Examensarbete

LEAP
A Platform for Evaluation of Control Algorithms

Examensarbete utfört i Datorseende
vid Tekniska högskolan i Linköping
av

Kristoffer Öfjäll

LiTH-ISY-EX--10/4370--SE

Linköping 2010
LEAP
A Platform for Evaluation of Control Algorithms

Examensarbete utfört i Datorseende
vid Tekniska högskolan i Linköping
av

Kristoffer Öfjäll

LiTH-ISY-EX--10/4370--SE

Handledare: Fredrik Larsson
isy, Linköpings universitet

Examinator: Michael Felsberg
isy, Linköpings universitet

Linköping, 1 juni, 2010
Most people are familiar with the Brio labyrinth game and the challenge of guiding the ball through the maze. The goal of this project was to use this game to create a platform for evaluation of control algorithms. The platform was used to evaluate a few different controlling algorithms, both traditional automatic control algorithms as well as algorithms based on online incremental learning.

The game was fitted with servo actuators for tilting the maze. A camera together with computer vision algorithms were used to estimate the state of the game. The evaluated controlling algorithm had the task of calculating a proper control signal, given the estimated state of the game.

The evaluated learning systems used traditional control algorithms to provide initial training data. After initial training, the systems learned from their own actions and after a while they outperformed the controller used to provide initial training.
Abstract

Most people are familiar with the Brio labyrinth game and the challenge of guiding the ball through the maze. The goal of this project was to use this game to create a platform for evaluation of control algorithms. The platform was used to evaluate a few different controlling algorithms, both traditional automatic control algorithms as well as algorithms based on online incremental learning.

The game was fitted with servo actuators for tilting the maze. A camera together with computer vision algorithms were used to estimate the state of the game. The evaluated controlling algorithm had the task of calculating a proper control signal, given the estimated state of the game.

The evaluated learning systems used traditional control algorithms to provide initial training data. After initial training, the systems learned from their own actions and after a while they outperformed the controller used to provide initial training.

Sammanfattning

Många känner till och har provat labyrintspelet från Brio. Målet med detta projekt var att skapa en plattform baserat på detta spel för att utvärdera styralgoritmer. För att visa några av möjligheterna med plattformen har olika styralgoritmer implementerats och utvärderats.

Styralgoritmen som utvärderas har möjlighet att styra spelet genom två servon. En kamera tillsammans med datorseendealgoritmer används för att skatta tillståndet hos spelet.

Bland de utvärderade styralgoritmerna finns några som är baserade på självlärande system. Dessa är designade för att startas helt olärda. En traditionell styralgoritm används för att generera träningssdata i början varefter det lärande systemet tar över kontrollen och fortsätter lära. Efter en stunds träning presterar det lärande systemet bättre än styralgoritmen som genererat träningsdata.
Acknowledgments

Many people have contributed, directly or indirectly, to this work. I would like to thank my examiner Michael Felsberg and my supervisor Fredrik Larsson for supporting the idea of the labyrinth platform and for providing many new ideas. As the number of possible ways of using the platform seem to far outweigh the time available, the help of selecting what to do and what not to do, has been greatly appreciated.

I would also like to thank my fellow masters students and all people at the Computer Vision Laboratory for fruitful discussions in the masters’ room as well as in the fika\textsuperscript{1} room. The people of the Automatic Control group also deserves mentioning albeit most of the ideas related to automatic control were not implemented.

Finally, I would like to thank all my new and old friends making my time spent at the university really enjoyable.

\textsuperscript{1}A verb indicating ingestion of coffee, tea or such in combination with pastries and/or cookies, often in a social context. The word may also be used as a noun referring to an occurrence of the said act.
Contents

1 Introduction ................................................. 5
   1.1 Goal .................................................. 6
   1.2 Outline ............................................. 6

2 System Overview ........................................... 9
   2.1 Hardware ............................................ 9
      2.1.1 Terminology .................................. 10
   2.2 Software ........................................... 12
      2.2.1 The LEAP Software ......................... 12
      2.2.2 The User Interface .......................... 14
      2.2.3 The Record Viewer .......................... 14

3 Background Theory ........................................ 17
   3.1 Control ............................................. 18
      3.1.1 Dynamic Systems ............................ 18
      3.1.2 Traditional Automatic Control .......... 19
      3.1.3 Learning System ............................. 19
         3.1.3.1 Locally Weighted Projection Regression .... 19
   3.2 Estimation ......................................... 22
      3.2.1 Motivation .................................... 22
      3.2.2 Homography Estimation .................... 22
      3.2.3 Marker Detection and Tracking .......... 24
         3.2.3.1 Detection ................................ 24
4 Hardware
4.1 Controlling the Game .................................................. 32
   4.1.1 Controlling the Servos ........................................ 33
4.2 Vision Hardware ..................................................... 34
4.3 Alternative Maze .................................................... 35
4.4 Hardware Model ...................................................... 36
   4.4.1 Physical Modeling .............................................. 36
   4.4.2 Identification .................................................. 38
5 Experimental Setup .................................................... 41
5.1 Performance Measure ............................................... 42
5.2 The Target Point Algorithm ...................................... 43
5.3 Adapting the PtD Controller ...................................... 44
5.4 Mappings for the Learning System ............................... 44
5.5 Evaluation Scenarios ............................................... 46
   5.5.1 Scenario 1: Simple Trajectory ............................... 46
   5.5.2 Scenario 2: Simple Trajectory with Modified Behavior ... 46
      5.5.2.1 Scenario 2a: Offset Change ............................... 48
      5.5.2.2 Scenario 2b: Reverse Control ............................. 48
   5.5.3 Scenario 3: The BRIO Maze .................................... 48
6 Results and Discussions ................................................. 51
6.1 Data Presentation .................................................... 51
6.2 Scenario 1: Simple Trajectory .................................... 52
   6.2.1 Discussion ...................................................... 52
6.3 Scenario 2: Simple Trajectory with Modified Behavior ....... 54
   6.3.1 Discussion ...................................................... 54
6.3.1.1 Scenario 2a ........................................ 54
6.3.1.2 Scenario 2b ........................................ 55
6.4 Scenario 3: The BRIO Maze ................................. 59
   6.4.1 Discussion ......................................... 59
6.5 Final Results ........................................... 62

7 Conclusions and Future Work ................................. 65
   7.1 The Platform ......................................... 65
   7.2 The Evaluated Controlling Algorithms .................... 66
   7.3 Future Work ........................................... 67

Bibliography ................................................................ 69

A Software .................................................................. 71
   A.1 The LEAP Software ...................................... 72
      A.1.1 Multithreading ...................................... 72
      A.1.2 The Frame .......................................... 74
      A.1.3 The Frame Cycle .................................... 74
      A.1.4 The LEAP Interface ................................. 76
      A.1.5 The Data Logging Subsystem ...................... 76
         A.1.5.1 The Message Queue ............................ 76
         A.1.5.2 The Frame Queue and Frame Writing ........ 77
         A.1.5.3 The Timing Module ............................. 77
      A.1.6 The Image Source Subsystem ...................... 78
         A.1.6.1 The Camera Module ............................ 78
         A.1.6.2 The Vision Routines ............................ 79
      A.1.7 The Controller Subsystem ........................... 80
         A.1.7.1 The State Estimation Filter .................... 81
         A.1.7.2 The Controller Container ....................... 81
         A.1.7.3 Controlling Algorithms ......................... 81
         A.1.7.4 The Control Signal Output Module .......... 82
   A.2 The User Interface ........................................ 82
   A.3 The Record Viewer ........................................ 83
List of Figures

2.1 The system .................................................. 10
2.2 Terminology .................................................. 11
2.3 System overview ............................................. 13
2.4 User interface ............................................... 15

3.1 PID controller ............................................... 18
3.2 Learning controller ......................................... 20
3.3 Image rectification ......................................... 24

4.1 The game ..................................................... 32
4.2 Servo installation ........................................... 33
4.3 Vision marker ............................................... 34
4.4 Alternative maze ........................................... 35

5.1 RMSOE illustration .......................................... 42
5.2 Illustration of the target point algorithm ................. 43
5.3 Target trajectory for scenarios 1, 2a and 2b ............... 47
5.4 Target trajectory for scenario 3 ............................ 49

6.1 Scenario 1 results ........................................... 53
6.2 Scenario 1 raw results ....................................... 53
6.3 Scenario 2a results ......................................... 56
6.4 Scenario 2b results ......................................... 56
6.5 LWPR-4 trajectories for runs 200-203, scenario 2b ........ 57
6.6 LWPR-4 trajectory for run 204, scenario 2b ............... 57
6.7 PID controller trajectories, scenario 2a .................... 58
6.8 Scenario 3 results ........................................... 60
6.9 LWPR-4 trajectories for early runs, scenario 3 ............ 61
6.10 LWPR-4 trajectories for late runs, scenario 3 ............ 61

A.1 Software overview .......................................... 73
A.2 Frame cycle ............................................. 75
1

Introduction

The BRIO labyrinth game has challenged humans since 1946. The objective is simple: guide the ball through the maze by tilting the plane while avoiding the holes. Most people who have tried it can tell that in practice, the game is really not that simple. By means of computer vision and servo actuators, the challenge can now be handed over to the machines.

There exist numerous algorithms for learning systems and automatic control. The Labyrinth Evaluation of Algorithms Platform, LEAP, is able to test how well these algorithms cope with reality. If the evaluated algorithm fail to complete the original maze, the difficulty can be adjusted by using alternative mazes.
1.1 Goal

The stated goal of this project has been to develop an evaluation platform for learning systems and automatic control algorithms. In order to achieve this, two main issues have been addressed. First, the evaluated algorithm has had to know the position of the ball. Secondly, this algorithm has had to be able to affect the game. Additionally, everything has had to be done in real time.

To achieve the stated goal, the labyrinth game has been fitted with computer controlled actuators. Computer vision has been used to enable the algorithm-at-test to receive information regarding the position of the ball. In addition to the software necessary to extract the position of the ball from the camera images, a few different control algorithms have been implemented.

Another goal has been to enable almost arbitrarily positioning of the camera relative to the labyrinth, only requiring that the maze should be visible from the camera position and cover a significant part of the image.

1.2 Outline

Chapter 2 contains a brief overview of the system and its main components. Terms used to refer to specific parts of the system are introduced.

In chapter 3, a summary of the main algorithms used in the system is provided.

A thorough description of the system hardware is available in chapter 4. The system software is thoroughly described in appendix A. The software appendix is intended for the reader interested in implementation details and is not necessary in order to understand the remainder of this thesis.

The experimental setup used when evaluating the controlling algorithms is presented in chapter 5. Results and discussions regarding these experiments can be found in chapter 6.

Finally, in chapter 7, conclusions regarding the performed experiments and the evaluation platform itself are presented. The chapter also contains ideas for future work.
2

System Overview

In this chapter an overview of the system is presented. This chapter is divided into two sections presenting the hardware and software respectively. The terminology used to refer to specific hardware parts of the system is introduced in section 2.1.1.

An image of the system is shown in figure 2.1 and a schematic view of the system is presented in figure 2.3.

