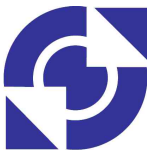
Ball-handling motion control for soccer playing mini-robots

A Master's Thesis
in Computer Science

by

Maarten Dimmen Buth



Dr. A. L. Schoute
Dr. M. Poel
Dr. ir. A.F. van der Stappen

Distributed and Embedded Systems Group (DIES)
Department of Computer Science
University of Twente
The Netherlands

April 2006

Abstract

This thesis describes improvements with respect to motion planning and control in the MI20 robot soccer system. Technologies from the fields of artificial intelligence, control theory, embedded systems and motion planning are used. To compete in the MiroSot category with mini-sized mobile robots requires playing skills for ball handling control. The existing implementations of these playing skills were unsatisfactorily in prediction, preplanning and control. Therefore the planned paths in motion planning are extended with the more flexible Bézier splines. By adding time or distance as parameter to such splines, trajectories results, that completely specifies the desired motion. To keep a mobile robot on a trajectory a timed controller is used. This control system uses a velocity profile and a trajectory as feed-forward control and uses feed-backward control based on measurements to correct the robot due to the uncontrolled dynamics.

The results show a more flexible planning when spline curves as trajectories are used. It requires less computation time due to the ease of differentiating and integrating by numerical methods. The new control system shows an accurate trajectory tracking when using feed-forward as well as feed-backward control at low speeds. A disadvantage of trajectory tracking is the dependency on accurate position data. Without accurate measurements, tracking robots with high velocities is limited. High velocities of the robots are also limited by the uncontrollable robot dynamics.

Samenvatting

Deze scriptie beschrijft verbeteringen ten aanzien van bewegingsplanning en aansturing van mobiele mini robots in het MI20 robot voetbal systeem. Hiervoor wordt gebruik gemaakt van onderzoeksgebieden in kunstmatige intelligentie, wiskundige systeemtheorie, embedded systemen en bewegingsplanning. Om competitief te zijn in de MiroSot categorie zijn vaardigheden vereist die de bal kunnen controleren. De huidige implementatie van deze vaardigheden is onbevredigend in voorspellingen, het flexibel herplannen van paden en het besturen van de robot over deze paden. Deze paden worden flexibeler door Bézier splines te gebruiken met een tijdsafhankelijke parameter wat gezamenlijk trajecten genoemd wordt. Doordat de robot op zijn traject moet blijven wordt er een nieuwe controller gebruikt die op tijd bijstuurt. Deze controller bestaat uit een open-loop controller en een gesloten-loop controller. Het traject en een snelheidsprofiel worden gebruikt voor de open-loop controller. De gesloten-loop controller maakt gebruik van metingen om de robot stuur signalen uit de open-loop te beïnvloeden om zo het dynamische gedrag te corrigeren.

De resultaten laten zien dat het gebruik van splines als trajecten een behoorlijke flexibelheid geeft ten opzichte van de huidige paden. Doordat trajecten eenvoudig numeriek te differentiëren en te integreren zijn, zijn er weinig berekeningen nodig waardoor de benodige processor tijd beperkt blijft. Een nadeel van de nieuwe controller is de afhankelijkheid van nauwkeurige meet gegevens. Zonder nauwkeurige meetgegevens is de controller beperkt in het laten rijden van robots op hoge snelheden. Beperkingen ten aanzien van de hoge snelheden hebben ook te maken met de dynamische aspecten welke steeds belangrijker worden naar mate de snelheid groter wordt.

Preface

This Master's thesis describes the results of my graduate work at the MI20 project during the last ten months of my study. It has been a tremendous experience working on the motion planning and control part in the MI20 robot soccer environment, which is an ideal testbed for advanced technology research.

In spite of the fact that we where not always able to be competitive in the MiroSot category, due to hardware and software problems, it was educational and enjoyable. Especially the experience during the journeys in Germany (Paderborn), France (Belfort) and Austria (Vienna) where enjoyable. Just like doing interviews with radio and television during the European championship 2005 organized at the University Twente.

I owe much gratitude to the following people. In the first place my parents which have assisted my with moral support and have had patient with me during my $6\frac{1}{2}$ years of study. The (former) members of the MI20 team and especially Paul de Groot for supplying a well designed framework for the revised MI20 system. Charles Petit and his wife for having a great time in France and of course I would like to thank my supervisors Albert Schoute and Mannes Poel for the great support, both morally and technically. And last but not least my lovely girlfriend Marianne Kiers for the endless patient and love she has given to me especially at the end of writing this Master's thesis.

Enschede, April 2006
Maarten Buth

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Robot soccer

In a robot soccer game autonomous robots move through a field without any human interaction. Many kinds of robot soccer competitions exist. We compete in the MiroSot (Micro-Robot World Cup Soccer Tournament) category. In this category rules are defined by the FIRA[1]. The game is played with wheeled mobile robots, the robot sizes are 7.5 cm in cubic, see figure 1.1. An overview of a MiroSot category game is shown in figure 1.2. In the MiroSot category several leagues are defined that differ in the size of the field and the number of robots allowed on the field. We currently participate in two of them, namely the MiroSot middle and large League. The middle league uses five robots on a field of 220cm x 180cm, while the large league uses seven robots on a field of 280cm x 220cm. Each robot

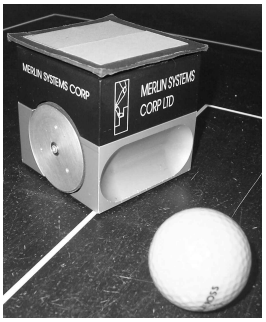


Figure 1.1: Robot and ball

is equipped with a color patch on the top side, which makes it possible to recognize the robot within a camera image. A camera is mounted above the playing field to capture these images. These images are received by a central computer system that has to determine appropriate commands for each team robot. These commands are send by wireless communication channels to the team robots.

[1]Federation of International Robot-soccer Association (http://www.fira.net)

1.2 Overview of the MI20 system

The MI20 robot soccer project started in 2002 at the University of Twente as a co-operation between the research groups Distributed and Embedded Systems (DIES) and Human Media Interaction (HMI). The MI20 system is a software system on a central computer. It is an ideal test bed to experiment with different kinds of techniques and at the same time it has to be competitive in the MiroSot category. The MI20 software system has to control all the team robots during the game. The research topic deals with improvements concerning this software system. Therefore an overview is given of the used hardware and the current software system.

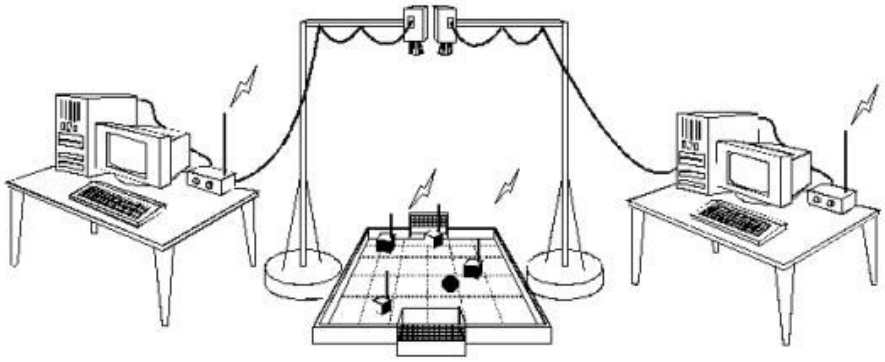


Figure 1.2: Robot soccer game setup

1.2.1 Hardware overview

The robots we use are all differential drive mobile robots. They turn by differentiating the wheel speeds. The current robots[2] (called GermanBots) of the MI20 team are bought from the University of Dortmund. Since 2005 the MI20 team is in the possession of MiaBots, bought from Merlin Systems Corp. Ltd[3].

Central computer system A central computer with a linux operating system[4] is used for receiving camera images. On this computer system the MI20 software is used for sending appropriate wheel velocities.

[2] http://www.robosoccer.de/
[3] http://www.merlinrobotics.co.uk/merlinrobotics/
[4] Fedora core 4

Communication For each MiaBot a separate channel is used to communicate with, while every GermanBot communicate through the same channel. A radio frequency (RF) receiver on board of the GermanBots uses frequencies of 418MHz and 433MHz to communicate with. This on board receiver is available for receiving velocities for the right and left wheel. These velocities are send via a radiometrix BIM Transceiver Module shown in figure 1.3 to the robot. This radiometrix module is connected to the RS232 port of the computer system. The velocities are send in a package broadcasted to all team robots. A MiaBot is capable to receive velocities as well as predefined commands see the MiaBot protocol[21]. To communicate with a MiaBot a Bluetooth interface with a range of 10 meter is used. Therefore the computer uses an USB-Bluetooth dongle and the MiaBot uses a Bluetooth receiver on board

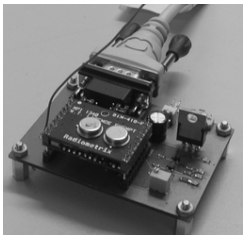


Figure 1.3: BIM transceiver module

Ball An 45.93 grams orange golf ball is used. This diameter may not be less than 42.67 mm and its shape may not differ significantly from a symmetric sphere[5].

Robot Types Both robot types have two electronic motors connected with two wheels. These two motors are powered by six 1.5V batteries. Both motors are controlled by a PID (Proportional Integral Derivative) control loop. This PID control is integrated in the on board controller and attempts to drive the wheels to the received velocities. The two robot types differ in shape and weight, but differ the most in performance. The MiaBot excels more at properties like fast, small, robust, secure, competitive and scalable, while the GermanBot has better grip and a better road-holding. Both robot types need an unique hardware identifier. The GermanBots are equipped with a dip switch to setup an unique identifier, as a result of this it is limited to handle eight GermanBots. The MiaBots are also equipped with a dip switch, but this is not used, because each Bluetooth link is a dedicated secure two-way channel established exclusively between the two devices (the pairing). Therefore each MiaBot consist of an unique 48-bit hardware address.

Camera To measure the position of the ball, the position of each robot and the orientation of each team robot a Sony digital camera is used with an ieee1394 (Fire

[5]http://en.wikipedia.org/wiki/Golf_ball

wire) interface. This camera generates thirty images (called images frames) per second and is connected to the computer system.

Colors In MiroSot each robot is required to have a color patch on top for recognition. The team color contains either a blue or yellow region and must cover at least thirty percent of the top side. These color requirements are needed to identify robot positions on the camera images and to be able to distinguish the robots from both teams. The MI20 team has identical color patches chosen to reduce CPU load during color segmentation. The prohibited colors to use are the color assigned to the opponent and the orange color as assigned to the ball.

1.2.2 Software overview

The MI20 software system written in C++ is responsible for detecting the robots in the field and sending the appropriate wheel speeds. In September 2005 a start is made to revise the software system. The new software system is currently in operation, so for more details see inside documents. Nevertheless we enumerate the most important shortcomings of the old software system

- A response control delay of ±7 image frames, which corresponds to $\frac{7}{30}$ seconds.
- Image processing overload leading to a loss of ±2 out of 10 image frames.
- A CPU load of 80% up to 100%.
- Up to 8% CPU usage for Inter-Process Communication.
- An unstructured software system without design patterns.

Adequate ball-handling in a highly dynamic environment requires a stable and structured software framework. This framework should allow accurate motion planning and control in order to shoot a ball of 42.7mm in a field of 2200mm by 1800mm with 5 uncontrolled opponent robots and 5 controlled team robots in the 5x5 league.

The old software system given in figure 1.4 is revised into the new software system given in figure1.5. Mainly the inter process communication is redesigned whereas most software processes are preserved. The new inter process communication is based on an event driven model.

Vision Vision is the first module in the control loop, it receives the raw video data from the camera. The vision module segments these image frames by means of color segmentation (See previous theses of N.S. Kooij[17] and E.M. Schepers[23]). This process detects the ball, team and opponent robots and assigns position to it and assigns an orientation to each team robot. Together this data is called a snapshot and is sent to the state estimator. The vision module also calibrates the field by making use of four orange markers placed on known positions.

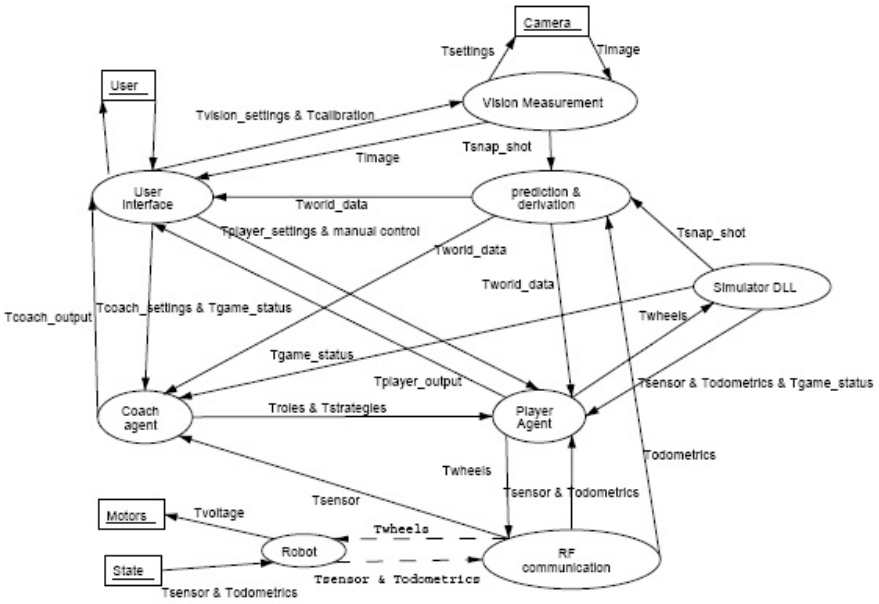


Figure 1.4: Overview of the old software system

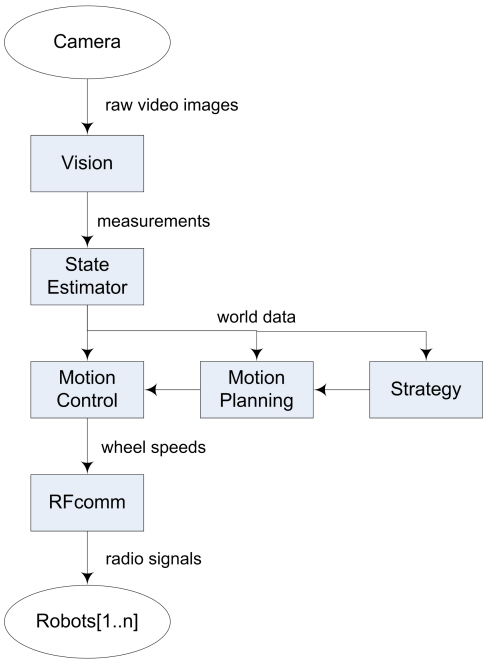


Figure 1.5: Overview of the new software system

State Estimator The task of the state estimator is to provide a consistent state by tracking each robot and the ball. Because our team robots have identical top sides, the state estimator has to identify the robots continuously. This is reached by combining the measurements with the predictions. The measurements of ball and robots are subject to measurement noise. To acquire stable state estimation the measurement data is filtered with an Extended Kalman filter.

