CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING

DIPLOMA THESIS

Positional Visual Feedback Control

Department of Control Engineering

Prague, 2007 **Author: Tomáš Němec**

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 18. 1. 2007

[illegible]
podpis

Poděkování

Na tomto místě bych rád poděkoval všem, kteří mě při studiu i vzniku této práce podporovali.

Především pak vedoucímu práce Ing. Pavlu Burgetovi za vstřícnost a cenné připomínky a rady nejen při pracech na projektu, ale i při vytváření této zprávy.

Velké díky patří rovněž manželce za trpělivost a motivaci k práci a také rodině, za psychickou a materiální podporu v průběhu celého mého studia.

Abstrakt

Diplomová práce se zabývá zavedením polohové vizuální zpětné vazby do řízení modelu přehazujícího kulečníkové koule. Byl sestaven stavový model systému, tvořeného jednou vyhozenou kulečníkovou koulí. Na základě tohoto modelu byl navržen Kalmanův filtr, odhadující stav systému. Data jsou do Kalmanova filtru dodávána z rychlé průmyslové kamery. Z odhadovaného stavu je počítán vhodný bod zachycení, který je posílán do řídicího PLC. Komunikace s modelem používá protokol TCP/IP. Pro rychlé zpracování obrazu byla použita knihovna HALCON. Výsledek práce je demonstrován funkčním programem ve Visual C++, umožňujícím modelu díky zavedené polohové vizuální zpětné vazby přehazovat kulečníkovou kouli. Závěr práce je věnován návrhu bezpečnostních prvků podle EN 954-1, kategorie 3.

Abstract

This diploma thesis deals with closing a positional visual feedback to the control of model throwing billiard balls. The state model of the system was constructed, for one thrown billiard ball. Based on this state model, a Kalman filter, estimating the system's state, was designed. A high speed industrial camera supplies the Kalman filter with the measured data. From the observed system state, the suitable catch point is calculated and sent to the control PLC. The communication with the model uses standard TCP/IP protocol. For fast image processing, HALCON library was utilized. The result of this project is a working application in Visual C++, which thanks to visual feedback from the camera juggles with one billiard ball. The last chapter of this thesis is devoted to design of safety elements according to standard EN 954-1, category 3.

České vysoké učení technické v Praze - Fakulta elektrotechnická

Katedra řídicí techniky ***Školní rok:*** 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Tomáš N ě m e c

Obor: Technická kybernetika

Název tématu: Řízení polohy pomocí vizuální zpětné vazby

Zásady pro vypracování:

1. Seznamte se s možnostmi snímání polohy letících předmětů v omezeném prostoru pomocí rychlé kamery.
2. Ke stávajícímu modelu připojte kameru a uzavřete tak zpětnou polohovou vazbu v řídicím systému. Z naměřených dat vytvořte matematický model pohybu předmětu a využijte jej k určení polohy předmětu ve stanoveném čase.
3. Proveďte taková opatření, která zajistí, aby byl model dostatečně robustní a aby jej bylo možné bezpečně použít ve výuce.

Seznam odborné literatury: Dodá vedoucí práce.

Vedoucí diplomové práce: Ing. Pavel Burget

Termín zadání diplomové práce: zimní semestr 2005/2006 (změna zadání: 18.12.2006)

Termín odevzdání diplomové práce: leden 2007

[illegible]

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

L.S.

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-8-

1 zast. [illegible]

prof. Ing. Zbyněk Škvor, CSc.
děkan

V Praze dne 21.12.2006

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This diploma thesis continues in project begun by (Pšenička, J. 2005), where a mechanical model of a tool capable of throwing and catching of billiard balls was constructed. The mechanical model consists of four servo drives controlled by four B&R servo amplifiers and one B&R PowerPanel PLC. The teaching tool was designed to demonstrate fast and accurate motion control of the four drives by throwing and catching a billiard ball through a steel ring or by juggling with one or two billiard balls.

1.1 Control structure

The control was done either with NC actions and timers, where the corresponding movements were preprogrammed in the control PLC or the more complex movements, as throwing a ball from one side of the model to another, were realized as cam profiles, saved in ACOPOS servo amplifiers. In both cases, there is no feedback to the controlling PLC from the real system, which can result in inaccuracy caused by variable static friction, the ball's rotation and other events resulting in slight changes of the flight trajectory.

In my diploma thesis I am describing a method of closing a positional visual feedback from a high speed CMOS camera. The method was inspired by similar works for instance by (Frese, U. et al. 2000) and in the terminology of visual servoing it is a position-based system with end-effector open loop. The flying ball's trajectory is modeled by a Kalman filter, which, based on the measured data, estimates its state. The ball's state is described by its current position and velocity. From the estimation a suitable catch-point is predicted and the servos' positions adjusted according to the prediction. Image processing is done with HALCON image processing library which is widely used in various industry branches.

1.2 Practical use

Although this project will serve as a teaching and demonstration tool at Department of Control Engineering of CTU in Prague, the applied principles can be used and are used in industry. Visual feedback or directly visual servoing have a great potential when utilized in robotic manipulators for fast automatized object sorting on conveyor belt, in the stock, by welding etc.

With increasing computational power of computers and high speed image acquisition hardware, the visual feedback control will be used more often even in systems, where real time control is needed.

1.3 Structure of this work

Chapter 2 of this work sums up some basic theory to visual servoing problematics. The crucial works are mentioned and the most common control structures of visual feedback described.

The following three chapters are an overview of techniques utilized in this work. Chapter 3 discusses the vision system. In its first part the image acquisition hardware is introduced, the second part of the chapter is devoted to image processing software. Among other, calibration principles are explained. The fourth chapter is a very short introduction to Kalman filtering. The necessary steps to Kalman Filter design and tuning are presented and the catch point predictor is worked out. Chapter 5 contains the necessary information to the communication interface used to connect the measuring PC to the control PLC.

Chapter 6 gives the reader an idea of the core algorithm principles, on the arisen problems and their solution. Problems like light reflexion, ball gleam, missing measurement due to occlusion, timing, etc. are in detail discussed. The solution of most problems is shown or it is suggested for further model development.

Chapter 7 solves the concrete implementation problems. It uses the knowledge introduced in previous chapters and applies it in the programmed application in Visual Studio 2005. First part of the chapter explains the control structure of the model. Follows the control program structure, where the main classes of the program are explicated.

Chapter 8 dwells on safety functions used in the model. In the chapter's introduction, the basic principles of machine safety are indicated, then the concrete realization of some safety circuits in the project is shown.

Chapter 2

Visual servoing

The term visual servoing has been used since the 1970s. It was introduced by Hill and Park, in 1979. In their work, they tried to distinguish their approach from earlier experiments, where systems alternated between picture taking and moving, so called look-then-move control, instead of having a real closed visual loop.

Visual servoing is combination of several research fields, among which belong for example high speed image processing, control theory, dynamics, kinematics, or real-time computing. The akin technical branches can be considered e.g. active vision, or structure from motion.

A very nice review of the most important works on visual servoing was summarized by (Corke, P. I. 1994). In following section, I will try to introduce briefly some of the works, relative to my project. Some of them are interesting in the means of technology utilized, other are pioneering works in visual servoing and therefore should be mentioned as well. The last two projects were realized lately and are built on commonly available technology for machine vision industry. Because the two last projects solve a similar problem, this thesis was inspired in particular by them.

2.1 Other works

One of the first experiments with closing a visual position loop, were made by (Shirai, Y., Inoue, H. 1973). They used visual feedback loop to increase accuracy of the robot position. From the visual processing methods edge extraction and line fitting were used to find a position of a box to be grasped. The cycle of this pioneering work was as long as 10s.

Another already mentioned work, which utilizes visual servoing and even mentioned this term probably for the first time was done by (Hill, J., Park, W. T. 1979). Binary image processing was used, because of its processing speed and reliability. Simple depth estimation was done by measuring distance between known features. In Hill and Park's experiments planar and 3D visually guided motion is introduced, as well as tracking and grasping of moving parts.

Bolt-insertion task was solved by (Geschke, C. 1981). Interesting on this work was that real stereo vision was replaced by a single camera and a special mirror arrangement. As image processing a thresholding was done, after which an image feature was searched. The sampling frequency of this model was 10Hz.

A real-time camera rocket tracking system was introduced by (Gilbert, A. L. 1980), which was able to reach a sampling frequency of 60Hz with a special image processing hardware.

Interesting work was done by (Webber, T. E. and Hollis, R. L. 1988), who developed a planar position micromanipulator. Visual feedback is used here, to damp the robot's motor vibrations. High accuracy, required by precision manufacturing task, is reached. As an image processing method, correlation was used, with a sampling rate of 300Hz.

Other applications of visual feedback control, dwelling on road vehicle guidance, were introduced in the late 1980s. One of the first works was done by (Dickmanns, E. D. and Graefe, V. 1988), and it describes an experiment of a visually guided vehicle reaching a speed of 96km/h along 20km long test track.

Other experiments try to emulate human skills at ping-pong, juggling, or e.g. ball catching. One of the most known experiments in ball catching was described by (Frese, U. et al. 2000). Their ball catching robot system was built in German Aerospace Center (DLR) and was demonstrated at Hannover Fair 2000. The system is based on DLR robotic arm controlled by a VxWorks PC. The image processing is done with a stereo camera system and is computed on Pentium II PC, running GNU/Linux operating system. To track the flying ball, an Extended Kalman Filter is used, so as to take the air drag into account. The catch point is predicted by numerical integration of the measured data. Sampling frequency of 50Hz was reached. As standard PAL cameras were used, it was not possible to reach higher frequency.

Another interesting application in this field is a High-speed Dartboard (Kormann, B. et al. 2006), which is a students' team project from TU-München. It is a positioning dartboard, which modifies its position according to the flying dart, so that it would hit the dartboard's center. A thrown dart is tracked with two high speed cameras. The dartboard is positioned with fast pneumatic actuators. Image processing is based on MVTec HALCON machine vision. The image processing runs on a double processor AMD PC with 2GB RAM. Although non-real-time operating system Windows XP runs on the PC, very good sampling frequency of 100 Hz is reached.

2.2 Structures of visual servoing systems

The visual servoing systems can be divided into four groups, depending on the structure of the control loop. The classification was introduced by (Sanderson, A. C. and Weiss, L. E. 1980).

If the control architecture is hierarchical and uses the vision system to provide position set-point to the joint-level controller as in figures 2.1 or 2.2, it is referred to as dynamic look-and-move system. On the other hand, direct visual servo structures 2.3 and 2.4, eliminate entirely the robot controller and use visual servo controller to compute joint inputs on its own.

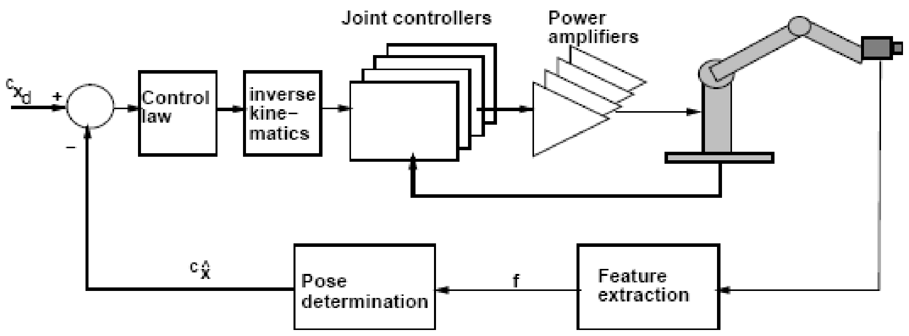


Figure 2.1: Dynamic position-based look-and-move structure. Taken from (Corke, P. I. 1994).

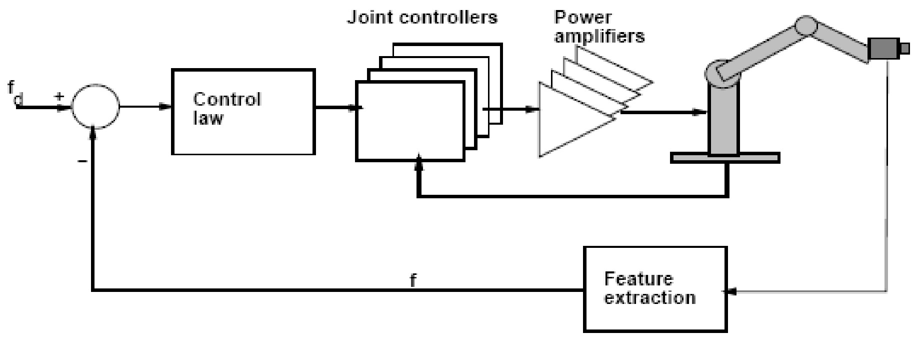


Figure 2.2: Dynamic image-based look-and-move structure. Taken from (Corke, P. I. 1994).

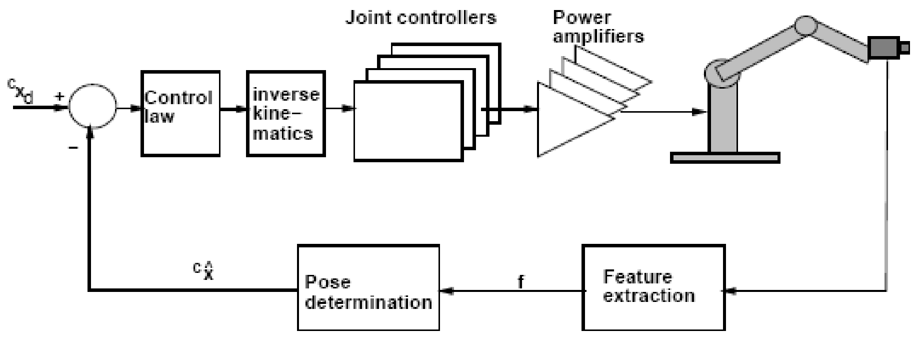


Figure 2.3: Position-based visual servo structure. Taken from (Corke, P. I. 1994).

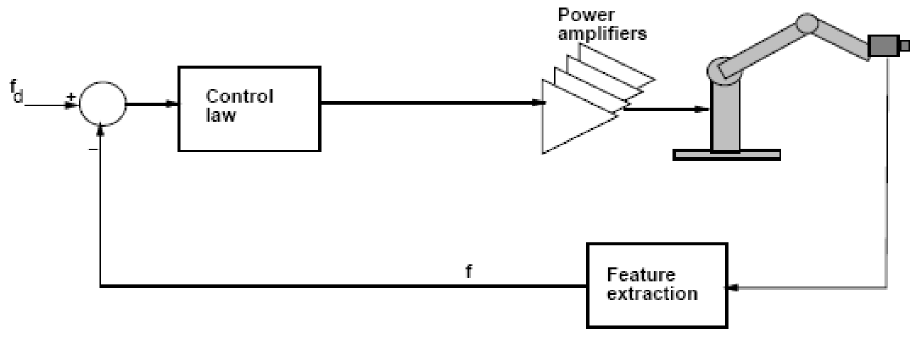


Figure 2.4: Image-based visual servo structure. Taken from (Corke, P. I. 1994).

However, most of the systems built use the look-and-move construction 2.1 or 2.2. There are several reasons for that. The most serious is the complexity of control algorithm, which would be used to directly control the robot's end effector just from vision information. Not speaking about relatively low sampling rates of the vision subsystem. Another reason is that most of the used industrial robots and servo amplifiers already have the joint controller built in, and accept Cartesian velocities and positions on their inputs.

Another classification of visual servoing systems distinguishes position-based and image-based control. The primary difference in both methods is that by position-based method, the image features are extracted and converted into Cartesian coordinate system using the camera internal and external parameters together with the geometric model of the target. The control of servo drives of the manipulator is computed through robot's inverse kinematics, to which as input the coordinates of the robot's end effector are given.

In contrary the image-based system does not need to convert the image features to the world coordinate system, as it uses directly those extracted features, to get the manipulator's end position. The image-based method is better in the means of computational complexity. It even eliminates errors, caused by imprecise camera calibration. However, feedback lead from image-based method is highly non-linear and coupled system.

In addition to this categorization, we can distinguish two other types of visual feedback, depending on what is observed. If only the target is tracked, without measuring the position of manipulator's end-effector, the system is referred to as endpoint open-loop (EOL). If both, the target and end-effector are observed, we speak of endpoint closed-loop (ECL) systems. The main difference between both systems is that the accuracy of EOL systems directly depends on its hand-eye calibration, whereas ECL systems can perform tasks with accuracy independent of hand-eye calibration.

2.2.1 Position-based systems

Position-based systems extract the features from the picture and convert it to the world coordinates. So as to extract the 3D coordinates from 2D image projected to camera sensor plane, some additional information is needed.

This information is in practice gained in various ways. The most common technique is stereo vision. Stereo vision works on principle of taking the picture of the same scene from two different, but known view points. Then, in both pictures the same feature points are searched and from their position the view depth can be extracted.

