All Botz
The University of Auckland RoboSoccer Team

Jacky Baltes, Nich Hildreth, Robin Otte, Yuming Lin
University of Auckland
Email: j.baltes@auckland.ac.nz
www.tcs.auckland.ac.nz/~jacky

October 28, 1998

Abstract

This paper provides general information about research at the University of Auckland into autonomous agents in highly dynamic environments, in particular in the RoboCup environment. The first part of this paper describes the design of our vision server and our camera calibration method and shows some of our experiences in developing our RoboCup entry. This may help other interested parties in avoiding similar pitfalls in the future. The second part of the paper describes our general architecture, which consists of a distributed, cooperative agent architecture with powerful path planning and controller components. The path planner is an anytime planning algorithm that supports reactive as well as strategic reasoning of the agent. The controller uses a fuzzy logic approach. Lastly, the paper describes our work on the design of a case-based reinforcement learner.

1 Introduction

This paper describes current research at the University of Auckland on RoboSoccer. The Centre for Imaging Technology and Robotics (CITR) is a part of the Computer Science Department with strong links to Electrical Engineering. In 1998, the CITR offered for the first time a graduate course on "Intelligent Active Vision." This focus of this course are issues in intelligent control of autonomous vehicles in complex, highly dynamic environments. The course emphasizes high level reasoning and the integration of strategic planning with low level reasoning processes (behaviors, real time control).

As a practical environment for students to work in, we set up an "Intelligent Active Vision" laboratory. We purchased commercially available toy RC cars. The cars allow for proportional control of steering and velocity. The transmitter of these RC cars were disassembled and we built a parallel port interface for the transmitters. The parallel port interface uses a PIC micro controller and gives us 33 different speed settings (16 forward, halt, and 16 backwards)[1] and 33 different steering angles (16 right, straight, and 16 left).

Position and orientation feedback are controlled by a global vision system, which will be described in more detail in section 2.

[1] In hind-side, it turns out that all of the forward speed were too fast

The task of the students was to control the car around a simulated race track. This race "Aucklandianapolis" was very successful and proved to be very popular among students. Staff and students even from other faculties attended the races. A more detailed description can be found in [Bal98].

Given the surprisingly successful completion of the time trials, students decided that they would like to tackle more challenging problems, which lead us to look at Robo Soccer as a new domain.

This paper presents work in progress. The video server used by our system is described in section 2. Section 3 introduces the architecture of our system. Two important subcomponents of this architecture, the path planner and the controller are described in sections 4 and 5 respectively. Section 6 describes extensions to the controller that allow it to learn the correct control output using reinforcement learning. Section 7 concludes and indicates directions for future research.

2 Video Server

Currently, we are using a Pentium 200 PC running Linux as our video server. The video server hosts a Matrox Meteor compatible frame-grabber and is connected to a commercially available Cam-corder.

The camera is mounted on a tripod (about 2.7m above the playing field). Given the current setup, our playing field is limited by the field of view of the camera (approximately 2x4m). Because the camera is mounted at an angle, the exact playing area is a trapezoid rather than a rectangle.

2.1 Camera Calibration

The problem of camera calibration is a very fundamental problem in Computer Vision. The input to a calibration method is a set of known coordinates and the output is a set of external and internal parameters of the camera model. Given this calibrated camera model, it is possible to determine the real world coordinates of image points (if at least one dimension is known) or compute the image coordinates for known real world coordinates.

Traditional camera calibration relies on the availability of known image coordinates for some known world points, that is the real world coordinates for at least 12 image points must be known. Once a sufficient number of matching points have been found, well known camera calibration techniques can be used. For example, the Tsai calibration method uses an eleven parameter model with six external and five internal parameters [Tsa87]. In our work, we are using a public domain implementation of the Tsai calibration method, which is available from the WWW [Wil95].

Since we are often asked to give demos of our system for different occasions, we needed an accurate, cheap, portable, and fast method for camera calibration.

