Cerberus'05 Team Report

H. Levent Akın[1]
Çetin Meriçli[1]
Tekin Meriçli[2]
Kemal Kaplan[1]
Buluç Çelik[1]

Artificial Intelligence Laboratory
Department of Computer Engineering
[1]Boğaziçi University
34342 Bebek, İstanbul, Turkey
{akin, cetin.mericli, kaplanke, buluc.celik}@boun.edu.tr

[2]Marmara University
Department of Computer Engineering
Göztepe, İstanbul, Turkey
tmericli@cs.utexas.edu

November 7, 2005

Contents

Chapter 1

Introduction

The “Cerberus” team made its debut in RoboCup 2001 competition. This was the first international team participating in the league as a result of the joint research effort of a group of students and their professors from Boğaziçi University (BU), Istanbul, Turkey and Technical University Sofia, Plovdiv branch (TUSP), Plovdiv, Bulgaria. The team competed also in Robocup 2002 and Robocup 2003. Currently Boğaziçi University is maintaining the team and it is the only team participating from East Europe. This year, despite the fact that it was the only team competing with ERS-210s (not ERS210As), Cerberus won the first place in the technical challenges.

The entire software system of Cerberus was designed and developed from scratch this year. As opposed to the previous years, we began with developing a framework which makes all of our modules platform and hardware independent and allows us to transfer from or to the robot any input, output or intermediate data of the modules. This infrastructure enables us to have a considerable speed-up during development and testing. The first goal of our team was to develop a platform where we can combine research and development techniques with software engineering methodologies. After a thorough inspection of previous works of our team and other teams, we decided to divide the project into two main parts as follows:

- Cerberus Station
- Cerberus Player

In the next subsections we describe these parts.

1.1 Cerberus Station

This is the off line development platform where we develop and test our algorithms and ideas. The whole system is developed by using Microsoft .NET technologies and contains a set of monitors which enable visualizing several phases of image processing, localization, and locomotion information. We have included record and replay facilities which allow us to test our implementations without deploying the code on the robot each time. It is possible to record live images, classified images, regions found, perceived objects and estimated pose on the field in real time to a log file and replay it in different speeds or frame by frame. Cerberus Station also contains a locomotion test unit in which all parameters of the motion engine and special actions can be specified and tested remotely. For debugging purposes, a telnet client and an exception monitor log parser are also included in the station. Since each sub-module of the robot code is hardware independent, all modules can be tested and debugged in the station. This hardware and platform independence provides great savings on development time when combined with the advanced raw data logging and playback system. Cerberus Station communicates with the robots via TCP and uses a common serializable message structure for information exchange.

1.2 Cerberus Player

Cerberus Player is the part of the project that runs on the robots. Most of the classes in *Cerberus Player* are implemented in a platform independent manner, which means we can cross-compile them in various operating systems like OPEN-R, Windows or Linux. Although, robot dependent parts of the code are planned to run only on the robot, a simulation system for simulating locomotion and sensing is under development. The software architecture of *Cerberus Player* consists of four objects:

- Core Object
- Locomotion
- Communication
- Dock Object

In the following subsections we describe these objects.

1.2.1 Core Object

The main part of the player code is the *Core Object.* This object coordinates the communication and synchronization between the other objects. All other objects are connected to it. *Core Object* takes the camera image as its main input and sends the corresponding actuator commands to the locomotion engine. *Core Object* is the container and hardware interface of *Vision*, *Localization* and *Planner* modules. This combination is chosen because of the execution sequence of these modules. All of them are executed for each received camera frame and there is an input-output dependency and execution sequence that is from *vision* → *localization* → *planner*.

1.2.2 Communication Object

Communication Object is responsible for receiving game data from the game controller and managing robot-robot communication. Since the *TCPGateway* object which was the only communication interface allowed so far will no longer be supported, both the game controller and robot-robot communication infrastructure have been rewritten. They both use UDP as the communication protocol.

1.2.3 Dock Object

Dock Object is the object which manages the communication between a robot and the *Cerberus Station*. It redirects the received messages to *Core Object* and sends the debug messages to the station. *Dock Object* uses TCP to send and receive serialized messages to and from *Cerberus Station*.

Chapter 2

Vision Module

The Vision module is responsible for information extraction from received camera frame. Image processing starts with receiving a camera frame and ends with an egocentric world model consisting of a collection of visual percepts as shown in Fig. 2.1. The Vision module is written from scratch this year and several novel approaches for image processing have been introduced.

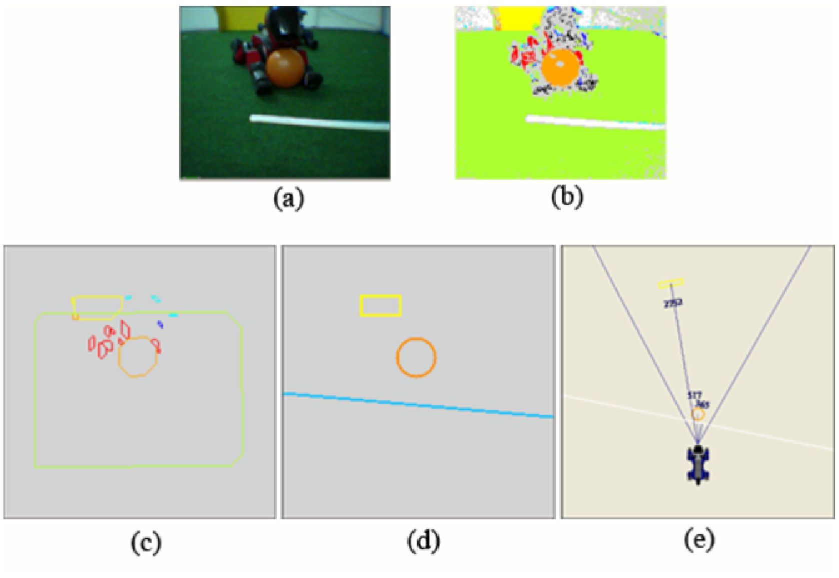


Figure 2.1: Phases of image processing. a) Original image, b) Color classified image, c) Found blobs, d) Perceived objects e) Egocentric view

2.1 Color Classification

First, we have decided not to use previously implemented color classification methods like decision trees and nearest neighbor [1]. Instead, we have implemented a Generalized Regression Network (GRNN) [2] for color generalization. After labeling a set of images with the proper colors, a GRNN is trained with the labeled data. After the training phase, the network is simulated for the input space to generate a color lookup table for four bits (16 levels) of Y, six bits (64 levels) of U and six bits of V. The resultant color lookup table is very robust to luminance changes and allows our vision system to work without using any kind of extra lights other than the standard ceiling fluorescents. This is proven by being one of the few teams which managed to score a goal in the variable lightning challenge.

2.2 Finding Regions

This sub-module is responsible for processing a labeled image and extracting the potentially significant regions for the perception sub-modules.

We use a new approach for this sub-module. Instead of using run length encoding (RLE), we use an optimized region growing algorithm that performs both connected component finding and region building operations at the same time. This algorithm works nearly two times faster than the well known RLE-Find connected components-build regions approach.

The approach which uses RLE first runs RLE on the image, connects the runs to find the connected components, filtering out potentially insignificant components, and finally rotating the regions. Fig. 2.2 shows an RLE sample.

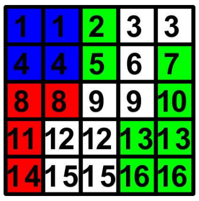


Figure 2.2: An RLE sample

Our new approach, on the other hand, uses a region growing algorithm directly on the raw image. Starting from leftmost top pixel, each pixel is processed in the following manner:

- If it was not labeled, it receives a new label
- It is compared with its consecutive pixel on the right. If they are of the same color and the consecutive pixel is labeled with a different label from the pixel which is being processed, then the labels of both

pixels are noted to be united. If they are of the same color and the consecutive pixel has no label, then the consecutive pixel is labeled with the same label with the pixel which is being processed.

- It is compared with its consecutive pixel at the bottom. If they are of the same color, then the consecutive pixel is labeled with the same label with the pixel which is being processed.

At this point, the sub-regions and notes indicating the subregions to be combined are prepared in just one pass on the image. Next, the sub regions are combined according to the prepared notes. Fig. 2.3 shows a sample with the new approach.

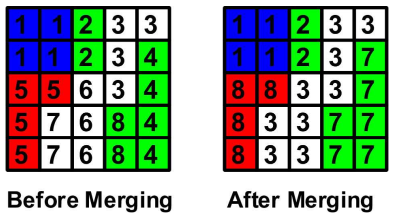


Figure 2.3: A sample with the new approach

After obtaining the combined sub-regions, they are filtered for significant regions and rotated as in other approaches.

2.3 Line Perception

Line perception process is an important part of the vision module, since it provides important information for the localization module. The sample images from line perception process are shown in Fig. 2.4. The proposed approach is as follows:

- Hough transform is applied on the white pixels which are close enough to green pixels using Robert's Cross on their Y band as the first operation.