On similar principle works the so called depth from motion. The different views are taken from a moving single camera along a known trajectory. This principle is used by end-effector mounted cameras of robot manipulators.

Other method, also used in this project, uses photogrammetric techniques. The 3D scene is reconstructed from a single calibrated camera. The geometric model of the searched object, however, must be known. The process of camera calibration is described in chapter 3.

2.2.2 Image-based systems

Image based methods do not depend on camera calibration. Their main advantage is therefore the absence of accuracy distortion caused by imprecise camera calibration or lower computational demands resulted in by no need of extracting of coordinates from the picture etc. However, this method can be used only in rare examples, where the highly non-linear relation between image features and robot set points can be easily described.

Further details discussing various image-based methods can be found for example in (Corke, P. I. 1994).

Chapter 3

Vision system

The vision system generally consists of image acquisition hardware and image processing software. Various types of cameras are used for machine vision. Cameras can work on various sampling frequencies. There are line scan cameras, which acquire images by lines, similarly as an ordinary CRT TV paints the picture on the screen, or area scan cameras, which acquire images from all sensor pixels in the same moment. Other parameters are determined by the chip type (CMOS, CCD, NMOS, CID, etc.), its resolution and pixel size, the data transfer speed from the camera to the frame grabber or the maximum sampling frequency of the camera.

There are two different lens types used in the machine vision tasks. The pinhole lens project the image to camera sensor similarly as the human eye does. The farther the observed object from the camera is, the smaller is the projected picture on the chip plane. We say, that the perspective projection is effected and call the combination of pinhole lens and camera as a pinhole camera model.

The telecentric lens type, in contrary, effects the parallel projection of the world coordinates onto the image plane. This causes objects having the same size in the image independent of their distance, in certain distance range, from the camera. This model is called telecentric camera model.

In the first section I am going to describe my image acquisition hardware. The second section describes the image processing software.

3.1 Image acquisition hardware

In the project, the high quality products of Mikrotron company were used, for image acquisition. Those have shown a great reliability in various industry branches, where high speed machine vision is needed. Camera, as well as the frame grabber work with a great reserve, which can be used in further project enhancements.

3.1.1 Camera

In the context of this chapter's introduction, I am using a pinhole camera model together with a high speed CMOS area scan camera. The resolution of the CMOS chip is 1280 x 1024 px and the sampling frequency 500 fps by full resolution. The sensor dimensions are 15.36(H) x 12.29(W) mm, which makes the chip diagonal of 1,25" with a pixel size of 12 x 12 μm.

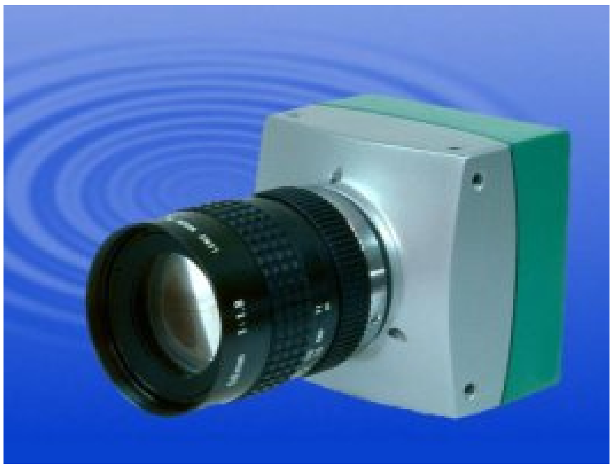


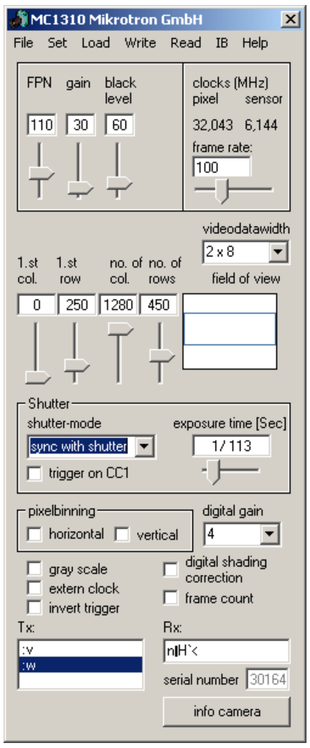
Figure 3.1: Mikrotron MC1310. High speed CMOS camera.

The connection between the camera and the frame grabber is done via Base/Full Camera Link interface, which is designed specially for machine vision tasks. The main difference between Base and Full versions of Camera Link interfaces are in the transfer data width, which is by Base Camera Link 2 x 8 Bit or 2 x 10 Bit, and by Full Camera Link 8 x 8 Bit or 10 x 8 Bit. The maximum data rate reached by Camera Link is 132 MBps by the Base version and up to 660 MBps by Full version[1].

[1]This is video data rate, from the camera, to the frame grabber. With 64-bit PCI-X interface, max. 528 MBps data rate is possible, between camera and computer RAM.

The camera is configured using profiles. A profile is described as a certain configuration stored in camera's registers. MC1310 has fifteen FPGA registers, each 10-bits wide, eight D/A registers, 8-bits wide, and one clock select register, 4-bits wide. There are 17 profiles stored in a nonvolatile memory of the camera. They are one Power Up profile, 8 user profiles, and 8 factory profiles in the camera. User profiles can be changed by the serial interface, built into the camera's Base Camera Link connector. The factory profiles are read-only. To the power up profile, any of the user profiles can be stored.

The configuration can be comfortably changed by a configuration tool, shown in fig. 3.2, supplied with the camera. Using this tool, a `*.mcf` configuration file is created and given as a parameter to the frame grabber configuration file.



FPN 110
gain 30
black level 60
clocks (MHz) pixel 32,043 sensor 6,144
frame rate: 100
videodatawidth 2 x 8
1.st col. 0
1.st row 250
no. of col. 1280
no. of rows 450
field of view
Shutter
shutter-mode sync with shutter
exposure time [Sec] 1/ 113
☐ trigger on CC1
pixelbinning ☐ horizontal ☐ vertical
digital gain 4
☐ gray scale
☐ extern clock
☐ invert trigger
☐ digital shading correction
☐ frame count
Tx: :v :w
Rx: nH`<
serial number 30164

Figure 3.2: Mikrotron MC1310 configuration tool.

3.1.2 Frame Grabber

The camera is connected to an INSPECTA-5 frame grabber. It is a 64-bit PCI-X card with 256 MB on-board memory for fast image sequences and with the availability of connecting line as well as area scan cameras via Base or Full Camera Link interface. Further technical details can be found in the product's data sheet.

The setting up of a frame grabber is done with a `*.cam` configuration file. This configuration file is editable in a simple text editor and its structure is similar to any windows `*.ini` file. For each frame grabber configuration, there is a separate section. In the following paragraph, there are the most important parameters for each configuration.

```
...
[FGCONFIG]
CamString=.\camera.mcf
LineLen=1280
NumLin=450
...
PixelRouter=0x14
...
```

The section starts with a section name, here it is `[FGCONFIG]`. Follows the camera string `CamString`, that specifies the camera parameter file, here `camera.mcf`. This file can be exported by the Camera configuration tool, see fig. 3.2. Then the vertical and horizontal resolutions of the camera are set and they must correspond with the settings in `camera.mcf` configuration file. The `PixelRouter` parameter settings depends on the tap size and used interface.

The frame grabber configuration file is important for the final application. It is used in HALCON as a parameter file, anytime the programmer needs to use the image acquisition hardware.

3.1.3 Lens

When selecting lens, we are restricted by several factors. The most important is the size of the lens. Our camera, similarly as other cameras for machine vision systems, supports industrial standard C-Mount. However, most C-mount lenses are constructed for sensor sizes of 1/3″ to 1/2″, which is for sensor with a diagonal of 1.25″ insufficient. Another restriction is caused by the fast sampling frequency. In my project, I wanted to reach a sampling frequency of 100 Hz. The short exposure time, requires a big aperture size.

The desired focus of the lens can be computed from picture 3.3. The d is the distance from the measured object to the optical axes of the lens and the f focal distance of the lens. The model to be observed is ca. 2.6 m high and 1.0 m wide. If we place the camera chip so that the longer side would correspond with the longer side of the object, we can easily compute the cameras focal length 3.1.

$$f = \frac{7.68}{1300} \cdot d \tag{3.1}$$

If we know, that the camera will be approximately 11 m far from the model, we get $f = 65mm$.

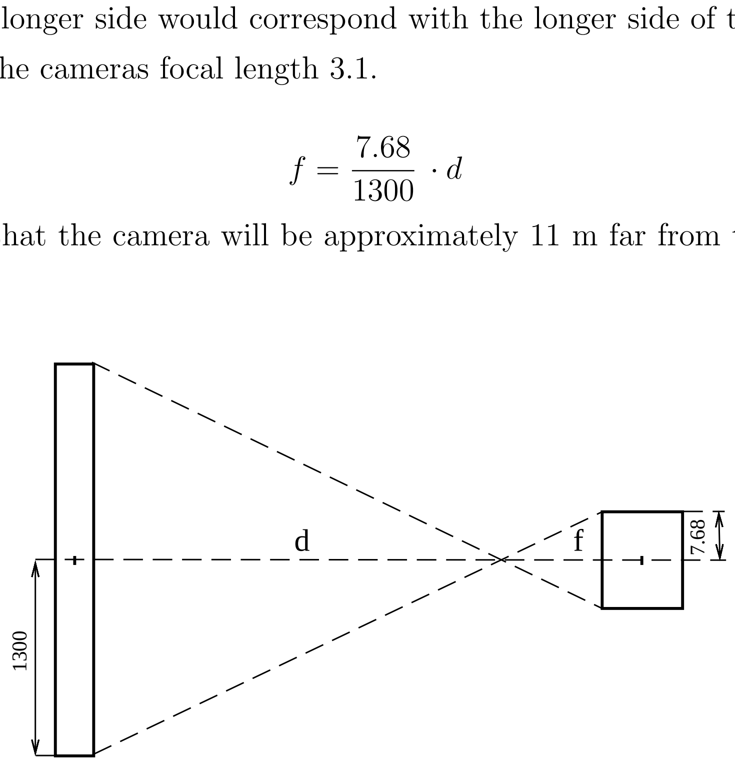


Figure 3.3: Calculation of lens parameters.

In my project I used the closest available C-Mount lens, with the highest available optical diameter. As the picture depth is not that important, the aperture was selected low, so that the light sensitivity was increased. Further details see table 3.1.

Table 3.1: Lens optical parameters

Parameter	Value
Optical diameter	$1''$
Apperture	$F > 1.8$
Focus	$f = 75mm$

3.2 Image processing software

For solving the task of image processing, HALCON software was applied. HALCON is a comprehensive machine vision library, in which basic, as well as advanced machine vision algorithms are implemented and thus the machine vision application development is made faster and more comfortable.

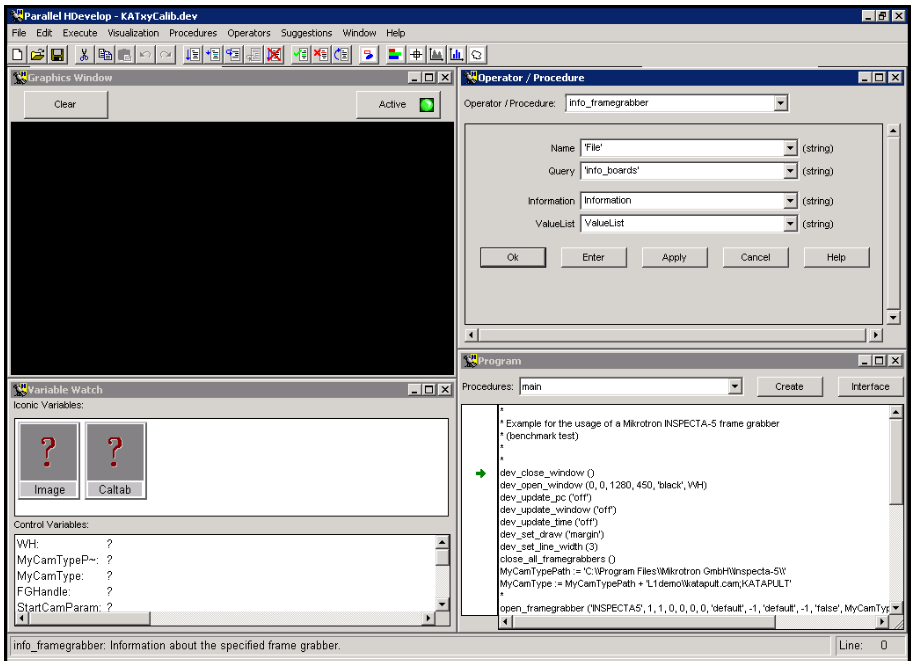
HALCON, however, is not just a plain machine vision library, but it contains many tools helping the programmer to prototype and debug the machine vision application quickly. The basic tool for prototyping machine vision tasks is an integrated development environment called HDevelop.

For the end-user application, HALCON provides user programming interfaces for C and C++ languages, as well as COM interface, which is independent of the programming language. The HALCON library has even the possibility of using automatized parallelism on several levels[2], which is useful if multiprocessor or dual-core machines would be used in the future.

3.2.1 HDevelop

HDevelop can be characterized as a rapid prototyping tool for machine vision tasks. The screen shot of the HDevelop environment can be seen in fig. 3.4. The development environment provides the user with all the necessary tools for quick machine vision algorithm development.

[2]this depends on the selected operator


Parallel HDevelop - KATxyCalib.dev
File Edit Execute Visualization Procedures Operators Suggestions Window Help
Graphics Window
Clear
Active
Operator / Procedure
Ok Enter Apply Cancel Help
Program
Create
Interface
Variable Watch

Operator / Procedure: info_framegrabber

Name 'File' (string)
Query 'info_boards' (string)
Information Information (string)
ValueList ValueList (string)

Procedures: main

```
*
* Example for the usage of a Mikrotron INSPECTA-5 frame grabber
* (benchmark test)
*
*
dev_close_window ()
dev_open_window (0, 0, 1280, 450, 'black', WH)
dev_update_pc ('off')
dev_update_window ('off')
dev_update_time ('off')
dev_set_draw ('margin')
dev_set_line_width (3)
close_all_framegrabbers ()
MyCamTypePath := 'C:\\Program Files\\Mikrotron GmbH\\Inspecta-5\\'
MyCamType := MyCamTypePath + 'L1demo\\katapult.cam;KATAPULT'
*
open_framegrabber ('INSPECTA5', 1, 1, 0, 0, 0, 0, 'default', -1, 'default', -1, 'false', MyCamTyp
```

Iconic Variables:

Image Caltab

Control Variables:

```
WH:            ?
MyCamTypeP~:   ?
MyCamType:     ?
FGHandle:      ?
StartCamParam: ?
```

info_framegrabber: Information about the specified frame grabber.

Line: 0

Figure 3.4: HDevelop - rapid prototyping machine vision development environment.

HDevelop environment consists of four windows. The operators and functions are written in the upper right Operator/Procedure window. This window guides the programmer through all the parameters that have to be given to the currently used function, and with a single mouse click can show the appropriate documentation. The algorithm being developed is typed into the lower right Program window. The lower left Variable Watch window shows the user the output of the currently typed algorithm. This window is divided into two parts. In the upper part, iconic variables are shown. Iconic variables are variables characterized as matrices like images, regions of images, image areas etc. The lower part shows the conventional scalar or array variables. The upper left window provides the actual view of currently processed image.

3.2.2 Code export

HDevelop can be used to develop a standalone applications running with a run-time license directly in HDevelop environment. However, more often the vendor wants to build in the machine vision algorithm into a more complex application. HDevelop provides several mechanisms for transfer of the code into other environments. The easiest way is to export the code into C, C++ or COM. After the export, the application can be immediately used without any programming.

However, in the most cases, HDevelop is used just for tuning of the machine vision algorithm, so that all the parameters were set properly and the final algorithm is then reprogrammed manually. The main advantage of this way is, that the final code written by the programmer is much more transparent than the exported code. Other disadvantage of code export is, that in C++ export the C-like procedural operators are used, instead of C++ objects.

3.3 Camera calibration

To understand better the process of camera calibration, it is useful to know something more about the point projection from the world coordinate system to the image coordinate system. I will try to sum up briefly this transformation, as it was in detail described for instance in (MVTec, App. Guide 2005). This transformation consists of several operations. First, the transformation from the world coordinate system to the camera coordinate system, then a projection of transformed picture into the image plane, a transformation modeling the lens distortion, and finally the transformation into image coordinates. In the following, let us have a closer look on individual steps. I will consider just the pinhole camera model. For telecentric camera model equations, see the original tutorial in (MVTec, App. Guide 2005).

The rigid transformation from the world coordinate system (WCS) to the camera coordinate system (CCS) can be mathematically described as a multiplication by a homogeneous transformation matrix $\mathbf{H^c_w}$. The transformation from the WCS into CCS is then described by equation 3.2.

$$\mathbf{p^c} = \mathbf{H^c_w}\mathbf{p^w} \tag{3.2}$$

Now follows the projection of the point, given in CCS into the image plane, that is from the three dimensional space to the plane. This step is for the pinhole camera model described by the equation 3.3.

