Self-Playing Labyrinth Game Using Camera and Industrial Control System

Daniel Persson

Fredrik Wadman

LUND UNIVERSITY

Department of Automatic Control

Msc Thesis
ISRN LUTFD2/TFRT--5948--SE
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Media-Tryck
Lund 2014

Abstract

In this master's thesis, an industrial control system together with a network camera and servo motors were used to automate a ball and plate labyrinth system. The two servo motors, each with its own servo drive, were connected by joint arms to the plate resting on two interconnected gimbal frames, one for each axis. A background subtraction-based ball position tracking algorithm was developed to measure the ball-position using the camera. The camera acted as a sensor node in a control network with a programmable logical controller used together with the servo drives to implement a cascaded PID control loop to control the ball position. The ball reference position could either be controlled with user input from a tablet device, or automatically to make the labyrinth self-playing.

The resulting system was able to control the ball position through the labyrinth using the camera for position feedback.

Keywords ball and plate, computer vision, background subtraction, PID, object tracking

Acknowledgments

We would like to thank our supervisors Alfred Theorin and Anton Cervin for their support during the work on this master's thesis. Special thanks to Anton for formulating the task and giving us the opportunity to do it and to Alfred for sticking with us during the spring.

We would also like to express our gratitude towards Thomas Montgomery and the people at Alten for their encouragement and appreciation of our work during the many hours spent at the office; especially Jan Lohman, who implemented the original labyrinth system and helped us a lot with understanding it and the tools used to configure it.

Further, we would also like to thank Johan Leveau at ABB for supplying equipment, especially the gear heads that substantially helped to improve the performance of the system.

Contents

List of Figures

List of Tables

1

Introduction

Alten has developed a large (almost 1×1 m) labyrinth game, where the challenge consists of navigating a ball through a labyrinth equipped with an industrial control system from ABB, while avoiding obstacles in the forms of holes (see Figure 1.1). The control system is hidden inside the labyrinth and consists of a *programmable logic controller* (PLC), two servo motors with drives, an inclinometer and a WLAN router for external communication. The user commands are in the form of desired game board angles (i.e. setpoints) that are sent from an Android tablet application.

The aim of this master's thesis is to develop a fully automatic, self-playing labyrinth game based on computer vision feedback. The position of the ball should be estimated using an Axis camera mounted above the game board. The first step is to implement a mode where user commands from the tablet are interpreted as ball position setpoints rather than angle setpoints, and where the actual ball position is regulated by the system. In the next step, the ball should follow a pre-specified path from start to finish on the board.

A key question to investigate in the thesis is how to best use computer vision to implement a ball tracking algorithm, as well as how to deal with the timing, synchronization and communication between the various components of the system in order to achieve short cycle times and delays. The PLC, the camera and the tablet run different operating systems and are all programmed in different languages, which represents another challenge.

1.1 Goal

The goal of the project can be broken down into some intermediate goals:

- Design and implement a ball tracking algorithm for the camera.
- Implement a distributed control system managed by the PLC.
- Implement an Android client application.
- Optimize and tune the system to allow the labyrinth to be self-playing.

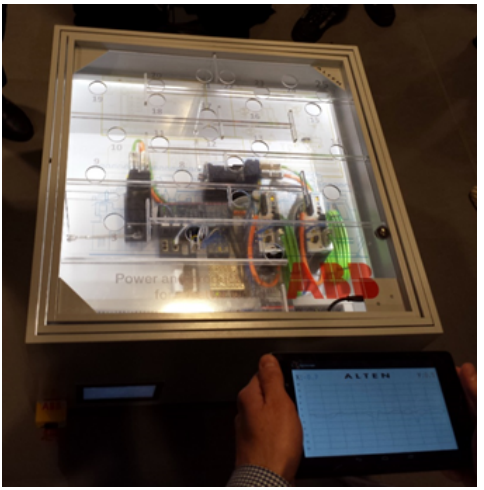


Figure 1.1 The labyrinth game developed by Alten

Completing these goals will achieve the specified task.

1.2 Individual Contributions

The work on this master's thesis has had a big focus on implementation of the different components: the camera, PLC and Android applications. The first objective of the work was to find a good algorithm for the ball tracking. This led to the investigation and testing of the methods described in Section 2.2 which was done together. Daniel's previous experience and knowledge of C programming proved valuable when implementing the camera application. Daniel also stands for the PLC implementation. Fredrik has led the work on the Android application. The final tuning of the system was done together.

1.3 Outline

This thesis is organized as follows: Chapter 2 contains background information about related work as well as some condensed information on subjects such as computer vision and tracking, i.e. filtering, transforms and background subtraction algorithms. The background chapter also contains a section on relevant control theory used in this thesis. Chapter 3 describes all different hardware components used for the system. It does also contain information on related software used together with the hardware components. Chapter 4 is the largest chapter and covers the implementation of all different system parts from system design to actual software. In Chapter 5,

resulting data presenting the performance of the completed system can be found, and in Chapter 6, the implications of these results are discussed. The last chapter, Chapter 7, draws conclusions from the observed results and the previous discussion. This chapter also includes possibilities for future work that could be done to improve or increase the functionality of the labyrinth game system.

2

Background

2.1 Ball and Plate

2.1.1 Introduction

The labyrinth system is essentially a ball and plate process which in turn is a generalization of the ball and beam process. In the ball and beam process the ball is moving in one dimension back and forth on a beam, whereas in the ball and plate process, the ball moves in two dimensions. The step between these two is not very large, but makes the recognition of the ball location using a camera a bit more difficult. Both the beam and plate based processes are well known in the automatic control society and are widely used for educational purposes.

2.1.2 Related Work

Many different implementations of both ball and plate and ball and beam processes have been made utilizing different kinds of ball tracking methods and controlling setups. Here follows a sample of related work.

Ball on Plate Balancing System by Andrews, Colasuonno and Herrmann
In [Andrews et al., 2004] the process of designing, developing and controlling a ball on plate system is presented. The ball position is measured using the resistive touch screen which act as plate and controlled by a full state-feedback controller. One of the things that distinguishes this report is that all components used are kept track of and a price of the system is estimated.

LEAP, A Platform for Evaluation of Control Algorithms by Öfjäll
In [Öfjäll, 2010] the original BRIO labyrinth game is used as the hardware. The goal is to develop an evaluation platform for control algorithms, hence the name LEAP, (*Labyrinth Evaluation of Algorithms Platform*). The control algorithms used are classical PID and also a learning algorithm, *Locally Weighted Projection Regression*. The ball position is tracked using a camera feeding images to a computer that runs an *approximate median background model*. The computer also runs software to send control signals to the servos controlling the plate angles and receive to user input.

Dynamic Ball and Plate Control verification on Magnetic Suspension Platform Using Enforced Fuzzy Logic Control by Lin and Huang
The paper presented by Lin and Huang, [Lin and Huang, 2012], presents a ball on plate system consisting of a touch panel connected in the center to a pedestal that can tilt its top using magnetic suspension. The ball position is tracked using the resistive touch panel and controlled using *enforced fuzzy logic*.

Vision Algorithms for Ball on Beam and Plate by Espersson
In Esperssons' master's thesis report, [Espersson, 2013], a computer vision tracking system is introduced for tracking a ball on a beam and also on a plate. The algorithm uses a *Hough line transform* to locate the beam using a USB webcamera. The ball is located using Hough circle transform that detects the edges of the ball. This is sufficient as both the beam and plate are monochromatic. Using special markers in the ends of the beam and corners of the plate the angles can also be measured. The control is executed using cascaded PID controllers.

Vision Based Balancing Tasks for the iCub Platform by Levinson, Silver and Wendt
In [*Vision Based Balancing Tasks for the iCub Platform: A Case Study for Learning External Dynamics*], a case study for controlling a ball and beam system and a ball and plate system using a humanoid robot is described. The robot holds the beam or plate in its hand and uses one of its eye-cameras, centered on the process, as vision input. The computer vision is done by color channel separation which requires some different colored markers and a colored ball for detection of the different components. The control is done by a state feedback controller with integral action.

2.1.3 System Model

The ball and plate system is easily broken down to its one-dimensional sibling, the ball and beam process, from which the full system can be described by extending the model to two dimensions. In Figure 2.1 the process is visualized, where the loose arm represents the connection with the motor shaft with angle α, and θ is the angle of the beam. Letting J be the moment of inertia of the ball, and R be the ball radius, the dynamics of the ball in one direction are derived as

$$0 = \left(\frac{J}{R^2} + m\right)\ddot{r} + mg\sin\theta - mr\dot{\theta}^2,$$

which when linearized around $\theta = 0$ gives

$$\left(\frac{J}{R^2} + m\right)\ddot{r} = -mg\theta.$$

The angle of the beam is related to the motor shaft connection angle α as $\theta = \frac{d}{l}\alpha$. Substituting this in the equation above gives

$$\left(\frac{J}{R^2} + m\right)\ddot{r} = -mg\frac{d}{l}\alpha.$$

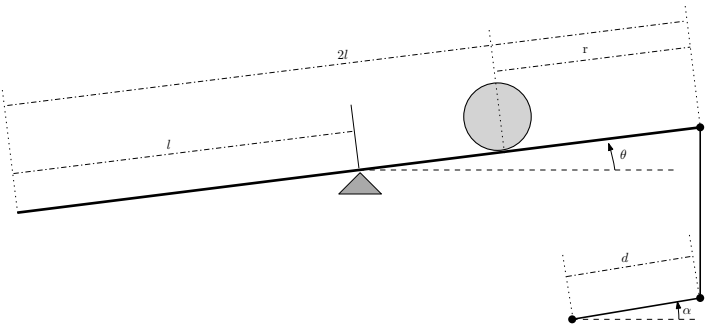


Figure 2.1 Beam system

By taking the Laplace transform of this and rearranging it, the transfer function is found as

$$P(s) = \frac{R(s)}{\alpha(s)} = -\frac{mgd}{l\left(\frac{J}{R^2}+m\right)}\frac{1}{s^2},$$

which in state space form will look like

$$\begin{bmatrix}\dot{r}\\ \ddot{r}\end{bmatrix} = \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix}\begin{bmatrix}r\\ \dot{r}\end{bmatrix} + \begin{bmatrix}0\\ \frac{-mgd}{l(\frac{J}{R^2}+m)}\end{bmatrix}\alpha,$$

$$y = \begin{bmatrix}1 & 0\end{bmatrix}\begin{bmatrix}r\\ \dot{r}\end{bmatrix}.$$

where y is the output. From this one can see that the ball dynamics is a double integrator process which with sampling time h and using zero-order-hold sampling translates to the discrete system

$$\mathbf{r}(t_{k+1}) = \begin{bmatrix}1 & h\\ 0 & 1\end{bmatrix}\mathbf{r}(t_k) + \begin{bmatrix}\frac{h^2}{2}\\ h\end{bmatrix}\left(\frac{-mgd}{l(\frac{J}{R^2}+m)}\right)\alpha(t_k).$$

By replacing r with x and y as $\mathbf{r} = \begin{bmatrix}x & y & \dot{x} & \dot{y}\end{bmatrix}^T$ and letting α be the motor shaft angle in the x-direction and β the angle in the y-direction, the system is expanded to two dimensions. Assuming that the plate is quadratic, and that the motor shaft arms are of equal lengths, one gets the following discrete system:

$$\mathbf{r}(t_{k+1}) = \begin{bmatrix}1 & 0 & h & 0\\ 0 & 1 & 0 & h\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\mathbf{r}(t_k) + \left(\begin{bmatrix}\frac{h^2}{2}\\ 0\\ h\\ 0\end{bmatrix}\alpha(t_k) + \begin{bmatrix}0\\ \frac{h^2}{2}\\ 0\\ h\end{bmatrix}\beta(t_k)\right)\left(\frac{-mgd}{l(\frac{J}{R^2}+m)}\right).$$

2.2 Computer Vision

2.2.1 Object Detection and Tracking

One large area in computer vision is object detection and tracking. A lot of focus has been put on detection of humans, and especially face recognition, which is now commonly found in cameras for automatic focusing and also in other applications. There are a few different techniques for finding and tracking objects in a picture or video, a few of them are given a brief description here.

Kalman Filtering The *Kalman filter*, first proposed by Rudolf E. Kalman in [Kalman, 1960], has been used in computer vision tracking since the mid 1980s [Szeliski, 2011]. There have been many different implementations done over the years spanning both regular linear systems and some extended Kalman filters for tracking more complex trajectories. Here follows a short description of the Kalman filter. For more extensive reading see [Glad and Ljung, 2003] and [Cuevas et al., 2005].

A discrete, linear dynamical system can be described mathematically by a state space model with a state or state vector $\mathbf{x}_k$, observed states $\mathbf{y}_k$, control input $\mathbf{u}_k$ and matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ on the form

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}\mathbf{x}_{k-1} + \mathbf{B}\mathbf{u}_k + \mathbf{w}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{v}_k, \end{aligned}$$

where $\mathbf{w}_k$ is the process noise and $\mathbf{v}_k$ is the measurement noise. Here they are both assumed to be additive, white and Gaussian noise with zero mean and covariance matrices $\mathbf{Q}_k$ for $\mathbf{w}_k$ and $\mathbf{R}_k$ for $\mathbf{v}_k$. The measurement noise is assumed to be uncorrelated with the process noise, i.e. $E[\mathbf{v}_n \mathbf{w}_n^T] = 0$. In vision tracking there is no control input; hence, $\mathbf{u}_k \equiv 0$ and the matrices $\mathbf{B}$ and $\mathbf{D}$ are irrelevant.

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, $\tilde{\mathbf{x}}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k$. The estimated state $\hat{\mathbf{x}}_{k|k-1}$ is calculated using the covariance information $\mathbf{Q}_k$, $\mathbf{R}_k$ and the state estimate covariance $\mathbf{P}_{k|k-1}$ for weighting. The subscript $n|k$ denotes the estimation at time n given system updates until time k.

The next state of the system is estimated as

$$\begin{aligned} \hat{\mathbf{x}}_{k+1|k} &= \mathbf{A}\hat{\mathbf{x}}_{k|k} \\ \mathbf{P}_{k+1|k} &= \mathbf{A}\mathbf{P}_{k|k}\mathbf{A}^T + \mathbf{Q}_k. \end{aligned}$$

When the measurement at $k+1$ is available the prediction is updated as

$$\begin{aligned} \hat{\mathbf{x}}_{k+1|k+1} &= \hat{\mathbf{x}}_{k+1|k} + \mathbf{K}_{k+1}\left(\mathbf{y}_{k+1} - \mathbf{C}\cdot\hat{\mathbf{x}}_{k+1|k}\right) \\ \mathbf{P}_{k+1|k+1} &= \mathbf{P}_{k+1|k} - \mathbf{K}_{k+1}\mathbf{C}\mathbf{P}_{k+1|k}, \end{aligned}$$

where $\mathbf{K}$ is the Kalman gain, which in turn is given by

$$\mathbf{K} = \mathbf{P}_{k+1|k}\mathbf{C}^T \left(\mathbf{C}\mathbf{P}_{k+1|k}\mathbf{C}^T + \mathbf{R}_k\right)^{-1}.$$

2.2.2 Filtering

Low-Pass Filtering *Low-pass filters* are used to attenuate high-frequency signals while allowing lower frequencies to pass through. Low-pass filters are therefore useful to remove the effects of high-frequency noise on signals. A simple example of a low-pass filter is an exponentially weighted smoothing of an input signal x_t at time t. The filtered output signal y_t is given by

$$y_t = \alpha x_t + (1-\alpha)y_{t-1},$$

where α is a smoothing factor ($0 < \alpha < 1$). The filter output depends only on the current input and the previous output and is therefore efficient to implement.

Gaussian Smoothing *Gaussian smoothing* or *blurring* is a common filtering technique used in image processing to smooth images and thereby reduce high-frequency noise [Szeliski, 2011]. The filtered image is calculated by convolving the image with a matrix produced from a two-dimensional Gaussian function

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

with a specific standard deviation σ.

2.2.3 Image Gradients

An image gradient gives the change of color intensity in an image and is defined as

$$\nabla f(x,y) = \frac{\partial f}{\partial x}\hat{\mathbf{x}} + \frac{\partial f}{\partial y}\hat{\mathbf{y}},$$

and can be calculated using e.g. the one-dimensional finite differential operators $[-1,0,1]$ and $[-1,0,1]^{\mathrm{T}}$ using the surrounding pixels, or more advanced operators such as the *Sobel operator*.

