Institute for Automation

Department Product Engineering,

Montanuniversität Leoben

Computer Vision Based Realtime Opto-Mechatronic Tracking System

MASTER THESIS

Ernst Johann Theußl

Field of Study:
Mechanical Engineering

Supervisor:

ASS.PROF. DIPL.-ING. DR.MONT. GERHARD RATH

O.UNIV.-PROF. DIPL.-ING. DR.TECHN. PAUL O'LEARY

March, 2018

Abstract

For purpose of high accuracy image acquisition of moving objects with a finite shutter speed of the observing camera, a camera-tracking mount is required to avoid blurred objects during the exposure time. The tracking mount could have one-, two- or multi-axis kinematics with a fixed location and camera perspective or variable location with the requirement of calibration of the relative camera coordinate system. The calibration calculates the correlation of camera position, kinematics of the mount and the movement of observed objects. For testing, in this thesis a two-axis equatorial mount is used to track astronomical objects such as stars and satellites. The exact solution of the forward kinematics is calculated, which includes the coordinate transformation of the projection of astronomical objects, represented by the camera data, and the mount coordinate system. Further a reduced kinematics is derived from the exact solution, to provide a generic two-axis object tracking system using inverse kinematics. A high-performance optimised multiple object tracking software is developed, which detects objects and calculates the two-axis movement of the mount in real-time, based on the two-dimensional deviation in the acquired image. For this reason, optimised computer-vision based algorithms on a SoC (system-on-a-chip) system are used together with a newly developed electronic control interface.

Zusammenfassung

Zum Zweck der hochpräzisen Bilddatenerfassung von bewegten Objekten mit Kamerasystemen mit einer endlichen Verschlusszeit ist es notwendig, dass das Beobachterkoordinatensystem den Objekten mit einer Nachführung folgt, da diese ansonsten unscharf dargestellt werden. Diese Nachführung kann eine einachsige, zweiachsige oder mehrachsige Nachführkinematik mit einer feststehenden Position im Raum und gleichbleibender Kameraposition sein oder eine variable Position von Nachführung und Kamera haben, die es notwendig machen, das System zu kalibrieren. Die Kalibration berechnet den Zusammenhang der Bewegung eines Objekts, der Kameraorientierung und der Kinematik der Nachführung. In dieser Arbeit wird für Versuchszwecke eine bestehende, zweiachsige sogenannte parallaktische Nachführung zur Verfolgung astronomischer Objekte, wie beispielsweise Sterne und Satelliten, verwendet. Dazu wird eine exakte Lösung der Vorwärtskinematik berechnet, welche die Transformation der Projektion astronomischer Objekte, die den Kameradaten entsprechen, auf das Montierungskoordinatensystem vollzieht. Für eine weitere allgemeine Anwendung der Objektverfolgung mittels einer zweiachsigen Kinematik wird aus dieser Transformation eine vereinfachte Rückwärts-Kinematik abgeleitet. Um die Nachführung mit zweiachsiger Kinematik zu steuern, wird eine optimierte Mehrfach-Objektserkennungssoftware entwickelt, welche die zweidimensionale Abweichung am Beobachtungsbild in die gegebene kalibrierte Kinematik umrechnet. Zu diesem Zweck werden optimierte Computervision-Algorithmen in Verbindung mit einer speziellen und hochempfindlichen Astrokamera, einem System-on-a-Chip System und einem eigens entwickelten elektronischen Steuer-Interface verwendet.

Statutory declaration

I declare in lieu oath, that I wrote this thesis and performed the associated research myself, using only literature cited in this volume.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich diese Arbeit selbständig verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und sonst keiner unerlaubten Hilfsmittel bedient habe.

Leoben, Tue 6th Mar, 2018

Ernst Johann Theußl
Matr.Nr.: 00630777

Acknowledgement

I dedicate this thesis to those who have supported me in this work and especially those who have made it possible to me to pursue my interests and always encouraged me to live my creativity as an essential part of my life.

-

I would like to thank Gerhard and Paul for the opportunity to write a thesis out of my personal idea and supporting me. In general, I would like to thank the chair of automation for their great support.

-

Great thanks also to the University of Leoben, especially to Prof. Paris, for the financial research funding, which made it possible to improve the equipment and thus the result of this thesis.

-

Many thanks to Prof. Gfrerrer for his great engagement regarding to the exact kinematic solution.

-

I would like to thank Richard Kunz for having been such an excellent physics teacher who has supported me already in my younger years.

-

”Teleskop Service GmbH”, Mr. Rolf Nicolay, stood by my side when I needed a hardware exchange for this project very quickly. Many thanks for that and your trust.

-

Last but not least I want to thank my family, who made my study and creative excursions possible. Especially I thank my grandpa († 12/25/1999) for his inspiring reflections on physics and mathematics.

-

Contents

List of Figures

List of Tables

Introduction

Computer vision based opto mechatronic object tracking in this thesis is used for the control of a mechanical system through a software-hardware solution consisting of a system-on-a-chip computer, a camera and a suitable optics for the magnified observation and tracking of for away objects. The mechanical system is an existing two-axis equatorial telescope mount, with the possibility to control the two axis with defined speed. For this reason, a new electrical control interface is developed. Further, a compact structure of components are placed on the mount, including the mentioned required components. For the purpose of easy tracking control, the input can be done via a touch screen and a developed interface. To developed software is focused on high accuracy tracking and a user friendly control interface, including parallel tasks, such as image acquisition with a light sensitive camera. The thesis leads through the whole development process, starting at computer vision, assembling all required components, required coordinate transformation of the kinematics and the final result, a derived reverse kinematics.

Chapter 1

Computer Vision

Computer vision with its algorithms is the feature of machines to see, understand images and evaluate captured images or videos to gather desired information's and further the analysis, inspection, process control and mechanical guidance of visual and mechanical systems. The aim is to extract real world data in numerical and computer understandable data. For this purpose, the amount of acquired data has to be reduced and classified for essential information. [1]

1.1 Object Tracking

The potential of computer vision controlled one or more axis kinematics is enormous. The implementation of theoretical algorithms got possible with the increase of computational power of microprocessors or whole system-on-a-chip systems, which provides real-time image or video processing. It is an essential factor for accurate control in closed-loop systems, which requires fast positioning data. Another cue is the robustness of computer vision algorithms in real environment and filtering the substantial image data in order to avoid too intense computational calculations. For this reason, most of the existing algorithms are designed for special applications with exact framework conditions, to provide a stable functionality [1], which is not always possible. As example, the difference in needed resources between tracking an special coloured sphere in a certain distance and multiple objects without a unique shape in any distance of the camera is very different. Mostly there are combinations of algorithms in advanced and complex systems to produce the most stable state of the art success in tracking. In this thesis a set of algorithm is applied on non unique different sized objects in a consistent framework with the possibility of the change of the number of objects and the special feature of automatic alignment and calibration of the used kinematics depending on the camera position and kinematic itself. In particular a video stream divided into image frames is processed with the aim of the transformation of the image coordinate and the difference of objects from frame to frame

into the given kinematics coordinates and controls.

1.2 Application

Many applications make it necessary to track objects for different purposes. That could be a movement oft the camera for high accuracy image acquisition for reasons of the finite shutter speed of cameras, security features in autonomous driving vehicles, artificial orientation of robots in moving environments, satellite tracking, and so on. Fast object tracking requires fast image processing algorithms and computing power. For this reason, the application of image processing or computer vision is a relatively young scientific discipline. Even if the computer power increased, computer vision is based on optimized matrix based image processing algorithms to reduce the required power and provide the functionality even on small SoC systems in real time [2]. In the literature and as example for typical object tracking problems, some different kind of features are presupposed in single or multiple distinguishable object detection and tracking. A lot of tracking algorithms for multiple object detections are using unique object features such as colour or shape and size to differentiate between objects . In some cases it is necessary to differentiate between moving objects without unique features or differentiate between moving objects with a controlled and tracked camera. For each case, different algorithms are essential to determine a working and stable object tracking mechanism [3].

1.3 Algorithms

As already mentioned there is a variety of different algorithms for different purposes of object tracking. The achieve better results, even combinations of different algorithms are possible. Considered separately the basic algorithms will be described in the following sub chapters.

1.3.1 Colour Based Object Tracking

For single and multiple object detection it is possible to track different objects with different colours in an easy way. Every pixel value with its colour is computed and evaluated for the desired pixel value (colour) in RGB or intensity grey scale images. Connected same-colour-pixels are summarised to unique labelled objects, which is possible for every distinguishable colour. The reduce the computing power, the background or all inessential pixel values are removed through a threshold algorithm, while an algorithm calculates the moments of each object and returns its centre, area and boundary pixel values. This algorithm is less dependent on the object speed cause of the colour recognition of the desired tracked objects [1].

1.3.2 Shape and Size Based Object Tracking

Objects that differ in colour from the background can also be examined on their shape and size. On the digital image just a two dimensional projection of the object can be observed, so the most easy algorithms are based on spherical objects which only change their size dependent on their distance to the camera based on the optical illustration and not their shape such as cubes or other more complex objects. Based on spherical object tracking, algorithms for circle detection can be applied with a second differentiation for multiple objects including features such as colour or size. To reduce the required computational power a radius range has to be given to avoid the detection of faulty objects like small edges [1].

1.3.3 Tracking of Non Unique Feature Objects

Multiple non unique coloured or shaped objects with a noticeable difference in colour to the image background need another kind of algorithm for distinguishable object tracking, which is a essential part of this thesis. To track multiple objects in this way, just the position of each object, except overlapping ones, is evaluated. To track objects over any number of frames, a comparison of all possible positions from frame n to frame n+1 is necessary. All distances are compared with a Gaussian probability function to obtain the most appropriate position in the following frame.

1.4 Limits of Multiple Non Unique Object Tracking

Limits are given by the relative movement of the tracked objects and the shutter speed of the image sensor and the program cycle speed. Neglecting the program cycle speed, a physical parameter determines the possible speed: The available light-data for object detection per frame depends on the image sensor sensitivity and the optics. Regarding the used tracking algorithm, the distance between the movement of all objects are calculated and compared in a Gaussian distribution, to find the most likely distance of the objects between frame n and frame n+1. However, if the distance of movement between the compared frames, depending on the shutter speed, is too long, no accurate algorithm can be applied and the objects get lost. Thus the smaller the shutter speed of the camera is, the smaller the distances of constant moving objects between the frame becomes, which increases the accuracy and stability. This circumstance shows up the importance of compliance among object speed and shutter speed of non unique multiple object tracking.

Chapter 2

Astronomy

2.1 Two-Axes Equatorial Mount

To observe distant objects, an optical instrument with a suitable and adjusted focal length is required. The focal length depends on the distance between the observer and the object and additionally on the required accuracy of tracking. The focal length in conjunction with the camera sensor resolution represents the limit of motion detection. To guide the optical observation instrument, so called mounts are used. To hold the position of an observed object at the same position for the observer, an apparent movement of objects, coming into existence through the movement of the earth, which must be balanced. For astronomical applications are usually equatorial mountings used, with the special feature of parallelism of one mount-axis and the earth axis [4]. Thereby the parallel axis rotates with the same angular speed as the earth, but opposite. This condition allows a one axis object tracking, just in case, the mount is perfectly aligned. If not the, a two-axis tracking has to be applied to compensate the apparent movement, with calculated angular speeds of both axes.

2.2 Astronomical Coordinate Systems

To specify the position of astronomical objects, depending on time, date and observer position, special coordinate systems are required. The basis of that kind of coordinate systems are spherical coordinates, which split up into the absolute and relative coordinate system. The relative coordinate system, uses for its basis the origin of the observer, while absolute systems uses a neutral basis like the centre of the earth. Mostly relative coordinate systems are used to avoid further calculations for exact positions of astronomical objects. Using that common coordinate system, available GoTo mounts uses a relative coordinate transformation, by having the observer coordinates, time and date saved due initialization of the mounting. By that, it is possible the calculate object positions and

describe their position through two angles, thus one angle is the height over the observers position horizon, the second angle is given between the north-south line, also called the meridian, and the object, measured positive clockwise from the north. However, it is always important to note, that this coordinates are not persistent by their coordinate-time-date-dependence [4].

