An Adaptive Nested Particle Filter with Advanced Weighting to handle Localization and Tracking under extreme sensory occlusion

by

Kedar Pramodan Marathe

(Under the direction of Prashant Doshi)

Abstract

Robot localization under extreme occlusion is a common scenario in the real world. Examples include an autonomous car following another car along long stretches of highways, robot assistants following a human around, hospitality robots, robot pets, etc. The challenge here is that the robot's sensors get occluded to a large degree and for substantial amounts of time by the leader that the robot is trying to follow. In this scenario, localization using a usual MCL particle filter gets worse over time since observations consistently keep deviating from static expectations. In this thesis I present an Advanced Weighting approach for the Nested Particle Filter, to maintain localization under extreme occlusion while simultaneously tracking the leader. I also adapt KLD sampling to Nested MCL, which makes this approach highly scalable by allowing for dynamic variation in particle counts at both levels of nested particles, instead of ad-hoc static counts.

Index words: Particle Filter, Monte Carlo Localization, Robot localization under sensory occlusion, KLD Sampling, Likelihood field, Mixture MCL

An Adaptive Nested Particle Filter with Advanced Weighting to handle Localization and Tracking under extreme sensory occlusion

by

Kedar Pramodan Marathe

B.E., University of Pune, 2009

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Master of Science

Athens, Georgia

2014

An Adaptive Nested Particle Filter with Advanced Weighting to handle Localization and Tracking under extreme sensory occlusion

by

Kedar Pramodan Marathe

Approved:

Major Professor: Prashant Doshi

Committee: Walter Potter
Khaled Rasheed

Electronic Version Approved:

Julie Coffield
Interim Dean of the Graduate School
The University of Georgia
December 2014

Dedications

I dedicate this thesis to my parents, without whom none of this would have been possible. They taught me not only to think independently, but also to stay humble and true to myself.

Acknowledgments

I would like to give my deepest gratitude to Dr. Prashant Doshi for not only being a great guiding force, but also a firm hand that kept me constantly on my toes, striving to achieve the best I could. A special thanks to Kenneth Bogert who is a PhD student in the THINC lab under Dr. Doshi, whose invaluable insights and critical attitude was a catalyst throughout the process.

I would like to thank Dr. Potter and Dr. Rasheed for their excellent guidance and tutoring over the years. They are some of the best teachers I've had the privilege to learn from. The Institute for Artificial Intelligence at UGA deserves a special mention for letting me use the workstations and robots from the AI lab for my physical experiments.

Lastly, I would like to thank my fiancée, my two sisters, and all the dear friends I have made over these years at UGA, for standing firmly by my side during this time.

Contents

List of Figures

Chapter 1

Introduction and Background

Robot Localization has been recognized over and over again as one of the most critical challenges that needs to be tackled in order to make robots viable in the real world. Monte Carlo Localization or MCL has proved to be one of the most successful methods for efficient and reliable localization [9]. Coupled with models that account for real world noise and unexpected sensory occlusions, and along with methods like Mixture MCL, localization can be made quite robust, such that it performs well even in crowded real-world environments[10, 8]. The augmentation of MCL with KLD sampling allows for an approach that is computationally efficient and highly scalable and flexible[8, 1].

Despite all these robust techniques for handling noise and unexpected dynamic objects in the environment, the MCL approach performs badly in a scenario that is all too common: the follower behavior. This behavior can be seen in many different situations such as an autonomous car following another car in a convoy-like formation on long patches of a highway, or a robot assistant following a human around in settings like hospitals, laboratories, nursing homes, or even a robot pet that follows a child around the house. Maintaining it's self localization while simultaneously tracking the "leader" is quite a common task, which is not handled sufficiently just by sensor models that assign noise to unexpected observations. In this scenario, occlusion of the sensors occurs to a large degree and for extended durations of time, which causes localization to get worse and worse over time. There have been a few attempts at accounting for sensory occlusion as described in [2] and [5]. In [2], they mainly deal with localization errors due to unpredictable and sporadic occlusion, and come up with sensor models that filter out noise based on a probability model of expected sensory readings. This does not help us in a scenario where the sensory occlusion is persistent and forms a large portion of our sensory input. The problem is that if the sensory occlusion does not go away for extended durations of time then modeling such noise based on a predetermined assumption of sensor's probability of measurements does not help. In [5], they designed a system for accounting for occlusion in the Robocup environment. They deal with a sensory model that relies on color-based processing of sensory data, which also needs 360-degree sensory vision. Obviously, this is a setting that is not always easily available in most robots. We introduce a more generalized algorithm that will work in most common environments, with less than 360 degree sensory input as well as simplistic external leader detection mechanisms, that are independent of the algorithm.

In this thesis, we introduce an Advanced Weighting method that modifies expectations based on the location of the leader. This method not only maintains localization of the follower throughout the follower behavior, but also simultaneously tracks the leader during this time. We further introduce a modification to the traditional Nested Particle Filter that makes it significantly more scalable and efficient. The leader in our scenario can be a human, a robot or any other kind of agent. We use two identical robots for our experiments to keep things simple. The leader is interchangeably called the "leader" or "$robot_j$", and the follower is called either the "follower" or "$robot_i$". Throughout the thesis, we are only concerned with the localization and tracking that happens within $robot_i$, so whenever we talk about $robot_j$'s particles, we are talking about the second level particles within first level particles of $robot_i$ that are being used to track $robot_j$.

1.1 What is "Robot Localization"?

The problem of robot localization pertains to determining the location and orientation of a robot in a given know environment. Consider that we are given a map of an environment in which the robot is placed. The problem of figuring out the (x, y, θ) of the robot that corresponds to its true location on that map is the "localization problem"[8]. The "map" that is given to the robot can either be a blueprint of the environment obtained from external sources, or a learned map that is generated by running the robot through the environment performing a "map learning" task using "Simultaneous Localization and Mapping" or SLAM algorithms, that can generate maps of the given environment. Any of the competent SLAM algorithms such as the ones described in [8] could work well. We use the gmapping package that is part of the Robot Operating System [7]. Gmapping implements a grid-based SLAM approach as described in [4] and [3]. In the current context, we are only interested in localization, and not the map learning task. So, we perform the map learning beforehand. We then assume that the robot is given a map of the environment. To be specific, we use a grid based map that informs the robot which areas of the environment are "occupied" by obstacles such as doors, walls, pillars, and other objects that can obstruct movement.

Robot Localization is a difficult problem to solve for several different reasons-

- Robot sensors are noisy and not always reliable.
- Robot movement and odometry information has noise in it.
- Real world environments have dynamic components to it that are hard to predict such as humans moving around, furniture locations that keep changing, doors and windows that open and close, changing lighting conditions within indoor as well as outdoor environments.

1.2 Nested Particle Filtering

1.2.1 Overview of Monte Carlo Localization

Monte Carlo Localization or "MCL" is an approximate method for localization. It is based on the Markov assumption that the robot's position and orientation or "pose" at any given time t is dependent only on it's location at time *t-1* and the action at time *t-1*. This Markov assumption is based on a static assumption of the robot's environment. Which means that while the robot is moving, everything else in the environment is assumed to remain static.

The belief over the robot's pose is represented by a probability distribution over the environment space at time *t-1*, which acts as a "prior". This distribution can either be given to the robot beforehand, or the robot could have no information about its location to begin with, in which case it is represented by a uniform distribution over the entire map. In MCL, this is represented by sampling over the prior distribution, where each sample is a hypothesis about the robot's pose at time *t-1*. These hypotheses are called "particles" which contain the location and orientation of the robot, along with a "weight" which represents the likelihood of that particular hypothesis being true. In case of initial distributions, the weights are uniform. This is called the *"Sampling"* step.

Then, based on the action that the robot takes which is given by the odometry, particles are transitioned forward to the next time step t. New samples are thus generated by sampling from the Transition Model. This generates another set of particles from the prior based purely on the transition model. This is the "propagation" step.

Now, the set of particles generated from the propagation step need to be reconciled with the real world sensory input. This is because the transition model introduces noise, since the precise action cannot be given by the robot's odometry. Thus, the distribution represented by the unweighted newly propagated particles has a divergence from the true posterior distribution.

We now use the sensor model to do "Importance Weighting" over the propagated particles. This is done by comparing the observational expectations of each particle with the actual observations that are recorded by the sensors. The particles whose expectations match the observations closely receive high weights, and the ones that differ receive lower weights. After the importance weighting is done, the particles along with their weights together represent an approximation of the true posterior distribution at time t. This is the *"Weighting"* step.

Resampling with replacement is then performed over this posterior distribution to get a new set of particles that represents the new belief about the location of the robot. The probability of a particle getting resampled is proportional to its weight. This ensures that only the poses which are closest to the truth are carried forward into the next iteration, and we don't waste resources in keeping track of weak particles. This is the "resampling" step.

All the above steps $Sample \rightarrow Propogate \rightarrow Weight \rightarrow Resample$ are performed repeatedly for every iteration until the particles start converging to a location. Once the particles sufficiently converge near the true location of the robot we can say that the robot is "localized".

1.2.2 Nested MCL

Now that we have briefly reviewed the typical MCL algorithm, let us look at the Nested MCL algorithm. The Nested Particle Filter or "NPF" for short, is a Particle Filter modified to not only localize the subject robot ($robot_i$), but to also simultaneously track another agent ($robot_j$), using logic that is similar to the one used for localizing $robot_i$. The Nested Particle Filter (NPF) as introduced in [6], was used mainly for estimating the location of an opponent to predict which landmark will be moved by the opponent. This is useful because if we can predict which of the landmarks has moved, we can then use that information to ignore that landmark in landmark based localization. In this thesis, we use NPF in a different way . We

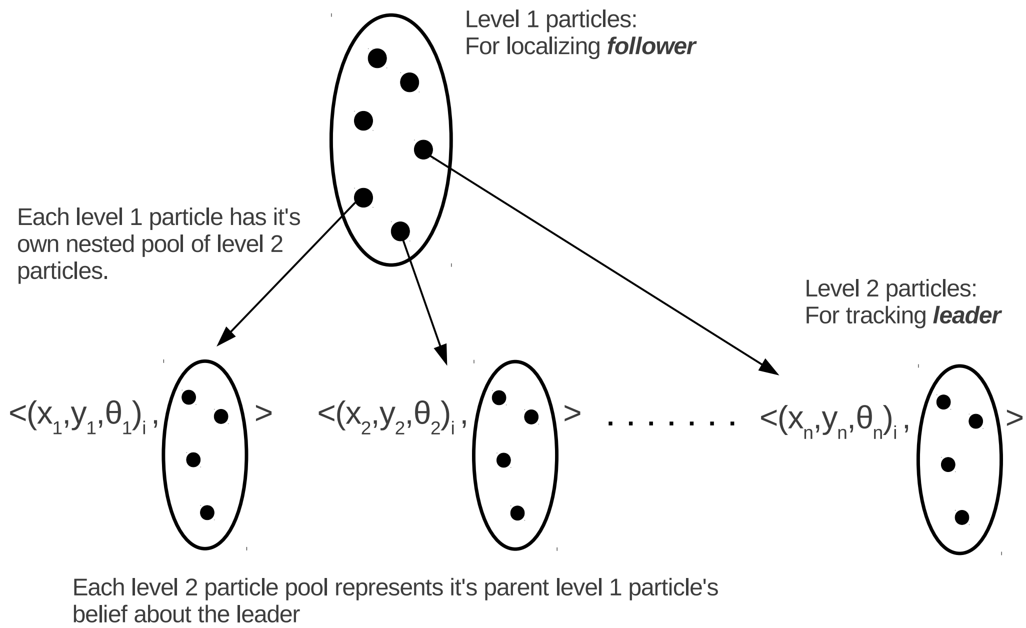


Figure 1.1: Nested Particle Structure

use the information about the leader's location to maintain self-localization by accounting for the resultant sensory occlusion. We are not concerned with any movement in landmarks that might or might not happen due to the leader.

We insert a pool of particles that represents $robot_i$'s hypotheses about the possible location of $robot_j$ into each particle of $robot_i$. Its like saying "If I am at a location $(xn_i, yn_i, \theta n_i)$, then $robot_j$'s location must be given by a set of particles $\{(x1_j, y1_j, \theta 1_j), (x2_j, y2_j, \theta 2_j), ..., (xn_j, yn_j, \theta n_j)\}$ ". These particles in the second level pools are called "Level-2 Particles" since they are all nested inside the first level particles of $robot_i$. We call particles at the first level "Level-1 Particles" and the second level particles "Level-2 Particles". Please refer to figure 1.1 for some more clarity.

