ABSTRACT

Title of Document:	IMPLEMENTATION OF REAL-TIME SIMULTANEOUS LOCALIZATION AND MAPPING WITH PARTICLE FILTER
	Jaymit Pradip Patel, Master of Science, 2012
Directed By:	Professor Christopher C. Davis Department of Electrical and Computer Engineering

The goal of this thesis is to use Particle Filters to Simultaneously Localize a mobile robot in an unknown environment and produce an accurate Map. The theory behind Monte Carlo Localization and Occupancy Grid Maps is introduced and compared with improvements to the Particle Filter such as the Shared Gridmaps and Variance Sampler. A Particle Filter algorithm is developed to use sonar measurements to create occupancy maps, and inertial sensors and wheel encoders to update robot's odometry. The Algorithm is applied to a four-wheel robot in an indoor environment with hallways and is successful in creating detailed maps of the test location and accurate estimate of the robot's state.

IMPLEMENTATION OF REAL-TIME SIMULTANEOUS LOCALIZING AND MAPPING WITH PARTICLE FILTER

By

Jaymit Pradip Patel

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park, in partial fulfillment
of the requirements for the degree of
Master of Science
2012

Dedication

To my parents: for their unconditional love and support.

Acknowledgements

First, I would like to thank Dr. Gilmer Blankenship for being a great mentor to me. He has been very supportive and encouraging throughout my graduate years. I also want to thank John Karvounis, Michael Stanley, and Jared Napora. Their work at the Autonomous Systems Lab has been a key to this thesis. I really appreciate all the help and advice they gave me. The ASL was not the same without them.

Table of Contents

List of Figures

Chapter 1: Introduction

This chapter will introduce basic concepts related to mobile robots, robot localization, and mapping. It will also outline the objective of this thesis.

1.1 Mobile Robots

Mobile Robotics is an active research area in engineering with numerous commercial and military applications. For the purpose of this thesis, a mobile robot is defined as a wheeled platform capable of using its sensors for observations while moving through a 2D indoor environment with its actuators. Thus, the robot's state in this environment is given by its relative location from the start location (origin) and its orientation: (x, y, θ).

A robot uses its actuators to interact with the environment. For example, motor-driven wheels can be used to change its orientation and location simultaneously. There are also numerous sensors available to observe environmental changes. Inertial Sensors provide the robot's acceleration and angular rates using accelerometers and gyroscopes, respectively. Additionally, range data to obstacles in the environment can be retrieved through the use of laser or sonar based range sensors.

1.2 SLAM

Simultaneous Localizing and Mapping (SLAM) is a key component in mobile robotics. To achieve any non-trivial task, a robot must be aware of both its environment and the location of itself in the environment. If the robot has neither a map nor its location, it must estimate them both simultaneously.

When a robot is introduced to a new environment, it can use its sensors to create an estimated map of its immediate surroundings. Feature-based maps contain only the unique features (landmarks) of the environment which can solve the localizing problem. Feature maps are often used when the robot sensors can identify landmarks in the environment regardless of their relative location to the robot. These landmarks are stored as static features in the map to describe the environment. [1] and [2] are examples, where feature-based maps are constructed using vision and sonar range sensors, respectively, to extract natural features from their surrounding such as corners or edges in indoor environments.

In the absence of precise sensors, Occupancy Grid Maps are commonly used to obtain a location based representation of the entire environment. Occupancy maps use a binary random variable to describe the location represented by each cell of the map as either occupied or free [3 p. 285]. This map representation is often used with sonar and laser sensors as seen in [4] and [5]. However, these approaches need to use unique observation models or map representation to ensure an accurate map can be constructed in real-time.

Given a map of its environment, the robot can estimate its global state within the map, or improve its local state estimate. The odometer data provides a useful initial state estimate for the robot; however, the robot continues to accumulate error in its estimate from the odometer noise as it moves through the environment. Feature based maps are often used with Gaussian filters such as the Kalman filter to improve the robot's local state estimate as well as obtain an estimate of the global state. Such filters store the robot and landmark states and continuously improve them with robot model and new measurements [3 p. 204]. Another common approach to localization is the Particle Filter. Particle Filter uses finite number of state estimates (particles) from a posterior distribution, each with an associated likelihood. This approach can be combined with Occupancy maps, to obtain particle state likelihood by comparing local measurement data with the map.

1.3 Objective

The objective for this thesis is to implement an online SLAM strategy to map an unknown environment with a mobile robot in real-time. This algorithm will use Monte Carlo Localization to improve the robot state estimate while simultaneously developing a Grid map estimation of the environment. Furthermore, the robot will perform this algorithm "online". In theory, this means the algorithm will only use current and past measurement data to obtain the current robot state and map. This also means that in practice, the SLAM algorithm will update the robot state and map using the current data before processing new sensor measurements.

A four-wheeled robotic platform is used to passively explore an indoor environment with hallways; specifically, the physical tests were conducted in one of the wings of A. V. Williams Building at University of Maryland. Wheel encoders and inertial sensors are used for base odometer data, while sonar range sensors are used to generate accurate maps.

The challenge in this thesis lies in producing accurate occupancy maps in an environment with mostly long corridors that look alike. An added difficulty is the use of noisy gyroscope and sonar sensors.

Chapter 2: SLAM Theory

This Chapter focuses on the main concepts involved in performing SLAM for mobile robots. It introduces the basic theory for Monte Carlo Localization and Occupancy Grid Maps. Since this thesis is focused on wheeled robots and sonar sensors, example odometry and sensor models will be shown.

2.1 Monte Carlo Localization

```
1:   Algorithm MCL($X_{t-1}, u_t, z_t, m$):
2:       $\bar{X}_t = X_t = \emptyset$
3:       for $n = 1$ to $N$ do
4:           $x_t^{[n]}$ = sample_motion_model($u_t$, $x_{t-1}^{[n]}$)
5:           $w_t^{[n]}$ = measurement_model($z_t, x_t^{[n]}, m$)
6:           $\bar{X}_t = \bar{X}_t + \langle x_t^{[n]}, w_t^{[n]} \rangle$
7:       endfor
8:       for $n = 1$ to $N$ do
9:           draw $i$ with probability $\propto w_t^{[i]}$
10:          add $x_t^{[i]}$ to $X_t$
11:      endfor
12:      return $X_t$
```

Figure 1- Monte Carlo Localization [3 p. 252]

Unlike Gaussian Filters, Particle Filters can directly process raw sensor values. This is especially useful when using beam sensors such as sonar range sensors, since the large noise of the sonar measurements make it difficult extract features from the raw data. Additionally, range sensors can provide a lot of negative information, such as the empty region between the sensor and the detected obstacle, which may be ignored in feature-based algorithms. Also, particle filters can be applied to global localization in addition to local position tracking, for which Gaussian Filters are less applicable [3 p. 233].

Monte Carlo Localization (MCL) is a mobile robotic approach to Particle Filter Localization. Its simplicity and wide range of application has made it a popular algorithm in robot localization since its introduction [6] . Figure 1 shows the general algorithm for MCL. MCL is an extension of Particle Filter algorithm, which itself is based on the Bayes Filter. This filter is used recursively to calculate the current state estimate by using the estimate from the previous time step. The algorithm is applied to the robot state at each time step recursively. Its input are the previous set of particles (X_{t-1}), the new control data (u_t), the new sensor measurements (z_t) and a shared map of the environment (m). The algorithm can be separated into two major parts: particle update and particle resampling. Lines 3-7 show the particle update stage. The odometer model is sampled for the new particle state estimate ($x_t^{[n]}$) using the control data and previous particle state estimate ($x_{t-1}^{[n]}$), as shown in Line 4. The next line calculates the importance weight of each particle using the measurement model. This model provides the probability of the input sensor measurement given the particle's state and the shared map. With occupancy grid maps, measurement models often use either ray casting or map matching to determine the likelihood of the sensor measurement. Ray casting is used with range sensors such as sonar or laser sensors to compute the true distance of an obstacle given a sensor measurement and using the sensor's pre-computed probability distribution to find the associated likelihood of the sensor measurement [3 p. 158]. In map matching, the sensor measurement distribution is not needed. Instead, a local map containing only the recent measurement data is used to compare to a global map that contains the rest of the past measurements. This comparison results in a weight that is higher if the two maps are more similar to each other. Thus, during loop closing, when a particle revisits a known area, an accurate state estimate will generate a high weight value and will be more likely to survive during the resampling process (Lines 8-11). In resampling, new particles are randomly drawn from the temporary particle set ($\bar{X}_t$) using the particle weights as the discrete probability distribution of the particles. This new set of particles (X_t) is retained to use with the next iteration of the algorithm.

The particle filter SLAM approach differs slightly from the MCL. Instead of using a common map amongst all of the particles, each particle maintains its own unique map. Furthermore, each particle updates its map estimate as well as its state estimate at each time step.

2.2 Odometry Model

The sample_motion_model shown in Figure 1 adopts a probabilistic motion model to advance the robot state estimate by integrating the control data and error into previous state estimate. Either velocity commands to the controller or robot's odometer data can be used to model the robot's motion. Velocity based models can provide the control data as the motion is carried out; therefore, it can be used to predict robot motion. In contrast, odometer based models are more accurate since they use the sensed motion provided by the odometer measurements after the robot has already executed the controls [3 p. 132].

