# DESIGN AND IMPLEMENTATION OF MULTI-AGENT VISUAL-SLAM ALGORITHMS ON AUTONOMOUS ROBOTS

by

Nezih Ergin Özkucur

B.S., in Computer Engineering, Marmara University, 2006

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2009

# ACKNOWLEDGEMENTS

First of all, I would like to thank to my supervisor Prof. H. Levent Akın for his endless patience, support and encouragement during this research. This thesis would not be possible without his comprehensive support.

I also would like to thank to my thesis committee members Asst. Prof. A. Taylan Cemgil for his presence in our meetings and indispensable comments about this work and Assoc. Prof. Borahan Tümer for his influence and discussions on my career decisions.

I am also grateful for the support of Prof. Okyay Kaynak who provided their robot platform for this thesis.

I have met the most outstanding friends in AILAB. Their friendship and help made everything easier.

I would like to express my deepest gratefulness to my parents for their endless self-sacrifice, patience and support.

I also would like to thank Ebru Toker, who always stand by me and supported all the time.

This Thesis has been supported by the Scientific and Technological Research Council of Turkey (TUBİTAK) 2228 National Graduate Scholarship Program. This research also has been supported by Boğaziçi University Research Fund Project under grant BAP06HA102 and TUBİTAK under grant 106E172.

# ABSTRACT

## DESIGN AND IMPLEMENTATION OF MULTI-AGENT VISUAL-SLAM ALGORITHMS ON AUTONOMOUS ROBOTS

The Simultaneous Localization and Mapping (SLAM) problem is one of the most challenging problems in robot navigation. The problem addresses autonomously exploring and mapping an unknown environment without prior knowledge (of features). The robot should generate the map of the environment and estimate its pose with respect to the map. An extension of this problem to the distributed multi-robot platform is a popular research topic for its challenges and commitments. Multiple cooperative robots exploring an area would decrease exploration time and increase the accuracy.

This thesis introduces the application of two successful SLAM solution techniques to the multi-robot domain using visual sensors and non-unique landmarks. There are two contributions to the literature: Evolutionary Strategies (ES) is used to calibrate the parameters of the Extended Kalman Filter-SLAM (EKF-SLAM) method with supervised data, and a novel map merging method with uncertainty propagation is introduced for the Fast-SLAM algorithm. The developed algorithms are tested in both simulated and real robot experiments and the improvements and applicability of the developed methods are shown with the results.

# ÖZET

## ÇOK ROBOTLU GÖRSEL SLAM YORDAMLARININ OTONOM ROBOTLAR ÜZERİNDE TASARIM VE GERÇEKLENMESİ

Eş zamanlı yer bulma ve haritalama (SLAM) problemi robot gezinmesi alanındaki en zor problemlerden biridir. Bu problem, ortam hakkında önceden bir bilgi verilmeden, bilinmeyen bir ortamın otonom olarak keşfedilmesini ve haritalanmasını içerir. Robot, ortamın haritasını çıkarmalı ve aynı zamanda kendi yerini bu haritaya göre tahmin etmelidir. Bu problemin dağıtık çoklu robotlar uzerine genişletilmesi de zorlukları ve vaatleri sebebiyle popüler bir konudur. Aynı ortamı yardımlaşarak gezen çoklu robotlar keşfetme süresini azaltabilir ve tahminin doğruluğunu artırabilir.

Bu tez, görsel algılayıcıları olan ve aynı işaretçili haritada çoklu robotlar üzerinde en başarılı SLAM çözme yöntemlerinin uygulanmasını sunar. Literatüre iki katkı sağlanmıştır: EKF-SLAM yönteminin parametreleri denetlemeli veri ile Evrimsel Stratejiler yöntemi tarafından kalibre edilmiştir. Yeni bir belirsizlik yayılması ile harita birleştirme yöntemi Fast-SLAM yöntemi için sunulmuştur. Deneyler hem benzetim ortamında hem de gerçek ortamda denenmiş ve geliştirilen yöntemlerin ilerleme ve uygulanabilirliği test sonuçları ile gösterilmiştir.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS

| | |
|---|---|
| $t$ | Time step at the execution of a discrete time system |
| $x_t$ | State vector at time $t$ |
| $z_t$ | Observation vector at time $t$ |
| $u_t$ | Control input at time $t$. |
| $bel(x_t)$ | Belief state at time $t$ |
| $\mu_t$ | Mean vector of the belief state at time $t$ |
| $\Sigma_t$ | Uncertainty covariance of the belief state at time $t$ |
| $A_t$ | Linear process model matrix at time $t$ |
| $B_t$ | Linear control input model at time $t$ |
| $\epsilon_t$ | Process noise vector at time $t$ |
| $R_t$ | Process noise covariance at time $t$ |
| $P$ | Initial error covariance of the state. |
| $\delta_t$ | Observation noise vector at time $t$ |
| $Q_t$ | Observation noise covariance at time $t$ |
| $C_t$ | Linear measurement model matrix at time $t$ |
| $K_t$ | Kalman Gain at time $t$ |
| $g(x_{t-1}, \mu_t)$ | Non-linear process function. |
| $h(x_t)$ | Non-linear observation function. |
| $G_t$ | Jacobian of $g()$ with respect to $x_t$ at time $t$. |
| $H_t$ | Jacobian of $h()$ with respect to $x_t$ at time $t$. |
| $W_t$ | Jacobian of $g()$ with respect to $\epsilon_t$ at time $t$. |
| $V_t$ | Jacobian of $h()$ with respect to $\delta_t$ at time $t$. |
| $S_t$ | Cross-correlation matrix and state. |
| $X_t$ | Sigma point set in UKF, particle set in Particle Filter. |
| $Z_t$ | Observation set. Observation sigma point set in UKF. |
| $K$ | Number of observations. |
| $w_i$ | Importance weight of particle $i$. |
| $N$ | Number of particles. |
| $p$ | Robot pose. |

| | |
|---|---|
| $p_x$ | $x$ coordinate of the robot pose. |
| $p_y$ | $y$ coordinate of the robot pose. |
| $p_\theta$ | Orientation of the robot. |
| $\theta_i$ | Orientation of the $i$th landmark with respect to the robot. |
| $l_i$ | Distance of the $i$th landmark to the robot. |
| $\Delta x$ | Displacement of a robot in the $x$ axis. |
| $\Delta y$ | Displacement of a robot in the $y$ axis. |
| $\Delta\theta$ | Change in the orientation of a robot. |
| $M$ | Map. |
| $m_i$ | $i$th entity in the map. |
| $p_x^i$ | $x$ coordinate of the $i$th landmark. |
| $p_y^i$ | $y$ coordinate of the $i$th landmark. |
| $tr_x$ | Translation parameter on $x$ axis. |
| $tr_y$ | Translation parameter on $y$ axis. |
| $tr_\theta$ | Rotation parameter between coordinate frames. |
| $c$ | Grid map matrix. |
| $c_{i,j}^t$ | Probability of existence of an obstacle in cell $(i, j)$ at time $t$. |

# LIST OF ABBREVIATIONS

| | |
|---|---|
| KF | Kalman Filter |
| EKF | Extended Kalman Filter |
| UKF | Unscented Kalman Filter |
| PF | Particle Filter |
| SLAM | Simultaneous Localization and Mapping |
| V-SLAM | Visual Simultaneous Localization and Mapping |
| ES | Evolutionary Strategies |
| SIFT | Scale Invariant Feature Transform |

# 1. INTRODUCTION

One of the most challenging problems in robot navigation is the simultaneous localization and mapping problem (SLAM). The SLAM problem has attracted many researchers in the field of mobile robotics. The problem addresses a robot generating a map of an unknown environment and localizing itself in this map.

In the localization problem, the robot is given a sequence of possibly noisy landmark observations, and for all the observations, the location of the landmarks are known more or less prior to execution. However, if the landmark positions are not available, the robot should both estimate the real positions of the landmarks and localize itself with the estimated landmarks. From the other point of view, in the mapping problem, the position of the robot is given, and it estimates the positions of the unknown landmarks. If the robot position and orientation (pose) is not given, the robot needs to estimate its pose using the landmarks. This is a chicken-egg problem and since the odometry readings are noisy in real applications, the error in the estimations are unbounded. The problem attracts researchers because the solution provides more autonomy to robots and more realistic application domains.

The SLAM problem addresses the estimation of both unknown landmarks in the environment and the robot pose with noisy observations and odometry readings. Depending on the application domain, the environment features are unknown and the observations are indistinguishable, meaning that the landmarks may not be unique. Most of the solution methods to this problem make probabilistic estimations with different assumptions on probability distributions of both the noise and the estimations.

The scanning range finders are suitable for mapping applications, however visual sensors are cheaper and can provide more information of the environment than range finders. However, observing a feature in an image of unknown environment requires Computer Vision algorithms which are computationally demanding methods. The SLAM problem becomes more interesting when it is distributed to more than one

robot. The robots do not know the initial positions of each other, they do not have a common coordinate frame but they are aware of each other and they collaborate to form the map.

The SLAM problem may be in many forms according to the application domain or the task of the robot. The robot may be sent to explore a cave without being lost or it might be expected from the robot to generate a precise map of a museum. In this thesis, we do not have a specific high level task for the robot. Instead, we focus on the Visual-SLAM (V-SLAM) problem on a multi-robot platform. Two basic SLAM solution methods are applied to a real robot platform and the experimental study is done on both simulation and real world. We provided visual landmarks for the robots and inspected the accuracy and the applicability of the methods on distributed multi-robot platforms.

The main contributions of this thesis can be listed as follows:

- Calibrating the parameters of the SLAM methods with supervised data. Tuning the parameters of the applied method is not trivial in some cases, and hand tuned (manually tuned) parameters may not be optimal. In this contribution, we searched the parameter space for more optimal solutions and tested with experiments.
- Using two robots and applying a map-merging method to the Fast-SLAM [1] algorithm with propagating the uncertainty in the map estimates. The methods are implemented for real robots and tested on both the simulation and in real world.

The rest of the thesis is organized as follows: The definitions of SLAM problem and solution methods in the literature are given in Chapter 2. The specifications and limitations of our problem definition and the underlying hardware and software platform are detailed in Chapter 3. In Chapter 4, our methods and implementation details for the complete system are explained. The experiment setups, conditions and the testing results are given in Chapter 5. In Chapter 6, the results are discussed and

some future works are pointed.

# 2. BACKGROUND

The real-world localization and mapping problem addresses a key problem in robot navigation. It is complicated and hard. Nothing is perfect in real world. Sensors are noisy and have limited capabilities. Actuators may result in unexpected results. These conditions pushed robotic researchers to find probabilistic solutions [2]. In this chapter, we will cover the terminology, types and most widely used methods for the localization and mapping problem. Most of the probabilistic solutions are subsets of state estimation techniques. To understand the methods better, we will start with the definition of the *state estimation problem* and its specific implementations. Then, the problem definition will be given and the most successful methods in the literature will be elaborated on.

## 2.1. State Estimation Problem

State estimation problem, also known as recursive Bayesian estimation, is a general problem which addresses recursively estimating an observable random variable given control inputs and perceptions. The system is often modelled as a hidden Markov model where the random variable is the state, observations are the perceptions of the agent and the belief state is a probability distribution on the state space. At time $t$, we will denote the state as $x_t$, the perception as $z_t$, the control input as $u_t$, and the belief state as $bel(x_t)$. The evolution of the system and the filter is shown in Figure 2.1. The recursive filter means that, a belief at time $t$ is calculated from the belief at time $t-1$.

The actual difficulty in estimating a random variable is representing the belief state. Since the space is continuous, representing the actual belief state is impossible in practice. There are different state estimation algorithms which overcome this difficulty for different sets of assumptions.

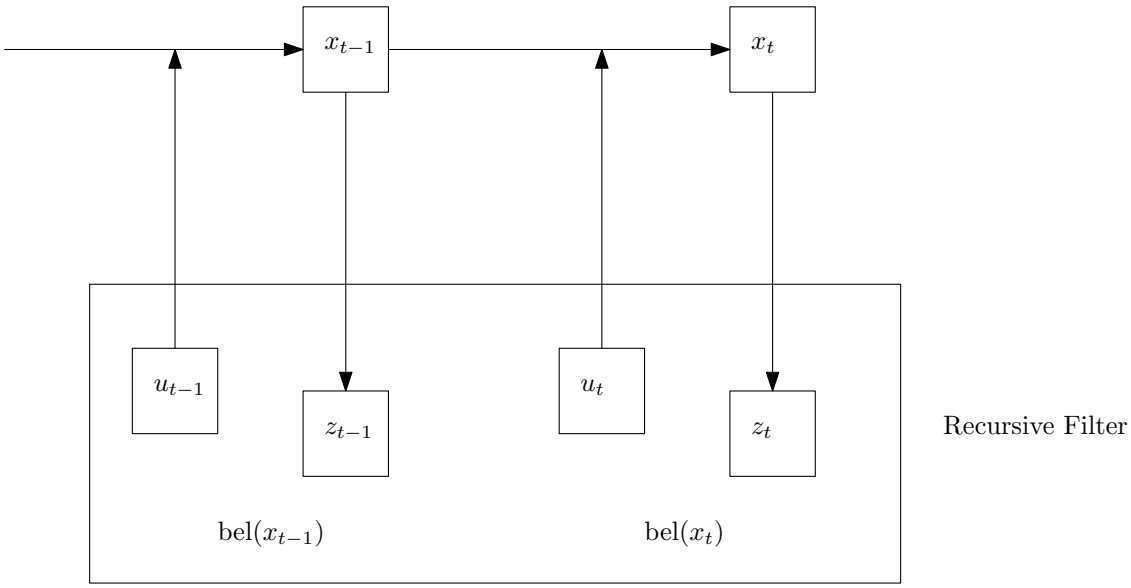



Figure 2.1. Recursive State Estimation.

### 2.1.1. Bayes Filter

The most general algorithm for calculating the belief state is the Bayesian Filter. Each recursive state is completed in two steps. First the belief $bel(x_t)$ is **predicted** using the control $u_t$ and the previous belief $bel(x_{t-1})$. In the second step, the predicted belief is **corrected** with the perception $z_t$ [2]. The pseudo-algorithm for the Bayes Filter is given in Figure 2.2.

**Algorithm** $BayesFilter(bel(x_{t-1}), control, perception)$

1: **for all** *possibleStates* **do**
2: $predictStateWithControl$
3: $updatePredictionWithPerception$
4: **end for**

Figure 2.2. The pseudo-algorithm for Bayes Filter.

The first step in the algorithm is called the prediction step, where the belief state is predicted with the control input and the uncertainty increases. Subsequently, the second step is the update step, where the predicted belief state is corrected with the observations and the uncertainty decreases.

One important feature of the Bayes Filter is that at time $t$, the belief state

only depends on the previous belief state, control input and observations. The formal algorithm is given in Figure 2.3.

**Algorithm** $BayesFilter(bel(x_{t-1}), u_t, z_t)$

1: **for all** $x_t$ **do**

2: $\quad \hat{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1}$

3: $\quad bel(x_t) = \eta p(z_t|x_t) \hat{bel}(x_t)$

4: **end for**

Figure 2.3. The Bayes Filter algorithm with equations.

The $\hat{bel}(x_t)$ is the predicted, but uncorrected belief state. The corrected state estimation is represented as $bel(x_t)$. In the rest of the Thesis, "hat" symbol indicates the predicted value of a variable.

The distribution $p(x_t|u_t, x_{t-1})$ specifies the control model where $u_t$ represents the control input and $p(z_t|x_t)$ specifies the observation model where $z_t$ represents the observation. To apply any kind of Bayes filter to a problem, these distributions should be specified.

The Bayes Filter is defined in continuous state spaces and most of the implementations are special cases of the Bayes Filter with different assumptions.

## 2.2. Parametric Filters

One family of Bayes Filter implementations is the parametric filters. These filters assume the probability distribution of the belief state comes from a distribution with some parameters and attempt to estimate the parameters of the distribution. This probability distribution is generally assumed to be Gaussian.

### 2.2.1. Kalman Filter

The Kalman filters are special case of representing the belief space, where the belief space is a Gaussian distribution over the state space with a mean vector $\mu_t$ as

the best estimate and a covariance matrix $\Sigma_t$ as the uncertainty in the belief [3].

$$bel(x_t) = \{\mu_t, \Sigma_t\} \tag{2.1}$$

The two dimensional state case is given in Figure 2.4.


Figure 2.4. Gaussian state representation for a two dimensional state space

Three assumptions must be made to ensure that $bel(x_t)$ is a Gaussian.

- The initial belief is normally distributed.

$$bel(x_0) = N(x_0; \mu_0, \Sigma_0)$$

- The control model is a linear function of state, control and additive noise.

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{2.2}$$

$$p(x_t|u_t, x_{t-1}) = N(x_t; A_t x_{t-1} + B_t u_t, R_t) \tag{2.3}$$

where $A_t$ and $B_t$ are the matrices of the prediction model equation at time $t$. $R_t$ is a covariance matrix with $\epsilon_t$ in its diagonal and represents the uncertainty in the control input.

- The observation model is a linear function of the state.

$$z_t = C_t x_t + \delta_t \tag{2.4}$$

$$p(z_t|x_t) = N(z_t; C_t x_t, Q_t) \tag{2.5}$$

where $C_t$ is the matrix of the observation model equation at time $t$. The $Q_t$ is a covariance matrix with $\delta_t$ in its diagonal and represents uncertainty in the observation at time $t$.

The Kalman Filter assumes that all the uncertainty distributions in the belief state, perception model and the state evolution model (control model) are Gaussian. So initially, the filter must start with a relatively good state estimate. The recursive algorithm of Kalman Filter is given in Figure 2.5.

**Algorithm** $KalmanFilter(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1: $\hat{\mu}_t = A_t \mu_{t-1} + B_t u_t$
2: $\hat{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
3: $K_t = \hat{\Sigma}_t C_t^T (C_t \hat{\Sigma}_t C_t^T + Q_t)^{-1}$
4: $\mu_t = \hat{\mu}_t + K_t(z_t - C_t \hat{\mu}_t)$
5: $\Sigma_t = (I - K_t C_t)\hat{\Sigma}_t$

Figure 2.5. The Kalman Filter algorithm.

The first two lines are the prediction step and lines 3-5 are the update step. $K_t$ is called the *Kalman Gain* and specifies how much should the state be updated with the observation. Its value depends on the uncertainty in the state and the observation.

The uncertainties of the perception model and the state transition model ($R_t$ and $Q_t$) are the key parameters for a good and accurate Kalman Filter. So in some works these uncertainties are estimated in a calibration step. One such implementation is given in [4] and is called Adaptive Kalman Filter. The idea is calibrating the noise parameters separately and running the filter with the estimated parameters.

In some applications modelling the actual system is almost impossible. In [5] the imperfect modelling of parameters are scaled into perfect model parameters with unknown extra variables. These introduced extra variables are included into the state space and estimated along with the model parameters. This is another solution to the sensitivity of Kalman filter to the noise parameters and is called Augmented Kalman Filter.

### 2.2.2. Extended Kalman Filter

The Kalman Filter assumes that both the control and the observation models are linear. Non-linear systems can be handled by linearization. In Extended Kalman Filters (EKF), the prediction model equation 2.2 is replaced by

$$x_t = g(u_t, x_{t-1}, \epsilon_t) \tag{2.6}$$

where $g$ is a non-linear function and the observation model equation 2.4 is replaced by

$$z_t = h(x_t, \delta_t) \tag{2.7}$$

where $h$ is a non-linear function.

The non-linear functions are linearized by Taylor Series [3]. At time $t$, the approximation of prediction model is

$$g(u_t, x_{t-1}, \epsilon_t) = g(u_t, \mu_{t-1}, \epsilon_t) + G_t(x_{t-1} - \mu_{t-1}) \tag{2.8}$$

where $G_t = \frac{\partial g(u_t, \mu_{t-1}, \epsilon_t)}{\partial x_{t-1}}$ and is the Jacobian of $g(u_t, x_{t-1}, \epsilon_t)$. Similarly at time $t$, the approximation of observation model is

$$h(x_t, \delta_t) = h(\mu_t, \delta_t) + H_t(x_t - \mu_t) \tag{2.9}$$

where $H_t = \frac{\partial h(\mu_t, \delta_t)}{\partial x_t}$ is the Jacobian of $h(x_t, \delta_t)$.

The noise terms are also inputs of the non-linear models, so the noise covariances should be linearized. The process noise covariance $R_t$ is transformed by $W_t R_t W_t^T$ where $W_t = \frac{\partial g(u_t, \mu_{t-1}, \epsilon_t)}{\partial epsilon_t}$. Similarly the measurement noise covariance $Q_t$ is transformed with $V_t Q_t V_t^T$ where $V_t = \frac{\partial h(\mu_t, \delta_t)}{\partial delta_t}$.

With the linearity approximations, the Extended Kalman Filter algorithm is given in Figure 2.6.

**Algorithm** $ExtendedKalmanFilter(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

1: $\hat{\mu}_t = g(u_t, \mu_{t-1})$
2: $\hat{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + W_t R_t W_t^T$
3: $K_t = \hat{\Sigma}_t H_t^T (H_t \hat{\Sigma}_t H_t^T + V_t Q_t V_t^T)^{-1}$
4: $\mu_t = \hat{\mu}_t + K_t(z_t - h(\hat{\mu}_t))$
5: $\Sigma_t = (I - K_t H_t)\hat{\Sigma}_t$

Figure 2.6. The Extended Kalman Filter algorithm.

The linearity approximations may fail in some systems, however EKF is still widely used. An application of an EKF algorithm can be seen in Cerberus team [6] and Nubots team [7] in Robocup Four-Legged league [8] where the problem is estimating the position and speed of the ball with respect to the robot.

### 2.2.3. Unscented Kalman Filter

The EKF algorithm is a good way to apply Kalman Filter to non-linear systems, however it suffers from the derivation in the linearity approximation. One alternative approach is to represent the Gaussian belief distribution as deterministically calculated set of points. The set of points, $X$, is called *sigma points* and represents both mean and covariance of the Gaussian distribution.

For the $n$ dimensional Gaussian distribution, there are $2n + 1$ sigma points and

associated weights.

$$
\begin{aligned}
X^0 &= \mu && (2.10)\\
w_m^0 &= \frac{\lambda}{n+\lambda} && (2.11)\\
w_c^0 &= \frac{\lambda}{n+\lambda} + (1-\alpha^2+\beta) && (2.12)\\
X^i &= \mu \pm (\sqrt{(n+\lambda)\Sigma})_i \quad \text{for} \quad i = 1,2,...,2n && (2.13)\\
w_m^i &= w_c^i = \frac{i}{2(n+\lambda)} \quad \text{for} \quad i = 1,2,...,2n && (2.14)\\
\lambda &= \alpha^2(L+\kappa) - L && (2.15)
\end{aligned}
$$

In this representation, $\alpha$,$L$ and $\kappa$ are the parameters. The prediction is performed with noise free calculation of new sigma points with the non-linear prediction function and deriving the mean and covariance with the new sigma set and weights. Note that just like the EKF prediction and update equations in Figure 2.6, the process noise covariance $R_t$ is the parameter in the prediction step and $Q_t$ is the observation noise covariance.

$$
\begin{aligned}
\Psi^i &= g(X^i) && (2.16)\\
\hat{\mu} &= \sum_{i=0}^{2n} w_m^i \Psi^i && (2.17)\\
\hat{\Sigma} &= \sum_{i=0}^{2n} w_c^i (\Psi^i - \hat{\mu})^T + R_t && (2.18)
\end{aligned}
$$

In Figure 2.7, the prediction of the new sigma point set ($\Psi$) is given for a two dimensional state space.