The $Sample \rightarrow Propogate \rightarrow Weight \rightarrow Resample$ steps are modified to become recursive for the Nested Particle Filter or "NPF".

Propagation

Propagation of level-2 particles is carried out a little differently from the propagation of level-1 particles. For level-1 particles, we receive information about the robot's action. This information is used to propagate all the level-1 particles forward. But for level-2 particles, since there is no communication between the two robots, we have no information about the action taken by $robot_j$ to propagate the level-2 particles within $robot_i$. Instead we make an assumption about the Transition Model for $robot_j$ and propagate the level-2 particles according to this Transition Model.

The Transition Model that is assumed in our case is a naive probability based model. It assigns a probability of 90% for an action of moving straight ahead, and a probability of 5% each to turning by 30 degrees on either right or left. This is the general case. For ensuring that level-2 particles do not move through walls, we make another assumption. The probability of moving straight is proportional to the distance of a level-2 particle from the nearest obstacle directly in front of it. This means, if a level-2 particle has an obstacle up close in front of it at the minimum safe distance specified, then the level-2 particle will have 0% probability of going straight and 50% probability of turning in either direction. Also, we assume that $robot_j$ moves at the same speed as $robot_i$

Weighting

Weighting of level-2 particles is divided into 2 parts:

1. When the leader ($robot_j$) IS VISIBLE to the follower ($robot_i$):

   When the leader is visible to the follower, weighting for level-2 particles is done using the usual zero centered Gaussian with mean located at a pose where the parent level-1 particle observes the leader. Of course, if the level-2 particle is calculated to be in an exceptional location like inside a wall then it is penalize with a pre-determined low weight. This predetermined low weight is calculated from the Gaussian using a "maximum occupancy distance" (*max_occ_dist*) that is passed as a parameter to the algorithm.

2. The leader is NOT VISIBLE to the follower:

   This weighting scheme is called "Negative Weighting", since the absence of an observation of the leader serves to determine weights of level-2 particles. The way this works is-

   - If a level-2 particle is in the visible field relative to its parent level-1 particle, then we give it a predetermined low weight, using the *max_occ_dist*. This is because we know that the leader is not visible although it would have been visible if it was at the location of this particular level-2 particle.

   - Similarly, a level-2 particle that is not in the field of vision of its parent level-1 particle when the leader is NOT visible gets a pre-determined higher weight (calculated using *max_occ_dist*/2). This is because the level-2 particle complies with the expectation that the leader is not visible when it is not observed.

The procedure for both *Propagation* and *Weighting* is very similar. For both, we do a simple recursive traversal through each level-1 particle and its corresponding level-2 pool, propagating and weighting as we go.

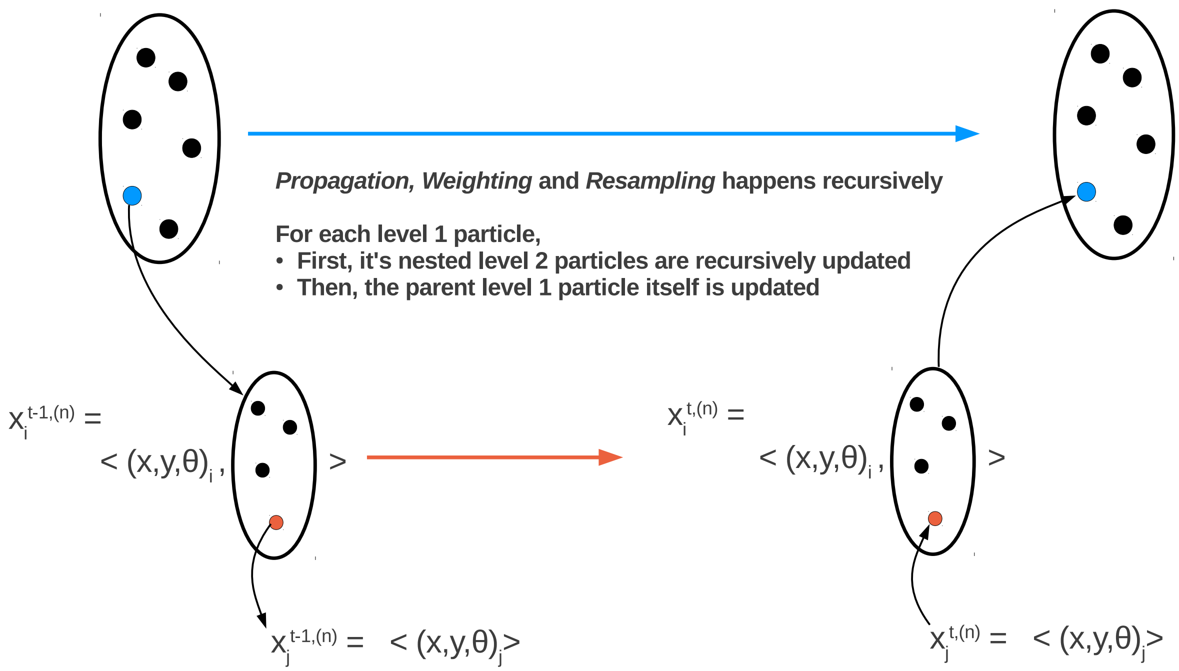


Figure 1.2: Nested Particle Recursive Updates

For example, for weighting, the steps would be as follows:

1. For each level-1 particle

   (a) Go to the level-2 pool within the level-1 particle (child pool)

   (b) Weight each Level-2 Particle within the level-2 pool using the observed location of $robot_j$

   (c) Return to the parent level-1 particle and weight this particle using the sensory observations.

2. Repeat until all level-1 particles and their corresponding level-2 particles have been weighted.

3. Stop

Resampling

The *Resampling* procedure is similar, but there is a subtle difference. Here, we have to resample a level-1 particle first, and then resample the level-2 particles that belong to the corresponding level-2 pool. We cannot process the level-2 particles first like in propagation or weighting because its necessary to first know if the parent level-1 particle will survive resampling. This proceeds as follows:

1. Resample a level-1 particle from the pool of weighted level-1 Particles, with probability of getting resampled proportional to the level-1 particle's weight.
    (a) Go to the level-2 pool within the resampled level-1 particle (child pool)
    (b) Resample from within the level-2 pool using the weight of each level-2 particle. Resample level-2 particles until the preset maximum level-2 population limit is reached.
2. Repeat until the preset maximum level-1 particle population is reached.
3. Stop
   NOTE: The population limit is a preset parameter and is a per-pool limit set for each level of nesting. So its different for level-1 particles and level-2 particles.

You might have noticed that in NPF, the individual steps of Propagation, Weighting and Resampling each happens like a Depth First Search tree traversal algorithm where each level-1 particle is a parent node and the level-2 particles can be thought of as leaf nodes. It is also worthwhile to note that just as in tree traversal, we can traverse the entire tree with either Depth First Search or Breadth First Search and still visit all the nodes since these are both complete search strategies. There is a way of improving efficiency of this traversal when it pertains to Nested Particle Filtering. We will discuss this in greater detail in the latter section on Adaptive NPF.

1.3 Mixture MCL

The problem with Nested PF is that tracking the leader ($robot_j$) using level-2 particles is not easy, since we cannot afford to have lots of level-2 particles per level-2 pool. Besides the meager level-2 particle counts, we have more problems with tracking such as two layers of noise which get added to tracking performance. At the first level, level-1 particles themselves have noise coming from localization inaccuracies. When we move deeper to the second level of particles, this noise gets carried forward and more noise pertaining to tracking is added to it. The second level noise comes from sources such as noise in observations of $robot_j$, our lack of knowledge about $robot_j$'s odometry, etc. All these shortcomings result in having very few or some times no level-2 particles at all that are anywhere close to the true location of $robot_j$. This is similar to the "Kidnapped Robot" problem in robot localization where the robot localizes to a wrong location and is then unable to recover because it does not have any particles anywhere near its true location.

One of the ways to deal with these issues is to use Mixture MCL, which is a more robust method and is capable of dealing not only with the kidnapped robot problem, but also works well with a fewer particles.

1.3.1 Sampling from the Dual

The idea of mixture MCL is to sample a small fraction of the particles from the observation function instead of the transition function, and "mix" these samples with the ones sampled from the transition function [10]. Sampling from the observation function is called as sampling from the "Dual". It involves generating samples that would match most closely with the observations, and then using the transition function to weight the generated samples, instead of the other way around. We adapt this approach to nested particle filtering based on the observations of the leader ($robot_j$) that are constantly being received.

We generate a small fraction (5%) of level-2 samples from the observed pose of $robot_j$ when it is visible, and insert them along with other level-2 samples generated from the transition function. These samples, since they are generated from the observed location of $robot_j$, tend to match more closely with the true location of $robot_j$ even if other particles diverge significantly. Since a major portion of the scenario we are concerned with involves directly observing the leader at close distances, these samples tend to remain close to the true pose especially during the convoy-like behavior. This proves to be extremely helpful in maintaining tracking accuracy of the level-2 particles.

The performance of tracking is significantly improved with mixture MCL, even with meager level-2 particle counts, and goes a long way in aiding the Advanced Weighting approach which depends heavily on tracking accuracy. In our experiments, we will compare the performance of the Advanced Weighting approach with and without the use of Mixture MCL for tracking, to highlight its importance.

Chapter 2

Advanced Weighting

2.1 The need for Advanced Weighting

Now, let us go back to the scenario where the robot needs to localize itself while following a leader which could be a human, or another robot or some other kind of agent. This scenario poses a challenge to robot localization, because there is large scale sensory occlusion that occurs in following around another dynamic agent for such extended durations of time. The current approaches to localization, even with robust sensor models that account for noise cannot handle such a scenario. Random unexpected noise, which is what is used in both the Beam Model and the Likelihood Field Model is not a sufficient factor to account for persistent large scale occlusion [8]. Also, if we wanted to simply figure out which portion of the sensory input was coming from the leader, and which portion was coming from the static obstacles, we could perhaps remove the leader and look at the sensory readings, then in turn remove the obstacles and look at the sensory readings. This would allow us to isolate sensory readings coming from the leader. But this approach is naive and would only work in a simulation where removing individual obstacles or leader is easily accomplished, but not in the real world.

Because of this, traditional localization methods that work in a usual setting that has only random sporadic noise, do not work here. What happens when the robot needs to break away from the follower behavior to perform some other task? If it does not maintain its localization during the follower behavior, then after the follower behavior ends, the robot is left clueless regarding its whereabouts, and hence unable to move around in the given environment. This is the reason why we need to use Advanced Weighting.

2.2 Likelihood Field Model

Although the Advanced Weighting (AW) method can be used with other sensory models as well, the Likelihood Field model has been chosen in this case due to its efficiency, speed and ease of implementation.

Let us review the Likelihood Field model in brief as described in [8, pg 169], so as to help in understanding it's adaptation of AW. The key idea of Likelihood Field model is to project the end points of a range sensor scan into the global coordinate space of the given map of the environment. Since we already have the particle poses for the robot as well as the angle of the projected sensor beam and its range reading, we use simple trigonometric transformations to project these end points forward onto the map from the particle pose. Once the end points are found, we check to see whether the end point location corresponds to an obstacle on the map, since obstacles are what generate range readings. An "occupied" location at an end point represents an observation that matches expectations, and vice-versa.

Let $x_t = (x \quad y \quad \theta)^T$ be the robot's pose at time t as represented by a particle. Let z_t^k be the measurement by sensor beam k at time t. Let $(x_{z_t^k} \quad y_{z_t^k})$ denote the end point calculated from the range reading z_t^k. Also, let m represent the map of the given environment.

There are 3 types of sources of noise that are assumed-

- Measurement noise: This noise corresponds to the discrepancies within the sensor when taking measurements. In the 2D space that we consider, it can be calculated by measuring the distance ($dist$) between the end point location and the nearest known obstacle on the map. The probability of this particular measurement is then calculated by a zero-centered Gaussian which represents the sensor noise:

$$p_{hit}(z_t^k|x_t, m) = \varepsilon_{\sigma_{hit}}(dist) \tag{2.1}$$

- Failures: any max-range readings are modeled by a constant large probability p_{max}.
- Unexplained Random Noise: A uniform distribution p_{rand} is used to model this.

The likelihood for this sensor reading is then given by-

$$p(z_t^k|x_t, m) = z_{hit} \cdot p_{hit} + z_{rand} \cdot p_{rand} + z_{max} \cdot p_{max}$$

Here, z_{hit}, z_{rand} and z_{max} are parameters of the sensor model, that are derived independently. Refer to [8] for more details about their derivation.

The final probability of the particle is then calculated by multiplying the likelihoods of all the individual sensor readings. This is the same as in the beam model.