Strategy This module is entrusted with assigning actions to the corresponding robots. Strategy is a long term process, so not every frame a new action is generated. For this module it is necessary to rely on well performing playing skills from motion planning.

Motion Planning The playing skills invoked by the strategy must be planned to perform well. The motion planning task is to generate paths intelligently to deal with solving collisions and to generate an optimal path as possible.

Motion Control Generated paths are executed by motion control. Its task is to keep the robot on track. This is done by reducing the error between the measured state of the robot and the theoretically generated path while maintaining the time constraints. That is why motion control needs world data from the state estimator and path data from motion planning in order to generate wheel velocities.

RFcomm Wheel velocities are send to each robot by matching each robot id to a communication channel with either a BIM Transceiver or Bluetooth interface. Wheel velocities for the GermanBots are first collected in a package before broadcasting. The wheel velocities for the MiaBots are sent in a secure two-way channel.

User interface The User interface is used for representing the system state to human observers. It is very useful to select a game state, or to calibrate the field by means of a well designed graphical user interface. As it is not required for controlling robots it is not taken into the system overview. The user interface listen to new world data events and represent this valuable feedback on the screen and the user is able to interrupt the game at all times.

Simulator For creating the perfect world the simulator creates snapshots with wheel speeds sent by motion control and present it to the state estimator. The most important property is to simulate the real world by modeling physical and dynamic aspects, thus collisions between robots, robot-wall collisions, robot-ball collisions and ball-wall collisions has to be considered for example. There exists also a given FIRA simulator, which is more suitable for simulating strategies.

1.3 Problem definition

In the old system both motion planning and motion control were build without a clear defined interface, which has led to an unstructured software system. The Extended Kalman Filter in the state estimator does not provide stable world data to motion planning and control in the old system. For example, if the robot is commanded to move while colliding, the filter has to detect that measured data is more reliable than the sent control signals. The old system uses planned motion which is not flexible enough and does not prevent planning paths through the border of the playing field. The cost of building a path was not considered in the old system, just like it had no time constrained parameter. An issue of controlling a robot is that it has to consider dynamic aspects to keep the robot on the path, therefore velocities have to be restricted in relation to the sharpness of the curve. The most essential problems of the existing motion planning and control are the following shortcomings:

- The MI20 system contains a huge frame delay and a couple of frames frequently disappears due to the violation of real time constraints.
- The accuracy of the measurements is low.
- Playing skills are not capable to control a rolling ball.
- The shoot skill with the old motion planning works only with a non moving ball.
- Planned motion is not flexible enough for ball-handling skills.
- Control of planned motion is not time dependent.

1.4 Objectives

The robot soccer game of the MI20 team offers an ideal test bed to design and evaluate approaches in motion control and planning technology. The research topic of this thesis is to improve motion planning and to introduce a new control mechanism for trajectory tracking to improve ball-handling skills. We attempt to reach this by designing time-based trajectories in an efficient, flexible and optimal fashion. These trajectories will be controlled with a new designed time-based motion controller to keep the robot on track. Measurement data is important for accurate tracking, therefore we have to upgrade the used filter with system constraints. All this is done under restrictions with regard to the nonlinearity of the system model, the physical nature of wheeled mobile robots and system delay. Aspects that are of specific interest are:

- Is a better measurement and prediction accuracy of robots and ball possible?
- Can we create paths which are flexible, scalable and controllable with respect to robot dynamics?
- How accurate are paths tracked with motion control?
- What is the maximum linear velocity with a given angular velocity?
- How well do ball-handling skills perform with these trajectories in a high dynamic environment?

With these objectives, we attempt to improve prediction, preplanning and control to obtain better playing skills of the robots, which are necessary to win games.

1.5 Thesis outline

This thesis will further introduce properties and constraints with regard to the used wheeled mobile robots, which are given in chapter 2. This chapter also focuses on improvements required for better prediction and research of the motion planning and control topic. The design of the planning trajectories is described in Chapter 3. Chapter 4 describes the controller design to track these planned trajectories. The design and improvements of playing skills is described in Chapter 5. The last two chapters cover the performance, evaluation tests and conclusions.

Chapter 2

Improving motion in the MI20 system

Moving a robot in a controlled fashion is one of the basic needs to play a robot soccer game well. Fundamental improvements have to be done in the MI20 robot soccer system with regard to motion control. First, the position estimation of the mobile robot must be properly performed and precisely given by the state-estimator based on vision data. Second, motion planning has to find an optimal path from the start pose to the goal pose with avoiding collision to obstacles, and minimizing performance requirements. But before we consider these motion control improvements, it is necessary to introduce some basics about our differential drive robots.

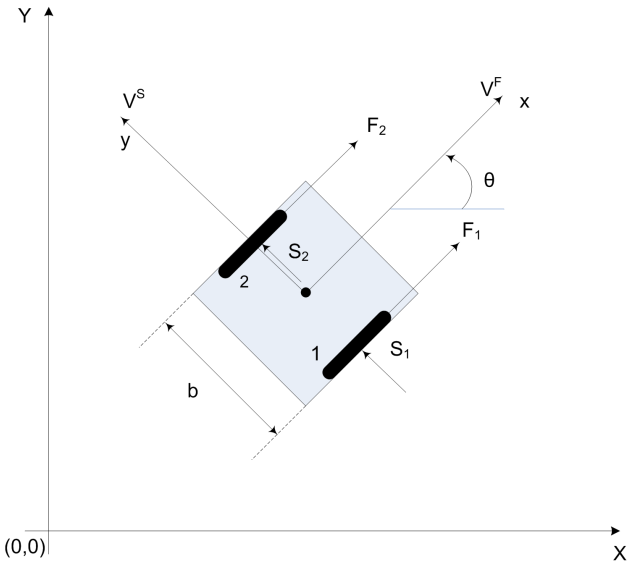


Figure 2.1: Robot architecture and symbols

2.1 Introduction into differential drive robots

Our robots are two wheeled differential drive mobile robots, like figure 2.1. To represent such a physical robot an orientation θ and a 2-tuple point (x, y) in a two dimensional Cartesian coordinate system[1] is used. Such a point is the position of the robot and is relative to the fixed environmental frame $\mathbf{F}_W$, which is our playing field. Where the robot orientation is denoted by the angle between the heading direction and the x-axis. To move a differential drive robot within the environmental frame, we have to know the kinematic model and the constraints it has in the environmental frame with respect to its own configuration.

Configuration of a physical robot The idea is to represent a robot A as a single point and thus reduce the motion planning problem to planning for a point. The reference point is defined as the origin of the robot frame $\mathbf{F}_A$. To specify the configuration of a physical robot A, it is enough to specify the position and orientation of the frame $\mathbf{F}_A$ with respect to $\mathbf{F}_W$. Where the configuration $\mathbf{q}$ is the state (see eq.2.1) of the robot and also called the pose or posture of the robot.

$$q = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.1}$$

Configuration space of the physical robot The configuration space of a physical robot is the space $\mathbf{C}$ of all valid configurations of the physical robot in its environment. The definition of the configuration space is given in Latombe[18].

Holonomic and nonholonomic constraints A differential drive robot can reach any position and orientation by moving backward and forward while turning appropriately. Thus it does not have a so called holonomic constraint which reduces the dimension of the configuration space and is often written in the form

$$G(q) = 0$$

However a differential drive robot is limited in its movements, which is the so called nonholonomic constraints[8]. These constraints include restriction on instantaneous state changes and take the form of

$$G(q, \dot{q}, \ddot{q}, ...) = 0$$

For our differential drive robot a limitation is the fact that it can not move transversally instantaneously. The mathematically representation of the nonholonomic constraints for our differential drive robot is given by

$$\dot{x}.sin(\theta) - \dot{y}.cos(\theta) = 0 \tag{2.2}$$

[1] http://en.wikipedia.org/wiki/Cartesian_coordinate_system

So if $\theta = 0$, then the velocity in the y direction is zero ($\dot{y} = 0$) and if $\theta = \frac{\pi}{2}$, then the velocity in the x direction is zero ($\dot{x} = 0$). While planning and tracking paths, the system has to consider these constraints.

Kinematic model The (first order) kinematics deals with the relationship between control parameters and the behavior of a system in state space[8]. So in our case it is concerned with the motions of a robot without reference to dynamic forces. The differential equations that describe the kinematics of the robot are given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.3}$$

Usually a differentially drive robot has two control inputs; the velocities of the left and right wheels, v_l and v_r. These are related to the linear or tangential velocity v and angular velocity ω by

$$\begin{aligned} v_r &= v + \tfrac{\omega L}{2} \\ v_l &= v - \tfrac{\omega L}{2} \end{aligned} \tag{2.4}$$

The forward and inverse kinematics are two well known problems in robotics. The forward kinematics problem determines the pose that is reachable given the control inputs of the robot. It also describes a nonholonomic constraint on the motion of the robot that can not be integrated into a positional constraint. For a differential drive robot the forward kinematics becomes:

$$\begin{aligned} x(t) &= \tfrac{1}{2} \int_0^t [v_r(t) + v_l(t)] cos[\theta(t)] dt \\ y(t) &= \tfrac{1}{2} \int_0^t [v_r(t) + v_l(t)] sin[\theta(t)] dt \\ \theta(t) &= \tfrac{1}{l} \int_0^t [v_r(t) - v_l(t)] dt \end{aligned} \tag{2.5}$$

But the more interesting question and more difficult to answer is just the inverse, that is, how do we select control inputs to reach a global pose or follow a specific trajectory. This is also called the inverse kinematics problem and is difficult to solve because of the non-integrability of nonholonomic constraints. Solving this is exactly what we need.

Dynamic model of the robot Dynamics or second order kinematics of nonholonomic differential drive mobile robots relates wheel forces to the kinematic model, see figure 2.1. When forces due to rolling of wheels, drag forces and other passive

forces are neglected the following equations describe the robot dynamics:

$$\begin{aligned} M\ddot{x} &= \sum_{i=1}^{i=2} F_i cos\theta - \sum_{i=1}^{i=2} S_i sin\theta \\ M\ddot{y} &= \sum_{i=1}^{i=2} F_i sin\theta - \sum_{i=1}^{i=2} S_i cos\theta \\ J\ddot{\theta} &= (F_1 - F_2)\frac{b}{2} \end{aligned} \tag{2.6}$$

In eq.2.6 F indicates the longitudinal (tractive) force between the wheel and the surface due to slip. S the lateral (cornering) force between the wheel and the surface due to skid, J indicates the moment of inertia and M the mass of the robot. For a controller design based on this dynamic model the reader is referred to F. Šolc and B. Honzik[6]

2.2 State estimation

The vision and state estimator modules are originally developed by N.S. Kooij[17]. Later on, the vision and state estimation were improved by E.M. Schepers[23]. These modules detect and predict each robot and the ball on the field. The vision module measures the position of the ball, team robots, opponent robots and the orientation of all team robots. This measurement is called a snapshot and is passed on to the state estimator. The state estimator has to generate world data from it, which is used by other modules of the system. To generate correct world data the state estimator has to track the ball and each robot. Due to the same color patch of each team robot the state estimator has to keep track of the identifier which is assigned to each team robot.

The state estimator receives, each ±33ms, snapshots from the vision module and each snapshot consist of a state per robot, see eq.2.1. Due to the physical environment (e.g. temperature, vibrations) each measurement and thus each snapshot contains some random fluctuations which is called noise. The objective is to achieve more accuracy in the estimated state of the robots and the ball. Therefore filters are used to reduce these measurement noise.

The Kalman filter introduced by R.E. Kalman[16] is a widely used method for tracking due to its simplicity, optimality, tractability and robustness. Unfortunately we have to deal with a nonlinear system model while Kalman filters are based on linear system models. Therefore the Kalman filter is extended to EKF (Extended Kalman Filter) which simply linearizes all nonlinear models so that the traditional linear Kalman filter can be applied. Although the EKF is a widely used filtering strategy, it has led to a general consensus within the tracking and control community that it is difficult to implement, difficult to tune, and only reliable for systems which are almost linear on the time scale of the update intervals [24].

A Kalman filter uses the commanded linear and angular velocity to update the estimate of the state between measurements. Therefore the commanded control

signals sent to the robots are also sent to the EKF. When using the control signals in the EKF they have to be matched with the measurement and therefore we have to mention that we have to deal with a physical delay. This delay exist of sending appropriate commands to the robot (±2 frame), physical nature of mechanical systems (±1 frames) and measurements taken by the camera (±1 frame). The old state estimator consists of an EKF but does not work properly due to:

- The time update within the EKF uses control signals and due to the ±4 measurement frames delay the state estimator has to buffer these signals with a FIFO queue. Then the EKF is able to match the current measurement with control signal sent ±4 measurement frames ago for each team robot. In the old EKF the included time shift to handle this delay did not work properly.
- Through the nonlinear nature of nonholonomic robots they are approximately linear on the time scale of the update intervals. Because collisions may occur, which is of course not desirable, the estimated state can be too far from the actual state. Then the linear approximation to the system's behavior will not be sufficiently accurate. This can lead to increasingly erroneous state estimates, a phenomenon known as *filter divergence*[8].
- The many collisions of the ball during a game makes the ball behavior becomes so nonlinear, that the estimated state is frequently too far from the actual state.

During revision the objective of stabilizing the measurements was simplified by using the EKF only for team robots. To get a more accurate state of the opponents it is better to first improve the vision and subsequently to use an EKF. In the vision thirty percent of the opponent is known, by knowing the other colors used by the opponent the state derived from the image will be more accurate. Due to the nonlinear behavior of the ball the weighted average is taken (described in chapter 5), while taking collisions into account, instead of using the EKF.

2.2.1 Extended Kalman Filter

The state is measured at discrete times while the system dynamics are continuous, therefore the EKF is also referred as a Continuous-Discrete Extended Kalman Filter. An EKF estimates the state of the robot and then obtains feedback in the form of noisy measurements. Therefore the equations used in the EKF are divided into: time update equations and measurement update equations, see figure 2.2. The time update equations are responsible for predicting the state for the moment the next measurement will come in. After a measurement is made, the predicted state is compared with the measurement and a new estimate of the state is made based on prediction and measurement see [12]. The collision correction of S. Grond [13] is not implemented yet, but will certainly increase the stabilization.