2.2.4 Hough Transform

The *Hough transform* is a well-known technique that uses a voting-system to find line locations in images [Szeliski, 2011]. It can also be used to locate other features, for example, circles. Consider an image f that includes a circle with a known radius r at an unknown location in the image. The image gradient gives a vector-field oriented perpendicular to the edges in the image, and assuming each edge pixel could be part of the circle edge, there are two potential positions

$$f(x,y) + r\frac{\nabla f}{\|\nabla f\|}$$

and

$$f(x,y) - r\frac{\nabla f}{\|\nabla f\|}$$

at a distance r and perpendicular to the edge which could be the circle center. These two positions are therefore voted on by the current pixel. After all pixels have voted, the position with the highest number of votes should be considered the most likely location for the sought after circle.

2.2.5 Binary Morphology

Introduction *Binary morphology* is a mathematical theory for geometrical processing and analysis on binary structures such as binary images or matrices [Szeliski, 2011].

Dilation One common operator is *dilation*. Dilation means that given a binary matrix $\mathbf{A}$ and a structuring element $\mathbf{B}$, $\mathbf{A}$ is dilated so that for any marked pixel in $\mathbf{A}$, the surrounding pixels marked in the structuring element $\mathbf{B}$ oriented on the currently evaluated pixel in $\mathbf{A}$ are also marked, i.e.

$$\mathbf{A} \oplus \mathbf{B} = \bigcup_{b \in \mathbf{B}} \mathbf{A}_b,$$

or the union of the positive translations of $\mathbf{A}$ with the coordinates $b \in \mathbf{B}$.

For example, with

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

and $\mathbf{B}$ having its origin in the center, the dilation is

$$\mathbf{A} \oplus \mathbf{B} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Erosion Another common operator is *erosion*, which instead unmarks pixels in an image $\mathbf{A}$ if they are not covered by the structural element $\mathbf{B}$, i.e.

$$\mathbf{A} \ominus \mathbf{B} = \bigcap_{b \in \mathbf{B}} \mathbf{A}_{-b},$$

or the intersection of the negative translations of $\mathbf{A}$ with the coordinates $b \in \mathbf{B}$.

For example, with

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and $\mathbf{B}$ having its origin in the center, the erosion is

$$\mathbf{A} \ominus \mathbf{B} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

2.2.6 Image Moments

Image moments are used in computer vision to e.g. compute pixel mass centroids, and are defined as a weighted average

$$M_{p,q} = \sum_x \sum_y x^p y^q I(x,y)$$

for a grayscale or binary image I. With $p = 0$ and $q = 0$, $M_{0,0}$ gives the pixel mass or sum of all pixel values for an image. $M_{1,0}$ will give the moment in the first direction, and $M_{0,1}$ will give a moment in the second direction. From mechanics it is known that the center of mass, or centroid, of the image can be calculated as

$$(\bar{x}, \bar{y}) = \left(\frac{M_{1,0}}{M_{0,0}}, \frac{M_{0,1}}{M_{0,0}} \right).$$

2.2.7 Background Subtraction

Introduction *Background subtraction* is a set of techniques used in image processing and computer vision to separate foreground and background in images, i.e. to find areas that are changing over time in image sequences. There are a multitude of different techniques used and this section will briefly give an introduction to some of them. For an introduction on other techniques, as well as performance evaluation, see [Toyama et al., 1999] or [Piccardi, 2004].

Frame Differencing Given that you have knowledge of the exact background composition of a specific image it seems very natural that the foreground of the image can be extracted by removing the known background using a simple difference, e.g. with a image represented as a matrix $\mathbf{I}$ so that $\mathbf{I}(x,y)$ is the color value of the pixel at location (x,y),

$$\mathbf{I}_{\text{foreground}}(x,y) = \left| \mathbf{I}(x,y) - \mathbf{I}_{\text{background}}(x,y) \right|.$$

This does, however, require that background pixels are static and that foreground pixels are non-static.

With a video sequence this can be used to detect movement between frames by simply using the last frame as the background reference for the next frame, e.g.

$$\mathbf{D}_k = |\mathbf{I}_k - \mathbf{I}_{k-1}|$$

will yield the absolute difference or movement $\mathbf{D}_k$ comparing the current frame $\mathbf{I}_k$ to the previous $\mathbf{I}_{k-1}$. This simple approach can unfortunately not be expected to generate too good results as there may be other differences between the frames except what is defined as movement. Changes in lighting and noise are examples, and they create frame differences that generally are uninteresting when trying to detect movement of objects.

Running Average Models From photography it is known that when a picture is taken with a very long exposure time only the background is captured, and the light reflected by passing objects have such a small effect on the resulting image (short fraction of the total exposure time) that they are not visible. The same technique can be applied to extract the background from a video sequence by simply averaging the frames to produce a background. The impact of non-static pixels will then only be small and thus not visible. For example, one could at a current frame use the mean value of the previous frames $\mathbf{I}$ as the background $\mathbf{B}$ by taking

$$\mathbf{B}_{k+1} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{I}_i,$$

or in a recursive manner,

$$\mathbf{B}_{k+1} = \frac{(k-1)\mathbf{B}_k + \mathbf{I}_k}{k},$$

with $\mathbf{B}_1 = \mathbf{0}$, and thus get the foreground $\mathbf{F}_k$ at time k

$$\mathbf{F}_k = |\mathbf{I}_k - \mathbf{B}_k|.$$

The background can, however, not be expected to be constant over longer time intervals. Lighting might, for example, change during the course of a day, and it might be better to model the background as the average of limited number of sequential frames, or by using an adaptive model (or low-pass filter, see Section 2.2.2), where the latest frame is weighted into the model using an adaptation factor α, e.g.

$$\mathbf{B}_{k+1} = \alpha\mathbf{I}_k + (1-\alpha)\mathbf{B}_k.$$

Adaptive Two Part Filter A method using a two part background subtraction is proposed in [Gruenwedel et al., 2013]. The approach is to combine an autoregressive moving average filter with two background models at different adaption speeds. The first filter models the long-term background and the second filter works with a higher adaption rate and models the short-term background.

This approach uses the two different background states to find the current foreground: a short-term background $\mathbf{B}_s$, and a long-term background $\mathbf{B}_l$. From these background states, two binary foreground masks, $\mathbf{F}_s$ and $\mathbf{F}_l$, short-term and long-term respectively, are calculated. The short-term foreground is calculated by simply thresholding the current image frame $\mathbf{I}_k$ against the current short-term background state using the threshold T_l, i.e.

$$\mathbf{F}_{s,k}(x,y) = \begin{cases} 1 & \text{if } \left|\mathbf{I}_k(x,y) - \mathbf{B}_{s,k}(x,y)\right| \geq T_l \\ 0 & \text{otherwise.} \end{cases}$$

To calculate the long-term foreground one starts by thresholding, as in the short-term foreground, but using a higher threshold $T_h \geq T_l$ (in order to avoid smaller changes that are handled by the short-term model) to get

$$\mathbf{A}_k(x,y) = \begin{cases} 1 & \text{if } \left|\mathbf{I}_k(x,y) - \mathbf{B}_{l,k}(x,y)\right| \geq T_h \\ 0 & \text{otherwise.} \end{cases}$$

The long-term foreground should also consider pixels nearby already marked pixels in $\mathbf{A}$ if they are also marked in $\mathbf{F}_s$; thus, by applying a morphological dilation to $\mathbf{A}$ one can get the long-term foreground

$$\mathbf{F}_{l,k} = (\mathbf{A}_k \oplus \mathbf{C}) \circ \mathbf{F}_{s,k}$$

where $\circ$ denotes the *Hadamard product* (element-wise product) and $\mathbf{A} \oplus \mathbf{C}$ denotes the morphological dilation of $\mathbf{A}$ using the structural element

$$\mathbf{C} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix},$$

i.e. a diamond shape. This allows pixels that are marked as foreground pixels in the short-term model to be marked as part of the foreground as long as they are close to pixels detected as foreground pixels in the long-term model.

Now, the detected foreground $\mathbf{F}$ is calculated by combining the short-term- and long-term foregrounds,

$$\mathbf{F}_k = \mathbf{F}_{s,k} \circ \mathbf{F}_{l,k} = \mathbf{F}_{l,k},$$

which results is the long-term foreground since the short-term foreground is used in conjunction with a dilation to mark pixels in the long-term foreground.

In order to apply the same method on the next frame in a video, the background states have to be updated. The short-term background should simply be updated by a linear interpolation (or low-pass filter) using the current frame, i.e.

$$\mathbf{B}_{s,k+1} = \alpha_s \mathbf{I}_k + (1 - \alpha_s)\,\mathbf{B}_{s,k},$$

where $0 < \alpha_s < 1$ is the short-term adaptation rate.

The long-term background should, however, only be updated in areas without foreground activity in order to avoid taking into account temporary variations; thus,

$$\mathbf{B}_{l,k+1} = (\mathbf{1} - \mathbf{F}_k) \circ \alpha_l (\mathbf{I}_k - \mathbf{B}_{l,k}),$$

where $0 < \alpha_l < 1$ is the long-term adaptation rate.

Since the long-term background state should describe the current image background over a longer period of time than the short-term state, the adaption rates should be related as $\alpha_s > \alpha_l$, so that the short-term model is updated faster than the long-term model.

Approximate Median Background Model The *Approximate Median Background Model* method was originally described in [McFarlane and Schofield, 1995] and has also been used in traffic surveillance video tracking [Sen-ching S. Cheung and Kamath, 2007]. The background is modeled using a reference image which is updated with every new frame. Each pixel in the reference image is compared with the corresponding image pixel in the image. If the value of the pixel in the reference image is larger than in the current frame the reference pixel value is decreased, and if the pixel value is greater in the current frame the reference pixel value is increased. The pixel values of the reference image will converge to a value where half of the updating values are larger than, and half less than this value–the median. Using the median instead of the mean creates a better resistance towards outliers; hence, a moving target in the image will not be added to the background as fast using the median as it would using the mean. This means that in order to create a valid reference image, the target must be moving or not present in the initial frames.

The implementation of this algorithm is very simple and thus also very computationally efficient. The notation of the algorithm is found in Table 2.1 and the pseudo code for the algorithm tried in this report in Algorithm 1. The method is widely known and this algorithm structure is found in [Wood, 2007].

$x_t(i)$	pixel value at time *t* for pixel *i*
f_t	frame at time *t*
$m(i)$	median estimate for pixel *i*
α	adaption rate
T	threshold
$\hat{B}$	binary background/foreground image

Table 2.1 Notation for approximate median algorithm

Kalman Filter Background Estimation Kalman filtering is a widely used technique for tracking linear dynamical systems. Many different versions have been proposed for background modeling, where the simplest ones use only the pixel intensity. One of the simpler versions described in [Karmann and Brandt, 1990] uses

Algorithm 1 Approximate Median Filtering

```
for all pixels in f_t do
    if x_t(i) > m(i) then                          ▷ Approximate median
        m(i) ← m(i) + α
    else
        m(i) ← m(i) − α
    end if
    if |x_t(i) − m(i)| ≤ T then                    ▷ Segment background/foreground
        B̂ ← 1
    else
        B̂ ← 0
    end if
end for
```

the intensity and its temporal derivative to create a background model. This method has been described in numerous papers, for example in [Sen-ching S. Cheung and Kamath, 2007], and in [Ridder et al., 1995]. The intensity of a pixel $s(x,y)$ at time t is the system input. The state of the system at time t_i is represented by $\hat{s}(x,y,t_i)$, and $\hat{\tilde{s}}(x,y,t_i)$ is the estimated variance. The estimation is updated as follows:

$$\begin{bmatrix} \hat{s}(x,y,t_i) \\ \hat{\tilde{s}}(x,y,t_i) \end{bmatrix} = \begin{bmatrix} \tilde{s}(x,y,t_i) \\ \tilde{\tilde{s}}(x,y,t_i \end{bmatrix} + \mathbf{K}(x,y,t_i) \left(s(x,y,t_i) - \mathbf{H} \begin{bmatrix} \tilde{s}(x,y,t_i) \\ \tilde{\tilde{s}}(x,y,t_i) \end{bmatrix} \right),$$

where the prediction is calculated as

$$\begin{bmatrix} \tilde{s}(x,y,t_i) \\ \tilde{\tilde{s}}(x,y,t_i) \end{bmatrix} = \mathbf{A} \begin{bmatrix} \hat{s}(x,y,t_{i-1}) \\ \hat{\tilde{s}}(x,y,t_{i-1}) \end{bmatrix}$$

with the constant matrices $\mathbf{A}$ and $\mathbf{H}$:

$$\mathbf{A} = \begin{bmatrix} 1 & 0.7 \\ 0 & 0.7 \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}.$$

The Kalman gain is set to

$$\mathbf{K}(x,y,t_i) = \begin{bmatrix} k_1(x,y,t_i) \\ k_2(x,y,t_i) \end{bmatrix},$$

$$k_1(x,y,t_i) = k_2(x,y,t_i) = \alpha \cdot m(x,y,t_{i-1}) + \beta \cdot \left[1 - m(x,y,t_{i-1})\right],$$

where $m(x,y,t_{i-1})$ is a thresholded binary map of the background/foreground segmentation

$$m(x,y,t_{i-1}) = \begin{cases} 1 & \text{if } d(x,y,t_{i-1}) \geq T_h \\ 0 & \text{else,} \end{cases}$$

$$d(x,y,t_{i-1}) = \left| s(x,y,t_{i-1}) - \hat{s}(x,y,t_{i-1}) \right|,$$

with a constant threshold T_h. The number 1 represents foreground pixels, and the number 0 represents background pixels. If the difference between the estimated background pixel value and the measured value is larger than the threshold, the pixel belongs to the foreground, in which case α is the Kalman gain, and if the pixel belongs to the background, i.e. $d(x,y,t_{i-1}) < T_h$ then β is the gain factor. The threshold has to be chosen in such a way that illumination changes are adapted fast into the estimated background. The parameters α and β can be thought of as different adaption rates where α is a slow adaption rate to keep foreground pixels from blending in to the background and β is a fast adaption rate to suppress background noise from the foreground estimation, i.e. $\alpha < \beta$.

2.3 Control Theory

2.3.1 PID

Introduction The labyrinth control system consists of several control loops based on *PID* control. The control signal $v(t)$ of a PID controller at time t can be mathematically described as

$$v(t) = K\left(e(t) + \frac{1}{T_I}\int^{t} e(\tau)\,\mathrm{d}\tau + T_D\frac{\mathrm{d}}{\mathrm{d}t}e(t)\right),$$

where the error is defined as $e(t) = y_{\mathrm{sp}}(t) - y(t)$, y_{sp} is the setpoint, K is the proportional gain, T_I is the integral time, and T_D is the derivative gain of the controller [Årzén, 2012]. This simple variant can be improved further by limiting the derivative gain, adding *anti-windup* to the integrator, and *setpoint weighting*.

Setpoint Weighting Setpoint weighting is easily implemented on the proportional part $P(t)$ as

$$P(t) = K\Big(\beta y_{\mathrm{sp}}(t) - y(t)\Big) \qquad 0 \le \beta \le 1,$$

where the parameter β is used to weight the setpoint so that the proportional part of the controller only operates on a fraction of the reference signal. It has been empirically shown that using setpoint weighting can lead to smaller overshoots [Årzén, 2012].

Setpoint weighting is also typically used on the derivative part $D(t)$, e.g.

$$D(t) = K\,T_D\frac{\mathrm{d}}{\mathrm{d}t}\Big(\gamma y_{\mathrm{sp}}(t) - y(t)\Big) \qquad 0 \le \gamma \le 1,$$

and the weight γ is most commonly set to zero [Årzén, 2012] so that the derivative part only operates on the measurement signal.

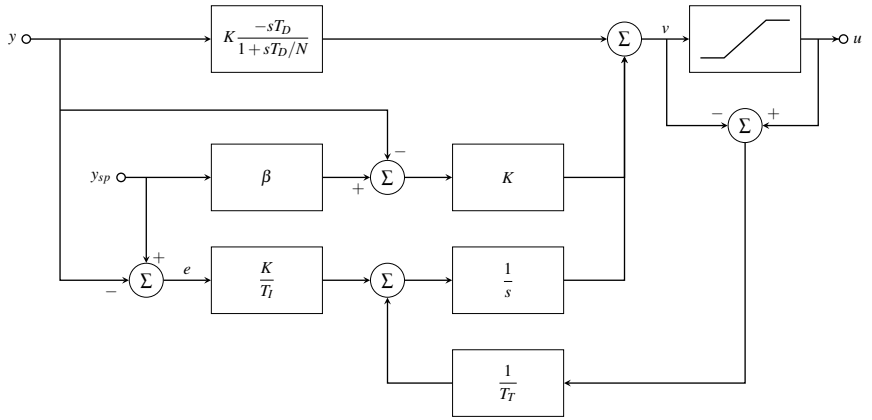


Figure 2.2 Block diagram of a PID controller with setpoint weighting, limited derivative gain and tracking-based anti-windup

Derivative Gain Limit A pure derivative term will result in large amplification of measurement noise [Årzén, 2012]. By approximating the derivative transfer function as

$$sT_D \approx \frac{sT_D}{1+sT_D/N},$$

the high frequency gain is limited by N, which is called *maximum derivate gain*. The purpose of limiting the derivative gain is to avoid the risk of measurement noise amplification in the control signal.

