution. This being said, we get:

$$Cov\big(x_k\big) = E\Big[\big(x_k - \hat{x}_k\big)\big(x_k - \hat{x}_k\big)^T\Big] \triangleq P_k \tag{9}$$

Note that $V$, $W$, and $P$ are all matricies, and that $P$ is the covariance matrix of $x$

So the prediction of the covariance matrix is:

$$P_{k+1} = E\Big[\big(x_{k+1} - \hat{x}_{k+1}\big)\big(x_{k+1} - \hat{x}_{k+1}\big)^T\Big] \tag{10}$$

We now substitute equations (1) and (5) into equation (10).

$$
\begin{aligned}
P_{k+1} &= E\Big[\big(F_k x_k + G_k u_k + v_k - F_k \hat{x}_k - G_k u_k\big)\big(F_k x_k + G_k u_k + v_k - F_k \hat{x}_k - G_k u_k\big)^T\Big] \\
&= E\Big[\big(F_k(x_k - \hat{x}_k) + G_k(u_k - u_k) + v_k\big)\big(F_k(x_k - \hat{x}_k) + G_k(u_k - u_k) + v_k\big)^T\Big] \\
&= E\Big[\big(F_k(x_k - \hat{x}_k) + v_k\big)\big(F_k(x_k - \hat{x}_k) + v_k\big)^T\Big] \\
&= E\Big[\big(F_k(x_k - \hat{x}_k) + v_k\big)\big((x_k - \hat{x}_k)^T F_k^T + v_k^T\big)\Big] \\
&= E\Big[F_k\big(x_k - \hat{x}_k\big)\big(x_k - \hat{x}_k\big)^T F_k^T + F_k\big(x_k - x_k\big)v_k^T + v_k\big(x_k - x_k\big)^T F_k^T + v_k v_k^T\Big] \\
&= E\Big[F_k\big(x_k - \hat{x}_k\big)\big(x_k - \hat{x}_k\big)^T F_k^T\Big] + E\Big[F_k\big(x_k - x_k\big)v_k^T\Big] + E\Big[v_k\big(x_k - x_k\big)F_k^T\Big] + E\Big[v_k v_k^T\Big]
\end{aligned}
$$

Since $v_k$ is independent from $F_k$, $x_k$ and $\hat{x}_k$, we get:

$$
P_{k+1} = E\Big[F_k\big(x_k - \hat{x}_k\big)\big(x_k - \hat{x}_k\big)^T F_k^T\Big] + E\Big[F_k\big(x_k - x_k\big)\Big]E\Big[v_k^T\Big] + E\Big[v_k\Big]E\Big[\big(x_k - x_k\big)F_k^T\Big] + E\Big[v_k v_k^T\Big]
$$

Since $E[v_k] = E[v_k]^T = \mathbf{0}$ we get:

$$
\begin{aligned}
P_{k+1} &= E\Big[F_k\big(x_k - \hat{x}_k\big)\big(x_k - \hat{x}_k\big)^T F_k^T\Big] + E\Big[v_k v_k^T\Big] \\
&= F_k E\Big[\big(x_k - \hat{x}_k\big)\big(x_k - \hat{x}_k\big)^T\Big] F_k^T + E\Big[v_k v_k^T\Big] \\
&= F_k P_k F_k^T + V_k
\end{aligned}
$$

We have now found the equation for the prediction of the covariance matrix

$$
P_{k+1} = F_k P_k F_k^T + V_k \tag{11}
$$

We will now derive the equation for the updated state.
We would like this update to somehow merge both the prediction that we have already made, as well as the measurment that the robot has received from its sensors. Therefore, we may write:

$$
\hat{x}_k = \hat{x}_k^- + K_k\big(y_k - H_k \hat{x}_k^-\big) \tag{12}
$$

Where $K_k$ is called the Kalman Gain and can be thought of as a blending factor.
First note that $H_k \hat{x}_k^-$ is the same as a predicted measurement. Therefore, we can see Equation (12) as modifying the state prediction by some blending factor times the difference between the actual, measured output, and the predicted output.
Note that the state prediction will be altered more if this difference is greater, and altered less if the difference is smaller.

We must now decide what forms the optimal Kalman Gain, $K_k$
For this, we return to the Covariance Matrix. We will illistrate a 3x3 example for clarity. Let

$X_1, X_2, X_3$ be three random variables, and $X$ denote the vector consisting of these three variables. Then we have:

$$Cov(X) = E\left( \begin{bmatrix} X_1 - \mu_{x_1} \\ X_2 - \mu_{x_2} \\ X_3 - \mu_{x_3} \end{bmatrix} \begin{bmatrix} X_1 - \mu_{x_1} & X_2 - \mu_{x_2} & X_3 - \mu_{x_3} \end{bmatrix} \right)$$

$$= \begin{bmatrix} Cov(X_1, X_1) & Cov(X_1, X_2) & Cov(X_1, X_3) \\ Cov(X_2, X_1) & Cov(X_2, X_2) & Cov(X_2, X_3) \\ Cov(X_3, X_1) & Cov(X_3, X_2) & Cov(X_3, X_3) \end{bmatrix}$$

$$= \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & Cov(X_1, X_3) \\ Cov(X_2, X_1) & Var(X_2) & Cov(X_2, X_3) \\ Cov(X_3, X_1) & Cov(X_3, X_2) & Var(X_3) \end{bmatrix}$$

As this example illistrates, the diagonal elements of the Covariance Matrix, $P$ form a vector containing the variances of the elements of the state vector, $x$

Now we can think of a normal distribution with a large variance to have a large degree of uncertainty, and also a normal distribution with a small variance to have a small degree of uncertainty. Therefore, we will choose the Kalman Gain, $K$, which minimizes the trace of the Covariance Matrix. We will acomplish this by taking the derivative of the trace of the Covariance Matrix with respect to the Kalman Gain, setting it equal to zero, and solving for the Kalman Gain.

Here, the trace is a scalar, and we define the derivative of a scalar with respect to a matrix as

$$\frac{ds}{dA} = \begin{bmatrix} \frac{ds}{da_{11}} & \frac{ds}{da_{12}} & \cdots \\ \frac{ds}{da_{21}} & \frac{ds}{da_{22}} & \cdots \\ \vdots & \vdots & \end{bmatrix}$$

We will now prove the following two Lemmas:

**Lemma1:** Let $A, B$ be matrices, and $AB$ be a square matrix. Then $\frac{d[trace(AB)]}{dA} = B^T$

*Proof.*

We know the definition of matrix multiplication is such that

$$(ab)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Then the trace of $AB$ will be the sum of the diagonal elements, thus the sum while setting $i = j$

$$trace(AB) = \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{ki}$$

We now take the derivative with respect to $A$ as defined above and get

$$\frac{d[trace(AB)]}{dA} = \begin{bmatrix} \frac{d}{da_{11}} \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{ki} & \frac{d}{da_{12}} \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{ki} & \cdots \\ \frac{d}{da_{21}} \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{ki} & \frac{d}{da_{22}} \sum_{i=1}^{n} \sum_{k=1}^{n} a_{ik} b_{ki} & \cdots \\ \vdots & \vdots & \end{bmatrix}$$