In the update step, the expected observation is represented similarly with sigma points calculated from the belief state and the mean and covariance for observation is

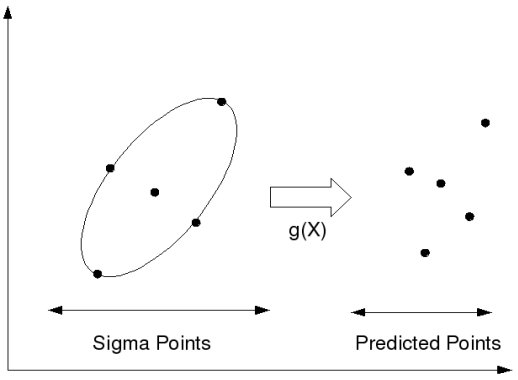



Figure 2.7. Prediction of the sigma points in a two dimensional state space.

derived with the calculated sigma points with the equations:

$$\overline{Z}_t = h(\overline{X}_t) \tag{2.19}$$

$$\hat{z}_t = \sum_{i=0}^{2n} w_m^i \overline{Z}_t^i \tag{2.20}$$

$$S_t = \sum_{i=0}^{2n} w_c^i (\overline{Z}_t^i - \hat{z}_t)(\overline{Z}_t^i - \hat{z}_t)^T + Q_t \tag{2.21}$$

where $\overline{Z}_t$ is the prediction of $Z_t$ and $S_t$ represents the covariance (uncertainty) of the observation expectation and will be used to compute the *Kalman Gain*.

Now we have the sigma points and the derived mean and covariance values for both the state and the observations. To calculate the updated state the following equations are used:

$$\hat{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^i (\overline{X}_t^i - \hat{\mu})(\overline{Z}_t^i - \hat{z}_t)^T \tag{2.22}$$

$$K_t = \hat{\Sigma}_t^{x,z} S_t^{-1} \tag{2.23}$$

$$\mu_t = \hat{\mu}_t + K_t(z_t - \hat{z}_t) \tag{2.24}$$

$$\Sigma_t = \hat{\Sigma}_t - K_t S_t K_t^T \tag{2.25}$$

where $\hat{\Sigma}_t^{x,z}$ is the cross-correlation between the observation and the state. This value is used together with the observation expectation uncertainty to calculate the *Kalman Gain* $K_t$.

This algorithm is called Unscented Kalman Filter (UKF) [9]. The UKF is a derivative-free filter which means the derivatives of the functions $g$ and $h$ are not needed.

## 2.3. Non-Parametric Filters

Non-parametric filters approximate the belief state distribution $bel(x_t)$ without any probability model. Such filters either decompose the state space into discrete values, or represent the distribution with sampling. These kinds of approximations allow solutions of systems with more complex and multimodal distributions.

### 2.3.1. Particle Filter

Particle filters have recently become the most post popular family of state estimation techniques. This approach uses a sample-based state representation and can be applied to a wide range of state estimation problems including uni-modal and multi-modal distributions [2],[10].

In the particle filter method, a state $X_t$ at time $t$ is represented with a set of particles where each of them $x_t^i$ represents a candidate of the actual state.

$$X_t = \{x_t^1, x_t^2, ..., X_t^N\} \tag{2.26}$$

In Figure 2.8, an example belief state for a robot position in the soccer field is given.

At each iteration of the algorithm, first the particles are translated with the control input. After that, an importance weight ($w$) is assigned for each particle with the perception. The last step is the heart of the particle filter, where the particles are sampled according to their importance weights. This sampling step forms the proposal distribution, represented by the predicted belief state, to converge to the target distribution, the corrected belief state. The overall algorithm is given in Figure 2.9.

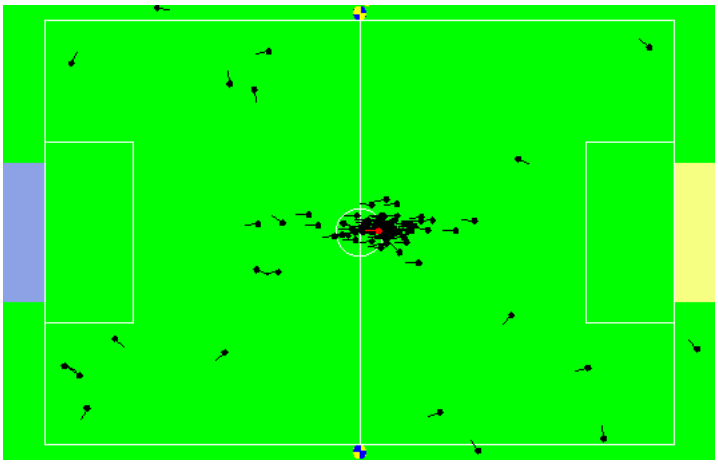


Figure 2.8. Example belief state of a robot position in the soccer field. Each dot represents a possible location of the robot.

**Algorithm** $ParticleFilter(X_{t-1}, u_t, z_t)$

1: $X_t = \hat{X}_t = \{\}$
2: **for** $i = 1$ to $N$ **do**
3: sample $x_t^i \sim p(x_t|u_t, x_{t-1}^i)$
4: $w_t^i = p(z_t|x_t^i)$
5: $\hat{X}_t = \hat{X}_t + < x_t^i, w_t^i >$
6: **end for**
7: **for** $i = 1$ to $N$ **do**
8: draw $i$ with probability $w_t^i$
9: $X_t = X_t + < x_t^i >$
10: **end for**

Figure 2.9. The Particle Filter algorithm.

The representation of belief state and very few assumptions on probability distributions make particle filters distribution free state estimation algorithm. The introduction of weight assignment also relaxes the noise distribution assumptions on the measurement model. One can treat each particle as a state and assign weights accordingly.

### 2.3.2. Rao-Blackwellisation of Particle Filter

Rao-Blackwellised particle filter is a derivation of the particle filter which is based on the Rao-Blackwell theorem [11]. In this method, the state features are divided into two subsets where, one can infer the probabilities of values in one subset with the probabilities of the values in the other set. Suppose our complete state at time $t$ is $x_t$, and the observation at time $t$ is $z_t$ and we want to find a posterior $p(x_t|z_t)$. If our model permits a factorization such that $x_t$ is split into two subsets $k_t$ and $l_t$ which satisfies:

$$p(x_t|x_{t-1}) = p(l_t|k_{t-1}, l_{t-1})p(k_t|k_{t-1}) \tag{2.27}$$

such that $p(l_t|z_t, k_t)$ is analytically tractable, then we can estimate $k_t$ with a filter and estimate $l_t$ analytically.

Using this idea, Rao-Blackwellised particle filter estimates one subset of features with a particle set and tracks the rest of the features with another estimator, usually with Unscented Kalman filter.

One can easily set up a system where he separates the state feature set and tracks each variable with independent estimators. However, the power of this youngest state estimation method comes from not only dividing state features into two subsets but also inferring one subset from another. When the number of state features are large, particle filters suffer from a large increase in the number of particles. When the particles are generated for only one portion of features, however the necessary number of particles decreases.

Particle filters have found some suitable application topics. In mobile robot localization, the simultaneous localization and mapping problem (SLAM) is one of them which we will examine in Section 2.5 . Another application topic in computer vision is object tracking. In the problem of object tracking, some shape features and location information of an object is estimated in time. In [12] this problem is modelled for the Rao-Blackwellised particle filter where the state features are shape features and

location features. The shape features of the object is inferred and tracked with location information. And the location information is estimated with particles.

## 2.4. Mobile Robot Localization Problem

The localization problem is the problem of estimating the position of a robot given the map of the environment, observations and odometry readings. Before proceeding with the types of localization problem and solution methods, the elements should be explained.

### 2.4.1. Robot Pose

The state to be estimated in the localization problem is the position of the robot in the 2-dimensional environment at time $t$.

$$p_t = \{p_{x,t}, p_{y,t}, p_{\theta,t}\}$$

where $p_x$ and $p_y$ are cartesian coordinates of the robot and $p_\theta$ is the orientation of the robot.

In some domains, 3-dimensional position may be estimated where state vector has six elements (three for position, three for orientation). However, it is beyond the scope of this work.

### 2.4.2. Maps

In the localization problem, the robot is given a map of the environment in the beginning. A map contains the exact locations of all relevant entities in a finite environment. There are two types of map of the environment.

- **Location Based Maps** In location based maps, each map entry in the map holds a property for a location in the map.

- **Feature Based Maps** In feature based maps, each map entry in the map holds the location and properties for an object in the map. The feature based maps generate two kinds of problem;
  - The **Known Correspondence Problem**, where a robot can match the perceptions with the entities in the map,
  - The **Unknown Correspondence Problem**, where a robot cannot match the perceptions with the entities in the map,

For the feature based maps, we will denote a map as $M$. The map represents all the known landmarks.

$$M = \{m_1, m_2, ..., m_L\} \tag{2.28}$$

$$\text{where} \quad m_i = \{p_x^i, p_y^i\} \tag{2.29}$$

Each entity $m_i$ in the map represents the cartesian coordinate of the corresponding landmark in the global coordinate frame.

### 2.4.3. Observations

The robot has several sensors and can observe the entities from the map it is given. The observation format may vary with sensor type or observation algorithm type, generally the representation of a set of observations is:

$$Z_t = \{z_t^1, z_t^2, ..., z_t^K\} \tag{2.30}$$

where $z_t^i$ can be the bearing $\theta_i$ and the distance $l_i$, or both distance and bearing $\{l_i, \theta_i\}$ of the landmark to the robot.

The observation may come from different sensor types and the observation model $p(z_t^i|x_t)$ of each type is given to the robot. For instance, if the observation comes from a laser scan as distance readings, and the map is given as a location based map

(occupancy grid map), then, a likelihood field is formed and observation model is calculated accordingly as in Figure 2.10.

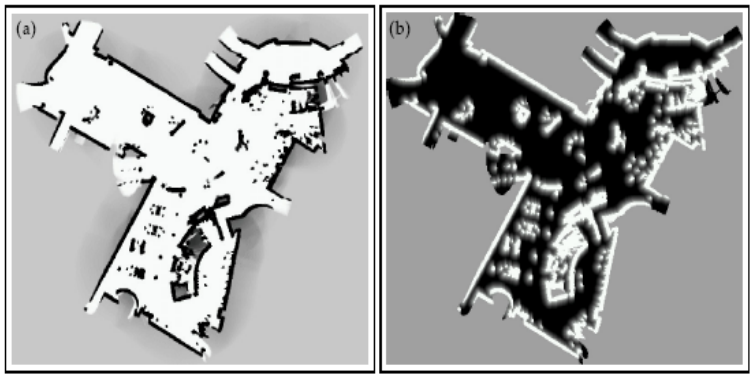



Figure 2.10. Observation model for a laser sensor. On the left, occupancy grid map. On the right, pre-calculated likelihood field. White regions represents observations with high probability [2].

If the observation comes from a feature based map, the observation model is generally selected Gaussian as in Eq. (2.5). If the observation model is stated with the function

$$\hat{z}_t = h(x_t) \tag{2.31}$$

the element-wise definition for distance and bearing type observation is

$$\begin{bmatrix} \hat{l}^i \\ \hat{\theta}^i \end{bmatrix} = \begin{bmatrix} \sqrt{(p_y - p^i_{y,\text{map}})^2 + (p_x - p^i_{x,\text{map}})^2} \\ \text{atan}(\frac{p_y - p^i_{y,\text{map}}}{p_x - p^i_{x,\text{map}}}) - p_\theta \end{bmatrix} \tag{2.32}$$

where $p^i_{x,\text{map}}$ and $p^i_{y,\text{map}}$ are the cartesian coordinates of the $i$th landmark given in the map. Since the uncertainty model is Gaussian

$$p(z^i_t|x_t) = N(z^i_t; h(x_t), R_t) \tag{2.33}$$

where $R_t$ is the uncertainty covariance. The likelihood of an observation given the state

can be calculated as

$$p(Z_t|x_t) = \prod_{i=1}^{K} \frac{1}{(2\pi)|R_t|^{1/2}} \exp(-\frac{1}{2}(z_t^i - h(x_t))^T R_t^{-1}(z_t^i - h(x_t))) \tag{2.34}$$

An example observation model in feature based map is given in Figure 2.11. Triangle is the robot, filled circle is the exact location (entity in the map) of the landmark, empty circle is the observation and the ellipse represents Gaussian uncertainty of the observation model. The $p(z_t^i|x_t)$ can be calculated with the probability distribution function of the Gaussian distribution.

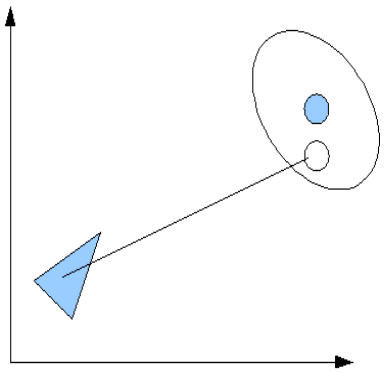


Figure 2.11. Observation model in a landmark based map.

### 2.4.4. Odometry

The odometry readings in a robot are control inputs for the localization problem. At each time step, it is assumed that robot can tell how far it has travelled with directions. The odometry readings are often represented as

$$u_t = \{\Delta x, \Delta y, \Delta\theta\}$$

where $\Delta x$ is the forward displacement, $\Delta y$ is sideways displacement and $\Delta\theta$ is the change in orientation. The uncertainty is modelled as Gaussian distribution. In Figure 2.12, a sample displacement reading is given.

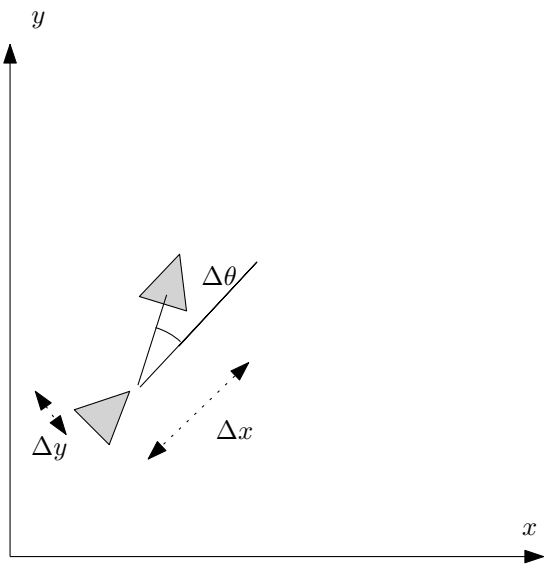



Figure 2.12. Odometry readings of a robot.

The control model function of a robot is

$$\hat{x}_{t+1} = g(u_t, x_t) \tag{2.35}$$

and the element-wise definition is

$$\begin{bmatrix} \hat{p}_{x,t+1} \\ \hat{p}_{y,t+1} \\ \hat{p}_{\theta,t+1} \end{bmatrix} = \begin{bmatrix} p_{x,t} + d_{x,t}\cos(p_{\theta,t}) - d_{y,t}\sin(p_{\theta,t}) \\ p_{y,t} + d_{x,t}\sin(p_{\theta,t}) + d_{y,t}\cos(p_{\theta,t}) \\ \Delta\theta_t + p_{\theta,t} \end{bmatrix} \tag{2.36}$$

Since the uncertainty model is Gaussian

$$p(x_{t+1}|u_t, x_t) = N(x_{t+1}; g(u_t, x_t), Q_t) \tag{2.37}$$

where $Q_t$ is the uncertainty covariance. The likelihood of a state given the previous state and the odometry can be calculated as

$$p(x_{t+1}|u_t, x_t) = \frac{1}{(2\pi)|Q_t|^{1/2}}\exp(-\frac{1}{2}(x_{t+1} - g(u_t, x_t))^T Q_t^{-1}(x_{t+1} - g(u_t, x_t))) \tag{2.38}$$

### 2.4.5. Localization Problem Types

The localization problem types can be classified by their platform or problem domain:

- **Local versus Global Localization** In *local localization*, also known as position tracking, we assume the initial position of the robot is known. The rest of the estimating process is basically updating the pose with motion commands and perceptions. Whereas, in *global localization*, the initial position of the robot is not known, so the robot can not make an initial guess, but tries to estimate the location in the map. To go further if robot can be kidnapped any time, it is called *kidnapped robot problem*.
- **Passive versus Active Localization** If the robot can control its motion commands, this kind of problem is called *active localization* and if robot cannot control itself, the problem is called *passive localization*.
- **Single-Agent versus Multi-Agent Localization** If there are multiple communicating and cooperating robots, than the problem is *Multi-Agent Localization*, else it is *Single-Agent Localization*.

The current localization algorithms are all based on the state estimation techniques. But since there are various localization problem types and solution techniques, we can classify localization solution techniques with their approaches.

- **Kalman Filter Based** These approaches represent the belief state distribution as a Gaussian and update belief states with Kalman Filter variants. They have lower performance requirements but cannot solve kidnapping situations or global localization problem.
- **Monte Carlo Based** These algorithms use the belief state distribution with samples, so they can represent arbitrary distributions. This allows them to solve kidnapping situations and global localization problem.
- **Markov Based** These algorithms represents the state space with small approximation errors. However, resource constraints are very tight and cannot scale

larger problems.

### 2.4.6. EKF-Localization

EKF-Localization is the earliest practical localization algorithm which is based on Extended Kalman Filters. With the algorithm in Figure 2.6 in section 2.2.2, applying EKF to this problem is trivial. The observation model is as defined in equation 2.32 and the odometry model is as defined in equation 2.32. This algorithm runs in feature based maps. For unknown correspondence problem, a maximum likelihood estimation is used for perceptions. The maximum likelihood approach will be detailed in section 2.5.3.

The EKF needs a good estimate on initial belief to start running. So EKF-Localization can only solve local localization problem. The lack of multi-modal belief representation decreases the performance of this algorithm in kidnapping situations.

### 2.4.7. Monte-Carlo Localization

Monte Carlo localization (MCL) is the most popular localization algorithm. It uses the particle filters to estimate the robot pose. Each particle represents a pose candidate of the robot and the overall particles represents a distribution free belief state [13].

The algorithm of MCL is given in section 2.3.1 in Figure 2.9. In the first sampling step (prediction step), the particles are sampled as

$$\hat{x}_{t+1}^{i} = g(u_t, x_t^i) + \epsilon \tag{2.39}$$

where $\epsilon$ is a random variable sampled normally with mean 0 and covariance $Q$. The importance weights are calculated with the equation 2.34.

MCL can solve both local and global localization problems and also kidnapped

robot problem. The belief state is not limited for a distribution like in EKF-Localization. This makes MCL more reliable when there are obstacles or repetitive structures in the environment [2].

### 2.4.8. Grid Localization

Grid localization, also called Markov Localization, is another approach to solve the localization problem. The 2D environment is partitioned into finite number of grids and belief state is represented as a set of probability value for each grid. This algorithm accepts location based maps [2].

The partitioning of the grid is the main parameter to tune. If the regions are large and update intervals are small, the algorithm may fail in updating grids correctly. Or alternatively if regions are small, the size of belief state becomes very large and the problem may be intractable.

### 2.4.9. Related Work

In [14], a hybrid localization method called Reverse MCL, which combines MCL and Markov Localization (ML) is introduced. The ML and MCL methods can solve the global localization problem and the kidnapped robot problem. However they may fail in converging fast. The ML algorithm has a shortcoming in grid size tuning when small update time steps exist. And the MCL may diverge when noise is too high.

In Reverse-MCL, ML algorithm is run in the first place. Normally, the robot pose has three dimensions, but in this method, the grids are formed for only two dimensional position of the robot. When the ML has converged to some level (the convergence criteria are parametric), samples for the MCL algorithms are generated for the grids whose likelihoods are high. Then, MCL starts to run and converges. If MCL diverges, the ML step is taken again.

The reason why this method is called “Reverse” is that in normal MCL, first the

samples are generated, updated and then robot pose is extracted. However in Reverse-MCL, first the robot pose is estimated and samples are generated. By this way, the ML step of the algorithm can quickly recover from the kidnapped robot situation.

There is also an extension to the Reverse-MCL algorithm by representing the probabilities of each grid by fuzzy sets instead of constant values [15]. The probabilities are calculated from a fuzzy membership function where the domain set is the difference between observed landmark distance and the expected landmark distance. With this method, the bias of the algorithm is reduced and the stability is increased. And also the robot can recover from kidnapped robot situation faster.

In [16], a multi-robot extension to the MCL algorithm is introduced. In the problem setup, the robots have laser range finders and a color camera. The robots have colored markers on their body to recognize each other. The map representation is a grid based map and the inputs for the localization algorithm are landmark observations, odometry readings and other robot observations.

In the algorithm, in addition to the normal MCL steps, one more step is added for the case of observing other robots. The algorithm works as follows. When a robot is not detected by another robot, it runs its own MCL module. If a robot detects another robot, it constructs a belief state for the location of the detected robot by generating a sample set and sends this state to the detected robot. For the detected robot, merging two belief states represented by particles is not a straight forward job. To do this, a density tree is generated with the samples of two belief set. The nodes on the tree are decision boundaries and the leaves are markers for a region on the map. The generated density tree is simply a piecewive constant representation of the belief state. It includes small but large number of grids where the input samples are dense, and big but small number of grids where the input samples are sparse. When the detected robot generates the tree, it resamples the sample set for its belief state by using the tree. By this way, merging belief state problem is solved.

## 2.5. Simultaneous Localization And Mapping

Simultaneous localization and mapping (SLAM) is the problem of estimating the robot pose along with the map. Unlike the localization problem, the robot is not given a map and has to build the map while localizing itself on it [2]. Not having a map of the environment makes localization hader and not knowing the initial pose makes the map making harder. As a result, the problem is harder than localization because the error in map estimation leads errors in robot pose and error in robot pose leads errors in map estimate, so the error grows in time and is unbounded. In this section the problem and related work is inspected from different perspectives.

### 2.5.1. Belief State Representation and Updating

The current probabilistic solutions to the SLAM problem use parametric or non-parametric filter based methods and its variants. The state to be estimated is a combined vector of robot pose and all the known entities of the map. As the robot explores, the state vector grows and it theoretically has no limit. Tracking the belief state of such a state is hard and there are different approximations in different approaches.

The most widely used approach is to represent the belief state as a single Gaussian distribution [17]. The belief state is a mean vector and a covariance matrix which also represents correlations between map entities and the robot pose. The prediction and update steps are performed with the *EKF* algorithm. This is the earliest approach to the SLAM problem and is called EKF-SLAM. This algorithm is detailed in Section 2.5.5.

As detailed in Section 2.3.2, the belief state can be factorized into subgroups and more accurate estimators can be applied. In [1], this idea is applied to the SLAM problem. Given a robot pose, the landmark position estimations becomes independent. The state space is factored such that robot pose is estimated with a particle set and the known landmarks are estimated as independent Gaussian distributions for each particle separately. While the particles are predicted with the odometry readings, the

landmarks are updated with the observations. This algorithm is called *Fast-SLAM* and detailed in Section 2.5.6.

In Fast-SLAM, the prediction is made by sampling the particles with the odometry and odometry uncertainty. In cases where odometric noise is high, the particles may fall into positions which is unlikely given the current observations. These particles are eliminated in the resampling step. In [18], a more accurate sampling method is introduced. The observations are also used in the sampling of particles. As a result, the unlikely sampled particles are eliminated in the sampling step instead of the resampling step.

The odometry information is generally provided by the robot hardware. However with visual sensors, it is possible to obtain a physical displacement values from the observations. These methods are are called *vision-based ego-motion* and explained in Section 2.5.7. In [19, 20], the Fast-SLAM algorithm is used, and in the prediction step, some particles are predicted using the odometric readings, while the rest of them are predicted with vision-based odometry. It is reported that this method is more robust than Fast-SLAM with high odometry noise.