The need for portability and speed of the calibration method ruled out traditional methods of using feature points inherent in the scene (since these feature points will not be available when moving to different rooms) or of painting feature points into the scene (a labor intensive and error prone task for a large set of points). The creation of a special calibration pattern of sufficient size and with a sufficient number of points proved to be impractical. For example, a large wooden board with calibration points (a) would be difficult to move, (b) may not fit into rooms that do not have similar geometry

Figure 1: Calibration Pattern as seen by the Camera

(e.g., a part of the rectangle is cut out by a wall), and (c) expensive and labor intensive to manufacture.

However, we clearly needed a portable calibration pattern, so we decided to use readily available material. We looked at a number of possibilities including carpets (have a dense texture and are expensive) and linoleum carpets (accurate pattern, but expensive and has an undesirable warping property).

In the end, we decided to use a duvet cover (250x200cm) with a square pattern on it. The back half of the duvet cover was removed to reduce artifacts due to the transparency of the cloth material. The duvet cover is well suited for our environment, since it is easily portable and can be adapted to different room outlays[2]. Drawbacks are that the cloth material stretches and warps. Both drawbacks can be minimized through the handy use of an iron. However, they can not be eliminated and thus introduce errors, which limit the accuracy of the camera calibration that can be obtained.

Figure 1 shows a picture of the calibration duvet cover as seen by the video camera.

Given the picture shown in fig. 1, our system uses a semi-automatic method for calculating the matching points. In the preprocessing step, the user removes unwanted parts of the picture, such as the table top on the left side of the calibration picture. Secondly, the color image is converted into a gray scale image and thresholded, so that only the white squares are left in the image. Currently, we are only a global threshold value on the red channel, which was sufficient for our environment.

After this initial preprocessing step, the system automatically computes the matching points. First, the system uses a simple pattern to find the white squares in the picture. This step ignores small artifacts and handles missing squares. The centre point of each square is computed by calculating the moments along the x and y direction. Then, the squares are sorted. This sorting step is of critical importance, since if it is done in the wrong order, the assigned real world coordinates will be wrong, which will result in inaccurate calibration.

After approximate real world coordinates have been assigned to the centres of all squares, the system uses four edge detection steps to find the coordinates of all four

[2]It is also a handy blanket for my graduate students when they get caught up in their work and end up sleeping in the lab

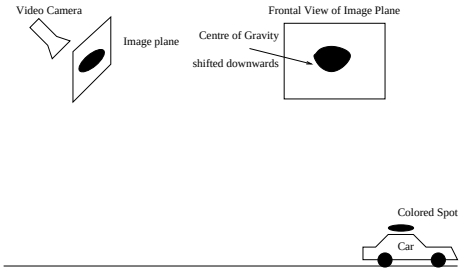

Figure 2: Distortion of a circle's centre of gravity under perspective distortion. The centre of gravity is shifted downwards as shown in the frontal view of the image plane.

corners. If a corner has been identified, it is assigned a real world coordinate by the geometry of the calibration pattern (each white square is 8 cm by 8.1 cm).

This means that the assignment of the real world coordinates to the corners is independent of the assigned real world coordinates of the centres of the squares themselves. This is an important feature in our algorithm, since the centres of objects are distorted by the perspective projection and are moved to the lower end of the picture, which means that they are unsuitable for applications that require a high accuracy. This problem is also discussed in section 2.2. Of course, given an accurate camera model, this perspective distortion can be compensated for, but this leads to a chicken and egg problem, since we are using this information to calibrate the camera.

Also we found in our tests that this two-stage approach (sort centre of squares, find corners for each square) works better than assigning world coordinates to all corner points. Missing squares or missing data points makes this one step assignment very difficult and error prone.

After the computation of the matching points, we use a PD implementation of Tsai's camera calibration to compute the parameters of the camera model [Wil95].

2.2 Image processing

The video server provides the clients with information about position, orientation, and velocity of objects in the domain. All this information is returned in real world coordinates, which has the advantage that the camera calibration (as discussed in subsection 2.1) information is only required on the video server, not the clients who perform all their calculations in real world coordinates.