Each term is of the form $\frac{d}{da_{lm}}\sum_{i=1}^{n}\sum_{k=1}^{n}a_{ik}b_{ki}$. We can see that all terms of this sum will be constant with respect to $a_{lm}$ except for the terms in which $i=l$ and $k=m$
Therefore, the matrix simplifies to

$$\begin{aligned}\frac{d[trace(AB)]}{dA} &= \begin{bmatrix} \frac{d}{da_{11}}a_{11}b_{11} & \frac{d}{da_{12}}a_{12}b_{21} & \ldots \\ \\ \frac{d}{da_{21}}a_{21}b_{12} & \frac{d}{da_{22}}a_{22}b_{22} & \ldots \\ \vdots & \vdots & \end{bmatrix} \\ &= \begin{bmatrix} b_{11} & b_{21} & \ldots \\ b_{12} & b_{22} & \ldots \\ \vdots & \vdots & \end{bmatrix} \\ &= B^T \end{aligned}$$

□

**Lemma2:** Let $A, B$ be matrices, and $B$ be a symmetric matrix. Then $\frac{d[trace(ABA^T)]}{dA} = 2AB$

*Proof.*
We know the definition of matrix multiplication is such that

$$(abc)_{ij} = \sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}c_{lj}$$

Therefore, we get

$$(aba^T)_{ij} = \sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}a_{jl}$$

Then the trace of $ABA^T$ will be the sum of the diagonal elements, thus the sum while setting $i=j$

$$trace(ABA^T) = \sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}a_{il}$$

We now take the derivative with respect to $A$ as defined above and get

$$\frac{d[trace(AB)]}{dA} = \begin{bmatrix} \frac{d}{da_{11}}\sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}a_{il} & \frac{d}{da_{12}}\sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}a_{il} & \ldots \\ \\ \frac{d}{da_{21}}\sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}a_{il} & \frac{d}{da_{22}}\sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}a_{il} & \ldots \\ \\ \vdots & \vdots & \end{bmatrix}$$

Each term is of the form $\frac{d}{da_{qm}}\sum_{i=1}^{n}\sum_{l=1}^{n}\sum_{k=1}^{n}a_{ik}b_{kl}a_{il}$. We can see that all terms of this sum will be constant with respect to $a_{qm}$ except for the terms in which $i=q$ and $k=m$ or for which $i=q$

and $l = m$. Therefore, each term will simplify to

$$\begin{aligned}
&\frac{d}{da_{qm}} \sum_{l=1}^{n} a_{qm} b_{ml} a_{ql} + \frac{d}{da_{qm}} \sum_{k=1}^{n} a_{qk} b_{km} a_{qm} \\
&= \sum_{l=1}^{n} b_{ml} a_{ql} + \sum_{k=1}^{n} a_{qk} b_{km} \\
&= \sum_{k=1}^{n} a_{qk} b_{mk} + \sum_{k=1}^{n} a_{qk} b_{km} \qquad \text{Dummy index/commutivity} \\
&= \sum_{k=1}^{n} a_{qk} b_{km} + \sum_{k=1}^{n} a_{qk} b_{km} \qquad \text{B is symmetric} \\
&= 2 \sum_{k=1}^{n} a_{qk} b_{km} \\
&= 2(ab)_{qm}
\end{aligned}$$

Therefore, the matrix simplifies to

$$\begin{aligned}
\frac{d[trace(ABA^T)]}{dA} &= \begin{bmatrix} 2(ab)_{11} & 2(ab)_{12} & \dots \\ 2(ab)_{21} & 2(ab)_{22} & \dots \\ \vdots & \vdots & \end{bmatrix} \\
&= 2AB
\end{aligned}$$

□

Now that we have these two Lemmas at our disposal, we will derive the optimal Kalman Gain. Recall from Linear Algebra the following identities:

$$\big(A + B\big)^T = A^T + B^T \tag{13}$$

$$\big(AB\big)^T = B^T A^T \tag{14}$$

Now if we plug equation (2) into equation (12), we get:

$$\hat{x}_k = \hat{x}_k^- + K_k\big(H_k x_k + w_k - H_k \hat{x}_k^-\big)$$

Now, plugging this into equation (9), we get:

$$P_k = E\bigg[\Big(x_k - \hat{x}_k^- - K_k\big(H_k x_k + w_k - H_k \hat{x}_k^-\big)\Big)\Big(x_k - \hat{x}_k^- - K_k\big(H_k x_k + w_k - H_k \hat{x}_k^-\big)\Big)^T\bigg]$$

Now, using the linear algebra facts (13) and (14) and regrouping some terms we get:

$$P_k = E\bigg[\Big(\big(x_k - \hat{x}_k^-\big) - K_k\Big(H_k\big(x_k - \hat{x}_k^-\big) + w_k\Big)\Big)\Big(\big(x_k - \hat{x}_k^-\big)^T - \Big(\big(x_k - \hat{x}_k^-\big)^T H_k^T + w_k^T\Big)K_k^T\Big)\bigg]$$

Now, doing out the FOIL multiplication gives:

$$\begin{aligned}
P_k = E\bigg[&\Big(x_k - \hat{x}_k^-\Big)\Big(x_k - \hat{x}_k^-\Big)^T - \Big(x_k - \hat{x}_k^-\Big)\Big(\big(x_k - \hat{x}_k^-\big)^T H_k^T + w_k^T\Big)K_k^T \\
&- K_k\Big(H_k\big(x_k - \hat{x}_k^-\big) + w_k\Big)\Big(x_k - \hat{x}_k^-\Big)^T + K_k\Big(H_k\big(x_k - \hat{x}_k^-\big) + w_k\Big)\Big(\big(x_k - \hat{x}_k^-\big)^T H_k^T + w_k^T\Big)K_k^T\bigg]
\end{aligned}$$

Linearity of Expectation tells us that $E[X+Y]=E[X]+E[Y]$. Also, if $X$ and $Y$ are independent, then $E[XY]=E[X]E[Y]$. Therefore, since $w(k)$ is independent from everything but itself and $w_k^T$ and the fact that $E[w_k]=E[w_k^T]=0$, we get:

$$\begin{aligned}P_k = E\Big[(x_k-\hat{x}_k^-)(x_k-\hat{x}_k^-)^T\Big] - E\Big[(x_k-\hat{x}_k^-)(x_k-\hat{x}_k^-)^T H_k^T K_k^T\Big] \\ - E\Big[K_k H_k (x_k-\hat{x}_k^-)(x_k-\hat{x}_k^-)^T\Big] + E\Big[K_k H_k (x_k-\hat{x}_k^-)(x_k-\hat{x}_k^-)^T H_k^T K_k^T\Big] + E\Big[K_k w_k w_k^T K_k^T\Big]\end{aligned}$$