In [21], the SLAM problem is applied on a platform consisting of a camera moving in the 3D space. There is no odometry information but it is assumed that the camera motion is smooth and the speed does not change frequently. The velocity parameters of the camera are added to the state vector. The prediction and update steps are performed using EKF-SLAM algorithm for the augmented belief state.

### 2.5.2. Map Representation

The complexity and the performance of the SLAM algorithm highly depends on the map representation. Most of the approaches represent the map as a set of landmarks [17, 19, 20]. However there also exists different approaches.

In the Fast-SLAM based approaches, the map is represented with the maps of

each particle, namely the estimated map is a set of maps, each estimated by a particle. In this representation each map may not have the same number of landmark and the positions of the same landmark may not be the same in each map. In [1], the landmark states of each particle is stored as the leaves of a tree and the complexity for accessing the state of a landmark is decreased from linear to logaritmic.

While the robot explores, the map size grows and updating the state requires more time. Updating local regions of the map is introduced in [22]. A hierarchical map is represented as a graph where each node is a local occupancy map and the links are the connection between local maps. The robot builds the local occupancy grid map until the uncertainty on the pose is bigger than a predefined value or the local map size reaches a limit. Than a new local map is started and linked to the previous one.

In [23], just like the previous approach, there are local maps which are called *patches*. Each patch has a bounding polygon, but instead of linking the patches to each other, each center has a point with respect to the global coordinate frame. In this representation, overlapping patches are allowed. This representation is called the *Manifold*. In [24], the algorithm is applied in the Virtual Robots League in Robocup [8].

### 2.5.3. Data Association Problem

Generally, in the SLAM problem, the robot does not know the environment or the landmarks within, and identifying a landmark with only observations is almost impossible. In such cases, the robot has to match the observed landmarks with the previously seen landmarks or state that the observation is a newly discovered landmark. This problem is called *Data Association Problem*. The main characteristic of this problem is that past knowledge on observations and the belief state is used for reasoning. Since the belief state and observations have uncertainties, this problem is not trivial.

The most straightforward approach is the maximum likelihood or also known

as nearest neighbor approach [25]. The idea is that the location of the observation is inferred with the best estimate of the current state and compared with previous known landmarks. If the closest landmark is not closer than a specified parameter, the observation is stated as a new landmark. In Figure 2.13 the algorithm is given. Input are a single observation $z_t$, best pose estimation $p_t$ and the known landmarks $M$. Returns the index of the associated landmark.

**Algorithm** MLDataAssociation($p_t, z_t, M$)

1: $pos \leftarrow calculateGlobalCoordinate(p_t, z_t)$
2: $minDist \leftarrow MAXINT$
3: $minDistIndex \leftarrow -1$
4: **for** $i = 1$ to $size(M)$ **do**
5: $\quad dist \leftarrow distance(pos, m_i)$
6: $\quad$ **if** $dist < minDist$ **then**
7: $\quad\quad minDist \leftarrow dist$
8: $\quad\quad minDistIndex \leftarrow i$
9: $\quad$ **end if**
10: **end for**
11: **if** $minDistIndex = -1$ **then**
12: $\quad addToM(pos)$
13: $\quad$ **return** $size(M)$
14: **else**
15: $\quad$ **return** $minDistIndex$
16: **end if**

Figure 2.13. The Maximum Likelihood algorithm for data association.

The likelihood metric in algorithm 2.13 is only the distance. In applications where $M$ includes Gaussian uncertainties - both in EKF-SLAM and Fast-SLAM (Section 2.5.5,2.5.6) does - the likelihood function is used as in equation 2.34.

Sometimes the data association algorithm may fail due to noise and associate the observation with a false landmark. This problem is focused in [25]. An observation has a likelihood value for each landmark, and instead of selecting the most likely one, the landmark is selected randomly weighted with their likelihoods. A landmark also cannot be associated with more than one observation at a time step, so there is

mutual exclusion in decisions. Finally, last modification is utilizing the non-existing observations. There are existence probabilities of each landmark, and if an observation is expected for a landmark but not observed, the existence probability of it is reduced. This also enables landmark pruning from the map.

The indentity of the landmarks are not given to the system, but sometimes they can be extracted from the raw sensor data. In [26], the platform uses a camera to observe landmarks and the perception algorithm also extracts features unique to landmarks and these features are used in the perception to decide landmark associations (explained in Section 2.5.7).

### 2.5.4. Loop Closure or Revisiting Problem

The loop closure problem is a larger scaled version of the data association problem. As the robot explores the environment, after exploring different areas, it may revisit the same section of the map it already explored. For example, robot may have explored a portion of a corridor and it can enter the same area from another location. Since the odometric error grows incrementally, the robot may not decide that it has been in that location. Most of the discussed approaches rely on the data association algorithm detailed in Section 2.5.3.

A different approach is applied in [22]. The map representation is a graph where each node is a local grid map and links are the transformation parameters between them as detailed in Section 2.5.2. When the robot decides initializing a new local map, it first checks the observations with the existing local maps. If a match is discovered, instead of creating a new node, it uses the existing node as the local map.

In the *Manifold* map representation as detailed in Section 2.5.2, the map is formed by local *patches* with a bounding polygon and a center point. As the robot explores it creates new patches without concerning the existing patches, because the map representation allows overlapping patches. This method enables *delayed loop closure*, searching for overlaps after exploration of areas. If overlapping patches are found, they

are merged into single patch so that the map is pruned.

### 2.5.5. EKF-SLAM

In this section the mathematical details of the EKF-SLAM algorithm is covered. The EKF-SLAM is the application of the EKF algorithm to the SLAM problem. The EKF algorithm is given in Figure 2.6. The state to be estimated is:

$$x_t = \{p_{x,t}, p_{y,t}, p_{\theta,t}, p^1_{x,t}, p^1_{y,t}, p^2_{x,t}, p^2_{y,t}, ..., p^L_{x,t}, p^L_{y,t}\} \tag{2.40}$$

where

$$p_t = \{p_{x,t}, p_{y,t}, p_{\theta,t}\} \tag{2.41}$$

is the pose and

$$M_t = \{p^1_{x,t}, p^1_{y,t}, p^2_{x,t}, p^2_{y,t}, ..., p^L_{x,t}, p^L_{y,t}\} \tag{2.42}$$

is the map. The odometry reading in each step is:

$$u_t = \{\Delta_{x,t}, \Delta_{y,t}, \Delta_{\theta,t}\} \tag{2.43}$$

and the prediction model function is:

$$p_t = g(u_t, p_{t-1}, \epsilon_t) \tag{2.44}$$

$$\begin{bmatrix} p_{x,t} \\ p_{y,t} \\ p_{\theta,t} \end{bmatrix} = \begin{bmatrix} p_{x,t-1} + (\Delta_{x,t} + \epsilon^1_t)\cos(p_{\theta,t-1}) - (\Delta_{y,t} + \epsilon^2_t)\sin(p_{\theta,t-1}) \\ p_{y,t-1} + (\Delta_{x,t} + \epsilon^1_t)\sin(p_{\theta,t-1}) + (\Delta_{y,t} + \epsilon^2_t)\cos(p_{\theta,t-1}) \\ p_{\theta,t-1} + \Delta_{\theta,t} + \epsilon^3_t \end{bmatrix} \tag{2.45}$$

where $\epsilon_t$ is the normally distributed random noise with covariance $R_t$. The process Jacobian $G_t$ is:

$$G_t = \frac{\partial g(u_t, p_{t-1}, 0)}{\partial p_{t-1}} = \begin{bmatrix} 1 & 0 & -\Delta_{x,t}\sin(p_{\theta,t}) - \Delta_{y,t}\cos(p_{\theta,t}) \\ 0 & 1 & \Delta_{x,t}\cos(p_{\theta,t}) - \Delta_{y,t}\sin(p_{\theta,t}) \\ 0 & 0 & 1 \end{bmatrix} \tag{2.46}$$

The Jacobian of $g(u_t, p_{t-1}, \epsilon_t)$ with respect to $\epsilon_t$ is:

$$W_t = \frac{\partial g(u_t, p_{t-1}, \epsilon_t)}{\partial \epsilon_t} = \begin{bmatrix} \cos(p_{\theta,t-1}) & -\sin(p_{\theta,t-1}) & 0 \\ \sin(p_{\theta,t-1}) & \cos(p_{\theta,t-1}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.47}$$

The observation model function $h(x_t, \delta_t)$ is:

$$z_t^i = h(p_t, \delta_t) \tag{2.48}$$

$$\begin{bmatrix} l_t^i \\ \theta_t^i \end{bmatrix} = \begin{bmatrix} \sqrt{(p_{x,t} - p_{x,t}^i)^2 + (p_{y,t} - p_{y,t}^i)^2} \\ \operatorname{atan}(p_{y,t}^i - p_{y,t}, p_{x,t}^i - p_{x,t}) \end{bmatrix} \tag{2.49}$$

The measurement model Jacobian $H_t$:

$$l_t^i = \sqrt{(p_{x,t} - p_{x,t}^i)^2 + (p_{y,t} - p_{y,t}^i)^2} \tag{2.50}$$

$$a = \frac{p_{x,t} - p_{x,t}^i}{l_t^i} \tag{2.51}$$

$$b = \frac{p_{y,t} - p_{y,t}^i}{l_t^i} \tag{2.52}$$

$$H_t = \begin{bmatrix} a & b & 0 & \dots & 0 & -a & -b & 0 & \dots & 0 \\ -\frac{b}{l_t^i} & \frac{a}{l_t^i} & -1 & \dots & 0 & \frac{b}{l_t^i} & -\frac{a}{l_t^i} & 0 & \dots & 0 \end{bmatrix} \tag{2.53}$$

where entity $-a$ corresponds to the index $3 + 2i$. At each time step the update is performed for each observation separately. This is why the corresponding entries in the Jacobian $H_t$ are zero.

The error covariance includes all the correlation values between all landmarks and the robot pose. This is the most informative state representation among other SLAM solutions. It does not approximate any relations but only have the Gaussian uncertainty assumption. The prediction step is simple since only the robot pose is predicted. However, update state changes all the state vector and covariance matrix and has quadratic time complexity.

One alternative solution to the complexity issue is introduced in [27]. The uncertainty is represented with the inverse of error covariance, which is the idea in Extended Information Filters. In this representation, as the correlation between two entities increases, the corresponding value in the inverse of covariance matrix gets closer to one. And the corresponding values for uncorrelated entities are zero. With the assumption that most of the distant landmarks are uncorrelated, the uncertainty matrix becomes sparse and complexity in the update step is reduced to order of number of landmarks.

### 2.5.6. Fast SLAM

Fast-SLAM is basically the application of the Rao-Blackwellised particle filters to the SLAM problem [1],[28]. The belief state representation is a set of particles with each element including the position of the robot and Gaussian state for position of the objects.

$$\mathbf{X_t} = \{x_t^1, x_t^2, ..., x_t^N\} \tag{2.54}$$

$$x_t^n = \{p_{x,t}^n, p_{y,t}^n, p_{\theta,t}^n, p_{x,t}^{n,1}, p_{y,t}^{n,1}, \Sigma_t^{n,1}, p_{x,t}^{n,2}, p_{y,t}^{n,2}, \Sigma_t^{n,2} ..., p_{x,t}^{n,L}, p_{y,t}^{n,L}, \Sigma_t^{n,L}\} \tag{2.55}$$

At each step of the algorithm, the robot pose portion of the particles are predicted using odometry. The importance weights are then calculated with the map of each particle and observations. The maps of each particle are updated with observations by the EKF time update equations as in Section 2.2.2. For a particle $x_t^n$, the pose is $p_t^n$

and map entity $i$ is $\{p_{x,t}^{n,i}, \Sigma_t^{n,i}\}$. The measurement model function is:

$$z_t^i = h(x_t^n, \delta_t) \tag{2.56}$$

The measurement model Jacobian $H_t$ is:

$$\begin{aligned} l_t^{n,i} &= \sqrt{(p_{x,t}^n - p_{x,t}^{n,i})^2 + (p_{y,t}^n - p_{y,t}^{n,i})^2} && (2.57)\\ a &= \frac{p_{x,t}^n - p_{x,t}^{n,i}}{l_t^{n,i}} && (2.58)\\ b &= \frac{p_{y,t}^n - p_{y,t}^{n,i}}{l_t^{n,i}} && (2.59)\\ H_t &= \begin{bmatrix} a & b \\ -\frac{b}{l_t^{n,i}} & \frac{a}{l_t^{n,i}} \end{bmatrix} && (2.60) \end{aligned}$$

In Fast-SLAM, the decisions in the update step are made on a particle basis. All the particles have their own maps and data association decisions are given for each particle independently. This enables representing different maps. In EKF-SLAM, there is a single map and wrong decisions leads to huge errors in the later steps. However in Fast-SLAM, particles with wrong decisions may be eliminated in time.

### 2.5.7. SLAM With Visual Sensors

Visual-SLAM (V-SLAM) addresses the SLAM problem in the case when the only sensor of the robot is stereo or monocular camera. The cameras are cheaper and generally provide better knowledge about the environment [26]. The usage of cameras requires vision processing and information extraction in the unknown environment. The robot should extract identifiable features from an image and also should identify them in an image taken from a different perspective.

In [29], the V-SLAM problem is addressed with an offline method. It has three steps. First, the robot navigates an environment and records a video sequence and

generates a 3D model by processing the frames in the video. Then, a 3D map is built with the motion model of the robot and odometry readings. Finally, the robot pose is estimated by localization algorithms given the generated map.

The SLAM problem domain requires re-observation of a landmark from different distances and orientations. In [30, 26, 31], the Scale Invariant Feature Transform (SIFT) Descriptors are used to extract features from the image. SIFT Descriptors [32] are a set of interest points or keypoints extracted from an image and identify the features of the objects in the image. They are also invariant to image scale and rotation. In the applied SLAM algorithms, local graphs are formed with the SIFT descriptors and each local graph represents an observation of a landmark. In domains with stereo camera, the local graphs in the two images are matched with each other and distance of the feature can be calculated with the differences in the graphs. The graphs are also stored in a long term memory to compare with newly observed landmarks, so a part of the data association problem is addressed in the preception step.

In a platform with a monocular camera, the bearing information of an object can be extracted but it is impossible to extract the distance information from SIFT descriptors. In [19, 20], this problem is addressed. Instead of initializing a new landmark as soon as it is observed, the initialization step is delayed. As the robot observes a new object in the image, it initializes a Gaussian distribution representing its position in the 2D space with a very high uncertainty at the axis along robot to object. In the consecutive observations from different directions, it generates new Gaussian states for the same landmarks. When there are sufficient number of observations, multiple Gaussian hypothesis are merged into one and more accurate estimation which also initializes the landmark with the distance information.

In [21], the positions of the SIFT graphs in the image is estimated with a Gaussian distribution and in the consecutive frames, the local perimeter of the estimation is searched for the same SIFT graph. This idea assumes that the position of the object in an image does not excessively change.

### 2.5.8. Multi-Robot Challenge

The SLAM problem in the single robot case has successful solutions in the literature. The robots can navigate and explore both unknown indoor and outdoor environments. However using multiple cooperative robots may decrease the exploration time, increase accuracy and eliminate failure risk of the whole system. The solutions to the multi-robot SLAM problem are in their early stages and need exploiting for robust real time applications.

The challenges in multi-robot SLAM problem may vary with the application or environment type, but they have some common challenges. While specifying these challenges, the aim of multi-robot SLAM is assumed to reducing exploration time and increasing the map accuracy.

- If the initial positions of the robots are assumed to be known, the local maps of the robots should be merged and updated.
- If the initial positions are not known, it means coordinate frames are not the same for robots and correct transformation parameters must be found.
- Landmark identities are generated online, so the robots must negotiate on the landmark identities.
- Local maps may not intersect, so the robots have to meet at some point.
- If the problem involves more than two robots and robot-to-robot observations are used, the robots must be identified.
- The robots must perform collaborative exploration for efficiency.
- All the operations must require reasonable amount of time, memory and communication resources.

Some researchers introduced solutions where, it is assumed that two robots can observe each other at the same time [33] or robots know their initial positions. In this case finding the coordinate transformation parameters is obsolete. The landmark identities are merged with nearest neighbor approached and a joint map is maintained.

In another approach [34], the transformation parameters are searched with the heuristic of geometric configuration of landmarks in the local maps. If a good hypothesis is found, joint map is maintained. This approach does not rely on robot to robot identification, instead, assumes that explored areas of robots intersect.

In [35] and [36], the most complete multi-robot algorithm among others is introduced. The approach is designed for large group of robots and tested with 100 robots exploring an area with laser range finders and cameras. Robots generate hypothesis for coordinate transformation parameters by localizing themselves in other robots' maps. They forms small groups when they observe each other and share their maps.

# 3. PROBLEM AND PLATFORM SPECIFICATIONS

## 3.1. Problem Specifications

Our goal in this Thesis is to establish a multi-robot system in which the robots work together to create the map of a real world indoor environment. As stated in Section 2.5.8, the multi-robot SLAM problem is a new and ongoing research topic and none of the approaches are sufficient for a real world application without a prior knowledge and assumptions. We want our problem specification to be as realistic as possible and feasible enough to be solved with the current solution methods and our capabilities. For this purpose, we limit the problem specification in different aspects.

The environment is an indoor environment. An outdoor environment is much more difficult and unpredictable than an indoor environment. SLAM algorithms are highly sensitive to odometric errors [25]. Generally, robots for indoor environments have smaller odometric errors than outdoor robot platforms. Lighting conditions, which directly affect visual perceptions, are fixed in indoor environments and increases perceptual accuracy. Another reason for indoor environment is that our hardware platform is an indoor robot and will be explained in Section 3.2.

We want to develop algorithms which can be used on fully autonomous systems, so our robot platform will be fully autonomous and this implies the real-time execution of the applied algorithms.

Range finders are accurate and reliable sensors and very suitable for mapping applications. However, visual sensors are cheaper and can provide more information than range finders. Therefore, *Visual-SLAM* problem has took attention from researchers and it is showed that using a visual sensor in a completely unknown environment as a perception source brings several challenges [26, 37] and addressed by computer vision research field. In our work, the main observations for SLAM algorithms are provided by a visual sensor, but to ease the challenge, external guidance for perceptual module

are placed to the environment. In Figure 3.1, an image of a landmark from the camera of the robot is given. The visual guidance are prepared identically so that a robot can recognize but cannot identify them.

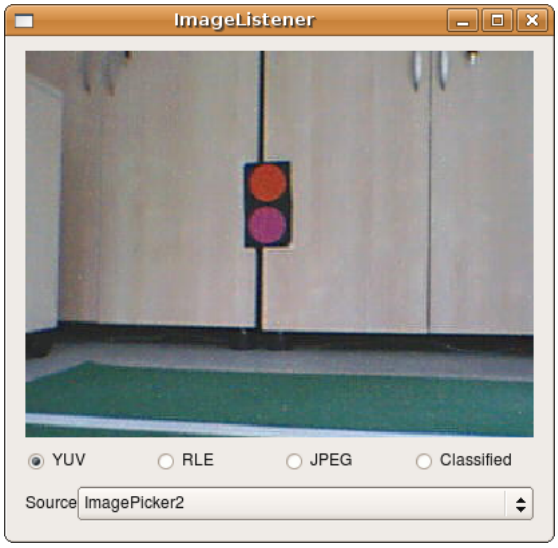



Figure 3.1. Sample visual guide from the camera of the robot.

There are two robots and they can observe the distance and bearing of each other visually. They do not know the initial position of the other, but we assume that during exploration, they will be in a situation such that one robot can observe other. The action planning to the goal of meeting with unknown initial positions is another problem and out of scope for this work. Though the robots should have an autonomous planning module capable of exploring enough area and obstacle avoidance.

## 3.2. Platform Specifications

### 3.2.1. Hardware Platform

The hardware platform for the application is the Festo Robotino robot [38]. The Robotino is an indoor mobile robot with on-board computing devices produced for education and research purposes. The hardware features we are interested in our work are:

- On-board computer with 500 MHz CPU, 64 MB RAM and 256 MB flash disk.
- It has swedish wheels which provide omnidirectional movement ability.
- The visual sensor is a webcam with a USB connector and can provide images with 320x240 resolution and 50 degrees field of view.
- There are bumpers around the bounds of the chasis.
- A wireless access point enables connectivity to any wireless network.
- In addition to the standard hardware, we equipped the robot with a URG laser range finder device [39]. The range limit is 5 meters and field of view is 240 degrees. However, physical placement of the laser device allows only 140 degrees of field of view.

The robots are marked with a colored paper visible from all angles and enables visual robot observations. A pre-built overhead camera system [40] is used for performance measurements in the real world experiments. The overhead camera system can tell the location and orientation of specific markers with respect to the global coordinate system. A specific marker is placed on top of the robots for overhead camera usage. In Figure 3.2, an image of the robot platform is given.

Figure 3.2. Festo Robotino robots are used as the hardware platform.

### 3.2.2. Software Platform

The on-board computer has a Debian/Linux based real time operating system and hardware communication is established with a library called *RobotinoAPI*. The processing power of the on-board computer is poor for our needs, therefore we run our controller code on another host and communicate with the robot via wireless network.

For the communication between the robot and the controller code, the *Player/Stage* framework is used [41]. The *Player/Stage* is a software tool for developing robot controllers. Most of the robot platforms have similar hardware and controller codes are developed almost same way except small changes in the low level hardware communication. The idea in *Player/Stage* project is to define abstract device interfaces and make controller codes transparent from the robot hardware. The hardware specific implementations of device interfaces are called *drivers* and developed separately for each robot or device.

The abstraction makes things simpler in the controller software. For example instead of defining a servo motor as an interface, the mobility of a robot is defined as a mobile object in 2D coordinate frame and control commands defined in the interface are reduced to velocity commands for robot. The robot specific drivers provide the mobility interface by implementing their own hardware controllers.

One advantage of such a software design is that once a controller code is implemented, it can be used for another hardware platform without modification. This hardware platform is also simulated in *Stage* module of the tool. The *Stage* is a simulator which implements all of the defined hardware interfaces including the camera in a 2D world. We use *Stage* to simulate the Robotino and indoor environment.

In our platform implementation, a driver for Robotino is built. It provides two interfaces; *BlobFinder* and *Position2D*. The driver for URG laser device (laser interface) is pre-built in the *Player*. Figure 3.3 shows the interaction between driver, controller and simulation. The *BlobFinder* interface provides landmark observations as blobs in

the image as well as their distance. The *Position2D* interface accepts motion commands as speed values in forward, sideways and angular dimensions. It also provides odometry information. The interface implementation details will be explained in Sections 4.1 and 4.2. We are not concerned with the simulation since all the interfaces are implemented in the simulation code and provide perfect knowledge.

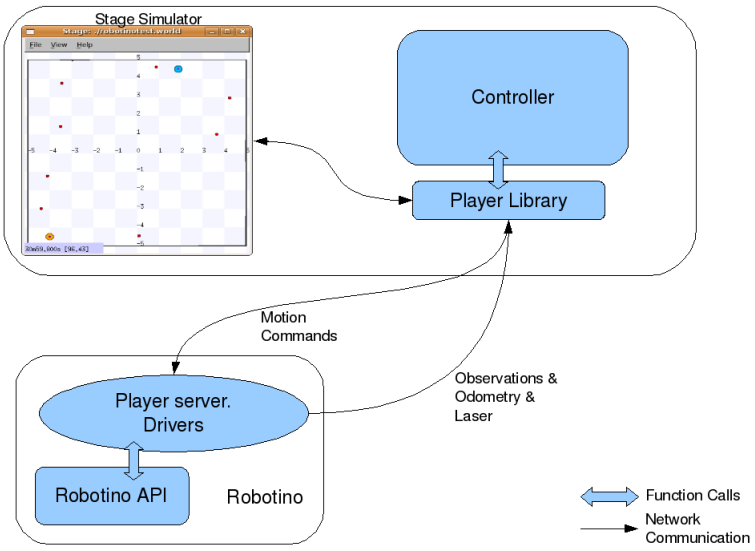



Figure 3.3. Player/Stage platform.

