DEVELOPING A PROBABILISTIC POST PERCEPTION MODULE FOR MOBILE ROBOTICS

by

Can Kavaklıoğlu

BS, in Computer Science, University of Warwick, 2006

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

Graduate Program in System and Control Engineering

Boğaziçi University

2009

ACKNOWLEDGMENTS

I would like to thank Prof. Dr. H. Levent Akın for making all this thesis work possible. His efforts in forming the Cerberus Team is the reason this thesis exists.

I would also like to thank Prof. Dr. Okyay Kaynak for appearing my thesis jury and his attention in my previous work during my masters education.

Asst. Prof. Taylan Cemgil has been very kindly helpful with particular parts of this thesis work. I would like to thank him for his generous help and also appearing in my thesis jury.

The captain of the Cerberus Team, Çetin Meriçli, has been an inspiring guide from the beginning as well as during this thesis. Thanks to him for his valuable advices and friendship.

Cerberus team member, Ergin Özkucur, deserves appreciation for providing us a solid infrastructure. Without his high quality work setting us an example and providing crucial tools, much of this thesis would not be possible.

All past and present members of the Cerberus Team (Barış Gökçe, Barış Kurt, Tekin Meriçli) and members of the AILab of Boğaziçi University deserve a warm thank you for providing probably the most comfortable environment for research in the whole world.

I would also like to thank Serhan Daniş for providing the visual data and data generation tools for the autonomous vehicle experiments.

Mert Cevahir has been very helpful for the installation of the overhead camera system, thanks to him for his help.

I tried to keep a list of people to thank, but this line is for all the others I have forgotten to thank, thank you.

Last but not least at all, I would like to thank my family for their support and affection during my entire life.

Finally I would like to express my gratitude and love to my dear wife Beyza for her crucial support. Life and all things shine with your peaceful essence.

ABSTRACT

DEVELOPING A PROBABILISTIC POST PERCEPTION MODULE FOR MOBILE ROBOTICS

All autonomous robots need to gather information about their surroundings. Once information about the environment is accurately extracted a great deal of further research is possible ranging from human computer interaction to high level action planning. However experience indicates achieving accurate perception can be a challenging problem.

One of the problems is erroneous perception of the unrelated features of the environment as landmarks. Due to imperfect nature of sensors, methods should be developed to compensate for the uncertainties introduced by the low level perception algorithms. Looking from this point of view, the self localization literature may be seen as an effort to resolve the uncertainties introduced by lower level perception algorithms.

This thesis proposes an algorithm to work between low level visual detection algorithms and higher level modules of a robot. The algorithm aims to select and remove erroneous perception information generated due to misplaced landmarks. Defining landmarks and *correct* locations for landmarks might sound very environment dependent at first. However this is not the case due to the generic definition of a landmark and correct location. Using the *meta pose* instead of a specific pose as the state space, the method becomes easily portable with a few alterations in the parameters of the underlying probabilistic framework. Experiments are performed in two different environments to show general nature of the proposed algorithm.

ÖZET

MOBİL ROBOTİK UYGULAMALARINDA KULLANMAK ÜZERE BİR OLASILIKSAL ALGILAMA SONRASI İŞLEME BİRİMİ GELİŞTİRİLMESİ

Bütün otonom robotlar verilen görevlerine bulundukları ortamdan algılar toplayarak başlarlar. Algılar kesin olarak bilinebilirse, insan robot iletişiminden üst seviye görev planlama gibi bir çok ileri araştırma alanı üzerinde çalışılabilir. Ancak önceki çalışmalar algıları eksiksiz ve kesin olarak toplanmasının oldukça güç olduğunu göstermiştir.

Başta gelen problemlerden birisi ortamda bulunan ilgilsiz nesnelerin işaretçiler olarak algılanmasıdır. Temelde algılayıcıların mükemmel olmayan doğası gereği oluşan, alt seviye algılama algoritmalarındaki belirsizliklerin telafi edilmesi için çeşitli yöntemler uygulanması gereklidir. Bu açıdan bakıldığında robot pozisyonlaması literatürünün tamamının alt seviye algılayıcılarının getirdikleri belirsizliklerin çözülmesi amacı taşıdığı söylenebilir. Zira mükemmel algılar var olabilseydi, pozisyonlanma en basit yöntemlerle bile çözülebilirdi.

Bu tezin amacı alt seviye görsel algılama birimleri ile yüksek seviye birimler arasında çalışarak, hatalı algılanmış nesneleri seçip ilgili algıları silebilecek bir algoritma geliştirilmesidir. Belirli bir grup nesneden ve *hatalı* algılardan bahsetmek ilk bakışta önerilen çözümün son derece ortam bağımlı olduğunu düşündürebilir. Ancak gerçek pozisyon bilgisi yerine *meta posizyon* bilgisinin kullanılması önerilen yöntemin olasılıksal alt yapısındaki birkaç parametrenin ayarlanamsıyla kolayca başka ortamlarda da kullanılabilmesini sağlar. İki farklı ortamda gerçekleştirilmiş deneyler önerilen yöntemin kolayca genellebildiğini göstermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

DARPA	Defense Advanced Research Projects Agency
EKF	Extended Kalman Filter
FPS	Frames Per Second
HMM	Hidden Markov Model
LPE	Localization Performance Experiment
MCL	Monte Carlo Localization
MPE	Mean Particle Error
PPM	Post-Perception Module
SPL	Standard Platform League
SLAM	Simultaneous Localization And Mapping
VSLAM	Visual Simultaneous Localization And Mapping

1. INTRODUCTION

A robot can be defined as an agent which interacts with its environment to achieve a specific task. Robots are ever more increasingly employed with different tasks to perform various goals ranging from assembling product parts in a factories to handling dangerous radioactive material. Some of these tasks require robots to be *autonomous*. Especially in case of autonomous robots, general plan formation has to rely on the perception capabilities, in other words to the available sensors to gather sufficient information about the environment. Only after collection of some perception data, further processing can be conducted to decide on a plausible action and then perform that action. Without any perception information in most cases, robots can not act sensibly and *autonomous* robots can not act at all since prior information can not be precise enough to accomplish any type of goals.

Often mere existence of the perception is not enough. Perception information must reveal statistically significant information about the environment. In some cases this requirement may increase sensor device cost or sensor data processing equipment cost. Such costs are unavoidable since robots simply can not act without reliable perception information.

In practice, requirements on sensor data can vary dramatically according to the complexity of the task at hand. If a robot's task is to manipulate a machine part in a factory, complex reasoning about low level object perception may a not be necessary due to the great level of control designers have on the environment. Any complex reasoning can be avoided with a good environment layout. Noise such as visual occlusions, surfaces with different reflection parameters can be avoided to achieve a satisfactory level of perception quality in such well controlled environments. Although these robots may still be autonomous, well controlled environment and immobility of robots simplify the problem great deal in contrast to more uncertain situations. For example the robotic arm of a space craft must be able to handle a much greater range of possible situations with a much greater precision and fault tolerance. The uncontrolled envi-

ronment makes using simple low level perception mechanisms very hard since effects of the environment on the sensors is not constant and not always predictable. Besides a robotic arm in a space shuttle can not be as easily replaced as the counterpart in a factory, which brings up the fault tolerance issue.

In fact, it is not necessary to go out into the space to face problems of the uncontrolled environments, most of the mobile robotics tasks include similar problems. More innocent looking requirements can easily present similarly hard problems. In the Urban Challenge[1] of DARPA participant teams use many sensors besides expensive 3D laser scanners with high energy requirements weighing over 10 kilograms. But teams still face hard problems of figuring out cars and lanes in the surrounding environment due to extensive noise present in the sensor readings. A different case is the four legged Sony Aibo robots trying to score goals on the the Standard Platform League field of RoboCup[2] with a single camera and limited field of view. In these examples numerous problems of the autonomous mobile robotics such as the following can be stated. In the Urban Challenge accommodating expensive price and processing power requirements due to the large amount of data supplied by the many sensors scanning the environment is a non trivial task. In the four legged robots case there are not many sensors but even processing the images captured with the single camera becomes a challenging problem due to the relatively low computing power available. Available on board resources may not always be plenty enough for autonomous mobile robots to achieve their tasks while handling the inherent uncertainties of the uncontrolled (or even semi-controlled) environments.

Considering the practically unconstrained nature of the uncertainties involved, it becomes apparent why the current research is directed at not reducing the uncertainties in the targeted environments but to overcome the uncertainties using probabilistic methods and act rationally given the current state of the robot and available perception information.

This thesis aims at providing another uncertainty handle to be used in autonomous mobile robotics research, which provides a prior belief over the sensor state

space. Using this prior estimate, robot will be able to distinguish between perception information which doesn't fit to the robot's expectations.

Chapter 2 provides detailed information about the background of the problem by describing related research conducted in our laboratory. Next the problem is introduced. Finally literature survey section provides some pointers to the relevant literature.

Chapter 3 describes the solution formulation in detail. First the heuristic solution is described and then the probabilistic solution.

Chapter 4 provide detailed results of the experiments performed and makes some comments about the results.

Chapter 5 comments on the overall results and provides some future work pointers.

2. BACKGROUND

2.1. Application Areas

This section describes the target environments in which the work introduced in this thesis is relevant for and provides some information about those specific areas of research. In particular the work done in our laboratory, AILAB of Computer Engineering Department of Bogazici University, is described to provide some insight before introducing the specific problem this thesis aims at solving.

2.1.1. RoboCup

One of the primary research areas of our laboratory is preparing a soccer team of robots to participate in the soccer tournament organized annually by the RoboCup committee. The goal of the committee is stated[3] follows

By the year 2050, develop a team of fully autonomous humanoid robots that can win against the human world soccer champion team.

The games are divided into four major branches and further into several sub branches for each branch. The main branches are, RoboCupSoccer, RoboCupRescue, RoboCup@Home, RoboCupJunior. There is also an academic symposium along with the games where researches exchange research ideas and present their current work.

Our laboratory participates in the Standard Platform League[2] (SPL) of the RoboCupSoccer and Rescue Simulation League of the RoboCupRescue branches. SPL team, Cerberus, has been participating in the RoboCupSoccer since year 2001 [4] [5] [6] [7] [8] [9] [10]. In our recent work, some probabilistic methods have been implemented along with some new perception implementations. These improvements resulted with a robust localization performance of our robots in games of the RoboCup 2008. Since localization plays a crucial role for our planner, which decides on the actions of each

robot, our robots were performing noticeably better than previous years.

SPL provides a well defined set of problems for researchers. The technical committee of SPL releases a rule book defining specific issues about the games including the size and landmarks of the field every year. Although the field elements are color coded, the limited camera properties of the robots, multiple robot interactions and the dynamic surroundings of the field create a great deal of uncertainty in the actual game fields. RoboCup symposium contains numerous publications about dealing with the mentioned uncertainties.

2.1.2. SPL Related Work

Particular parts of the development conducted for SPL team of our laboratory is directly related with this thesis. An overview of our current robot control design will be helpful for appreciation of the targeted problem. Figure 2.1 displays important modules of the current design.

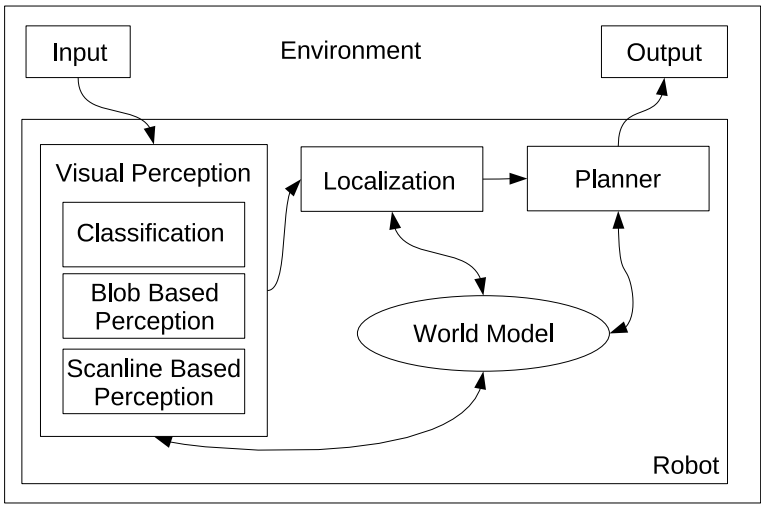


Figure 2.1. Software architecture of the current robot soccer software

As Figure 2.1 shows since the robot in question is an autonomous mobile robot, the process needs to start with an input from the environment. It is not impossible to come up with a design which does not rely on environment input, but in that case the robot will not be able to act rationally in its environment and only be able to move around the field randomly.

Once input, which is an image from the camera of the robot in this case, is received, the vision module starts processing. There are three main sub parts of the vision module. In the classification step, a corresponding *classified image* is created which represents the received image in a much smaller color space. Running further visual algorithms on classified images rather than the raw image increases overall efficiency considerably. Raw color space can have tens of thousands or even millions of possible colors, which renders complex perception algorithms computationally intractable for most mobile computational platforms. Using a classified image, which only contains around ten possible colors, perception algorithms can use higher level color names in their internal reasoning in a computationally feasible fashion.

Once a classified image is generated, higher level perception algorithms are run to perceive any available objects in the current image. In fact scanline based perception methods do not require a classified image, which reduces the time spent with classifying all of the image. Further details of perception algorithms can be found in Appendix B.

Once image processing module completes generating perception information, the results are stored in a *World Model Object*, which is shared among modules running on a robot. There are three types of World Model Objects on a running robot's memory. Perception World Model stores detected object locations relative to robot itself. Egocentric World Model stores objects locations tracked with a Kalman filter. Global World Model merges information gathered from other robots in the environment so that each robot can benefit from information collected any other robot in the environment. Further details of the overall processing design and development strategies of our robot software can be found in Appendix A.

The next step in robots' processing is the localization module, in which the robot tries to localize itself on the given map of the environment using the information available in the shared World Model Object. Currently our team uses a particle filter based algorithm at this step. Details of localization algorithms used by the team can be found in the latest team report[10].

The Planner runs as the last step to generate a higher level action command in order to direct robot's behavior. The behaviors are a set of empirically derived finite state automata, aiming to perform the most rational action at any given time. Inter-robot communication is also important for the planner module. Much more intelligent scenarios can be achieved through communication among robots.

There are also further modules not shown on this figure such as the motion module used for translating higher level commands of the planner module to low level joint configurations to be applied to the joints of the robot.

Please note that this robotic control design is not restricted to only soccer playing robots. Requirements of any autonomous mobile robot will always be the same, a perception mechanism will be required to sense the environment, a localization module will be necessary to locate the robot in the environment and a planner module must exist to respond with rational reactions to the changes in the environment. Our approach described in detail in Appendix A, meets these requirements with a portable software implementation. Thus the result of our research and development remains platform independent.

2.2. The Problem

Ongoing research always generates problems and new problems require new solutions to be implemented. This thesis searches for a solution to one such problem. In the sections below first practical considerations, which contributed to the formulation of this problem, are discussed. Available probabilistic methods are discussed next and the last section provides the formal statement of the problem.

2.2.1. Practical Considerations

Consider a hypothetical controlled experiment where a human child is brought up from infancy. In the child's linguistic education the color *orange* is taught to the child as color *green* and *orange* is taught to be the color *green*. This child will have no

way of knowing the *actual* names of the colors unless in some way the correct naming is given to him. This hypothetical experiment is a good example of a robot processing the input it receives from the environment. As the light in the environment changes color classification methods may disfunction and the robot may easily *misperceive* observed colors. If such errors occur at any point of a robot's processing, the robot may easily fail to complete its mission.

Proceeding to the next step of the visual processing of a robot, further complications are possible. In fact errors become even easier to make at higher levels of processing since the elements of the processing become more detailed and hypothetical. For example most of the time, color calibration procedures can be employed to correct the color classification methods and they can be re-used in case current parameters of color calibration fail. However if a perception module gives wrong results, for example perceive an obstacle which does not exist, then there is not much a robot can do other than becoming more confused as time passes.

Since perception information is such a critical resource for further processing, a robust perception mechanism needs a method of assessing its output. Generally *confidence values* are provided along with the perception information, so that further processes can have a handle on the uncertainties related with the perception method. Due to severe implications of perception errors only perception information with high confidence values are considered by higher levels of perception. This way it is possible to reduce some of the systematic errors in the perception methods and errors related with noise in the environment.

Unfortunately such an approach can handle only a small portion of the errors. Further heuristic methods can be employed to overcome a larger set of errors using *sanity checks*, which involve applying domain dependent constraints on the perception output so that perception information violating the environment constraints can be eliminated. If the developers spend enough resources on a thorough set of sanity checks most robotic systems can be tuned to work well enough for a set of given tasks as long as there are no drastic changes in the environment or the task definition.

However these systems can never guarantee completeness. The state space of an image is very large in terms of today's computational power. Unlike a human brain which has the power of processing very wide angle high resolution images coming from two sources and at the same time perceiving almost unlimited number of objects in these images, the robots have very limited perception capabilities. Typically the resolution of images used by robots is no more than 320x240 pixels and yet processing every pixel of this image in real time is well beyond most processing power. A pixel on an image can display 256^3 different values in the commonly used RGB color model. If the image has 320x240 pixels, then there are $(256^3)^{320*240} = 3.07 * 10^{554,858}$ possible numerically distinct images, in which case enumerating through the possible images to test the sanity check set is a non-trivial task. Consider the next available higher resolution $(256^3)^{640*480} = 8.95 * 10^{2,219,433}$, which shows enumeration is quickly out of the question as resolution increases. In fact there are methods to reduce these huge numbers down for example using a classification step, which reduces the number of possible colors for a pixel from 256^3 to around 10, which makes $(10^3)^{320*240} = 1*10^{76,800}$ states, and further methods could be introduced to reduce the size of the state space even more. However all these reductions will introduce numerous assumptions which can cause systematic errors. For example, if the classification method is not working as expected due to some reason then colors may be *misperceived*, in which case the perception problem arises in a different form this time.

In other words sanity checks on visual input can almost never be complete because the constraint set can not be fully tested due to the huge state space. The work introduced in this thesis aims to provide an alternative method to sanity checking with a probabilistic method, which distinguishes between correct and misperceived perception information.

2.2.2. Considerations on Some Probabilistic Methods

In the common general design of robotic control algorithms, the output of the perception module is received by the localization module, which aims to locate the robot in the environment. Using the confidence values and any other available information,

the localization module tries to handle the uncertainties associated with its input. Since the quality of localization information is generally mission critical, any improvement to reduce uncertainties of localization module is also generally critical.

There are numerous popular probabilistic methods used to reduce uncertainties at the localization level. The most fundamental probabilistic method is the Kalman Filter, which is primarily used to track a signal in time. Kalman Filter[11] can provide considerable improvements for noisy sensor input when its parameter are appropriately tuned according to a given problem. For example if a visual perception method tries to estimate the distance of an object, it is bound to make errors due to almost unlimited number of noise sources. Joint angle readings of the robot will be noisy, classification will be noisy, perception method will not be detailed enough, thus the final estimated object position reading will contain some amount of error for all of the readings. A Kalman Filter applied to these readings can filter out this noise from multiple readings given they are all coming from the same source. Simple Kalman Filter is limited to linear transition functions, Extended Kalman Filter (EKF)[11] relaxes this assumption. There are different Kalman Filters designed for different needs.

