S-LOC AND MY ENVIRONMENT: A NEW LOCALIZATION SYSTEM FOR AUTONOMOUS ROBOTS

by

Buluç Çelik

B.S. in Computer Engineering, Eastern Mediterranean University, 2003

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

2005

ACKNOWLEDGEMENTS

First of all, I would like to thank to my supervisor Prof. H. Levent Akın for his encouragement, guidance and enthusiastic support. This thesis would not be possible without his contributions.

I would like to thank to the people of AILab and Cerberus Team, who has taken place in every single moment of this work with their valuable friendship, discussions, brilliant ideas and support. Their friendship and professional collaboration meant a great deal to me.

My appreciations go to my sincere jury members Prof. Cem Say and Prof. Okyay Kaynak.

I would like to thank all my teachers, who have shared their knowledge and ideas with me over the years, and enlightened my path.

I would like to express my gratitude to my family for their endless love and support as well as helping me to form my personality. I owe them everything.

In addition, I am also grateful to my real friends for their morale support, friendship and remarkable discussions about the life, universe and everything.

ABSTRACT

S-LOC AND MY ENVIRONMENT: A NEW LOCALIZATION SYSTEM FOR AUTONOMOUS ROBOTS

The localization problem is the detection of the pose of a robot relative to the environment using the information about the environment sensed by the robot when the starting position is unknown. In short, it is answering the question "Where am I?". Localization is an active field of study where many approaches are introduced into the literature.

Robot soccer is a good platform to develop and test localization techniques since the robots have limited and noisy sensorial information as in the real life and the environment is also highly dynamic.

In this work, a new module that will stand between the perception module and the other modules that uses its output and a new localization technique are introduced, and they are together proposed as a new localization system.

The proposed new module is My Environment, which stores the perceptional data and provides a filtered and more robust data, and the new localization technique is S-Loc, which is a sample based localization technique where only one sample is used for each perception data.

This system is implemented in Cerberus'05, which won the Technical Challenges in the RoboCup 2005 - Sony Four-Legged League. This success, together with the experimental study, has shown that the proposed solution has a high performance for the application domain.

ÖZET

S-LOC VE MY ENVIRONMENT: OTONOM ROBOTLAR İÇİN YENİ BİR LOKALİZASYON SİSTEMİ

Lokalizasyon problemi, başlangıç konumunun bilinmediği durumlarda çevreden algınan bilgileri kullanarak bir robotun çevresine göre konumunu belirlemesidir. Kısaca, "Neredeyim?" sorusunu yanıtlamaktır. Lokalizasyon, birçok yöntemin literatüre dahil edildiği aktif bir çalışma alanıdır.

Robotların gerçek hayattaki gibi sınırlı ve gürültülü algısal bilgilere sahip olmalarından ve çevrelerinin oldukça hareketli olmasından dolayı, robot futbolu lokalizasyon tekniklerinin geliştirilmesi ve denenmesi için iyi bir ortamdır.

Bu çalışmada, yeni bir lokalizasyon tekniği ve algılama modülüyle algılama modülünün çıkışını kullanan öteki modüller arasında bulunacak yeni bir modül oluşturulmuş, bunların ikisi birlikte yeni bir lokalizasyon sistemi olarak önerilmiştir.

Önerilen yeni modül, My Environment, algısal verileri depolar ve filtrelenmiş, daha sağlıklı veriler sağlar. Yeni lokalizasyon tekniği olan S-Loc ise her algısal veri için sadece bir örneğin kullanıldığı örnek tabanlı bir lokalizasyon tekniğidir.

Bu sistem, RoboCup 2005 - Sony Dört-Ayaklılar Ligi'nin Teknik Yarışmalar kategorisini kazanan Cerberus'05 bünyesinde gerçekleştirilmiştir. Bu başarı, deneysel çalışmalarla birlikte önerilen çözümün uygulama alanı için yüksek bir performansa sahip olduğunu göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

a	The odometry data in ML
AA_i^j	The relative angle of the i^{th} record of the j^{th} object in ME
$Bel(l)$	The belief that the robot is at the location l in ML
c_j	The confidence estimation of the j^{th} object in ME
CA_i^j	The confidence of the i^{th} record of the j^{th} object in ME
d	The previous relative distance in odometry update of ME
d'	The new relative distance in odometry update of ME
d_j	The relative distance estimation of the j^{th} object in ME
DA_i^j	The distance of the i^{th} record of the j^{th} object in ME
f_{HP}	The function that returns a history coefficient according to the number of percepts available in S-Loc
$f_{wc}(n_j)$	The function which gives the weight for the confidence of an objects with n_j known records
f_{wpu1}	The function that returns a weight component for a pose according to the perception for which the pose sample is calculated
f_{wpu2}	The function that returns a weight component for a pose according to the current perceptions
f_{wpu3}	The function that returns a value related to the difference in the x-coordinate and the y-coordinate
$f_{ws}(i)$	The function which gives the weight of the i^{th} record for a static object
KA_i^j	The flag which is equal to one if the i^{th} record of the j^{th} object exists and zero otherwise in ME
n_w	The window size in ME
o	The observation data in ML
PA	The percepts array in S-Loc
PA^i	The perception data of the i^{th} perceived landmark which is stored as the i^{th} element of the perception array
PA_c^i	The perception confidence of the i^{th} landmark in the percepts array

PA_d^i	The perceived relative distance of the i^{th} landmark in the percepts array
PA_k^i	The flag of perception array that is one if the i^{th} object is perceived
PA_x^i	The actual x-coordinate of the i^{th} landmark in the percepts array
PA_y^i	The actual y-coordinate of the i^{th} landmark in the percepts array
PA_θ^i	The perceived relative angle of the i^{th} landmark in the percepts array
PE_c	The confidence of the pose estimate before a perception or odometry update in S-Loc
PE_c^*	The confidence of the updated pose estimate in S-Loc
PE_x	The x-coordinate of the pose estimate before a perception or odometry update in S-Loc
PE_x^*	The x-coordinate of the updated pose estimate in S-Loc
PE_y	The y-coordinate of the pose estimate before a perception or odometry update in S-Loc
PE_y^*	The y-coordinate of the updated pose estimate in S-Loc
PE_θ	The orientation of the pose estimate before a perception or odometry update in S-Loc
PE_θ^*	The orientation of the updated pose estimate in S-Loc
PS_c^i	The confidence of the i^{th} pose sample in S-Loc
PS_w^i	The weight of the i^{th} pose sample in S-Loc
PS_x^i	The x-coordinate of the i^{th} pose sample in S-Loc
PS_y^i	The y-coordinate of the i^{th} pose sample in S-Loc
PS_θ^i	The orientation of the i^{th} pose sample in S-Loc
s_j	The speed estimation in current pose estimation for dynamic objects of ME
w_j	The total weight for the j^{th} object in ME
Δx	The signed distance the agent moved in sideways in odometry update of ME
Δy	The signed distance the agent moved on its orientation in odometry update of ME

α_j	The relative direction of the speed estimation
θ	The previous relative angle in odometry update of ME
$\Delta\theta$	The angle the agent has turned in odometry update of ME
θ'	The new relative angle in odometry update of ME
θ_j	The relative angle estimation of the j^{th} object in ME

AI	Artificial Intelligence
A-MCL	Adaptive Monte Carlo Localization
EKF	Extended Kalman Filter
MCL	Monte Carlo Localization
ME	My Environment
Mix-MCL	Mixture Monte Carlo Localization
ML	Markov Localization
ML-EKF	Markov Localization - Extended Kalman Filter
R-MCL	Reverse Monte Carlo Localization
SLAM	Simultaneous Localization And Mapping
S-Loc	Simple Localization
SRL	Sensor Resetting Localization
UDP	User Datagram Protocol

1. INTRODUCTION

The localization problem can be defined as the detection of the pose (i.e. the position and the orientation) of a robot relative to the environment, using the information about the environment sensed by the robot when the starting position is unknown. In short, localization is the problem of having the robot answer the question: *"Where am I?"* for itself.

A robot uses its sensors (such as infrared, camera, etc.) to gather information about the environment. Except for specially designed toy problems, these sensors and the environment are uncertain, and therefore the results are mainly erroneous and inaccurate. For this reason, the localization problem still remains to be nontrivial and challenging. Consequently, localization is an active field of study and although much effort has been put into it, there is room for more.

Triangulation is one the basic localization approaches for mobile robots [1]. It is a well-known technique for estimating the position of a robot using perception of fixed points (i.e. landmarks) in the environment.

Another approach is based on fuzzy logic [2, 3]. This approach is a grid-based one where uncertainty is represented in terms of fuzzy membership functions.

A well-known approach for localization is based on the Kalman filter (Kalman-Bucy filter) [4, 5]. Kalman filter uses Gaussian distributions to represent all the densities such as positions, odometric and sensory measurements. This way, it integrates uncertainty into the computations.

Markov Localization (ML) is a grid based method which is similar to the Kalman filter approach except that it does not make a Gaussian distribution assumption but allows any kind of distribution to be used instead [6, 7]. Although this feature brings flexibility, it also brings computational overhead.

Monte Carlo Localization (MCL) algorithms represent a robot's belief by a set of weighted samples [8, 9]. These samples approximate the posterior probability of the robot's pose by using a Bayesian formulation of the localization problem.

During the study of this thesis many such localization techniques were taken into consideration especially for the RoboCup 2005 Sony Four-Legged Robot League. Considering the advantages and the disadvantages of these techniques, it was decided to implement a new localization technique, Simple Localization, together with a version of MCL.

The RoboCup is an international organization which aims to build a fully autonomous humanoid robot soccer team and beat the official human world soccer champion team by the year 2050 [10, 11]. Currently, a number of different RoboCup soccer leagues that focus on different aspects of this challenge exist under RoboCup.

The Sony Four-Legged Robot League is one of these leagues. In the Sony four-legged league, two teams each consisting of four Sony AIBO robotic dogs compete against each other in a soccer game. Four unique bi-colored beacons are placed in the game area, which is 6m by 4m, in order to provide information for the localization. Robots operate fully autonomously. Any human intervention other than placing robots on the field is strictly prohibited.

The localization module of the Cerberus Team (Boğaziçi University- Turkey) was a test bed for this thesis. Other than the experiments presented in this report, the proposed solution is tested at both the soccer competition and the technical challenges of the RoboCup 2005 Sony Four-Legged League.

The Cerberus team participated with Sony Aibo ERS-210 robots whereas all the other teams used Sony Aibo ERS-7 robots, which was a newer model with many advances. This drawback caused the early elimination of the team from the soccer competition.

At the technical challenges, the Cerberus team, having the highest total score from three independent challenges, became the champion. The role of the localization module, and of this thesis study, was significant in this success.

This thesis proposes a new localization approach together with an intermediate module between the perception and localization modules. The proposed solution is the use of these two modules.

Having the perception data buffered and filtered in the first module, called My Environment module, and then using the output of this module the new localization module, called Simple Localization module, very accurate results are shown to be obtained by its performance in both the experiments and the RoboCup 2005 competitions.

In Chapter 2, the localization problem is discussed and some well-known techniques are defined. My Environment module, which is an important part of the proposed system, is explained in Chapter 3. The proposed localization algorithm, S-Loc, is discussed in Chapter 4.

In Chapter 5, the application domain to which the proposed solution is applied in practice is explained. The experiments and their results are discussed in Chapter 6. Finally, Chapter 7 contains the summary of the work and points the possible future work.

2. BACKGROUND

Localization is an active field of robotics where many approaches are introduced into the literature. Starting with a short description of localization, some of the most common ones of these approaches are covered in this chapter.

2.1. Localization Problem

The global localization of autonomous robots is a very challenging issue, as the sensing devices that provide the data are unreliable, the provided data are generally highly noisy, the environment is dynamic and highly unpredictable, and the number of useful (i.e. distinguishable, recognizable) features of the nature which could be used for self-localization of the robot is inadequate.

There are two sub problems of the localization problem. They are position (pose) tracking and global self-localization. Pose tracking is the problem of keeping track of the robot's pose (position and orientation) using odometry, assuming that the initial pose of the robot is already known [12]. There are two main drawbacks. The first one is that dead reckoning error grows cumulatively in time. The cumulative error in the orientation of the robot is especially very critical as it may lead the future odometry updates to be done in the wrong direction. The second drawback is actually the fact that initial position is usually not likely to be known in real time applications.

On the other hand, global self-localization is the problem of making the robot find its location in the environment where no priori information is available. Sensors are used to make perceptions and gather information about the environment. Because of the high uncertainty and noise of the sensors and the dynamic nature of the environment, global self-localization is a very hard task and is a challenging problem for researchers. The proposed solutions range from the most basic methods based on simple geometric calculations where uncertainty is not considered at all to complex statistical methods using sophisticated models.