Another tool used during development is the *BOUNRobotics* software, developed by robotics group of our University [42]. This tool is mainly for debugging and monitoring a robot controller code and it provides basic robotic utility routines. It has two main modules, *BOUNLib* and *CerberusStation*. *BOUNLib* is a library including core functionalities, data representations utility routines. *Kalman Filters* are implemented as *BOUNKalman* library as a part of *BOUNLib* library. It also can log data and send robot states over UDP protocol.

The *CerberusStation* is a monitoring tool with a graphical interface. It can receive log messages sent from a robot using the *BOUNLib* library. It can also replay a recorded log file or even test our code with the inputs logged in the file. The interaction of the driver, controller and *CerberusStation* is given in Figure 3.4.

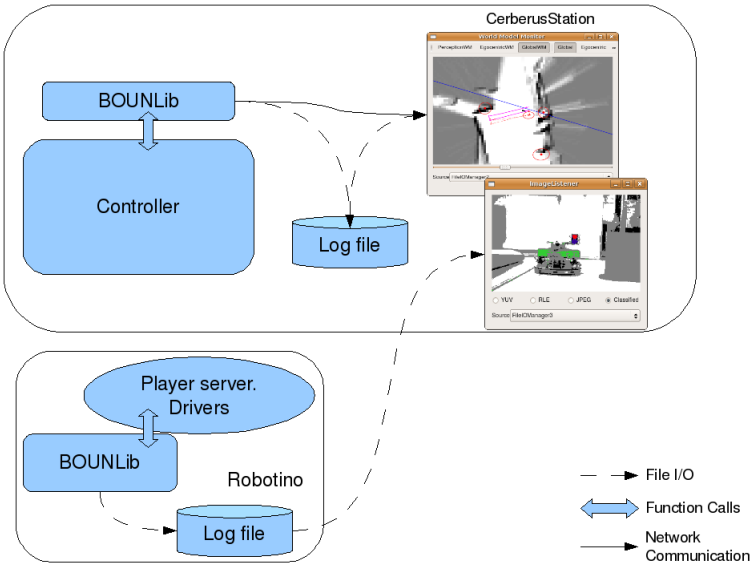



Figure 3.4. Usage of *BOUNLib* library and *CerberusStation* tool.

To summarize the platform capabilities, we can run our algorithms on real robots and simulation without any modification. We can record log files of system inputs and outputs and replay them. We can test our algorithms offline from log files. We can view all the robot state visually both onlie or offline.

# 4. METHODOLOGY

In Section 3.2.2, the problem scope is specified and the underlying platform to solve it is given. In this chapter the methods for solving the problem is introduced and explained in detail.

The system is separated into different modules with their own tasks. Each module has a specific task and provides outputs as inputs for other modules. The modules and their tasks are as follows.

- **Driver** module runs on the robot. It provides landmark observations and executes motion commands while providing odometry readings.
- **Slammer** module runs on the controller host and implements SLAM algorithms. Inputs are odometry, observation and messages from the other robot. The output is the landmark map and the self position estimate.
- **Grid Mapper** module runs on the controller host. It takes position estimate and laser readings as input and provides a grid map of the environment.
- **Planner** module runs on the controller host. It takes laser readings and position estimate as inputs and outputs necessary movement commands.

The modules and their interactions are given in Figure 4.1. Each module runs sequentially and waits for the dependent module if needed. The only outside interaction is a robot with the same system so the system is fully autonomous and distributed.

## 4.1. Landmark Detection

As mentioned in section 3.2.2, the *driver* module provides a *BlobFinder* interface. This interface specifies a driver which finds connected components in an image and calculates the parameters of a region. In our domain, a landmark is defined with two colors, but in the driver, we find the color groups and merge them as if they are one object in the image.

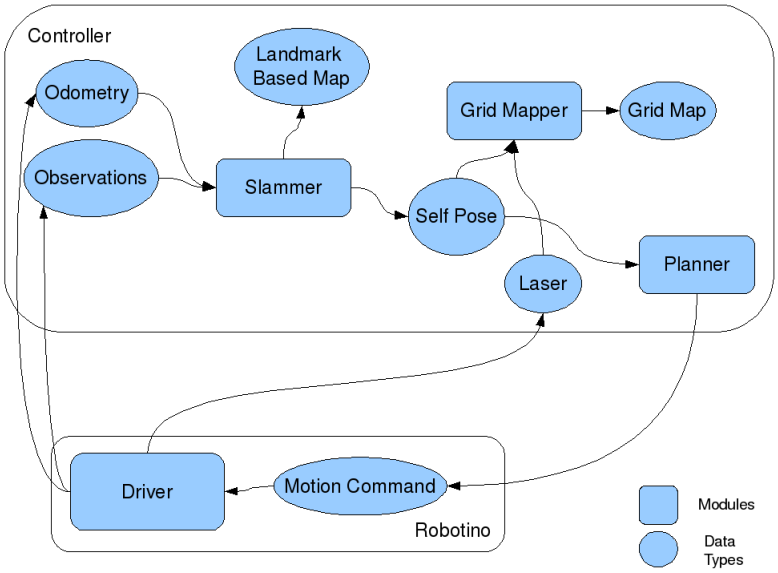



Figure 4.1. The main modules, data types and their interactions.

### 4.1.1. Color Classification

In the implementation on the Robotino driver, we used the *BOUNImage* library in *BOUNLib*.The *BOUNImage* library can take a raw image (RGB, YUV, YUV2), classify pixels into color groups and finds regions. The *RobotinoAPI* provides raw image in the jpeg format but we convert it to the RGB format. In the raw image format, each pixel is represented with a three-byte value and can be one of the $255^3$ values. Since it is impossible to efficiently operate on such an input space, the colors are classified into fewer color groups. Each group represents a possible color on the environment. These are white, green, yellow, blue, robot-blue, orange, and red. There is actually one more group which stands for none of the groups and noted as the "ignore" group.

To summarize the problem, we have an image with a resolution of $320x240$, where each pixel has a range of $255^3$ values and we want to assign each pixel to one of the eight color groups. This is a *regression* problem and we used Generalized Regression Neural Network (GRNN) to solve this problem [43]. GRNN is a neural network which can approximate a function (linear or nonlinear) with $n$ dimensional domain to $m$ dimensional output domain.

In our problem, the input space has three dimensions and the output space has eight dimensions. Before training a GRNN network, a training set must be formed. For this purpose, we use our *Labeler* software in *CerberusStation*. In the *Labeler*, an image in its raw format is converted to RGB format and is displayed. The user marks the sample pixels with their color group. When sufficient number of samples are collected from different images, it is saved as the training set. In Figure 4.2, a screen shot from *Labeler* software is shown.

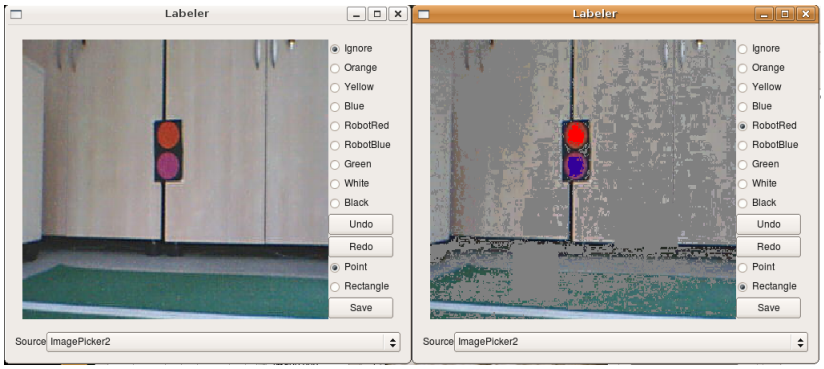



Figure 4.2. Screenshot from *Labeler* software. Grey region is the ignored color group.

After the creation of the training set, GRNN is trained. To efficiently get outputs from the GRNN network, a look up table is constructed for all possible inputs. To look up the color group of a pixel, $R$, $G$ and $B$ values are used to calculate the unique index and the value at that index gives the color group ID. In Figure 4.3, output of a trained GRNN is tested on a sample image.

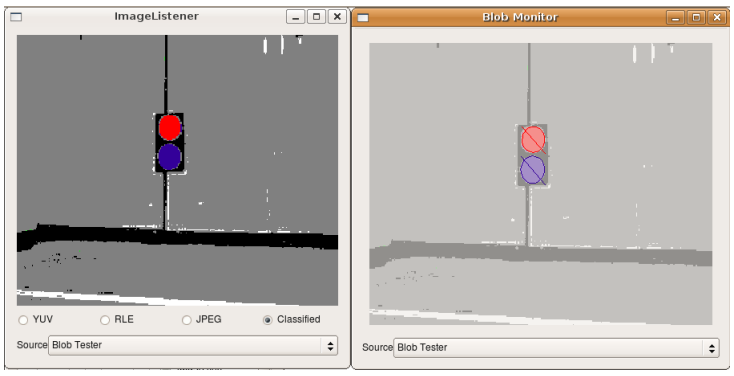



Figure 4.3. A classified image constructed with a trained GRNN and formed regions.

### 4.1.2. Distance Calculation

On the classified image, pixel groups with same color group form a region. The orientation of the recognized object to the robot is calculated by the ratio of region center to the center to the image and the field of view for the camera. However calculating the distance of the object is not trivial as calculating the orientation.

To calculate the distance of a visually observed object, either a stereo vision system or using consecutive frames of single camera as stereo vision is needed [26]. We have a single camera, but the sizes of the objects in the environment are fixed and known. The distances of objects can be inferred by the size feature of the objects. In visual observations, the size of an object can be measured by the width or height of the object in pixels. As the object gets closer, the size of the object in the image increases and this increase has a nonlinear relationship with the actual distance of the object. In our approach, we represent this relationship with the function $y = a \times (x^b) + c$ where $y$ is the distance, $x$ is the width or height of the object in pixels, and $a$, $b$ and $c$ are parameters of the function.

Now the problem is reduced to finding the three optimal values for $a$, $b$ and $c$. To optimize these parameters, a training set is collected from the environment and the parameters are estimated with the least squares method. The pixel widths are found from the classified image, so this training procedure is repeated for each object type and after each color calibration process.

## 4.2. Odometry Calibration

The *Position2D* driver provides odometry readings to the controller. The odometry readings are actually provided by the *RobotinoAPI*. In order to apply current pose estimation methods, the odometric measurements should have zero mean noise. However we observed non-zero mean noise in our experiments. These errors are called systematic errors [44] and should be eliminated. Otherwise none of the algorithms can bound the odometric error over time.

The motion commands are one of three types; go forward, turn right or turn left. The speed parameter is constant in all types of motion commands. Since we have systematic errors, when we apply the same motion command sufficient number of times, we expect similar amount of difference between the calculated and the observed poses. The purpose of the odometry calibration is to estimate the systematic errors in different action types and modify the odometry readings with the opposite of the error.

The first action type is going straight forward. To measure the error, the robot is moved forward until the cumulative odometric measurements report near 1.5 meters displacement. The displacement does not have to be exact 1.5 meters since we are only interested with the difference between the reported position by the robot and the measured position. Figure 4.4 represents the experiment setup for collecting required data.

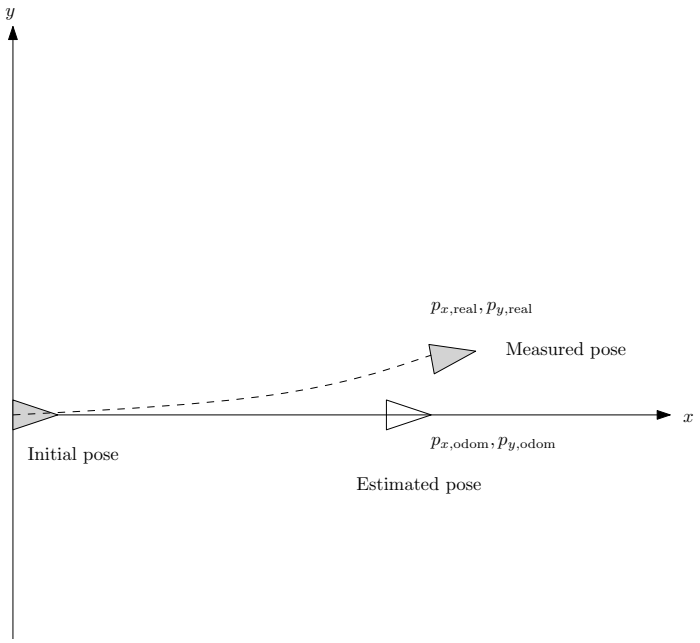



Figure 4.4. Setup for odometry calibration procedure.

The starting position of the robot is selected as the origin of the coordinate system and the robot is headed to the $x$ axis. Let $(p_{x,odom}, p_{y,odom}, p_{\theta,odom})$ represent the pose reported by the odometric measurements and $(p_{x,real}, p_{y,real}, p_{\theta,real})$ represent the point measured by the overhead camera in the robot coordinate frame in the end of the run. The difference is the cumulative result of the readings during $\Delta t$ time steps.

To collect sufficient statistics, the experiment is repeated $n$ times and the estimation of systematic error is calculated as the mean of them. With these information, the modified odometry measurement in each time step for forward motion becomes:

$$\Delta x = \Delta x_{orig} + \frac{\sum_{i=1}^{n} \frac{p^i_{x,real} - p^i_{x,odom}}{\Delta t^i}}{n} \tag{4.1}$$

$$\Delta y = \Delta y_{orig} + \frac{\sum_{i=1}^{n} \frac{p^i_{y,real} - p^i_{y,odom}}{\Delta t^i}}{n} \tag{4.2}$$

The same procedure is repeated for the other two types of actions, with the exception that displacement in position is ignored, because the robot can turn around at the same point. Only the systematic error in the orientation is estimated and modified. Similar to Eq.(4.1), the corrected odometry measurement for the turning type actions becomes:

$$\Delta\theta = \Delta p_{\theta,orig} + \frac{\sum_{i=1}^{n} \frac{p^i_{\theta,real} - p^i_{\theta,odom}}{\Delta t^i}}{n} \tag{4.3}$$

## 4.3. Single Agent SLAM

The *Slammer* module is the main focus of our work. The robot is put in an unknown environment and this module is responsible for creating a landmark based map and estimating the pose with respect to this map. In this section, the methodology is covered for a single agent.

The initial position of the robot with respect to the global coordinate frame is not given to the robot, so the starting position is simply selected as the origin for the rest of the execution time and the $x$ axis lies along the robot direction. In other words, the robot is initially at the point $(0, 0)$ with a heading of 0 degrees. In the rest of the document, the global coordinate frame refers to the coordinate frame originated from the starting position.

The state prediction and update steps are implemented in two ways; *EKF* method and *Fast-SLAM* method. The input and output of the two methods are the same, so the module can easily configured to run with either of them.

### 4.3.1. Data Association

The *Driver* module provides landmark observations as $Z_t = \{z_t^1, z_t^2, ...z_t^K\}$ at time $t$ where $z_t^i = \{l_t^i, \theta_t^i\}$. The distance $l$ and orientation $\theta$ are robot centered. However the map representation is $M = \{m_1, m_2, ..., m_L\}$ where each entity $m_i = \{p_{x,m}^i, p_{y,m}^i\}$ represents the positions of the known landmarks in the global coordinate frame. Before updating any belief state, we have to match observations to the known landmarks or declare a new landmark.

For this problem, the *Nearest Neighbor* approach is used as in Figure 2.13. However the algorithm is naive in initializing a new landmark. If an observation is not close enough to any known landmark, a new landmark is immediately created and started to estimated. However, since the observations are noisy, the new landmarks are delayed in an initialization set. The set is called the *candidate set* and in addition to the landmark information, it keeps counts of encounters for each landmark candidate. When a candidate is seen enough times, it is transferred to the actual landmark set. The point estimate of the landmark is initialized as the mean of the encounters.

### 4.3.2. EKF-SLAM Method

The EKF and EKF-SLAM method is detailed in sections 2.5.5 and 2.2.2. In our problem, in order to use the same algorithm, the observations are passed to the data association algorithm as in section 4.3.1. Once a new landmark added to the map, the state vector and error covariance is initialized for the corresponding landmark. For simplicity and real-time constraints, the maximum number of landmarks. are limited. If the data association algorithm decides more than allowable number of landmarks, they are omitted on the EKF state. The EKF-SLAM mode of the *Slammer* module is given in Figure 4.5.

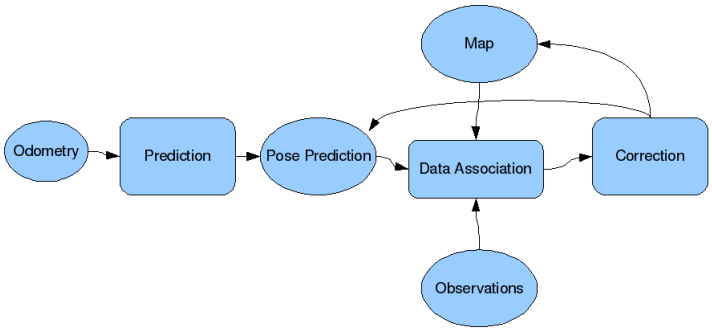



Figure 4.5. EKF-SLAM implementation flowchart.

The parameter selection for the EKF is not a trivial problem. There is a set of parameters and there is a tradeoff between landmark accuracy and the pose accuracy. If the parameter are set such that the odometric noise is assumed low and observation noise is assumed high, the algorithm may not bound the odometric errors and diverge. Similarly if the observation noise parameters are set too low, the robot would believe any noisy observation and map accuracy would drop down. The state vector size is 3 for robot pose and 2 for each of $L$ landmarks totaling $3 + 2L$. The parameter set of the EKF-SLAM is:

- A 2x2 measurement noise covariance $Q$.
- A $3 + 2L$ x $3 + 2L$ process noise covariance $R$.
- A $3 + 2L$ x $3 + 2L$ initial error covariance $P$.

Our platform provides true robot position during an experiment in both real and simulated world. Therefore, we decided to use *Evolutionary Strategies* [45] for the optimization of EKF-SLAM parameters.

The parameter set is large for an efficient optimization application. But if we make some assumptions, the parameter set size can be reduced. If we assume uncorrelated noise, the off-diagonal elements becomes obsolete. In addition, if we assume same noise parameters at $x$ and $y$ axes in the coordinate system, the noise parameters

become:

$$Q = \begin{pmatrix} \omega_1 & 0 \\ 0 & \omega_2 \end{pmatrix} \tag{4.4}$$

$$R = \begin{pmatrix} \omega_3 & 0 & 0 & 0 & 0 & ... \\ 0 & \omega_3 & 0 & 0 & 0 & ... \\ 0 & 0 & \omega_4 & 0 & 0 & ... \\ 0 & 0 & 0 & \omega_5 & 0 & ... \\ 0 & 0 & 0 & 0 & \omega_5 & ... \\ . & . & . & . & . & . \\ . & . & . & . & . & . \end{pmatrix} \tag{4.5}$$

$$P = \begin{pmatrix} \omega_6 & 0 & 0 & 0 & ... \\ 0 & \omega_6 & 0 & 0 & ... \\ 0 & 0 & \omega_7 & 0 & ... \\ 0 & 0 & 0 & \omega_8 & ... \\ . & . & . & . & . \\ . & . & . & . & . \end{pmatrix} \tag{4.6}$$

If we name the parameters:

- $\omega_1$ is the observation distance noise.
- $\omega_2$ is the observation orientation noise.
- $\omega_3$ is the process pose displacement noise.
- $\omega_4$ is the process pose orientation noise.
- $\omega_5$ is the process landmark displacement noise.
- $\omega_6$ is the initial pose displacement noise.
- $\omega_7$ is the initial pose orientation noise.
- $\omega_8$ is the initial landmark displacement noise.

The vector $\omega = \{\omega_1, \omega_2, ..., \omega_8\}$ forms the chromosome of an individual. The training data is collected from simulator and logged to a file without noise. At the

training step, for each individual, random Gaussian noise injected to the perfect inputs and the individual is tested for the path until the end of episode in the log file. The fitness function is the negative of the cumulative mean square error during an episode. To avoid unnecessary continuation of an episode, if the error between the estimated and the true pose is very large, the episode is stopped and a small fitness value is returned. For the real world, the procedure is same except noise is not injected to the logged data.

### 4.3.3. Fast-SLAM Method

The second applied method is the Fast-SLAM algorithm as described in the Section 2.5.6. Like the EKF-SLAM algorithm, this method needs identified landmark inputs to work, however, in Fast-SLAM method instead of only one single map estimate, all the particles have their own map estimate. In our implementation, each particle decides the observations based on their own map with the algorithm described in Section 4.3.1. This may lead results such than an observation is matched with a previously seen landmark for one particle and declared as a new landmark for other particle. The general flowchart for FastSLAM algorithm is given in Figure 4.6. Recall that particle $i$ has a pose estimate $(p_t)^i$ and a map estimate $M_t^i$ at time $t$. The

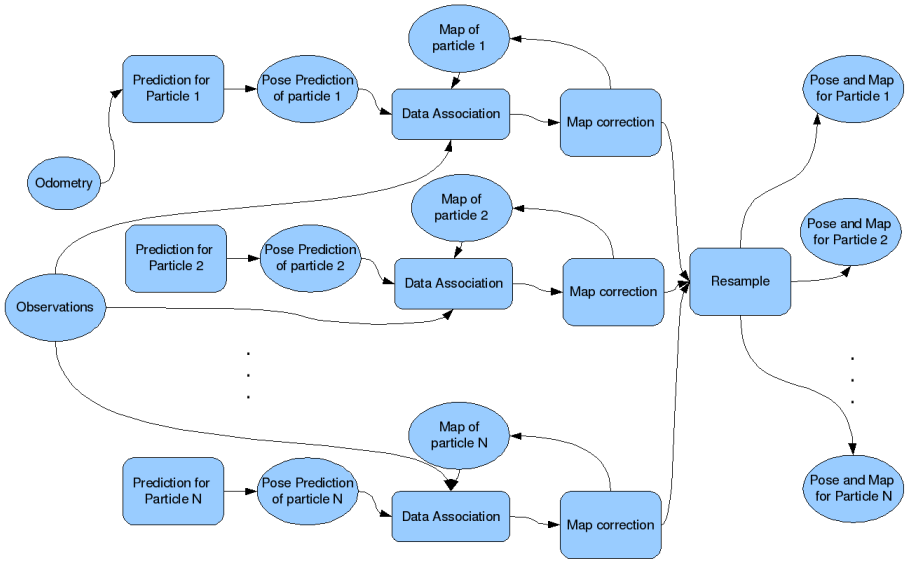



Figure 4.6. Fast-SLAM implementation flowchart.

algorithm for the applied Fast-SLAM algorithm is given in Figure 4.7. The first step of the algorithm is the prediction of pose for each particle. The lines 1 to 3 are implementation of the sampling from Gaussian distribution. Poses of each particle is calculated with odometry and random Gaussian variables are added to the components $p_x, p_y$ and $p_\theta$. The function $normrand(\mu, \sigma)$ stands for a random variable drawn from Gaussian distribution with mean $\mu$ and variance $\sigma$.