Another branch of probabilistic methods is the particle filters[11], which also promises a great deal in filtering noise out of the readings. An implementation of particle filters, Monte Carlo Localization (MCL)[11] is used in our laboratory with satisfactory results. The essence of particle filter based localization algorithms lies in the usage of many particles to represent a robot's estimated position on a given map. Each particle represents a particular position and orientation on the given map. The particle set is updated for each observation received and also for each odometry information received. A weight value is assigned to each available particle defining the fitness of the particle given the information available. During the update process the best particles, the particles with higher weight values, have higher probability of being chosen for the next set. At the end of the update step a new set of particles are generated with some random particles added. This method enables the robot to track its position on a given map by removing some effects of the noise in the sensor readings. Using a large set of particles many possible trajectories can be assessed at a

single update cycle. Generally if the sensors are not producing spurious readings, but only noisy output, the MCL method can nicely filter out the noise and achieve sensible localization result.

As mentioned above probabilistic filters provide very nice tools to filter out the noise from sensor readings. Unfortunately in autonomous mobile robotics noise is not the only problem related with the sensor readings. Spurious readings also pose a great difficulty in higher level processes. For example consider an autonomous car driving down a street and the car relies on an visual edge detector module to detect left and right sides of a road. If this edge detector detects a traffic light pole as a valid road edge, the car may fail to drive as expected. In case such spurious perception occurs higher modules do not have much to do. Even the probabilistic filtering techniques will most likely fail in such cases since they are primarily designed to filter out noise from readings, not to distinguish between correct and incorrect perception information. The work introduced in this thesis aims at detecting the unexpected perception information so that the filters can work with the *correct* set of readings.

2.2.3. Problem Statement

As explained in the above sections primarily due to the huge size of the visual state space heuristic *sanity checks* fail to present a complete solution and probabilistic methods employed by the localization algorithms are inherently not very suitable for spurious landmark detection. To eliminate such erroneous detection of landmarks a new post-perception module (PPM) is necessary. This module should receive perceived landmarks from the lower level perception module and apply domain specific knowledge embedded by the developers of the module to select possible and impossible landmarks. The method should be able to in some way track incoming perceptions so that upon the arrival of an unexpected perception a signal can be generated for further processing. However the method should not be using any localization information since it should be processing before the localization module.

2.3. Literature Survey

In essence this thesis aims at providing a method for improving visual perception. Visual perception literature is a diverse field of research and has a very rich literature. All levels of computational activity involving visual input needs to develop some kind of visual perception method. The result is a very wide range of publications varying from automatic ball tracking in soccer matches[12] to image guided surgery methods[13]. Most of the visual perception literature is directly related with the constraints of the environment and the goals of the system under consideration. Thus it becomes hard to locate generally applicable studies. In this section similar and inspiring works from the literature will be listed.

2.3.1. The SLAM Problem

Simultaneous Localization and Mapping (SLAM)[11] field faces a similar problem of reliably detecting landmarks in the environment. Since the primary source of information is the perception information, any inaccuracies related to it can cause huge errors in localization and mapping purposes. In addition generally SLAM algorithms' running cost grow according to the landmark set size. These facts make the landmark extraction and data association tasks very crucial steps in the SLAM algorithms. Achieving a good set of landmarks is a topic of ongoing research. In Information Management Kalman Filter method[14] the authors suggest introducing uncertainties related with the sensor readings into the internals of Kalman Filter using a probabilistic information matrix, which enables better tracking of the correct landmarks and removal of *misperceptions*. Keeping the landmark set size under control is also a possible approach since with a lower number of landmarks more processing time can be reserved for higher level localization algorithms. Using junction trees of Bayesian Network literature has been suggested[15] to keep a more meaningful and constant size of landmark set. The *thinning* on the junction tree is performed with a maximum likelihood method.

Although SLAM literature sounds close to the work presented in this thesis, the

perception methods employed (mainly laser in SLAM problems) and goals of the system (producing a map of an environment) makes the research interests quite different in the end. A branch of the SLAM literature uses visual input for landmark detection, Visual Simultaneous Localization and Mapping (VSLAM), have more similar problems with the work presented in this thesis. In [16] authors present new visual landmark detection filters, which find better landmarks by matching a database of landmarks. If more significant landmarks, such as particular shaped structures, can be effectively chosen as landmarks, overall performance increases. An effect of more robust perception is demonstrated in [16]. As more reliable perception methods become available, simpler higher reasoning may suffice for tasks at hand. For example authors of [16] use a rather simple EKF-SLAM algorithm and yet still report successful results mostly due to their improved visual landmark detection methods.

The work presented in this thesis aims to achieve similar results. For example if better perception can be made available requirements of color classification step can be relaxed. By using a more general color classification robustness against light changes in the environment will be increased.

Given that robust color classification is a research area in itself [17] the importance of high quality perception once more becomes apparent. To achieve a more general color calibration, researchers have come up with numerous techniques. Some examples include online calibration of the color calibration module [18] [19] [20] to compensate for changing environment parameters. Probabilistic methods such as Bayesian modeling have also been applied to the color classification problem [21]. Automatic calibration detecting the need for such a process is also an important research question[22].

2.3.2. Model Based Visual Perception Methods

When computer vision first become more and more available, one of the most popular methods was the model based vision methods[23], where the computational models tried to match features of observed images to a database of possible objects. Most methods first tried to extract some previously defined set of features and then

match the extracted features to an object's features. Unsurprisingly such processes were mostly used in factories or other such highly controlled environments. Handling the uncertainty were not the primary issue back then since there were neither computational power nor suitable methods developed to compensate for the uncertainties of the uncontrolled environments such as the ones we are dealing today.

The model based work has been inspiring for the design of the method used in this thesis. We know the robot moves in a particular environment and we also know some properties of the environment. If we can incorporate some of this knowledge to the robot's reasoning, impossible perception information can be highly reduced if not totally eliminated.

2.3.3. Visual Modeling

Some of the recent research aims to provide a more general environment model to improve local object detection algorithms. In [24] the authors argue that without an overall understanding of a scene, single feature detection is bound to fail. Thus their methodology tries to model the viewpoint of the camera and relative relations about the objects in the image. The results indicate integrating the model into the detection reasoning improves the overall performance of object detection.

In addition to static object detection, dynamic object detection is also possible with the recent research results[25]. Combining object detectors, a structure from motion module, 3D localization and multi-view/multi category object recognition researchers are able to detect pedestrians and cars from a moving vehicle. The most inspiring point of this research is using only images received by two cameras working in stereo, which demonstrates impressive detection of objects and localization is possible without sophisticated laser range sensors or odometry.

2.3.4. Multi Hypothesis Tracking and Data Association

Tracking targets has been an interesting research topic from beginnings of the computing. It is not surprising to see some of earliest the tracking research (even with experiments conducted on a UNIVAC 1100 computer) directed towards producing robust tracking systems for target tracking[26] in various domains such as ballistic missile defense, air defense, air traffic control, ocean surveillance, battlefield surveillance. Reasons for such research interest in tracking are not hard to find since it is a hard problem for humans. On the contrary computers can run through a greater number of sensors and make faster and more informed tracking possible. Such cases occur frequently in military scenarios where the domain is very well defined providing clear cuts between correct and incorrect decisions.

Kalman filters have been used in tracking targets with success for a long time[26]. However with multiple targets to track a single Kalman filter instance is no more a satisfactory solution since the Kalman filter aims at reducing the noise associated with a single target on a tracking measurement series. Data association appears as a greater problem as measurement gets more and more noisy. Since most probabilistic filters such as the Kalman filter, assume a Gaussian noise model, as noise characteristics deviates from Gaussian many problems arise with the associated filtering techniques[27]. Multiple target tracking is one of the problematic cases, where noise associated with a different target might interfere with other targets. Data association literature aims at characterizing sources of such complex noise patterns to produce a correct association between the sensor readings and tracking targets, Probabilistic Data Association Filter[27], Joint Probabilistic Data Association Filter[27] and Multi Hypothesis Tracking[28] can be listed to name a few such methods.

2.3.5. Hidden Markov Model

In this section details of Hidden Markov Model implementations are presented to provide a background for the proposed solution.

Hidden Markov Model (HMM)[29] [30] essentially is a probabilistic filtering method, which can be used to track various possible paths in a state space. Given a state space definition, a transition matrix and an observation model, a HMM implementation can track the incoming signal of the observed state as well as providing an expectation for the next observation state. At any time point t in the received observation sequence, HMM keeps a probability distribution over the defined state space. The peak of the distribution represents the most likely state. According to the characteristics of the received sequence there can be states with very similar expectation values. However as time progresses the model becomes more effective in tracking the expected signal.

In essence HMM implementations track all possible paths in time. As Figure 2.2 shows all states are fully connected to each other, which means all possible state transitions are tracked. Full state space tracking does not come for free. The requirement of storing all states may be intractable for very large state spaces. However with an efficient state space design it is possible to reduce requirements of the module so that the implementation becomes feasible.

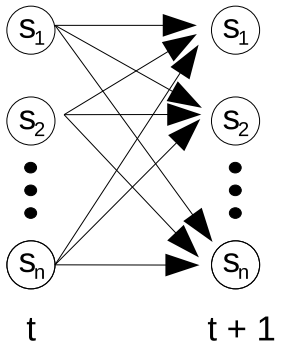


Figure 2.2. Graphical representation of state space of a Hidden Markov Model

As time advances from t to $t + 1$ each probability value of all states are updated with the transition matrix, which represents the prior information about the probability of transitions from each state to all other states. When an observation is received, the observation model, which is another matrix of probability values, is used to judge from which state this observation is more likely to be observed. Following this procedure for each observation in a given sequence, provides a probability distribution over the state

space, peaking at the most likely state at time t.

The main parameters of the model are the state transition matrix and the observation model matrix. These matrices represent how transitions in time and received observations alter the state probability vector. Matrices can be produced by experts in the application field or alternatively a learning method can be employed to learn a set of values from previous observations. The reason why hand coded parameters can work good enough is that it is actually possible to generate these parameters with some reasoning. The transition matrix represents the probabilities of transitions between states. Since the states are clearly defined by the model, we know how the transitions should occur. Similarly observations are generated according to the current state of the robot, in which case we also know what the observations should be like. Any learning method employed will be trying to learn the parameters defined by the environment we already know about. Unlike more unobservable cases where the actual model is not known but guessed.

One of the most traditional HMM learning algorithms is the Baum-Welch Algorithm[29], which is an expectation maximization based method. A more recent algorithm is called the Ensemble Learning[31], which aims to solve a few problems of the Baum-Welch Algorithm such as overfitting and uncertainty handling. Performing online learning is also possible through the online HMM learning methods such as an online version of the Baum-Welch algorithm[32].

2.3.6. Literature Survey Summary

Literature is full of various perception mechanisms working at all levels of visual processing and localization. However due to the very diverse application areas of computer vision and related processing algorithms the research areas are very diverse as well.

The work presented in this thesis is located between the lower level perception mechanisms and higher level localization algorithms. The position of the method in

the order of processing brings advantages of both worlds. Being above lower level perception mechanisms means any low perception method can be employed with the introduced method. Being below higher level algorithms increases the utility of the method as it can be used in many different environments by different higher level algorithms such as tracking methods or localization methods.

3. PROPOSED SOLUTION

The requirement statement provides a challenging problem of identifying unexpected perception generated by the lower level perception modules. The characteristics of the lower level perception modules can vary greatly and most of the times, the parameters of the modules' can be adjusted according to the environment and the task at hand. In our previous experiences we observed that the perception modules are not sufficiently reliable for the purposes of higher level modules. Thus all of our low level perception modules are configured very conservatively, where the modules report only if there is a lot of evidence for the existence of the target landmark. This configuration eliminates most (but never all) of the spurious perception with the cost of not perceiving some of the obvious cases. If we could assume a perfect environment, such as a simulation environment, the lower level perception modules could be used much more efficiently and all of the further processing modules could benefit from this better perception information available.

The module introduced below is designed to work between the lower level perception and the higher level modules to provide a method of detecting and removing spurious perception detection among the correct perception, which in turn will enable better utilization of lower level perception modules and remove the need of the error prone and inefficient process of hand coded spurious landmark removal procedures. The place of the module in the current design is shown in the Figure 3.1.

The initial heuristic design, which has been useful in formulation of the probabilistic solution, is presented in the first part of this section. Then the final design is presented in detail in the rest of the section.

3.1. Heuristic Solution Design

The initial design, shown in Figure 3.2, presents a more refined requirement analysis. The model should be able to cope with multiple perceptions of the same

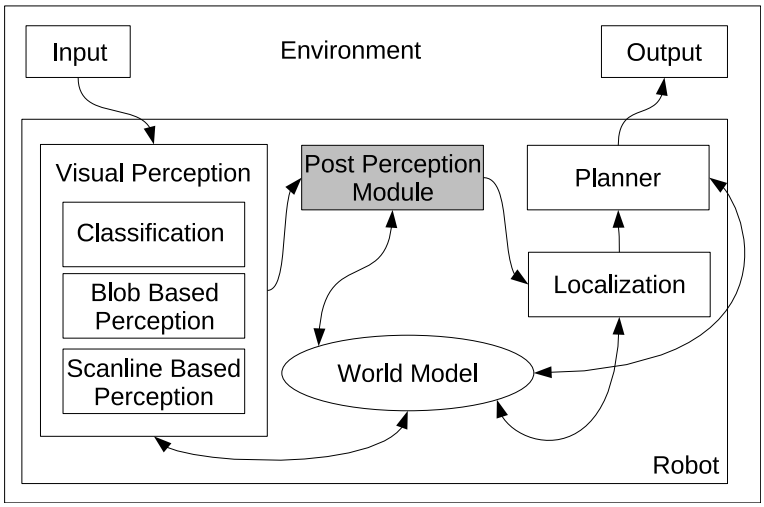


Figure 3.1. Software architecture of the current robot software with the new module added

and/or different objects. The model should also be able to handle conflicting perception information. To meet all these requirements, the first idea is placing the received perception information into a set of maps, which then can be searched for the richest map. Following paragraphs present a detailed explanation.

Upon receiving some perception information the algorithm iterates through the existing maps. If any map matches with the new perception information, the corresponding map is updated with the new perception information. If none of the existing maps match, then a new map is created and the received perception information is added. Following this procedure provides a set of maps each of which is free of the erroneous effects of the *misperceptions* since all maps are consistent within each other.

There are several problems with this heuristic solution, due to the inherent problems of the heuristic methods.

Any implementation of this formulation is bound to be a very specific implementation, which can only work in a particular environment. For instance in line 6 of the algorithm in Figure 3.2, the location of the perceived object and the location of the object in the environment is compared. The *map* in the algorithm must be defined in terms of euclidean distances, meaning any map can define a single environment for

Algorithm 1: MPPM

```
input  : PerceptionVector
output: Best map available
foreach Perception in the PerceptionVector do
    FittingMap ← −1; EmptyMap.reset();
    BestMapIndex ← −1; PruneCount ← 5;
    //Check all available maps for fitting;
    foreach Map in the MapVector do
        if closeEnough (Perception.location, Map (Perception).location) then
            FittingMap ← Map
        end
    end
    if FittingMap equals -1 then
        FittingMap ← EmptyMap; MapUpdateCountVector ← PruneCount
    end
    //Update fitting map and store empty map if necessary ;
    FittingMap.relevantPerceptionUpdate (Perception) ;
    if EmptyMap.isInitiated then MapVector ← FittingMap
end
//Prune non updated tables ;
for i=0 to MapVector.size () do
    if i not equals MapVector.indexOf (FittingMap) then
        subtractOne (MapUpdateCountVector [i]) ;
        if MapUpdateCountVector [i] equals 0 then
            MapVector [i].remove; MapUpdateCountVector [i].remove;
        end
    end
end
if MapVector.size () > 1 then
    BestMapIndex ← MapVector.mostConfidentMap
end
return MapVector [BestMapIndex ];
```

Figure 3.2. MPPM Algorithm

each implementation. Similarly observations will be defined very specifically reducing the generality of the solution furthermore. The main problem is the requirement of a map of the environment, which is a very limiting and reduces the generality of the solution.

The term *location of the perceived object* can be very dangerous since if the mentioned location is a global location, then this term assumes the existence of a decent localization algorithm, which is an unacceptable requirement for a module working before such a localization algorithm. Using relative location may be possible since this only requires the distance perception of objects. However relative locations may not provide the effect of the global locations for the described algorithm.

Although in practice processing multiple maps may not cause performance problems, since our past experience indicates the number of maps do not go beyond five in most cases, theoretical complexity is $O(p * m)$, linear in the size of perception number (p) and number of maps (m), due to the requirement of inspecting every object in every available map once for every perception received. In other the words complexity grows linearly as the number of maps increases. Pruning the unused maps and the confidence calculations of the available maps also account for practical complexity considerations but not for the theoretical complexity concerns.

3.2. Probabilistic Solution Design

After further considerations on the initial design a probabilistic solution was formulated, which does not involve limitations of the heuristic solution and has more room for further extensions. From the theoretical point of view the heuristic solution given in the initial design actually provides a subset of the HMM solution. Algorithm in Figure 3.2 generates a set of possible maps given a sequence of perception information, which are analogous to the states of a HMM. New maps are generated if a perception information does not fit any of the existing maps. The fitting map for a received perception information is updated with that perception. The problem with this approach is that the *fitness* of a perception has to be defined so that new maps

can be generated. Defining such a fitness function for assessing the new perception information is the actual problem in the first place.

In contrast with the heuristic solution, the probabilistic Hidden Markov Model can be used to track all possible maps in a more efficient manner. Once a sufficiently expressive and flexible enough state representation is found, the dynamics of the HMM can be employed to generate a smooth tracking of the states of a system. Important internal parameters of a HMM are the transition model and observation model. Sections below describe the HMM formulation used in the probabilistic solution design.

3.2.1. State Definition

The first idea for the state definition is the observations themselves. However if the state represents observations themselves it becomes hard to come up with a continuous problem formulation. Splitting the physical environment into grids can be considered, which however is not a very useful due to many limitations and not many benefits.

After further considerations on the definition of the state space, a hybrid formulation was conceived, called the *meta pose*, in which a small set of discrete values represent *all* the physical conditions a landmark can be observed. This definition removes the localization requirement of the module because we are no more interested in a specific position of a robot. Instead the system requires an indication of a meta pose definition of a set of possible physical positions. The indication is the observation robot receives. Thus all the system needs reduces to be the output of the lower level perception modules, with no dependencies to the higher level modules.

Using the *meta pose* definition, it is possible to define higher level maps, which can be used in multiple domains. Maps are defined using landmarks indicating particular positions in the environment. It is also possible to define a map using the meta pose information of the landmarks. For instance, in a soccer field two goals can be considered as landmarks. Meta pose of these goals indicate all possible physical

configurations which the corresponding goal is visible. The second goal on the field will be presenting another landmark, thus providing a second meta pose definition, representing all possible physical configurations which can result in observation of the second goal. So the high level map in this case will have two landmarks, representing each goal. It is easy to see further reasoning can be applied to the received perception information with these definitions. For example, two landmarks may not be observed at the same time due to physical limitations. It is possible to reflect this reasoning using a HMM implementation, where the states of the model indicate various landmarks on the field and their various relations.

Now that the state is independent of the physical configuration of the robot, the problem is constrained with the available perception information. This simplification also generalizes the applicability of the solution to the other domains. Since the physical configuration parameters are not considered by the system, the visual input gathered by any other system can use the proposed solution with other data for other domains.

The next step is to define the observation and transition matrices, which provide the platform specific parameters of the system. An example instance of these matrices is given in Section 4.1 and sections below describe their formulation strategies.

3.2.2. Transition Model Definition

Transition model is defined with a matrix, formed by a set of vectors. Number of vectors is equal to the number of states of the HMM. Each vector defines transitions from a single state to all other states. Defining a transition matrix requires good knowledge of the state definition. With more information available about the states and their relation with each other the better the transition matrix can be specified.

In case states are not observable and not much is known about the states of the system, transition model can also be learned with one of the learning methods mentioned in Section 2.3.5.