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{\mathbf{f}}{\mathbf{z^c}} \begin{pmatrix} \mathbf{x^c} \\ \mathbf{y^c} \end{pmatrix} \tag{3.3}$$

After the image was projected into the image plane, the lens distortion modifies the $(u, v)^T$ coordinates. The lens distortion can be modeled by the equation 3.4

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \frac{2}{1 + \sqrt{1 - 4\kappa(u^2 + v^2)}} \begin{pmatrix} u \\ v \end{pmatrix}, \tag{3.4}$$

where κ parameter models the magnitude of the distortion. Its effect is depicted in figure 3.5.

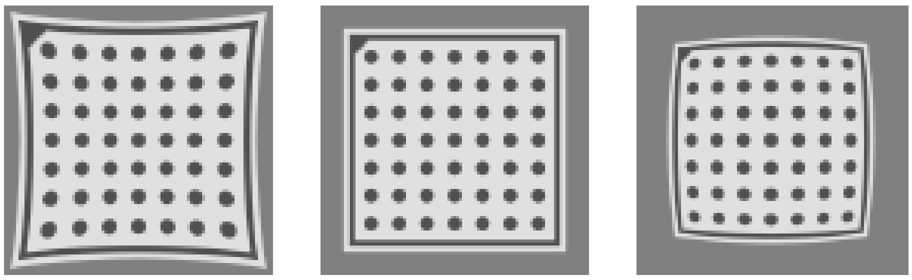


Figure 3.5: Demonstration of radial distortion for $\kappa > 0$ (left), $\kappa = 0$ (middle), $\kappa < 0$ (right). Taken from (MVTec, App. Guide 2005).

Advantage of modeling the radial distortion by equation 3.4 is, that the vector $(u, v)^T$, can be analytically expressed as

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{1}{1 + \kappa(\tilde{u}^2 + \tilde{v}^2)} \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix}. \tag{3.5}$$

Final step of the whole transformation is the transformation of the $(\tilde{u}, \tilde{v})^T$ vector in the image plane coordinate system into the image coordinate system. This can be expressed by equation 3.6.

$$\begin{pmatrix} r \\ c \end{pmatrix} = \begin{pmatrix} \frac{\tilde{v}}{S_y} + C_y \\ \frac{\tilde{u}}{S_x} + C_x \end{pmatrix} \tag{3.6}$$

In this transformation the S_x and S_y are the scaling factors, which by the pinhole camera model represent the distances between single sensor elements on the camera chip. The point $(C_x, C_y)^T$ is the so called principle point of the image, which by a pinhole camera is the perpendicular projection of the optical center into the image plane.

Looking at the whole transformation process from the WCS to the image coordinates, we need to know the six parameters, shifts and rotations along the coordinate axes, stored in the matrix $\mathbf{H}_{\mathbf{w}}^{\mathbf{c}}$, they are called the exterior or also extrinsic camera parameters and the six parameters $(f, \kappa, S_x, S_y, C_x, C_y)$, which we call interior or intrinsic camera parameters.

The process of finding intrinsic and extrinsic camera parameters is called the camera calibration.

Chapter 4

Data filtration and system modeling

The data coming from the camera are strongly distorted by inaccuracy of the measurement. What we need is to measure somehow the ball's position together with its speed. This would be very ill conditioned problem if we looked at the system just like onto a black box, not knowing anything about its physical structure. But we know quite a lot about the ball's trajectory. Not taking into account the air drag, we can describe the ball's trajectory as a linear differential equation.

The solution of this problem is straightforward, by applying the so called Kalman filter, which was introduced by (KALMAN, R. E. 1960). In the following sections a short description of Kalman filter is shown. Then a Kalman filter for the billiard ball's trajectory is prototyped in MATLAB and a catch point prediction based on the system's states is calculated.

4.1 The discrete Kalman filter

Very good introduction to discrete Kalman filtering was shown in (WELCH, G., BISHOP, G. 2004) or (HAVLENA, V., ŠTECHA, J. 2000), where even some further background to the system theory can be found. For the completeness, I will try to summarize some of the crucial ideas of Kalman filtration.

4.1.1 Filter equations

The Kalman filter solves the general problem of estimating the state $x \in \Re^n$ of an n-dimensional discrete system, described by the difference equation

$$\begin{aligned} x(t+1) &= \boldsymbol{A}x(t) + \boldsymbol{B}u(t) + v(t) \\ y(t) &= \boldsymbol{C}x(t) + \boldsymbol{D}u(t) + e(t), \end{aligned} \tag{4.1}$$

where the $v(t)$ and $e(t)$ are the non-correlated discrete white noises in the system and in the measurement.

In the stochastic problem formulation, the Kalman filter is the optimal state observer in the LMS[1] means.

The algorithm of Kalman filter consists of two basic steps. The prediction and the correction step.

In the prediction step, the Kalman filter tries to predict the evolution of the system in the upcoming time sample, according to the data it has measured so far, while in the correction step, the prediction is compared to the real measurement, and coefficients of Kalman filter are adjusted according to the measured error.

4.1.2 The prediction step

The prediction step is described by equations

$$\begin{aligned} \hat{x}(t+1|t) &= \boldsymbol{A}\hat{x}(t|t) + \boldsymbol{B}u(t) \\ \boldsymbol{P}(t+1|t) &= \boldsymbol{A}\boldsymbol{P}(t|t)\boldsymbol{A}^T + \boldsymbol{Q}. \end{aligned} \tag{4.2}$$

In equations 4.2, the $\boldsymbol{P}$ matrix is the error covariance matrix and its trace is minimized by the Kalman filter algorithm.

What the equations 4.2 do, is very simple. First, the so called a priori value for $\hat{x}(t+1|t)$ is computed from the knowledge of previous evolution of the state and from the system description, given by the system matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. The same is it for the error covariance matrix. Its a priori value is computed from the previous system error evolution and the preset value of the distortion, that is to be expected in the system.

The a priori values of state and error covariance computed in the prediction step are the values, that are most likely to be the real measured values, according to the information we have on the system and its distortion.

[1]Least mean squares

4.1.3 The correction step

The correction step is expressed by the equations

$$\begin{aligned} \boldsymbol{K}(t) &= \boldsymbol{P}(t|t-1)\boldsymbol{C}^T(\boldsymbol{C}\boldsymbol{P}(t|t-1)\boldsymbol{C}^T + \boldsymbol{R})^{-1} \\ \hat{x}(t|t) &= \hat{x}(t|t-1) + \boldsymbol{K}(t)(y - \boldsymbol{C}\hat{x}(t|t-1) - \boldsymbol{D}u(t)) \\ \boldsymbol{P}(t|t) &= \boldsymbol{P}(t|t-1) - \boldsymbol{L}(t)(\boldsymbol{C}\boldsymbol{P}(t|t-1)\boldsymbol{C}^T + \boldsymbol{R})\boldsymbol{L}^T(t). \end{aligned} \tag{4.3}$$

The a priori values of state and error covariance computed in the prediction step are used in the correction step. The computed a posteriori values of the state and error covariance are corrected with the newly measured point. The expected measurement error is reflected in this step as $\boldsymbol{R}$ matrix. It expresses, how much we trust the measurement.

4.1.4 The computation cycle

The cycle of Kalman filter is depicted in fig. 4.1. As initialization data to the computation, come the a priori values for the state $\boldsymbol{x}$ and error covariance matrix $\boldsymbol{P}$.

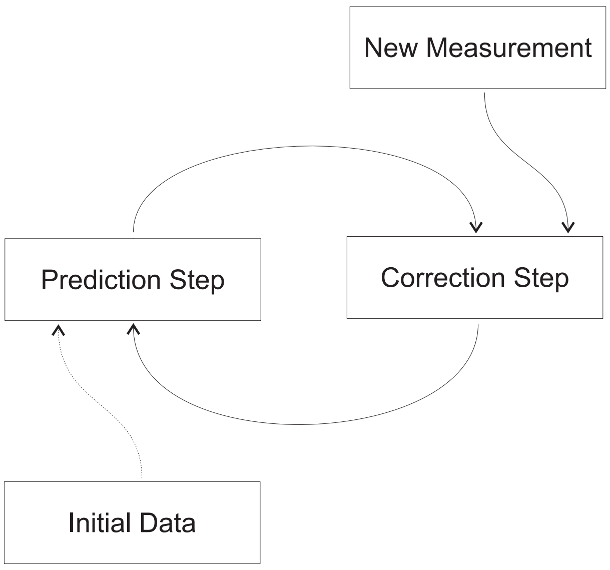


Figure 4.1: Kalman Filter cycle.

The most important parameters of Kalman filter to be tuned are the system and measurement errors $\boldsymbol{Q}$ and $\boldsymbol{R}$. The relation between the both values expresses, which values do we trust more. If to the information contained in the system equations, or to the information gained from the measurement.

There exist modifications of Kalman filter for non-linear systems, for systems with color-noise etc.

4.2 System modeling

When modeling a thrown billiard ball, I did not take into account an air drag. There are several reasons for that. First, I am modeling a flight trajectory of a billiard ball, which is heavy enough, compared to e.g. a tennis ball, used by (Frese, U. et al. 2000), to be effected by air drag. Then, the modeling of air drag would lead a strong nonlinearity into the system in the form of an air drag factor, and I would have to use computationally more complex extended Kalman filter. Moreover, the estimation of another parameter would lead to further uncertainty in the system. The used simplification has shown to be sufficient for this case.

Modeling a falling body, the single axes can be considered as independent. By the billiard ball, the most complex movement is done in the vertical direction, as it is effected by g. I will describe just one axes in the vertical direction, because the other axes are the same, except to $g = 0$, thus having constant velocity. I modeled the states of the system as follows

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -g, \end{aligned} \tag{4.4}$$

where the state x_1 is the current position of the ball and x_2 the actual velocity. If I rewrite those states into the state equations, I get

$$\begin{array}{ccccccc} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} & = & \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} & + & \begin{bmatrix} 0 \\ 1 \end{bmatrix} & u \\ \dot{\boldsymbol{x}} & = & \boldsymbol{A} & \boldsymbol{x} & + & \boldsymbol{B} & u \end{array} \tag{4.5}$$

4.2.1 Filter parameters

The most difficult was to set up properly the parameters $\boldsymbol{Q}$ and $\boldsymbol{R}$. Empirically it has shown up, that matrices could be set as

$$\begin{aligned} \boldsymbol{Q} &= \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix} \\ \boldsymbol{R} &= [100] \,. \end{aligned} \tag{4.6}$$

The meaning of the matrix coefficients, can be understood as that we are most sure about the position information in the system. The modeled speed of the system is more noisy, but most fuzzy is the actual measurement.

Moreover, it has shown up, that it is useful to set the beginning system covariance to very high value $P_0 = 10^6 \cdot \boldsymbol{Q}$. The meaning of this step is, that I tell the system like this, that it should set up the parameters of the observed system as fast as possible with the information from the measurement. The more measurement it then gets, the more important becomes the information from the system. This is of great importance, because if the system would receive some noisy data a few samples before the catch point, whole measurement would be negatively influenced by it. This, however, corresponds the reality. At the beginning I do not know, where the ball is, and what is its speed, so I need to trust the measurement more, than at the end, where it is unlikely that the ball would change its trajectory suddenly.

The quality of the set parameters is demonstrated by the plotted error covariance in fig. 4.2. The diagonal elements of $\boldsymbol{P}$ are shown in x and y axes. They represent the decrease of error in estimating the balls position and its velocity in single axes. The plot is made of data, that were logged by the real application.

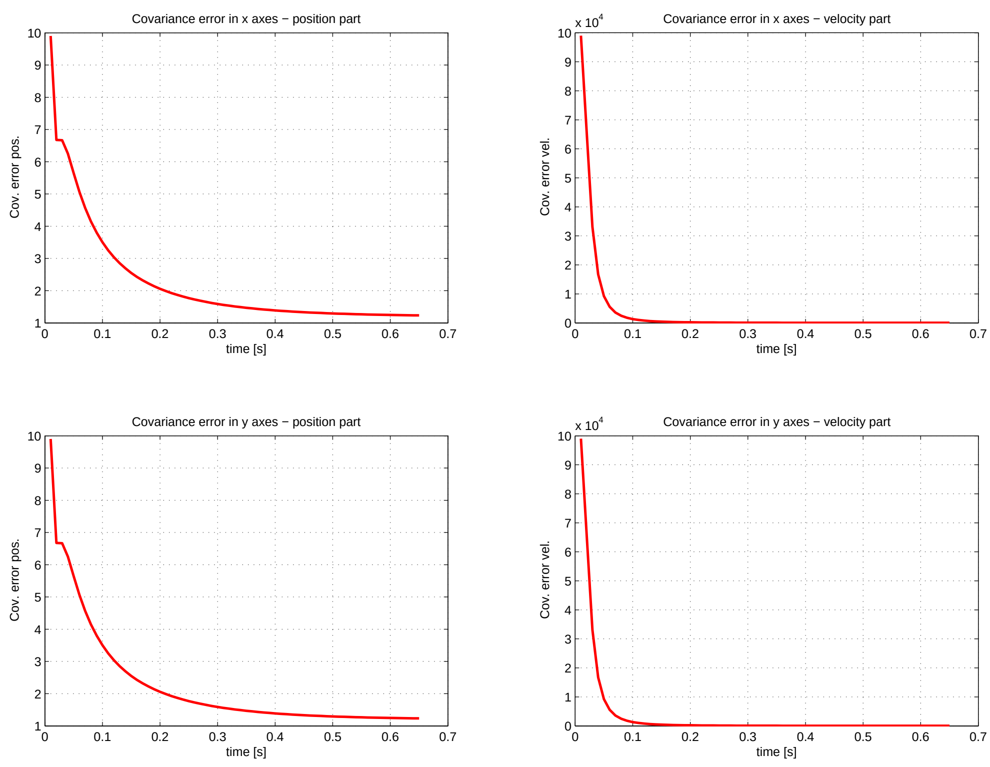


Figure 4.2: Error covariance in x and y axes.

4.2.2 Catch point prediction

The prediction of the catch point is done on data observed by the Kalman Filter. The observed variables are the actual positions in both axes x_0 and y_0 and the actual speeds v_{x0} and v_{y0}.

The prediction is done from the linear throw in homogeneous gravitation field. In general, the trajectory of a throw under an angle α can be described by equations

$$\begin{aligned} x &= x_0 + v_{x0}t \\ y &= y_0 + v_{y0}t - \frac{1}{2}gt^2, \end{aligned} \tag{4.7}$$

From these equations, after separating time t, the catch point either by given x-

coordinate 4.8 or given y-coordinate 4.9 can be computed.

$$\begin{aligned} t &= \tfrac{x_{fixed}-x_0}{v_{x0}} \\ y &= y_0 + v_{y0}t - \tfrac{1}{2}gt^2, \end{aligned} \tag{4.8}$$

$$\begin{aligned} t &= \tfrac{v_{y0}\pm\sqrt{v_{y0}^2-2gy_{fixed}+2gy_0}}{g} \\ x &= x_0 + v_{x0}t. \end{aligned} \tag{4.9}$$

The positive value of the two solutions for t in 4.9 must be taken into account.

Chapter 5

B&R Automation Net

B&R Automation Studio is a software for programming various B&R Automation Targets. To those targets belong B&R PLCs, Power Panels, ACOPOS servo drives, distributed inputs and outputs etc. There runs a special software on an Automation Target, called B&R Automation Runtime. B&R Automation Studio communicates with the B&R Automation Runtime software, running on various B&R Automation Targets via B&R Automation Net. This chapter describes the communication with B&R Automation Net via Process Visualization Interface. The details to Automation Studio and Automation Runtime can be found in (Pšenička, J. 2005).

5.1 Process Visualization Interface (PVI)

Within PVI Base System it is possible to access the B&R Automation Net from any Windows 95/98/ME/NT/2000/XP or Windows CE computer. PVI is used to design simple human-machine interface to controlled processes or to generally access the connected PLC from within a control application. With PVI it is possible not only to access variables stored in the PLC, but also to call some of its services, like downloading and uploading of program modules, querying of its status, calling warm or cold reset, etc. In this section I will briefly describe the structure of the interface and method of accessing PVI modules on various levels.

5.1.1 Structure of PVI

The structure of PVI interface is schematically described in figure 5.1. The lowest level is formed by PVI Line objects. They determine the communication protocol to be used and are connecting objects within PVI with objects outside of PVI (e.g. on a PLC).

The program interface to the application is based on Microsoft's Component Object Model (COM) technology. This is the lowest access level to PVI available to the programmer. PVI COM interface is even used by higher level applications like OPC, DDE, or Web Servers to provide data for applications using those servers.