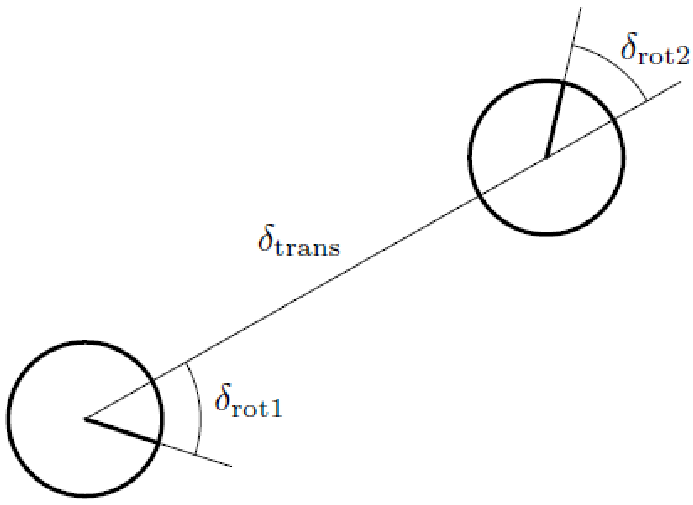


Figure 2- Odometry Model [3 p. 133]

Figure 2 shows the robot motion according to an odometry model. The model represents a robot motion as an initial rotation by δ_{rot1}, followed by translation of δ_{trans}, and a final rotation by δ_{rot2}. Such an odometry model is used to create a sampling algorithm which can be used in the particle update stage of MCL. Figure 3 shows the algorithm used to sample this motion model given the odometer measurements. To be consistent with the MCL Algorithm, the inputs for this algorithm are still control data (u_t) and previous state estimate (x_{t-1}); however, the current and previous odometer measurements ($\bar{x}_t$ and $\bar{x}_{t-1}$) are used instead of actual controls. Lines 2-4 calculate the first rotation from previous state (δ_{rot1}), the translation (δ_{trans}), and the final rotation (δ_{rot2}) according to the Odometer model shown in Figure 2. The next three lines subtract the normally distributed noise, assumed to be present in the odometry readings, to produce the correct rotations and translation values ($\hat{\delta}_{rot1}$, $\hat{\delta}_{trans}$, and $\hat{\delta}_{rot2}$). The **sample** method used in this algorithm samples a random value from a normal distribution with variance described by its input. The variables α_1, α_2, α_3, and α_4 are the noise parameters specific to the robot's odometry measurements. The variance to mean ratio of odometer error is a good start for these parameters; however it is ideal to overestimate these errors to improve the performance of the Particle Filter [3 p. 112]. Finally, the new estimated state (x_t) is calculated by applying the odometer motion model to the error-corrected rotation and translation values.

```
1:  **Algorithm sample_motion_model_odometry($u_t, x_{t-1}$):**
2:      $\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
3:      $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
4:      $\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$
5:      $\hat{\delta}_{rot1} = \delta_{rot1} - \textbf{sample}(\alpha_1|\delta_{rot1}| + \alpha_2|\delta_{trans}|)$
6:      $\hat{\delta}_{trans} = \delta_{trans} - \textbf{sample}(\alpha_3|\delta_{trans}| + \alpha_4(|\delta_{rot1}| + |\delta_{rot2}|))$
7:      $\hat{\delta}_{rot2} = \delta_{rot2} - \textbf{sample}(\alpha_1|\delta_{rot2}| + \alpha_2|\delta_{trans}|)$
8:      $x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1})$
9:      $y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1})$
10:     $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$
11:     return $x_t = (x', y', \theta')^T$
```

Figure 3 - Sampling Odometer Motion Model [3 p. 136]

2.3 Occupancy Grids

Grid Maps represent the robot's environment as a set of uniformly spaced discrete locations (cells) instead of continuous space. In occupancy grid mapping, each of the cells has an associated binary random variable, which describes whether that location contains an obstacle. An occupancy grid algorithm uses an inverse sensor model to update the probability of each cell being occupied (occupancy value) for each new sensor measurement, given an accurate state estimate. An inverse sensor model algorithm is shown in Figure 4 for range sensors with cone shaped sensor beam.

```
1:  **Algorithm inverse_range_sensor_model($i, x_t, z_t$):**
2:      Let $x_i, y_i$ be the center-of-mass of $m_i$
3:      $r = \sqrt{(x_i - x)^2 + (y_i - y)^2}$
4:      $\phi = \text{atan2}(y_i - y, x_i - x) - \theta$
5:      $k = \text{argmin}_j \left|\phi - \theta_{j,sens}\right|$
6:      if $r > \min\left(z_{max}, z_t^k + \frac{\alpha}{2}\right)$ or $\left|\phi - \theta_{k,sens}\right| > \frac{\beta}{2}$ then
7:          return $l_0$
8:      if $z_t^k < z_{max}$ and $|r - z_{max}| < \frac{\alpha}{2}$
9:          return $l_{occ}$
10:     if $r \leq z_t^k$
11:         return $l_{free}$
12:     endif
```

Figure 4- Inverse Range Sensor Model [3 p. 288]

This algorithm calculates the distance (r) and orientation (ϕ) from the center of each cell (x_i, y_i) in the map to the sensor location (x, y). If the current cell (i) is outside of the sensor beam width (β) of its closest sensor (k), or it is farther than the sensor's max range (z_{max}) or the closet sensor's detected range (z_t^k); then the cell occupancy value is not updated. Otherwise, the occupancy value is increased by l_{occ} if the cell's distance is within $\alpha/2$ of z_t^k or by l_{free} if it is not. Here, l_{occ} and l_{free} are the log-likelihood values instead of probabilities. A log-likelihood is calculated by taking the logarithm of the probability of an event divided by the probability of its negate. This representation simplifies the occupancy update step, since the Bayesian rule is not needed to update the probability of a cell being occupied; instead, each cell adds the return value from inverse sensor model to its previous log-likelihood value [3 p. 94].

Chapter 3: Particle Filter Improvements

This chapter will discuss the specific algorithms used in this thesis to improve the Particle Filter SLAM described in the previous chapter. Improvements to Monte Carlo Localization and Occupancy Grid Maps are introduced to meet the objective of performing an online SLAM in real-time.

3.1 Improved Occupancy Grid Maps

The occupancy grid maps represent the robot environment as a field of binary random variables with an associated probability of occupancy. The SLAM algorithm described in Chapter 2 uses this map representation for each particle, which has a memory cost in proportion to $n * m$, where n is the number of particles and m is the number of grid cells used in a single map. A larger particle set can avoid issues such as particle depletion, where particles with accurate state estimates are removed during resampling [3 p. 113]. Additionally, the number of grid cells is directly related to the level of detail of the map, a finer resolution grid map requires more cells. As such, there is a tradeoff between the memory cost of the Particle Filter and its accuracy. However, a reduction in the memory cost of each map will allow for the same grid map resolution with a larger number of particles. One such approach creates Shared Gridmaps [7] to be used with a collection of particles, instead of each particle having its own unique map.

A normal occupancy map maintains a unique map spanning the entire observed environment. This map must be either initialized to a large grid of the maximum possible environment size or must be constantly grown whenever the robot reaches the edge of the current map. In shared gridmaps, each particle contains a set of separate grid patches. Each of these patches is similar to the occupancy grid map, but it is of fixed size and is centered at a different location. For instance, with a grid cell resolution of 0.1 meter per cell, a patch with a size of 5 meter would contain 2500 cells with 50 columns and 50 rows. Each particle map can be initialized with a single patch and additional patches are created when the robot location is outside of all of the current patches.

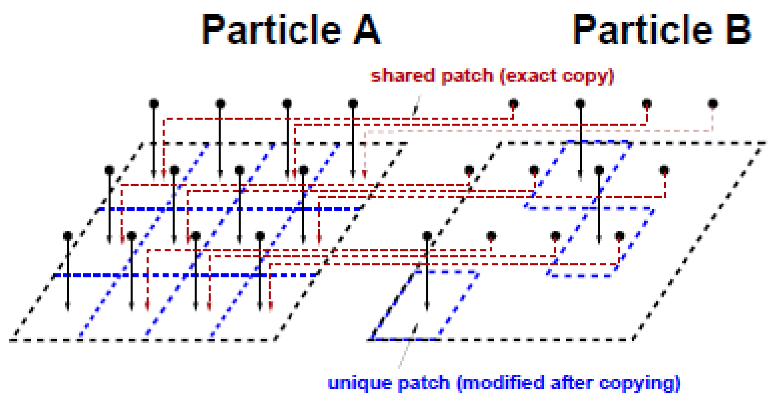

Particle A
Particle B
shared patch (exact copy)
unique patch (modified after copying)

Figure 5 - Shared Gridmaps Representation [7]

During the resampling process of Particle Filter SLAM, the maps of particles with high weights are copied and given to new particles, while the particles with low weights are removed. So, between two resampling stages, all particles copied from the same particle from the previous step will largely contain the same maps, with the area of the maps where the particles are currently located having any differences. Thus, these similarities between particle maps are redundant. Shared Gridmaps reduce this redundancy by only providing the reference to the original map during the resampling process instead of copying the entire map. When each particle wants to update its map, it can copy the individual patch to its own map and modify it, as seen in Figure 5. In comparison to the original occupancy grids, which keeps a unique map in memory for each particle, the Shared Gridmaps approach is much more memory efficient [7]. This approach effectively reduces the number of maps in memory during each loop closing and also reduces the algorithm time since all particle maps are not being recopied after every resampling.

3.2 Subsampling Observations

The original Monte Carlo Localization is too complex in practice to achieve the goal of this thesis: to use Particle Filter SLAM in real time and obtain accurate state and map estimate. However, this algorithm can be modified in several ways to reduce its computational cost [3 p. 244]. First, subsampling the sensor observation spatially and in time can speed up the map update algorithm. Instead of updating each particle's map at each time-step, the update can be delayed until the robot has moved on for a sufficient distance from the previous location or if a pre-determined time period has passed since the last update. This delay can be justified if the robot motion for each time step is less than the resolution of a grid cell. Similarly, motion updates can also be delayed by geometrically integrating the odometry readings over short time periods.

