UPPSALA
UNIVERSITET

Gifr

Main Report

An undergraduate project in the course Project DV
Department of Information Technology
Uppsala University, fall 2003

Jakob Ahlén	jaah7883@student.uu.se
Jonas Eriksson	joer5789@student.uu.se
Thierry Galle	thga3581@student.uu.se
Joakim C. Kekonius	joca2513@student.uu.se
Johannes Kählare	joka3199@student.uu.se
Daniel Lanner	dala8399@student.uu.se
Erik Magnusson	erma1771@student.uu.se
Nonis Malamas	noma5953@student.uu.se
Thomas Nyström	thny7987@student.uu.se
Alexander Pettersson	alpe6413@student.uu.se
Magnus Pålsson	mapa5173@student.uu.se
Peter Sandström	pesa8079@student.uu.se
Teppo Siirilä	tesi4151@student.uu.se
Mattias Sterner	mast7883@student.uu.se

http://www.robocup.it.uu.se/gifr

Uppsala University
Department of Information Technology

Abstract

This paper presents our approach to the Sony Four Legged Robot League of RoboCup. Our system consists of motion, vision, localisation, calibration, communication, distribution, and strategy modules. We use several techniques for these modules: localisation using a Monte Carlo algorithm, a world view in the form of a fully replicated distributed database, working inter-robot communication, and a powerful color calibration tool using color maps with convex hull generation through a Graham Scan algorithm that also works as a testing and debugging tool.

Table of Contents

Formulas and figures

1 Introduction

There are a number of international research initiatives in the field of robotics. One of these is the RoboCup competition, which combines exchange of ideas and solutions with a soccer championship. In RoboCup, there are several different leagues, in which different categories of robots compete. The common goal of all leagues is to promote and stimulate development and research in robotics. Most of the participants in RoboCup are universities from around the world. Since 2000 Uppsala University has participated in several different leagues.

In RoboCup 2003, which was held in Italy, a student project group from the Computer Science Program at Uppsala University for the first time competed in the Sony 4-legged league, which uses Sony's AIBO robot platform. This project was called Dynamo Pavlov and the team also participated in the German Open local RoboCup competition. Dynamo Pavlov was a result of the undergraduate Computer Science project course running during the fall semester of 2002.

It was decided that the project course during the fall of 2003 should continue developing software for the Sony 4-legged RoboCup league. The focus was now on distribution and communication and implementing the soccer software as a complex distributed system. This year's project is named Gifr[1], ran over the entire semester, and corresponds to 15 weeks of full time studies.

In order to develop a successful distributed soccer system a number of issues have to be addressed, that pertain to several different subject areas. The system requires a functioning motion system, which deals with mechanical robotics, as well as functioning vision system, which deals with digital image analysis and color calibration. Furthermore, the system deals with artificial intelligence, distributed systems, real time systems, and communication.

The focus has been on the distribution issue, which is why the main goal of the project has been to implement a distributed system that enables a team of Sony AIBO robots to play soccer.

[1] Gifr: From Norse mythology, the name of one of the dogs belonging to the great god Odin.

1.1 System requirements

The system requirements of the Gifr system were agreed upon during the first few weeks of the project and are specified in detail in Appendix A. In order to achieve a functioning system, these fundamental requirements must be fulfilled:

- The rules of the RoboCup soccer competition must be implemented in the system.
- There must be a function motion system which allows for movement in all directions and special features such as kicks and saves.
- There should be a working localisation and positioning system.
- Different game strategies need to be developed so that different situations in the game can be handled.
- There needs to be a functioning vision system which provides enough visual data for it to be useful in the image recognition system.
- Communication between the robots must function so that the distribution of relevant data between them is possible.
- An efficient color calibration tool is necessary.
- The system should synchronize its world state based on a distributed view, i.e. it should update the world state to all robots connected to the system.

1.2 Methodology

The first few weeks of the project were devoted to writing a project plan and agreeing on the requirements specification. During that period information was gathered on existing tools and frameworks for the Sony AIBO platform at the same time as a PC lab with Linux and all software development tools necessary for the platform was set up. Some time was also spent getting familiar with the robots and their environment as well as the previous year's Dynamo Pavlov project.

The analysis resulted in a number of conclusions. The most significant of these was that the system was to be developed based on the Tekkotsu[2] framework rather than on the Dynamo Pavlov project code. Since the Tekkotsu framework has a very well-documented and clear

[2] Tekkotsu: "Iron bones" in Japanese, An application framework for the Sony AIBO platform developed by Carnegie Mellon University

system design we also came to the conclusion that our own system could easily adopt the fundamental characteristics of Tekkotsu's design.

The next step in the process was to get familiar with Tekkotsu and decide which modules the Gifr system needed to implement. In the Gifr project a combination of the Rational Unified Process (RUP) and the Extreme Programming (XP) methodology was used for the development process. This meant that a lot of freedom was given to the development groups while, at the same time, quite a lot of emphasis was put on documentation. A number of important milestones were scheduled, where testing and integration of the modules was to take place.

2 System overview

2.1 Hardware

The hardware platform for the Gifr project is the Sony AIBO ERS-210A robotic dog. The AIBO has 20 degrees of freedom with a continuous range of motion and a variety of other output mechanisms including sound and LEDs. In terms of input it has a variety of sensors, the most important of which are a color CMOS camera, an accelerometer, a infrared proximity sensor, and joint position & load sensing.

The AIBO is equipped with a 11Mbit/s wireless network interface, a memory card slot which accepts memory cards with a capacity up to 16 MB, and 32 MB of RAM. The CPU is a 64-bit RISC processer with a clock frequency of 384 MHz.

2.2 Software

The operating system running on the Sony AIBO is called Aperios. Aperios is a multi-threaded real-time OS developed by Sony. The OS kernel source is not available for developers outside Sony, which is a drawback in terms of control over processes running on the AIBO.

The software development kit provided by Sony for the AIBO is called OPEN-R. OPEN-R consists of a set of routines and data structures used for controlling the robot and provides the basic interface to the hardware. The OPEN-R SDK can be used to create Aperios objects, that are processes that perform application specific functions. Each Aperios object communicates with other objects by sending messages containing data, via shared memory regions. Message passing uses a hand-shake protocol with senders and observers.

The OPEN-R SDK is sufficient for developing a software system for the AIBO. The drawback to using OPEN-R is that the programming structure required is relatively complicated and the resulting applications are often inefficient, unless extra effort is put into minimizing the number of Aperios objects (in which case the intuitive understanding of the source code can be significantly hampered). Therefore we chose to use the Tekkotsu framework, which both provides basic functionality such as low level vision and motion, as well as an extra abstraction layer which among other things handles Aperios inter-object communication in an efficient manner.

The programming language used in the Gifr project is C/C++, except for the Calibration tool which is written in Java. The development environment has consisted of a PC lab with GNU Linux and other development tools such as CVS, Bugzilla, and Doxygen.

3 System design

The Gifr system is designed based on both the Tekkotsu system design and the structure of the Dynamo Pavlov project code.

3.1 The Tekkotsu framework

The Tekkotsu framework is a free, open-source development framework released under the GPL. The goal of Tekkotsu is to provide a basis for rapid development of new AIBO applications.

Architecturally, Tekkotsu consists of only three Aperios objects (two if sound is disabled), which enables efficient run-time performance since inter-object communication, which otherwise can be a lag factor, is minimized. The three objects (or processes) are:

- MainObj - Most of the bulk processing, such as vision, decision making, tracking state of the world.
- MotoObj - Updating positions of joints and values of LEDs.
- SoundPlay - Mixes and sends sound data to the system to ensure playback doesn't stutter.

Tekkotsu builds on the basic functionality provided by the OPEN-R operating system. It is written in C++, (like the underlying system APIs) and makes full use of inheritance and templates. The main functionality of Tekkotsu is provided through events and behaviours. In essence, behaviours are functions that send commands to the robot. Behaviours can listen for certain events to occur and be triggered by them, as well as post new events that will trigger other behaviours. Timers are also handled as events.

