RICE UNIVERSITY

Pose Estimation With Low-Resolution Bearing-Only Sensors

by

Joshua B. Rykowski

A THESIS SUBMITTED
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE

Master of Science

APPROVED, THESIS COMMITTEE:

[illegible]

Dr. James McLurkin
Assistant Professor of Computer Science

Alan L Cox

Dr. Alan L. Cox
Associate Professor of Computer Science
and Electrical and Computer Engineering

[illegible]

Dr. Marcia K. O'Malley
Associate Professor of Mechanical
Engineering

Houston, Texas

December, 2011

ABSTRACT

Pose Estimation With Low-Resolution Bearing-Only Sensors

by

Joshua B. Rykowski

Pose estimation of neighboring robots is a key requirement for configuration control behaviors in multi-robot systems. Estimating pose is difficult without system constraints, it is even more challenging when using minimalistic sensing alongside limited bandwidth. Minimal sensing models are a well studied field in robotics and are relevant to our particular hardware platform, the r-one, which has sensors that only measure a low-resolution bearing to neighboring robots. These bearing-only sensors are simpler to design with and cheaper to deploy in large numbers. In this thesis, I focus on the r-one multi-robot system which is capable of coarsely measuring the bearing, but not the distance, to neighbors. These sensors have a angular resolution of only 22.5 degrees due to the construction of the infrared system. I develop a particle filter algorithm that allows the r-one robot to estimate the pose of a neighbor using the infrared communication system and odometry measurements. This algorithm relies on the fusion of a coarse bearing measurement and neighbor velocities and is optimized to use the smallest communications bandwidth possible. I tested this algorithm with a simulation to demonstrate its effectiveness across varying sensor setups, neighbor update periods, and number of particles.

Acknowledgments

There are two people who share the credit for supporting me throughout my thesis adventure. They are my wonderful and devoted wife, Carrie and my ever patient and ever understanding advisor, James McLurkin.

To Carrie, thank you so much for your support and your devotion during this academic ordeal! You took care of our home and our wonderful children (Julianne, Preston, Emmi, and Bryce) with such grace and aplomb. Through all of the late nights, early mornings, last minute "I need another hour to run another simulation" (only to come home several hours later) you kept the home fires burning and always had a positive attitude and knew just what to say when my motivation waned. You are the reason I am a better person. I love you and our family!

To James, thank you so much for your wisdom, guidance, and patience. My time under you definitely started off at a snails pace for a variety of reasons. However, you listened when it was needed, you understood what I was saying and stepped back when I needed it and stepped in when I thought I didn't need it but really did. I will always be an "epsilon away...". It was a pleasure to work with you!

Finally, I would like to thank my other committee members, Marcia O'Malley and Alan Cox, for their understanding and patience.

Sometimes you just need a little faith.

Table of Contents

Illustrations

Tables

Chapter 1

Introduction

Range Matters. Imagine yourself on Sicily Drop Zone at Fort Bragg, North Carolina immediately following a 2300 time-on-target airborne "jump". Knowing that the assembly area for your unit is about a kilometer east of the drop zone you quickly extricate yourself from the parachute harness, pull out your night vision goggles (NVGs) (the monocle version which limits your depth perception) and begin to expeditiously move (i.e. run) towards the assembly area.

You fail to remember that looking through this particular version of the NVGs is akin to closing one eye (it severely limits your depth perception). All of a sudden you see a flash of light and then you are looking up at the stars, listening to someone chuckle in the distance. Your inability to estimate range caused you to clothesline yourself on a low tree branch in view of the rest of your unit. Our new robot, the r-one, faces a similar problem, without the ability to estimate range between themselves and their neighbors, these robots are destined to live a life of simple, trivial tasks and always be fearful of low branches!

The ability to quickly and accurately obtain a good estimate of the pose, (x, y, θ), of neighboring robots is a requirement to produce many complex behaviors in multi-robot systems. These behaviors include, but are not limited to: a robot maintaining a critical distance from its neighbor, a robot ensuring reliable communications by maintaining a maximum distance, or many robots performing SLAM (simultaneous localization and mapping)[1, 2]. Many systems can measure range directly with the robot's sensors, but our low-cost robot has a minimalistic sensor system that only measures angles between neighboring robots. This thesis describes a particle filter algorithm that can estimate the pose of a neighboring robot with low-resolution bearing sensors.

1.1 Motivation

A multi-robot system has can accomplish tasks that a single robot cannot. An example of this would be conducting surveillance, or collecting data, over a large area. This is simply not possible with a single robot because they cannot be in multiple locations simultaneously. However, a multi-robot system can disperse and ensure coverage of the area of interest. Also there are tasks that a multi-robot system can accomplish much more efficiently. A typical task that a multi-robot system will be able to accomplish much more efficiently than a single robot is exploration. A single robot can explore any size area. However, a multi-robot system will be able to explore the same area in a fraction of the time.

The multi-robot system design to originate from the Multi-Robot Systems Lab at Rice University, the r-one robot shown in Figure 3.1(a), attempts to fill a gap in the short list of educational multi-robot systems. Currently, it has a cost of around $220 per robot, a large sensor suite including a gyro, accelerometer, wheel encoders, and light sensors. It includes a radio for global control, an infrared beacon for ground-truth localization, and an infrared inter-robot communication and localization system. The robot can be programmed in C, or can run an embedded Python interpreter to make programming more accessible to younger, less experienced, students. We are currently using the platform in an introductory first-year engineering course and a graduate level course on robotics, with good success to date.

However, this platform lacks sensors that can directly measure range to neighbors. However, we can use its bearing measurements to infer the range. In this thesis, I estimate the range to neighboring robots with a particle filter algorithm on the r-one. This estimation also recovers the full pose of the neighbor, (x, y, θ).

1.2 Contributions

The importance of this pose estimator is that it fills a critical gap in the current capabilities of the r-one robot without increasing the cost of the platform. Currently, the r-one platform can only measure limited geometric information its neighbor without any idea of the range to the neighbor or the neighbor's (x, y) position in the sensing robots coordinate system. The resolution of bearing that the r-one determines is approximately $22.5°$, which is very coarse. Section 3.1.1 provides an in-depth explanation on the current sensor capabilities of the r-one.

In developing my particle-filter algorithm, I have successfully extended the state estimation capabilities of the r-one platform without adding to the overall cost, either in build cost or materials cost. This extension is useful in that the r-one can now be used to perform even more complex tasks than it was capable of performing. An example of this increased capability would be adding formation control to flocking r-ones. By being able to determine distance between each other the flock of r-ones can morph their shape to be able to move through a dynamic environment, or maintain a specified distance to neighboring robots.

Chapter 2

Related Work

There are three main areas of work that support my thesis. These include multi-robot systems, minimalistic sensing and probabilistic robotics. In this chapter I describe what defines a true distributed multi-robot system and discuss literature on current full and partial multi-robot systems. The review of minimalistic sensing is split into two major subsections. The first subsection discusses the *Power of Robots* which establishes a dominance relation to stratify different robot platforms. The second subsection focuses on coordinate systems and uses the dominance relation to rank-order the current multi-robot systems in terms of their coordinate systems. The final focus is on probabilistic robotics, more specifically, particle filters for state estimation. I finish with a discussion about the various particle filter implementations.

2.1 Multi-Robot Systems

I define a true distributed multi-robot system as meeting four distinct criteria:

- Distributed Sensing
- Distributed Actuation
- Distributed Computation
- Distributed Communications

According to this list, there are very few multi-robot systems that meet all four of these requirements. A majority of multi-robot systems meet only a subset of this criteria, I refer to these as partial multi-robot systems. These partial multi-robot systems are still relevant in that they are used to research and solve problems within their particular criteria-space. This ultimately advances the knowledge for that specific research domain within multi-robot systems.

The General Robotics, Automation, Sensing and Perception (GRASP) laboratory at the University of Pennsylvania utilizes a quadrotor multi-robot system for research on coordinated, dynamic flight [3]. Their particular setup meets the distributed actuation criteria mentioned above. However, they utilize the Vicon Motion Capture System to provide all of their sensing instead of relying on distributed sensing [4]. This allows their laboratory to constrain the problem-space of their multi-robot system to better study specific aspects of their system.

Another current partial multi-robot system, this one built for educational outreach, is the Educational, Research-Oriented, Sensing, Inexpensive robot, eROSI, built by the Center for Distributed Robotics at the University of Minnesota. This particular multi-robot system possesses a moderate array of sensors that include light sensors, infrared range sensors, encoders, a camera and a bluetooth module. The approximate cost of each eROSI is $500.00, placing it at almost double the cost of the r-one but still substantially cheaper than other multi-robot systems [5].

There are four multi-robot systems that meet all four criteria and they are: the r-one robot, discussed in detail in Section 3.1, the SwarmBot multi-robot system, the Khepera III system, and the e-Puck with the infrared turret. The first two were designed and built by James McLurkin with the latter while he was with iRobot and the former with the Multi-Robot Systems Lab at Rice University.

The r-one multi-robot system is covered in detail in Section 3.1 [6]. However, a brief overview of the r-one follows. It is a 10 cm robot which allows many robots to be used in a small space. The infrared system is the primary mode of inter-robot communications. It was designed as an extremely low-cost solution for educational outreach and meets that goal with a total per robot cost of approximately \$ 220.00. The robot can be programmed in either Python or C (using the FreeRTOS kernel). This supports both undergraduate introductory courses and advanced robotics courses for graduate students.

The SwarmBot multi-robot system was developed by James McLurkin and represents a robust system with a mature set of algorithms [7, 8, 9]. The hardware setup consists of a complex sensor suite the consisting of bump sensors, light sensors, a camera, encoders, and a infrared communications system (which can measure the full pose of neighboring robots, bearing, orientation, and range). The robots also use a proprietary operating called the Swarm Operating System (SwarmOS^{TM}) which provides an API for writing applications for the SwarmBot. It uses the ThreadX real-time operating system from Express Logic that supports threads, mutexes, semaphores, queues and memory allocation. This multi-robot system uses a synchronous distributed communications model. All of the SwarmBots transmit with the same transmission period ensuring that each SwarmBot will receive only one message from each of its neighbors during the period. This provides an upper bound on the time each SwarmBot has to wait for a message and lets us model the entire system as a synchronous distributed system. This is described in further detail in Section 3.1.1.

The Khepera III multi-robot system was produced by K-Team Corporation with assistance from the Distributed Intelligent Systems and Algorithms Laboratory at EPFL [10, 11]. This particular platform is an extensible design that supports multi-robot education and research with a sensor suite that includes nine infrared range detectors for obstacle detection, five ultrasonic sensors for long range obstacle detection, and two cliff detectors. The cost of this multi-robot system is prohibitive for large populations at approximately $3,500 per robot.

The ePuck multi-robot system was developed as an educational outreach tool by EPFL [12, 13]. With the addition of a separate communications board this is a true multi-robot system. There are a couple key features that make this an attractive educational tool. First, it is designed to be programmed over bluetooth and does not need to be plugged into a computer. Second, it has a removable battery pack allowing for extended usage time. However, it does have some limitations as well. First, it is expensive with a price that exceeds $1,000, even more so with the range and bearing board, which increases the cost by approximately $500 . Second, the curriculum is built around their definition of a swarm which is approximately three robots. The caveat to this limitation is that the system will scale to a larger size if that is desired.

2.1.1 State of the Art in Outdoor Multi-Robot Systems

The Multi Autonomous Ground-robotic International Challenge (MAGIC) was a competition sponsored by the United States and Australian Departments of Defense in an effort to push the development of the next-generation of the fully autonomous ground vehicles [14]. More specifically, the need to fill a specific technology gap for urban combat exists and this competition sought to address that gap.

The winner of MAGIC 2010 was Team Michigan lead by Professor Ed Olson. The heart of this system is the inclusion of a robot-operator interface. The global state of the map and robot status is integrated into a single user interface. The overall commander can take control of any robot at any time and have his orders supersede any of the current robot behaviors. Even though their multi-robot system was custom built to the MAGIC 2010 specifications the chassis can be modified easily to relax the constraints placed on their design during entry into the competition.

2.2 Minimal Sensing

The theme of minimal sensing is to determine how much a robot can accomplish with the least amount of capabilities, or the simplest sensor to achieve a given task. This is a subfield of probabilistic robotics that is still being studied in depth and has been for over 20 years now.

Erickson et al. provides a spartan approach by using only a clock and a bump sensor to localize a robot within a complex environment [15]. In this work, the researchers simplify their world model to only include points along the boundary of environment and ignore all points inside the boundaries. This novel approach to redefining the environment allows them to develop an algorithm that reaches localization given only the initial orientation of the robot and an extremely limited sensor suite and not the robot's initial position within the environment. Their algorithm is built around a heuristic that looks at the entropy of the system and seeks to reduce that entropy by choosing motions accordingly.

Yu et al. provides a proof of concept of minimal sensing and control with a simple system modeled on a dubins car [16]. This model has only three states that allows it to follow a neighbor and still have all of the agents converge. This paper is useful in that it shows a minimal sensing model can still cause agents to converge without measuring range or bearing. This convergence is accomplished by means of a simplified control law. Additionally, the agents do not communicate at all. Even with this incredibly constrained system, they prove that any number of agents, placed randomly within their environment, will always converge.

Another approach to minimal sensing is to use a sensor network that tracks a target with a binary sensor using a particle filter [17]. Aslam et al. describes a system with two types of sensors in the network, one that senses the target moving towards the sensor and another that senses the target moving away. These simplified sensors emit a single bit of information which allows the researchers to completely remove the bandwidth constraint faced by any multi-agent systems. A drawback to this approach is that the sensor network can only reliably determine the motion of the target. By extending the model and combining the single bit with local proximity information garnered from an infrared system the location and direction can be determined.

2.2.1 Low-Cost Sensors

A low-cost lidar alternative, referred to as a laser distance sensor (LDS), was introduced at the 2008 ICRA Conference with a total production cost of $30.00 and built with commercial off the shelf components [18]. This LDS has an accuracy of 3 centimeters out to 6 meters, 10 Hertz acquisition, and 1 degree resolution over a full 360 degree scan. The inclusion of this sensor could greatly extend the r-one's capabilities at a minimal increase in cost. However, the major drawback is the lack of actual design plans available to the public. Currently, the only way to get an LDS is to purchase a Neato XV-11 and remove the LDS. This has already been accomplished through a "hack" competition sponsored by www.robotbox.com and now there are various APIs that allow anyone to use the LDS in a custom application [19].

The introduction of the Microsoft Kinect and its subsequent hacking and repurposing provides yet another vector with which to provide low-cost sensing. The Microsoft Kinect camera utilize the RGB-D (red, green, blue and depth) camera technology developed by PrimeSense [20]. The depth refers to the pixel distance from the camera and is captured at the same time as the RGB image to provide a threedimensional image. The capabilities of these cameras are limited in the following ways: they only provide depth up to a distance of 5 meters, their depth estimates are noisy and their field of view is approximately 60°. Even with these limitations, this sensor can extend the capabilities of a multi-robot system by providing an inexpensive means to ascertain range information as shown by Henry et al. [21]. Additionally, Herbst et al. provides a variation of SLAM, simultaneous localization and mapping, that utilizes the Kinect RGB-D camera to identify details of objects within a robot's environment to enhance the robot's performance [22].

2.2.2 Power of Robots

This thesis resides in the shadow of a much larger discussion about the types and complexity of sensors required to perform a particular task, with two primary examples coming from Rus et al. and Erdmann et al. [23, 24].

The paper by Rus et al. looks at whether or not a multi-robot system requires an explicit control law to be able to accomplish the complex of changing the orientation of an object [23]. The authors determine that planning, global control, and explicit communication is not a requirement to complete these tasks. This correlates with the *Power of Robots* theme in that it attempts to empirically derive the minimal amount of sensing required to have a mulit-robot system change the pose of an object.

Erdmann proposes a method of reverse engineering sensors based on a robot's task, its actions, and the uncertainty in control [24]. The bottom line of this paper is that the author attempts to address three issues. The first is to utilize simple sensors to improve the reliability of operations. Second, Erdmann attempts to circumvent the sensing uncertainty issue by trying to design simple sensors that can measure the particular uncertainty directly. Finally, he tries to define the amount of information required for different task strategies.