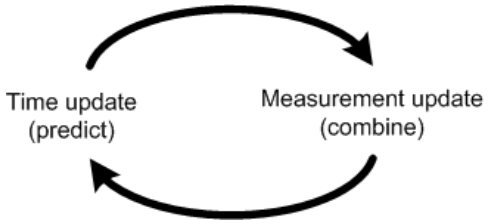


Figure 2.2: Phases in the Extended Kalman Filter

For simplicity the actual state of the robot at time k is changed from $q_k = (x, y, \theta, \dot{x}, \dot{y}, \omega)^T$ see [23] and [13] to $q_k = (x, y, \theta)^T$. The orientation in this state is an average of three independent orientations derived from the image in the vision module for each team robot. Two for each color segment and one based on the bisectrix of the two segments. This state q_k changes as a function of time and control input u_{k-1} and process noise ν. Therefore the following plant model is defined according to Gregory Dudek and Michael Jenkin [8].

$$q_k = \Phi(q_{k-1}, u_{k-1}) + \nu_k \tag{2.7}$$

In where the control input is a velocity vector given by eq.2.8 and is sent to the robot and to the EKF at time k.

$$u_k = (v_k, \omega_k)^T \tag{2.8}$$

The EKF simply linearizes the nonlinear plant model, which involves calculating the Jacobian [25] of the plant model $\nabla\Phi(q_{k-1}, u_{k-1})$ and using this as a linear estimate of $\Phi(\cdot)$. Each image frame from the vision a measurement z_k is taken, which is related to a system state q_k as given in eq.2.9 and describes how the frame data vary as a function of the system state.

$$z_k = h[q_k, k] + w_k \tag{2.9}$$

Where z_k is the k^{th} measurement and w_k the measurement noise function. Because w_k and ν_k are uncorrelated with each other, the noise is Gaussian with statistics given by

$$\begin{aligned} \nu_k &\sim \mathcal{N}(0, R_\nu) \\ w_k &\sim \mathcal{N}(0, R_w) \end{aligned} \tag{2.10}$$

Time update In between the measurements, the extended Kalman filter must adapt $\hat{q}_k$ as good as possible. Where $\hat{q}_k$ is the estimate of the robot state q_k at time k. The time update consists of a prediction of the state, using the plant model from eq.2.7, and control signals from eq.2.8, so the resulting propagation is given by eq.2.11.

$$\hat{q}_k^- = \Phi(\hat{q}_{k-1}, u_{k-1}) \tag{2.11}$$

Where the prediction of the robot state at time k is $\hat{q_k}^-$ and the prediction of the covariance P_k of the estimate is given by eq.2.12.

$$P_k^- = A_k P_{k-1} A_k^T + V_k R_{\nu_{k-1}} V_k^T \tag{2.12}$$

Where A and V are the Jacobian matrices of partial derivates of $\Phi(\cdot)$ with respect to the state q and to the process noise ν.

$$A_{i,j} = \frac{\partial \Phi_i}{\partial q_j}(\hat{q}_{k-1}, u_{k-1}) \qquad V_{i,j} = \frac{\partial \Phi_i}{\partial \nu_j}(\hat{q}_{k-1}, u_{k-1})$$

Measurement update The time update state estimate in the previous step is corrected based on the feedback from the measurement that has been taken. In eq.2.13 the Kalman gain K is calculated, which has to balance how much to trust the state update using P^- the estimate of P and how much to trust the measurement using R_w. The lower the measurement error relative to the process error, the larger the Kalman gain will be.

$$K_k = \frac{P_k^- H^T}{H P_k^- H^T + R_w} \tag{2.13}$$

Where H is the partial derivative of the measurement function h with respect to the state q:

$$H_{i,j} = \frac{\partial h_i}{\partial q_j}(\hat{q}_{k-1}^-)$$

Next, the state estimate is updated using the Kalman gain K from eq.2.13.

$$\hat{q}_k = \hat{q}_k^- + K_k[z_k - h(\hat{q}_k^-)] \tag{2.14}$$

Where $\hat{q_k}^-$ is the predicted state of actual state q_k at the time of the next measurement and $\hat{q_k}$ is the estimated state of actual state q_k. Now we can complete the estimation of the error in the new state, P, using the Kalman gain matrix K.

$$P_k = (I - K_k H_k) P_k^- \tag{2.15}$$

2.3 Motion planning

In a robot soccer game it is crucial to move to a specific position or posture, it is also an important area in robotics research [14]. A path generator is required, which has to find an optimal path from the start posture to the goal posture while avoiding collision to obstacles, and minimizing a path cost function while satisfying system constraints and performance requirements. The current system exists of two different approaches. First, a reactive motion control, originally developed by W.D.J. Dierssen[7]. Second, a deliberate motion control originally developed by T. Verschoor[26]. After introducing these approaches other methods are introduced which do more satisfy our objectives for motion planning.

Reactive approach The reactive motion control makes use of the Vector Field Histogram (VFH) method [5] by classifying all path directions into two categories: passable and not passable. It uses a predefined set of rules on the current state, and uses no knowledge about the past nor predicts future states. When using only the current state it is impossible to plan a task beforehand. Finally it will reach its end state, but it is unknown how and when it is achieved.

Deliberate approach The deliberate approach or proactive planning focuses on planning paths. The old deliberate approach makes use of S-curves to plan paths from initial pose to a goal pose. Later on we describe more details of the deliberate approach with S-curves. The path generation with S-curves is low cost in CPU load but is not flexible for preplanning and does not easily take avoidance of static or dynamic collisions into account.

Motion Planning objectives To contact the ball, we have to realize a time constrained motion. A deliberate approach is used because it is easier to handle time constraints with a deliberate approach than a reactive approach. The deliberate approach should require therefore the following properties.

- Flexibility in preplanning of curves at run time.
- Generating and updating curves should be cheap in CPU usage.
- A curve should be easily extended with a time dependency.
- The curve should be a smooth curve, since we have to control a nonholonomic mobile robot.

Therefore the following methods describe different curves properties with regard to flexibility.

Clothoid curve The clothoid curve is interesting because its curvature varies linearly with the curve length. This property results in a more accurate tracking when linearizing our nonlinear model. It is one of the simplest examples of a curve that can be constructed from its curvature, although the math is somewhat complicated. Another disadvantage of the clothoid is that it is difficult to determine the inverse kinematics, which makes the essential preplanning for ball-handling inflexible.

S-curves Another method to create curves is to use the old S-curves as already mentioned. It is possible to parameterize the curve with time t, which is proposed by J. van der Linden[20]. Planning with S-curves has some serious drawbacks. First, the S-curve is not flexible enough in re-planning with regard to moving targets. Secondly, obstacle avoidance is not an easy job with S-curves. An advantage of the S-curves is the smoothness, which is easy to track when using nonholonomic robots.

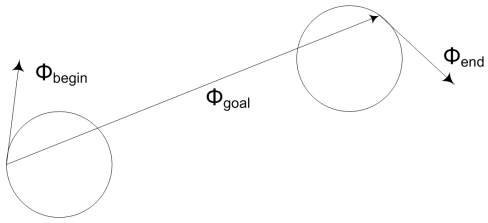


Figure 2.3: S-curve

Bézier spline curves Two of the most popular representations for curves are Bézier and B-spline curves. The Bézier spline curve[2] was formally presented in [3] and has since then been a very common way to display smooth curves, both in computer graphics and mathematics.

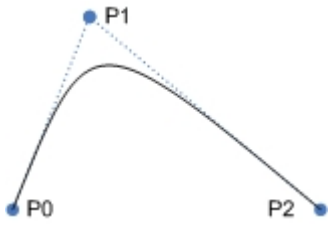


Figure 2.4: Example of a quadratic Bézier spline

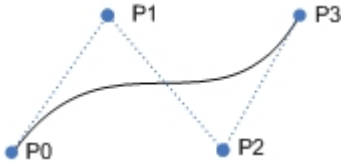


Figure 2.5: Example of a cubic Bézier spline

Bézier curves are formed mathematically from piecewise approximations of polynomial functions with zero, first and second order continuities. An example of a quadratic (2th order polynomial) and cubic (3th order polynomial) Bézier curve

[2]Shortened Bézier curve or Bézier spline

is given in figures 2.4 and 2.5. The points that define a spline are known as control points (e.g. in figure 2.4 these are P0,P1,P2). These points together form a polygon, which is called the Convex Hull. The polynomial functional representation of the Bézier curve it is standard parameterized with a value (often u) between 0 and 1. Shifting the value u in the range $[0, .., 1]$ is called interpolating the curve. Bézier curve properties are explained in more detail in chapter 3. They offer the following useful side effects:

- Bézier curves can virtually represent any desired shape, which gives a huge flexibility in creating curves to desired end poses.
- Detection of collisions is easily done by means of the convex hull property.
- It is easy to move a control point, which is useful in re-planning curves.
- A time or distance dependent variable can be added as parameter and mapped to the u value, which offers a time dependent curve(see [15]).
- Bézier curves are easily clamped inside a certain area (e.g. the playing field), by choosing the control points properly.
- The number of control points P_i is directly dictated by the degree n of the Bézier spline as $n + 1$.

Hermite spline curves The Hermite curve[2], named after Charles Hermite, specifies two endpoints and two other points to determine the tangents to these endpoints (see figure 2.6). A Hermite curve uses the tangent vector of the endpoint as a geometric constraint of the curve whereas Bézier uses another point relative the origin as constraint. A property of the Hermit curve is that it will pass through all of the control points. However, it uses a different set of basis functions compared with the Bézier spline and behaves differently. It also interpolates all of its control points, but does not have a convex hull property, which complicates collision detection. Another disadvantage is that is does not guarantee C_1 continuity (see chapter 3 for more details about continuity).

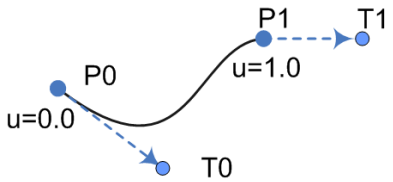


Figure 2.6: Example of a Hermite spline

B-Splines The B-spline curve is a generalization of the Bézier curve that consists of segments, each of which can be viewed as an individual Bézier curve with some additions (see figure2.7). One of these additions is that changes to a control point only affects the curve in that locality, so only a specified part of the B-Spline near that control point will be changed. This property is useful while avoiding obstacles. Another useful addition for avoiding obstacles is that adding multiple points (knot points) at or near a single position will draw the curve toward that position. And any number of points can be added without increasing the degree of the polynomial. However it also has disadvantages such as that is does not necessarily pass through any control point. And a B-spline curve can be quite expensive to calculate, especially in the non-uniform case. For more details about B-splines we refer to [2].

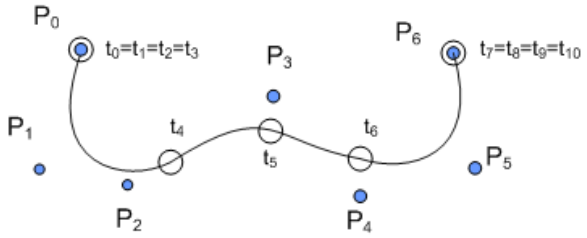


Figure 2.7: Example of a B-Spline with knot vector $T = \{t_0, t_1, ..., t_{10}\}$ and control points $P_0, ..., P_6$

Conclusion The type of curve to use as representation for a path between initial pose and goal pose is a matter of searching for the most suitable solution. By comparing the properties of the curves the parameterized Bézier curve is chosen mainly due to the following advantageously properties.

- Bézier curves always go through the first and last control point.
- A control point of a Bézier curve is easily changed, which facilitates replanning.
- Bézier curves contain a convex hull property, which facilitates collision detection.
- A time parameter is easily linked to a curve segment.

As a result of these useful properties the curves explained in chapter 3 will be of the Bézier type.

2.4 Motion control

From the control point of view, three different tasks are distinguished for mobile robots[1], see also figure2.8.

- *Point to point motion*: The robot must reach a desired goal configuration starting from a given initial configuration without planning a path in between.
- *Path following*: A reference point on the robot must follow a given path from initial state.
- *Trajectory following* : A reference point on the robot must follow a trajectory in the cartesian space (i.e., a geometric path with an associated timing law) starting from a given initial configuration.

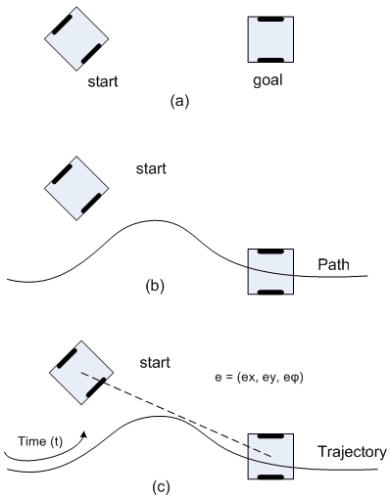


Figure 2.8: (a) Point to Point motion; (b) Path following; (c) Trajectory following;

The old motion control was based on path following by following the S-curves with PID control, pure-pursuit tracker and vector-pursuit tracker see [26] and [22]. In this thesis we will concentrate on a time-based motion control. The robot usually moves in the environment with obstacles, limitations and other demands which all somehow define a desired robot path. These are all in favor in using trajectory following control.

Smooth trajectories generated by the motion planning must be transformed into robot inputs. Robot inputs calculated from the trajectory are the feed-forward inputs. The basic control of a mobile robot in an obstacle-free environment must be solved to keep the robot on track. Therefore a feed-backward control is necessary to compensate initial errors and other disturbances during operation. We shall ignore second order kinematics, for example given in eq.2.6, because these are too difficult to handle. However controlling mobile robots considering only first order kinematics is very common in literature [19], [11] as well as in practice.

Chapter 3

Motion planning with splines

The objective of motion planning is to design a path in Cartesian space and to add a time parameter to it. Such a time-space path is used by motion control to move the robot along the path. The following describes the mathematics of the so called Bézier parametric curve. It is attributed and named after a French engineer[1]. The Bézier curve describes only the path in Cartesian space, therefore in the next chapter a velocity profile is added, which together forms a trajectory.

3.1 Bézier spline curve

In order to move from the current pose to the goal pose a path has to be planned. The most essential property of paths is that it leads to a specific end pose. Additionally one has to take into account some other desired properties such as generating paths with the lowest curvature, avoid areas and obstacles or updating paths with minimum CPU usage. Therefore the Bézier spline curve is chosen due to the following useful properties:

- Bézier splines consist of n control points with $n > 1$, which all influence the spline.
- A Bézier spline goes always through the first and last control point.
- The bounded polygon of the control points can be used for collision detection as the spline remains inside this polygon. Even a stronger property exist as shown in section 3.2.
- Smoothness is achieved by C^1 continuity.
- Bézier splines are easily joined together and offer local control in each segment.