2.1 Hardware

The LEAP\(^1\) hardware consists of a Labyrinth game, servo actuators and a camera. A desktop computer is used to run the software. The game is based on a standard Brio labyrinth game which has been slightly modified in order to allow a computer to control the game. More details regarding the hardware can be found in chapter 4.

The modification of the labyrinth game consists of adding two standard servos usually used in radio controlled vehicles. Instead of using a radio receiver to control the servos, the servos are connected to a controller card that directly interface the computer.

A camera is used to capture images of the game from which the state of the game can be estimated.

\(^1\) Labyrinth Evaluation of Algorithms Platform
2.1.1 Terminology

The different parts of the system are denoted as follows. In this section, numbers within parentheses refer to labels in figure 2.2.

The game frame(1) of the labyrinth consists of the red part of the game which has contact with the supporting surface, normally a table, and in its turn supports the other parts of the game.

Attached to the game frame is the outer gimbal ring(2) or just outer gimbal. The outer gimbal is attached to them game frame in a way that allows rotation of the ring relative to the game frame around an axis parallel to the supporting surface.

Attached to the outer gimbal ring is in its turn the inner gimbal ring(3) or simply the inner gimbal. The inner gimbal is free to rotate relative to the outer gimbal around an axis orthogonal to the axis of rotation between the frame and the outer gimbal. When the outer gimbal ring is in its neutral position, the axis of rotation between the inner and outer gimbals is parallel to the supporting surface.

The maze(4) is the part of the game that may contain holes and obstacles. This is where the ball will move unless it falls through a hole. The maze is rigidly attached to the inner gimbal ring. The maze may be exchanged with mazes of different difficulty. The two gimbal rings allow the normal of the maze to obtain an arbitrary orientation relative to the game frame.

The maze plane coordinate system or a point in maze coordinates refers to a two
2.1 Hardware

A dimensional cartesian coordinate system in a plane coinciding with the maze. The origin is fixed in one of the corners of the maze and the coordinate system uses millimeters as units of length. To estimate the orientation of the maze relative to the camera, there are four markers, each marked with number 5 in figure 2.2.

Attached to the frame are also the two servos controlling the tilt of the gimbal rings by means of ball joints and control rods. Each of the two servos are connected to one of the outer and inner gimbal ring respectively. When it is necessary to distinguish between the two servos, they are called the outer servo(6) and the inner servo(7) respectively. The outer servo is the servo controlling the outer gimbal ring while the inner servo control the inner gimbal ring.

When referring to the axes of the maze plane coordinate system, the terms outer direction and inner direction are used. All these outer and inner terms have a logical connection. As an example, changing the deflection of the outer servo causes the outer gimbal ring to move relative to the frame. Depending on the position of the ball, this will probably change the component in the outer direction of the acceleration of the ball.

A servo controller card is used to generate control pulses to the servos.

The ball ejection plate is a tilted plate attached inside the frame making the ball travel to the ball collector(8) if the ball falls through a hole in the maze.
2.2 Software

The software in the system may be further divided into the LEAP software, the user interface and the record viewer. Note that the user interface, which enables the user to interact with the LEAP software, is dependent on the operating system, and not a part of the LEAP software. More details regarding the software can be found in appendix A.

Within the LEAP software resides the controlling algorithm, that is, the algorithm being evaluated.

In this section and in appendix A, the term frame is used frequently. In the context of the software, a frame refers to an image with some additional data belonging to that image.

2.2.1 The LEAP Software

The LEAP software is the central part of the system performing tasks such as capturing and analyzing images, estimating the state of the system as well as calculating the control signal and sending this to the servo controller.

The controlling algorithm is one of the core components within the LEAP software. The purpose of the controlling algorithm is to calculate the appropriate control signal given the estimated state of the system.

By changing controlling algorithms, the performance of different algorithms may be measured and compared. In the LEAP system, there might be several controlling algorithms loaded at the same time. The algorithm actually controlling the game is called the active controlling algorithm. The active controlling algorithm may be changed at any time while the system is running.

As a part of the image analysis needed to estimate the state of the game, the images acquired from the camera are rectified. This means that the images are resampled to a grid aligned with the maze. To do this, the positions of four markers in the image are found and tracked. If these markers can not be found automatically, their initial positions may be supplied from the user interface.

The LEAP system uses multithreading to be able to do several things simultaneously. This is primarily needed in order to be able to capture and process new frames while waiting for old frames to be written to disk. Writing old frames to disk provides for offline analysis of data and offline training of learning systems.
Figure 2.3. Schematic overview of the system.
2.2.2 The User Interface

As the name suggests, the user interface acts as an interface between the LEAP software and the user. Starting the user interface will start the rest of the LEAP system. The user interface can be seen in figure 2.4.

Via this graphical interface the user may control the LEAP software to:

- Start and stop capturing of images.
- Start and stop writing of images, control signal and state data to file.
- Load and save controlling algorithms.
- Change the active controlling algorithm.
- Change parameters in the loaded controlling algorithms.
- View the image captured by the camera as well as the rectified image. Information regarding the state of the system is superimposed on the images.
- Manually initialize the marker tracker.

2.2.3 The Record Viewer

When running the system, images and data may be saved to disk. Using the record viewer, these data can be played back and analyzed. The images may also be exported as bit mapped pictures.

As the recorded data also contain the full state of the system as well as the applied control signal, the record viewer may use the recorded data to train learning systems offline. As the data files usually are of considerable size (approximately 15 megabytes per recorded second), image data may be removed to reduce the file size and facilitate faster training.
Figure 2.4. The user interface.
To play the game, some kind of control signal has to be applied. The proper control signal is dependent on the current as well as the desired state of the game. In section 3.1, a few methods able to generate the control signal, given the current and the desired state, are presented.

To estimate the current state, methods from section 3.2 may be used.

How the desired state is generated depends on the performed experiment. The generation of the desired states used in the performed experiments in this thesis are described in chapter 5.
3.1 Control

The platform is designed to be used for evaluating different ways of controlling the game. To demonstrate the functionality of the platform, a few different controlling algorithms are implemented. In addition to the automatic control algorithms described below, it is also possible to control the game manually by means of a joystick.

3.1.1 Dynamic Systems

Consider a system currently in state $x$, applying a control signal $u$ will put the system in another state $x^+$. A system is said to be dynamic if the new state $x^+$ does not only depend on the applied control signal $u$ but also on the old state $x$.

One first approach to control such a system might be to consider the difference between the output of the system and a desired output from the system. This difference is called the control error. The control signal can be chosen proportional to the control error. By considering how the control error change over time, a better control algorithm may be created. This is done, to some extent, in the PID controller described in section 3.1.2.

A model of the system can be used to predict the resulting new state $x^+$, given the current state $x$ and the applied control signal $u$. Using the system model, a controller more tuned to the current system can be calculated.

If the system model is invertible, the so called inverse dynamics can be calculated. This can be used to directly calculate the control signal given the current state and a desired state. Another option is to directly estimate the inverse dynamics from measured system data. This is the approach used by the learning system in section 3.1.3.
3.1 Control

3.1.2 Traditional Automatic Control

The traditional automatic control is in this thesis represented by a PID controller. An introduction to PID controllers and control theory in general can be found in [6].

A general PID controller is described by

\[ u(t) = Pe(t) + D \frac{de(t)}{dt} + I \int_{0}^{t} e(\tau) \, d\tau \]  \hspace{1cm} (3.1)

where \( u(t) \) is the control signal and \( e(t) \) is the control error.

The parameters \( P \), \( I \) and \( D \) are used to adjust the influence of the proportional part, the derivative part and the integrating part respectively. A schematic view of a system containing a PID controller can be seen in figure 3.1.

The derivative part usually stabilizes the system as the control signal is reduced when the system moves toward the reference. The integrating part may remove any static control error. In the case of the LEAP system, the integrating part applies the control signal needed to keep the maze horizontal.

3.1.3 Learning System

Using learning systems, the inverse dynamics might be learned instead of modeled. Consider a system currently in state \( x \), applying a control signal \( u \) will put the system in another state \( x^+ \). Learning the inverse dynamics means that given the current state \( x \) and a desired state \( x^+ \), the learning system should be able to estimate the required control signal \( u \) bringing the system from \( x \) to \( x^+ \). In figure 3.2, the learning system in the left box should produce a control signal based on the current and desired state of the system.

This can be seen as approximating the mapping

\[ \begin{pmatrix} x \\ x^+ \end{pmatrix} \rightarrow (u) . \]  \hspace{1cm} (3.2)

There are several ways of expressing the system state as well as there are several algorithms useful for learning the mapping in equation (3.2).

3.1.3.1 Locally Weighted Projection Regression

For learning the mappings above, Locally Weighted Projection Regression, LWPR, may be used [11]. The idea is to use the output from several local linear models weighted together to form the output. The parameters of these models are adjusted online by the algorithm to fit the training examples presented to the learning system.
In LWPR, the output from the $K$ local models are weighted together as

$$\hat{y} = \frac{\sum_{k=1}^{K} w_k y_k}{\sum_{k=1}^{K} w_k} \quad (3.3)$$

to form the output of the system. In the equation $y_k$ and $w_k$ are the output and weight for each local model. The output of the whole system is denoted $\hat{y}$. In this section, $k$ will be used to index the local linear models.

The weight, $w_k$, for each local model is determined by the distance from the input to the center of the respective local model in input space. Each local model has its own distance metric described by the matrix $D_k$. The distance metric is subject to change as the area of validity of the local model is discovered.

The center of the local model, $c_k$, is set when the local model at hand is created whereafter it does not change. Employing a Gaussian kernel, the weight for a certain local model $k$ when presented with the input $x$ is calculated as

$$w_k = \exp \left( -\frac{1}{2} (x - c_k)^T D_k (x - c_k) \right). \quad (3.4)$$

When creating a new local model, the matrix $D_k$ is usually set to a diagonal matrix and the LWPR algorithm may be set to keep the distance matrix diagonal.

Each local model consists of a few linear one-dimensional function approximations along different directions in the input space. Each projection direction and its corresponding linear coefficient is adjusted to fit the training examples.

The output of each local model is calculated as

$$y_k = \beta_k^0 + \sum_{i=1}^{r_k} \beta_{k,i} u_{k,i}^T (x_{k,i} - x_k^0) \quad (3.5)$$

where $r_k$ is the number of projection directions in the current model $k$, $\beta_k^0$ and $x_k^0$ is the mean of input and output training data seen so far, $u_{k,i}$ are the projection directions and $\beta_{k,i}$ the corresponding regression variables.
3.1 Control

The input is in $x_{k,i}$, where $x_{k,1} = x$ is the unaltered input and $x_{k,i+1}$ is generated from $x_{k,i}$ by removing the part of the input-output relation explained by the projection regression $i$.

When the system has several outputs, one independent LWPR network is created for each output. The LWPR algorithm is more thoroughly described in [11]. The choice of variable names in this short presentation is mostly consistent with the mentioned paper. Some exceptions have been made where this makes the variables more easily distinguishable.

Partial Least Squares

To find the projection directions $u_i$ and the linear coefficients $\beta_i$ used in LWPR, Partial Least Squares, PLS, is used. The basic idea of PLS is presented below. A more compact description of the algorithm using matrix notation is available in [11].