Anti-Windup When using a controller with integral action on a system that can be saturated, the integral action can result in the actuator remaining saturated as the error changes. This is called *integrator windup*, and in the labyrinth system, it is necessary to implement anti-windup such that this unwanted phenomena is avoided. There are several ways of implementing anti-windup. One easy way to avoid windup is to introduce *conditional integration*, i.e. shutting of the integral part when the system is far from steady-state. Another alternative is to use *tracking based anti-windup* by adding a new parameter T_T, tracking time, and change the update formula for the integral part $I(t)$ to

$$I(t) = \int^{t} \frac{K}{T_I} e(\tau) + \frac{1}{T_T}\Big(u(\tau) - v(\tau)\Big)\,\mathrm{d}\tau,$$

where $u(t) = \mathrm{sat}(v(t))$ is the saturated control signal.

A block diagram showing the structure of a PID controller with setpoint weighting ($\gamma = 0$), maximum derivative gain N and tracking-based anti-windup is shown in Figure 2.2.

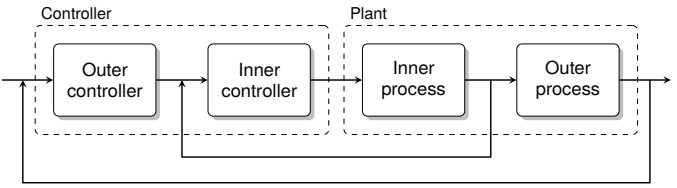


Figure 2.3 Example of a cascaded control loop

Cascading The servo drives used to control the servo motors of the labyrinth have a number of *cascaded control loops*. At the lowest level, one has to control the current output to the motors; however, it is also desirable to control the velocities and positions of the motors, as well as the angular position of the labyrinth frame board, and also the ball position.

A useful property of PID controllers are that they can be cascaded efficiently so that the output of one controller is used as input to another one. One advantage of using cascaded PID loops is that at the inner process, i.e. the plate angle, disturbances can be handled faster and corrected before causing disturbances on the primary process, the ball position, [Hägglund, 2000]. An example of a cascaded loop is seen in Figure 2.3.

Discretization Since a digital control system is to be implemented, any controllers used must be discretized. The sampling time used is denoted h and the sample points t_k. The proportional part with setpoint weighting translates directly to the discrete case as

$$P(t) = K(\beta y_{sp}(t) - y(t)) \quad \longrightarrow \quad P(t_k) = K(\beta y_{sp}(t_k) - y(t_k)).$$

The integral part needs to be approximated using an appropriate method. Some methods that might be of interest are *forward differences*, *backward differences* or *Tustin's approximation*. Using a forward difference approximation leads to the integral part

$$I(t_{k+1}) = I(t_k) + \frac{Kh}{T_i} e(t_k).$$

Adding tracking-based anti-windup to this gives

$$I(t_{k+1}) = I(t_k) + \frac{Kh}{T_i} e(t_k) + \frac{h}{T_t}(u(t_k) - v(t_k)),$$

where $u(t_k)$ is the saturated control signal and $v(t_k)$ is the non-saturated control signal.

The derivative part also needs to be approximated. Using any of the three approximation methods previously mentioned leads to the same form,

$$D(t_k) = a_d D(t_{k-1}) - b_d(y(t_k) - y(t_{k-1})),$$

but with different formulae for calculating a_d and b_d. Using the most common method, backward differences, a_d and b_d are defined by

$$a_d = \frac{T_d}{T_d + Nh}, \qquad b_d = \frac{KT_dN}{T_d + Nh}.$$

This version of the derivative part includes a limitation of the gain as discussed earlier.

The control signal can finally be written as

$$v(t_k) = P(t_k) + I(t_k) + D(t_k).$$

To read more about the results of approximating the integral and derivative part, see [Årzén, 2012] where all of the above formulas are collected, or [Wittenmark et al., 2012] for an implementation example.

2.3.2 Event Based Control

The traditional focus of automatic control is periodic control systems [Åström, 2008]. In such periodic control, continuous signals are sampled at fixed intervals to provide their discrete counterparts. However, for some systems it may be interesting to instead consider *event based control*, in which asynchronous events dictate control signal updates. If one has limited computational resources it may be preferred to only perform calculations when they are actually needed. And for some systems there might be no reason to update the control signal until a specific event occurs, e.g. when a signal passes a certain trigger level (such as when controlling a tank level).

It has been shown that event-based PID controllers work well for some systems [Årzén, 1999].

2.4 Pathfinding

For the labyrinth to be able to be self-playing it must somehow be able to control setpoints automatically. This can, for example, be done by using a set of waypoints or connected nodes that are automatically traversed. The first waypoint will be the first setpoint, and when the ball is close enough to the current setpoint, the setpoint is changed to the next waypoint. To be able to determine these waypoints automatically, one can use *pathfinding*. To find the best way through a graph of connected nodes, there exist several search algorithms. A seasoned but still popular and effective algorithm is *A** (pronounced A-star). The A* algorithm was first described in [Hart et al., 1968] and will be given a short introduction here.

2.4.1 A*

Given a map of points (where every point represents either clear space or an obstacle), a starting point and a finish point, one wants to find the best way from start to finish while avoiding obstacles.

The A* algorithm utilizes two sets, one open and one closed. The open set contains points that might fall along the path and therefore need to be checked while the closed set contains points that have been visited and do not need to be rechecked.

Begin at the starting point, add it to the open set, and look for adjacent points that are in reach, optionally including diagonal moves, and add them to the open set while defining the starting point as their parent. The starting point might now be dropped from the open set and added to the closed one.

Next subject is to calculate the cost of stepping to each point. The total cost, F, is the sum of the past cost, G, and the future cost, H,

$$F = G + H.$$

The past cost is the cost of reaching the particular point from the starting point, counting that each step has a specified cost. The future cost is an admissible heuristic estimate of the distance to the finish point, for example, this might be the straight line to the finish point. It is advantageous to use rounded numbers so that computations are kept simple.

Using the cost F as comparison, the point with the lowest F in the open set is chosen as the next point. The new point is dropped from the open set and added to the closed set. For all reachable adjacent points that are not already in the closed set, check if the cost G using this path is lower than before. If so, the current point should be considered the parent. All adjacent points that are not already in the open or closed set should be added to the open set, saving the current point as their parent.

The procedure is now repeated until the finish point is added to the closed set or the open set is empty, in which case there is no path. With the finish point on the closed set the trace of parents will show the best path from start to finish point.

3

Hardware

3.1 Introduction

The labyrinth consists of a $0.80 \times 0.80 \times 0.45$ m (w $\times$ d $\times$ h) plywood box with a transparent plexiglass plate top mounted in a two-piece wooden gimbal frame, see Figures 3.1 and 3.2. Beneath the gimbal frame is a tilted plexiglass plate to catch the ball when it falls through the holes in the labyrinth and to dispose it in a tube that dispenses the ball. A list of the available hardware is presented in Table 3.1. All control components are mounted on the bottom of the box, except for the panel which is mounted on the side of the box for easy access. Inside the box there is also a WLAN router to allow external connections from an Android device, and three LED strips that can be controlled for visual effect. The camera is mounted on a fixed stand above the maze to get a good overview of the entire game board. The ball used is a simple steel ball with a diameter of 3 cm.

Device	Type	Manufacturer	Quantity
PLC	AC500	ABB	1
Control panel	CP635	ABB	1
Servo drive	MicroFlex e150	ABB	2
Motor	BSM63N-275AF	Baldor	2
Gear head	alphira 060-100	Wittenstein	2
Inclinometer	ACS-020	Posital	1
Camera	P3367	Axis	1
Tablet	Nexus 7 / K008	ASUS	2

Table 3.1 Hardware components

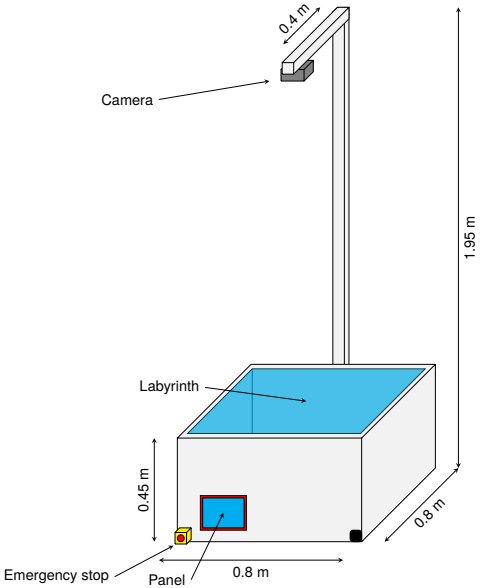


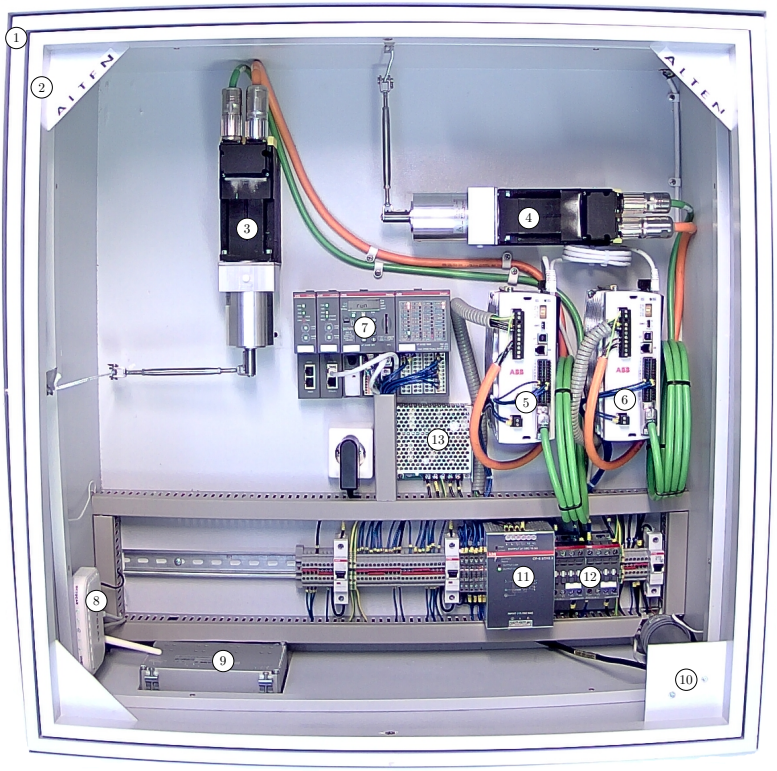
Figure 3.1 Overview of the labyrinth and the camera attachment

3.2 Servo Drives

The labyrinth setup includes two *MicroFlex e150* servo drives which each contain hardware for alternating current output to control an AC motor. The drives operate at 230 VAC one- or three-phase input with 24 VDC control input. The drives control the feed to the servo motors and contains three cascaded control loops and some feed-forward and filtering functionality. The innermost loop controls the current feed, and on top of this loop there is another for controlling the motor velocity. The outermost loop is used for position control, i.e. the rotation angle of the servo axis. The motors have line encoders for feedback, with a line count of 2500. The drives support *EtherCAT* communication (see Section 3.3) and existing software libraries for the AC500 PLC allow for communication between PLC and drives to e.g. alter setpoints and monitor error statuses.

The parameters of the drives can be modified using the accompanying *Mint Workbench* software and a USB connection. The Mint software has several features such as:

- Automatic tuning of control parameters for different bandwidths with or without load.



1. Outer gimbal frame (*y*-axis control)
2. Inner gimbal frame (*x*-axis control)
3. Motor (*x*-axis)
4. Motor (*y*-axis)
5. Servo drive (*x*-axis)
6. Servo drive (*y*-axis)
7. PLC with I/O and EtherCAT
8. Router
9. Control panel
10. Inclinometer
11. 24 VDC PSU
12. Contactors for motor supply current
13. 12 VDC PSU

Figure 3.2 The labyrinth system

- Manual fine tuning of control parameters with possibilities to measure physical parameters such as electrical properties, and calculation of control parameters.
- Possibilities to run tests and to check performance.
- The ability to change numerous parameters other than directly control-related ones, with ability for custom programming of the drives.
- Monitoring and resetting of error alarms, etc.

3.3 EtherCAT

EtherCAT, or *Ethernet for Control Automation Technology*, is an Ethernet based *fieldbus* system. Fieldbus is a common name for a number of computer network protocols used in industry for distributed communication in automated systems which have real-time communication requirements. A *distributed real-time control system* consists of a network of different nodes such as sensors, actuators and controllers [Nilsson, 1998]. The controller nodes will read measurement values from the sensor nodes and calculate control signals that are sent to the actuator nodes.

Ethernet is probably the most used technology for local area networks. It is, however, not intended to be used for real-time communication [Nilsson, 1998]. But due to Ethernet being a commonly used technology, several real-time communication technologies have been based on it. Some examples other than EtherCAT are *PROFINET* and *Ethernet Powerlink*.

EtherCAT overcomes the limitations of Ethernet by having slave devices read data from and write data to frames as the frames are passing through the devices [*EtherCAT*]. By also sharing frames between devices, the frame delays are reduced to nanoseconds and the usable data rate increased to over 90 % [*EtherCAT*].

3.4 Controller

The ABB *AC500* PLC supports flexible software development and supports multiple tasks. As a modular scalable device, it is easy to add and configure different types of input-, output- and communication modules.

The PLC is accompanied by the *Control Builder Plus* software and additional *CoDeSyS* and *Panel Builder* software that can be used for:

- hardware and parameter configuration
- I/O mapping
- programming
- visualization

- debugging, diagnostics and tracing

The PLC programming environment supports all the programming languages defined in the IEC 61131-3 standard:

- ladder diagram
- function block diagram
- structured text
- instruction list
- sequential function chart

In addition to these languages, it is also possible to use the C programming language together with an external compiler.

In the used setup, both Ethernet and EtherCAT modules are available for communication, and for I/O, both digital and analog inputs and outputs are available.

3.5 Camera

The Axis *P3367* camera supports a resolution of 1600×1200 pixels at 30 frames per second and is running a Linux-based operating system on a *ARTPEC-4* processor architecture.

3.5.1 VAPIX

VAPIX is an Axis developed API (*application programming interface*) based on standard HTTP and RTSP protocols for communicating with and controlling Axis cameras. The API is simple to use and there is support for several other features than just taking pictures or streaming video, such as:

- audio
- motion detection
- pan, tilt, zoom
- triggers and events
- parameters
- image overlays and view areas
- I/O and serial ports

Since one can simply use HTTP, it is easy to implement applications using VAPIX using most languages and on most platforms. Most commands can be issued directly from a web browser. For VAPIX documentation see [*VAPIX*®].

3.5.2 ACAP

Since the camera runs Linux it is possible to also develop applications that can be run on the camera itself. For this purpose, Axis has a SDK (*software development kit*), ACAP (*Axis Camera Application Platform*), which includes C libraries with interfaces for:

- media and image capture
- application parameters
- HTTP interfaces
- events
- license keys
- fixed point math
- image processing

Development using ACAP is perhaps not as straight-forward as using VAPIX, but it does allow for e.g. an image processing server that can send results to clients.

The image processing library included is named RAPP (*Raster Processing Primitives*) and contains many useful operations. The library is open-source and the documentation is available in [*RAPP User's Manual*].

3.6 Tablet Devices

The *Nexus 7* is a tablet device with a 7" LED touch screen running Android OS 4.3 (JellyBean). It has a quad-core 1.2 GHz processor and features an accelerometer and a gyro which are interesting for this application.

The purpose of the tablet devices is for a user to interact and control the system. Since the tablets contain gyroscopes, it is possible to tilt them to provide angular setpoints to control the labyrinth board angles in two dimensions. They should also be used to control ball position setpoints, ideally from a labyrinth layout where the current ball position can be viewed and the setpoint can be set on the screen.

4

Implementation

4.1 System Design

4.1.1 Hardware

The goal with the implementation is to allow a user to be able to control the position of the ball in the labyrinth using a tablet device, or to allow the labyrinth to be self-playing. The camera should measure the ball position and the PLC should control the servo drives which in turn control the servo motors. The servo motors are connected to two joint arms which in turn are connected to the two gimbal mounted frames. The rotation of the motors will then translate to rotation of the two frames which are resting on ball bearings. Each motor will control a separate axis, one x-axis and one y-axis.