[1] Pierre Bézier (1910-1999)

When generating spline curves, the nonlinear model of our robots has to be taken into account. The degree and thus the number of control points influence the controllability of the robot when tracking the spline. The objective is to plan an optimal smooth curve to the end pose. Planning paths is done with the following three curve types:

- 1th degree polynomial (Linear curve)
- 2th degree polynomial (Quadratic curves)
- 3th degree polynomial (Cubic curve)

By selecting these types enough flexibility in motion planning is created. By selecting a cubic curve (see figure 2.5) the end orientation is freely selectable, this is not if a quadratic curve is selected, see figure 2.4. If a linear curve is created as trajectory the robot has to plan a turn first in the desired orientation. The cubic curve is used to fit the initial as well as the end orientation. Building a Bézier spline requires the Bernstein polynomial evaluation to provide a so called basis-weighted function.

Basis-weighted function Each Bézier spline consists of control points P_i used to position and modify the spline. The first and last points define the so called endpoints. The cubic Bézier spline consists of four control points, P0 through P3. The two intermediate points P1 and P2 are used to specify the start and end tangent vectors, hence the curve passes through P0 and P3 while approximating the other two control points. Some kind of basis function is needed to tell how much weight each control point is given on the curve. The weighting function is defined by the Bernstein polynomials, also called basis functions:

$$B_{i,n}(u) = \binom{n}{i} u^i (1-u)^{n-i}, \qquad u \in [0..1] \tag{3.1}$$

As the robot moves along the trajectory, it is drawn toward control point i by an amount determined by $B_{i,n}(u)$, where i is the given control point and n the degree of the curve.

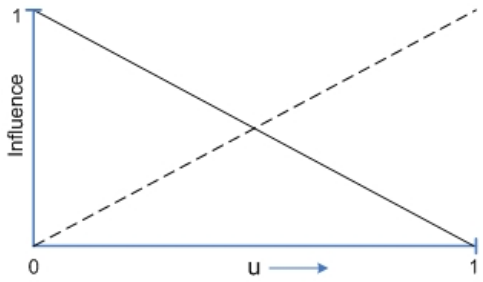


Figure 3.1: Basis function of linear Bézier splines

In the figures 3.1, 3.2 and 3.3 the basis function is given for the first, second and third degree Bernstein polynomials. These figures show two typical Bézier basis function properties, namely they all sum to unity and are all symmetric. By increasing u in figure 3.1 the linear curve is interpolated from control point P_0 to P_1.

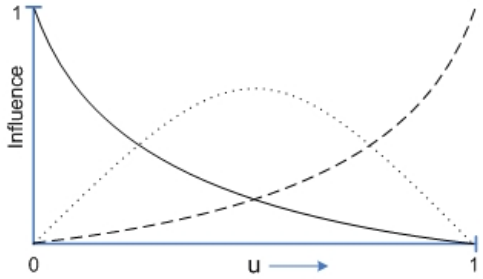


Figure 3.2: Basis function of quadratic Bézier splines

As the number of control points increases it is necessary to have higher order polynomials. Figure 3.2 shows a quadratic polynomial basis-weighted function and figure3.3 the cubic polynomial basis-weighted function.

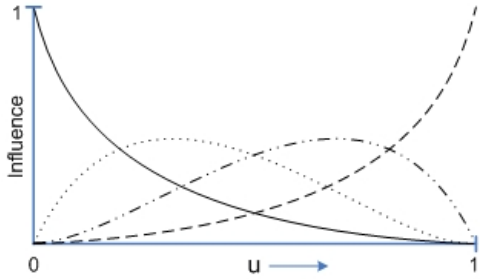


Figure 3.3: Basis function of cubic Bézier splines

Bézier Spline definition An n^{th} degree Bézier spline is defined by

$$C(u) = \sum_{i=0}^{n} P_i B_{i,n}(u) \tag{3.2}$$

Where $\{P_i\}$ forms a set of control points and $\{B_i\}$ are basis-weighted functions. Planning paths for the linearized model require rather straight curves. It is also known that each control point affect the curve. Thus if a flat curve is needed, like in our case, control points have to be chosen in a proper way or curves have to be made up of several Bézier splines. The length of an endpoint vector is defined as the distance between the end point and its neighbor point. Whenever a control point is planned out of the playing field it is simply clamped.

Computation The Bézier spline is very cheap to evaluate. This evaluation is done with an iterative scheme, which allow even a very small size of Δu without much computational load. There exists an iterative as well as an efficient recursive scheme. The step size do not need be uniform, but can be made variable depending on the local curvature.

Joining splines When two splines are joined, a first degree of continuity across the boundary is desired. If the n derivative of a function is continuous this is called C^n continuity we define

- Positional continuity C^0
- Tangential continuity C^1
- Curvature continuity C^2

Joining Bézier splines requires C^1 tangential continuity. As example two cubic curves are joined in figure 3.4. This implies positional continuity, which is achieved since both end positions of the curve are coincidental. If this is not the case, another curve has to connect the two curves or the endpoints have to move.

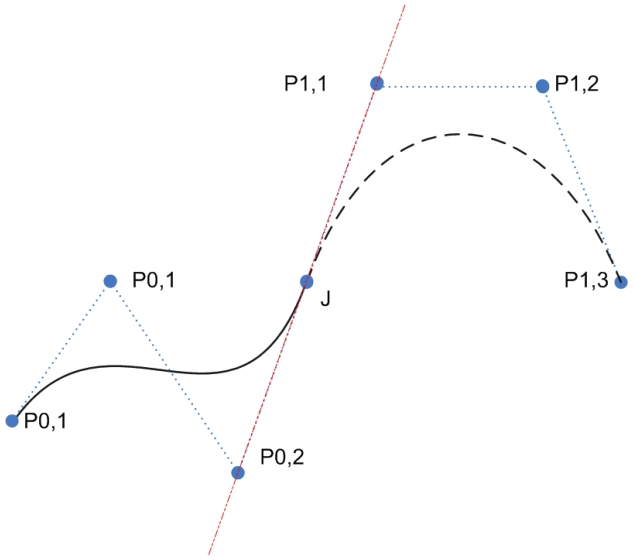


Figure 3.4: Joining two cubic curves

Positional continuity is not enough, the splines may still meet at an angle. Therefore we need also tangential continuity for a smooth curve, which require the end vectors of the curves to be parallel. When this is achieved it causes the curve to fall on a tangentially continuous edge, which makes this level of continuity sufficient for joining curves. Unless there has been chosen to drive with a certain velocity through the curve, curvature continuity is required and is achieved by sharing the same parametric second derivatives where they join.

The example in figure 3.4 interpolate their endpoints, so C^0 is achieved by the shared control point J. C^1 is achieved by setting $P_{0,3} = P_{1,0} = J$, and making $P_{0,2}$, J and $P_{1,1}$ collinear with $J - P_{0,2} = P_{1,1} - J$. Reaching C^2 continuity requires that control points $P_{0,1}$ and $P_{1,2}$ are chosen so that $\angle\{P_{0,1}, P_{0,2}, J\}$ equals $\angle\{J, P_{1,1}, P_{1,2}\}$ and the length of $\{P_{0,2}, J\}$ and $\{J, P_{1,1}\}$ are of the same magnitude.

Parametric Function mapping When using Bézier splines as trajectories, the travel distance property parameter of our robots have to match with the curve parameter u. Therefore we use the property of mapping the parameter $u \in [0..1]$ to the curve length or travel time t. With this a Bézier spline can be parameterized with time t or distance s. In our 2D Cartesian plane the curve is a function in the X-axis and Y-axis and is defined by

$$C(u) = \{C_x(u), C_y(u)\} \tag{3.3}$$

Where $C_x(u)$ and $C_y(u)$ are functions of the parameter u (hence parametric). Given a value of u, the function $C_x(u)$ gives the corresponding value of x, and $C_y(u)$ the value of y. We parameterize the curve with the arc length s described in the next paragraph. With this arc length s and parameter u a table is made to match each Δu with the traveled distance reached so far.

For example, a curve with total length $s_{tot} = 500mm$, matches the length parameter s=0 with the curve parameter u=0 and the length *s=500* with u=1. So when we have to know the robot position (x(u),y(u)) the matching u must be found in a table, therefore the traveled distance so far has to be known. This is known by the velocity profile after which we have to select the best matching s stored in the table in order to know the value of u. Searching is done by using a binary search because this uses at most O(log n) time.

Arc length A necessary value to know is the length of the Bézier spline when using these Bézier splines as trajectories. This length is needed for the velocity profile to determine the travel time at a given velocity. This velocity profile is described in Chapter 4. The arc length of a Bézier spline, see [25] is calculated by

$$L = \int_0^1 \sqrt{\dot{C}_x(u)^2 + \dot{C}_y(u)^2}\, du \tag{3.4}$$

Curvature The curvature can be regarded as a measure of the rate of change of tangent direction θ of the curve with respect to the curve length parameter s, see figure 3.5. This curvature is defined by

$$\kappa = \left|\frac{d\theta}{ds}\right| \tag{3.5}$$

Where θ is the angle between the tangent line at P_1 and the tangent line at P_2 and s is the arc length parameter. Thus, the curvature is the absolute value of the rate

of change of θ with respect to arc length. For our parametric Bézier curve $x = x(t)$ and $y = y(t)$ this becomes

$$\kappa = \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{[\dot{x}^2 + \dot{y}^2]^{\frac{3}{2}}} \tag{3.6}$$

Where the dots indicates derivatives with respect to time t, so $\dot{x} = \frac{dx}{dt}$.
Use $\theta = arctan(\frac{dy}{dx})$ to find $\frac{d\theta}{dt}$ and than use the chain rule to find $\frac{d\theta}{ds}$. This prove is given in appendix A.

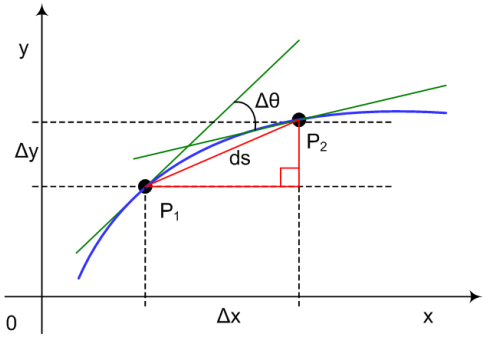


Figure 3.5: Curvature κ

3.2 Collision detection and avoidance

Collisions in a robot soccer environment are part of the game. In most cases they are not productive and can better be avoided. Therefore collision detection and avoidance must be applied in several layers of the MI20 software structure. The planning of curves is one part of that.

Collision detection Detecting collisions while planning curves is subdivided into

- Collisions with walls
- Collisions with team robots
- Collisions with opponent robots

A useful property of Bézier curves for collision detection is that the curve stays always within the Bézier polygon. This polygon is formed by the convex hull of all control points see figure3.6.

Collision detection between a robot and the border of the playing field is done by checking if all control points are chosen in the playing field. Collisions with other robots included team robots are detected by checking if the robots are within convex hull. Polygons take still a large area if control points are chosen far away from the end points. Therefore the convex hull becomes large when a path is

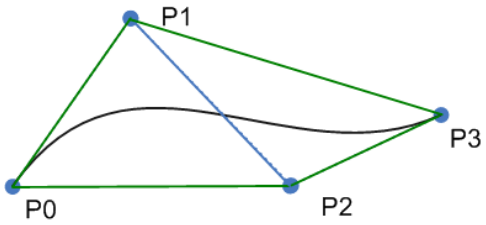


Figure 3.6: Convex hull property

planned to the other side of the field. A disadvantage of a large convex hull is that it detects more collisions then a small one. That is why the polygons are reduced to triangles for each curve. For example, if we split up the cubic curve $P_0, ..., P_3$ in 2 triangles we know that the curve stays within the two triangles P_0, P_1, P_2 and P_1, P_2, P_3. This covering area is mathematically represented with

$$\bigcup_{i=1}^{n-1} \Delta(P_{i-1}, P_i, P_{i+1})$$

This can be extended by adding a time property to each triangle for locking a certain interval.

Collision avoidance The collisions with the borders can easily be solved by clamping the control points within the playing field. Here the size of the robot must also be taken into account. With planning clever trajectories, collisions with team robots should almost be absent. If in spite of clever planning a collision occurs, then some kind of dynamic collision avoidance is needed. For collision avoidance in the previous version of the MI20 software we refer to J. van der Linden[20]. Avoiding collisions with opponent robots and team robots is a huge subject on itself, therefore it is not an integral part of this thesis.

Chapter 4

Motion control of a mobile robot

The control module has to keep the robot on track as good as possible by sending appropriate wheel speeds. Therefore a time-space path in the form of a parameterized smooth Bézier curve is followed, see chapter 3. A velocity profile is used as time constraint during this operation. This velocity profile together with the time-space curve is called a trajectory. The control signals derived from such a trajectory are used as open-loop control. Due to imperfect controllable dynamics, open-loop control signals must be corrected with measurements, therefore a closed loop controller is designed to correct the robot given the filtered and predicted measurements.

4.1 Trajectory tracking

Trajectory tracking is keeping the robot on the planned curve with regard to the time constraints from the velocity profile. Steering the robot across a trajectory is done by varying the left and right wheel speeds of the robot. These wheel speeds must be constructed by the motion control module and are the inputs for the robots. Due to the imperfect controllable dynamics (e.g. slippery wheels) we have to know how much we have to steer in which direction for correcting the orientation error and how much we have to accelerate or decelerate for reaching the planned pose $(x, y, \theta)^T$ at time t. These errors must be expressed into inputs for the robot, therefore a reference trajectory and reference robot are defined.

- A reference trajectory indicates the theoretical and thus ideal trajectory.
- The reference robot is the place the robot has to be according to our reference trajectory.

Each frame, filtered and predicted measurements lead to a state update by the state estimator. To avoid confusion, reference robots are indicated with a desired state and the measured robots are indicated with an actual state[1]. The objective of

[1] We shall use q_d to indicate the desired state and q_a for actual state

time-based motion control is to solve the error at time t between the desired state and the actual state of the robot as shown in figure 4.1.

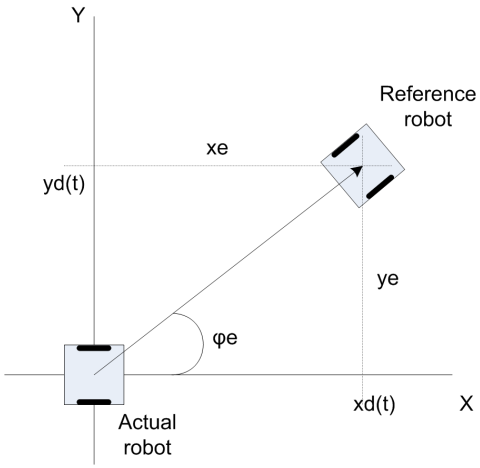


Figure 4.1: Robot frame error

There is only one drawback to solve first. Controlling the actual robot by setting the wheel speeds is defined relative to the robots own coordinate frame, whereas the measurements of the real robot and the planned trajectory are defined relative to the world frame, shown in figure4.2.