Given a set of input vectors $x_1, \ldots, x_n$ and corresponding outputs $y_1, \ldots, y_n$, PLS projects and regresses the data along $r$ orthogonal directions. For $i = 1, \ldots, r$, the following steps are performed:

First, the direction, $u_i$, within the input space with maximum correlation with the output is found as

$$u_i = \sum_{j=1}^{n} y_j x_j. \quad (3.6)$$

Note that any non-spherical distribution of the input vectors $x_j$ will bias the projection direction. The input vectors are projected onto $u_i$ according to

$$s_{i,j} = x_j^T u_i, \quad j = 1, \ldots, n. \quad (3.7)$$

The parameter $\beta_i$ in the linear model $y = \beta_i s$ is fitted to the projections in a least squares\(^1\) manner according to

$$\beta_i = \frac{\sum_{j=1}^{n} y_j s_{i,j}}{\sum_{j=1}^{n} s_{i,j}^2}. \quad (3.8)$$

Finally, the residual error is calculated and used as data for the next iteration. This is done according to

$$y_j = y_j - s_{i,j} \beta_i \quad \hat{j} = 1, \ldots, n \quad (3.9)$$

$$x_j = x_j - \frac{s_{i,j} u_i}{|u_i|^2} \quad j = 1, \ldots, n. \quad (3.10)$$

In LWPR, this algorithm is implemented in an incremental fashion. Also, (3.10) is modified to make the distribution of $x_1, \ldots, x_n$ more spherical each iteration.

---

\(^1\)Minimizing $\epsilon_i = \sum_{j=1}^{n} (y_j - \beta_i s_{i,j})^2$. 

3.2 Estimation

For the controllers to be of any use, the state of the system has to be known or at least estimated. In this section, some algorithms used to estimate the state of the game are presented. A short motivation for the use of these algorithms is presented in 3.2.1.

3.2.1 Motivation

The state of the system might, among other variables, contain the position and velocity of the ball. These should preferably be expressed in a coordinate system fixed to the maze, as the maze move when the game is played. To estimate the state, images of the game are captured.

Assuming the maze is approximately planar and that the used camera can be described well enough by the pinhole camera model, a homography may be estimated and used to transform the image to the maze plane coordinate system. As expressed in section 3.2.2, this requires four points and their corresponding projections on the camera sensor to be known.

The maze is fitted with four markers and a method for finding the projections of the markers is described in section 3.2.3.

Given these rectified images, the position of the ball has to be found. This may be done by creating a model of the background and look for any difference between the rectified image and the background model. Different ways of creating a background model are presented in section 3.2.4.

Finally, given the position of the ball, the remaining state variables have to be estimated. This is possible using a Kalman filter and a model of the system. The Kalman filter is described in section 3.2.5.

3.2.2 Homography Estimation

A homography may be used to describe the mapping of points on a plane to an image sensor in a camera. This is shown in [4]. In this case, the hyperplanes have two dimensions. Using homogenous coordinates, the homography may be described by a three by three matrix $H$ as

$$
\begin{pmatrix}
  x \\
  y \\
  1
\end{pmatrix}
\sim
H
\begin{pmatrix}
  u \\
  v \\
  1
\end{pmatrix}
=
\begin{pmatrix}
  \hat{h}_{11} & \hat{h}_{12} & \hat{h}_{13} \\
  \hat{h}_{21} & \hat{h}_{22} & \hat{h}_{23} \\
  \hat{h}_{31} & \hat{h}_{32} & \hat{h}_{33}
\end{pmatrix}
\begin{pmatrix}
  u \\
  v \\
  1
\end{pmatrix}.
$$

In the equation, $(u, v)$ is the coordinates in the image of the projection of a point $(x, y)$ on the plane. This is illustrated in figure 3.3. The point $(u, v)$ in the camera image corresponds to the point $(x, y)$ in the maze.
The notation $\mathbf{a} \sim \mathbf{b}$ is used to indicate that $\mathbf{a}$ and $\mathbf{b}$ represent the same element in a projective space. In this case, that is,

$$\mathbf{a} \sim \mathbf{b} \iff \mathbf{a} = \begin{pmatrix} a_1 \\ a_2 \\ 1 \end{pmatrix} = \begin{pmatrix} b_1 s \\ b_2 s \\ s \end{pmatrix} = \mathbf{b}s \quad (3.12)$$

holds for some $s \neq 0$.

From above it is apparent that scaling all elements in $\mathbf{H}$ with a nonzero constant will result in the same relation. $\mathbf{H}$ thus have eight degrees of freedom and (3.11) can be expressed according to

$$\begin{pmatrix} x s \\ y s \\ s \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}. \quad (3.13)$$

Carrying out the matrix multiplication yields

$$xs = h_{11}u + h_{12}v + h_{13}$$
$$ys = h_{21}u + h_{22}v + h_{23}$$
$$s = h_{31}u + h_{32}v + 1. \quad (3.14)$$

Substituting $s$ in the two first equations results in

$$h_{31}ux + h_{32}vx + x = h_{11}u + h_{12}v + h_{13}$$
$$h_{31}uy + h_{32}vy + y = h_{21}u + h_{22}v + h_{23}$$
$$s = h_{31}u + h_{32}v + 1. \quad (3.15)$$

Note that the equations are linear in the elements of the homography matrix and that the third equation is linearly dependent on the first two. Skipping the last equation, the relations can be expressed according to

$$\begin{pmatrix} u \\ v \\ 1 \\ 0 \\ 0 \\ 0 \\ -ux \\ -vx \\ 0 \\ 0 \\ u \\ v \\ 1 \\ -uy \\ -vy \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix}. \quad (3.16)$$

Stacking the equations from four known points and their known projections, the elements of $\mathbf{H}$ may be solved for, and the homography may be used to calculate the position of any point in the plane given its projection in the image. Inverting the homography enables the opposite.

A proof that a homography actually may be used to describe the relation mentioned as well as an introduction to projective geometry, projective spaces and homogenous coordinates is given in [4].
3.2.3 Marker Detection and Tracking

In this context marker detection and tracking is about finding a specified number of markers in an image and tracking these markers in temporally subsequent images.

The algorithm can be divided into two parts, initialization and tracking. The first part concerns finding the markers when the location of the markers in the previous frame is not known. This is described in the detection section 3.2.3.1. The second part tracks the found markers. This is described in the section 3.2.3.2. To be robust against changes in light intensity, the marker detection and tracking is based on color rather than intensity.

### 3.2.3.1 Detection

A distance image, $D$, is created as

$$D(x) = ||I(x) - c_m||.$$  \hfill (3.17)

The position $x$ varies over the whole input image $I$, $c_m$ is the color of the marker and the norm $||:||$ is the euclidean distance in the color plane.

A filter enhancing local minima of the same size as the expected size of the markers is applied to the distance image. This is followed by marker detection. Searching for $n$ markers, the following is performed.\(^2\)

\(^2\)The suggested pseudocode include traversing the image at least $n + 1$ times. The algorithm actually used only visit each pixel once.
for i = 1 to n+1
    find and save position of global minimum, z, in image I
    for all pixels x in I where distance(x, z) < r
        set x to positive infinity
    end
end

The parameter \( r \) is the minimum spatial distance between two markers. The detections, \( x_i \), are sorted such that \( D(x_i) < D(x_{i+1}) \).

By detecting one more minimum than asked for, a certainty measure may be calculated as

\[
C = \frac{D(x_{n+1}) - D(x_n)}{D(x_n)}.
\]  

(3.18)

Simply put, when the relative difference between \( D(x_n) \) and \( D(x_{n+1}) \) is small, it is an indication that either less than \( n \) markers or more than \( n \) markers were detected. In either way, the detected marker positions are uncertain.

### 3.2.3.2 Tracking

If the position of a marker is known in the previous frame, the search in the current frame may be limited to a small area centered around the previous known location of the marker. This assumes that the markers move with limited speed.

Calculating the distance image and minima enhancing filtering is only done in this small area, for each marker. The new marker positions can now be found by finding the minimum in each area, assuming that the areas do not overlap.

### 3.2.4 Background Model

With a model of the background, a captured image may be compared to the model to detect anything in the image that is not a part of the background, that is, foreground. A few examples of different background models can be found in [2].

#### 3.2.4.1 Approximate Median Background Model

For a long enough time interval, the median value over time for each pixel belongs, with high probability, to the background.\(^3\) This can be used to create a background model from a sequence of images without the need of a specific image known to contain only background.

\(^3\) This may in fact be used to define what the term background should mean.
As calculating the median requires some computational effort and also cannot be done without all images available, an approximate median background model, as in [12], may be used.

Every time a new image is available, the background model is updated as follows.

```plaintext
for each pixel i
    if new_value(i) > model(i)
        model(i) += alpha
    else
        model(i) -= alpha
end
```

When the model converges, approximately half of the new values will be greater then the model (and the other half will be smaller). That is, the model is close to the median of the data.

The parameter $\alpha$ controls the rate of convergence of the background model. Using a small value makes the model react very slowly to changes in the background. The downside of using a large value is that if the foreground remain stationary for some time, the background model may adapt to the foreground.

### 3.2.4.2 Gaussian Mixture Model

Sometimes, the median background model will not be sufficient. This might be the case when the background contains moving boundaries, where the pixels close to the edge periodically change. In that case, the background model should be a mixture of two models, each describing the conditions on either side of the boundary. Using normal distributions to model each side of the edge, the background model ends up being a **Gaussian Mixture Model**, gmm, [8], also described in [12].

When using gmm to model image background, each pixel has its own model as described by

$$ P(I|\Gamma) = \sum_{k=1}^{K} w_k P_k(I|\Gamma_k) $$  \hspace{1cm} (3.19)

where $P(I|\Gamma)$ is the probability of the pixel attaining the intensity $I$ given that it belongs to the background $\Gamma$. The model is a mixture of $K$ normal distributions each weighted with $w_k \geq 0$ subject to

$$ \sum_{k=1}^{K} w_k = 1. $$  \hspace{1cm} (3.20)

The weights $w_k$ can be interpreted as the probability of the pixel being generated by the submodel $\Gamma_k$, given that the pixel belong to the background. That is
$w_k = P(\Gamma_k | \Gamma)$. Assuming that the pixel value is generated by one and only one of the submodels, equation (3.19) is a variation of the law of total probability.

The last factor in equation (3.19), $P_k(I | \Gamma_k)$, is the probability of a pixel attaining the intensity $I$ given that it belongs to the submodel $\Gamma_k$. Assuming normal distributions of the submodels, the probability can be expressed as

$$P_k(I | \Gamma_k) = \int_{I-0.5}^{I+0.5} \frac{1}{\sigma_k \sqrt{2\pi}} e^{-\frac{(i-\mu_k)^2}{2\sigma_k^2}} di. \quad (3.21)$$

Note that the intensity values, $I$, are assumed to be integers. This, in combination with the integral in (3.21), is necessary for expressions like $P(I | \Gamma)$ to represent probabilities. Otherwise, these would be probability density functions. In practice, the integral is usually approximated using the value of the integrand in the middle of the interval, that is, sampling the density.

Using an incremental update algorithm, the parameters $w_k$, $\mu_k$ and $\sigma_k$ can be adjusted to fit the data seen so far. Usually $K$, the number of submodels, is fixed.