A gateway between PVI COM and PVI Lines is formed by PVI Manager. All the communication between the PLC and the PC is passed through the PVI Manager.

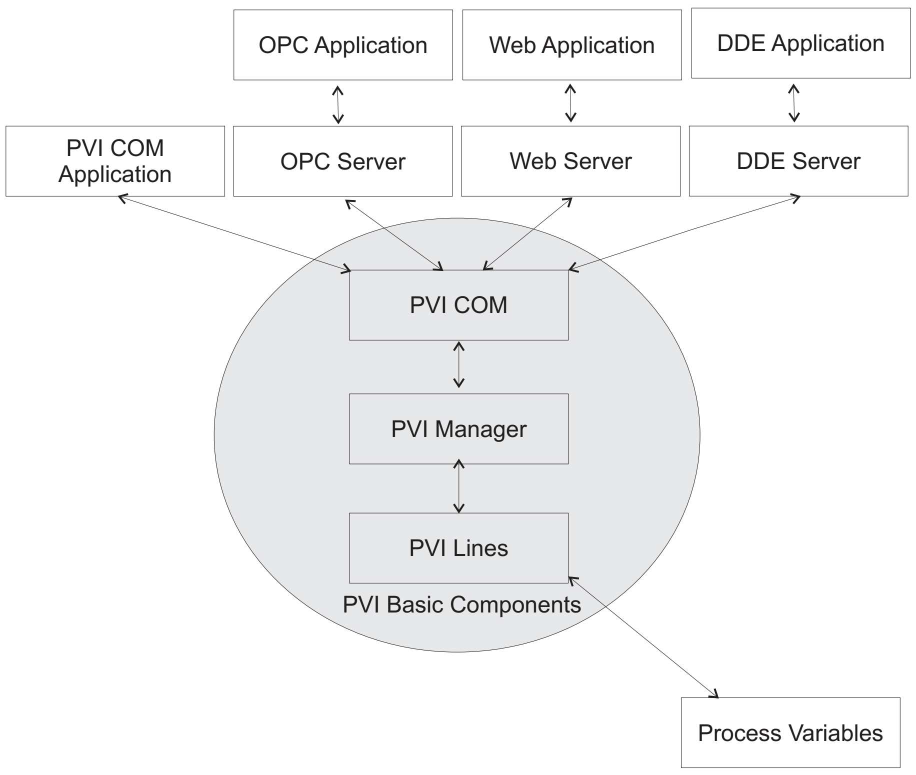


Figure 5.1: Structure of Process Visualization Interface.

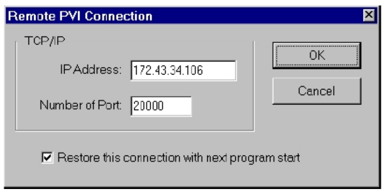
5.1.2 PVI Manager

PVI Manager is an application running in the background as a process or service. It manages the process data into program or data objects to make them accessible to PVI.

PVI Manager services the timing of the communication and it routs the communication to specified stations.

PVI Manager has no graphical user interface and can be monitored by PVI Monitor. Besides overview of operating status of the PVI Components, PVI Manager options can be changed with PVI Monitor. The most important are the Remote PVI Connection options, see the figure 5.2, and the PVI Manager Properties, fig. 5.3.

When configuring the Remote PVI Connection options, the IP address and port number of the remote PLC is given as a parameter. This IP address must be properly set in the Automation Studio as e.g. shown in (PŠENIČKA, J. 2005).



Remote PVI Connection

TCP/IP

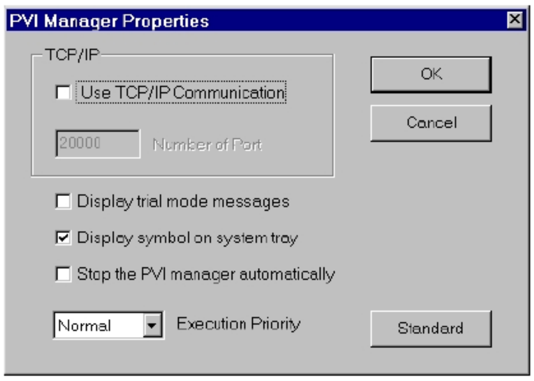
IP Address: 172.43.34.106

Number of Port: 20000

☑ Restore this connection with next program start

Figure 5.2: PVI Monitor - Remote PVI Connection dialog box.

In the PVI Manager Properties dialog box, the execution priority of the PVI Manager process can be changed. However, TCP/IP communication option is more important. This enables connection of application running on remote PCs to the PVI Manager.



PVI Manager Properties

TCP/IP

☐ Use TCP/IP Communication

20000 Number of Port

☐ Display trial mode messages

☑ Display symbol on system tray

☐ Stop the PVI manager automatically

Normal Execution Priority

Figure 5.3: PVI Monitor - PVI Manager Properties.

5.1.3 Establishing connection

First step to set up a successful communication with PVI Manager is to establish a PVI connection. This is done with `PviInitialize` operator. Its call creates a new connection to PVI Manager running on a computer connected to the control PLC, with that we are about to communicate. It can be connected either to a local machine (local connection) or to a different PC (remote connection) identified with accessible IP address. Its IP address and port number are given to the `PviInitialize` operator as parameters.

In case PVI Manager and PVI COM application run on the same computer and the local connection is selected for connecting to it, the PVI manager is started automatically with `PviInitialize` function call. On the other hand, if both applications run on separate computers, then to establish connection, the PVI manager has to be started manually on the remote machine, connected to the Automation Net.

Connection status can be controlled by global PVI events. This is useful by remote connection especially. The events are created with `PviSetGlobEventMsg` operator. I am using three global PVI events, which monitor establishing of new connection, loss of connection, and demand on recreating of all of the PVI object.

5.1.4 Creating PVI Objects

After the connection was successfully created, the object structures for process images can be set up. Using process image, the user can access data in PLC variables or execute special services.

The process image is represented in the PVI manager as an object structure, in which the individual objects are represented in a tree - Process Object Tree. Each process object is represented by a name, unique on the particular tree level, and can be accessed by specifying the object path.

Each PVI Line has slightly different structure of Process Object Tree. For the PVI Line Ina2000, the Process Object Tree structure is depicted in figure 5.4.

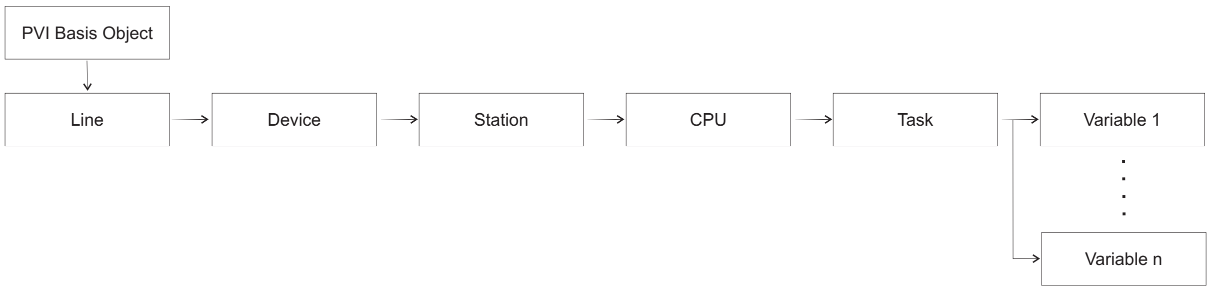


Figure 5.4: Process image structure.

There are several types of process objects that must be created to get access to particular PLC variable. The root process object is a base object, which is always present in the Process Object Tree and does not need to be created. Line object determines the PVI Line, which specifies the line used for the connection. Device, station, and CPU objects are used to connect to certain PLC by specifying its IP address, Ethernet Powerlink address, its name etc. Module, task, and variable objects determine the variable, from certain program module in a specified task.

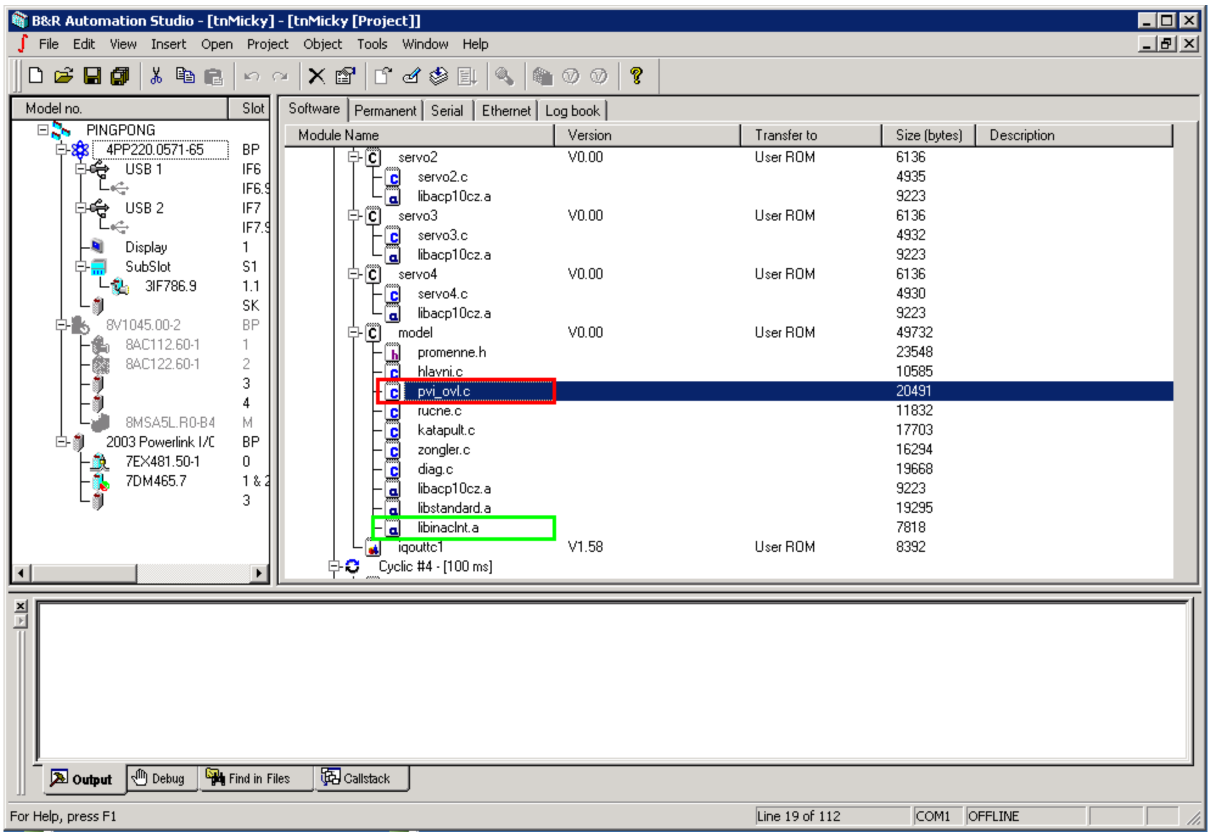
The process objects are created using `PviCreate` operator. As shown in fig. 5.4, line, device, station, CPU, and task objects must be set up, before any variable object is created.

The communication with the PVI interface is done with windows messages. If the value of a certain variable is to be read, `PviReadRequest` function is called. Optionally, this function call can be omitted, if cyclic reading is set. As soon as the variable value is available, it is signaled by sending a windows message `WM_PVI_EVENT`. It is important that those messages are caught by appropriate function in the C++ program, as ignoring them would lead to filling up of the windows message queue and freezing of the control application. Reaction to each message requesting the read operation is done with `ReadResponse` function.

Writing of a new value works on similar principle as reading. When writing a new value into a variable, `PviWriteRequest` operator is called. The result of writing operation is signaled again with a windows message `WM_PVI_WRITE_RESP`, which has to be responded to with `PviWriteResponse`.

5.1.5 Preparing B&R Automation Target

So as the connection was successful, Automation Target has to be configured as Ina2000 Client. This is done in Automation Studio just by inserting `libinaclnt.a` library to the task, that is going to communicate with the outer world, see fig. 5.5. Then, by including the `inaclnt.h` header file in the PLC program, all the variables of the task can be accessed by PVI.


B&R Automation Studio - [tnMicky] - [tnMicky [Project]]
File Edit View Insert Open Project Object Tools Window Help
Software Permanent Serial Ethernet Log book
Output Debug Find in Files Callstack
For Help, press F1
Line 19 of 112
COM1
OFFLINE

Model no.	Slot
PINGPONG	
4PP220.0571-65	BP
USB 1	IF6
	IF6.9
USB 2	IF7
	IF7.9
Display	1
SubSlot	S1
3IF786.9	1.1
	SK
8V1045.00-2	BP
8AC112.60-1	1
8AC122.60-1	2
	3
	4
8MSA5L.R0-B4	M
2003 Powerlink I/C	BP
7EX481.50-1	0
7DM465.7	1 & 2
	3

Module Name	Version	Transfer to	Size (bytes)	Description
servo2	V0.00	User ROM	6136	
servo2.c			4935	
libacp10cz.a			9223	
servo3	V0.00	User ROM	6136	
servo3.c			4932	
libacp10cz.a			9223	
servo4	V0.00	User ROM	6136	
servo4.c			4930	
libacp10cz.a			9223	
model	V0.00	User ROM	49732	
promenne.h			23548	
hlavni.c			10585	
pvi_ovl.c			20491	
rucne.c			11832	
katapult.c			17703	
zongler.c			16294	
diag.c			19668	
libacp10cz.a			9223	
libstandard.a			19295	
libinaclnt.a			7818	
iqouttc1	V1.58	User ROM	8392	
Cyclic #4 - [100 ms]				

Figure 5.5: Green - the inserted library for selected Line, red - the program communicating with PVI.

Chapter 6

Algorithm design

The whole flight of the thrown billiard ball lasts about 700 ms. In that time, its speed in two axes has to be measured precisely enough, to determine the desired catch point as early as possible, so that it could be sent to the drives.

Before the whole program was implemented in Visual Studio, all the used algorithms were tested in HDevelop and Matlab, to find a suitable program structure and solve the arisen problems fast in prototyped program.

6.1 Acquisition speed

So that the image acquisition was as fast as possible, only the necessary image data must be transferred. At the same time, the maximum possible resolution ought to be kept, so that the precision of the whole system was not negatively influenced.

In this phase of the project, Basic Camera Link version was used. This has a restriction of data transfer up to 132 MBps, but does not need PCI-X bus to be used.

6.1.1 Image size

To satisfy the data rate available by Base Camera Link interface, the image cannot be transferred to the computer memory in the full resolution. This is, however, not necessary. The active area, where the ball can occur is circumscribed by a rectangle 2.2 m high and 0.9 m wide. By preserving the maximum available precision, it is fully sufficient to transfer just the selected region of 1280 x 450 px.

If the transferred data for each pixel is set to 8 bits by frequency of 100 Hz, the total amount of data transferred per second will be 57.6 MB. With this value we are in the half of the PCI bus limit. Higher sampling frequency would bring me increased system accuracy, but the currently installed lightening conditions do not allow using lower exposure times.

The desired region of interest (ROI) can be set by the camera configuration tool. I shifted the ROI to the center of the chip, since the center is the most illuminated part of it.

Because the whole scene is even by 100 Hz too dark, the whole image is digitally amplified. The effect of the digital amplification is mainly for tuning of the image processing algorithm. The greater contrast improvement can be achieved only by better illumination.

6.1.2 Grabbing modes

The selected grabbing mode is very important for the whole algorithm. In HALCON, several grabbing modes are possible. The basic command for synchronous image acquisition is the `grab_image` operator. Its timing is schematically shown in fig. 6.1.

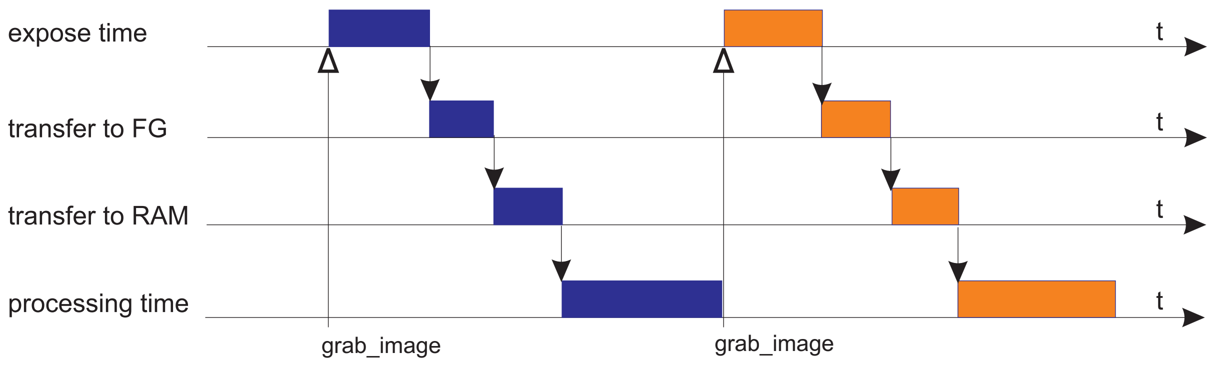


Figure 6.1: Timing of synchronous grabbing.