3.2.3. Observation Model Definition

Observation model is defined with another matrix, also formed by a set of vectors. These vectors define the observation probabilities of other observations given a state number. Similar to the transition matrix domain knowledge helps greatly to formulate the observation matrix since in practice values of the observation matrix represent the accuracy information about the sensors of the system. Similar to the case with the transition model this matrix can also be learned with a learning method.

3.3. Proposed Solution Summary

Comparing the heuristic solution with the probabilistic solution it becomes apparent that the probabilistic solution is more robust. To illustrate some of the points for that claim the paragraphs below reviews the disadvantages of the initial design.

In terms of generality the heuristic solution is severely limited with its designed environment due to its requirements related with the map of the environment. The probabilistic solution relaxes this requirement with its more general state definition, which does not require a map of euclidean distances, thus applicability in unknown environments is easier. The only parameters of a HMM that may be specific for an environment are the transition and observation matrices. Most of the time the transition matrix can be assumed to be a set of Gaussian vectors since most models will be assuming intermediate states, where the neighboring states of any state are the most likely next states. The observation matrix represents the characteristics of the sensors used, which will need adjustment in case of sensor changes. In some cases the observation matrix made of Gaussian vectors may also be used, in which case no change in the solution would occur for those environments.

The *meta pose* state representation removes any possible needs for higher level modules. Thus the solution becomes more robust with less number of dependencies to other modules.

In terms of complexity, the probabilistic solution also outperforms the heuristic solution. The heuristic solution works for each map available and processes all observations for each of these maps. The probabilistic solution consists of a single HMM update for each received observation, which consists of multiplication of a vector (current state vector) and two matrices (transition and observation models). If the state space is of size n, then the vector is of size n and the matrices are of size nxn. Since the values of the vectors and matrices are known at compile time, there is much room for optimization. In the most extreme case, all possible values can be calculated before hand to implement a lookup table for the update procedure. Therefore it is possible to perform a HMM update in constant time, which makes the complexity of the system $O(1)$.

3.4. Implementation

In this section, first the target environments are described, then the implementation details for each environment will be described.

3.4.1. Target Environments

To apply the specified probabilistic model, first the states of the system and their dynamics in the given environment should be provided. There are two environments to be used in the experiments. The paragraphs below describe each of them in detail.

3.4.1.1. RoboCup 2008 SPL Field. The primary environment for this thesis is the RoboCup 2008 SPL Field [33] shown in Figure 3.3. In our laboratory we have a scaled down version (5200 x 3300 mm) of this field with the color coded beacons and goals on corresponding sides of the field. The controlled lighting is also very useful to achieve a stable testing environment. On this field Sony's Aibo robots (see Appendix D for details) play soccer during the SPL games. Each team has five robots on the field during the twenty minute games.

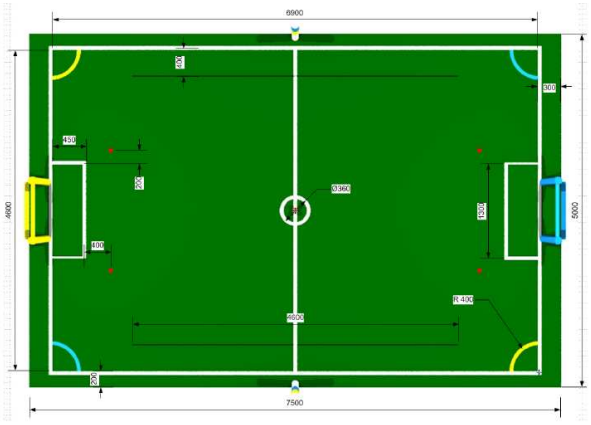


Figure 3.3. Official RoboCup 2008 Standard Platform League Field

There are many problems in this domain, where the level of uncertainty is quite high due to the robot's noisy sensors, semi controlled lighting, features introduced by the spectators, etc. The robots use the color camera located in front of their heads as their main sensor. Thus robust visual feature detection is an important research topic for this domain.

The problem we aim to investigate among the set of problems of SPL environment is the spurious landmarks generated by the low level visual detection modules. Due to the importance of the visual landmark detection in the domain, a module which picks out spurious landmarks can be very useful during the games. To create spurious landmarks a set of wrong landmarks are introduced into our scaled environment as can be seen in Figure 3.4, which are also called misperception sources throughout the text.

3.4.1.2. Autonomous Car. To show the generality of the proposed solution a second test environment is used. Data collected by the camera of an autonomous vehicle developed by our laboratory[34] is processed. Two different perception information generated by two different procedures is tested using the same log files. These experiments also provide an insight about how the complexity of the underlying perception method effects the performance of the proposed algorithm. Some details about the vehicle used is presented in the Appendix D.

The first perception method is one step above a toy solution where a simple visual

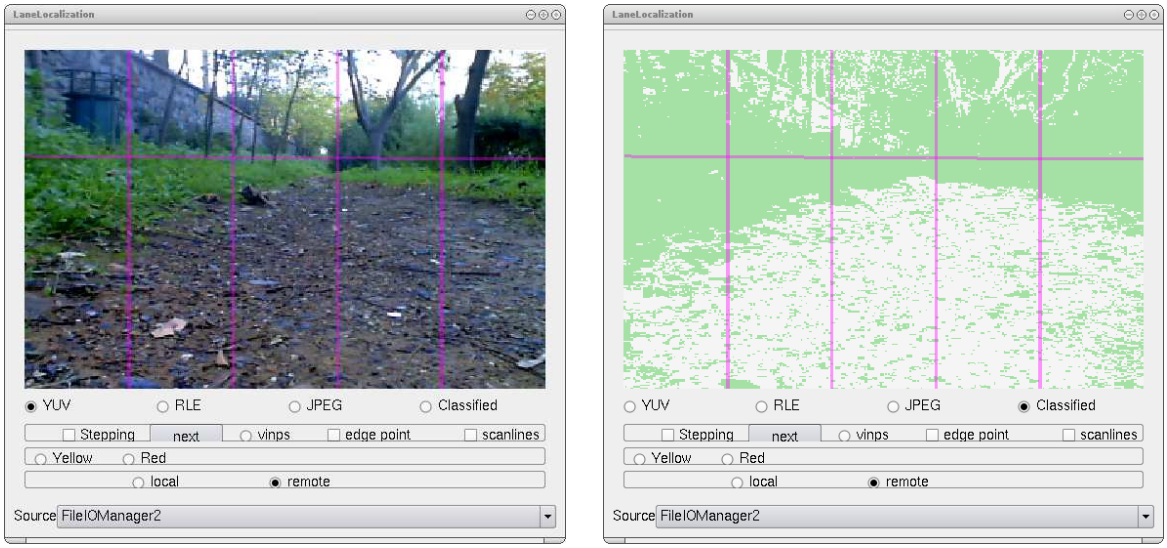
(a) Opposite colored goal around the Aibo goal

(b) Misplaced beacon at the upper right

(c) Misplaced beacon at the upper middle

(d) Misplaced beacon at the middle left

Figure 3.4. One spurious goal and two spurious landmarks are placed into the standard SPL environment



(a) A raw image (b) Corresponding classified image

Figure 3.5. Images received from the car's camera

pseudo lane detection module is tested with the proposed method. The lane detection module tries to find the center of the road. But due to the imperfect classification, in some frames the center of the road jumps to other frames, causing *misperceptions*. The proposed system uses its internal belief state to mark such a transition invalid. Example input from the car's camera is presented in the Figure 3.5. Details of the center of the road detection algorithm is provided in the Appendix B.

The second perception method is developed to be used by the autonomous vehicle itself. Since outdoor conditions are targeted, it is a much more computationally demanding algorithm [34]. Working of the algorithm can be summarized as follows:

1. The images gathered by vehicle are striped into ten columns, each of which are then divided into ten squares.
2. Each square is processed with several visual algorithms and a quantitative result, called *energy*, is achieved representing the corresponding square.
3. Using the hand labeled distance data as supervision, a linear regression is performed to match the distance supervision data with each stripe's energy.
4. In the run time, the trained linear regressor is used to match the energy of each stripe in the received image.

5. Output of the linear regressor is used to drive the autonomous vehicle's steering.

The target of the proposed method in this case is to smooth out the spikes of wrong readings of the linear regressor, which are analogous to divergence of the particles in MCL of the Aibo experiments. These *misperceptions* may occur due to many sources of inherent uncertainty of the outdoor environment and also due to the assumptions of the perception method.

3.4.2. HMM implementation

The HMM implementation is based on the general HMM implementation (in Matlab code) presented in the lecture notes of the Bayesian Statistics and Machine Learning course [30] of Computer Science Department of our university. The implementation is modified to fit for the purposes of this thesis. The code is tested using visual detection logs collected from an Aibo robot. For further experiments to be conducted on the Aibo robot and also to be able to conduct faster batch experiments, a C++ implementation of the HMM code was written and included among the BOUNLib architecture.

The C++ implementation working on the Aibo robot, works after the lower level visual landmark detection algorithms and removes the perception information related with the unexpected states, which are defined as the states with probability values less than a set threshold. In other words the true positive results of the system indicate the misplaced landmarks on the field. The false positive results are the deleted landmarks, which are actually correct.

The implementations are flexibly designed so that they are easily configurable for other environments. The number of states, transition and observation matrices are easily changeable in the Matlab implementation and in C++ implementation they are read from configuration files so that even recompiling the code is not necessary.

The code base assumes only a single perception may occur in a frame. However

in practice a single frame may have multiple perception information on it. To resolve this problem, in case multiple perceptions arrive at the same time, the HMM is updated once for each perception information. This solution does not bring any practical computational load to the implementation because the HMM update is already a very simple computation. A simple test shows 850 updates takes less than a millisecond on an Intel T8100 processor computer.

4. RESULTS

A series of experiments have been conducted on two different platforms to show applicability and generality of the proposed method. Sections below describe the extensive set of experiments that were performed with the Aibo robot and a few simple experiments on the autonomous robot environment and their results.

4.1. Experiment Setup

4.1.1. Aibo Experiments

A series of experiments with the Aibo robot were conducted. In the first set of experiments performance of the proposed module was tested with its effects on the number of misperceptions. In the second set of experiments effects of the proposed module on the current localization algorithm was tested.

4.1.1.1. Hidden Markov Model Parameters (Expert Coded). As described in the proposed solution, the states of the implemented HMM are chosen to be *meta states* on the field. Each meta state provides an indication of all possible physical positions, from which a specific observation can be observed. In the Aibo field the selected landmarks are two beacons and vertical bars of the goals, numbering is shown in Figure 4.1. The seventh state represents the meta state in which no observation is received.

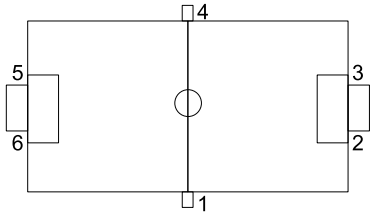


Figure 4.1. Numbers of the meta states chosen from landmarks on the field.

Transition Matrix Formulation: Columns of the transition matrix, shown in Table 4.1, correspond to the individual states of the HMM. Thus column 1 represents the first state of the HMM and transitions from this state to all other possible states. The formulation of the transition matrix introduces a number of assumptions, which may be the starting point for further research. The following paragraphs describe these assumptions.

Table 4.1. Transition matrix used by the Aibo post-perception module

	1	2	3	4	5	6	7
1	0.32	0.16	0	0	0	0.16	0.14
2	0.16	0.32	0.16	0	0	0	0.14
3	0	0.16	0.32	0.16	0	0	0.14
4	0	0	0.16	0.36	0.16	0	0.14
5	0	0	0	0.16	0.36	0.16	0.14
6	0.16	0	0	0	0.16	0.32	0.14
7	0.5	0.5	0.5	0.5	0.5	0.5	0.14

The cells with the value 0 are taken to be 0.0001 in the implementation for all matrices so that the probabilities will not get stuck with the value zero in them due to the multiplicative behavior of the number zero.

State transitions are assumed to have a Gaussian probability distribution. Whenever the robot thinks it is observing a landmark, the next state is assumed to be around the previously observed state with a Gaussian probability distribution peaking over the previously observed state. The only exception is the seventh state, which is the next state with a probability of 1/2 at all times. This value is set after observing sample log files gathered from the robot, in which half of the images have no perception information in them. This is partly due to the strict conditions set for the lower level perception modules, trying to avoid misperceptions.

The transition probability between all states, except the seventh state, is dis-

tributed with a Gaussian probability distribution with standard deviation of 1. The mean of the Gaussian function changes for each state, making sure the Gaussian distribution peaks over that state. The Gaussian is always centered on the current state when a transition will occur. The values of the Gaussian function are decreased with the probability given for the seventh state, so that the sum of the all possible states will sum up to one. However, since we do not want negative values, some values are rounded up to a value close to zero. Even though the transition values may not sum up to one exactly, the normalization performed at some stages of the run time takes care of this problem.

Gaussian transition probability distribution is a reasonable assumption, which comes from the observation of the map of the environment. When a landmark is perceived, observing that landmark and the landmarks beside the perceived landmark becomes more probable in the next state. This assumption can be used in other domains as well, where the states are expected to be observed in an ordered fashion such as the Aibo robot's environment.

If the current state is the seventh state, which means there were no observations in the current state, it is not easy to make a guess about which state may be the next state. So the seventh transition matrix column has a uniform distribution. This is also an assumption, further work may provide a dynamic function to work in this column, which may change the values of this column according to observations in a specified time window.

Observation Matrix Formulation: Table 4.4 shows the observation matrix used in the experiments. The rows provide information about the observations received in the corresponding state. For example, the second row provides information about the second meta state, which corresponds to the right yellow goal bar. In other words, the value in the second column in the second row of the matrix states the probability of being meta state number 2, when a right yellow goal bar is observed. As can be seen in the table, the diagonal values are all the same. Another example follows, the first value in the second row states the probability of being in meta state number 2,

when a right yellow goal bar is perceived on a right beacon. The matrix contains all combinations of possible misperceptions.

The formulation of these values are made using the observations and prior expertise on the subject. Our goal perception module can rarely report goal bars on beacons thus, all such values are given a value of 0.15. The other misperceptions are defined similarly.

Just like the transition matrix, the seventh state requires special treatment since it is not trivial to make any judgment about the received perception information when current state is the seventh state. Any perception may be or may not be correct when the system is in the seventh state, because the robot is not in any of the physical meta states indicating a prior position of the robot.

The seventh column in the seventh row is lower than the other diagonal values. The experiments indicate that if it is given any higher value, the large number of images without any perception quickly change the current state to the seventh state. To avoid this situation and make the effects of the received perceptions persist longer in a given sequence, the system believes receiving no observation to be an indication of the seventh meta state more slowly.

Table 4.2. Observation matrix used by the Aibo post-perception module

	1	2	3	4	5	6	7
1	0.5	0.12	0.12	0.01	0.12	0.12	0.01
2	0.15	0.5	0.15	0.15	0.01	0.01	0.03
3	0.15	0.15	0.5	0.15	0.01	0.01	0.03
4	0.01	0.12	0.12	0.5	0.12	0.12	0.01
5	0.15	0.01	0.01	0.15	0.5	0.15	0.03
6	0.15	0.01	0.01	0.15	0.15	0.5	0.03
7	0.125	0.125	0.125	0.125	0.125	0.125	0.25

Threshold Formulation: The final parameter of the HMM is the unexpected state threshold, which defines how much probability is required for a state to be an expected state. With the above parameters the value of 0.1 was found to be appropriate.

4.1.1.2. Hidden Markov Model Parameters (Learned). To test the performance of the learning procedure a set of HMM parameters are learned with the Baum-Welch Algorithm[29]. Data required to run the learning algorithm is collected in a simulation environment. The Aibo robot performs the same behavior in the simulation environment as if it is in the real soccer field. During the actions of the robot received perception information is recorded. The robot is randomly placed in predefined intervals so that a diverse set of perceptions are available. The final set of collected perception data has 65019 number of perception information, half of which is used as training data and second half is used as validation data. Once parameters are learned they are tested on the real world data. Tables in this section show the learned matrices, which do not look like the expert coded matrices at all. The threshold is coded by the expert after several test with the real world data, which is 0.000001.

Table 4.3. Learned transition matrix

	1	2	3	4	5	6	7
1	0.97	0	0.52	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0.02	0.55	0	0	0.9	0.02	0
6	0	0.08	0.48	0	0.02	0.97	0
7	0.01	0.37	0	1	0.04	0.01	1

4.1.1.3. Misperception Elimination Experiment. The first test of the introduced post-perception module is performed while an Aibo robot gets into a specific position on

Table 4.4. Learned observation matrix

	1	2	3	4	5	6	7
1	0	0	0.02	0.98	0	0	0
2	0	0.1	0.15	0.03	0.26	0.32	0.15
3	0.06	0.37	0.35	0.05	0	0	0.16
4	0	0	0	0.05	0.89	0	0.06
5	0	0	0	0	0	0	0.99
6	0.98	0.01	0	0	0	0	0
7	0	0	0	0	0.36	0.63	0

the field. The planner code [10] running during this experiment is the planner code used during the RoboCup's SPL games. The player goes to its position on the field using its localization information. Figure 4.2 shows the movement of the robot on the field during the experiments. The collected data is assessed for false positive and true positive values of the system.

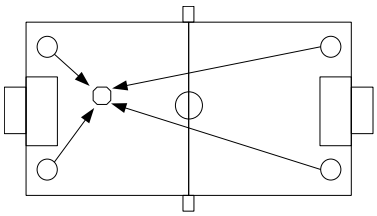
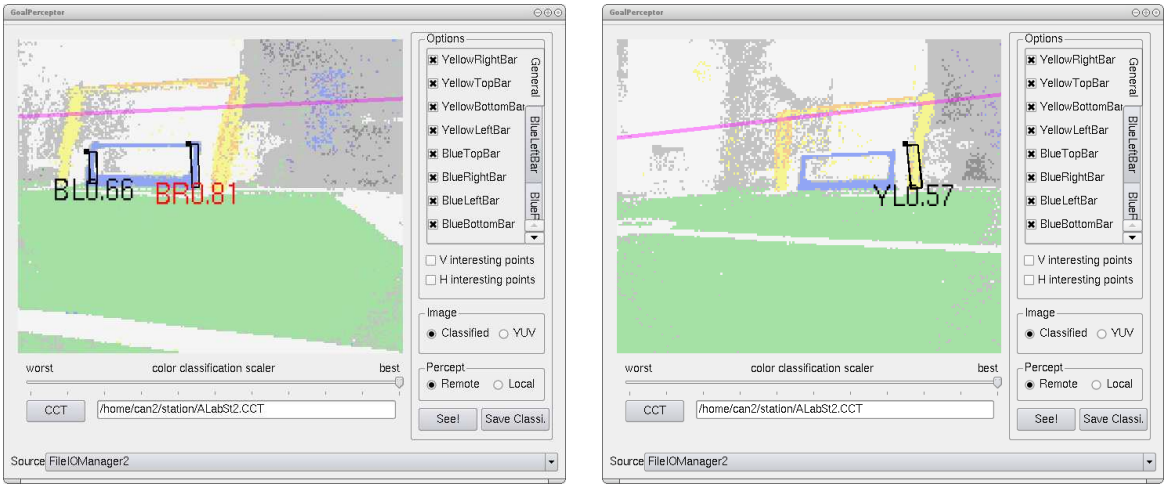


Figure 4.2. Circles indicate the starting points of the robots. The octagon shows the target of the robot in the experiments.

As described in the Section 3.4.1.1, there is a misplaced yellow goal over the blue goal in the field, which causes misperceptions. The aim of this experiment is to show that the proposed system is able to remove the misperception information generated by the misplaced goal.

During this experiment visual data is also collected. Using the visual data it is possible to exactly mark the misperceptions, which serves as a very good measure of the perception quality. Figure 4.3 shows some sample images. The classified images are viewed in the goal perception testing plugin, which is described in Appendix B.