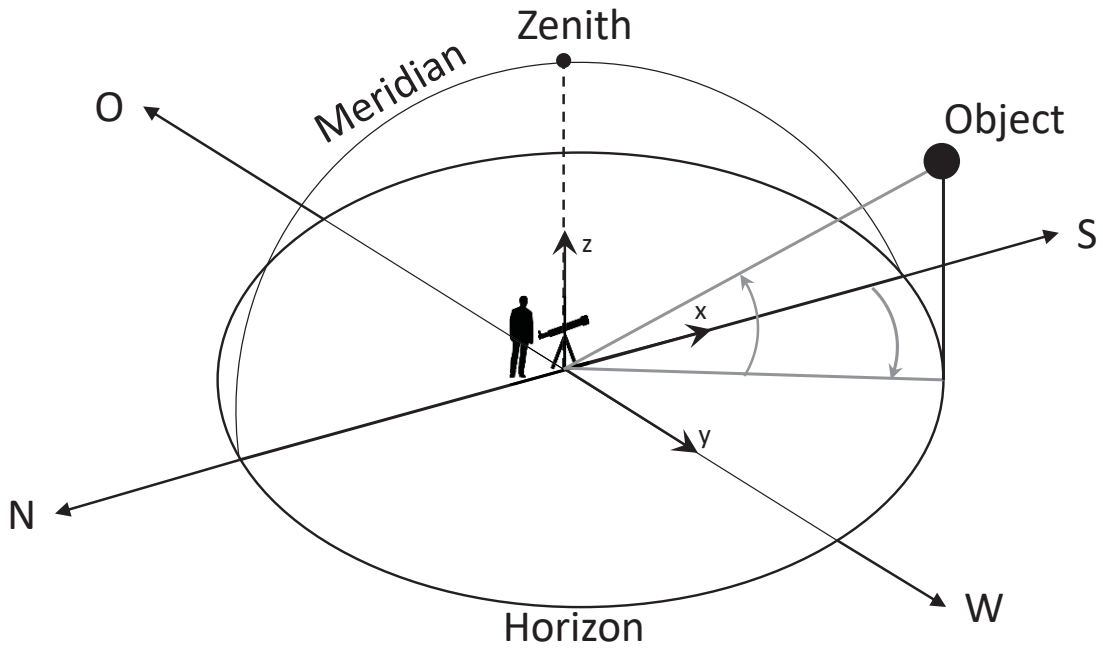

Figure 2.1: Astronomical coordinate system

2.3 Autoguiding

Autoguiding is the in the literature known term of tracking a star with a given telescope mount in relation to the misalignment. Through the control of both axes, right ascension and declination, in positive and negative direction, the star is guided over the whole observation and stands still in the image coordinate system. That includes the correction of the movement with two axis, which causes a third, unguided movement, thus results into an image field rotation in dependence of the misalignment. This condition takes less effects in finite exposure or observing times. The guiding function is a closed loop system, with a cyclic target-actual comparison of astronomic objects from image frame to image frame. The images are taken through a so called guidescope, with a typical focal length of about 180 mm, which guarantees in combination with a sensitive camera, accurate tracking possibilities.

2.4 Image Field Rotation

Due a misalignment of the mount a accompanied deviation of the right ascension axis results. Considering a two-axis kinematics, this leads to a rotation of the acquired image about its centre. The angular speed is depending on the deviation, while the visibility of it depends on the focal length of the observer optics and the duration of observation. This angular speed of image rotation is given by the derivation of the parallactic angle η. Given is any point on the celestial sphere, where a spherical triangle can be obtained through the observed object, the celestial north pole and the zenith. The inner angle of the observed point is designated as the parallactic angle η, together with the right ascension angle t and the azimuth angle Az [5].

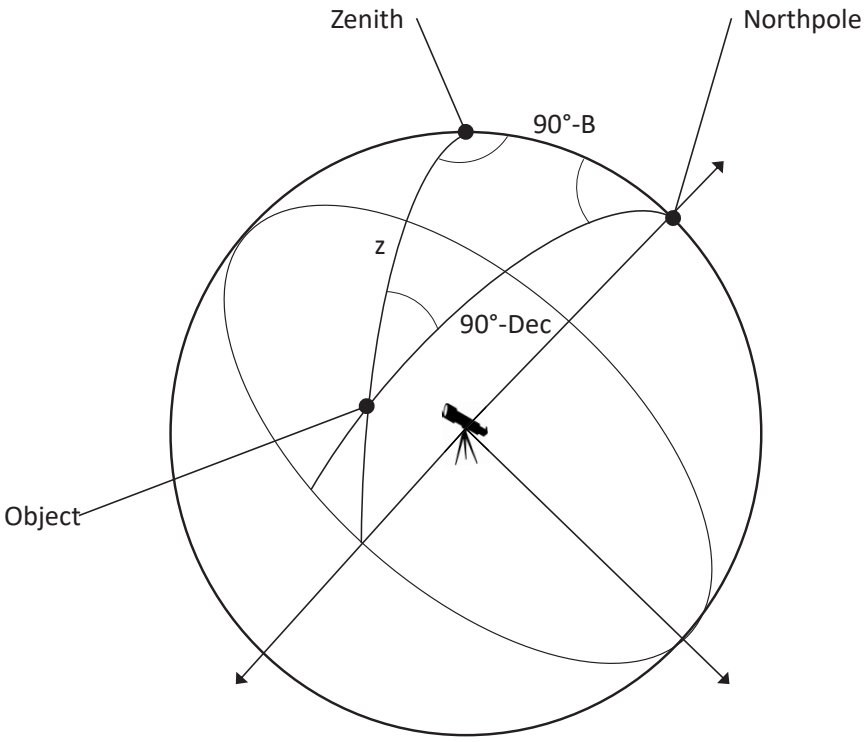

Figure 2.2: Spherical triangle

The sine rule applied on the spherical triangle gives

$$\frac{\sin(90^\circ - B)}{\sin(\eta)} = \frac{\sin(z)}{\sin(t)} \tag{2.1}$$

with $\sin(90 - B) = \cos(B)$ leads to

$$\sin(z)\sin(\eta) = \cos(B)\sin(t). \tag{2.2}$$

To obtain the correct quadrant of η, the sine-cosine rule is applied, which gives

$$\sin(z)\cos(\eta) = \cos(90-B)\sin(90-\delta) - \sin(90-B)\cos(90-\delta)\cos(t) \tag{2.3}$$

with $\sin(90-B) = \cos(B)$ and the division of (4) and (6) gives, with the determination of η

$$\eta = \arctan(\frac{\sin(t)}{\tan(B)\cos(\delta) - \sin(\delta)\cos(t)}). \tag{2.4}$$

Through the time derivative

$$\dot{\eta} = \frac{\mathrm{d}\eta}{\mathrm{d}t} \tag{2.5}$$

$$\dot{\eta} = \Omega\frac{\cos(\varphi)\cos(A)}{\cos(h)} \tag{2.6}$$

when

$$\Omega = \frac{1U}{d} = \frac{360^\circ}{86146.091s} = 4.178 \cdot 10^{-3}\frac{deg}{s}. \tag{2.7}$$

[5] The angular speed increases with the deviation of the misalignment. The visibility results in circular arches while long exposures of the observed area on the edges of the image. However, this phenomenon has a rather small effect and is neglected in the reduced kinematics, since this rotation has to be considered only for astronomical objects. Thus, a generic solution of two axis tracking is intended.

t ... right ascension

δ ... declination of the object

B ... latitude

η ... parallactic angle

z ... zenith

Chapter 3

Test Setup

To test the developed software, especially the kinematics control behaviour, a so-called equatorial telescope mount of the manufacturer 'SkyWatcher' (EQ6) is used. The mount provides high precision guiding of the whole observer system, consisting of a guide-scope with the astro-camera, a 102mm opening achromatic refractor (TS Optics Photo-Line 102 mm [6]), an APS-C or full frame observer camera and a system-on-a-chip computer (Raspberry Pi). The focal length of the guide-scope is 180 mm and 700 mm of the refractor telescope, which is the main observing optics for visual use by eye or camera. To control the mount, a suitable 4 channel electrical input is given by the manufacturer of the mount. The complete test setup is shown in figure 3.1.

3.1 EQ6 Mount

The used mount is a so called equatorial mount of the manufacturer 'SkyWatcher'xy. The particularity of this kind of mount is the parallelism of the right ascension axis of the mount and the axis of the earth. If the alignment is perfectly done, without any deviation in parallelism, the apparent movement of the stars can be balanced by one axis. This alignment-task is not easy to handle, so it is assumed that an standing deviation exists. To correct the deviation while tracking with an imperfect alignment, a two-axis movement correction is necessary, which can be provided through two stepper motors, which moves primary the right ascension axis and the secondary declination axis. For high accuracy positioning with the given stepper motors, the maximum load of equipment on the mount is about 15 kg. In stock the mount is controlled via a hand control unit, which allows additional settings such as observer position, stepper motor rate, ST4 port speed and so on. The alignment process includes three degrees of freedom. The first degree of freedom is represented by the parallelism to the ground of the mount, which is adjusted with a spirit level on it. The second degree of freedom is the pile high which has to be adjusted via a screw on the mount. The third degree of freedom is the azimuth angle, which can

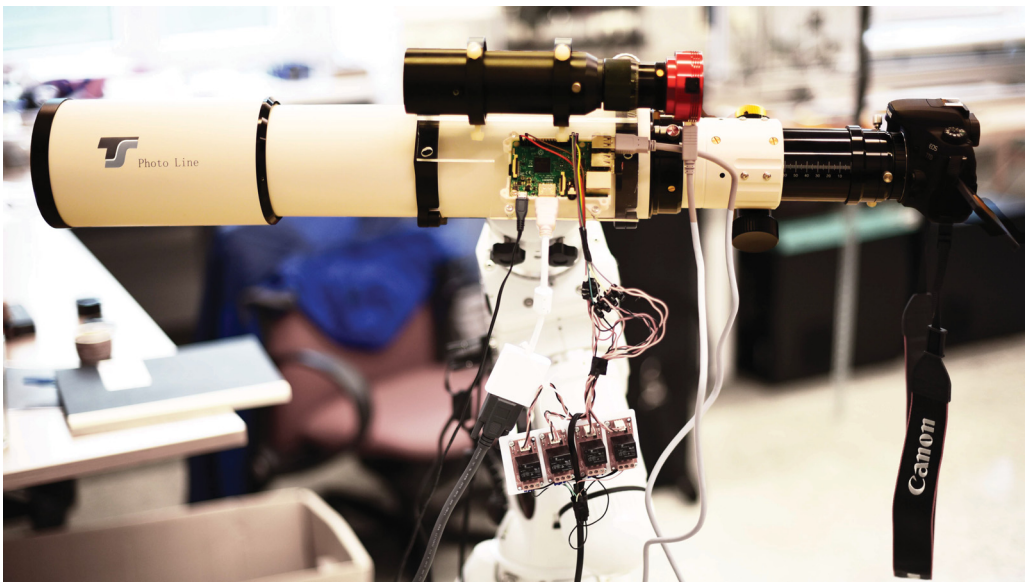

Figure 3.1: Test setup

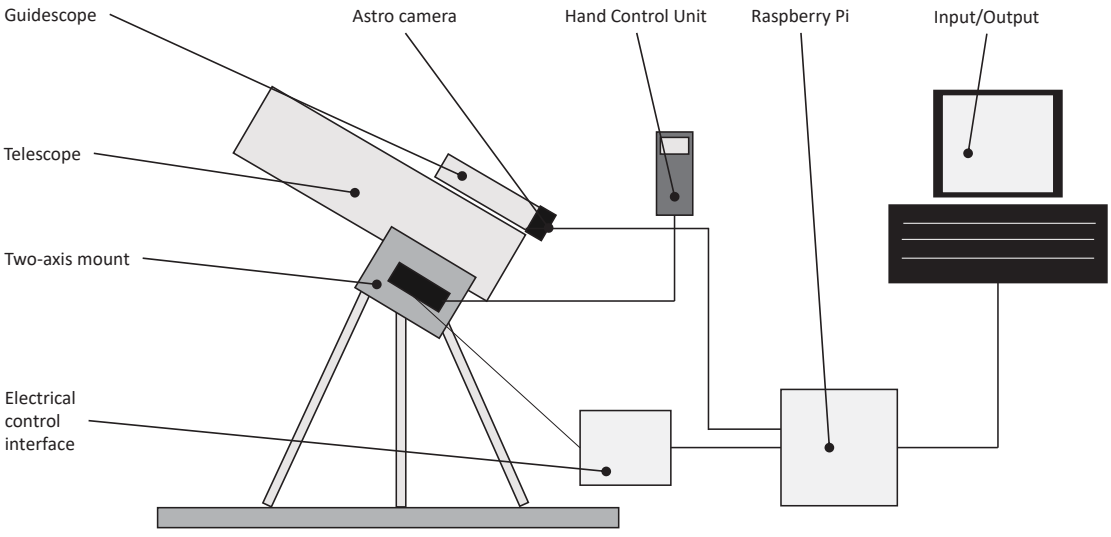

Figure 3.2: Overview test setup

only be aligned by the declination of the right ascension axis and the rotation centre of the sky, which can be found through a pole finder in the night sky. The small eccentricity distance to Polaris depends on the current time of day and date and can be setted up in the finder scope. The finder scope or pole finder is placed in the hollowed right ascension axis with the ability of fine adjustment the azimuth angle with screws on the head of the mount [7].