Some of the proposed solutions give sufficient results, but because of the limitations of the memory and processing power they are not suitable for all applications. Faster algorithms with less memory requirements are required especially for real-time applications, such as robotic soccer competitions where the environment is highly dynamic and the robot's computational capacity is limited with low onboard resources. While the existing fast solutions generally give imprecise results, the accurate approaches suffer from slowness and high memory usage. Even though some of these fast approaches may produce precise local results, they fail to find the global pose.

2.2. Localization Techniques

Many techniques have been introduced so far which range from the most basic methods based on simple geometric calculations to complex statistical methods and hybrid solutions using sophisticated models to integrate uncertainty. Below, some of the well-known techniques are described briefly.

2.2.1. Triangulation

The triangulation technique uses the geometric properties of triangles to compute the pose of a robot [1]. Triangulation can be done using only the distances or it can be done using primarily angle measurements.

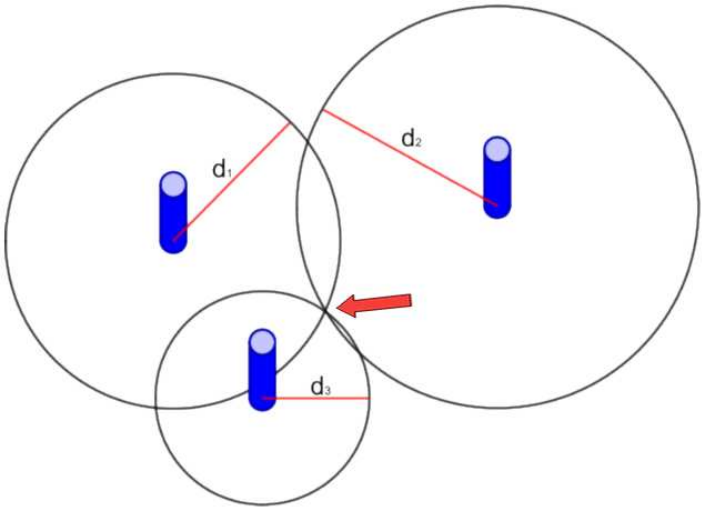

Figure 2.1. Triangulation using only distances

In the first way, the position of the robot is computed by measuring its distance from multiple reference positions. Calculating a robot's position in two dimensions requires distance measurements from 3 non-collinear points as shown in Figure 2.1. On the other hand, the orientation of the robot cannot be calculated without the use of the angle measurement with respect to the reference points.

The second way is similar to the first one except that, instead of distances, angles are used for determining the position of the robot. In general, for two dimensional localization, two angle measurements and one distance measurement is required as shown in Figure 2.2.

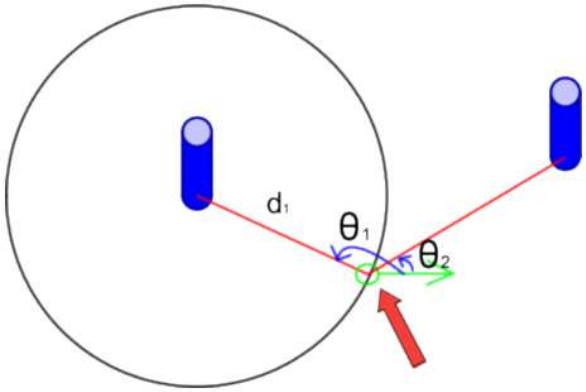

Figure 2.2. Triangulation using two angle measurements and one distance measurement

Although two angle and one distance perceptions are enough for triangulation, with noisy data, whenever there are more perception data available, they should be used in order to reduce the error. In such cases there are more than one poses estimate. There are different ways of combining these estimates to obtain a final position estimate. One simple way is to compute their average.

2.2.2. Fuzzy-Localization

Fuzzy-Localization is another approach which is based on fuzzy logic [2, 3]. This approach is grid-based and the uncertainty is represented in terms of fuzzy membership functions. Fuzzy sets are used to represent the range r and heading θ of the robot. For the representation of the uncertainty in sensor readings, the trapezoidal membership

functions are useful. Using an appropriate bias value helps recovering from kidnapping problem, where the robot is kidnapped to test the robustness. Trapezoidal fuzzy sets are represented with a tuple of the form (θ, Δ, α, h, b), where θ is the center, Δ is the width of the core, α is the slope, h is the height and b is the bias as in Figure 2.3.

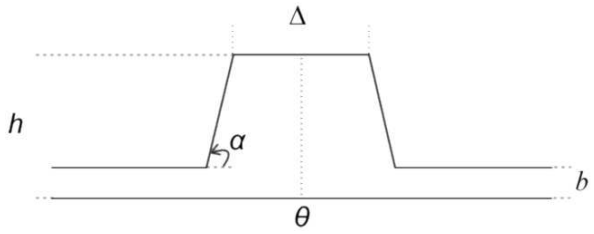

Figure 2.3. Trapezoidal fuzzy sets

Bias is used to integrate the uncertainty that the solution might be somewhere else. This technique does not critically rely on the accuracy of these parameters [2]. The tuples of the form (r, Δ, α, h) represent the distance from the observed landmark to every grid. A two-dimensional fuzzy grid map $Gt(x, y)$, where each grid measures the degree of possibility is used to represent the robot's position at time t. For perception update, the probability distribution $St(x, y|r)$ represents the possibility of having the robot be in (x, y) using the information that the observed landmark is at distance r from the robot.

A predict-observe-update cycle is followed in this approach. In the observe step the perception information is used to calculate the probability distribution of the grids, whereas the predict step makes use of the odometric information to update the current position. Finally, in the update step the probability distribution of the grids is integrated into the fuzzy grid map.

2.2.3. Kalman Filter Method

Another well-known approach for the localization problem is the Kalman filter (Kalman-Bucy filter), which makes the assumption of Gaussian distributions to represent all densities including positions, odometric and sensory measurements to integrate uncertainty into the computations.

Odometry updates and perception updates are performed on the pose estimations using the property that Gaussian distributions can be combined using multiplication [4, 5].

Kalman filter method fails to make global localization, and it can not recover from total localization failures because of the drawback that only one pose hypothesis can be represented [13]. Therefore, this method is usually used to locally track the object's pose together with another method, which is in charge of the global localization, forming hybrid approaches.

However, it is a very efficient algorithm that gives precise results in the absence of kidnapping and total localization failures. On the other hand, additional coefficient and covariance computations can increase the computational time undesirably [5].

For non-linear and potentially numerically unstable system models, Extended Kalman Filter (EKF) is more useful where uncertainty is represented by the first and second moments of the density [14].

2.2.4. Markov Localization Method

Markov Localization (ML) is a localization method similar to the Kalman filter approach, on which many studies are based in the literature. ML differs from the Kalman filter approach with not making a Gaussian distribution assumption but allowing any kind of distribution to be used [6, 7]. This aspect brings flexibility, however it also adds a computational overhead.

ML is a grid based method that uses odometry measurements a and perceptional measurements o [15]. The key idea is that the robot maintains a belief over its position. The belief of the robot at time t will be denoted as $Bel^{(t)}(L)$. Here L is a three-dimensional random variable composed of the robot's x-y position and its heading direction θ. This way, $Bel^{(t)}(L = l)$ is the belief that the robot is at the location[1] l.

[1]The *pose* and *location* are used interchangeably

The initial knowledge of the robot is reflected by $Bel^{(0)}(L)$. When the initial pose of the robot is not known, $Bel^{(0)}(L)$ is initialized by a uniform distribution. $Bel^{(t)}(L)$ is the posterior with respect to all data collected up to time t as in Equation 2.1.

$$Bel^{(t)}(L) = P(L^{(t)}|d^{(t)}) \tag{2.1}$$

where $d^{(t)}$ denotes the data collected *up to* time t. For the perception update, the last item in $d^{(t)}$ is a perception data, $o^{(t)}$. Using the Markov assumption, $Bel^{(t)}(L = l)$ is calculated for each l as in Equation 2.2, and it is updated as in Equation 2.3.

$$\begin{aligned} Bel^{(t)}(L = l) &= P(L^{(t)} = l|d^{(t)}) \\ &= \alpha\, P(o^{(t)}|L^{(t)} = l)\, Bel^{(t-1)}(L = l) && (2.2) \\ Bel(l) &\leftarrow \alpha\, P(o|l)\, Bel(l) && (2.3) \end{aligned}$$

where α is a normalizer independent from l; $P(o^{(t)}|L^{(t)} = l)$ is the probability of making the perception $o^{(t)}$ given that the robot is at l at the time t; and $Bel^{(t-1)}(L = l)$ is the belief that the robot was at l at the time $t - 1$.

For the odometry update, the last item in $d^{(t)}$ is an odometry data, $a^{(t)}$. Using the Theorem of Total Probability and exploiting the Markov property, $Bel^{(t)}(L = l)$ is calculated for each l as in Equation 2.4, and it is updated as in Equation 2.5.

$$\begin{aligned} Bel^{(t)}(L = l) &= P(L^{(t)} = l|d^{(t)}) \\ &= \int P(L^{(t)} = l|a^{(t)}, L^{(t-1)} = l')\, Bel^{(t-1)}(L = l')\, dl' && (2.4) \\ Bel(l) &\leftarrow \sum_{for\ each\ l'} P(l|a, l')\, Bel(l') && (2.5) \end{aligned}$$

2.2.5. Monte Carlo Localization

Monte Carlo Localization (MCL) is very similar to Markov localization, but approximates belief functions using a Monte Carlo method. Instead of dividing the space

into grids, MCL uses a sample-based representation [8, 9]. In MCL, beliefs are represented by a set of K weighed samples (particles) which are of the type ((x, y, θ), p), where $p \geq 0$ is a numerical weighting factor and sum of all p is 1. Odometric and sensory updates are similar to ML.

Most of the MCL based approaches suffer from the kidnapping problem, since the main theory converges to a pose and then it collapses when the current estimate does not fit observations. There are some extensions to MCL that addresses this problem such as adding random samples at each iteration. Sensor Resetting Localization (SRL), Mixture MCL (Mix-MCL), and Adaptive MCL (A-MCL) are some examples to such methods.

In SRL, a small amount of uniformly distributed random samples is added when the likelihood of the current observation is below than a threshold [16]. Mix-MCL, which has been developed for extremely accurate sensor information, additionally gives weights to these random samples according to the current probability density p(l) [13].

The key idea of the Adaptive MCL is to use a combination of two smoothed estimates (long term and short term) of the observation likelihoods [13]. In A-MCL, samples are added only when the difference between short-term estimate (slow changing noise level in the environment and the sensors) and the long-term estimate (rapid changes in the likelihood due to a position failure) is above a threshold. In 2002, University of Washington team applied this approach to the RoboCup domain [17].

2.2.6. Markov Localization - Extended Kalman Filter

Markov Localization - Extended Kalman Filter (ML-EKF) method is a hybrid method of ML and EKL. Taking into consideration the fact that ML is more robust and EKF is more accurate, it aims to make use of the advantages of both methods. So ML-EKF finds the location of the robot roughly using grid based ML, and then uses EKF inside this area to find a more accurate solution [13].

2.2.7. Reverse - Monte Carlo Localization

Although ML is robust and converges fast, it is also coarse and computationally complex. On the other hand, sample based MCL is not as computationally complex as ML, and gives accurate results; but it can not converge to a position as fast as ML, especially in the case of an external impact on the position of the robot such as kidnapping. In addition, the number of samples used in MCL is generally very high for covering the complete space and converge to the right position.

There are several extensions to MCL for adaptive sample size usage, but these techniques still do not solve the slow coverage problem. To address this issue, Reverse - Monte Carlo Localization (R-MCL) converges to several cells by ML, and then inside these bulk of grids, produces a limited number of samples to find the final position [18, 19, 20]. The average of these samples gives the final position and the standard deviation might give the uncertainty of the final position as in the MCL based methods.

3. MY ENVIRONMENT

My Environment is the first module introduced in this thesis work. This module was initially designed as a part of the localization module and to increase the performance of the localization. It was decided to be a separate module as not only the localization module, but the other modules could also benefit from its output.

3.1. Introduction

For a human to predict his/her pose, i.e. his/her coordinates and the orientation, vision is the primary input. Estimating the distance and the orientation with respect to known static objects, he/she can calculate his/her pose. These estimates are valid not only when they are seen, but also for a period of time after they were perceived, with having the estimates' confidence decreasing in time.

The buffering of objects in the environment can be done either with their actual poses or their poses with respect to the observer. Buffering the actual poses of objects in time, the coordinates of the objects with respect to the environment are calculated using the current perceptions, and stored in an array of data structures as shown in Figure 3.1.a. The odometry update and the instantaneous pose calculations are very simple and could be done at a low cost.