The labyrinth size has been kept small enough to allow it to be moved easily through doors. This means that the space for mounting devices is limited, and that if a motor should spin out of control, the inertia can damage other equipment such as the PLC; therefore, a maximum operating angle has to be defined for the motors. This angle is approximately 30° or $\pm 15°$. Should any of the motor angles exceed this limit, the power driving the motors has to be cut immediately. Also, with moving parts, it is important to have an emergency button to provide the same functionality. The 230 VAC power supply to the servo motors is controlled by the PLC using *contactors* (or electronic switches), which can be switched from the 24 VDC digital output of the PLC.

4.1.2 Network Design

The PLC is selected to be the master device in the network, to which other devices should connect. The PLC is most suitable for this as the control signals to the servo drives have to be sent on the EtherCAT network, which only the PLC and the drives are connected to.

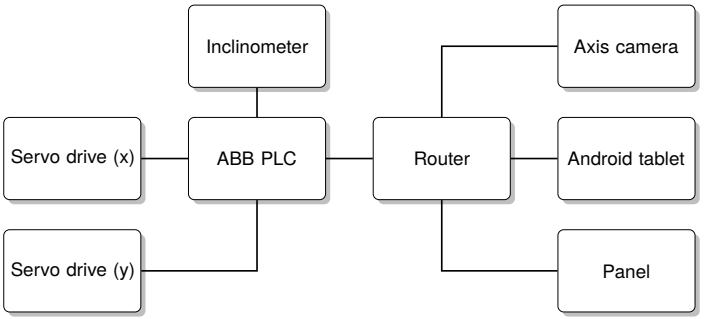


Figure 4.1 System network overview

Since it is possible to develop software to be run directly on the camera, and as the camera is designed to be able to handle image processing, it is suitable to implement the tracking algorithm directly on the camera. This is also efficient in the sense that no images have to be sent over the network, and thus the latency between an image being captured and processed can be minimized. The camera should therefore be treated as a position sensor in the distributed network. The position sensor should ideally be implemented to run in an event-based fashion. When tracking is active, any new position measurement should be relayed to the system controller as fast as possible, and having the master device or PLC request data from the position sensor would only create unnecessary overhead.

Since the latency of the system should be minimized, it is sensible to transfer all data using UDP datagrams rather than TCP packets in order to avoid the overhead introduced by package acknowledgments and ordering restrictions that exist when using TCP. This does imply that signals might not be received in the exact order they were sent: However, considering the size of the local network used, this is unlikely, and should it occur, it can simply for the position sensor be considered as measurement noise.

The result is a star-topology network as shown in Figure 4.1. The PLC acts as master device or control node, the camera and tablet as sensor nodes, and the servo drives as both sensor and actuator nodes. To calibrate the servo drives correctly, there is also an inclinometer, which can measure angles in two dimensions. The inclinometer is a completely analog device which is connected to the analog input of the PLC.

4.2 Controlling Setpoints

The system setpoints are to be controlled from a tablet device and sent to the PLC. Two different types of control modes should be available:

- frame angle control
- ball position control

With the frame angle control mode, the user should be able to use the gyroscope that is built into the tablet device to provide the reference angles in two dimensions. This means that the user can simply tilt the tablet device in a desired angle, and that angle should then be reflected by the labyrinth.

In the other mode, the position of the ball should be controlled. By tilting the tablet device it should be possible to control the change of the reference position, e.g. a small angle will move the reference position at a slow speed. It should also be possible to change the setpoint by touch input, e.g. by pressing a position on the screen.

Finally it would also be interesting to provide a multiplayer-mode, where two users have one tablet each, and each user controls one of the axes, either in angle or position control mode.

To provide feedback to the user, measurement values should be transferred to the tablet(s) during all modes. These measurement values should include both the current actual angles and the current ball position. The angle values can then be drawn on the tablet screen together with the reference values. The ball position should be visualized on either a drawn labyrinth reflecting the real one to indicate the tracked position, or using an image stream from the camera as a background.

4.3 Selection of Ball Tracking Algorithm

4.3.1 Background Subtraction

It became clear early on that a background-subtraction based algorithm was needed to cope with the background noise from all background components as well as from the moving frame. Initial attempts with gradients for circle detection with the *Hough transform* (see Section 2.2.4) for feature extraction proved very hard because of the background complexity. Figure 4.2(a) shows the labyrinth background and Figure 4.2(b) show the gradient magnitudes calculated from the same image. The image gradients show many circular features and it is hard to locate the ball. The Hough-transformed image using the approximate ball radius is seen in Figure 4.2(c), and shows that the ball is not easy to differentiate from the background.

By using Python together with *SciPy*, *NumPy* and the *OpenCV* computer vision library it was possible to test the performance of the background subtraction techniques discussed in Section 2.2.7. The adaptive background algorithm described in [Gruenwedel et al., 2013] proved suitable due to it being relatively easy to implement on an embedded system as the algorithm is not as computationally intensive as other algorithms. The adaptive algorithm is also supposed to perform as well as other more complex background subtraction algorithms such as those based on *mixtures of Gaussian* models [Gruenwedel et al., 2013].

Thus, the foreground $\mathbf{F}_k$ at time k, containing any currently moving objects, is composed of a short-term foreground $\mathbf{F}_s$ and a long-term foreground $\mathbf{F}_l$,

$$\mathbf{F}_k = \mathbf{F}_{s,k} \circ \mathbf{F}_{l,k},$$

where

$$\mathbf{F}_{s,k}(x,y) = \begin{cases} 1 & \text{if } |\mathbf{I}_k(x,y) - \mathbf{B}_{s,k}(x,y)| \geq T_l \\ 0 & \text{otherwise,} \end{cases}$$

and

$$\mathbf{F}_{l,k} = (\mathbf{A}_k \oplus \mathbf{C}) \circ \mathbf{F}_{s,k}$$

with

$$\mathbf{A}_k(x,y) = \begin{cases} 1 & \text{if } |\mathbf{I}_k(x,y) - \mathbf{B}_{l,k}(x,y)| \geq T_h \\ 0 & \text{otherwise.} \end{cases}$$

So far there are two threshold parameters T_l and T_h ($T_l < T_h$) which control how the foregrounds are created from the short-term background $\mathbf{B}_s$, the long-term background $\mathbf{B}_l$ and the current image frame $\mathbf{I}_k$. The matrix $\mathbf{C}$ denotes the diamond-shaped structural element (see Section 2.2.5) used for the morphological dilation when calculating the long-term foreground.

The adaption rate parameters α_s and α_l ($\alpha_s > \alpha_l$) control how fast the backgrounds are updated as the input images change. The short-term background is updated using the exponentially weighted low-pass filter

$$\mathbf{B}_{s,k+1} = \alpha_s \mathbf{I}_k + (1 - \alpha_s)\mathbf{B}_{s,k},$$

and the long-term background is updated as

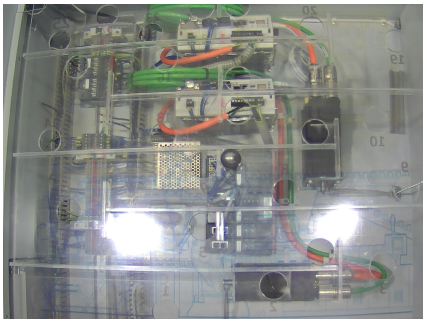
$$\mathbf{B}_{l,k+1} = (\mathbf{1} - \mathbf{F}_k) \circ \alpha_l(\mathbf{I}_k - \mathbf{B}_{l,k}).$$

However, background subtraction and foreground detection is not sufficient on its own for tracking, and thus the algorithm needs to be extended.

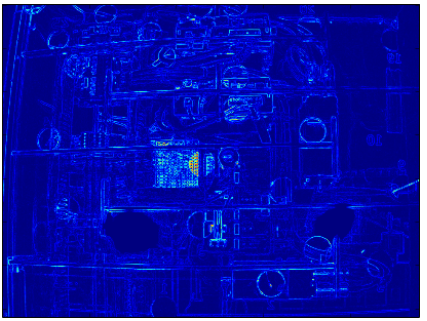
4.3.2 Glare Removal

One major issue to tackle is the glare effects due to the labyrinth having reflective boards of plexiglass, with one of them attached to the moving gimbal frame (see figure 4.2(a)). A very small movement of the plexiglass frame may, depending on the surrounding lighting conditions, move the reflections across the image.

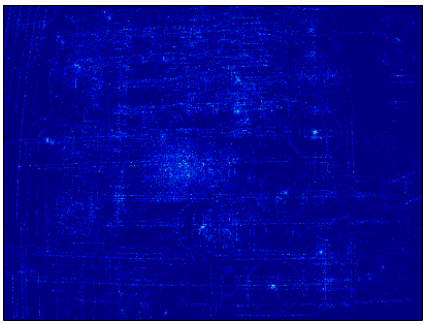
The moving reflections caused by the change of the plexiglass angle creates significant illumination changes between frames and causes unwanted foreground detection. However, since the ball is spherical, any light directed at the ball will be reflected in several different directions and thus assure that the ball will never be as illuminated as a plane reflecting a direct light source such as the plexiglass does.



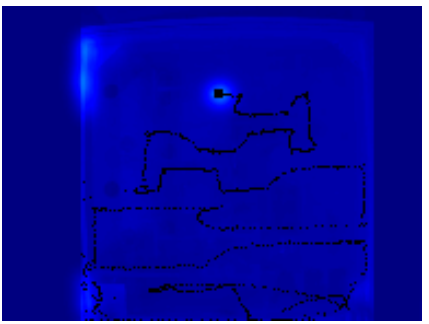
(a) Background and glare effects on image



(b) Gradient magnitudes



(c) Hough transformed image for circle-features



(d) Example of tracking scenario with foreground detection of edges

Figure 4.2 Image examples

Using the spherical properties of the ball, it is possible to remove the glare effects on the foreground detection by simply using a glare threshold T_g, and only accept pixels below this threshold as part of the foreground; thus, the glare pixels $\mathbf{G}$ for the current image frame $\mathbf{I}_k$ are given by

$$\mathbf{G}_k(x,y) = \begin{cases} 1 & \text{if } \mathbf{I}_k(x,y) \geq T_g \\ 0 & \text{otherwise,} \end{cases}$$

and the foreground is thus redefined to ignore the glared areas as

$$\mathbf{F}_k = \mathbf{F}_{s,k} \circ \mathbf{F}_{l,k} \circ (\mathbf{1} - \mathbf{G}_k) = \mathbf{F}_{l,k} \circ (\mathbf{1} - \mathbf{G}_k).$$

4.3.3 Filtering

When the gimbal frame moves fast, foreground detection will occur at the edges of the labyrinth. This is because the movement causes pixel intensity changes in lines along the moving edges. This behavior can be observed in Figure 4.2(d), where the lighter areas indicate foreground detection. The moving ball is detected, but there

is also foreground detection along the edges of the upper left corner; however, as these detections occur in lines in contrast to the pixel-masses from the moving ball, it is possible to apply a low-pass filter to the foreground to reduce the edge detection values relative to the ball detection values as there is no foreground detection on the sides of the frame edges.

So, after having calculated the binary foreground matrix for the current image frame, the foreground should be converted to a non-binary matrix, to which e.g. a Gaussian filter can be applied. The filtering process will also eliminate outlier pixels such as image noise, which can come from e.g. blinking status LEDs on the PLC, etc.

4.3.4 Position Estimation

Once the foreground has been filtered and smoothed one can search the image for its maximum value and its location. This location should then represent the approximate center of the foreground object moving the most; thus, this location can be considered to be the current ball position. If the ball lies still for some time it will blend into the background model and therefore not be detected. Though, the last detected position is still known by the system and when the ball starts moving it will appear in the foreground again.

Earlier, the reflection from the spherical ball was discussed. The glare removal stage is based on the fact that light will be spread when hitting the reflective ball. But a light source above the labyrinth will still cause some reflection on the central part of the ball, which will be considered as glare and be removed. This would then imply (of course depending on the filtering stage) that under such circumstances, the maximum foreground location will not be located at the central part of the ball, but rather, closer to an edge.

Also, when the ball is not moving at all, there should be no new position detections; however, small image noise such as e.g. shadows, can introduce small foreground detections. To avoid such false positives, another threshold is introduced. A detected ball position should only be accepted if the maximum filtered foreground value is larger than a specified threshold T_f.

4.3.5 Local Search

One observation is that in-between sampled image frames, the ball can only be expected to move a certain distance from its previous position; thus, one can limit the search for a new ball position to an area around the previous ball position.

This area does depend on the speed of the ball. If the ball is lying still, it can only accelerate a certain amount before the next image is sampled and can thus only move a limited distance. But with a high velocity, the ball can travel a much larger distance.

To be able to decide around which position the local search should be conducted, a Kalman filter is applied to predict the most likely position of the ball in the next frame. The tracking in the next frame is then limited to an area around the predicted position.

By limiting the image tracking algorithm to a small area, the computational cost required is significantly reduced especially for filtering operations as the number of required pixel operations is reduced. Also, by limiting the search to a localized area, the number of false positives could be expected to be reduced.

The introduction of localized search allows for improvements to the ball position detection in the foreground. Instead of using the maximum filtered foreground value, it is now possible to calculate the centroid, or pixel mass center of the localized area. This can be done by calculating the image moments in both dimension as well as the total pixel mass sum to get a position

$$(\overline{x},\overline{y}) = \left(\frac{M_{1,0}}{M_{0,0}}, \frac{M_{0,1}}{M_{0,0}}\right).$$

This would not have been very effective over a large area containing outliers as the moments would potentially add undesirable weights to the total moments depending on outlier locations.

Also, rather than thresholding the maximum value to decide whether a detection should be accepted or not, one can now use the local pixel mass sum $M_{0,0}$ instead.

4.3.6 Kalman Filter

To use a Kalman filter to estimate the ball position in the next frame one must first select a model for the ball position. For each axis of the labyrinth, the angle will determine the ball acceleration unless there is a wall obstructing the ball. Including angle measurements to the model would mean expanding both the model itself and also the communication in the system why it is not implemented. Also if the ball is leaning against a wall the angle measurement would not be of any use since the acceleration of the ball would still be zero.

A simple model will be to consider the changes in acceleration as white noise with some estimated variance. With a position $x(t)$ at time t, the acceleration will then be

$$\ddot{x} = w(t)$$

where $w(t)$ is white noise with a variance σ_w^2. Assuming that the white noise will be constant between samples, i.e. $w(t) \equiv w(t_k) \equiv w_k$ for $t \in [t_k, t_{k+1}]$ the state space model becomes

$$\begin{aligned} x_{k+1} &= x_k + h\dot{x}_k + \frac{h^2}{2} w_k \\ \dot{x}_{k+1} &= \dot{x}_k + h w_k, \end{aligned}$$

with a sampling period h. On matrix form this becomes

$$\begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ \dot{x}_k \end{bmatrix} + \begin{bmatrix} \frac{h^2}{2} \\ h \end{bmatrix} w_k.$$

Assuming that the white noise is Gaussian with zero mean, i.e. $\mathbf{w}_k \sim N(0, \mathbf{Q}_k)$, the process noise covariance matrix $\mathbf{Q}_k$ becomes

$$\mathbf{Q}_k = \mathrm{E}[\mathbf{w}_k \mathbf{w}_k^{\mathrm{T}}] = \begin{bmatrix} \frac{h^2}{2} \\ h \end{bmatrix} \begin{bmatrix} \frac{h^2}{2} & h \end{bmatrix} \sigma_w^2 = \begin{bmatrix} \frac{1}{4}h^4 & \frac{1}{2}h^3 \\ \frac{1}{2}h^3 & h^2 \end{bmatrix}.$$

Also, with a measurement noise $\mathbf{v}_k \sim N(0, \mathbf{R}_k)$ having a variance σ_v^2 the measurement covariance matrix can be written as

$$\mathbf{R}_k = \mathrm{E}[\mathbf{v}_k \mathbf{v}_k^{\mathrm{T}}] = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \sigma_v^2,$$

and since there is no control feedback ($\mathbf{B} = 0$) the filter model becomes

$$\begin{aligned} \mathbf{x}_k &= \mathbf{A}\mathbf{x}_{k-1} + \mathbf{w}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{v}_k. \end{aligned}$$

The full state transition matrix for the two positions $x(t)$ and $y(t)$ with corresponding velocities is then

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & h & 0 \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

and the full process noise covariance matrix

$$\mathbf{Q}_k = \begin{bmatrix} \frac{1}{4}h^4 & 0 & \frac{1}{2}h^3 & 0 \\ 0 & \frac{1}{4}h^4 & 0 & \frac{1}{2}h^3 \\ \frac{1}{2}h^3 & 0 & h^2 & 0 \\ 0 & \frac{1}{2}h^3 & 0 & h^2 \end{bmatrix}$$

assuming that the x and y positions are uncorrelated in their movement. Only the positions can be measured and thus only the positions should be extracted from the state vector. The velocity can not be measured and should be estimated by the filter; therefore, the matrix $\mathbf{C}$ should be defined as

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

to extract the measured position values from the estimated states when calculating the Kalman filter innovation.