(a) Image with no misperceptions (b) Image with a misperception

Figure 4.3. Classified images from the Aibo robot's memory during the misperception elimination experiment

4.1.1.4. Localization Performance Experiment. Another method of assessing the performance of the proposed module is to see the effects on the localization algorithm. Since localization is a key module of our robotic control system, any positive effects will be very valuable. Figure 4.4 shows the path the robots take during the experiments. This path is particularly selected to be towards the misplaced goal landmark because with no misperceptions the effects of the module is not observable.

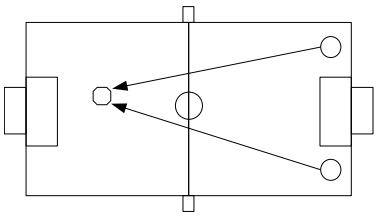


Figure 4.4. Circles indicate the starting points of the robots. The octagon shows the target of the robot in the experiments.

Figure 4.5 show the pictures of the Aibo robot in the two starting positions. For each experiment the robot goes to its position from its relative starting point. Once the robot gets to its target, the robot is stopped and logs are collected. This process is repeated five times starting from both starting locations. Thus for each experiment there are ten sets of logs.

Figure 4.5. Aibo in starting positions for the localization performance experiments

In this experiment the overhead camera system (described in detail in Appendix C) developed as a part of this thesis is used to provide the ground truth location of the robot as supervisor input. The success of the localization algorithm is compared against this ground truth value supplied by the overhead system.

Different environment and system parameters have been tested with this setup. Two sets of misplaced landmarks were tested. With the misplaced yellow goal, two misplaced landmarks are introduced, the misplaced right and left yellow bars. This setup was the primary setup. To test the system with a grater number of misplaced landmarks, two more beacons were placed in wrong positions increasing misplaced landmark number from two to four.

After some initial experiments it has been observed that if the robot tries to localize by running the post-perception module its behaviors might change, which results in different experiment results. Thus experiments with both the post-perception module and without it have been conducted. In cases where the robot the post-perception module was disabled, the results were obtained by running the localization module offline using the log files recorded during the execution.

Another parameter is the sanity checks, there are two sets of sanity checks on the current Aibo player code. One set of sanity checking code checks, called *egocentric checks*, observation of the beacons around goal bars. The other, called *seen twice*

checks, makes sure a perception is received twice before it is used for purposes of the localization algorithm. These algorithms are also experimented with using the offline debugging environment.

4.1.2. Autonomous Car Experiments

To show the generality of the proposed a method a set of experiments on a different platform is also performed. The vehicle was introduced in Section 3.4.1.2. A simple perception is developed to find the center of the road with the tools available in our laboratory. The details of the perception is provided in Appendix B. This perception provides a measure of the center of the road in terms of the five consecutive states, which can be seen in the figures of the Section 3.4.1.2. A second set of experiments are conducted with a more complex algorithm designed to run in a greater variety of outdoor environments[34].

4.1.2.1. Hidden Markov Model Parameters. In the first set of experiments the vehicle can be in one of the five available states, which can be regarded as *meta pose*. Using the meta pose generalization allows us to efficiently discretize the state space with only five states representing all possible physical orientations of the robot in relation to the target lane. Once we have a meta state definition, the proposed system can easily be applied to the system with the appropriate parameters.

Transition Matrix Formulation: The transition matrix parameters can be the same as explained in the previous section. Since transitions between states are expected to occur continuously, the Gaussian assumption holds in this environment as well.

Only exception is the final states of each side of the state vector because unlike the soccer field, the state space of the car experiments do not complete a cycle. Each side of the vector corresponds to the each sides of the camera image without any connections to the other side of the image.

Observation Matrix Formulation: In this case due to lack of experience in the domain, the observation parameters can not be set a specifically as in the previous section. Thus the observation matrix is also chosen to be the same as the transition matrix, once more following a Gaussian model, which assumes when an state observation is received the peak is the corresponding meta pose. But probability of observing the other state decreases with the increasing distance from the mean of the Gaussian. In other words only neighboring states are possible observations once an observation is received from the environment and further away states have low probability. Further work on the perception mechanism would reveal more information about the possibilities and a better observation model can be achieved.

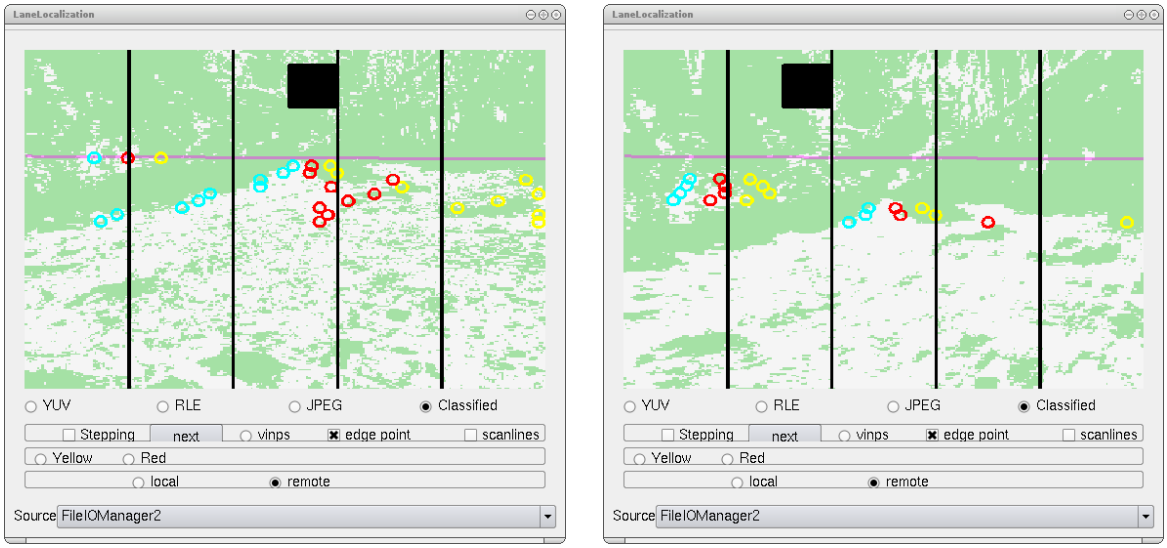
The images provided by the vehicle indicate some sudden deviations of the road center at times due to classification errors as seen in Figure 4.6. The proposed system catches some of these deviations. Since this experiment only tries to show generality of the proposed method, no further optimization attempts have been conducted to achieve better results. Also the supervision is not as successful as the previous experiment, since it is done manually using the received images.

In terms of the HMM parameters the second set of experiments are completely analogous to the first set of experiments with the only exception of the state number. Since the more complex method has ten stripes on the image, the HMM also has ten states. The transition matrix and observation matrix are only scaled accordingly.

4.2. Experiment Results

4.2.1. Misperception Elimination Results on Aibo Platform

For this experiment four set of logs were collected. Table 4.5 reports results of five trials. Figure 4.2 however, has four starting points. The reason for an extra trial is due to a localization failure in one of the trials. In experiment number 4, the trial with the highest number of misperceptions, the localization algorithm lost its position at a point, thus the robot was unable to go to its place. Thus that trial is repeated.



(a) Correct road center detection (b) 3 frames later, misperceived road center

Figure 4.6. Effects of the classification errors in the autonomous vehicle experiments

This situation once more showed how disastrous can the effects of the misperception can be.

Table 4.5. Results for the misperception elimination experiment

	Tr. 1	Tr. 2	Tr. 3	Tr. 4	Tr. 5	Avg.
Number of Misperceptions	4	2	19	26	23	15
Number of Misperception Reports	77	91	102	118	103	98
Frames With Observation	392	463	340	383	351	386
Frames With No Observation	318	346	517	487	492	432
Number of Total Frames	710	809	857	870	843	818
Misperception Elimination	50%	100%	89%	81%	78%	80%

The very low number of misperceptions in trials 1 and 2 is due to the starting points of the robot. The robots in these cases start from the sides of the blue goal, which has the misplaced yellow goal over it. Since the robots face away from the misplaced landmark, the number of misperceptions is quite low in these cases. Whereas in the other cases, the number of misperceptions increases, as the robots start the experiment from the other side of the field, facing the misplaced goal most of the duration of the

experiment.

The misperception report numbers, which indicate the false positive results, are quite high. These results indicate the cost of running the proposed method to be losing a portion of the perception information. There are a few reasons behind this cost, which may be dependent to the environment. We discuss below some of these reasons.

The most influential one is the number of frames with no observation. Due to a large number of frames with no perception information, the HMM goes to its seventh state, in which it can not believe any of the incoming perceptions. The first few of perception information is discarded after having no observations for some time. Considering our current sanity checking algorithms require at least two consecutive perceptions of the same landmark, some improvement may be achieved. For example consider this scenario, the right bar of the goal is perceived, then left bar of the goal is received. The current sanity checking algorithm will need two consecutive perceptions of each of those landmarks. However the proposed method may not require a second perception of the second landmark due to the connection the HMM makes between these physically neighboring landmarks.

Another reason for the high number of false positives is due to the very constrained lower level perception modules, which do not report landmarks unless they are very sure of their existence. If these constraints are relaxed, a greater number of perception information may be available, reducing the number of the seventh state problem. If the proposed module can handle the misperceptions well, the using the relaxed lower level modules may result with a better performance.

The overall results are promising. 80 percent of the misperceptions are removed with the cost of 22 percent percent of the perception information over a total of 4089 frames. If we consider frames with no observation valuable, then the ratio of false positive results decrease to 10 percent.

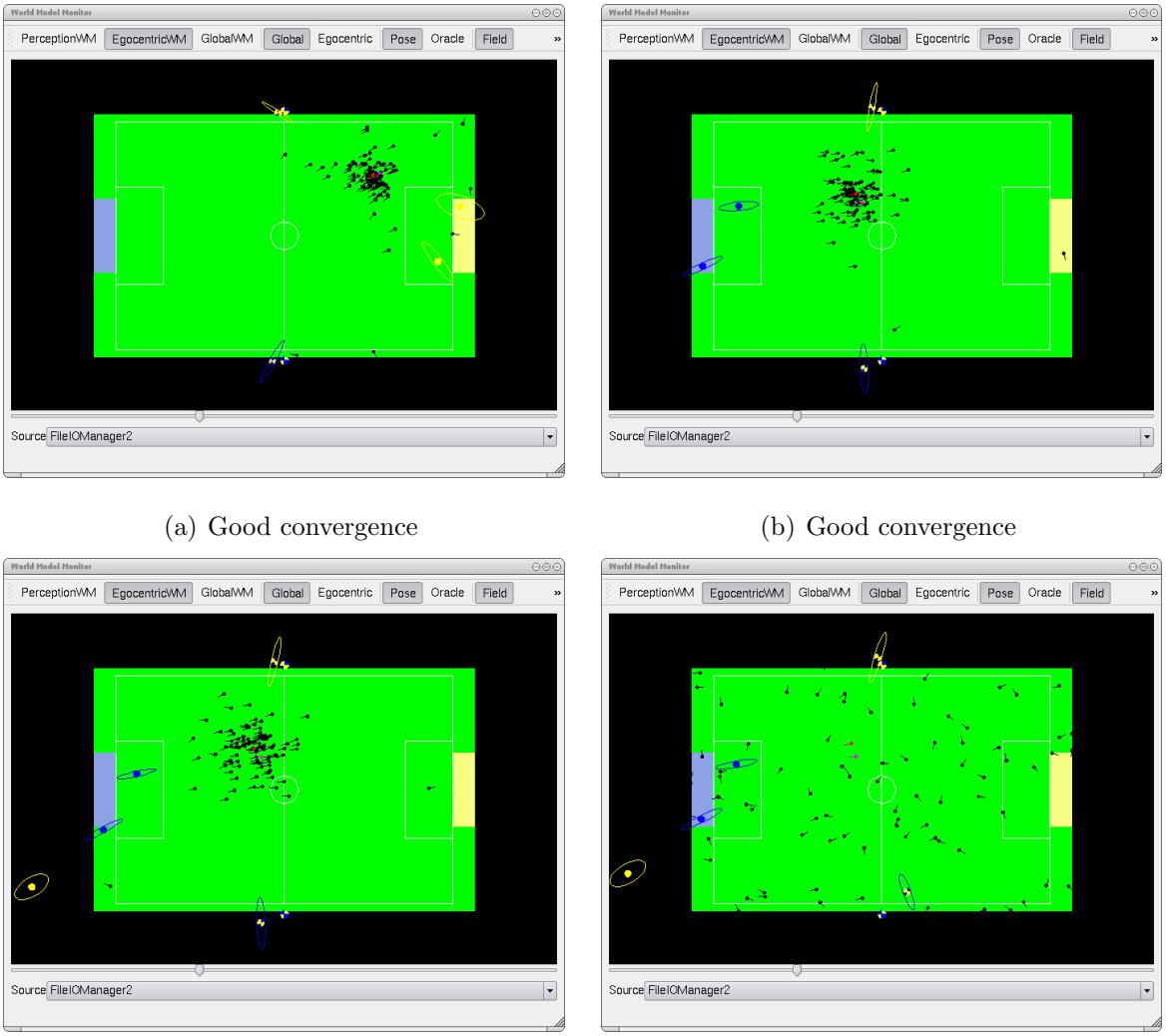
4.2.2. Localization Performance Results on Aibo Platform

4.2.2.1. Effects of The Proposed Module on the Localization. The current localization used in our Aibo player code is a Monte Carlo Based Localization[10] [35] algorithm. There are 100 particles each of which estimate the pose of the robot. The algorithm uses odometry information gathered from the robot's internal sensors and the visual perception information gathered from the environment to propose an estimated localization information.

The primary effect of the proposed post-perception module is removing the misperceptions from the localization input. In cases where misperceptions appear consistently in the input of localization, the particles diverge and the localization algorithm only uses the odometry information, which can be sufficient enough to provide a good estimate for some time. Generally the correct perception comes before too long and overall a good estimate of the localization information is available to our robot.

Figure 4.7 shows the phases of particles during a typical experiment, where the robot starts from a corner of the field facing the misplaced yellow goal. In distant parts of the field the misperception does not appear and the particles converge fine as seen in Figures 4.7(a) and 4.7(b). In Figure 4.7(c) one of the bars of the yellow goal appear as a misperception and nine frames later (around 0.27 seconds) the particles diverge as seen in Figure 4.7(d). In the figures the red point shows the pose estimate of the robot and the pink point shows the ground truth supplied by the overhead camera system.

The Experiments show that the proposed system can remove such misperceptions and help the particles of the localization algorithm. Indeed the second version of this very same log file with the post-perception module working does not have the misperception show in the Figure 4.7(c) and 4.7(d). The effects of the proposed system on the particle error is demonstrated better with a time graph. In Figure 4.8 the average particle errors of each frame are shown for the corresponding log file. In Figure 4.8(b) all particles are included into the average. However due to the random resampling step of the localization algorithm, taking a smaller subset of better particles, such as the



(a) Good convergence

(b) Good convergence

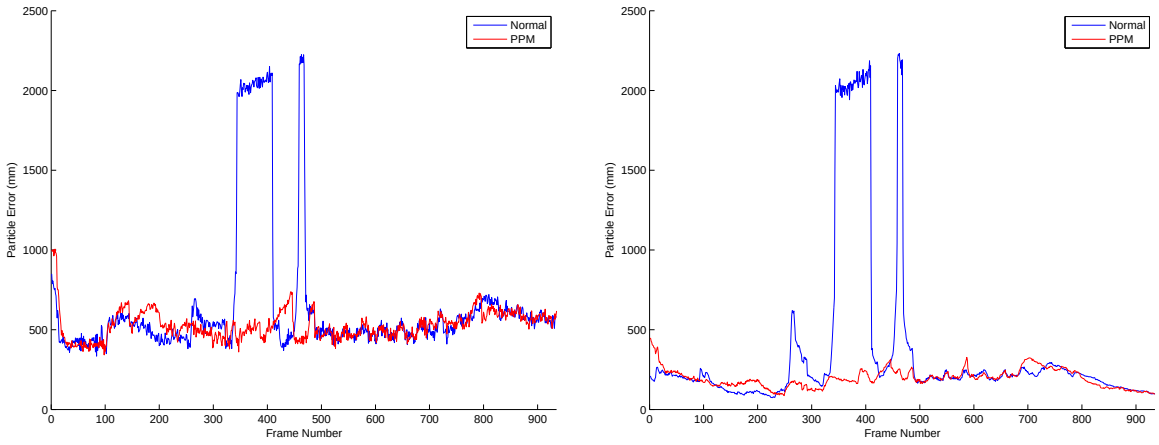
(c) First misperception frame still has good convergence

(d) Convergence is lost after nine frames

Figure 4.7. Effect of misperception on the localization algorithm

best 50 particles, provides a better graph with the same primary properties.

In the experiments the robot was observed to be able to act reasonably even though particles diverge. This is due to a feature of the localization algorithm, which stops the vision update on the pose of the robot in case the particles diverge. With this feature the robot may still have a good estimate of its position on this field using the previous vision updates and the odometry information available. Figure 4.7(d) shows this feature. Although the particles diverge, the red dot is still close to the pink dot. Thus even though mean particle error increases greatly the final pose estimate does not get very large.



(a) All particles (b) Best 50 particles

Figure 4.8. Particle errors for log number 2 exp. 2, averaged over 10 runs of all logs

After explaining the effect of the proposed module, the sections below provide results from batch experiments performed. Six different sets of data are collected to fully understand the effects of the module. Then offline experiments are performed on the log files gathered from these experiments. Table 4.6 gives parameters of the experiments.

Table 4.6. Experiment parameters

	Exp. 1	Exp. 2	Exp. 3	Exp. 4	Exp. 5
Misplaced landmarks	2	4	2	2	2
Proposed method on robot	off	off	on	off	on
Active localization	on	on	on	off	off

The robot's behavior on the field changes dramatically if it is performing active localization or passive localization, thus experiments testing the results of this behavior is also conducted. In active localization, the robot only tries to improve its localization by constantly looking around, trying to observe landmarks. In passive localization, the localization algorithm still runs but when the robot is actually performing another task such as chasing a ball, the number of observed landmarks decrease.

Similarly running the proposed method on the robot seems to result in behavior changes as well. Thus experiments have been conducted with this parameter on and off. As expected number of misplaced landmarks greatly change the outcomes.

The five set of data is extensively tested. In the cases where the proposed method is not run on the robot it is possible to observe the effects of the method with the offline testing tools. Thus a series of results are collected with each data set. The available parameters were *egocentric checks*, *seen twice checks* and the number of particles.

Using the offline experiment setup, it was possible to measure the effects of the stochasticity of Monte Carlo based localization algorithm. Due to the stochastic nature of the algorithm, it gives different results each time it is run, thus to report a meaningful improvement the effects of stochasticity is also required to be measured. To measure the effect, each of the offline experiments, where the measured localization results are generated, are conducted ten times and the variance of the results are reported.

The sections below provide results for each of the localization performance experiments (LPE). All error values are in millimeters.

The *standard deviation of pose error* and *standard deviation of particle errors* values indicate the standard deviation of the mean values of pose error and particle error respectively over the 10 runs of the stochasticity tests. To show any benefit of the proposed method, the results must be better than the reported standard deviation values. Otherwise they might just indicate a lucky run of the MCL process instead of a benefit of the proposed method.

The mean error values are calculated over all frames of all logs of the experiment with runs for both the proposed module working (the *With PPM* column) and not working (the *Normal* column).

The mean particle errors are found by calculating the error of best particles in a frame and averaging the error values, providing a single error value for the whole log

file. Then mean error values for all logs are averaged to reach a combined value for the experiment.