To make this task easier, the cars use color coded dots (about 8cm in diameter). Initially, two types of color coding were used: (a) two colored circles (one red, one green) attached to the roof of the car. In both cases, the pixel coordinates of the position of the car were determined by the x and y moments of the projection of a circle onto the image plane, that is the centre of gravity of the circle. This method is fast, but has the disadvantage that the moments of a circle are distorted when projected onto the image plane (see Fig. 2). Currently, this effect is ignored, since the position information returned from the video server is accurate enough (usually less than 3cm error) for our applications.

After finding the real world coordinates of these circles, the orientation of the car is determined by a line through the center of these two circles. However, since the determined centers of circles are distorted, orientation information calculated using

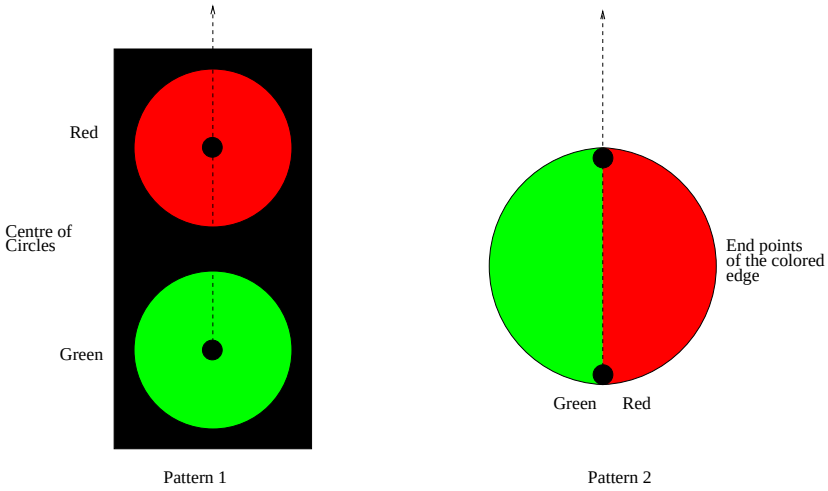

Figure 3: Comparison between the two calibration patterns that we used in our work. Pattern 1 calculates the orientation by the line through the centre of the two circles. Pattern 2 uses the end points of the red-green colored edge.

this method has a larger error (about 10 deg).

Another approach we tried was to use a pattern of two different colored semi-circles. In this case, the orientation is determined by the end-positions of the edge. This method is not susceptible to the problem of distortion due to the perspective projection. Initially, using an ad hoc method to calculate the end points of the edge resulted in better orientation information (about 3 deg). Figure 3 shows the two different patterns.

Currently, the video server is able to find the position of cars accurate to about 3cm and their orientation accurate to within 10 degrees. Other clients connect to the video server via UDP and the video server will send a message with the following information for each object (e.g., ball, car) : (a) the time stamp of the current frame, (b) whether the object has been found in the current frame, (c) its position, (d) its orientation, (e) its displacement along the x coordinate from the previous frame, (f) and its displacement along the y coordinate from the previous frame.

3 General Architecture

Figure 4 shows the overall architecture of the agent. The design goals for the agent architecture were: versatility, extendibility, and robustness.

- *Versatility* is the ability of an agent to be used for a variety of tasks. Instead of being limited to a single task, albeit a very challenging one (e.g., playing soccer), our research goal is to develop an architecture that can perform different tasks in the mobile robot domain, such as parallel parking, time trials, and office delivery.
- *Robustness*. The architecture should be robust in the sense that if some of its capabilities are reduced or removed, it should still provide a reasonable level of performance. This means in particular that if communication with other agents