Since the tracking algorithm requires an initial position, it can be assumed that a correct one will be provided initially and that the state estimate matrix $\hat{\mathbf{x}}_k$ can be initialized with the initial position and that the state estimate covariance matrix $\mathbf{P}_k$ can be set to zero initially.

The Kalman filter estimated position can now be used to select around which location the local search should occur, and the full tracking algorithm is given in Figure 4.3.

4.3.7 Normalized Parameters

The ball detection algorithm features a number of parameters whose values have to be determined. It is, however, clear that illumination plays a great role in how these values should be set. In a dark environment, threshold values have to be set much lower compared to a bright environment. For example, consider the visibility of a shadow. In a very bright environment, the darkness of a shadow will stand out significantly from its surroundings, whereas in a dark environment it will be hard to differentiate. It is thus reasonable to assume that if parameter values are set relative to the current illumination level, they will scale better with changes in illumination. All threshold values should therefore be set between zero and one to be considered relative to the current illumination level; thus, applying a threshold T to a matrix $\mathbf{A}$ to create a resulting matrix $\mathbf{B}$ will result in

$$\mathbf{B}(x,y) = \begin{cases} 1 & \text{if } \left|\frac{\mathbf{A}(x,y)}{\max \mathbf{A}}\right| \geq T \\ 0 & \text{otherwise.} \end{cases}$$

4.4 Camera Implementation

4.4.1 Overview

The camera implementation is done in C using the Axis ACAP SDK. The resulting code is then cross-compiled and wrapped in a camera application package that can be transferred to the camera for execution. The P3367 camera can handle a maximum frame rate of 30 frames per second. To be able to utilize all frames, the ball tracking algorithm will have to be able to meet a deadline of ≈ 33 milliseconds per frame. Also, higher processing speed will reduce the measurement latency in the control system. Some approaches for improving the performance are:

- Avoidance of floating point number calculations by using fixed-point numbers instead.

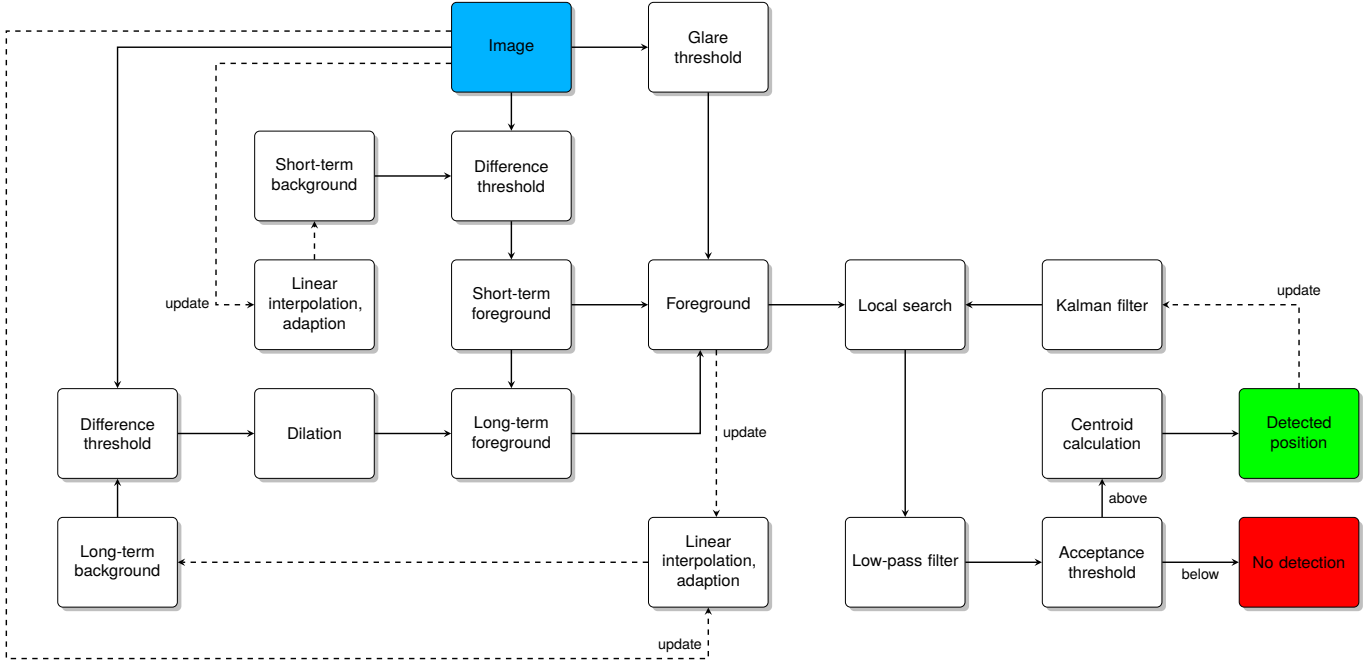


Figure 4.3 Overview of the full tracking algorithm. An input image results in either a new detected position or no detection, after which the background models are updated as indicated by the dashed lines.

- Correct handling of multidimensional arrays, e.g. memory layout requirements such as alignment, and respecting the layout requirements for loop-processing, etc.

- Correct usage of pointers and pointer restrictions to avoid context-switching overhead and to allow the compiler to produce more efficient code.

- Sub-image processing.

The ACAP SDK provides some of this functionality already with the included fixed-point mathematics library and the RAPP image processing library. The fixed-point library uses 32-bit wide datatypes and supports several mathematical functions. The RAPP image processing library includes a vast number of image processing functions, and other functions, such as (from [*RAPP User's Manual*]):

- aligned image buffer allocation

- pixelwise operations

- statistical operations

- spatial filtering

- border padding

- geometrical transformations

- conditional operations

- integral images, etc.

The RAPP library handles two different types of images: unsigned 8-bit images and binary images. The 8-bit image pixels can assume values between 0-255. For binary images, each pixel only uses a single bit of the image buffer. For performance, all image buffers must fulfill certain alignment requirements [*RAPP User's Manual*]. The entire image buffer must be aligned, and the row dimension, or the number of bytes to the pixel at the same column on the next row must also be aligned. The actual image data inside the buffer must also start at an alignment boundary or alignment multiple with a potential bit-offset value for binary images; thus, when allocating image buffers for the tracking algorithm, it is important that these requirements are followed. Some image processing operations, such as filtering, are calculated by convolving the image using a certain filter kernel. For border pixels, this means that pixels outside the actual image may be processed depending on the kernel size; thus it is also useful to allocate images with a certain number of pixels used for padding.

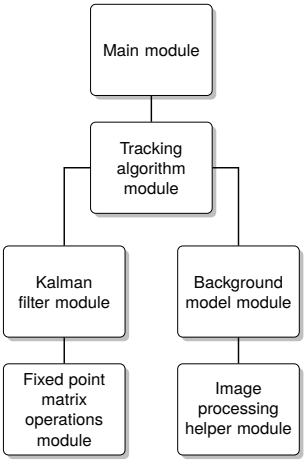


Figure 4.4 Camera application modules

4.4.2 Module Implementation

To cope with the buffer requirements, a module was implemented to provide consistent allocation and initialization of image buffers. This helper module also contains functions for calculating pointers to image start positions inside buffers and the values for used row dimensions.

The actual tracking algorithm can be layered by breaking it down into separate modules as seen in Figure 4.4. The *background module* handles the adaptive background models and produces a foreground from a passed image buffer.

The *Kalman filter module* contains the Kalman filter implementation, which takes a measured position as input to produce an estimated position as output. The Kalman filter module relies on an extension module of the fixed-point mathematics library that provides matrix allocation and matrix operations such as addition, multiplication, transpose multiplication and inversion.

The *tracking module* uses the background module and the Kalman filter module to implement the tracking algorithm by using the Kalman filter output to select a region for localized sub-image search.

4.4.3 Application Implementation

The ACAP SDK provides a parameter interface that allows for predefined parameters to be edited directly from the camera web interface using the included event-model. This allows for easy parameter changes.

The application requires two different threads. The first thread is used to receive UDP datagrams, and it will be blocked when waiting for new datagrams to appear in the buffer. The second thread will run the actual image capturing, tracking and sending of the resulting position datagrams. The image capturing and tracking are implemented using a simple state machine with four modes: *idle, starting, running* and *stopping*. A mutex allows the state machine to be controlled from the datagram reception thread.

The parameter interface requires parameter handlers to be registered for the different parameters. These handlers are then entered externally, which means that mutex locks are also required as memory barriers for parameter values.

The media stream from which images are captured is configured to receive Y800 monochrome 8-bit images which can be directly copied to the tracking algorithm input buffer.

The state machine execution is detailed in Figure 4.5. During the idle mode, the camera will simply wait for instructions to start and then change mode. When starting, the image media stream will be started, the initial ball position will be read and the tracker will be initiated by allocating all buffers required by the different modules. When the start-up procedure is completed, the state machine will be in the running state. When running, the application will wait for the next image frame from the media stream and then copy the image data to a buffer. The buffer will be processed by the tracking algorithm, and if the algorithm detects a new ball position, this position will be sent to a configured control server. Before repeating the procedure, the application will check for instructions to stop the tracking. When stopping, allocated resources will be freed and states will be reset before again entering the idle mode.

4.5 Control

4.5.1 Driver Control Loop

Each servo drive includes a cascaded control loop as seen in Figure 4.6.

The many available control parameters of the servo drive loops can be automatically tuned using the Mint Workbench software using a USB connection. For further fine tuning, the parameters can be changed manually.

The only parameter that has to be set by the operator is the controller bandwidth, which is individually set for each control loop. The control loops are cascaded in the following order: a PI control loop for current control, a PI control loop for velocity, and a PID control loop for the angular position. The filters seen in Figure 4.6 are not used in this implementation. For the drives to work properly, a fairly low controller bandwidth for the position and velocity has to be chosen at 250 rad/s. The PI loop for the current is automatically tuned with a bandwidth of 2000 rad/s.

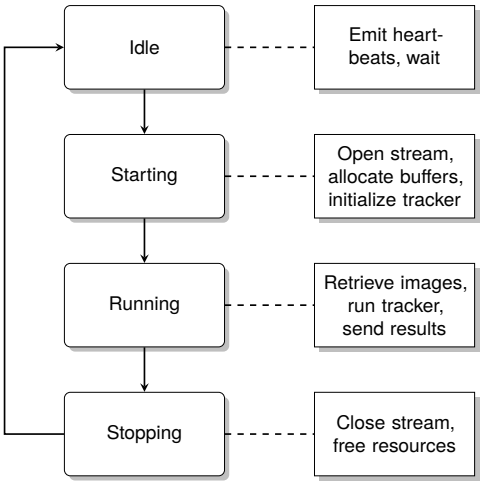


Figure 4.5 Camera implementation state machine

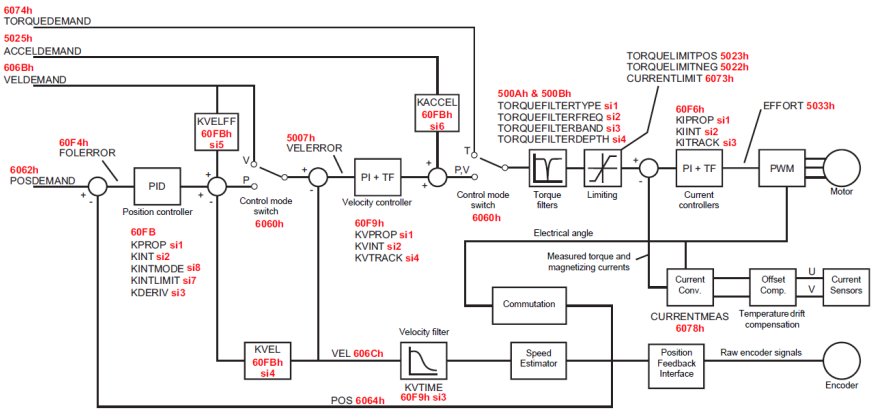


Figure 4.6 The cascaded servo drive control loop. From [*User's manual: MicroFlex e150 servo drive* 2013].

4.5.2 Ball Position Control

Using an event-based controller for the ball position is tempting. If the labyrinth process tries to keep the ball at a certain position, it is reasonable to assume that the control signal does not need to change unless the ball position deviates and an error is introduced; however, a potential issue with event-based control for the system is that unless the proportional gain is large enough to overcome friction, the integral part will not increase to do so.

The speed of the camera is dependent on several factors. First of all, the frame rate is limited to 30 frames per second by the hardware. Then each frame has to be processed by the tracking algorithm and the result be sent in a datagram over the network. The camera also runs other applications, such as a web server, and the task scheduling will also affect the speed. The tracking algorithm speed is also varying as the speed of image operations depends on image data.

When the camera actively supplies new position measurements, the ball position control signal should be updated as these measurements are received by the PLC. If the ball is not moving, no new position measurements will be received. The control signal should then be updated periodically at a predefined interval set at the achievable frame rate of the camera when running the tracking algorithm. The controller should also be reset every time the tracking is restarted so that the previous integral part is zeroed.

The ball position measured by the tracking algorithm may be rather noisy as the algorithm will not always be able to locate the exact center of the ball. The positions will be varying in an area close to the center, which can be considered as measurement noise. The measurement noise may be problematic for the position controller, particularly effecting the derivative part. An additional low-pass filter is therefore introduced to filter the position reference signal as it is fed to the controller.

The full control loop including the ball position control is shown in Figure 4.7.

4.6 Communication Protocol

4.6.1 Overview

The camera should act as an event-triggered sensor and push ball position measurements to the PLC as soon as new values are available. The PLC should process the values and send control signals to the servo drives. The tablet device(s) should send the control loop reference values. The communication protocol should use UDP datagrams which implies potential data loss. The protocol details are available in Appendix A.2.

4.6.2 Communication Between Camera and PLC

When it comes to the network communication between the camera and the PLC, the following requirements apply:

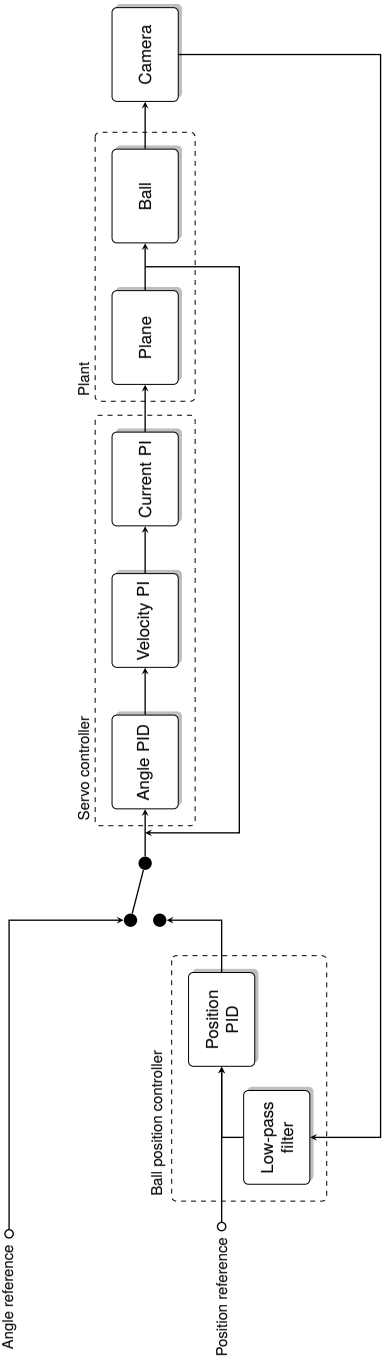


Figure 4.7 The full system control loop

- The PLC must be able to change the mode of the camera.
- The PLC must be able to set the initial ball position used by the tracking algorithm.
- In order for the PLC to know whether or not the camera is connected to the control network and running, a heartbeat functionality is needed.
- The camera must be able to send measurement values to the PLC.
- The camera must be able to acknowledge commands sent from the PLC.

The initial position is required to start the tracking, and sending a new initial position can be combined with starting the tracking. By using four bytes it is possible to send two coordinates with values 0-65535, which should be more than sufficient. The camera can then acknowledge that it has received the initial position and is starting by returning a simple acknowledgment byte. To stop the tracking when it is not needed, the PLC should simply be able to send a stop byte and receive a stop-acknowledgment byte.

The heartbeat functionality is needed by the PLC to ensure that the whole system is functional. It would be no point to allow a user to control the ball position if the camera is not running correctly. The camera will be emitting heartbeat bytes. When idle, the heartbeat will be sent periodically at a predefined interval, and when the tracking is active, the heartbeat will be sent when no new position measurements are available. When position measurements are available, they will instead suffice as a guarantee that the camera is running. Should the PLC not receive heartbeat messages or position measurements under a predefined timeout interval, the camera should be considered as disconnected from the network.