Now we can recognize $P_k^- = E\Big[(x_k-\hat{x}_k^-)(x_k-\hat{x}_k^-)^T\Big]$

and $W_k = E\Big[w_k w_k^T\Big]$ giving us:

$$P_k = P_k^- - P_k^- H_k^T K_k^T - K_k H_k P_k^- + K_k H_k P_k^- T H_k^T K_k^T + K_k W_k K_k^T \tag{15}$$

We now have a formula for the covariance matrix in terms of the Kalman Gain. Therefore, we can take the derivative of its trace and set it equal to zero to find the optimal Kalman Gain.
Noteing that $P^{-T}=P^-$ since $P$ is symmetric, and that $trace[A]=trace[A^T]$ for any matrix $A$, we will use Lemma1 and Lemma2 proved earlier to get:

$$\begin{aligned}\frac{d[traceP]}{dK} &= \frac{d}{dK}trace\Big[P_k^- - P_k^- H_k^T K_k^T - K_k H_k P_k^- + K_k H_k P_k^- H_k^T K_k^T + K_k W_k K_k^T\Big] \\ &= \frac{d}{dK}trace\Big[P_k^- - (K_k H_k P_k^-)^T - K_k H_k P_k^- + K_k H_k P_k^- H_k^T K_k^T + K_k W_k K_k^T\Big] \\ &= \frac{d}{dK}trace\Big[P_k^- - 2K_k H_k P_k^- + K_k H_k P_k^- H_k^T K_k^T + K_k W_k K_k^T\Big] \\ &= -2(H_k P_k^-)^T + 2K_k H_k P_k^- H_k^T + 2K_k W_k \\ &= -2P_k^- H_k^T + 2K_k\big(H_k P_k^- H_k^T + W_k\big)\end{aligned}$$

Setting this equal to zero, we can solve for

$$K_k = P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \tag{16}$$

Having now solved for the optimal Kalman Gain, we may plug this equation back into equation (12) to get a formula for the state update

$$\hat{x}_k = \hat{x}_k^- + P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1}\big(y_k - H_k \hat{x}_k^-\big) \tag{17}$$

Having solved for the optimal Kalman Gain, we may go back and plug this into equation (15) to

get an update formula for the covariance matrix.

$$
\begin{aligned}
P_k = P_k^- &- P_k^- H_k^T \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big)^T \\
&- \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big) H_k P_k^- \\
&+ \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big) H_k P_k^- H_k^T \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big)^T \\
&+ \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big) W_k \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big)^T \\
= P_k^- &- P_k^- H_k^T \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big)^T - \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big) H_k P_k^- \\
&+ P_k^- H_k^T \Big( H_k P_k^- H_k^T + W_k \Big)^{-1} \Big( H_k P_k^- H_k^T + W_k \Big) \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big)^T \\
= P_k^- &- P_k^- H_k^T \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big)^T - \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big) H_k P_k^- \\
&+ P_k^- H_k^T \Big( P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \Big)^T
\end{aligned}
$$

$$
P_k = P_k^- - P_k^- H_k^T \Big( H_k P_k^- H_k^T + W_k \Big)^{-1} H_k P_k^- \tag{18}
$$

So we now have found formulas for both the prediction and update steps.
Note that we clearly need the prediction of the covariance matrix in order to perform the update steps, so we were correct to derive this prediction.

**Prediction:**

$$
\hat{x}_{k+1} = F_k \hat{x}_k + G_k u_k \tag{19}
$$

$$
P_{k+1} = F_k P_k F_k^T + V_k \tag{20}
$$

**Update:**

$$
\hat{x}_k = \hat{x}_k^- + K_k \big(y_k - H_k \hat{x}_k^-\big) \tag{21}
$$

$$
P_k = P_k^- - K_k H_k P_k^- \tag{22}
$$

$$
K_k = P_k^- H_k^T \big(H_k P_k^- H_k^T + W_k\big)^{-1} \tag{23}
$$

### 2.3.3 Kalman Conclusion

Given the derivation above, Linear Kalman Filter SLAM becomes as simple as running the matrices described in section 2.3.1 through equations 19-23 in the Predict-Update fashion shown in Figure 5. Note that for most SLAM robots, including this project, the prediction step takes the form of odometry. It is much easier to read odometry data than to predict such data based on the actions that will be applied to the motors. This also implies that both the prediction and the update steps actually occur at the same time step. However, the theory is the same since the odometry data still forms a prediction that can be updated by the sensor measurements.

# 3 Implementation

The hardware for this project was chosen to have the necessary components to implement a SLAM algorithm. Therefore, the robot needed the following:
(1.) A base capable of supporting the rest of the hardware. The base had to be capable of receiving and executing motor commands, as well as generating and sending odometry data.
(2.) A vision sensor capable of generating RGB images of the environment in real time. This was necessary in order to identify and track landmarks in the environment.
(3.) A distance sensor capable of determining the distances to each landmark in the environment.
(4.) A processor capable of interfacing with the aforementioned hardware and containing the processing power to run a SLAM algorithm in real time.

The chosen robot system is shown in Figure 7 below. An iRobot Create was chosen as the robot base, an Xbox Kinect was chosen to act as both the vision and distance sensor, and an HP laptop running LabVIEW was chosen as the processor. The individual components are described in the following sections.

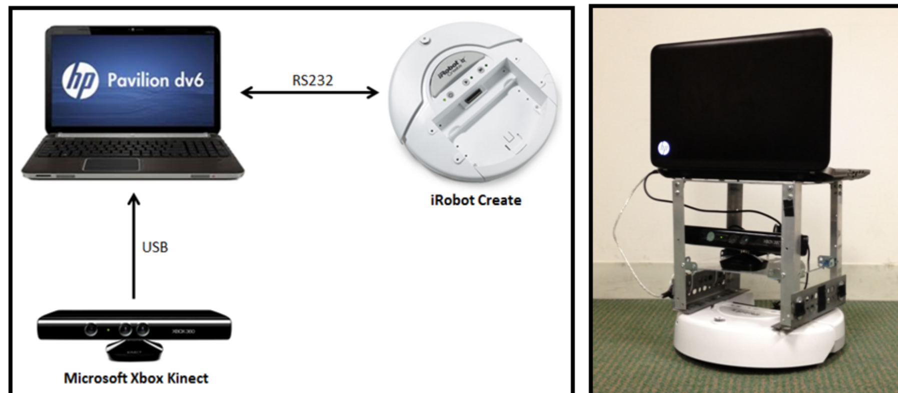


**Figure 7:** Our Robot

## 3.1 iRobot Create

The iRobot Create is a robotics platform manufactured by the iRobot Corporation. It is similar to the Roomba robotic vacuum cleaner, but contains no vacuum, and is easily programmable through RS232 Serial Communication. The iRobot Create was specifically chosen due to the ease with which it can be programmed, as well as the accuracy of its odometry. While custom shaft encoders could have been constructed, they would not have been nearly as accurate as a commercial product, nor would this process have aided in a better understanding of the SLAM algorithm, which was the main goal of the project. The iRobot Create is also relatively inexpensive, and was readily available in that the Trinity Engineering Department owns three of them.