The background models, one for each pixel, may be used to create a probability image $B(x)$ when presented with a new image $I(x)$. The probability image can be generated according to

$$B(x) = P_x(I(x) | \Gamma) \quad (3.22)$$

where $P_x$ is the background model for pixel $x$.

The probability image contains the probability of every pixel attaining the current value given that the pixel belongs to the background. Pixels having an intensity with a low probability in the corresponding background model may be seen as potential foreground pixels.

This might seem strange. Estimating $P_x(\Gamma | I(x))$, that is, the probability of the pixel belonging to the background given the current pixel value, would be closer to the intended purpose of the model.

Using Bayes’ rule, $P(A \cap B) = P(A)P(B | A)$, the relation between $P_x(I(x) | \Gamma)$ and $P_x(\Gamma | I(x))$ may be expressed as

$$P_x(\Gamma | I(x)) = \frac{P_x(\Gamma)}{P_x(I(x))} P_x(I(x) | \Gamma). \quad (3.23)$$

It may be noted that the authors of [8] model both the foreground and background using a Gaussian mixture model. The classifying decision is then made by comparing the probability of the current value of the pixel being generated from a background or foreground distribution.
3.2.5 Kalman Filter

Given a state space model of a system and some observations dependent on the state of this system, a Kalman filter [5], may be used to estimate the full state of the system. This is possible even if some state variables can not be directly measured, as long as the system is observable.

For more information on time continuous and time discrete state space models see [6] and [3]. The latter also contain a thorough description of the Kalman filter.

The filter used in this thesis is based on a time discrete system of the form presented in (3.24) where \( n \) is the time index, \( \mathbf{x} \) is the state vector, \( \mathbf{u} \) is the input and \( \mathbf{y} \) is the output. The process noise is represented by \( \mathbf{w} \) with covariance matrix \( \mathbf{Q}_n \) while the measurement noise is represented by \( \mathbf{v} \) with covariance matrix \( \mathbf{R}_n \). The noise is assumed to be white and all terms are assumed to have zero mean. In this version of the Kalman filter, the process noise is assumed to be uncorrelated with the measurement noise, that is, \( E[\mathbf{w}(t)\mathbf{v}(\tau)^T] = 0 \). The matrices \( \mathbf{A}, \mathbf{B} \) and \( \mathbf{C} \) are used to represent the state space model of the system.

\[
\begin{align*}
\mathbf{x}_{n+1} &= \mathbf{A}\mathbf{x}_n + \mathbf{B}\mathbf{u}_n + \mathbf{w}_n \\
\mathbf{y}_n &= \mathbf{C}\mathbf{x}_n + \mathbf{v}_n
\end{align*}
\] (3.24)

The Kalman filter can be shown to give the optimal linear estimation of the state vector with respect to the variance of the state error. If the noise is both Gaussian and white, the Kalman filter provides the optimal state estimation, that is, there is no non-linear filter providing better estimations [3].

Subscripts on the form \( n|k \) will be used to denote an estimation of the value at time \( n \) given measured data until time \( k \). Additionally \( \hat{\mathbf{x}}_{n|k} \) is the estimated state and \( \mathbf{P}_{n|k} \) is the covariance matrix of the estimated state. Using the covariance information \( \mathbf{P}_{n|k}, \mathbf{Q}_n \) and \( \mathbf{R}_n \) for weighting, the Kalman filter fuses information from measurements and the system model to estimate the state of the system.

When the control signal \( \mathbf{u}_n \) is available, the next state may be estimated as

\[
\begin{align*}
\hat{\mathbf{x}}_{n+1|n} &= \mathbf{A}\hat{\mathbf{x}}_{n|n} + \mathbf{B}\mathbf{u}_n \\
\mathbf{P}_{n+1|n} &= \mathbf{A}\mathbf{P}_{n|n}\mathbf{A}^T + \mathbf{Q}_n.
\end{align*}
\] (3.25)

When the measurement at \( n + 1 \) is available the prediction is updated as

\[
\begin{align*}
\hat{\mathbf{x}}_{n+1|n+1} &= \hat{\mathbf{x}}_{n+1|n} + \mathbf{L}_{n+1}(\mathbf{y}_{n+1} - \mathbf{C}\hat{\mathbf{x}}_{n+1|n}) \\
\mathbf{P}_{n+1|n+1} &= \mathbf{P}_{n+1|n} - \mathbf{L}_{n+1}\mathbf{C}\mathbf{P}_{n+1|n} \mathbf{C}^T.
\end{align*}
\] (3.26)

where the Kalman gain \( \mathbf{L} \) is

\[
\mathbf{L}_{n+1} = \mathbf{P}_{n+1|n}\mathbf{C}^T(\mathbf{C}\mathbf{P}_{n+1|n}\mathbf{C}^T + \mathbf{R}_n)^{-1}.
\] (3.27)
In this chapter, a detailed description of the hardware is given. The platform is based on a Brio labyrinth game acquired from the local toy store. The game is shown in figure 4.1.

The game is slightly modified to allow controlling the game from a computer. Provisions for easier estimation of the game state have been added. An alternative maze, completely free of holes and obstacles, has also been created.

In addition to the game, the system employs a camera and means of directing it toward the game, that is, a tripod.

The modifications made to control the game are presented in section 4.1 together with a description of the control hardware. In section 4.2, the vision hardware is described as well as the modifications made to simplify tracking.

Section 4.3 describes the alternative maze while section 4.4 present a state space model of the hardware.
4.1 Controlling the Game

The original means of control consists of two knobs each controlling the tilting of the maze in one of the two directions by pulling cables attached to the middle of each of the four sides of the maze.

This controlling mechanism has been removed and replaced with two standard RC\textsuperscript{1} servos with linkage to the gimbal rings. The installation is shown in figure 4.2. Two Hitec HS-475HB servos are used [10].

In the figure, the servos are shown with large deflections from the neutral position. When the servos are in their neutral position, all parts of the linkage are at right angles resulting in an approximately linear transformation from servo deflection angle to maze tilt angle.

There are two reasons to remove the original means of control. First, the servos would render the original control useless. Secondly, the original control might interfere with the servos making the whole system less responsive.

\textsuperscript{1}Equipment normally used for Radio Controlled vehicles.
4.1 Controlling the Game

4.1.1 Controlling the Servos

The servos are controlled by sending pulses to the servos. The servo arm tries to maintain an angular position proportional to the width of the pulse. A pulse width of \(1500 \mu s\) makes the servo attain its central position. By changing the width of the pulse up to \(\pm 600 \mu s\), the servo moves from its central position.

An ordinary RC receiver demodulates a signal consisting of one pulse for each servo stacked after each other with a small pause between each pulse. After the pulse to the last servo, a longer pause inform the receiver that the following pulse should be directed to the first servo. The receiver just split the signal during the pauses and forward the correct pulse to the correct servo.

In this case, a controller card is used to generate the pulses directly. The controller card receives the desired pulse widths for each servo from the computer. The controller then generates pulses with the correct width until new pulse width information is received.

The used controller card is of type SSC-32 from Lynxmotion [7]. It is based on an Atmega16 one chip computer from Atmel and communicates with the desktop computer via a standard RS-232 link. The controller card is in its turn controlled by sending simple ASCII\(^2\) character strings telling which servo, or which servos, to move and their corresponding new positions. These ASCII-strings are sent by the output part of the LEAP software.

---

\(^2\)American Standard Code for Information Interchange.
For viewing the game, an IEEE-1394 Marlin F145C2 camera from Allied Vision Technologies is used [1]. At the current configuration, the camera delivers 15 color images per second with a resolution of 800 by 600 pixels. The camera driver makes the frames available to the LEAP software in YUV422 format.

The camera, mounted on a tripod, is directed towards the game. The game board is modified in order to simplify the estimation of the homography between the maze plane coordinate system and the image coordinate system. The modifications consist of four spherical, colored markers recessed into some of the obstacles in the maze, see figure 4.3.

The markers have known positions in the maze plane coordinate system. One of the markers has a different color to distinguish it from the others. This odd colored marker makes it possible to resolve which of the four possible orientations of the maze plane coordinate system is correct.
4.3 Alternative Maze

The original maze contains a lot of holes and obstacles. Due to the small distances between these objects in the maze relative to the size of the ball, there might be a problem detecting any significant behavior of the controlling algorithm. To accommodate better possibilities of observing progress by the controller, an alternative maze has been made.

The surface of the original maze is also quite uneven. This is especially evident in some places where large dents in the surface appear. These dents are the result of casting the obstacles. To spread the obstacle material to all obstacles from one point, there are paths of obstacle material under the maze base. These paths result in recession of the maze base wooden fiber plate visible where there are no obstacles on the upper side.

The alternative maze consists of a transparent plastic board that can be placed on top of the original maze with a tight fit inside the inner gimbal. In addition to being free of holes and obstacles, the alternative maze has a much smoother surface than the original maze. The game with the alternative maze in place can be seen in figure 4.4.
4.4 Hardware Model

To estimate the state of the system using a Kalman filter, a linear state space model of the system is needed. System identification was performed to generate such a model.

The system with the original maze contains severe nonlinearities where the ball hits the obstacles. Therefore the model was created using the hole and obstacle free alternative maze and the model is not expected to be valid where the ball is in contact with the edges of the maze.

One of the requirements on the filter is to estimate the ball velocity. In order to make sure that ball velocity is one of the states, a gray box model is used.

Physical relations are used to set up the equations in the model while system identification is used to estimate certain parameters. The model is divided into two submodels, each is derived the same way describing motion in the outer and inner direction respectively.

4.4.1 Physical Modeling

In this section, a model for one of the directions is derived. The model for the other direction is similar, only the values of the parameters differ.

The servo is modeled as a proportionally controlled motor with a gearbox. The servo motor (dc-motor) and gearbox is modeled as

\[\ddot{\theta} = -a\dot{\theta} + bv\]  

(4.1)

where \(\theta\) is the output axis angle and \(v\) is the input voltage. The model has two parameters \(a\) and \(b\). The dots represent derivation with respect to time.

The servo motor is controlled by means of a proportional controller with a pulse width modulated output. As the pulse frequency is very fast compared to the servo dynamics, the output may be viewed as controlling the voltage. The feedback is thus modeled as \(v = K(K_2u - \theta)\) where \(u\) is the reference signal, proportional to the desired servo angle. \(K\) and \(K_2\) are free parameters. In the finished model, \(u\) will be the input from the servo controller card.

Inserting the controller into equation (4.1) yields

\[\ddot{\theta} = -bK\theta - a\dot{\theta} + bKK_2u.\]  

(4.2)

Note that this is a general second order system.

If one would derive the equation describing the relation between servo angle, maze tilt and finally ball acceleration, one would end up with an equation containing

---

3A gray box model is something between a model fully derived from physical relations (sometimes called a white box model) and a model created by just considering input and output relations (a black box model), hence the term gray box model.

4See section 2.1.1.
4.4 Hardware Model

a few trigonometric functions. The physical layout of the control linkage enables the relation to be approximately described as linear for small deflections but with an additional offset. The offset is necessary as setting the control signal to zero may not cause the maze to be horizontal.

The model used for ball motion is

\[ \ddot{y} = c(\theta + \theta_0) - d\dot{y} \quad (4.3) \]

where \( y \) is the ball position and \( \theta_0 \) is the maze offset. In the equation, \( d \) model friction and \( c \) is the linearized relation between maze tilt and ball acceleration. Both parameters contain the ball mass.