The *mean pose error std deviation* values show the standard deviation of the error of the estimated pose of the robot over all generated runs of the logs for this experiment.

The most expressive value for the experiments is the *mean particle errors std deviation* rows, which best represents the particles diverging. As explained above since the localization algorithm may stop using the particle information if particles diverge, and continue only using odometry information, the mean pose errors do not always indicate the effects of the proposed module.

The experiments were also conducted with the egocentric checks on. However the results do not show a reportable difference. So those experiments are not reported.

4.2.2.2. Experiment 1. In this experiment the robot performs active localization with two misplaced goal bars. The collected logs are analyzed and results are given in Table 4.7. The results indicate the following:

1. The localization algorithm runs consistently, with a standard deviation of around 10 mm for mean pose error using 100 particles.
2. The mean pose error has a minor benefit (around 6 percent less mean error) from the proposed method under the conditions of this experiment. However, the mean error's standard deviation can be lowered by 26 percent using the module.
3. The particle errors indicate the benefit of the proposed algorithm, since they are directly effected with the alterations made by the proposed module. The mean particle errors are found to be 27 percent better when the module is used. Also the particle errors were more consistent with a 48 percent better mean particle errors standard deviation value.

Table 4.7. LPE 1 - Egocentric checks off, seen twice checks on, 100 particles

	Normal	With PPM
Mean pose error	252	244
Std deviation of pose error	11	8.6
Mean pose error std deviation	332	208
Mean particle errors	395	289
Std deviation of particle errors	4.2	3.8
Mean particle errors std deviation	245	126

Further runs of the offline experiment is run with varying particle numbers, which is a core parameter of the MCL process. Tables 4.8 and 4.9 provide the results of this experiment, which do not indicate any important changes in the relation of *Normal* and *With PPM* cases. As the number of particles increases the values get better and as the number of the particles decrease the success of the algorithm drops and the difference in the success of the algorithm does not seem to be altered greatly with the change in number of particles.

Table 4.8. LPE 2 - Egocentric checks off, seen twice checks on, 25 particles

	Normal	With PPM
Mean pose error	256	258
Std deviation of pose error	19	13
Mean pose error std deviation	320	322
Mean particle errors	414	305
Std deviation of particle errors	4.6	19
Mean particle errors std deviation	277	150

The last table of this experiment Table 4.10 shows the results for the case the *seen twice checks* are turned off, which indicate the following:

Table 4.9. LPE 3 - Egocentric checks off, seen twice checks on, 500 particles

	Normal	With PPM
Mean pose error	253	244
Std deviation of pose error	4.3	10
Mean pose error std deviation	339	309
Mean particle errors	350	253
Std deviation of particle errors	3.6	3.3
Mean particle errors std deviation	210	75

1. The particle errors are effected significantly as more misperceptions arrive. The benefit of the proposed module becomes more apparent in such cases with a 40 percent better mean particle error rate.
2. The particle errors standard deviation also falls with the help of the proposed module, 34 percent better result.
3. The mean pose error also seems to drop but due to the effects of the odometry, further experiments are necessary before more comments can be made.

Table 4.10. LPE 4 - Egocentric checks off, seen twice checks off, 100 particles

	Normal	With PPM
Mean pose error	317	279
Std deviation of pose error	32	9.2
Mean pose error std deviation	302	381
Mean particle errors	854	516
Std deviation of particle errors	13	18
Mean particle errors std deviation	530	348

Figure 4.9 shows the effect of turning the *seen twice checks* off. Compared to Figure 4.7, which shows the same log file's output with the *seen twice checks* on, a third peak has appeared, indicating a third divergence of the normal Aibo player's localization particles. On the other hand the player with the post-perception module

suffers at the beginning to find the correct set of landmarks. However once they are found, it does not lose track of the correct landmarks. This is good demonstration of the proposed method's aim.

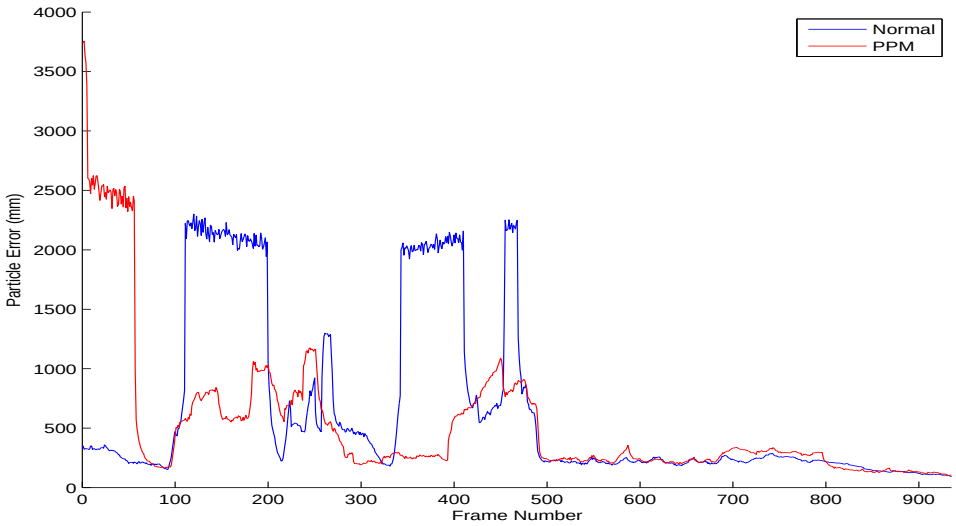


Figure 4.9. Particle errors for log number 2 experiment 2, averaged over 10 runs of all logs (with *seen twice checks* off)

4.2.2.3. Experiment 2. In the second experiment there are four misplaced landmarks, which creates a harder environment for the robot. The results are shown in Table 4.11, which indicate the following:

1. All values are worse compared to the first experiment where there were only two misplaced landmarks. The pose estimates and particle errors are around 70 percent worse.
2. The increase in standard deviation of pose errors and particle errors indicate the effects of stochasticity becomes more apparent. However they are still small compared to the mean pose and mean particle error values.
3. The proposed module still provides improvement over the normal player code. The mean particle errors are 18 percent better and particle errors' standard deviation was 28 percent less.

Table 4.11. LPE 1 - Egocentric checks off, seen twice checks on, 100 particles

	Normal	With PPM
Mean pose error	980	917
Std deviation of pose error	56	49
Mean pose error std deviation	866	823
Mean particle errors	1198	972
Std deviation of particle errors	69	38
Mean particle errors std deviation	484	346

In the experiments where the different number of particles were tested, the results were very similar to the first experiment. As the number of particles increases the error values decrease and as the number of particles decreases error values increase. However the effect is neither unexpected nor significant just as it is the case in the first experiment.

The results of the experiments with the *seen twice checks* turned off, indicate some parameters of the localization algorithms need to be adjusted. The findings are presented in Figure 4.12. Although particle errors are 23 percent less in the case with the proposed method, the mean pose estimate error suffers slightly with a 5 percent worse result. These findings may indicate that improving the localization algorithm may provide better solutions. On the other hand since the *large std deviation of pose error* (91) is greater than the absolute difference (40) between the *normal* and *ppm* cases, the difference could as well be due to the stochasticity of the MCL algorithm.

4.2.2.4. Experiment 3. In the third experiment the robot player on the field runs the post-perception module on board. Thus running the post-perception module offline means running the proposed module twice on the same data, which is not an expected case. So the results in Table 4.13 only have the *Normal* column on them, which already has the post-perception module effects in it.

Table 4.12. LPE 4 - Egocentric checks off, seen twice checks off, 100 particles

	Normal	With PPM
Mean pose error	675	715
Std deviation of pose error	91	35
Mean pose error std deviation	577	790
Mean particle errors	1679	1281
Std deviation of particle errors	27	24
Mean particle errors std deviation	807	633

Table 4.13. LPE 1 - Egocentric checks off, seen twice checks on, 100 particles

	Normal
Mean pose error	227
Std deviation of pose error	5.7
Mean pose error std deviation	222
Mean particle errors	299
Std deviation of particle errors	7.6
Mean particle errors std deviation	126

Compared to the first experiment, which is the most similar experiment to this experiment, the results look promising. The *mean pose error* is 10 percent less and *mean particle errors* are 24 percent less, which was also observed from the behavior of the robot.

Different number of particles yield similar results to the previous experiments, so they are not reported.

Turning the *seen twice checks* off, provides interesting results with the *mean particle errors*, as seen in Table 4.14. When compared with the first experiment, the mean pose error is about the same and the mean particle errors is about 22 percent

Table 4.14. LPE 4 - Egocentric checks off, seen twice checks off, 100 particles

	Normal
Mean pose error	318
Std deviation of pose error	24
Mean pose error std deviation	369
Mean particle errors	660
Std deviation of particle errors	16
Mean particle errors std deviation	414

less. This shows the mean particle errors can be kept lower when the post-perception module is run on the robot.

4.2.2.5. Experiment 4. In the fourth experiment, the robot performs *passive localization*. In other words the actual task of the robot is not localizing itself on the field. In this case the robot goes to the same position as previous experiments, but instead it chases the ball placed at that location. Since the ball chasing behavior does not provide as much perception information, the overall errors are greater than the previous cases.

In Table 4.15 the results of the initial experiment are given, which show a better mean pose error of 14 percent when the post-perception module is used. Compared with the information from the first experiment, where active localization is performed, this value suggest that as the perception information becomes more critical, the effects of the proposed modules become more apparent. In the first experiment, the mean pose error value was not improved with the post-perception module. On the other hand since there are less number of perceptions available in passive localization, the improvement was possible with the use of the proposed module.

Experiments with different number of particles have similar effects to the above

Table 4.15. LPE 1 - Egocentric checks off, seen twice checks on, 100 particles

	Normal	With PPM
Mean pose error	618	530
Std deviation of pose error	31	30
Mean pose error std deviation	531	480
Mean particle errors	791	598
Std deviation of particle errors	28	27
Mean particle errors std deviation	347	261

Table 4.16. LPE 4 - Egocentric checks off, seen twice checks off, 100 particles

	Normal	With PPM
Mean pose error	699	542
Std deviation of pose error	34	29
Mean pose error std deviation	601	449
Mean particle errors	1166	690
Std deviation of particle errors	27	33
Mean particle errors std deviation	574	332

mentioned experiments, so they are not reported.

The results in Table 4.16 once more shows as perception information becomes more critical, for example in cases where the misperception ratio increases, the benefits of the proposed module become more important. In this experiment, the *seen twice checks* are turned off, which means the number of misperceptions is more as well as the correct perception information. In the case the proposed module is active, the mean pose error is 22 percent less and mean particle errors is 40 percent less.

4.2.2.6. Experiment 5. Fifth experiment tests the behavior of the robot with the post-perception module running during the experiments, in which the robot is chasing the

Table 4.17. LPE 1 - Egocentric checks off, seen twice checks on, 100 particles

	Normal
Mean pose error	538
Std deviation of pose error	77
Mean pose error std deviation	471
Mean particle errors	675
Std deviation of particle errors	78
Mean particle errors std deviation	325

ball. The best comparison case is the fourth experiment with the same settings except when the post-perception module runs onboard. Table 4.17 shows the results. As is the case with the third experiment, since the module is working on the robot, it is run a second time in the offline testing, which means the values reported as *Normal* already contain the effects of the module.

Compared with the *normal* case of the fourth experiment mean pose error is 13 percent less, and the mean particle errors is 15 percent less. These results show using the proposed method on the robot improves the localization performance of the robot.

Table 4.18 reports the results of the experiment performed with *seen twice checks* off, which provides a greater number of misperceptions. Compared to another similar environment of the experiment four, there is a 20 percent less amount of mean pose error and 26 percent less amount of mean particle error.

Perhaps more importantly the mean pose error is within the boundaries of the standard deviation of mean pose error. In other words it is possible to conclude that the proposed module can be used instead of the *seen twice checks*, which is a crude check removing a lot of the perception information in noisy environments with low amount of perception information available.

Table 4.18. LPE 4 - Egocentric checks off, seen twice checks off, 100 particles

	Normal
Mean pose error	558
Std deviation of pose error	54
Mean pose error std deviation	488
Mean particle errors	852
Std deviation of particle errors	58
Mean particle errors std deviation	428

4.2.2.7. Time Graphs From The Experiments. Due to the uncontrollable internal mechanics of the localization, such as not using the visual updates in case the particles diverge, the overall numerical results may be confusing. To better indicate the effects of the proposed module this section provides some graphs, which better point out the effects of the proposed module.

The best indicator of the proposed module is the particle errors, which provide the convergence and divergence behavior depending on the received misperceptions. The proposed module has several effects on the on the received perceptions. Since the unexpected perceptions are removed, the sudden changes are less common in the graphs with the effects of the module. When the module is successful, erroneous perception information is not sent to the localization module, thus divergences are less common.

Each graph in the Figure 4.2.2.7 displays particle error graphs from several experiments under different conditions. In these graphs, the effects of the proposed method can be observed. In Figure 4.10 the normal run of the present localization code diverges three times, whereas the proposed module handles these divergences by removing the responsible misperceptions.

Figure 4.11 presents results from the second experiment, where there are four misperception sources instead of two, proposed module handles some misperceptions but repentant misperceptions are too convincing for the module after some time. To

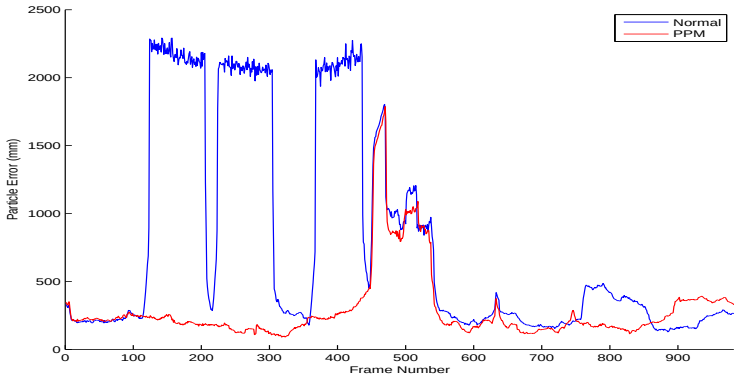


Figure 4.10. Graph from first experiment

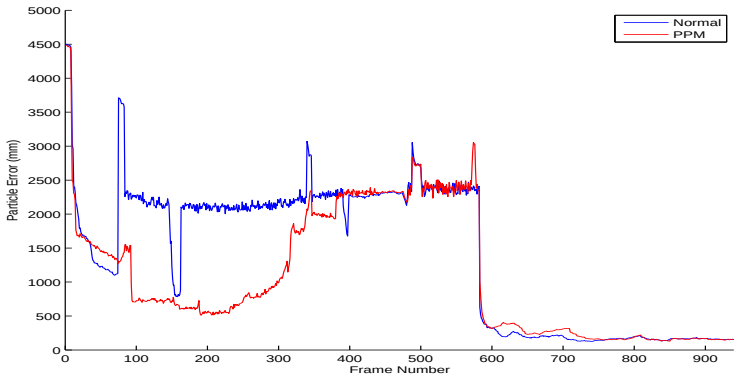
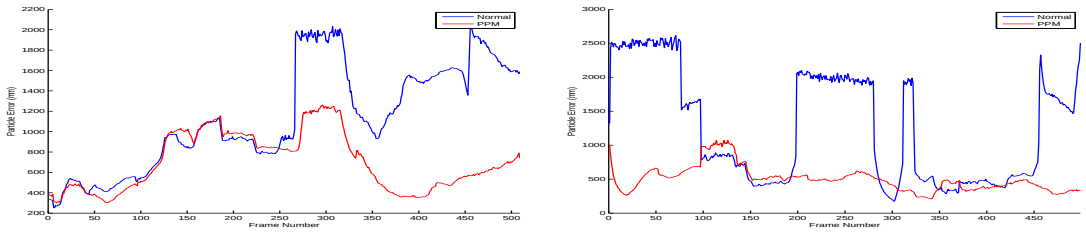


Figure 4.11. Graph from second experiment

get an even better result in this case, running the proposed module could be run on the robot, which causes robot to act more sensibly removing some of the misperceptions.

Figure 4.12(a) and 4.12(b) show results from the fourth experiment, where the robot does not perform active localization as is the case in other experiments. Since the robot does not explicitly search for landmarks in the environment, less perception information is available, which makes the misperceptions much more harmful in terms of the overall localization performance. In the Figure 4.12(b) the *seen twice checks* are off, thus even a larger number of divergences are present in the normal run of the robot. However the post-perception module has managed to remove most of the misperceptions providing a better input set for the localization algorithm.



(a) (b)

Figure 4.12. Graphs from the fourth experiment

Table 4.19. Misperception elimination experiment results (learned parameters)

	Tr. 1	Tr. 2	Tr. 3	Tr. 4	Tr. 5	Avg.
Number of Misperceptions	4	2	19	26	23	15
Number of Misperception Reports	90	108	75	128	96	99
Frames With Observation	392	463	340	383	351	385
Frames With No Observation	318	346	517	487	492	432
Number of Total Frames	710	809	857	870	843	817
Misperception Elimination	50%	100%	61%	67%	74%	70%

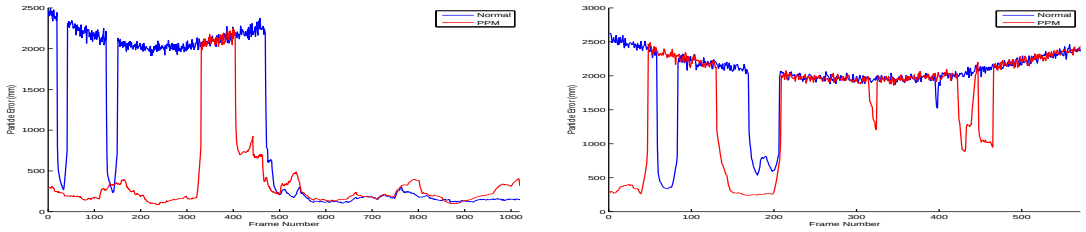
4.2.3. Learned Parameter Set Results on Aibo Platform

4.2.3.1. Misperception Elimination Results. Table 4.19 shows the results of the misperception elimination experiment with the HMM parameters learned in the simulation environment, which seem to be almost as good as the results achieved with the hand tuned parameter set shown in Table 4.5. On average hand coded results are 10% more successful in eliminating the misperceptions.

4.2.3.2. Localization Performance Results. Table 4.20 shows the *mean particle error* values of all experiments, which are the best indicator of the proposed module's effects on localization algorithm. Columns with the *EC* label indicate results achieved with the expert coded parameter set and *L* label indicates the results from the learned set parameter experiments. *Effect* columns present the improvement achieved by the usage

Table 4.20. Mean particle error values with learned parameters

	EC-Normal	EC-PPM	Effect	L-Normal	L-PPM	Effect
Exp1 - MPE	854	516	39%	856	494	42%
Exp2 - MPE	1679	1281	24%	1679	1401	17%
Exp4 - MPE	1166	690	40%	1160	845	27%



(a) (b)

Figure 4.13. Graphs from the first (a) and second (b) experiments with the learned parameter set

of the module in mean particle errors (MPE).

As provided in the table overall performance of the learned parameters seems to be close to the expert coded parameter set. However to better understand the particle errors characteristics a closer look at each experiment is necessary, which is available with the particle localization graphs over a single experiment. Figure 4.2.3.2 presents the mean particle error graphs, indicating the performance of the learned parameter set. Graphs show the proposed module is more vulnerable to misperceptions with the learned parameter set, which is infact not very surprising. Since the simulation environment does not have any misperception introduced, the model does not include the effects of possible misperceptions.

4.2.4. Autonomous Car Experiments

The results of the first set of experiments on the autonomous vehicle platform are presented in the Figure 4.21. Over a total of 1739 frames, on average 47 correct

misperceptions were found and 28 false positive reports were made. The number of false positives needs to be less. However considering a simple Gaussian observation model and unsophisticated perception method, these results are a satisfactory way of showing some improvements can be achieved in other domains as well.