4.6.3 Communication Between Tablet Devices and PLC

The communication between the PLC and the tablet device(s) has slightly more requirements than the communication between the camera and the PLC:

- The tablet devices should be able to connect and disconnect from the PLC.
- Once a tablet device connection has been accepted by the PLC, the PLC should acknowledge the connection success to all connected tablet devices.
- Connections to the PLC should be maintained by answering periodic heartbeats emitted from the PLC.
- The tablet devices should be able to select the current control mode: angle or position control. The mode changes must include the initial ball positions to initiate the tracking algorithm.
- The PLC should confirm control mode changes.

- Both angle and position setpoints should be sent from the tablet devices as the values change.
- Angle and position measurement values should be sent from the PLC to the tablet devices as the values change.
- Upon errors, e.g. if the emergency stop button is pressed, the tablet devices should be informed.
- The PLC should be able to handle connections from two tablet devices simultaneously.
- The tablet devices must be informed if they both are connected at the same time, implying that the multiplayer mode is enabled.
- When the multiplayer mode is active, the tablet devices need to be informed about the setpoints of each other as well as which axis is controlled by which device.

A connection should be initiated by sending a connection byte to the PLC, which the PLC will reply to if it has a free connection slot available and the connection is accepted. This will initiate the periodic heartbeat byte messages emitted from the PLC to the connected tablet. If the tablet receives a heartbeat message, it should respond back with the same heartbeat to inform the PLC that it is still connected. By failing to respond to heartbeat messages for a certain connection timeout, a connected tablet device will be considered to have disconnected; thus, to disconnect, the tablet device will just have to stop responding to messages from the PLC.

The message for changing modes will include the 4-byte initial ball position and will be acknowledged by the PLC sending back a status message. The status message will contain information about the current mode and if multiplayer is active. If multiplayer is active, the assigned axis to be controlled will also be included in the status message. Should a status message be lost, the PLC will resend one upon receiving a new mode change request, or by receiving an illegal message during the current mode. The position setpoints are sent as 4 bytes with the desired location in reference to the camera image coordinate system. The angle setpoints are also sent as 4 bytes, but have to be translated. An angle setpoint value of θ degrees will be transformed to an integer value x to be sent as

$$x = \text{round}(100 \cdot \theta) + 18000.$$

This allows angle values to be sent in a range of $-180°$ to $180°$ with a resolution of $0.01°$. The measured angle and position values are sent in the same way as the setpoint messages, but to the tablets as measurement messages. Errors are relayed to the tablets by reserving an error mode bit in the status messages.

When the multiplayer mode is active, the PLC logic will handle the relaying of setpoints across the tablet devices automatically. New status messages will be sent out if a second tablet connects, and the PLC will start to relay measurement values between the tablet devices. Also, when multiplayer is active, it is possible for the tablets to send shorter setpoint messages with values for only one axis.

4.7 PLC Implementation

4.7.1 Introduction

The PLC logic is implemented using the high level structured text programming language. The implementation is structured in different layers using different *function blocks*. Function blocks are essentially functions with input and output signals, that have an internal state. The function blocks are handled and executed by different programs, which in turn are executed by different tasks.

4.7.2 Details

An overview of the implementation layers is given in Figure 4.8. There are two *axis programs*, each executing an *axis function block*, one for each axis. These blocks handle the motion control communication with the servo drives by reading global angle setpoint variables and sending motion requests to the drives.

The *incliner program* is used to read analog signals from the inclinometer and rescale the values to actual degrees, and the *LED program* controls a set of LED arrays for visual connection feedback to users. The *emergency stop program* monitors the servo drives for errors as well as checks if the emergency stop button has been pressed. In case of errors, the contactors supplying the servo motors with current will be disabled.

The *main program* contains all communication and essentially acts as a bridge between the camera, the control loop and the tablet devices by running a number of different function blocks and relaying signals in-between them. The main program also manages the tablet connections based on client-unique IP addresses.

The *client reception function* block checks the datagram buffer on the port open for client connections and parses the byte data into different output signals corresponding to the different commands that can be sent to the PLC together with parameters. The *client status*, *measurements* and *multiplayer* blocks are used to send the corresponding messages to connected devices. Input parameters, such as angle values are automatically converted and put into byte buffers that can be sent to a specified client. Each client has its own heartbeat block, which uses a timer to send heartbeat messages at a predefined interval.

The *camera manager function block* manages the camera transfer and reception blocks. The manager will internally make sure that acknowledgments are received from the camera–if not, commands are automatically resent if possible. The manager will also be monitoring the heartbeat status of the camera.

The *PID function blocks* contain the discrete PID controller implementation based on Section 2.3.1. Each block has inputs for reference signal, measurement signal, resetting and different control parameters. The blocks will themselves measure sample times between invocations that are used to calculate the control signal outputs. It is also possible to disable integral or derivative action if needed. Listing 4.1 contains part of the PID function block implementation and shows how the control signal is calculated.

Listing 4.1 Part of PID function block implementation

```
CurrentTime := SYS_TIME(EN := TRUE); (* Get current system time *)
h := CurrentTime - LastSampleTime;
LastSampleTime := CurrentTime;

P := K * (SetPoint - Actual); (* Calculate proportional part *)

IF Td <> 0.0 THEN (* Calculate derivative part *)
  ad := Td / (Td + N * h);
  bd := K * N * ad;
  D := ad * D - bd * (Actual - PreviousActual);
  PreviousActual := Actual;
ELSE
  D := 0.0;
END_IF;

v := P + I + D;

Output := LIMIT(LimitMin, v, LimitMax); (* Limit the output signal *)

IF Ti <> 0.0 THEN (* Update integral part *)
  bi := K * h / Ti;
  ar := h / Tt;
  I := I + bi * (SetPoint - Actual) + ar * (Output - v);
ELSE
  I := 0.0;
END_IF;
```

The PID function blocks are managed by the *cascade controller blocks*. The cascade controller blocks take a control mode input signal which will determine whether the outermost cascaded ball position loop will be used or not. If is the position loop is to be used, the PID blocks will be used to calculate the angle reference signal output of the cascade blocks. The angle reference signal is written to global angle setpoint variables, which the axis control blocks will use to update the servo drive setpoints.

4.7.3 Scheduling

The PLC scheduling is done manually. A set of different tasks are created, and each task is assigned one or more programs to execute. There are three different types of tasks:

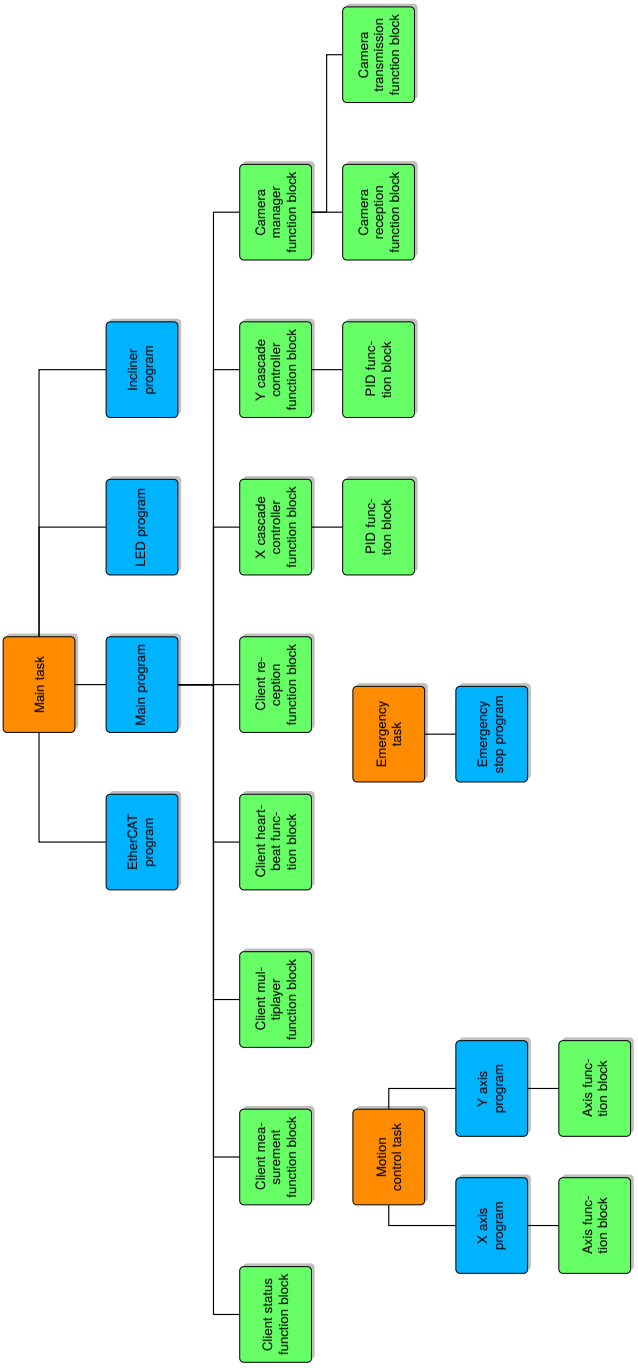


Figure 4.8 PLC implementation structure overview

- cyclic
- freewheeling
- event-triggered

Cyclic tasks are released at specified intervals and event-triggered tasks are released when specified internal or external events occur. Freewheeling tasks are immediately released when the PLC starts, and when completed, restart again immediately. All tasks are also assigned static priorities, which dictate how preemption is performed. If two tasks to be executed have the same priorities, the task with the longest waiting time will be executed first [*User Manual for PLC Programming with CoDeSys 2.3* 2010].

For this implementation; three tasks are required as shown in Figure 4.8. A high-priority cyclic emergency task will monitor the emergency stop and axis errors at frequent intervals.

The axis control or servo drive communication will be handled in a separate task. This task will have a medium priority, and it will be triggered by an external coupler event from the EtherCAT bus to synchronize with the telegrams.

The third task will be the main task that handles the camera and tablet communication as well as the ball position controllers. The main task will have lowest priority and will be freewheeling. The main task will thus execute continuously unless preempted by the other tasks.

The PID controllers should, however, not be allowed to freewheel in their execution as that may lead to them being executed too fast compared to the position measurement sample rate, which might cause problems (see Section 2.3.2). A timer will be used to control how often the control signals will be updated at a minimum.

4.8 Tablet Device Implementation

4.8.1 Overview

The tablets used are Android-based and the control application has therefore been implemented in Java. The implementation was based on an existing partial implementation which only supported angle control, but most of it was redone to match the updated functionality requirements. Here follows some information about the classes used. For a full walk-through of the different views in the application, refer to Appendix A.1.

4.8.2 Details

This section is based on Figure 4.9.

The `MainActivity` provides the initial start screen where application settings can be altered. The communication with the PLC does not start until the “Play” button is pressed, which starts the `GyroscopeActivity`.

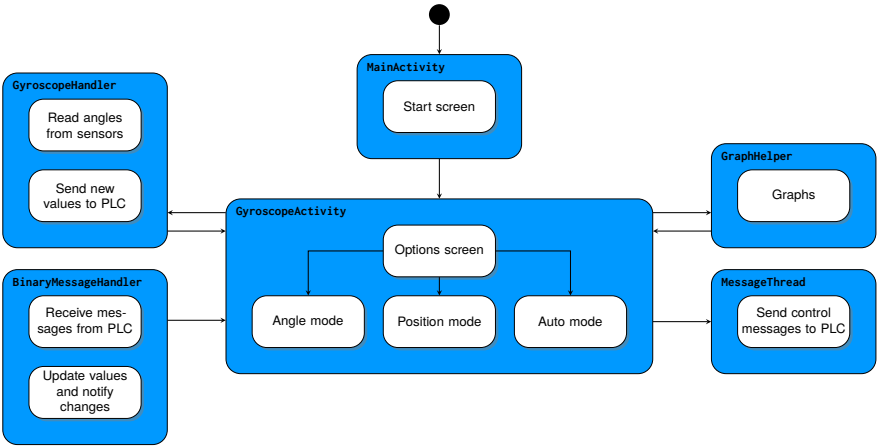


Figure 4.9 Overview of the tablet implementation classes and data flows

The `GyroscopeActivity` is the actual main activity which holds together all components of the application including user input and graphical output. For communication there are two main classes, a `BinaryMessageHandler` for receiving messages and a `MessageThread` for sending control messages to the PLC.

The `GyroscopeHandler` class extends `AsyncTask` which is used to handle UI operations in the background of an `Activity` (in this case the `GyroscopeActivity`). By accessing the data from the tablet accelerometer, it can read at what angle the tablet is held. This angle is also the output to the PLC. This class is also used to send position reference values. When controlling the ball position, the value sent is calculated by simply adding the angle values in each direction to the previous position. Optionally one can set a gain for the angle and position to get a more or less sensitive control.

The `BinaryMessageHandler` class also implements `AsyncTask` and is used for receiving all messages from the PLC. The content of a message is parsed using a helper class and then sent forth to the content-corresponding function in the `GyroscopeActivity`. This class is also responsible for keeping up the connection by sending heartbeat messages in response to the ones from the PLC.

The `MessageThread` has a queue for sending connect and mode change messages from the `GyroscopeActivity` to the PLC. Since the messages are UDP datagrams there is no direct confirmation of reception on the other end; therefore, the messages will be sent again and again until confirmation is received via the `BinaryMessageHandler`. The messages are then deleted from the `GyroscopeActivity`.

The GraphHelper class holds all the graphs for the UI, including both the actual graphs and the ball position graph. Most communication is sent from the GyrosopeActivity to the GraphHelper, for example, initialization of graphs and value updates, but some data is also collected from here, i.e. the initial reference position for the ball, which has to be prespecified for the camera software to initialize.

Automatic Mode To give the system its self-playing ability, setpoints are read from a text file using the Pathfinder. The setpoints are added to a list which is returned to the GyroscopeActivity, which starts an instance of BallGuide taking the list as input. The BallGuide will then go through the list sending a new setpoint as the ball closes in on the current one. The actual position is constantly updated and the error between that and the setpoint has to be below a specified threshold before the next setpoint is sent.

5

Results

5.1 Camera Implementation & Tracking Algorithm

5.1.1 Parameters

The tracking algorithm parameters were tuned by hand to reduce false positives (or noise) with a fast response time. The optimal values are found in Table 5.1. The Kalman filter predictions did not improve the localized search, and hence, no filter parameters are included in the table.

5.1.2 Performance

To achieve the maximum possible camera frame rate of 30 frames per second, the tracking algorithm has to meet a deadline of ≈ 33 ms per frame. Figure 5.1 shows a cumulative normal distribution plot of the time required for the tracking algorithm to process a single frame using a resolution of 320×240 pixels, corresponding to 3 pixels per cm, under varying movement of the tracked ball. The processing speed at this resolution averages 19 milliseconds and is skewed towards smaller values. According to the plot, the deadline should be met with a probability of almost 99 %.

Parameter	Symbol	Value
Local area size	-	16-24 pixels
Glare threshold	T_g	0.4-0.5
Search threshold	T_f	0.06-0.08
Short-term background model threshold	T_l	0.05
Short-term background model adaptation rate	α_s	0.7-0.8
Long-term background model threshold	T_h	0.1
Long-term background model adaptation rate	α_l	0.1

Table 5.1 Tracking algorithm parameters

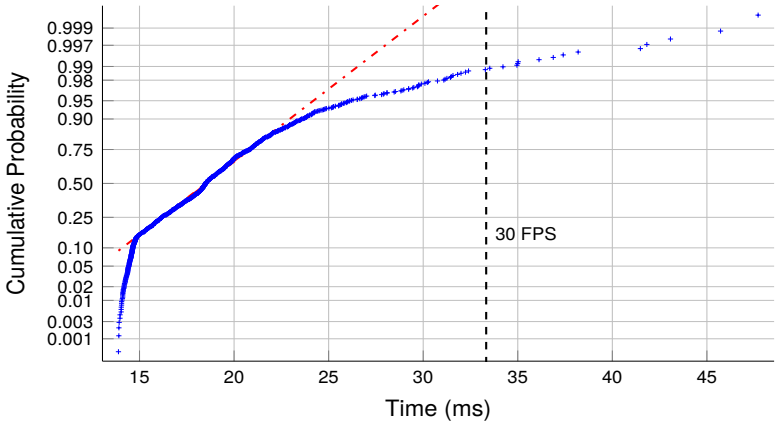


Figure 5.1 Tracking algorithm time per frame

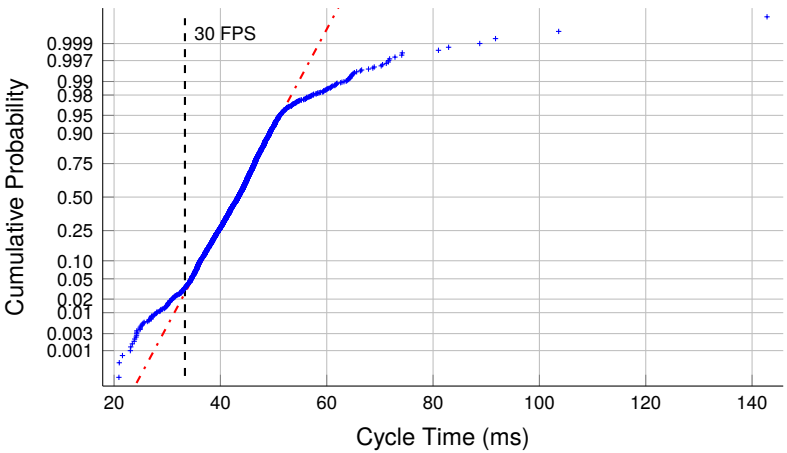


Figure 5.2 Camera tracking algorithm cycle time

Figure 5.2 shows a cumulative normal probability plot of the total cycle time required for the camera to get the next image pointer, perform the tracking and send the resulting position (if a new position is found) using the same resolution of 320×240 pixels under varying movement of the tracked ball. The total cycle time is on average slightly more than twice as large as the actual tracking algorithm processing time, implying a smaller frame rate than 30 frames per second. At 95 % probability, a frame rate of ≈ 20 frames per seconds is achieved.