Using the state vector

\[
\begin{pmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 x_4 \\
 x_5
\end{pmatrix} =
\begin{pmatrix}
 y \\
 \dot{y} \\
 \theta \\
 \dot{\theta} \\
 \theta_0
\end{pmatrix}
\quad (4.4)
\]

the combination of equations (4.2) and (4.3) can be expressed as the continuous time state space model

\[
\dot{x} =
\begin{pmatrix}
 0 & 1 & 0 & 0 & 0 \\
 0 & -d & c & 0 & c \\
 0 & 0 & 0 & 1 & 0 \\
 0 & 0 & -bK & -a & 0 \\
 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 x_4 \\
 x_5
\end{pmatrix} +
\begin{pmatrix}
 0 \\
 0 \\
 0 \\
 bK_2 \\
 0
\end{pmatrix} u
\quad (4.5)
\]

\[
y = (1 \ 0 \ 0 \ 0 \ 0) \dot{x}
\]

Finally using forward difference approximation of the derivative \( \dot{x} \approx \frac{x^{n+1} - x^n}{T} \Rightarrow x^{n+1} \approx x^n + T\dot{x} \) the time discrete model

\[
x^{n+1} =
\begin{pmatrix}
 1 & T & 0 & 0 & 0 \\
 0 & 1 - dT & cT & 0 & cT \\
 0 & 0 & 1 & T & 0 \\
 0 & 0 & -bKT & 1 - aT & 0 \\
 0 & 0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
 x_1 \\
 x_2 \\
 x_3 \\
 x_4 \\
 x_5
\end{pmatrix} +
\begin{pmatrix}
 0 \\
 0 \\
 bK_2T \\
 0
\end{pmatrix} u
\quad (4.6)
\]

can be obtained from (4.5). \( T \) is the sampling interval while \( n \) is the time index.
4.4.2 Identification

To identify the unknown parameters in the model, data was collected by manually controlling the game while recording the ball position and servo control signal.

The game was controlled as to avoid the ball hitting the edges of the maze while making inputs of varying amplitude and frequency. The identification data was thus collected under the influence of manual feedback rendering the control signal somewhat dependent on the output.

The nonlinearity at the maze edges made it inappropriate to use a completely random control signal. It may be noted that the sampling interval $T$ is fixed and dependent on the frame rate of the camera. Also, some of the parameters in (4.6) are grouped together before identification as all of them cannot be identified individually. Validation of the model was done both by comparing with validation data and manually by playing a simulation of the game based on the model.
To evaluate the performance of the different controlling algorithms, a few experiments have been carried out. The experiments are described in this chapter and the results are presented in chapter 6.

Each controlling algorithm has been evaluated by letting it guide the ball along different trajectories. The deviations from these target trajectories have been measured and a single performance measure for each run through one trajectory has been calculated. This measure, the Root Mean Squared Orthogonal Error, RMSE, represent how well the target trajectory has been followed. The controlling algorithms have been evaluated using a few different scenarios. The scenarios have used different setups and target trajectories.

The trajectory error measure, RMSE, is described in section 5.1. The scenarios are described in section 5.5. Three controlling algorithms were evaluated. They are denoted PID, LWPR-2 and LWPR-4. The controlling algorithms are described in sections 5.3 and 5.4. The evaluated controlling algorithms were supplied with target points along the desired trajectory. The algorithm generating these points is described in section 5.2.
5.1 Performance Measure

The ability of the evaluated control algorithms to follow a given trajectory is measured and compared.

This path following ability is measured as the root of the mean squared distance from the actual trajectory to the desired trajectory. The distance is measured orthogonal to the desired trajectory every frame. This measure is abbreviated RMSOE, Root Mean Squared Orthogonal Error. Figure 5.1 illustrates the measure.

This metric is designed with a few properties in mind. It provides one single value for each run. Using $L^2$ instead of $L^1$, the RMSOE put more weight on large deviations than on small. When using the real maze, small deviations would be acceptable while large deviations would risk the ball falling through a hole. The RMSOE is also designed not to favor any specific speed along the trajectory. On the negative side, it may be noted that the ball stopping on the target trajectory for a considerable time results in a lower value than might be expected from the rest of the run since one measurement is obtained for each frame. It may also be noted that RMSOE is not a metric in mathematical sense. As the distance is measured orthogonal to the target trajectory, switching the actual and target trajectories may not produce the same result.

In the experiments, the desired trajectory is followed back and forth several times. At each end of the trajectory, the RMSOE for the last leg is calculated and stored.
5.2 The Target Point Algorithm

As the evaluated algorithms expect a reference point or reference velocity for the ball, this has to be calculated given the target trajectory and the current ball position. In the performed experiments, this desired position, called the target point, is supplied by a deterministic algorithm.

When a new trajectory is first activated, the point on the target trajectory closest to the current ball position is found. This point is the target point. When a current target point exist, the target point is moved along the trajectory until outside a specified radius from the current ball position. Note that if the current target point already is farther away from the current ball position than the specified radius, the target point does not change. The two latest scenarios are illustrated in figure 5.2.

If the end of the trajectory is found before finding a point outside the search radius, a direction flag is toggled to follow the trajectory in the other direction.

The mentioned search radius is set by the controlling algorithm. In the performed experiments, the search radius is set to 15 mm.

Figure 5.2. Illustration of the target point algorithm. The bold black line is the target trajectory directed in the direction of the arrow. The gray dashed circle is the previous target point and the solid line circle is the new target point. The filled circle is the current ball position and the black dashed circle illustrate the search radius. Two different scenarios are shown.
5.3 Adapting the PID Controller

The PID controller described in section 3.1.2 only handles one dimensional inputs and outputs. Two of these controllers are used to control the game, one for each direction. The two controllers have no interconnection.

The control error for each direction is calculated as the difference between the current ball position and the target point. The target point is provided by the algorithm described in section 5.2. The output from the controllers are sent to the respective servo.

Dividing this two dimensional system into two parts assumes that the outer servo does not affect the position of the ball in the inner direction and vice versa. Further, only considering the relation between the ball position and the target point, not the absolute ball position, assumes that the system behavior does not change with the position of the ball. As a result of this, the PID controller is not expected to perform well when the ball is in contact with the edges of the maze.

The P, I and D parameters were manually adjusted by the author. In the following chapters, this PID based controlling algorithm will be referred to as the PID.

5.4 Mappings for the Learning System

The idea behind the controller based on a learning system is described in section 3.1.3. As mentioned, there are several different ways to describe the current and desired state of the game.

Two options on how to describe the current state have been evaluated. One is based on velocities and the other uses both velocities and the absolute position of the ball. The first option is similar to the PID with respect to what kind of information is available to the controlling algorithm. For the PID, the ball position relative to the target point, that is, the control error, can be seen as indicating the desired state.

The desired state of the game is expressed as a desired velocity of the ball in all the conducted experiments involving learning systems. This desired velocity has a constant speed and is directed towards the target point.

The learning systems are trained online. The current state and a desired state is fed into the learning system and the control signal calculated by the learning system is used. When the resulting state of this action is known, the triple old state, applied control signal and the resulting state is used for training. The learning systems are thus able to learn from their own actions.

In this section, the representation of the current and the desired state is denoted input. The control signal predicted by the learning system is the output. In the following equations, \( p \), \( v \) and \( u \) denote position, velocity and control signal respectively. A subscript, \( o \) or \( i \), indicates if the aforementioned value correspond
The superscript plus sign has a twofold interpretation. The interpretation of the
superscript plus sign depend on if the learning system is currently used to calculate
the control signal or if the learning system is being trained. In the first case, the
superscript indicate that the corresponding value is a desired value, that is, the
control signal output from the learning system should bring the corresponding
part of the state close to the value supplied as the desired value. In the second
case, that is, when the learning system is trained, the superscript plus sign values
are replaced with the corresponding values resulting from the action taken.

If not enough examples close to the input data has been seen by the learning
system, the system can not make a prediction. In that case, a PID controller will
be used instead. Thus, when starting an untrained system, the PID controller will
control the game completely. As the learning system gets trained, it will start to
decide the control signal more and more often. It is also possible to control the
game manually to provide training data.

The different evaluated learning systems will be denoted LWPR-n, where LWPR
indicate that Locally Weighted Projection Regression is used to learn the mapping
and n is the dimension of the input. LWPR is described in section 3.1.3.1.

The first alternative, LWPR-2 is

\[
\begin{pmatrix}
v_o^-
\end{pmatrix} \rightarrow (u_o)
\]

\[
\begin{pmatrix}
v_i^-
\end{pmatrix} \rightarrow (u_i).
\]

This setup makes the same assumptions regarding the system as those made for
the PID controller. First, the ball can not behave differently in different parts of
the maze. Secondly, the outer servo should not affect the ball position in the inner
direction and vice versa.

By adding the absolute position to the input vectors, LWPR-4 is obtained. The
resulting mappings are

\[
\begin{pmatrix}
v_o
p_o
p_i
v_o^-
\end{pmatrix} \rightarrow (u_o)
\]

\[
\begin{pmatrix}
v_i
p_o
p_i
v_i^-
\end{pmatrix} \rightarrow (u_i).
\]

This learning system should have the possibility to handle different dynamics in
different parts of the maze. Still it is assumed that the control signal in one
direction has little effect on the ball movement in the other.

5.4 Mappings for the Learning System
Depending on the framerate of the camera and the dynamics of the system, the correlation might be low between the applied control signal and the state of the game in the frame acquired directly after the change in control signal. To increase the correlation, the state resulting from a certain action can be measured a few frames later. In the performed experiments, the resulting state has been measured four frames later.

To find proper values for the parameters in the LWPR algorithm, a simple experiment was conducted. Some training and evaluation data was gathered from the system. A computer was set to train several learning systems offline with different parameter values using the training data. The learning systems were then evaluated offline using the evaluation data. The parameters of the learning system with the best result was used.

5.5 Evaluation Scenarios

In this section, the scenarios used to evaluate the algorithms are presented. Each scenario is given a number to be referenced in chapter 6. To avoid the need of picking up the ball manually, all scenarios use the hole and obstacle free alternative maze.

5.5.1 Scenario 1: Simple Trajectory

The trajectory of the first scenario is one period of a sine wave. The wave length is 150 mm and the amplitude is 50 mm. The trajectory is shown in figure 5.3.

All controlling algorithms are expected to be able to pass this scenario with reasonable results.

5.5.2 Scenario 2: Simple Trajectory with Modified Behavior

The second scenario uses the same trajectory as in the first scenario. However the behavior of the system is changed in one half of the maze, the modified area.

This change consists of modifying the control signal to the outer servo when the ball is in the lower half of the maze, below the dotted line in figure 5.3. The control signal modification is performed just before the signal is sent to the servo controller card. Seen from the controlling algorithm, this modification seem to be a part of the system to be controlled.

The purpose of this scenario is to evaluate the controlling algorithms aware of absolute position. These are expected to perform better than the controlling algorithms only aware of relative position or velocity. Two versions of the scenario exist.
Figure 5.3. The target trajectory for scenarios 1, 2a and 2b (dashed line). The dotted line indicate the border between the normal area and the modified area used in scenarios 2a and 2b.
5.5.2.1 Scenario 2a: Offset Change

In this scenario, a constant is added to the servo control signal when the ball is in the modified area. That is, the horizontal maze offset is changed.