Its main disadvantage of synchronous grabbing is, that while the image is acquired, the image processing is not possible, because the program waits, till the operator returns the grabbed image.

With `grab_image_async`, asynchronous grabbing is available. The fig. 6.2 shows its timing. This means, that while the acquired image is transferred from the camera to the frame grabber and to computer's RAM, the processing of image previously saved in the RAM can be done.

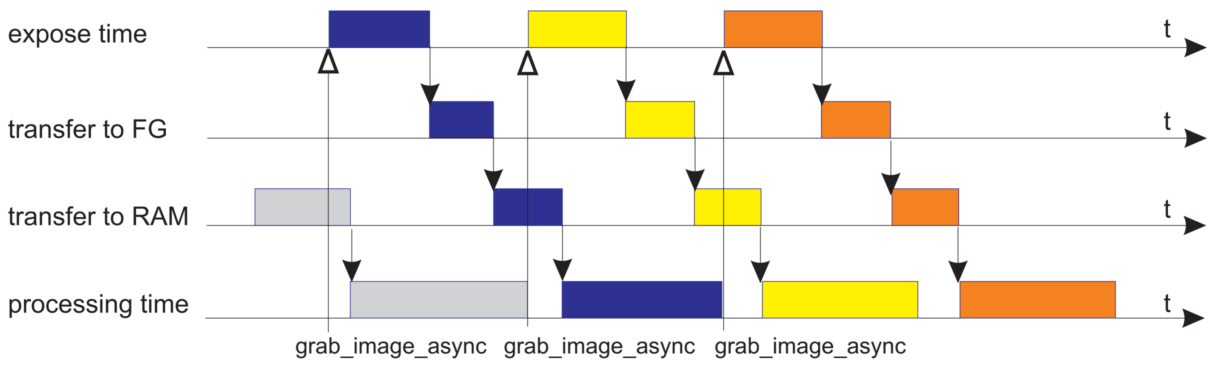


Figure 6.2: Timing of asynchronous grabbing.

The disadvantage of asynchronous grabbing is, that the program does not know, how old the processed image is. It is also important, that as much operating memory is reserved, as needed for at least two grabbed images. In case only one image could be saved in RAM, it could happen, that it would be overwritten by the newer one, while being processed.

The time distances between separate frames are due to non real time operating system neither by synchronous nor by asynchronous grabbing mode constant. The command to the camera, that the exposure should start, always comes in slightly different moment. The solution of this problem is in the so called continuous grabbing.

The continuous grabbing mode must be set with `set_framegrabber_param` command. The timing of continuous grabbing is depicted in fig. 6.8.

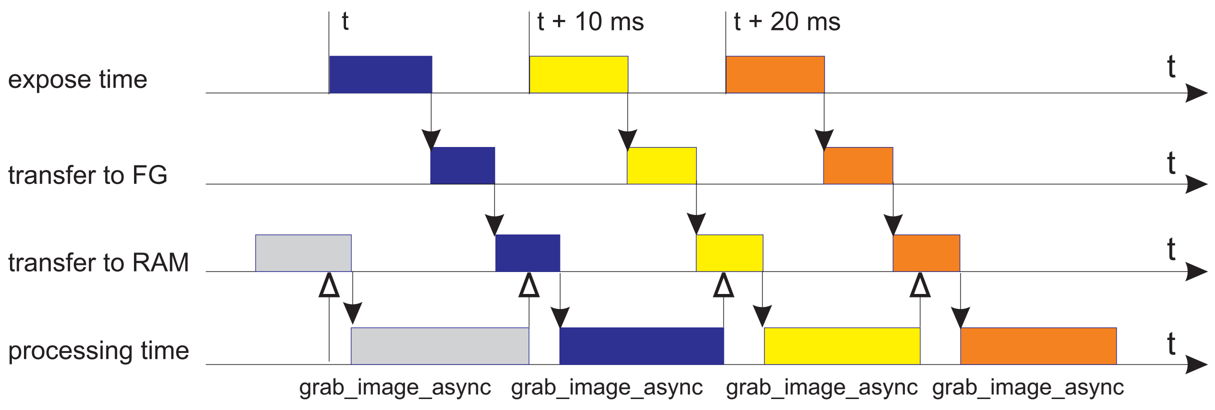


Figure 6.3: Timing of continuous grabbing.

After the continuous grabbing mode was set, the camera starts exposing new images with a predefined frequency and sends them instantly to the frame grabber, which transfers them to the computer's RAM. The older images are automatically overwritten. The images can be read from RAM with `grab_image_async` operator.

The continuous mode, however, transfers every single grabbed image to RAM using DMA transfer via PCI. This can abnormally increase the traffic on PCI bus and thus delay other PCI transfers in the computer. Therefore, the continuous grabbing is preferable only if all the images ought to be processed.

6.2 Image processing

The image processing algorithm has to be as simple as possible, to manage to extract the ball's position in the worst case within 10 ms. The algorithm was prototyped in HDevelop and several methods were tried, before some good results were achieved.

6.2.1 Fixing background

There were several ways, how the image processing could be done. First idea was to fix the background image and subtract every newly acquired image from it. This method was utilized for instance by (Frese, U. et al. 2000).

By doing this, many problems arose. Firstly, the image was disturbed by the reflexions on the metal background wall of the model. Then, the fixed background image was becoming “old” by increasing number of samples. The automatic camera light accommodation together with slight camera vibrations caused the newly acquired image after about 5 s to be almost completely different compared to the fixed one.

By (Frese, U. et al. 2000), this problem was solved by automatic accommodation of the fixed background image. This, however, would not eliminate other problems, as for instance the moving drives, that were detected as image change as well.

6.2.2 Thresholding

Because of the experienced problems, I decided to adjust the scene in such a way, that the image processing could be evaluated as just a simple thresholding of the scene. I applied a black flat painted wall paper on the model's background and all the bright parts, except the linear motor belts, of course. This brought a great improvement in the means of reflexions on the metal parts. They were completely eliminated.

Now, with a black background, white billiard ball's were preferably chosen for juggling, whereas their segmentation is straight forward, by applying a simple thresholding function.

6.2.2.1 Fast threshold

In HALCON, more thresholding functions can be utilized. I used the `fast_threshold` operator. Its advantage is in the way it looks for the pixels that should be selected. It does the thresholding in two steps. At first, all pixels along rows and columns with certain minimum distances are searched.

In the next step the neighborhood of all previously selected points is processed. The minimum search size is set so that the object, to be found, was caught in the first step in the created pixel net. The billiard ball is in diameter ca. 100 px.

I set the minimum search value to 20 px, as sometimes only a part of the ball is seen, or is illuminated from the top more etc.

6.2.2.2 Connecting regions

The thresholded image is represented as a set of selected pixels. To form a region of those pixels, they have to be connected with `connection` operator. Continuous regions are formed like this.

6.2.2.3 Object segmentation

To find a white billiard ball, we have to specify it more precisely. I made this specification on the ball's shape. With `select_shape` operator, I characterize the area range, that the ball can occupy, and its circularity. For the area, values between 300 to 1100 work fine, and the circularity is set to be between 0.5 to 1, where 1 stands for a perfect circle.

6.2.2.4 Ball coordinates

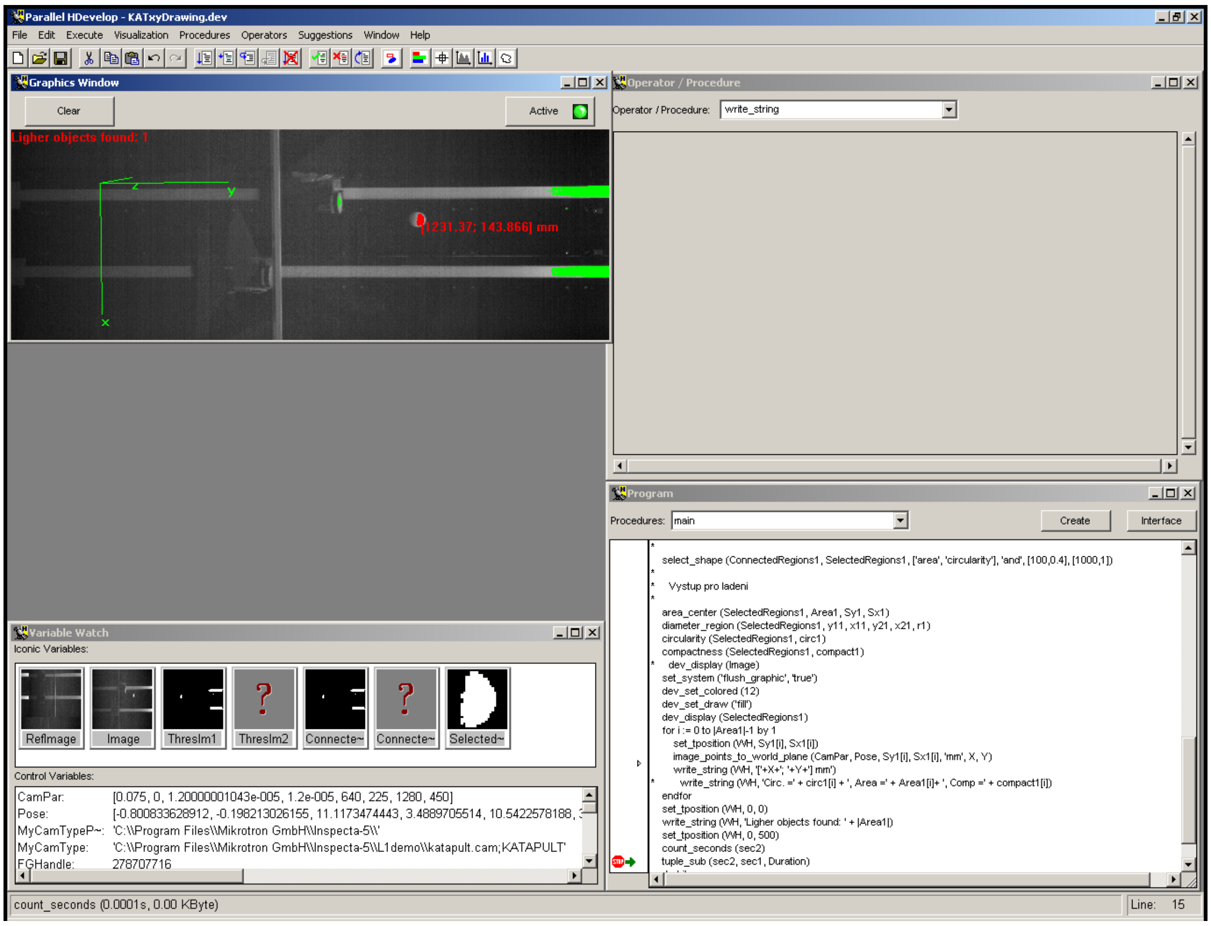
The coordinates of the found ball are gained with `area_center` operator. The returned values are coordinates in image pixel coordinate system. To get the real position of the ball in the world coordinate system, the pixel values have to be converted with the information from the camera pose, and its internal parameters. This is done by `image_points_to_world_plane` operator.

6.2.2.5 Final algorithm

Summarizing previously described steps, the final algorithm is schematically drown up by the code below. The result of the vision algorithm is seen in fig. 6.4. Because the camera is positioned, so that the longer model side was measured with the chip's side with higher resolution, the whole picture is rotated 90° clockwise. It could be rotated back with homogeneous transformation, but the rotation is not necessary and would waste the computation time.

```
...
open_framegrabber(...)
set_framegrabber_param(...,'continuous_grabbing','enable')
...
while(1)
    ...
    grab_image_async(...)
    fast_threshold(...)
    connection(...)
    select_shape(...)
    area_center(...)
```

```
        image_points_to_world_plane(...)
        ...
    end
```



```
*
  select_shape (ConnectedRegions1, SelectedRegions1, ['area', 'circularity'], 'and', [100,0.4], [1000,1])
*
*   Vystup pro ladeni
*
  area_center (SelectedRegions1, Area1, Sy1, Sx1)
  diameter_region (SelectedRegions1, y11, x11, y21, x21, r1)
  circularity (SelectedRegions1, circ1)
  compactness (SelectedRegions1, compact1)
*   dev_display (Image)
  set_system ('flush_graphic', 'true')
  dev_set_colored (12)
  dev_set_draw ('fill')
  dev_display (SelectedRegions1)
  for i := 0 to |Area1|-1 by 1
     set_tposition (WH, Sy1[i], Sx1[i])
     image_points_to_world_plane (CamPar, Pose, Sy1[i], Sx1[i], 'mm', X, Y)
     write_string (WH, '['+X+'; '+Y+'] mm')
*      write_string (WH, 'Circ. =' + circ1[i] + ', Area =' + Area1[i]+ ', Comp =' + compact1[i])
  endfor
  set_tposition (WH, 0, 0)
  write_string (WH, 'Ligher objects found: ' + |Area1|)
  set_tposition (WH, 0, 500)
  count_seconds (sec2)
  tuple_sub (sec2, sec1, Duration)
```

```
CamPar:        [0.075, 0, 1.20000001043e-005, 1.2e-005, 640, 225, 1280, 450]
Pose:          [-0.800833628912, -0.198213026155, 11.1173474443, 3.4889705514, 10.5422578188,
MyCamTypeP~:   'C:\\Program Files\\Mikrotron GmbH\\Inspecta-5\\'
MyCamType:     'C:\\Program Files\\Mikrotron GmbH\\Inspecta-5\\L1demo\\katapult.cam;KATAPULT'
FGHandle:      278707716
```

```
count_seconds (0.0001s, 0.00 KByte)
```

Figure 6.4: Tracked billiard ball.

6.3 Problems in vision system

The vision system is very sensitive to various reflexions and light condition changes during the day time. The experienced problems and their solution is described in following chapters.

6.3.1 Reflexions

The whole model is from safety reasons enclosed in a plexi glass case. This is useful for the safety, but not for image processing, since the plexi glass can be under certain circumstances a source of parasitic light reflexions.

There are several kinds of reflexions. First, if the light reflexion is big, in some day time even over one fourth of the observed area, the ball cannot be seen at all in the place of the reflexion. This reflexion cannot be eliminated by the algorithm, of course.

Second type is not that large as the first one and does not cover the area with the ball. It can be eliminated by the algorithm, but in the same time the algorithm becomes slower, because of the principle of the `fast_threshold` function, where more areas have to be searched in its second step.

Last case of light reflexions is approximately as big as the billiard ball, and is evaluated as a billiard ball as well. I designed an auxiliary simple algorithm that eliminated most of those reflexions. By tracking the ball's position, I throw away the found ball's which are too far from the previous ball's position, thus unlikely to belong to the ball's trajectory.

6.3.2 Occluded ball

The front door of the model is fixed by a steel bar. If the flying ball is hidden behind the steel bar, it cannot be found by the algorithm. I decided to solve this problem on the filtration level and evaluate the position, when the ball is not found, as 0 value. To further coordinate processing in the Kalman filter, I send zeros for all coordinates and so let the filer know, that there was a sample but was not measured.

If the Kalman filter, instead of a valid value, receives 0 for all coordinates, it uses previously predicted values as corrected ones and retriggers the prediction step, instead of correction step. This method has shown up to be very reliable if previous few samples had already been measured.

6.3.3 Ball gleam

The light illuminating the ball from the top creates gleams on the ball's surface. Usually, the gleam contrast over takes the dark areas of the ball, and they merge with the background. Thus the gleam is evaluated as the ball and the ball's center coordinates are shifted little bit upwards, depending on the ball's height. This is even noticeable in fig. 6.4, where the green area does not cover the whole ball.

This effects the final Kalman filter measurements, and can lead to control inaccuracy. The problem could be solved by the sideway lights installed into the model's walls.

I minimize its effect by offsetting the catch point position downwards along the y-axes. This helps in predicting the ball's position, but does not solve the accuracy problem in certain lightening conditions.

6.4 Filtration algorithm

The Kalman filter as in detail described in Chapter 4 was programmed. However, some specific functions had to be included in its functionality.

6.4.1 Missing data

The machine vision delegates the responsibility of managing missing data to the filtration algorithm. The solution is here easier.

```
int CKalman1D::measure(double y)
{
  predictionStep();
  if(y!=0)
  {
    correctionStep(y);
  }
  else
  {
    x1 = x_apriori1; ...
    p11 = p_apriori11; ...

    predictionStep();
  }
  ...
}
```

When measured new data, the prediction step of Kalman filter is triggered. Then, the measured data is tested, if it is valid. In case a real measurement is present, an ordinary correction step is run, in other case the a priori data of the state and error covariance are used as computed data from the correction step, and the prediction step is triggered again to do the time evolution of the whole system. The example of substituting the measurement with estimated data is to see in the plot 6.5.