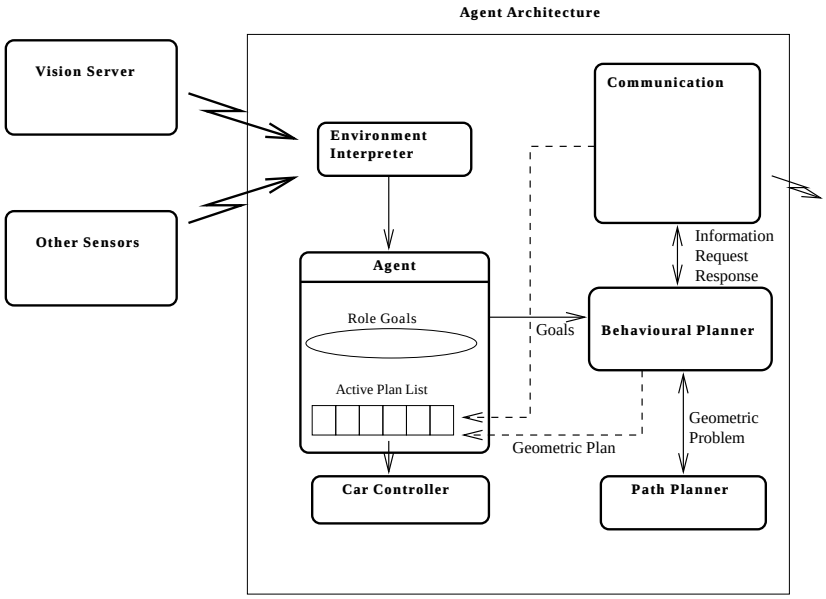

Figure 4: The Agent Architecture.

is interrupted, the agent should be able to perform at least some limited functionality on its own.

- *Extendibility*. It should be easy to add additional behaviors and functionality to the agent so that it is able to improve its performance on a task.

Our agent is based on a distributed role based architecture using a behavioral planner. Vision as well as other sensory information is received by the agent and processed in the *environment interpreter*. This interpreter will pass some of the information on directly to the agent, such as the current position or orientation. However, some other features of the environment may require significant processing (e.g., is our team on the offense or defense). The agent continually checks the incoming environment information and selects individual goals from its role goal base. As will be shown in subsection 3.1, different players have different roles and are therefore trying to achieve different goals.

Whenever the agent finds a suitable goal it is passed to the behavioral planner. The behavioral planner contains a database of procedural information for achieving different goals in different situations. For example, it contains a low level behavior to orient itself towards the ball as part of its overall scoring behavior. If the agent is unable to achieve a goal directly, it can invoke the communications or path planner components.

The communication component can send and receive messages from other players. In general, the communications can be grouped into three main categories: information ("I am trying to score a goal"), request ("I am open, pass the ball to me"), and response communication ("I will pass the ball to you").

The path planner solves geometrical problems, such as moving the player from one position to another one. The details of the anytime path planner are described in section 4.

Once a feasible plan has been found by the path planner, the plan is send to the agent, who puts it onto the *Active Plan List*. The *rank* of a plan in the queue is determined by the environment (e.g., if there is no opponent in the way, this plan is prefered over one that has an opponent in the way.), the goal that the plan is trying to achieve (e.g., we prefer to score goals over defensive goals), and the complexity of the plan (e.g., we prefer small plans that have a higher chance of success).

The agent selects the most promising plan from the active plan list and compares its rank with the currently executing plan. If the new plan has a higher rank, the currently executing plan is pre-empted and put on the active plan list. The controller will then start to execute the new plan. Details of the controller are described in section 5.

An important feature of the architecture is its extendibility. Other skills can be added to an individual agent. The architecture uses the concept of roles, which are described in subsection 3.1 to manage the possible explosions of actions that the agent may perform.

3.1 Roles

Roles are as important in RoboCup as they are in real soccer, since otherwise all players would simply chase the ball and get into each others way. The idea is that invididual agents are assigned specific roles on the playing field.

Our architecture currently includes the following five players: center, striker, defense, and right and left wing. The individual players have a number of restrictions put on them to simplify planning. Firstly, each role has a certain set of possible goals, i.e., things that it is trying to achieve, associated with it. Secondly, the individual players are assigned certain zones, which limits their movement. Figure 5 shows the zones of the different players.

For example, a striker only has one possible goal in its behavioral planner, scoring a goal, and is limited to the striker zone (3/4 of the playing field closest to the opponents goal).