However, there is no consensus in the current literature of how to best understand and stratify the relative "power" of robots and sensors [25]. The general theme seems to be constructing an ordering based on minimal robot/sensor capability that can solve tasks. The introduction of sensors with greater capabilities incurs the penalty of system complexity alongside increased cost (which can be mitigated in some cases, see RGB-D and LDS in Section2.2). Finally, for multi-robot systems with large populations sensor cost is a limiting factor: when you add an additional sensor to the sensor suite you have to multiply the cost by the number of robots that will be in your swarm.

My particle filter technique settles into this subfield rather well. I take an extremely simple bearing system that has an approximate resolution of 22.5° and compute a usable range from it. This competes with the current solutions that are currently available for the e-Puck and the Khepera III [11, 13]. In regards to minimalistic sensing my particle filter algorithm presents an alternate method which will meet the same neighbor localization capabilities as the two commercial solutions but not add any to cost to the current r-one bill of materials.

2.2.3 Coordinate Systems

O'Kane et al. define a dominance relation that states a robot *dominates* another robot if it can collect at least as much information as another robot [25]. This relationship is further clarified through the use of *robotic primitives*, or self contained "instruction sets" for the robot that may involve sensing, motion or both. Building upon this dominance relation I then add coordinate systems to further stratify the multi-robot systems. McLurkin defines four different coordinate systems, I consider

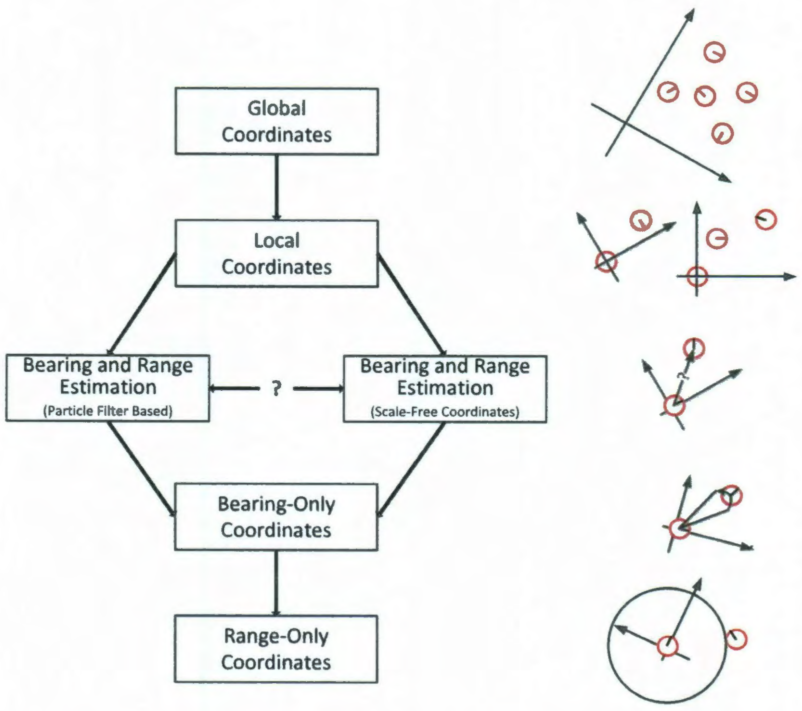

Global Coordinates
Local Coordinates
Bearing and Range Estimation (Particle Filter Based)
?
Bearing and Range Estimation (Scale-Free Coordinates)
Bearing-Only Coordinates
Range-Only Coordinates

Figure 2.1 : A representation of the dominance relationship of coordinate systems.

these coordinate systems to be *robotic primitives*, of which I only focus on global and local coordinate systems [9].

With this dominance relationship as a backdrop I can then define a partial order of systems for the current multi-robot systems based on the dominance relationship of coordinate systems. In Figure 2.1 you see two columns. The column on the left represents the dominance relation of the coordinate systems and the column on the right is an illustration of the coordinate. Starting from the top the first coordinate system is a global coordinate system. This coordinate system is the most dominant of all the coordinate systems due to its ability to represent information for any of the subsequent coordinate systems. Additionally, another reason this coordinate system is the most dominant is that each robot's pose measurement of their neighbors is consistent.[†] This is represented on the right as a single frame of reference for all robots.

The next coordinate system is the local coordinate system. In this particular coordinate system the robot population is split up into local "neighborhoods" or subsets of the overall robot population. These neighborhoods each have their own local coordinate system that allows the robots within the neighborhood to know the pose measurement to all the other local robots relative to their local coordinate axis. However, each neighborhood of robots does not know the information within nearby neighborhoods. This is depicted on the right as two separate neighborhoods, each with their own local coordinate system.

The next step down in this dominance relationship is a bearing and range estimation coordinate system. This includes both my particle filter implementation and another method currently being researched in the Multi-Robot Systems Lab at Rice University called scale-free coordinates. Both of these coordinate systems rank below local coordinate because they both only produce an estimate of the pose for the neighboring robot. The question mark in between the two coordinate system merely indicates that we do not currently know the relationship between the two coordinate systems in terms of which one is more dominant than the other. The depiction of these coordinate systems on the right indicates the bearing and range to a neighboring robot. Again, the question mark indicates that the range is only an estimate and not the actual range between the two robots.

The particle filter implementation is a probabilistic technique that attempts to determine some sort of hidden state variable. This is an variant of the Bayes filter algorithm that utilizes a collection of estimates, called particles, to represent the

[†]I.e. the measurement between any two robots is the same from each robot's point of view

probability distribution function of the hidden state variables. This is discussed in detail in Chapter 4.

As mentioned above, an alternate approach to pose estimation on the r-one platform is a technique referred to as *Scale-Free Coordinates* [26]. This technique utilizes the same multi-robot system as I do and attempts to mitigate the same hardware limitations as well. The main difference is the algorithm producing the estimated pose. I rely on a probabilistic method to infer the hidden state variables with my particle filter implementation. The scale-free implementation leverages trigonometry and the local network to produce a reasonable estimate of the range between robots, up to a scaling factor. The advantages of the scale-free technique are shared with my technique; no additional cost added to the r-one platform while increasing the r-one's capabilities. A major disadvantage of this technique is the increased utilization of bandwidth due to the communications required to compute the geometry of the local neighborhood. This computation is covered in detail in Section 3.2. However, this technique represents a promising alternate approach to determine an estimate of the pose of neighboring robots.

Continuing down the dominance relation I arrive at the bearing-only coordinate system. This coordinate system is currently implemented on the r-one and only allows the robot to measure the bearing to its neighbor and not the range. This is indicated on the the diagram on the right by showing a sector measurement to the neighbor and nothing more.

Finally, the lowest coordinate system on the totem pole is the range-only coordinate system. This coordinate system is the lowest simply because there is very little you can do with knowing only the range and nothing more. If you have bearing-only you can at the very least navigate by landmark recognition as described by Loizou and Kumar [27]. In this paper they propose a provably correct bearing-only navigation controller that is biologically inspired and utilizes landmarks. This coordinate system is depicted on the right as a circle centered on the robot. The neighboring robot can be anywhere along the edge of this circle.

In light of this partial order I am trying to extend the capabilities of the r-one to include a reasonable range measuring system that does not increase the cost of the platform. This would raise the r-one's current coordinate system to bearing and range-estimation placing it between bearing-only and local coordinates.

2.3 Probabilistic Robotics

Probabilistic robotics revolves around the idea of attempting to explicitly represent the uncertainty that is inherent in robotic systems. This uncertainty stems from four areas:

- Environment
- Robot
- Limited, Noisy Sensors
- Inaccurate Models

By utilizing a probabilistic method one can then mitigate these four areas of uncertainty.

A particular subset of probabilistic robotics is state estimation which refers to a robots attempt to determine some hidden state based on sensor data. A particle filter is one method that can be used for state estimation. Particle filters have been studied for almost 20 years and are a mature area of research for robotics. Dellaert, Fox, Thrun and Burgard used a particle filter to solve robot localization using a method they refer to as Monte Carlo Localization [28]. This was an incredible breakthrough as the robot localization problem was widely recognized as the "most fundamental problem to provide a mobile robot with autonomous capabilities" [29].

2.3.1 Main Resources

First, and foremost, I consider the textbook, Probabalistic Robotics, written by Fox, Thrun, and Burgard to be the primary resource for anything relating to probabilistic robotics [30]. It does a great job of building up levels of complexity within probabilistic robotics. It starts with the theory behind probabilistic robotics by introducing the Bayes Filter algorithm. It then moves into discussing variations of the Bayes Filter algorithm to include Kalman Filters, Extended Kalman Filters, and Particle Filters. It also discusses different methods to model the possible sensors. Finally, it culminates with simultaneous localization and mapping (SLAM).

Rekleitis provides an exceptional technical report that was a byproduct of the culmination of this thesis on particle filters for mobile robot localization [31]. It provides a detailed description of a particle filter, both in plain text and in pseudo-code. He also addresses the types of error that arises for a robot when it is either translating or rotating and how to model that error. Appendix D provides substantial information on the main resampling methods currently used. This appendix also provides enough information on the variations on resampling so that anyone can pursue the pertinent literature on their own.

2.3.2 Techniques That Worked

Out of the myriad of specialized techniques available for particle filters I only found two techniques that really assisted my particle filter in attaining convergence quickly. These techniques are discussed in the subsequent paragraphs.

The Sensor Resetting Localization (SRL) algorithm produced by Lenser et al., although developed for Monte Carlo localization (MCL), proved to be quite useful during my particle filter algorithm development [32]. The authors determined three key problems with MCL; MCL requires more particles during global localization than when tracking, MCL cannot handle large modeling errors, finally MCL does not handle unexpected or unmodeled robot movement. To mitigate with these three problems their SRL algorithm adds a resampling phase to the MCL algorithm. SRL determines if the probability of an area designated by the samples is low then the algorithm replaces those samples with samples taken from the current sensor measurement. This allows the robot to attain global localization quickly and with less samples. This increases the overall efficiency of the algorithm and reduces its computational load. It is also robust against a poor robot model in that once the systemic error rises the poor samples will be replaced with current sensor samples, this makes the robot resistant to unmodeled movements as well. This proved to be incredibly useful in my particle filter implementation. By utilizing this method I was able to use current sensor measurements to "bias" the *probability distribution function* of the particles so that the estimated pose would be auto-corrected if it was not close to the current sector being sensed.

Jensfelt et al. provides a unique resampling method to combat an issue that faces all particles filters, the reduction of the probability distribution function to a single point [33]. This collapsed distribution provides an extremely poor estimate of the state with particle filters. This ultimately will cause the particle filter to fail. Their modification allows the current sensor reading to contribute more to the weighting of the particles then the current output of the predictive model. This is accomplished by re-injecting particles based on the current sensor reading during the sensor update.

These particles are given the maximum weight of the current particles and replace the particles with the lowest weight. I use this reinjection method for the multiplicative weighting implementation of my particle filter.

Liu et al. describes the basis of sequential importance sampling (used in particle filters) in depth in this particular paper [34]. The main contribution to my thesis comes in the form of a heuristic that allows me to declare when I want my particle filter to resample. This heuristic is referred to as the *effective sample size*, or ESS. This refers to the equivalent number of independent and identically distributed samples at time t. Essentially, this heuristic allows the particle filter to resample when there are a certain amount of near-zero weight particles as these particles will not affect the estimated pose. The issue with this heuristic and the r-one sensor model is that we have a binary sensor, either you are in the sector or you are not. This does not produce a sample set of independent and identically distributed particles. However, I was able to use this heuristic with the multiplicative weighting version of the particle filter that I developed.

Fox et al. describes a situation in which their particle filter implementation performs poorly with both high sensor noise and low sensor noise [35]. Even though this article deals with mobile robot localization, more specifically Monte Carlo localization, it helped me to realize the a flaw in developing my particle filter algorithm. The introduction of noise in both the odometry measurements and the bearing measurements allowed me to produce a particle filter that would converge. In fact, they offer three analytical "approaches" to accommodate highly accurate sensors. A drawback to their approach seems to be the particle set size, which they declare "optimal" between 1,000 and 5,000. Right now we are currently using 500 particles with anything above 1,000 particles I exclude due to the computational constraints of the r-one.

2.3.3 Techniques I Considered

There are several particle filter techniques that seemed viable at first. However, upon further research I did not utilize the following techniques for various reasons discussed below.

The first technique that I considered and then discarded is a real time particle filter algorithm (RTPF) designed to handle sensor information that arrives significantly faster than the update rate of the filter (250ms in our case) [36]. The authors explain that their RTPF does not throw out any sensor data in between sensor updates, instead they consider all sensor updates by sampling over all of the sensor readings within a predetermined window size, this produces a state that is a mixture of all of the sample sets. This is mainly due to the fact that n samples are distributed among the k observations within an estimation window. At each timestep only n/k particles are needed, reducing the computational load on the r-one and allowing for more particles. In my algorithm I only sample at every update period and I do not care about sensing in between these update periods, making this technique moot.

Resampling is referred to as the "trick" of the particle filter where the particles are forced back to the posterior $bel(x_t)$ [30]. The simplest resampling algorithm used in particles filters is the Sample with Replacement algorithm. One drawback to this particular algorithm occurs after the generation of an array of random numbers uniformly distributed in $[0, 1]$. Once this array is created it is then sorted. This is typically implemented as a quicksort and takes $O(nlogn)$ time. Carpenter et al. provides a linear time ($O(n)$) algorithm by using the cumulative sum of the negative logarithm of random numbers uniformly distributed in $[0, 1]$ [37]. This produces a sorted sequence of random numbers uniformly distributed in $[0, 1]$. This particular resampling method is not used in my particle filter algorithm due to ease of implementation. However, if I need to further increase my efficiency I would rework my algorithm to use this resampling method.

Kullback-Liebler distance sampling, KLD-sampling, is an another technique developed to increase the efficiency of a particle filter algorithm by dynamically adjusting the particle sample size over time [38]. First, the Kullback-Liebler distance refers to dissemblance between two probability distribution functions. By putting a bound on the error being introduced by the re-injection of particles, the the Kullback-Liebler distance determines the maximum number of particles needed to meet that bound. Any time the number of particles required to represent the posterior probability is reduced, the computational load is also reduced. I initially contemplated using this resampling method to make my algorithm more efficient. However, the memory saved by the reduction in particles would be overshadowed by the computational requirement of calculating the Kullback-Liebler distance, especially on the r-one platform.

2.4 Summary

Now that I have described the pertinent multi-robot literature, works in minimalistic sensing, and references for probabilistic robotics the next chapter discusses preliminary information to set the stage for my particle filter algorithm. Following that will be a brief overview or probabilistic robotics with a focus on particle filters. Finally, I will describe my particle filter algorithm and explain the data produced by my algorithm.

Chapter 3

Preliminary Information

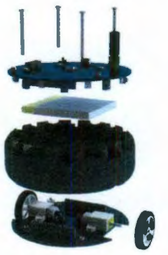
In this chapter I give a brief overview of the r-one multi-robot system built by the Multi-Robot Systems Laboratory at Rice University. I then discuss reciprocal orientation and the different methods to determine reciprocal orientation and their associated bandwidth costs. Finally I will close the chapter with a description of the data collection method.

3.1 The R-one Robot

Figure 3.1(a) shows a fully assembled robot, and Figure 3.1(b) shows the exploded diagram. The sensor suite consists of a 2-axis gyro, 3-axis accelerometer, and 3 visible-light photo resistors. The robot has two motors with quadrature encoders to measure position and velocity. The robot includes 8 IR transmitters, 8 IR receivers, a 2.4 GHz radio with 2Mbps data rate, and a USB port. To interact with the user, the robot has 3 push buttons and 3 arrays of five LEDs each in red, green, and blue.

The robot is controlled by a Texas Instruments Stellaris LM3S8962 microcontroller. The CPU core is an ARM Cortex-M3 running at 50 MHz with 256 KB of Flash memory and 64 KB of SRAM. This particular CPUt limitation must be considered in this work because it places an upper bound on the number of particles the r-one can reasonably create and evolve.

(a) The r-one robot.


(b) Exploded CAD view.

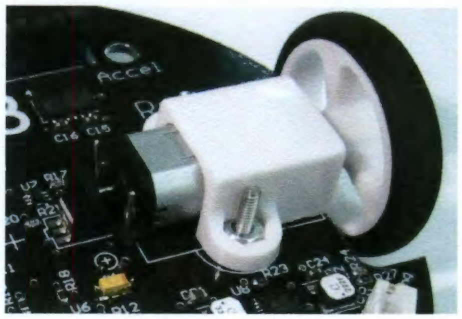

(c) The r-one encoder.