After predicting the particles, for each observation $z_t^i$, the corresponding index $k$ in the map of each particle $M^j$ is found with the data association algorithm in Figure 2.13. If the map size increases after data association algorithm, this means the observation is declared as a new landmark. The importance weight is calculated with the distance between the observation and the corresponding entity in the map. If it is the case that the observation initializes a new landmark, the importance weight would be maximum and this may lead misqualifiation of the particles with wrong landmark decisions. To avoid this, if a particle declares an observation as a new landmark, the importance weight is set to a small value so that the importance weight of other particles are bigger. In the other case, the map entity is updated with the *EKF* equations in the algorithm 2.6.

At the resampling step, each particle is copied to the new set according to their importance weight. A particle with higher importance weight is copied more than a particle with lower importance weight. And finally, if the new set has missing particles due to floor approximation in step 24, they are filled with randomly selected particles.

4.3.3.1. Information Extraction. The Fast-SLAM algorithm represents both the pose and the map estimate with a set of particles. However in other algorithms, we may need a single best estimation of both pose and the map. In other words, a valuable information should be extracted from the particles. The common approach is calculating the weighted average of the particles. However, to obtain a smoother estimate over time, we calculate the weighted average with the a subset of best particles. Let $N'$ be the number of elements in the smaller subset. Only best $N'$ particles are used

**Algorithm** $FastSLAMStep(X_{t-1}, Z_t, u_t)$

**for** $j = 1$ to $N$ **do** {For each particle}
  $(\hat{p_t})^i \leftarrow g((p_{t-1})^i, u_t) + [normrand(0, \omega_3), normrand(0, \omega_3), normrand(0, \omega_4)]$ {Calculate new pose and add random dissortion}
**end for**
**for** $i = 1$ to $K$ **do** {For each observation}
  **for** $j = 1$ to $N$ **do** {For each particle}
    $previousMapSize \leftarrow size(M_t^j)$
    $k \leftarrow MLDataAssociation((\hat{p_t})^j, z_t^i, M_t^j)$ {Algorithm in Figure 2.13.}
    **if** $previousMapSize < size(M_t^j)$ **then** {If $z_t^i$ is declared as a new landmark.}
      $w_i \leftarrow aSmallValue$
    **else**
      $EKFUpdate((\hat{p_t})^j, z_t^i)$ {Perform EKF update for the $k$th map entity.}
      $w_i \leftarrow p(z_t^i|(\hat{p_t})^j, m_{t,k}^j)$ {Eq.(2.34)}
    **end if**
  **end for**
  $normalizeWeights$
  $resampleIndex \leftarrow 1$
  **for** $j = 1$ to $N$ **do** {For each particle}
    **for** $a = 1$ to $\lfloor w_j N \rfloor$ **do**
      $x_t^{resampleIndex} \leftarrow \hat{x_t}^j$
      $resampleIndex \leftarrow resampleIndex + 1$
    **end for**
  **end for**
  **for** $j = resampleIndex$ to $N$ **do** {Fill remaining particles randomly}
    $index \leftarrow uniformrand(0, 1) * N$
    $x_t^{resampleIndex} \leftarrow \hat{x_t}^{index}$
    $resampleIndex \leftarrow resampleIndex + 1$
  **end for**
**end for**

Figure 4.7. The applied FastSLAM algorithm.

in information extraction. The pose estimation is:

$$p_t = \frac{\sum_{i=0}^{N'} (p_t)^i w_i}{\sum_{i=0}^{N'} w_i} \tag{4.7}$$

Similarly, the map estimate is the weighted average of best $N'$ particles. Recall from Eq.(2.55), the map estimate also contains a covariance for each entity.

$$m_{j,t} = \frac{\sum_{i=0}^{N'} m_{j,t}^i w_i}{\sum_{i=0}^{N'} w_i} \tag{4.8}$$

$$\Sigma_{j,t} = \frac{\sum_{i=0}^{N'} \Sigma_{j,t}^i w_i}{\sum_{i=0}^{N'} w_i} \tag{4.9}$$

## 4.4. Multi-Robot SLAM

In our problem specification, as described in Section 3, the robots do not know the initial positions of each other and they will meet at some time during execution. Our focus in the multi-robot case is merging the maps of two robot so that they will know the explored area of each other. However, since the estimations have uncertainties of both applied algorithms for single case, we should also propagate the uncertainties while merging the maps.

The behavior of the robots when they meet will be explained in Section 4.6. In this section, we assume the robots have observed each other, negotiated and exchange internal states. Our algorithm is also decentralized, and we will cover the algorithm from the point of view for one robot. The other robot will be mentioned as "other robot", but note that the same algorithm is executed for the other robot at the same time.

The EKF-SLAM and Fast-SLAM have different state representations but we can gather a common state representation from both of the algorithms. From the EKF-SLAM algorithm, we already have mean vectors of map entries as in Eq.(2.40) and the 2x2 covariance matrices are extracted from the diagonal of $(3 + 2N)$x$(3 + 2N)$ state

covariance. From the Fast-SLAM algorithm, the map entries are extracted as detailed in Section 4.3.3.1. The message contents received from the other robot is:

- $M_{other}$: the map of the other robot with the common representation. Each entity $m_{other,i}$ contains the position $p_{other,i}$ and a 2x2 covariance $\Sigma_{other,i}$ of the $i$th landmark.
- $p_{other}$: the pose of the other robot. Note that $p_{other}$ is the other robots pose and $p_{other,i}$ is the position of the $i$th landmark of other robot.
- $z_{other,self} = \{l_{other,self}, \theta_{other,self}\}$ is the observation of the other robot to self robot.

In the single robot case, recall that robots has their own coordinate frame and the origin is the starting position of the robot. The first problem is to find a transformation matrix from other robot's coordinate frame to the global coordinate frame. The transformation matrix if formed with the translation values $tr_x, tr_y$ and rotation $tr_\theta$:

$$T = \begin{bmatrix} \cos(tr_\theta) & -\sin(tr_\theta) & tr_x \\ \sin(tr_\theta) & \cos(tr_\theta) & tr_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.10}$$

The definitions of the message contents and transformation parameters are visualized in the Figure 4.8. The diagram of overall map merging algorithm is given in Figure 4.9. Finding the transformation parameters is an analytic geometry problem. The distance obseravtions $l_{self,other}$ and $l_{other,self}$ may be different, so in calculations, the distance between robots is taken as the mean of two values. The transformation parameters

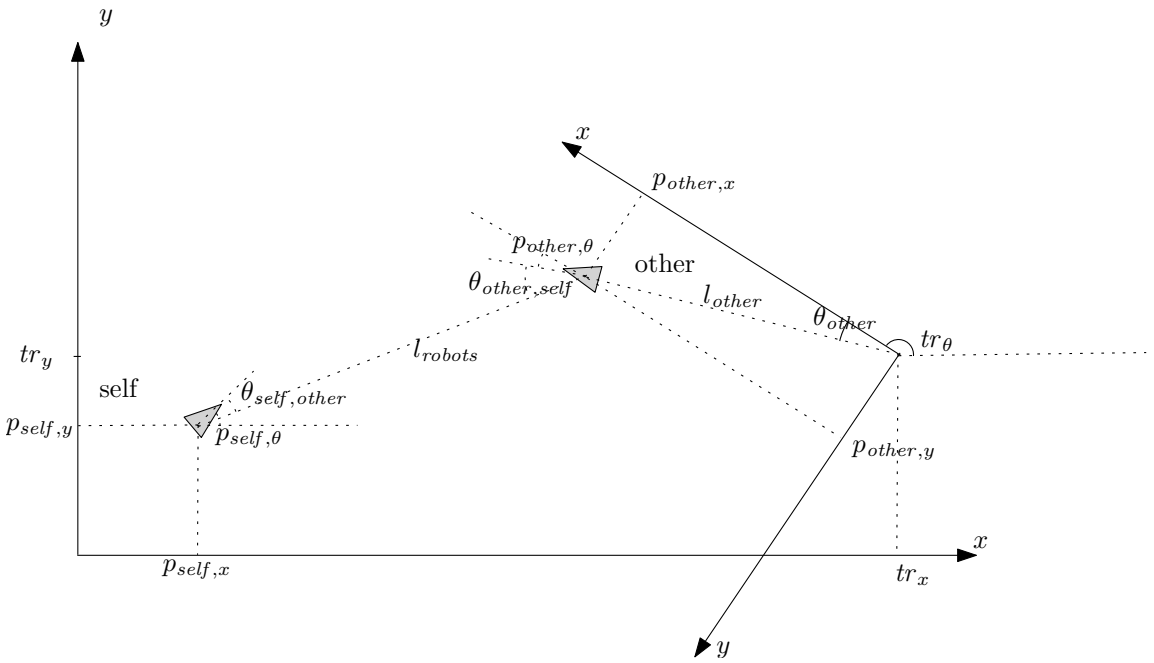



Figure 4.8. The variables when robots observe each other.

are:

$$l_{robots} = (l_{self,other} + l_{other,self})/2 \quad (4.11)$$

$$l_{other} = \sqrt{(p_{other,x}^2 + p_{other,y}^2)} \quad (4.12)$$

$$\theta_{other} = \text{atan2}(p_{other,y}, p_{other,x}) \quad (4.13)$$

$$tr_\theta = \pi + \theta_{self,other} + p_{self,\theta} - \theta_{other,self} - p_{other,\theta} \quad (4.14)$$

$$tr_x = p_{self,x} + l_{robots}\cos(p_{self,\theta} + \theta_{self,other}) + l_{other}\cos(tr_\theta + \theta_{other} + \pi) \quad (4.15)$$

$$tr_y = p_{self,y} + l_{robots}\sin(p_{self,\theta} + \theta_{self,other}) + l_{other}\sin(tr_\theta + \theta_{other} + \pi) \quad (4.16)$$

Once the transformation parameters are found, all the map entities from other robot should be transformed to the self global coordinate frame. To find the transformed map entity $m_{other,i'}$, the point $p_{other,i}$ and the covariance $\Sigma_{other,i}$ are transformed with the transformation matrix $T$ as:

$$\begin{bmatrix} p_{other,x,i'} \\ p_{other,y,i'} \\ 1 \end{bmatrix} = T \begin{bmatrix} p_{other,x,i} \\ p_{other,y,i} \\ 1 \end{bmatrix} \quad (4.17)$$

$$\Sigma_{other,i'} = T^T \Sigma_{other,i} T \quad (4.18)$$

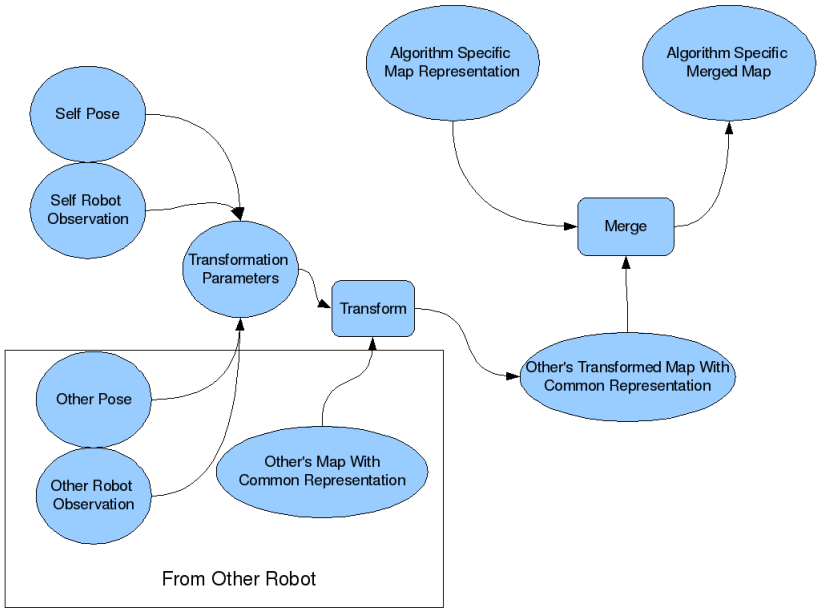



Figure 4.9. Diagram of multi-robot map merging algorithm.

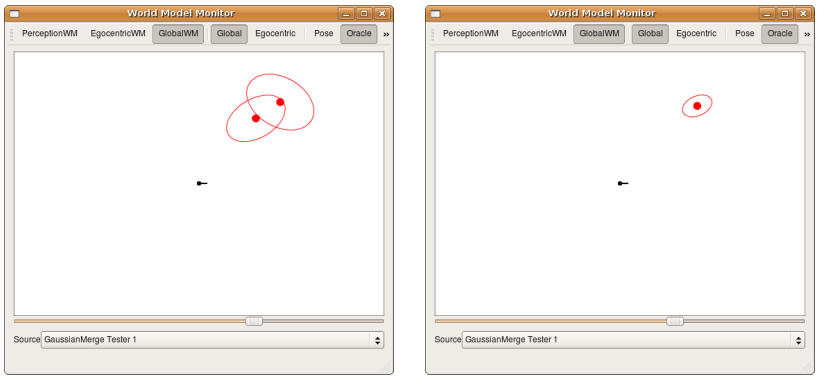
When all the map entities are transformed we have the map of other robot as the locations of landmarks and corresponding 2x2 uncertainty covariances with respect to the self coordinate frame. They should be merged with the map of the current applied SLAM algorithm (EKF-SLAM or Fast-SLAM).

For the EKF-SLAM method, an entity from the transformed map of other robot can either be already exists in the self map, or not. To find the correspondences, nearest neighbor algorithm is used as in Figure 2.13. Once the correspondences are found, the unmatched entities from the other map are just copied into the EKF-SLAM state. The correlations between the new landmarks and the existing landmarks are omited and set to 0. For the matched entities, the merged entity in the EKF-SLAM covariance is found with the equations:

$$\Sigma_{merged} = \Sigma_{self} - \Sigma_{self}\left[\Sigma_{self} + \Sigma_{other'}\right]^{-1}\Sigma_{self} \quad (4.19)$$

$$p_{merged} = p_{self} + \Sigma_{self}\left[\Sigma_{self} + \Sigma_{other'}\right]^{-1}\left(p_{other'} - p_{self}\right) \quad (4.20)$$

Where $\Sigma_{self}$ is extracted from the diagonal entries of the EKF-SLAM covariance. The Gaussian merging equations (4.19 and 4.19) are applied from [46] and is visualized in Figure 4.4.




(a) Two Gaussian (b) Merged Gaussian

Figure 4.10. Merging two Gaussian

For the Fast-SLAM method, instead of a single map, we have $N$ maps in the particle set, so the map of other robot should be merged with the map of each particle. Just like in the EKF-SLAM case, the corresponding entities are found between the map of each particle and the map of other robot. Unmatched entries are copied to the map of the particle and matched entities are merged with the equations (4.19 and 4.19).

## 4.5. Grid Mapping

In the SLAM algorithms in this work, landmark based maps are used. In addition to the landmark based map, we also create a location based map for better monitoring the robot state and have more informative map. An occupancy map is a location based map which contains the probabilities of the occupancy of any location in the area. However, if we want to estimate the occupancy map, since the area is a continuous plane, we would need infinite memory. Instead, we can approximate an occupancy map by dividing the area into small identical grids and representing the occupancy probabilities of the grids. This approximate map is called the occupancy grid map [2].

We denote the occupancy grid map with the matrix $c$. An element $c^t_{i,j}$ is the probability of existence of an obstacle within the corresponding cell at time $t$. The size of the matrix depends on the grid size and the bounds of the area to be covered. For instance if the interested area has 5 meters width in both dimensions, and the grid width is 10 centimeters, the matrix $c$ has $\frac{500}{10} = 50$ rows and columns.

At the beginning, all the cells are initialized with 0.5 probability. At each time step, the value of each cell is updated with the returned laser readings. The cells can be in one of three cases:

- The laser has returned from that cell. A positive evidence.
- The cell is in the trajectory of a laser beam but the beam returned higher distance value. A negative evidence.
- The cell is out of range. No evidence.

If the cell $c_{i,j}$ has a positive evidence, the value is updated as:

$$c^t_{i,j} = \min(c^{t-1}_{i,j} + \tau, 1) \tag{4.21}$$

Where $\tau$ is a small update value. Similarly if the cell $c_{i,j}$ has a negative evidence, the value is updated as:

$$c^t_{i,j} = \max(c^{t-1}_{i,j} - \tau, 0) \tag{4.22}$$

The cell value remains same if there is no evidence. The laser readings and evidences are visualized in Figure 4.11.

## 4.6. Planning

The Planner module provides autonomous exploration capability to our robots. It reads all the observations and state estimations, and outputs motion commands to the robot. The planning module should provide:

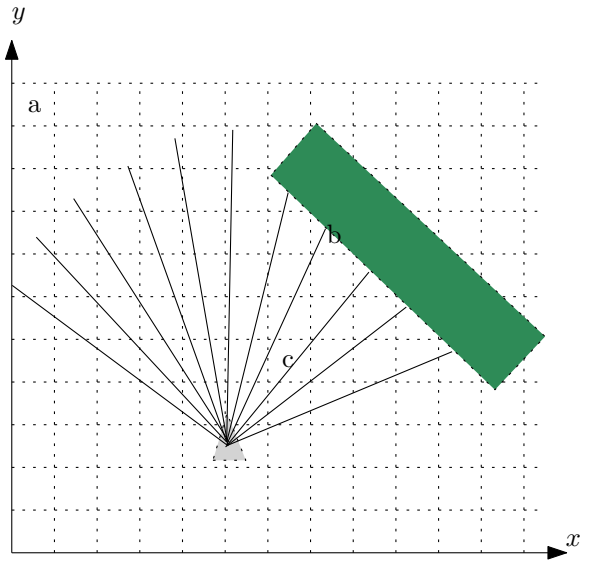



Figure 4.11. The three types of evidence in the grid mapping. In cell (a), there is no evidence, in cell (b) there is a positive evidence, in cell (c) there is a negative evidence.

- Simple obstacle avoidance behaviors.
- An exploration strategy that covers enough area.
- Ability to negotiate with other robot.

### 4.6.1. Exploration

The robot explores an unknown environment so it can encounter obstacles while exploring and should avoid them. On the other hand, it should explore the area and may encounter new landmarks. While doing all of these, it should make minimum amount of turns as possible because turnings increase odometric errors [47].

Our domain is an indoor environment, the obstacles are generally smooth surfaces of wall and the area is rectangular. The robot is equipped with a laser range finder. With the light of these, it is assumed that in the case of an encounter with an obstacle, the laser readings probably form a line.

In the low level reactive behaviors, if the robot encounters an obstacle ahead, it first tries to form a line with the laser readings. If a line can be formed, robot turns right such that it is parallel to the line. Then, it moves straight ahead until it encounters

another obstacle. The state machine of the algorithm is given in Figure 4.12. The line forming algorithm is the Random Sample Consensus (RANSAC) algorithm [48]. The algorithm forms lines with least squares method and excludes outlier points. Then forms another line to remaining points and excludes outliers. This iteration ends with a resulting line if there is not any outlier left and have enough sample points to form a line, or results without a line. The robot always turns to the same side so that it can

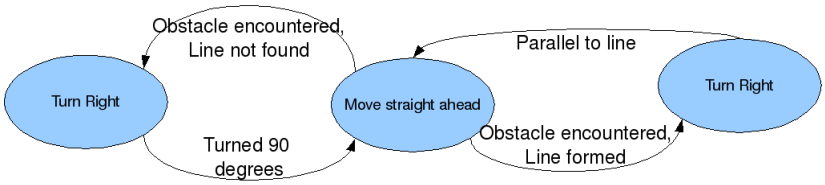



Figure 4.12. State machine of exploration algorithm.

follow the walls and explore surroundings. The robot also turns parallel to the line so that it would deliberately perform minimum number of turns. The expected behavior in the idea case is given in Figure 4.13.

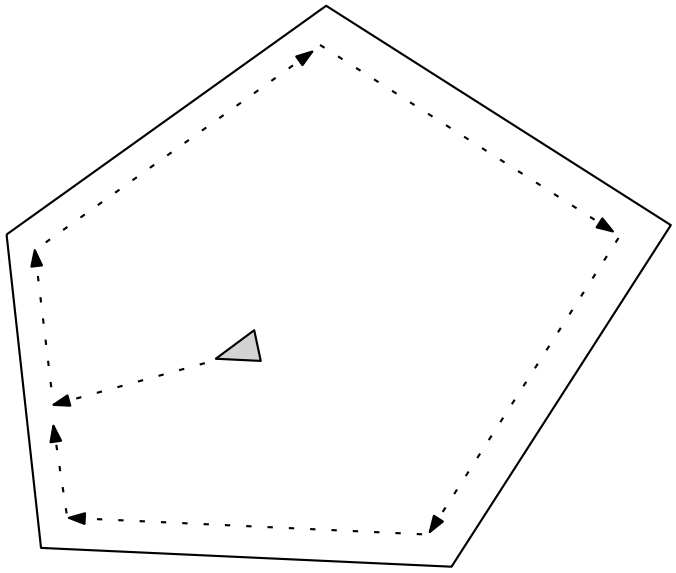


Figure 4.13. Expected behavior in ideal case.

### 4.6.2. Coordination and Negotiation

From the point of view of one robot; there is another robot and initial position is not known. At any time, it may observe other robot or be observed by it. The robot should be ready and react to both situations. The robots are constantly send messages to each other to inform about their state. Whenever a robot observes another, it informs the other robot that it is observed. After first encounter, the observer robot stops exploration, turns to the observed robot and waits for acknowledgment. At the same time, observed robot also stops exploring and starts turning to one side until it observes the other robot. When the two robots see each other, they exchange the maps and continue to exploration. The finite state machine of the algorithm is given in Figure 4.14. To avoid stuck on waiting other robot, if a robot waits for other robot

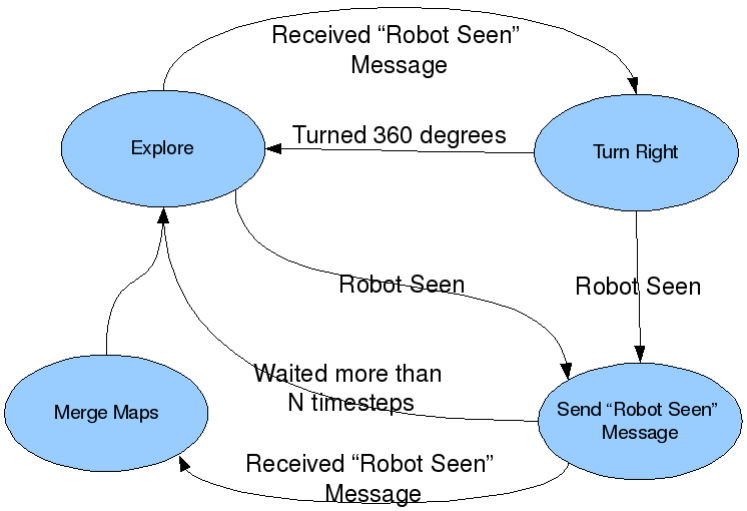



Figure 4.14. State machine for multi-robot negotiation.

more than some time steps, it skips the negotiation and keeps exploring.

# 5. RESULTS

In this Chapter, the experiment setups are explained and results are discussed. The experiments are performed in simulation and the real world. The simulation experiments provide a testbed with controlled conditions and comparison of the algorithms on exactly same conditions and inputs. The real world experiment setups aimed to inspect the applicability of the methods in the real world.