To solve this, the pose error relative to the world frame is transformed into a pose error relative to the robot frame. Therefore we use a so called transformation matrix expressed in eq.4.1.

$$T(\theta) = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

The error state $\tilde{q} = (x_e, y_e, \theta_e)^T$ relative to the robot frame given in eq.4.2 has to be determined in order to generate appropriate wheel speeds. The difference between the state of the actual robot $q_a = (x_a, y_a, \theta_a)^T$ and the state of the reference robot $q_d = (x_d, y_d, \theta_d)^T$ represent this error. The error state is transformed to the robot frame eq.4.2 by means of the transformation in eq.4.1.

$$\begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} = T(\theta). \begin{bmatrix} x_a - x_d \\ y_a - y_d \\ \theta_a - \theta_d \end{bmatrix} \tag{4.2}$$

With this error a controller is designed described by G. Klančar and D. Matko [9]. This controller is divided into feed-forward control and feed-backward control.

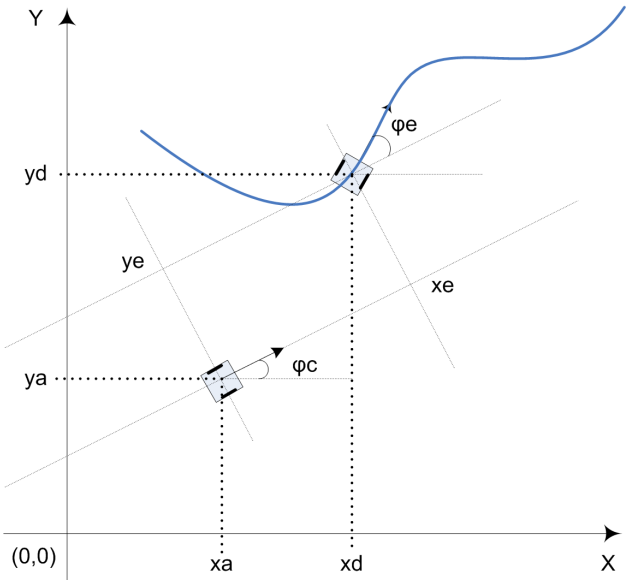


Figure 4.2: World frame error

4.2 Feed-forward control design

Controlling the robot with feed-forward or open-loop denotes that it has to determine appropriate wheel speeds from the reference trajectory. Recall eq.2.4 where the wheel speeds are related to the linear velocity v and angular velocity ω.

$$\begin{aligned} v_r &= v + \frac{\omega L}{2} \\ v_l &= v - \frac{\omega L}{2} \end{aligned} \tag{4.3}$$

The reference velocity vector (v_d, ω_d) is solvable by solving the ω_d from the curvature and determining the v_d from the velocity profile. The velocity profile determine the desired linear velocity at time t and is described in section 4.2.1. The desired angular velocity at time t is derived by the curvature κ at time t and the desired linear velocity v_d at that time. Therefore recall the curvature equation

$$\kappa = \frac{|\dot{x}(t))\ddot{y}(t) - \ddot{x}(t)\dot{y}(t)|}{[\dot{x}(t)^2 + \dot{y}(t)^2]^{\frac{3}{2}}} \tag{4.4}$$

Then the angular velocity is derived by

$$\omega_d(t) = \kappa.v_d(t) \tag{4.5}$$

Before obtaining the desired feed-forward linear velocity from the velocity profile we have to restrict the desired linear velocity to our nonholonomic constraint, see eq.2.2. For example, when the measured orientation error e_3 is $\pm\frac{1}{2}\pi$ than the

desired velocity is transversal to the orientation, which shall only enlarge the error. As a consequence of our nonholonomic constraint this will result in:

$$\begin{aligned} u_{d1} &= v_d.cos(e_3) \\ u_{d2} &= \omega_d \end{aligned} \tag{4.6}$$

Where u_d is the desired feed-forward velocity vector.

4.2.1 Velocity profile

A velocity profile is necessary for motion control to specify the desired reachable velocity at a time t. Both robot types have their own specific profile parameters for the velocity profile given in table 4.1. The MiaBot acceleration and deceleration parameters are adaptable by changing the registers on board[21]. The theoretical properties are given by table 4.1.

	GermanBot	Miabot
Acceleration	$1.8m/s^2$	$2.0m/s^2$
Deceleration	$-1.8m/s^2$	$-2.0m/s^2$
Top speed	$2.0m/s$	$3.5m/s$

Table 4.1: Robot properties

The velocity profile is used by motion control, which will use time-stamps from the world data to know the feed-forward velocity at that time-stamp. To calculate the velocity profile we just use the physical laws for the velocity and distance.

$$\begin{aligned} v_d(t) &= \alpha t + v(t_0) \\ s(t) &= \tfrac{1}{2}\alpha t^2 + v_d(t_0).t + s(t_0) \end{aligned} \tag{4.7}$$

With these equations we are able to calculate the total time needed for traveling the curve. Therefore the velocity profile is divided into three parts, the acceleration part, a constant velocity part and a deceleration part see also an example in figure 4.3.

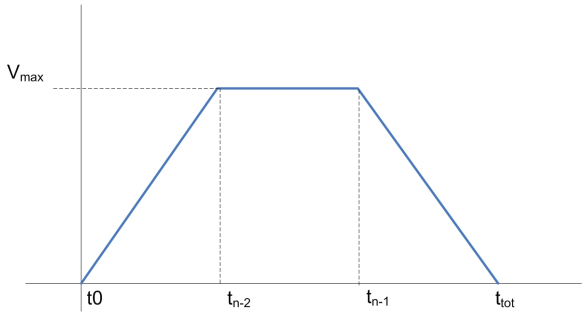


Figure 4.3: Example of a velocity profile

The desired velocity can be given by the strategy module and is limited by the top speed of the specified robot type. Since the distance of the curve the robot has to travel is already known we are able to calculate the total time needed for accelerating and decelerating. When the desired speed can not be reached due to a too short trajectory it will just as much accelerate as decelerate. Otherwise the distance left is used to travel the curve with a constant velocity.

The strategy is able to set the velocity at the end of the trajectory, which is useful when the robot has to kick the ball. Another adaptable parameter is the begin velocity. When a trajectory consist of several spline segments it is desirable to match the end velocity and begin velocity for the joined trajectory. When a new move is planned in the same direction the robot was already moving, than the begin velocity is calculated with the Pythagoras sum (see eq.4.8). How the robot is driving through the curve, forward or backward, is dedicated by the sign.

$$v_a(t) = \pm\sqrt{\dot{x_a}^2(t) + \dot{y_a}^2(t)} \tag{4.8}$$

Limiting curvature velocity When using a velocity profile it has to deal with the curvature of the trajectory the robot has to follow. Each team robot has curvature limitations due to its dynamics, therefore a maximum velocity should be used at which the robot is controllable. To determine this maximum curvature velocity we use the radius. The radius is definite by:

$$R = \frac{1}{||\kappa||} \tag{4.9}$$

If $\kappa \to 0$ then the curve becomes a linear curve or also called straight line. With a linear curve the highest reachable velocity is feasible due to the linearity, if the curve length is long enough. When the curve is sharp, the κ in eq.4.9 is huge, then the radius will become small, which limits the maximum velocity.

Each time a velocity is derived from the profile it should be restricted to the maximum allowable velocity for that curvature. Therefore a table should be made on basis of the radius (see figure 4.4), since the maximum allowable radius is easier to obtain by differentiating left and right wheels.

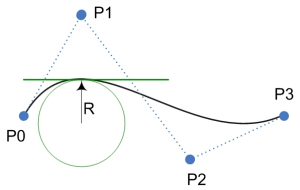


Figure 4.4: Curvature of a Bézier spline curve

4.3 Feed-backward control design

To minimize the errors between the actual robot compared to the reference robot a control algorithm is designed to correct the robot to follow the reference trajectory precisely. These initial errors, effects of noise and disturbances are corrected with a feed-backward control also called closed loop control. The feed-forward together with the feed-backward is our control system. The measured error state $\tilde{q}$ is used as input for our control system and the robot velocity vector (v, ω) is used as control input u. Our control system is defined by:

$$\dot{\tilde{q}} = A.\tilde{q} + B.u \tag{4.10}$$

The trajectory tracking is transferred into a conventional control problem, which has to determine the robots control signals u_1 and u_2. Such that the feed-backward system is asymptotically stable; that is, the error configuration $\tilde{q}$ tends to its equilibrium 0 eventually. Therefore the robot velocity vector input u and accordingly the output of the control system is obtained by calculating the difference between the feed-forward velocity vector u_d and the feed-backward velocity vector v.

$$\begin{aligned} u_1 &= u_{d1} - v_1 \\ u_2 &= u_{d2} - v_2 \end{aligned} \tag{4.11}$$

The schematic of the obtained controller with the feed-forward control and feed-backward control is shown in figure 4.5.

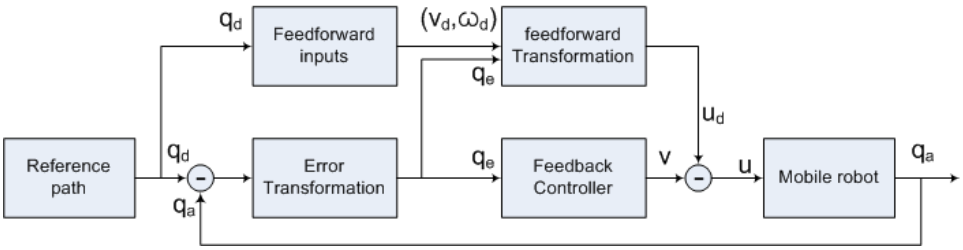


Figure 4.5: Feedbackward and feedforward control scheme

In our control scheme, the feed-backward controller cancels the effects of noise, disturbances and initial state errors, while the feed-forward control uses the robot inverse kinematics to calculate feed-forward inputs. From control point of view the robot must be controllable, this prove is given in appendix B. Accordingly the feed-backward controller has to solve the 2-tuple output vector v with a 3-tuple input error state $\tilde{q}$, see the feed-backward control block in figure 4.5. Therefore a 3x2 kalman controllability matrix K is specified as.

$$v = K.\tilde{q} \tag{4.12}$$

When we unfold this equation this becomes:

$$\begin{bmatrix} v1 \\ v2 \end{bmatrix} = \begin{bmatrix} k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,1} & k_{2,2} & k_{2,3} \end{bmatrix} . \begin{bmatrix} x_e \\ y_e \\ \theta_e \end{bmatrix} \tag{4.13}$$

The difficulty of controlling a drift-less nonholonomic mobile robot is the restriction of the degree of freedom which gives nonlinear behavior, see[1]. In other words, due to the fact that we have only two control signals, while we have three errors to solve, force us to share at least one control signal. Therefore we define the following linear feedback law:

$$\begin{aligned} v_1 &= -k_{1,1}.x_e \\ v_2 &= -k_{2,1}.sign(v_d(t)).y_e - k_{2,3}.\theta_e \end{aligned} \tag{4.14}$$

The error in the x-as of the robot frame is solved by the linear velocity with a definite Kalman value. While the errors in the y-as and orientation of the robot frame are solved by the angular velocity with definite Kalman values. This feedback law is also an intuitive one. When the robot has an error in the x-as of the robot frame, it was too slow or too fast in the past. By controlling the linear velocity we are able to decrease this error. When the robot has to move more to the left or to the right or has to solve an orientation error it has to control the angular velocity. As a result of this the Kalman controllability matrix is given by

$$\begin{bmatrix} v1 \\ v2 \end{bmatrix} = \begin{bmatrix} -k_{1,1} & 0 & 0 \\ 0 & -sign(v_d(t)).k_{2,2} & -k_{2,3} \end{bmatrix} . \begin{bmatrix} e1 \\ e2 \\ e3 \end{bmatrix} \tag{4.15}$$

Where $sign(v_d(t))$ is the robot drive direction (forward or backward), because if we drive backwards than we have to steer in the opposite direction.

The next question is how to determine the suitable controller gains. From control point of view the following closed-loop characteristics are used, also see [10] and [1].

$$(\lambda + 2\zeta a)(\lambda^2 + 2\zeta a\lambda + a^2), \qquad \zeta, a > 0 \tag{4.16}$$

Having constant eigenvalues (one negative real at $-2\zeta a$ and a complex pair with natural angular frequency a > 0 and damping coefficient $\zeta \in (0, 1)$) can be obtained by choosing the gains in eq.4.15 as

$$k1 = k3 = 2\zeta a, \qquad k2 = \frac{a^2 - w_d(t)^2}{\|v_d(t)\|} \tag{4.17}$$

However, k2 will go to infinity as $v_d \to 0$ therefore, a convenient gain scheduling is achieved by letting a = a(t) = $\sqrt{\omega_r^2(t) + bv_d^2(t)}$ where the factor $b > 0$ has been introduced as an additional degree of freedom.

Chapter 5

Development of ball handling algorithms

With the use of Bézier splines as adaptable trajectories, we are capable to create better ball-handling algorithms due to the useful properties of splines. The ball-handling algorithms will focus on the shoot and catch algorithm. In both algorithms the trajectory controller will be used.

5.1 Shoot skill algorithm

Shooting a ball in a desired orientation is essential in a robot soccer game. The existing shoot algorithm works satisfying with a non moving ball, see T. Verschoor [26]. Our improvements of the shoot skill will aim on a moving ball. To shoot such a ball it has to be intercepted by the robot, the point where that happens is called the interception point. Intercepting the ball requires an accurate ball prediction and adaptable trajectory with time constraints.

Ball prediction The old ball prediction is improved by a more stable ball tracking method, a better ball prediction and by adding a ball collision model.

First, the ball position is not filtered by the EKF as already mentioned. Due to the many collisions the ball becomes nonlinear in behavior and therefore the estimated state is frequently too far from the actual state. To get stable and accurate ball data a history queue is added to track the ball the last four measurements. Extracted from these measurements the weighted average of the positions is taken which makes the ball prediction more accurate. Also, with such a history queue, a more stable orientation of the velocity vector is achieved.

Secondly, as the velocity vector is made more accurate, the prediction of a certain time ahead will be more accurate too. At the moment there is no static friction taken into account, because this is depending on the playing field and it does not improve the prediction significantly. Since the used golf ball contains dimples it is still difficult to predict the ball when its velocity is very low. Therefore

the prediction of the ball position is quite uncertain when the velocity of the ball is low.