This is a Matlab plot made of real measured data. Noticed shall be the 11th sample, which is missing and is correctly substituted by the predicted value. As soon as 20 samples are present, the prediction starts and is plotted with a blue line over the really measured points in the future.

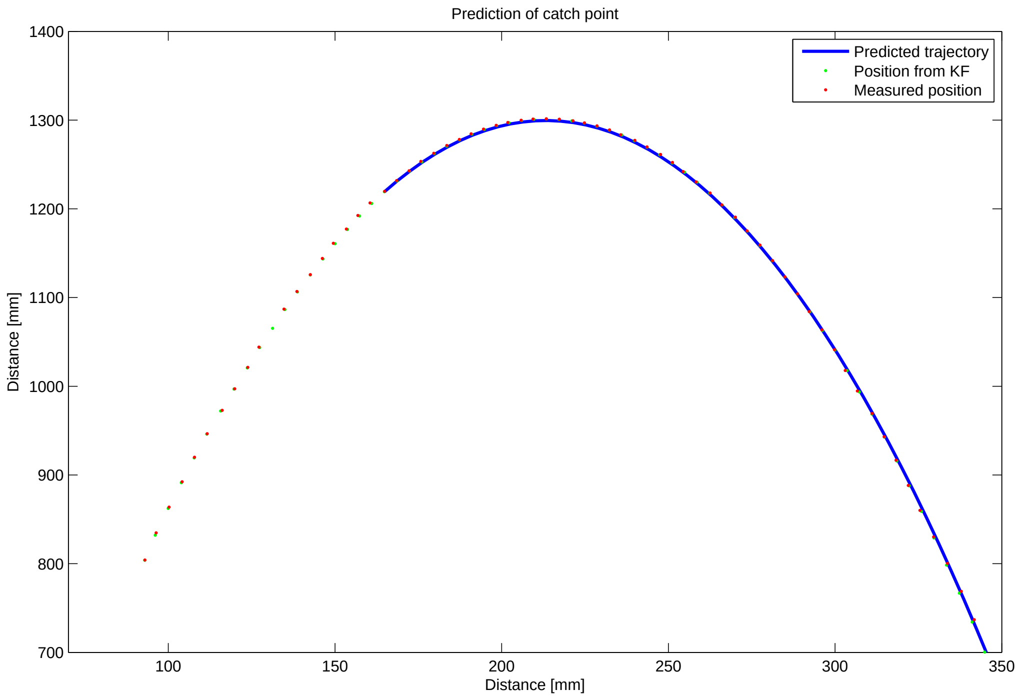


Figure 6.5: Example of predicted trajectory from the first few samples.

6.4.2 Starting filtration

With every new measurement, the Kalman Filter is initialized and restarted. The start of a new measurement is detected in the picture by finding a ball with a minimal defined speed. The minimal speed is characterized by minimal distance, the ball must cover between two following pictures. By initialization, all the intern variables are set to the start values.

6.5 PLC control program

The task of the PLC is to control the drives so that the ball was thrown in a plane, reachable by the catching grips. Further, the task for visual feedback must have a connection to the measuring computer. As the communication is based on PVI, the PLC task must include the functionality of Ina2000 client. The whole sequence of throwing the ball, catching it, and damping the catch must be synchronized and controlled by the PLC.

6.5.1 Structure of the program

First idea was to control the drives with NC actions. However, I was not able to synchronize the movements with timers and NC actions, so that the ball always flew in the plane reachable by the grips. The flight trajectory was by each throw slightly different. That is why in the final projects cam profiles were used. I designed cam profiles for throw from both the left and the right grip.

The application created in (Pšenička, J. 2005) was used. I extended the existing task `model` of one function encapsulating three state machines controlling the drives' states and communicating with the tracking PC. Then necessary cam profiles were added, following the name convention of the original program, they are `vacka16`, `vacka26`, `vacka36`, and `vacka46`.

I divided the control of the model into two parts, the left and the right catcher. Each of the parts is controlled with one state machine. They are drawn up in fig. 6.6.

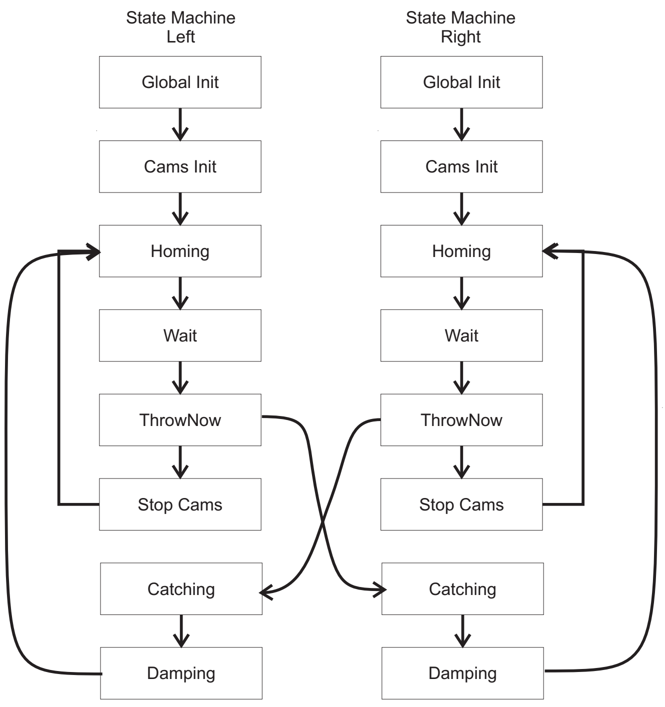


Figure 6.6: State machines controlling the drives.

6.5.1.1 Throwing

After initialization and homing of all drives into the start position, it is waited, till the throwing command comes. When thrown, the state machine on the opposite side is set to catching mode. In the catching state, the PLC is ready to read the catch point data from the PC.

6.5.1.2 Catching

The catch point data are actualized, until the command to start damping comes from the PC. It was empirically found, that the suitable time to start damping is around 150 ms before the fall. The signal is delayed because of the network latency. Slight delay is caused even by the COM objects themselves.

6.5.1.3 Damping

The catching mode is stopped and the damping is triggered. In the damping phase, the grip is moved ca. 15 cm downwards to damp the ball's fall. After the catching mode is finished, the drives return to the home position again and the process can start from the beginning.

The process of catching was recorded by trace tool at servo 4, see fig. 6.7. The part no. 1 is the adjusting to the sent catch point and instant actualization with the coming measurement, the phase 2 describes the damping of the ball's fall and the last phase 3 is a return to the home position.

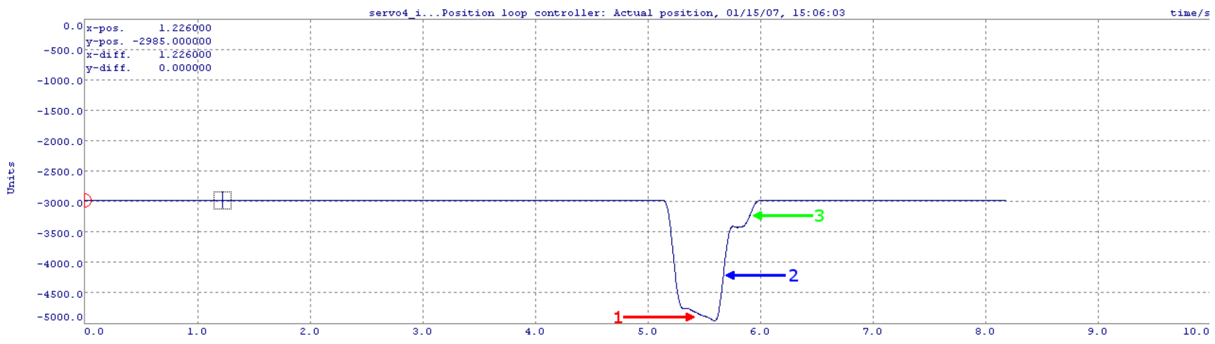


Figure 6.7: Trace record of the 4th servo drive while catching the ball.

6.5.2 Single ball juggling

The demonstration program alternates between left and right part throwing. The switching is controlled by another, third, state machine, which is independent of the two state machines described in fig. 6.6 and it just triggers the left and right throwing and guarantees that the both parts are prepared before the throw.

6.6 Timing

The timing of the whole system is drawn up in fig. 6.8. After initialized, the measurement is clocked by the camera sampling frequency. The tracking of a ball starts, as soon as ball motion is detected in the scene. This means that the triggering of the motion, is in fact derived from the PLC, which controls the drives throwing the balls.

After the motion is recognized, the measured samples are collected for 200 ms (20 samples). As soon as more then those 20 samples are measured, the predicted value starts to send the catch point to the PLC, while still collecting measured samples. The measuring is stopped, when the catch moment approaches to less than 150 ms. In the same time, command to PLC is sent, to start the damping procedure.

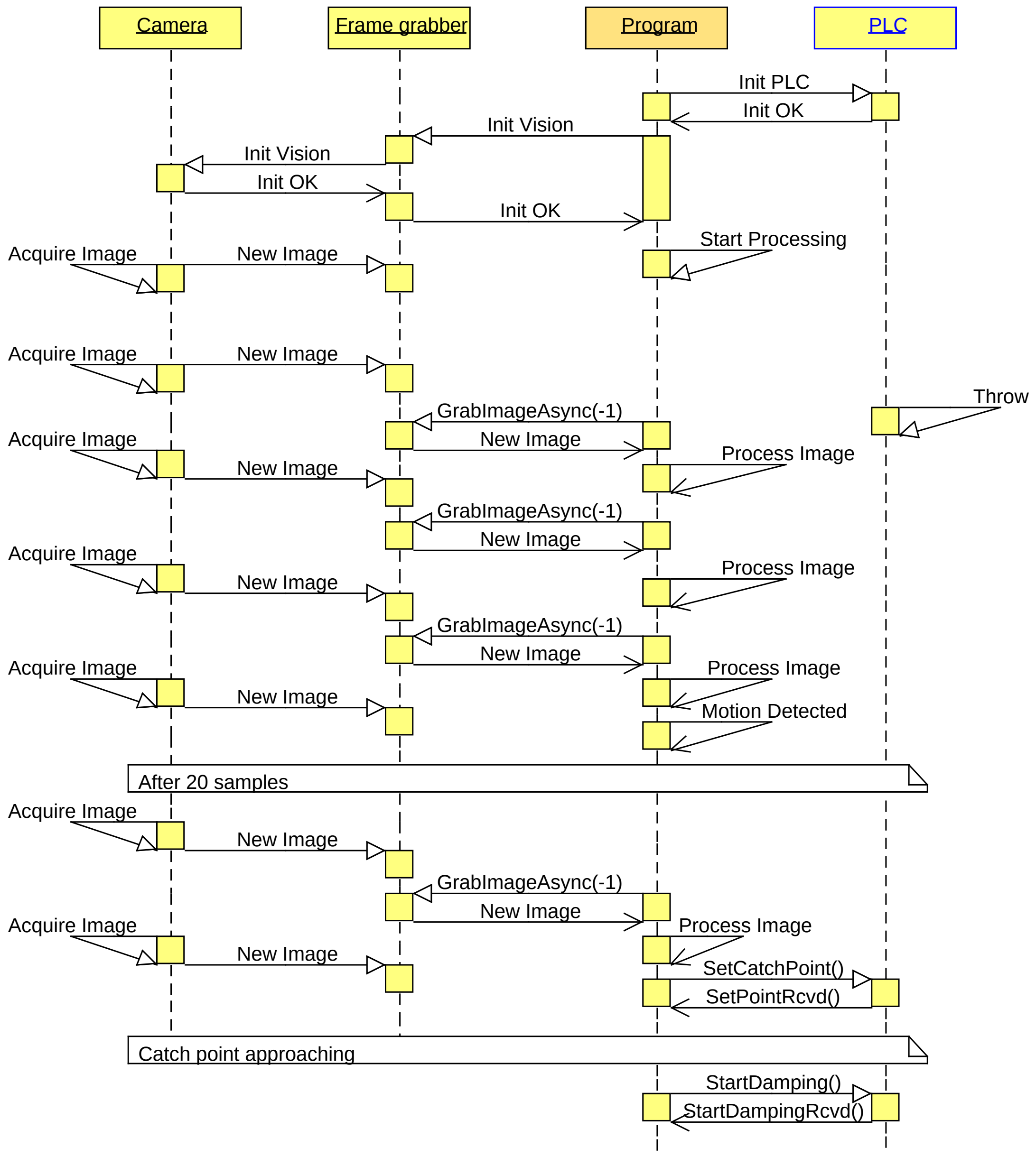


Figure 6.8: Timing of the whole control.

Chapter 7

Implementation

The most suitable operating system for a real time control task would be some hard real time operating system or real time extensions for Windows. Nevertheless, in the time the project was designed, the only available frame grabber drivers were supported for Windows 2000/XP. Although Windows have no real time capability, it is possible to reach fairly good timing results by disabling some unnecessary background processes and services, as experienced for instance in (Kormann, B. et al. 2006). The timing jitter and overall timing accuracy is reduced at minimum by using the camera clock as the clock for the whole measurement, which guarantees, that the sampled scene is acquired at constant time distances. Every time a camera acquires a new image, its processing is triggered automatically.

The control software was designed in Visual Studio 2005, using MFC (Microsoft Foundation Classes). Design with Visual Studio and MFC has its main advantage in the possibility of later simple export to use with RTX real-time extension for Windows, as soon as frame grabber drivers support RTX. Moreover, MFC are the classes designed directly by Microsoft and should be the lowest level tool for creating Windows applications.

7.1 Control structure

The closed position control loop is schematically shown in the figure 7.1. The model itself, as described by (Pšenička, J. 2005), communicates between the PLC and servo drives using Ethernet Powerlink protocol. The Ethernet Powerlink environment is called B&R Automation Net, as described in Chapter 5. To gain access to B&R Automation

Net, PVI is used. In the time this diploma thesis was created, no Ethernet Powerlink PCI card was available on the market. That is why, a standard Ethernet adapter was used and interconnection between PVI and the PC was done via TCP/IP protocol.

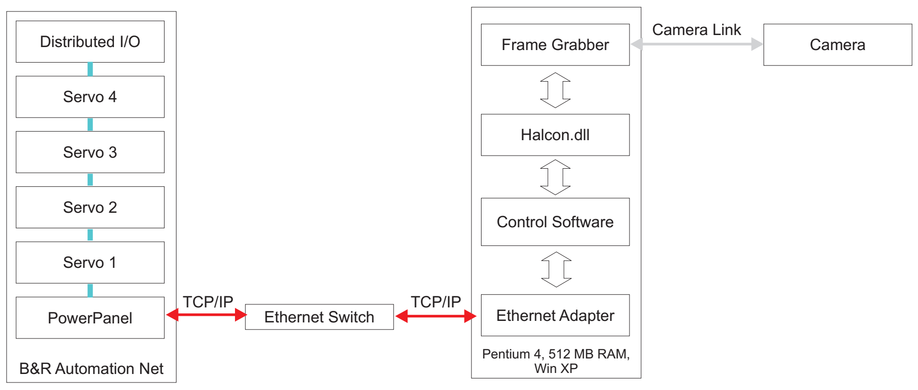


Figure 7.1: The control structure of the model.

The measurement of the ball's position runs on a standard PC Pentium 4, 3.0 GHz with 512 MB RAM, running Windows XP. The structure of the control program is discussed in the following sections.

7.1.1 Measuring program

The measuring program consists of three main classes, which communicate with each other using Windows messages or special member functions. Each of the main classes represents a dedicated function block devoted to special functionality and each of them has also its own dialog window.

All the three main classes are owned by an overview class `CKAT`. `CKAT` is the main dialog class interconnecting all the main parts of the program. The `CKAT` dialog window displays the most important delay times in the program and is also the starting point to access other subclasses as `CCamera`, to see the image processing, `CKalman`, to get an overview of the data filtration and prediction process, and `CMotion`, to access the communication module. The program structure is schematically shown in fig. 7.2.

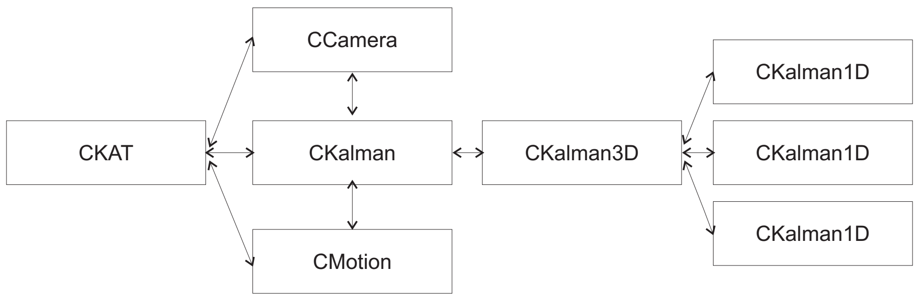


Figure 7.2: The control structure of the model.