Tekkotsu provides certain services that enable basic functionality on the AIBO. Gifr makes use of some of these services and modifies others. Some of the functionality that Gifr makes full use of are:

- Low-level vision processing – uses the CMVision module developed by Carnegie Mellon University
- Motion primitives – walking style from CMU as well as other basic motion functions
- Utility functions – timers, mutex, self-righting after falls, etc.

The main bulk of the Gifr project, however, either consists of modifications or extensions of Tekkotsu modules or completely new modules that are integrated with the underlying Tekkotsu system.

3.2 Overview

The Gifr system has been divided into a number of modules according to functionality. Some of the modules are quite large, and consist of both extensions and modifications of underlying Tekkotsu modules as well as new Gifr-specific code. The modules that make up the Gifr system are:

- Motion – extensions of the Tekkotsu motions primitives and custom motion sequences for kicks, saves, etc.
- Vision – extensions of the Tekkotsu low-level vision system as well as a high-level system which interacts with the Localisation and Strategy modules.
- Localisation – a system for the AIBO to localize itself on the field.
- Calibration – a stand-alone calibration and configuration tool for the Gifr system.
- Communication – low-level communication routines
- Distribution – a system for managing information that needs to be distributed to the whole team.
- Strategy – the main soccer-playing AI-routines.
- Goalkeeper – specialised AI for the goalkeeper.
- Penalty – specialised AI for penalty shot situations.

3.3 Motion

The motion module uses the walking style from the Tekkotsu framework, which was developed by the CMU RoboCup soccer team. The walking engine uses *hermite splines*, a mathematical function that interpolates between given control points. By using this method to calculate the joint positions, smooth changes are accomplished between the different joint positions. The motion sequences which are used for shots and saves do not use splines, instead they just set the joints to the given positions that predefined in a file.

All motion is executed by the Motion Manager in Tekkotsu, which provides mutual exclusion and the possibility to set priority on the tasks sent to it. There can only be one motion execution at a time, so it is not possible to run a motion sequence at the same time as walking

forward. But it is possible to execute different motions on different parts of the AIBO at the same time, e.g. walking forward while sweeping with the head.

The walking engine and head motions are controlled by timers and are executed *n* times per second. These parts do not use the Tekkotsu event system because it tended to overflow the event queue. The things that are controlled by events in the motion module are the motion sequences and tail motions, because these parts do not need to run very often. Therefore it would be waste of processing power to check this on regular time interval.

3.3.1 Walking

The AIBO can walk in all directions; forward, backward, sideways to left and right, rotate left and right as well as combinations of these at the same time. The walk engine keeps track of the AIBOs movements on the field with pretty good accuracy, unless we collide with some other AIBO or some obstacle. This information is used in the *Monte Carlo* algorithm in the localisation module.

Some different walking styles were tested, but the one that was chosen is the default walking style in Tekkotsu, since it is relatively fast and camera jitter due to the walking is acceptable. An interface is used to pass parameters about walking speed in different directions. Values that can be used for localisation are received from the walking engine.

3.3.2 Head motion

Head motion can be used for searching for the ball, beacons, and goals. Head motions that can be done are pan (look to both sides), tilt (look up and down) and roll (look cute). These can be combined together to create more complex motions. A simple motion is just to set a standing position, but executing example a kick requires several different joint updates to make a complete motion.

3.3.3 Motion sequences

With help of motion sequences it is possible to set postures for all joints on the AIBO. This can be one simple frame or a more complex series.

3.3.4 Kicks

The Gifr system has three different types of kicks. All kicks are based on that the ball is between the front legs of the AIBO. Kick number one shoots the ball straight forward, using both front legs. The second kicks make a sweeping movement with one of the front legs, left

or right depending on which direction the shot should be directed at. This kick shoots the ball 45 degrees to the left or right. The third "kick" uses the head to shoot the ball to the right or left. This is handy if the AIBO is stuck with the ball to one side.

3.3.5 Grab ball

This motion is used to grab the ball when it is between the AIBO's front legs. It is performed using front legs and the head. Grab the ball can be used before shooting or when the AIBO needs to rotate with the ball.

3.3.6 Goalkeeper save

This action is performed when the goalkeeper needs to do a save motion. The AIBO puts out its front leg to both sides. When doing this the AIBO takes up more space in the goal, which makes it harder for the opponent to score.

3.3.7 Stand up

Simply puts the AIBO in a standing position.

3.3.8 Auto get up

If the AIBO trips over and falls on the back or on the side, it has two different sequences depending on if it falls on the back or the side. It uses the accelerometers that are built in to the AIBO to detect that is has tripped over.

3.3.9 Conclusions

By using the Tekkotsu framework there was no need to do any work with inverse kinematics and develop a custom walking engine. This freed people to work on other parts of the project. The Tekkotsu walking engine works very well in most cases and provides the functionality needed since it was developed for RoboCup soccer. Some minor strange things happen when the AIBO is walking forward and starts to rotate – sometimes it throws out one front leg to the side. This makes the AIBO jerk a little, which could lead to unpredictable changes in the walking path that are not desirable.

3.4 Calibration

A graphical user interface for calibration and configuration of the system has been developed.

3.4.1 Color Calibration

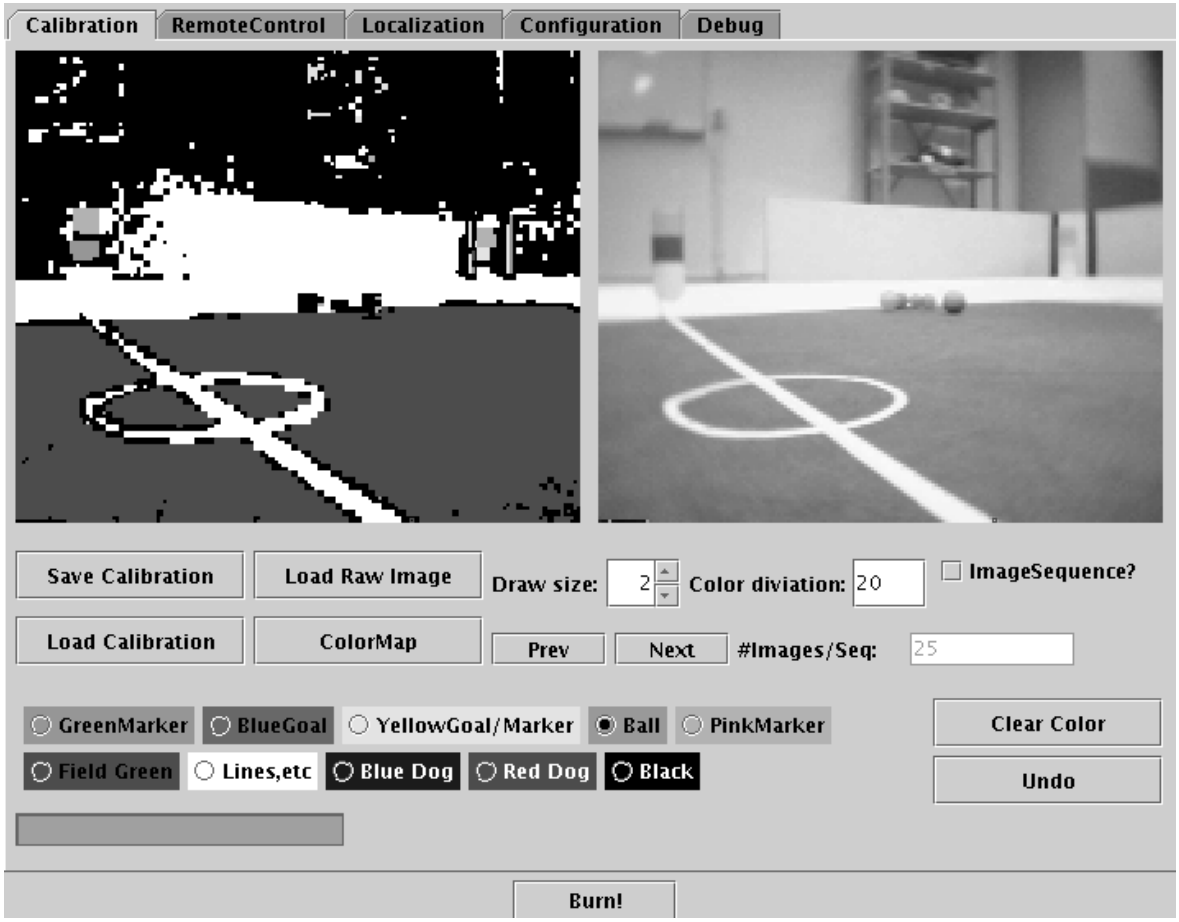

Fig. 1 Color Calibration GUI

The core of the color calibration tool is Tekkotsu's segmenting tool which we have modified to our needs. What is needed to calibrate in order for the AIBOs to be able to play are 6 different markers, the two goals and the ball. Other optional things that can be calibrated are the lines, field color and other AIBOs.