Buffering the relative poses of the objects, the perceived distances and relative angles are directly stored in an array of data structures which are used as buffers as shown in Figure 3.1.b. This way, the odometry update and the instantaneous pose calculations are relatively more complex, and since they require trigonometric functions, their cost is higher.

Even though the cost of buffering relative poses of objects is more than the cost of buffering actual poses of objects in time, as buffering relative poses does not involve localization it is more robust. Involving localization in the calculation results

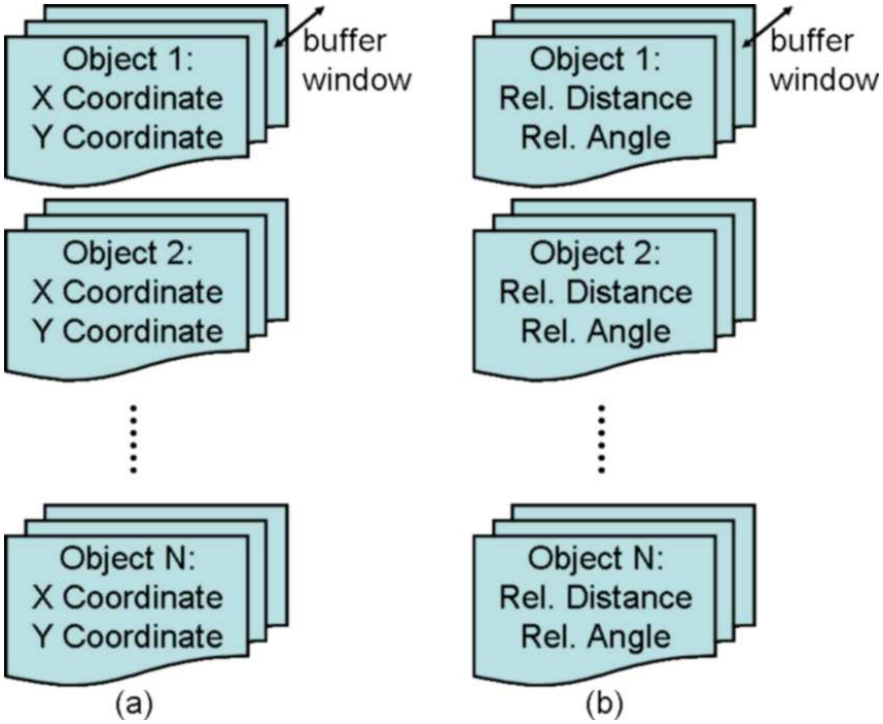

Figure 3.1. Buffering the poses of objects: (a) Buffering actual poses, and (b) Buffering the relative poses

in involving localization error in the output. For each stored value on an object's pose history, having instantaneous localization error added to the perception error, the estimations on the current pose of the object would involve more noise.

In addition, buffering the relative poses of the objects makes it meaningful to buffer the poses of the static objects. This is very valuable for localization in two ways. First, the processed static object poses would be more robust and lead to more accurate agent pose estimations. Secondly, this buffering will make it possible to process more static objects than that are seen in any moment of time, as long as the buffered relative poses could give a proper estimate for the current pose of the static object.

3.2. General Outline of ME

My Environment (ME) is a module between the perception module, or any other module that handles the perception of the environment objects, and the other modules that use the output of the perception module as shown in Figure 3.2. In some

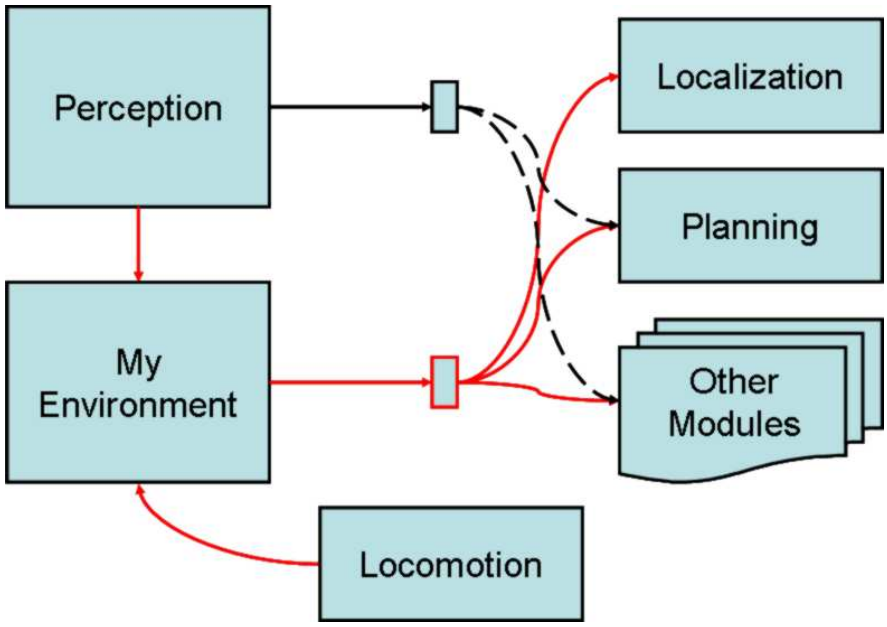

Figure 3.2. Interaction of ME with the other modules

exceptional cases, where the position and the relative angle data are not sufficient, the perception data may be needed to be used directly. For instance, in the robot soccer domain, the ball tracking behavior for the head, where the coordinates of the perceived ball region on the camera frame image may be used to calculate the next pan - tilt parameters, uses the perception output directly.

Localization is one of the modules that require the perception data. During the perception update of the localization module, the perceived static objects are used to estimate the current pose of the agent.

The input of the perception module is the current internal state of the agent and the latest camera frame image. Input from a camera is often very noisy and may cause false perceptions. Not only the distance of an object could be perceived wrong, sometimes the object could be recognized as another object. If the data from the perception module are used as they are, these errors could result in unwanted behavior in other modules.

Filtering the output of the perception module in time using the past perceptions of the objects, more stable and robust data could be provided for the localization module

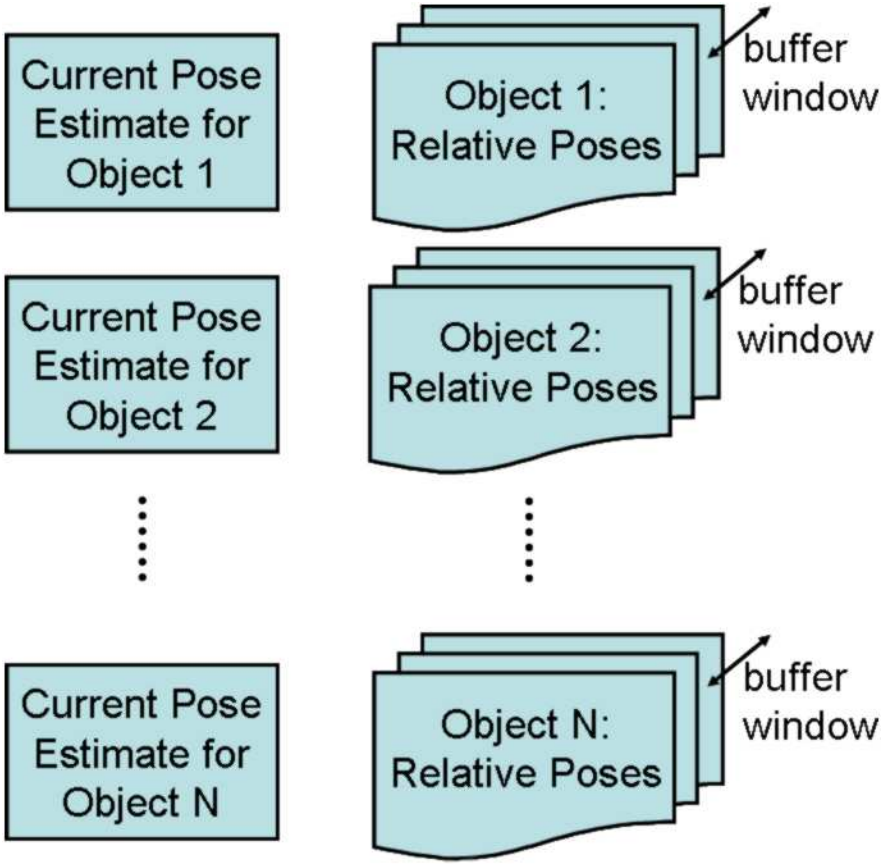

Figure 3.3. The data structure of ME

as well as other modules. This way the effect of false perceptions and recognitions would be decreased.

It should be kept in mind that for a mobile agent, using the past perceptions could lead to problems if the motion of the agent is not reflected on the past perceptions.

3.3. Architecture of ME

There are two kinds of objects in the ME architecture: static objects and dynamic objects. Each object, either static or dynamic, has a buffer window for storing the most recent perception data for that object, and an additional buffer for the current estimation for that object. The data structure of ME is shown in Figure 3.3.

Each static object entry has a distance, relative angle and a confidence regarding its perception. In the case when a static object has its own orientation, the orientation

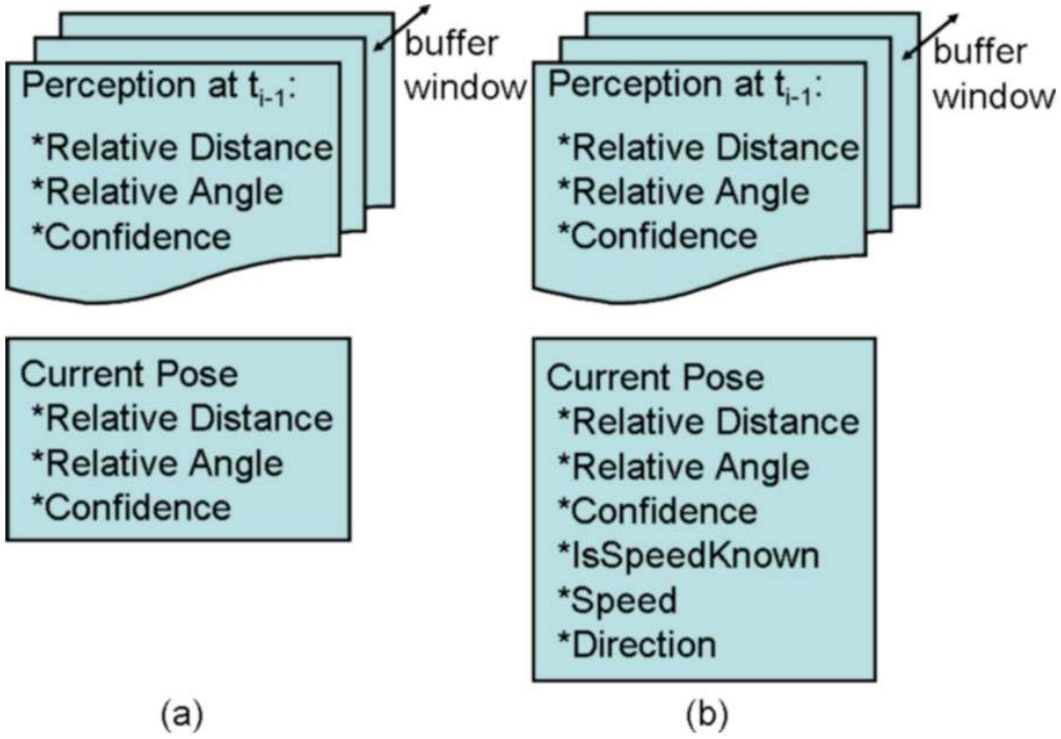

Figure 3.4. Data structures of (a) static objects, and (b) dynamic objects

values are also to be stored in the buffer. For dynamic objects, the velocity should also be calculated, but as it is not directly extracted from a single camera frame, it is not necessary to be buffered. In Figure 3.4a and 3.4b the data structures for static and dynamic objects are defined.

The window size is a hyper-parameter of ME. The noise in odometry, the dynamicity of the agent's pose, and the frequency of the processed vision frames play an important role in the selection of a good window size parameter. For dynamic objects, the speed of such objects should also be taken into account.

For instance, for Cerberus'05, the robots used have a maximum speed of approximately 30 cm/s (with ERS 210s), the odometry is quite noisy as the robots are legged, and the average vision frame processing time was 18 fps. For such conditions, the windows size for static objects and dynamic objects were chosen to be 32 and 12 respectively.

3.4. Procedures of ME

There are five main procedures of ME. Other than *initialization*, the first two procedures, *perception update* and *odometry update*, are triggered as new perception data from the perception module and new odometry data from the locomotion module arrive. The other two, *current pose estimation for static objects* and the *current pose estimation for dynamic objects*, are called inside the *perception update* procedure.