5.5.2.2 Scenario 2b: Reverse Control

In this scenario, the different behavior of the system is achieved by reversing the deflection of the outer servo when the ball position is in the modified area.

5.5.3 Scenario 3: The BRIO Maze

The third scenario uses the trajectory from the original maze. The trajectory is shown in figure 5.4.

Note that although the trajectory is from the original maze, the scenario uses the alternative maze. This is mainly due to the current lack of an automated way of putting the ball back in the maze after falling through a hole.
Figure 5.4. The target trajectory for scenario 3.
6

Results and Discussions

In this chapter, the results of the conducted experiments are presented. Each scenario is presented in its own section followed by a discussion. In section 6.5, a comparison of all evaluated controlling algorithms and scenarios is presented.

6.1 Data Presentation

For each experiment and controlling algorithm, a graph of the RMSOE for each run along the target trajectory is presented. As the RMSOE varies quite much between runs, a moving average is applied to the data to make the trends more visible in certain figures. This average value for each run is the average of that run and the 5 previous runs. For the first 5 runs, all previous runs are used in the average. Generally, the first run, starting from the initial position of the ball, is omitted.
6.2 Scenario 1: Simple Trajectory

In figure 6.1 the results for scenario 1 are presented. The data is averaged. For completeness, the same data without averaging is presented in figure 6.2.

6.2.1 Discussion

The results for the first scenario are mostly as expected. All controlling algorithms handle this scenario well.

The performance of the PID is roughly constant over the runs. The LWPRs both perform better than the PID after some training. As seen in figure 6.2, the difference is clear even if the variations between runs are considered. In the end, the learning systems outperform the controlling algorithm used to provide initial training data to these systems.

The two LWPRs perform equally after some runs. The LWPR-4 need more training before the RMSE drops. This might be expected as more dimensions in the input require more training data.

The error curve of the LWPR-4 drops down to the same level as LWPR-2 initially but then rise again between runs 20 and 30. This is due to the ball occasionally visiting new areas of the maze where LWPR-4 can not make predictions of the control signal yet. The LWPR-2 only uses velocity information. Thus, it may utilize information gathered from a different part of the maze.
Figure 6.1. Results for scenario 1, averaged data.

Figure 6.2. Results for scenario 1, non-averaged data.
6.3 Scenario 2: Simple Trajectory with Modified Behavior

Scenario 2 is divided into two parts. The results from the first part, scenario 2a, can be seen in figure 6.3. Figure 6.4 shows the result from scenario 2b. Only LWPR-4 is able to follow the trajectory in scenario 2b and thus there are no results for the other controlling algorithms in the latter figure.

For LWPR-2 to be able to complete scenario 2a, the desired speed toward the target point has to be set higher than for the other scenarios and control algorithms. Keeping the standard speed makes the ball stop when it has entered the modified area of the maze with different offset.

In figure 6.7, some PID trajectories from scenario 2a are shown. The trajectories from runs 200, 201, 202 and 203 by the LWPR-4 in scenario 2b is shown in figure 6.5. This is followed by run 204 in figure 6.6.

6.3.1 Discussion

The two subscenarios are discussed separately. Scenario 2a is discussed in section 6.3.1.1 and scenario 2b is discussed in section 6.3.1.2.

6.3.1.1 Scenario 2a

The trajectories by the PID in scenario 2a have two very specific features (see figure 6.7). When the ball crosses the line from one half of the maze to the other, the change in offset creates a distinct deviation from the target trajectory. This deviation remain until the integrating part of the regulator has compensated for the offset. When following the trajectory in the other direction, the same thing happens. As before, the performance of the PID does not change with the number of runs.

As noted earlier, the LWPR-2 needs a higher desired speed toward the target point to be able to complete runs in scenario 2a. Without the higher target speed, the LWPR-2 learns the required control commands in the first half of the maze. When entering the second half, with modified offset, the learned control signals are not strong enough to compensate for the offset and the ball stops. The LWPR-2 get no new information how to make the ball start to move in the correct direction and since the LWPR-2 can make predictions, the PID is not used.

When using a higher desired speed toward the target point, the control signal applied by LWPR-2 is strong enough to make the ball move even in the area with different offset. When the ball is moving, useful training examples are generated making the LWPR-2 learn the offset used in the modified area. This re-learning is repeated every time the ball crosses the line between the areas of different offset.
6.3 Scenario 2: Simple Trajectory with Modified Behavior

Studying LWPR-2 in figure 6.3, the error first gets smaller, then raises and finally gets smaller again. Correlating that trend with observations made while the system was running, a few things may be noted. The smallest errors corresponds to occasions when the LWPR-2 seemed to be trained for an offset somewhere in the middle between the two different offsets in the scenario. The higher errors in the middle was recorded when the LWPR-2 performed better in one half while performing worse in the other. That is, the system was trained for one of the offsets.

The LWPR-4 on the other hand, is aware of the absolute position of the ball. This enables the LWPR-4 to take care of both offsets. In figure 6.3 it can be seen that after some training, the LWPR-4 reaches a RMSOE not very much higher than the final value in scenario 1.

Changing the learning rate for LWPR-2 might change the results in scenario 2a. Different learning rates have not been tested.

6.3.1.2 Scenario 2b

LWPR-4 was the only algorithm able to complete scenario 2b. Both PID and LWPR-2 compensated in the wrong direction when the ball entered the reversed servo area.

The LWPR-4 uses the PID regulator when not enough training data has been seen in the current region. This creates problems when entering the reversed servo area as the PID regulator can not handle this region. In that area, the LWPR-4 has to be presented with useful training examples by controlling the game manually. After some initial training data, the error decreases. The error continue to decrease as the LWPR-4 learns from its own actions. This can be seen in figure 6.4. Four runs from the end of the scenario are shown in figure 6.5. As can be seen, the LWPR-4 handles the reversed servo area with only small deviations from the target trajectory close to the border.

During run 204 a spike in the error is recorded. This is due to the ball getting outside the area known by the LWPR-4 while in the reversed servo area. The LWPR-4 can not make a prediction and the PID is used instead. Using the PID in the reversed servo area results in the ball moving far from the target trajectory. The PID is used until the ball state happen to enter an area where predictions can be made by the LWPR-4. The trajectory of this run is shown in figure 6.6. Note that this is the run directly after the last run shown in figure 6.5.
Figure 6.3. Results for scenario 2a, averaged data.

Figure 6.4. Results for scenario 2b for LWPR-4. No other controlling algorithm was able to complete any run. Non averaged data.
Figure 6.5. Four runs (200, 201, 202, 203) by the LWPR-4 in scenario 2b. Cyan lines indicate forward runs, blue lines are used for reverse runs. The dashed black line is the target trajectory.

Figure 6.6. Run 204 by the LWPR-4 in scenario 2b.
Figure 6.7. Eight runs by the PID controller in scenario 2a. Cyan lines indicate forward runs, blue lines are used for reverse runs. The dashed black line is the target trajectory.
6.4 Scenario 3: The BRIO Maze

The result from scenario 3 is shown in figure 6.8. Neither the PID nor LWPR-2 is able to complete the third scenario. These controlling algorithms appear not to be able to handle the parts of the trajectory close to the edges of the maze. The observed chain of events is the ball hitting the edge of the maze followed by overcompensation by the controlling algorithm. The ball hits the same edge again and the process is repeated. The amplitude of the oscillations increase. An indication of a similar phenomenon can be seen in the lower right part of figure 6.9. In this case, the oscillations are damped when the LWPR-4 steps in.

A few trajectories performed by the LWPR-4 are presented in figures 6.9 and 6.10. Figure 6.9 contains trajectories from the beginning of the experiment while figure 6.10 contains the trajectories of the last runs in the experiment.

6.4.1 Discussion

This scenario provides an indication of what is required to complete the real maze. In the real maze, the ability to control the ball close to the maze edges and obstacles is even more important.

When starting the PID and LWPR-2 controlling algorithms in the middle of the maze, they are able to follow the trajectory fairly well until the ball reaches a part of the trajectory located close to the edge of the maze.

Since the PID and LWPR-2 are not able to handle the parts of the trajectory close to the edges of the maze, these controlling algorithms will probably not be able to successfully complete the real maze. Trials using the PID in the real maze also show strong oscillations with the ball bouncing off the obstacles.

Concerning the LWPR-4, as can be seen in figure 6.8, more runs are required for the RMSOE to stabilize in scenario 3 compared to the other scenarios. During the early runs, the LWPR-4 also has problems close to the edges. The trajectories of a few early runs are shown in figure 6.9.

The initial oscillations close to the edges are due to the PID being used when the LWPR-4 can not make a prediction. As the LWPR-4 gets more training, it learns how to control the ball close to the edges.

A few later runs are shown in figure 6.10. As can be seen, there are no oscillations close to the edges. The trajectories from the later runs are closer to the target trajectory than the trajectories from the early runs. The tendency of the ball not to follow the same trajectory in the forward and reverse runs can to some extent be explained by the trajectory following algorithm used. Also the tendency to cut corners can be explained by this algorithm. Using a smaller search radius should make the ball follow the corners more closely. With enough training data, the LWPR-4 should be able to successfully complete the real maze.
Figure 6.8. Results for scenario 3 for LWPR-4. No other controlling algorithm was able to complete any run. Averaged data.
Figure 6.9. Trajectories from early runs by the LWPR-4 in scenario 3. Cyan lines indicate forward runs, blue lines are used for reverse runs. The dashed black line is the target trajectory.

Figure 6.10. Trajectories from late runs by the LWPR-4 in scenario 3. Cyan lines indicate forward runs, blue lines are used for reverse runs. The dashed black line is the target trajectory.
Results and Discussions

<table>
<thead>
<tr>
<th>Scenario</th>
<th>PID</th>
<th>LWPR-2</th>
<th>LWPR-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scenario 1</td>
<td>6.0 (0.8)</td>
<td>3.5 (0.5)</td>
<td>3.7 (0.9)</td>
</tr>
<tr>
<td>Scenario 2a</td>
<td>15.5 (1.6)</td>
<td>11.5 (1.8)</td>
<td>6.5 (1.6)</td>
</tr>
<tr>
<td>Scenario 2b</td>
<td>x</td>
<td>x</td>
<td>5.6 (1.1)</td>
</tr>
<tr>
<td>Scenario 3</td>
<td>x</td>
<td>x</td>
<td>3.8 (0.6)</td>
</tr>
</tbody>
</table>

Table 6.1. Mean rmsoe for 30 runs in the end of each scenario. The standard deviations are given within parentheses. An x indicates that the current controlling algorithm could not finish that particular scenario.

6.5 Final Results

In this section, all results of the different combinations of controlling algorithms and scenarios are compared.

Table 6.1 contains the mean rmsoe of 30 runs at the end of each experiment. The mean is taken over consecutive runs starting after the rmsoe value has stabilized. For the LWPR-4 and scenario 2b, the interval is chosen not to include any rmsoe spike. One such spike may be observed close to run 200 in figure 6.4. An x indicates that the current controlling algorithm could not finish the particular scenario.