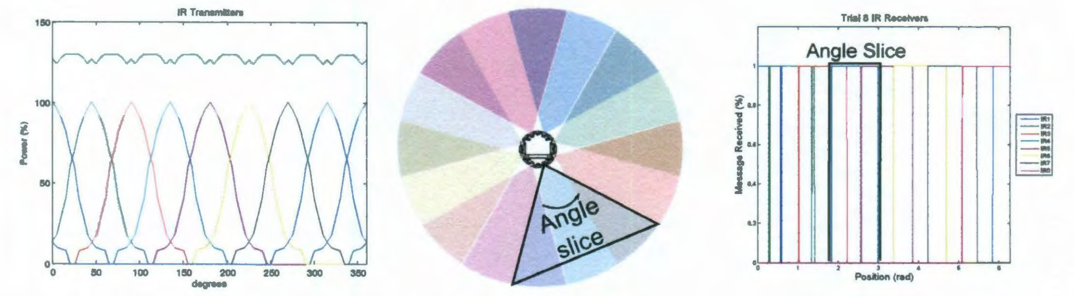
Figure 3.1 : **a:** The r-one robot. **b:** Exploded CAD view of the robot assembly. The robot is composed of two circuit boards bound together with a circular shell and four screws. **c:** The motors and encoders mount directly to the circuit board.

The motors and encoders mount directly to the bottom circuit board, shown in Figure 3.1(c). The encoders have 0.0625 mm/tick linear resolution at the wheel.

The 2.4 GHz radio on the robot can be used for inter-robot communication, but is designed for centralized command and control. The primary means of inter-robot communication is the local IR communication system described in the following section.

3.1.1 Inter-Robot Communication and Localization

Each robot has a set of eight IR transmitters and eight IR receivers. The transmitters transmit in unison, and were designed into the shell to provide a nearly radially-uniform energy emissions pattern. Figure 3.2(a) shows the predicted angular output based on the IR emitter specifications and the design of the plastic shell. This shows a power variation of 4%, but we have not verified this performance on physical hardware yet. Because the communications bandwidth is very limited (see below), we selected a maximum range of around 1.0 meters in order to limit the number of neighbors and



(a) Predicted IR transmitter power output. (b) Top CAD view of IR receiver overlap. (c) Measured IR overlap.

Figure 3.2 : **a:** The transmitter and shell are designed to produce a radially uniform power output. **b:** This is a top view of a CAD model of each IR receiver's detection region. Each receiver detects signals in a 68°arc. These regions overlap to form 16 distinct sectors. A message from a neighboring robot will be received on one or two receivers, and can be processed to determine the direction to within 2.5°. **c:** Experimental verification of the overlap of the receiver regions. The plot is showing the angle each receiver can detect an incoming message. The average width is 68°, which matches the CAD model. The corresponding arc from the top view is highlighted in black.

messages that are received by each robot.

Each robot has eight IR receivers, arranged so that their reception regions overlap as shown in Figure 3.2(b). The shell is designed to limit the detection arc of each receiver to 68°. By noting which receiver(s) detect a neighboring robot, the bearing can be estimated to a resolution of approximately 22.5°. Figure 3.2(c) shows the measured reception arc from each receiver with color corresponding to Figure 3.2(b). The reception arc varies from 63°to 74°with a mean of 68°over 10 trials.

The receivers are standard Sharp IR remote control devices, with 38khz modulation and a maximum bit rate of 1200bps. A simple TDMA scheme is utilized: the robots to transmit at periodic intervals, but with a random offset, similar to the ALOHA protocol [39]. This will limit the effective usable bandwidth before network congestion causes saturation. The protocol is similar to RS232 8N1 and produces a

Message Size (bytes)	Message Size (bits)	Transmit Time (ms)
3	42	33.6
4	51	40.8
5	60	48

Table 3.1 : An illustration of the desired message size, in bytes, the actual number of bits sent and the corresponding transmit time.

message size shown in Table 3.1. My algorithm requires a message size of three bytes, which takes 33.6 milliseconds to transmit. These bandwidth constraints place a limit on robot density and algorithm complexity, but increasing bandwidth would require using more expensive or even custom receivers.

Neighbor Transmission Period (Rounds)

Each robot transmits a message at a fixed periodic interval, meaning that every robot will receive messages from each of its neighbors only once per period. This acts as a *synchronizer* and makes the programming model for the r-one to be a synchronous distributed system from each robots point of view [7]. For this method to work, each r-one has to abide by the rule that each robot has the same neighbor update period. The diagram in Figure 3.3 illustrates this point.

In Figure 3.3 the Δt represents the neighbor update period, or the round. Once a robot sends out their message they wait Δt time before sending out another message. The vertical lines illustrate that based on robot A's initial transmission, no robot communicates more than once during each round. The vertical lines are arbitrary and are merely there to provide a concrete representation of the current round. The vertical lines could have easily been drawn based on any of the three robot's commu-

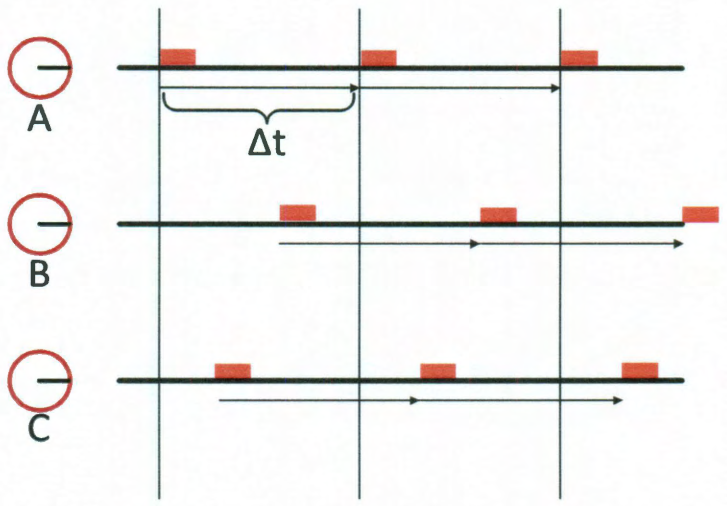


Figure 3.3 : A simple diagram illustrating how the neighbor period update makes the r-one a distributed, synchronous system. Each robot transmits only once during the neighbor period, Δt. Since each robot has the same Δt the system looks like a distributed, synchronous system.

nications, the result would the same; no robot would transmit more than once per round.

There are two major constraints that rounds places on my particle filter algorithm (or any algorithm the r-one is asked to execute). The first constraint deals with the the amount of updates. Because of the rounds, each r-one will only be able to transmit and receive infrequent updates. Essentially, they are not allowed to broadcast information whenever they want to. The second constraint is based on the length of the round. My particle filter algorithm requires a fair amount of computation to create the particles, evolve/weight the particles, and calculate a weighted average of the estimated pose. With this in mind the r-one must be able to accomplish all computations during the current round. If it is not able to complete all of the computations then it will miss a round. Too many missed rounds may cause the complex behavior that is relying on the pose estimate from this algorithm to become unstable or even fail. For a three byte message and four neighbors, the r-one has 250 milliseconds for each round. All computation must be completed in this time.

3.2 Methods to Determine Orientation

As mentioned earlier in this chapter, orientation is the angle measurement from the neighboring robot to the sensing robot from the neighboring robot's point of view. See Figure 3.7 for a graphical representation of orientation.

Currently the r-one is can only measure orientation of neighboring robots using network communication. The first measurement is bearing-only and the other two are methods of determining orientation. The first measurement is bearing-only. This allows the r-one to determine the angle to its neighbor defined by its own local coordinate system. This particular method does not add any additional communications to compute. The information required for this to work is already included in the announcement message that the r-one sends out each communications round which contains its ID, its current translational velocity, and its current rotational velocity. When a neighboring r-one receives an announcement message it records the senders id and the sector that it received the message on, creating a neighbor list that resolves the neighbor's id to the bearing. This method is $O(1)$ since the r-one will only ever send one announcement message regardless of the number of neighbors.

The next method of determining orientation is reciprocal orientation. This particular method requires the most of the r-one in terms of communications complexity. To perform this method the r-one transmits its announcement message and then transmits a series of orientation messages, one for each neighbor. Each message contains the neighbors id and bearing. If a robot receives an orientation message that has its id in it then it knows the neighbors measurement of its bearing from the neighbors point of view, or the reciprocal orientation. The communication complexity of this method is large. The r-one must transmit one announcement message plus an additional message for each of it's neighbors resulting in a communications complexity of $O(1 + \Delta)$ with Δ representing the maximum number of neighbors.

The final method is a type of a Gray code described as a bit pattern overlap method. This method will be researched, implemented and tested during the summer of 2011. Essentially, the r-one will transmit an additional bit of data at the end of its announcement message for each transmitter that it has. The data within the message will collide exactly and will not be affected. The additional bits of data, a single bit representing the particular transmitter the data originated from, will collide constructively allowing the remaining bits to represent the sector. Even though this has a relatively low communications complexity cost, the price is paid in having to put more effort in engineering the hardware to ensure that the messages collide constructively. The communications complexity of this method is $O(s)$ with s representing the number of transmitters the robot has.

3.3 Data Collection

Data collection on multi robot systems requires the user to know the ground truth positions of the robots, or the robot's location in an external coordinate frame. There are many means of determining a robot's global position: GPS [40], a Vicon-like tracking system [3], radio-acoustic ranging [41, 42], or camera-based tracking [43, 44, 45, 46]. However, GPS is unavailable indoors, a Vicon system is expensive, and radio acoustic systems work well, but increase the complexity of each robot.

Camera-based tracking systems are currently the most common low-cost method for providing ground-truth. These systems must have the ability to uniquely identify individual robots in the camera image. Fiducial tracking can find multiple markers, and with initialization, identify unique robots [43]. In a uniform environment, robots can be tracked by color alone as with SwisTrack [44]. Bar code tags such as AprilTags [45] provide unique IDs without initialization, as well as 6-DOF pose estimation. An alternative to the bar code tags is to track IR beacons on each robot [3, 46]. The beacons transmit a pattern unique to each robot. One beacon per robot and one camera allow 2-DOF position to be measured directly. Multiple cameras or beacons can be used for full 6-DOF pose estimation [47].

Ground truth poses of the robots for this thesis are measured by a vision-based localization system called the AprilTag system [45]. Figure 3.4(a) shows the actual AprilTag global localization system that is used for the r-one. The system has three main components: the robots running the experiment, a tripod and boom mounted

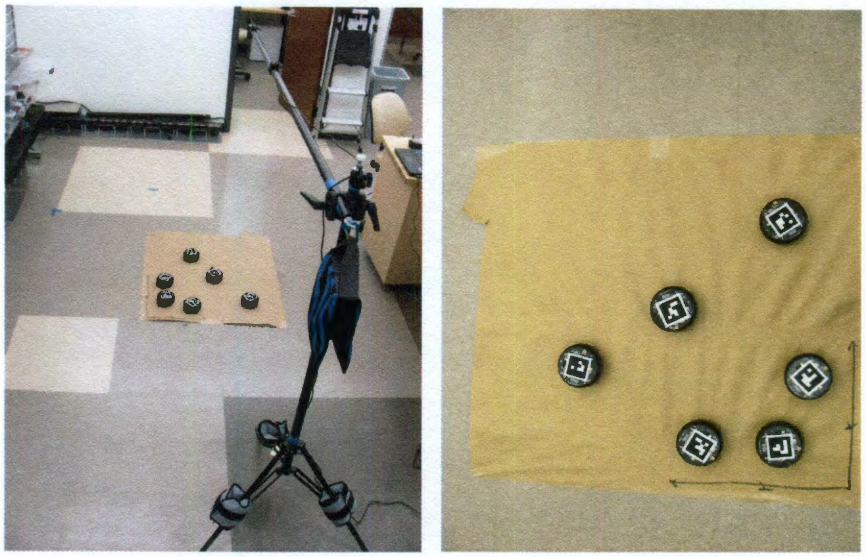

(a) Tripod and Boom. (b) Robots in Camera Space.

Figure 3.4 : **a:** The tripod and boom setup used for collection of location information. **b:** Camera view of rone robots.

digital camera, and a server computer with data logging software. With this system, a camera server collects and displays all ground-truth estimates of individual robot positions based on the IDs of the unique fiducials. This system has been tested and has a mean position error of 6.56*mm* and a mean orientation error of 9.6*mrad*. This error was determined after 583 testing iterations. I then combine the position data with log data from the sensing robot to produce a unified file for analysis.

3.4 Robot Model

Depicted in Figure 3.5 is the robot model for the r-one. two types of velocities indicated for the r-one. The first is translational velocity, or tv. This is the velocity of the robot along the x-axis and it is measured in millimeters per second. The second is rotational velocity, or rv. This is the velocity of the robot as it rotates and it is

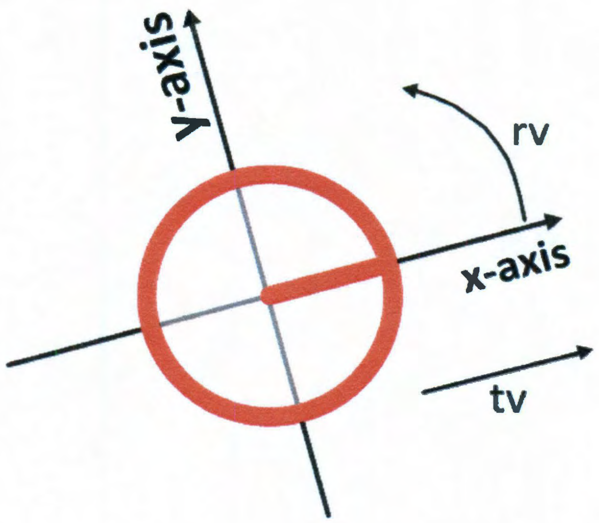


Figure 3.5 : A representation of robot model for the r-one.

measured in milliradians per second. With these two velocities one can depict the r-one moving in a straight line, rotating in place or a combination of the two. As indicated in Equation 3.1 this is the control input, u at time t (u_t) which refers to the tv and rv of the neighbor robot received through the infrared communications system.

$$u_t = (tv_{nbr}, rv_{nbr}) \tag{3.1}$$

3.5 Sensing Model

Figure 3.6 depicts the infrared communications system or the sensor model. This particular system uses the eight infrared transmitters linked together to transmit the robot's messages. The receivers overlap to create 16 distinct *sectors* to measure the bearing with a resolution of approximately $\frac{\pi}{8}$ radians. Initial testing in the lab indicated a binary transition from one sector to the neighboring sector with no overlap.

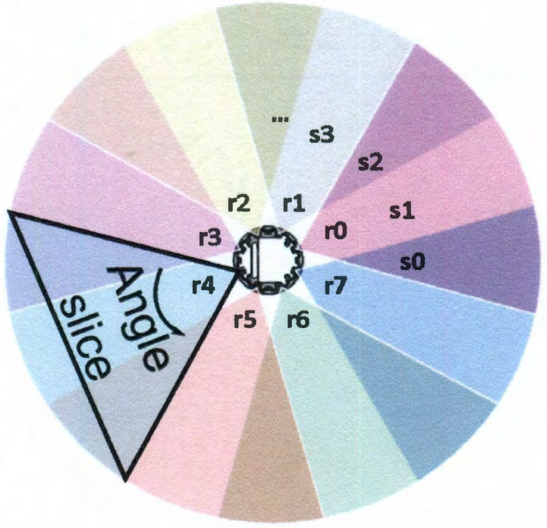


Figure 3.6 : A representation of sensor model for the r-one. The r's represent the eight infrared receivers. The s's represent the 16 bearing sectors.

However, this sensor model will cause a particle filter failure due to the lack of error in the model. Error was added to the sensing model to indicate a 95% probability that a neighbor is actually in the sector that the robot senses it in and 5% probability that the neighbor is in one of the other 15 sectors. Equation 3.2 represents the final sensor input definition:

$$z_t = \begin{cases} S_n & \text{with p(0.95)} \\ S_{\overline{n}} & \text{with p(0.05)} \end{cases} \tag{3.2}$$

3.6 State Model

Pose, in a multi-robot system, is a measurement of the neighboring robot's position in the sensing robot's local coordinate system. In this thesis I define the *sensing robot* as the robot that is attempting to determine the pose of another robot which is referred to as the *neighboring robot.* Additionally, pose represents the *state* model of the system at time t, or x_t.