- Two thresholds are used to check each entry in the table prepared in Hough transform. The first threshold is the minimum acceptable value for the line's entry, whereas the second one is minimum acceptable value for the sum of entries of the line and its neighbors. For a line entry to be accepted, it should also be a local maximum.

- For a chosen line, to decrease the quantization error, the weighted average of its angle and the perpendicular distance are taken, where weights are the values in the Hough transform table.

- Now, we have a more or less fine tuned line, but it is still the border of the field line. Especially in images where field lines are close to the camera, the lines occupy a thick region. The selected line is shifted along its normal vector orientation. The amount and the direction of the shift are calculated by following the normal line at different intervals.

- The lines are rotated according to the pan and the tilt of the camera.

- Then, the lines are mapped to the real 3D field using geometrical transformations.

- Once the lines are mapped to the field, they are still relative to the camera. As they need to be relative to the chest of the robot, they are transformed accordingly.

- Finally, extra copies of the same line, which is a rare but possible situation, are eliminated.

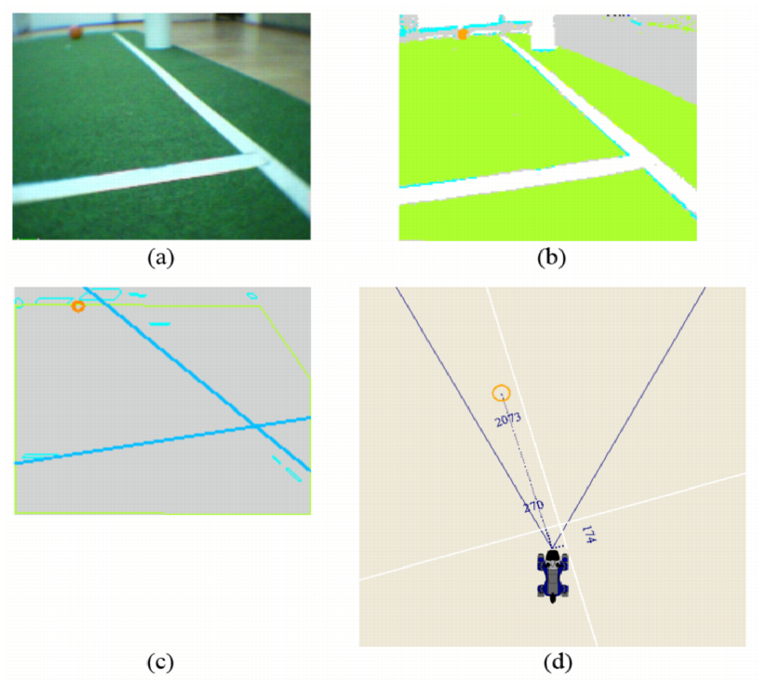


Figure 2.4: Phases of Line Detection. a) Original image, b) Color classified image, c) Perceived lines e) Egocentric view

2.4 Object detection

The classified image is processed in order to obtain blobs. Here, we use a new approach. Instead of using run length encoding (RLE), we use an optimized region growing algorithm that performs both connected component finding and region building operations at the same time. This algorithm works nearly two times faster than the well known *RLE → Find connected components → build regions* approach. Another novel approach used is the concept of a *bounding octagon of a region.* Since the robot must turn its head in order to expand its field of view, it is necessary to rotate the obtained image according to the actual position of the head. However, since rotation is a very expensive operation, it is not wise to rotate the entire image. For this reason typically only the identified regions are rotated. Since octagons are more appropriate for rotation than rectangular boxes, using octagons instead of boxes to represent regions reduces the information loss due to rotation.

Our vision module employs a very efficient partial circle fit algorithm for detecting partially occluded balls and the balls which are on the borders of the image as shown in Fig. 2.5. Since accuracy in the estimation of ball distance and orientation is needed mostly in cases where the ball is very close and it is so often that the ball can only be seen partially in such cases, having a cheap and accurate ball perception algorithm is a must. The equation for estimating the ball radius is

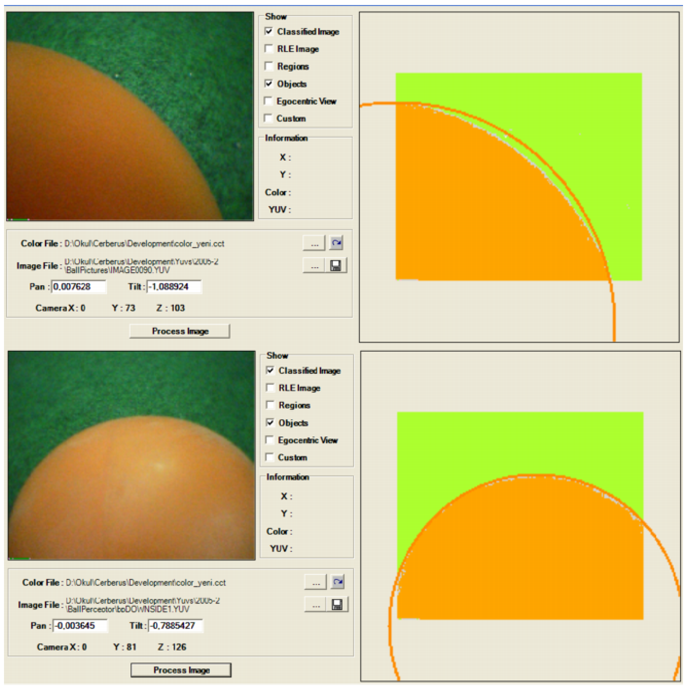
$$r = \frac{h}{2} + \frac{s \times s}{8 \times h} \tag{2.1}$$

where r is the radius, h is the height of the ball region and s is the width of the ball region. Although this estimation can be used for different types of ball segments (i.e ball segments in different parts of the captured image), the ball center estimation requires separate handling of different partial image conditions.

The recognition of objects on the field is based on objects' colors and sanity checks performed on the candidate regions. The sanity check process has three phases.

- **Phase 1.** The candidate region should satisfy object specific precondition checks. For example, the lower edge of the goal or the lowest point of the ball should not be outside of the field region. Of course this control requires a field region (i.e. a merged green region classified as the field) and a threshold value which can be modified for each object type.

- **Phase 2.** A probability value is assigned for the candidate field. The probability calculation is based on the properties of the object. For



Color File : D:\Okul\Cerberus\Development\color_yeni.cct
Image File : D:\Okul\Cerberus\Development\Yuvs\2005-2\BallPictures\IMAGE0090.YUV
Pan : 0,007628 Tilt : -1,088924
Camera X : 0 Y : 73 Z : 103

Color File : D:\Okul\Cerberus\Development\color_yeni.cct
Image File : D:\Okul\Cerberus\Development\Yuvs\2005-2\BallPerceptor\boDOWNSIDE1.YUV
Pan : -0,003645 Tilt : -0,7885427
Camera X : 0 Y : 81 Z : 126

Figure 2.5: Two examples of detecting partial balls via circle fit

example, distance between regions, with respect to the region sizes, has an important effect on probability of being a pair of beacon regions.

- **Phase 3.** Some postconditions checks are performed on the surviving candidate regions. The first postcondition check is usually the probability thresholds which prun

The final checks for each object is well-documented in the source codes which can be accessed from our team's web page.

The vision module is one of the fastest vision systems having the features described above developed on AIBOs. On our current robots (ERS-210 with 200 MHz processor) a frame is processed in approximately 50 ms which provides a 20 frames per second speed.

Chapter 3

Localization

Localization is one of the main research interests of our research group. We have a number of different localization engines [3, 4, 5, 6]. We have used our most recent work (which later turned into an M.Sc. thesis) in the competition called *S-LOC*. In this section, *S-LOC* and history based egocentric world modeling approach called *My Environment* are presented.

3.1 My Environment

My Environment was initially designed as a part of the localization module and aimed to increase the performance of localization. It was then decided to be a separate module such that not only the localization module, but the other modules could also benefit from its output.

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

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

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

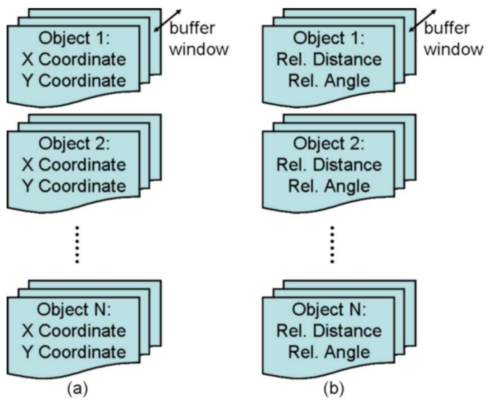


Figure 3.1: Buffering the poses of objects: (a) Buffering actual poses, and (b) Buffering the relative poses

Although the cost of buffering relative poses of objects is greater than the cost of buffering actual poses of objects in time, buffering relative poses is more robust since it does not involve localization. Involving localization in the calculation results in involving localization error in the output. For each stored value on an object's pose history, having instantaneous localization error added to the perception error, the estimations on the current pose of the object would suffer from more noise.