The problem with this model is that dynamic agents like people or other robots in the environment which could cause short readings are not modeled. The assumptions about random noise are sufficient if the noise is low, but it cannot account for persistent large scale occlusions of the kind that we are trying to address in our approach.

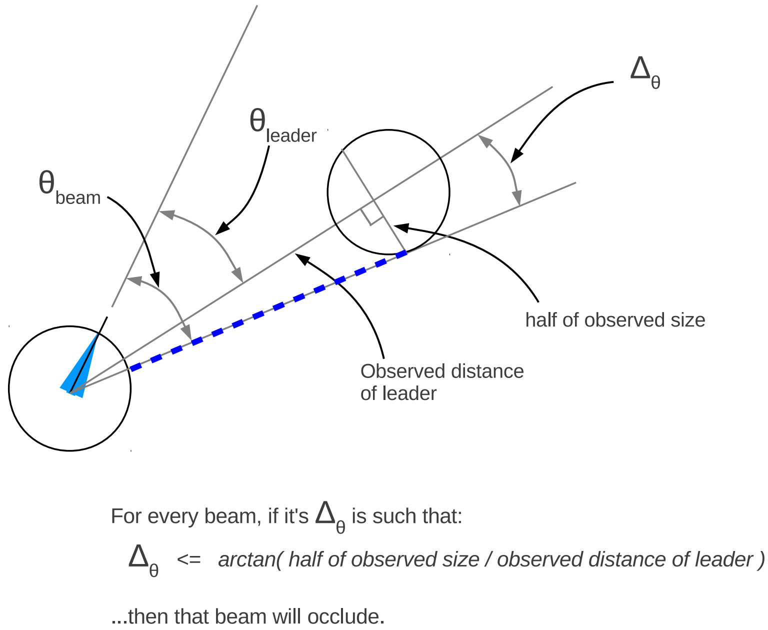


Figure 2.1: Occluding beam detection

2.3 Using Level-2 Particle Weights

The way to account for the persistent occlusions during a follower behavior is to create another likelihood field based on the location of the leader, similar to the one for the static obstacles, and use it to calculate the probabilities for the sensory beams that experience the occlusion.

We already have a set of level-2 particles within each level-1 particle, which represents our belief about the location of the leader. This belief can act as the likelihood field for observations of the leader. Hence, we can use this belief to calculate the probabilities of the occluding sensor beams.

2.3.1 Detecting the occluding beams

The first step is to figure out which sensor beams are getting occluded due to the leader. This can be determined by using an external detection mechanism. For example a color blob detector from a color sensing camera, or some other feature detection tool, which gives us the relative location and visible size of the leader from the follower. Using the size and the relative location of the leader, it is a matter of simple trigonometry to figure out which sensor beams will be occluded.

For each sensor beam, let the angle between the beam and the relative location of the leader be Δ_θ. This angle is given by-

$\Delta_\theta = abs(\theta_{leader} - \theta_{beam})$

θ_{leader} is provided by the external leader detector, and θ_{beam} is know since it is part of the robot's sensory characteristics.

Let the observed size of the leader be s_l and the observed distance of the leader be z_l. Now, to know if the beam is getting occluded by the leader we check the following simple trigonometric condition-

$$\Delta_\theta \leq \arctan\left(\frac{(s_l/2)}{z_l}\right)$$

If the above condition is true, then we can assume that the beam is getting occluded. This means the range reading is coming from a hit to the leader rather than something else. Note: It is possible that the hit is not coming from the leader despite these calculations, which can happen due to several reasons. Handling of these discrepancies is discussed in more detail in the next part where we use the level-2 particle poses for finding the likelihoods of occluding beams.

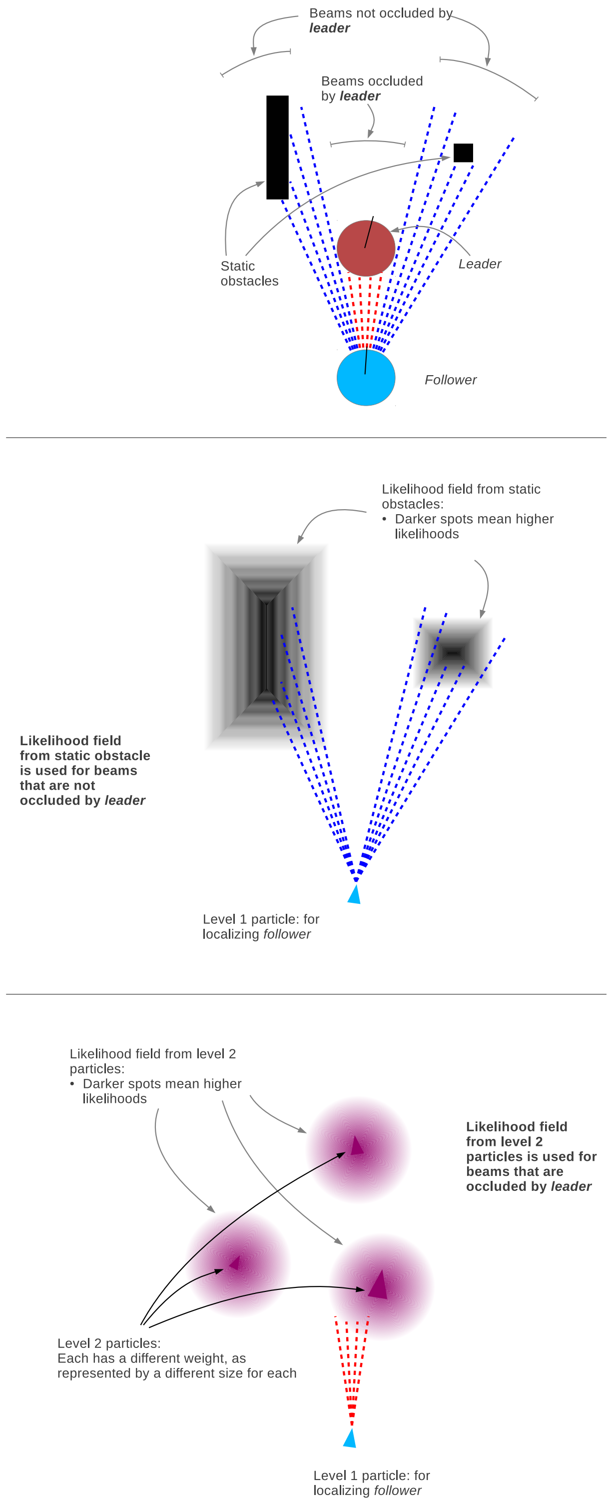


Figure 2.2: Two different likelihood fields for occluding and non-occluding beams

2.3.2 Calculating likelihoods of occluding beams

Once we have determined which of the sensor beams are getting reading from the leader, we can use the alternate likelihood field represented by the level-2 particles to perform weighting for these occluding beams specifically. We still need to handle noise that is encountered in sensor readings to make Advanced Weighting possible. Based on the assumption of noise made for the usual Likelihood Field model, we determine the following types of sources of noise for Advanced Weighting:

- Measurement noise
- Sensor failure
- Unexpected random measurements
- False positives in identification of occluded beams

  Note: False *negatives* in identification might also be present, but they do not cause noise for the Advanced Weighting part. They will cause noise for the usual likelihood weighting, but assuming that the false negatives are a negligible proportion of the sensor readings, they will likely be handled by the assumptions of noise by the standard likelihood field model.

Measurement Noise

The very first thing that we need to take care of is the measurement noise, which is intrinsic to any sensor. We use similar logic for handling this as we did with the usual Likelihood field model earlier. With reference to equation 2.1, we make a few modifications to the equation for occluding beams. Let z_t^{ko} represent the measurement received from an occluding beam "k" at time t. Let $m_{level-2}$ represent the belief over the location of the leader. We calculate the distance ($dist_{level-2}$) between an occluding beam's end point and the nearest level-2 particle, similar to the distance to the nearest obstacle calculated in the usual Likelihood field model. The probability is then calculated by again using a zero-centered Gaussian using this distance as follows:

$p_{hit}(z_t^{ko}|x_t, m_{level-2}) = \varepsilon_{\sigma_{hit}}(dist_{level-2})$

In equation 2.1 the belief over the locations of obstacles is represented by an "occupancy grid" represented by the map m which tells us whether or not an obstacle is present at a particular location. This belief is given to us as being true at the location of the calculated nearest obstacle specified by the map m. Hence, we do not have to worry about the probability of the obstacle being truly at that location.

The level-2 particles on the other hand, do not represent that the leader is actually present at that particular location. They represent a hypothesis about the leader being there, and come with a weight that represents the probability about that hypothesis being true. Just as x_t represents the location of the current level-1 particle that is being weighted, $x_t^{level-2}$ represents the location of the level-2 particle nearest to the end point of the sensor beam, from which we calculate $dist_{level-2}$.

So The p_{hit} in this case is really given by:

$$p_{hit}(z_t^{ko}, x_t^{level-2}|x_t, m_{level-2}) = p_{hit}(z_t^{ko}|x_t, x_t^{level-2}, m_{level-2}) \cdot p(x_t^{level-2}|m_{level-2})$$

Here, $m_{level-2}$ is the pool of level-2 particles, representing a belief about leader's location. Since p_{hit} is independant of $m_{level-2}$ given the nearest level-2 particle location...

$$= p_{hit}(z_t^{ko}|x_t, x_t^{level-2}) \cdot p(x_t^{level-2}|m_{level-2})$$

Substituting the respective values of each term...

$$= \varepsilon_{\sigma_{hit}}(dist_{level-2}) \cdot bel^{level-2}(x_t^{level-2})$$

Now, since the belief over the leader's location at $x_t^{level-2}$, or $bel^{level-2}(x_t^{level-2})$, is given by the weight of the level-2 particle at $x_t^{level-2}$ we can write...

$$= \varepsilon_{\sigma_{hit}}(dist_{level-2}) \cdot Weight^{level-2}(x_t^{level-2})$$

Noise due to sensor failures:

Same as Likelihood Field model. We assign a large likelihood p_{max} to these max range sensor readings, when present.

Noise due to unexplained random measurements:

Same as Likelihood Field model. We assume a constant uniform probability p_{rand}.

False positives in identification of occluded beams:

These can be caused due to things like the presence of physical holes in leader which cause sensor beams to pass through, or due to errors in leader detection mechanism, etc. When they occur, they are usually readings from sensor beams that actually never hit the leader, but are still determined to be occluding beams. These types of readings are hard to detect and model. A way to detect these "false positive" occluding beams is to check for the closest level-2 particle distance. If this distance is larger than a threshold max-occupancy distance, we assume that the sensor reading is not coming from the leader. We use the usual likelihood calculation method for these false-positive sensory readings, instead of using the Advanced Weighting method. *Note: The max-occupancy distance is a parameter that is specified at the* *beginning, and is also used in the standard Likelihood Field model to cut off the Gaussian after a large enough distance when the probability values start becoming negligible.*

2.4 Experiments to test Advanced Weighting

2.4.1 Experimental Setup

We use two robots for our experiments. Both are turtlebots 2.0 robots, equipped with a kobuki base and a kinect sensor that has a laser range finder and a color-sensing camera. The robots have a differential drive mechanism with 3 wheels, which allows for in-place rotation for turns.

The algorithm implementation is done using the open source Robot Operating System (ROS) on Ubuntu Linux [7]. The version specification is ROS Electric on Ubuntu 11.10. The AMCL package from ROS repositories is used as a starting point for the implementation, with modifications added for Advanced Weighting and Nested Particle Filtering. The KLD sampling part of the AMCL package is kept intact to allow for efficient level-1 particle localization, so that the level-1 particle count can be scaled down when the robot is well localized. For the moment, we will not use KLd sampling to adapt level-2 particle population sizes. KLD sampling will be discussed in more detail in chapter 3 . For now, it suffices to say that it is a more efficient version of the MCL algorithm that adapts the level-1 particle count as needed to vastly improve efficiency without compromising localization accuracy.

The external mechanism used for leader detection is a colored blob detection package called "cmvision", which gives us the location of the detected colored blob relative to the camera frame. We use one of the robots as the leader. To make the leader detectable by the follower, we put a distinctly colored light-weight box on top of it, and also put colored paper around it . Based on the relative angle and range at which the leader is detected, we are able to find the relative location of the leader from the follower. This is used for determining which laser beams are hitting the leader. These are the beams which we will use for the Advanced Weighting part.

The experiments are done in simulation using the 3D simulation package Gazebo within ROS. We use two separate 3D environments that are constructed to-the-scale using Google's "Sketchup" software, from the blueprints of University of Georgia's Boyd GSRC building. The two environments used for the simulations are the 1st floor and 5th floor.