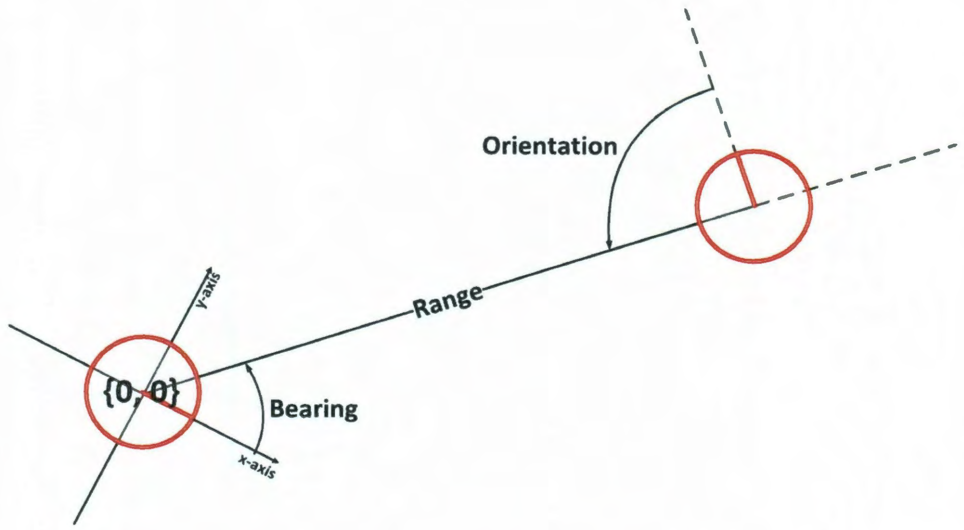


Figure 3.7 : A representation of $(bearing, range, orientation)$ for two robots. The sensing robot is on the left (centered on the origin) and the neighboring robot is on the right.

There are two representations for the pose of a neighboring robot, the primary being $(bearing, range, orientation)$, which is illustrated in Figure 3.7. *Bearing* is the angle measured from the sensing robot to the neighboring robot, or the heading to the neighboring robot. *Range* is the measure of the distance between the center of the sensing robot and the center of the neighboring robot. Finally, *orientation* is the angle measurement from the neighboring robot back to the sensing robot. Note that the bearing of the neighboring robot from the sensing robot is the same as the orientation of the sensing robot to the neighboring robot.

Pose can also represented as (x, y, θ), illustrated in Figure 3.8. This is the method that I use in my particle filter. With this method the (x_{nbr}, y_{nbr}) values are the location of the neighboring robot in the sensing robot's coordinate system. The θ_{nbr} is the *heading* of the neighboring robot in the sensing robot's coordinate frame.

The pose representation depicts the state model for the r-one multi-robot system,

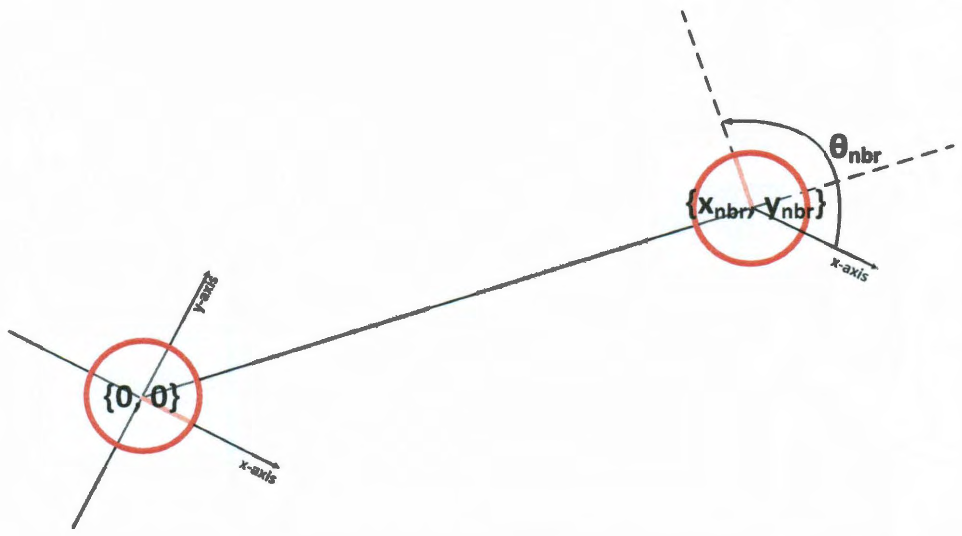


Figure 3.8 : An alternate representation of (x, y, θ) for two robots. The sensing robots is on the left (centered on the origin) and the neighboring robot is on the right.

illustrated in Figure 3.8. The *state* at time t is represented as x_t. Equation 3.3 represents the state model:

$$State = x_t = (x_{nbr}, y_{nbr}, \theta_{nbr}) \tag{3.3}$$

Chapter 4

State Estimation

State estimation refers to a robots attempt to determine some hidden state based on sensor data. The robot has to estimate its state by inferring quantities from its sensor data because these sensors often cannot directly measure the state variables. Additionally, the robot's sensors only carry partial information about the quantities being measured and the noise of the sensor further compounds the issue. The goal of state estimation is to attempt to recover state variables from this noisy information. In my case, I wish to estimate the pose, $(x_{nbr}, y_{nbr}, \theta_{nbr})$, of a neighboring robot. Furthermore, our sensor model is nonparametric which precludes many common state estimation techniques.

4.1 Probabalistic Robotics

The primary concept in probabilistic robotics is that of *belief*, which represents a robots estimate about the state of the environment. A robot may exist at a known pose, referred to as its actual pose, in a global coordinate system but that pose can never be measured directly, even with precise sensors. Instead of measuring the pose directly the robot must infer its pose, referred to as the estimated pose, from its sensor data.

Belief is represented as a posterior probability, which is the probability of the current state variables conditioned on the sensor inputs. This is updated in two steps. First, one calculates the belief at time t after a control input but before receiving the sensor information. This is referred to as $\overline{bel}$, and is portrayed in equation 4.1. It represents the state, x, at time t given the control input, u, and the sensor input, z, at time $t-1$.

$$\overline{bel}(x_t) = p(x_t | z_{1:t-1}, u_{1:t}) \tag{4.1}$$

Belief, as indicated in Equation 4.2, represents the state at time t given the entire history of the control inputs and all of the current sensor inputs.

$$bel(x_t) = p(x_t | z_{1:t}, u_{1:t}) \tag{4.2}$$

Probabilistic robotics rely on the Markov assumption which states that the past and future data are conditionally independent as long as the current belief of the system, x_t, is known. Essentially since the current belief only depends on the previous belief (x_{t-1}), all other states can now be disregarded and the recursion stopped after the previous state is accounted for. This reduces the above equations to the following:

$$\overline{bel}(x_t) = p(x_t | u_t, x_{t-1}) \tag{4.3}$$

$$bel(x_t) = p(x_t | z_t, x_{t-1}) \tag{4.4}$$

4.1.1 Bayes Filter Algorithm

The Bayes filter algorithm is the most general algorithm for calculating the belief. This algorithm recursively calculates the belief bel at time t, and is shown in Algorithm 1. There are three initial states to this algorithm. If the robot has knowledge of the initial state of the system, x_0, then $bel(x_0)$ is initialized with a point mass distribution about x_0. However, if the initial state of the system is unknown, $bel(x_0)$

is initialized with a uniform distribution over the entire state space. Additionally, partial knowledge of $bel(x_0)$ can be expressed as a non-uniform distribution.

```
1 Algorithm Bayes filter( $bel(x_{t-1}), u_t, z_t$ ):
2 for all $x_t$ do
3     $\overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1}$;
4     $bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)$;
5 end
6 return $bel(x_t)$
```

Algorithm 1: A generalized form of the Bayes filter algorithm.

This algorithm contains a fundamental equation which defines the Bayes filter:

$$bel(x_t) = \eta p(z_t|x_t) \int p(x_t|u_t, x_{t-1})bel(x_{t-1})dx_{t-1} \tag{4.5}$$

where η is a normalizing constant, z_t is the sensor measurement update and u_t is the control update. This belief update equation can only be implemented in the strictest of cases and quickly becomes intractable as the cases become more generalized. The intractability is a result of attempting to compute the integral for a *probability distribution function.* However, this algorithm has lead to several other techniques that sidestep this intractability such as Kalman filters, extended Kalman filters, particle filters and many more.

4.2 Particle Filter

A particle filter is a variant of the Bayes filter describe in Section 4.1.1 and like the Bayes filter it recursively estimates the posterior belief, *bel*, over the state of the system x_t. However, instead of attempting to represent the probability distribution function in its entirety it utilizes a sample-based approach. This approach is approximate and nonparametric, allowing it to represent distributions other than Gaussian that could also potentially be multi-modal. It uses these samples to track hidden state variables as they evolve over time. These samples are referred to as *particles* and represent the variable of interest by associating each particle with a weight that indicates the importance of that particle, or how "good" of an estimate the particle depicts. The estimated state or the variables of interest, $bel(x_t)$ can be as simple as a weighted average of all of the particles.

There are two main steps in a particle filter algorithm: prediction and update. In the prediction step each particle is modified according to the robot model with velocity information from the neighboring robot, to include the addition of random noise. During the update step each particle's weight is updated according to the current sensor information. Particles with small weights are discarded through a technique referred to as resampling, described in detail in Section 4.2.1 below. The current estimate of the variable of interest can be calculated in three ways. First, the weighted mean can be computed; second, the particle with the highest weight can become the estimate and, third, the weighted mean inside a small window surrounding the particle with the highest weight (also called the robust mean). Each of these estimation methods have pros and cons: a weighted mean fails when there are multi-model distributions, using the particle with the highest weight introduces a discretization error, and the robust mean is computationally expensive (although it is the best method).

4.2.1 Resampling

A major issue that arises as the particle filter progresses is the depletion of the particle population. As the particles evolve, several particles may drift far enough from the sensor input for their weight to become negligible and have no effect on the probability

```
Algorithm Particle filter($x_{t-1}, u_t, z_t$):
  $\overline{\chi}_t = \chi_t = \emptyset$
  for $m = 1$ to $M$ do
      $sample x_t^m \sim p(x_t \mid u_t, x_{t-1}^m)$
      $w_t^m = p(z_t \mid x_t^m)$
      $\overline{\chi}_t = \overline{\chi}_t + \langle x_t^m, w_t^m \rangle$
  end
  for $m = 1$ to $M$ do
      draw $i$ with probability $\alpha\ w_t^i$
      add $x_t^m$ to $\chi_t$
  end
  return $\chi_t$
```

Algorithm 2: A basic implementation of a particle filter algorithm [30].

distribution function of the state variables you are tracking. The estimated sample size, ESS$_t$, is a hueristic developed by Liu et al. [48] that describes the number of near-zero-weight particles that reside in the distribution. These equations, 4.6 and 4.6 details the coefficient of variation of the weights of each particle. ESS can be used as a threshold to indicate when to to resample. When the ESS rises above a certain threshold the particle population is resampled and the particles with low weights are dropped in favor of replicating the particles with the higher weights.

$$cv_t^2 = \frac{var(w_t(i))}{E^2(w_t(i))} = \frac{1}{M}\sum_{i=1}^{M}(Mw(i) - 1)^2 \tag{4.6}$$

$$ESS_t = \frac{M}{1 + cv_t^2} \tag{4.7}$$

Resampling Techniques

There are three commonly used algorithms to resample particles. They are *Select with Replacement*, *Linear Time Resampling*, and Resampling by Liu et al.. In every algorithm the input is an array of the weights of the particles and the output is an array of the indices that indicate the particles that are going to be propagated forward. Select with Replacement is the simplest resampling algorithm and tests have shown no reasonable increase in performance of the other algorithms over this simple algorithm which is precisely the reason why I decided on this algorithm for my particle filter. [31].

In the Select with method, each particle is selected to propagate forward with a probability proportional to its weight; the higher the weight of the particle the better the chance it has of being propagated forward. Another method proposed by Carpenter et al. is Linear Time Resampling. In this method a sorted random number sequence is generated in linear time by manipulating the negative logarithm of N random numbers, after that the algorithm is the same as Select with Replacement [37]. The third algorithm is Resampling by Liu et al. put forth in their paper on Sequential Importance Sampling. This technique uses a function of the weights of the particles $a_j = f(w_j)$ to determine which particles are propagated forward. If a_j is greater than or equal to one then k copies of that particle are propagated forward ($k = a_j$). If the output is less than one the particle survives with a probability equal to a_j [34].

4.3 Limitations of a Multi Robot System on State Estimation

The initial limitation of the r-one on state estimation is the low-cost factor. This particular limitation is the main constraint which affects all of the other limitations. In an effort to produce a multi- robot system that is economically viable for any type of institution or research environment, design decisions were made that placed limits on the hardware procured for the build (see Section 3.1).

One particular limitation of the r-one on state estimation is a direct result of the choice of microcontroller. As stated in 3.1 the rone is controlled by a Texas Instruments Stellaris LM3S8962 microcontroller. While this microcontroller has plenty of I/O it does not have a floating point unit (FPU) which would allow it to perform floating point calculations. The addition of a microcontroller that includes an FPU would add to the overall cost of each robot, defeating the goal of maintaining the r-one as a cheap, yet robust, multi-robot platform. Also, the decision to use the LM3S8962 was based on the fact that it was an "system on a chip" solution that did not require external I/O controls or external memory, thus reducing the overall system complexity.

Another limitation of the r-one on state estimation is the limited amount of bandwidth for the IR communication system. Refer to 3.1.1 for the more information about the IR system in the r-one. With the current protocol the typical message size is 60-bits for a 5-byte message payload, which takes 48 milliseconds to transmit. With a neighbor period of 1000 ms each robot can effectively maintain a neighbor list that contains 10 neighbors, any more than that and your IR network becomes saturated due to communication collisions.

Chapter 5

Simulation Results

To test the effectiveness of my particle filter algorithm, without needing to implement the algorithm on the actual r-one hardware, I needed a simple r-one simulator. It is this requirement that lead me to take an existing simulator and adapt it to my needs. This simulator effectively incorporates the state model, robot model and sensing model put forth in Chapter 3.

5.1 Rone Simulator

The current r-one simulator is written in Java with the Model-View-Controller architecture. It was originally coded by Elizabeth Fudge and Siegfried Bilstein; both undergraduate students in the Multi-Robot Systems Lab at Rice University. I took this existing framework and extended it to simulate my particle filter.

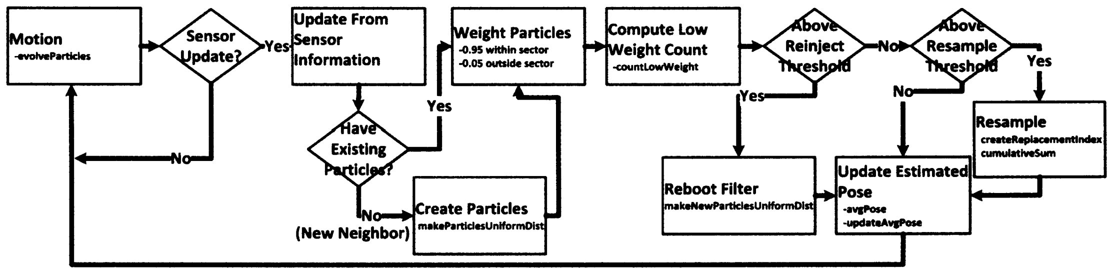


Figure 5.1 : Flowchart of the particle filter algorithm implemented in Java.

5.1.1 Particle Estimator Robot Class

The ParticleEstimatorRobot class inherits all of the methods from the RobotVertex class and adds two specific methods for the particle filter. The first method developed is the *updateSensorInformation.* This method does not need any input and the return type is *public void.* The first step in this method is to look at the list of current neighbors and as long as this is populated it completes the following steps for each neighbor in the list. First it calls the parallel method (*updateSensorInformation*) contained in the ParticleRobotNeighbor class, and discussed below. It then determines the number of low weight particles for the current collection of particles by calling the countLowWeight method contained in the ParticleRobotNeighbor class. A check is made to see if the low weight count of the particles is below two predetermined thresholds. If it is below the resample threshold then the then the particles are resampled by calling the resampleParticles method in the ParticleRobotNeighbor class. If it is below the reinject threshold then the particles are reinjected by calling the makeNewParticlesUniformDist method. The final action this method completes is to call the updateAvgPose method in the ParticleRobotNeighbor class. This provides the latest estimate of the pose of the robot's neighbor by computing a weighted average across all of the particles.