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

3.1.1 General Outline of ME

My Environment (ME) is a module between the perception module, or any other module that handles the perception of the environment objects, and the other modules that use the output of the perception module as shown in Figure 3.2. In some exceptional cases, where the position and the relative angle data are not sufficient, the perception data may be needed to be used directly. For instance, in the robot soccer domain, the ball tracking behavior for the head, where the coordinates of the perceived ball region on the camera frame image may be used to calculate the next pan - tilt parameters, uses the perception output directly.

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

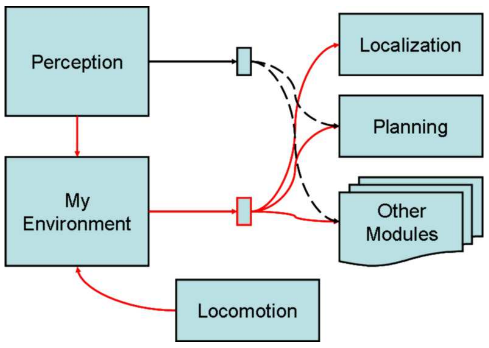


Figure 3.2: Interaction of ME with the other modules

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

By filtering the output of the perception module and using the past perceptions of the objects at the same time, more stable and robust data could be provided for the localization module as well as other modules. This way the effect of false perceptions and recognitions would be decreased.

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

3.1.2 Architecture of ME

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

Each static object entry has a distance, a relative angle and a confidence regarding its perception. In the case when a static object has its own orientation, the orientation values are also to be stored in the buffer. For dynamic objects, the velocity should also be calculated, but as it is not directly extracted from a single camera frame, it is not necessarily be buffered. In Figure 3.4a and 3.4b the data structures for static and dynamic objects are defined.

The window size is a hyper-parameter of ME. The noise in odometry, the dynamicity of the agent's pose, and the frequency of the processed vi-

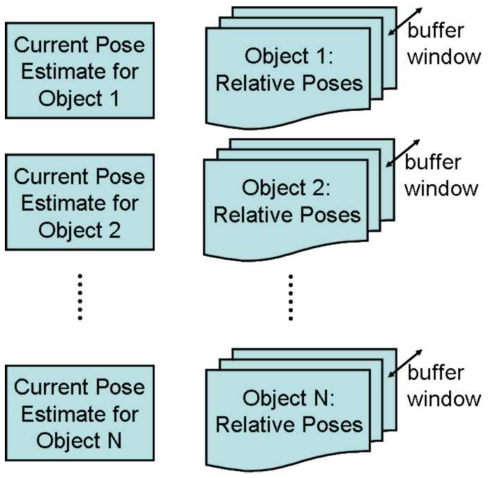


Figure 3.3: The data structure of ME

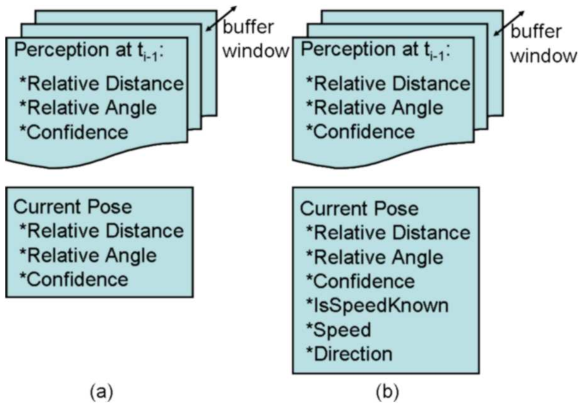


Figure 3.4: Data structures of (a) static objects, and (b) dynamic objects

sion frames play an important role in the selection of a good window size parameter. For dynamic objects, the speed of such objects should also be taken into account.

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

3.1.3 Procedures of ME

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

Initialization

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

Perception Update

For each dynamic and static object, the oldest record on the buffer is deleted and a new record is stored from perception if the object is seen at the moment, otherwise it is marked that there is information available about the pose of the object at that time. After updating the buffer with the latest perception output, an estimation is done for each object concerning its pose by calling the appropriate procedure in Section 3.1.3 or Section 3.1.3.

Odometry Update

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

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

$$\begin{aligned}
c &= cos(\theta) \times d - \Delta x && (3.1)\\
s &= sin(\theta) \times d - \Delta y && (3.2)\\
d' &= \sqrt{c^2 + s^2} && (3.3)\\
\theta' &= tan^{-1}(s/c) + \Delta\theta && (3.4)
\end{aligned}$$

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

Current Pose Estimation for Static Objects

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

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

$$\begin{aligned}
w_j &= \sum_{i=0}^{n_w} KA_i^j \times CA_i^j \times f_{ws}(i) && (3.5)\\
\Delta x_j &= \frac{\sum_{i=0}^{n_w} KA_i^j \times DA_i^j \times cos(AA_i^j) \times f_{ws}(i)}{w_j} && (3.6)\\
\Delta y_j &= \frac{\sum_{i=0}^{n_w} KA_i^j \times DA_i^j \times sin(AA_i^j) \times f_{ws}(i)}{w_j} && (3.7)
\end{aligned}$$

$$\begin{aligned}
n_j &= \sum_{i=0}^{n_w} KA_i^j && (3.8)\\
c_j &= \frac{\sum_{i=0}^{n_w} KA_i^j \times CA_i^j \times f_{ws}(i)}{w_j} \times f_{wc}(n_j) && (3.9)\\
d_j &= \sqrt{\Delta x_j^2 + \Delta y_j^2} && (3.10)\\
\theta_j &= tan^{-1}(\Delta y_j / \Delta x_j) && (3.11)
\end{aligned}$$

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

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

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

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

Current Pose Estimation for Dynamic Objects

The procedure of the current pose estimation for the dynamic objects is the same as the current pose estimation for the static objects except that for dynamic objects the speed and the direction of the speed should also be calculated when possible. For each dynamic object, this function is called once in every *perception update*. Using the same equations in Section 3.1.3 the pose estimation, with the exception of the speed related variables, is performed, but the $f_{ws}(i)$ function is replaced with $f_{wd}(i)$, which gives the weight of the i^{th} record for the dynamic object.

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

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

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

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

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

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

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

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

3.1.4 Advantages and Disadvantages of ME

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

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

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

It should also be noted that using ME, the agent would observe dynamic objects slower than they are. This is because of using the previous poses of the dynamic object in the calculation of the current pose. This problem could be minimized theoretically by adding the velocity of the previous estimation times the time passed to the previous records of that object, but this could bring more noise than it makes corrections as the velocity estimations could be noisier than the relative position estimations.

3.2 S-LOC: Simple Localization

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

- Triangulation is a simple and accurate technique, but is not robust. It is too much effected by noise, specially by the false perceptions [7].
- The major disadvantage of Kalman Filter methods is that they do not have the capability of recovering from kidnapping [10, 11].
- ML approaches are generally expensive, where false perceptions could be big problems [12, 13].
- Raw MCL cannot recover from kidnapping, but a version of it, SRL is implemented [14, 15, 16].
- ML-EKF is also another expensive technique, which would not be preferred in a case where a much lower cost algorithm could give accurate and robust results [17, 18].
- Fuzzy localization techniques generally have high computational complexity, and do not give results with enough accuracy that are worth the cost [8, 9].
- R-MCL is also another technique, which is used in the experiments for comparison purposes [3, 4, 5].

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

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

Perception update, as it depends on the perceptional information, usually includes high amount of noise. Although the agent is dynamic, its pose should not be highly unstable, i.e. the pose should not jump to different

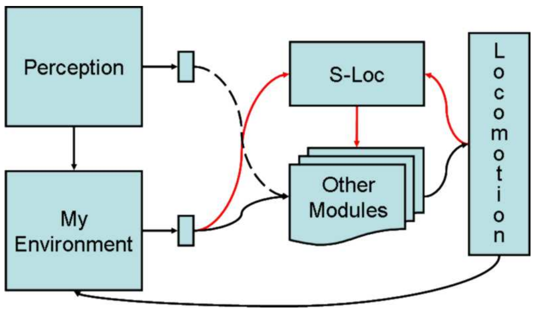


Figure 3.5: The relationship of the S-Loc module with the other modules

poses that are far away on the field frequently. Using a memory for the previous pose estimate, and updating it with the current estimate could handle the big fluctuations and increase the robustness to the false perceptions of static landmarks.

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

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

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

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

3.2.1 General Outline of S-Loc

S-Loc is a localization module. It needs the perception data and the odometry data for updating the pose estimate, which it provides as the output. This pose estimate is then used in other modules. The relationship of the *S-Loc* module with the other modules is shown in Figure 3.5.