**Figure 8:** iRobot Create

The iRobot Create is set up to receive 8-bit opcodes at 57600 baud that instruct it in what to do. For this project, the only functionalities that were utilized were sending motor commands, and receiving odometry data from the built in shaft encoders. The necessary opcodes for each of these functions are shown below in Table 1. The four functions that were realized in the project are shown in Table 2 below. A separate LabVIEW VI was written to achieve each.

The Initialize VI configures the serial port to 57600 baud, 8 data bits, and then sends the opcodes $128_{10}$ $132_{10}$ to start the Create and put it in full mode. It also reads the encoders as discussed below to clear their values.

The Send Motor Commands VI sends the opcode $145_{10}$ in order to enable motor control. It then sends four bytes that specify the speed of the motors in mm per second. The first two bytes are the high and low bytes of the right motor speed, and the second two bytes are the high and low bytes of the left motor speed.

The Get Odometry Distance VI sends the opcode $142_{10}$ in order to request a data packet from the sensors. It then sends the opcode $19_{10}$ to specify that the odometry distance should be sent. The Create sends the change in average distance traveled by both wheels since the last time it was queried.

The Get Odometry Angle VI sends the opcode $142_{10}$ in order to request a data packet from the sensors. It then sends the opcode $20_{10}$ to specify that the odometry angle should be sent. The Create sends the change angle since the last time it was queried.

| **Opcode** | **Function** |
|---|---|
| $128_{10}$ | Start Create |
| $131_{10}$ | Put Create in Safe Mode |
| $132_{10}$ | Put Create in Full Mode |
| $145_{10}$ | Allow Motor Control |
| $142_{10}$ | Request Sensor Data Packet |
| $19_{10}$ | Identify Encoder Distance Packet |
| $20_{10}$ | Identify Encoder Angle Packet |

Table 1: Create Opcodes

| **Initialize** | **Send Motor Commands** | **Get Odm. Distance** | **Get Odm. Angle** |
|---|---|---|---|
| $128_{10}$<br>$132_{10}$ | $145_{10}$<br>$RH_{10}$<br>$RL_{10}$<br>$LH_{10}$<br>$LL_{10}$ | $142_{10}$<br>$19_{10}$ | $142_{10}$<br>$20_{10}$ |

Table 2: Create Functions

## 3.2 Kinect

The Xbox Kinect, as described in Section 2.1, was chosen for the precise reason that it is capable of fulfilling two roles. The Kinect is capable of generating both an RGB image stream, as well as calculating the distance to every point in view. This is extremely useful for this project since the distances to each landmark must be found. The Kinect allows for a step to be cut out of the process. Image processing code had to be written to identify and track landmarks, but since the

Kinect allowed for the distance to a specified point on the image to be returned, there was no need to match a landmark in an RGB image to a distance measurement from a separate sensor.

### 3.2.1 OpenNI Interface

The OpenNI interface was chosen to provide the necessary drivers needed to communicate with the Xbox Kinect. Currently there are three driver packages available for the Kinect  Microsoft Kinect SDK, OpenNI, and the drivers provided by Ryan Gordon. After spending a considerable amount of time during the first semester trying to install the Ryan Gordon drivers, communication with National Instruments suggested the Microsoft Kinect SDK. While this package had been developed by Microsoft and was relatively simple to get running, it was found that there were far too many limitations on the image sample rate as well as the functional capabilities. The Microsoft SDK allowed the Kinect to report both RGB and Depth data, but a processor speed of 2.66 GHz was required for the image stream to run at a reasonable frame rate. Also, this package did not include the tools to correlate the RGB and depth data. Continuing our research, it was discovered that the OpenNI interface provided faster image processing and a more seamless integration with LabVIEW. The OpenNI package solved both of the previously addresse issues. There was no limitation on processor speed, and there was a built in function to return the distance to any pixel in the RGB image. These qualities made the OpenNI interface ideal for this project. The drivers were discovered on the main OpenNI website and installed through a complete automatic installer found at [15].

The OpenNI package installs drivers necessary to interface with the Kinect, as well as a vast library of C++ code to utilize the many Kinect functions. Therefore, in order to run the Kinect, the correct sequence of Methods from this C++ code had to be called from LabVIEW. In order to expedite this process, some sample code was downloaded from [16]. The VI's from this sample code were able to be rearranged in order to generate an RGB image and then find the distance to a specified point on that image.

### 3.2.2 Landmark Extraction

The robot now needs to be able to analyze that image and determine 1) if there are landmarks present and 2) where those landmarks are located. In order to demonstrate a proof of concept for SLAM, very simple landmarks were chosen. Red, blue, and green cylinders were used so that they would be easy to identify and track, allowing the main focus of the project to be on the SLAM algorithm. Therefore, having predetermined the landmarks for the environment, the robot is supplied with a small image file of its target landmarks as shown in Figure 9. The only data that the

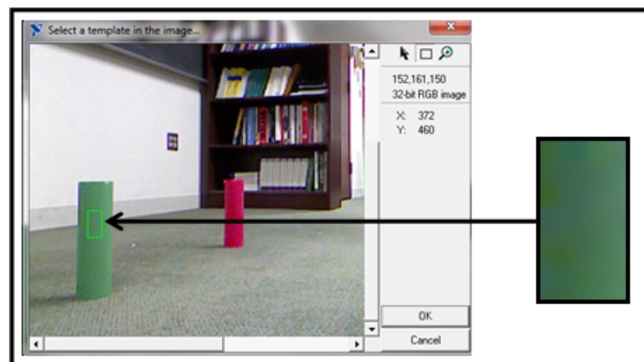


**Figure 9:** Landmark Extraction

robot knows about its environment are the RGB template samples taken from a calibration image for each landmark. The green box, which has been expanded in Figure 9, will act as a color sample for the green landmark. With this information, the robot is constantly checking its field of view for an identical color sample.

The Detect LMs (Landmarks) VI performs the task of extracting the pixel coordinates for each landmark. LabVIEW makes this process relatively simple through the Vision Assistant. The programmer only has to specify the color template to be tracked, and the Vision Assistant creates code to perform this function. Once the sample RGB color template has been found in the live image, the robot then begins to extract depth data from the image based on the indices in which it was found. This is done by calling a sequence of new OpenNI methods. The methods locate the landmark and its pixel location within the image. In this VI, three landmarks are being detected simultaneously: the red, blue, and green landmarks.

The Convert Kinect Data to r,theta VI is responsible for converting the sensor data from the Kinect into an actual measure value. Depth is simply extracted since it is already a value of distance in mm. However, the angle of the landmark from Kinect still needs to be determined. Using basic trigonometry a conversion equation was calculated which output an angle value as a linear function of the X coordinate of the located landmark.