3.3 Particle Weight Calculation

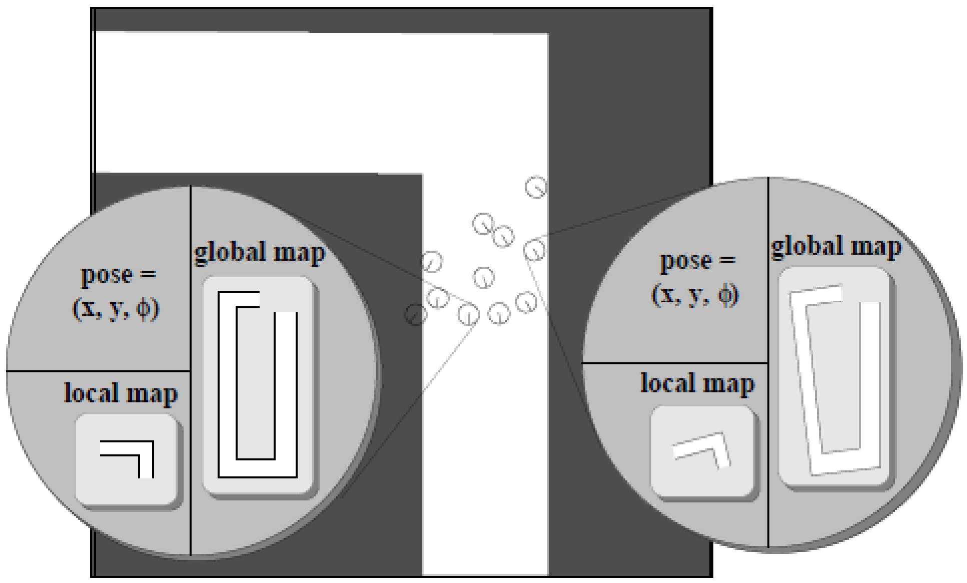


Figure 6 - Local Map Representation [7]

If the map updates are delayed until the robot has moved on for a certain distance, the weight calculation should also be delayed, since the measurement model will not produce a new weight until the map has been updated. It may also be necessary to delay weight calculations until the map has been updated multiple times. Noisy sonar sensors, in particular, would require several map updates to form reliable occupancy maps. Furthermore, the map and weight delay is advantageous when using a map matching algorithm, such as that used in [7], to calculate a particle's weight value. In that approach, each particle contains a local and a global map along with its state estimate (see Figure 6). The global map is not updated at each time step and only the most recent sensor observations are used to update the particle's local map instead. During weight calculation, the local map is compared to the global map. The weight function calculates a total match value, which is increased if a cell in global and local map is occupied (occupancy value is above an "occupied" threshold) or decreased if a cell is occupied in local map but free (occupancy value is below “free” threshold) in the global map. An exponential function is applied to the match value to obtain the actual weight for the particle. Since both of these maps are built from different observations, a high match value indicates an accurate localization in the global map. This method is similarly used in [8] to correct odometry error during localization. For resampling, each particle’s weight can be divided by the total weight to obtain an accurate discrete probability distribution with a sum of 1.

After a weight calculation, the local map information can be processed into the global map. Until this step, the global map has no knowledge of the robot’s current environment since the map updates have been performed only on the local map. A simple algorithm can store the most recent sensor measurement and odometry pairs in a delayed queue until they can be processed by the global map. However, if the occupancy maps store the log-likelihood value instead of the occupancy probability, the local map can be used directly to update the global map at once. Just as in the Inverse Range Sensor Model, each local map’s grid cell’s log-likelihood value is added to the corresponding global grid cell value. Afterwards, the current local map is discarded before processing new map updates.

3.4 Particle Resampling

As with the map and weight updates, particle resampling can also be delayed to achieve reduction in computation time. Specifically, particle resampling should be done at most with the same frequency as weight updates. Additionally, resampling is unnecessary when the robot is in an unexplored location. If the particle’s local map is created from an unexplored area of the environment, the corresponding global map will be unknown. So, the map-matching algorithm, described above, will return a weight of 1. If all or most of the particles have the same weight, the resampling process of the MCL, may replace some of the "accurate" particles unnecessarily since new particles would have equal probability of being drawn from any of the old particles. On the other hand, if the robot is returning to a known location (loop closing), the particles' weights will vary greatly. Those particles with accurate state and map estimates will have a higher weight than those that have many conflicts between their local and global maps. So, the measured variance of the particle weights will be close to zero when the robot is exploring unknown area and so resampling should not be performed. While a high variance during loop closing will indicate that the particles should be resampled [3 p. 109]. This approach further decreases computational complexity since particle resampling will be less frequent.

```
1:     Algorithm Low_variance_sampler($X_t, W_t$):
2:         $\bar{X}_t = \emptyset$
3:         $r = \text{rand}(0: M^{-1})$
4:         $c = w_t^{[1]}$
5:         $i = 1$
6:         for $m = 1$ to $M$ do
7:             $u = r + (m - 1) \cdot M^{-1}$
8:             while $u > c$
9:                 $i = i + 1$
10:                $c = c + w_t^{[i]}$
11:            endwhile
12:            add $x_t^{[i]}$ to $\bar{X}_t$
13:        endfor
14:        return $\bar{X}_t$
```

Figure 7- Low Variance Sampler [3 p. 110]

Another way to reduce the computational cost of resampling is to use the Low Variance Sampler shown in Figure 7. An efficient weighted random sampling algorithm has a complexity of $O(n\,log\,n)$, since n iterations of binary search ($O(\log n)$) over a sorted particle weight set is required to sample n new particles. In comparison, low variance sampler is more efficient with complexity of $O(n)$. The sampler uses only one random number and selects a particle with probability in proportion to a particle's weight. The u calculated in line 7 lies in $[0,1]$ and points to the first particle in the weight list for which the cumulative sum (c) up to that particle is greater than u.

Chapter 4: Hardware and Software

This chapter describes the robot hardware used in the physical experiments, including the specific sensors used for SLAM. It also discusses the software and algorithms used for the experiments.

4.1 Robot Platform

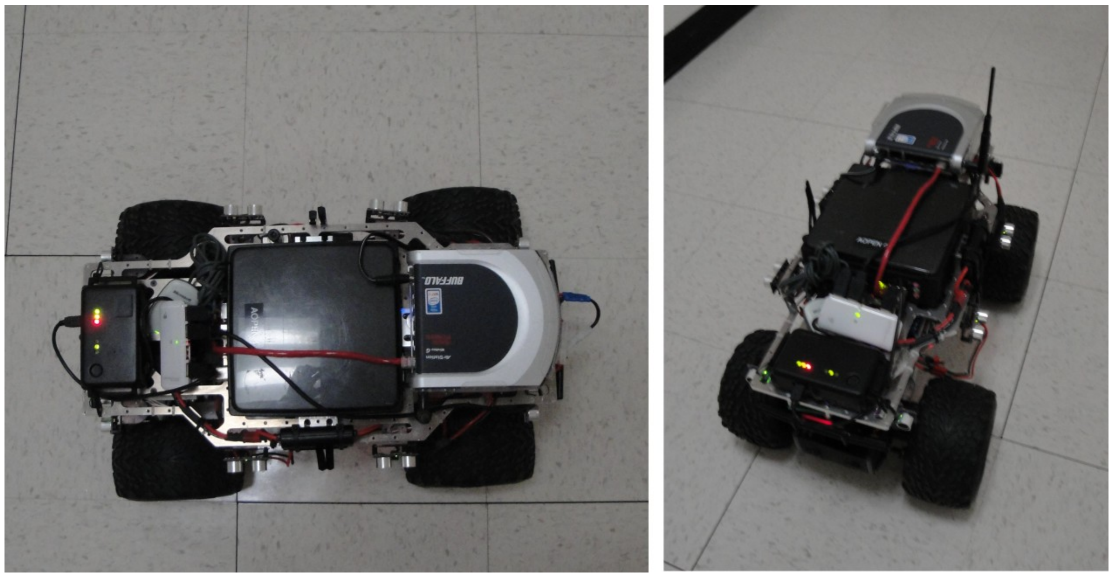


Figure 8 - Tamiya Robot (Left: Top-Down View with Robot Facing Left, Right: Robot facing towards the bottom)

A four-wheeled mobile robot with various sensors was used for the experiments in this thesis (see Figure 8). This platform was constructed by the graduate students of Autonomous System Laboratory (ASL) [9][10][11]. The base chassis of the platform is the TXT-1, Tamiya Remote Control Truck. The TXT-1 is actuated through two servos for front and rear steering as well as a motor controller for speed control. An 8.4V Ni-MH Battery powers the steering servos and the motor controller. The original RC receiver was replaced with a Parallax Servo Controller, which is connected to the onboard computer via USB. An Xbox 360 Wireless Receiver is connected to the computer to retain the ability to manually control the robot with a remote controller.

The stock cover for the robot was replaced with a custom built aluminum platform [11]. The platform is used to hold the onboard computer and router as well as to mount the necessary sensors to the robot. The AOpen miniPC, housing an Intel Core 2 Duo 2.0 GHz processor and 2.0 GB RAM, process all sensor data and controls robot actuation. A Buffalo WHR-HP-G54 Wireless-G Router is used to control the robot from a base computer and can also be used to communicate with other robots. Both the computer and the router are powered through multiple 11.1V Li-Ion Batteries via a DC-to-DC converter.