2.4.2 Settings used for the experimental runs

The different stages that we use for the experiments are designed to replicate a real world follower scenario that we are concerned with. We ensure that the occlusion is significant and for long durations of time during the experiments. We use different configurations of particle counts at the two levels. We also perform tests without Mixture MCL for the level-2 particles. Thus, we test the effect of all these different configurations on the performance of NPF with Advanced Weighting (or "NPF-AW").

Stages of the experiment

The pattern of experiments on both the 1st floor and 5th floor environments is similar. Each experimental run is divided into 3 parts:

1. **Initial semi-global localization:** The follower robot robot starts off in a location where it cannot see the leader, and it is not localized to begin with. But it has some idea about the general area that it starts in, which is represented by a Gaussian distribution of particles centered in the general area of the follower with a very large covariance. The follower seeks to reach a waypoint, from where it will be able to spot the leader.

2. **Convoy Stage:** Once the follower spots the leader, it will attempt to get into a convoy like formation, trying to follow the leader, maintaining a constant close distance

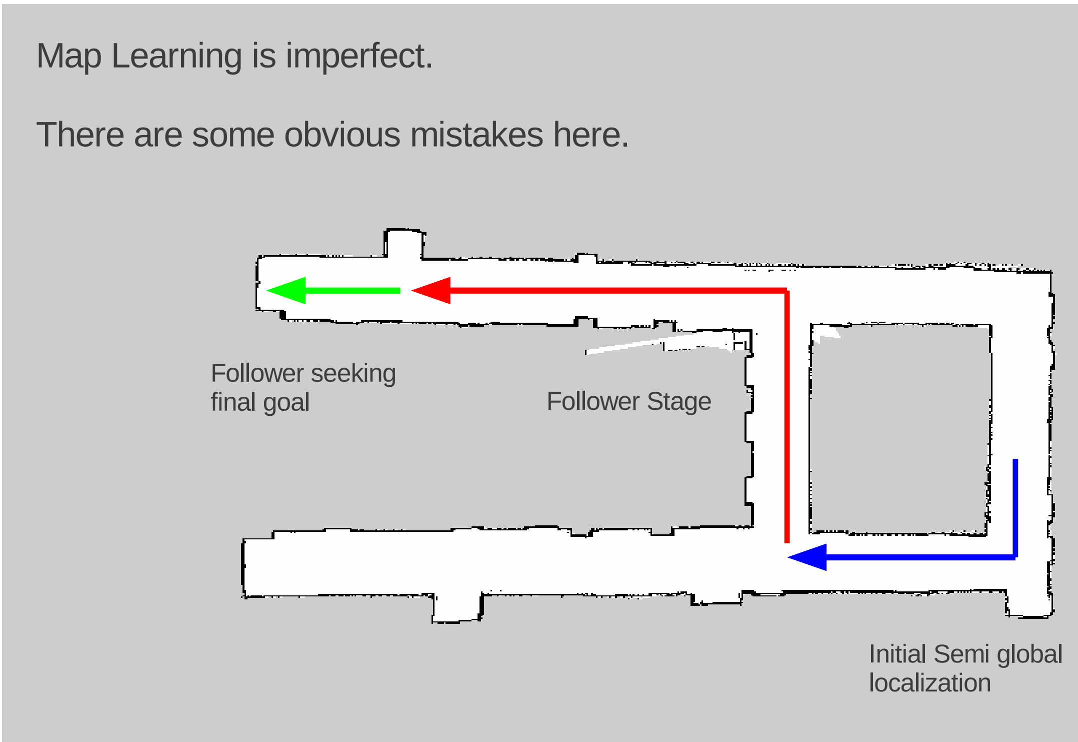


Figure 2.3: Experiment Stages on Floor 1 Learned Map

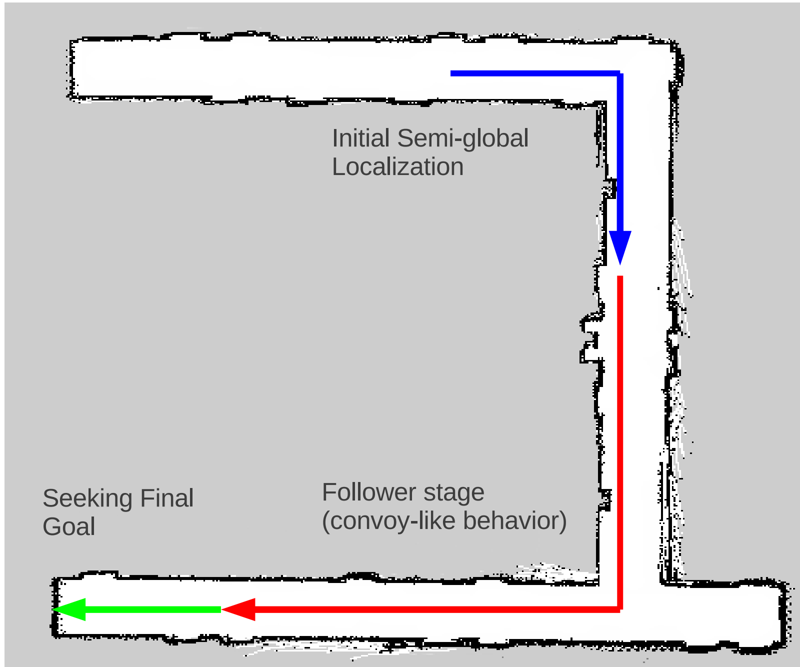


Figure 2.4: Experiment Stages on Floor 5 Learned Map

(between 0.5m-1m) to the leader during this time. The leader will lead the follower to another waypoint.

3. **Seeking final goal:** Upon reaching the general area of the second waypoint, the follower will break away from the convoy and try to reach a final goal location.

   Note: Since the follower might not be well-localized at the end of the convoy stage, we specify a wide region around the second waypoint, which should be consistently detectable by the follower as a cue to start seeking the final goal.

If the follower maintains its localization during this trajectory, it will be able to easily find a path to the final goal and reach it successfully. On the other hand, if its localization gets thrown off, it will either never find a path to the final goal or repeatedly keep trying to find a path until a pre-determined time limit is reached. We set this time limit to be 10 minutes, which is quite generous in both the environments and allows sufficient time for the follower to do multiple attempts at reaching the final goal.

Configurations used

The maximum number of level-1 particles is set to 5000, which was found to be an optimal number necessary for the initial semi-global localization part. Please note that the actual level-1 particle count goes down once the follower localizes itself near the end of the initial semi-global localization stage. The is due to KLD sampling at the 1st level of particles.

The level-2 particle counts per pool are constant throughout, since we have not inculcated KLD sampling for level-2 particles yet. We set different numbers of level-2 particles per level-1 particle to see what effect it has on the performance. The different level-2 particle counts used are 10, 5 and 1.

And lastly, we perform experiments with mixture MCL turned off for the level-2 particles, with 10 level-2 particles per level-1 particle. We perform 50 runs with each combination of particle counts for floor 1, and 30 runs for floor 5.

2.4.3 Performance measures

- Localization accuracy: We measure the Mean Squared Error for level-1 particles of the follower, measured from the true pose of the robot given by the simulator. This is a good measure of the follower's localization accuracy.

- We carry out 50 runs with each configuration for the 1st floor, and 30 runs with each configuration for the 5th floor. We measure the percentage of successful outcomes of each configuration of the experiment.

Final outcomes have the following categories:

Successful: Reached the final goal

Unsuccessful: Unable to find path to final goal

Unsuccessful: Timed out after repeated attempts at reaching final goal

2.4.4 Experimental results and discussion

If we compare simple NPF versus NPf-AW across any of the measures, it becomes obvious that Nested PF with Advanced Weighting (NPF-AW) performs better in every case. This is true for either of the two experimental environments. The point to be noted from graphs 2.6 and 2.8 is that Advanced Weighting needs sufficient level-2 particles in the level-2 pools for maintaining level-1 particle localization. These level-2 particles also need to be tracking the leader sufficiently well in order to maintain level-1 particle localization, as is made obvious by the deterioration in performance when Mixture MCL for level-2 particles is turned off. This means that in an ideal scenario we need atleast 5,000 level-1 particles and 10 level-2 particles per pool. So, for taking full advantage of Advanced Weighting, we need atleast $5,000 + 50,000 = 55,000$ combined total of particles.

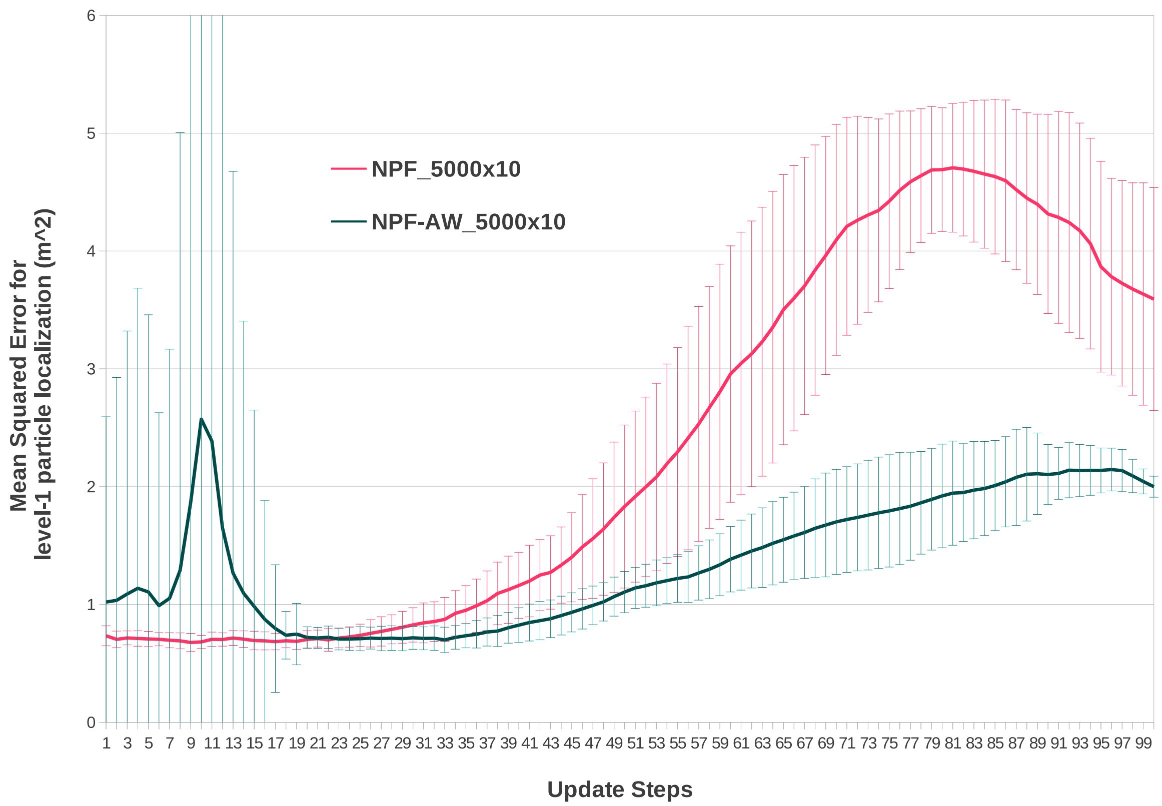


Figure 2.5: Effect of Advanced Weighting on Mean Squared Error of level-1 particles on floor 1

This figure compares Mean Squared Error for level-1 particle localization with and without Advanced Weighting for Floor 1 Experiments. Clearly, the error is maintained at a lower level due to Advanced Weighting as the occlusion persists. The initial spike in localization error for NPF-AW with 5000x10 particles is due to the fact that 55,000 total particles is a huge computational load, which causes the algorithm to run a little slower per-step of the updates. This is why the follower is not always fully localized at the start of the follower phase, causing a spike in initial localization error. This irregularity is also evident from the huge variation in the localization error towards the beginning.