A defense player's only goal is to stop a ball from going into its team's goal. Therefore, a defensive player will never even attempt to shoot at the other goal and is limited to movement in its own quarter of the field.

The right and left wing players have a variety of possible goals, which include scoring as well as blocking opponents and passing a ball. They are limited in movement to the right and left half of the field respectively, but can move along the full length of the field.

The center player is the only player with no restrictions on its movement. Its primary goal is to stay close to the ball and to support the striker.

The roles of a player are static and will not change during execution of the game. The different roles are implemented through different sets of behaviors in the behavioral planner.

4 Path planner

Path planning is an important problem in strategic planning for a mobile robot. Given the current position and the desired goal configuration, the mobile robot must come up with a sequence of movements that will take it from the initial state to the goal.

Currently, we are using Bicchi's path planner [BCS95]. This planner is based on Reeds and Shepp curves and finds the shortest path of bounded curvature amongst

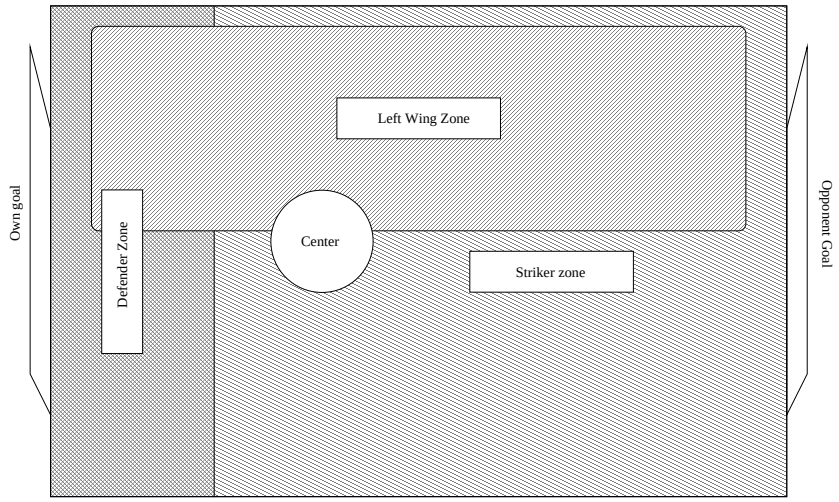

Figure 5: The zones of the individual soccer player roles in our architecture are shown in this figure. The zone of the right wing is symmetric to that of the left wing, but is left out for readibility.

polynomial obstacles. . This planner is very similar to visibility graph based planners in holonomic vehicles, but the vertices are augmented by circles of maximum turning radius. The planner adds circles with the maximum turn radius around the vertices of all obstacles and then finds the shortest path consisting of arcs around the circles and straight lines between circles from the start to the goal. The straight lines between circles connect the points on the circle with identical tangents. Thus there are at most four lines between circles. Any lines that intersects an obstacle is removed. A very simple example is shown in Fig. 6.

To find the shortest path, the planner performs a search through the space of possible plans. We currently use A^* with the distance to the goal as heuristic function. This works well as long as the car does not need to backup to reach the goal. Execution times on a Pentium 200 PC are between one to ten seconds for most problems. However, our cars have a comparatively large turning radius (ca 90cm) for our playing area, so that the car often has to backup to complete a turn. In this case, execution times are much worth, since almost always does the car have to move away from the goal. Since the heuristic evaluation function is based on the distance to the goal, the planner will search all possible ways of moving forward first, before backing up.

Working in a highly dynamic environment, the agent must often react very quickly ("instinctively") to oncoming threats and will therefore have not sufficient time to generate a complete path. Often even the problem of formalizing the domain in a representation suitable to support planning is too expensive. Therefore, in recent years, planners with very limited representations and quick reaction times have been successfully developed for a number of domains, the most famous one being Agre's Pengi system. On the other hand, however, strategic planning is important for the agent to achieve some long term goal. This will, for example, prevent the agent from painting himself into a corner.