3.2 ST4-Connector

In stock, the mount assumes a perfect alignment and rotates with the opposite sidereal speed around the right ascension axis. This leads to errors, so it is necessary to correct that with the movement of two axes. For this, a predefined electrical input of the mount is given (ST4 port) which is a 5 pin active low RJ12 port [7], providing ground, right ascension +/- and declination +/- connection with the given and adjustable speeds of table 3.2 and 3.1 in both axes. The speeds are a multiple of the sidereal speed or the simple sidereal speed. This port is a standardised port and can be used in many astronomical devices. To control this port via the GPIOs of the Raspberry Pi, an additional electrical interface is necessary which fulfils the current and voltage conditions of the Raspberry Pi's GPIO ports (3,3V 16mA).

Table 3.1: RA+/RA- speeds

Setup Hand Control Unit	RA+	RA-
0.25	1.25	0.75
0.5	1.5	0.5
0.75	1.75	0.25
1	2	0

Table 3.2: DEC+/DEC- speeds

Setup Hand Control Unit	DEC+	DEC-
0.25	0.25	-0.25
0.5	0.5	-0.5
0.75	0.75	-0.75
1	1	-1

As seen in table 3.1, the mount is able to move in positive right ascension axis with twice of the sidereal speed, which effects in a simple sidereal speed, caused by the opposite speed of the speed of the earth. The declination axis as seen in table 3.2 gives positive and negative sidereal speeds. In theory and perfect aligned mount, only the right ascension axis has to be driven with the single sidereal speed [7]. Misaligned mounts have to be controlled in two axes.

3.3 Raspberry Pi and Periphery

To run the developed software and control the electrical interface, a Raspberry Pi 3 model B 64 bit, with the operating system ”Jessie”, is in use. To guarantee enough storage, the partition is expanded to the full available space of the memory card (16GB) right after the installation of the operating system. Two separate tests are done with the stock PiCamera and then with special astronomic camera. The PiCamera is connected via a camera adapter cable to the camera port of the Raspberry Pi. It is supported by standard drivers. This camera can be run in Python and openCV without any software adjustments. The situation is different with the astronomy camera. It comes without any native driver support for Python or openCV. This camera is connected via a USB 3.0 port.

3.4 Touch Displays

For easy controlling the program, two different displays in different sizes are tested. The first tests are done with a small 3.2” after market touch display connected via GPIO ports on the 40 pin connector of the Raspberry Pi. In the application the 3.2” display turns out to be too small and will be replaced by a 7” screen in the following tests.
The installation of the 3.2” display is done by

```
1  sudo nano /boot/config.txt
2  dtparam=spi=on
3  dtoverlay=waveshare32b:rotate=270
4  sudo nano /boot/cmdline.txt
5  fbcon=map:10
6  sudo nano /usr/share/X11/xorg.conf.d/99-calibration.conf
7  Section "InputClass" Identifier "calibration" MatchProduct "ADS7846
       Touchscreen" Option "Calibration" "160 3723 3896 181" Option "
       SwapAxes" "1" EndSection
8  sudo nano /usr/share/X11/xorg.conf.d/99-fbturbo.conf
9  Option "fbdev" "/dev/fb1"
10 cd /tmp wget http://www.joy-it.net/anleitungen/rpi/tft32b/waveshare32b-
       overlay.dtb
11 sudo cp waveshare32b-overlay.dtb /boot/overlays/
```

3.5 OpenCV

OpenCV is an open source program library of high performance computer vision algorithms for the purpose of machine vision and image processing which can be used under the BSD license [8]. Available is this library for many platforms such as Python,

Java, C++ and C and was introduced by Intel. However, the platform independence makes it necessary to compile the whole library for the desired platform, which is a time-consuming task [9].

3.5.1 Installation of OpenCV

The installation is done by the following commands as root in the console

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install build-essential cmake pkg-config
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev
libpng12-dev
sudo apt-get install libavcodec-dev libavformat-dev
libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install libgtk2.0-dev

sudo apt-get install libatlas-base-dev gfortran
sudo apt-get install python2.7-dev python3-dev

wget -O opencv_contrib.zip https://github.com/Itseez/opencv_contrib/
    archive/3.1.0.zip
unzip opencv_contrib.zip

wget https://bootstrap.pypa.io/get-pip.py
sudo python get-pip.py

cd ~/opencv-3.1.0/
mkdir build
cd build
cmake -D CMAKE_BUILD_TYPE=RELEASE \
-D CMAKE_INSTALL_PREFIX=/usr/local \
-D INSTALL_PYTHON_EXAMPLES=ON \
-D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib-3.1.0/modules \
-D BUILD_EXAMPLES=ON ..

make -j4
make clean
make

sudo make install
sudo ldconfig
```

After a successful installation, it is tested by importing openCV into a Python program:

```
1 import cv2 as cv
```

3.6 Usage of the PiCamera in Python

After connecting the camera via a special camera cable to the Raspberry Pi and activation in the Raspberry-Pi setup menu, it can be accessed in the console and in Python. The camera provides up to 1080p video streams at 30 frames per second. To use the camera in Python especially with openCV, a supported driver has to be loaded via the command

```
1 sudo modprobe bcm2835-v412
```

in the console or

```
1 os.system(sudo modprobe bcm2835-v412)
```

in the Python program. Otherwise the resulting errors are data type related and not easy to assign.

3.7 Usage of the Astro Camera ZWO Asi 120MM-S

This camera comes with a high sensitive CCD-sensor, which allows higher frame rates or shorter exposure times, depending on the desired application. The camera is connected via the USB 3.0 port of the Raspberry Pi. A native driver support for the operating system is given. The communication follows a manufacturer related communication protocol, which must be used. So the image data is read via the USB 3.0 interface and the proper communication protocol which allows live camera controls such as gain, exposure time and as well reading data like the sensor temperature. The camera manufacturer for stock provides some programs, to capture images and videos with a pre defined exposure time and gain value. To communicate with Python, a certain camera manufacturer software development kit (SDK) is necessary. The SDK provides the communication protocol within control commands for C/C++. To access the control commands, a wrapper is used to translate the existing C-library to a Python library and make it accessible.

In this case a camera of the manufacturer ZWO ASI is used. To install the C/C++ Python wrapper [9] for Python 3, the wrapper is available on PIP with the command *pip3 install zwoasi*. After the installation, the camera has to be initialized in the Python program with

```
1 import zwoasi as asi
2 asi.init("path")
```

where the path-variable is the path to the dynamic library of the ZWO ASI camera: *libASICamera.so*. The driver file is available on the manufacturer homepage. To control the camera, the following commands are mostly used [10]:

```
camera.set_control_value(asi.ASI_GAIN, CamGain)
camera.set_control_value(asi.ASI_EXPOSURE, 100)
camera.set_control_value(asi.ASI_WB_B, 99)
camera.set_control_value(asi.ASI_WB_R, 75)
camera.set_control_value(asi.ASI_GAMMA, 50)
camera.set_control_value(asi.ASI_BRIGHTNESS, 150)
camera.set_control_value(asi.ASI_FLIP, 0)
```

There are two possible ways to capture data from the camera:

- capture a single frame
- start video capture and grab frames

Capturing single images takes much longer and doesn't allow frame rates like in the video mode. For every single frame, the camera initializes with the belonging control data, captures an image and closes the capture process. The frame rate depends on the initializing speed of the camera. An advantage of that kind of use is the lower CPU load of the Raspberry Pi, which is about 1/5 of the video capture mode. The image is taken with the set control parameters and therefore the acquisition duration depends on the exposure time, the initialization time of the camera and the transfer time to the Raspberry Pi. Under bright conditions the total duration is mainly dependent on the initialization time, while under dark conditions and longer exposure times, it's mainly the exposure time, that determines the total cycle time.

```
camera = asi.Camera(0)
image = camera.capture()
```

The video capture mode starts a continuous streaming mode with the possibility of reading the current image data. It must be started in a different way and has to be closed after video stream acquisition. The possible frame rate mostly, depending on the exposure time, is much higher than in single capture mode, but it requires more CPU power, which is about 30 percent of the Raspberry Pi total power. It is possible to manipulate the camera control in every cycle before accessing the video frame without loss of time. This makes the camera video stream, within its ability to control, highly dynamic and customise able.

```
camera = asi.Camera(0)
camera.start_video_capture()
frame = camera.capture_video_frame()
```

```
4 camera.stop_video_capture()
```

3.8 Threading

Threading is the simultaneous execution of two or more independent processes [11], with the ability to communicate between them. In this case, it is used to combine time critical calculations, movement control of the mount, the GUI and reading image data from the guiding camera. The cycle time of the calculations and control commands within the GUI should be as short as possible to get a fast running program and obtain a smoothly working GUI without time lags caused by button inputs for operation control. While reading frames from the camera, the program must wait and blocks other processes in a non-threaded program. Thus the cycle time of the whole program depends on long image exposure times just on reading data from the camera, which expresses in spontaneous working user inputs, delayed calculations and video visualisation. In the threaded program, the camera reads its frames simultaneously to the calculations during the program cycle, to guarantee a cycle time depending on the calculations. This time is mostly correlating to the number of recognized objects and the associated computational cost. To process the image data resource-saving, a suitable exposure time and gain value must be applied, to avoid image noise and the accompanying false detection of objects.

3.9 GUI - PyGame

PyGame is an open source Python library based on SDL (simple directmedia layer) developed primarily for programming games in Python in a high level programming language. The advantage of PyGame is the easy to use library for graphical applications, keyboard and mouse inputs and the optimized performance even on SoC systems such as Raspberry Pi. The requirement in this case is the combination of different party libraries work together in one program: Image acquisition through threading considering the special camera communication protocol through a Python/C wrapper, openCV algorithms and a graphical user interface. This includes the conversion of the image format of the video stream between openCV and PyGame for the special requirements of PyGame [12].

The program has past the first tests with a simple GUI controlled by keyboard commands entirely written in Python and openCV. To control a program in this way, a short break waiting for keyboard interrupts is necessary, which reduces the speed of the whole program. In the PyGame GUI an exception handler interrupts the program for keyboard and mouse actions without any time delay, which increases the performance, especially in combination with a threaded video stream capturing. To control the program in the first tests, four keyboard inputs are used, with a single delay time of 30 ms, which gives a sum

of 120 ms and results in a cycle time greater than 120 ms. Compared to the camera exposure times in a non threaded program, the input delay time is in a similar or even higher order than the expsoure time of the camera. These time critical applications produce a haltingly process of calculations and user-input-possibilities. To optimise the cycle time, the PyGame library combined with a parallel image acquisition must be applied.

3.9.1 Installation of PyGame

The common installation process of the this widespread library is very easy and is done by entering *sudo apt − get install python − pygame* in the console. Also a installation for Python 3 via pip is available through *pip3 install pygame* for Python 3. After a successful installation, the PyGame library can be included through