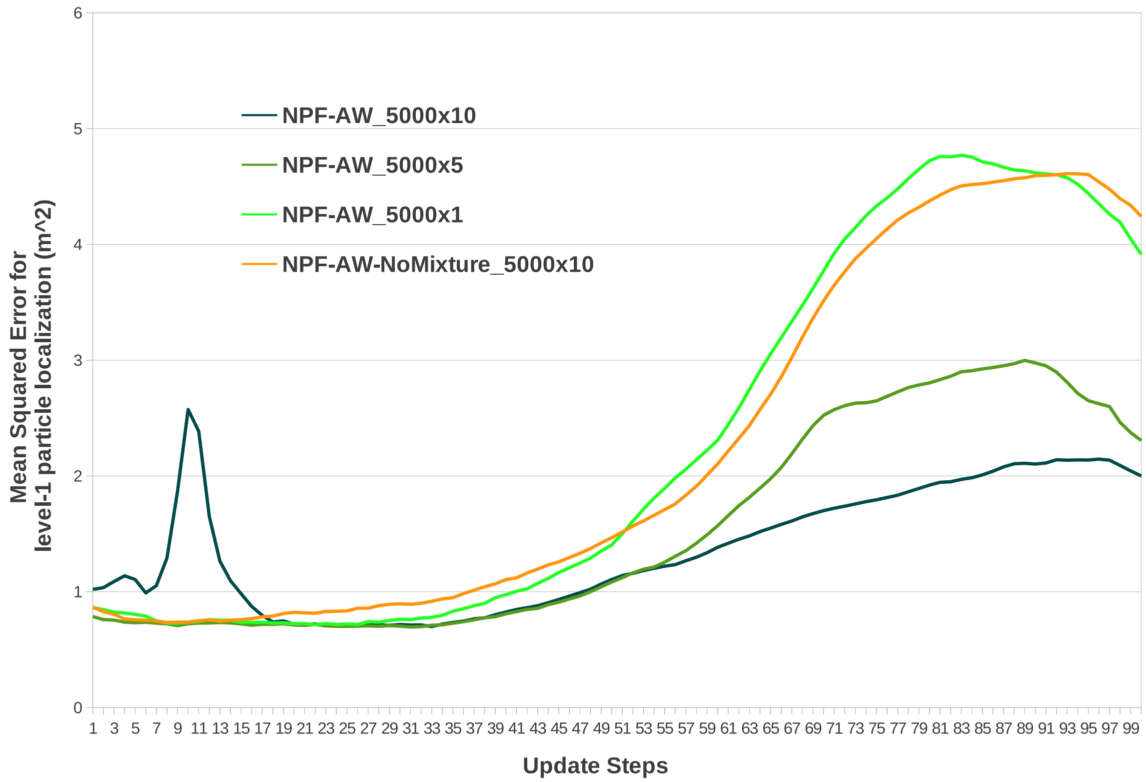


Figure 2.6: Varying configurations with Advanced Weighting on floor 1
Comparison of different configurations of the Nested Particle Filter with Advanced Weighting (NPF-AW) for Mean Squared Error of level-1 particle localization on Floor 1 Experiments. As the tracking gets worse due to fewer level-2 particles or level-2 particles not tracking well, the localization error gets larger.

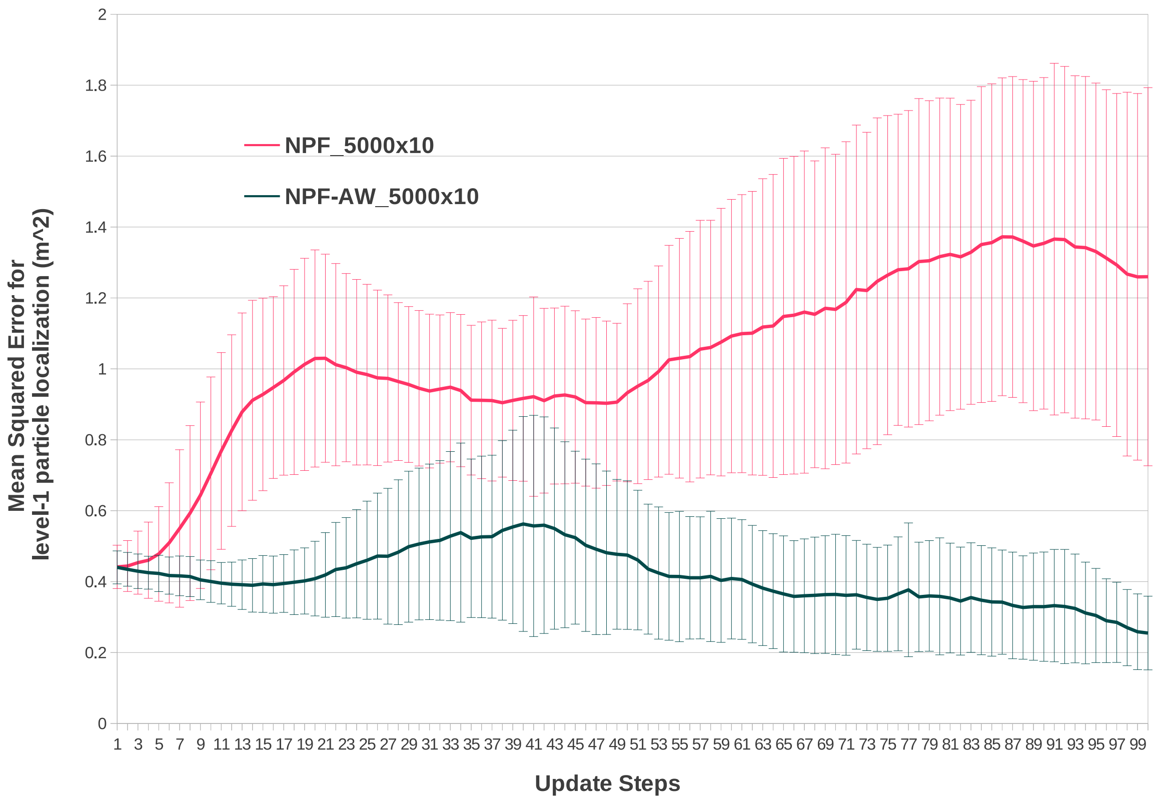


Figure 2.7: Effect of Advanced Weighting on Mean Squared Error of level-1 particles on floor-5

Comparison of Mean Squared Error for level-1 particle localization with and without Advanced Weighting on Floor 5 Experiments. Clearly, the error is maintained at a lower level due to Advanced Weighting as the occlusion persists.

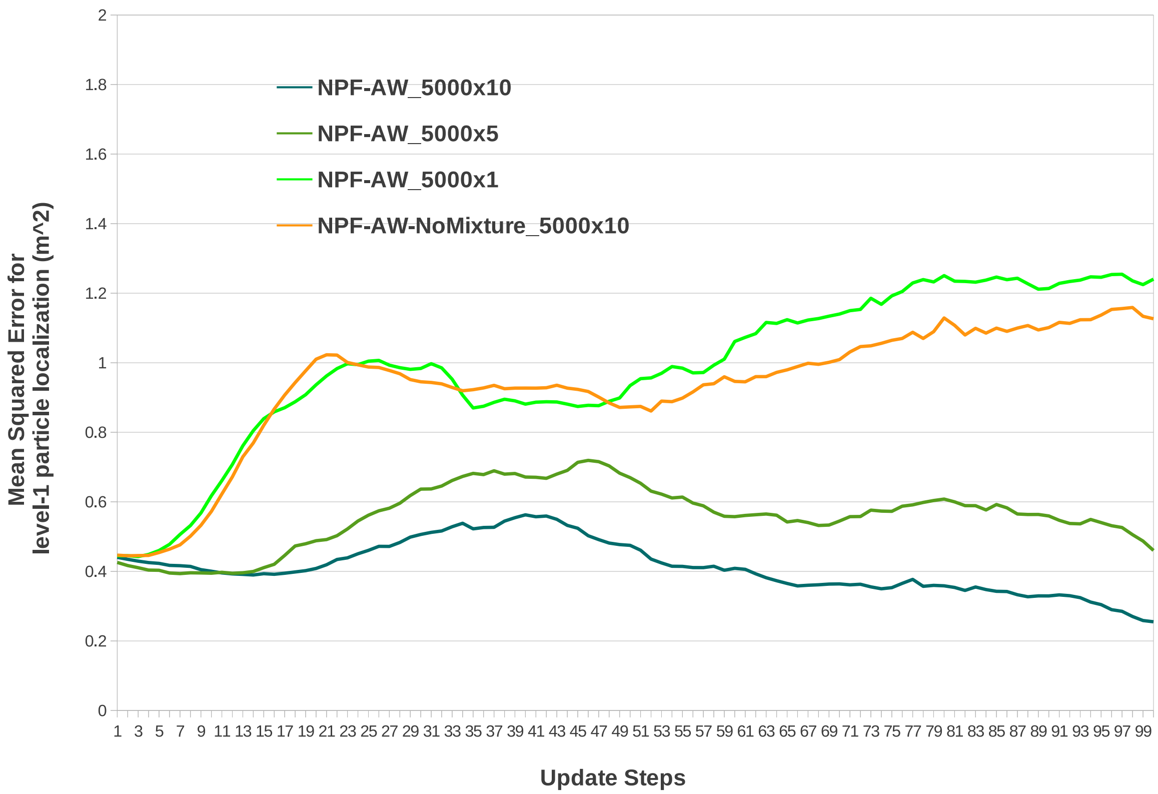
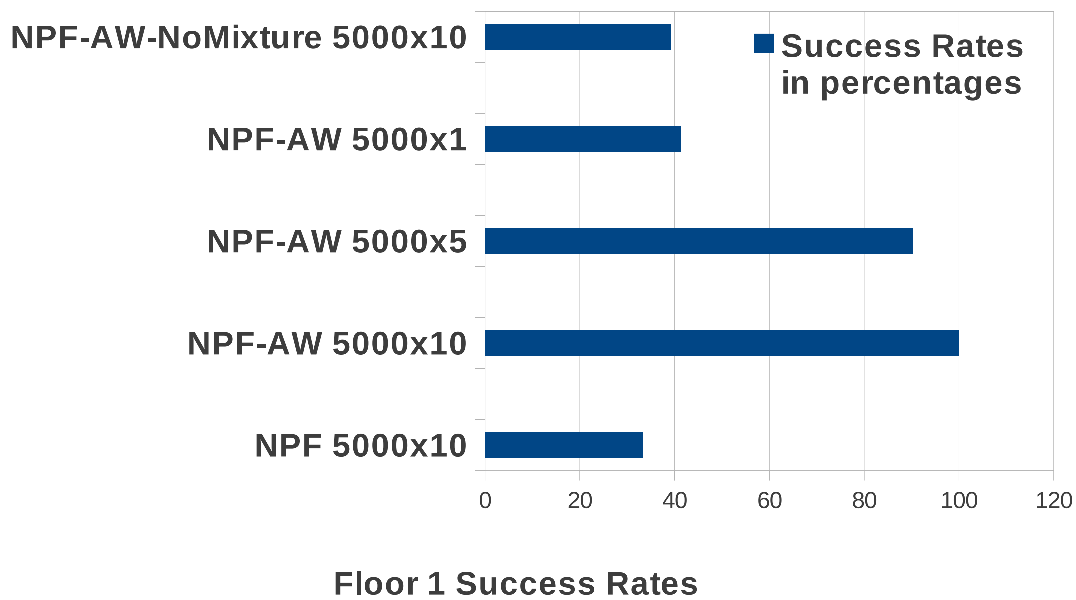


Figure 2.8: Varying configurations with Advanced Weighting on floor 5
Comparison of different configurations of NPF-AW for Mean Squared Error of level-1 particle localization on Floor 5 Experiments. As the tracking gets worse due to fewer level-2 particles or level-2 particles not tracking well, the localization error gets larger.



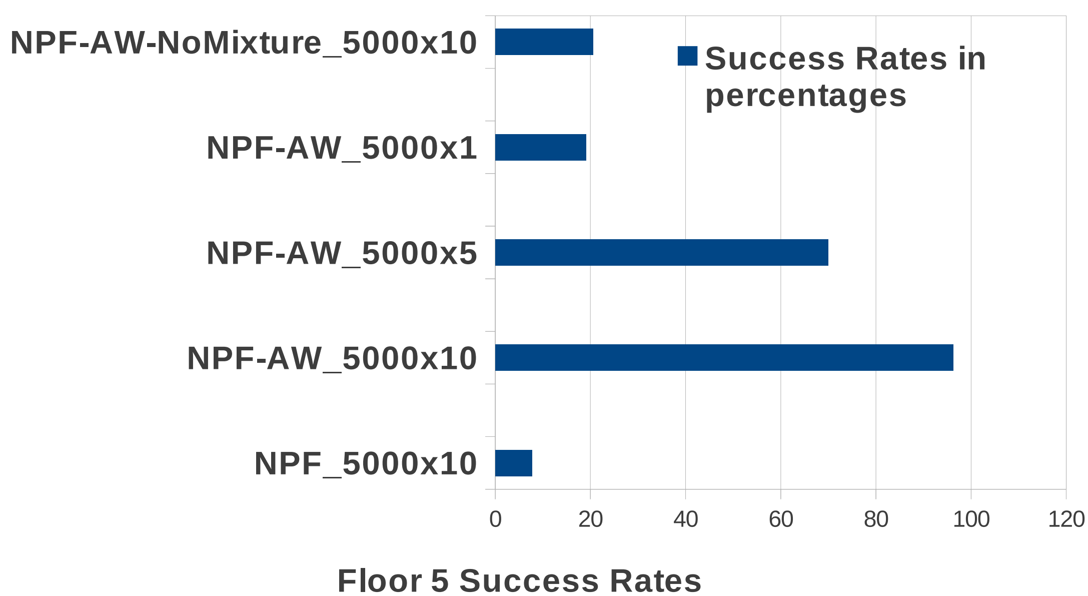


Figure 2.9: Comparison of Success Rates with and without Advanced Weighting
Comparison of Success Rates between various configurations of the NPF-AW approach and NPF (without Advanced Weighting). It is clear that NPF-AW, when provided with sufficient level-2 particles that are tracking the leader well, performs significantly better.