7.1.1.1 CCamera

This class takes care of communication with the frame grabber and does the whole image processing. In the class's dialog window, the currently processed image is displayed. To save some computation time, the redrawing of the image window is disabled, if the dialog window is closed. Therefore it is important that the dialog window is used just for debugging purposes.

The dialog has three buttons. One for initialization of the frame grabber and two for starting and stopping of the image processing. When initialized, the instance of `HFramegrabber` class is created and parametrized with a `*.cam` file, described in chapter 3. Then continuous grabbing is set as a special parameter of the frame grabber, so that the grabbing with a given frequency could be started. From that time, camera sends in 10 ms intervals the grabbed pictures into reserved place in the computer's RAM and the stored pictures can be accessed by `GrabImageAsync(-1)` operator. By the initialization, image processing thread is created and waits, until the START button is pressed. The most important part of the code is shown below.

```
GrabbedImage = fg_thread->m_pFG->GrabImageAsync(-1);
...
hBalls = GrabbedImage.FastThreshold(180,255,20).Connection()
          .SelectShape("area","and",100,1100)
          .SelectShape("circularity","and",0.5,1);
...
area   = hBalls.AreaCenter(&ix,&iy);
...
image_points_to_world_plane(...,...,ix,iy,...,&wx,&wy);
```

```
...
dlg_kalman->ball1->measure(wx,wy,0);
...
```

In this part of the program, the image is acquired from the frame grabber first and saved to the variable `GrabbedImage`. Then it is binarized by `FastThreshold` function. This means that all the image pixels with gray value lying in interval $180 - 255$ will be represented as 1 and the rest as 0. With `Connection` operator, the gained continuous areas are connected into bigger continuous elements, which are then searched for specific circular objects of certain size with `SelectShape` operators. The center of the found balls is converted into the world coordinate system, and sent to the Kalman filter.

7.1.1.2 CKalman

This class encapsulates the Kalman filter together with the predictor. As shown in figure 7.2, the `CKalman` class consists of a `CKalman3D` class, which represents one tracked ball. As described in the chapter 3, each coordinate is tracked with a separate Kalman filter, represented by `CKalman1D` class.

Moreover the predictor is a part of `CKalman3D` class. The prediction is done with a simple equation for throw in homogeneous gravitation field. They are simple throw equations 4.8 or 4.9. As input come the data from the Kalman filter. The quality of the predicted catch point from the 20 measuring points is shown in the plot 6.5. It has shown up, that the simplified assumption of not taking the air drag into account is correct.

7.1.1.3 CMotion

`CMotion` class is the implementation of PVI interface to communicate with a special task stored in the control PLC. The class works either in manual mode so that the user can remotely send the drive to certain position at certain speed, or in automatic mode, where the manual values are bypassed by automatically predicted catch point coordinates. Further details can be seen in chapter 5.

For the accurate control, calibration of all the drives is needed. This is mainly because the both vertical sides have the end switches in slightly different heights. Thus, the right part is ca. 7 mm lower then the left one. Accurate positions can be gained from a simple HDevelop application. The read coordinates of end positions for each drive are given to the final control application, so that the coordinates of the drives were the same as the coordinates measured by the camera.

7.1.1.4 Program configuration

Most of the program parameters are configured over `Config.ini` file, which is placed in the program's working directory. The structure of the configuration file consists of three sections.

In `[EECalib]` section the end effector calibration is saved, together with some auxiliary control parameters. The `[ConnectionCfg]` section contains the connection information, needed for communication with the control PLC and finally the section `[ObjectsCfg]` saves all the necessary parameters for creating of PVI objects, communicating with the control PLC.

7.1.1.5 Program ouput

For other data processing, all the measured trajectories are logged into `*.csv` files for every measured trajectory. Besides the measured data, the Kalman filter states are logged as well, together with error covariance trace and the Kalman gain.

With every program start, the files `log_YYMMDD_HHMMSS.csv`, `log_YYMMDD_HHMMSS_kf_x.csv`, and `log_YYMMDD_HHMMSS_kf_y.csv` are created. They can be further processed in MS Excel, or imported to Matlab etc.

7.2 Camera calibration in HDevelop

The proper calibration of the camera has a great influence on the final success of the whole visual feedback. Because I am using position based visual servoing, the accurate conversion of pixel values to the world coordinate system is crucial. The precise calibration is even more critical, because end effector open loop strategy was used, meaning that only the ball's position is measured with the camera, not the position of the models' ball grips.

For camera calibration, HALCON provides a special calibration board and functions, designed to be used with this calibration board. For this purpose, two applications in HDevelop were designed. First application is responsible for finding the camera extrinsic parameters, the second one helps the user to determine the end effector position accurately, so that the appropriate constants for the program could be set.

7.2.1 Calibration plate

The calibration plate can be bought from MVTec or generated by HDevelop in several sizes. The suitable size used for calibration is chosen, so that the calibration plate would form approximately one third of the processed image size. Together with the calibration board comes the calibration file, which describes the dimensions and specific distances of the calibration board.

When calibrating the camera for measuring the position of a known object with a single camera, the calibration board is to be placed on the plane, where the measurement will be done, eventually on the plane parallel to the measurement plane. I chose to place the measurement board on a parallel plane, to the back wall of the model.

The back wall of the model was selected, because it could guarantee, that the calibration board won't be sloping in any direction, when fixed to the linear motors of the model. Important is, that the coordinate system's origin must be shifted to the measurement plane along the z-axis, after the calibration.

7.2.2 Camera pose

The application for determining the camera pose opens the frame grabber, finds the calibration plate, and according to the measured positions of the points on the calibration plate determines the camera position.

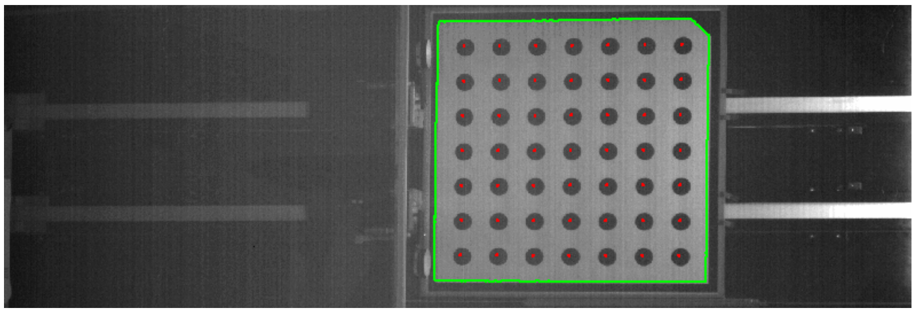


Figure 7.3: Found calibration board.

The calibration process itself in HDevelop is not difficult. The main parts of the code can be schematically described as follows.

```
...
StartCamPar := [0.075,0,1.2e-5,1.2e-5,640,225,1280,450]
```

```
open_framegrabber(...,FGHandle)
...
caltab_points('caltab.descr',X,Y,Z)
grab_image(Im,FGHandle)
find_caltab(Im,Caltab,'caltab.descr',...)
find_marks_and_pose(Im,Caltab,'caltab.descr',StartCamPar,...,StartPose)
camera_calibration(...,StartCamPar,StartPose,'pose',CamPar,Pose,...)
...
```

The operator `caltab_points` reads the coordinates of the calibration point centers from the `'caltab.descr'` calibration board description file. Then the acquisition of the image follows and the used calibration board is searched with function `find_caltab`, in the gained picture. As a result, the region, where calibration board was found, is returned.

With operator `find_marks_and_pose` the calibration marks are extracted from the image, and the camera pose is roughly estimated. This estimation is then passed to the `camera_calibration` function, which computes the parameters more precisely.

In my case, I do not let the function compute the interior camera parameters. It has shown up to be more reliable to set the parameters to fixed values, and let the algorithm determine the camera's pose only. The interior camera parameters are known from the documentation to the camera and lens. More critical are here the exterior camera parameters. The slightest error in camera's angle for instance can cause the ball's flight trajectory to be non-linear. If incorrectly calibrated, this can be recognized on the Kalman filter measurement. The estimated speed in vertical direction must be constant after ca. 10 samples. If it has ascending or descending character, something is wrong.

7.2.3 End effector position

The second calibration application is just a simple tool for finding the end effector position and to shift the origin of the coordinate system. The origin is shifted, so as to be in the lower left corner reachable by the left grip. The positions of the grips are measured, by servos set vertically to values 1000 and 1400 units in horizontal axes and 100 and 18100 in the vertical direction. The measured positions are written to `Config.ini` file as end effector calibration data and unless camera or the model move, they stay constant.

7.3 Application testing

While testing the application it has shown, that one of the weakest parts is the connection between the PLC and the computer. To the switch between them, even other computers are connected, and this was a source of troubles mainly by higher network loads. Sometimes, the catch point position arrived that late, that the servo drives could not reach the set point in time.

Originally application of Ethernet Powerlink PCI card was considered, but unfortunately, no such card was commercially available yet. What could increase the reliability of the connection would be to prefer the traffic between the PLC and PC on the switch level, or to make the connection with crossed Ethernet cable without the need of switch.

The changing time delay has even effect on the prefix time, where the damping of the fall should be sent from the PC. While during the day, higher value is desired, in later hours when the network traffic is lower, the delay must be set appropriately decreased, too.

Chapter 8

Safety functions

The need of safety functionality of the model is highly demanded. The safety elements must minimize the possibility of severe injuries caused by moving parts of the model. To satisfy the safety requirements the standard EN 954-1 was followed. In the first part of this chapter, I will try to summarize the specification of single EN 954-1 norm safety categories, then I will explain, what was done and which components were used in this project. Finally, advantages of safety PLC are pointed out.

8.1 Safety categories

The standard EN 954-1 describes five safety classes to specify what happens to the system if one or more faults occur. The safety classes are divided into 5 categories B, 1, 2, 3, and 4. The detailed description of single categories and their use can be found in (Defren, W., Kreutzkampf, F. 2003).

The first two categories B and 1 are less demanding and are characterized mainly by the used components in safety circuits. Category B requires, that the used safety components must withstand the expected influence. Category 1 adds the requirement of using well tried safety components and principles. In both classes, occurrence of a single fault can lead to loss of safety function, however the probability of fault is lower by category 1.

All the higher categories are distinguished mainly by the structure. Category 2 adds a requirement of safety function check in suitable intervals by the machine control system. This requirement guarantees, that the safety function can be lost only if a fault occurs between the checks. By the next check, the fault is found. Category 3 demands that a single fault in any of the used parts does not lead to loss of the safety function. This single fault should be detected by the check. Category 4 adds the requirement, that accumulation of faults must not lead to loss of safety function.

8.2 Safety elements

The safety circuit is shown in fig. 8.2. It was designed, so that it would correspond to the category 3 of EN 954-1. The circuit has three main safety functions: supply voltage monitoring, emergency stop button, and door lock.

8.2.1 Voltage monitoring

The supply voltage on the three phases is monitored by Pilz S3UM relay, so that the model was disconnected from the main power supply in case of over or under voltage. The relay is connected with manual reset button. By power failure, the emergency stop is raised. The structure corresponds with category 3 of EN 954-1, because two steps have to be done, to recover power supply. The voltage must be in the defined range and reset button must be pressed.

8.2.2 Emergency stop

Second safety relay Pilz PNOZ X2P, disconnects the power supply after emergency stop button is pushed. The relay itself is constructed, so that it would correspond to the category 4 of EN 954-1. The button is connected to the safety relay with double wire. The dual channel relay secures that if one single fault occurs, the safety function is maintained. The emergency stop button is placed on the operating panel in the lower right corner, see fig. 8.1. The relay is connected with manual reset button, so that the power voltage could not be recovered immediately after unblocking the emergency stop button. The structure meets the requirements of EN 954-1, category 3, as well, because all the parts from the button to the relay are doubled.

8.2.3 Door lock

The safety door lock Möller AT0-11-24DMT is connected to another PNOZ X2P relay with a single wire. The door lock prevents the plexi glass door from opening when the model is operating. The lock is controlled from the operation panel, by a key in upper right corner, see fig. 8.1. The area of the model can thus be accessed in two steps, which is required by the category 3 of EN 954-1. The key must be unlocked and the door lifted.

When unlocked, the drives are blocked in two ways. The quick stop input is activated, and the drives' controllers are switched off by deactivating the enable input. After closing the door and turning the key to the locked position, the quick stop is deactivated again, and after a preset time on the Möller DIL ET 11-M-A, all the drives are enabled and the controllers can be switched on again.

8.2.4 Operating mode indication

The operating status of the model is indicated by a three-color signal light on the operation panel. This signal light is connected to distributed input/output module, see fig. 8.1. It is controlled from the PLC and depending on the used functionality various operating states can be indicated.

8.3 Use of safety PLC

Before the tool is available for students, B&R safety PLC, which was released short before finish of this work, will be installed. This will be subject of other diploma thesis. The task of the safety PLC will be to monitor all the traffic going via Ethernet Powerlink from the control PLC to the drives and evaluate the meaning of the commands and their effect. If it should happen, that the commands to be processed could lead to damage of the model, quick stop must be immediately raised. The program of the safety PLC must not be accessible from Automation Studio, so that it could not be reprogrammed by the students.

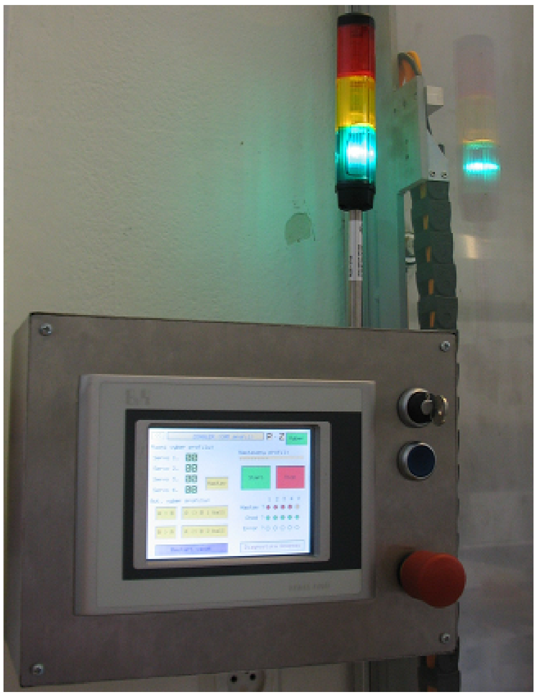


Figure 8.1: Operating panel with door lock, reset button, and emergency stop button.

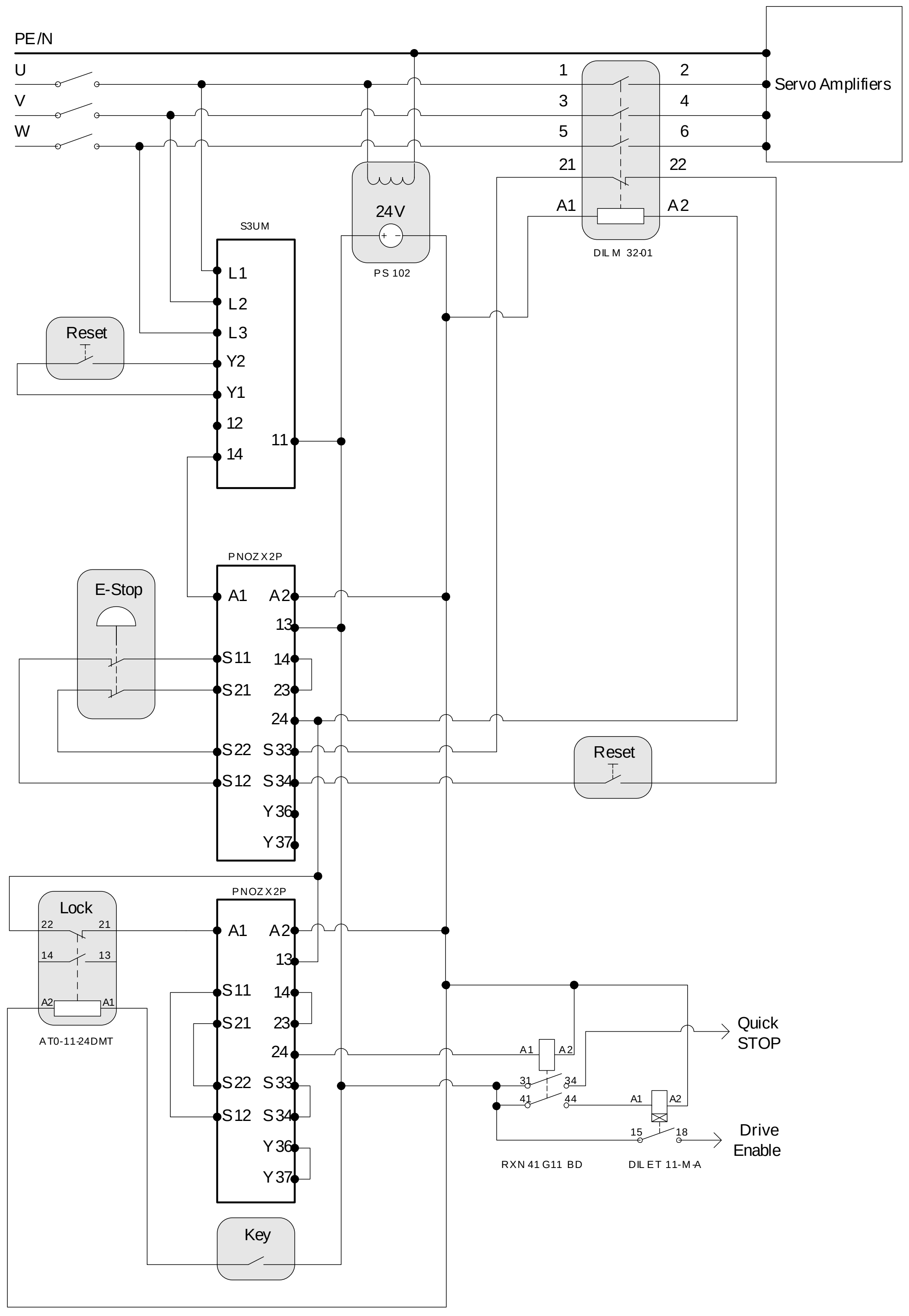


Figure 8.2: Safety circuit.