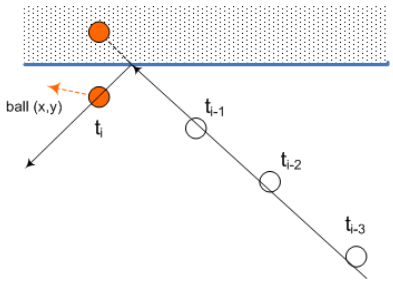


Figure 5.1: Ball-wall collision with noisy ball position measurements

And third, when predicting the ball, collisions with the wall and the robots are considered. A velocity damping factor is used to reduce the size of the velocity vector if a collision occurs, this improves the accuracy of the prediction. A collision with a wall is detected by the fact that the predicted position one frame ahead is outside the playing field, see figure 5.1. The velocity vector has a certain orientation. When a collision occurs with the wall this orientation is changed by a mirrored ball position. Because the orientation of the velocity vector is the average over the last four measurements, it actually takes four measurements to get the proper orientation after a collision. It is recommended to change the orientation after a collision immediately, for example the velocity vector can be mirrored. When the ball is predicted a certain time ahead, collisions with robots are detected by using the position of the specified robot in the current snapshot. In other words, it is assumed that robots are non moving objects when predicting the ball.

The ball-wall collision model in the old MI20 system is also improved by taking the corners of the playing field into account. As a result of this collision model we are able to predict the ball at any time, but it may be obvious that predicting a small time ahead is more accurate than predicting a large time ahead.

5.1.1 Calculate interception point

For shooting a ball in a specific direction a trajectory has to be planned toward the interception point. This point is the first point in time the robot is able to hit the ball as shown in figure 5.2. The shoot direction in the end pose of the trajectory is determined by the target point to shoot at. To simplify the shoot problem we solve this problem in an obstacle free environment. When obstacles do exist, a collision free trajectory has to be planned, which is still not possible.

The approach of calculating the interception point with a Bézier spline is to search for the first moment in time T in which the robot is feasible to hit the ball. Therefore the robot must be feasible to travel within time T, to the position the ball is predicted at time T. The position of the ball predicted at time T is called the interception point, only if the robot is feasible to travel to that point in time

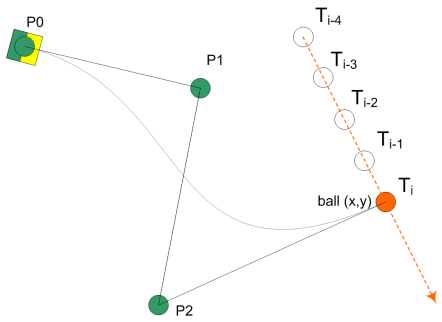


Figure 5.2: Kicking a ball in a specific orientation

T. For searching an appropriate time T a bisection search is used (see further on). To check whether it is feasible to travel within time T to the interception point the length of the spline has to be known. This length is used in the velocity profile to calculate the time, the robot needs for traveling.

Interception point update Once an interception point is calculated and a shoot action is initiated, the interception point has to be kept up to date. Due to ball dynamics and imperfect controllable robot dynamics the interception point is a dynamic point. Therefore the interception point has to be recalculated each frame, but this is an expensive operation of ±3 ms.

The idea is to use the already known interception time T. This time T is equivalent to the time totally needed for traveling the curve, thus the time left is also known. This time left is used to predict the position of the ball and this position is used to update the spline, see figure 5.3. Because the spline changes, the interception time changes too, which introduces disturbances. These disturbances are solved by the time-based controller, which is capable to raise or lower the velocity and to solve trajectory errors. When the spline is changed, the length is changed too and as a result of that we have to re-parametrize the spline.

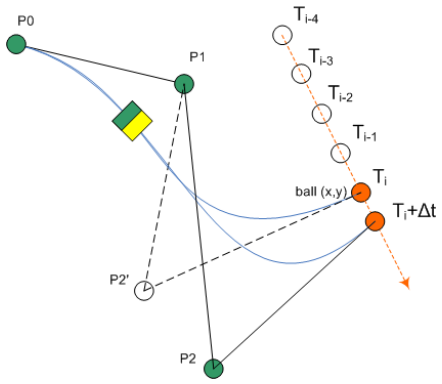


Figure 5.3: Interception point update

Problems arise in a game situation when the ball is taken away by the opponent. In such case the strategy has to decide what to do next. Adjusting the trajectory is a problem if the robot has almost finished the trajectory. After the robot has passed the spline for 75% the robot will meet difficulties during tracking the trajectory if it is changed. Therefore the interception point is updated up to $u < 0.75$.

Bisection algorithm To search for the interception time T a bisection search is used based on R.W Beard[4]. The idea is to use a function g which calculates the length of the trajectory from the start pose to the ball pose at time T. Function g calculates also the distance the robot can travel in time T according to the given velocity profile. The ball pose is a predicted ball position at time T and the orientation is the orientation we want to shoot at. The direction to shoot at is a variable determined by the strategy.

Before the search starts, a time range to search in has to be defined see figure 5.4. The maximum time is found by using the g function and increment time T with large time steps (seconds in our case) until the interception point is feasible within that time T. The initial time T_0 is still set on 0. It is recommended finding an optimal minimum for this, but this is not trivial for a ball that can bounce.

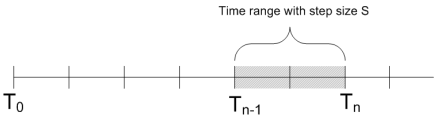


Figure 5.4: Time range in which the interception point is feasible

Because two distances in the g function are compared, the search has to be stopped if the difference between these two distances is smaller than ±5mm. This is the so called offset ϵ. The bisection algorithm has the advantage of quickly converge to $\|g(T)\| < \epsilon$

Bisection algorithm code

1. Set $T = T_0$.
2. Increment T with step size S until $g(T) > 0$.
   Set $T_2 = T$ and $T_1 = T - S$.
3. While $\left\|g(\frac{T_1+T_2}{2})\right\| > \epsilon$

   if $g(\frac{T1+T2}{2}) > 0$ set $T2 = \frac{T1+T2}{2}$,

   else set $T1 = \frac{T1+T2}{2}$
4. Set $T = \frac{T1+T2}{2}$
5. With $g(t) = V_{max}t - \|pathLength(\quad path(InitialPose, ballPose(t))\quad)\|$

Shoot types For shooting a ball we define three shoot types:

- Kick
- Dribble
- Panic Kick

The objective of a kick is to shoot in a desired orientation with a desired velocity. A kick is constructed from a cubic Bézier spline which is planned to the interception point and after hitting the ball the kick action just stops.

When it is desired to dribble with the ball to a specified point a dribble must be selected, see figure 5.5. The first phase of the dribble equals the kick. The second phase of the dribble is just a spline joined to the spline from the first phase. The idea is to plan this second spline to the goal of the opponent. The second phase starts after the ball is hit. While being in the second phase we continuously have to check if the ball is in front of the robot. This check is required because our robots do not have a mechanism to hold the ball and due to the dynamic behavior of colliding with hard objects it is possible to lose the ball. Taking the uncontrollable movements of the ball in front of the robot into account, the best approach is to accelerate in the second phase. Therefore it is useful to plan a linear trajectory as second phase, because with a linear trajectory the robot is able to reach higher velocities then with trajectories containing curvatures. When losing the ball during a dribble, the robot loses the dribble functionality and as a result of that it stops.

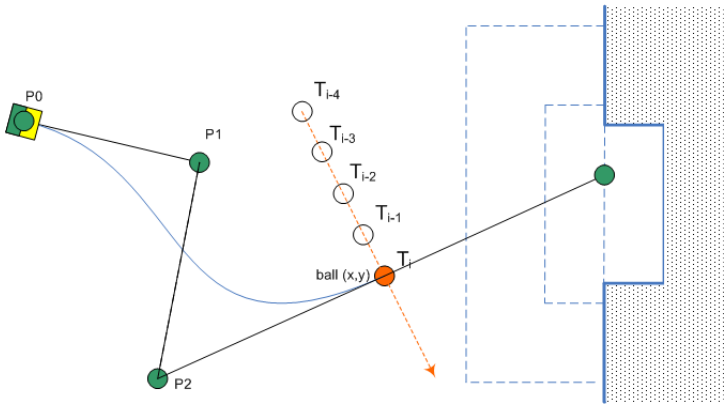


Figure 5.5: Dribble shoot

The objective of a panic kick is to shoot the ball as quick as possible in any direction except of the own goal. Before a panic shoot is executed the motion planning detects if it will shoot on the own goal, if so it just refuses to execute. For planning a panic shoot a turn and a linear spline is used, see figure 5.6. When calculating the interception point, the time to turn as well as the time to traverse the linear curve has to be considered.

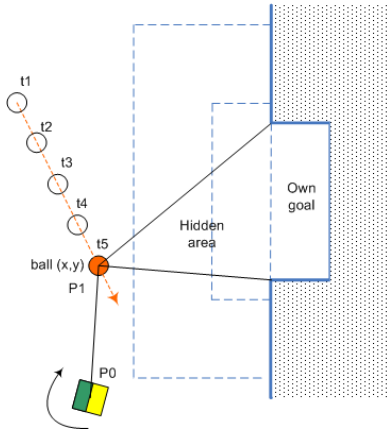


Figure 5.6: Panic shoot

5.2 Defend skill algorithm

The goal keeper algorithm made by W.D.J Dierssen[7] is based on a reactive motion with a single goalkeeper. Due to revising of the software structure for the MI20 system the reactive goalkeeper is not implemented yet. Instead of this we will investigate how well the feed-backward control from the motion control module alone will perform as reactive approach.

The catch skill is extended with a two robot defend system in which two objectives are postulated. First, keeping the ball out of our goal and secondly, avoiding to be penalized. The FIRA formulated rules about penalty kicks. We are penalized when

- Defending with more than one robot in the goal area.
- The kick out by the goalkeeper robot, of the ball from its goal area, is not within ten seconds.
- Three robots of one team stay inside their own penalty area.

Motion planning To comply with these rules two robots are assigned to the penalty area from which one is assigned to the goal area. The robot in the goal area is called the goal keeper and the other is called first defender. The idea is to test how well the controller functions when a reactive approach is used. Therefore we define a straight line for both robots, one in the penalty area and one in the goal area. Each frame a point on the line is selected without considering the points selected in the past. We keep both catch types simple by defining the following cases in which we react differently.

1. The ball is rolling toward our goal.
2. The ball is not in the penalty area and is not rolling toward our goal.

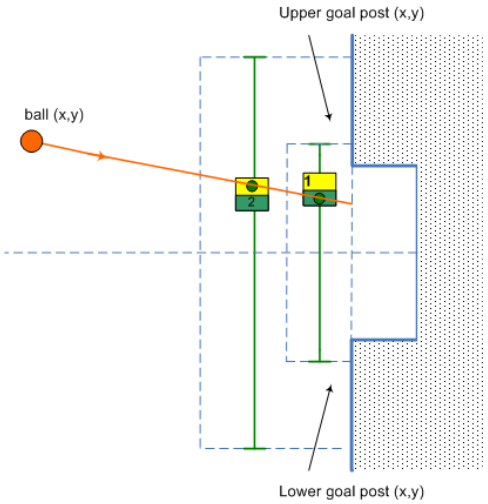


Figure 5.7: Defend system as test case with a ball rolling toward our goal

In the first case when the ball moves toward our goal, we chose the cross point of the defend line and the line of the ball orientation as destination point. To maximize goal coverage, both robot sizes are used in accordance to figure 5.7.

When the ball is not in the penalty area and is not rolling in the direction of our goal, the best we can do is to select a point which covers the goal the most. By defining lines from the ball to both goal posts and at the middle of the goal line we get two separate triangle areas. The chosen destination points on the defend lines are the points in the middle of the triangle, see figure 5.8.

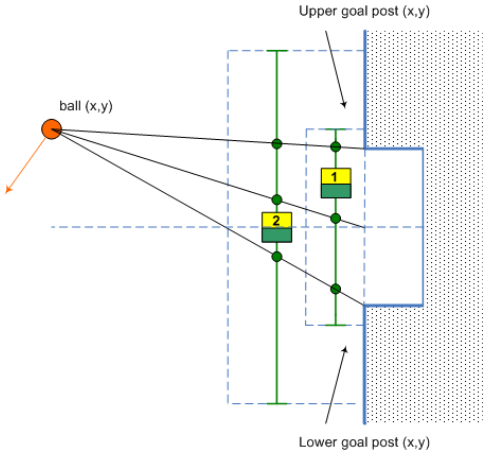


Figure 5.8: Defend system as test case with a ball rolling in another direction then our goal

In the case the ball passed the first defender, a carombole effect is possible. Therefore we emphasize that this ad hoc defend system is pure for testing motion control for the catch skill. Another approach is to plan a trajectory each frame, but this requires more computation time. Or we can plan a linear spline as trajectory only once, but this requires a more complex velocity profile.

Motion control The idea is to investigate the performance of the trajectory controller as defend skill when the feed-forward control is neglected. Therefore the motion planning only has to specify a destination state each frame for each defending robot. The feed-backward control tries to solve the error between the initial state and the destination state. The objective after determining a state is to move as quickly as possible toward that new point. Therefore the controller with only feed-backward has to be tuned differently.

Chapter 6

Testing and evaluation

This chapter describes how well the new motion planning and control performs with regard to ball-handling skills compared to the reactive approach of [7] and proactive approach from [26]. The accuracy of measurements and ball-handling skills are tested and judged qualitatively. Evaluating the accuracy of trajectory tracking is judged quantitatively.

6.1 Data filtering accuracy

The idea was to improve world data by making it more stable and accurate. The reason to revise the EKF in the old MI20 system was caused by the state estimator, which did not function properly. The failures were discovered by testing the following:

- When a team robot color patch was moved by hand without sending appropriate wheel speeds, the EKF did not respond.
- When the control signals where sent while the robot did not respond, the EKF generated world data that was completely based on the control signals.

It resulted in neglecting the measurements in the extended Kalman filter (EKF). The old EKF has been revised and is tuned so that it trusts the measurements from the snapshots a lot more. The test shows that the revised EKF now trust the measurements too much. This resulted in unsatisfying accuracy on high speeds. For example, when our robot is traveling with 1.6 m/s and our camera measured 30 snapshots each second a maximum deviation of 53 mm can exist. Although the EKF filters data well on low speeds, it has to be tuned better to find a better balance between measurements and control inputs.

To prevent a huge CPU load, like in the old MI20 system, each algorithm is tested on the amount of CPU usage. The computation time needed for a complete Kalman cycle is measured and takes 42μs.

6.2 Trajectory tracking accuracy

Using Bézier splines with a time profile as trajectories gives much flexibility. The curve is, by means of the control points, easily clamped inside a certain area (i.e. the playing field). Clamping the trajectory within the playing field prevent planning trajectories through the wall, what was frequently done with S-curves in the old MI20 system. Other advantages of building and parameterize a Bézier curve are the costs of $\pm 60 \mu s$ CPU usage and that splines are numerically easy to differentiate and to integrate. All together the Bézier spline works quite well and is very useful as a trajectory building tool.