Images of markers, goal and ball are required and these images must be in YUV format, which is a color format where every pixel is constructed by three values; luminance (Y) and two color components. When loaded, these images are converted to RGB so that it is easier for humans to see what the images represent and how they should be calibrated, instead of looking at a picture with strange colors. After loading, a colormap is created where all the different colors have been converted into x coordinates and y coordinates that are plotted onto the colormap. Every RGB color is first converted into a HSB format, which is a color format where every pixel consists of three values; hue value, a saturation value and a brightness value. From there

the hue value is taken to be the x coordinate, and the saturation value to be the y coordinate. The colormap is used to show the areas that are created later in the calibration.

When we pick a color and start selecting which colors on the image are to be classified as the specific color, the following happens:

1. The algorithm searches through all the pixels in the image currently showing for similar colors to those colors inside the selected area in the image and taking into account the color deviation. E.g: Select a color with values r, g and b. Look for colors which lies in the range r ± color deviation, g ± color deviation, b ± color deviation.
2. Convert all the colors we found into two-dimensional points as described above and store the points along with the previous points if any.
3. An area is created from the points above with a *Graham Scan* algorithm to get a convex hull area. This because all the points generated in step 2 should be inside the area.
4. The Area is shown on the colormap. This area is altered as new points are found or generated.

The Graham Scan algorithm is a very fast (O(n log n)) way to calculate a convex hull from a set of points, which is the main reason why the algorithm is used. It works the following way:

1. Find the point with the lowest y coordinate.
2. Sort the remaining points in increasing order of the angle they make with the point in step 1 and the x-axis.
3. Go through the points and calculate whether or not moving from the last good point to the next point causes a ”left turn” or a ”right turn”. If a ”right turn” is found then the second-to last point is not in the convex hull and is taken away from future calculations. The process is continued as long as the last three points is a ”right turn”.

During the saving process 65,536 different YUV colors are generated and converted into points. These points are checked whether they lie inside any of the areas created in the above process. If the point lies inside the area it is stored away in a byte array. This generates a threshold file that is used on the AIBO to classify the colors into regions. A second file, color file, is created also specifying which areas should be presented as what colors. The threshold file contains the mapping from color space to an index value of a corresponding color in the color file. In other words, the threshold file defines the regions of color space which should be considered a color x, and then the xth line of the color file describes that color.

3.4.2 Image Sequences

When the image sequence button is pressed, the program assumes that there a specified number of images, which can be changed. It will then look through all the pixels in the rectangle in all the images. This because then we find the slight color changes that occur in different frames. With this method some of the color changes are found.

3.4.3 Remote Control

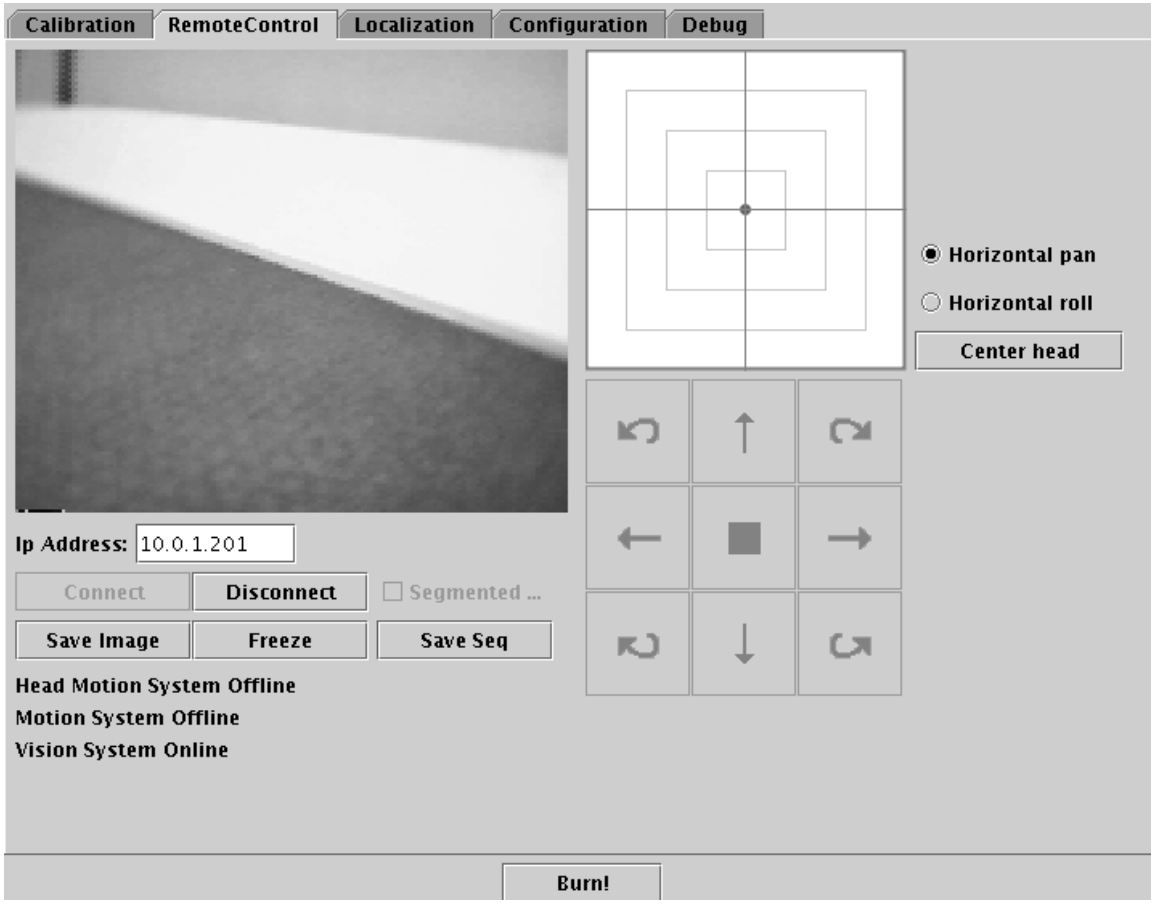

Fig. 2 Remote Control GUI

The purpose of this tool is to be able to remote control an AIBO and to take images of the different things needed to calibrate the colors. To be able to do that, one must first connect to the AIBO by inputting its IP number in the field.

If the correct behaviours are running on the AIBO then it is possible to steer the AIBO with the controls given. The head is controlled with the point and the body is controlled with the arrows under the point.