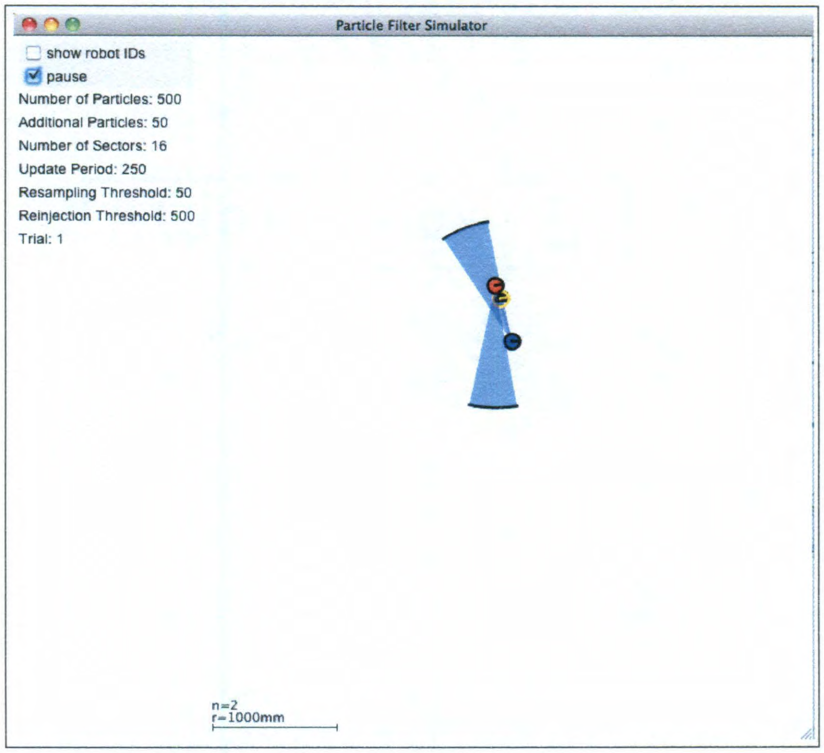
The second method located within this class is the updateNeighborParticles method. While there is a neighbor in the neighbor list this method calls evolveParticles in the ParticleRobotNeighbor class, which will be described in detail below. However, it provides the motion to the particles based on the transmitted tv and rv values of the neighbor.

5.1.2 Particle Robot Neighbor Class

The first important method in this class is the createReplacementIndex method. This method is the critical requirement of the resampling portion of the particle filter and it returns an array of integers which represents the indices of the particles that will to be propagated forward. This method is based on the Select with Replacement algorithm in Rekleitis' technical report [31]. Refer to Section 4.2.1 for more information about this resampling technique.

Another method located in this class is the resampleParticles method. This method starts out by calling the createReplacementIndex discussed above. It then uses the output array of indices to create a new set of particles to be propagated forward. Then the reinjection of new particles based on the current sensor measurement occurs by calling the makeNewParticlesUniformDist method which simply replaces existing particles with the lowest weight with a predeterimined amount of new particles based on the current sensor measurement. Finally, the weights of the particles are normalized which is a requirement for the *Effective Sample Size* heuristic to function correctly.

Finally, the last major method in this class is the updateSensorInformation method. The first step of this method is to determine which sector the neighbor is in, which allows the sensing robot to determine sector changes. Next the method weights the particles using the sensor model giving a particle in the sector a weight of 0.95 and a particle out of the sector a weight of 0.05.



Number of Particles: 500
Additional Particles: 50
Number of Sectors: 16
Update Period: 250
Resampling Threshold: 50
Reinjection Threshold: 500
Trial: 1

Figure 5.2 : A screenshot of the Java particle filter simulator in motion.

5.2 Metrics

In measuring the effectiveness of my particle filter I am concerned with two metrics: speed and accuracy. Accuracy is determined by calculating the difference between the ground-truth pose and the estimated pose produced by the particle filter. I split the accuracy measurement into two parts. The first part measures the linear distance between the (x, y) of the actual and estimated pose based on the sensing robot's coordinate system. The second part measured the difference between the θ of the estimated pose and the θ of the ground-truth pose, again based on the sensing robot's coordinate system. The second metric measured was the time it took the estimated pose to converge with the ground-truth pose. Once the estimated pose was with 10% of the communications radius of the r-one, approximately one meter, I declared the estimated pose to be converged and measured the current time. I denote this convergence on the plots with a black, horizontal line at 0.1 meters and a black vertical line denoting the time of convergence in milliseconds. One item to note is that the system may seem like it converges initially but then the error grows. During this initial period the system does not converge, the illusion of convergence is simply a byproduct of the initial placement of the neighboring robot. This is discussed in detail in Section 5.5.3. Additionally, the longer that the neighboring robot stays within the sensing robot's sector the estimated pose may diverge from the actual pose. This is due to the amount of information that can be garnered from motion updates versus sensor updates and is discussed in greater detail in Section 5.5.2.

5.3 Simulation Setup

A proper assessment of the effectiveness of my particle filter implementation required a baseline data set. This data set was the result of what I refer to as the "vanilla" particle filter implementation. This particular variation did not include any resampling or reinjection of new particles and used a simple multiplicative weighting scheme. Fox, Burgard and Thrun actually refer to this particular implementation as a "usually inferior" implementation of a particle filter [30].

I initialized this implementation with 2,000 particles and allowed those particles to evolve, be weighted according to the sensor measurement, and calculate the weighted average of those particles to produce the estimated pose. To determine the algorithm's sensitivity to different parameters I varied three key design parameters. The first parameter is the number of particles the particle filter is initialized with. I start with a population of 125 particles and then doubled it each iteration until I reached 4000 particles. The second parameter is the update period for the sensors in milliseconds. I started with an update period of 50 milliseconds and doubled it until I reached an update period of 400 milliseconds. The final parameter is the number of sectors that the robot uses to sense other robots with, this dictates the resolution of the bearing measurement for the r-one. I started with four sectors and doubled the number of sectors until I reached 64 sectors. However, the remaining variables for each iteration remained in a default configuration consisting of the following values:

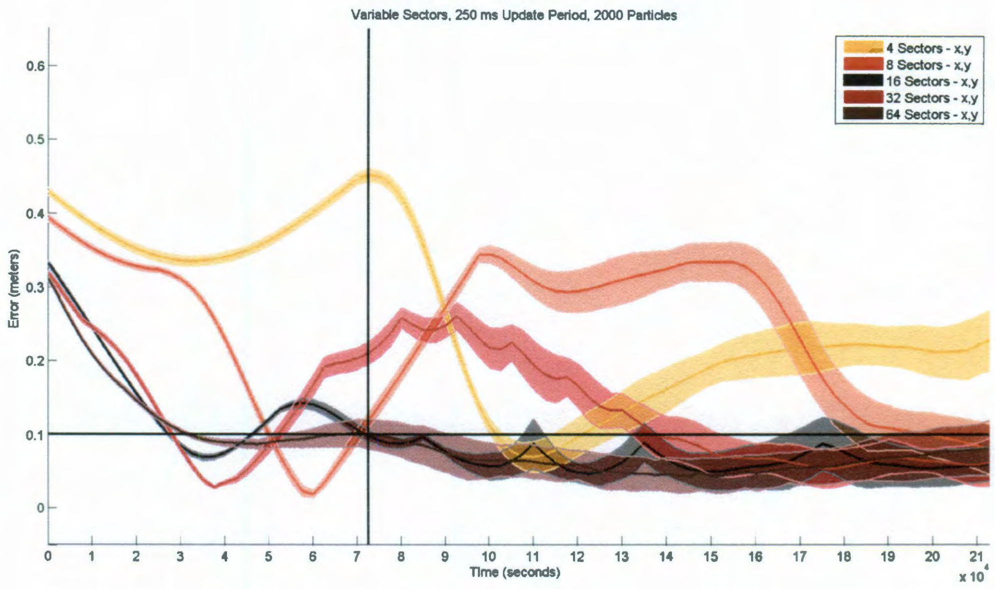
- Number of Sectors: 16
- Number of Particles: 2000
- Update Period: 250 milliseconds
- Resampling Threshold: 10% of the Number of Particles
- Reinjection Threshold: 100% of the Number of Particles

Furthermore, to collect enough data to collate and analyze I ran each iteration a total of 12 times. The data collected included the estimated pose and the actual pose alongside descriptor data that allows me to parse the data and sort it accordingly. To accomplish this I added a section in the code that captured all of the variables and output them to a comma-separated values file which was then subsequently manipulated using a Python script. This data was then entered into Matlab to produce the plots.

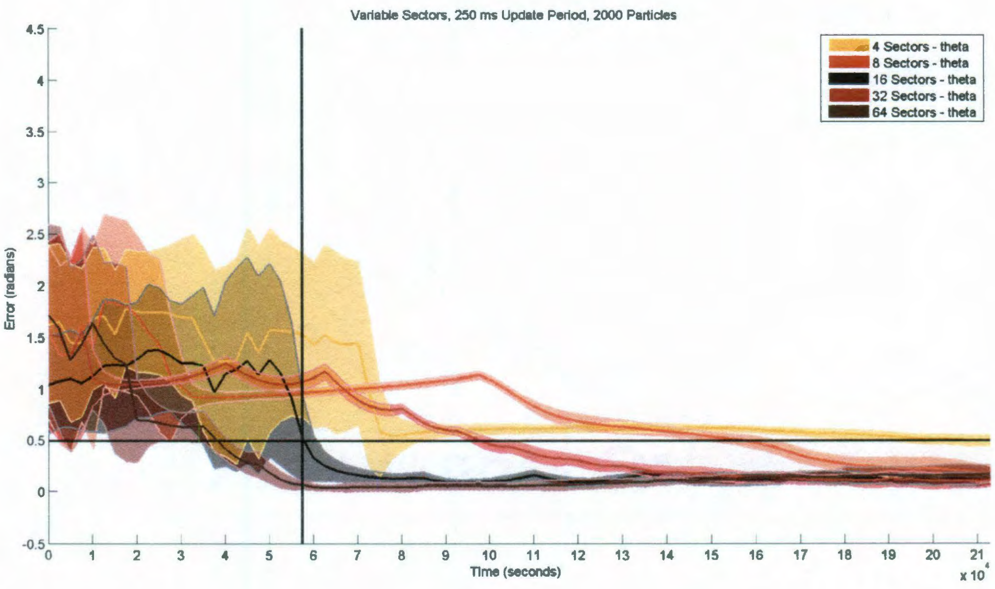
I accommodated varying all of these parameters through the work smarter and not harder idea of automation. To accomplish this I wrapped the main loop of the simulator with three different while loops, one for each parameter I wanted to test. This allowed me the freedom to hit run once and walk away from the simulator and allow it to output all of the files I needed.

5.4 Simple Particle Filter Implementation Data and Results

At first glance this simple implementation appears to allow the r-one to determine the range to its neighbor much more efficiently than my final particle filter implementation. This is supported by Figures 5.3(a) - 5.5(b) which show that the overall error for both the (x, y) position and θ declining over time. It would seem that this is the solution to our range problem. However, I believe that this particular implementation is not robust enough to quickly handle the r-one making course changes or other dynamic movements. The particles are only created when the neighbor is first sensed and those particles are never resampled nor are new particles reinjected, instead all of the original particles are allowed to evolve until the neighbor is no longer sensed. If the neighbor were to make a 90° rotation and continue to move forward the estimated pose would correct only a little but not enough to allow the estimated

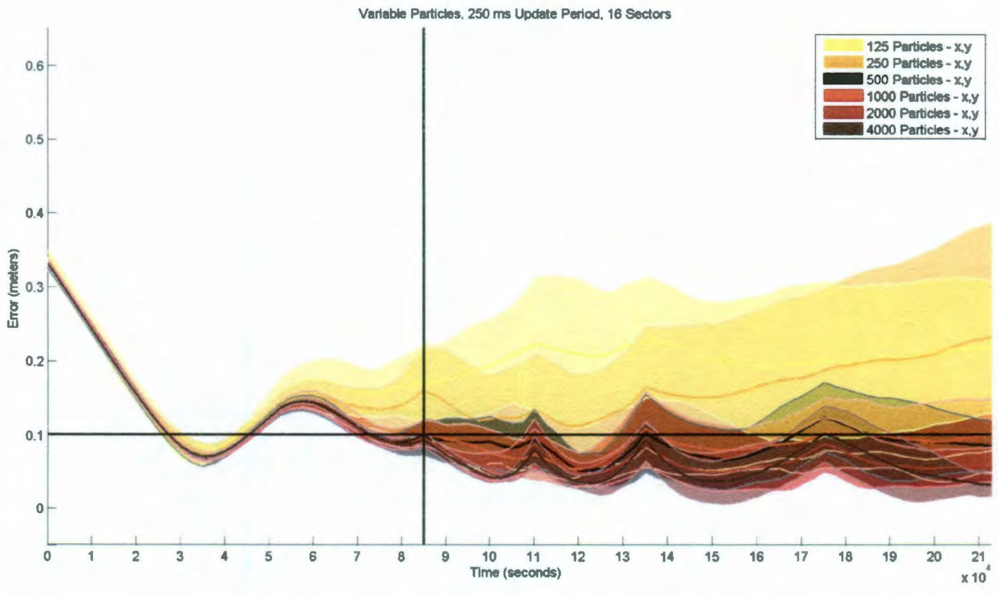


(a) X,Y Error for Sectors.

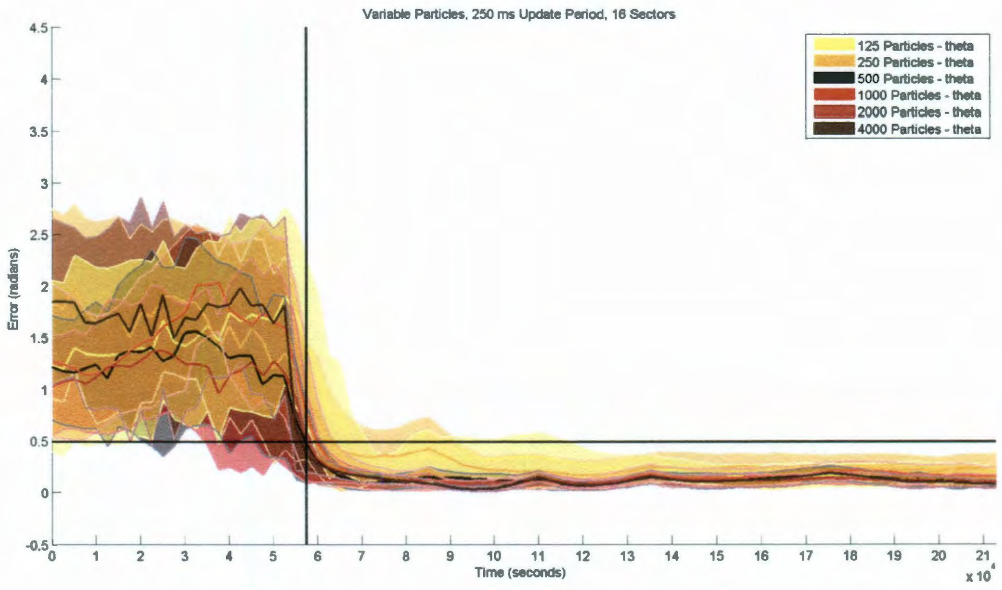


(b) Theta Error for Sectors.

Figure 5.3 : Vanilla particle filter standard deviation for both (x,y) and θ while varying the number of sectors.