While the Xbox Kinect was a pain for nearly the entire year, it proved its worth in the end. The Kinect was able to very accurately track and determine the distance to the landmarks within its view.

### 3.3 SLAM

The SLAM part of the LabVIEW program functioned by manipulating a single data type, or cluster, as is called in LabVIEW. This data type contained four components:

(I.) Landmark Array: This is an array containing data about every landmark that has been observed in the current time step. Each element of the array contains the following:
- (a.) The landmark ID: A unique ID is assigned to each landmark the first time it is observed and is used to distinguish between different landmarks
- (b.) The observed x and y coordinates of the landmark at that time step
- (c.) The covariance matrix associated with that landmark.

(II.) System Matrices: This is a data type (cluster) containing the F, G, H, V, W, K matrices as described in section 2.3.1

(III.) Kalman Estimate Matrices: This is a data type (cluster) containing the X, P matrices as described in section 2.3.1

(IV.) LM ID's: This is an array containing the ID's of every landmark that has been observed since the program began.

### 3.3.1 Initialization

In order for the program to run correctly, each matrix had to initialized to the correct value. Clearly, the robot starts having recognized no landmarks, and thus, $n = 0$. This results in $F$ and $G$ getting set to the $3 \times 3$ identity matrix. Similarly, $H$ and $K$ get set to null, empty matrices. It will be assumed that the robot begins at position $x = 0$, $y = 0$, $\theta = 0$ resulting in $x$ being set to the $3 \times 1$ zero matrix. For this project, $\theta$ is considered to be bearing, or degrees from north. Therefore, the robot is considered to start facing north, and a positive change in $\theta$ is defined as a clockwise rotation. However, it should also be noted that from the robot's perspective, landmarks are identified by the traditional definition of angle, where zero is along the x-axis, and a positive increase is a counterclockwise rotation. These two angles are illustrated in Figure 10.

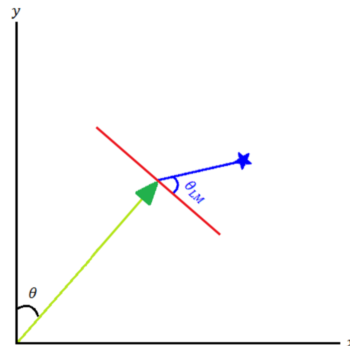

**Figure 10:** Angles

The only subjective matrices in terms of initialization are $V$, $W$ and $P$. The initial inclination might be to set them all to zero matrices. However, referring to section 2.3.1, this would imply that the variance of both odometry and measurement would be zero. This is clearly false since the entire purpose of SLAM is to account for error. Therefore, it is safe to assume that all of the variables are independent, and thus that the Covariances are zero. However, the diagonal elements of each of these matrices must be nonzero to indicate that some error is present. For this project, the variances were assumed to all be one. Note that these can be changed to account for either odometry or sensing being more accurate. With this in mind, $P$ was set to the $3 \times 3$ identity matrix, $W$ was set to a null matrix, but the covariance matrix associated with each landmark was set to default to a $2 \times 2$ identity matrix which will be added to $W$ as landmarks are observed. (This will be discussed in Section 3.3.7) $V$ was also set to a $3 \times 3$ identity matrix. However, it should be noted that when $V$ increases size, the added diagonal elements should be set to zero. This reflects the fact that since no landmarks are moving, there is no process error associated with them. The resulting initialization gives the following matrices:

$$F = G = V = P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad H = W = K = [] \qquad X = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

### 3.3.2 Input Vector

As discussed in Section 2.3.1, the Predict step uses odometry data in order to make a prediciton of the robot state. Therefore, the first step is to calculate the input vector, $u$, from the odometry data. As described in section 3.1, the iRobot Create returns odometry data such that distance is reported as the average of the distances that both wheels have traveled since the last time they were polled, and the angle is the change in angle since the last time it was polled. The input vector must be of the form:

$$u_k = \begin{bmatrix} \Delta x_r \\ \Delta y_r \\ \Delta \theta_r \end{bmatrix}$$

Therefore, in order to get calculate the input vector, some calculations must be done. From Figure 11, we see that the reported odometry data of arclength, $S$, and change in angle, $\Delta\theta$, can be transformed into equations for $\Delta x, \Delta y$.

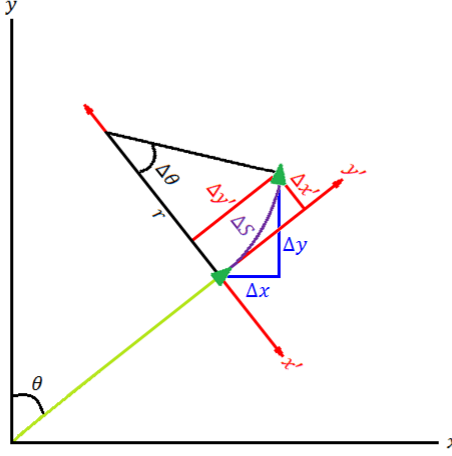


**Figure 11:** Odometry Diagram

We will first find $\Delta x', \Delta y'$. From basic trigonometry, we know that:

$$r = \frac{\Delta S}{\Delta \theta}$$

Solving for $\Delta y'$, we get:

$$\Delta y' = r sin(\Delta\theta)$$
$$\Delta y' = \frac{\Delta S}{\Delta \theta} sin(\Delta\theta)$$

Solving for $\Delta x'$, we get:

$$r - \Delta x' = r cos(\Delta\theta)$$
$$\Delta x' = r\big(1 - cos(\Delta\theta)\big)$$
$$\Delta x' = \frac{\Delta S}{\Delta \theta}\big(1 - cos(\Delta\theta)\big)$$

Now, we may form the vector:

$$\begin{bmatrix} \Delta x' \\ \Delta y' \end{bmatrix} = \begin{bmatrix} \frac{\Delta S}{\Delta \theta}\big(1 - cos(\Delta\theta)\big) \\ \frac{\Delta S}{\Delta \theta} sin(\Delta\theta) \end{bmatrix}$$

If we rotate this vector counterclockwise by $\theta$, we will get the correct values for $\Delta x_r$, $\Delta y_r$. Therefore, we will multiply the previous vector by a rotation matrix in order to get $u_k$

$$\begin{bmatrix} \Delta x_r \\ \Delta y_r \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} \frac{\Delta S}{\Delta \theta}\big(1 - cos(\Delta\theta)\big) \\ \frac{\Delta S}{\Delta \theta} sin(\Delta\theta) \end{bmatrix}$$

This therefore, gives that:

$$u_k = \begin{bmatrix} \Delta x_r \\ \Delta y_r \\ \Delta\theta_r \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{\Delta S}{\Delta\theta}\big(1 - cos(\Delta\theta)\big) \\ \frac{\Delta S}{\Delta\theta} sin(\Delta\theta) \\ \Delta\theta \end{bmatrix}$$

We now have the input vector assuming that $\Delta\theta \neq 0$. However, if $\Delta\theta = 0$, the equations derived above will be dividing by zero and will be incorrect. Therefore, a special case was programmed to handle this event. If $\Delta\theta = 0$, it means that the robot has just traveled straight ahead, and we get:

$$u_k = \begin{bmatrix} (\Delta S)\, sin(\theta) \\ (\Delta S)\, cos(\theta) \\ 0 \end{bmatrix}$$

#### 3.3.3 State Prediction

With the input vector, and all of the updated SLAM matrices, the Kalman Predict, or state estimation, step is relatively simple. The $x$ and $P$ matrices are manipulated according to equations (19) and (20).