3.4.4 Localization helper

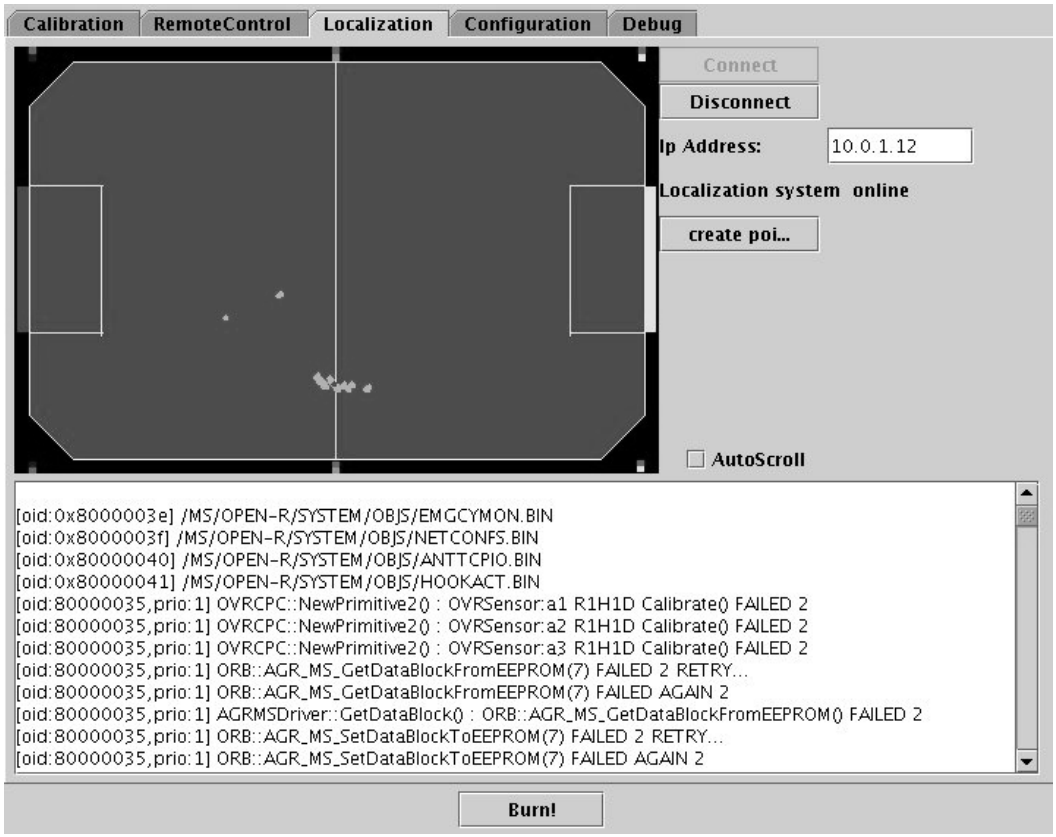

Fig. 3 Localisation helper GUI

The purpose of the localization panel is to see where the AIBO thinks it is on the field.

Using this is simple, simply input the IP adress of the AIBO and press connect. A telnet window is also available to see debug information.

3.4.5 Configuration

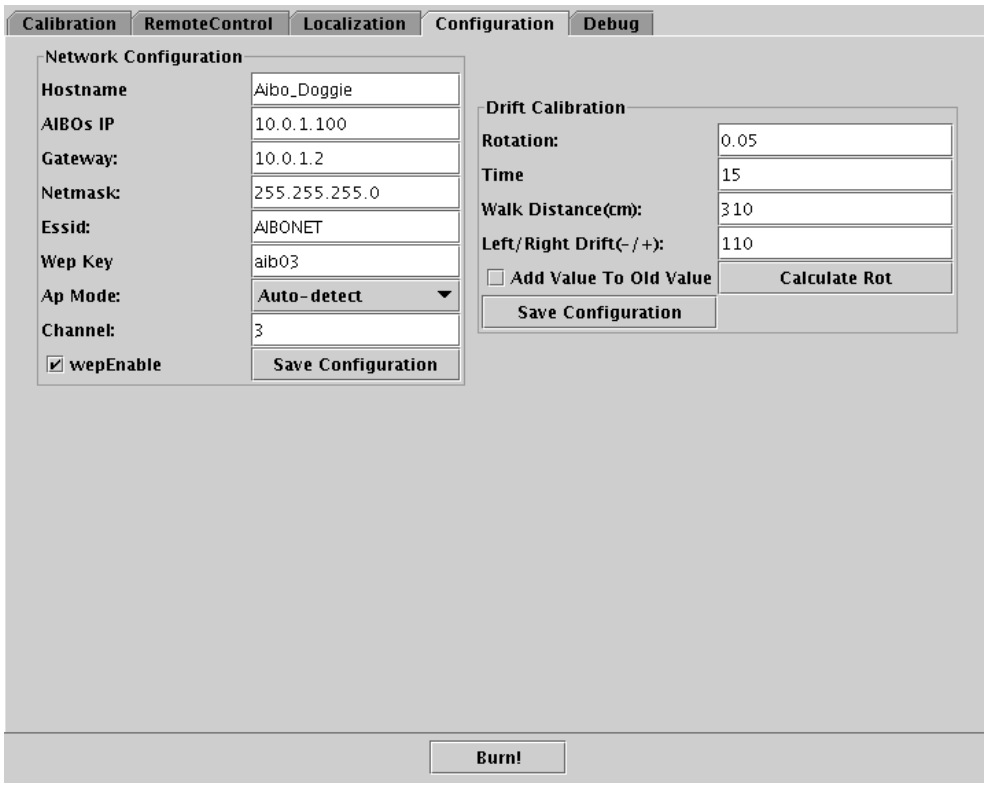

Fig. 4 Configuration GUI

The purpose of the configuration panel is to simply configure an AIBO's IP address and other things needed to be able to connect to an AIBO. Some drift compensation information can also be calculated here. To do that, let the AIBO walk straight for a specified time, measure the distance walked and how much left or right it drifted. Input those values in the different fields to calculate the compensation value. This value can be used to compensate for the drift.

3.4.6 Debug

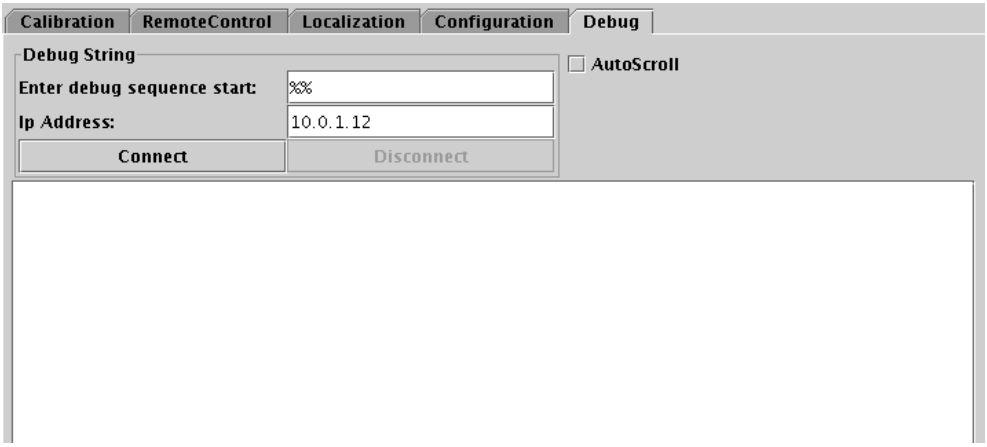

Fig. 5 Debug GUI

The purpose of the debug panel is to sort out the needed debug information from the telnet window. This is useful when lots of different debug messages are output to the telnet and you want to watch for a specific message type.

3.4.7 Conclusion

The main reason for using Java instead of C/C++ was that Tekkotsu used Java in their tools, which allowed us to use some of their code in our calibration tool instead of writing similar code from scratch. Due to this we could concentrate on the conceptual areas of the problem instead of the GUI aspects.

3.5 Vision

The Vision module is responsible for analysing images from the AIBO camera and producing information about objects that the robot currently sees. It has two major stages: the *low level vision system* performs operations on the image to find regions of the same color within the image, and the *high level system* then uses these classified regions to determine the position of objects relative to the robot.

3.5.1 Low Level Vision

The lower level of the vision system has not been modified very much by the Gifr project, since everything was already implemented in Tekkotsu. Tekkotsu uses a library called

CMVision[3] developed by James Bruce, then student at Carnegie Mellon University, as his undergraduate thesis. The system looks at images in YUV format, from the AIBO's camera, and produces regions of pixels in the image that are 4-connected (explained below). These regions are what is mainly used by the high level system to locate objects in the image.

The regions are found by running through several different steps, the first of which is *segmenting* the image according to a lookup table for all different color values. So every pixel is given a symbolic color, instead of three values for YUV. For example the symbolic color Orange is classified as three ranges of values, one in Y-space, one in U-space and one in V-space. If the YUV-values for a pixel in the image are all within the three ranges, the pixel is classified as Orange. This produces a so called color map for later use.