```
1 import pygame
2 from pygame.locals import *
```

Chapter 4

Electrical Interface

As already mentioned, a special electrical interface is necessary to control the mount via the ST-4 interface. Different interfaces are tested, calculated and evaluated for an optimal solution regarding especially the integrability, required space, electrical safety for the mount interface, simplicity of use and robustness. The tested interfaces are

- relay circuit,
- transistor circuit,
- I2C interface,
- and optocoupler circuit.

4.1 Relay Circuit

In order to achieve a four port electrical control of the mount, a combination of four TinkerKit relays is tested. To operate the relays, an addition power source of 5 V is required, which can be taken from the Raspberry Pi GPIO 5 V power pin. This circuit guarantees a galvanic isolation [13] between the control interface and the mount RJ12 input connector, which satisfies the electrical safety. A great disadvantage regarding the mechanical parts of relays, is the inertia of the magnet contact. The mass respectively the inertia causes a minimal switching time. Also not to be neglected is the space used by the TinkerKit relays. The minimal switching time is especially unstable for small deviations, where particularly short pulses are needed. Too long pulses are leading to overshoots in the deviation control. Nevertheless, in the test setup it works very well for long exposures. Just for short exposures the overshoot was visible in the observer camera through a little haziness at long focal lengths. Not to be despised is also the noise of the switching relays [14].

4.1.1 Transistor Circuit

Also tested is a four channel NPN transistor BC547B circuit in common emitter setup. It is calculated with a transistor amplification, regarding to the data sheet, of $hfe = 300$ and a base current of 3 mA at 3,3 V to satisfy the GPIO conditions [13]. The series resistor of the transistor base is calculated by

$$R_{base} = \frac{V_{GPIO} - V_{Diode}}{I_{Base}} = \frac{3.3 - 0.7}{3.10^{-3}} = 866\,\Omega. \tag{4.1}$$

This gives a 866 Ω resistor, thus the next larger resistor with a value of $1k\,\Omega$ is selected. To

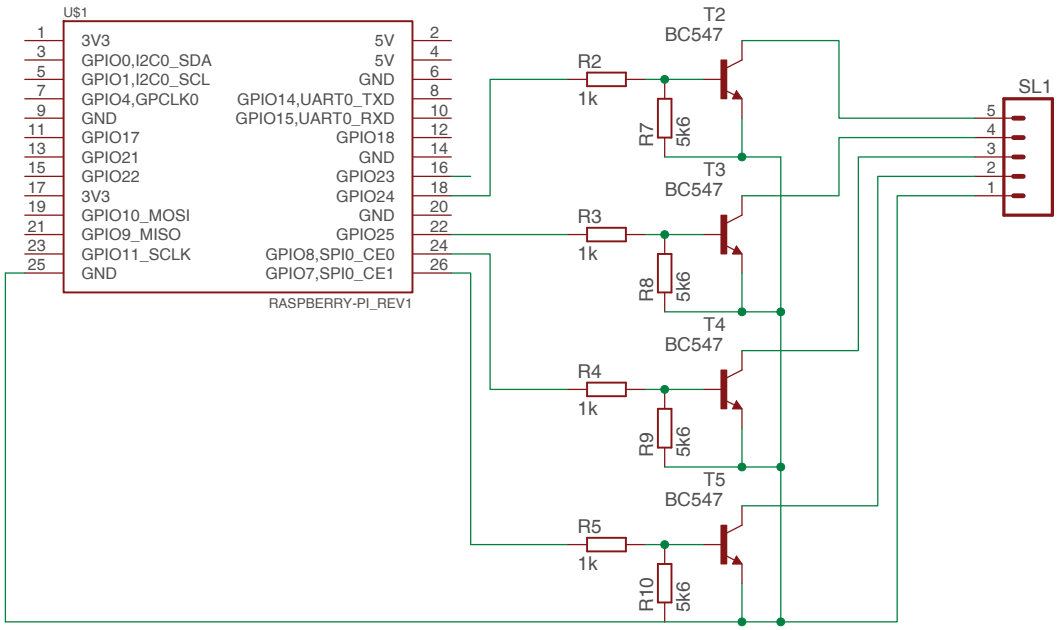

Figure 4.1: 4 Channel transistor circuit

reduce leakage current, an additional resistor parallel to the base-emitter route is applied, with a value of $5.6k\,\Omega$. One great advantage of that circuit is the fast switching time, the integrability into combined circuits and the less space of the four channel transistor circuit. The fast switching time reduces the overshoot of the control, depending on the deviation of the tracked object.

4.1.2 I2C Circuit

In this case an eight channel I2C I/O board I2C-0C8055 (Fig. 4.2) is used, which consists essentially of a I2C controller and a transistor driver (ULN2803). To control the I2C board, the Raspberry Pi has to be configured, thus the I2C function must be activated in the Raspberry Pi setup and initialised in the Python program.

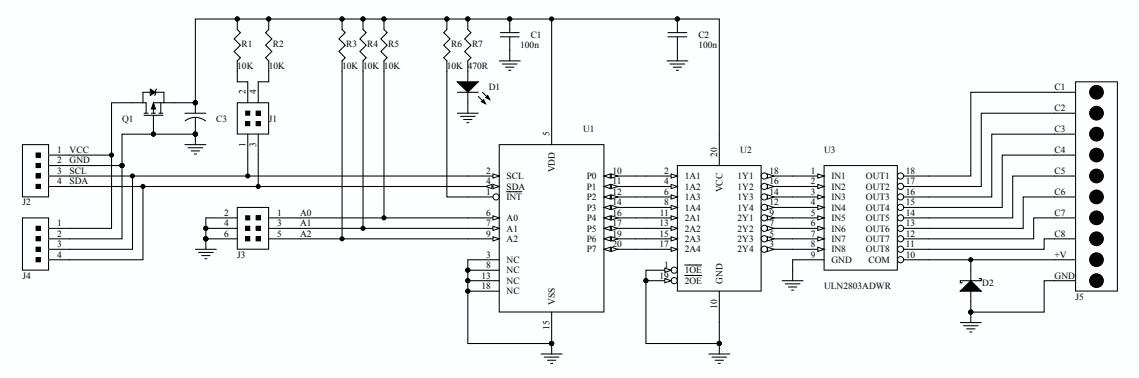

Figure 4.2: 8 channel I2C I/O board - datasheet

Initialisation in Python:

```
1 bus = smbus.SMBus(1)
2 address = 0x20 I2C device address I/O board
```

Addressing the ports of the I2C driver and controller in Python:

```
1 bus.write_byte(address,0xff) # all low
2 bus.write_byte(address,0xfd) #RA+
3 bus.write_byte(address,0xfe) #RA-
4 bus.write_byte(address,0xfb) #DEC+
5 bus.write_byte(address,0xf7) #DEC-
```

The interface is connected to the I2C ports of the GPIO ports of the Raspberry Pi as shown in table 4.1. Further no additional power source is required. The transistor driver of

Table 4.1: I2C Ports

I2C Interface Port	Raspberry Pi Port
SDA	GPIO2
SCL	GPIO3
GND	GND

this interface with its Darlington transistor arrays provides switching times similar to the standalone transistor solution, which is enough for the guiding pulses. The bottleneck of this solution is the timing of the I2C bus protocol. The addressing of the channels is done regarding to the data sheet via a translation between the binary addresses to hexadecimal addresses for Python. The binary addresses are shown in figure 4.3.

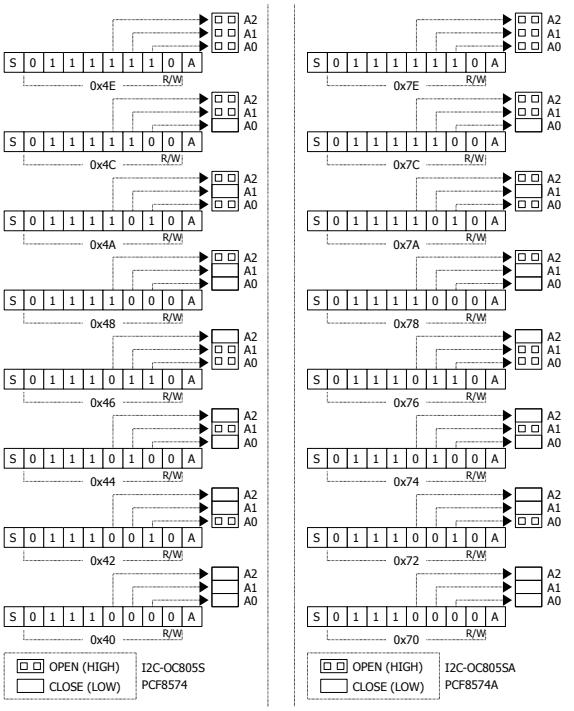

Figure 4.3: Addressing I2C - datasheet

4.2 Optocoupler Circuit

Optocouplers (Fig. 4.4) provide a galvanic isolation [13] between the mount and the control unit and fast switching times similar to the transistor circuit. Depending on the optocouplers, a previous transistor circuit is required, to not exceed the current specifications of the Raspberry Pi. This circumstance reduces the integrability compared to the standalone transistor circuit.

4.3 Summary

Regarding to the evaluation factors (integrability, required space, electrical safety for the mount interface, simplicity of use and robustness), the transistor circuit is selected for further work and the final PCB design.

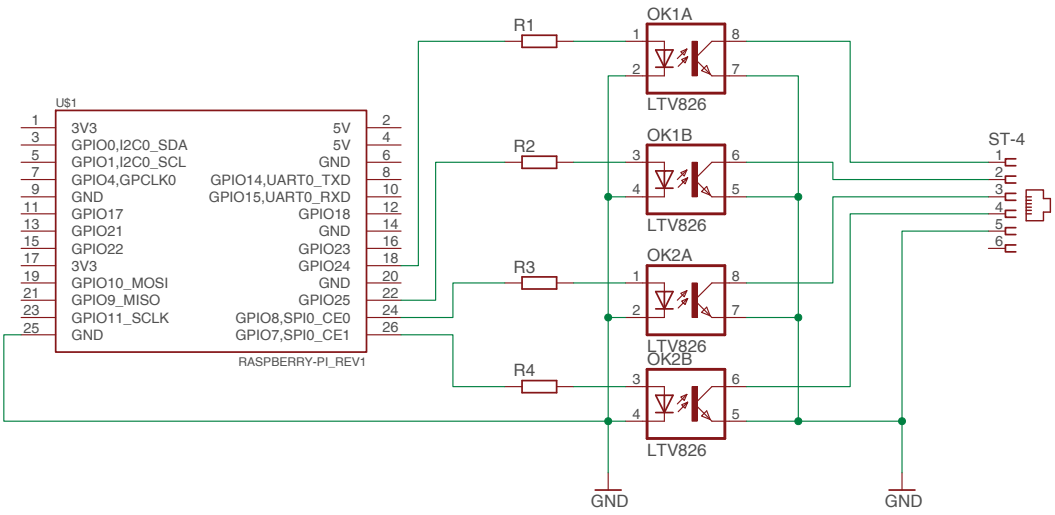

Figure 4.4: 4 channel optocoupler circuit

Chapter 5

PCB Design

5.1 Combined Circuit

To provide an all-in-one solution such as 12 V and 5 V power supply for the Raspberry Pi, the electrical control interface, the touch display and the mount itself, a combined circuit is designed. The requirement of different input voltages facilitates the practical use of the system offside the availability of electricity out of the socket with a power supply. Mostly 12 V car battery packs are in use, because of their capacity. Meanwhile the development of high capacity 5 V battery packs as an available and light-weight solution, concerning the weight of the battery, provides a good alternative to the mostly heavy-weight 12 V car batteries. The designed electrical circuit integrates all necessary inputs and outputs and additionally the power source of the system, matching the requirements. For the schematic design and the PCB layout, the free to use program DesignSpark PCB is used. With its integrated part finder of known companies like RS-Components, the availability of parts and footprints for lay outing are given.

5.2 DesignSpark PCB

As already mentioned, a special program with an easy import of parts including foot prints is used for the combined circuit. To provide a satisfying and robust connectivity for controlling the mount, input voltage and the Raspberry Pi connections are special connectors necessary. The circuit board is a double layered with a 40 pin connector on one side to attach the board to the Raspberry Pi. On the other side (layer) the other required parts are soldered, such as transistors, RJ12 connector and so on. The availability of these special parts and the provided foot prints have to be taken into account, which must be considered in the search of the parts. DesignSpark PCB with the additional library loader can import the parts with their footprints via a special online search engine available on [15]:

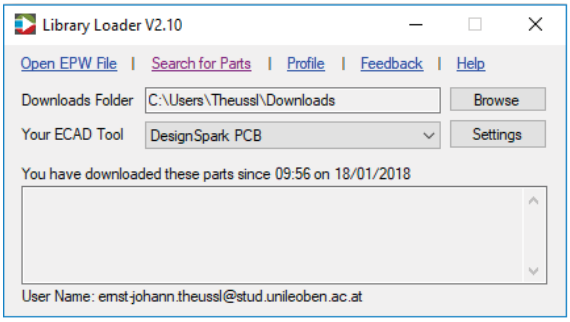

Figure 5.1: DesignSpark library loader

https://rs.componentsearchengine.com

5.3 Steps from Circuit to the finished product

Every design of circuit boards is separated into some steps: Calculate the circuit considering the requirements, a schematic drawing of the circuit and the final layout. In the chapter 4 possibilities for controlling the mount are calculated and tested. For the first tests, the different circuits are build on prototype circuit boards and compared to each other. On the basis of connectivity and available space, the prototype of a transistor based circuit is further used. The advantage of the transistors compared to relays is their switching time, which affects the accuracy control and the less space they need. Compared to the I2C solution, a more integrated circuit board is possible, without having extra boards, which must be attached by wires in a housing.

5.4 Schematic Design

After calculating and comparing the different circuits, the transistor variant is drawn in DesignSpark PCB (Fig. 5.2). To create a step by step solution, a new project in the program has to be created. The first step is to search for available parts, which are given in a parts list, made after the first prototypes via the RS-Components search engine. The individual components of the circuit are describes in table 5.1 and figure 5.2. The $12\,V$ and $5\,V$ supplies are given by existing DC boost up and down converters. They are attached by the connectors shown in figure 5.2 by number 7 and 8.

5.4.1 PCB Design - Board Layout

The next step is the placement of the components on the circuit board and give easy access to the input and output connectors (Fig. 5.3). The whole circuit board has equal dimensions as the Raspberry Pi, with a global 40 pin connector on the lower side, while all electrical parts are on the upper side to guarantee the best compactness. To realise this,

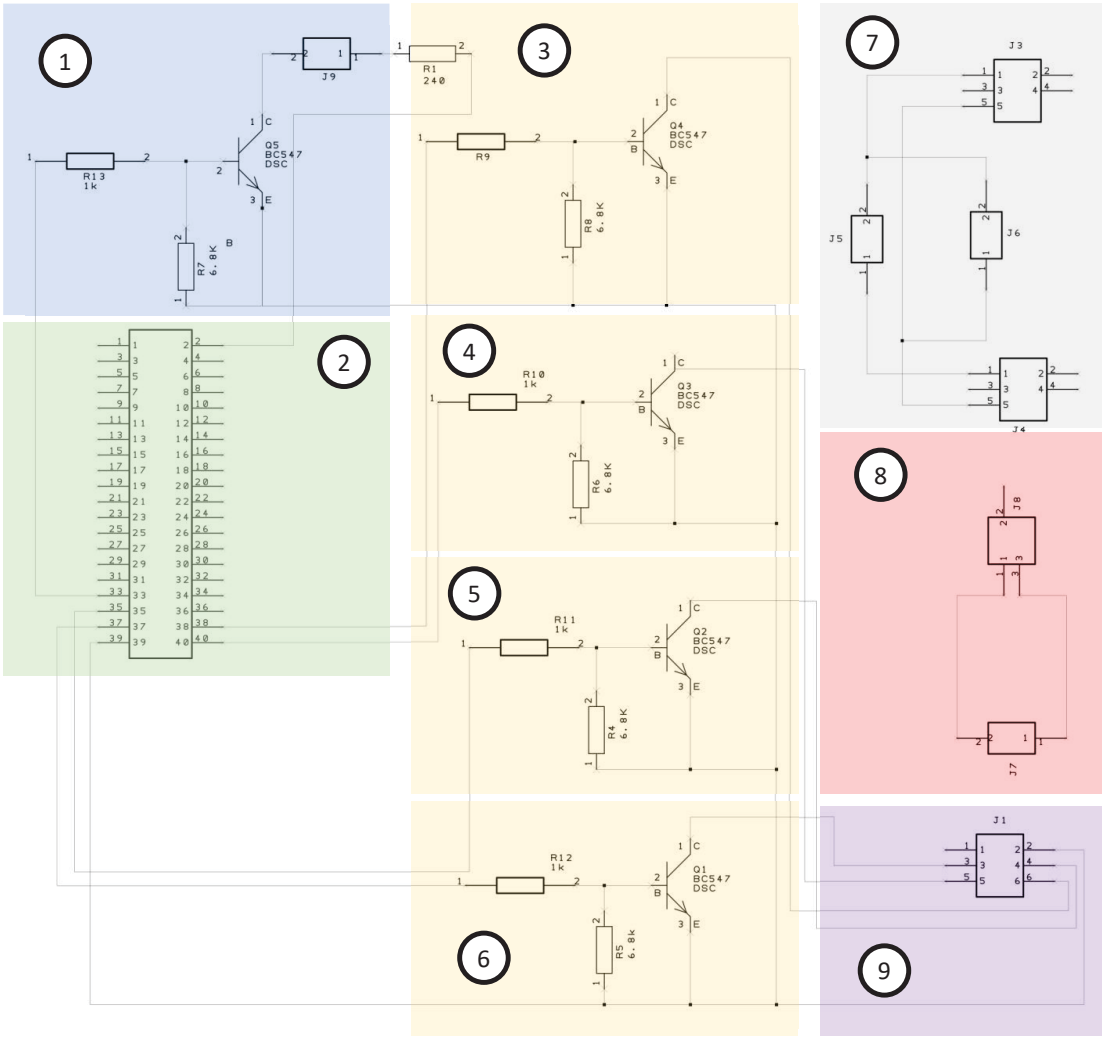

Figure 5.2: Schematic

a double layer board is necessary. In the PCB design the components can be switched between the layers and placed by drag and drop on each layer. The board wiring in DesignSpark PCB can be done with the 'auto-route all nets' function with additional conditions, such as used layers and number of vias. After a successful routing, the wiring can, if necessary, manipulated by hand for position and thickness. A 3D rendered image can now be viewed as a final check of the placed parts and wires of the circuit (Fig. 5.4). The 3D model includes all the footprints and available 3D data of each component.

Table 5.1: Connections

Number	Purpose	Raspberry Pi GPIO	40 Pin Connector Port
1	Status LED Driver	33	
2	40 Pin Connector	-	-
3	RA+ Driver	20	38
4	DEC+ Driver	21	40
5	RA- Driver	26	39
6	DEC- Driver	19	35
7	5V Input Voltage Connector	NC*	NC*
8	12V Input Voltage Connector	NC*	NC*
9	ST4 Output	-	-

Table 5.2: Parts list

Position	Pieces	Part Description	Order Number	Price p. Pcs.
1	1	Molex RJ12	95501-2661	0.73
2	1	Connector 40 Pin	681-6794	4.07
3	5	BC548B	761-9828	0.188
4	5	R 1k	125-1142	0.035
5	5	R 6k8	132-696	0.091
6	3	2 Pin Connector	4838461P	0.103
7	1	Micro USB Connector	8183361	1.53
8	1	DC Connector	8786787	0.954

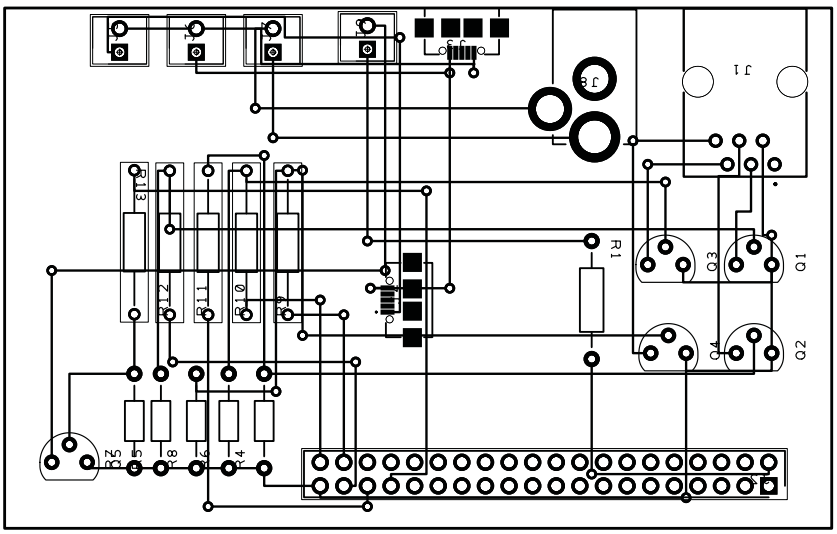

Figure 5.3: Layout

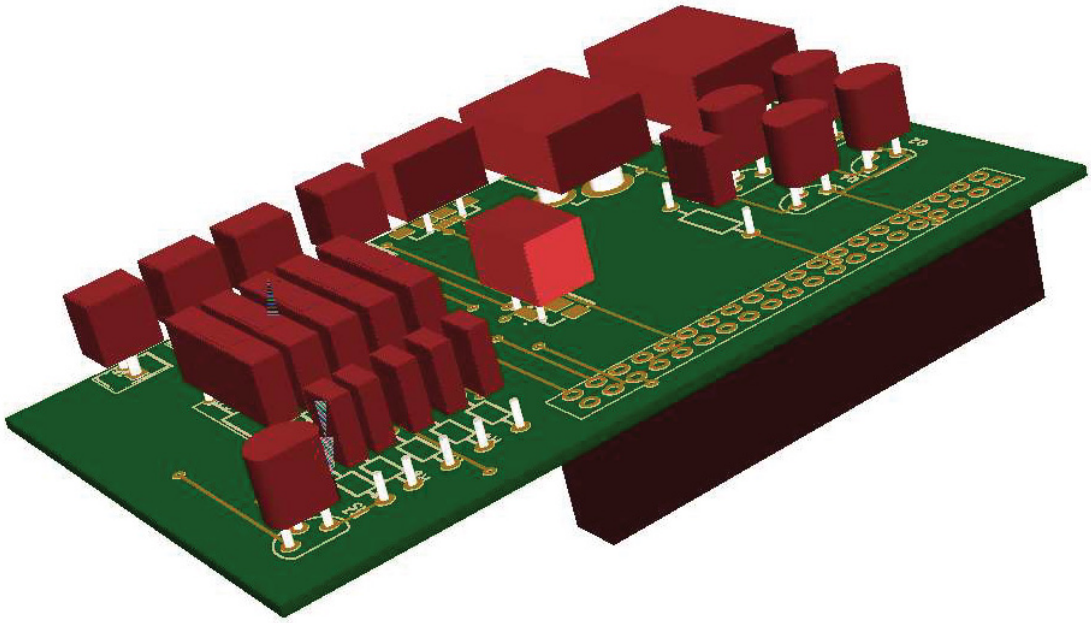

Figure 5.4: PCB design 3D rendering

Chapter 6

Kinematics

The relatively kinematic view requires the transformation of coordinate systems from the space system with the movement of heavenly bodies into the mount coordinate system and further into the image coordinate system to recalculate the misalignment of the mount in dependence of the star and observer position [16]. To this, the correction pulses for the mount control are applied [17].

6.1 Transformation: Telescope to space system

6.1.1 E1 to E0

Transformation from the local coordinate system Σ_1 on earth into space system Σ_0:
The fixed space coordinate system has the origin in the centre of the earth, while ϕ represents the geographical latitude and u the rotation of the earth about its axis.

$$B_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ r\cos(\phi)\cos(u) & \cos(\phi)\cos(u) & -\sin(u) & -\sin(\phi)\cos(u) \\ r\cos(\phi)\sin(u) & \cos(\phi)\sin(u) & \cos(u) & -\sin(\phi)\sin(u) \\ r\sin(\phi) & \sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \tag{6.1}$$

6.1.2 E2 to E1

Rotation about the vertical telescope mounting axis ($\Sigma_2 \rightarrow \Sigma_1$):
The vertical telescope axis goes through the zero point, the centre of the earth. α represents a rotation about the x axis of Σ_1, which is the first error parameter. If $\alpha = 0$, no error on the alignment occurred and the y-axis of the mount is parallel to the tangent of the circle of latitude. This angle results in a lefter or righter position of the equatorial

telescope axis compared to its nominal-position (Polaris.)

$$B_{12} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{6.2}$$

6.1.3 E3 to E2

Rotation about the horizontal telescope mount axis ($\Sigma_3 \rightarrow \Sigma_2$):
This rotation is the second error parameter of the alignment and is represented by β. If the angle $\beta = \phi$ no error at the alignment occurred. This angle results in a higher or lower position of the equatorial axis compared to its nominal-position (Polaris).

$$\mathsf{B}_{23} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\beta) & 0 & \sin(\beta) \\ 0 & 0 & 1 & 0 \\ 0 & -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{6.3}$$

6.1.4 E4 to E3

Rotation about the hour axis ($\Sigma_4 \rightarrow \Sigma_3$):
A rotation about the z-axis of the telescopes equatorial axis $-u$, the negative speed of the rotation of the earth. If $\alpha = 0$ and $\beta = \phi$, that angular speed compensates the rotation of the earth, no additional movement of the mount is necessary.

$$\mathsf{B}_{34} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(u) & 0 & \sin(u) \\ 0 & -\sin(u) & \cos(u) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

6.1.5 E5 to E4

Transformation of the telescope or camera coordinate system Σ_5 to the telescope mount coordinate system Σ_4. It is a shift of the telescope coordinate system on the z-axis of the mount system into the camera coordinate system.

$$\mathsf{B}_{45} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ a & 1 & 0 & 0 \\ b & 0 & 1 & 0 \\ c & 0 & 0 & 1 \end{bmatrix} \tag{6.5}$$

6.1.6 Summary: E5 to E0

The summarised transformation consists of the multiplication of each transformation

$$\mathsf{B}_{05} = \mathsf{B}_{01}\mathsf{B}_{12}\mathsf{B}_{23}\mathsf{B}_{34}\mathsf{B}_{45}. \tag{6.6}$$

A point x with homogeneous coordinates in the camera coordinate system $[1, x5, y5, z5]^T$, has the following transformed homogeneous space coordinates:

$$\begin{bmatrix} 1 \\ x_0 \\ y_0 \\ z_0 \end{bmatrix} = \mathsf{B}_{05} \begin{bmatrix} 1 \\ x_5 \\ y_5 \\ z_5 \end{bmatrix} \tag{6.7}$$

Distant points such as astronomical objects are represented as homogeneous coordinates as $[0, x, y, z]^T$.The matrix $\mathsf{B_ij}$ has the form

$$\mathsf{B}_{ij} = \begin{bmatrix} 1 & \mathsf{o}^{\mathsf{T}} \\ \mathsf{d}_{ij} & \mathsf{A}_{ij} \end{bmatrix} \tag{6.8}$$

where A_{ij} is an orthogonal $3x3$ matrix (rotation matrix), d_{ij} the translation part with 3 components and o^{T} a null vector. So the transformation of a distant point of the camera coordinate system to the space coordinate system is done with

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} = \mathsf{A}_{05} \begin{bmatrix} x_5 \\ y_5 \\ z_5 \end{bmatrix} \tag{6.9}$$

and

$$\mathsf{A}_{05} = \mathsf{A}_{01}\mathsf{A}_{12}\mathsf{A}_{23}\mathsf{A}_{34}\mathsf{A}_{45} \tag{6.10}$$

so the result of A_{05} is

$$\mathsf{A}_{05} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{6.11}$$

with

$$\begin{aligned} a_{11} = {} & (\cos(\alpha)\sin(\beta)\sin(\phi) + \cos(\beta)\cos(\phi) - \cos(\alpha))\cos^2(u) \\ & + \sin(\alpha)(\sin(\phi) - \sin(\beta))\cos(u)\sin(u) + \cos(\alpha) \end{aligned} \tag{6.12}$$

$$\begin{aligned} a_{12} = {} & \sin(\alpha)(\sin(\beta) - \sin(\phi))\cos^2(u) \\ & + (\cos(\alpha)\sin(\beta)\sin(\phi) + \cos(\beta)\cos(\phi) \\ & - \cos(\alpha))\cos(u)\sin(u) - \sin(\alpha)\sin(\beta) \end{aligned} \tag{6.13}$$

$$a_{13} = (\sin(\beta)\cos(\phi) - \cos(\alpha)\cos(\beta)\sin(\phi))\cos(u) + \sin(\alpha)\cos(\beta)\sin(u) \tag{6.14}$$

$$\begin{aligned} a_{21} = {} & \sin(\alpha)(\sin(\beta) - \sin(\phi))\cos^2(u) \\ & + (\cos(\alpha)\sin(\beta)\sin(\phi) + \cos(\beta)\cos(\phi) \\ & - \cos(\alpha))\cos(u)\sin(u) + \sin(\alpha)\sin(\phi) \end{aligned} \tag{6.15}$$

$$\begin{aligned} a_{22} = {} & (\cos(\alpha)\sin(\beta)\sin(\phi) + \cos(\beta)\cos(\phi))\sin^2(u) \\ & + \cos(\alpha)\cos^2(u) + \sin(\alpha)(\sin(\beta) - \sin(\phi))\cos(u)\sin(u) \end{aligned} \tag{6.16}$$

$$a_{23} = -\sin(\alpha)\cos(\beta)\cos(u) - (\cos(\alpha)\cos(\beta)\sin(\phi) - \sin(\beta)\cos(\phi))\sin(u) \tag{6.17}$$

$$a_{31} = -(\cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi))\cos(u) - \sin(\alpha)\cos(\phi)\sin(u) \tag{6.18}$$

$$a_{32} = \sin(\alpha)\cos(\phi)\cos(u) - (\cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi))\sin(u) \tag{6.19}$$

$$a_{33} = \cos(\alpha)\cos(\beta)\cos(\phi) + \sin(\beta)\sin(\phi) \tag{6.20}$$

6.2 Transformation of Far Away Points

To calculate back from distant points in the space system to the camera coordinate system, A_{05} must be inverted

$$\mathsf{A}_{50} = A_{05}^T = \begin{bmatrix} a_{11} & a_{21} & a_{31} \\ a_{12} & a_{22} & a_{32} \\ a_{13} & a_{23} & a_{33} \end{bmatrix} \tag{6.21}$$

. In case $\alpha = 0$ and $\beta = \phi$, $\mathsf{A}_{05} = \mathsf{A}_{50} = \mathsf{I}$, a unit matrix is the result. The inverse transformation gives

$$\begin{bmatrix} x_5 \\ y_5 \\ z_5 \end{bmatrix} = \mathsf{A}_{50} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \tag{6.22}$$

which means, that x_0, y_0, z_0 are coordinates of a distant point in terms of the space system. The equation is the parametrisation of the orbit of the object in terms of the camera coordinate system. The parametrisation follows

$$\begin{bmatrix} x_5 \\ y_5 \\ z_5 \end{bmatrix} = \begin{bmatrix} x_5(u) \\ y_5(u) \\ z_5(u) \end{bmatrix} \tag{6.23}$$

As example, the coordinates of Polaris are approximately $x_0 = y_0 = 0$ and $z_0 = 1$

$$\begin{bmatrix} x_5 \\ y_5 \\ z_5 \end{bmatrix} = \begin{bmatrix} a_{31} \\ a_{32} \\ a_{33} \end{bmatrix} = \begin{bmatrix} -(\cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi))\cos(u) - \sin(\alpha)\cos(\phi)\sin(u) \\ \sin(\alpha)\cos(\phi)\cos(u) - (\cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi))\sin(u) \\ \cos(\alpha)\cos(\beta)\cos(\phi) + \sin(\beta)\sin(\phi) \end{bmatrix} \tag{6.24}$$

$$\begin{aligned} &= \begin{bmatrix} 0 \\ 0 \\ \cos(\alpha)\cos(\beta)\cos(\phi) + \sin(\beta)\sin(\phi) \end{bmatrix} \\ &+ \cos(u) \begin{bmatrix} -(\cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi)) \\ 0 \\ 0 \end{bmatrix} \\ &+ sin(u) \begin{bmatrix} -\sin(\alpha)\cos(\phi) \\ -(\cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi)) \\ 0 \end{bmatrix} \end{aligned} \tag{6.25}$$

6.2.1 The Central Projection of the Orbit

The image plane is parallel to the x_5, y_5 plane, with a distance of $z_5 = f$, where f is the focal length of the optics. The projection resulting from the telescope focus gives

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -f \cdot \dfrac{x_5}{z_5} \\ -f \cdot \dfrac{y_5}{z_5} \end{bmatrix}. \tag{6.26}$$

The central projection of the distant orbit of Polaris results into a circle centred at $\boldsymbol{O}$:

$$\begin{aligned} &\begin{bmatrix} x(u) \\ y(u) \end{bmatrix} = \\ &\frac{f\cos(u)}{\cos(\alpha)\cos(\beta)\cos(\phi) + \sin(\beta)\sin(\phi)} \begin{bmatrix} \cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi) \\ -\sin(\alpha)\cos(\phi) \end{bmatrix} \\ &+ \frac{f\sin(u)}{\cos(\alpha)\cos(\beta)\cos(\phi) + \sin(\beta)\sin(\phi)} \begin{bmatrix} \sin(\alpha)\cos(\phi) \\ \cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi) \end{bmatrix}. \end{aligned} \tag{6.27}$$

At long exposures this circle is visible with its resulting radius on the image, which can be determined by

$$\begin{aligned} r^2 = &\frac{f^2}{(\cos(\alpha)\cos(\beta)\cos(\phi) + \sin(\beta)\sin(\phi))^2} \\ &\cdot \left[(\cos(\alpha)\sin(\beta)\cos(\phi) - \cos(\beta)\sin(\phi))^2 + \sin^2(\alpha)\cos^2(\phi) \right] \end{aligned} \tag{6.28}$$

If this radius is known through an evaluation of observed objects due long time image acquisition, with the knowledge of their coordinates in the space system, it is possible to calculate the misalignment, respectively the deviation angles α and β and further the two-axis control pulses.

6.2.2 Reduced Kinematics

Further a reduced kinematics is derived of the exact solution, to provide a generic applicable two-axis object tracking system through reverse kinematics. For that reason, the movement of an object in the observer coordinate system (two-dimensional image) is compared to the movement of the axes of the kinematics. In this process the position of the object is stored into a database, distinguished by right ascension and declination axis. In the first step, the right ascension axis is moved for a defined time and number of data points, the same goes on for the other axis. The stored data is linearly fitted and differentiated by the axis to calculate the slope of each function. This slope is the rotation angle α of the axis of the kinematics and camera related coordinate system (camera position and kinematics behaviour). The calculation of the slope of the second axis gives a second slope β, which should be $\alpha + 90$ measured in degrees. Due mechanical inaccuracies the real slope is different to this assumption, but will be neglected in the next steps because just small deviations are considered [18].
With

$$rot_{angle} = \alpha \tag{6.29}$$

and assuming an orthogonal coordinate system

$$\beta = \alpha + 90 \tag{6.30}$$

the rotation matrix about the z-axis in homogeneous coordinates [19] is represented by

$$\mathsf{D} = \begin{bmatrix} cos(\alpha) & -\sin(\alpha) & 0 \\ sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{6.31}$$

considering the additional shift T of the coordinate system

$$\mathsf{T} = \begin{bmatrix} 1 & 0 & x_{ref} \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix} \tag{6.32}$$

with x_{ref} and y_{ref} are the coordinates of the tracked object in the camera coordinate system

$$\boldsymbol{O} = \begin{bmatrix} x_{ref} \\ y_{ref} \\ 1 \end{bmatrix}, \tag{6.33}$$

which represents the origin of the transformed coordinate system. The full transformation of the coordinate system for the reverse kinematics is:

$$\Sigma_2 \rightarrow \Sigma_1 = \mathsf{DT} = \begin{bmatrix} cos(\alpha) & -\sin(\alpha) & 0 \\ sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & x_{ref} \\ 0 & 1 & y_{ref} \\ 0 & 0 & 1 \end{bmatrix}, \tag{6.34}$$

which gives the transformed object coordinates $\boldsymbol{O'}$ for any coordinate $\boldsymbol{R}$ (x,y)

$$\boldsymbol{R} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{6.35}$$

$$\boldsymbol{O'}^T = \boldsymbol{R}^T \mathsf{DT} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & -(x_{ref}\cos(\alpha) - x_{ref}\sin(\alpha)) \\ \sin(\alpha) & \cos(\alpha) & -(x_{ref}\sin(\alpha) + y_{ref}\cos(\alpha)) \\ 0 & 0 & 1 \end{bmatrix} \tag{6.36}$$

$$\boldsymbol{O'} = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}. \tag{6.37}$$

Any deviation measured in the image data is the transformed deviation in the new x' and y' coordinate system, which now can control the axes of the kinematics. The orthogonality error is calculated in degrees as

$$ortho_{error} = 90 - |\alpha| - |\beta|. \tag{6.38}$$

Anticipated, this error only effects in positions distant from the coordinate system origin. Because objects are approximated to the axis deviations, the error is decreasing in order to the distance to the origin. In tests, even large deviations compared to the image resolution were controllable. In case

$$\boldsymbol{R} = \boldsymbol{O}, \boldsymbol{O'} = \boldsymbol{O}^T \mathsf{DT} = 0 \tag{6.39}$$

the transformation of the point $\boldsymbol{R}$ is equal to $\mathbf{0}$, which means, that no deviation occurs.

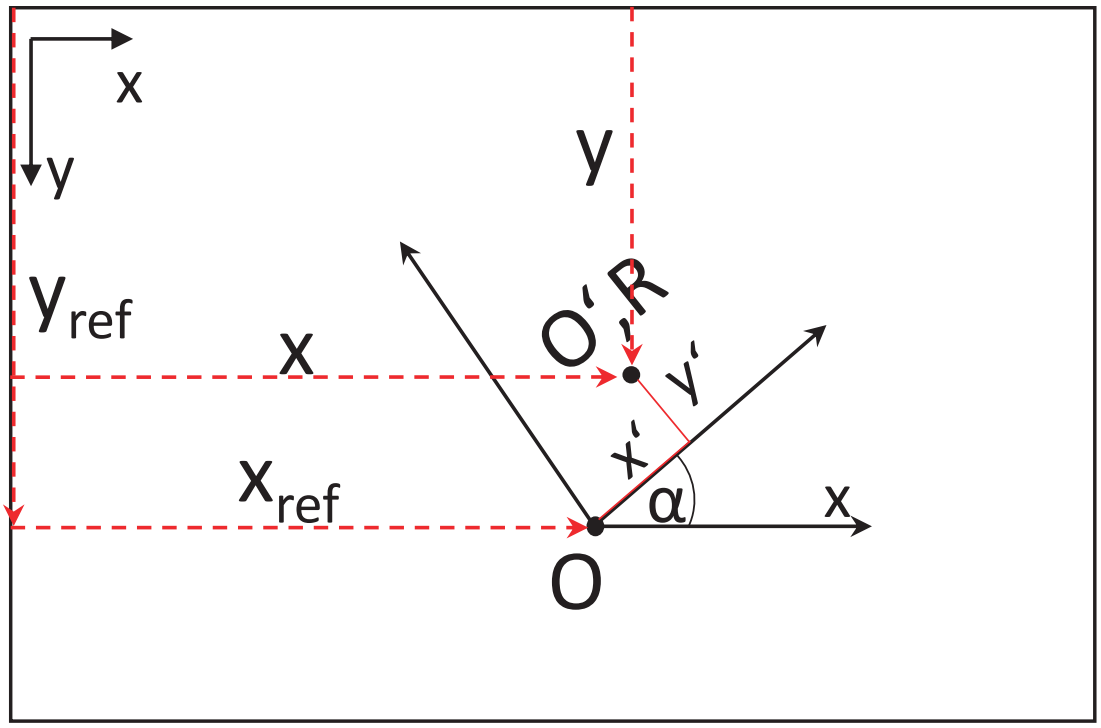

Figure 6.1: Reduced kinematics

Chapter 7

Optics

Regarding to calculations, the aperture of the optical system is responsible for the light, which can be captured by the optics and further by the CCD-sensor. The aperture is given as the quotient of the opening pupil (effective lens diameter) and the focal length. The opening pupil gives the size of the optical system and the ability to gather image information, while a larger focal length limits the field of view. The focal length in combination with the CCD-sensor size and resolution physically defines the tracking accuracy with its movement per pixel. Depending on the application, the distance to the tracked object and its speed, the optical system has to be chosen. In this case a small refractor lens with a focal length of 180mm f/3.3 is used, which provides enough light especially under low light conditions. The refractor also supports a T2 and a 1,25” connection for direct camera usage.