3.4.1. Initialization

As an initialization, it is necessary for the pose estimations to set all the buffers in the windows of all the objects, for both the static objects and the dynamic objects. If there is no prior information about the pose of an object when the system starts, all the buffers in its window are to be marked as unknown. If the initial pose of an object is known a priori, then this can be provided to the system by filling the buffers in its windows according to that knowledge.

3.4.2. Perception Update

For each dynamic and static object, the oldest record on the buffer is deleted and a new record is stored from perception if the object is seen at the moment, otherwise it is marked that there is information available about the pose of the object at that time. After updating the buffer with the latest perception output, an estimation is done for each object concerning its pose by calling the appropriate procedure in Section 3.4.4 or Section 3.4.5.

3.4.3. Odometry Update

For the odometry update of each record, three trigonometric functions and a square root is used. For a ME with n_o number of objects and a window size of n_w, there are $n_o \times n_w$ number of records. These calculations increase the cost of ME, but they are mandatory for reasonable ME estimations.

Since all the information in the buffers of all objects' windows is relative to the agent, on each movement of the agent, they need to be modified. For each record new relative distance and the relative angle values have to be calculated using Equations 3.3 and 3.4.

$$c = cos(\theta) \times d - \Delta x \tag{3.1}$$

$$s = sin(\theta) \times d - \Delta y \tag{3.2}$$

$$d' = \sqrt{c^2 + s^2} \tag{3.3}$$

$$\theta' = tan^{-1}(s/c) + \Delta\theta \tag{3.4}$$

where θ, d, Δx, Δy and $\Delta\theta$ are the previous relative angle, previous relative distance, the signed distance the agent moved in sideways, the signed distance the agent moved on its orientation and the angle the agent has turned, respectively. The new relative distance and the new relative angle are represented with d' and θ' respectively.

3.4.4. Current Pose Estimation for Static Objects

For each static object, this function is called once in every *perception update*. Using the window sized perception data records; a pose estimation is made for its use in other modules of the agent's architecture like localization and planning.

In Equations 3.9, 3.10 and 3.11 the confidence estimation, the relative distance and relative angle of the static object are calculated.

$$w_j = \sum_{i=0}^{n_w} KA_i^j \times CA_i^j \times f_{ws}(i) \tag{3.5}$$

$$\Delta x_j = \frac{\sum_{i=0}^{n_w} KA_i^j \times DA_i^j \times cos(AA_i^j) \times f_{ws}(i)}{w_j} \tag{3.6}$$

$$\Delta y_j = \frac{\sum_{i=0}^{n_w} KA_i^j \times DA_i^j \times sin(AA_i^j) \times f_{ws}(i)}{w_j} \tag{3.7}$$

$$n_j = \sum_{i=0}^{n_w} KA_i^j \tag{3.8}$$

$$c_j = \frac{\sum_{i=0}^{n_w} KA_i^j \times CA_i^j \times f_{ws}(i)}{w_j} \times f_{wc}(n_j) \tag{3.9}$$

$$d_j = \sqrt{\Delta x_j^2 + \Delta y_j^2} \tag{3.10}$$

$$\theta_j = tan^{-1}(\Delta y_j / \Delta x_j) \tag{3.11}$$

where j is the index of the object, n_w is the window size; w_j is the total weight for the j^{th} object, $f_{ws}(i)$ gives the weight of the i^{th} record for a static object; $f_{wc}(n_j)$ gives the weight for the confidence of an object with n_j known records in its windows; KA_i^j is a flag which is equal to one if the i^{th} record of the j^{th} object exists (i.e. the object was perceived at the time that record was buffered) and zero otherwise; DA_i^j is the distance of the i^{th} record of the j^{th} object; AA_i^j is the relative angle of the i^{th} record of the j^{th} object; CA_i^j is the confidence of the i^{th} record of the j^{th} object; c_j is the confidence estimation of the j^{th} object; d_j is the relative distance estimation of the j^{th} object; and θ_j is the relative angle estimation of the j^{th} object.

The function $f_{ws}(i)$ is a monotonically increasing function. The value of $f_{ws}(i)$ is to be arranged such that the more recent a record is the more weight it will receive, but at the same time it will not let too small number of records (i.e. one or two) dominate the value of the weight. Although a linear function could easily be used for that purpose, a sigmoid function could be expected to give better results if configured properly for the application.

The function $f_{wc}(i)$ is also a monotonically increasing function for favoring the confidence with respect to the number of records of which poses are available.

If n_j is zero, meaning that none of the buffers in the window stores a perceived pose, then no estimation could be made and the object's pose is set as unavailable. After the confidence is calculated, if it is below a predefined threshold, the object's pose is also set as unavailable.

3.4.5. Current Pose Estimation for Dynamic Objects

The procedure of the current pose estimation for the dynamic objects is the same as the current pose estimation for the static objects except that for dynamic objects the speed and the direction of the speed should also be calculated when possible. For each dynamic object, this function is called once in every *perception update*. Using the same equations in the Section 3.4.4 the pose estimation, with the exception of the speed related variables, is made, but the $f_{ws}(i)$ function is replaced with $f_{wd}(i)$, which gives the weight of the i^{th} record for a dynamic objects.

If an object is dynamic, perceiving the object in different poses may be either due to noisy perception or the object's movement. If the object's recent poses are buffered and used for the estimation of the current pose, the weights of the most recent records should be higher than they are for static objects, where they should be always perceived in the same pose. As a remark, it should be noted that the effect of the movement of the agent is eliminated with the motion update procedure.

The function $f_{wd}(i)$ is also a monotonically increasing function as the function $f_{ws}(i)$ is, but favors the most recent records more than $f_{ws}(i)$. Since the older records are less important for the dynamic objects, the window size for the dynamic objects could be set less than the window size set for the static objects.

The speed of a dynamic object is estimated only if the pose of the object was available from the previous run, otherwise the speed is set as unavailable.

In Equations 3.13 and 3.14 the speed and relative direction of the speed of the dynamic object are calculated.

$$h_j = d_j^2 + d_{j-1}^2 - 2 \times d_j \times d_{j-1} \times cos(\theta_j - \theta_{j-1}) \tag{3.12}$$

$$\alpha_j = \Pi - cos^{-1}(\frac{h_j - d_j^2 + d_{j-1}^2}{2 \times h_j \times d_{j-1}}) - \theta_{j-1} - \theta_j \tag{3.13}$$

$$s_j = \frac{\sqrt{h_j}}{\Delta t} \tag{3.14}$$

where j is the index of the object, d_j is the relative distance estimation of the j^{th} object; θ_j is the relative angle estimation of the j^{th} object; α_j is the relative direction of the speed estimation; and s_j is the speed estimation.

3.5. Advantages and Disadvantages of ME

ME provides more stable results for both static and dynamic objects. For static objects, especially in the case when localization does not give accurate results, the pose of the static object at ME would be more robust. For localization, the ME output poses can be used as if they were perceived from the sensors at that time. This way, perceived objects are not forgotten just after they are perceived, but remain in ME for a period of time. In addition, the noisy perception, which from time to time may lead to false object detections, could be stabilized.

ME provides more stable results for dynamic objects as well. Using the ME output instead of the perception output directly, instantaneous fluctuations in the pose of the object are smoothened. Losing the dynamic object in some camera frames and perceiving it again frequently, which is not a rare thing in robotics perception modules, could lead to oscillations in the operations and the outputs of some of the modules. ME smoothens these oscillation with its pose estimation, where it uses the recent perceptions to calculate the current pose.

These advantages have a cost. The space needed to store the pose buffers and current estimations of objects in ME grows linearly with the product of window size of the pose buffers and the number of objects in ME. The complexity of the ME procedures is $O(n_w \times n_o)$, where n_w is the window size and n_o is the number of objects in ME. Using the output of the perception module, both the processing and memory expenses of ME will be saved, but if the system can afford these expenses, the benefits of ME could be worthwhile.

It should also be noted that using ME, the agent would observe dynamic objects slower than they are. This is because of using the previous poses of the dynamic object

in the calculation of the current pose. This problem could be minimized theoretically by adding the velocity of the previous estimation times the time passed to the previous records of that object, but this could bring more noise than it corrects as the velocity estimations could be noisier than the relative position estimations.

4. S-LOC: SIMPLE LOCALIZATION

S-Loc is the localization technique proposed by this thesis. It used not only in experiments but in also the Cerberus'05, which participated in RoboCup 2005 - Sony Four-Legged League and won the technical challenges.

During the development of the localization module of Cerberus'05, many techniques were taken into consideration.

- Triangulation is a simple and accurate technique, but is not robust. It is too much effected by noise, specially by the false perceptions [1].
- Fuzzy localization techniques generally have high computational complexity, and does not give accurate enough results that are worth that cost [2, 3].
- The major disadvantage of Kalman Filter methods is that they do not have the capability of recovering from kidnapping [4, 5].
- ML approaches are generally expensive, where false perceptions could be big problems [6, 7].
- Raw MCL cannot recover from kidnapping, but a version of it, SRL is implemented [8, 9, 16].
- ML-EKF is also another expensive technique, which would not be preferred in a case where a much lower cost algorithm could give accurate and robust results [13, 21].
- R-MCL is also another technique, which is used in the experiments for comparison purposes [18, 19, 20].

Considering the points above, it was decided to implement S-Loc together with a version of SRL. The existing R-MCL module implemented in our laboratory is also used in the experiments.

4.1. Introduction

In general, the localization process has two main steps. The first one is the perception update, which is based on the perceptions in order to calculate the estimated pose of the agent. Since the movement of the agent changes its pose, the second step, the odometry update, is necessary for reflecting the effect of the movement on the calculations and the estimations.

Perception update, as it depends on the perceptional information, usually includes high amount of noise. Although the agent is dynamic, its pose should not be highly unstable, i.e. the pose should not jump to different far poses on the field frequently. Using a memory for the previous pose estimate, and updating it with the current estimate could handle the big fluctuations and increase the robustness to the false perceptions of static landmarks.

In order to use triangulation, three objects, of which perceptions are not available so often at the same time, are needed to be perceived. Also, even if three objects are available, in the case where one of the perceptions is wrong or is highly noisy, the calculation will lead to a very noisy pose estimation.

In the MCL, there is a large number of sample poses, for which many calculations should be made in order to find their confidences. Generally, most of these samples do not hold any useful information. Also, noisy perception data may lead to unstable pose estimations.

The principle of ML leaves open how the robot's belief is represented and how the conditional probabilities are computed. Existing ML approaches mainly differ in the representation of the state space and the computation of the perceptual model. These approaches are generally expensive, since the space is discretized and for each perception and for each location, the probabilities should be calculated at each frame. False perceptions could also be major problems.

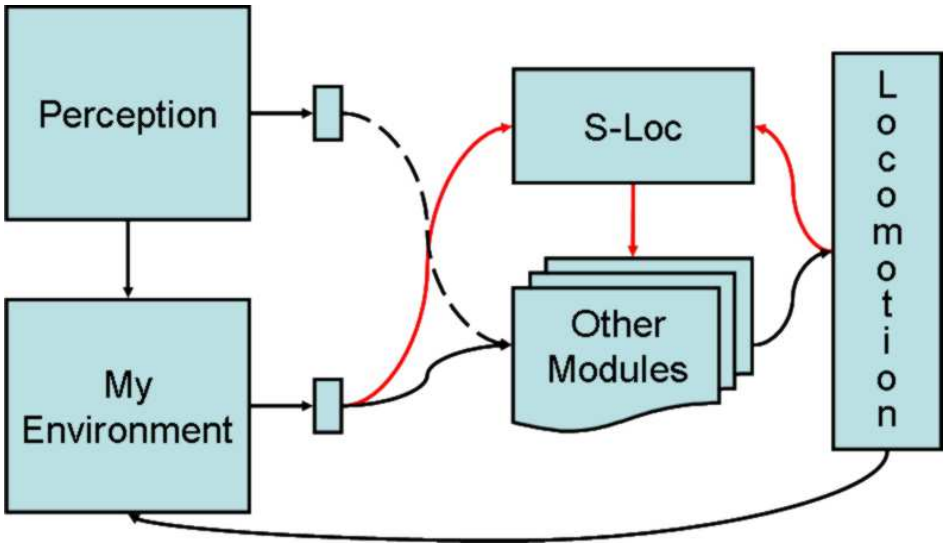

Figure 4.1. The relationship of the S-Loc module with the other modules

In the perfect, noiseless case, the odometry data should be continuous and the pose should be updated continuously as the agent moves. On the other hand, in the real world case, the odometry data is generally very noisy, especially when the agent uses legs for locomotion; and arrives at discrete times, for instance after a step is completed. Both of these make the previous pose estimates less confident for the current estimate calculations.