The second step is to find the *connected components* in the image. As mentioned above, all pixels in a region must be 4-connected, this means only pixels that are adjacent horizontally or vertically are connected, not diagonally adjacent pixels. To speed up the process of finding connected components the color map you *run-length encode* the color map before starting. This step produces a set of linked lists (one for each symbolic color) where every element is a region and the lists are sorted with the largest regions first.

The final step is *region merging*, which is done mainly to remove noise from the image. Here you consider merging several regions of the same color into one region, even though they are not connected. For regions to be merged together the area of the resulting region divided by the area of its bounding box must be greater than some threshold value, which can be set differently for every color.

3.5.2 High Level Vision

The task of the high level vision system is to determine the position of objects on the field relative to the AIBO. To accomplish this we use the color regions found by the low level vision system. We search for each object among the largest regions of the appropriate color. Each object can only be found once in each image (e.g. there can never be two blue goals in one image). The objects that our system can recognize at this time are:

- The orange game ball

[3] J. Bruce, T. Balsh, M. Veloso, (http://www-2.cs.cmu.edu/~jbruce/cmvision/)

- The blue and yellow goals
- The six different beacons on the field

Future improvements might include functionality for recognition of *other robots*, both of own and opposing team. For each object found in the image the system computes the *distance* to the object (in mm.), *two angles* to the object (one to the right side and one to the left side of the object) and a *confidence* value, ranging from zero to one. The confidence represents how sure we are that the recognized region in the image is in fact the real object, a value of one meaning we are very sure.

To calculate the distance in 3D-space to objects we use something called the inverted pinhole camera model.

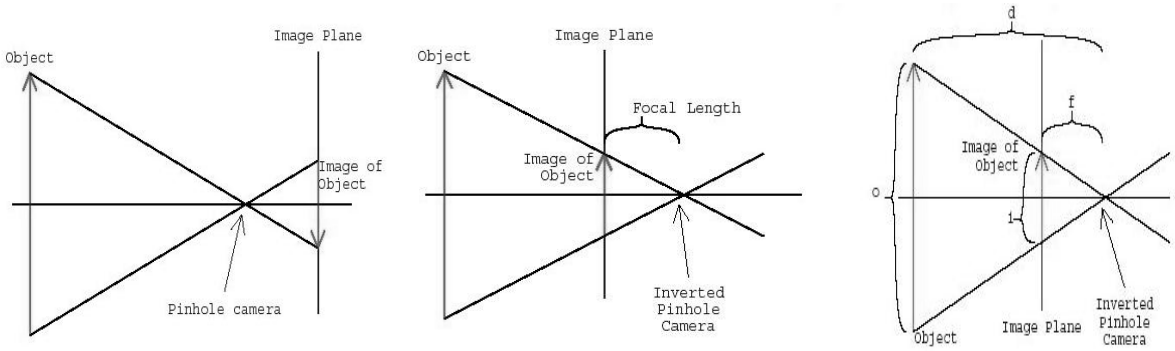

Fig. 6 The pinhole camera model, Carnegie Mellon University, 2003

We compute the distance (d) to the object using the relationship with the size of the object in the image (i), the size of the real object, and the focal distance of the camera (f):

$$d = \frac{f \cdot o}{i}$$

Formula 1 Distance to objects

3.5.3 The Ball

We search for the ball among the ten largest regions of orange color in each image. Among these ten regions we decide which one best fulfils the expectations of the system, and if this has a greater confidence than some threshold value, we save this region and the computed values (distance, angles, confidence) for later use by the strategy module.

The confidence value of the ball is computed by multiplying together some different factors, and adding a bonus if the region has a large number of orange pixels. The following properties are to be expected from a ball:

- The ball should be wide and tall enough (3 pixels) and have a large enough area (7 pixels)
- The ball should be located no more than 5 degrees above the robot's head
- The bounding box of the region should be square
- The area of the region compared to the area of the bounding box should match that expected from a circle.

3.5.4 The Goals

We search for the goals (yellow and blue) among the four largest regions of appropriate color in each image. Among these four regions we decide which one best fulfils the expectations of the system, and if this has a greater confidence than some threshold value, we save its values for later use by the Monte Carlo Localisation.

The confidence value of the goals is computed by multiplying together factors decided by these properties:

- The area of the goal should be big
- The ratio between the height and width of the region in the image should be close to 1.5 (the real goal has a width of 600mm and a height of 400mm)

3.5.5 The Beacons

We search for the six different beacons among pairs of regions with the right color. The ten largest pink regions are paired with the ten largest yellow, blue and green regions and each case is compared to the requirements for a beacon. For each beacon, one set of values can be saved for later use by the Monte Carlo Localisation.

The confidence value for each beacon is computed by multiplying together factors decided by these properties:

- The areas of the two colored regions should be equal
- The height of the regions should be equal to the distance between the centers of the regions

- The size of the two regions

3.6 Localisation

Localisation is the problem of deciding the robot's position on the field. To do this you need information about the position of objects on the field relative to the robot. The objects used for localisation are the six beacons and the two goals. If two or more objects are seen in one image from the camera you can decide the robots position at that moment using strictly geometrical calculations. However this only gives a position at that precise time, and it may not even be correct. Since we process about 25 images each second, we will know our position for about 1/25th of a second. But we wish to know our position accurately at all times. To achieve this we have implemented a method known as *Monte Carlo Localisation*.

3.6.1 Monte Carlo Localisation

This method uses a number of samples distributed on the playing field, where every sample represents a position of the robot on the field. Every sample has a position (x, y) on the field, an angle representing which way the robot is facing, and a quality representing how sure we are that this sample is correctly placed.

To compute the actual position of the robot, the field is divided into a number of sub-cubes and which sub-cube contains the most samples is determined. You then compute the mean of all samples in this sub-cube. The validity of the mean position (how sure we are that it is the correct position) is computed by comparing the qualities of all samples in the sub-cube with the qualities of all other samples. The more samples that are in the correct sub-cube, the greater the validity of the mean position.

Shown below are two examples of how a distribution of samples might look during a game.

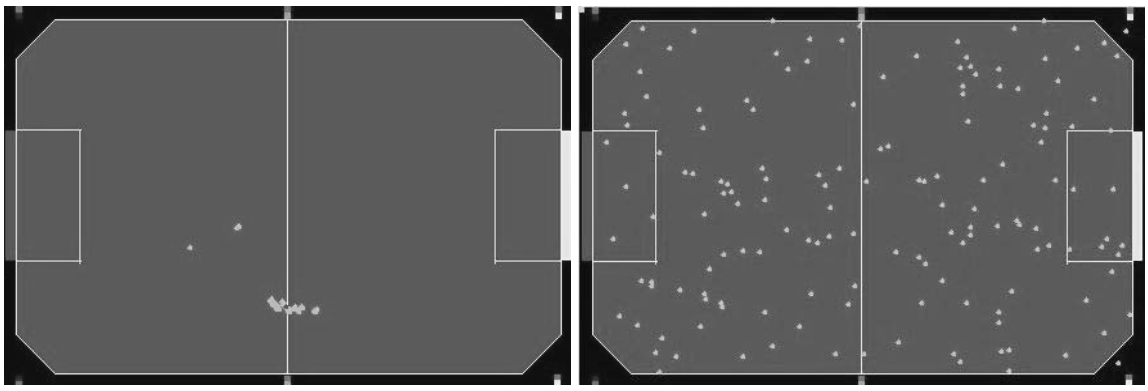

Fig. 7 Monte Carlo Localisation

The red dot is the calculated mean position, with the angle the dog is looking in drawn as a line. The first image shows the AIBO knowing its position very accurately, with all the samples located in the same region of the field. In the second image the AIBO has no idea where it is really located, as all the samples are scattered across the field. The mean position and angle are still printed, but it will now have a very low validity.