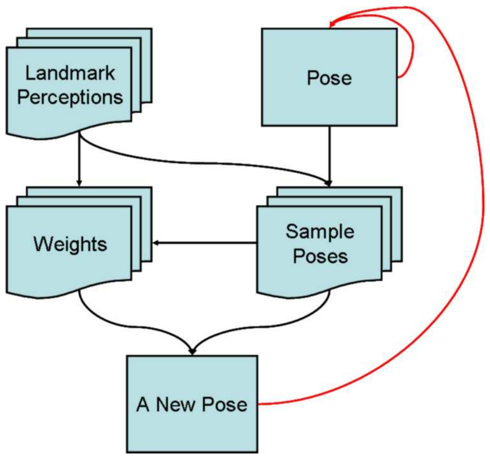


Figure 3.6: The perception update process

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

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

3.2.2 Architecture of S-Loc

The perception update of the S-Loc depends on the perception of landmarks and the previous pose estimate. The perception update process is shown in Figure 3.6. Even if the initial pose estimate is provided wrong, it acts as a kidnapping problem and is not a big problem as S-Loc will converge to the actual pose in a short period of time if enough perception could be obtained during this period.

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

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

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

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

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

In the case of having no perception at a certain time, the current pose estimate could be obtained by decreasing the confidence of the previous pose estimate.

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

3.2.3 Procedures of S-Loc

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

Initialization

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

Perception Update

As shown in Equations 3.15, 3.16, 3.17, 3.18 and 3.19, the first pose sample is the previous pose estimate.

$$PSA_x^0 = PE_x \tag{3.15}$$
$$PSA_y^0 = PE_y \tag{3.16}$$
$$PSA_\theta^0 = PE_\theta \tag{3.17}$$

$$
\begin{aligned}
PSA_c^0 &= PE_c && (3.18)\\
PSA_w^0 &= PE_c \times f_{wpu2}(PSA_x^0, PSA_y^0, PSA_\theta^0, PA) && (3.19)\\
PA_k^0 &= 1 && (3.20)
\end{aligned}
$$

where PS_x^0, PS_y^0, PS_θ^0, PS_c^0 and PS_w^0 are the x-coordinate, y-coordinate, orientation, confidence and weight of the first pose sample; PE_x, PE_y PE_θ, and PE_c are the x-coordinate, y-coordinate, orientation and confidence of the pose estimate before the perception update; PA, percepts array, is the collection of perception data of all the perceived landmarks together with their coordinates that are known initially; f_{wpu2} is the function that returns a weight component for a pose according to the current perceptions; and PA_k^0 is set to one in order to have the first element of pose sample array included in the proceeding calculations.

Then, a separate pose sample is calculated for each perception as in the Equations 3.22, 3.23, 3.25, 3.26 and 3.27.

$$
\begin{aligned}
\alpha_i &= tan^{-1}\left(\frac{PE_y - PA_y^i}{PE_x - PA_x^i}\right) && (3.21)\\
PSA_x^i &= PA_x^i + PA_d^i \times cos(\alpha_i) && (3.22)\\
PSA_y^i &= PA_y^i + PA_d^i \times sin(\alpha_i) && (3.23)\\
\beta_i &= tan^{-1}\left(\frac{PSA_y^i - PA_y^i}{PSA_x^i - PA_x^i}\right) && (3.24)\\
PSA_\theta^i &= \pi + \beta_i - PA_\theta^i && (3.25)\\
PSA_c^i &= PA_c^i && (3.26)\\
PSA_w^i &= f_{wpu1}(PA^i) \times f_{wpu2}(PSA_x^i, PSA_y^i, PSA_\theta^i, PA) && (3.27)
\end{aligned}
$$

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

The function f_{wpu1} returns the first component of the PSA_w^i for the argument PA^i. It may return PA_c^i directly or any other number that gives the confidence that the perception is correct. Since the accuracy of the pose sample will be taken into account by the function f_{wpu2}, this function is independent of the corresponding pose sample. The purpose of this function is to decrease the weight of the pose samples, of which the perception is less confident. In the case where the perception module does not provide healthy confidence values, the perceived relative distance of the landmark can be used for the calculation of the return value. In such a case, a properly configured sigmoid function can be very suitable. If the landmarks are of different types and are known to have different perception accuracy, then this could also be reflected on the return value.

The function f_{wpu2} returns the second component of the PSA_w^i. The return value is related to the accuracy of the pose sample according to all the perceived landmarks. For each perceived landmark, the position of the perceived landmark is calculated by adding the perceived distance on the perceived relative angle to the pose sample, and the resulting position is compared to the actual position of the landmark. The difference gives the error. The return value should be a function of the error as in Equation 3.30.

$$par_x^j = \left| PA_x^j - (PSA_x^i + PA_d^j \times cos(PSA_\theta^i + PA_\theta^j)) \right| \quad (3.28)$$

$$par_y^j = \left| PA_y^j - (PSA_y^i + PA_y^j \times sin(PSA_\theta^i + PA_\theta^j)) \right| \quad (3.29)$$

$$f_{wpu2} = \prod_{j=1}^{N_L} PA_k^j \times f_{wpu3}\left(par_x^j, par_y^j\right) \quad (3.30)$$

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

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

The calculation of the new pose estimate is the last step of the perception update. Except the new orientation estimate, all the estimation values are the weighted average of the recent calculation, which is in turn a weighted average of sample poses, and the corresponding previous estimate value. The new orientation estimate is calculated by using the new coordinate estimates and the perceptions. The new values of pose estimate are calculated from the Equations 3.33, 3.34, 3.37, and 3.38.

$$hp = f_{HP}\left(\sum_{j=1}^{N_L} PA_k^j\right) \quad (3.31)$$

$$tw = \sum_{j=0}^{N_L} \left(PA_k^j \times PSA_w^j\right) \quad (3.32)$$

$$PE_x^* = hp \times PE_y + (1 - hp) \times \frac{\sum_{j=0}^{N_L} \left(PA_k^j \times PSA_x^j \times PSA_w^j \right)}{tw} \quad (3.33)$$

$$PE_y^* = hp \times PE_y + (1 - hp) \times \frac{\sum_{j=0}^{N_L} \left(PA_k^j \times PSA_y^j \times PSA_w^j \right)}{tw} \quad (3.34)$$

$$\beta_i = tan^{-1} \left(\frac{PE_y^* - PA_y^i}{PE_x^* - PA_x^i} \right) \quad (3.35)$$

$$wa_i = f_{wpu1}(PA^i) \times f_{wpu2}(PE_x^*, PE_y^*, \beta_i, PA) \quad (3.36)$$

$$PE_\theta^* = tan^{-1} \left(\frac{\sum_{i=1}^{N_L} \left(PA_k^i \times sin(\beta_i) \times wa_i \right)}{\sum_{i=1}^{N_L} \left(PA_k^i \times cos(\beta_i) \times wa_i \right)} \right) \quad (3.37)$$

$$PE_c^* = hp \times PE_c + (1 - hp) \times \frac{\sum_{j=0}^{N_L} \left(PA_k^j \times PSA_c^j \times PSA_w^j \right)}{tw} \quad (3.38)$$

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

Odometry Update

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

It should also be noted that, in the case where the frequency of the odometry update is much less than the frequency of the perception update, transforming the odometry data to lower values may lead to better results since the original odometry data is the result of the agent's motion from the previous odometry update to the current one, and this would last for more than one perception updates.

In Equations 3.39, 3.40 and 3.41 the new (updated) coordinates and orientation of the pose estimate is calculated.

$$PE_x^* = PE_x + \Delta x \times sin(PE_\theta) + \Delta y \times cos(PE_\theta) \quad (3.39)$$

$$PE_y^* = PE_y + \Delta y \times sin(PE_\theta) - \Delta x \times cos(PE_\theta) \quad (3.40)$$

$$PE_\theta^* = PE_\theta + \Delta\theta \quad (3.41)$$

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

3.2.4 Advantages and Disadvantages of S-Loc

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

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

In comparison with triangulation, S-Loc does not calculate the best estimate according to the perception of the moment, but makes the estimation in a way that it converges to that point in a short period of time. On the other hand, the effect of the false perceptions is greatly decreased as the sample pose of such a perception would have a relatively small confidence and will not play a big role in the pose estimation. In this way, the robustness is increased without decreasing the performance.

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

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

Chapter 4

Planning

4.1 Potential Field Planner

We have used finite state machines (FSM) in our planner module in the previous years. However, since the soccer domain is a continuous environment and the sensory input, which determines the state transitions, is quite noisy, discrete planners like FSM suffers from oscillations and quantization errors. Therefore, we adopted a modified version of Potential Field Planner (PFP) previously used for wheeled mobile agents in our lab [19]. Potential Fields is a method of planning robot trajectories based on combining the vector fields induced by mapped objects, such as repulsive fields for obstacles and attractive fields for goals. In the implementations of potential fields for robot control, only the force vector for the current robot position is calculated for each object or schema. The sum of these vectors is then used to control the instantaneous motion of the robot. The process is then repeated for the next robot position to handle any new perceptual information that may have become available.