4.2 TRX INU

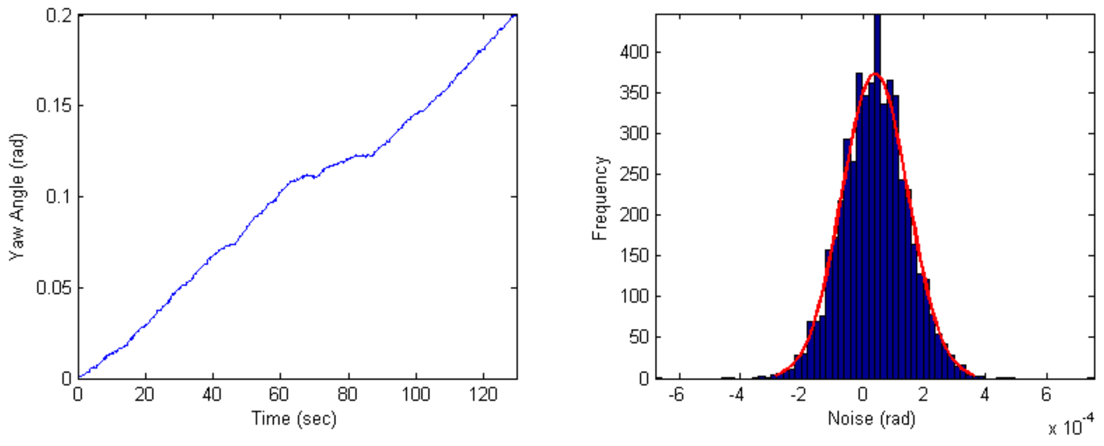


Figure 9- TRX INU Noise

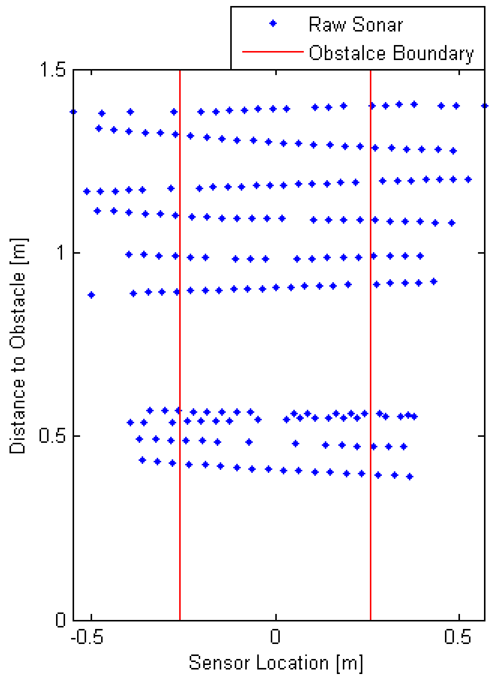
A TRX Systems Inertial Navigation Unit (INU) is mounted in front of the robot and provides the robot's heading. It also is connected to the computer via USB. The TRX INU is a custom-built device that provides calibrated orientation data using a proprietary filter to reduce drift that results when using MEMS Inertial

Measurement Units (IMU). An IMU provides acceleration and angular rates in the sensor frame using accelerometers, and gyroscopes. The angular rates are integrated to get an estimation of the robot's roll, pitch, and yaw angles relative to the sensor's initial orientation. Since the INU is mounted to the robot and the robot only moves in the 2D plane parallel to the floor, only the yaw angle is used in this thesis to obtain a general heading of the robot's motion. However, the yaw angle obtained from the calibrated INU orientation data is obtained from a noisy gyroscope measurement. This noise produces a drift in the orientation when the angular rate is integrated, even if the robot is motionless. Figure 9 shows the drift and the noise in the yaw orientation angle while the robot is kept still for couple minutes. The left plot is the actual yaw orientation provided by the INU. The right plot shows a histogram of the change in the orientation angle with a Gaussian fit over the data. Although the noise appears to be Gaussian, it fails the Kolmogorov-Smirnov Test. However, it is a good approximation for the noise model [10].

4.3 Wheel Encoder

The TRX INU provides robot's acceleration data along with orientation. However, in general, acceleration data gives poor results for the robot's location. The noise present in the acceleration data is integrated twice when using acceleration to retrieve the distance travelled by the robot. This process produces unreliable location estimates. Therefore, this thesis uses encoder measurements along with INU orientation for odometry calculations. Four Digital Optical Encoders mounted to each wheel shaft are used to measure the distance travelled by each wheel [11]. The encoder measures the wheel rotation rate by emitting light onto a coded reflective wheel and measuring the number of reflections in an interval. This encoder is a Quadrature Encoder, which uses two out of phase tracks on the wheel, to obtain the direction of wheel rotation along with the rate. The encoders were calibrated by moving the robot in a straight line using line following [10]. Using the total distance travelled by the robot and the number of wheel rotations, a parameter value for each wheel encoder was obtained. The encoders were found to contain Gaussian noise with a standard deviation to mean ratio of 0.02 [10]. A USB data collection board, designed at the ASL, is used to collect the measurements from the microcontroller of each encoder and transfer to the computer. A second such sensor board was used to collect the sonar measurements.

4.4 Sonar Sensor



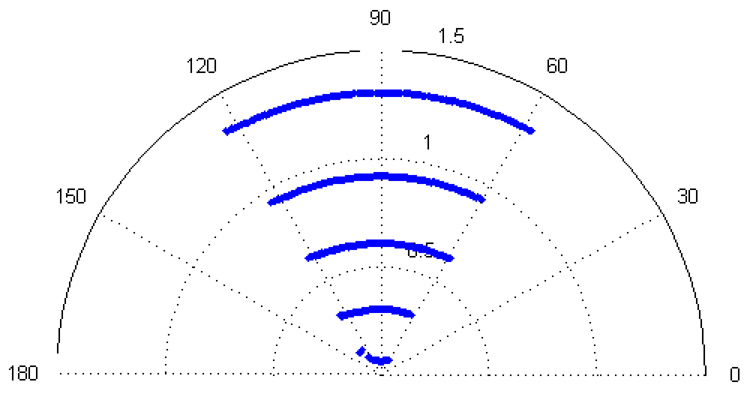


Figure 10 - Raw Sonar Data

The robot uses eight Parallax Ping))) Ultrasonic Distance Sensors. Two sensors are mounted to each side of the aluminum platform. However, the rear sensors are not used during data collection since the operator is constantly behind the robot. These sensors transmit an ultrasonic (40 kHz) burst and record the time until an echo is returned, which is used to calculate the distance to the detected object [12]. The sensor has a specified range of 3 meters and has 90 degree field of view. Figure

10 shows sonar data collected with a single Ping sensor. The bottom image is a polar plot where each point represents a single sensor value (distance) and the corresponding angle (theta) of the sensor when the sensor detects an obstacle. The data was collected with a servo-mounted sensor placed in front of a wall while the servo is rotated from 45 to 135 degrees. Unlike the specifications, the sonar sensor has an approximate 60 degree field of view. The top image shows the plot of sonar range values vs. the sensor location relative to an obstacle. Here, the robot was driven several times at different ranges in parallel to a flat obstacle with width of 0.52 meter. The vertical red lines show the edges of the obstacle. This sonar sensor detects the 0.52 meter wide obstacle for approximately 0.9 meter, at almost all ranges. This shows that the sonar sensor has a rectangular model with width of 0.4 meter instead of a cone-shaped model when encountering parallel obstacles like walls in an indoor environment. Also, the raw sonar values had a variance to mean ratio of less than 0.00015 for distances greater than 0.2 meter. Figure 11 shows the rectangular sonar model overlaid on the test result of rectangular obstacle in parallel to the sensor, provided by the sensor's data sheet [12].

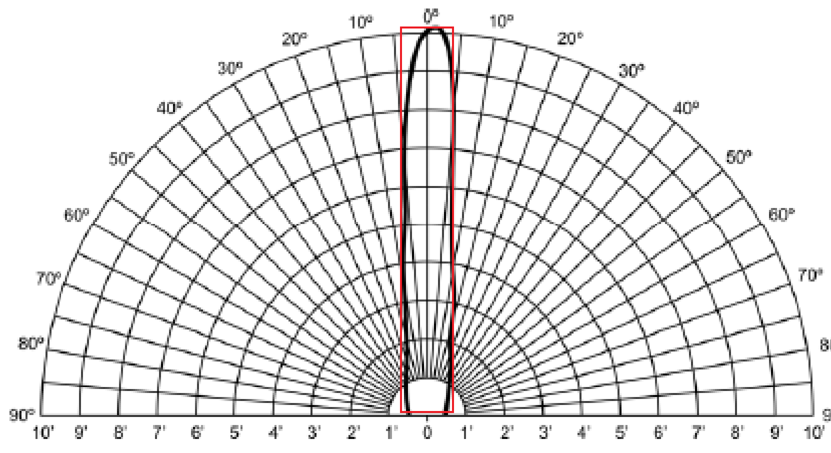


Figure 11 - Sonar Model [12]

4.5 Software

This thesis primarily used C# programming language. The ASL Software Framework developed by the Autonomous Systems Lab was used on the robot's Windows XP Operating System to run the robot and collect data for the Particle Filter.