For each processed image every sample undergoes a number of different steps that are part of the Monte Carlo algorithm:

1. the **motion** update uses information from the robot's engines - telling us how much we have moved since the last image - to move each sample a little bit

2. the **vision** update compares information from the vision module, telling us the angle to a beacon or goal that was seen in the current image, and compares with a supposed angle from each sample to that same beacon/goal (from now on referred to as a *flag*). The supposed angle is calculated using the sample's angle on the field and position. If the two angles are not equal or close then the sample's quality should be decreased.

3. **resampling** removes improbable samples and replaces them with new more probable samples. For each sample a test is performed comparing its quality to a random value, and if the test fails, the sample is replaced by either a template (see below) or a random sample. Also, for each processed frame, there is a chance that improbable samples will be removed and replaced with copies of more probable samples.

Another important step is the calculation of **templates**. A template is a supposed position of the AIBO, calculated using angles and distances to several flags. Each time the vision-module sees a flag, the angle and position to it are saved in a buffer of flags. Each frame, a number of new templates are calculated using the information in this buffer. The templates can then be inserted into the sample set, replacing the improbable samples that were removed in the resampling step.

There are two different ways of computing new templates. The first uses the angles to three different flags, and the second uses the angles to two flags, and the distance to one of the flags. Strictly geometrical formulas are then applied to calculate the template.

The system has worked very well, and has allowed us to know our position on the field correctly at most times. We also do not need to worry about when an AIBO is 'teleported',

when penalized by the referee, because this works automatically with the Monte Carlo algorithm.

For the next RoboCup the two flags in the middle of the field will be removed, but this should not be a problem for our algorithm. A possible problem might lie in the fact that flags cannot be seen all across the field, but this is not a problem in the localisation. If the underlying functions in the vision system are improved a bit the performance of the localisation might be improved, but there is no real problem with the algorithm, just small bugs in the mass of underlying systems that add up to some small problems with the localisations.

3.7 Strategy

To have a well planned strategy was an important part of the Gifr project. Since last year's project had difficulties with the communication this is the major difference between the Gifr project and Dynamo Pavlov. It was desirable for the dogs to have specific roles, not to act only individually but as a team. This way it was possible to avoid "rugby-like" behaviour, where the dogs end up in a big mess that destroys every possibility to play nice soccer. The team play is supposed to be based upon the collective knowledge of all the dogs, which clearly is an advantage compared to solitary AIBOs.

3.7.1 Goalkeeper

The goalkeeper has his own specific role, for which certain conditions prevail. During the game it is not possible for any other AIBO to take over the role as goalkeeper dynamically. This means that the goal will be empty if the goalkeeper crashes or is removed as a consequence of penalty. These unique situations might be something the field players have to consider.

The actions of the goalkeeper are mainly based on the fact that he must not end up too far from the own goal. Therefore this is the first question that is asked in the decision tree. It is very important for the goalkeeper to know its own position with great accuracy. In our system the goalkeeper is supposed to stay close to the goal line, turned towards the ball to facilitate savings and other appropriate actions by getting confident and fresh information through the own camera. In some situations it is possible that the ball, even though it is close to the goal, is not visible to the own camera. The knowledge about the ball position is then based on the information from the other AIBOs, if it is available and trustworthy.

The behaviour of the goalkeeper is static in the sense that it acts the same way no matter how the field players are placed.

3.7.2 Field players

The field players differ from the goalkeeper by their dynamic behaviour and that they move over larger areas. Since we decided to use zone play, there are several tactical roles that an AIBO can have:

- Single defender
- Single attacker
- Left defender
- Right defender
- Left attacker
- Right attacker

Combinations of these roles depend on the number of active players and the chosen team positioning. Other roles were also considered. Two of those were supporting attacker and sweeper. The supporting attacker is supposed to be prepared to act as a second wave in the offence, the sweeper is supposed to defend the area closest to the own goal and just get the ball out of there. To have the ball, or to be the one that goes for the ball, can also be seen as a role but this role is not restricted to a specific zone. This is also the case for the supporting attacker, of course.

We have decided to use two basic variants of the placement of the field players. These may vary during game according to the number of players. How to place the field players depends on what tactics one wants to use. Some factors that decide the choice of tactics are:

- The way the opponent team plays, if it has been studied in advance (long-term factor)
- The desired result in the game (long-term factor)
- Penalties and crashes during game (short-term factor)

3.7.3 Zone play

The main reason for using zone play is to avoid that the AIBOs end up in a big clump. This was obviously the case in last year's project, mainly because of difficulties with the wireless communication. Another reason for zone play is the advantage of knowing that the dogs are

located in specific areas. If this is the case there is a greater possibility that a ball that is shot into the offensive half actually reaches one of the team mates. The distances that the AIBOs have to move should also be smaller than if zones are not used.

One thing to notice is that the zones overlap a bit. This to avoid inefficiency if the ball ends up just outside the AIBO's opinion of where the ball is. This can still occur, but we wanted to investigate whether or not the advantages of zone play outweighs these.

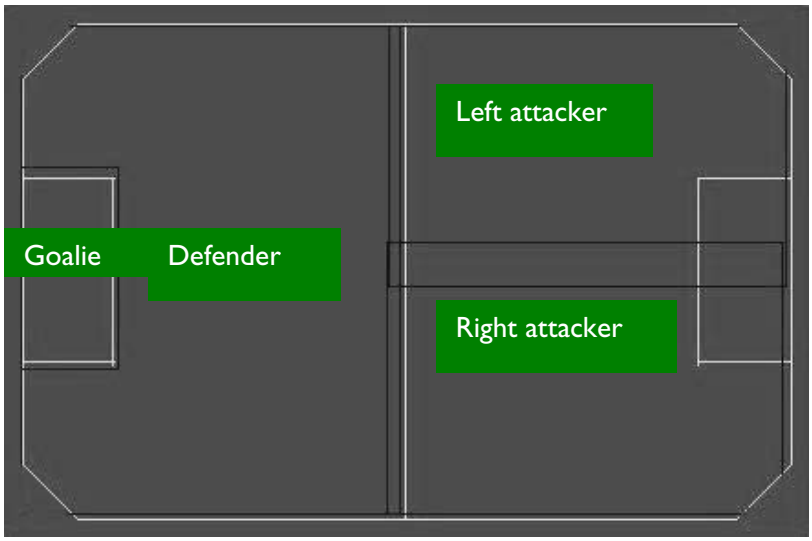

Fig. 8 Zone play with standard positioning

3.7.4 Standard positioning

The standard disposition of the field players that we have chosen is a pretty offensive variant with one defender, a left attacker and a right attacker, as can be seen above.

When a player is removed from the game because of a penalty the disposition is changed. The new disposition consists of one defender and one attacker. This new disposition can be reached in two ways;

- If an attacker is taken from the field, the remaining attacker gets the role *single attacker*.
- If the defender is taken from the field, the attacker closest to the own goal gets the role *single defender* and the other the role *single attacker*.

3.7.5 Defensive positioning

An alternative positioning can be used when the focus lies on a strong defence. This can be suitable when the team is playing a very good team, the team is winning or the current result is satisfying for some reason.

When a player is removed as a consequence of penalties, a change to an even stronger defence is made. This disposition, consisting of two defenders, can be reached in two ways;

- If the attacker is taken from the field, nothing needs to be done
- If any of the defenders are removed, the attacker is set to take this role

3.7.6 Special situations

It is possible that a situation occurs when only **one** field player remains as a result of penalties and crashes. A natural action would be to give this AIBO the role single defender. We have discussed the need for a specialized role, the sweeper for these situations. But when considering the change of roles one has to consider the time it takes for the role changes to be completed.

If the goalkeeper is not in place in the goal, the team is obviously much more vulnerable against attacks. A simple measure to take is to set the team to defensive positions. Another measure would be to use the specialized role mentioned above, the sweeper, to protect the area in front of the own goal. Testing during game play is the only way to know whether it is reasonable to make these changes.

3.7.7 Decision trees

The actions of the AIBOs are decided by the decision trees that have evolved during the development. First of all there are separate decision trees for the goalkeeper and the field players. In the diagrams below the ovals represent questions that are to be answered by yes or no, depending upon the knowledge of the AIBO. This knowledge is based both on observations and calculations made by the AIBO in question and knowledge from other AIBOs.