Chapter 3

Adaptive NPF

One of the main concerns with Nested PF is that the number of particles needed to have viable tracking as well as localization is extremely high. The usual Nested Particle Filter as described in section 1.2 does not scale well for larger pools of level-2 particles or if we want to track more than one other robot. Take for example, the best performing configuration from the experiments at the end of chapter 2. We needed to use 5000 level-1 particles and 10 level-2 particles per level-1 particle, bringing the total combined particle count to
5,000 level-1 + 5,000x10 level-2 = 55,000 total particles
Besides, during the initial stage of the experiments, the level-2 particles were not really much useful since we were still trying to localize the level-1 particles before we could even attempt to track the leader. 55,000 is an extremely large number of particles to update at every iteration. Moreover, what if we needed to scale this up to track another leader? What if we needed to insert another level of nesting? The number of particles required goes up exponentially with each new level of nesting. The problem here is that we are using a static count of particles that is set at the beginning as an ad-hoc parameter to the algorithm. In this chapter we introduce a way of making this approach more scalable.

3.1 Adaptive MCL using KLD sampling

KLD sampling has been proved to be an excellent method for determining how many particles are needed to approximate the true posterior to a sufficient degree of accuracy [1, 8]. The name KLD-sampling comes from Kullback-Leibler Divergence, which gives a measure of the difference between two probability distributions. In KLD sampling, the logic is that given a statistical bound on the quality of an approximation of the true posterior distribution, we can determine how many samples are needed to achieve that quality. The bounds specified are thus: If we are given an error ε and a probability $1-\delta$, then KLD sampling can determine how many samples we need so that with probability $1-\delta$ the error between the true posterior and the sample-based approximation is less than ε. ε and δ are provided as parameters to the KLD-sampling algorithm, and for each iteration of filtering, KLD sampling tells us how many samples we need to generate.

We then only resample as many particles as we need. This allows us to scale down the number of level-1 particles ($robot_i$'s particles) as they start localizing well. This frees up more space and computational power to insert more particles into tracking the leader ($robot_j$). Thus, instead of specifying the particles at either level of the particle filter, we just specify one combined population limit which represents the maximum combined number of particles that we can handle. Initially we use all these particles to only localize the follower, not allocating any level-2 particles to track the leader. The heuristic assumption here is that if the follower itself is not localized first, then it's very difficult to accurately track the leader at the same time. Once the follower is localized to such a degree that its particle count gets reduced sufficiently, then that space becomes available to insert level-2 particles. Thus, we start inserting level-2 particles that will track the leader.

This allows us to dynamically scale the particle counts up or down at both the levels. It makes the algorithm a lot more efficient. It also allows us to insert a lot more particles at the level-2 particle level than what we can afford to set when using static population limits. Thus, it becomes possible to track the leader using a lot more particles, and improves tracking.

3.1.1 KLD Sampling

There is an efficient implementation of the KLD sampling algorithm that has been inculcated into MCL. It works on-line during the resampling step, using a moving threshold to determine after generating each new sample if more samples need to be generated.

This moving threshold for KLD sampling, as proved in [1], is given by:

$$M_x := \frac{k-1}{2\varepsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\}^3 \tag{3.1}$$

Where,

M_x is the moving threshold

k is the number of map grid cells with atleast one particle

ε is the permissible error

$1-\delta$ is the probability of the error

$z_{1-\delta}$ represents the upper $1-\delta$ quantile of the normal distribution

3.1.2 Adaptive MCL algorithm

The threshold in equation 3.1 is used to determine after a new sample is taken, whether we need to generate more samples. A concise version of the algorithm using this threshold is as follows:

1. Propagate all the particles forward using the motion model
2. Weight the particles using observations
3. Resampling step:
   (a) Sample a new particle based on the probability proportional to the particles weight
   (b) Add the new sample to the current pool of particles.
   (c) Check to see if the threshold for KLD sampling is reached, or if the maximum particle population is reached.
   If either of the two is reached, stop.
   Otherwise sample more particles.

3.2 KLD Sampling for Nested PF

We use the KLD sampling algorithm discussed in section 3.1.2 and adapt it for nested particle filtering. We have to make a few modifications to the resampling step of the Nested Particle Filter. Recall that resampling in NPF happens like a Depth First Search tree traversal. The level-1 particle gets resampled first, immediately followed by resampling of level-2 particles within that particular level-1 particle, before moving on to other level-1 particles.

Now, the problem with KLD sampling in this approach is that we cannot find out how many particles we need at the 1st level of particles until we finish resampling level-1 particles. Thus, if the level-1 particle count is adaptively reduced due to better localization, we have no way of knowing this until the end of level-1 particle resampling. But by this time, level-2 particles will already have been resampled if we follow the DFS traversal strategy.

In order to take advantage of any changes in level-1 particle counts, and insert more level-2 particles accordingly, we need to change our resampling strategy from a Depth First to a Breadth First traversal. So, all the level-1 particles will get resampled first, without touching the level-2 particle pool within each level-1 particle. After resampling of level-1 particles ends, we get a count of the level-1 particles that have been resampled. We then go one level down, and resample level-2 particles from each level-2 particle pool.

We modify the maximum permissible population limit per-level-2-pool of the level-2 particles according to the following formula:

$$population\_limit_{level-2} = \frac{population\_limit_{total} - currrent\_count_{level-1}}{currrent\_count_{level-1}} \tag{3.2}$$

Where,

$population\_limit_{level-2}$	is the maximum permissible level-2 population that can be supported
$population\_limit_{total}$	is the combined population limit that is set at the beginning
$currrent\_count_{level-1}$	is the current level-1 particle count after level-1 particle resampling

Note that the chief function of the above formula is to calculate how much space remains after resampling level-1 particles, and divide it equally among all the level-2 pools.

Using the above formula and the BFS traversal strategy,we can now show how KLD sampling can be used for varying population limits of level-2 particles as part of the NPF resampling step. The Propagation and Weighting steps in NPF remain as they are, and we can still use Advanced Weighting as-is.

Adaptive Resampling for NPF-

1. Resample all level-1 particles first, keeping their corresponding level-2 particle pools intact.

2. Calculate the new population limit for level-2 particles ($opulation\_limit_{level-2}$) according to the equation 3.2.

3. Loop through each resampled level-1 particle

   (a) Go to the level-2 pool of the current level-1 particle

   (b) Resample from this level-2 pool until either the KLD threshold is reached or $population\_limit_{level-2}$ is reached

4. Stop.

Using this adaptive approach for level-2 particle resampling, the Nested Particle Filter can be made highly scalable. We will perform experiments using this Scalable NPF algorithm combined with Advanced Weighting to measure the efficiency of this approach.

3.3 Experiments to test Adaptive NPF with Advanced Weighting

3.3.1 Experimental setup

The experimental setup for testing Adaptive NPF with Advanced Weighting (ANPF-AW) is the exact same as for the earlier (NPF-AW), including tests with mixture MCL turned off. The only difference we make in the configuration is that we vary the combined maximum population instead of the level-2 particle counts. The 3 different settings used for ANPF-AW are with 5000, 750 and 400 max combined population. Please note that 750 and 400 populations are not sufficient for consistently localizing even level-1 particles. We encountered about 25% and 70% success rate for localization with 400 and 750 particles respectively in the initial semi-global localization stage. These success rates are too low, especially considering that we get close to 100% success rate with 5000 particles. So, using such a low particle counts is not advisable in general. But, since we are only concerned with the convoy-stage localization performance, we use these acutely low particle counts with the specific purpose of hindering the performance of the ANPF-AW algorithm during the convoy stage. Hence, we only consider the experimental runs where the initial semi-global localization does succeed, and the follower moves to the convoy stage.

For each different setting we perform 50 runs with floor 1 and 30 runs with floor 5 as before.

3.3.2 Experimental results and discussion

The main aim of experiments to test Adaptive NPF with Advanced Weighting is to measure its performance in comparison to the simple NPF with Advanced Weighting. This is because except for the adaptive resampling that we use in ANPF-AW, the remaining algorithm is identical to NPF-AW. We are also going to compare the performance with the basic NPF algorithm to highlight the performance improvement.

In addition to Mean Squared Error of level-1 particles and the success rates we measured earlier, we also measure the performance of tracking itself. We will compare the percentage of level-2 particles that are within 1m of the true pose of the leader. This is a good measure of the tracking performance. Also, since it is a comparison of proportions of level-2 particles, the variation in total level-2 particle counts is accounted for. We show that the tracking performance is consistently better for Adaptive NPF-AW than simple NPF-AW. This also makes it obvious how the tracking performance deteriorates when we turn off Mixture MCL for level-2 particles.

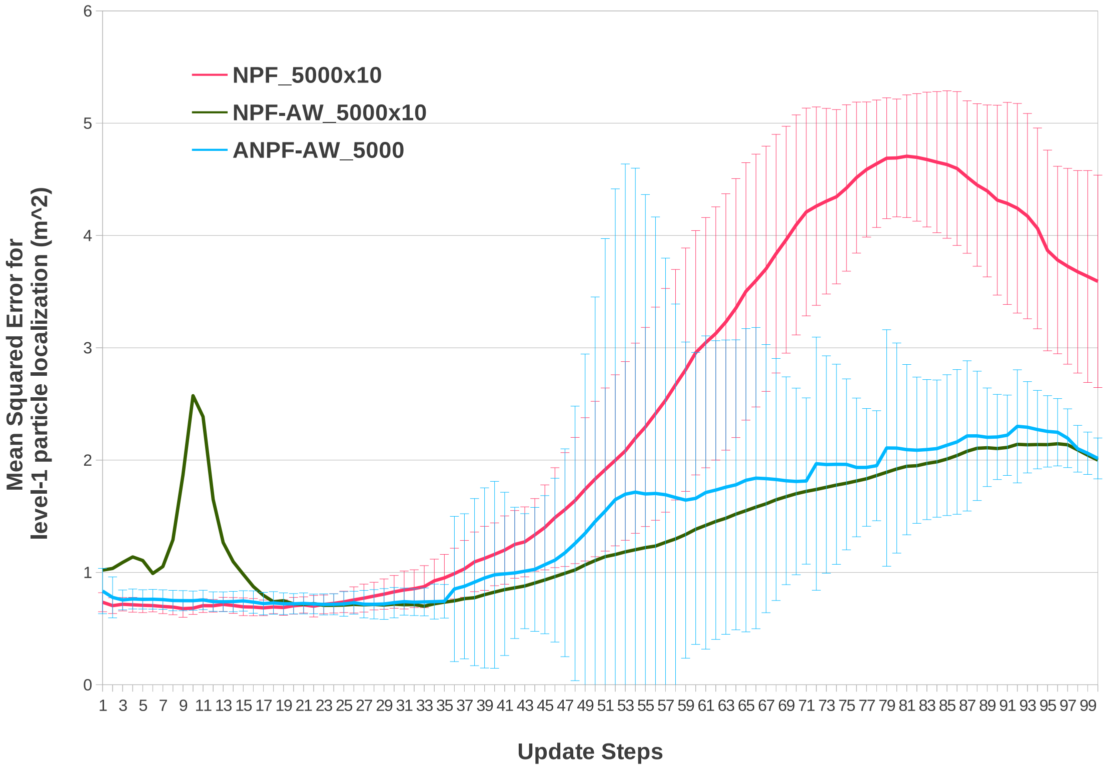


Figure 3.1: Mean Squared Error of level-1 particles on floor 1 using Adaptive NPF with Advanced Weighting
This figure compares Mean Squared Error for level-1 particle localization with and without Adaptive NPF for Floor 1 Experiments. We can see that the performance of Adaptive NPF with Advanced Weighting is similar to that of NPF-AW when given enough particles. In fact, with ANPF-AW we need almost 10 times fewer particles than with NPF-AW for similar performance.