We suggest an anytime path planner. As for any planning system, the input is a description of the current state and the goal states and the output of the path planner is

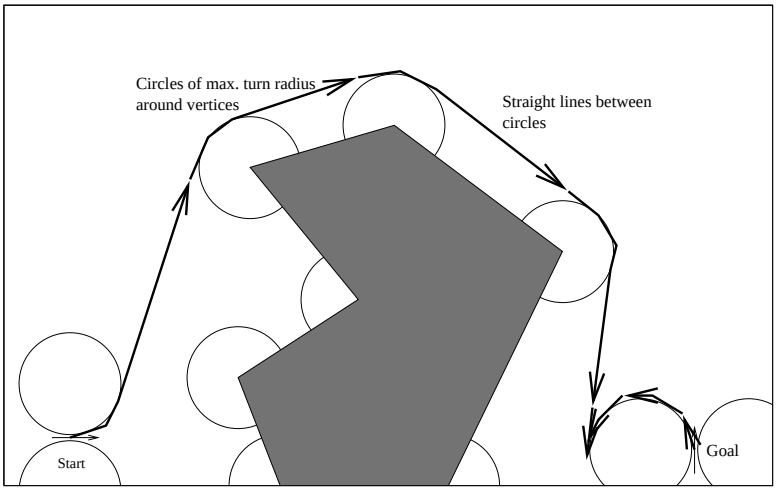

Figure 6: Sample execution of Bicchi's path planner is a simple path planning problem.

a plan to reach the goal state. However, the space of possible plans is searched in such a way that the controller can continously ask the path planner for the current best plan. This means that if a truck is threatening to crush the agent, the controller can ask for a plan to get out of the way right now. In this case, the planner will return its best guess immediately. On the other hand, if more processing time is available, the planner will continue working on the plan and improve it.

The initial plan is formed by finding the orientation from the current position to the goal, which can be done very quickly and will provide the initial guess. The anytime path planner then checks to see whether the current plan is free from obstacles or not. Should the direct route intersect the side of an obstacle, the two vertices on either side of the obstacle become the goal nodes and the planner is called recursively to generate plans from the initial state to the obstacle vertex and from the vertex to the goal. An example of the operation of the any-time path planner for the same problem as the one shown in Fig. 6 is shown in Fig. 7. The initial plan directly from the start to the goal fails. The vertices of the side that are first intercepted by the plan are used as subgoals. In this case, a plan for the upper vertex is generated. This plan will also fail and the refinement of the plan continues recursively.

Currently, the planner uses depth-first search, which means that the returned plan is not necessarily optimal. Therefore, the planner uses a quality function that will reject inefficient plans, for example plans that contain a lot of reversals.

Note that the generated plan is a holonomic plan and may violate some of the non-holonomic constraints of the car (e.g., an immediate turn by 90 degrees). A post-processing step is therefore necessary to turn the holonomic plan into a feasible non-holonomic one. Assuming that the environment is not too cluttered with obstacles, sufficient free room should be available for this conversion.

The plan generated by the path planner consists of straights and maximal turns to the left and to the right. However, because of uncertainty in the actuators, there is a

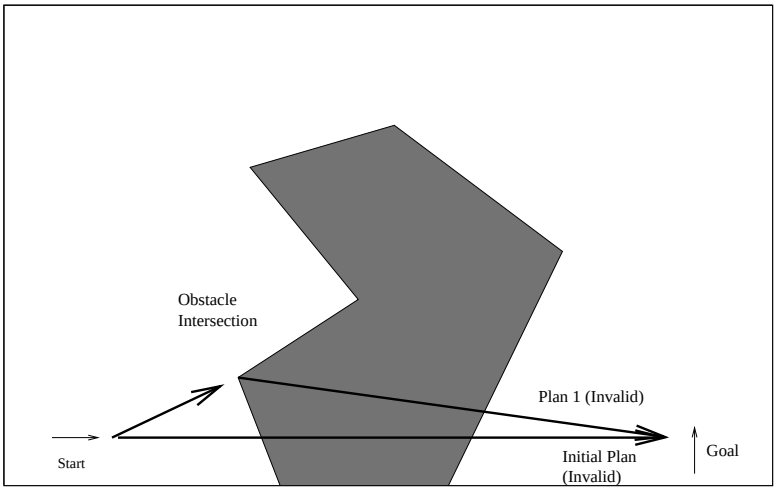

Figure 7: Sample execution of the anytime path planner on a simple path planning problem.

significant error when, for example, going along a straight line. Therefore, we need a controller to supervise the execution of the plan.