The answer to the question leads further down the decision tree to another question, represented by another oval, or to a certain action to be performed. Actions are represented by rectangles and are the leaves in the decision tree.

Below are diagrams of the two decision trees where strict zone play, with a certain overlap, is used.

Goalkeeper

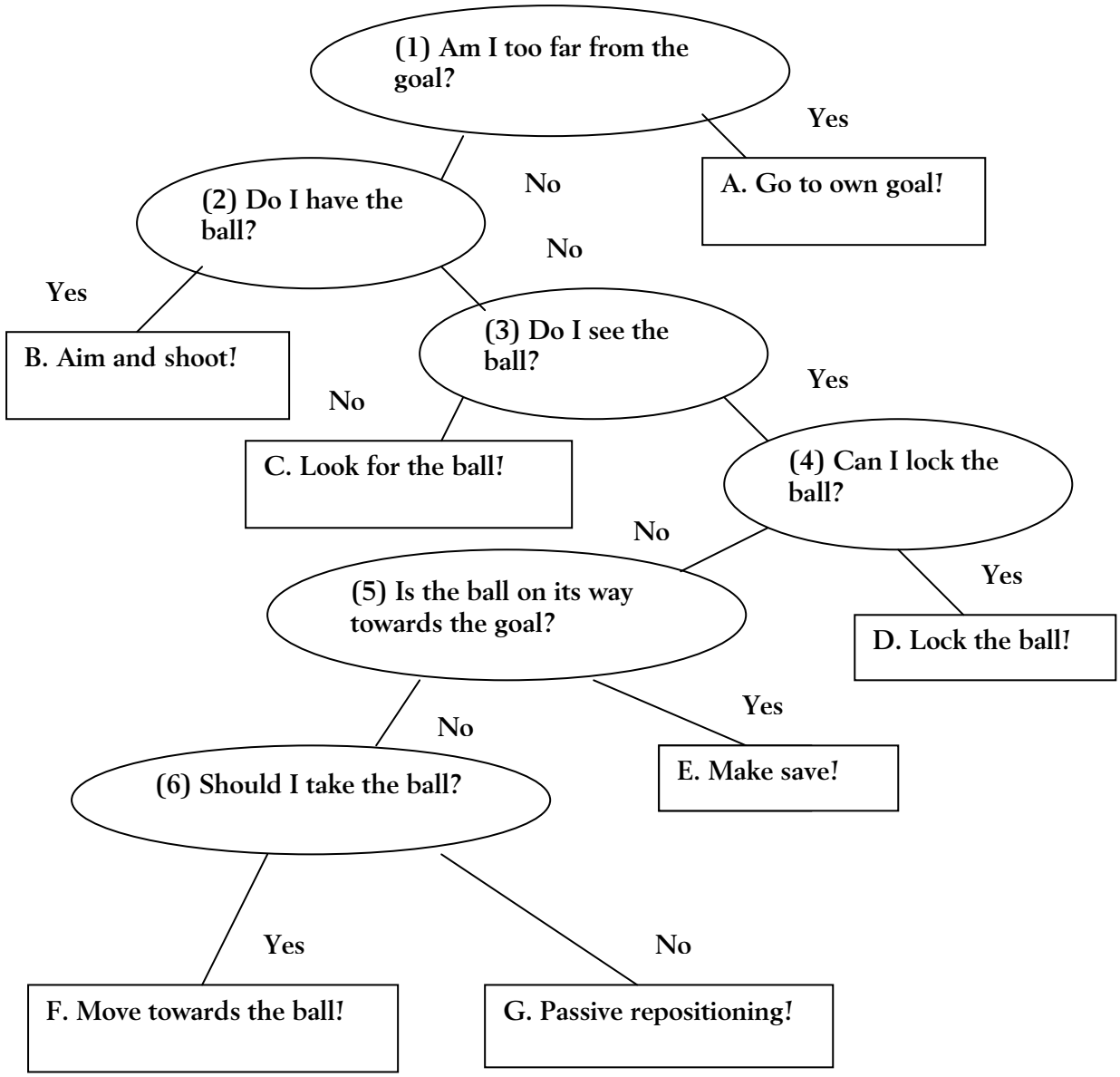

Fig. 9 Goalkeeper decision tree

The goalkeeper's decision tree is independent of the current strategy of the field players, but there are some things to consider. These are the definition of the goalkeeper's zone as well as when the goalkeeper should take the ball.

Field players

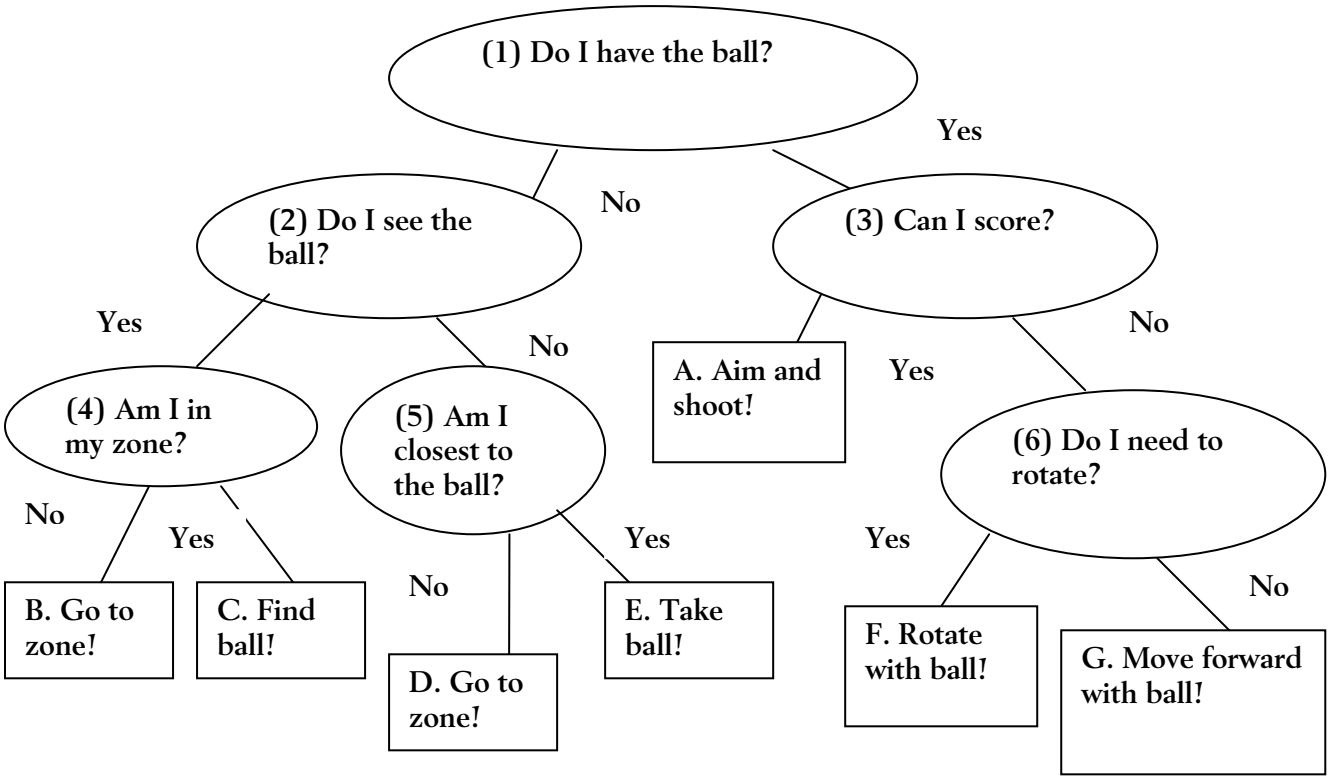

Fig. 10 Field player decision tree

3.7.8 Penalty behaviour

The penalty behaviour is an independent behaviour that will occur if a soccer game is tied.

The basic idea of the behaviour is:

- Whenever the ball is found, no matter when in the algorithm, the AIBO goes to it and tries to score.
- First it rotates 360 degrees and tries to localise the goals and the ball.
- If no ball is found the AIBO starts to go towards a goal (which goal depends on the start up position) and thereby starts walking around the field in a predetermined way to search for the ball.