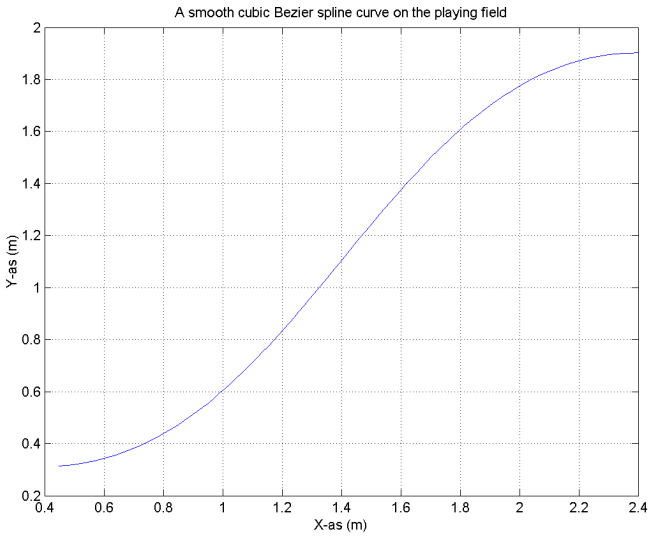


Figure 6.1: An example of a path on the playing field

A recommendation when building a Bézier spline is to optimize the spline by selecting the maximum velocity feasible for the curvature. For example, one may vary the length of an end point vector. The current spline builder uses 30% of the Euclidean distance between the initial pose and the end pose for the length of an end point vector. Because the direction of the used robot doesn‘t matter, the current spline builder builds the spline with the minimum κ value. Other opportunities are building a spline in forward or backward direction of the robot or building the shortest spline.

It is frequently desired by the commanding strategy to go as quickly as possible to the end pose. Therefore a maximum velocity has to be chosen for each curvature value. This is depending on the robot type, because each robot type has other dynamics. Only the MiaBot was used for testing the trajectory tracking. The communication to the GermanBots frequently falter and some GermanBots uses 80% of the sent control inputs, which makes them more difficult to control.

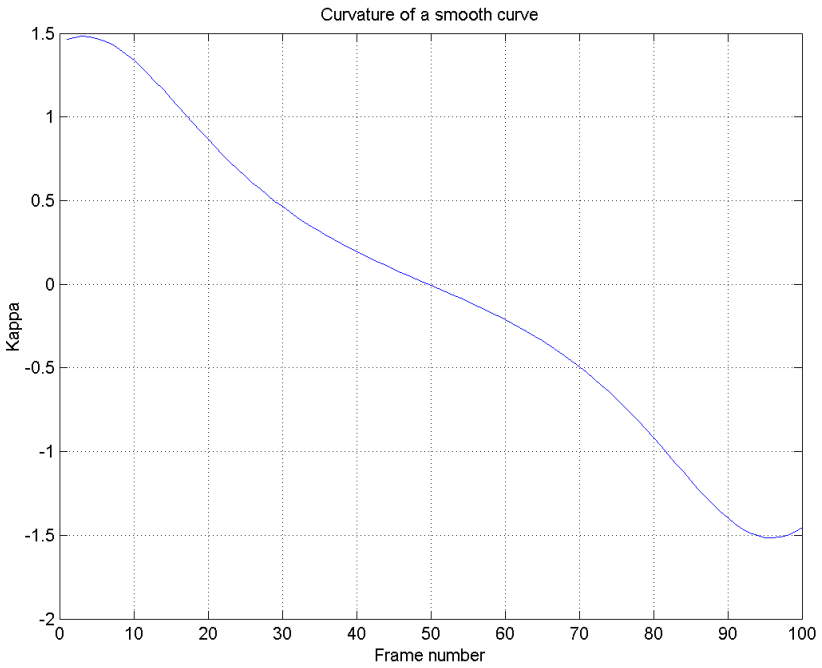


Figure 6.2: The curvature of the smooth path

In figure 6.1 a trajectory is made on the playing field. Figure 7.1 in Appendix C shows the connection between the control points of such a spline. The curvature of the smooth path from figure 6.1 is shown in figure 6.2. Where the maximum curvature κ is ± 1.5, which corresponds to a radius of $66,667$ cm (see eq.4.9). This is in accordance to the sharpness of the curve in figure 6.1. The idea to limit the maximum velocity for a curvature can be done by testing at which velocity the robot remains controllable, given the turn radius.

For successful control the feed-forward controller as well as the feed-backward controller is necessary. In figure 6.3 both controllers are used to track the first part of the smooth curve build in figure 6.1. The feed-forward control uses robot inverse kinematics to calculate feed-forward inputs from the reference trajectory. The feed-backward cancels the effects of noise, disturbances and initial state errors. The simulator as well as practice proves that a mobile robot can follow the desired reference trajectory. Figure 6.4 shows the result if only feed-backward or only feed-forward is used with a velocity of 200mm/s.

Using high velocities is only possible with smooth curves otherwise it becomes uncontrollable. This uncontrollability emerges from the fact that when controlling the robot with high velocities it should be frequently and carefully corrected. Therefore it requires accurate and also more frequent measurements from the state estimator than it produces now. The result of following a trajectory at 1200mm/s

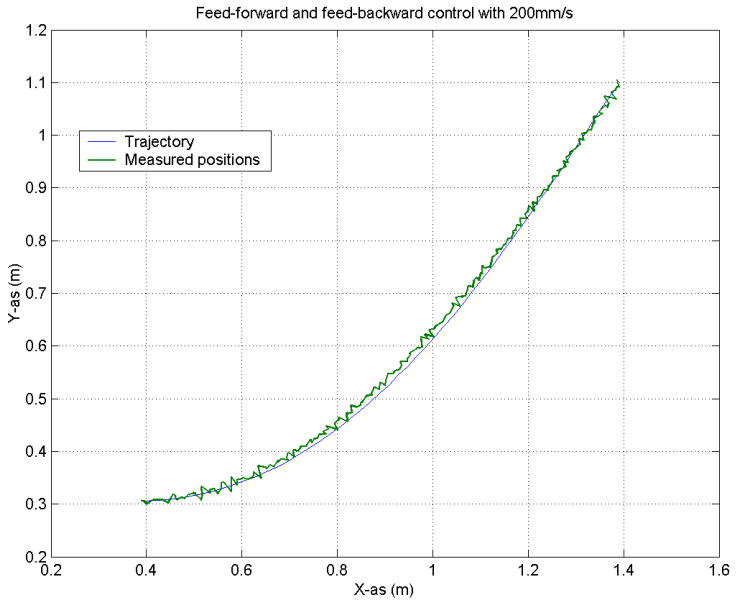


Figure 6.3: Combined feed-forward and feed-backward control tracking example with 200mm/s

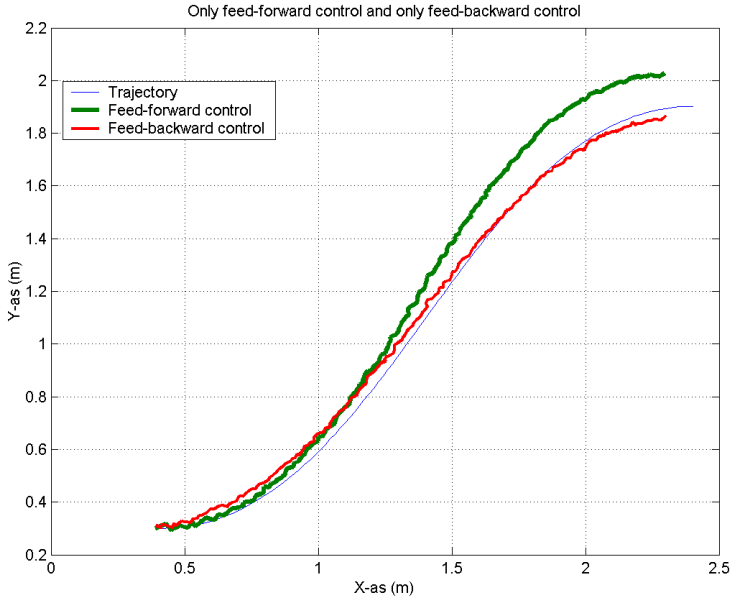


Figure 6.4: Only feed-forward and only feed-backward control tracking example with 200mm/s

is given in figure 6.5. The figure shows a good result for a quite high velocity, this is attributed to a smooth curve. Figure 7.2 in Appendix C gives also a good result due to the low velocity. But because the curve in figure 7.2 is much sharper than figure 6.5, the result when controlling the robot with 1000mm/s is worse. The computation time needed for a complete control cycle has been measured and takes 23μs.

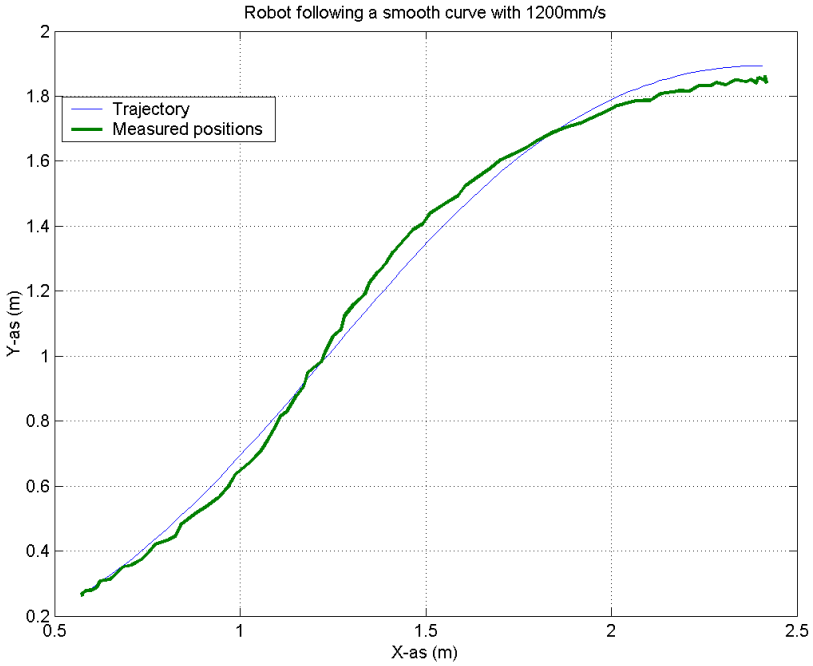


Figure 6.5: Feed-forward and feed-backward control with 1200 mm/s

An important factor is how accurate the robot arrives on the end pose. Therefore a smooth curve and a MiaBot are used to test the accuracy of arriving on the end pose. Table 6.1 shows the mean end pose error of following the curve four times with different velocities. Where the absolute deviations in the x and y position are given in mm and the orientation is given in radians. The table shows that the deviation becomes larger when the velocity increases, which is according to our expectations.

When positional errors during trajectory tracking become large the controller has to correct a lot, which may lead to an unstable control system. Therefore accurate measurements are important. The mean position error of a trajectory is related to the absolute positional errors measured each frame. The mean position error is repeated four times for each velocity and is given in table 6.2. The table shows that raising the velocities cause a larger mean error, which is obvious since higher velocities are more difficult to control. The maximum position errors are the max-

Velocity	End pose deviation (x, y, θ)
200 mm/s	(15, 8, 0.035)
400 mm/s	(24, 6, 0.208)
600 mm/s	(19, 11, 0.091)
800 mm/s	(26, 25, 0.107)
1000 mm/s	(48, 27, 0.180)
1200 mm/s	(75, 48, 0.411)

Table 6.1: End pose deviation in mm and rad of the MiaBot after four tests

imum absolute errors measured during trajectory tracking for different velocities. These maximum positional errors are given in table 6.2. According to our expectations the table shows that the maximum error becomes larger when the velocity increases.

Velocity	Mean Error in position (x,y)	Max Error in position (x,y)
200 mm/s	(3 , 4)	(27, 29)
400 mm/s	(8 , 9)	(38, 24)
600 mm/s	(9 , 11)	(48, 40)
800 mm/s	(22 , 25)	(62, 53)
1000 mm/s	(26 , 21)	(94, 77)
1200 mm/s	(56 , 36)	(134, 102)

Table 6.2: Mean and max deviations in mm of the MiaBot after four tests

6.3 Ball-handling algorithms

The evaluation of the implemented ball-handling algorithms has to deal with the performance of motion planning and control during the execution of the specified skill. For both skills the ball-handling algorithms are suffering from limitations due to the imperfect controllable dynamics.

6.3.1 Shoot skills

Shooting a moving ball is implemented and works under the following conditions.

- Since the robot is constrained by a nonholonomic property, it is not always feasible to hit the ball in a specified direction. This is especially the case when the ball is predicted aside of the robot like figure 6.6. In the case the ball is predicted aside the robot the given end velocity is also frequently not reachable.
- Since the curvatures used in a shoot skill are not restricted, the velocity profile is frequently not suited for the trajectory. This causes uncontrollable

behavior of the robot. Therefore the shoot skill does only work with a low velocity.

- During execution of a shoot skill, opponents are ignored. In spite of the fact that they are ignored, they do exist in a real game situation. This can lead to a ball which is radically changed when it is pinched by the opponent and this makes updating the interception point invalid. Such situations are not handled by the shoot skill. In such situations it is better to let the strategy decide what to do next (e.g. initiate a new shoot action).

- Updating the spline during a shoot skill becomes more difficult to track if the robot has already passed the last but one control point.

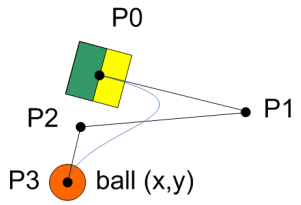


Figure 6.6: Difficult to kick, due to the sharp curvature which is forced by the selected control points [$P0, .., P3$]

Due to the improved motion planning module the scoop skill in the old system has become superfluous. The scoop skill tries to scoop the ball from the wall if it is in fixed contact with it. The new system considers the size of the robot and never plans a trajectory outside the playing field and thus is always capable to scoop the ball. Calculating the interception point takes an average time of ±3ms. Therefore the robot which has to shoot should be chosen carefully.

When comparing the new shoot algorithm with the old one, there is a large improvement noticeable. With our shoot algorithm we are able to score a moving ball while the old system was not. Scoring a non moving ball works fine in both cases but is not a real challenge.

The kick type of the shoot skill works fine with the conditions given above. The dribble skill has the kick as its first phase, but extends on it in the second phase when robot possess the ball. It appears that controlling the ball after a first collision is very hard. The ball is frequently lost due to the collision. Actually in most cases it is more a matter of luck instead of a well functioning dribble skill when pushing the ball to a desired end position.

The panic shoot does frequently suffer from an incorrect preceding turn, which causes the panic shoot to fail. When simulating these three shoot types, they do function very well. The only disadvantage of the simulator is that it does not contain a physical model of collisions. Therefore it is difficult to evaluate a simulated dribble with the ball.

6.3.2 Catch skill

Evaluating pure feed-backward control is done by testing the catch skill. The test is to choose destination points on the line planned for the catch skill. The goal keeper as well as the first defender robot has to travel to the planned destination state without feed-forward control. Using only feed-backward control makes this a reactive approach and is therefore differently tuned by choosing other $k_{1,2,3}$ values. This result in a stronger velocity correction, which is more inaccurate due to slippery wheels.