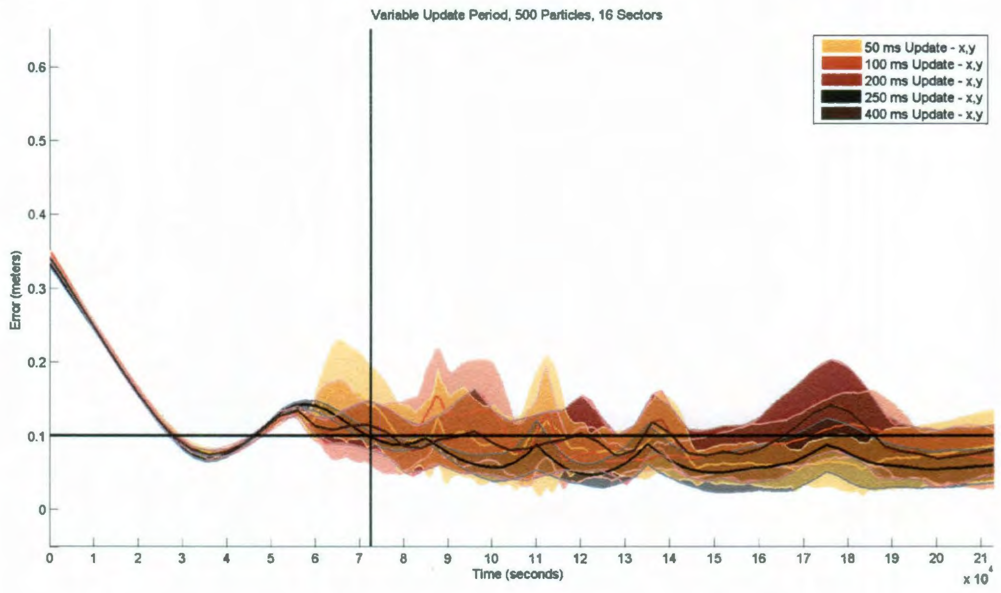


(a) X,Y Error for Particles.

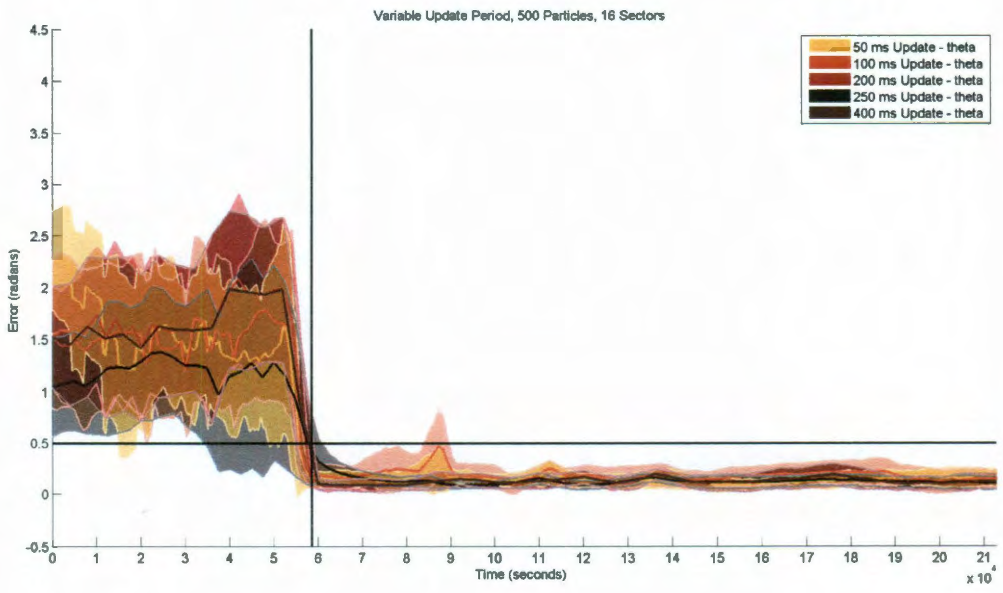


(b) Theta Error for Particles.

Figure 5.4 : Vanilla particle filter standard deviation for both (x,y) and θ while varying the number of particles.



(a) X,Y Error for Update Period.



(b) Theta Error for Update Period.

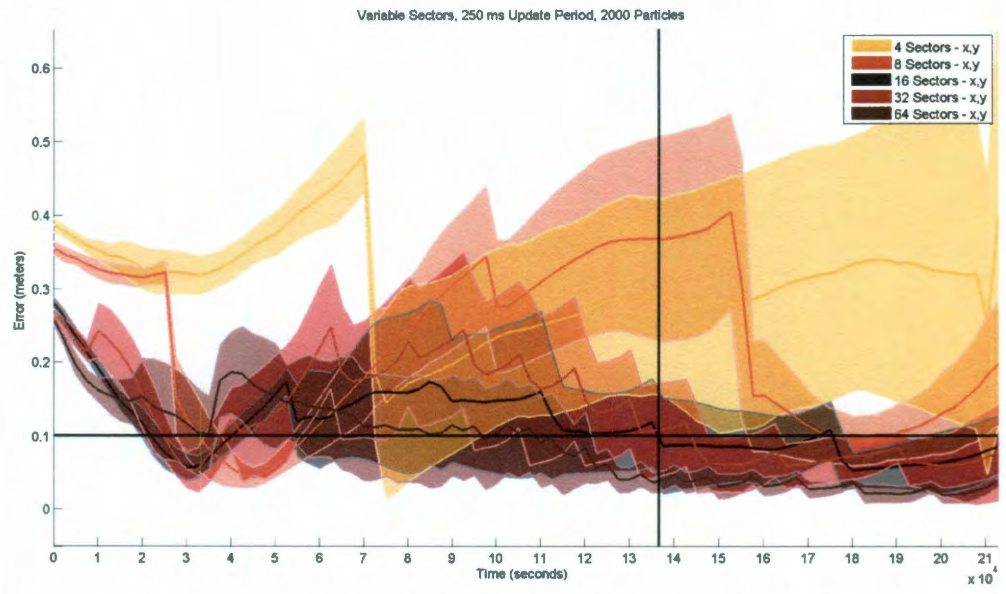
Figure 5.5 : Vanilla particle filter standard deviation for both (x,y) and θ while varying the update period.

pose to represent the actual pose.

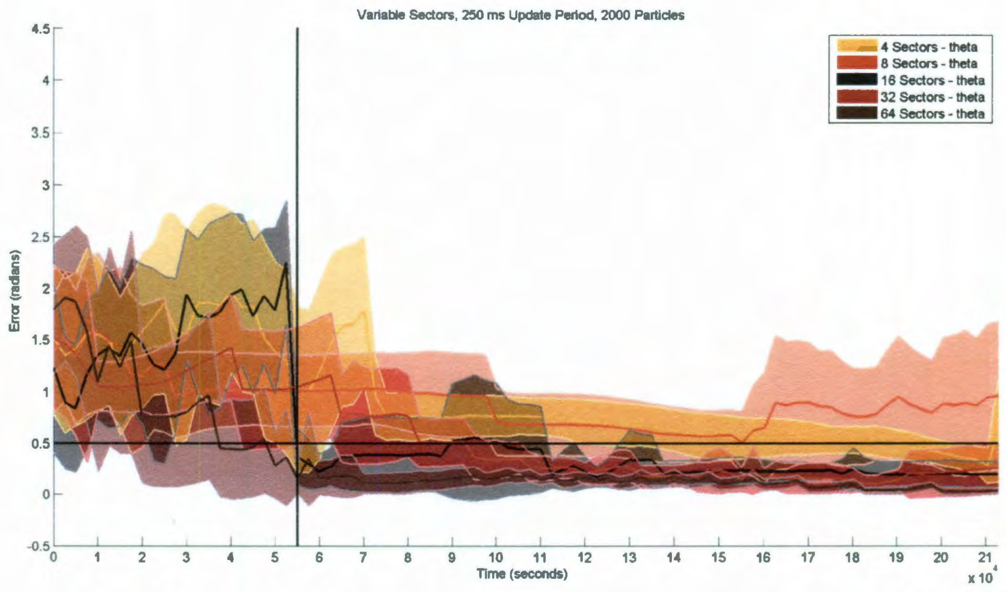
5.5 Final Particle Filter Implementation and Results

The plots from the final particle filter implementation are much noisier than their "vanilla" counterparts. However, looking beyond all of the noise it is clear to see that the error for both (x, y) and θ decline over time. Additionally, this implementation will be able to handle dynamic motion, unlike the simple implementation. The theta plots indicate that the error in theta is invariant with all of the parameters tested. The estimated theta reasonably converges between 5 and 6 seconds for most of the tests.

The first two plots display the positional and bearing errors while varying the number of sectors that the r-one uses to sense neighbors. The reason for this is twofold: first, it is to see if the design choice of 16 sectors was a reasonable one and to see if the particle filter reacts better with a different number of sectors, possibly warranting a design change. For the four sector case the particle filter performed horribly. This is the result of three factors. First, the number of particles, 500 in this case, is not enough to reasonably represent the probability distribution function. Second the greatest amount of information is gleaned during a sector transition and for the four sector scenario there are only two sector transitions. Third, the longer the r-one stays within a sector the more the belief spreads out without any new information to act on. The eight sector scenario performs marginally better than the four sector scenario. The 16 and 32 sector scenarios performed reasonably well and were closely matched in performance. The final scenario, that of 64 sectors, performed the best out of all the scenarios. Again, this falls in line with I mentioned earlier about the four sector scenario, only it is the reciprocal of those points. Since

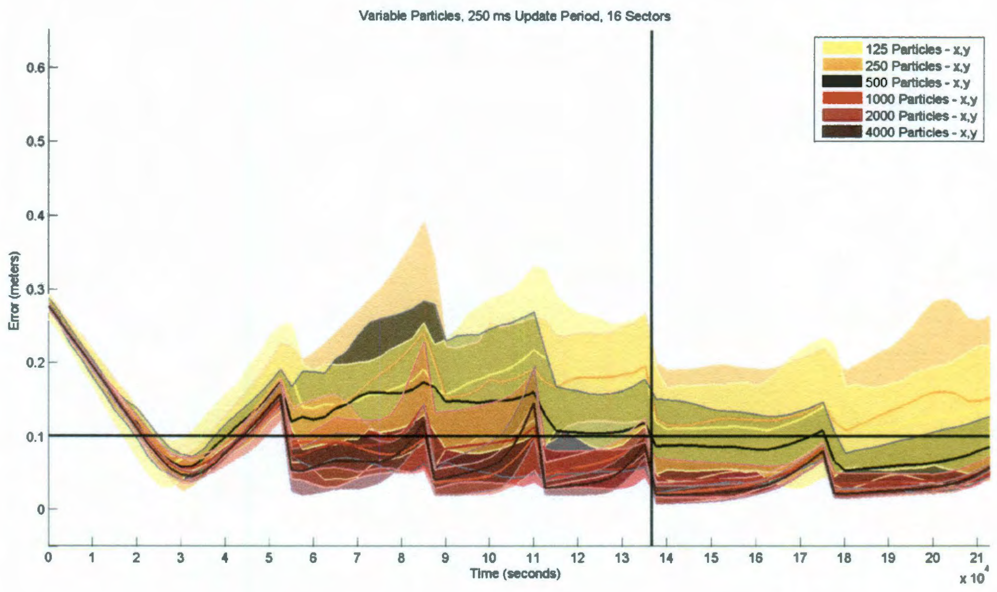


(a) X,Y Error for Sectors with Resampling.

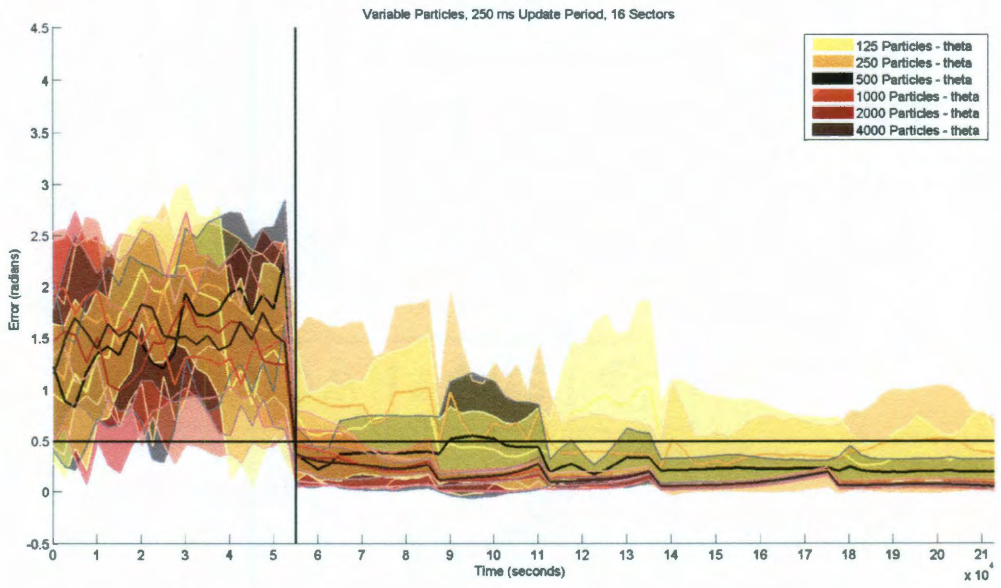


(b) Theta Error for Sectors with Resampling.

Figure 5.6 : Optimized particle filter standard deviation for both (x,y) and θ while varying the number of sectors.

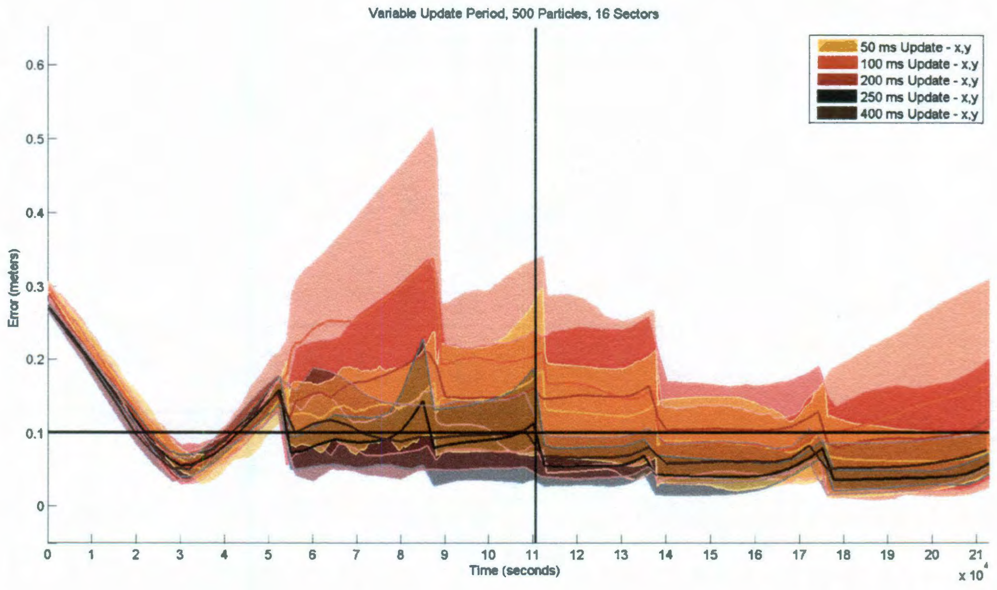


(a) X,Y Error for Particles Resampling.

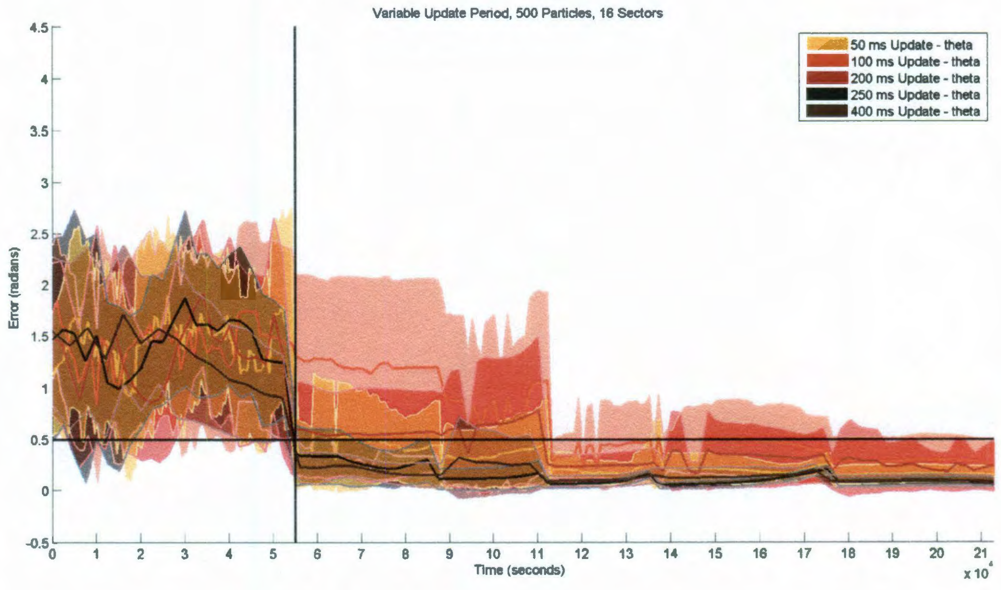


(b) Theta Error for Particles Resampling.