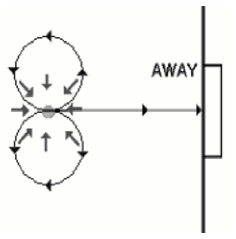


Figure 4.1: Ball Field

Most of the objects on the field require only an impulsive force for preventing collision. Nevertheless, we use an attractive force to move a robot to a specific location. On the other hand, the attractive simple fields use a constant force directed to the center of the field. The field generated by the ball is the combination of two potential fields. The first one is a simple attractive field. The second one is a special circular field located above and below the line between the ball and the center of opponent goal as shown in Fig.4.1. The direction of the field is perpendicular to the line segment between the robot and the center of the nearest circle. On the other hand, the magnitude of the simple attractive field is constant.

4.2 Task Allocation

Although PFP approach provides a fast reactive controller for a single agent, soccer is a cooperative game and the robots should cooperate and collaborate with the teammates. For this purpose, we employ a market driven task allocation scheme [20, 21, 22]. In this method, the robots calculate a cost value (their fitness) for each role. The calculated costs are broadcasted through the team and based on a ranking scheme, robots chose most appropriate role for their costs. Here, each team member calculates costs for its assigned tasks, including the cost of moving, aligning itself suitably for the task, and cost of object avoidance, then looks for another team member who can do this task for less cost by opening an auction on that task. If one or more of the robots can do this task with a lower cost, they are assigned to that task, so both the robots and the team increase their profit. Other robots take actions according to their cost functions (each takes the action which is most profitable for itself). Since all robots share their costs, they know which task is appropriate for each one so they do not need to tell others about their decisions and they do not need a leader to assign tasks. If one fails, another would take the task and go on working.

The approach is shown in the flowchart given in Fig. 4.2. The robot with the smallest score cost C_{ES} will be the primary attacker. Similarly the robot, except the primary attacker, with the smallest C_{def} cost will be the defender. If C_{auc} is higher than all passing costs ($C_{bid(i)}$) then the attacker will shoot, else, it will pass the ball to the robot with the lowest $C_{bid(i)}$ value. The cost functions used in the implementations are as follows:

$$\begin{aligned}
C_{ES} &= \mu_1.t_{dist} + \mu_2.t_{align} + \mu_3.cl_{goal} && (4.1)\\
C_{bid(i)} &= \mu_1.t_{dist} + \mu_2.t_{align} + \mu_3.cl_{teammate(i)} + C_{ES(i)}, i \neq id && (4.2)\\
C_{auc} &= C_{ES(id)} && (4.3)\\
C_{def} &= \mu_5.t_{dist} + \mu_6.t_{align} + \mu_7.cl_{defense} && (4.4)
\end{aligned}$$

where id is the id of the robot, t_{dist} is the time required to move for specified distance, t_{align} is the time required to align for specified amount, μ_i are the weights of several parameters to emphasize their relative importance in the total cost function, cl_{goal} is the clearance from the robot to goal area-for object avoidance, $cl_{defense}$ is the clearance from the robot to the middle

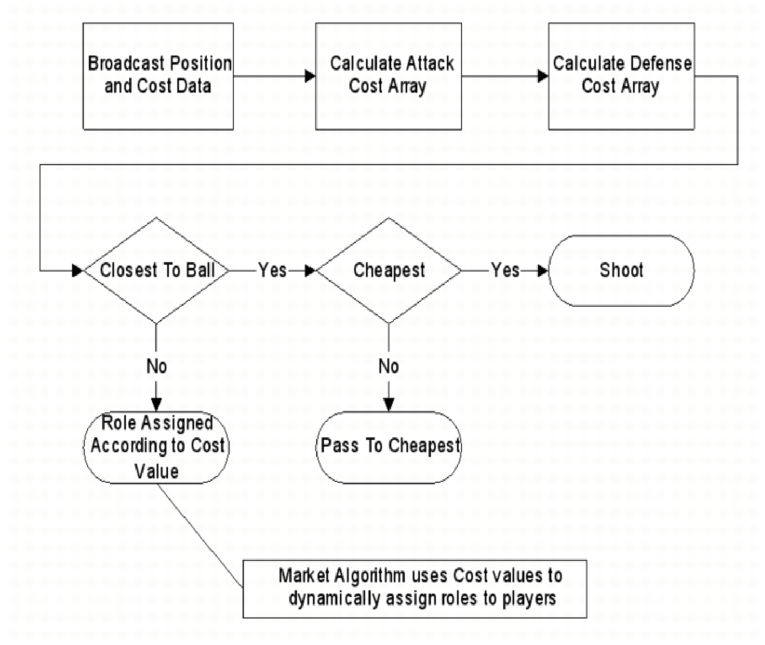

Broadcast Position and Cost Data
Calculate Attack Cost Array
Calculate Defense Cost Array
Closest To Ball
Yes
Cheapest
Yes
Shoot
No
No
Role Assigned According to Cost Value
Pass To Cheapest
Market Algorithm uses Cost values to dynamically assign roles to players

Figure 4.2: Flowchart for task assignment

point on the line between the middle point of own goal and the ball-for object avoidance, and similarly $cl_{teammate(i)}$ is the clearance from the robot to the position of a teammate. Each robot should know its teammates score and defense costs. In our study each agent broadcasts its score and defense costs. Since the auctioneer knows the positions of its teammates, it can calculate the $C_{bid(i=id)}$ value for its teammates.

The game strategy can easily be changed by changing the cost functions in order to define the relative importance of defensive behavior over offensive behavior, and this yields greater flexibility in planning, which is not generally possible.

Chapter 5

Motion

This year, ParaWalk engine, which is developed by rUNSWift 4-legged robot soccer team of UNSW, was used until the development of our own motion engine is completed [23]. Also the qualification video was shot with ParaWalk because of the ongoing testing of the new motion engine.

At the lowest level, the Aibo's gait is determined by a series of joint positions for the three joints in each of its legs. More recently, trajectories of the Aibos four feet through three-dimensional space have been used to develop a higher level representation for Aibo's gait. An inverse kinematics calculation is then used to convert these trajectories into joint angles. Among higher-level approaches, most of the differences between gaits that have been developed for the Aibo stem from the shape of the loci through which the feet pass and the exact parameterizations of those loci.

5.1 Kinematic Model

At the position level, the problem is stated as, "Given the desired position of the robot's paw, what must be the angles at all of the robots joints?".

In our approach, inverse kinematics techniques are used to calculate the desired joint angles while the paw is moving along the path determined by the locus of the leg. The locus is divided into *pStep* (to be explained in Section 5.5) points, and each of these points has (x,y,z) values. When the paw is to move to the location of the next point on the locus, the following formulas are used to calculate the necessary joint angles in order to make the paw move towards this point. A simple representation of an Aibo leg is illustrated in Figure 5.1.

The law of cosines is used for calculating the knee angle (θ_3).

$$d^2 = x^2 + y^2 + z^2 \tag{5.1}$$

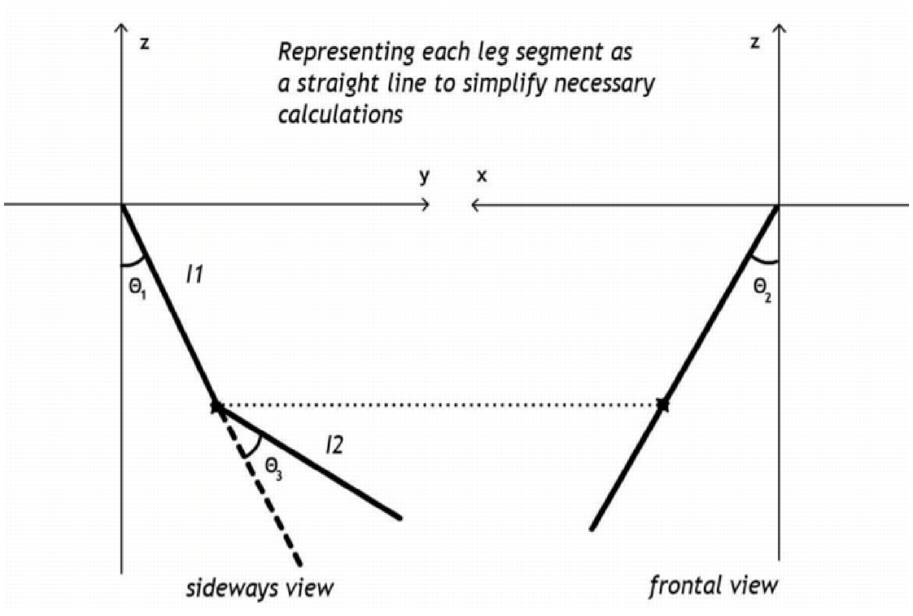

Representing each leg segment as a straight line to simplify necessary calculations
z
y
x
z
Θ1
l1
Θ2
l2
Θ3
sideways view
frontal view