Table 4.21. First experiment results for the autonomous vehicle platform

	Trial 1	Trial 2	Trial 3	Average
Total frame number	862	556	321	579
True positive number	52	63	26	47
False positive number	37	35	13	28

The second set of experiments are unsurprisingly more successful in terms of number of the ratio of the true positive and the false positive reports as can be seen in Table 4.22. The ratio of false positive reports to all reports in the first set of experiments is 37 percent on average, whereas in the second set of experiments it is 18 percent on average. This improvement can be credited to the more complex lower level perception method, which apparently does not produce as many *misperceptions* as the simple perception method.

Table 4.22. Second experiment results for the autonomous vehicle platform

	Trial 1	Trial 2	Trial 3	Average
Total frame number	71	60	66	66
True positive number	14	10	4	9
False positive number	4	1	0	2

Figure 4.14 shows the states of the HMM during the first trial. All runs of the proposed model have a similar HMM graph, where states are aligned vertically and the darker areas show more probable states. In the autonomous vehicle experiments these graphs are more interesting since they actually show the path the vehicle goes through during the trial.

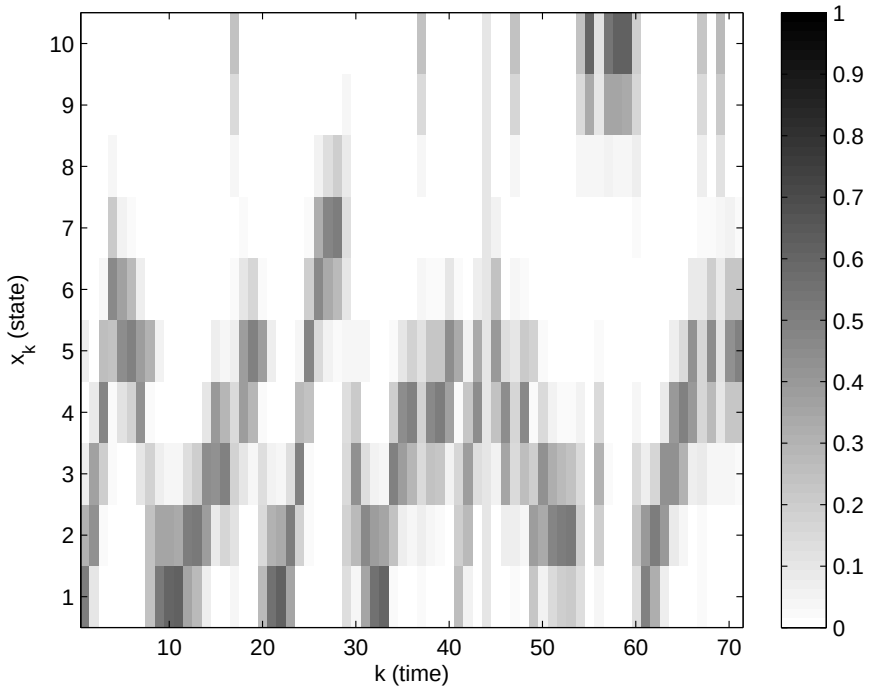


Figure 4.14. HMM states of the first trial of the second experiment

5. CONCLUSIONS

Due to computability problems mainly created by the huge state space, visual perception problems are bound to be solved by suboptimal methods, which come with many problematic assumptions. These assumptions may not work as expected in real environments. Most problems with these methods boil down to inaccurate perception of the landmarks, either by not perceiving them or by *misperceiving* them. The misperceived landmarks, in other words landmarks which are not actually there, are targeted by this work.

The proposed method uses a Hidden Markov Model of the environment, which creates an expectation of the landmarks to be perceived. Using this probabilistic expectation, it is possible to detect unexpected landmarks. The results of the experiments conducted in two environments indicate it is possible to generate better results using the proposed method.

Use of the proposed approach makes it possible to achieve better localization performance at the expense of losing some of the perception information. In an environment with six landmarks, two consistent misperception sources can be tolerated, but experiments show four of consistent misperception sources may cause problems. In this case since there are too many misperception sources, the module may start expecting misperceptions. However, since the method is designed for rare cases of misperceptions so even two *consistent* misperceptions sources is enough to show the validity of the results in terms of the primary goals of this thesis.

Further experiments conducted with the autonomous vehicle indicate that the proposed algorithm is a general enough, to be used in smoothing of any perception gathered by any robot or system with minor modifications according to the domain of the target environment.

5.1. Future Work

The underlying probabilistic framework creates a lot of room for improvements. There may be many different additions to the current design of the module. For example adding a submodule which provides expected perceptions may be very beneficial for higher level modules.

With the proposed method in place, relaxing constraints of lower level perception may be possible. In which case there would be more perceptions available, selecting correct misperceptions may be easier. The higher modules may work much better if more perceptions are available.

Learning the HMM parameters with a more robust method may also be a future work. Due to the undeterministic effects of the misperceptions in the environment any learning procedure must be designed carefully, otherwise as Section 4.2.3.2 indicates, effects of the misperception in the environment may not be handled well enough. A supervision input, such as the location performance data provided by the overhead camera system, is necessary for such a learning algorithm so that effects of the misperceptions can be calculated to guide the learning procedure. Using an online procedure will be much more robust since the robot can only then be able to reason about the temporal nature of the misperceptions in the environment.

It is also possible to make better use of simpler perception modules using the proposed module. Consider a blob based goal perception which marks any large goal colored blob as a goal target. This would be a very beneficial perception if the robot was actually facing the goal. However it can not be used as a general input to the localization because such a simple perception module will generate a lot of misperceptions. If an accurate set of perception sources are available to the proposed module, simpler perceptions can be more actively deduced to be correct or incorrect. Thus computationally cheaper algorithms may be more efficiently used.

APPENDIX A: ROBOTIC SOFTWARE ARCHITECTURE DESIGN

A.1. Development Method

In order to efficiently develop robotic software, every piece of code must be written with great care. However out experiences indicate care is never enough, only using a code base for a period of time can guarantee satisfactory output. In such a fragile environment maintaining a code base in time becomes more important since as time passes code base becomes more valuable.

On the other hand, new code come with many classical pitfalls of the software engineering practices. One of the most common issues is turning the code base into spaghetti code, which is not hard to achieve in a middle scale project with multiple people working at different times and not necessarily in coordination.

Thus building a time tested code base requires a solid structure, which will make sure newly included code pieces can not harm the overall design. Otherwise newly added code will easily cause time loss as well as rendering previous code base useless.

Our design makes a critical separation between platform specific code and platform independent code. There are two main branches of code during the development process. The *BOUNLib* branch contains all of the platform independent general purpose code. Any platform specific code, such as a robot's code contains instantiations of the BOUNLib components. A similarly approach is also taken within the platform specific code. There are subbranches within the platform specific code, which enable unit testing of individual parts of the code using our debugging utility Cerberus Station[10].

As platform specific code becomes more mature, it is possible to generalize such code parts and move them into the BOUNLib. If we get one step closer to the implementation level, such a generalization effort involves parametrization of some constant

values, providing interfaces to various parts of the platform specific code.

Explained levels of generality are implemented with shared libraries. Each component of the code base is organized into a shared library so that any other piece of code can use any part of the system. Using shared libraries provides ease of implementation and generality. A simple test code can be implemented before using the shared library. Once the simple test program works as expected any other piece of code can use the shared library without any changes in the library itself.

A.2. Code Structure

Our laboratory's past experiences indicate some kind of a shared object is necessary to form a common data structure so that numerous different modules can interact with each other without explicitly depending on each other in compile time. The *BOUNCore* component of the *BOUNLib* serves for this purpose. All classes within the *BOUNLib* can access the data structures of *BOUNCore* and read/write data on the shared *BOUNCore* object.

Similarly on the platform dependent side of the code base a class called *Robot*, which inherits from the *BOUNCore*, acts as the shared object. All parts of the platform dependent code base communicate with each other among the *Robot* object.

With this structure it is possible to maintain a convenient object, which includes all of the important information. Storing, viewing or any other operation required to be performed on the robot or in the debug mode using the Cerberus Station can be conveniently performed on a single interface by any module.

A.3. Summary

The efforts of our laboratory using the above explained development methodology and code structure have proved to be useful and fruitful in last two years. Using many independent sections of the code base, all developers were able to work without causing

any conflicts to other parts of the code. The resultant code base has time tested quality, easy to understand structure and conveniently extendable modules, which is about all any developer would like to achieve with his work.

APPENDIX B: PERCEPTION MODULE DETAILS

This appendix provides detailed information about the lower level perception modules, tools used in development of the modules and presents some examples of the detection procedure.

As a part of this thesis a scanline based lower level vision system was implemented. The main source for inspiration was the scanline based vision system of the German Team[36] of RoboCup SPL. In our current implementation of the Aibo player code, the scanline vision algorithm is primarily used for the detection of the goals. Beacons and the ball detection algorithms use blob based methods, which will be briefly described as well.

The generic scanline detection algorithm is very flexibly implemented. For the purposes of the second target environment a very simple lane center detection algorithm was implemented within a few hours. In the sections below this algorithm is also described.

B.1. Lower Level Vision Module Overview

Once an image is received from an image source, such as the camera of a robot, it is impractical to process the received image in raw format if not impossible with computer power provided by the current standard mobile processors. In order to reduce the search space to a tractable size, the first step in low level vision modules is to *classify* the image. In the classification procedure every of the pixels of the image is mapped to a corresponding discrete color such as green, orange, ignored etc. The fastest method of performing the projection process is using a lookup table, called a *color table*, in which each possible color is mapped to a discrete color.

B.1.1. Color Table Generation

The *color table* provides the fastest method of classifying images available to us. This process also takes care of a great deal of uncertainty, which makes generation of a robust color table one of the most crucial procedures of the robotic research. Thus our laboratory has developed a few tools to conveniently generate color tables.

To generate a new color table, we use our labeler tool [10] which provides a way of labeling values in YUV color space with the associated discrete colors. Next a Matlab program uses the generalized regression neural network method to train a neural network with the labeled values. The trained network provides a discrete color index given a color value in the YUV color space. The network also takes the distance of the labeled pixel from the center of the image as an input for both training and testing, in order to compensate for the high levels of radiometric distortion of the Aibo robot's camera. Finally another Matlab program creates the color table by matching every possible color in the YUV color space with the corresponding answer of the trained neural network.

Another use of the color table lookup method is altering the system behavior according to the changing tasks. For example the overhead camera system described in the Appendix C uses a custom color table, which only contains information about the colors related with the overhead camera system, instead of our regular color tables used in robotic soccer.

B.1.2. Classification Performance Considerations

Although classification procedure provides a tractable method of computation for our further algorithms, it is still not the optimal method for visual detection algorithms. The main problem with classification performance of an image is due to the number of pixels in an image. Even in a very small image, such as the image received by the color camera of the Aibo robot, which has 208 pixels in a row and 160 rows of pixels in an image, there are a total of 33280 pixels. Even the simplest possible processing, such as

a color table lookup, becomes expensive when it is performed for each pixel. Indeed we found the classification step being the one of the most expensive task in terms of computational time required in our Aibo soccer player code.

The historical requirement of classifying all of the image is due to the blob based methods employed by out laboratory. Since blob based methods trace blobs of same colored pixels, it is more efficient to classify all of the image once and look for blobs in the classified image.

Using scanline perception algorithms it is possible to eliminate the need of classifying all of an image. Since scanline methods only deal with the pixels on the scanlines drawn on the raw image, number of pixels to classify can be reduced according to the performance requirements with the cost of losing visual landmark detection precision. In practice we have observed that a moderate number of scanlines can both drastically reduce the number of pixels to be classified and still provide impressive visual detection results.

Sections below demonstrate scanline based perception methods. An example of the blob based methods is presented in the Section C.3.2 of the Appendix C.

B.2. Sample Scanline Based Perception Implementations

B.2.1. Goal Perception of Aibo

Scanline based goal perception aims to detect the goal bars of each goal individually so that multiple landmarks can be perceived from a single goal structure. To achieve this a four stage procedure is designed and implemented, which can be used to perceive left and right goal bars individually. Due to the generic design of the perception stages and the underlying scanline framework, implementing top and bottom bars is only a task of mirroring left and right bar *percepters*. Similarly beacon perception can be implemented with a slight variation of the bar perception.

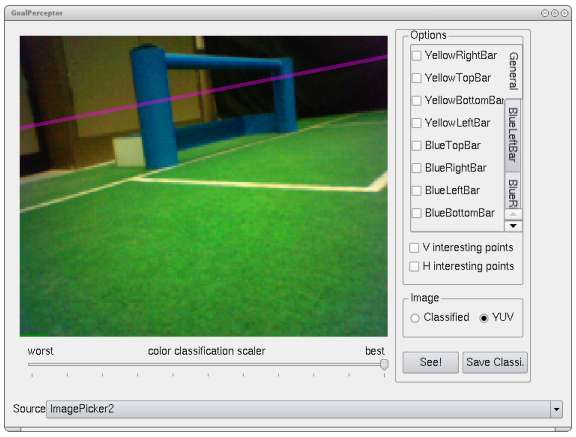


Figure B.1. Cerberus Station Goal Perception Test Plugin

Tests of the percepter are conducted using the newly Goal Perception Tester Plugin of the Cerberus Station debugging tool as can be seen in Figure B.2.1. Using this visual tool greatly enhances the testing procedure. Employing the *BOUNio* library's message logging infrastructure, it is possible to observe run time performance as well as inspecting recorded logs within the goal perception tester plugin.

The goal perception task is divided between several classes, namely main vision object, goal percepter, goal bar percepter and bar percepter. Bar percepter performs the four stage bar perception procedure, goal bar percepter investigates if the found bar is the correct bar and the main vision object initiates the procedure.

Initially the main vision class runs a series of scanlines vertical to the horizon plane and classifies the YUV image along these scanlines as shown in Figure B.2.1. With procedure it is possible to avoid classifying all of the image, which accounts to a large portion of the computational cost in previous blob based implementations.

Generated classified points, called *important points*, are passed to the goal percepter class. There are four goal bar percepter classes, one for each vertical bar of the two goals, blue left, blue right, yellow left, yellow right. Each of these goal bar percepters have three bar percepters. Each goal percepter inspects all of these three bar perceptions to pick the best candidate to be chosen as the perceived goal bar. Finally bar percepter follows the four stage procedure to detect the goal bar candidates as

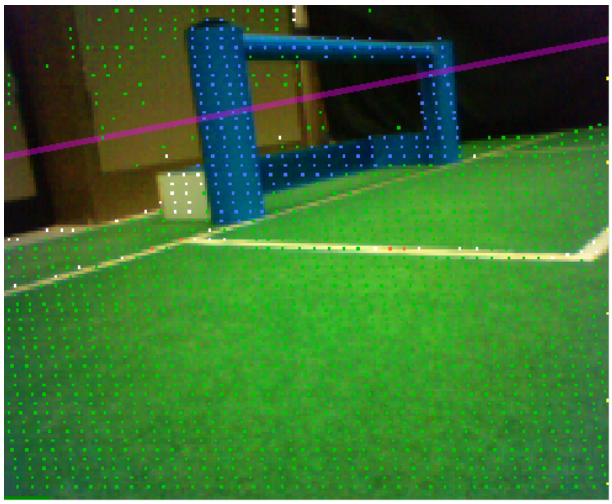
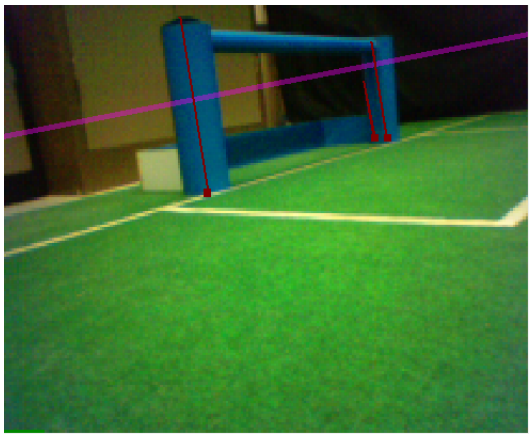


Figure B.2. Important points

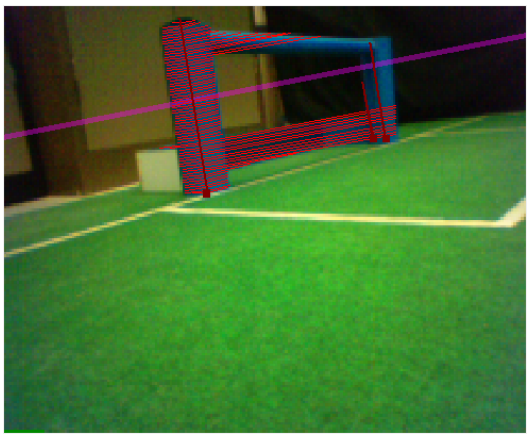
follows (shown visually in Figure B.2.1).

1. Find vertical base: The *important points* are scanned for the largest continuous segment of the given color. The segment can be discontinued with a value defined by a constant, which can be adjusted according to the performance of the classification available.
2. Ranger lines: From the detected vertical base, perpendicular scanlines are drawn to locate the ends of the colored region.
3. Bound detection: Ranger lines are passed through a histogram filter to find the most likely edge of the bar.
4. Bounding box: The four corners of the detected bounds are stored as the perceived bar plane.

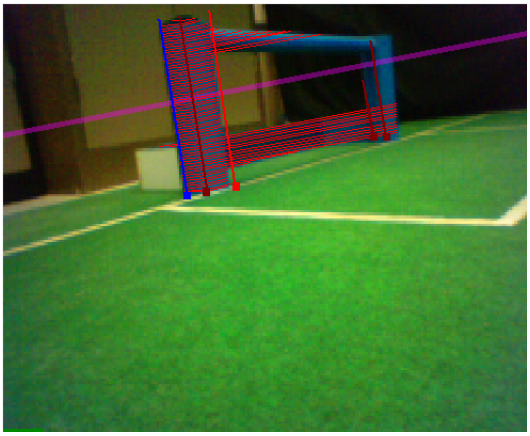
Once candidate bars are generated, respective goal bar percepters search for the best available vertical goal bar of their respective colors. To achieve quantified confidence results, goal bar percepters draw squares, scaled to the size of the perceived bar, around the expected positions of the colored regions, as seen in Figure B.4(a). Within these squares further scanlines are run to detect the ratio of pixels with the requested color. Figure B.4(b) shows the resulting confidence values.