Figure 5.7 : Optimized particle filter standard deviation for both (x,y) and θ while varying the number of particles.



(a) X,Y Error for Update Period Resampling.



(b) Theta Error for Update Period Resampling.

Figure 5.8 : Optimized particle filter standard deviation for both (x,y) and θ while varying the update period.

there are several more sector transitions there is more information being provided to the particle filter. Also, the time spent in each sector is minimal which does not allow the belief to spread out very much at all. Finally, the number of particles is more than adequate to reasonably portray the *pdf* since the overall area of the sector is much smaller.

The second plots display the positional and bearing errors while varying the number of particles used to bootstrap the particle filter. As I mentioned earlier, if you do not have enough particles to represent the *pdf* then the particle filter will find it difficult, if not impossible, to estimate the hidden state variables you would like it to ascertain. This is supported with the plots; as the number of particles go up the time to converge, the standard deviation, and the amount of divergence between sector transitions all decline. The sweet spot seems to be right at 1,000 particles for this system. However, due to the memory and computational constraints on the r-one I decided to use 500 particles for my filter and to tune all of my parameters accordingly since the 500 particle case allows for a reasonable convergence time.

The third set of plots display the positional and bearing errors while varying the update period of the particle filter. This period determines when the r-one will receive information from the infrared communications system. With this information it can then update the weights of the particles and resample or reinject if it is required. The plots indicate that the r-one does not require a longer update period. In fact the r-one is insensitive to changes in the update period. is at its peak. The convergence time among all of the different update periods are similar. It is because of this fact that the design decision to utilize settle on a 250 millisecond update period for the r-one is reinforced.

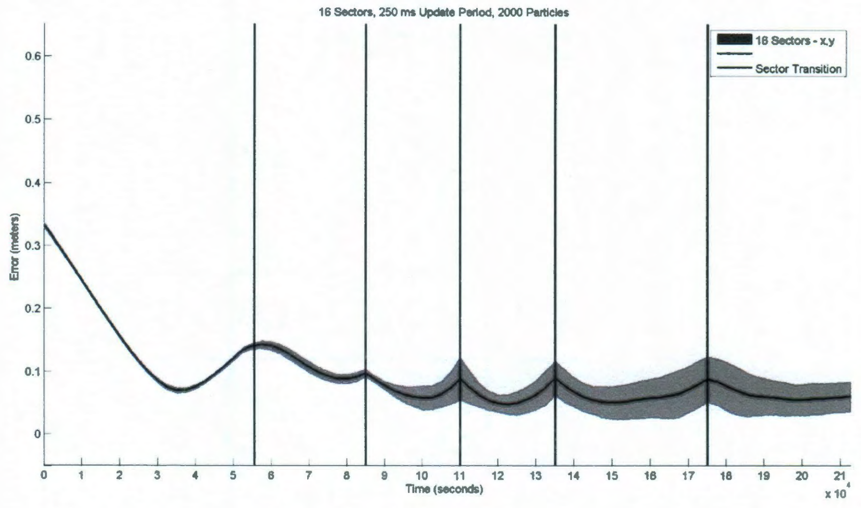
5.5.1 The Effect of ESS on the Final Implementation

Even though the ESS heuristic first discussed in Section 2.3.2 works for an interim implementation of my particle filter (one with multiplicative weighting and resampling), it fails miserably when I add in the actual weighting scheme based on the sensor model portrayed in Section 3.5. As pointed out earlier, this sensor model does not produce a typical probability distribution function. Instead, it is binary in nature by only producing two weights. In fact Liu goes on to state that if the weights of particles are approximately similar then resampling only serves to decrease the efficiency of the sampled representation [49]. This resulted in the development of a simplified method to determine when to resample. Instead of relying on the ESS heuristic I simply count the number of low weight particles in the current particle distribution. If that number rises above a preset threshold then I have the particle filter algorithm resample or reinject.

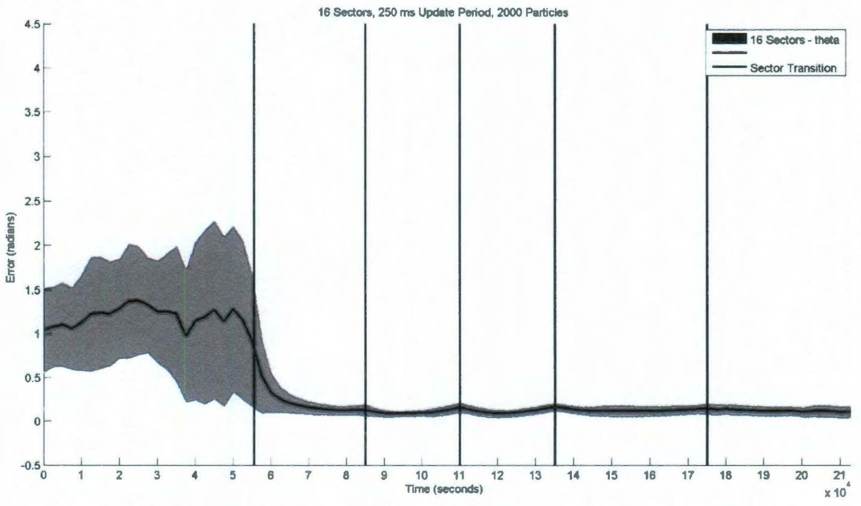
5.5.2 Impact of Sector Changes

To better understand the impact that the sector transitions had on my particle filter I decided to simplify two of my earlier plots by plotting the 16 sector data and adding visual indicators where the sector transitions occurred. Figures 5.9(a) and 5.9(b) is the result of this simplification. The accuracy of the transition indicators is coarse because they are the result of logging the time when the neighboring robot crosses into an adjacent sector in a file separate from the other data. However, the placement transition indicators need not be in the exact location because even the coarse locations correlate very well with the effect that sector transitions are having on the mean.

One major takeaway from this particular figure set is that the longer that a neigh-



(a) X,Y error with sector transitions.



(b) Theta error with sector transitions.

Figure 5.9 : Error between sector transitions for 2,000 particles, 16 sectors, and a 250 millisecond update period.

boring robot stays in one of the sensing robot's sectors the worse its estimate of its pose gets. This is because the most information gleaned from this particle filter is right at a sector transition. The particles that make this sector transition at approximately the same time the robot's sensors detect a change in sector will continue to maintain their high weight while all other particles will be reduced to the lower weight. You can notice that the the error drops quickly immediately following a sector transition and reaches a minimal point while the robot traverses the remainder of that sector only to start increasing again. Again, this is because the longer a robot stays within a sector the more that the probability distribution function representing the *belief* spreads out. This results in the estimated pose drifting away from the actual pose.

5.5.3 Poor Experiment Design

After reviewing the all of the plots I noticed an anomaly where the error quickly gets smaller and the estimated position almost converges on the actual position before getting large again. To further investigate, I drew the simplified plot with 16 sectors, a 250 millisecond update period, and 2,000 particles on the white board. I then added the sector transition lines to the plot. Finally, directly below the plot, I sketched the actual path of the robot and the estimated path of the robot. The red robot and dashed line represents the actual robot location and the path it takes. The green robot and dashed line represents the estimated robot location and determined by the particle filter.

Looking at this hybrid plot it is clear to see that the initial reduction in error and subsequent increase in error is an artifact of the estimated position crossing the path of the actual robot. This makes it appear that the estimated pose converges with

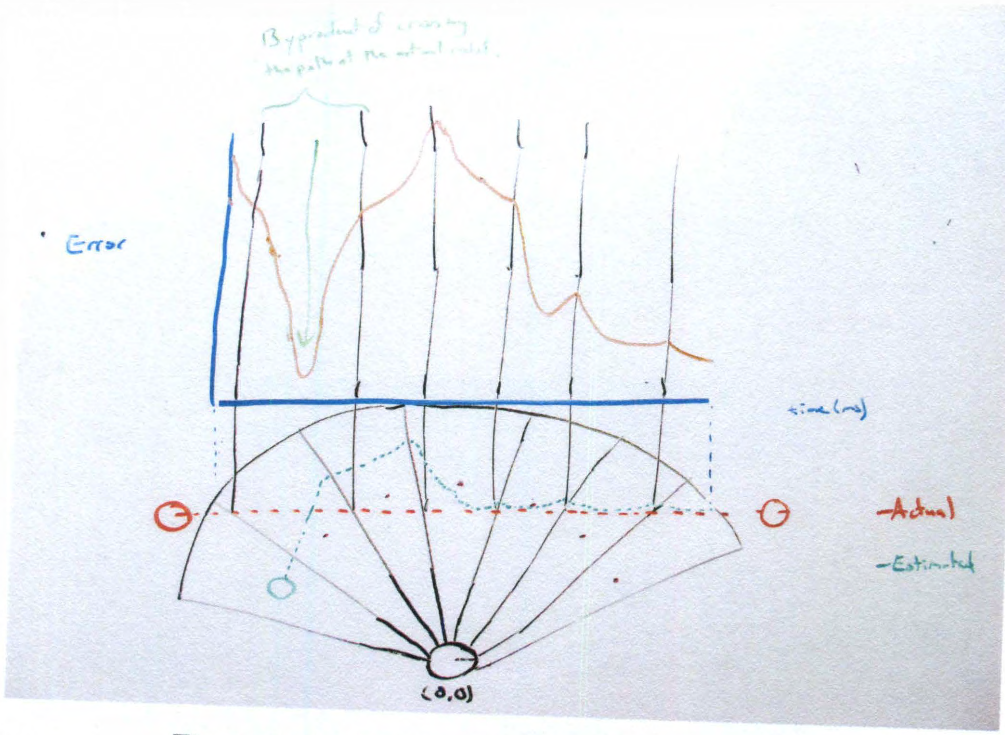
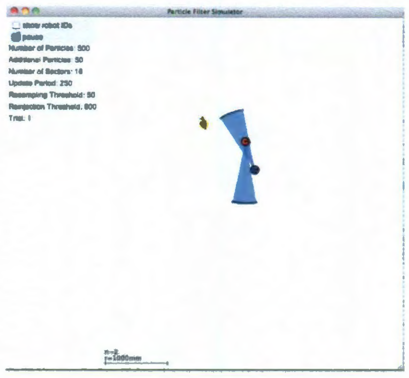


Figure 5.10 : Recognizing bad experiment design.



Particle Filter Simulator

show robot IDs

pause

Number of Particles: 500

Additional Particles: 50

Number of Sectors: 16

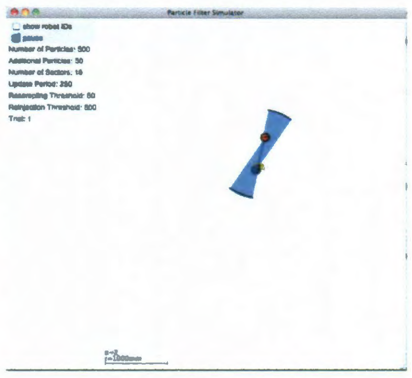
Update Period: 250

Resampling Threshold: 50

Reinjection Threshold: 800

Trial: 1

(a) All the particles fall outside of the sector after a transition.



Particle Filter Simulator

show robot IDs

pause

Number of Particles: 500

Additional Particles: 50

Number of Sectors: 16

Update Period: 250

Resampling Threshold: 50

Reinjection Threshold: 800

Trial: 1

(b) The particles get "stuck" at the point of the sector.

Figure 5.11 : Degenerate Cases of my particle filter algorithm.

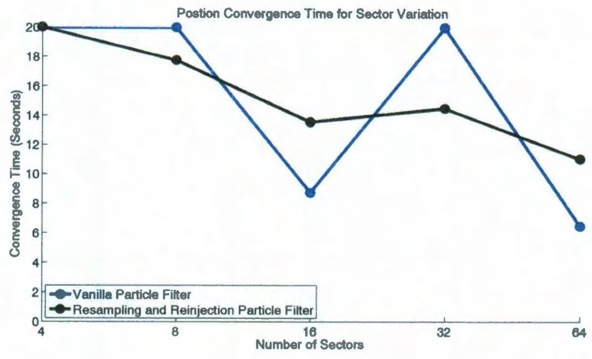
the actual pose. This is merely poor experiment design. To get a better handle on the how quickly this system converges I would need to disregard the first two sector transitions, if not the first three, and then determine when the system converges. Moving forward I addressed this issue by moving the starting point down and the ending point up. This set of points will ensure that the estimated pose does not cross the path of the actual pose. This will induce a greater error at the outset since the probability distribution will have to diverge more before getting a good estimate of the pose.

5.5.4 Degenerate Cases

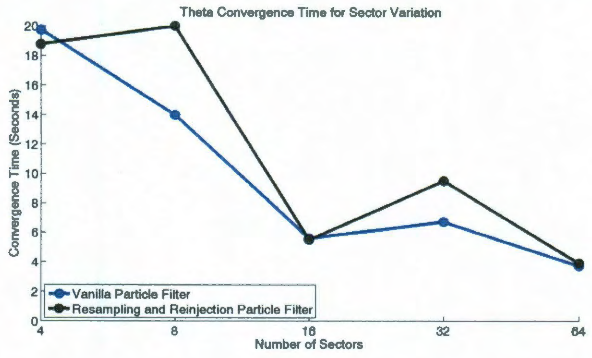
After observing my implementation of the particle filter over several iterations I noticed that there are two degenerate cases that occasionally occurred. Both of these cases are illustrated in Figures 5.11(a) and 5.11(b).

The first degenerate case, Figure 5.11(a), is a result of all of the particles ending up outside of the sector during a sector transition. This typically happens when all of the particles, and subsequently the estimate, are close to the far edge of the sector. During a sector transition none of the particles make the transition to the new sector at the same time so all particles are weighted down accordingly. Resampling does nothing to correct this degenerate case because the goal of resampling is to propagate particles with higher weights and since they all have the same weights the particles get "stuck" outside of the sector. To deal with this degenerate case I added a second threshold to check for during the update phase of the particle filter, this is the reinjection threshold. When this threshold is met the particle filter is rebooted by discarding the old particles and making a new uniform distribution of particles. To detect when all of the particles leave the sector completely the limit of this threshold is set at the initial number of particles.

The second degenerate case, Figure 5.11(b), occurs when the only particles remaining end up near the point of the sector. Once there they are unable to move towards the actual pose of the neighboring robot. This is due to the error model for tv and rv. By introducing a constant error to the tv and rv we produce a nearly round cloud of particles that depicts the estimated pose. If this cloud of particles gets near the point of the sector the error model produces particles that are always inside and outside of the sector. The outside particles are immediately weighted down and discarded during the next resampling phase, but the inside particles are kept. Since we are losing a significant portion of our particles but keeping incorrect ones, the particle filter cannot converge. To deal with this degenerate case I am making the assumption that the robots will only pass that close for a short time. Therefore I weight particles that are near the point of the sector as if they were outside of the sector. This ensures that the particle cloud never gets stuck.



(a) Convergence Time for Position with Sector Variation.

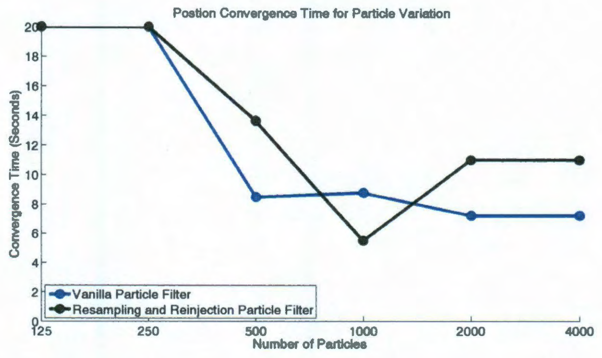


(b) Convergence Time for Theta with Sector Variation.