Figure 5.1: Simple kinematic model representation for front right leg.

where (x, y, z) represents the 3d coordinates of the paw according to the shoulder.

$$I_1{}^2 + I_2{}^2 - 2I_1I_2\cos(\pi - \theta_3) = d^2 \tag{5.2}$$

$$\theta_3 = \pi - \arccos(\frac{I_1{}^2 + I_2{}^2 - d^2}{2I_1I_2}) \tag{5.3}$$

Then the abductor angle θ_2 is calculated.

$$\theta_2 = \arcsin(\frac{x}{I_1 + I_2\cos(\theta_3)}) \tag{5.4}$$

Finally, rotator angle θ_1 is calculated.

$$\theta_1 = \frac{y\cos(\theta_2)(I_1 + I_2\cos(\theta_3)) + zI_2\sin(\theta_3)}{yI_2\sin(\theta_3) - (z\cos(\theta_2)(I_1 + I_2\cos(\theta_3)))} \tag{5.5}$$

5.2 Walking Styles

To produce a walking motion, the legs must not be at the same position on the walk locus at the same time. Essentially, the legs must move out of phase of each other. Human walking actually uses a similar approach. One leg is lifted, moved forward, and then dropped, while the other stays where it was. Once the first step has been taken, the other leg is then lifted and basically mirrors the same action taken by the first leg.

Different gait types can be obtained by shifting the movement phases of each leg in different manners. Timing of each leg and resulting walking type is shown in Figure 5.2

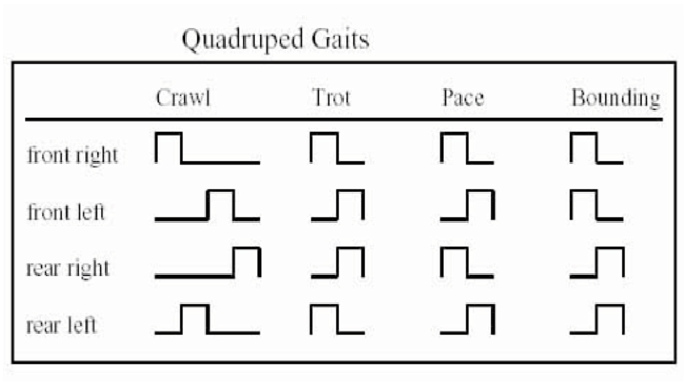


Figure 5.2: Different timing of each legs motion results in different walking styles [24].

5.3 Omnidirectional Motion

Omnidirectional walking can be thought as the motion of a shopping cart, and can be obtained by treating the legs as wheels. This is illustrated in Figure 5.3.

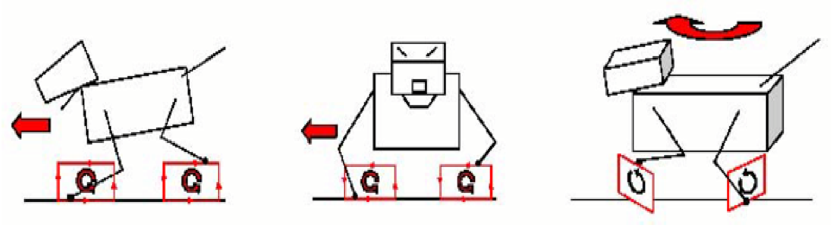


Figure 5.3: Using legs as the wheels of a shopping cart [23].

In order to achieve this motion, 3 walk components named *forward*, *sideways*, and *turn* are used. These components are represented as 2-dimensional vectors and they are added vectorally in order to obtain one resulting vector and its symmetric part according to the initial paw location. These two resulting vectors together produces the limits of the locus; that is the limits of each leg's area of operation. The body of the robot can be approximated as a rectangle from the top view and the angle of the turn components can be calculated as the arctan($bodyWidth/bodyHeight$). The operation of obtaining locus limits by using forward, sideways, and turn components is illustrated in Figure 5.4.

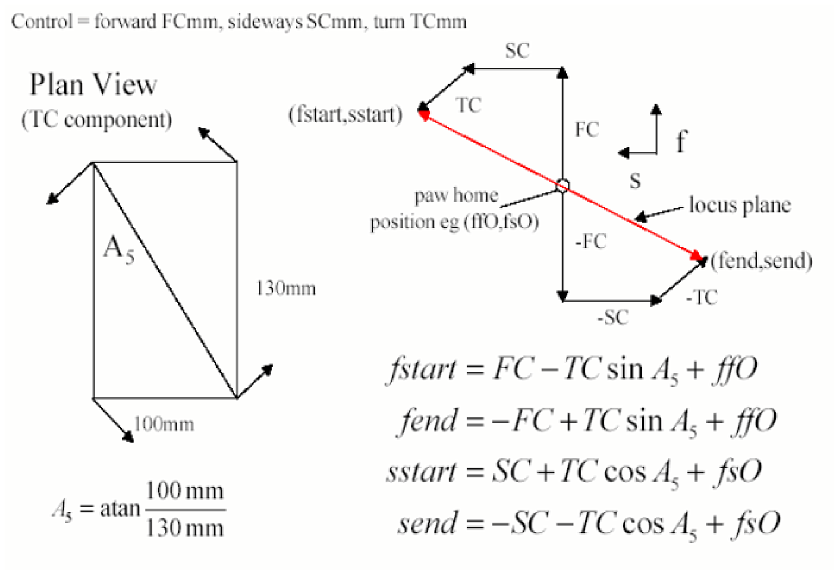

Control = forward FCmm, sideways SCmm, turn TCmm
Plan View
(TC component)
A_5
130mm
100mm
$A_5 = \text{atan}\frac{100\text{ mm}}{130\text{ mm}}$
SC
TC
(fstart,sstart)
FC
f
s
paw home position eg (ffO,fsO)
locus plane
-FC
(fend,send)
-TC
-SC
$fstart = FC - TC\sin A_5 + ffO$
$fend = -FC + TC\sin A_5 + ffO$
$sstart = SC + TC\cos A_5 + fsO$
$send = -SC - TC\cos A_5 + fsO$

Figure 5.4: Forward, sideways, and turn components are added vectorally to obtain a resulting vector; which indicates the direction and limit of the paw movement [25].

Trot gait, in which the diagonally opposed legs are synchronized, is used as the primary gait type. Omnidirectional motion is inherited from ParaWalk as it is. The only difference is the meaning of sideways and turn components. In ParaWalk, default sideways direction is leftwards, and default turn angle increases counterclockwise. In our approach, default sideways direction is rightwards, and default turn angle increases clockwise. Resulting motions for different combinations of walk components is shown in Figure 5.5.

5.3.1 Representing the Locus

According to the research done so far, rectangular, trapezoidal and half elliptic loci are not effective; in fact they have a hindering effect on robot's movement. Especially the movement of rear legs is the cause of this effect. While performing these kinds of movements the leg touches the ground in the same direction of the movement, which in turn decreases the robot's momentum at that time.

Proposed locus is in the shape of an ellipse cut from below in some proportion. This shape can be approximated by a hermite curve and it is illustrated in Figure 5.7.

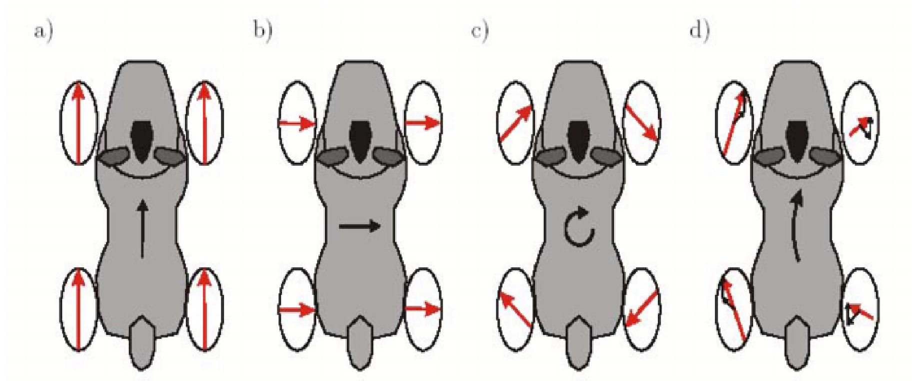


Figure 5.5: Resulting motions with different combinations of forward, sideways, and turnCW parameters: (a) anly forward, (b) only sideways, (c) only turnCW, (d) forward and turnCW together [24].

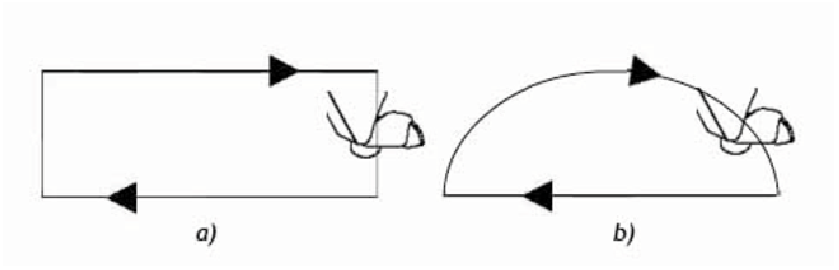


Figure 5.6: Movement of the paw on a (a) rectangular locus and a (b) half elliptic locus.