#### 3.3.4 Measurement Vector

The measurement vector is generated from the observed $x, y$ coordinates of each landmark found by the Kinect. The process of generating the measurement vector is shown in Figure 12. The first step is to sort out all of the new landmarks. This is necessary since there is no state prediction for these landmarks, and therefore, there will be nothing to update.

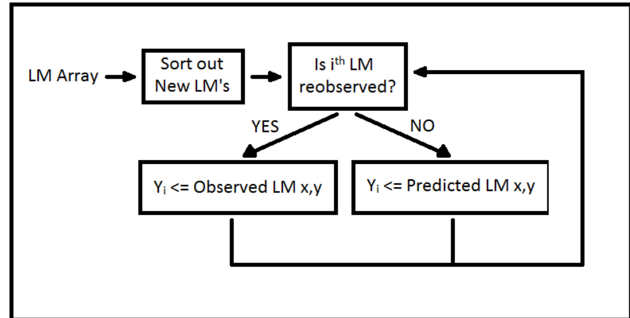


**Figure 12:** Measurement Vector Process

The next step is to sort through each of the previously observed landmarks to see if each has been reobserved in this time step. If the landmark has been reobserved, its $x, y$ coordinates are multiplied by a rotation matrix to get them in the same format as the overall coordinate frame, and added to the measurement vector. If the landmark has not been reobserved, the prediction for where the landmark will be is added to the measurement vector. By setting the measurement of a nonreobserved landmark to its predicted $x, y$ coordinates, the innovation error for that landmark will equal zero. This ensures that the landmark may remain in the state vector while not affecting the estimates of the other landmarks. In this fashion, only reobserved landmarks will factor into the state update.

#### 3.3.5 Calculate Kalman Gain

The Kalman Gain matrix is calculated based on equation (23). However, it was modified to account for nonreobserved landmarks. If the Kalman Gain matrix were allowed to remain unaltered, the innovation errors for reobserved landmarks would alter the estimate of the nonreobserved landmarks. This is not desirable because it is possible, and becomes ever more likely with larger environments, that a nonreobserved landmark will not be particularly close to the reobserved landmarks, and

therefore, should remain unaffected. In order to combat this problem, the rows in the Kalman Gain matrix corresponding to nonreobserved landmarks were zeroed out. This ensured that they would not be affected by the innovation errors in the reobserved landmarks.

### 3.3.6 State Update

With the measurement vector, Kalman Gain matrix, and all of the updated SLAM matrices, the Kalman Update, or state update, step is relatively simple. The $x$ and $P$ matrices are manipulated according to equations (21) and (22).

### 3.3.7 Landmark/System Update

After the Kalman Prediction and Update steps have been completed, any newly observed landmarks need to be added to the robot state. Note that these are landmarks that were ignored during the previous Kalman Update step since there was no prediction for them. A number of things have to happen in order to add new landmarks to the state. Let $m$ denote the number of landmarks to be added. The $F$ matrix grows by $2m$ rows and $2m$ columns to become simply a larger identity matrix:

$$F_k = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

The $G$ matrix grows by $2m$ rows of zeros to become:

$$G_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{bmatrix}$$

The $H$ matrix grows by adding $m$ more $H_i$ matrices to the preexisting $H$ matrix, and updating the $H_i$ matrices already in the $H$ matrix. As an example, if the total number of landmarks comes to 3, the $H$ matrix becomes:

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

The $W$ and $P$ matrices grow by $2m$ rows and $2m$ columns and simply insert the covariance matrix associated with each landmark to the bottom right according to the following:

$$W_k = \begin{bmatrix} W_{k-1} & 0 & 0 \\ 0 & Var(x_m) & Cov(x_m, y_m) \\ 0 & Cov(y_m, x_m) & Var(y_m) \end{bmatrix}$$

$$P_k = \begin{bmatrix} P_{k-1} & 0 & 0 \\ 0 & Var(x_m) & Cov(x_m, y_m) \\ 0 & Cov(y_m, x_m) & Var(y_m) \end{bmatrix}$$

The $V$ matrix simply grows by $2m$ rows of zeros and $2m$ columns of zeros to become:

$$V_k = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

The $X$ matrix grows by $2m$ rows as the observed $x$ and $y$ coordinates of each landmark are multiplied by a rotation matrix to get them in the format of the overall coordinate frame, and added to the state vector. The $X$ matrix becomes:

$$x_k = \begin{bmatrix} x_r \\ y_r \\ \theta_r \\ x_1 \\ y_1 \\ \vdots \\ x_n \\ y_n \end{bmatrix}$$

Finally, the Landmark ID's array must grow by $m$ elements to include all of the landmarks that have been observe to this point.

# 4 Results

## 4.1 VFH

During the first semester, in order to familiarize the team with programming in LabVIEW, and interfacing with the iRobot, a VFH algorithm was written. Since the Kinect was not yet functioning, this program was implemented using the Hokuyo laser range finder. The laser range finder continuously takes scans of the environment in front of the robot and returns distance readings from -120° to 120°. The program written to implement VFH sorted through the data and found all passable regions using a 1000mm threshold. The robot then was then programmed to drive toward the middle of the widest region. At first the algorithm seemed not to work, but it was discovered that the array holding the distance measurements was arranged with the readings from 120 at the front of the array. This meant that the array was essentially arranged with readings from right to left rather than the left to right orientation that was assumed. Therefore, simply inverting the array solved the problem, and the robot was able to perform basic obstacle avoidance. If the goal of this project was to implement robust VFH, much more work would be required. For example, coding would be required to ensure that the robot did not turn around. Also, obstacles at different ranges would be treated differently. However, the goal here was to familiarize ourselves with coding in LabVIEW with our system. We wanted to be able to interface with the sensors and send commands to the iRobot. This goal was accomplished through this task.

## 4.2 Filter Simulation

In order to demonstrate the Kalman Filter theory, a simple experiment was designed. A gaussian noisy signal was generated with a mean of -0.39. The Kalman Filter was applied to the signal by making the state vector a $1 \times 1$ matrix containing only the value of the signal. There was no input to the system so $G = 0$. The current state maps directly to both the next state and the measurement, so $F = H = 1$. Also, the covriance matrices were $1 \times 1$ matrices with very small values, the measurement variance being larger than the process variance. $X$ and $P$ were initialized to zero and one respectively. With these parameters, the result was that the noise of the signal was almost completely filtered out. The simulation can be seen in Figure 13. The filtered output can be seen to approach the actual value as time progresses. This simulation was very successful, and shows the desired behavior of the Kalman Filter that was to be applied to the SLAM robot.