5 Controller

The task of the controller is to follow and execute a path or trajectory that was generated by the path planner. As mentioned previously, a path consists of either straight lines or maximum turns to the right or left. We are currently focusing on the path tracking problem, but hope to extend our work to the trajectory tracking problem in the near future.

We used Balluchi's sliding mode controller [BBBC96]. This controller makes two simplifying assumptions: (a) the velocity of the car is static, and (b) the control outputs are maximum turns left or right (*bang-bang control*). In other words, the controller does not change the speed setting and only chooses between full right and full left. The controller does not use knowledge about the structure of the path, in particular the path's curvature, but only uses the distance and heading errors to the nearest point on the path. One disadvantage of sliding mode controllers is that inputs to the actuators are discontinuous, that is the controller may try to steer the car instantaneous from full left to full right, which is a physical impossibility. A smoothing function is used to smooth out the controls.

This controller performed well in simulations, but in practice caused some problems. In fact, most of the time spent on Aucklandianapolis by students was trying to improve the control. The first problem was the bang-bang control itself. Imagine being a passenger in a car whose driver turns full left as soon as he drifts a few centimeters to the right of the centre line. Should you survive such a ride, you may just decide to use public transport more often. This problem was overcome by smoothing out the control

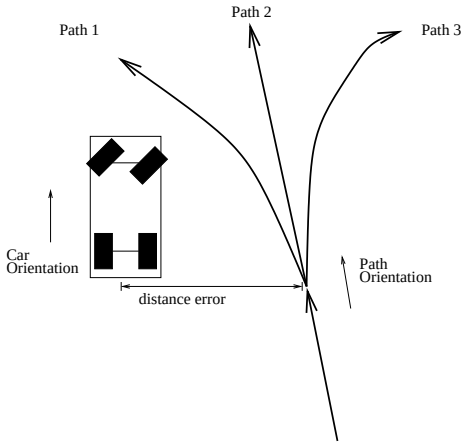

Figure 8: Problem showing the need for at least partial knowledge of the path curvature.

of the car whenever it is close to the desired path, i.e., its distance and error headings are small. In this case, instead of full turns, the controller would use small turns only. Although this works well for tracking straight lines, it is detrimental when trying to follow a maximum turn circle, which is often created by the path planner. In this case, anything but full turns will lead to the car drifting off the desired track very quickly.

Another disadvantage of the bang-bang controller is that it is very sensitive to delay. There is a small but noticeable delay between a frame being captured by the video camera and the information being received by the client (controller). In a sense, a bang bang controller waits for the last possible instance to change the direction of the car, but since the car has already traveled further in the delay between the information being send from the video server and being received by the the controller, the controllers output will occur too late.

Another big disadvantage is that the controller ignores the curvature of the path[3]. A problem that illustrates the need to know the curvature of the path is shown in Fig. 8. Although the position and orientation errors are identical for all three parts, turning to the right is only correct if following path 2 or 3. If the car follows path 1, a full turn to the right forces the car to intercept the path at almost a 90 degree angle (a configuration that is difficult to recover from).

Based on these observations, we are currently developing a fuzzy logic controller for the car. The fuzzy controller mimics the behavior of Balluchi's controller if either the distance or orientation error are large, since in these cases Balluchi's controller works very well.