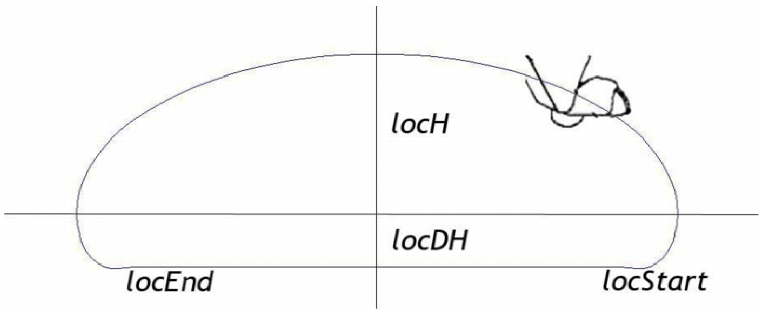


Figure 5.7: Proposed locus in the shape of a hermite curve.

With the introduction of elliptic locus, this effect is avoided since the leg touches the ground after moving in the reverse direction of the movement for a short period of time. This movement type guarantees that the moment of the robot is not hindered but increased. Also, elliptic locus makes the movement of the leg smoother.

5.4 Object-oriented Design

Locomotion module is designed by using an object-oriented approach. First of all, robot is thought as a single object composed of many other objects. Specifically, an AIBO robot physically consists of four legs, a head, and a tail, each of which carries different number of joints. Each Leg has three Joints, which are the rotator, the abductor, and the knee joints. The Head has three Joints, which are pan, tilt, and roll joints. Finally, the Tail has two Joints, which are pan and tilt joints. All these objects are defined as a separate class. The classes used for the locomotion module is shown in Figure 5.8.

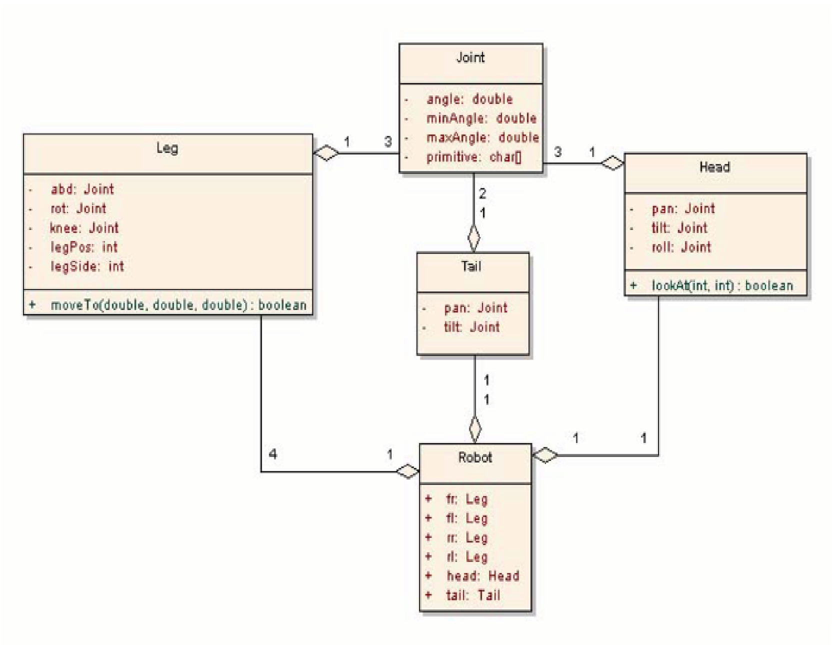


Figure 5.8: Class diagram showing the relations between classes used in the locomotion module.

Leg class has a method named *moveTo* for calculating the required joint angles to be able to reach a specific point in a 3-dimensional space. It performs the aforementioned inverse kinematics calculations and determines the knee, abductor, and rotator angles of the leg, respectively.

Besides these robot related classes, there are two very important classes. One is MotionManager class, which is responsible for the coordination of all movements, and the other is GA class, which is responsible for generating an initial population according to the sample string provided, and then performing the main GA operations (reproduction, crossover, mutation) on each population in order to generate parameter lists to be used during experiment processes.

5.5 Parameter Optimization

There are 11 parameters used by the new walking engine. These parameters can be categorized as step duration related, locus related, and initial paw locations related parameters.

1. Step duration related

   pStep: Number of steps needed to complete one full step (i.e. the paw comes back to its initial position).

2. Locus related

   fLocH: Height radius of the ellipse to be used as the locus for front legs.

   fLocDH: Perpendicular distance of the center of the ellipse of the front locus from the initial paw location.

   bLocH: Height radius of the ellipse to be used as the locus for rear legs.

   bLocDH: Perpendicular distance of the center of the ellipse of the rear locus from the initial paw location.

3. Initial paw locations related

   hF: Height of the chest of the robot from ground.

   hB: Height of the back of the robot from ground.

   fs0: Sideway distance of the paws of the front legs from shoulder.

   ff0: Forward distance of the paws of the front legs from shoulder.

   bs0: Sideway distance of the paws of the rear legs from shoulder.

   bf0: Forward distance of the paws of the rear legs from shoulder.

GA is used for optimization of these parameters. Initially, a set of hand-tuned parameters are given to the GA engine, and it produces some predefined number of chromosomes by distorting the parameter values on the sample string to obtain a generation. Structure of our parameter set and a sample string are shown in Figure 5.10 and Figure 5.11, respectively.

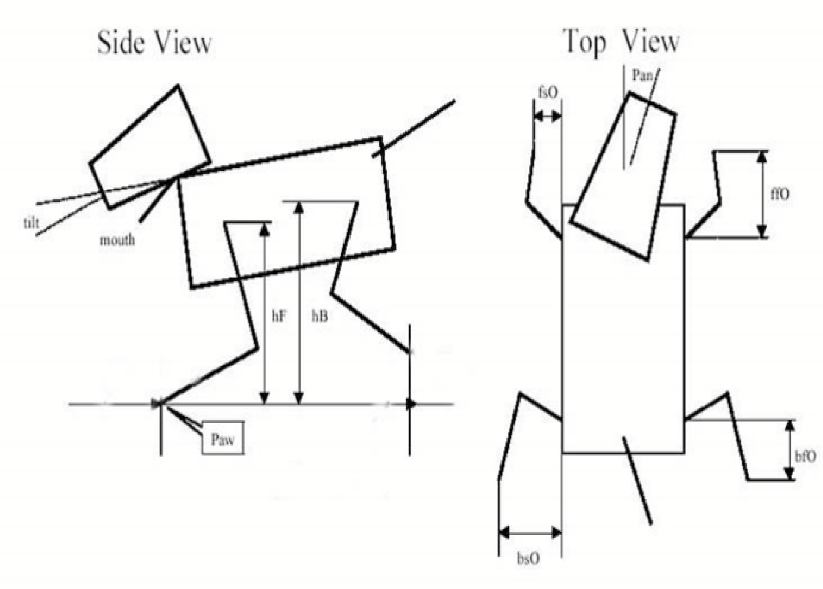


Figure 5.9: Parameters related to initial paw locations [24].

pStep	fLocH	fLocDH	bLocH	bLocDH	hF	hB	fs0	ff0	bs0	bf0

Figure 5.10: String representation of walking parameters.

80	20	10	25	10	-55	-104	6	52	12	16

Figure 5.11: A sample string.

Once a generation is constructed, the process begins. GA engine starts with the first chromosome and replaces the current parameter values on the robot side with these new values taken from this chromosome and the robot starts walking. After the robot completes 8 steps, the fitness of this parameter set (chromosome) is calculated according to our fitness function. Since the main objective of this process is to obtain the optimal parameter set that provides the faster walking without any diversion, our fitness function is constructed in such a way that it would promote moving straight forward, and punish either rotational or sideways diversion. Our fitness function is

$$fitness = 0.9 * fwd - 0.4 * sdwDiv - 0.4 * rotDiv \tag{5.6}$$

where *fwd* is total forward distance reached, *sdwDiv* is sideways diversion, and *rotDiv* is rotational diversion. When all the chromosomes are tried and associated with a fitness value, the reproduction process begins. During reproduction process, a random number between zero and total fitness values of all chromosomes is generated and a chromosome is selected by using the well-known roulette wheel technique. After that second chromosome is selected similarly. Then these chromosomes exchanges some patterns according to a crossover probability rate and generates two new chromosome to be copied into the new generation. Also, during crossover process, some parameter values can be changed according to a mutation probability rate.