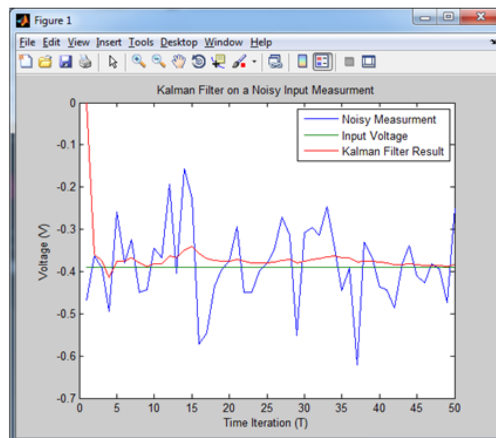


**Figure 13:** Filter

## 4.3 SLAM Simulation

Before the code was written for the actual robot, a simulator was built to show a proof of concept. At first, quite a bit of time was spent writing a complicated simulator that allowed the user to drive the "robot" around its environment using the arrow keys on the keyboard. The angle of the robot could be changed with the side arrows, and the position changed by the up and down arrows. A semicircular region in front of the robot was programmed to show up as green in order to represent the range of the robot's sensors. Error was simulated in both the robot position and landmark positions by adding a random number generated from a Gaussian distribution with zero mean and a small variance to both the $x$ and $y$ coordinates. It was at this point that the problem of conflicting coordinate frames was discovered. As discussed in Section 3.3.1, the robot's coordinate frame is different from the overall coordinate frame. In this initial simulator, a large amount of math was done attempting to convert values from one coordinate frame to the other. This code included a different case depending on which quadrant of the graph the robot was facing. While it seemed to work, the code was very inefficient and certainly must have contained errors that were not noticed in the small amount of testing that was done. When the SLAM portion of this simulator was programmed, there started to be quite a bit of confusion as to which coordinate frame certain points were in, and whether the math was done correctly. Due to this confusion, this simulator was abandoned before completion.

A second attempt at a simulator was much more simplified in order just to demonstrate the SLAM process. In this simulator, the robot position was simply controlled by front panel controls for $x$ and $y$ position. The angle of the robot could not be changed, and therefore, the majority of the conflicting coordinate frame problem was ignored. The error in both the robot and landmark positions was calculated in the same fashion as before. The actual SLAM portion of the code was written by defining a functional global variable for each matrix in the system, and either reading or writing to these variables when necessary. The results of this simulator can be seen in Figure **??**. The robot position was plotted as a point at every time step so as to see the variance in the state estimation. The image on the left is the noisy input of the robot position as it moved around, and

the image on the left is the Kalman estimate of the robot and landmark positions. This experiment met with partial success. The landmark positions moved slightly at first, but quickly converged to positions close to their actual values. However, the robot position remained just as noisy as the input signal. Upon reflection, it seems that this was the case due to the manner in which the error was generated. The error generation for the robot position was not indicative of a real world scenario. This experiment modeled the error as changing the robot's position estimate by a random value at every time step. However, in a real scenario, the error that is occurring due to odometry is much different. There may be a slight offset in the odometry prediction, but it would not be as unpredictable as this simulation. Therefore, the simulation was considered a partial success, and the programming of the actual robot code began.

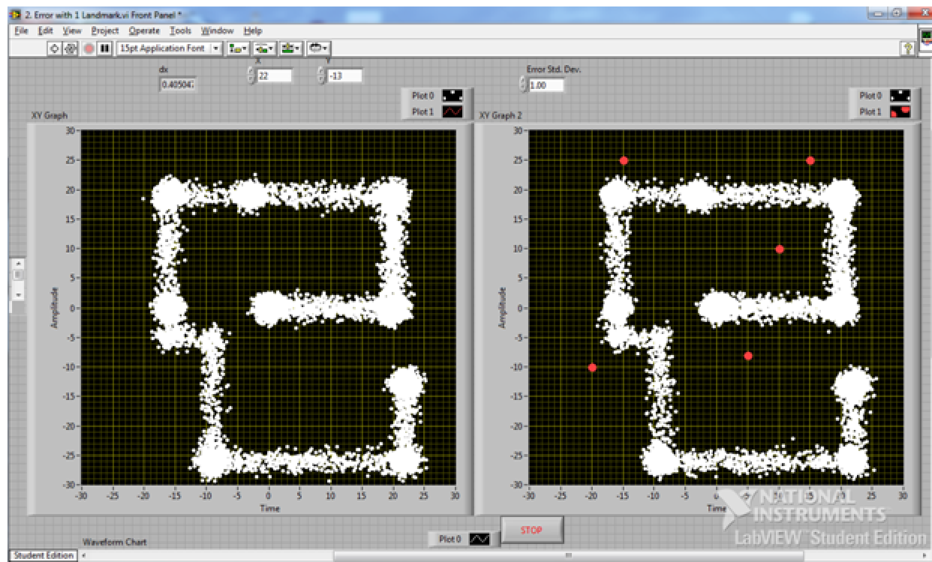


**Figure 14:** Simulator

### 4.4 SLAM Experiments

After programming the simulator, a few lessons were learned. Firstly, the method of using functional global variables for each matrix was not optimal. It made the code bulky, more difficult to understand. Therefore, after consulting [1], the SLAM data type was created as discussed in Section 3.3. This made the code much simpler as a code style was adopted where this data type was simply fed through multiple subvi's in each loop. Theses subvi's manipulated the data stored in the SLAM data type as discussed in Section 3.3. Also, it was theorized that there must be a simpler method than the attempted math in converting between coordinate frames. After some research, the concept of a rotation matrix was discovered in [1]. This made the conversion extremely straight forward. With these new methods, the main program was written for the robot.

#### 4.4.1 Single Landmark

As a initial test, the SLAM robot was tested with only a single landmark. While this worked somewhat, the availability of only one landmark did not allow the robot much of a reference, and the map was not particularly accurate.

### 4.4.2 Three Landmarks

Finally, the SLAM code was executed with three landmarks in the environment. The results can be seen in Figure **??**. This experiment was remarkably successful. When the robot initialized, it found only the green and blue landmarks. It began to navigate with those two, and quickly also found the red landmark. However, the initial estimate of the red landmark's position was not particularly good. However, as the robot moved toward the landmarks, it was able to correct the position of the red landmark to approximately the correct location. The robot also made small corrections to its own location as it moved based on the innovation error it observed from the landmarks. This sensor error is the primary reason why the robot's path is noisy. However, the SLAM algorithm worked to perfection, and clearly demonstrated a proof of concept.