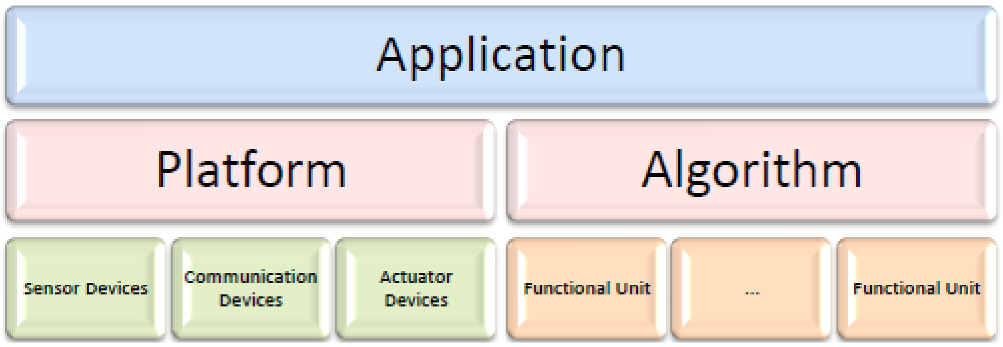


Figure 12- ASL Framework [10]

The ASL Framework (shown in Figure 12) uses C# events to asynchronously distribute data between different parts of the software. Any application developed in this framework is divided into a Platform and an Algorithm. The Platform forms a group of all the Devices available on the robot, such as sensors and actuators. These devices contain code to interface with the actual hardware to either collect sensor data or send actuation data. The Platform controls the flow of sensor data and actuations to or from the Algorithm. The Algorithm is the counter to the Platform. It contains smaller units of code called Functional Units and controls data to and from the Platform as well as among the Functional Units. Each Functional Unit processes data from sensors or other Functional Units to perform a task and can also send out actuator commands to the Platform. Since the framework uses asynchronous events, the Functional Units can process data from multiple sensors in parallel, while overflow data not processed in time is discarded to keep the Algorithm running in real-time. This framework also provides hardware abstraction inherently. The algorithm and platform are unaware of each other, and simply process the incoming sensor and actuator data, respectively, and output the actuator and sensor data. Thus, any algorithm and platform developed in this framework can be replaced in-between application runs. Additionally, the ASL Library already contains many Devices and Functional Units that are used for this thesis.

4.6 Tamiya Platform

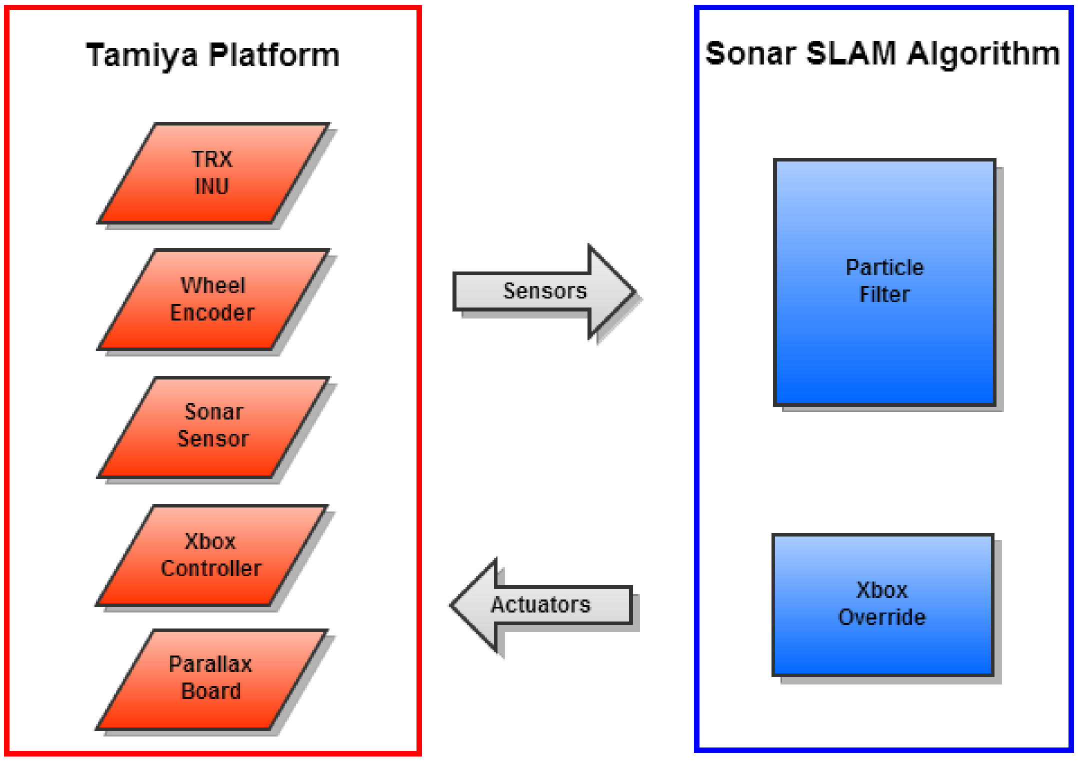


Figure 13- SLAM Application

Figure 13 shows the flow of sensors and actuators between the Tamiya Platform and the SLAM Algorithm. The Tamiya Platform contains devices previously developed by the Autonomous Systems Lab. It includes the three sensor devices: TRX INU, Wheel Encoder, and Sonar Sensor. These devices communicate through USB serial ports to their respective hardware described in the previous section and provide the orientation, encoder, and sonar measurements, respectively, to the algorithm. Additionally an Xbox Controller device is used to collect user commands to remotely control the robot. This device uses Microsoft XNA Studio library to connect to an Xbox controller. Finally, the Tamiya Platform accepts steering and motor actuation commands from the algorithm and sends them to the Parallax Servo Controller device. This device separates the acutation commands and sends them to the appropriate channels on the Parallax board to control robot steering servos and motor controller.

The Tamiya Platform is interchangeable with the Virtual Platform. This platform simulates a robot by replaying sensor data from a previously recorded sequence. The Virtual Platform is useful for fast algorithm development and running multiple tests on the same data.

4.7 Sonar SLAM Algorithm

The Sonar SLAM Algorithm contains two Functional Units. The Xbox Override Functional Unit is used to read the Xbox controller sensors from the Platform and converts them to appropriate steering and motor commands to send back to the Parallax Device. Figure 14 shows the detailed flow of the Particle Filter Functional Unit. This Functional Unit maintains a set of particle objects and performs the update and resample stages of the Particle Filter as described in Chapter 3. Each Particle object itself contains a state estimate of the robot, and local and global occupancy maps. The sensor inputs to the Particle Filter are buffered using a single queue. The Particle Filter is busiest during Map Updates and Resampling; however, at other times, this functional unit is simply recording odometry measurements. Bufffering the inputs allows the filter to distribute the time spent on the long subunits evently throughout its run. This increases the number of particles that the Particle Filter can process without falling behind to the sensor inputs.

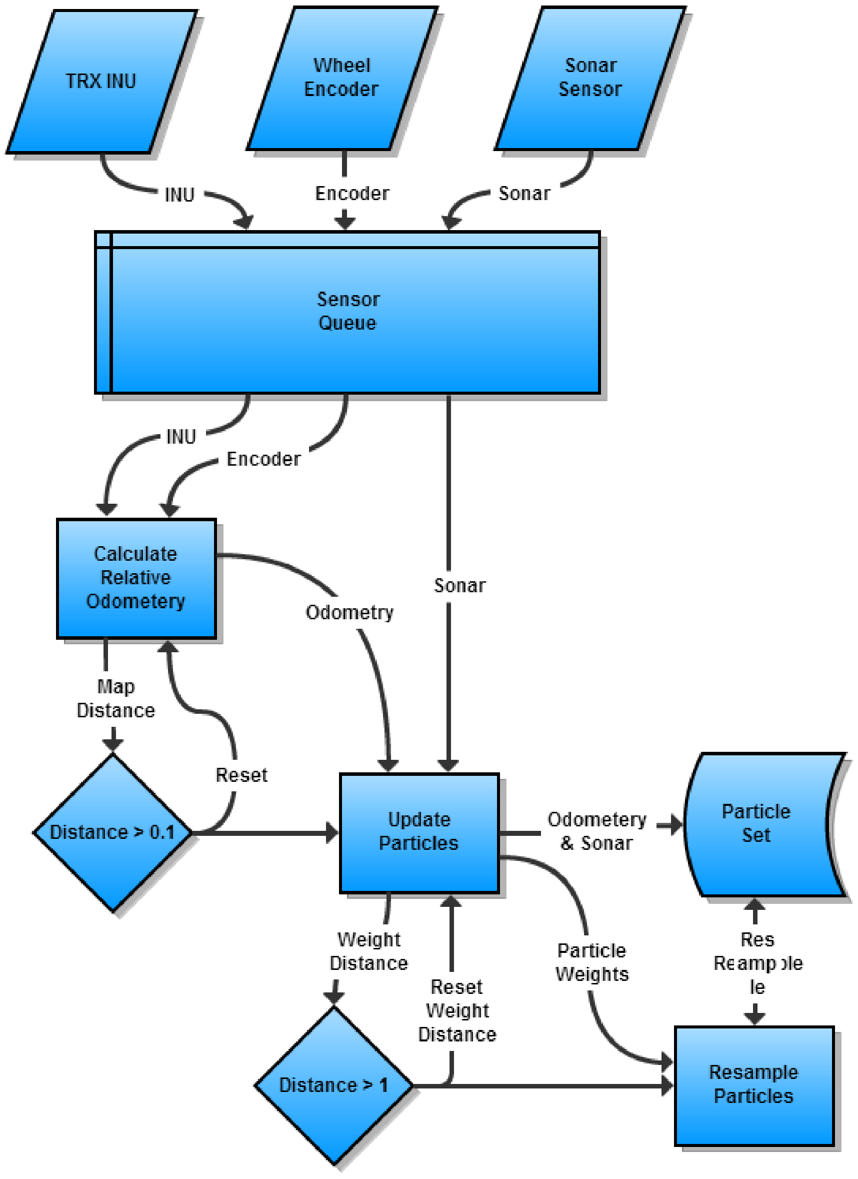


Figure 14- Particle Filter Functional Unit

The INU orientation and encoder measurments are used to calculate the odometry data relative to the last odometry update. Since the INU provides the absolute orientation from the initial heading, only the last INU measurement is kept while waiting for the odometry update step. However, the Wheel Encoders provide the relative distance travelled by the robot, so the encoder measurements are integrated until the next odometry update. The Odometry Motion Model discussed in Section 2.2 was customized to use the wheel encoders and the INU data. The difference in orientation from the INU is divided in half to obtain the initial rotation (δ_{rot1}) and the final rotation (δ_{rot2}). This maintains the assumption that the robot has traveled at a direction between its current and previous orientations. The integrated encoder distance is set as the translation (δ_{trans}). The cross α parameters (α_2 and α_4) are not used since the rotation and translation are obtained from independent sensors; while α_1 and α_3 are set to $2e^{-4}$ and $4e^{-4}$, respectively.