However, there are some differences between our approach and simple GA according to the techniques used for performing crossover and mutation operations. In our approach, there are more than one crossover point determined by the range of parameter category; that is locus related parameters are exchanged in their own range, and initial paw locations related parameters are exchanged in their own range. These clusters are shown in Figure 5.12. Before performing crossover operation, one of the three clusters is selected, and then the crossover operation is performed within this cluster as shown in Figure 5.13. In mutation operation, the amount of distortion is determined according to the fitness value of this chromosome. That is, if the fitness values is small, the amount of distortion is greater, and if the fitness value is high, which means that this parameter set is good enough, then the amount of distortion is smaller.

Finally, Table 5.1 shows the performances of different walking engines, and improvement provided by our proposed engine.

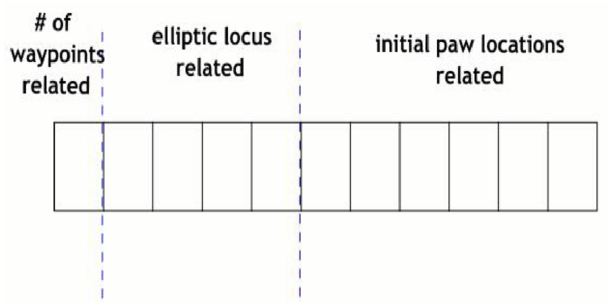


Figure 5.12: A sample chromosome seperated into three parts. These parts are related to number of waypoints on the locus, shape of the front and back loci, and initial paw locations, respectively.

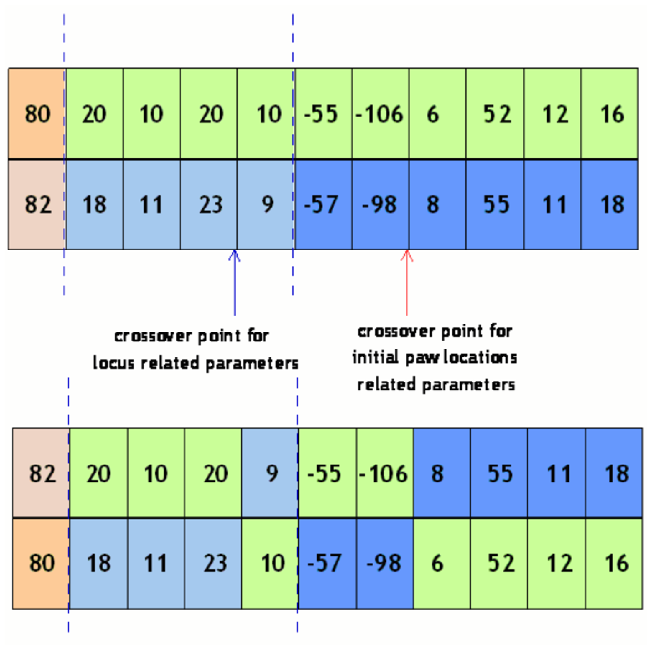


Figure 5.13: Technique used for the crossover operation in our approach.

Table 5.1: Walking engine vs. Performance.

Walking Engine	**Performance**
Default Sony-type	$\approx$ 4 cm/sec
ParaWalk	$\approx$ 21 cm/sec
Our engine (hand crafted)	$\approx$ 25 cm/sec
Our engine (optimized)	$\approx$ 27 cm/sec

Chapter 6

Results

6.1 Games

This year, Cerberus was the only team competing with ERS-210 robots with 200MHz CPU speed. The rest of the participants were using ERS-7s which have faster servos, more memory and a faster CPU. During the preparation to the competition, we have assumed that even though we have one of the fastest locomotion engine in ERS-210s, the speed of our robots will be much slower than the ERS-7s. Therefore, in order to save time, we have decided to use a non-grabing behavior which is based on an "approach the ball and kick it towards an appropriate direction" approach. In fact, that was a mistake, because most of the teams had a walking speed nearly equal to ours and we have suffered so much from spending time to align to the ball for kicking. This was the major reason for losing soccer games.

6.2 Technical Challenges

Contrary to the games, we have shown a good performance in the challenges and won the technical challenges! Even though we could not show a good performance in the soccer games, our modules were so powerful when considered individually and they brought the first place to us.

6.2.1 Open Challenge

In the open challenge, we presented a pool game in which the robot tried to score by hitting a ball with another ball. Unfortunately, we had a "Demo Syndrome" and our robot failed to kick the ball during the demo. This resulted in quite low points from the audience.

6.2.2 Localization Challenge

In the localization challenge, our robot could not hit the points but the adaptation speed of the robot to the unknown environment was too fast so our traversal time of five points was very short. As a result, we received a high point from that challenge.

6.2.3 Variable Lighting Challenge

The variable lighting challenge was the one in which we thanked so much to our really great vision module. It's quite hard to believe but we did not prepare especially for this challenge. During the competition, other teams were taking sample pictures by darkening the spots when there aren't games in the fields and working so hard on preparing their robots for the challenge. We did not take any pictures especially for this challenge, we also did not train a special color table for this purpose. We just used the very same color table which we used in the games. When the challenge started, everyone was surprised because all the other teams were assuming that the changes in the lighting will occur by darkening the environment. But the TC had decided not to remove the regular spots (because removing the spots might affect the neighbor field which was the one that the third place and the final games would be played on), but to introduce extra spots emitting yellow-white light.

Our robot was among a few robots that could manage to score. Even in the parts that the extra spots were flashing and the ball was right in the middle of the region that the light of extra spots fell on, it did not lose the ball even for a short duration of time and scored a good goal. That goal brought us the trophy.

The winning prize was an ERS-7 and it was so meaningful to win an ERS-7 with a 4 years old ERS-210.

Bibliography

[1] YILDIZ, O. T, L. Akarun and H. L. Akın, "Fast nearest neighbour testing algorithm for small feature sizes", *Electronics Letters*,Vol 40, No 3, pp. 171-172, February 2004.

[2] Schioler, H. and Hartmann, U., "Mapping Neural Network Derived from the Parzen Window Estimator", *Neural Networks*, 5, pp. 903-909, 1992.

[3] Kose, H and H. L. Akın, "Experimental Analysis And Comparison Of Reverse-Monte Carlo Self-Localization Method", CLAWAR/EURON Workshop on Robots in Entertainment, Leisure and Hobby, December 2 – 4, 2004, Vienna, Austria.

[4] Kose, H and H. L. Akın, "Robots From Nowhere," *RoboCup 2004: Robot Soccer World Cup VIII,* LNCS 3276, pp.594-601, 2005.

[5] Kose, H and H. L. Akın,"A fuzzy touch to R-MCL localization algorithm", Robocup 2005 Symposium. (accepted)

[6] Kaplan, K. B. Çelik, T. Meriçli, Ç. Mericli and H. L. Akın, "Practical Extensions to Vision-Based Monte Carlo Localization Methods for Robot Soccer Domain" Robocup 2005 Symposium. (accepted)

[7] Hightower, J., and G. Borriello, "A Survey and Taxonomy of Location Systems for Ubiquitous Computing", Technical Report UW-CSE Tech Report No:01-08-03, 2001.

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

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

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

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

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

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

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

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

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

[17] Gutmann, J.S., and D. Fox, "An Experimental Comparison of Localization Methods Continued", In Proc. of the 2002 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS02), Lausanne, Switzerland, pp 454-459, 2002.

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

[19] KAPLAN, K. and H. L. Akın, "A Controller Design for Soccer Robot Teams", *IJCI Proceedings of International XII Turkish Symposium on Artificial Intelligence and Neural Networks TAINN 2003*, 1, 1, July 2003.

[20] KOSE, H., Ç. Meriçli, K. Kaplan and H. L. Akın, "All Bids for One and One Does for All: Market-Driven Multi-Agent Collaboration in Robot Soccer Domain", *Computer and Information Sciences-ISCIS*

2003, 18th International Symposium Proceedings, LNCS 2869, pp. 529-536, 2003.

[21] KOSE, H., K. Kaplan, C. Mericli and H. L. Akın, “Genetic Algorithms Based Market-Driven Multi-Agent Collaboration in the Robot-Soccer Domain”, FIRA Robot World Congress 2003, October 1 - 3, 2003, Vienna, Austria.

[22] KOSE, H, U. Tatlidede, C. Mericli, K. Kaplan and H. L. Akın, “Q-Learning based Market-Driven Multi-Agent Collaboration in Robot Soccer”, Proceedings, TAINN 2004, Turkish Symposium On Artificial Intelligence and Neural Networks, June 10-11, 2004, Izmir, Turkey, pp.219-228.

[23] Bernhard Hengst, Darren Ibbotson, Son Bao Pham, Claude Sammut
Omnidirectional Locomotion for Quadruped Robots
School of Computer Science and Engineering, University of New South Wales

[24] UNSW 2003 team report
“http://www.cse.unsw.edu.au/∼robocup/report2003.pdf”

[25] UNSW 2000 team report
“http://www.cse.unsw.edu.au/∼robocup/2002site/2000PDF.zip”