4.2. General Outline of S-Loc

S-Loc is a localization module. It needs the perception data and the odometry data for updating the pose estimate, which it provides as the output. This pose estimate is then used in other modules. The relationship of the S-Loc module with the other modules is shown in Figure 4.1.

The perception data can be supplied directly from a vision module (or any other perception module), or they can be supplied by the ME module where they are buffered and estimates using them are produced. It could perform better if the perceptional input is provided by the ME module, because the ME module provides more stable and robust data.

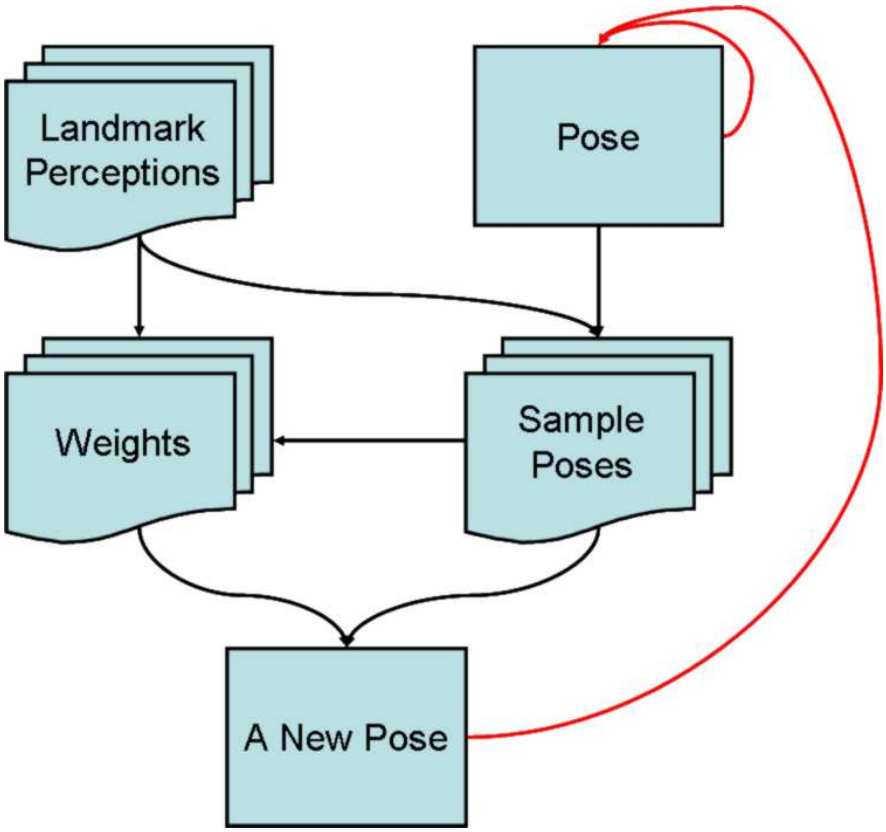

Figure 4.2. The perception update process

The locomotion module provides the odometry data at certain times, which is generally less frequent than the perception data. The effect of the movement of the agent should also be reflected on the pose estimate.

4.3. Architecture of S-Loc

The perception update of the S-Loc depends on the perception of landmarks and the previous pose estimate. The perception update process is shown in Figure 4.2. Even if the initial pose estimate is provided wrong, it acts as a kidnapping problem and is not a big problem as S-Loc will converge to the actual pose in a short period of time if enough perception could be made during this period.

For each perceived landmark, a sample pose is calculated according to this perception and the previous pose estimate of the agent. The previous pose estimate is also taken as a sample pose.

For each sample pose, using all the landmarks, the likelihood of this sample pose is calculated. This is done by assuming that the agent's actual pose is the sample pose being processed and calculating the difference of the perceived landmarks positions and their actual positions. Also, the confidence of the perception is reflected on the likelihood.

After these likelihood calculations are done for each sample pose, these likelihoods are used for calculating the weights of the corresponding sample poses, and a new pose is calculated as the weighted average of these sample poses.

The weighted average of these sample poses is then used together with the previous pose estimate to calculate the current pose estimate. The purpose of not using the weighted average of these sample poses directly is to provide system enough memory to prevent big jumps of the pose estimate and make it more stable.

After the current position of the agent is estimated, it could be safer to calculate the current orientation of the agent using the current position estimation and the perceptions.

In the case of having no perception at a certain time, decreasing the confidence of the previous pose estimate, the current pose estimate could be obtained as no further data are available.

The odometry update process is as simple as updating the pose estimation with the odometry data. As only the pose estimation is used from the previous cycle of every estimation, no more update or calculation is necessary. On the other hand, if the frequency of the odometry update is much less than the frequency of the perception update, then it may be better to lower the weight of the odometry data accordingly. This is because of having the original odometry data to be the result of the motion during more than one perception update.

4.4. Procedures of S-Loc

There are three main procedures of S-Loc. The *initialization* is the first one. Other two procedures, *perception update* and *odometry update*, are triggered as new perception data from the perception module and new odometry data from the locomotion module arrive.

4.4.1. Initialization

The only thing to be done in the initialization procedure is to initialize the pose estimate to initial value. It does not have to be the actual pose the agent will have at the beginning, since S-Loc module can recover from kidnapping. On the other hand, it shall still be set to a valid pose initially in order not to cause a problem in the proceeding calculations.

4.4.2. Perception Update

As shown in Equations 4.1, 4.2, 4.3, 4.4 and 4.5, the first pose sample is the previous pose estimate.

$$PSA_x^0 = PE_x \tag{4.1}$$

$$PSA_y^0 = PE_y \tag{4.2}$$

$$PSA_\theta^0 = PE_\theta \tag{4.3}$$

$$PSA_c^0 = PE_c \tag{4.4}$$

$$PSA_w^0 = PE_c \times f_{wpu2}(PSA_x^0, PSA_y^0, PSA_\theta^0, PA) \tag{4.5}$$

$$PA_k^0 = 1 \tag{4.6}$$

where PS_x^0, PS_y^0, PS_θ^0, PS_c^0 and PS_w^0 are the x-coordinate, y-coordinate, orientation, confidence and weight of the first pose sample; PE_x, PE_y PE_θ, and PE_c are the x-coordinate, y-coordinate, orientation and confidence of the pose estimate before the perception update; PA, percepts array, is the collection of perception data of all the

perceived landmarks together with their coordinates that are known initially; f_{wpu2} is the function that returns a weight component for a pose according to the current perceptions; and PA_k^0 is set to one in order to have the first element of pose sample array included in the proceeding calculations.

Then, a separate pose sample is calculated for each perception as in the Equations 4.8, 4.9, 4.11, 4.12 and 4.13.

$$\alpha_i = tan^{-1}\left(\frac{PE_y - PA_y^i}{PE_x - PA_x^i}\right) \tag{4.7}$$

$$PSA_x^i = PA_x^i + PA_d^i \times cos(\alpha_i) \tag{4.8}$$

$$PSA_y^i = PA_y^i + PA_d^i \times sin(\alpha_i) \tag{4.9}$$

$$\beta_i = tan^{-1}\left(\frac{PSA_y^i - PA_y^i}{PSA_x^i - PA_x^i}\right) \tag{4.10}$$

$$PSA_\theta^i = \pi + \beta_i - PA_\theta^i \tag{4.11}$$

$$PSA_c^i = PA_c^i \tag{4.12}$$

$$PSA_w^i = f_{wpu1}(PA^i) \times f_{wpu2}(PSA_x^i, PSA_y^i, PSA_\theta^i, PA) \tag{4.13}$$

where α_i and β_i are dummy angle variables; PSA_x^i, PSA_y^i, PSA_θ^i, PSA_c^i and PSA_w^i are the x-coordinate, y-coordinate, orientation, confidence and weight of the i^{th} pose sample; PA_x^i, PA_y^i are the actual x-coordinate and y-coordinate of the i^{th} landmark in the percepts array; PA_d^i, PA_θ^i and PA_c^i are the perceived relative distance, relative angle and the perception confidence of the i^{th} landmark in the percepts array; PA^i is the perception data of the i^{th} perceived landmark which is stored as the i^{th} element of the Perception Array; and f_{wpu1} is the function that returns a weight component for a pose according to the perception for which the pose sample is calculated.

The function f_{wpu1} returns the first component of the PSA_w^i for the argument PA^i. It may return PA_c^i directly or any other number that gives the confidence that the perception is correct. Since how accurate the pose sample is will be taken into account by the function f_{wpu2}, this function is independent of the corresponding pose sample. The purpose of this function is to decrease the weight of the pose samples, of which perception is less confident. In the case where the perception module does not

provide healthy confidence values, the perceived relative distance of the landmark can be used for the calculation of the return value. In such a case, a properly configured sigmoid function could be very suitable. If the landmarks are of different types and are known to be of different perception accuracy, then this could also be reflected on the return value.

The function f_{wpu2} returns the second component of the PSA_w^i. The return value is related to the accuracy the pose sample according to all the perceived landmarks. For each perceived landmark, the position of the perceived landmark is calculated by adding the perceived distance on the perceived relative angle to the pose sample, and the resulting position is compared to the actual position of the landmark. The difference gives the error. The return value should be a function of the error as in Equation 4.16.

$$par_x^j = \left| PA_x^j - (PSA_x^i + PA_d^j \times cos(PSA_\theta^i + PA_\theta^j)) \right| \tag{4.14}$$

$$par_y^j = \left| PA_y^j - (PSA_y^i + PA_y^j \times sin(PSA_\theta^i + PA_\theta^j)) \right| \tag{4.15}$$

$$f_{wpu2} = \prod_{j=1}^{N_L} PA_k^j \times f_{wpu3}\left(par_x^j, par_y^j\right) \tag{4.16}$$

where N_L is the number of landmarks; PA_k^j is one if the perception of the j^{th} landmark is available, and zero otherwise; and f_{wpu3} is a function that returns a value related to the difference in the x-coordinate and the y-coordinate.

The return value of the function f_{wpu3} is a value for the confidence of the corresponding sample pose for the corresponding perceived landmark. The greater the x-coordinate and y-coordinate differences provided as parameter to this function, the worse the sample pose fits to that landmark perception and therefore the less confidence the function shall return.

The calculation of the new pose estimate is the last step of the perception update. Except the new orientation estimate, all the estimation values are the weighted average of the recent calculation, which is in turn a weighted average of sample poses, and the

corresponding previous estimate value. The new orientation estimate is calculated using the new coordinate estimates and the perceptions. The new values of pose estimate are calculated from the Equations 4.19, 4.20, 4.23, and 4.24.

$$hp = f_{HP}\left(\sum_{j=1}^{N_L} PA_k^j\right) \tag{4.17}$$

$$tw = \sum_{j=0}^{N_L}\left(PA_k^j \times PSA_w^j\right) \tag{4.18}$$

$$PE_x^* = hp \times PE_y + (1-hp) \times \frac{\sum_{j=0}^{N_L}\left(PA_k^j \times PSA_x^j \times PSA_w^j\right)}{tw} \tag{4.19}$$

$$PE_y^* = hp \times PE_y + (1-hp) \times \frac{\sum_{j=0}^{N_L}\left(PA_k^j \times PSA_y^j \times PSA_w^j\right)}{tw} \tag{4.20}$$

$$\beta_i = tan^{-1}\left(\frac{PE_y^* - PA_y^i}{PE_x^* - PA_x^i}\right) \tag{4.21}$$

$$wa_i = f_{wpu1}(PA^i) \times f_{wpu2}(PE_x^*, PE_y^*, \beta_i, PA) \tag{4.22}$$

$$PE_\theta^* = tan^{-1}\left(\frac{\sum_{i=1}^{N_L}\left(PA_k^i \times sin(\beta_i) \times wa_i\right)}{\sum_{i=1}^{N_L}\left(PA_k^i \times cos(\beta_i) \times wa_i\right)}\right) \tag{4.23}$$

$$PE_c^* = hp \times PE_c + (1-hp) \times \frac{\sum_{j=0}^{N_L}\left(PA_k^j \times PSA_c^j \times PSA_w^j\right)}{tw} \tag{4.24}$$

where PE_x^*, PE_y^*, PE_θ^* and PE_c^* are the updated x-coordinate, y-coordinate and orientation of the pose estimate; and f_{HP} is a function that returns a history coefficient according to the number of percepts available.

4.4.3. Odometry Update

As the odometry update, the only necessary thing is the update the current pose estimation with the new odometry data. No more update or calculation is necessary, because other than the pose estimation nothing is used from the previous cycle of estimation.