## 5.1. Simulator Experiments

### 5.1.1. Dead Reckoning versus SLAM Algorithm

The first experiment is designed to observe the effectiveness of the SLAM algorithm. The path estimation with dead reckoning (raw odometric measurements) and the estimation of the SLAM algorithms are compared. The experiment setup is given in Figure 5.1. The rectangular objects are the unique landmarks and the octagon shaped object is the robot.

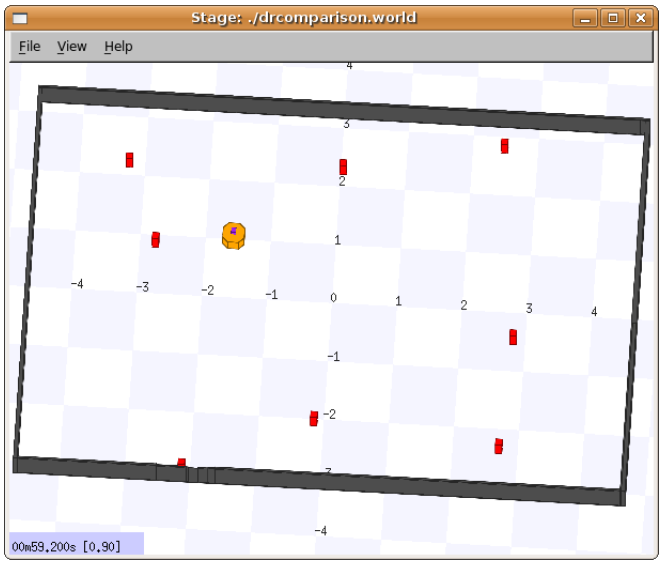
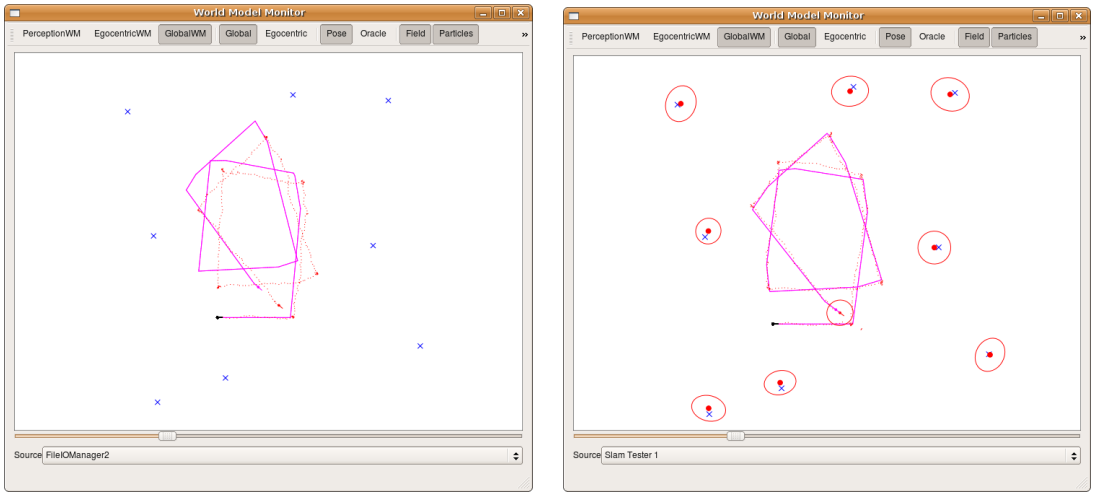



Figure 5.1. The experiment setup for single robot tests in the simulator.

The robot follows a predefined path. The exact location of the robot and the

odometry readings are taken from the simulator. In the test code, a Gaussian noise with zero mean and a variance is injected to both the odometry values and the observations. The noisy data is recorded to a file during the execution, and SLAM algorithms are tested with the CerberusStation tool.




(a) Dead reckoning (b) EKF-SLAM

Figure 5.2. Comparison of path estimates with dead reckoning and EKF-SLAM.

In Figure 5.2, the comparison of the path estimation with the dead reckoning and the EKF-SLAM is given. The snap shot is taken from the CerberusStation tool. The cross signs are the exact location of the landmarks, the points with ellipses are the estimation of landmark locations with their covariance. The ellipses are drawn with the standard deviation values and rotated with the the correlation values. The smooth lines are the exact position of the robot and the dotted path is the estimated path. The robot started from the lower-left of the path. Note that landmark locations are rotated as compared to the experiment setup in Figure 5.1 because the origin of the coordinate frame is the starting point of the robot and $x$ axis lies on the robot direction.

In Figure 5.3, the pose error for each time step is given. The pose error in each time step is the mean squared error of the 2D location of the pose. As can be seen, the dead reckoning error grows in time and is unbounded. The decrease in the dead reckoning error occurs on the intersections after changing the direction. On the other hand, the error of the EKF-SLAM estimation with the same inputs is bounded and

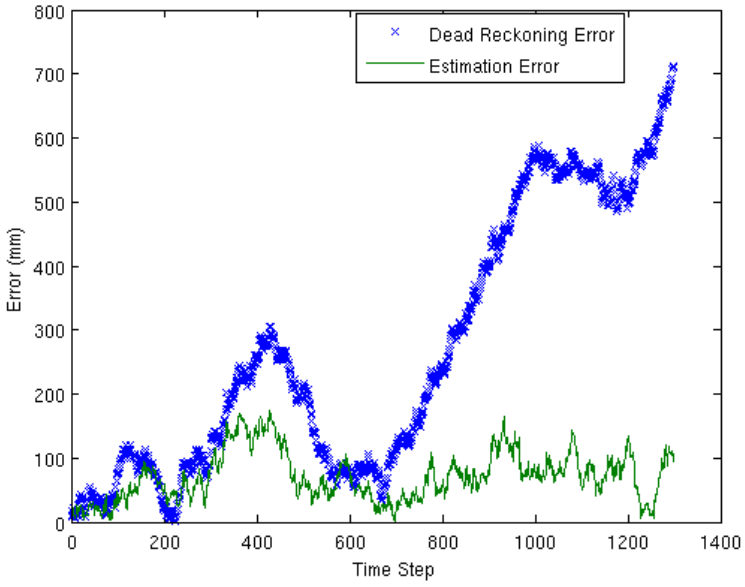



Figure 5.3. The comparison of pose errors for dead reckoning and EKF-SLAM.

can provide a good path estimate. As long as the algorithm observes a landmark, it corrects both the pose and the landmark position.

The same setup is applied for the Fast-SLAM algorithm. In Figure 5.4, the comparison of the dead reckoning pose error and the Fast-SLAM pose error is given. Like the EKF-SLAM Fast-SLAM can bound the odometric error. Note that another episode is applied for the Fast-SLAM algorithm and the injected noise differs from the previous run.

To obtain sufficient statistics, the experiment is repeated with the same configurations but with regenerated random variables five more times. Table 5.1 lists the results of errors in dead reckoning and the EKF-SLAM mothod in five different runs. The error is the mean of all pose errors over an episode. Figure 5.5 is the plot of the same set of experiment results.

Similar to EKF-SLAM, the experiments are repeated for the Fast-SLAM algorithm five times. Table 5.2 lists the errors for different experiments and Figure 5.6 plots the values in the list.

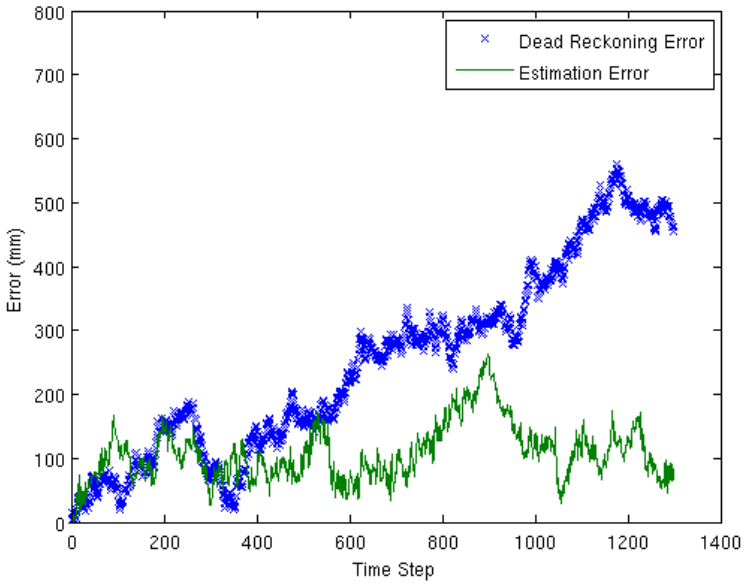



Figure 5.4. The comparison of pose errors for dead reckoning and Fast-SLAM.

Table 5.1. The mean pose error comparison of the episodes in repeated experiments.

| | EKF-SLAM | Dead Reckoning |
|---|---|---|
| Experiment 1 | 71.25 | 268.05 |
| Experiment 2 | 132.29 | 128.93 |
| Experiment 3 | 136.02 | 203.28 |
| Experiment 4 | 156.29 | 235.93 |
| Experiment 5 | 60.45 | 357.09 |

Table 5.2. The mean pose error comparison of the episodes in repeated experiments.

| | Fast-SLAM | Dead Reckoning |
|---|---|---|
| Experiment 1 | 130.91 | 170.03 |
| Experiment 2 | 99.02 | 142.52 |
| Experiment 3 | 147.83 | 277.99 |
| Experiment 4 | 108.11 | 249.9 |
| Experiment 5 | 134.19 | 347.9 |

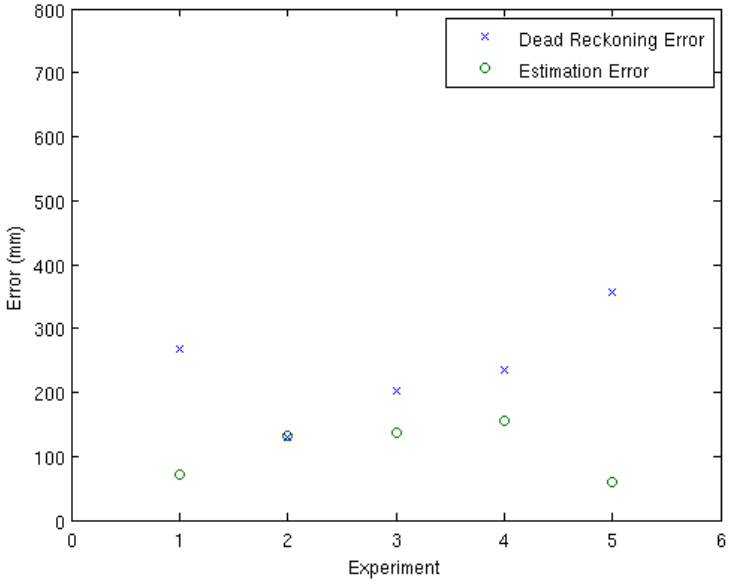



Figure 5.5. The mean pose error comparison of the episodes in repeated experiments.

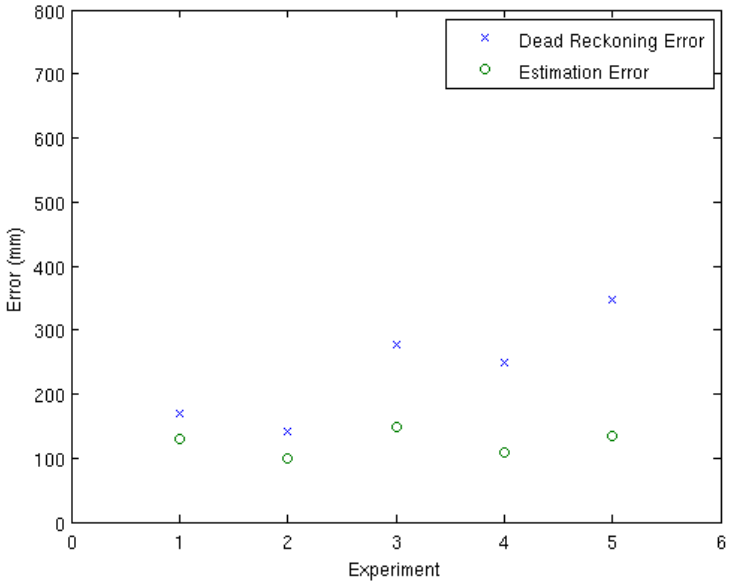



Figure 5.6. Results of pose errors of dead reckoning and Fast-SLAM in different experiments.

The results show that the odometric error is unbounded and the SLAM algorithms really improve the performance and can bound the odometric error if the noise distribution is Gaussian as assumed. Note that these results are not for comparing the EKF-SLAM and Fast-SLAM which is made in Section 5.1.3

### 5.1.2. Parameter Calibration in EKF-SLAM

One of the contributions of this Thesis is the calibration of EKF-SLAM parameters with Evolutionary Strategies (ES). The first set of experiments are designed for the inspection of the dependency of the parameter estimator algorithm on the training data. A complete episode in simulation with perfect knowledge is divided into four groups, each includes inputs and real position of a portion of the episode. The ES is trained with four different sets with the same parameters and the resulting parameter sets are tested on the other three sets. In other words, there are four different calibration results, trained with a small training set and three different validation sets.

Table 5.3. Results of calibrated parameters with four different training sets on different validation sets.

| | Set 1 | Set 2 | Set 3 | Set 4 |
|---|---|---|---|---|
| Training Result With Set 1 | 451.61 | 494.16 | 486.61 | 513.93 |
| Training Result With Set 2 | 1139.77 | 1090.36 | 1138.46 | 1111.17 |
| Training Result With Set 3 | 913.82 | 993.3 | 885.38 | 1017.07 |
| Training Result With Set 4 | 632.39 | 606.56 | 695.02 | 584.37 |

The error results of each trained parameters with each validation set is given in Table 5.3. The diagonal entries are the smallest values in their rows and columns. This results show us that our estimator has a variance for different sets of inputs. If we want the estimator to provide robust parameter sets, we should provide enough training samples to the estimator.

In the experiments above, none of the training sets include a loop in the environ-

ment; namely they have not encountered any previously seen landmark during their paths. In the actual calibration, we provided a long episode with at least two loops on the environment. The initial values for the search space is formed by tuning the parameters by hand and ES generated 100 generations with 100 individual. In addition, since we do not want the estimator to learn a specific case of noisy data, while performing the fitness value calculation episode for each individual, we regenerated the noise and injected to the inputs.

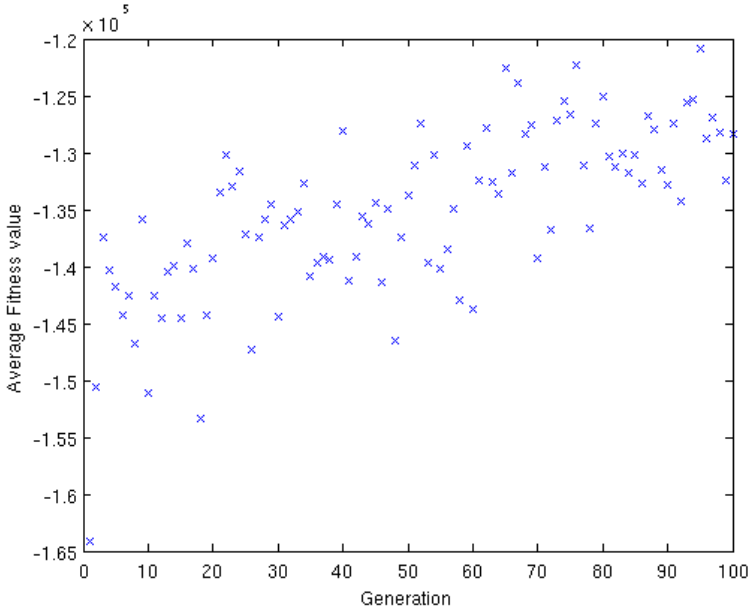
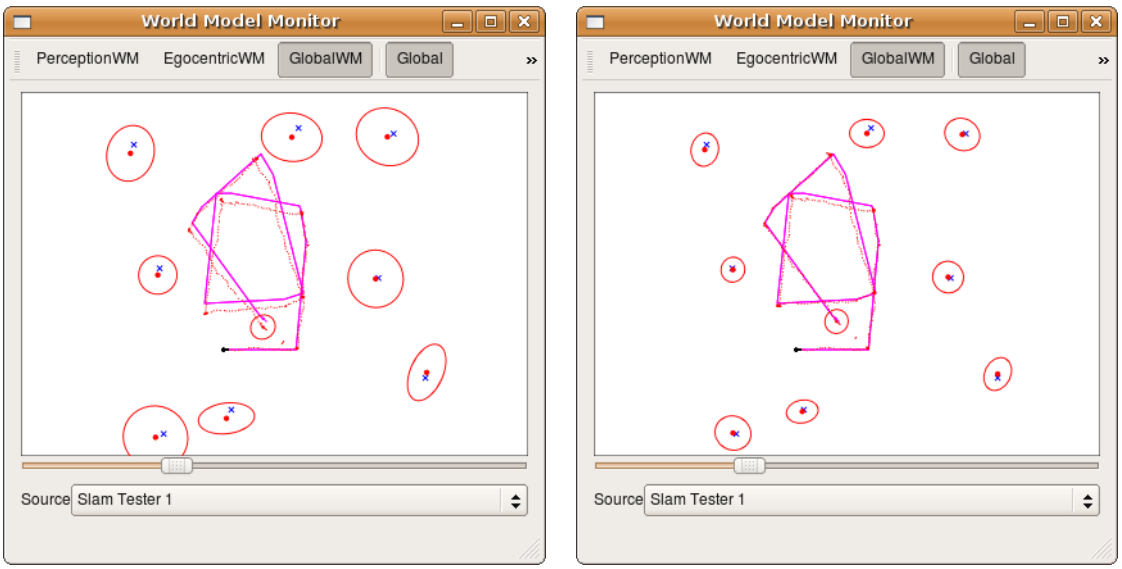



Figure 5.7. Average fitness values over generation number.

Figure 5.7 shows the average fitness values over generation. The values vary with generation due to changing noise in the input, however the overall fitness value increases over time.

In Figure 5.8, the paths and the map states in the end of the episode is given. The difference in the landmark covariances does not infer a comparison of the parameters, instead the accuracy of the landmark estimations and the path estimation shows that the calibrated parameters resulted with lower errors. Figure 5.9 shows the position error comparison during one episode. Since the algorithm is the same, the error pattern is similar in both runs, however calibrated parameters keep a lower bound in the pose error.




(a) Hand Tuned Parameters (b) Calibrated Parameters

Figure 5.8. Comparison of the hand tuned parameters and calibrated parameters in single episode.

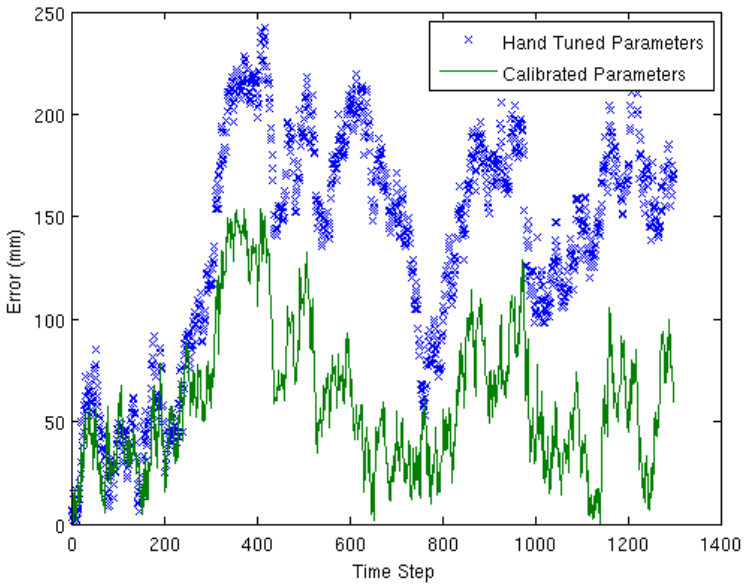



Figure 5.9. Comparison of pose errors for hand tuned and calibrated parameters in a single episode.

Another metric for the algorithm comparison is the map accuracy. The landmarks estimations change during an episode and unlike the pose errors, there can be jumps in the accuracy of a map estimate. When the robot completes a loop and re-observes a known landmark, the state of the landmark can change rapidly. Figure 5.10 shows such a situation. The plot is the mean squared error of one landmark. The robot first initializes the landmark near beginning of the execution. The straight segments indicates that landmark is not in the field of view, so it is not detected. Around time step 600, the robot re-observes the landmark and the error in estimation is corrected. The jump in the error may not always result with a decrease, the pose error might be higher in the second pass so that the landmark estimation error may increase, or a worse case is that the landmark may be declared as a new landmark because of large difference the observation and the estimation.

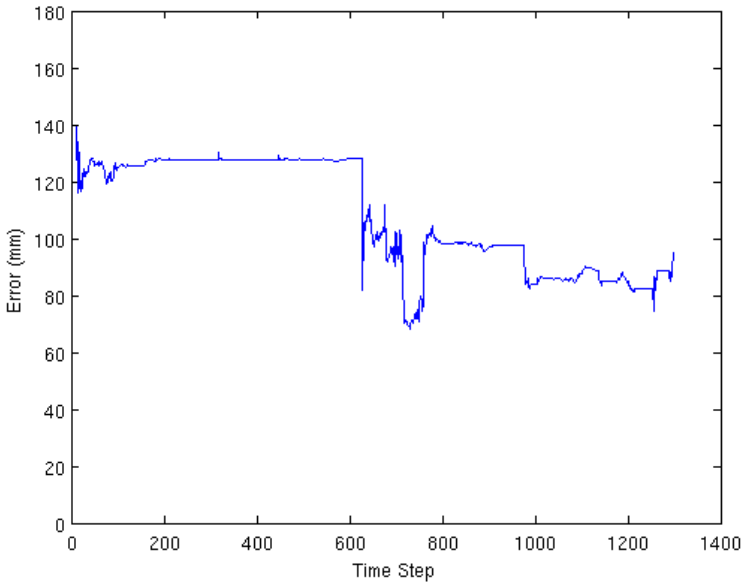



Figure 5.10. The MSE of a single landmark during an episode.

The map error metric for comparison of the SLAM algorithms is the mean of the MSEs of the known landmarks in a time step.In Figure 5.11, the map accuracy comparison of hand tuned parameters and the calibrated parameters are given. Just like the pose error comparison in Figure 5.9, the change pattern in the errors are similar due to same SLAM algorithm, however the calibrated parameters give better results.