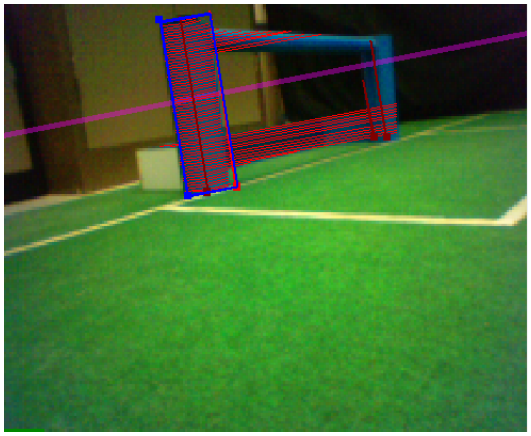
(a) Vertical bases



(b) Ranger lines for the largest vertical base

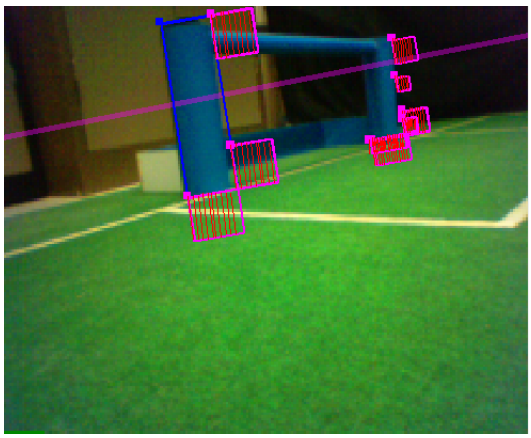


(c) Bounding lines

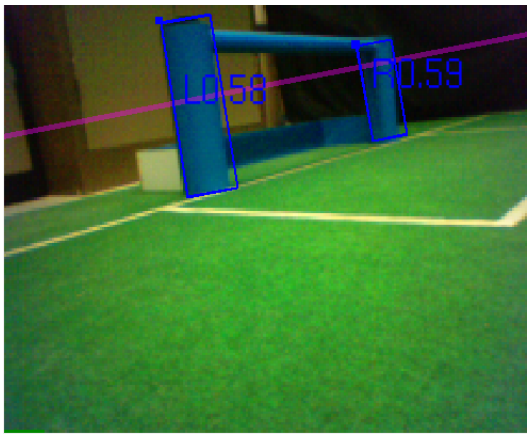


(d) Final bounding box

Figure B.3. Four stages of bar perception



(a) Confidence regions drawn for all possible left blue vertical goal bars



(b) Final confidence values for left and right blue vertical goal bars

Figure B.4. Final perception

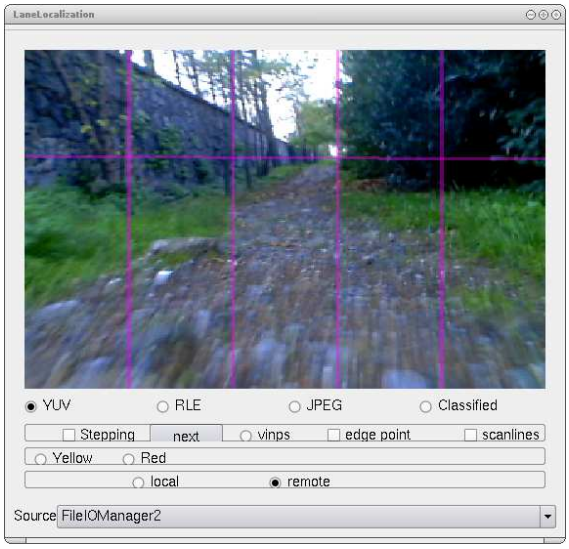
B.2.2. Mid-Lane Perception of The Autonomous Car

Using flexible the code base generated for the scanline perception implementing a new perception method became a not so challenging problem. This section presents an example of such a case, where a new implementation is created from scratch. The task at hand is to provide a very simple perception module to detect on which section does the center of the road in the images fall into. The data is collected by the vehicle while it is being driven on a pathway behind our department.

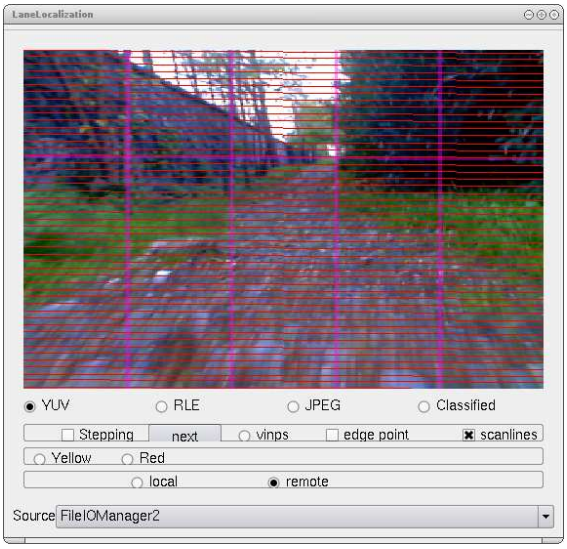
To achieve a meaningful classification of the received images, a new color table must be generated. This procedure involves labeling a few images with the labeler tool and generating a color table with the Matlab program, which in total takes about ten minutes. Next step is to generate a new plugin for the Cerberus Station, called Lane Localization Tester plugin. The plugin is actually created as the copy of another existing plugin. Thus not much new code is written to create a new plugin. Once we have a new plugin and a color table, classified image can be viewed through the plugin as seen on Figure 3.5.

Once we have our testing platform ready, we can start implementing our new perception method. After an hour or so and 223 lines of code, the perception method is ready, which draws scanlines on the received image, finds the right and left sides of the road and marks the middle of these line segments as the center of the road for that scanline. Then average of center points of all scanlines on the image represents the center of the road for that image. Figure B.5 shows the procedure. The black square at the top of the image represents the selected partition of the image in which the center of the road falls into. Therefore in the case of the Figure B.5 the result of the perception module is 3. Other possible sections of the image are labeled from 1 to 5.

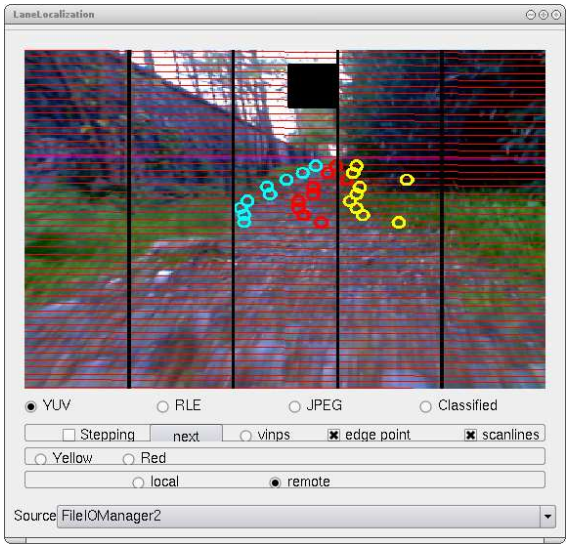
Generation of the mid-lane perception took about a single work day. This shows the software development principles are correct and the underlying scanline code base works conveniently.



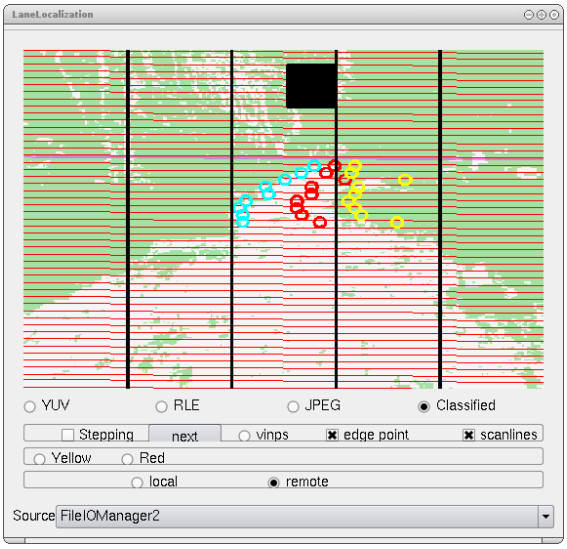
(a) Raw image from the vehicle



(b) Scanlines on the raw image



(c) Left/right/center of the road



(d) Corresponding classified image

Figure B.5. Mid-Lane perception procedure

APPENDIX C: OVERHEAD CAMERA SYSTEM

As a part of this thesis work an overhead camera system was developed. Such an external observation system plays critical role in a wide spectrum of research topics. For instance in neural network implementations a supervisor input is necessary to train the network before any reasoning can be made with the run time behavior of the system. The developed system is currently being used for two other master theses, which demonstrate other uses of an overhead camera system. In one of the theses[37], the system provides input for training phase of the evolutionary learning process for two legged walking algorithm. In the other thesis the input from the system provides a measure to calculate the error in the odometry information of a mobile robot[38]. Robots used in other experiments with their attached markers can be seen in Figures C.7(b) and C.7(c).

In this thesis the overhead camera system is used for quantitative assessment of the localization module. The system provides the position of the Aibo robot with a marker attached on its back (Figure C.7(a)) on the field. The data is used to calculate the localization module error. The sections below describe the overhead camera system in detail.

C.1. Requirements

Initial requirements can be listed as follows:

- Contain all of the field in its field of view.
- Accurate enough to provide mobile robotics localization tests.
- Flexible software infrastructure.
- Robust enough to work for long periods of time.
- Low component costs.

C.2. Structural Design

The field we want to observe in our laboratory is a scaled version of the RoboCup Standard Platform League field, its dimensions are 5200 x 3300 mm. As shown in Figure C.1 two USB 2.0 cameras are used to cover the entire field. Due to limitations with the ceiling height (2600 mm) of our laboratory, multiple cameras are necessary to cover the entire field. Cameras are of the shelf webcams. The only requirement is the large field of view. Cameras are labeled *wide angle lens* or *ultra wide angle lens* by the manufacturers to indicate wide angle field of view (around 70-80 degrees). Linux compatibility is also good feature to provide easy integration with our current software systems. The selected camera model complies with the (USB video class) UVC standard, which is well supported by the Linux operating system.

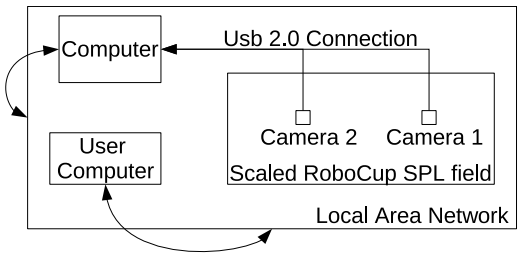


Figure C.1. Overhead camera system design

Different camera orientations could be used to increase the camera coverage with the cost of losing some measuring precision. For example a camera placed on one side of the field can observe the other side of the field. In this configuration camera coverage of the field increases greatly but due to the effects of the orientation of the camera on the received image, data collected from some parts of the field may have a larger amount of error than the other parts of the field.

The computer cameras are connected is quite an old desktop PC with a Celeron 2.0 GHz processor and 512 MB of RAM. The local area network connection in our laboratory is a standard 10/100 Mbps network with wireless access for user computers and a cable connection to the overhead camera system's computer. The cable connection can provide up to 8 MB/s connection speed, but due to large images sent by the camera (640x480) and the wireless connections of the users computers' viewing images on user

computers is not fast. Fortunately experiments do not require sending images over the network, some processing is done on the overhead camera system's computer and only application data is sent over the network which requires a few hundred kilobytes per second connection speed at most.

The cameras can send images at 640x480 at 30 frames per second (fps) in two different image formats. The first format is the YUY2 format (also called raw), the second is the compressed MPEG format. If the compressed format is used compression and decompression of an image creates an unnecessary layer of processing both for the camera and the camera computer. Since we would like to avoid any unnecessary computations, which may result in slow frames per second rate, raw format is a better choice. However in the raw format the USB 2.0 bus can not handle two cameras on a single bus. Although the camera computer contains 6 USB ports, apparently they are all connected to a single USB 2.0 bus, which can not handle two cameras. To solve this problem a separate PCI USB Interface Card is used, which provides a second USB 2.0 bus to the computer. With this configuration the computer can receive images from both of the cameras at the same time.

A major part of the system's design is to find a suitable arrangement of a solid structure to form a platform for placing the cameras. Simple metal bars and several brackets supplied from a local shop are used for this purpose. Four thick brackets are drilled into the ceiling forming a multi purpose platform for different configurations of metal bars. The thin bar seen in the Figure C.5(a) can be placed at any point between the two thicker bars, it can also be attached to a single thick bar to extend on either side of the thick bars. The solid structure forms a good platform for further research instruments as well.

C.3. Software Design

The system contains multiple cameras. Thus the software system must be able to obtain multiple images and perform other processes required on these images. The images do not arrive in a synchronized fashion and cameras are separate devices pro-

viding multiple memory buffers, thus software should also be able handle concurrency of processing multiple cameras. The system also needs a user interface, which will direct all these different possible cases of controls and processing. Below components of the software system design are described in detail.

C.3.1. Receiving Images From The Cameras

Due to the concurrent nature of the task, receiving images process should be designed as flexible as possible. To achieve this requirement every part of the receiving image process is designed and implemented separately, so that all parts can be used in any configurations required.

The driver supplied by the manufacturer of the used camera complies with the standard video4linux 2 (v4l2) API. Using relevant parts of the luvcview[39] source code a viewer class is designed, which simply controls the camera and provides a memory buffer once a frame is received. A callback mechanism is also implemented to provide easy access for other parts of the code using a subscription/unsubscription based system. Any other class can subscribe to the camera controller class and be called upon arrival of a new frame from the camera.

In one of the common usage styles of the multiple camera systems, received images from multiple cameras are combined into a single image. Further algorithms are run in the combined image and thus synchronization is handled at a lower level. Reviewing possible cases of marker detection quickly shows that in the designed system combining images procedure is not necessary. In the case of detection of a marker, which indicates an object of interest, by a single camera, there are no problems of synchronization. At first it might seem if a marker is detected by both cameras, synchronization problem might occur. However since cameras are stationary a marker detected by any number of cameras must have the very similar coordinates (if not the same) for all of the detected cameras. If any of the cameras give a different answer than others, then that camera must be re-calibrated. This argument shows there is no need to combine images received by the cameras into single image. Further sections below describe how the

control code is used by the user interface to control multiple cameras.

Cameras have several parameters which drastically change the numerical values of the colors in the images. Especially automatic changes made by the camera render the reliable marker detection algorithms useless. To achieve the similar numeric color values for each operation of the cameras, a set of manually selected parameters are set at the start up of the image receiving procedure. In some cases applying these parameters may require resetting all parameters before setting them, which is possible using the *luvcview* program. As a future work this resetting step will be embedded into the image receiving procedure.

Multiple cameras generate multiple sets of detected markers, which are combined into a single set of detected markers. If a single marker is detected by two cameras at the same time average of the detected values are taken, since they must be very close if not the same.

Detection of a single marker by multiple cameras is possible in our current configuration because we found that such a redundancy is necessary in order not to lose track of the markers while the robot attached with a marker passes through the middle of the field where field of view of the cameras end. If respective field of views of the cameras are aligned next to each other with no overlap, the markers are lost during the transition from one camera to the other since all parts of a marker is required to be observed for successful detection. For robust tracking of the markers cameras are configured in such a way that in the middle line of the field, both cameras' field of view overlap. With an overlap of a marker's length, it is possible to move a marker from one camera's field of view to another without any loss in tracking.

C.3.2. Detecting Markers

To simplify the detection of the targets on the field markers of different colors are designed. Markers are made of simple colored cardboard. The largest piece is the black colored base piece, which contains the three smaller circular cardboard pieces.

As shown in the Figure C.2 the smaller (7 cm diameter) of these three pieces provide the identity of the marker and the larger (9 cm diameter) piece provides the hint color.

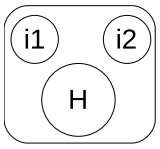
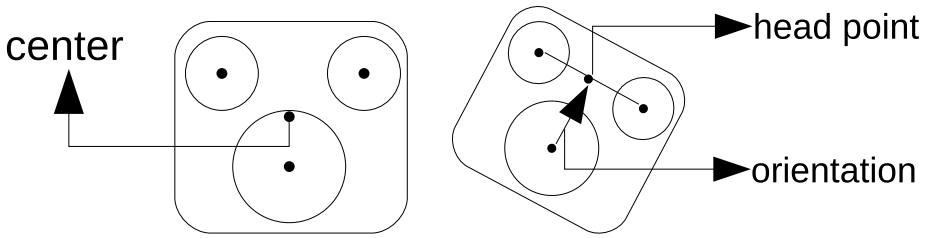


Figure C.2. Marker design

The marker detection algorithm is built upon the blob detection methods developed in our laboratory, which find the circular blobs in a given classified image generated with a custom color table. The details of classified image generation and other lower level vision module procedures are described in Appendix B. Figure C.7(d) provides an example classified image used in the overhead camera system.

Once the classified image and the blobs in the image are formed, the marker detection algorithm enumerates through the available blobs to locate the markers if there are any in the current image. Hint colored blobs are located first in the image. The distance of all identity colored blobs in the image are tested next to see if they are close enough to the hint colored blob to form a marker. If identity blobs are detected in the correct configuration for any marker, the center of all three blobs, as seen in Figure C.3 is marked as the new location of the corresponding marker. The orientation of the detected marker is calculated as the angle of the line drawn between the center of the hint colored blob and the head point in the field coordinate system. Head point is calculated as the midpoint between the two identity blobs.



(a) Marker center calculation (b) Marker orientation calculation

Figure C.3. Marker design

System is currently designed with nine different markers. Any number of addi-

tional markers may be designed by adding new identity colors. Images showing the steps of the detection process is presented in Section C.4.

C.3.3. Projection Calibration

In order to make use of the information obtained by the detected markers a projection method is necessary to convert the image coordinates into the field coordinate system. A generic calibration method can also be used to handle errors related with the camera parameters. The simplest conversion might be finding a set of coefficient values for estimating the length of a single pixel in the field coordinate system. However this method may cause errors if the camera is not perfectly parallel to the field, which is not an easy task to accomplish. To avoid such an error prone process and achieve a generic calibration method, multi-linear regression is selected as the calibration method. Using Matlab's related functions two set of parameters are generated. The first set is used to convert given x and y values in the image coordinates to x coordinate of the field coordinate system and the second set is used to convert given x and y values in the image coordinates to y coordinate of the field coordinate system.

Using such a training method provides a robust method of projection, hiding many possible sources of error such as camera orientation, camera lens specific distortion, intrinsic parameters of each camera, which are bound to be different for each individual camera. The two cameras used in the current installation are calibrated using this method, the error is found to be around 20 mm for each axis, which is acceptable as for purposes of this thesis. An example projection is shown in Figure C.4, Figure C.4(a).

C.3.4. Data Extraction

Data generated by the overhead camera system can be sent to specific IP addresses, broadcasted to the network with UDP messages or recorded to the local disk using the versatile BOUNio library[10] developed in our laboratory.

(a) Camera image (b) Corresponding detected marker

Figure C.4. Projection of the detected marker into field coordinate system

There are two different methods available for achieving the required data for further research related processing. The first is recording the robot log and overhead camera logs separately and merging them later with custom code. The second is receiving the overhead camera data from the robot code and recording the data within the robot log in real time. Any one of these two methods can be employed according to the needs of the experiment. The first scenario requires manual recording of the start and end of the robot log message timestamps, which may not be very accurate. In cases where very accurate timing is required second method is advised to be used.

C.4. Implementation

C.4.1. Structure

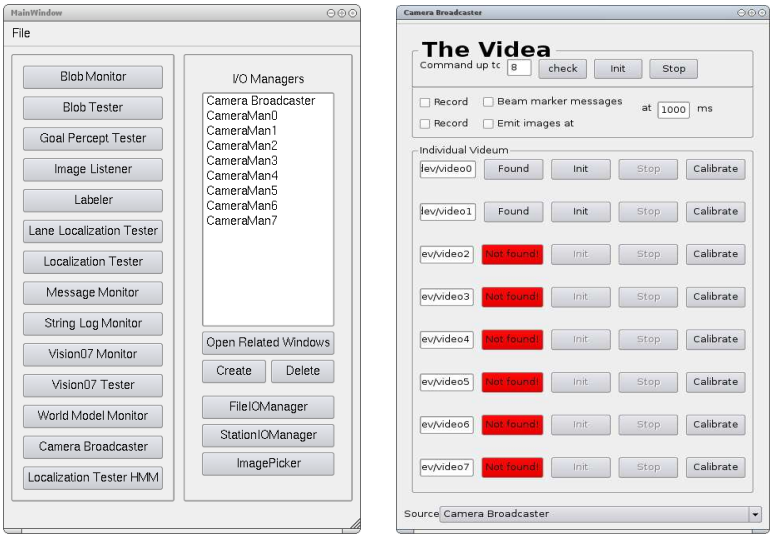
The final pictures of the system are given in the Figure C.5. The metal structure provides a multi purpose platform for much room for further devices.