It should also be noted that, in the case where the frequency of the odometry update is much less than the frequency of the perception update, transforming the odometry data to lower values may lead to better results since the original odometry

data is the result of the agent's motion from the previous odometry update to the current one, and this would last more than one perception update.

In Equations 4.25, 4.26 and 4.27 the new (updated) coordinates and orientation of the pose estimate is calculated.

$$PE_x^* = PE_x + \Delta x \times sin(PE_\theta) + \Delta y \times cos(PE_\theta) \quad (4.25)$$

$$PE_y^* = PE_y + \Delta y \times sin(PE_\theta) - \Delta x \times cos(PE_\theta) \quad (4.26)$$

$$PE_\theta^* = PE_\theta + \Delta\theta \quad (4.27)$$

where PE_x^*, PE_y^* and PE_θ^* are the updated x-coordinate, y-coordinate and orientation of the pose estimate; PE_x, PE_y and PE_θ are the x-coordinate, y-coordinate and orientation of the pose estimate before the odometry update; Δx, Δy and $\Delta\theta$ are the odometry data giving the change in the x-coordinate, y-coordinate and orientation.

4.5. Advantages and Disadvantages of S-Loc

In ML, for each landmark seen the probability distribution is modified accordingly, and as a result, the final probability distribution is expected to give the agents real pose. Instead of a probability distribution, a pose, which is most likely to be the actual pose according to the previous pose estimate, is used in S-Loc. This way, as it is in the ML, the pose estimate converges to the actual pose of the agent.

Considering only the most likely sample poses, S-Loc acts like a kind of ML but with a local coverage. Although it has a local coverage, it responds in a fast manner to the kidnapping problem, as the most likely sample poses could be far away from the previous pose estimate. In addition, calculating only a sample pose for each landmark, S-Loc has a much lower cost than ML.

Comparing with triangulation, S-Loc does not calculate the best estimate according to the perception of the moment, but makes the estimation in a way that it

converges to that point in a short period of time. On the other hand, the effect of the false perceptions is greatly decreased as the sample pose of such a perception would have a rather small confidence and will not play a big role in the pose estimation. This way, without decreasing the performance, the robustness is increased.

In a way, S-Loc works similar to the MCL as the sample poses are used in the same way they are used in MCL. The main difference is the selection of these sample poses. In MCL, there is a large number of pose samples, and they are populated according to their confidences, and randomly mutated for small changes. In S-Loc new pose samples are calculated for each estimation, and for each perceived landmark a pose sample is calculated. This way S-Loc becomes a much lower cost localization method with accurate pose estimation capability.

The memory used in the S-Loc increases the robustness of the system even further and the big jumps of the pose estimate are prevented.

5. APPLICATION DOMAIN

The proposed solution is the use of S-Loc localization technique together with a properly set ME module. It is applied on the Sony Aibo robots to compete in the RoboCup 2005 - Sony Four-Legged League.

The Cerberus'05, which is the project of Cerberus team from Bogazici University - Turkey, uses an implementation of S-Loc as its localization module, and ME as an alternative for the direct use of vision module outputs.

5.1. Robotic Soccer Domain

With the technological advances of the use of robots in industrial environments, scientists are often faced with issues on cooperation and coordination among homogeneous and/or heterogeneous robots and their autonomy in a workspace [10, 22]. These issues lead to the developments in multi-robot cooperative autonomous systems.

The promoters of multi-robot autonomous systems needed a model to test the theories being proposed, and to evaluate their efficacies and efficiencies. This way, they started focusing on robot soccer domain. Robot soccer makes heavy demands in many key areas of robot technology like mechanics, sensors and intelligence. It also provides a competitive setting that people around the world can understand and enjoy.

Robot soccer can be described as a competition of advanced robot technologies within a confined domain. It offers a challenging arena for the researchers who work with autonomous mobile robotic systems.

5.1.1. RoboCup Soccer

RoboCup is an international joint project which intends to promote AI, robotics, and related field [10]. It is an international research and education initiative which

attempts to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and analyzed.

Soccer game is chosen to be a central topic of research for RoboCup, aiming at innovations to be applied for socially significant problems and industries. The ultimate goal of the RoboCup project is to develop a team of fully autonomous humanoid robots that can win against the human world champion team in soccer by 2050.

In order to have a robot team to perform a soccer game properly, various technologies must be incorporated including: design principles of autonomous agents, multi-agent collaboration, strategy acquisition, real-time reasoning, robotics, and sensor-fusion. RoboCup Soccer is a task for a team of multiple fast-moving robots under a dynamic environment. RoboCup Soccer also offers a software platform for research for the simulation competitions.

Although soccer game is chosen to be a central topic, there are two more domains other than RoboCup Soccer. RoboCup initiated RoboCupRescue domain, which is another major application of RoboCup for search and rescue in large scale disaster, to specifically promote research in socially significant issues. The third domain, RoboCup Junior, is a project-oriented educational initiative that sponsors robotic events for young students.

The main focus of the RoboCup activities is competitive football, as the games are important opportunities for researchers to exchange technical information. In addition, they serve as a great opportunity to educate and entertain the public. RoboCup Soccer is divided into five leagues:

5.1.1.1. Simulation League. Simulation League is one of the oldest leagues in RoboCup Soccer, where independently moving software players (agents) play soccer on a virtual field inside a computer. Matches are divided into five minute halves.

5.1.1.2. Small-Size Robot League (f-180). In Small-Size Robot League, team consists of up to five small robots of no more than 18 cm in diameter play soccer with an orange golf ball on a field with the size of bigger than a ping-pong table. Matches are divided into 10-minute halves.

5.1.1.3. Middle-Size Robot League (f-2000). In Middle-Size Robot League, team consists of up to four middle-sized robots of no more than 50 cm diameter play soccer with an orange soccer ball on a field the size of 12x8 meters. Matches are divided into 10-minute halves.

5.1.1.4. Sony Four-Legged Robot League. Teams of four four-legged entertainment robots play soccer matches, which are divided into 10-minute halves, on a 4x6 meters field. As the robots used are SONY's AIBO, this is a software competition.

5.1.1.5. Humanoid League. This league was introduced in 2002. Biped autonomous humanoid robots play in *penalty kick*, and *1 vs. 1*, *2 vs. 2* matches whereas *free style* competitions are to be expected as well.

5.2. Sony Four-Legged League

Teams to participate in this league first go through a qualification, if they are not prequalified for their success in the previous year. Then, 24 qualified teams participate to the league, where SONY AIBO entertainment robots are used.

5.2.1. SONY's AIBO

The SONY AIBO models allowed in the Sony Four-Legged League are ERS-210 and ERS-7 that are shown in Figure 5.1, whereas the ERS-7 robots are recommended as they have a faster processor, a higher camera resolution, and they are the only models currently being sold that are permitted in RoboCup.

Absolutely no modification or addition to the robot hardware is permitted. No additional hardware is allowed including any off-board sensing or processing system. Besides the originally installed sensors on the robots, no additional sensor is likewise allowed.

Figure 5.1. SONY AIBO models: ERS-7 is on the left, and ERS-210 is on the right

5.2.2. Soccer Rules

Sony Four-Legged League has two competitions. The first one is the *Soccer Competition* [23]. At the first round robins, there are eight groups of three teams. The leaders of each group qualify for the second round robin. The seconds and thirds play an intermediate round, where the winners go to the second round robins.

In the second round robin, four teams constitute each one of four groups, where the first two teams from each group plays in the quarter-finals. Then, following semi-finals and the final game, the winner becomes the champion.

5.2.2.1. Setup of the Environment. The colors of the soccer field and the manual setup of the teams for kick-off are shown in Figure 5.2 [23]. All items on the RoboCup field are color-coded as follows:

- The field (carpet) is green.
- The field lines are white.
- The yellow goal is defended by the red team.

- The sky-blue goal is defended by the blue team.
- Four cylindrical beacons are placed on the edge of the field. The bottom part is always white. Each beacon is bi-colored and is unique with color sequence.

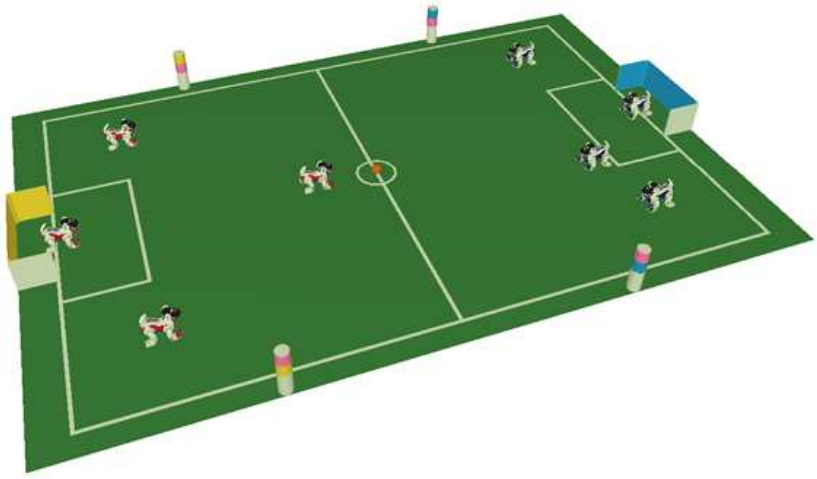

Figure 5.2. Field colors and manual setup for kick-off

5.2.2.2. Structure of the Game. Each team consists of no more than four completely autonomous robots including the goal keeper. As long as it uses the access points provided by the event organizers, any form of wireless robot-to-robot communication is allowed. GameController messages will be sent to the robots by a computer that will be provided by the event organizers. Other than this, the robots are to be completely autonomous. Any manual interaction with the robots, either directly or using some kind of communications mechanism, is not allowed. Team members can only handle one of their robots when an assistant referee gives it to them after they *request for pick-up*.

The games consist of three parts: the first half, a half-time break, and the second half. Each half is 10 minutes. The extra time over is referred to as lost time. The half-time break is another ten minutes, during which both teams may change robots, change programs, or anything else that can be done. A game can finish in a draw, if it is in the preliminaries. In the finals, penalty shoot-out employing sudden death is started after a five minute break follows any game that ends in a draw.

5.2.3. Technical Challenges

At the RoboCup 2005 Four-Legged League Technical Challenges Competition, three technical challenges were held [24]. The team that gets the highest overall score becomes the champion of the Technical Challenges. These challenges are as follows:

- The Open Challenge
- The Variable Lighting Challenge
- The Almost SLAM Challenge

5.2.3.1. The Open Challenge. This challenge is introduced to encourage creativity within the Legged League, in which teams are allowed to demonstrate an interesting research in the field of autonomous systems. Each team is expected to demonstrate their research on the RoboCup field during the three minutes of time given to them. Each team should also prepare a short, one page description of their demonstration. The teams are voted by the other entrants.

5.2.3.2. The Variable Lighting Challenge. The Variable Lighting Challenge is the second challenge which is intended to encourage teams to increase the robustness of their vision to illumination changes. It is based on a penalty shoot out where the team attempting the challenge places a single robot on the field. The team whose robot scores more goals into the opposite goal in three minutes gets more points. There are also two additional opponent robots placed on the field. Both of these robots are paused. One is placed in a goalie position whereas the other is placed in a defender position.

5.2.3.3. The Almost SLAM Challenge. The Almost SLAM challenge is intended to be a step towards moving the league away from strictly defined beacons to more generic localization information as in a soccer stadium. In order to achieve this, additional and initially unknown landmarks are placed around the field. The challenge is made of two stages. In the first stage, the robot is given time to explore the field while all existing and new landmarks are still in their positions. In the second stage, the regular beacons

and goals are covered up or removed and only the additional ones exist around the field. The robot must then move to a series of points, which were given at the beginning of the challenge, on the field using the information it learned during the first stage.

5.3. Cerberus'05

The entire software system of Cerberus was designed and developed from scratch for RoboCup 2005. The work began with developing a framework which makes all of the modules platform and hardware independent allowing the transfer from or to the robot any input, output or intermediate data of the modules. It was decided to divide the project into two main parts:

- Cerberus Station
- Cerberus Player

5.3.1. Cerberus Station

Cerberus Station is the offline development platform where the algorithms and ideas are developed and tested. The record and replay facilities are also provided which allows testing the implementations without deploying the code on the robot each time. The station was developed using Microsoft .NET technologies and it contains a set of monitors which enable visualizing several phases of image processing, localization and locomotion information.