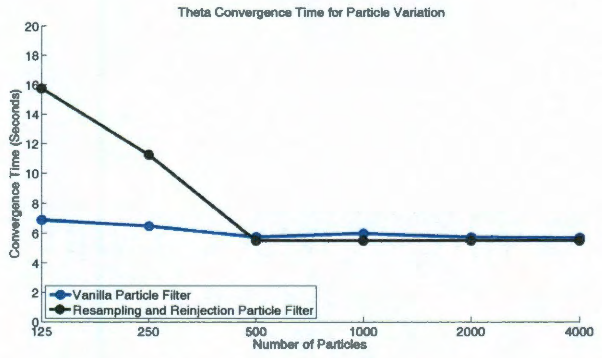
Figure 5.12 : Convergence times, in seconds, for both (x,y) and θ while varying the number of sectors.

5.6 Convergence Times

Figures 5.12(a) and 5.12(b) illustrate the convergence time while varying the sector parameter. There are two reasons for varying this parameter. First, I wanted to see how sensitive my particle filter was to the sector parameter. Second, I wanted to verify that the 16 sector design was a correct decision. I was able to verify that my particle filter algorithm is sensitive to the number of sectors which indicates that tuning is required if the number of sectors change on the r-one. The figure also verifies that the 16-sector design for the r-one allows for a low convergence time.



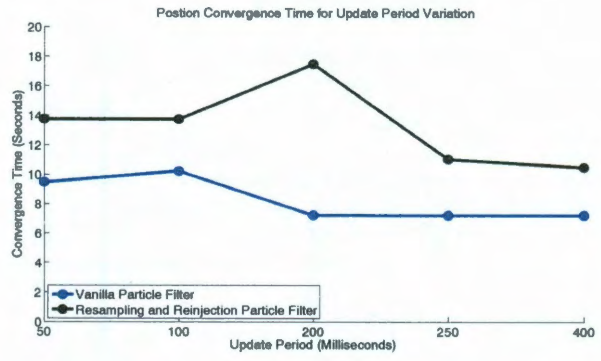
(a) Convergence Time for Position with Particle Population Variation.



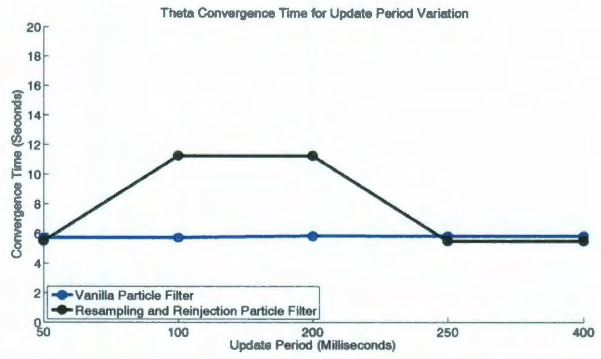
(b) Convergence Time for Theta with Particle Population Variation.

Figure 5.13 : Convergence times, in seconds, for both (x,y) and θ while varying the population of particles.

Figures 5.13(a) and 5.13(b) illustrate the convergence time while varying the particle population size parameter. The main reason for adjusting the number of particles that the particle filter is initialized with is to determine the absolute lowest number of particles that I can use while still reaching convergence in a reasonable amount of time. The figures indicate that 1,000 particles seems to be the optimal solution. However, this may change with more time invested in tuning the particle filter for 500 particles.



(a) Convergence Time for Position with Update Period Variation.



(b) [Convergence Time for Theta with Update Period Variation.

Figure 5.14 : Convergence times, in seconds, for both (x,y) and θ while varying the update period.

Figures 5.14(a) and 5.14(b) show the convergence time while varying the update period parameter. While changing the update period I discovered that my particle filter algorithm is relatively invariant to the length of the update period. However, thinking about this a little more thoroughly leads me to believe that this invariance exists due to the simple experiment design of having the neighboring robot travel in a straight line. To further investigate this issue I would need to run another round of experiments that have the neighboring r-one travel using complex patterns.

Chapter 6

Conclusions and Future Work

6.1 Limitations

A major limitation of my particle filter implementation is that it requires the robot being sensed to be moving. If it is not moving the amount of information garnered from the sensing robot is very little, a majority of the information provided in this particle filter comes when a robot crosses from one sector to the next.

A major limitation turned out to be the amount of memory that the r-one has. This memory limited the maximum number of particles to 1,000. However, at this amount the r-one would not be able to run any other task. This is why I focused on tuning the particle filter algorithm for 500 particles. This produced reasonable results while freeing up enough resources for the r-one to run other tasks concurrently with the particle filter algorithm.

6.2 Future Work

There are several avenues to move forward from this thesis. First, there is the implementing of this algorithm in C and on the r-one hardware to verify it's real-world effectiveness. Second, with the large number of parameters to tune to make this algorithm optimal one can devise a method to automate the parameter tuning. Finally, one can add an additional layer between the algorithm and the r-one hardware that will smooth the estimate using an exponential moving average.

6.2.1 Actual Robot Testing

There exists a rift between a real-world implementation of an engineering system and a software simulation of the system. Even though a developer may take every precaution available to them they can never make a simulator truly represent the real world. This is due to the inherent differences in the design approach and the assumptions made when modeling the subsystems and the interactions between them. Take the infrared messaging system as an example, more specifically the passing of the robot ID, translational velocity, and rotational velocity. In the simulator I simply use a setter/getter to pass this information from the origin to the destination. On the actual r-one itself it is not this simple, the r-one has to put the message together and the infrared message system has to transmit it. There are several potential error injection points in just in these two steps. With this in mind I wanted to conclude my thesis with the experimental results of running my particle filter on the r-one's.

The primary way forward is to conduct actual experiments with the r-one multi-robot system and my particle filter algorithm written in C. Seeing this algorithm run on the actual hardware would reinforce the findings made through the use of the simulator and allow me to garner just how computationally exhaustive this particle filter algorithm is and to see if it is a viable range estimation solution for the r-one.

6.2.2 Automated Parameter Tuning

Another avenue to pursue would be to determine the best set of parameters for this particle filter implementation. I believe the way forward for this would be to further extend the simulator and to wrap it with a genetic algorithm to test all of the parameters and to let it run. Once the framework was built up it could be used to determine the best parameters for any particular situation or the best set of parameters for a generalized situation. The latter would be the best product of this work since the state of a multi-robot system is always in flux due to movement of the robots themselves or to fluctuations in the inter-robot communications.

6.2.3 Pose Smoothing on the R-one

One thing I notice during the simulation runs is that the estimated pose has a tendency to “jitter” quite a bit. This is a result of the sensor model. With the sensor being an arc we get a lot of information from each sector transition and only a little information while within the sector. As the robot stays in the sector the estimate deviates from the actual position due to this lack of information. To combat this one could implement an exponential moving average for the estimated pose. This would allow the r-one to weight a large transition in the estimated pose lower than a smaller transition. This is based on the fact that the velocity controller for the r-one produces a smooth tv and rv and does not “jump around”. By smoothing the path of the estimated pose the r-one will produce a better estimate the pose of the neighboring robot.

6.3 Final Remarks

I have evaluated my particle filter algorithm using three key parameters; the number of sectors, the number of particles and the update period. I have determined that my particle filter algorithm is sensitive to the the number of sectors and the number of particles but insensitive to the update period. The sensitivity to the number of sectors is mitigated by the fact that the algorithm converges in a reasonable amount of time with 16 sectors, the current number that the r-one uses. The number of particles is also mitigated by the fact that the 500 particle case still converges within a reasonable amount of time as well. This is a number that the r-one hardware can efficiently implement. The insensitivity to the update period allows us to move forward without having to change the current update period on the r-one.

Finally, the main result of this work is the extension of the capabilities of the r-one robot. By implementing my specific particle filter algorithm and allowing the r-one to be able to estimate the range between itself and its neighbors I have raised the r-one in the dominance relation explained in Section 2.2.3. The r-one is now placed above the bearing-only coordinate system but below local coordinate systems as seen in Figure 2.1.

Bibliography

[1] P. Ranganathan, R. Morton, A. Richardson, J. Strom, R. Goeddel, M. Bulic, and E. Olson, "Coordinating a team of robots for urban reconnaisance," in *Proceedings of the Land Warfare Conference (LWC)*, Nov. 2010.

[2] W. Burgard, D. Fox, and S. Thrun, "Active mobile robot localization," 1997.

[3] N. Michael, J. Fink, and V. Kumar, "Experimental testbed for large multirobot teams," *Robotics & Automation Magazine, IEEE*, vol. 15, no. 1, pp. 53–61, 2008.

[4] "Vicon MX systems." http://www.vicon.com/products/viconmx.html, 2010.

[5] M. Walter, M. Anderson, I. Burt, and N. Papanikolopoulos, "The design and evolution of the eROSI robot," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 2984–2989, 2007.

[6] J. McLurkin, A. Lynch, S. Rixner, T. Barr, A. Chou, K. Foster, and S. Bilstein, "A Low-Cost Multi-Robot system for research, teaching, and outreach," *Proc. of the Tenth Int. Symp. on Distributed Autonomous Robotic Systems DARS-10, November*, p. 200, 2010.

[7] J. McLurkin, *Stupid Robot Tricks: A Behavior-Based Distributed Algorithm Library for Programming Swarms of Robots.* S.M. thesis, Massachusetts Institute of Technology, 2004.

[8] J. McLurkin, J. Smith, J. Frankel, D. Sotkowitz, D. Blau, and B. Schmidt, "Speaking swarmish: Human-Robot interface design for large swarms of autonomous mobile robots," Mar. 2006.

[9] J. McLurkin, *Analysis and Implementation of Distributed Algorithms for Multi-Robot Systems.* Ph.D. thesis, Massachusetts Institute of Technology, 2008.

[10] R. M. Harlan, D. B. Levine, and S. McClarigan, "The khepera robot and the kRobot class: a platform for introducing robotics in the undergraduate curriculum," in *ACM SIGCSE Bulletin*, SIGCSE '01, (New York, NY, USA), p. 105109, ACM, 2001. ACM ID: 364553.

[11] J. Pugh, X. Raemy, C. Favre, R. Falconi, and A. Martinoli, "A fast onboard relative positioning module for multirobot systems," *Mechatronics, IEEE/ASME Transactions on*, vol. 14, no. 2, pp. 151–162, 2009.

[12] C. M. Cianci, X. Raemy, J. Pugh, and A. Martinoli, "Communication in a swarm of miniature robots: the e-Puck as an educational tool for swarm robotics," in *Proceedings of the 2nd international conference on Swarm robotics*, SAB'06, (Berlin, Heidelberg), p. 103115, Springer-Verlag, 2007. ACM ID: 1763844.

[13] A. Gutierrez, A. Campo, M. Dorigo, J. Donate, F. Monasterio-Huelin, and L. Magdalena, "Open e-puck range & bearing miniaturized board for local communication in swarm robotics," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, p. 31113116, 2009.

[14] "MAGIC 2010: Super-smart robots wanted for international challenge.." http://www.dsto.defence.gov.au/MAGIC2010/, 2010.

[15] L. Erickson, J. M. OKane, and S. M. LaValle, "Probabilistic localization using only a clock and a contact sensor," in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2007.

[16] J. Yu, S. M. LaValle, and D. Liberzon, "Rendezvous without coordinates," in *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, p. 18031808, 2009.

[17] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus, "Tracking a moving object with a binary sensor network," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, p. 150161, 2003.

[18] K. Konolige, J. Augenbraun, N. Donaldson, C. Fiebig, and P. Shah, "A Low-Cost laser distance sensor," 2008.

[19] "The open lidar project - hack the neato XV-11 lidar for a $800 bounty! by gallamine | RobotBox." zotero://attachment/2/, 2010.

[20] "PrimeSense." http://www.primesense.com/, 2011.

[21] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments," 2010.

[22] E. Herbst, P. Henry, X. Ren, and D. Fox, "Toward object discovery and modeling via 3-D scene comparison," 2011.

[23] D. Rus and B. D. Jennings, "Moving furniture with teams of autonomous robots," 1995.

[24] M. Erdmann, "Towards Task-Level planning: Action-Based sensor design," *Robotics Institute*, Jan. 1992.

[25] J. M. OKane and S. M. LaValle, "On comparing the power of mobile robots," in *Robotics: Science and Systems*, 2006.

[26] A. Cornejo, J. McLurkin, S. Bilstein, E. Fudge, A. Lynch, and N. Lynch, "Computing Scale-Free coordinates on Multi-Robot systems," *The Ninth Int. Workshop on the Algorithmic Foundations of Robotics WAFR-10, November*, p. 1, 2010.

[27] S. G. Loizou and V. Kumar, "Biologically inspired bearing-only navigation and tracking," pp. 1386–1391, Dec. 2007.

[28] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," vol. 2, pp. 1322–1328 vol.2, 1999.

[29] I. J. Cox, "Blanche-an experiment in guidance and navigation of an autonomous robot vehicle," *Robotics and Automation, IEEE Transactions on*, vol. 7, pp. 193–204, Apr. 1991.

[30] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics.* MIT Press, 2005.

[31] I. M. Rekleitis, "A particle filter tutorial for mobile robot localization TR-CIM-04-02," *Centre for Intelligent Machines*, 2003.

[32] S. Lenser and M. Veloso, "Sensor resetting localization for poorly modelled mobile robots," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, p. 12251232, 2000.

[33] P. Jensfelt, O. Wijk, D. J. Austin, and M. Andersson, "Experiments on augmenting condensation for mobile robot localization," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 3, p. 25182524, 2000.

[34] J. S. Liu, R. Chen, and T. Logvinenko, "A theoretical framework for sequential importance sampling and resampling," *Sequential Monte Carlo Methods in Practice*, p. 225246, 2001.

[35] D. Fox, S. Thrun, W. Burgard, and F. Dellaert, "Particle filters for mobile robot localization," in *Sequential Monte Carlo Methods in Practice*, 2001.

[36] C. Kwok, D. Fox, and M. Meila, "Real-time particle filters," *Proceedings of the IEEE*, vol. 92, no. 3, p. 469484, 2004.

[37] J. Carpenter, P. Clifford, and P. Fearnhead, "Improved particle filter for nonlinear problems," *Radar, Sonar and Navigation, IEE Proceedings -*, vol. 146, no. 1, pp. 2–7, 1999.

[38] D. Fox, "Adapting the sample size in particle filters through KLD-sampling," *The international Journal of robotics research*, vol. 22, no. 12, p. 985, 2003.

[39] N. Abramson, "The aloha system: Another alternative for computer communications," Technical Report B70-1, University of Hawaii, Honolulu, Hawaii, Apr. 1970.

[40] J. Feddema, C. Lewis, and D. Schoenwald, "Decentralized control of cooperative robotic vehicles: theory and application," *Robotics and Automation, IEEE Transactions on*, vol. 18, no. 5, p. 852864, 2002.

[41] M. J. Mataric, *Interaction and Intelligent Behavior.* PhD thesis, Massachusetts Institute of Technology, 1994.

[42] N. B. Priyantha, A. Chakraborty, and H. Balakrishnan, "The cricket location-support system," in *Proceedings of the 6th annual international conference on Mobile computing and networking*, (Boston, Massachusetts, United States), p. 3243, ACM, 2000.

[43] M. Veloso, M. Bowling, S. Achim, K. Han, and P. Stone, "The CMUnited-98 champion Small-Robot team," in *RoboCup-98: Robot Soccer World Cup II*, vol. 1604 of *Lecture Notes in Computer Science*, pp. 77–92, ACM, 1998.

[44] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, "SwisTrack - a flexible open source tracking software for Multi-Agent systems," in *Proceedings of the IEEE/RSJ 2008 International Conference on Intelligent Robots and Systems (IROS 2008)*, p. 40044010, IEEE, 2008.

[45] E. Olson, "AprilTag: a robust and flexible multi-purpose fiducial system," tech. rep., University of Michigan APRIL Laboratory, May 2010.

[46] J. McLurkin, "Experiment design for large Multi-Robot systems," in *Robotics: Science and Systems, Workshop on Good Experimental Methodology in Robotics*, (Seattle, WA, USA), June 2009.

[47] A. Das, R. Fierro, V. Kumar, J. Ostrowski, and C. J. Taylor, "A Vision-Based formation control framework," *IEEE Transactions on Robotics and Automation*, vol. 18, pp. 813–826, Oct. 2002.

[48] J. S. Liu, "Nonparametric hierarchical bayes via sequential imputations," *The Annals of Statistics*, vol. 24, no. 3, p. 911930, 1996.

[49] J. S. Liu, *Monte Carlo strategies in scientific computing.* Springer Verlag, 2008.