C.4.2. Camera Broadcaster Plugin

The software driving the cameras is added to the Cerberus Station[10], which is a flexible visualization tool developed by our laboratory in C++ to better analyze and debug robotic applications. Simply adding a plugin to the station is enough to benefit from the robust messaging infrastructure provided by the BOUNio library. Using this infrastructure and logs collected from various robot platforms an accurate simulation of a robot performing a specific action can be achieved.

(a) Ceiling installation (b) Both cameras and camera computer

Figure C.5. Pictures of the final system in operation



(a) Cerberus Station (b) Camera Broadcaster Plugin

Figure C.6. User interface implemented as a plugin within the Cerberus Station

The Camera Broadcaster Plugin, shown in Figure C.6(b) provides control over multiple cameras using the generic design of a camera control interface class described in Section C.3.1. Each line of buttons controls a single camera. When the plugin is started it checks for the default device names such as */dev/video0*, which is the common convention for video device names. If any other video devices are present in the system their names follow the convention like so */dev/video1*, */dev/video2* etc. If a corresponding device is found the control buttons are enabled. If cameras are plugged in to the computer after starting the plugin, the *Not found* button can be clicked to check for individual video device names. If there are any special devices which do

not follow the conventional naming sequence, their names can be entered into the text fields of any line.

Once a camera is found, it should be initialized. During this step some information is printed on the console Cerberus Station is started, which may be helpful for some simple debugging if necessary. All cameras start in paused mode. In order to receive images from the camera *Play* button should be pressed, which appears when *Init* button is pressed. Once the frames are being received, the *Play* button is converted to *Pause* button, which may be used to pause the images without stopping the camera. *Stop* button stops the camera and terminates the camera controls.

The *Calibrate* button provides a very easy interface to the calibration utilities. As described in Section C.3.3 calibration process involves a multi-linear regression training phase, which outputs three coefficients for each target axis. These values are written in a configuration file and read by the Camera Broadcaster Plugin during the start up. These values can be generated automatically using the *Calibrate* button for each camera. The internal process of the calibration procedure is described below.

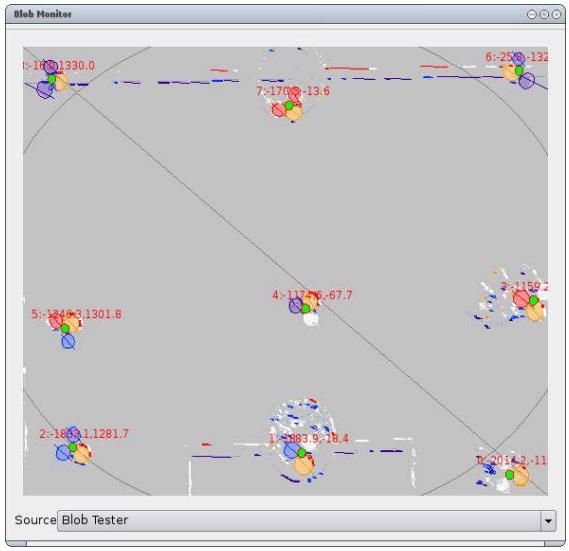
The field in our laboratory has labels on it which indicate particular coordinates. Each one of the nine markers are assigned to specific labels. Before the calibration process starts the markers are placed on to their respective labels on the field as seen in Figure C.7(b). The overhead camera receives images of the setup (Figure C.7(c)) and records a number of images. Markers on these images are detected (Figure C.7(d)) and detected values are recorded on to the disk in a log file. Then the log file is processed by a C++ code, which generates the input for the Matlab multi-linear regression training. The training procedure matches the label coordinates in field coordinates with the detected markers in the image coordinates, generating the new coefficients and recording them into the corresponding calibration file.

Thus calibration is as easy as placing the markers on the field, pressing the calibration button and restarting the Camera Broadcaster plugin. With such a convenient method available any changes in the camera configuration can be easily handled.

(a) Aibo robot with marker attached on its back

(b) Markers setup for calibration

(c) All markers setup for calibration observed by the overhead camera



(d) Corresponding detected markers

Figure C.7. Different views of several robots with markers attached

C.4.3. Data Generation

There are several methods by which data can be generated using the controls available through the control elements at the top of the Camera Broadcaster Plugin. The two primary methods are sending raw images or sending processed messages to the various destinations.

Images can be sent to test the color calibration procedure results by selecting the *Emit images* checkbox. In such a case, any widget listening to the *CameraManX*, where X is the number of the associated camera, can receive raw images for further processing.

An example is the *Blob Tester* plugin, which receives raw images and produces blobs and markers found on the given image and broadcasts the results. These results then can be viewed with the *Blob Monitor* plugin, as shown on Figure C.7(d). Images may also be observed through the *Image Listener* plugin. If recording emitted images is also required, *Record* checkbox beside the *Emit images* checkbox can be checked to enable logging to the disk.

For most experimentation purposes receiving the detected marker information is more suitable. Similar to the sending image case, detected marker messages can also be sent by the messages and recorded on the local disk. Unlike the emitting images, markers can be beamed to other network destinations, such as a remotely working Cerberus Station or another robot walking on the field. To send marker messages over the network *Beam marker messages* checkbox can be used, *Record* checkbox next to the *Beam marker messages* checkbox can be used to record messages to the local disk.

All messages sent by the Camera Broadcaster Plugin can be sent in any required time interval through the text box available at the top of the widget. The primary limit on the message speed is the camera fps, which is 30 fps with the current cameras. If images are viewed over a network connection the speed can be much slower due to the slow network connection speed. However as mentioned earlier, the experiments typically do not require raw images, they use only the detected markers messages, which are small messages compared to the raw images and therefore do not create any load on the network.

C.5. Specifications

The current specifications of the system meets the requirements and several experiments have been conducted successfully. The paragraphs below investigate each one of the requirements listed above.

The scaled field in our laboratory is *almost* covered fully. The cameras perfectly cover each side of the field individually. However due to the redundancy requirement

described in Section C.3.1, both cameras need to cover the middle line of the field. To provide the redundancy requirement for more robust marker detection, a marker's length of area from both sides of the field are left out of the field of view of the cameras.

Accuracy of the readings of the system are sufficient. The primary target application, the localization module generally has error rates in the order of 100 millimeters at its best performing cases. As explained in the Section C.3.3 the error rate of the overhead camera system is around 20 mm, which is accurate enough for localization experiments.

The explained infrastructure can handle any number of cameras, and almost all parameters of the cameras are configurable. The calibration utility provides easy recalibration whenever it is necessary. With the flexible underlying color calibration utilities it is quite easy to add new markers and other objects to track.

The system has been tested for extended periods of time by recording a very large image log file (1.6 GB) and observed to be working without any errors.

Total cost of the system is given in Table C.1, which is not a high cost in terms of any research equipment.

Table C.1. Total cost of the overhead camera system

Item	Cost (TL)
PCI USB interface	22
Metal bars, screws, bolts etc.	100
Two cameras	180
Total	302

APPENDIX D: ROBOTS

Two robots are used during the experiments conducted to produce results of this thesis. This section provides some information about these platforms.

D.1. Aibo Robot

Sony Corporation's Aibo entertainment robot is a very robust robotic platform primarily used by the RoboCup's SPL as a research platform until Sony stopped producing robots. The robots used by our laboratory survived four to five years without any repairs. Although mechanical parts are very robust the software platform is not as reliable. There is a great observability problem due to several points. Connectivity is the first issue. The only connection available to us is the wireless connection, which is not a reliable way to access a fragile research code. Especially in cases where the code running on the robot crashes badly, it is very hard to diagnose the problem since the robot's operating system dies along with the user code. Even if the robot does not stop working, since there is no console access to the robot. The only diagnostic method available is to print out debug statements to wireless terminal or store in the memory stick, which may or may not be available during a software crash. Closed source operating system is also a part of this problem. If the source code of the operating system was available some other interesting features could have been developed.

The robot has several sensors on board. The main sensor used by RoboCup researchers is the color camera located in front of the nose of the robot. Other available sensors are, microphones, electrostatic sensors, a few button interfaces, ir sensors, an accelerometer. In terms of output devices there is a speaker and a set of lights, which can be used for debugging purposes. There are many actuators to move the four legs, the head, tail and ears and further other details in the model information manual[40]. Figure D.1 shows the robot used in the experiments.

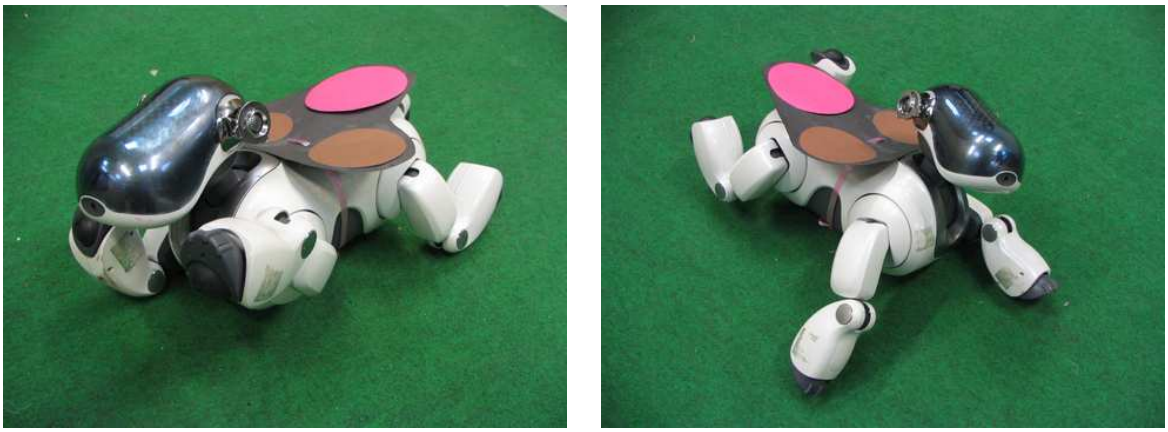


Figure D.1. Aibo robot used in the experiments

D.2. Autonomous Vehicle

The second testbed used in this thesis is the outdoor autonomous car developed during a masters thesis[34] of our laboratory. The vehicle has an on board full PC computer, which brings great flexibility to the platform. Currently the vehicle has a laser range finder and a color camera attached with room for further devices such as a GPS chip. The data collected from the camera of this platform is used as a second testbed for the proposed method. Figure D.2 shows an image the vehicle.

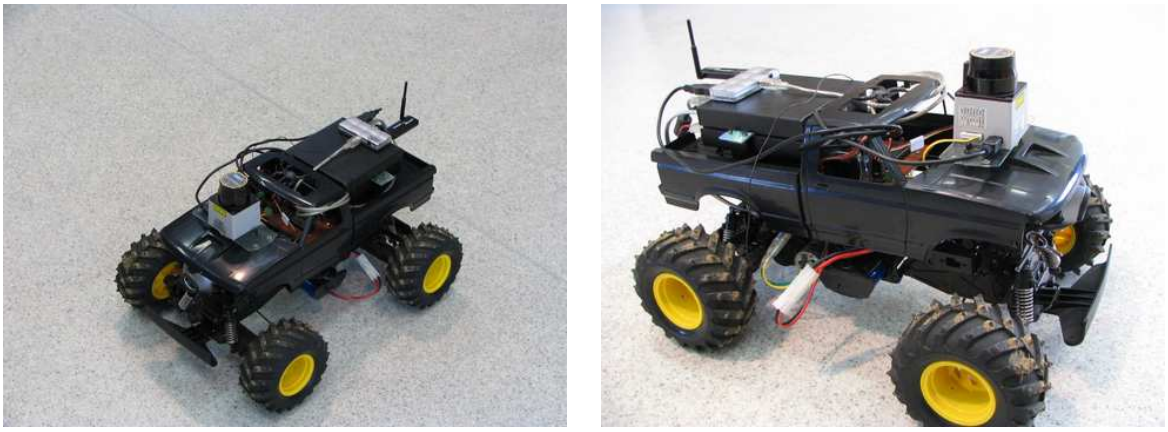


Figure D.2. Autonomous vehicle used in the experiments

REFERENCES

1. "Darpa Grand Challenge", http://www.darpa.mil/GRANDCHALLENGE/, 2009, maintained by the Defense Advanced Research Projects Agency.

2. "WebHome - WebSite - Standard Platform League", http://www.tzi.de/spl, 2009, maintained by the SPL Technical Committee.

3. "RoboCup Official Site", http://www.robocup.org/, 2009, maintained by the RoboCup Federation.

4. Akın, H. L. and et al., "Cerberus 2001 Team Report", Technical report, Boğaziçi University, 2008, available at http://robot.cmpe.boun.edu.tr/~cerberus/wiki/uploads/Downloads/Cerberus2001-TR.pdf.

5. Akın, H. L. and et al., "Cerberus 2002 Team Report", Technical report, Boğaziçi University, 2002, available at http://robot.cmpe.boun.edu.tr/~cerberus/wiki/uploads/Downloads/Cerberus2002-TR.pdf.

6. Akın, H. L. and et al., "Cerberus 2003 Team Report", Technical report, Boğaziçi University, 2003, available at http://robot.cmpe.boun.edu.tr/~cerberus/wiki/uploads/Downloads/Cerberus2003-TR.pdf.

7. Akın, H. L. and et al., "Cerberus'05 Team Report", Technical report, Boğaziçi University, 2005, available at http://robot.cmpe.boun.edu.tr/~cerberus/wiki/uploads/Downloads/Cerberus2005-TR.pdf.

8. Akın, H. L. and et al., "Cerberus'06 Team Report", Technical report, Boğaziçi University, 2007, available at http://robot.cmpe.boun.edu.tr/~cerberus/wiki/uploads/Downloads/Cerberus2006-TR.pdf.

9. Akın, H. L. and et al., "Cerberus'07 Team Description Paper", Technical re-

port, Boğaziçi University, 2007, available at `http://robot.cmpe.boun.edu.tr/~cerberus/wiki/uploads/Downloads/Cerberus2007-TDP.pdf`.

10. Akın, H. L. and et al., "Cerberus'08 Team Report", Technical report, Boğaziçi University, 2008, available at `http://robot.cmpe.boun.edu.tr/~cerberus/wiki/uploads/Downloads/Cerberus08Report.pdf`.

11. Sebastian Thrun, D. F., Wolfram Burgard, *Probabilistic Robotics*, The MIT Press Cambridge, Massachusetts London, England, 2005.

12. Choi, K. and Y. Seo, "Probabilistic Tracking of the Soccer Ball", *SMVP04*, pp. 50–60, 2004.

13. I. F. Talos, C.-F. W. e. a., L. O'Donnell, "Diffusion Tensor and Functional MRI Fusion with Anatomical MRI for Image Guided Neurosurgery", *Sixth International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI'03)*, pp. 407–415, 2003.

14. Schumitsch, B., S. Thrun, L. Guibas, and K. Olukotun, "The identity management Kalman filter (IMKF)", *Proceedings of Robotics: Science and Systems*, Philadelphia, PA, USA, August 2006.

15. Paskin, M. A., "Thin Junction Tree Filters for Simultaneous Localization and Mapping", Gottlob, G. and T. Walsh (editors), *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pp. 1157–1164, Morgan Kaufmann Publishers, San Francisco, CA, 2003.

16. Ahn, S., M. Choi, J. Choi, and W. K. Chung, "Data Association Using Visual Object Recognition for EKF-SLAM in Home Environment", *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pp. 2588–2594, Oct. 2006.

17. Nisticò, W. and T. Röfer, "Improving Percept Reliability in the Sony Four-Legged Robot League", *RoboCup*, pp. 545–552, 2005.

18. Gunnarsson, K., F. Wiesel, and R. Rojas, "The Color and the Shape: Automatic On-Line Color Calibration for Autonomous Robots", *RoboCup*, pp. 347–358, 2005.

19. Anzani, F., D. Bosisio, M. Matteucci, and D. G. Sorrenti, "On-Line Color Calibration in Non-stationary Environments", *RoboCup*, pp. 396–407, 2005.

20. Jüngel, M., "Using Layered Color Precision for a Self-Calibrating Vision System", *RoboCup*, pp. 209–220, 2004.

21. Dirk Schulz, D. F., "Bayesian Color Estimation for Adaptive Vision-based Robot Localization", *Proceedings of 2004 IEEEIRSJ International Conference on Intelligent Robots and Systems*, pp. 1884–1889, 2004.

22. Sridharan, M. and P. Stone, "Color Learning on a Mobile Robot: Towards Full Autonomy under Changing Illumination", *The 20th International Joint Conference on Artificial Intelligence*, pp. 2212–2217, January 2007.

23. Chin, R. T. and C. R. Dyer, "Model-based recognition in robot vision", *ACM Comput. Surv.*, Vol. 18, No. 1, pp. 67–108, 1986.

24. Hoiem, D., A. A. Efros, and M. Hebert, "Putting Objects in Perspective", *Proc. IEEE Computer Vision and Pattern Recognition (CVPR)*, Vol. 2, pp. 2137 – 2144, June 2006.

25. Leibe, B., N. Cornelis, K. Cornelis, and L. V. Gool, "Dynamic 3D Scene Analysis from a Moving Vehicle", *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'07)*, June 2007.

26. Reid, D., "An algorithm for tracking multiple targets", *Automatic Control, IEEE Transactions on*, Vol. 24, No. 6, pp. 843–854, Dec 1979.

27. Rasmussen, C. and G. D. Hager, "Probabilistic Data Association Methods for Tracking Complex Visual Objects", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 23, pp. 560–576, 2001.

28. Cox, I., "A Review of Statistical Data Association Techniques for Motion Correspondence", *IJCV*, Vol. 10, No. 1, pp. 53–66, February 1993.

29. Alpaydın, E., *Introduction to Machine Learning*, The MIT Press, 2004.

30. "CMPE 58K, Sp Top in CmpE: Bayesian Statistics and Machine Learning", `http://www.cmpe.boun.edu.tr/courses/cmpe58K/fall2008/`, 2009, maintained by A. Taylan Cemgil.

31. Mackay, D. J. C., "Ensemble learning for hidden markov models", Technical report, 1997.

32. Mongillo, G. and S. Deneve, "Online learning with hidden markov models", *Neural Comput.*, Vol. 20, No. 7, pp. 1706–1716, 2008.

33. "RoboCup Four-Legged League Rule Book", `http://www.tzi.de/spl/pub/Website/Downloads/AiboRules2008.pdf`, 2009, maintained by the SPL Technical Committee.

34. Daniş, S., *Development of a Multi-Sensored Autonomous Ground Vehicle*, Master's thesis, Boğaziçi University, 2009.

35. Kaplan, K., B. Çelik, T. Meriçli, Çetin Meriçli, and H. L. Akın, "Practical Extensions to Vision-Based Monte Carlo Localization Methods for Robot Soccer Domain", *RoboCup*, pp. 624–631, 2005.

36. Thomas Röfer, M. W., Tim Laue, "German Team RoboCup 2005", Technical report, Universitat Bremen, 2005.

37. Gökçe, B., *Design and Implementation of a Bipedal Walking Algorithm for Nao Humanoid Robots*, Master's thesis, Boğaziçi University, 2009.

38. Özkucur, E., *Design and Implementation of Multi-agent Visual-SLAM Algorithms on Autonomous Robots*, Master's thesis, Boğaziçi University, 2009.

39. "luvcview - QuickCam Team", `http://www.quickcamteam.net/software/linux/v4l2-software/luvcview`, 2009, maintained by Logitech video engineering team.

40. Sony Corporation, *OPEN-R SDK Model Information for ERS-7*, 2004.