5.3.2. Cerberus Player

The part of the project that runs on the robots is the Cerberus Player. Most of the classes in CerberusPlayer are implemented in a platform independent manner allowing the cross-compilation of them in various operating systems like OPEN-R, Windows or Linux. The software architecture of Cerberus Player consists of the following four objects:

5.3.2.1. Core Object. The main part of the player code is the Core Object which coordinates the communication and synchronization between the other objects. All other objects are connected to Core Object. It takes camera image as its main input and as a result of its processing, it sends the corresponding actuator commands to the locomotion engine. Core Object is the container and hardware interface of Vision, My Environment, Localization and Planner modules, where all of them are executed for each received camera frame. The basic input-output dependency and execution sequence is as follows:

$$Vision \rightarrow MyEnvironment \rightarrow Localization \rightarrow Planner$$

This is the basic linear illustration of the dependencies. For instance, ME module is between Vision and Localization module, but it also provides valuable information for the Planner module. The complete dependency of the modules is naturally more complex than this.

5.3.2.2. Locomotion. Locomotion object is responsible for all the actuator related operations on the robot, which performs a certain type of walk with certain parameters, a predefined special action, or sets the indicators according to the commands it receives from the Core Object.

5.3.2.3. Communication. Communication object is responsible for communication of the robot with the external entities during the games, which includes receiving game data provided by the game controller and managing robot to robot communication. Both communication mechanisms use UDP as the communication protocol.

5.3.2.4. Dock Object. Dock object is the object which manages the communication between the robot and the Cerberus Station and is used only for debugging purposes. During the games, this object is not installed on the robots. It sends the debug messages to the station and redirects the received messages to Core Object.

6. EXPERIMENTAL STUDY

Three localization modules used in the experimental study are developed and optimized to be run on actual robots of the Cerberus Team. On the other hand, the experimental data used are gathered in 2002 where the field setup was very different from it is in 2005 [13].

This leads to some changes between the test environment and the environment for which the modules are designed and optimized. In the data set used: the goal perceptions are also not available, the number of beacons and their positions in the field are different, even the size of the field is smaller. The frequency of odometry data's arrival is also higher than it is in Cerberus'05.

However, no modifications were made on the localization modules. In fact, these modules take the definition concerning the field setup from outside, and for the experiments, only these definitions are changed. They are optimized for this different environment.

6.1. Test Environment

The colors of the soccer field of the test environment are shown in Figure 6.1. The perception data contains relative positions of six landmarks which are located around the field. Goals and lines are not perceived. The field is three meters long and two meters wide. There are five points on the field that are visited by the robot in order.

The data set is recorded on this test field. Some other versions of the dataset where noise is added or the data is sparse are also used for experiments.

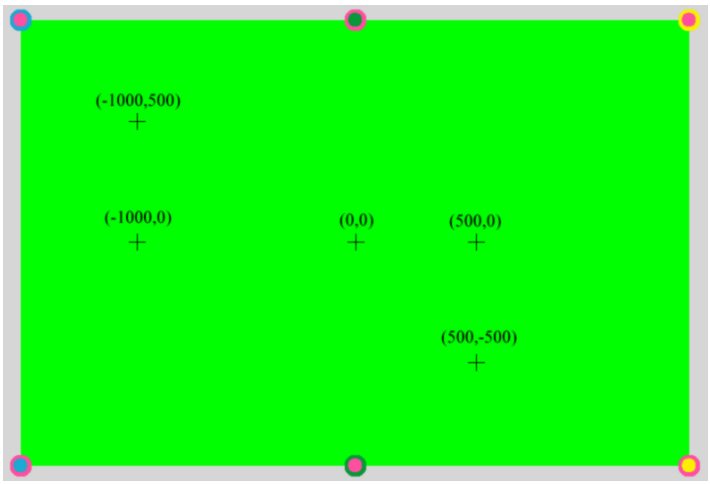

Figure 6.1. The soccer field of the test environment

6.2. Offline Testing Tool

The offline testing tool is developed during the thesis preparation for making the experimental study. It can perform four different tests on the data set using one of the available localization algorithms implemented in the Cerberus Station of Cerberus'05. This tool is embedded to the Cerberus Station. Its visual interface is as shown in Figure 6.2.

There are three display windows at the bottom of the visual interface. The first window is for monitoring the perception of the last frame. The second one shows the latest ME output. Finally, the third window shows the current pose estimation on the test field.

The tests can run on the data set in different speeds for analyzing the performance of the algorithm as well as debugging the implementation to fix it. For carrying large number of tests, this tool can also perform a given number of tests in batch mode.

6.3. Experiment 1: Robustness Test

Nine different data sets are used in the first experiment where 0 per cent, 10 per cent, 20 per cent, 30 per cent, 40 per cent, 50 per cent, 60 per cent, 70 per cent and 80

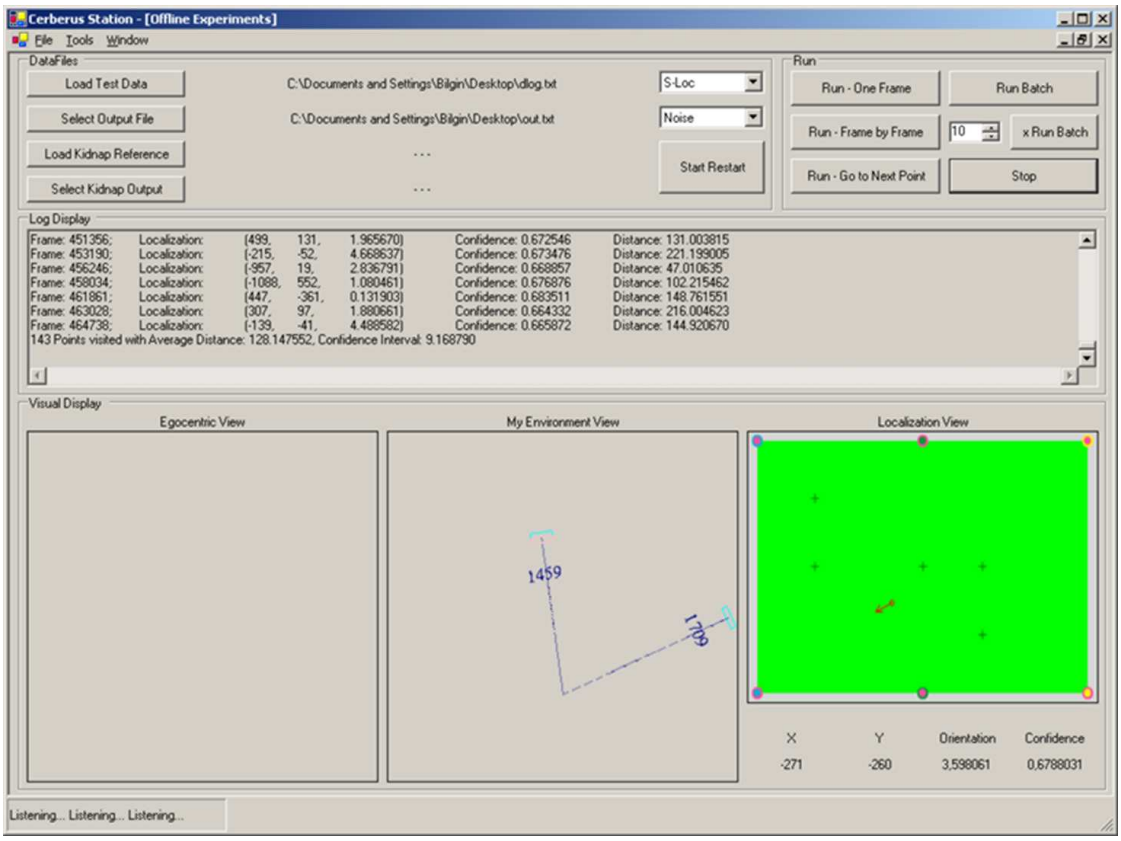

Figure 6.2. The offline testing tool

per cent noise is added to the raw data [13]. The addition of noise is done by replacing a certain fraction of landmark observations by random landmark data. The aim is to measure the robustness of the algorithms.

0 per cent noisy data set contains the raw data, where S-Loc outperforms the others. However, increasing the noise increases the error for S-Loc faster than SRL up to 70 per cent, and at 80 per cent S-Loc gives the best results again as shown in Figure 6.3 and Table 6.1. For SRL and R-MCL, the error distances are calculated over ten runs where in each run number of mark visits is 143. For S-Loc only one run is used, since no randomization is used in the algorithm.

During the design and implementation of the S-Loc algorithm, some observations are made on the actual robots and the data provided to the localization module in the actual application domain, which is Cerberus'05. It was noticed that the perception noise in the relative angle of the landmarks is much less than the noise in the dis-

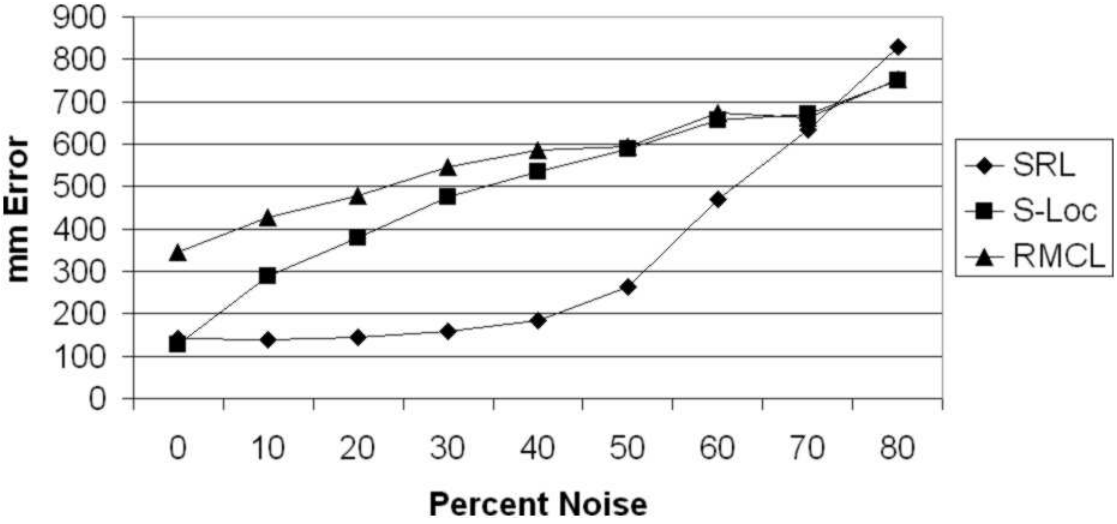

Figure 6.3. Results of the experiment 1

Table 6.1. Results of the experiment 1

Noise percentages	**Error of S-Loc** (mm)	**Error of SRL** (mm)	**Error of R-MCL** (mm)
0	128.1 ± 66.2	142.8 ± 100.5	346.5 ± 264.0
10	287.9 ± 221.0	139.7 ± 99.3	427.2 ± 368.2
20	377.9 ± 271.9	144.1 ± 111.5	477.5 ± 330.0
30	475.1 ± 318.3	159.0 ± 160.2	546.7 ± 375.2
40	535.5 ± 370.0	184.8 ± 229.1	584.5 ± 410.5
50	587.3 ± 405.7	262.2 ± 348.3	595.4 ± 415.4
60	656.2 ± 419.6	469.9 ± 493.1	673.1 ± 496.5
70	671.0 ± 453.2	633.4 ± 590.5	660.9 ± 503.9
80	749.0 ± 474.6	830.0 ± 633.9	751.5 ± 498.6

tance. The main problem with the perception was the noise in the distance estimation. Occasionally, false perceptions were also present.

It was also observed that using the ME module, the effect of infrequent false perceptions could be decreased. They are generally not a big problem for ME module, as not only the last perception but a window of the last perceptions are used to calculate the output.

Since the false perceptions are relatively rare, and having the relative angle perceptions the most reliable source of information about the environment, the S-Loc algorithm is optimized on the actual robots for depending on the relative angle data more than others.

In the way noisy data sets are prepared for the experimental study, only the false perception of the beacons is modeled, which constitutes only a small part of the noise problem and occurs infrequently [13]. Noise in the perception of the distance and the relative angle of the landmarks are not modeled. This way, only the robustness of the algorithms with respect to the false perceptions is evaluated.

6.4. Experiment 2: Sparsity Test

In the experiment 2, the accuracy of different methods are evaluated under sensor data sparsity. By discarding landmark observations from sensor data, new data sets are prepared. The test results are as shown in Figure 6.4 and Table 6.2. As in the first experiment, the error distances are calculated over ten runs for SRL and R-MCL where in each run number of mark visits is 143. For S-Loc only one run is used. The results are in millimeters.

The first data set, 1/1 fraction, is the raw data set. The other data sets are prepared using the raw data set by removing a certain fraction of the sensory information. For instance, 1/256 means that only one of each 256 sensory inputs is taken from the raw data set.