The hand tuned and calibrated parameters are tested five times and the pose

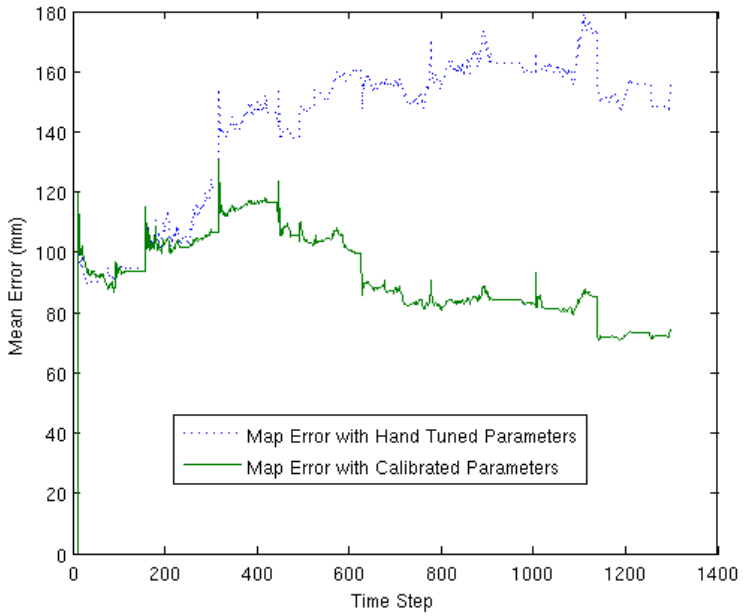
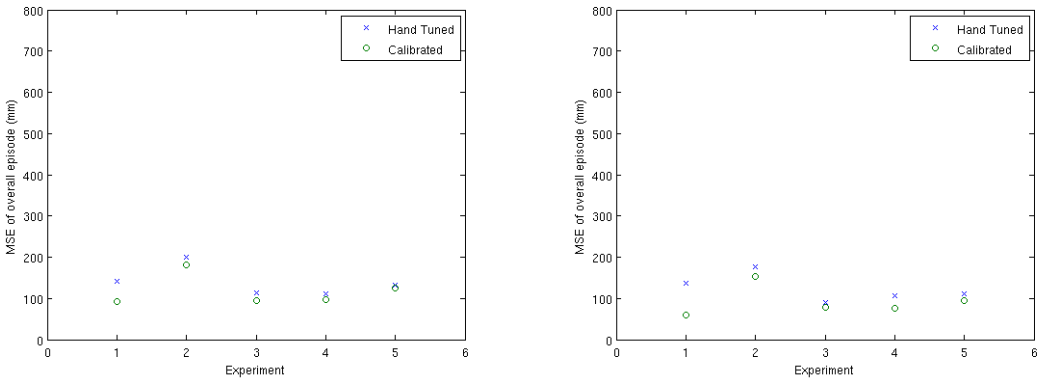



Figure 5.11. The overall map accuracy comparison of hand tuned parameters and calibrated parameters.

error and map accuracy comparisons are given in Figure 5.12. As can be seen, the calibrated parameters performed better than the hand tuned parameters.




(a) Pose Error (b) Map Error

Figure 5.12. Comparison of the hand tuned parameters and calibrated parameters.

### 5.1.3. Comparison of EKF-SLAM and Fast-SLAM

The EKF-SLAM and the Fast-SLAM both uses the Gaussian distributions on the observation and prediction models. So the calibrated parameters for the EKF-SLAM

are used in the Fast-SLAM algorithm. However, the Fast-SLAM has an important parameter, particle number, which affects the quality of the posterior distribution. As the number of particles increases, the error of the estimator decreases. The relation between the particle number and the performance is given in Figure 5.13. While comparing the Fast-SLAM with EKF-SLAM, we selected the particle number as 100 which gives reasonably low error and time complexity. The comparison of the path

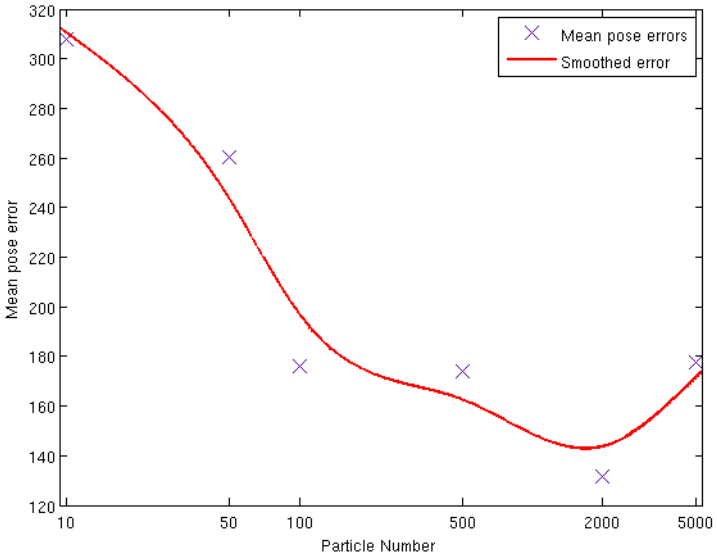
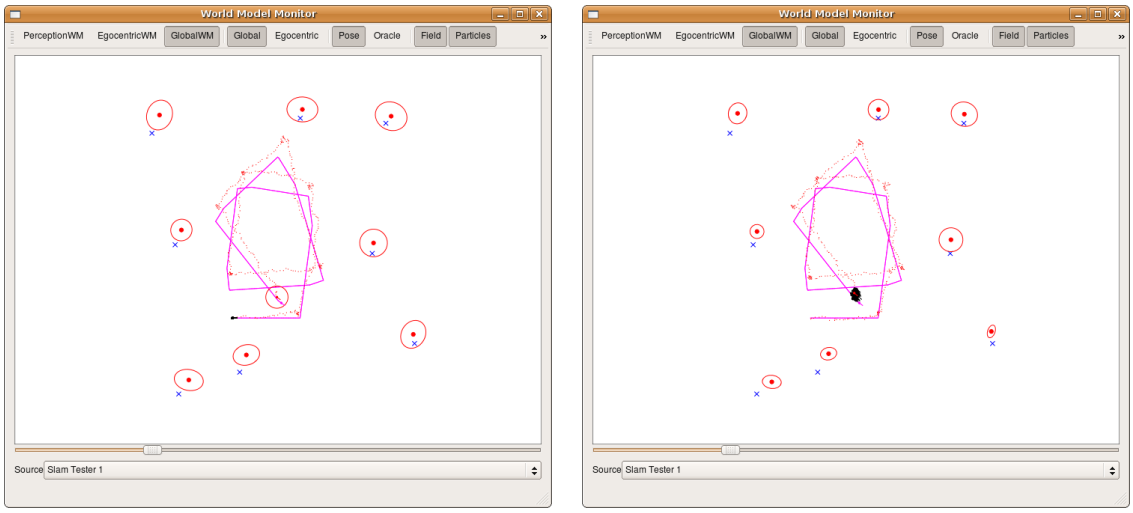



Figure 5.13. Relationship between the particle number and error in Fast-SLAM.

estimations of the algorithms with the same set of parameters and the same set of inputs is given in Figure 5.14. The dots in the end of the Fast-SLAM method represents particles of the pose belief, and the ellipse in the end of the path of the EKF-SLAM method represents the uncertainty covariance of the pose.

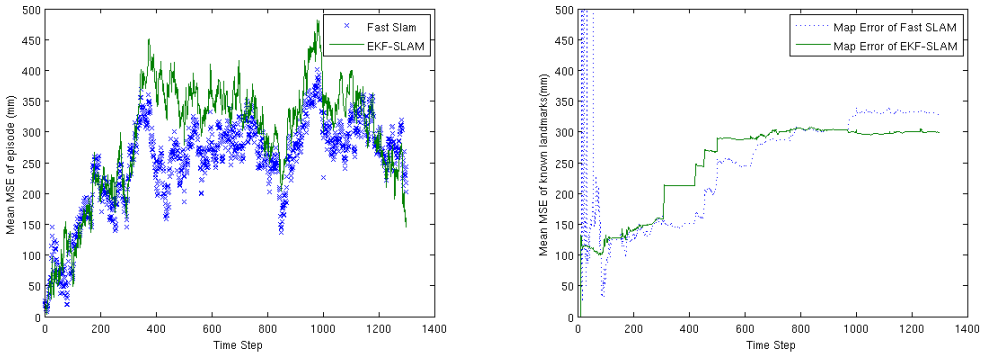
The injected noise parameters are increased in this experiment to observe the difference more clear. If we focus on the last state of the episode, (Figure 5.14), we observe that even though the same parameters are used, the uncertainty level of the landmarks in the Fast-SLAM algorithm is smaller than the EKF-SLAM algorithm. In other words, the ellipses for landmarks in the Fast-SLAM algorithm are smaller than the ellipses in EKF-SLAM algorithm. The reason of this is that the EKF-SLAM algorithm represents the augmented uncertainty with the robot pose, so the uncertainty of the pose is also utilized in the update steps. On the other hand, in the Fast-




(a) EKF-SLAM (b) Fast-SLAM

Figure 5.14. Path estimation comparison between EKF-SLAM and Fast-SLAM in a sample run.

SLAM algorithm, the uncertainty of the pose is represented with particles and the landmark uncertainties are uncorrelated. each particle updates the landmark state without uncertainty in the robot pose. The displayed landmark state in the Fast-SLAM method is the average of the landmark states of the particles as detailed in Section 4.3.3.1.

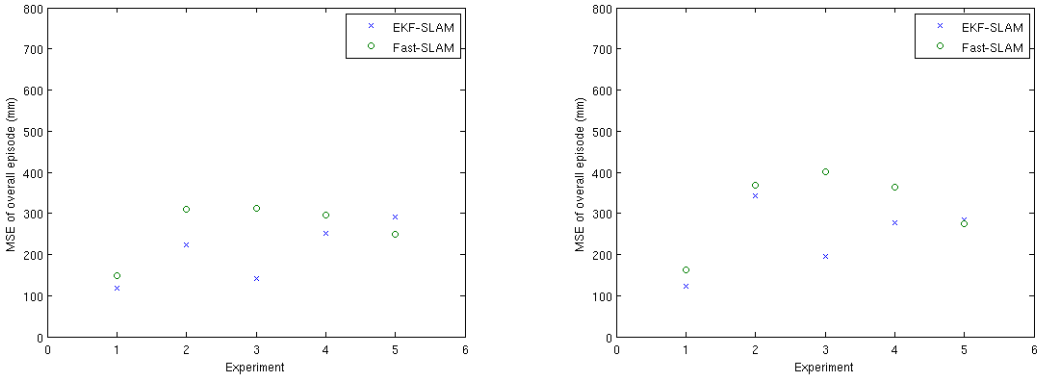



(a) Pose Errors (b) Map Errors

Figure 5.15. Comparison of the EKF-SLAM and Fast-SLAM errors in an episode.

The pose error and the map accuracy comparisons are given in Figure 5.15. The parameters of the algorithms are the same so they estimate the path with the same type of errors. However the map error of the Fast-SLAM in the beginning of the episode

peaks to large values. This is caused by our evaluation method. Since the SLAM algorithm decides the landmark identities itself, we cannot know the corresponding real landmarks. We can only assume that the order of the real observed landmarks is the same as the order of the landmarks in the map of the SLAM algorithm. The error calculations are made accordingly. This assumption generally holds and evaluates the algorithm. However, in the beginning of the Fast-SLAM algorithm, some particles may not obey the observation order of the landmarks. The large error becomes normal when inaccurate particles are eliminated.




(a) Pose Errors (b) Map Errors

Figure 5.16. Comparison of the EKF-SLAM and Fast-SLAM with different experiments.

The EKF-SLAM and Fast-SLAM comparison experiment is repeated five times and the overall result is given in Figure 5.16. As the experiments show, the EKF-SLAM performed better than Fast-SLAM in conditions with high Gaussian noise. The EKF-SLAM has more informative state representation and this result is intuitive.

### 5.1.4. Multi-Robot Experiment

The experiment setup of the multi-robot experiment is given in Figure 5.17. The robots starts from the position indicated in the figure, and follows a predefined rectangular path. After the fourth turn, the robots at a particular position can observe each other. At the points indicated in Figure 5.17, the robots exchange maps and explore the map of each other. Like the single agent experiments, Gaussian noise is

injected into all of the odometry readings and observations. The experiment is applied to both EKF-SLAM and Fast-SLAM algorithms.

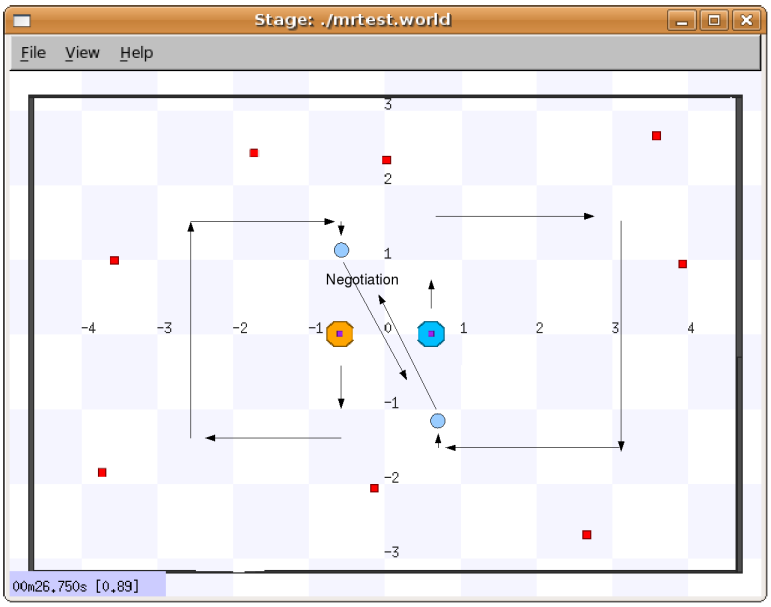
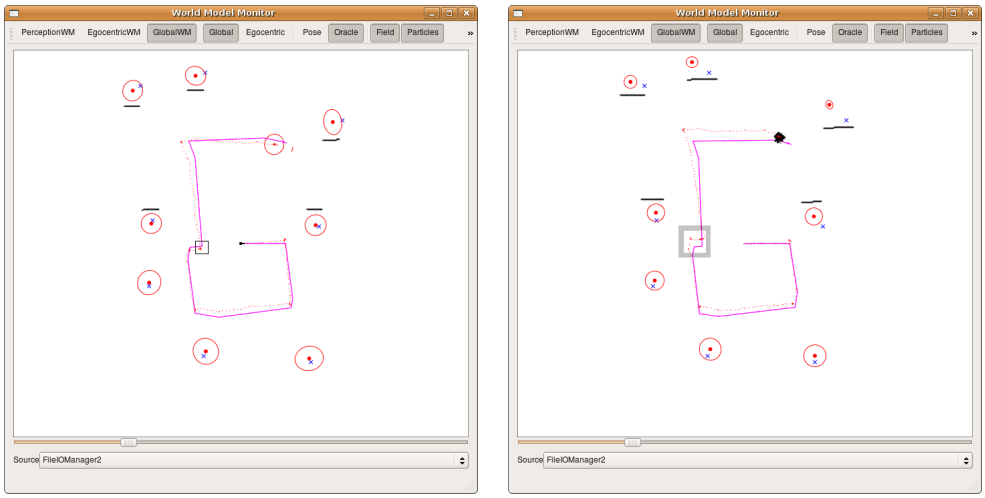



Figure 5.17. Experimental setup of the multi robot experiment.

In Figure 5.18, the results of the path estimations and final state of the map states are given for both EKF-SLAM and Fast-SLAM algorithms.

The position marked with the square in the path of the robot points the negotiation and map merging location. The underlined landmarks are added into the robot's map and the overlined landmarks are merged with the landmarks from other robot because they were already in the map of both robots. Note that the two figures are the results from the same robot with different algorithms. The displayed results are taken from the left hand side robot in Figure 5.17.

One notable result in Figure 5.18(b) is that the self explored landmarks are precise, however the newly added landmarks have a biased error. This error is caused by the observation and pose estimation errors in the moment of map merging. Recall that the transformation parameters are only dependent on pose estimations and robot observations. Another observation in the figure is that the error in the merged landmarks does not increase as the newly added landmarks, because the robot considers

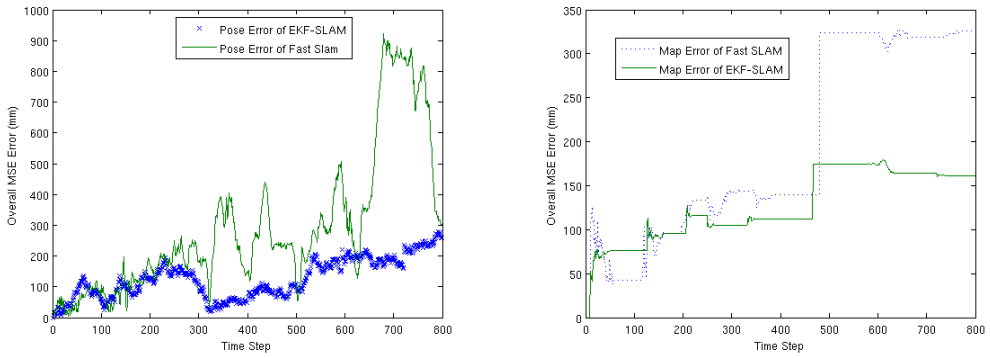



(a) EKF-SLAM (b) Fast-SLAM

Figure 5.18. Path and map estimation results in the multi robot experiment.

the common landmarks as a new evidence.

Unlike the single robot experiments, there are two robots on the environment and they should communicate at the same time. For this reason this algorithm should be applied fully online. The pose estimation errors and map estimation errors are reported for the algorithms in Figure 5.19, but recall that these algorithms are executed with different inputs and different noise values.




(a) Pose Errors (b) Map Errors

Figure 5.19. Pose and map errors for EKF-SLAM and Fast-SLAM methods in the multi-robot experiment.

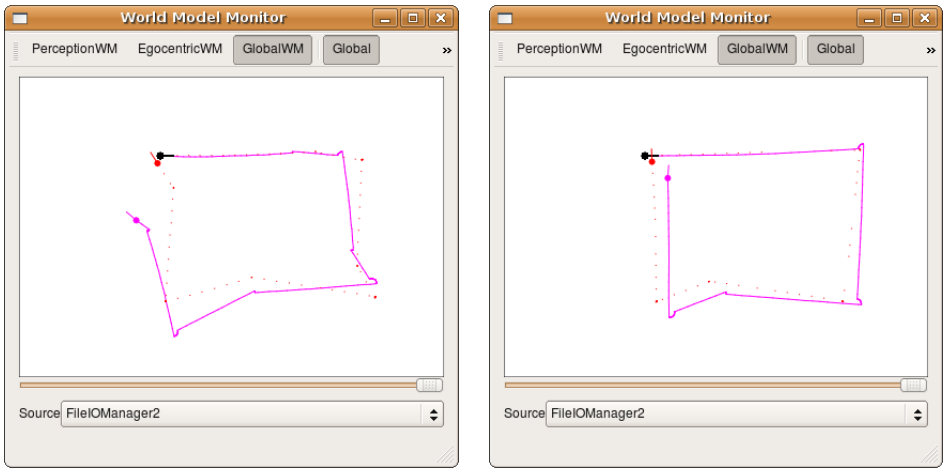
The most notable difference is the change in the map estimation error after the

map merging. As easily observed from Figure 5.18, the newly added landmarks have a bias. This bias increases the map estimation error after map merging.

## 5.2. Real World Experiments

### 5.2.1. Odometry Calibration Results

The odometry calibration procedure is applied to the real robots as described in Section 4.2. The three types of motion commands; forward, left and turn applied several times and the path is recorded with the overhead camera system. Since the overhead camera has its own coordinate system, the readings of the camera are transformed to the coordinate system of the robot. Each motion command is applied five times and the required odometry correction for all runs are recorded. The final odometry correction values are set as the mean of five runs. Figure 5.20 shows the comparison of the dead reckoning path estimations with both uncorrected odometry and corrected odometry.




(a) Uncorrected Odometry (b) Corrected Odometry

Figure 5.20. The comparison between uncorrected odometry and the corrected odometry readings.

The straight line shows the readings of the overhead camera and the dots represent the dead reckoning path. The differences between the followed paths are caused by the planning algorithm. The robot is moved on a predefined square, but the arrival to

the corners constraint is relaxed with both distance and orientation thresholds. So the robot corrects the direction as long as it is heading a wrong direction. However, we are actually concerned with odometric measurement quality and the main observation in the results in Figure 5.18 is the difference between the resulting dead reckoning position and the actual position.

### 5.2.2. Single-Agent EKF-SLAM

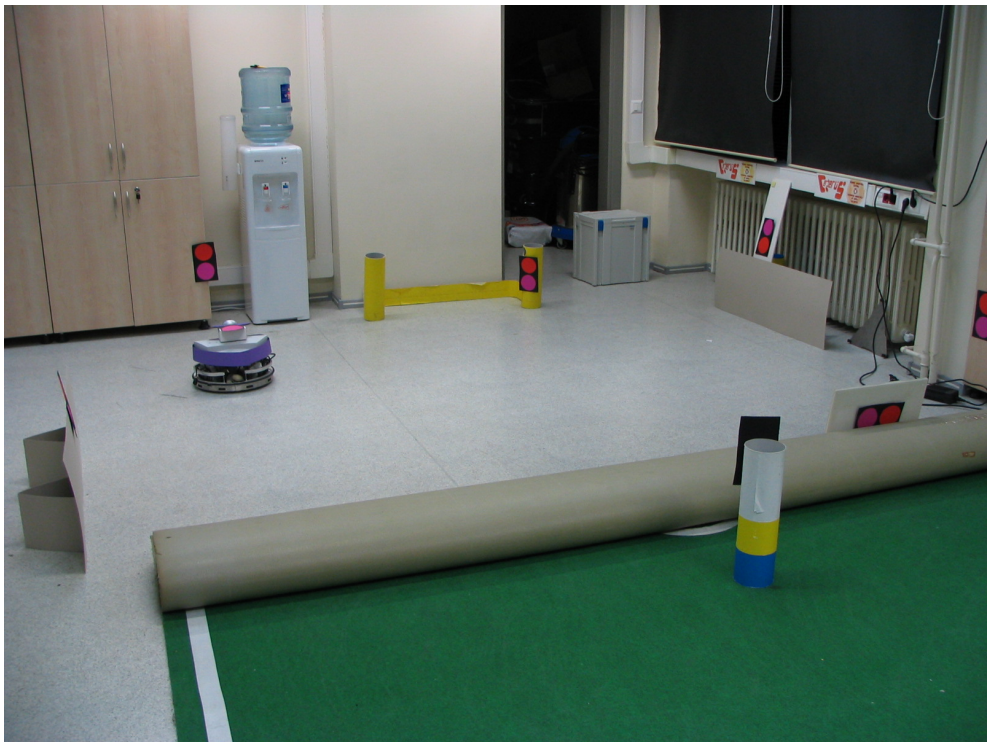
The EKF-SLAM algorithm is applied to the single real robot in the indoor environment. The experiment setup is given in Figure 5.21. The robot is standing in the starting position.

In this experiment, the autonomous planning algorithm as detailed in Section 4.6 has controlled the robot. The robot tried to avoid the obstacles with as few turns as possible and map the marked landmarks along with the occupancy grid mapping. In this experiment, the overhead camera did not cover the environment, so it did not provide actual position of the robot.

In Figure 5.22, the obstacle avoidance algorithm is demonstrated in the experiment. The color of the area represents the probabilities of the occupancy of the corresponding grid. The darker a grid is, higher the probability of being an obstacle. The line in front of the robot is the line formed by the odometry readings. Note that the line is not formed by the grid probabilities in the map, instead raw laser readings are used.

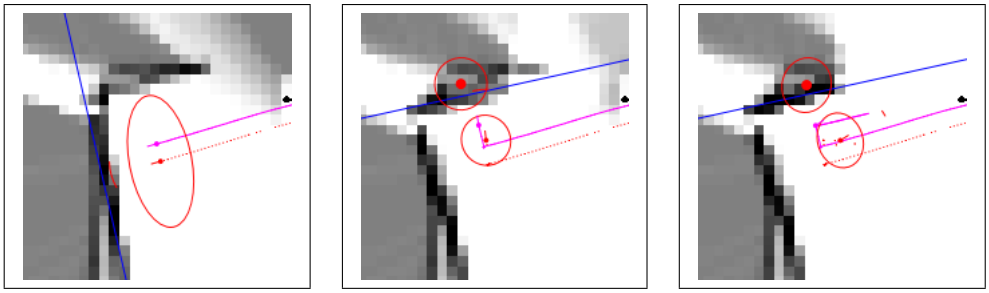
In Figure 5.23, the final state of the robot is given. In this Figure, since there is not the supervisor information, the continuous line is the dead reckoning path and the dots are the estimated path. On the execution, the robot have managed to correct its odometric errors and provided with an accurate map. While the dead reckoning pose error grows excessively with systematic errors, the EKF-SLAM algorithm did not take any false decisions on observation correspondences.