The PID and LWPR-2 perform approximately equal across the scenarios. The learning system generally performs better than the PID. As expected, these two controlling algorithms handle scenario 1 fairly well. However, these algorithms not using the absolute ball position do not perform well in scenario 2 but were expected to be able to follow the trajectory in scenario 3.

It should be noted that the PID regulator does not represent the best regulator control theory has to offer. Putting some more effort into regulator design, constructing a traditional controlling algorithm performing at least as good as LWPR-2 should be possible. As the current state of the system is estimated and a state space model of the system exist, using a state feedback controller might seem like a reasonable approach.

The LWPR-4 successfully follows the trajectories in all scenarios. The mean deviation is approximately equal for scenarios 1 and 3. The results from scenario 2 are not as low. Notably, the LWPR-4 performs better in scenario 2b than in scenario 2a. This might result from the border between the normal and modified area being easier to find in scenario 2b as the difference in system behavior is larger.

Note also the spike in figure 6.4. This type of spikes has not been observed when running scenario 2a. This is due to the ability of the PID to handle scenario 2a better than 2b. If the LWPR-4 can not make a prediction, the PID has a higher probability of making a good decision in scenario 2a.

In scenario 1, the LWPR-2 perform slightly better than the LWPR-4. Compared to the variation between different runs with the same controlling algorithm, this difference between the controlling algorithms is very small.
Conclusions and Future Work

In this chapter, conclusions regarding the evaluation platform and the evaluated controlling algorithms are presented. Suggestions of future work are also presented.

7.1 The Platform

An evaluation platform, as described in the introduction, has successfully been constructed. The platform has been used to evaluate a few different controlling algorithms.

Using off the shelf components, the platform hardware has been created. Together with the software, the platform has proven to provide a challenging task for the controlling algorithms being evaluated. The fact that some of the controlling algorithms, both traditional and learning systems, also manage to successfully control the labyrinth and navigate the target trajectories, proves that the platform may be used to evaluate such controlling algorithms.

Estimating a homography relating maze plane coordinates to image coordinates for every frame enables the camera to be moved almost arbitrarily relative to the maze. The maze naturally has to be visible in the camera image. The maze with obstacles not being completely planar poses some additional constraints on the camera movement. These constraints may be loosened using a background model capable of capturing the obstacles appearing to move.
Some experiments have been performed in the real maze. An automated way of placing the ball back in the maze after falling through a hole is needed for more extensive experiments. The platform currently uses a manual ball resetting system. An automatic system, using a Lynx-6 robotic arm [7], is scheduled as future work.

7.2 The Evaluated Controlling Algorithms

Both LWPR based controlling algorithms outperform the PID in all scenarios. From this, two conclusions may be drawn. First, it should be possible to design a much better traditional controller. Secondly, by learning from their own actions, the learning systems are able to perform better than the controlling algorithm used to provide initial training data.

The LWPR-4 requires more training data than LWPR-2. According to the authors of [11], this should not necessarily be the case. However, depending on the initial size of the local models, more local models are needed to fill the input space when more inputs are used.

Using more inputs to the LWPR also increase the risk of the game state ending up where there is no local model yet. If the controlling algorithm used to provide training data can not handle the current state, events seen in scenario 2b may occur. This type of controlling algorithm can not be recommended for critical systems. Although, LWPR-4 is the only evaluated controlling algorithm able to complete scenario 2b.

The real maze contains some more challenges than just the obstacles. As an example, the real maze contains dents deep enough to affect the ball. Of the evaluated controlling algorithms, the LWPR-4 is the most likely to be able to navigate the real maze.

All evaluated controlling algorithms assume that the labyrinth dynamics may be divided into two one-dimensional problems. Using more inputs to the LWPR, any dependencies between the two directions may be handled. Such dependencies are expected to be more apparent in the real maze.
7.3 Future Work

In this section, some ideas of future work are presented. These include both improvements of the platform and ideas of future experiments possible to conduct using the developed platform.

- Better background model. Currently, the system uses the approximate median background model. This model have some trouble close to the obstacles in the maze when the maze is moving. Ideas include Gaussian mixture models, or a computationally lighter model based on approximate mean and variance.

- Automatic resetting of the ball. Hardware and ideas how this could be accomplished exist. The work include modifying a small robotic arm to enable it to pick up the ball and adding a control program to the LEAP software.

- Evaluations using the real maze. This should be easy to perform when the automatic ball resetting is available.

- Evaluation of other learning systems than LWPR.

- Evaluating better traditional control algorithms. Ideas include state feedback controllers as a state space model of the system is already available and the system state is estimated.

- Multivariate controllers. All currently evaluated control algorithms use dual univariate controllers.

- Computer aided manual control. Using the joystick input as velocity reference to the controlling algorithm. This computer aided control may be compared with controlling the game manually.
Bibliography


The LEAP system is designed to be highly modular and to run online. New types of image sources and controlling algorithms can easily be added. To facilitate this, a few issues regarding the implementation had to be solved.

In this appendix, a detailed description of the software is presented. It is intended for the reader interested in implementation details or as an introduction to the source code. For an understanding of the primary part of this thesis, the software description in the system overview (chapter 2) is sufficient.

The software is divided into three parts, the LEAP software, the user interface and the record viewer. A schematic view of the software is shown in figure A.1.
A.1 The LEAP Software

The purpose of the LEAP software is to acquire images, extract state information from these images, generate a control signal, apply the control signal and finally store all useful information in a file. These tasks are performed by different subsystems. There are three subsystems, the image source subsystem, the controller subsystem and the data logging subsystem.

In this section a short description of the tasks of the different subsystems is followed by a general description of some of the features of the LEAP software. The end of this section contains a more detailed description of each of the subsystems. There is also a subsection describing the interface to the LEAP software.

Some of the common data structures and how they are passed around in the system is described quite early in this section. The reason is that an understanding of the main data flows in the system constitutes a large part of understanding the whole system.

The image source subsystem takes care of acquiring images and estimating the state of the system. Storing of data is taken care of by the data logging subsystem. The term useful information mentioned earlier includes the camera images, game state information and the applied control signal. The data logging subsystem also handles information messages, sent by different parts of the system, intended to be read by the user.

The last subsystem, the controller subsystem, is responsible for calculating a control signal that is expected to take the system from the current state to a desired state. This desired state is dependent on the experiment conducted but usually regard the position and velocity of the ball as to make the ball follow a given trajectory. The controlling algorithm is the central part of the controller subsystem.

A.1.1 Multithreading

Each of the three subsystems run in their own thread. This is beneficial as the subsystems often are involved in lengthy operations. Usually these operations do not require much processing power and multithreading allows other work to be done in the other subsystems. One example of this kind of operation is the image source subsystem waiting for the camera to capture the next image.

In the LEAP system, multithreading is necessary when writing frame data to a harddrive. If the operating system fills a block while writing a file, and the following block on the harddrive is not free, it may take a very long time to find a block where writing could be continued. If the LEAP system would have to wait for the file operation to finish before processing the next frame, the system would not be able to run in real time. One purpose of the data logging subsystem is to handle writing of frame data to disk in a way that makes it possible for the rest of the system to process new frames while old frames are written to disk.
A.1 The Leap Software

Figure A.1. Schematic overview of the software.

RV  Record Viewer
UI  User Interface
LI  LEAP Interface
ISS Image Source Subsystem
CS  Controller Subsystem
DLS Data Logging Subsystem
The downside of multithreading concerns thread synchronization and data integrity issues. An example of this is if one thread start copying a frame from the camera to primary memory and does not finish before another thread start reading the frame data from primary memory. In this case, the second thread would read corrupted frame data.

The problem above can be solved by using *mutual exclusion*, or *mutex*. This means that only one thread at a time will be allowed to access each shared piece of data. If another thread is already accessing the data, the thread asking for access will have to wait. Mutual exclusion can be achieved by letting each thread lock and release\(^1\) special signaling objects. These objects implementing mutex are also called *mutaxes*.

### A.1.2 The Frame

A *frame* contains a captured image, the rectification of that image and all data relating to the image. This related data is collected in a *frame descriptor*.

The *frame descriptor* is the central part of each frame. The descriptor contains pointers to the image and the rectified image, the size of the images as well as data estimated by the vision routines. These data contain among other things the ball position and the homography relating the image coordinate system of the current image to the maze plane coordinate system. After the frame has been processed by the controller subsystem, the applied control signal is also saved in the frame descriptor.

### A.1.3 The Frame Cycle

The frame cycle describes how and when frame data is used in the system. A schematic view of the frame cycle is presented in figure A.2.

To avoid repeated allocation and release of memory, a buffer capable of holding a specified number of frames is allocated when the system starts. The allocated memory is reused while the system is running and finally released when the system shuts down.

The image source subsystem keeps track of this frame buffer. Lists are used to store what positions in the buffer that contain valid frame data and what positions that may be used to store new frames.

The frame cycle start in the image source subsystem when a new image is acquired from the camera. The image together with the frame descriptor is placed in an empty slot in the frame buffer. The vision routines are applied to the image and information regarding the homography, the rectified image and ball position is saved in the frame descriptor.

\(^1\) *Resource Acquisition Is Initialization*, RAII, is a useful way of locking and releasing mutexes as well as other resources in C++. See [9].
Figure A.2. Schematic view of the frame cycle.
A reference to the frame is sent to the controller subsystem. The controller subsystem uses the active control algorithm to calculate a control signal that is sent to the servos. When the control signal has been applied, the frame is passed on to the data logging subsystem.

In the data logging subsystem, the frame is put in a queue of frames waiting to be written to file. The data logging subsystem writes frames to file in the order they were put in the queue. When the data in a frame has been written, the frame is returned to the image source subsystem.

In the image source subsystem, the slot in the frame buffer used by the returned frame is marked as ready to receive a new image from the camera.

### A.1.4 The LEAP Interface

In this section, the interface to the LEAP software is presented. Note that this is not the user interface, this is the interface the user interface uses to communicate with the LEAP software.

The outer interface to the LEAP software is divided into two parts. The first part defines the interface while the other part implements the interface. This is necessary to fully hide the implementation details of the LEAP software from the user interface.

The LEAP interface manages starting and stopping of the system, making sure that the subsystems are started and stopped in the correct order. When the system is running, this interface handling object mainly passes calls on to the correct subsystem.

### A.1.5 The Data Logging Subsystem

The data logging subsystem perform four tasks. These are maintaining the frame queue, writing frames to file, maintaining the message queue and timing different operations in the system.

The frame writing feature may be turned off. If this feature is deactivated, frames will be returned to the image source subsystem directly.

The queues are protected by two mutexes, one for each queue. This prevents data corruption while it allows one thread to access one queue while another thread accesses the other queue.

#### A.1.5.1 The Message Queue

The message queue is a FIFO\(^2\) buffer for messages with information, warnings or errors intended to be read by the user. Other parts of the LEAP software post

\(^2\text{First In First Out}\)
messages to this queue and the user interface retrieves them. When a message is retrieved, it is removed from the queue. In addition to a message descriptor in clear text, each messages may be supplied with a message level and what frame the message is related to, if applicable.