The nature of the feed-backward controller is based on correcting these disturbances by means of measurements. Due to dependency of the orientation accuracy from the EKF and the low image frame rate of the camera it is hard to correct disturbances with only a feed-backward control. Especially when in a game situation the opponent collide with the goal keeper and the goal keeper is pushed away, the feed-backward control has to recover from this undesired state. Therefore it is hard to tune the $k_{1,2,3}$ values to the optimum and is the reactive approach by using only the feed-backward control unsatisfying.

Chapter 7

Conclusions

This chapter describes conclusions concerning the results of the improved motion planning and control and points out research that has to be conducted in the future, to further improve the robot soccer system.

7.1 Results

We can conclude that an extended Kalman filter (EKF) will assist in getting more accurate world data. The old EKF is neglecting the measurements whereas our modified EKF neglects the control signals. Therefore a better EKF implementation is still necessary that helps in finding a better balance between the measurements and the control signals.

Using the Bézier splines as paths gives more flexibility than S-curves. For example, the planned paths are always clamped within the playing field. Therefore the scoop skill in the old MI20 system has become superfluous. The Bézier spline is easily updated, which is pretty useful for ball-handling skills. Consequently, the shoot skill is now able to shoot a moving ball, which is a very useful improvement. Using the convex hull property for collision detection is useful when it is not too large as described in section 3.2. We also conclude that a small step size is able for mapping distance as a physical value to a Bézier spline. Mapping time as physical value instead of a distance makes no difference in terms of CPU load.

Due to unreliable transmission and incorrect handling of control signals GermanBots were difficult to control, therefore we were especially focused on MiaBots. Controlling a mobile robot along a reference trajectory is presented with a controller consisting of a feed-forward and feed-backward part. The simulator as well as practice proves that a mobile robot can follow the desired reference trajectory. The controller is most suitable when using feed-forward as well as feed-backward control. Due to the unrestricted curvatures the controller functions quite well with low velocities. In practice velocities must be limited depending on the local curvature, as high velocities complicate controlling the robots due to the imperfect controllable dynamics.

7.2 Future work

When doing future work, it is recommend and essential to maintain a well designed software structure. This will keep the revised software structure modular and easier to update with new research technologies. There will always be work to do concerning improvements with regard to motion prediction, planning and control. Therefore the most important issues are enumerated below.

Filtering Filtering of the three independent orientations of the own team robots instead of using an average value, improves world data to be more accurate. It may be recommended to do research in nonlinear filtering, such as the use of an unscented Kalman filter or at least make the EKF a bit more control signal dependent. To make opponent measurements more accurate it is recommend knowing the other colors on their color patches too. Until now we are only using the thirty percent color region of their color patches, which makes the prediction quite imprecise.

Motion planning Extending Bézier splines into B-splines will give more flexibility for planning trajectories and avoiding obstacles, but it will also raise the complexity. To avoid obstacles it is recommended using Hermite-splines due to the property of crossing through each control point. It is also recommended to add time passing information to the polygon defined by only three subsequent control points as described in section 3.2. This will increase the accuracy of collision detection. An useful recommendation for the strategy module is the opportunity to get specific parameters of the planned trajectories, such as total travel time, etc. Furthermore it is useful to make the most optimal curvatures by choosing the most optimal endpoint length of Bézier curves, as this improves the travel time of the robot.

Motion control Controlling the robot is restricted due to constraints from the less controllable dynamics. Controlling nonholonomic robot dynamics like in eq.2.6 will lead to faster and better control, but these are also much more difficult to solve. It is advised to use a more advanced feedback controller able to also compensate large orientation errors or at least a feed-forward control for the defend system. Another recommendation is to use more accurate world data with high velocities. Therefore a camera with higher frame rate will help to improve trajectory tracking and as a result of that the robots can be better controlled.

Bibliography

[1] M. Vendittelli A. Luca De, G. Oriolo. Control of wheeled mobile robots: An experimental overview.

[2] F. Andersson. Bézier and b-spline technology.

[3] T.D. DeRose B.A. Barsky. *IEEE Computer Graphics and Applications*. IEEE Computer Society Press, January 1990.

[4] Randal W. Beard. Motion planning using waypoints. 2003.

[5] J. Borenstein and Y. Koren. *The Vector Field Histogram, fast object avoidance for mobile robots*. IEEE Journal of Robotics and Automation, 1991.

[6] F. Šolc and B. Honzik. Control of the mobile micro-robot. 2003.

[7] W.D.J. Dierssen. Motion planning in a robot soccer system. Master's thesis, University of Twente, 2003.

[8] G. Dudek and M. Jenkin. *Computational principles of mobile robotics*. Cambridge University Press, 2000.

[9] D. Matko G. Klančar. State space control of differential steering robot. 2005.

[10] A. Schaft van der G. Meinsma. Inleiding wiskundige systeemtheory.

[11] M. Vandittelli G. Oriolo, A. Luca. *Dynamic Feedback Linearization: Design, Implementation and Experimental Validation*. IEEE Transactions on Control Systems Technology, Secaucus, NJ, USA, 2002.

[12] G. Bishop G. Welch. An introduction to the kalman filter. 2001.

[13] S. Grond. Handling collisions in a robotsoccer environment. Master's thesis, University of Twente, 2005.

[14] Y.K. Hwang and N. Ahuja. Gross motion planning. a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.

[15] D-S Kwon J-H Hwang, R.C. Arkin. Mobile robots at your fingertip: Bézier curve on-line trajectory generation for supervisory control.

[16] R.E. Kalman. *A new approach to linear filtering and prediction problems*. T-ASME, 1960:35-45.

[17] N.S. Kooij. The development of a vision system for robotic soccer. Master's thesis, University of Twente, 2003.

[18] J.C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991.

[19] J.P. Laumond. *Robot Motion Planning and Control*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1998.

[20] J. van der Linden. Dynamic avoidance control of soccer playing mini-robots. Master's thesis, University of Twente, 2005.

[21] Merlin Systems Corp. Ltd. Miabot pro bt v2, user manual, rev. 1.2. 2002.

[22] M. Lundgren. Path tracking and obstacle avoidance for a miniature robot. Master's thesis, Umeå University, 2003.

[23] E.M. Schepers. Improving the vision of a robot soccer system. Master's thesis, University of Twente, 2004.

[24] Jeffrey K. Uhlmann Simon J. Julier. *A New Extension of the Kalman Filter to Nonlinear Systems*. The Robotics Research Group.

[25] J. Steward. *Calculus Early Transcendentals, fifth edition*. Thomson, 2003.

[26] T. Verschoor. Dynamic motion planning of soccer playing mini-robots. Master's thesis, University of Twente, 2004.

Appendix A

Prove of the derivation of the curvature[1].
In two dimensions, let a plane curve be given by Cartesian parametric equations $x = x(t)$ and $y = y(t)$. Then the curvature kappa is defined by:

$$\kappa \equiv \frac{d\theta}{ds} = \frac{\frac{d\theta}{dt}}{\frac{ds}{dt}} = \frac{\frac{d\theta}{dt}}{\sqrt{(\frac{dx}{dt})^2+(\frac{dy}{dt})^2}} = \frac{\frac{d\theta}{dt}}{\sqrt{\dot{x}^2+\dot{y}^2}} \tag{7.1}$$

where θ is the tangential angle and s is the arc length. The $\frac{d\theta}{dt}$ derivative in the above equation can be found using the identity

$$tan\theta = \frac{dy}{dx} = \frac{dy/dt}{dx/dt} = \frac{\dot{y}}{\dot{x}}$$

so

$$\frac{d}{dt}(tan\theta) = sec^2\theta\frac{d\theta}{dt} = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2}$$

and

$$\frac{d\theta}{dt} = \frac{1}{sec^2\theta}\frac{d}{dt}(tan\theta)$$

$$= \frac{1}{1 + tan^2\theta}\frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2}$$

$$= \frac{1}{1 + \frac{\dot{y}^2}{\dot{x}^2}}\frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2}$$

$$= \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\dot{x}^2 + \dot{y}^2} \tag{7.2}$$

Combining eq.7.1 and eq.7.2 then gives

$$\kappa = \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{(\dot{x}^2 + \dot{y}^2)^{\frac{3}{2}}}$$

[1] http://mathworld.wolfram.com/Curvature.html

Appendix B

Prove of the controllability To prove if our robot is controllable the kalman controllability matrix from eq.4.13 must have full rank see [10]

$$rank(B, AB, A^2B) = 3 \tag{7.3}$$

Therefore we start with the kinematics of the mobile nonholonomic robot, which is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos\theta & 0 \\ sin\theta & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{7.4}$$

The derived error state $\tilde{q}$ is given by

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x_a - x_d \\ y_a - y_d \\ \theta_a - \theta_d \end{bmatrix} \tag{7.5}$$

With eq.7.5 and eq.7.4 the following kinematic model is obtained:

$$\begin{bmatrix} \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{bmatrix} = \begin{bmatrix} cose_3 & 0 \\ -sine_3 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} v_d \\ \omega_d \end{bmatrix} + \begin{bmatrix} -1 & e_2 \\ 0 & -e_1 \\ 0 & -1 \end{bmatrix} . \begin{bmatrix} u1 \\ u2 \end{bmatrix} \tag{7.6}$$

Where v_d is the feed-forward linear velocity, ω_d is the feed-forward angular velocity and v1 and v2 are outputs from the closed loop.

$$\begin{aligned} u_1 &= u_{d1}.cos(e3) - v_1 \\ u_2 &= u_{d2} - v_2 \end{aligned} \tag{7.7}$$

From eq.7.7 expressing the closed-loop inputs and rewriting eq.7.6 results in

$$\begin{bmatrix} \dot{e}_1 \\ \dot{e}_2 \\ \dot{e}_3 \end{bmatrix} = \begin{bmatrix} 0 & u2 & 0 \\ -u2 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} . \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} + \begin{bmatrix} 0 \\ sin(e3) \\ 0 \end{bmatrix} .v_d + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} v1 \\ v2 \end{bmatrix} \tag{7.8}$$

Further on linearizing eq.7.8 around the operating point ($e1 = e2 = e3 = 0, v1 = v2 = 0$) results in the following linear model

$$\dot{\tilde{q}} = \begin{bmatrix} 0 & u_{d2} & 0 \\ -u_{d2} & 0 & u_{d1} \\ 0 & 0 & 0 \end{bmatrix} .\tilde{q} + \begin{bmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \Delta u \tag{7.9}$$

Which is in state space form

$$\dot{\tilde{q}} = A.\tilde{q} + B.u \tag{7.10}$$

Now the rank can be determined and when either v_d or ω_d are nonzero, the above linear system becomes controllable, since matrix

$$C = [B \quad AB \quad A^2B] = \begin{bmatrix} 1 & 0 & 0 & 0 & -\omega_r^2 & v_r\omega_r \\ 0 & 0 & -\omega_r & v_r & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

has rank 3 provided . Therefore, we conclude that the kinematic system eq.2.3 can be locally stabilized.

Appendix C

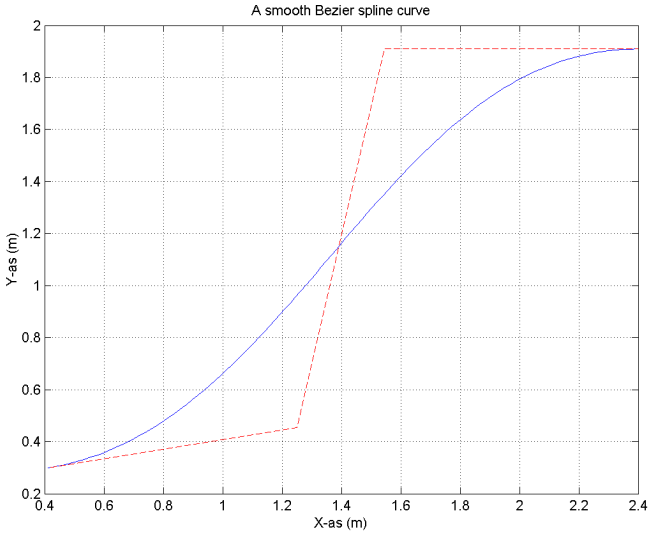


Figure 7.1: A smooth curve with connected control points

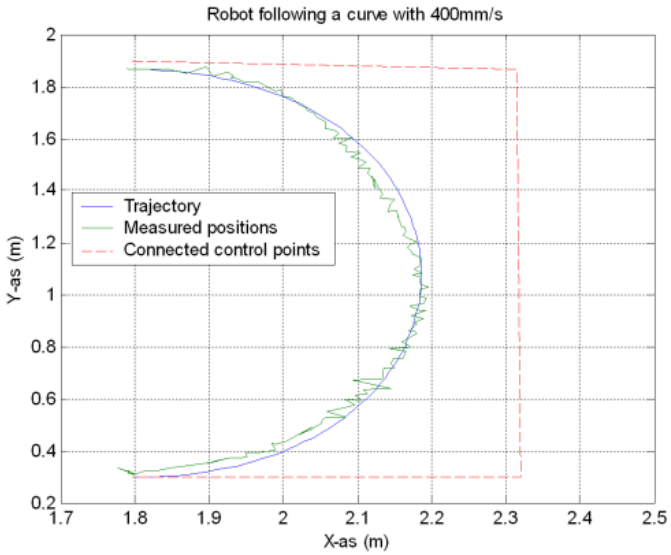


Figure 7.2: Robot following a curve with 400mm/s

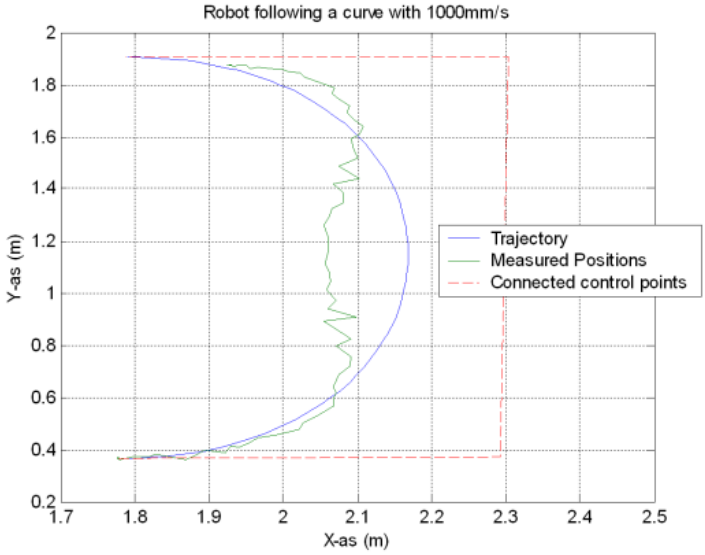


Figure 7.3: Robot following a curve with 1000mm/s