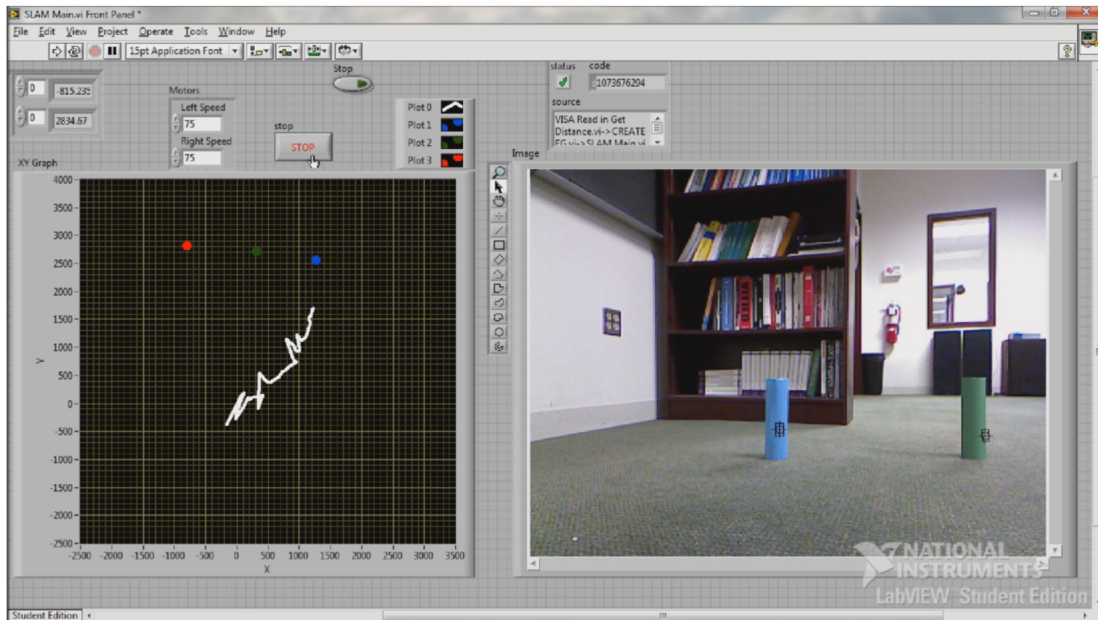


**Figure 15:** SLAM with Three Landmarks

# 5 Conclusion

This project was a remarkable learning experience. Clearly the hardest part was in understanding the theory behind the Kalman Filter, and how it can be applied to SLAM. The process of learning the SLAM theory, and implementing a SLAM robot required the use of knowledge gained from a large variety of Math, Engineering, and Computer Science disciplines. The design process encountered many difficulties, especially in setting up the Kinect drivers and debugging the final code. However, in the end, persistence and research prevailed, and a successful prototype was constructed.

It should also be noted that the Kalman Filter theory was not simply accepted at face value. It was rigorously proved, thus allowing for strong conceptual understanding. Also, the Kalman Filter was applied to SLAM in a somewhat different manner than it had been by former Trinity projects. A similar project was accomplished in 2010 as discussed in [1]. Many concepts were learned and borrowed from this project. However, the program was written slightly differently, and one particular fault in the previous project was addressed. The previous project required that all of the

landmarks be in sight at all time. The methods described in this project allow for landmarks to go unobserved and not disturb the estimates of reobserved landmarks. Nor do the estimates of the reobserved landmarks affect the nonreobserved landmarks.

It was remarkable to watch the final code run on the robot. This algorithm clearly demonstrates a very high level of intelligence. The fact that the robot was able to correct itself after making a bad estimate of a landmark position was extraordinary.

Finally, while this project has greatly influenced us, it is hoped that the work and documentation will allow future students to be able to understand and apply the SLAM algorithm to their own robots.

## References


[1] Wright, Adam A., Nathan M. Swaim, and Orko Momin. "Implementation of a Probabilistic Technique for Robotic Mapping." Thesis. Trinity College, 2010. Print.

[2] Brown, Robert Grover., and Patrick Y. C. Hwang. Introduction to Random Signals and Applied Kalman Filtering. New York: J. Wiley, 1992. Print.

[3] 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.

[4] Dissanayake, G., P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. An experimental and theoretical investigation into simultaneous localisation and map building (SLAM). Lecture Notes in Control and Information Sciences: Experimental Robotics VI, Springer, 2000.

[5] Durrant-Whyte, Hugh, and Tim Bailey. "Simultaneous Localization and Mapping: Part 1."CiteSeerX. The University of Sydney. Web. 19 Dec. 2011. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.135.9810>.

[6] Maybeck, Peter S. "Introduction." Stochastic Models, Estimation and Control. New York: Academic, 1979. Print.

[7] Simon, D. Kalman Filtering, Embedded Systems Programming, vol. 14, no. 6, pp. 7279, June 2001. <www.embedded.com/story/OEG20010529S0118>

[8] Thrun, Sebastian, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics. Cambridge, MA: MIT, 2005. Print.

[9] Welch, Greg, and Gary Bishop. "An Introduction to the Kalman Filter." Department of Computer Science, University of North Carolina at Chapel Hill, 24 July 2006. Web. 11 Nov. 2011. <http://clubs.ens-cachan.fr/krobot/old/data/positionnement/kalman.pdf>.

[10] Weill, Lawrence R., and Paul N. De Land. "The Kalman Filter; An Introduction to the Mathematics of Linear Least Mean Square Recursive Estimation." International Journal of Mathematical Education in Science and Technology 17.3 (1986): 347-66. Print.

[11] Marx, Kevin. "Kinect Sensor for Xbox 360 - Formerly Project Natal - InfoBarrel."InfoBarrel - Crowdsourcing Information — Make Extra Money Writing. InfoBarrel. Web. 19 Dec. 2011. <http://www.infobarrel.com/Kinect_Sensor_for_Xbox_360_-_Formerly_Project_Natal>.

[12] Fisher, Matthew. "Matt's Webcorner - Kinect Sensor Programming." Computer Graphics at Stanford University. Stanford University, 2012. Web. 19 Dec. 2011. <http://graphics.stanford.edu/ mdfisher/Kinect.html>.

[13] Nate. "How The Kinect Senses Depth." Web log post. NonGenre. Blogspot, 11 Dec. 2010. Web. 3 Dec. 2011. <http://nongenre.blogspot.com/2010/12/how-kinect-senses-depth.html>.

[14] "Laser Range Finder Overview." Acroname Robotics. Acroname Robotics. Web. 19 Dec. 2011. <http://www.acroname.com/robotics/info/articles/laser/laser.html>.

[15] Brekelmans, Jasper. "Bloginfo('name'); ?" Rants about Mocap, MoBu, Maya and Coding Brekel. Web. 07 May 2012. <http://www.brekel.com/?page_id=170>.

[16] "User Tracking with LabVIEW and Kinect Based on the OpenNI Interface." National Instruments Developer Zone. National Instruments, 21 July 2011. Web. <https://decibel.ni.com/content/docs/DOC-16978>.