(a) Experiment setup



(b) Experiment setup from different perspective

Figure 5.21. The experiment setup on the real world.



(a) Obstacle observed (b) Turned 90 degrees (c) Turn again to avoid

Figure 5.22. The obstacle avoidance and area coverage algorithm.

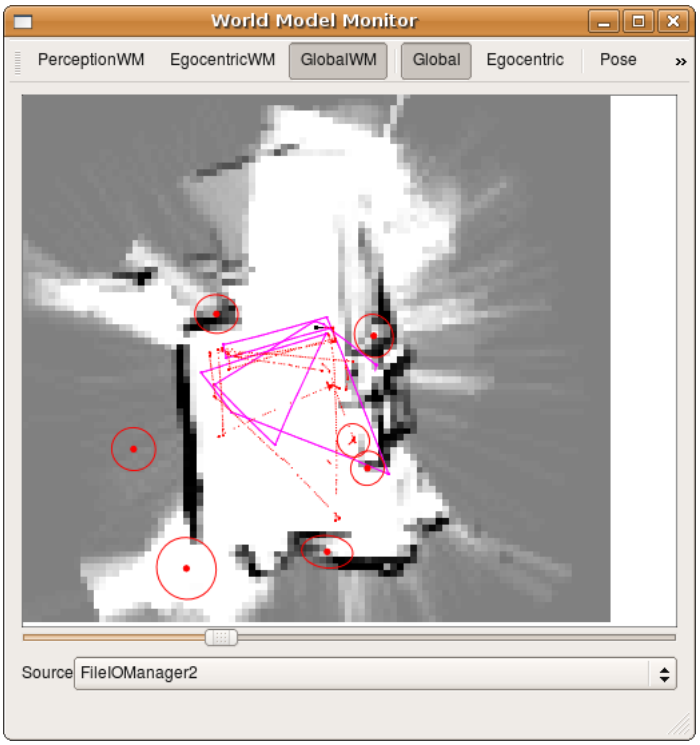
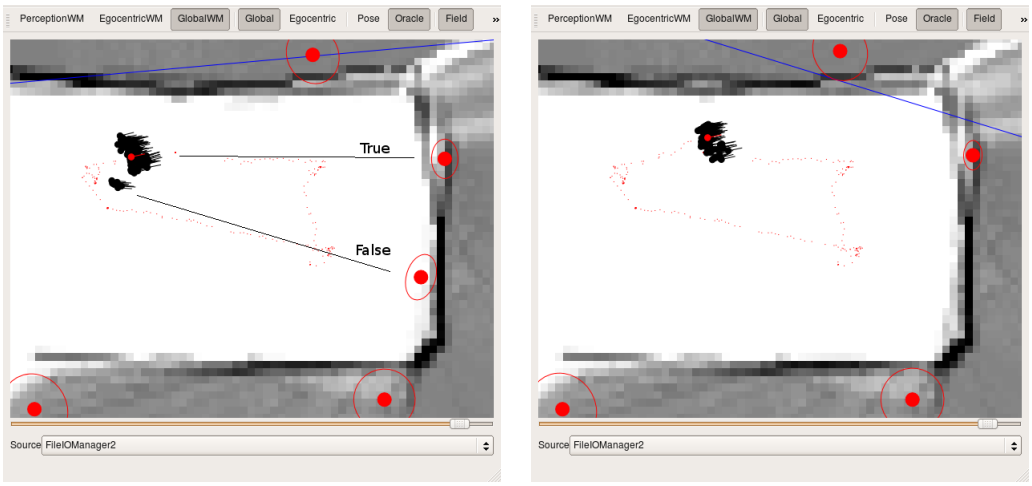



Figure 5.23. The state of the robot at the end of the episode.

### 5.2.3. Single-Agent EKF-SLAM

The Fast-SLAM experiment in the real world shows some characteristic features of the Monte-Carlo based approach. As mentioned in Section 4.3.3, each particle in the belief state has its own map estimation and this estimation may differ from each other. However in the correction steps, the particles with false estimations are eliminated. In Figure 5.24, two snapshots of an episode is given. In the first snapshot, some of




(a) Wrong landmark decision (b) Wrong decisions are eliminated

Figure 5.24. On the left snapshot, some particles has wrong decisions but they are eliminated after a few time steps.

the particles initialized a newlandmark, which is already in the map estimate, but observed with a high offset from the previously known location due to odometric errors and particle diffusion. However after a few time steps, the particles with wrong extra landmark in their maps are eliminated and given in the second snapshot. We see the false landmarks in the map estimate of the Fast-SLAM because the maps of each particle are merged into a single map and all the distinct landmarks are added.

### 5.2.4. Map-Merging Results

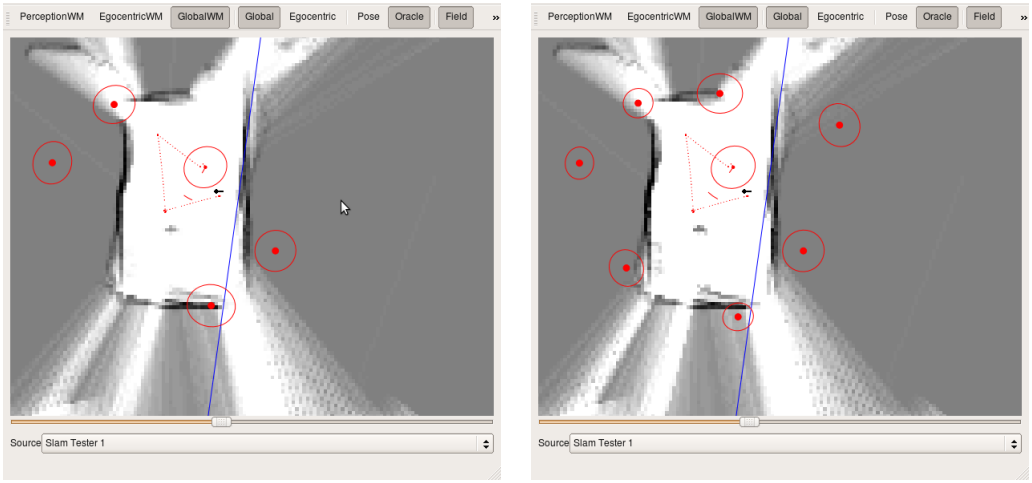
The real world experiment environment is given in Figure 5.25. The robots starts as back to back and since the obstacle avoidance behavior always turns to the same

side, the robots explore seperate regions and eventually one observes the other and the maps are merged after negotiation.

Figure 5.25. Experiment setup of the map-merging experiment.

In Figure 5.26, the results for the EKF-SLAM algorithm are given for both of the robots. Note that the origins of the coordinate frames of robots are the initial pose. The snapshots on the left column are the belief states just before the merging operation, where the robots are faced to each other. The other robot also affects the grid map and can be observed in the snapshots. Before the merging operation, the maps of the robots also contain estimations for same landmarks, however none of the robots know all of the landmarks. The snapshots on the right column are the maps after the map-merging operation. After this, both robots know all the landmarks in the environment.

In Figure 5.27, the results for the Fast-SLAM algorithm is given for both of the robots. The snapshots on the left column are the belief states just before the merging operation. Before the merging operation, the maps of the robots also contains estimations for same landmarks, however none of the robots know all of the landmarks. The snapshots on the right column are the maps after the map-merging operation. After this, both robots know all the landmarks just like in the EKF-SLAM approach.



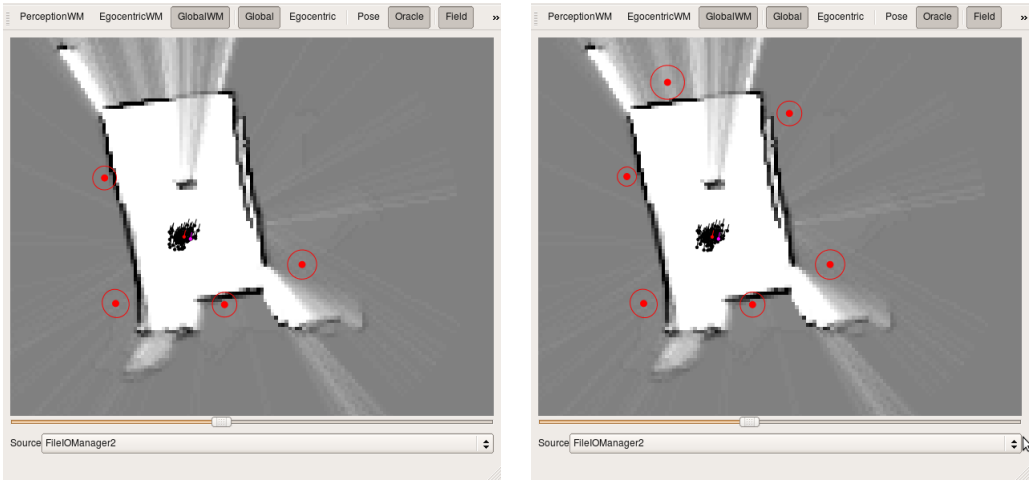

(a) State of robot 1 before merging

(b) State of robot 1 after merging

(c) State of robot 2 before merging

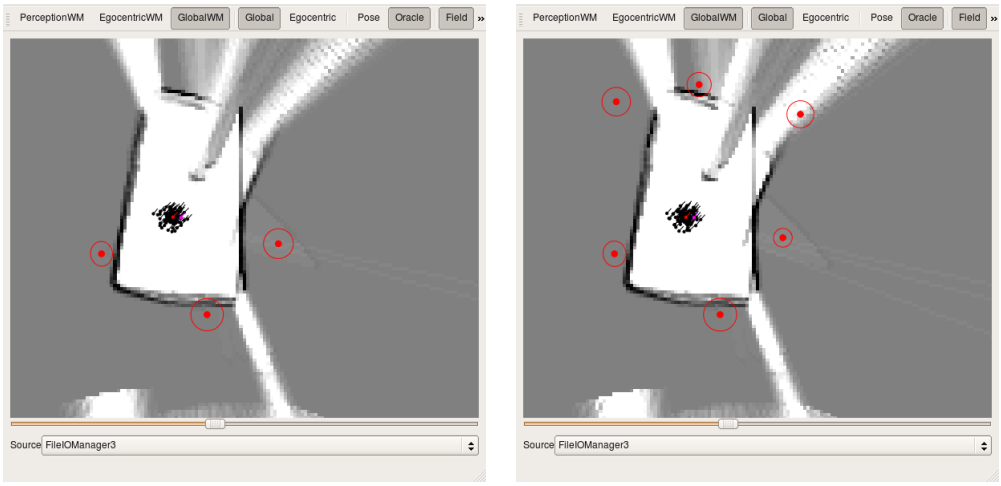
(d) State of robot 2 after merging

Figure 5.26. Map-merging result in the real world with EKF-SLAM method.




(a) State of robot 1 before merging

(b) State of robot 1 after merging




(c) State of robot 2 before merging

(d) State of robot 2 after merging

Figure 5.27. Map-merging result in the real world with Fast-SLAM method.

# 6. CONCLUSIONS

In this thesis, we proposed the application of the SLAM solution techniques to the distributed multi-robot domain. The SLAM problem may be in many forms and may require different high level tasks. In this work, we limit the problem as creating a map of an indoor structured environment. We provided visual landmarks, which violate the "complete unknown environment" assumption, but we still provided non-unique landmarks, which is the case for most of the SLAM applications. We developed and applied the algorithms for the two robot case. More than two robot case is the more interesting case but brings several more challenges like complex correspondence problems. Instead, we focused on map-merging between robots which do not know the initial positions of each other. For the single agent case, EKF-SLAM and Fast-SLAM algorithms are implemented.

We developed the robots as fully autonomous. For this purpose, we developed simple obstacle avoidance and area coverage algorithms. The multi-robot application also requires online execution of the algorithm, so our algorithms are designed to work online. By using our logging and monitoring tools, we can compare the two algorithms with the same set of inputs on the single robot case.

On the real robots, the systematic errors of the odometry values are calibrated and corrected. We also generated a grid map of the environment along with a landmark based map. However, we only used landmark based map in the SLAM algorithms, which is a more general case for SLAM problems.

On the methods, we have two notable contributions to the literature. First, we calibrated the noise parameters of EKF-SLAM algorithm using Evolutionary Strategies. The other contribution is that we adapted the map merging method for EKF-SLAM method from the literature to the Fast-SLAM method. The main difficulty in this adaptation is the representation of multiple maps in the Fast-SLAM method. To overcome this, we extracted a single map estimation from the Fast-SLAM belief state

along with the uncertainty and merged this map with each map of the other particle based belief state. This method also allows execution of different algorithms on different robots. The exchanged map estimation between robots has a common format, so the estimator of the incoming map estimation is not important for the merger algorithms.

On the testings, we used simulator and real robots. The simulator provided perfect knowledge and exact locations of every object. But in testings, we injected noise to the perfect knowledge to better evaluate algorithms. In the real environment, the exact locations of the robots are provided by the overhead camera system.

With the test results, we have shown the necessity of a SLAM method to avoid unbounded odometric errors during navigation. We also tested the parameter calibration procedure with different training sets and showed that the calibration procedure may have variance for different training sets. However we showed that when enough training samples are provided, the calibration procedure can give better results than hand-tuned parameters.

The test results also showed that EKF-SLAM gave slightly better results than the Fast-SLAM method when the noise is distributed as Gaussian. The reason is the state representation of the EKF-SLAM method with all the correlation values. The results also showed that adaptation of map merging from EKF-SLAM to Fast-SLAM works as expected.

## 6.1. Future Work

The SLAM problem has a broad range of application areas and each may have its own conditions, limitations or context knowledge. The method should be selected specific to that task and conditions. For example, in an outdoor environment, a laser range finder would return unreliable readings and odometry values might be very noisy. Instead of a laser range finder, camera images would result with more reliable landmarks and vision-based ego-motion method may be used as odometry calculations.

Whatever the application type, the SLAM problem characteristic is always same; fully autonomous system and completely unknown environment.

The major obstacle between this work and a real life SLAM application is the artificial landmarks. We should not provide artificial landmarks to the robot and extract features from images. Alternatively, laser readings should be used for extracting features in structured environments.

Another shortcoming of the method is the assumption of two robots. It would be more interesting when there are more than two robots. This problem requires resolving correspondence issues between robot observations.

Another improvement can be using a hybrid map representation of location based map and landmark based map. While landmarks provide position corrections, location based map provides human interpretable maps.

# REFERENCES


1. Montemerlo, M., "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association", *CMU Robotics Institute*, 2003.

2. Thrun, S., W. Burgard, and D. Fox, *Probabilistic Robotics*, MIT Press, Cambridge, MA, 2005.

3. Welch, G. and G. Bishop, "An Introduction to the Kalman Filter", Technical report, Chapel Hill, NC, USA, 1995.

4. Wang, F. and V. Balakrishnan, "Robust Adaptive Kalman Filters for Linear Time-Varying Systems with Stochastic Parametric Uncertainties", 1999.

5. Martinelli, A., N. Tomatis, and R. Siegwart, "Simultaneous Localization and Odometry Self Calibration for Mobile Robot", *Auton. Robots*, Vol. 22, No. 1, pp. 75–85, 2007.

6. "Cerberus Robocup 2005 Home", 2008, `http://robot.cmpe.boun.edu.tr/aibo/home.php3`.

7. Oliver, S. C., "The NUbots' Team Description for 2005", 2005, `citeseer.ist.psu.edu/675149.html`.

8. "Robocup 4-Legged League", 2008, `http://www.tzi.de/4legged/`.

9. Wan, E. and R. van der Merwe, "The Unscented Kalman Filter for Nonlinear Estimation", In Proc. of IEEE Symposium (AS-SPCC), Lake Louise, Alberta, Canada, Oct., 2000.

10. Rekleitis, I. M., "A Particle Filter Tutorial for Mobile Robot Localization", Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, 3480 University St., Montreal, Québec, CANADA H3A 2A7, 2004.

11. Doucet, A., N. de Freitas, K. Murphy, and S. Russell, "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks", *UAI*, pp. 176–183, 2000.

12. Khan, Z., T. R. Balch, and F. Dellaert, "A Rao-Blackwellized Particle Filter for EigenTracking", *CVPR (2)*, pp. 980–986, 2004.

13. Thrun, S., D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots", *Artificial Intelligence*, Vol. 128, No. 1-2, pp. 99–141, 2001.

14. Köse, H. and H. L. Akın, "The Reverse Monte Carlo Localization Algorithm", *Robot. Auton. Syst.*, Vol. 55, No. 6, pp. 480–489, 2007.

15. Köse, H. and H. L. Akın, "A Fuzzy Touch to R-MCL Localization Algorithm.", Bredenfeld, A., A. Jacoff, I. Noda, and Y. Takahashi (editors), *RoboCup*, Vol. 4020 of *Lecture Notes in Computer Science*, pp. 420–427, Springer, 2005.

16. Fox, D., W. Burgard, H. Kruppa, and S. Thrun, "A Probabilistic Approach to Collaborative Multi-Robot Localization", *Auton. Robots*, Vol. 8, No. 3, pp. 325–344, 2000.

17. Smith, R., M. Self, and P. Cheeseman, "Estimating Uncertain Spatial Relationships in Robotics", *Autonomous Robot Vehicles*, pp. 167–193, 1990.

18. Montemerlo, M., S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges", Gottlob, G. and T. Walsh (editors), *IJCIA*, pp. 1151–1156, Morgan Kaufmann, 2003.

19. Kwok, N. M., G. Dissanayake, and Q. P. Ha, "Bearing-only SLAM Using a SPRT Based Gaussian Sum Filter.", *ICRA*, pp. 1109–1114, IEEE, 2005.

20. Lemaire, T., S. Lacroix, and J. Solá, "A Practical 3D Bearing Only SLAM Algorithm", August 2005.

21. Davison, A. J., "Real-time simultaneous localisation and mapping with a single camera", *International Conference on Computer Vision*, pp. 1403–1410, 2003, http://dx.doi.org/10.1109/ICCV.2003.1238654.

22. Estrada, C., J. Neira, and J. D. Tardós, "Hierarchical SLAM: real-time accurate mapping of large environments", *IEEE Transactions on Robotics*, Vol. 21, No. 4, pp. 588–596, December 2005.

23. Howard, A., "Multi-robot Mapping using Manifold Representations", *ICRA*, pp. 4198–4203, IEEE, 2004.

24. Pfingsthorn, M., B. Slamet, and A. Visser, "A Scalable Hybrid Multi-Robot SLAM Method for Highly Detailed Maps", Proceedings CD of the 11th RoboCup International Symposium, 2007.

25. Montemerlo, M. and S. Thrun, "Simultaneous Localization and Mapping With Unknown Data Association Using FastSLAM", *ICRA*, pp. 1985–1991, IEEE, 2003.

26. Lemaire, T., C. Berger, I.-K. Jung, and S. Lacroix, "Vision-Based SLAM: Stereo and Monocular Approaches", *Int. J. Comput. Vision*, Vol. 74, No. 3, pp. 343–364, 2007.

27. Thrun, S., Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. F. Durrant-Whyte, "Simultaneous Localization and Mapping with Sparse Extended Information Filters", *I. J. Robotic Res*, Vol. 23, No. 7-8, pp. 693–716, 2004, http://ijr.sagepub.com/cgi/content/abstract/23/7-8/693.

28. Casella, G. and C. P. Robert, "Rao-Blackwellisation of Sampling Schemes", *Biometrika*, Vol. 83, No. 1, pp. 81–94, March 1996, http://dx.doi.org/10.1093/biomet/83.1.81.

29. Royer, E., M. Lhuillier, M. Dhome, and J.-M. Lavest, "Monocular Vision for Mobile Robot Localization and Autonomous Navigation", *Int. J. Comput. Vision*, Vol. 74,

No. 3, pp. 237–260, 2007.

30. Sim, R., P. Elinas, and J. J. Little, "A Study of the Rao-Blackwellised Particle Filter for Efficient and Accurate Vision-Based SLAM", *Int. J. Comput. Vision*, Vol. 74, No. 3, pp. 303–318, 2007.

31. Karlsson, N., E. D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. E. Munich, "The vSLAM algorithm for robust localization and mapping", *2005 IEEE International Conf. on Robotics and Automation, ICRA 2005*, 2005.

32. Lowe, D. G., "Distinctive Image Features from Scale-Invariant Keypoints", *Int. J. Comput. Vision*, Vol. 60, No. 2, pp. 91–110, 2004.

33. Zhou, X. and S. Roumeliotis, "Multi-robot SLAM with Unknown Initial Correspondence: The Robot Rendezvous Case", *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 1785–1792, Oct. 2006.

34. Thrun, S. and Y. Liu, "Multi-robot SLAM with Sparse Extended Information Filers", *I. J. Robotic Res*, Vol. 15, pp. 254–266, 2005, `http://www.springerlink.com/content/255jw4lnhhwq6k1l/`.

35. Konolige, K., D. Fox, B. Limketkai, J. Ko, and B. Stewart, "Map merging for distributed robot navigation", *Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, Vol. 1, pp. 212–217 vol.1, Oct. 2003.

36. Fox, D., K. Konolige, B. Limketkai, J. Ko, D. Schulz, and B. Stewart, "Distributed Multi-Robot Exploration and Mapping", *Computer and Robot Vision, 2005. Proceedings. The 2nd Canadian Conference on*, pp. xv–xv, May 2005.

37. Rekleitis, I., D. Meger, and G. Dudek, "Simultaneous Planning, Localization, and Mapping in a Camera Sensor Network", *Robotics and Autonomous Systems*, Vol. 54, No. 11, pp. 921–932, November 2006.

38. "Festo Robotino", 2008, http://www.festo-didactic.com/int-en/learning-systems/education-and-research-robots-robotino/.

39. "URG Laser Range Finder", 2008, http://www.hokuyo-aut.jp/02sensor/07scanner/urg.html.

40. Kavaklıoğlu, C., *Developing A Probabilistic Post Perception Module For Mobile Robotics*, Master's thesis, Boğaziçi University, Turkey, 2009.

41. Gerkey, B., R. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems", *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, jun 2003, `citeseer.ist.psu.edu/gerkey03playerstage.html`.

42. "Robotics Group of Bogazici University", 2008, `http://robot.cmpe.boun.edu.tr/`.

43. Alpaydın, E., *Machine Learning*, MIT Press, 2004.

44. Borenstein, J., "Correction of systematic odometry errors in mobile robots", *IROS '95: Proceedings of the International Conference on Intelligent Robots and Systems-Volume 3*, p. 3569, IEEE Computer Society, Washington, DC, USA, 1995.

45. Beyer, H.-G., *The Theory of Evolution Strategies*, Springer-Verlag, Heidelberg Berlin, 2001.

46. Stroupe, A., M. Martin, and T. Balch, "Distributed sensor fusion for object position estimation by multi-robot systems", *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, Vol. 2, pp. 1092–1098 vol.2, 2001.

47. Tolun, L., *Development of Area Coverage Algorithms For an Autonomus Cleaner Robot*, Master's thesis, Boğaziçi University, Turkey, 2000.

48. Fischler, M. A. and R. C. Bolles, "Random Sample Consensus: A Paradigm for

Model Fitting With Applications to Image Analysis and Automated Cartography",
*Commun. ACM*, Vol. 24, No. 6, pp. 381–395, 1981.