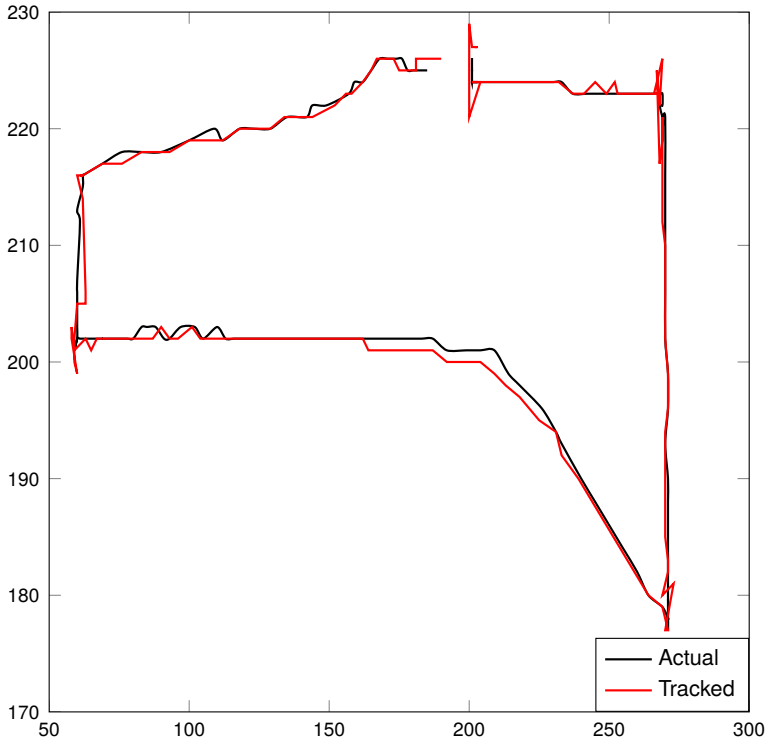


Figure 5.3 Tracked ball position in pixels compared to actual ball position

Figure 5.3 shows a sample tracking scenario with the actual path traveled by the ball against the positions reported by the tracking algorithm. As seen in Figure 5.4, the absolute error distance between the tracked position and the actual position is most commonly one pixel large, with larger errors being uncommon.

5.1.3 Latency

To evaluate the network performance, the latency between the camera and the PLC was measured with the results as a cumulative normal probability plot in Figure 5.5. The latency seems fairly normal distributed with a low mean of approximately 0.9 ms.

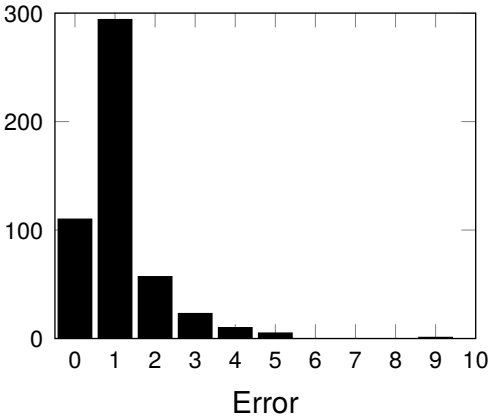


Figure 5.4 Histogram showing the absolute tracking error in pixels

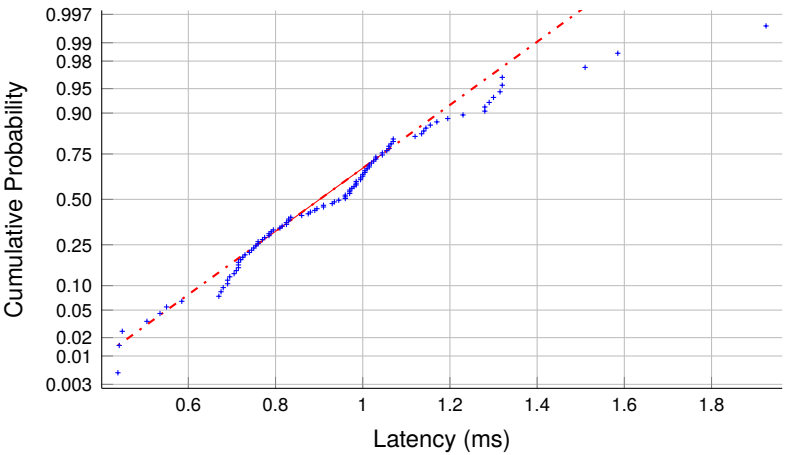


Figure 5.5 Latency between camera and PLC

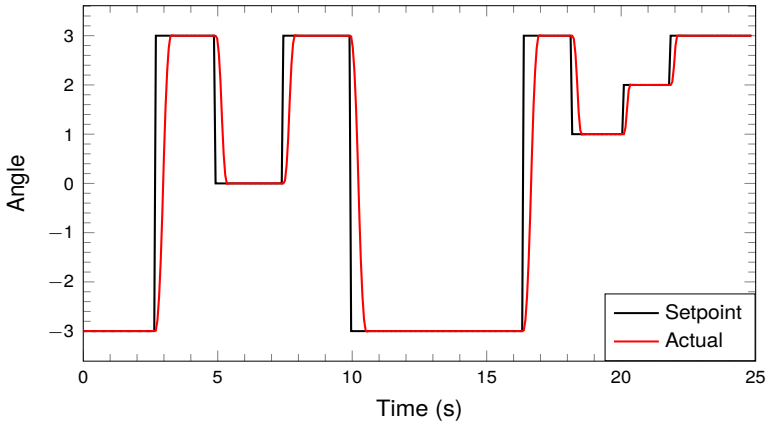


Figure 5.6 Angle control of *x*-axis

5.1.4 Scheduling

The execution times of the emergency and motion control tasks of the PLC were very low and did not cause any performance loss from preemption of the task handling the PID controllers and the client communication. Potential issues can however occur if a client sends too much data and the datagram buffers are filled faster than they can be processed and depleted.

5.2 Control

The performance of the control loops was evaluated for both angular control and ball position control with the complete physical setup. The results are presented below.

5.2.1 Angle Control

The performance of the servo drive angle control for some angle steps is seen in Figure 5.6. The maximum output is $\pm 3°$. All parameters and performance are equivalent for the two axes, why only one axis is addressed here.

5.2.2 Position Control

The used values for the ball position control loops for both axes are listed in Table 5.2. The controller gain is set relatively low but with a large derivative gain to prevent overshooting. The integral time is used primarily to overcome friction when reference changes are small. The maximum servo acceleration/deceleration is chosen so that the response is fast without loss of stability.

Parameter	Symbol	Value
Proportional gain	K	0.08
Integral time	T_i	10
Derivative gain	T_d	1.1
Integral reset time	T_t	30
Maximum derivative gain	N	20
Setpoint weight	β	1
Maximum servo acceleration/deceleration	-	$120 \cdot 3.6°/s^2$

Table 5.2 Position control PID parameters

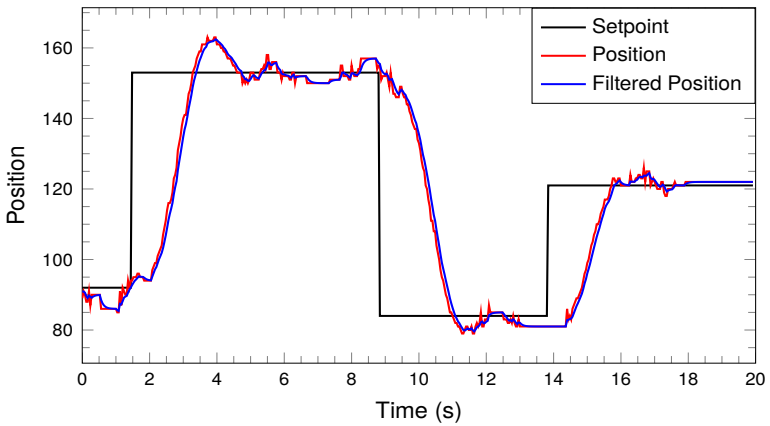


Figure 5.7 Position control in x-direction

The performance of the ball position control is shown in two examples in Figures 5.7 and 5.8, which show the responses of the loops for different reference signal changes. These figures clearly show that the ball position measurement signal is rather noisy but that the low-pass filter removes high frequencies that have a negative impact on the derivative action. The derivative action limits overshooting, but this also results in a limited response time.

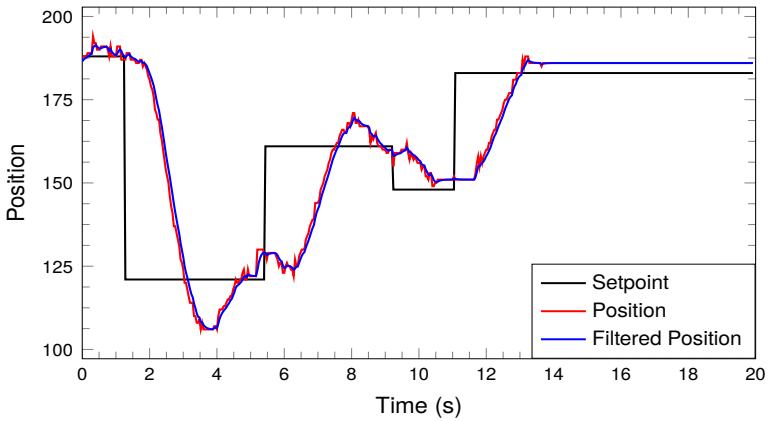


Figure 5.8 Position control in *y*-direction

6

Discussion

6.1 Tracking Algorithm & Camera Implementation

Tests with the tracking algorithm showed that the Kalman predictions used to determine the location for the localized search in the next frame did not show any tracking improvement as the used model only handles linear movement. Any sudden setpoint changes or wall collisions would result in bad predictions; thus, the size of the search area could not be reduced further and the Kalman filter predictions were removed from the final implementation.

A Kalman filter with linear movement could, however, be useful to follow the ball position in a scenario without obstacles. Also, if the ball should not always be visible for the camera, a Kalman filter can be used to estimate the position as long as the ball is hidden. There also exist extended Kalman filters which support more complex forms of movement, such as e.g. Brownian motion. This was, however, not investigated further since the tracking performed well without Kalman filtering and there was no desire to increase the computational load on the camera processor.

At the initial state of the project there were thoughts on implementing a correction for the barrel distortion seen in the images of the system, i.e. the lines at the edges are not straight but slightly bulged out. The distortion makes the ball location differ slightly from the actual position. At the end the conclusion was that it was not needed since it had such small impact on the measured position and that the ball positioning control would compensate for it.

6.2 PID Tuning

6.2.1 Servo Drives

Initially, the frame was attached directly to the motors without any gear-boxes. This meant that the motors only had a very small position range to operate in, and also, that they had to handle more inertia. When automatically tuned, the motors performed badly and the system would often start to oscillate on reference angle changes except very small ones. Also, any load disturbances on the frame would quickly make the

system oscillate. By manually reducing the controller gains, it was possible to control the angles in a limited range and with a limited controller speed. It was, however, not possible to add the additional ball position control loop without the system becoming unstable.

By adding gearboxes with a 1:100 scaling to the servo motors, the automatic tuning of the servo drives produced a very accurate angle control with fast response times. The gearing increases the operating range of the motors and reduces the torque required to control the game board angles.

6.2.2 Ball Position Loop

By low-pass filtering the ball position signal before feeding it to the controller, the derivative performance of the controller improved.

One important parameter for the stability of the position loop turned out to be the maximum allowed acceleration and deceleration of the servo motors. A low acceleration limit would mean that the response time of the loop was not fast enough to prevent large overshoots. However, a large limit would result in rapid decelerations that would cause vibrations in the entire frame. These vibrations are a result of the connections between the motors and the frames being slightly flexible with pin-connected joints. A more solid construction could possibly eliminate the vibrations and allow for higher accelerations and decelerations without loss of stability.

6.3 Network Delay Considerations

Since all components are physically close, there should not really be any significant bottlenecks introduced by network latency or delays. Also, the measured latency between the camera and PLC was very low. The speed of the tracking algorithm is the major contributor to delays.

6.4 Control

6.4.1 Angle Control

The performance of the angle control performed by the frequency drivers is adequate. The delays seen in Figure 5.6 are expected since the control signal is taking steps between the minimum and maximum output angle.

6.4.2 Ball Position Control

Looking at the plots in Figures 5.7 and 5.8, one can see quite long rise times. This is expected, since the system (i.e. the ball) is quite slow. The thing that needs to be avoided is overshoot and especially disturbance in the axis not responsible for the current movement when doing a one-dimensional move. The disturbance comes from the camera signal incorrectly estimating the exact center of the ball.

Something else that is not optimal is the locked position seen last in Figure 5.8, where the ball stops near the setpoint. This is due to friction, i.e. when the ball is still it needs a push to start moving again. A conclusion from this is that the integral action is not always able to satisfy its purpose. It would be good to implement some functionality that makes it possible to raise the gain of the integral action or implement some friction compensation, for example, *dither*.

One issue with the ball position control is that the only inclinometer available stopped functioning. This meant that the angle calibration had to be done manually and measured by the servo drives. An angle of zero degrees would therefore not necessarily be exactly zero degrees; thus, an output control signal with value zero would most often not result in the labyrinth plane resting completely horizontally. This stationary error was hard to overcome if the calibration differed to much from reality, as the proportional action would too large to be compensated for.

7

Conclusions

7.1 Summary

In this thesis, a background subtraction based algorithm for tracking a moving ball on a moving labyrinth plane has been designed. The algorithm has been used to implement a camera-based sensor that feeds measured positions to a controller in a distributed network. Using a cascaded PID control loop, the controller has been able to control the ball position on the labyrinth plane using reference signals from one or two tablet devices also connected to the network. A predefined setpoint layout can also be used to make the labyrinth self-playing.

7.2 Stability Issues

The system in its original state, without the gearboxes, would not have made the control feasible. Despite large efforts on controlling the ball position without the gearboxes, no satisfactory results were achieved. With the gearboxes added, the system performs well.

7.3 Pathfinding

The investigated pathfinding algorithm, A*, is implemented but not used in the clients due to time limitations and the perception that a simple list of prespecified setpoints would suffice for the application. There is also a binary map of the Alten maze available in the code.

7.4 Camera Vision

A drawback of using a transparent labyrinth is that the algorithm used for distinguishing the ball needs to subtract everything that is not moving. This means that if the ball moves very slowly the tracking might be lost due to high threshold values on the images. Another drawback is that the algorithm might favor one side of the

ball due to surrounding light sources. A brighter side of the ball will be easier to find, which makes the tracking somewhat biased and the positioning inexact. If the background was solid and without distractions, a gentler thresholding could be used. This would make the whole ball visible in the filtered image resulting in a more exact positioning. Even though it would be nice with better tracking, a non-transparent labyrinth would defeat the purpose of the labyrinth as a concept for showcasing the control system.

The combination of a short-term and a long-term model to form the background model used has been shown to be successful. The fast responses of the short-term model combined with the insensitivity to noise of the long-term model results in a background model combining these properties.

7.5 Further Work

There are a number of possibilities for future work to improve the performance of the labyrinth control:

- Increase stability by implementing new control method or observer.
- Replace camera with new one with more computational power (e.g. ARTPEC-5) and possibly higher frame rate.
- Implement an algorithm for finding the ball, such that no initial position has to be chosen.
- Implement an algorithm that automatically finds and zooms in on the labyrinth so that the camera placement does not affect the game.
- Implement friction compensation for the ball to start rolling when the ball velocity is zero.
- Construct new joint arms to replace the shaky ones connecting the motors and the gimbal frames.