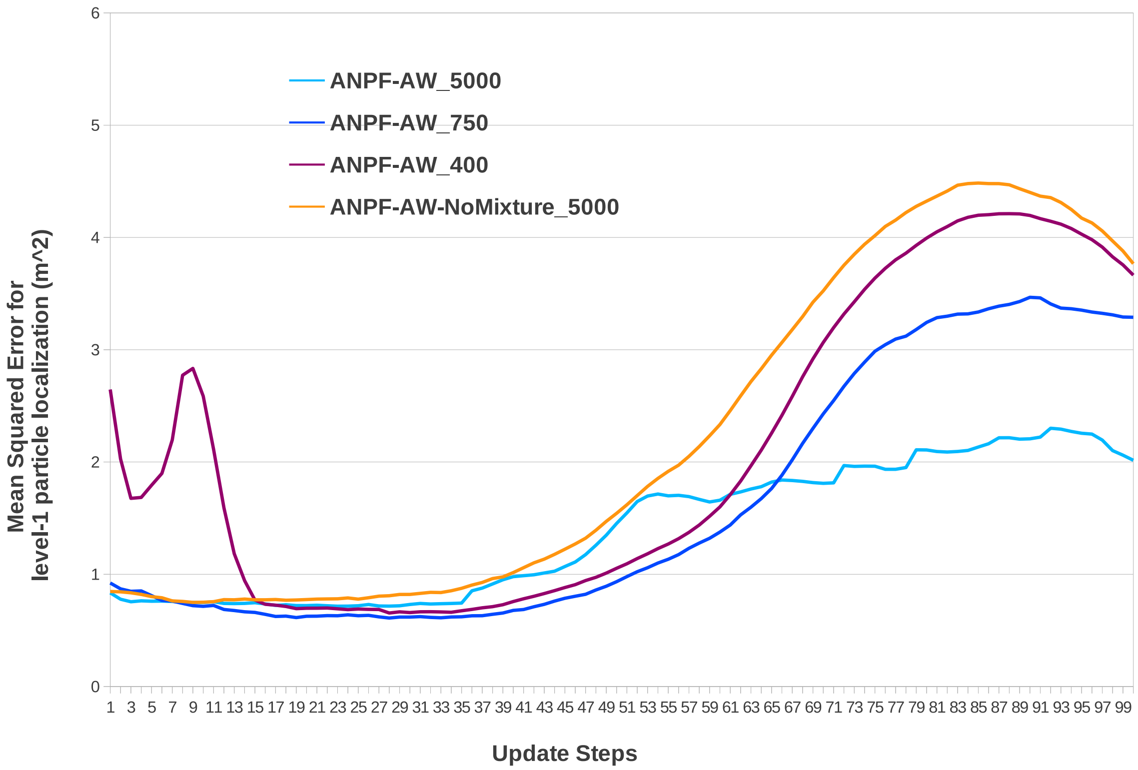


Figure 3.2: Mean Squared Error of level-1 particles on floor 1 with different configurations for Adaptive NPF with Advanced Weighting
This figure compares Mean Squared Error for level-1 particle localization using ANPF-AW for Floor 1 Experiments. We are comparing performance with the different configurations. It is easy to see that ANPF-AW relies on sufficient level-2 particle counts for maintaining localization as well. This is evidenced by the deterioration in performance when the total particle count is acutely low. Similarly, it also performs poorly when the level-2 particles are not tracking the leader well enough, as is seen when the Mixture MCL for level-2 particles is turned off.

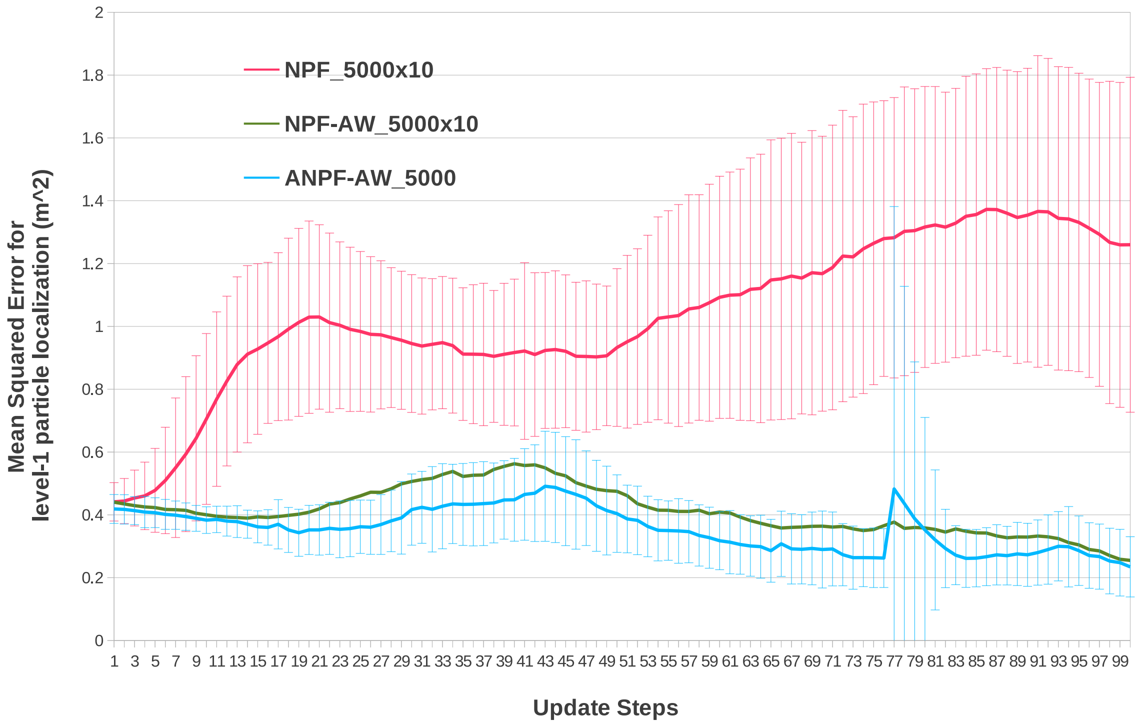


Figure 3.3: Mean Squared Error of level-1 particles on floor 5 using Adaptive NPF with Advanced Weighting
This figure compares Mean Squared Error for level-1 particle localization with and without Adaptive NPF for Floor 5 Experiments. We can see that the performance of Adaptive NPF with Advanced Weighting is similar to that of NPF-AW when given enough particles. In fact, with ANPF-AW we need almost 10 times fewer particles than with NPF-AW for similar performance.

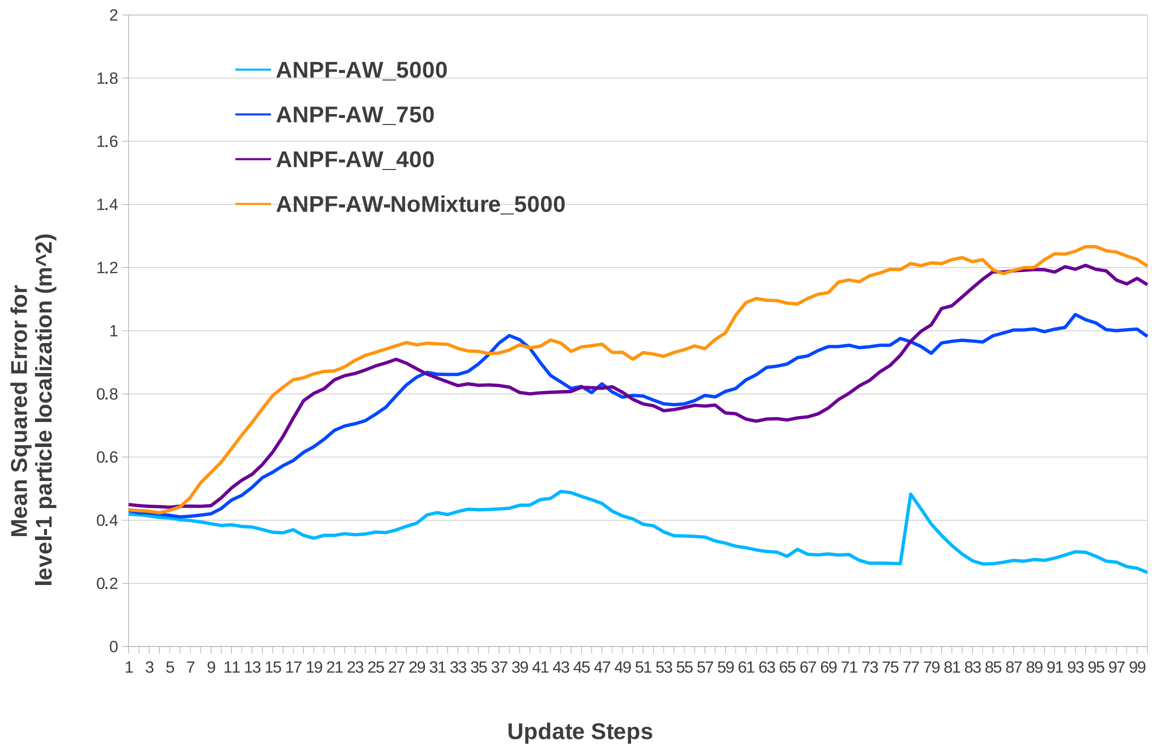


Figure 3.4: Mean Squared Error of level-1 particles on floor 5 with different configurations for Adaptive NPF with Advanced Weighting
This figure compares Mean Squared Error for level-1 particle localization using ANPF-AW for Floor 5 Experiments. We are comparing performance with the different configurations. It is easy to see that ANPF-AW relies on sufficient level-2 particle counts for maintaining localization. This is evidenced by the deterioration in performance when the total particle count is acutely low. Similarly, it also performs poorly when the level-2 particles are not tracking the leader well enough, as is seen when the Mixture MCL for level-2 particles is turned off.

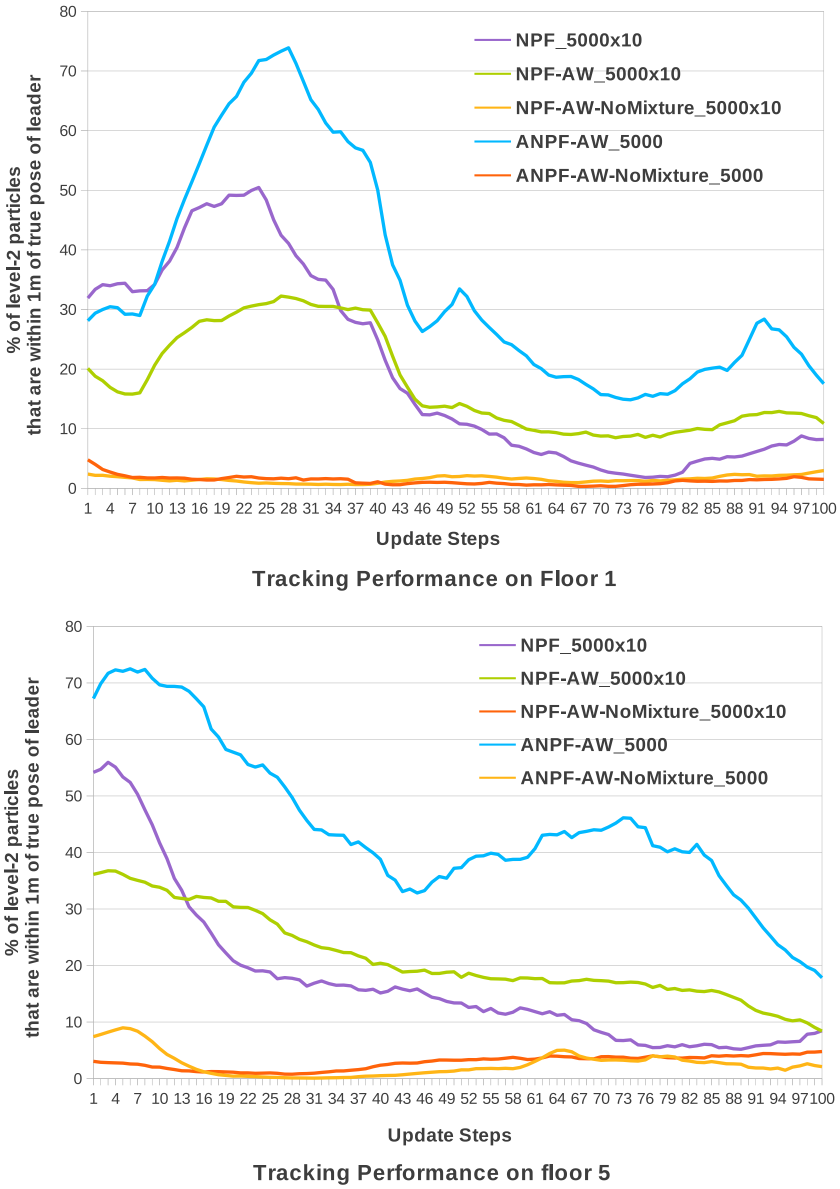


Figure 3.5: Tracking Performance on both floors

This figure compares the tracking performance of ANPF-AW with other configurations. It is quite clear from the above two graphs that tracking performance gets improved significantly with ANPF-AW. The ANPF-AW graph dominates all other approaches and configurations. This is even more notable since we are measuring the "percentage" of level-2 particles that are within 1m of true pose. This means that this performance is relative, regardless of how many total level-2 particles are present. The actual count of level-2 particles within 1m will be drastically higher for ANPF-AW since we have more level-2 particles in this approach overall.