When a new sonar measurement arrives, the particle filter performs the update step if the robot has moved on for 0.1 meter since the last map update. Each particle updates its own odometry model with the integrated encoder distance and the change in robot heading. Then, the last sonar measurment is used to update each particles' local occupancy grid map at their current location. The map update function uses a modified inverse range sensor model with a rectangular sonar model shown in section 4.4. The beam opening parameter (β) is set to 0.4 meter and range error parameter (α) is set to the map's grid cell size. If the particle is sharing its map, it makes its own copy before updating that shared patch.

During the map update, a particle also calculates a weight value if the robot has moved on for 1 meter since the last weight calculation. Each particle compares the current local map to the global map to obtain the weight value; then it queues the current local map to be integrated into the global map. Afterwards, the global map integrates the first map in the queue. For this thesis, a local map queue of size 1 was sufficient to delay the update of the global map. This delay is necessary to produce a unique local map which contains no similarities to the global map while the robot is in an unexplored area.

During resampling, a variance value of the particles' weights is calculated to verify if resampling should proceed. In experiments, a variance value above 0.05 generally indicated that the robot was in a previously explored region. The particles are resampled using the low variance resampler discussed in the previous chapter. As mentioned in that chapter, the new particles are given only the reference to the original occupancy maps.

A Data Collection Algorithm was used to collect and record robot sensor data for future testing using the Virutal Platform. For this purpose, it contained a Message Recorder Functional Unit, which converts any sensor or data object to binary data and records it and its associated time of arrival to the functional unit. The Virtual Platform contains a counter Message Reader Device, which converts the binary data back to sensor or data object.

Chapter 5: Results

This Chapter shows the results of the SLAM experiments conducted. Two exploration patterns are discussed and compared. It also discusses the effect of various Particle Filter parameters on the state error and map quality.

5.1 Experiments

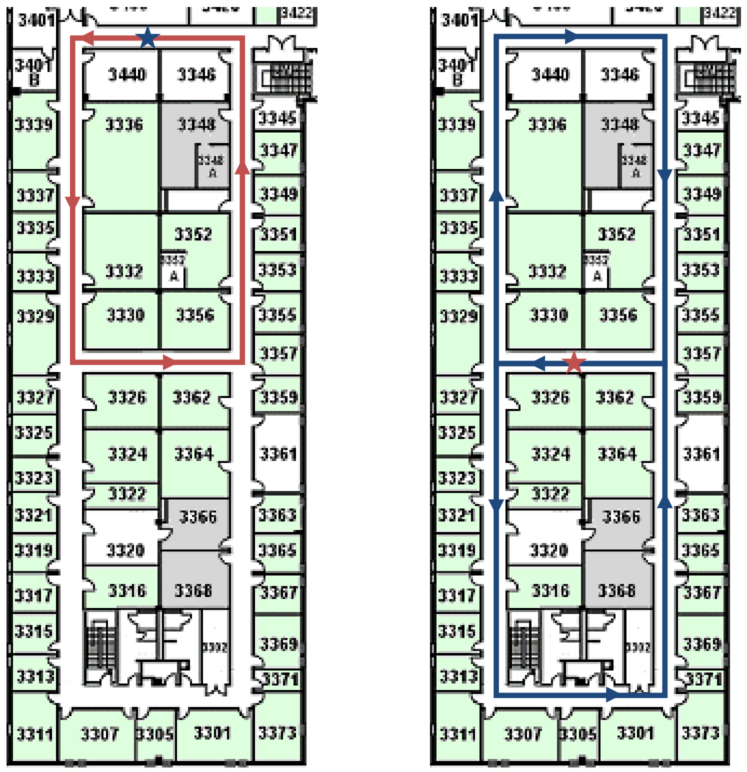


Figure 15- A.V. Williams Wing (3rd Floor) Floor Plan with Robot Path Overlaid (Left: Double Loop, Right: Infinity Loop) [13]

To test the Particle Filter, the robot was driven through the 3rd Floor wing of A.V. Williams Building at the University of Maryland. The first data set (Double Loop) was collected by running the robot counter-clockwise around the top half of the wing. Figure 15, on the left, shows the approximate robot path (the star marks the starting location of the robot) imposed on the floor plan of the test location. On the right is the approximate path of the second data set, during which the robot first makes a single counter-clockwise loop followed by a clockwise loop (Infinity Loop). The robot was driven at approximately 0.54 m/s for 87 meters and 0.5 m/s for 88 meters for the Double and Infinity Loops, respectively. Figure 16 shows the maps generated for both Loops using only the raw odometry and sonar information. The gray intensity of the map indicates the occupancy value. For example, the black pixels represent high probability of that location being occupied, while the white pixels represent unoccupied cells of the map. Without the Particle Filter, the maps are severely misaligned when the robot returns to a known location.

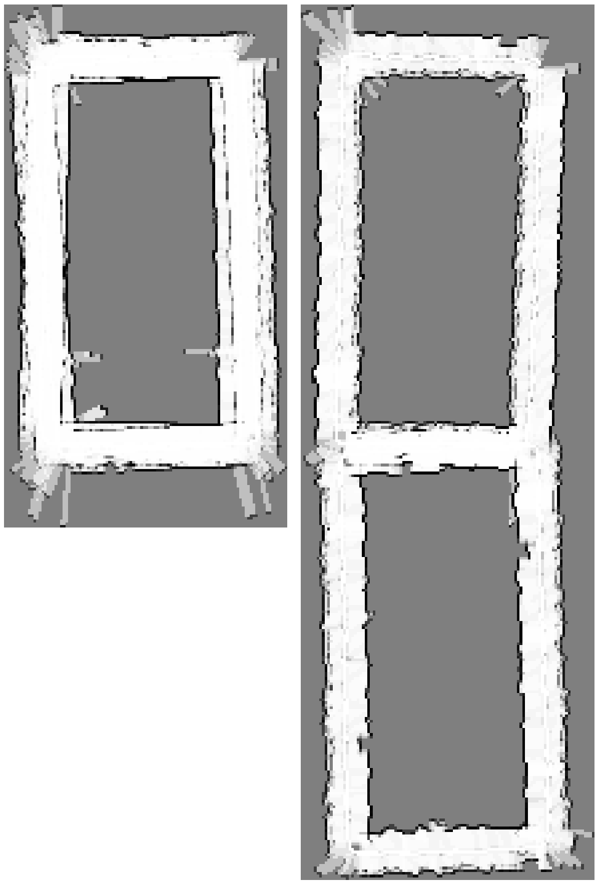


Figure 16 – Occupancy Grid Map Representations (Left: Double Loop, Right: Infinity Loop)