However, the fuzzy logic controller uses as inputs the distance error, orientation error, and curvature of the path. Instead of only controlling the steering angle, the fuzzy controller controls velocity as well as steering. An example of a fuzzy rule is shown below:

```
if distance error is SMALL and orientation error is
SMALL and curvature is STRAIGHT,                    .
then go STRAIGHT ahead and go FAST
```

[3]Balluchi et al show how the controller uses the sign of the curvature, however, we found while implementing the controller that the sign of the curvature cancels in the equations

The fuzzy sets (e.g., SMALL) used in the controller are indicated by uppercase in the example rule. Simulation has shown some promising results in the inverted pendulum domain for the fuzzy controller.

6 Reinforcement Learner

Although fuzzy controllers such as the one described in section 5 are conceptually simple, their success critically depends on the choice of parameter values. The controller in section 5, for example, has about 60 parameters. Finding a good set of values for these parameters is often non-intuitive.

Another approach to the problem of path tracking is the use of un-supervised learning methods. Especially Neural Nets (NN) and Genetic Algorithms (GA) have been used successfully in a large number of domains.

Reinforcement learning and Q learning are other learning methods that have been used extensively in robotics and AI [RN95]. The idea is to learn the correct actions for an agent in different states. The difficulty is that in the environment, the effect of actions can only be determined in some later stage. For example, a bad move in chess may lead one to loose a game, but often the loss will not occur until many moves later. Similarly in the path tracking domain, an incorrect steering command will only sometime later lead the car to miss the ball when trying to kick it. This time delayed reward leads to the *credit assignment problem*. For example, even though the controller did do the correct control actions for the last ten steps, the car missed the ball because of a steering decision eleven steps previously. The difficulty is to assign the blame for the failure to the actions that caused the failure.

Q learning needs to be adapted when used in the path tracking domain. Firstly the notion of discrete time and discrete actions (situation calculus) needs to be extended to include the continuous time and actions in which the agents execute. Secondly, the concept of a state needs to be defined.

In the reinforcement learner, we use a similar state representation as suggested by the controller. A state is defined by three variables, the distance error, the orientation error, and the curvature of the path. The reward is calculated as the sum of the position and orientation errors over a fixed number of previous time steps. Note that in the path tracking domain, reward is immediately available, since the current errors can be easily determined. Our Q learner, however, sums up a number of previous errors to get a better estimate of the quality of the current path.

To speed up the learning, the reinforcement learner uses a case based approach to select reward values from similar states, when no Q values for a given state are known.

7 Conclusion

This paper describes some of the main aspects of our work on automatic navigation of non-holonmic robots in highly dynamic environments and how we transfered our approach to the domain of RoboCup.

This is work in progress. RoboCup is a very interesting and challenging problem. Currently, we have far more questions than answers. For example, What is a good balance between strategic and reactive planning in environments such as RoboCup?

We hope that the integration of a fast anytime path panner, a robust controller and a distributed control architecture will lead to a system that will allow us to perform a

variety of tasks, such as parellel parking, racing, and RoboCup.

There are many possibilities for further research in this area. One possibility is to move from a global vision system to a local one. This is a route that we would definitely like to explore in the future. One of the advantages of the project so far are its relatively small cost. All components can be readily purchased. On the other hand, the use of toy RC cars caused some problems, which we hope to address in the future. We are working on a microprocessor board that can be installed in the cars to connect sensors and actuators as well as a motor drivers, which will improve our control over speed and steering.

References

[Bal98] Jacky Baltes. Aucklandianapolis homepage. WWW, February 1998. http://www.tcs.auckland.ac.nz/ jacky/teaching/courses/415.703/-aucklandianapolis/index.html.

[BBBC96] A. Balluchi, A. Bicchi, A. Balestrino, and G. Casalino. Path tracking control for dubin's cars. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[BCS95] Antonio Bicchi, Giuseppe Casalino, and Corrado Santilli. Planning shortest bounded-curvature paths for a class of nonholomic vehicles among obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1349–1354, 1995.

[RN95] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*, chapter 20, pages 598–624. Prentice-Hall Inc., Englewood Cliffs, New Jersey 07632, 1995.

[Tsa87] Roger Y. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.

[Wil95] Reg Willson. Tsai camera calibration software. WWW, 1995.