Chapter 9

Conclusion

The main goal of this diploma thesis was to close a positional visual feedback to the control of a model, juggling with a billiard ball. Billiard ball throw is a very fast process lasting around 700 ms. Within this time, the throw trajectory has to be measured and the servo drives adjusted to the computed catch point position.

The structure of the solution was designed, after studying similar visual servoing projects and after consulting the problem with the vision software and hardware producer.

Although real time operating system, as well as deterministic communication protocol could not be utilized, because either appropriate drivers or the product itself were not available on the market, this thesis proves on practical working application, that control of very fast processes, as flying billiard ball definitely is, is possible with nowadays machine vision systems even with conventional PC with non real time operating system.

During work on the project, various interesting problems arose and are discussed in this thesis. In machine vision, light reflexions, ball occlusion, ball gleam, or even missing samples had to be dealt with. Other problems had to be solved, while minimizing the image acquisition and processing time.

In the next extensions of the project, the third coordinate along the z-axis could be measured, by evaluating the size of the billiard ball. Nevertheless, to reach some reasonable performance by this method, better lightening must be installed. What could increase the overall quality of the vision system, would be the C-mount lens with a bigger picture diagonal. Considered could be even the application of F-mount lens with a C-mount adapter.

The measuring application can be easily extended with a tracker module distinguishing more balls, if more billiard balls should be used for juggling. Nevertheless, the current PC would have to be upgraded to use the full potential of the frame grabber with the

PCI-X bus and Full Camera Link interface. The application is programmed to use a multi processor computer, which would increase the image processing power, as well as more operating memory.

The automatic calibration method might be applied as well. If some calibration points were sticked to the back wall of the model a simple calibration function could be built into the measuring program and called with each program start. This would guarantee that the camera pose is always properly set.

Last part of the thesis was devoted to the safety standard EN 954-1. Some necessary safety equipment was installed to minimize the possibility of severe injuries caused by moving parts of the model in the control lessons. As future extension of the model, a safety PLC, increasing the model robustness, will be installed together with another servo drive, collecting the fallen billiard balls. .

Under `http://dce.felk.cvut.cz/juggler`, I designed a simple web page, for propagation purposes of the project, where the latest development of the model will be presented.

Bibliography

CORKE, PETER I., *Visual control of robot manipulators - A review*, Preston, Victoria, 3072, Australia: SCIRO Division of Manufacturing Technology, 1994.

DEFREN, W., KREUTZKAMPF, F. *Machine safety in the European Community.*, K.A.Schmersal GmbH,ISBN 3-926069-13-9, January 2003.

DICKMANNS, E. D. and GRAEFE, V. *Applications of dynamic monocular machine vision.*, Machine Vision and Applications, 1, pp. 241-261, 1988.

FRESE, U., BÄUML, B., SCHREIBER, G., SCHAEFER, I., MÄHNLE, M., HIRZINGER, G. *Off-the-shelf vision for a robotic ball catcher.*, Institute of Robotics and Machatronics, German Aerospace Center (DLR Oberpfaffenhofen), 82230 Wessling, Germany, 2000.

GESCHKE, C. *A robot task using visual tracking.*, Robotics Today, pp. 39-46, Winter 1981.

GILBERT, A. L., FLACHS, G. M., ROGERS, R. B., YEE, H. U. *A real time video tracking system.*, IEEE Trans. Pattern Analysis and Machine Intelligence 2(1), January 1980.

HAVLENA, V., ŠTECHA, J. *Moderní teorie řízení.*, Skriptum ČVUT-FEL,ISBN 80-01-02095-9, 2000.

HILL, J., PARK, W. T. *Real time control of a robot with a mobile camera.*, 9th International Symposium on Industrial Robots, pp. 233-246, March 1979.

KALMAN, R. E. *A new approach to linear filtering and prediction problems.*, Transcript of the ASME-Journal of basic engineering, pp. 35-45, March 1960.

KORMANN, B. et al. *Treffsichere Dartscheibe.*, TU-München, http://www.treffsichere-dartscheibe.de/, 2006.

KRUGLINSKI, D. J., SHEPHERD, G., WINGO, S. *Programujeme v Microsoft Visual C++.*, Computer Press Praha, ISBN 80-7226-362-5, 2000.

MVTEC *HALCON 7.1, Application guide*, MVTec Software GmbH, München, Germany, 2005.

PRATA, S. *Mistrovství v C++, 2. aktualizované vydání.*, Computer Press Brno, ISBN 80-251-0098-7, 2004.

PŠENIČKA, J. *Synchronizace servopohonů.*, Diploma Thesis, ČVUT-FEL, 2005.

SANDERSON, A. C. and WEISS, L. E. *Image-based visual servo control using relational graph error signals.*, Proc. IEEE, pp. 1074-1077, 1980.

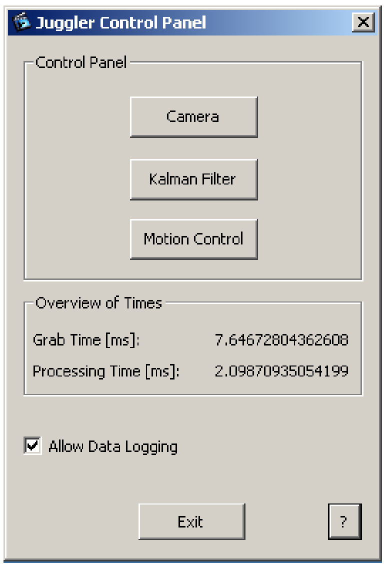
SHIRAI, Y.,INOUE, H. *Guiding a robot by visual feedback in assembly task*, Pattern Recognition, 5, pp. 99-108, 1973.

WEBBER, T. E. and HOLLIS, R. L. *A vision based correlator to actively damp vibrations of a coarse-fine manipulator.*, RC 14147 (63381), IBM T.J. Watson Research Center, October 1988.

WELSH, G. and BISHOP, G. *An introduction to the Kalman filter.*, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, Updated: April 5, 2004.

Appendix A

Program screenshot


Juggler Control Panel

Control Panel

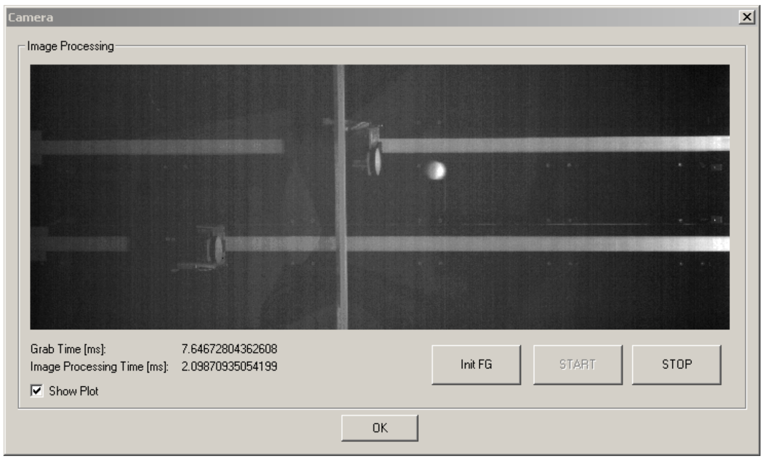
Overview of Times

Grab Time [ms]: 7.64672804362608

Processing Time [ms]: 2.09870935054199

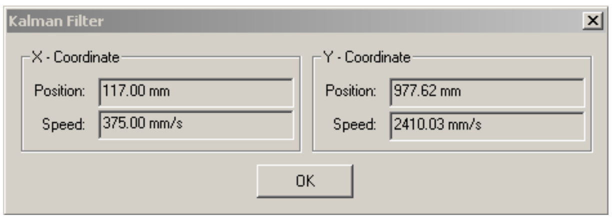
[✓] Allow Data Logging

Figure A.1: Screnshot of the measuring program's main dialog.



Grab Time [ms]: 7.64672804362608
Image Processing Time [ms]: 2.09870935054199
[✓] Show Plot

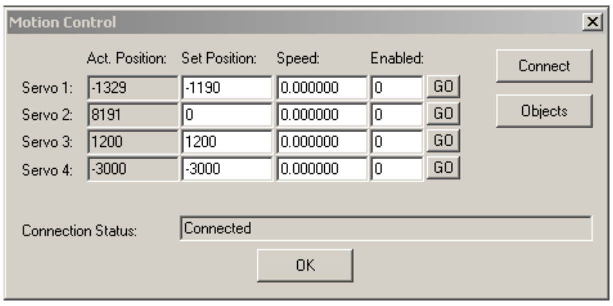
Figure A.2: Screnshot of the image processing window.



X - Coordinate
Position: 117.00 mm
Speed: 375.00 mm/s

Y - Coordinate
Position: 977.62 mm
Speed: 2410.03 mm/s

Figure A.3: Screnshot of the Kalman filter window.



	Act. Position:	Set Position:	Speed:	Enabled:	
Servo 1:	-1329	-1190	0.000000	0	GO
Servo 2:	8191	0	0.000000	0	GO
Servo 3:	1200	1200	0.000000	0	GO
Servo 4:	-3000	-3000	0.000000	0	GO

Connection Status: Connected

Figure A.4: Screnshot of the motion control window.

Appendix B

Photos

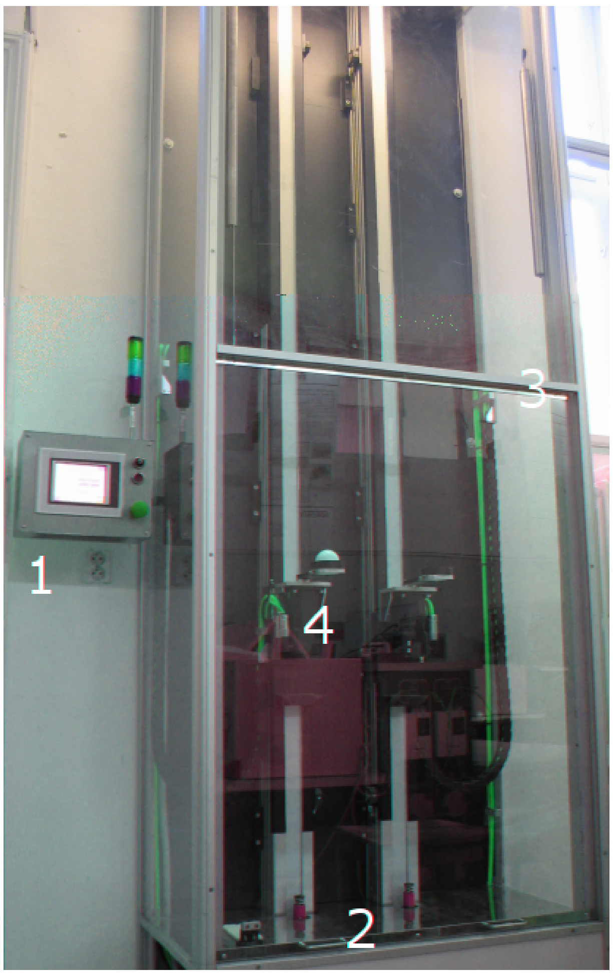


Figure B.1: The model: 1 - the operating panel, 2 - door, 3 - steel bar, 4 - ball grip.

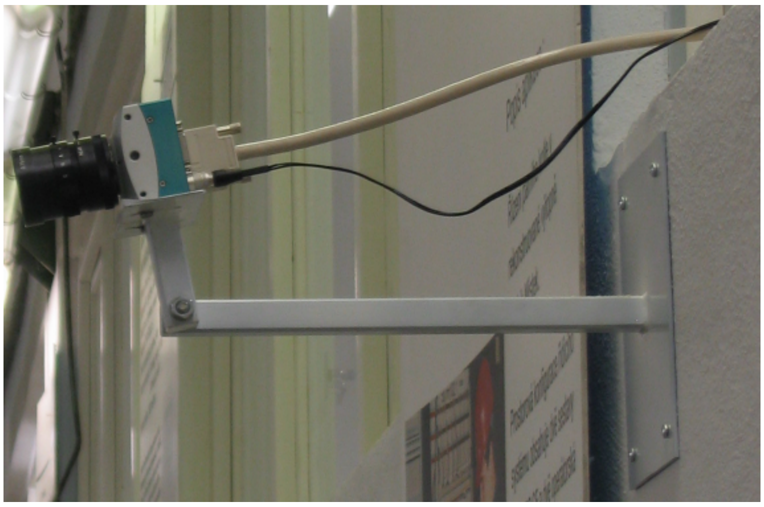


Figure B.2: Measuring camera.

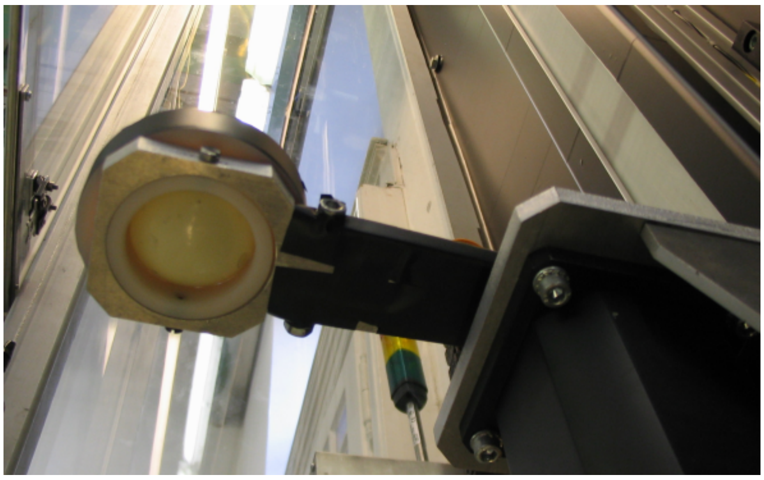


Figure B.3: Detail of a ball grip.

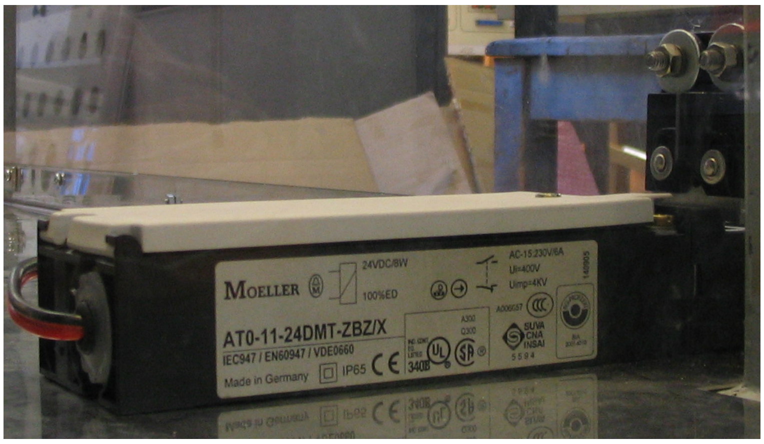


Figure B.4: Detail of installed safety lock.

Appendix C

Content of inserted CD

A CD is enclosed to this work, on which the source codes, data sheets and example programs are stored.

- `Doc\`: Used manuals, documentation, and data sheets.
- `DT\`
  - `LaTeX\`: Source code of this text in LaTeX.
  - `PDF\`: Final PDF file with the diploma thesis.
- `SW\`:
  - `PC\`:
    - `KAT\`: Measurement program with source codes.
    - `KatHDev\`: Applications in HDevelop. Calibration tool, end effector calibration, algorithm debugging.
    - `Matlab`: Matlab testing m-files.
  - `PLC\`: Control program for the Power Panel in Automation Studio.