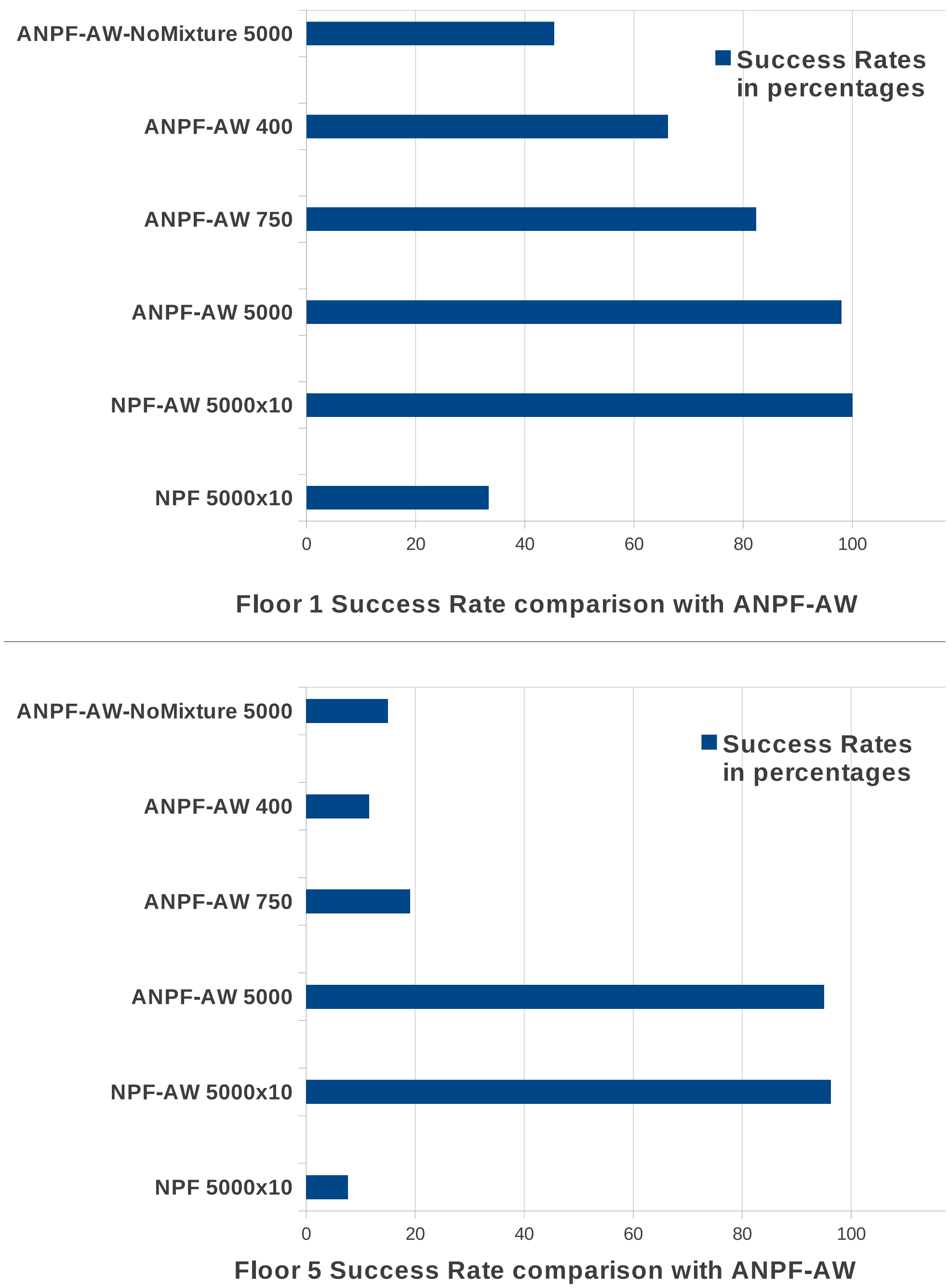
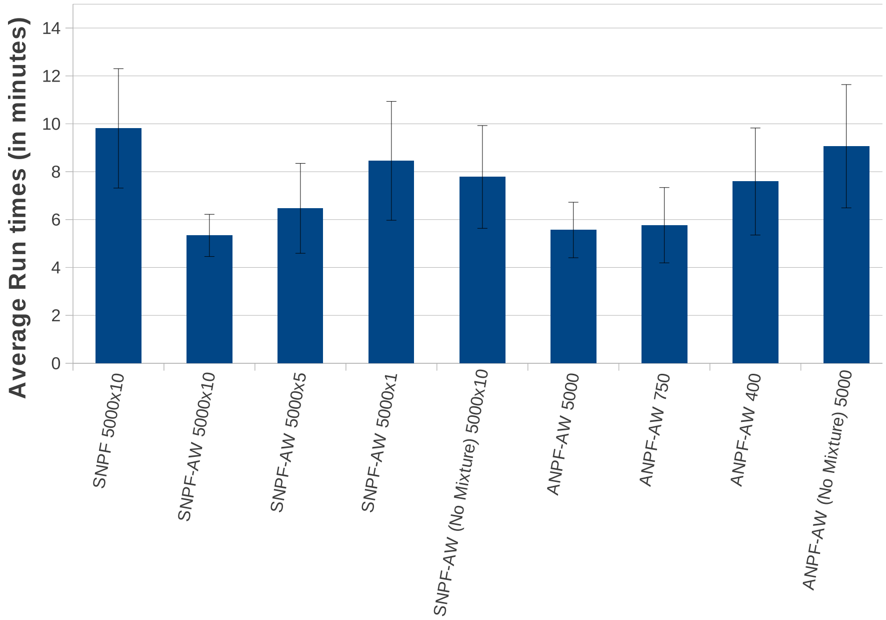
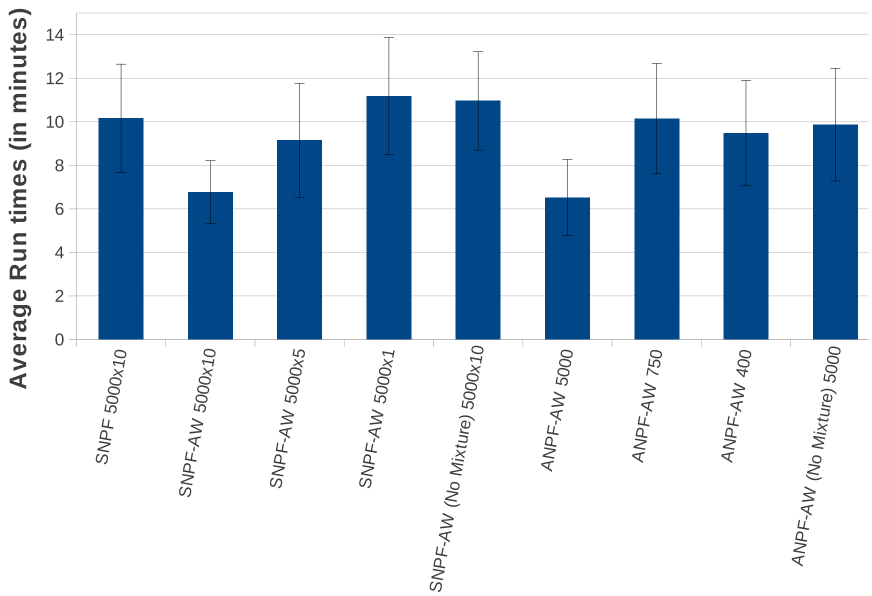


Figure 3.6: Comparison of Success Rates with Adaptive NPF-AW
Comparison of Success Rates between various configurations of the ANPF-AW approach and NPF and NPF-AW. We can see that ANPF-AW can perform just as good as NPF-AW with total number particles that is almost 10 times lower than NPF-AW.



Run Times for Different Algorithms and Configurations on floor 1



Run Times for Different Algorithms and Configurations on floor 5

Figure 3.7: Experiment run times for different algorithms and settings

These charts compare the run times of each algorithm and configuration on both the floors. We can see that the algorithms and configurations that have better success rates are also the ones that have shorter average run times. This means they finish faster. Since they also have higher success rates, these charts show that on an average they succeed, and succeed quickly.

Environment	**Simple NPF** [no Advanced Weighting, non-adaptive]	**Adaptive NPF with Advanced Weighting**
Floor 1	1 out of 10 : 10%	8 out of 10 : 80%
Floor 5	2 out of 10: 20%	6 out of 10: 60%

Figure 3.8: Comparison of success rates in physical runs
Comparison of simple NPF and Adaptive NPF with Advanced Weighting on both the floors. The rates are lower than in simulations, but still consistent with expectations. The likely reason for lower success rates in physical runs is due to noisier environmental factors than in simulations. Things like lighting conditions, terrain discrepancies, etc. cause greater noise in physical environments.

3.3.3 Physical Experiments

The final algorithm, Adaptive Nested Particle Filter with Advanced Weighting (ANPF-AW) was implemented and tested on a physical robot. We performed 10 runs each with ANPF-AW and the basic NPF algorithm, using the best configurations from simulations for each. We used 5,000 level-1 particles and 10 level-2 particles per pool for the simple NPF algorithm. For ANPF-AW we used 5,000 combined total of particles.

Since we cannot know the true poses when performing physical runs, we used the success rate to measure performance. We used the actual 1st floor and 5th floor corridors for our physical experiments, with the same configurations as in the simulations. Refer to chart 3.8 for success rates of the two algorithms in physical runs.

These success rates are slightly different from the simulated experiments, but not inconsistent with expectations. This validation by physical experiments is further proof that ANPF-AW is a significant improvement over the usual NPF.

Chapter 4

Conclusion and future work

From all the experimental results and performance evaluations shown in this thesis, it becomes clear that our final algorithm "Adaptive NPF with Advanced Weighting" or "ANPF-AW" performs significantly better than the existing algorithms under extreme sensory occlusion, and is highly scalable compared to the usual Nested Particle Filtering approach. This algorithm includes two innovations. The first one is Advanced Weighting which is a method for maintaining localization under extreme sensory occlusion. The second contribution is the Adaptive resampling approach that has been adapted for Nested Particle Filters. This second contribution makes Nested MCL highly scalable, and improves tracking performance.

Some of the things that can be done in the future with this approach are to test it in an even more rigorous setting and evaluate it's performance. For example, we could try tracking multiple other robots, with a more crowded environment involving many more humans generating sensory noise.

Also, the advanced weighting part can be made even more efficient by using heuristic models for nearest-neighbor detection of level-2 particles. This could speed up the advanced weighting calculations significantly.

Bibliography

[1] Dieter Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 22, 2003.

[2] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.

[3] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2432–2437. IEEE, 2005.

[4] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, 2007.

[5] Emanuele Menegatti, Alberto Pretto, and Enrico Pagello. Testing omnidirectional vision-based monte carlo localization under occlusion. In *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2487–2493. IEEE, 2004.

[6] Anousha Mesbah and Prashant Doshi. Individual localization and tracking in multi-robot settings with dynamic landmarks. In Francien Dechesne, Hiromitsu Hattori, Adriaan ter Mors, JoseMiguel Such, Danny Weyns, and Frank Dignum, editors, *Advanced Agent Technology*, volume 7068 of *Lecture Notes in Computer Science*, pages 277–280. Springer Berlin Heidelberg, 2012.

[7] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[8] Dieter Fox Sebastian Thrun, Wolfram Burgard. *Probabilistic Robotics*. The MIT Press, Cambridge, Massachusetts, 2006.

[9] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Monte carlo localization for mobile robots. In *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*, 1999.

[10] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. 2001.