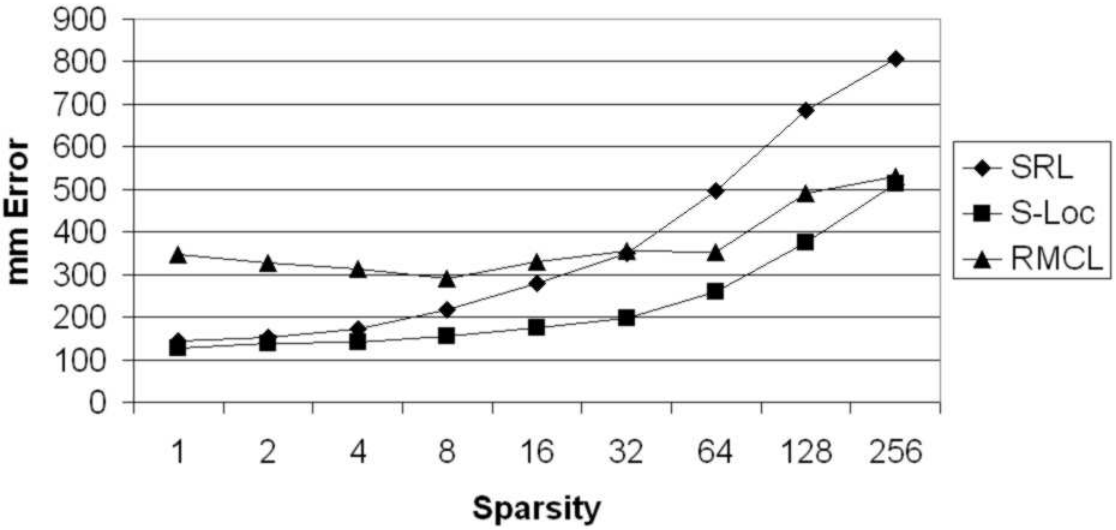

Figure 6.4. Results of the experiment 2

Table 6.2. Results of the experiment 2

Fraction	**Error of S-Loc** (mm)	**Error of SRL** (mm)	**Error of R-MCL** (mm)
1/1	128.1 ± 66.2	142.8 ± 100.5	346.5 ± 264.0
1/2	138.6 ± 69.5	151.0 ± 91.6	327.8 ± 269.4
1/4	141.2 ± 78.5	173.0 ± 109.9	313.7 ± 257.5
1/8	155.9 ± 89.1	218.0 ± 146.0	289.4 ± 232.2
1/16	175.0 ± 97.3	278.2 ± 181.6	330.8 ± 243.9
1/32	198.9 ± 107.9	350.8 ± 193.3	354.9 ± 242.6
1/64	260.1 ± 161.3	497.9 ± 298.8	352.9 ± 201.7
1/128	374.3 ± 273.0	686.8 ± 401.9	491.0 ± 370.9
1/256	513.2 ± 381.8	805.6 ± 484.1	531.4 ± 334.6

The results show that S-Loc outperforms the others in all cases. It should also be noted that S-Loc is using the ME output instead of using the perception data directly. This also has an important advantage especially against sparsity.

6.5. Experiment 3: Kidnapping Data

In the experiment 3, the ability of the methods to solve the kidnapped robot problem is analyzed. To do so, the average time the methods needed for re-localizing the robot after it has been manually displaced is computed over 22 kidnapping tests. For SRL and R-MCL, ten runs are used.

The results are shown in Table 6.3, where the results are in seconds. According to the experimental results, SRL recovers the earliest from kidnapping. It is followed by R-MCL and S-Loc.

Table 6.3. Results of the experiment 3

	S-Loc	**SRL**	**R-MCL**
Recover Time	$4.28 \pm 2.38\ s$	$2.80 \pm 2.44\ s$	$3.26 \pm 3.64\ s$

As S-Loc is a geometric method, where a large number of samples are not distributed over the field, the average time for recovering from kidnapping is longer than others, but it is still in an acceptable interval for robotic soccer domain.

6.6. Experiment 4: Running Time

In this experiment, which is carried on a Pentium 4 2.6 GHz computer with 512 MB ram, average processing time of the methods are tested. The average processing times and the number of processed frames per second are calculated for each method over the complete raw data set that contains 51523 frames ten times.

For SRL and R-MCL, the processing time interval is the average of each frame's time intervals for the motion and vision updates for localization module; whereas for S-Loc, the time intervals include motion and vision updates for the ME module as well.

As it is shown in the Table 6.4, where results are in microseconds, the total processing time for S-Loc and ME is much less than both SRL and R-MCL. S-Loc, SRL and R-MCL processed 8849.56, 1416.43 and 5291.00 frames per second respectively. The high speed of the proposed system makes is a very important characteristic especially for real-time systems.

Table 6.4. Results of the experiment 4

	S-Loc	**SRL**	**R-MCL**
Processing Time	113 μs	706 μs	189 μs

6.7. Final Discussion on the Proposed System's Performance

The experimental results show that the proposed system is a very fast system, and it gives satisfactory results even with sparse data. Because of its analytic nature with small number of perception based samples, it takes longer time to recover from kidnapping, but it is still a reasonable time compared with the other approaches.

Experiments show that the proposed system is not as robust as the other approaches used in the experiments against false landmark perceptions. However, the practical experiences show that when working on actual robots, where the noise on the perception data is different from it is modeled in the experiment 1, the proposed system is robust. The raw data set, where the proposed system outperforms the others, already contains the noise of the vision system and error made due to measurements and timing during recording it.

The proposed system is also used in Cerberus'05. In the RoboCup 2005 - Sony Four-Legged League, Cerberus Team won the Technical Challenges, where S-Loc and ME played an important role in the success.

Being the only team that is participating with the ERS-210s, the soccer competition was much harder for Cerberus Team. All of the opponents used ERS-7s with faster CPUs, higher resolution and frame rate cameras, stronger servo motors, and a new neck-head design that lets the robot control the ball much better.

7. CONCLUSIONS

Currently, localization is an active field of study and many approaches are being introduced into the literature, because it is a critical part of any autonomous mobile robotics research.

Due to the noisy characteristic of every real life environment, the localization modules of such projects need to be robust enough to handle the noisy perception data. The localization technique should also satisfy the accuracy requirements and be able to work real time with the available resources.

Robot soccer is a suitable test bed where localization techniques could be developed and tested as the robots have limited and noisy sensorial information as in the real life situation and their environment is also highly dynamic.

ME module, which is designed to buffer the perceptional information and calculate more robust estimates over them, and S-Loc technique, which is a sample based localization technique where only one sample is used for each perception data, are introduced in this work. Together, they are proposed as a new localization system, especially for the application area of robotic soccer.

The proposed system is implemented in Cerberus'05. The experimental study shows that the system is:

- very fast,
- able to work fine with sparse data,
- able to recover from kidnapping at a reasonable time,
- not as robust as sample based techniques against false landmark perceptions.

About the robustness of the system, the experiments only evaluate the performance against false landmark perception levels. However, observations on the actual robots

show that the main source of noise in perceptional data is the distance estimation. The practical experiences show that while working on actual robots, the proposed system is robust.

In the RoboCup 2005 - Sony Four-Legged League, Cerberus Team won the Technical Challenges, where the proposed system played an important role in the success. This success, together with the results of the experimental study, has shown that the proposed system has a high performance for the application domain.

As a future work, consideration of linear landmarks could also be added to the proposed system. The possibility of equivalent landmarks is also not discussed in this work. These two are necessary for the use of field lines in the robotic soccer, which is the application domain of this work.

REFERENCES

1. Hightower, J., and G. Borriello, *A Survey and Taxonomy of Location Systems for Ubiquitous Computing*, Technical Report UW-CSE Tech Report #01-08-03, 2001.

2. Buschka, P., A. Saffiotti, and Z. Wasik, "Fuzzy Landmark-Based Localization for a Legged Robot" *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)* Takamatsu, Japan, July 2000, pp. 1205-1210, 2000.

3. Saffiotti, A., A. Bjorklund, S. Johansson, and Z. Wasik, "Team Sweden", *RoboCup 2001: Robot Soccer World Cup V*, Springer-Verlag, Seattle, Washington, Lecture Notes in Computer Science Series, Vol. 2377, pp 725-729, 2002. 2001.

4. Stroupe, A. W., and T. Balch, "Collaborative Probabilistic Constraint Based Landmark Localization", *Proceedings of the 2002 IEEE/RSJ Intl. Conference on Intelligent Robots and Systems EPFL*, Lausanne, Switzerland, pp. 447-452, 2002.

5. Stroupe, A.W., K. Sikorski, and T. Balch, "Constraint-Based Landmark Localization", *RoboCup 2002: Robot Soccer World Cup VI*, Springer-Verlag, Fukuoka, Busan, Lecture Notes in Computer Science Series, Vol. 2752, pp 8-24, 2003. 2001.

6. Fox, D., W. Burgard, and S. Thrun, "Markov Localization for Mobile Robots in Dynamic Environments", *Journal of Artificial Intelligence Research*, Vol. 11, pp. 391-427, 1999.

7. Schulz, D., and W. Burgard, "Probabilistic State Estimation of Dynamic Objects with a Moving Mobile Robot", *Robotics and Autonomous Systems 34*, Elsevier, pp. 107-115, 2001.

8. Thrun, S., D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo Localization for Mobile Robots", *Artificial Intelligence*, Elsevier, Vol. 128, pp. 99-141, 2001.

9. Schulz, D., and W. Burgard, "Probabilistic State Estimation of Dynamic Objects with a Moving Mobile Robot", *Robotics and Autonomous Systems*, Elsevier, Vol. 34, pp. 107-115, 2001.

10. Robocup Organization, http://www.robocup.org/, 2005.

11. Sony Four-Legged Robot League, http://www.tzi.de/4legged/, 2005.

12. Fox, D., W. Burgard, H. Kruppa, and S. Thrun, *A Monte Carlo Algorithm for Multi-Robot Localization*, Technical Report CMS-CS-99-120, 1999.

13. Gutmann, J.S., and D. Fox, "An Experimental Comparison of Localization Methods Continued", *In Proc. of the 2002 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS'02)*, Lausanne, Switzerland, pp 454-459, 2002.

14. Maybeck, P. S., "The Kalman Filter: An Introduction to Concepts", I. Cox and G. Wilfong, editors, *Autonomous Robot Vefıcles*, Springer-Verlag, pp 194-204, 1990.

15. Fox, D., W. Burgard, F. Dellaert, and S. Thrun, "A Probabilistic Approach to Collaborative Multi-Robot Localization", *Special issue of Autonomous Robots on Heterogeneous Multi-Robot Systems*, Vol. 8, No. 3, pp. 325-344, 2000.

16. Lenser, S., and M. Veloso, "Sensor Resetting Localization for Poorly Modelled Mobile Robots", *Proc. ICRA 2000*, IEEE, Vol. 2, pp. 1225-1232, 2000.

17. Crisman, Z., E. Curre, C. T. Kwok, L. Meyers, N. Ratliff, L. Tsybert, and D. Fox, "Team Description: UW Huskies-01", *RoboCup 2001: Robot Soccer World Cup V*, Springer-Verlag, Seattle, Washington, Lecture Notes in Computer Science Series, Vol. 2377, pp 721-724, 2002.

18. Köse, H., and H. L. Akin, "Experimental Analysis and Comparison of Reverse-Monte Carlo Self-Localization Method", *Proceedings of CLAWAR/EURON Workshop on Robots in Entertainment, Leisure and Hobby*, December 2 - 4, Vienna, Austria, pp. 85-90, 2004.

19. Köse, H., and H. L. Akin, "Robots from Nowhere", *2004 Robocup Robot World Congress*, Lisbon, Portugal, Springer-Verlag, Lecture Notes in Computer Science Series, Vol. 3276, pp.594-601, 2005.

20. Köse, H., and H. L. Akin, "A Fuzzy Touch to R-MCL Localization Algorithm", *RoboCup International Symposium 2005*, Osaka, July 18-19, 2005. (Accepted).

21. Gutmann, J. S., "Markov-Kalman Localization for Mobile Robots", *Int. Conf. on Pattern Recognition (ICRP)*, Vol. 2, No. 2, pp. 601-604, 2002.

22. FIRA, http://www.fira.net/, 2005.

23. Sony Four Legged League Technical Commitee, "Sony Four Legged Robot Football League Rule Book", http://www.tzi.de/4legged/pub/Website/Downloads/Rules2005.pdf, 2005.

24. Sony Four Legged League Technical Commitee, "Technical Challenges for the RoboCup 2005 Legged League Competition", http://www.tzi.de/4legged/pub/Website/Downloads/Challenges2005.pdf, 2005.