A.1.5.2 The Frame Queue and Frame Writing

When a frame has been processed by the controller subsystem, the frame is put into the frame queue. Here the frames wait to be written to the data log file. This queue structure is needed to avoid slowing down the whole LEAP system if the hard drive writing starts lagging behind. The data logger contains its own thread taking care of writing data to the log file. Thus, this can be done in parallel with other tasks of the system. When a frame and its descriptor has been copied to the output buffer, the frame is returned to the image source to be filled with new data. If there is no data to be written, the frame writer thread is put to sleep waiting for a signal indicating that new frames has been put in the queue.

To avoid many short write operations to the hard drive, the output is explicitly buffered by copying data destined for the file to a reserved location in primary memory. When the pending data log operation does not fit in the remaining buffer, the whole buffer is written to the file at once. The buffer is also flushed when the data log file is to be closed.

The image source subsystem may set a flag in the frame descriptor indicating that the number of free slots in the frame buffer is low. If this flag is set, writing frames to file is temporarily stopped to catch up. Some frames are dropped, that is, returned to the image source without writing the data contained in the frame to the data log file. The data logger starts dropping frames directly when this flag is seen. If there already are some frames in the queue when a frame with this flag set is added, the already present frames are dropped. The dropping of frames continue until a frame descriptor, with the buffer warning flag cleared, is added to the queue.

The startup sequence requires the data logging subsystem to be initialized before the image source subsystem. The data logging subsystem must be provided with a reference to the image source subsystem to be able to return frames. If the frames can not be returned, the image source would run out of empty slots.

A.1.5.3 The Timing Module

The last task of this subsystem is to measure the time spent in certain processing steps for each frame. The subsystem does this by creating timestamps at different positions in the frame cycle and calculating the elapsed wall clock time between

---

3Wall clock time may different from e.g. the utilized processor time in a multithreaded environment.
pairs of timestamps. The timing module uses operating system dependent precision timers. The resolution of these timers are dependent on the hardware but it is usually the same as the clock frequency of the processor.

A.1.6 The Image Source Subsystem

The image source subsystem provides images to the LEAP system and process these images to extract information. The subsystem consists of two main parts, the camera module and the vision routines. The image source subsystem is designed to accept different types of image sources. Currently one source, based on an ieee1394 camera, is implemented.

The camera module is responsible for getting images to the LEAP system as well as handling the frame buffer. The vision routines are used to extract information from these images.

The image source subsystem expect the other two subsystems to be running when the image source starts. This ensures that the image source can post messages regarding camera initialization when starting and that the remainder of the system is ready to accept frames when the image source starts to acquire images.

A.1.6.1 The Camera Module

The primary objective of the camera module is to provide images to the LEAP system. For this, a camera is used. To communicate with the camera driver, the module uses the FireGrab API from Allied Vision Technologies, [1].

The secondary objective is to handle the frame buffer. As the frame buffer handling algorithm very much resembles the algorithm employed by the camera API and driver, the camera frame buffer handling will be briefly explained.

Camera Driver Frame Handling

The camera driver internally allocates memory to hold a select number of images. When an image arrives from the camera, it is put in a free image buffer. When requested by the camera module, the driver return a pointer to the next filled image buffer not already fetched, together with some information about the image. Note that the image data is not copied and the memory pointed to is allocated by the camera driver.

For the camera driver to know when it is possible to use an old buffer to hold a new image, the camera driver has to be told when the program has finished using the old image in question. This is called returning a frame although in reality, only some data indicating the identity of the image is sent to the driver.

The images pointed to by the pointers fetched from the camera driver are guaranteed not to be changed by the said driver until the corresponding frames are
returned. If the camera captures an image and the driver has no free buffer, the image will be lost.

**Camera Module Frame Handling**

The same system is employed by the image source. The image source allocates memory for a fixed number of frames. When creating the camera module, the number of slots\(^4\) in the frame buffer may be selected. The buffer is allocated when initializing and then reused as the program is running. Each slot in the buffer actually consist of two parts, one part is used by the camera API and the other by the LEAP system.

To handle the buffer, three queues exist. The first queue, the *free queue*, contains a list of free slots that may be used to store images and image information in. When a frame is filled and sent to be used by the system, the frame is removed from the *free queue* and put in the *active queue*. After being returned from the system, the frame is moved to the *returned queue*. Finally, when the camera driver is not busy, the image buffers used to store the images belonging to the frames in the *returned queue*, are returned to the camera driver. The corresponding frames are then moved to the *free queue*.

When shutting down the LEAP system, the camera module stops producing frames and waits for all frames to be returned.

### A.1.6.2 The Vision Routines

The purpose of the vision routines is to extract information regarding the ball position from the images. This is done by detecting the maze plane, estimating the maze plane homography, maintaining a background model and using the background model to find the ball.

**Maze Plane Detection**

Maze plane detection is based on four colored markers in the maze and the algorithm described in section 3.2.3. By detecting these markers in the image and by knowing their respective position in the maze plane coordinate system, a homography relating image coordinates and maze plane coordinates can be estimated.

The marker detection is based on color. This is simple to implement as the images are retrieved in the *YUV* format, where the *Y* channel contains luminance information and the *UV* channels contain color information. The color distance can be measured as the euclidean distance in the *UV* plane.

To be able to estimate the orientation of the maze without ambiguity, knowing which marker that corresponds to which known position is necessary. To get

\(^4\)Each slot is large enough to fit one frame.
this information, one marker has a different color compared to the other three. Additionally there are two assumptions regarding the position of the camera:

- The maze is not viewed from below.
- The maze is not viewed through a mirror.

Now the marker identity ambiguity can be resolved by means of a direction from centroid sort. First the centroid of the four detected markers in the image is calculated together with the direction from the centroid to each marker. The marker detections are then sorted in a clockwise order starting with the odd colored marker. When the positions and identities of the four markers are known, the homography can be estimated.

Some data is preserved between runs of the system. This data include marker color and last known position. If data from any previous run can not be found, the vision routines are reinitialized.

**Background Model and Ball Detection**

Using the estimated homography, the luminance channel of the image is transformed to the maze coordinate system. This rectified image stays the same when the maze is tilted by the servos, or for that matter, when the whole game is moved by means of manual interference.

From the rectified images, an approximate median background model is created. As the ball might remain stationary for a relatively long time, the background model update speed parameter has to be set relatively low. This makes the model react slowly to changes. To facilitate faster initialization, the background model is set to the first image captured and rectified after the program starts or after the marker positions are reinitialized.

When searching for the ball position, the difference between the background model and the current rectified image is calculated. After low pass filtering the difference image, the global maximum is considered a tentative ball position. By comparing the global maximum with the maximum values of smaller regions scattered around the image, a certainty measure for ball detection is created. If the certainty is high enough, the ball is considered detected at the tentative ball position.

**A.1.7 The Controller Subsystem**

The purpose of the controller subsystem is to use the ball information gathered by the image source subsystem to control the servos. To carry out its task, the controller subsystem contains a state estimation filter, a control signal output module and several controlling algorithms.

The controlling algorithms use the estimated state of the system to calculate a control signal. To handle all controlling algorithms, the controller subsystem em-
A.1 The Leap Software

employs a controller container. The controlling algorithm currently controlling the game is referred to as the active controlling algorithm.

A.1.7.1 The State Estimation Filter

The raw ball position detections are filtered using two Kalman filters. Each of the filters operate along one of the dimensions in the maze plane.

The filters use measured ball positions, the recorded servo outputs and the hardware models described in section 4.4. By fusing the information, more accurate position information as well as ball speed estimations can be produced. The filters also estimate the horizontal maze offsets, that is, the control signal required to orient the maze horizontally.

The reason this is done in the controller subsystem and not in the image source subsystem is that the filters use the applied control signal. This information is not available in the image source subsystem.

A.1.7.2 The Controller Container

The purpose of the controller container is to handle all loaded controlling algorithms. Tasks such as creating new instances of controlling algorithms, loading and saving of controlling algorithms and setting the active controlling algorithm can be performed.

The controller container takes handles all thread and data integrity issues for the controlling algorithms. All activations are made via the controller container which serializes the requests. This makes it easier to implement new controlling algorithms for use within the LEAP system.

The controller container maintain its own thread. This thread may be used by the active controlling algorithm to perform work while waiting for new frames to arrive.

A.1.7.3 Controlling Algorithms

The controlling algorithms are used to control the game. All of them have a common interface toward the controller container. Thus, it is easy to add new controlling algorithms to the system. In this section, the currently implemented controlling algorithms are briefly presented.

Manual Controllers

Two manual controllers are implemented. The first manual controller forwards the manual control signal directly to the servos. The second employs a so called moonlander style of control. This controller quantify the manual control signal
into three levels: forward, zero or reverse. The controller also integrates the signal before sending it to the servos. This means that the joystick is used to control the angular velocity of the maze instead of the angular position.

The PID Controller

The PID controller utilizes the ball velocity information available from the state estimation filter. Two controllers are used, one in each direction of the maze. A target ball position is used as reference. The integration is approximated by a cumulative sum of the control error. The $p$, $i$ and $d$ parameters of the controller may be changed while the controller is active.

The Learning System Controller

This is the core of the poodle.\footnote{Pudelns kärna (Swedish).} The learning system controller implement a controller based on LWPR. A target ball position is available as a reference to the system. Depending on the experimental setup, some calculations are performed to generate a desired state for the system. One example is to ask the system to maintain a constant speed towards the target ball position. For learning the mapping, code written by the authors of [11] is used.

A.1.7.4 The Control Signal Output Module

Sending the control signal to the servo controller card is handled by a serial communications module. The module is divided into two parts of which one defines the interface toward the rest of the LEAP system while the other implements the functions defined in the interface. Accessing the serial port is dependent on the operating system. By dividing the module, prototypes for the operating system calls are hidden from the LEAP system.

A.2 The User Interface

The purpose of the user interface is to allow the user to interact with the LEAP system. Graphical objects are displayed using the .NET framework. Mainly, the user interface presents the options available in the LEAP software to the user. A few features of the user interface are presented below.

- The current trajectory is fetched from the LEAP system. Using the homography to transform points in maze plane coordinates to image coordinates, the trajectory supposed to be followed by the ball is superimposed on the camera
image display. The path the ball actually has traveled is also displayed in a similar fashion.

- The messages sent to the message queue by the LEAP system are regularly fetched and displayed in a list.

- The interface contains means of manually calibrating the markers.

- When a new frame is captured, the user interface is notified and the image is displayed to the user. One additional image is also displayed, the user has the option of either displaying the rectified image, the background model or the difference between the two.

A.3 The Record Viewer

The record viewer is used to play data files recorded while running the LEAP system. As the data files might be quite large (approximately 15 megabytes per second of recorded data) and since the file format used supports it, the whole file is not read at once, only one frame is read at a time. By reading forward or seeking backward in the input file stream, the sequence may be played either forward or backward.

Using the record viewer, learning controllers may be trained offline. The training is done by presenting the recorded data from the opened data log file as examples to the controller. As only data present in the frame descriptor is needed, lean files may be used for training. Lean files are data log files from which the image data has been removed to reduce file size.

Note that due to limitations in the Microsoft operating system file handling and C++ interface, reverse play and stepping backward is not possible after playing past the first two gigabytes of data in a record file. Forward play, forward stepping and stop will continue to work.