5.2 Particle Count

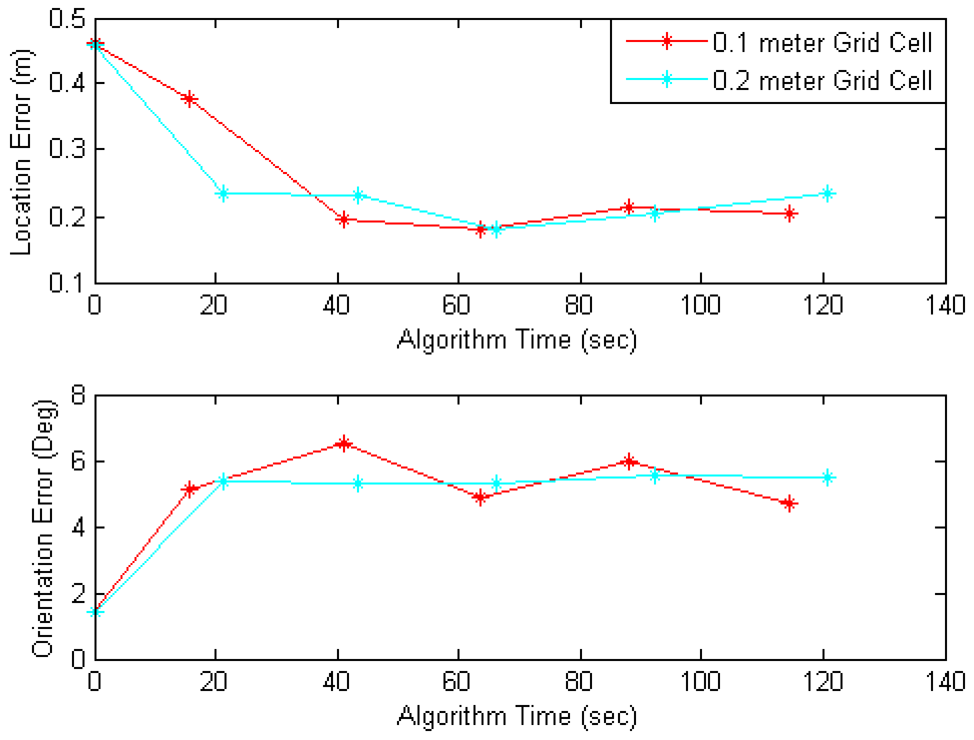


Figure 17 – Effect of Grid Cell Size on State Error

The number of particles is an important parameter for the Particle Filter. The approximation error goes to zero uniformly as the number of particles goes to infinity [3 p. 113]. A large set of particles is necessary to ensure the particles are not depleted from the correct region of states. Low error estimation or unlucky resampling may also cause the Particle Filter to diverge. However, since one of the goals of this thesis is to use the Particle Filter in real-time, the size of the particle set is limited to the hardware used. In experiments, the Sonar SLAM Algorithm was able to process approximately 1000 particles before any lag was seen. This was only possible for grid cell size of 0.2 meter; while 0.1 meter cell size limited the algorithm to 500 particles. A smaller grid cell size increases the level of detail in a map at the cost of memory and process time. Figure 17 shows the particle filter results with varying number of particles. The max time allowed for the algorithm to run in real-time was approximately 120 seconds. Each data point represents 5 runs of the Particle Filter with the Double Loop dataset using 100 to 500 particles for 0.1 meter grid cell and 300 to 1100 particles for 0.2 meter grid cell. A simulation run with a single particle was used as the initial error for comparison. The top plot shows the mean location error weighted with the final particle weights. The bottom plot shows the mean orientation error obtained similarly to the location error.

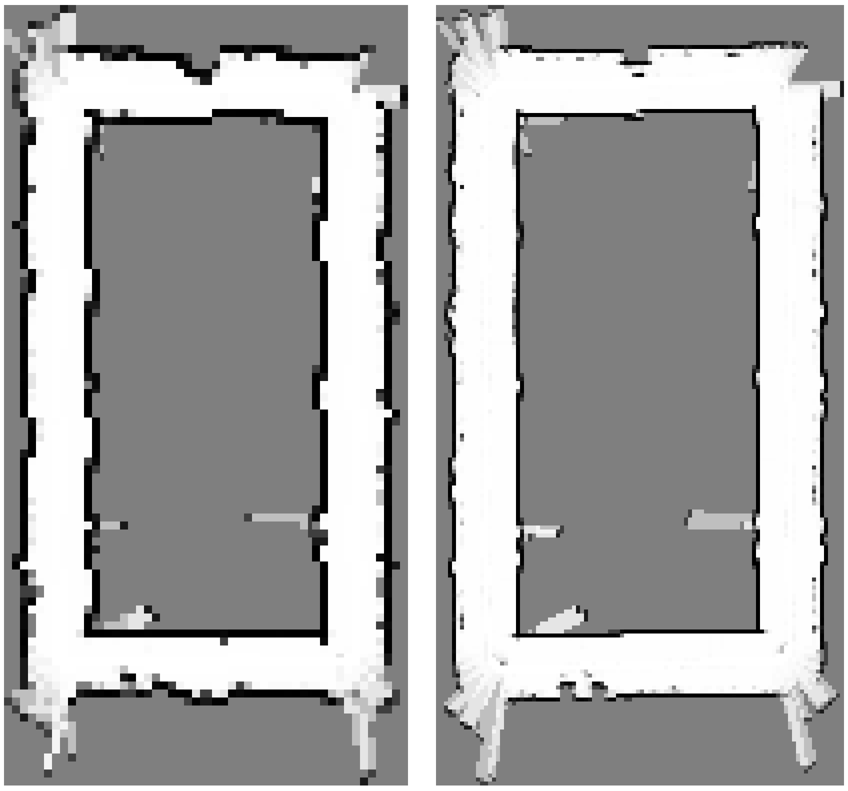


Figure 18- Best Double Loop Maps (Left: 0.2 Grid Cell Size, Right: 0.1 Grid Cell Size)

For both grid cell sizes, the Particle Filter converges with location error close to 0.2 meter and orientation error close to 5.5 degrees as the particle count is increased. Since the Particle Filter is able to contain more particles for higher cell size, the tests with 0.2 meter cell size perform better at lower time limits; however they had twice as much variance to mean ratio of the location error. Figure 18 shows highest weighted maps generated using 1100 particles for 0.2 meter cells and 500 particles for 0.1 meter cells.

5.3 Map Update Rate

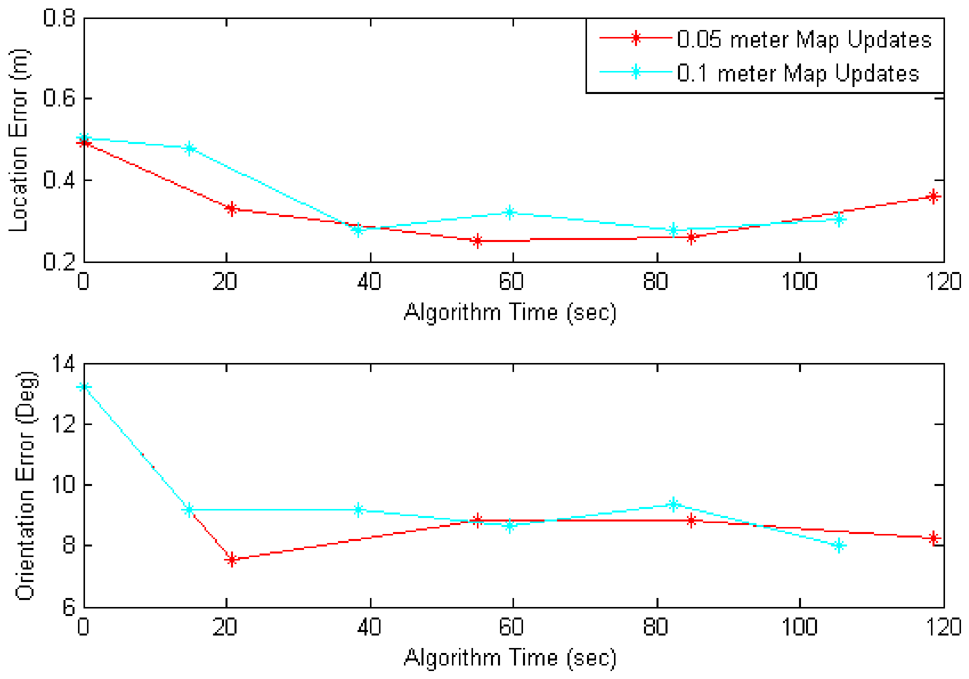


Figure 19 – Effect of Map Update Frequency on State Error

The particles process a new sonar measurement for every 0.1 meter the robot travels for the Double Loop dataset. This update rate was sufficient to update most of the grid cells occupied by the robot at least once by the second iteration of Double Loop. However, in the Infinity Loop, the robot revisits only a small portion of the environment, so the areas of the map only explored once are less reliable. Namely, 0.1 meter between map updates becomes too long and the resulting map contains more unknown or partially-known grid cells. To verify this, the Infinity Loop dataset was run with map updates after 0.1 meter and map updates after 0.05 meter. Figure 19 shows that change in update rate does not have a significant impact on the location and orientation error. As expected, more frequent map updates does improve the map quality as expected (see Figure 20); however, it also increase the algorithm time such that the maximum particle count is reduced from 500 to 400 with 0.1 meter grid cell size.