3.8 Communication

3.8.1 Conditions

The AIBOs are able to communicate through wireless network cards. In order to comply with the RoboCup rules of 2003, all wireless communication must use Sony's TCP Gateway program, which is running on each AIBO.

3.8.2 Solution

The Tekkotsu framework does not have built in support for the TCP Gateway. Therefore a slight modification of Tekkotsu's core files has been necessary.

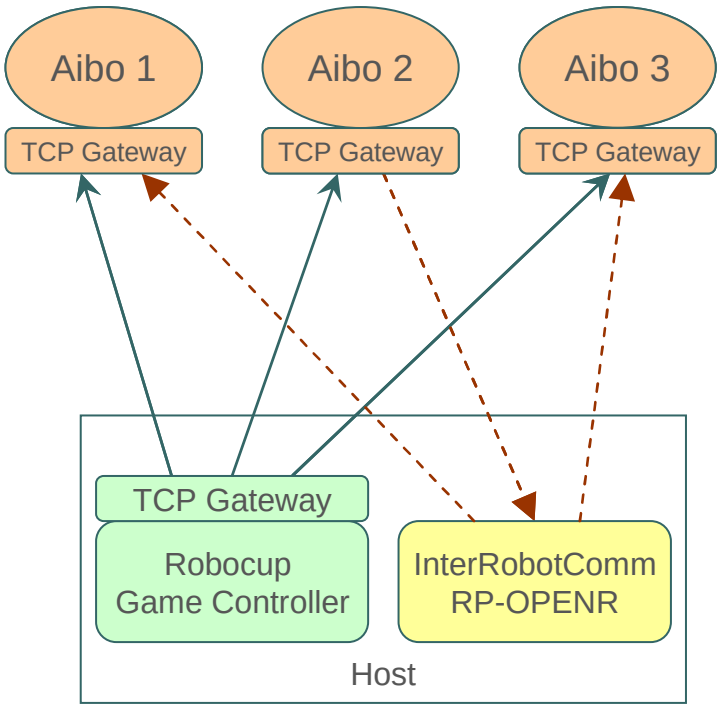

Fig. 11 Communication

All communication between AIBOs is broadcast, i.e. one AIBO sends data to all remaining AIBOs through the TCP Gateway, as shown by the red dotted arrows in the diagram below. The blue arrows indicate the one-way communication from the RoboCup Game Controller program, which all AIBOs must be able to receive.

3.9 Distribution

Every dog has its local decision tree, but to make a team of AIBO robots play more efficiently, communication is needed between the team mates. This means that we need a distributed system where important data is shared with all other dogs, but we don't need to have a leader dog. Hence, we use a simple distributed system, where data to be distributed is broadcasted to every dog in the team without being controlled by a leader process. This in essence constitutes a fully replicated distributed database.

There is no check if the packets were delivered, but since the system is not depending on distribution to work, this solution is good enough for us. Also, the data is sent often enough to correct transmission errors.

The sent and received data is merged into a summary of the team's knowledge about the current state; this state is referred to as the global world. Each dog has its own, local, copy of the global world.

3.9.1 Global world

The distributed global world contains information about all the dogs and the ball.

Information about each dog that can be found in the world is the dog's position in coordinates and confidence, its current action (what the dog is doing right now) and the distance to the ball and which zone the dog is positioned in.

The information about the ball that is stored in the global world is the ball's position in coordinates and confidence.

3.9.2 Methods in the distribution module

The data being sent and received is of the type DistributionData. DistributionData contains the sender dog, the ball, and a variable if we are to swap zone and the zone to swap to.

Data can be sent in two different ways: one where the dog will swap zone with another dog, and one without swapping. Zone swapping will, for instance, occur when the dog with the ball crosses another dog's zone. The two dogs will then swap zone. How data is sent depends on if a zone is given as argument to the method.

Data is always received with the same method, and this method will automatically take care of merging the global world and, if we should swap, performing a swap. The distribution module

also handles questions concerning the global world. For example there are methods to check if anyone is getting the ball or to check if the local dog is closest to the ball.

3.9.3 Conclusion

There is a functioning distribution of a global world, but it could probably be used even more than it is now in the strategy module. Due to lack of time, we will only use the distributed information in a fairly simple way for the most basic tactics. That is, we are distributing every dog's distance to the ball, so we can decide which dog shall go to the ball. We are also using the distribution for swapping zones. Still there are methods implemented in the distribution module, and much data with strategical information, that are currently not in use in the strategy.

This will be easy to use in future extensions of the system. Other possible extensions in the future could, for example, be to distribute data processing or information about the opponent team.

4 Evaluation

The main goal of this project has from the start been to implement a complex distributed system. For this to work, all modules in the system, from vision to strategy, must function and provide the data that is to be distributed. In that sense, we have fulfilled a goal, since the world view that is used in the soccer strategy is designed a fully replicated distributed database with meaningful game data. On the other hand, the system performance from a soccer perspective is not very satisfying. In its present state, this system would never win a RoboCup match, even though it in almost all aspects satisfies the formal requirements specification.

4.1 General aspects

A strength in the project has been the choice of using Tekkotsu as a development framework. Tekkotsu has provided an intuitive environment to program the AIBO robots in. It has naturally been helpful to have working low-level vision and motion modules to build on and the design and structure of Tekkotsu has also helped in creating a structure for the Gifr system. Furthermore, the testing and debugging tools in Tekkotsu have greatly simplified the development process. We are convinced that without Tekkotsu, we would never have accomplished the task we set out to do within the time limits of the project. The only major drawback with using Tekkotsu is that it forced us to abandon the previous year's system design and code, since it was based directly on Sony's OPEN-R API.

The overall system design in Gifr is adaptive and object oriented, much due to design already built into Tekkotsu. This will ease future extensions and changes to the system. The open source mentality of the RoboCup community has also been a great advantage in our development process.

4.2 Future improvements

4.2.1 Motion

Since the Gifr system uses Tekkotsu's motion module, it is already functioning. There are, however, some small improvements that could be made. In some situations the motion is jerky and unpredictable, which should be investigated. The reason for this is not entirely clear, but could pertain to a command overflow or some sort of state collision. Beyond this, there is of course always room for new motion features such as fancy kicks as well as general tweaking of existing motions.

4.2.2 Vision

An obvious and relatively important improvement to the vision system would be the recognition of other robots, both of own and opposing team. This is not possible in the current Gifr system.

4.2.3 Calibration

The calibration tool is quite well-developed in the Gifr system, but there is always room for improvement and new features. One new feature could be the calibration of the AIBO's walk depending on the surface texture. There are also one known bug which occurs when switching between target AIBOs that needs to be addressed.

4.2.4 Localisation

The system has worked very well, and has allowed us to know our position on the field correctly at most times. If the underlying functions in the vision system are improved a bit the performance of the localisation might be improved, but there is no real problem with the algorithm, just small bugs in the mass of underlying systems that add up to some small problems with the localisations. Possible extensions of the localisation module include localisation without beacons, which would enable us to participate in some of the extra challenges that take place in RoboCup.

4.2.5 Distribution

There is a functioning distribution of a global world, but it could probably be used even more than it is now in the strategy module. There are methods implemented in the distribution module, and much data with strategical information, that are currently not in use in the strategy. This will be easy to use in future extensions of the system. Other possible extensions in the future could, for example, be to distribute data processing or information about the opponent team.

4.2.6 Communication

In next year's RoboCup competition, it will be permitted to communicate directly between the AIBO's without using Sony's TCP Gateway. Since this form of communication is more efficient, it would be desirable to implement an extension which takes care of this.

4.2.7 Strategy

The decision trees used in the strategy module need a thorough revision, since this is probably the source of most of the current problems with soccer gameplay. A clear bug is that the robots freeze during play in certain situations. Certain improvements in the goalkeeper's behaviour are also necessary, most importantly it needs to take advantage of the distributed data about the ball's position in a better way. The goalkeeper also needs be able to position itself easier, possibly with the help of a customized localisation module. The navigation system used in the penalty behaviour also needs improvement.