By construction new arms between the motors and the frames it should be possible to avoid vibrations that currently occur. This would then allow control parameters to be changed so that the control can be performed faster without losing stability.

A new camera with more computational power could allow for a higher frame rate and a higher image resolution. A higher image resolution would provide more accurate control of the ball position as there would be more pixels per length unit. The higher frame rate would, if not limited by other factors, provide faster control of the system.

Bibliography

Andrews, G., C. Colasuonno, and A. Herrmann (2004). *Ball on plate balancing system*. Project report.

Cuevas, E., D. Zaldivar, and R. Rojas (2005). *Kalman filter for vision tracking*. Tech. rep. B 05-12. Institut für Informatik, Freie Universität Berlin.

Espersson, M. (2013). *Vision algorithms for ball on beam and plate*. Masters thesis. Department of Automatic Control; Lund University.

Öfjäll, K. (2010). *LEAP; A Platform for Evaluation of Control Algorithms*. Masters thesis. Linköpings Tekniska Högskola.

Glad, T. and L. Ljung (2003). *Reglerteori : flervariabla och olinjära metoder*. Lund : Studentlitteratur. ISBN: 9144030037.

Group, E. T. *Ethercat*. Gathered on 2014-05-25. URL: `http://www.ethercat.org/en/technology.html`.

Gruenwedel, S., N. Petrovic, L. Jovanov, J. Niño-Casta-Ñeda, A. Pižurica, and W. Philips (2013). "Efficient foreground detection for real-time surveillance applications." *Electronics Letters* **49**:18, pp. 1143–1145. ISSN: 00135194.

Hart, P., N. Nilsson, and B. Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths." *Systems Science and Cybernetics, IEEE Transactions on* **4**:2, pp. 100–107. ISSN: 0536-1567.

Hägglund, T. (2000). *Reglerteknik AK: Föreläsningar*. Department of Automatic Control; Lund University.

Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems." *Transactions of the ASME – Journal of Basic Engineering* 82 (Series D), pp. 35–45.

Karmann, K.-P. and A. von Brandt (1990). "Moving object recognition using an adaptive background memory." In: Cappellini, V. (Ed.). *Time-varying Image Processing and Moving Object Recognition,2*. Elsevier Publishers B.V., pp. 297–307.

Levinson, S. E., A. F. Silver, and L. A. Wendt. *Vision Based Balancing Tasks for the iCub Platform: A Case Study for Learning External Dynamics*. Tech. rep. Department of Electrical and Computer Engineering; The University of Illinois at Urbana-Champaign.

Lin, C. E. and W. C. Huang (2012). "Dynamic ball and plate control verification on magnetic suspension platform using enforced fuzzy logic control." *International Journal of Applied Electromagnetics & Mechanics* **40**:4, pp. 259 –281. ISSN: 13835416.

McFarlane, N. and C. Schofield (1995). "Segmentation and tracking of piglets in images." *Machine Vision and Applications* **8**:3, pp. 187 –193.

Nilsson, J. (1998). *Real-time control systems with delays*. Department of Automatic Control, Lund Institute of Technology: 1049. Lund offset.

Piccardi, M. (2004). "Background subtraction techniques: a review." In: *Systems, Man and Cybernetics, 2004 IEEE International Conference on*. Vol. 4, 3099–3104 vol.4.

RAPP User's Manual. Gathered on 2014-05-25. URL: http://www.nongnu.org/rapp/doc/rapp/.

Ridder, C., O. Munkelt, and H. Kirchner (1995). "Adaptive background estimation and foreground detection using kalman-filtering." In: *Int. Conf. Recent Advances Mechatronics 12*, pp. 193–199.

Årzén, K.-E. (1999). "A simple event-based pid controller." In: *Preprints 14th World Congress of IFAC*. Beijing, P.R. China.

Årzén, K.-E. (2012). *Real-Time Control Systems*. Department of Automatic Control; Lund University.

Sen-ching S. Cheung and C. Kamath (2007). *Robust techniques for background subtraction in urban traffic video*.

Åström, K. J. (2008). "Event based control." English. In: Astolfi, A. et al. (Eds.). *Analysis and Design of Nonlinear Control Systems*. Springer Berlin Heidelberg, pp. 127–147. ISBN: 978-3-540-74357-6. DOI: 10.1007/978-3-540-74358-3_9.

Szeliski, R. (2011). *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer London, 2011. ISBN: 9781848829350.

Toyama, K., J. Krumm, B. Brumitt, and B. Meyers (1999). "Wallflower: principles and practice of background maintenance." *Proceedings of the Seventh IEEE International Conference on Computer Vision* 1, p. 255. ISSN: 9780769501642.

User Manual for PLC Programming with CoDeSys 2.3 (2010). Version 6.0, CoDeSys V.2.3.9.24. 3S - Smart Software Solutions GmbH.

User's manual: MicroFlex e150 servo drive (2013). LT0291A04EN. ABB Oy.

VAPIX®. Gathered on 2014-05-25. URL: http://www.axis.com/techsup/cam_servers/dev/cam_http_api_index.php.

Wittenmark, B., K. J. Åström, and K.-E. Årzén (2012). *Computer Control: An Overview, Educational Version 2012*. Department of Automatic Control; Lund University.

Wood, J. (2007). *Statistical Background Models with Shadow Detection for Video Based Tracking*. Masters thesis. Linköpings Tekniska Högskola.

A

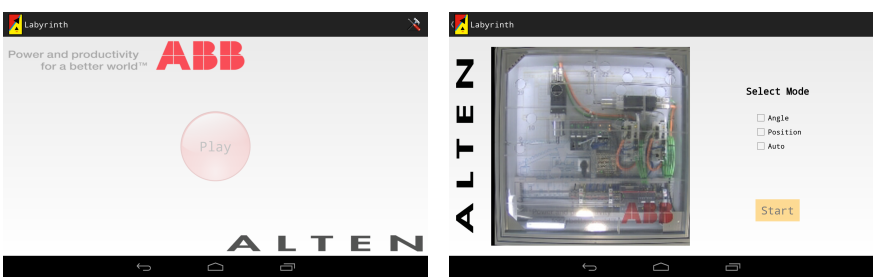
Appendix

A.1 Labyrinth – Android Application

Here follows a walk-through of the android application for the labyrinth game. The first screen that meets the player when launching the application is the start screen of the `MainActivity` as seen in Figure A.1(a). From here the settings menu is available where the player can adjust some parameters, such as the angle or position gain. When the "Play" button is pressed the `GyroscopeActivity` will be launched.

A.1.1 Start Screen

When the `GyroscopeActivity` is launched, it sends a connect message to the PLC and waits for acknowledgment. The PLC responds with a message that tells whether the system is ready or not. If it is not ready, there will be an error message telling that the motors need to be calibrated. If the system is ready and in level mode, the screen in Figure A.1(b) will be shown. Here the player chooses the initial ball position along with the desired game mode. When the start button is pressed, a new connect message will be sent to the PLC with the desired mode and initial ball position. The PLC acknowledges the message by a responding status message with the new mode



(a) `MainAcitivity` **(b)** `GyroscopeActivity`

Figure A.1 Start screens

and the selected mode layout appears on the screen. Whenever the player is in game mode, one can return to the start screen by pressing the back button in the bottom of the screen. This sends a new connect message telling the system to level out and stops the `GyroscopeHandler` so that no new setpoints are sent. The "Labyrinth" button at the top of the screen will be visible during all game modes. By pressing it the `GyroscopeActivity` is closed and the application returns to the `MainActivity` start screen.

A.1.2 Angle Mode

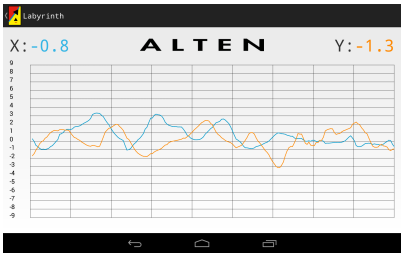
If the player chooses to play the angle mode the first screen will be Figure A.2(a). The values plotted are the x and y angles of the device with added gain and color coding according to the values shown in the top left and right corner. For easier control, a low gain is recommended. If the player taps the screen once, the second angle mode layout, shown in Figure A.2(b), will appear. Here there are two plots, one for x and one for y, showing the tablet angles and the angles of the plate. If the screen is tapped again, the third angle layout appears (see Figure A.2(c)). Here the two plots from the previous screen are mashed together and shown beside a camera image of the labyrinth, and the current tracked ball position is marked by a red cross. If the screen is tapped a third time, the first angle layout in Figure A.2(a) will appear again. Whenever a new screen is shown or something changes in the system, there will be a "toast" shown, as seen in Figure A.2(b), telling what is shown on the screen.

A.1.3 Position Mode

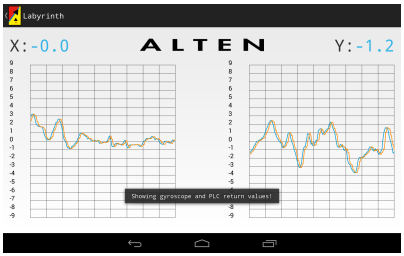
If the player chooses position mode, the layout will be that of Figure A.2(d). When in position mode, the `GyroscopeHandler` will send position setpoints by adding the current angle values of the tablet to the current position. This will move the position setpoint in a "ramping" way that ensures stability of the PLC controller. There is also a possibility to tap the screen to specify a setpoint.

A.1.4 Auto Mode

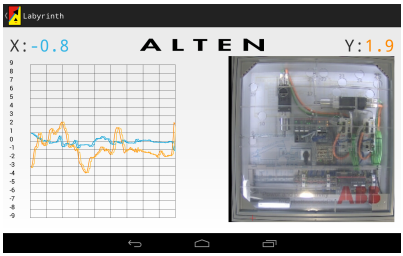
In auto mode, the player chooses which maze is used (see Figure A.2(e)). When a maze is chosen, some buttons will appear as in Figure A.2(f). Pressing "Calculate" will make the `Pathfinder` read the list of setpoints provided from a text file. Pressing "Start" will make the `BallGuide` start to move along the list of setpoints. The "Clear" button is used if the player wants to stop the path calculations or stop the `BallGuide` from executing the whole list of setpoints.



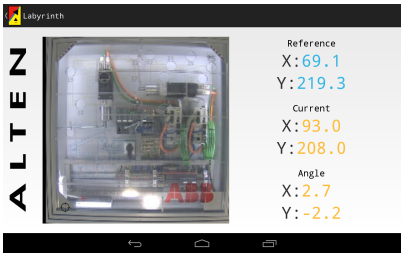
(a) Angle – Big graph



(b) Angle – Split graph

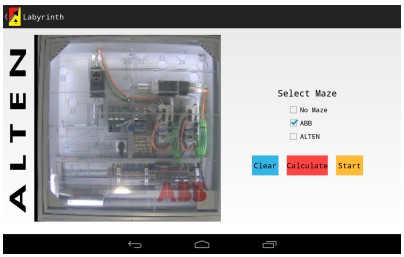


(c) Angle graph and camera image



(d) Position mode

(e) Auto mode



(f) Auto mode

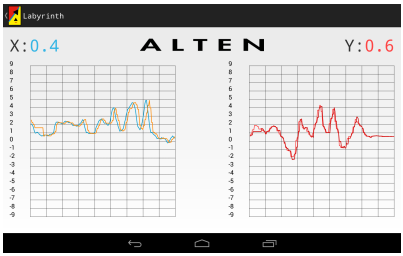
Figure A.2 Game mode layouts

A.1.5 Multiplayer

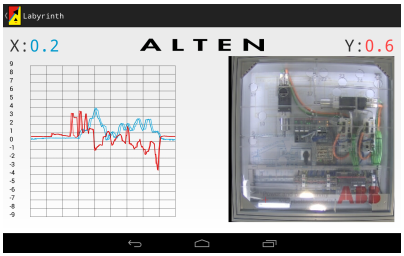
When playing in multiplayer mode, it is recommended to connect one tablet first and wait for the start screen. Then, the second player connects and gets the second player start screen. The start screens of the two players will now be the ones seen in Figure A.3(a) and A.3(b). The first player to connect will be in control of the x-axis as indicated by the arrow on the start screen, and the second player will be in control of the y-axis. When playing in angle mode, there are two layouts available. The split screen seen in Figure A.3(c) showing the other players reference value in red, and the layout with a mashed plot and a camera image seen in Figure A.3(d). The position mode layout is almost identical to the single player case except that the other players reference value is shown in red as in Figure A.3(e).

(a) First player start screen

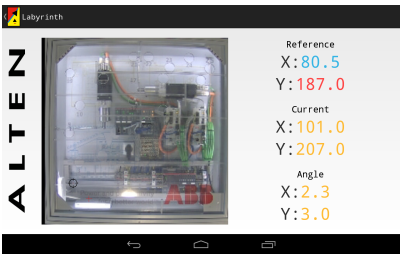
(b) Second player start screen



(c) Multiplayer split graph



(d) Multiplayer angle graph and camera image



(e) Multiplayer position mode

Figure A.3 Multiplayer layouts

A.2 Communication

A.2.1 Tablet to PLC

The UDP datagrams sent from the tablet to the PLC can have data lengths of one, three or five bytes depending on the message sent. The first byte of every message is a control letter that defines what action to take, except for the heartbeat, which only consist of one byte with value zero. Table A.1 contains details on the datagrams sent from the tablet devices to the PLC.

Message Type	Byte	Value	Description
Connect to PLC	0	C	
Heartbeat response	0	0	
Change to angle mode	0	A	
	1		X high
	2		X low
	3		Y high
	4		Y low
Change to ball position mode	0	P	
	1		X high
	2		X low
	3		Y high
	4		Y low
Change to off mode (level out)	0	O	
New angle values	0	B	
	1		X high
	2		X low
	3		Y high
	4		Y low
New position values	0	Q	
	1		X high
	2		X low
	3		Y high
	4		Y low

Table A.1 UDP datagrams sent from tablet to PLC

A.2.2 PLC to Tablet

When the PLC receives a connect message or if there is a change of some sort, for example when another player connects, a status update message is sent to the tablet as a confirmation. The standard in-game message sent is the measured values message, which tells the angle of the plate and the position of the ball. When in multiplayer mode, the measured values message is sent along with the setpoint from the other player so that the angle or position setpoint of the other player's device is available for plotting. See Table A.2 for details on the datagrams sent from the PLC to tablet devices.

Message Type	Byte	Value	Description
Connect message / Status update	0	S	
	1		Start / reset done
	2		Current mode
	3		Multiplayer
	4		Player number
Heartbeat	0	0	
Measured values	0	M	
	1		X high
	2		X low
	3		Y high
	4		Y low
	5		X high
	6		X low
	7		Y high
	8		Y low
Setpoint from other player	0	N	
	1		Setpoint high
	2		Setpoint low

Table A.2 UDP datagrams send from PLC to tablet

A.2.3 Camera to PLC

See table A.3 for details on the datagrams sent from the camera to the PLC.

Message Type	Byte	Value	Description
Ball position	0		X high
	1		X low
	2		Y high
	3		Y low
Heartbeat	0	0	
Start acknowledgment	0	A	
Stop acknowledgment	0	S	

Table A.3 UDP datagrams from camera to PLC

A.2.4 PLC to Camera

See table A.4 for details on the datagrams sent from the PLC to the camera.

Message Type	Byte	Value	Description
Start message without initial position	0	0	
Start message with initial position	0		X high
	1		X low
	2		Y high
	3		Y low
Stop message	0	S	

Table A.4 UDP datagrams from PLC to camera

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER´S THESIS

Date of issue
June 2014

Document Number
ISRN LUTFD2/TFRT--5948--SE

Author(s)
Daniel Persson
Fredrik Wadman

Supervisor
Anton Cervin, Dept. of Automatic Control, Lund University, Sweden
Alfred Theorin, Dept. of Automatic Control, Lund University, Sweden
Karl-Erik Årzén, Dept. of Automatic Control, Lund University, Sweden (examiner)

Sponsoring organization

Title and subtitle

Self-Playing Labyrinth Game Using Camera and Industrial Control System

Abstract

In this master's thesis, an industrial control system together with a network camera and servo motors were used to automate a ball and plate labyrinth system. The two servo motors, each with its own servo drive, were connected by joint arms to the plate resting on two interconnected gimbal frames, one for each axis. A background subtraction-based ball position tracking algorithm was developed to measure the ball-position using the camera. The camera acted as a sensor node in a control network with a programmable logical controller used together with the servo drives to implement a cascaded PID control loop to control the ball position. The ball reference position could either be controlled with user input from a tablet device, or automatically to make the labyrinth self-playing.

The resulting system was able to control the ball position through the labyrinth using the camera for position feedback.

Keywords
ball and plate, computer vision, background subtraction, PID, object tracking

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-76

Recipient's notes

Security classification