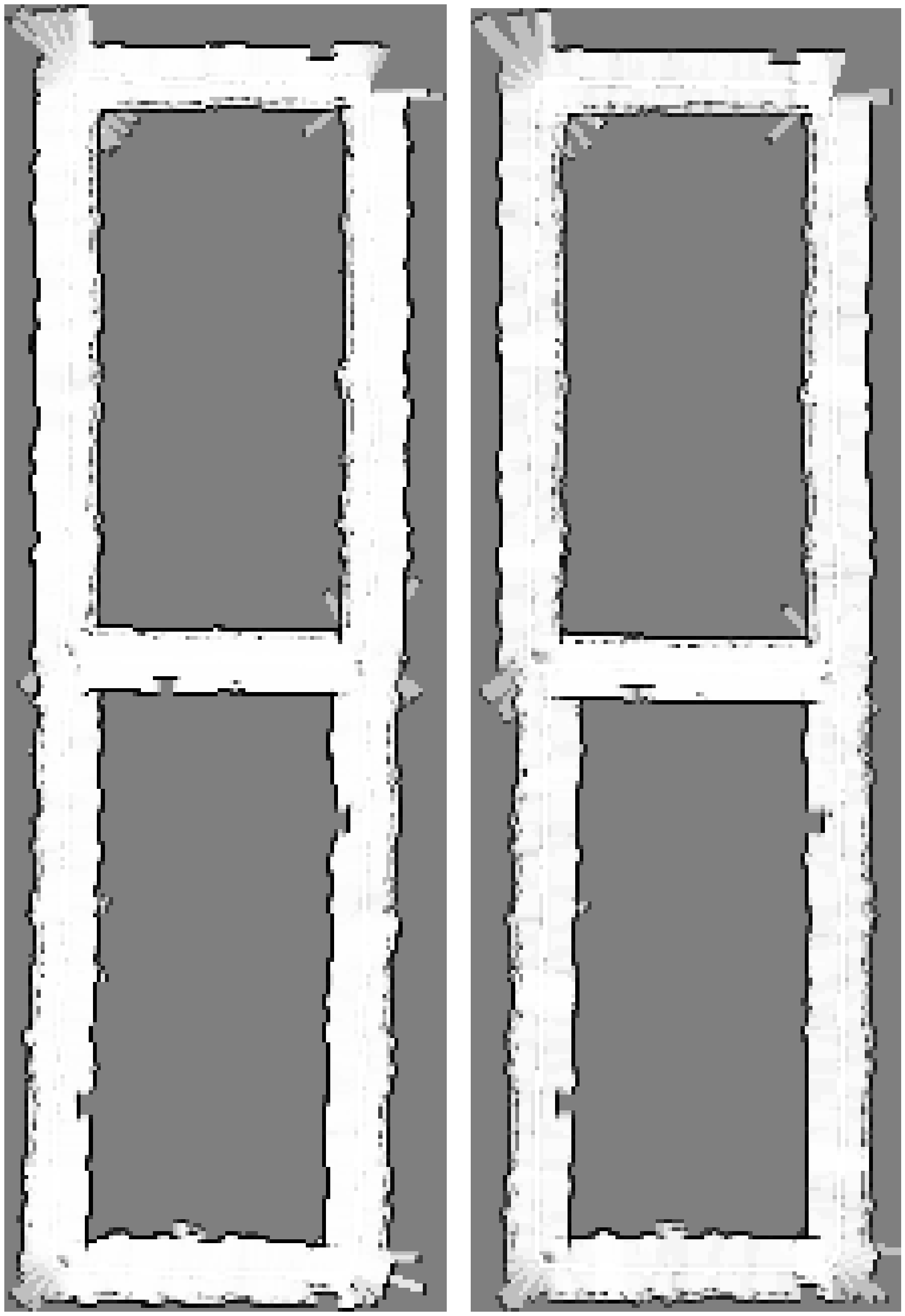


Figure 20 - Best Infinity Loop Maps (Left: 0.05 meter Map Update, Right: 0.1 meter Map Update)

5.4 Double Loop vs. Infinity Loop

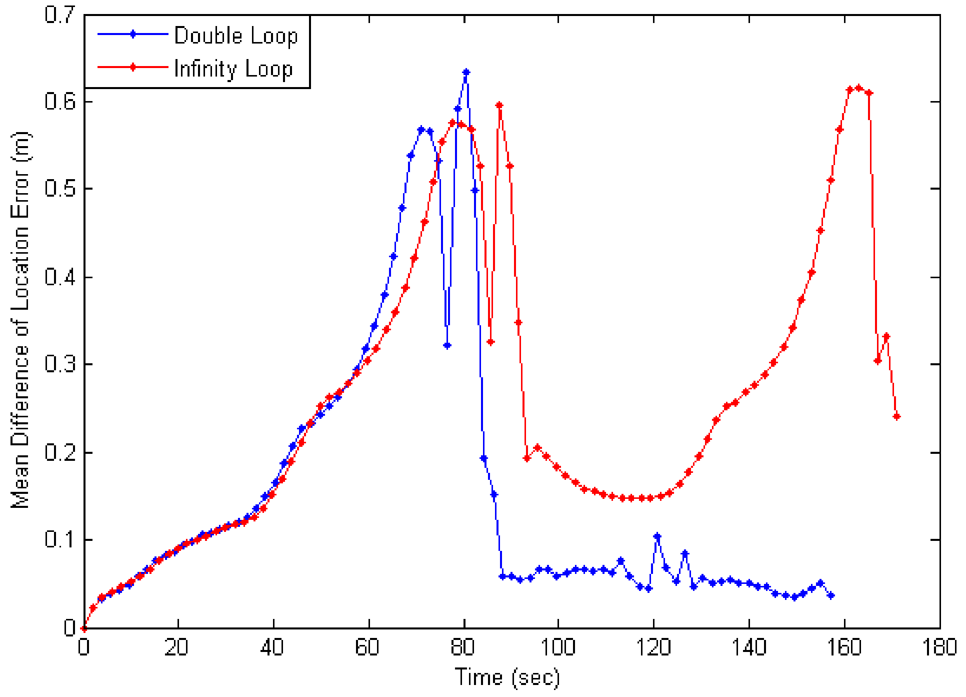


Figure 21 – Mean Difference of Particles' Location Error

Figure 21 compares the mean difference in location errors of the Particle Filter with the Double Loop and Infinity Loop datasets. The SLAM algorithm converged to mean difference of less than 0.1 meter when it resampled four consecutive times (4 meters) in the Double Loop dataset and continued to remain less than 0.1 meter as the robot traversed a previously explored path. However, for the Infinity Loop, the algorithm resampled for only 3 iterations and was unable to reduce the mean difference to less than 0.1. This created a larger uncertainty for the second loop for which the algorithm was again not able to converge.

Chapter 6: Conclusion and Future Work

The objective for this thesis was to implement an online SLAM strategy to map an unknown environment with a mobile robot in real-time. A Sonar SLAM Algorithm, shown in Chapter 4, was developed to incorporate real-time performance improvements to the Particle Filter. The Algorithm supported up to 500 particles for high resolution maps and 1000 particles for low resolution maps.

In the Double Loop dataset, the robot performs two iterations over the top section of the wing, so the Particle Filter performs resampling during the entire second iteration, which leads to lower variation in the particle locations and a better state estimate. However, in the Infinity Loop, resampling is only done during the shared middle section of the two loops. So, the Particle Filter is only resampling twice for 3.75 meters out of the 88 meters long path. This results in the particles being too dispersed to effectively close the loop when the robot returns to the start location.

This issue may be avoided if the robot sampled from a more accurate motion model. The odometry motion model used in this thesis is an approximate motion model that does not account for the actual motion of the robot. Using a linear or non-linear model estimate generated from the robot's control data and measurements, such as the ones generated in [11] or [10], would improve the accuracy of the odometer model and so may improve the overall performance of the particle filter. A more general way to improve the motion model is to use the sonar sensor observations along with odometry, while sampling a new particle location. By using the most recent sonar measurements, the odometer model can be adjusted to the current environment, such as to distribute the particles along the direction of travel when traveling in a long hallway. This approach requires a pre-computed range sensor model which can be calculated using the map generated in this thesis as an approximation. Such an improvement would increase computation complexity of the SLAM algorithm; however, a more efficient sampling will offset this cost by reducing the number of particles needed to generate an accurate map as seen in [14].

The particle filter can also be improved by performing active loop closing. In this thesis, the robot performed passive loop closing, since it was remotely controlled in a set trajectory throughout the run. In active loop closing, the algorithm controls the robot exploration in a way that the robot returns to known locations to close loop. [15] shows a possible implementation of an active loop closing algorithm. They suggest using a topological map with nodes representing locations visited by the robot. If the robot senses a shortcut, a small distance on the grid map, from the current location to one of the nodes of the topological map, it can try to move in that direction to reduce its location uncertainty.

Bibliography

1. *Vision-based SLAM using natural features in indoor environments.* **Miro, Jaime V., Dissanayake, Gamini and Zhou, Weizhen.** s.l. : Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. pp. 151-156.

2. *Simultaneous Landmark Classification, Localization and Map Building for an Advanced Sonar Ring.* **Fazli, Saeid and Kleeman, Lindsay.** 3, New York, NY : Cambridge University Press, 2007, Vol. 25, pp. 283-296.

3. **Thrun, Sebastian, Burgard, Wolfram and Fox, Dieter.** *Probabilistic Robotics.* Cambridge, MA : MIT Press, 2005.

4. **Varveropoulos, Vassilis.** Robot Localization and Map Construction Using Sonar Data. *The Rossum Project.* [Online] rossum.sourceforge.net/papers/Localization/PosPosterv4.pdf.

5. *DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks.* **Eliazar, Austin and Parr, Ronald.** s.l. : International Joint Conference on Artificial Intelligence, 2003, Vol. 18.

6. *Monte Carlo Localization for Mobile Robots.* **Dellaerty, Frank, et al.** Antwerp, Belgium : International Conference on Robotics and Automation (ICRA), 1999.

7. **Schroter, Christof, Bohme, Hans-Joachim and Gross, Horst-Michael.** Memory-Efficient Gridmaps in Rao-Blackwellized. *EMCR.* 2007.

8. *Robust map building for an autonomous robot using low-cost sensors.* **Schroter, Christof, Bohme, Hans-Joachim and Gross, Horst-Michael.** Hague, Netherlands :

Systems, Man and Cybernetics, 2004 IEEE International Conference on, 2004, Vol. 6.

9. **Napora, Jared.** *Implementation, Evaluation, and Applications of Mobile Mesh Networks for Platforms in Motion (Master's Thesis).* College Park, MD : Univerity of Maryland, 2009.

10. **Karvounis, John.** *Theory, Design, and Implementation of Landmark Promotion Cooperative Simulataneous Localization and Mapping (Doctoral dissertation).* College Park, MD : University of Maryland, 2011.

11. **Stanley, Michael.** *Implementation of Kalman Filter to Tracking Custom Four-Wheel Drive Four-Wheel-Steering Robotic Platform (Master's Thesis).* College Park, MD : University of Maryland, 2010.

12. PING)))™ Ultrasonic Distance Sensor (#28015) . [Online] 9 11, 2009. http://www.parallax.com/Portals/0/Downloads/docs/prod/acc/28015-PING-v1.6.pdf.

13. A. V. Williams. *Engineering Information Technology.* [Online] http://eit.umd.edu/maps/avwilliams.

14. **Grisetti, Giorgio, Stachniss, Cyrill and Burgard, Wolfram.** Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics.* 2007, Vol. 23, 1, pp. 34-46.

15. **Stachniss, Cyrill, Hahnel, Dirk and Burgard, Wolfram.** Exploration with active loop-closing for FastSLAM. *Intelligent Robots and Systems.* 2004, Vol. 2, pp. 1505-1510.