7.1 Camera

The camera and its sensor with its ability to control the camera settings is an essential part of the high accuracy tracking system. Typical parameters which has to be controlled are the exposure time respectively the shutter speed, gain, resolution, bandwidth, and so on.

7.2 Telescopes

For observation of distant objects, the most suitable telescope must be selected by the focal length, which gives the magnification and the aperture, determined through the lens diameter and focal length. There are many different design types of telescopes and each has advantages and disadvantages, such as image quality, diameter, weight, optical resolution and price.

7.2.1 Newton Telescopes

A newton telescope consists of a concave main mirror, which magnifies the image and projects it on a 45 degree fang mirror to the eyepiece or camera. The advantage of that kind of telescopes is the light weight main mirror and the simple design. The focus of the image is set by the distance of the eyepiece to the fang mirror and the main mirror. Due thermal influence on the material of the tubus, this distance increase or decrease depending on the temperature. This makes adjustments via a special adjustment laser necessary, which is a reasonable disadvantage of this design [4].

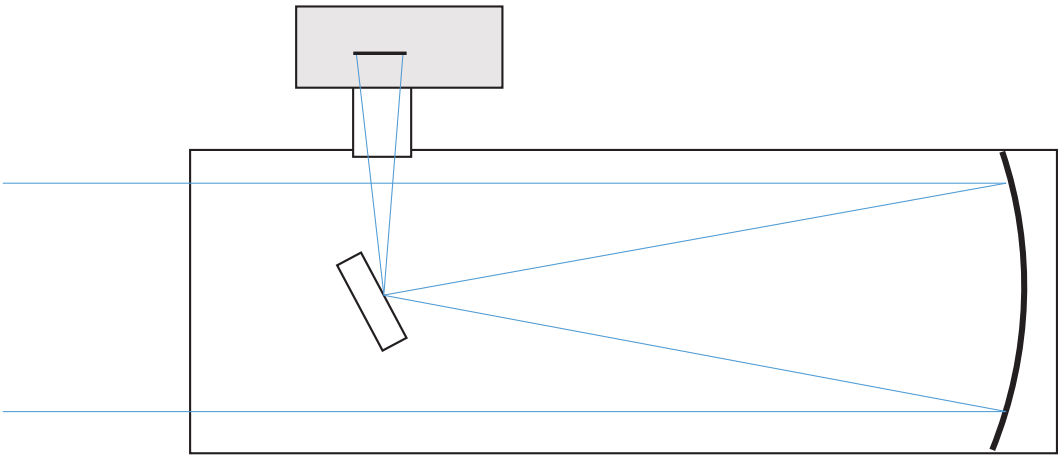

Figure 7.1: Newton reflector telescope

7.2.2 Refracting Telescopes

Refracting telescopes magnifies the image through a set of lenses in the tubus, thus the number of lenses and design gives the quality criteria of the whole telescope together with the lens materials. Lenses tend to colour errors and aberrations on their edges, therefore these errors have to be corrected through additional lenses or materials, like in achromatic refractor telescopes. These are designed with at least two or more corrected lenses. Further this telescope design doesn't need any type of adjustment at all, just a homogeneous temperature of the tubus and lenses, which can be achieved by a appropriate cooling or heating time under the needed external conditions [4].

7.2.3 Detectable movement of tracked objects

The least detectable movement of tracked objects results in a combination of the image sensor resolution of the observing camera and the focal length of the telescope with its

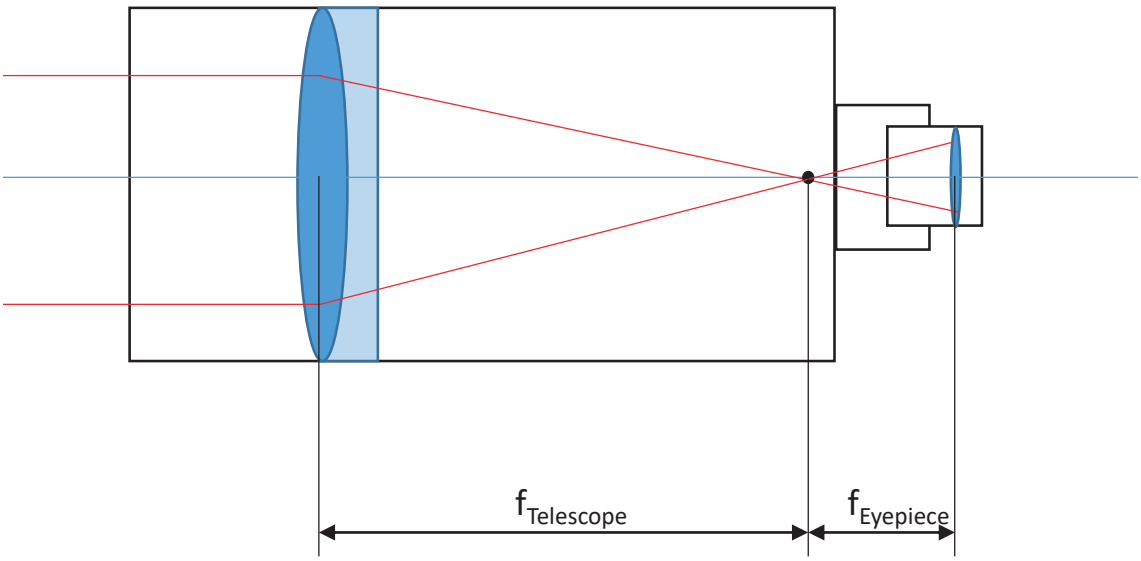

Figure 7.2: Achromatic refractor telescope

optical resolution. This factor can be expressed through the field of view equation [20]:

$$\alpha_v = 2 \cdot \arctan\left(\frac{d}{2 \cdot f}\right), \tag{7.1}$$

combined with the sensor width and height resolution

$$\alpha_v = 2 \cdot \arctan\left(\frac{w_{mm}}{2 \cdot f}\right), \tag{7.2}$$

and

$$\alpha_v = 2 \cdot \arctan\left(\frac{h_{mm}}{2 \cdot f}\right), \tag{7.3}$$

which gives the resolution of arc seconds per sensor width or height through

$$1\,rad = \frac{360^\circ}{2\pi} = 57,295^\circ, \tag{7.4}$$

$$1\,deg = \frac{57,295^\circ}{1\,rad} \cdot 60s = 57,295 \cdot 60\,s = 3438\,\text{arcsec}, \tag{7.5}$$

and leads to

$$b_{arcmin} = \frac{w_{mm} \cdot 3438}{f}, \tag{7.6}$$

and

$$h_{arcmin} = \frac{h_{mm} \cdot 3438}{f}. \tag{7.7}$$

Applied on the sensor resolution

$$h_{arcmin} = \frac{h_{mm}}{res_h}, \tag{7.8}$$

$$w_{arcmin} = \frac{w_{mm}}{res_w}. \tag{7.9}$$

The Dawes criteria describes the optical resolution of the optics, in dependence of the of the wavelength

$$\alpha = 1,02\frac{\lambda}{d} \tag{7.10}$$

with,
d ... lens diameter in mm
λ ... wavelength in mm
is neglected. Because the combination of sensor and optics determines the physical limit of the distinctness of observed objects.

7.2.4 Crop Factor and Aperture

The crop factor indicates the visible area of an image in dependence of the camera sensor and focal length of the telescope, which gives further the effective aperture of the sensor and optics combination. The crop factor is calculated as the quotient of sensor width of a full frame sensor and the used sensor width [21]

$$c = \frac{width_{ff}}{width_{crop}}, \tag{7.11}$$

$$k_{eff} = \frac{f \cdot c}{D}. \tag{7.12}$$

with,
k_{eff} ... effective aperture
c ... crop factor
D ... opening

This aperture determines the depth of field in photography and also the light collecting capability. Thus the objects are almost infinitely far away, the depth of field has no influence on the acquired image, but the light collecting capability does. The darker an object appears, the longer must be the exposure time, to gather the desired image data. This time decreases with a bigger light collecting capability, which is represented by a smaller k-value.

Chapter 8

Software Developement

After the requirements of the test setup are met and kinematics are calculated, the prerequisites for programming are given. This includes hardware requirements like the electrical interface, necessary software and driver installations and the calculated kinematics and algorithms for multiple non unique object tracking.

8.1 Program Overview

The software, written in Python, can be divided into the main parts

- image acquisition,
- image processing,
- calculations,
- controlling,
- presentation.

The aim of this software is to hold tracked clearly distinguishable moving objects on an exact position of the image and control a two-axis mount through calculated reverse kinematics. The difficulty of this project is the identification of multiple non unique objects in a high speed video streams. The faster the shutter speed in correlation to the object movement, the smaller the deviation of the object between video frame n and frame n+1 is. Once again, the advantage of a fast software and shutter speed is shown. To track the non unique feature objects, their centre points are calculated through computer vision algorithms and saved into a temporary object database. After the next frame n+1 is read, the centre points of frame n and n+1 are compared with a Gaussian distribution to determine the most suitable positions of the objects in frame n+1. The result is saved in a persistent object database with a consistent numbering of every object. The database also

includes the object number, the actual coordinates and an online indicator for each object. If an object leaves the field of view, the online indicator changes to 0 (= offline) and the coordinates will keep the last coordinates of the seen object. New objects, travelling into the image, won't be saved into the object database to keep the focus on the existing and desired objects.

The image acquisition part reads in the actual video data in a parallel program through a separate thread. In this project a special high sensitive astronomy camera is used to minimise the shutter speed and optimise the correlation between its speed and the tracked object movement. The data is read with a SDK (software development kit) written in the programming language C. To use it in Python, a wrapper for the SDK is required. With this library, all essential parameters such as shutter speed respectively the exposure time, image gain and white balance of the camera can be controlled live in the program. The image or video processing part is doing all the required calculation depending on the given pixel values. It starts with a dynamic threshold as the first step in segmentation, to reduce the data information for further processing.

In the calculation part the relative camera and kinematics behaviour is measured through a calibration method, which gives a relative coordinate system. This new rotated and shifted coordinate system is the base for all deviation measurements in the tracking process. The calibration uses a initial rotation of the kinematic axes to save the combined behaviour of camera orientation and kinematics to recalculate a reverse kinematics in order to control the mount.

To control the mount, the calculated relative coordinate system is set to the desired initial object position, which consists of a shift in x,y and a rotation about a calculated angle. The x,y values of the coordinate shift are the x,y coordinates of the desired object read out of the object database at the time of initialisation. If a deviation occurs in the transformed coordinate system, the values of the object are simultaneously the deviation values for controlling the closed loop system.

The presentation of the video stream, input buttons, camera parameters, coordinate information and camera controls are shown in an entire window, generated in PyGame.

8.2 Overview Software Functions

Table 8.1 gives an overview of essential functions of this program, which will be explained in more detail in this chapter. These functions are used in the first tests of the program, without a PyGame-GUI and also later, without any change.

Table 8.1: Function Overview

Function	Explanation
Initialize()	*Initialise GPIOs, load camera driver*
getCoordinates()	*calculate actual coordinates*
getInitialStarTable()	*read initial star database*
StarMatching(StarTable)	*apply the magic matching*
getsSlope()	*get slope of relative coordinate system*
RACalibration(StarTable, TrackedStar)	*right ascension calibration*
DECCalibration(StarTable, TrackedStar)	*declination calibration*
PrintLog()	*show calibration data points*
CoordinatesTransformation (StarTable,TrackedStar)	*transform coordinate system*
TrackingMarkers(StarTable, TrackedStar)	*highlight tracked object*
Tracking(StarTableTrans)	*track object*

8.3 Applied Computer Vision

8.3.1 Read Video Stream Introduction

In this thesis, two relevant types of cameras are used. The first test starts with the Raspberry-Pi camera, a out of the box solution for an easy beginning. The limits of this camera where quickly seen under low light conditions and the ability of adjusting parameters live due processing. For this reason, the implementation of an astronomy camera is necessary. This camera has a more light sensitive image sensor and live adjustable camera parameters via a SDK. Thus it is possible to detect darker objects and reduce the exposure time, which is an improvement factor in accurate tracking.

8.3.2 Read Video Stream: Pi Camera

After the Pi-Camera is connected to the Raspberry Pi, it can be accessed via the command line prompt and also in Python. To read images [22] in a openCV understandable format, the existing camera driver must be overwritten by the command in the command prompt:

```
1 sudo modprobe bcm2835-v412
```

after are loaded drivers loaded and the camera can be initialized

```
1 cap = cv2.VideoCapture(0)
```

where 0 is the hardware device number of the camera. After the successful initialisation, a video stream can be started through

```
ret,frame = cap.read()
```

The required image data is saved to the variable *frame* and is ready for further image processing. The number of frames per second in this case is depending on the cycle time and the shutter speed of the PiCamera.

8.3.3 Read Video Stream: ZWO Asi 120MM-S

In a similar way, but with many more requirements (chapter 3), the ZWO Asi camera can now be used in Python through:

```
camera = asi.Camera(0)
camera.start_video_capture()
frame = camera.capture_video_frame()
camera.stop_video_capture()
```

8.3.4 Converting the Video Stream

Again, the image data is saved into the variable *frame* in the same way and can be further processed as in the Pi Camera. The only difference is the data format of the image. The Pi Camera gathers an RGB image, which needs a three-dimensional data array while the ZWO Asi captures a grey level image, which is saved in a 2-dimensional array. At a certain point of the program, the same data format must be respected to fulfil the requirements of the applied algorithms. For this reason, the three-dimensional array must be converted to a two-dimensional grey level image.

```
gray = cv2.cvtColor(frame,cv2.COLOR_BGR2GRAY)
```

After that conversion, a threshold algorithm is applied to filter the image for bright elements (segmentation), which stand out from the dark background. A simple dynamic threshold value *Threshold*, which is 80 percent of the maximum pixel value of the image is used.

```
Threshold = frame.max()*0.8
ret, thresh = cv2.threshold(gray,Threshold,255,0)
```

8.3.5 Feature Detection

After 'thresholding' the image, bright objects are represented by bright pixels (value=255), while the background is black-coloured (value=0), which are good conditions to apply an edge or contour detection algorithm. The existing algorithm *findContours* provided by

the openCV library calculates all boundary pixels of the labelled objects and saves it into an array:

```
image, cnts, hierarchy =
cv2.findContours(thresh,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
```

For further processing, the given data by the contours algorithm is used to determine the moments of each recognised object:

```
for c in cnts:
 M = cv2.moments(c)
  if(M['m10']!=0):
   cX = int(M['m10']/M['m00'])
   cY = int(M['m01']/M['m00'])
    TempCoordinates = np.append(TempCoordinates,np.array([[cX,cY]])
    ,axis=0)
```

The result is an array of found moments of the visible objects in the image:

$$C = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_i & y_i \end{pmatrix} \tag{8.1}$$

8.3.6 Unique Labelling of Moving Objects

In case an object disappears or a new one is added and processed till the coordinate calculation, the position of the moments and their labels are changed in the moment matrix, which causes identification problems of single objects. This circumstance makes an supplementary calculation necessary to make every object clearly identifiable. For this reason the moments of frame n are saved temporary to a variable and compared to the next frame $n+1$ through a matching algorithm. This algorithm calculates all possible distances between the recognised objects, under consideration, that the dimension of each moment matrix matches the other. In case, the dimension doesn't match, additional calculations must be done. The aim of the matching algorithm is to make every object from a certain time, which is given by an user input, clearly identifiable and provide a persistent object database with a unique sorting. That means, that new objects won't be visible in the database, while disappeared objected are marked as offline (Table 8.2).

So the matrix of moments of objects of frame n with its dimensions [i x 2] is given by

Table 8.2: Object database

StarTable			
Object Number	***x-Coordinates***	***y-Coordinates***	***online indicator***
1	x_1	y_1	0 or 1
2	x_2	y_2	0or 1
$\vdots$	$\vdots$	$\vdots$	$\vdots$
n	x_n	y_n	0 or 1

$$\mathsf{C}_n = \begin{pmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_i & y_i \end{pmatrix}. \tag{8.2}$$

And the matrix of moments of objects of frame $n+1$ with its dimensions [j x 2] by

$$\mathsf{C}_{n+1} = \begin{pmatrix} \hat{x}_1 & \hat{y}_1 \\ \hat{x}_2 & \hat{y}_2 \\ \vdots & \vdots \\ \hat{x}_j & \hat{y}_j \end{pmatrix}. \tag{8.3}$$

Regarding these two matrices of moments, where i and j are the number of objects, the distances between all objects of frame n and $n+1$ are calculated with equation 8.4 and summarised in a distance matrix, given by equation 8.5.

$$r_{ij} = \sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2} \tag{8.4}$$

$$\mathsf{R} = \begin{pmatrix} \sqrt{(x_1-\hat{x}_1)^2+(y_1-\hat{y}_1)^2} & \sqrt{(x_1-\hat{x}_2)^2+(y_1-\hat{y}_2)^2} & \dots & \sqrt{(x_1-\hat{x}_i)^2+(y_1-\hat{y}_i)^2} \\ \sqrt{(x_2-\hat{x}_1)^2+(y_2-\hat{y}_1)^2} & \dots & \dots & \sqrt{(x_2-\hat{x}_i)^2+(y_2-\hat{y}_i)^2} \\ \vdots & \ddots & \vdots & \vdots \\ \sqrt{(x_i-\hat{x}_1)^2+(y_i-\hat{y}_1)^2} & \dots & \dots & \sqrt{(x_i-\hat{x}_j)^2+(y_i-\hat{y}_j)^2} \end{pmatrix} \tag{8.5}$$

Underlying a Gaussian distribution (equation 8.6), the single values are weighted by their probability of best suitable distance between frame n and $n+1$. The range of this probability starts at 0 - which is a very unlikely match, till 1, which gives the best match for the new position of an object of frame n in the frame $n+1$.

$$G_{ij} = e^{-\frac{r_{ij}^2}{2\sigma^2}} \tag{8.6}$$

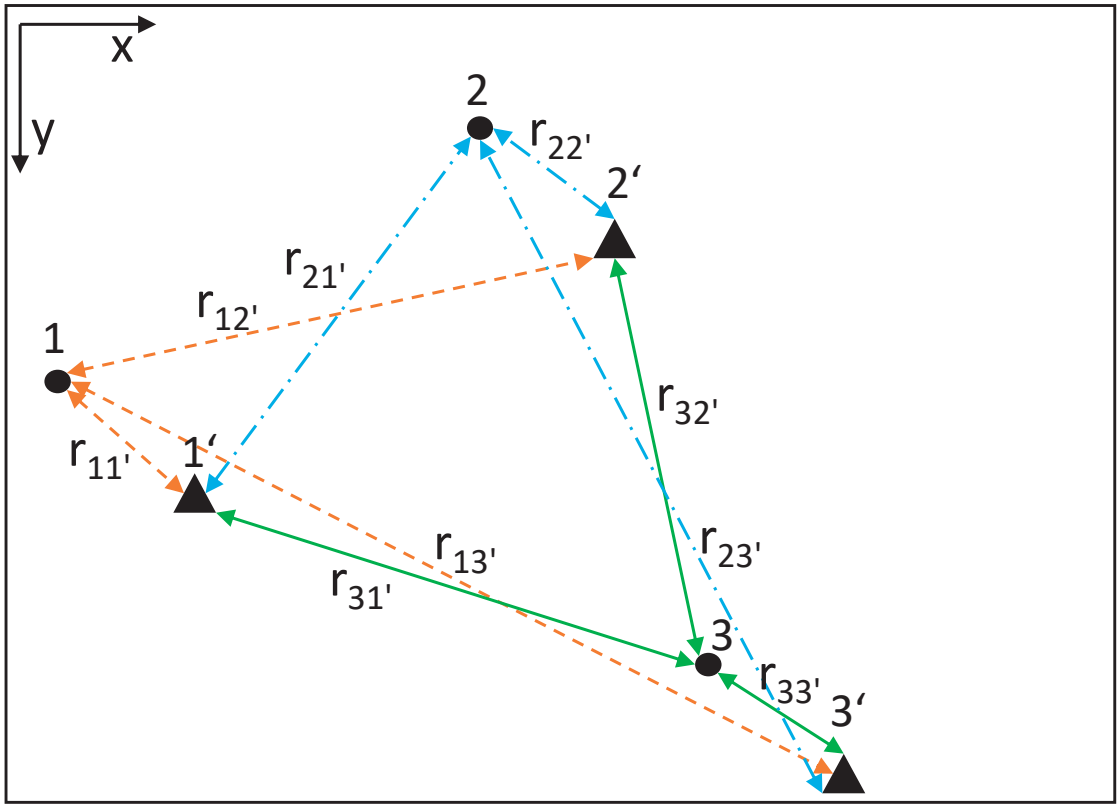

Figure 8.1: Distance determination

$$\mathsf{G} = \begin{pmatrix} e^{-\frac{r_{11}^2}{2\sigma^2}} & \dots & e^{-\frac{r_{1j}^2}{2\sigma^2}} \\ \vdots & \ddots & \vdots \\ e^{-\frac{r_{i1}^2}{2\sigma^2}} & \dots & e^{-\frac{r_{ij}^2}{2\sigma^2}} \end{pmatrix} \tag{8.7}$$

In the case, that

$$dim(\mathsf{C_n}) = dim(\mathsf{C_{n+1}}), \tag{8.8}$$

which means, that the number of objects of frame n and $n+1$ did not change, a diagonal matrix will be the result

$$\hat{\mathsf{G}} = round(\mathsf{G}) = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \\ 0 & 0 & \dots & 1 \end{pmatrix}. \tag{8.9}$$

As already mentioned, it is possible, that the number of objects decreases or increases in the following image frame, which leads to different dimensions of the compared matrices:

Case a: Number of objects decreasing:

$$dim(\mathsf{C_n}) > dim(\mathsf{C_{n+1}}) \tag{8.10}$$

Case b: Number of objects increasing:

$$dim(\mathsf{C_n}) < dim(\mathsf{C_{n+1}}) \tag{8.11}$$

Considering both cases and the requirement to calculate the distance matrix or proximity matrix, the dimensions of the two input matrices have to be adjusted by the following steps:
Extract the x any y coordinates from $\mathsf{C_n}$ and $\mathsf{C_{n+1}}$ through

$$\begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{pmatrix} = {\mathsf{C_n}}^{\mathrm{T}} \tag{8.12}$$

and

$$\begin{pmatrix} \hat{\boldsymbol{x}} \\ \hat{\boldsymbol{y}} \end{pmatrix} = {\mathsf{C_{n+1}}}^{\mathrm{T}}, \tag{8.13}$$

further

$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}, \boldsymbol{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}, \boldsymbol{I} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}, \tag{8.14}$$

where the length of the unit vector $\boldsymbol{I}$ corresponds to the length of C_{n+1}. Similar the matrix $\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$ and $\hat{\boldsymbol{I}}$ are formed, where the length of this unit vector $\hat{\boldsymbol{I}}$ corresponds to $\mathsf{C_n}$.

$$\hat{\boldsymbol{x}} = \begin{pmatrix} \hat{x}_1 \\ \vdots \\ \hat{x}_n \end{pmatrix}, \hat{\boldsymbol{y}} = \begin{pmatrix} \hat{y}_1 \\ \vdots \\ \hat{y}_n \end{pmatrix}, \hat{\boldsymbol{I}} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix} \tag{8.15}$$

with the outer product of

$$\hat{\boldsymbol{I}}\boldsymbol{x}^{\mathrm{T}} = \begin{pmatrix} x_1 & x_1 & \dots & x_1 \\ x_2 & x_2 & \dots & x_2 \\ \vdots & \ddots & \vdots & \vdots \\ x_n & \dots & \dots & x_n \end{pmatrix} = \mathsf{x_n} \tag{8.16}$$

$$\hat{\boldsymbol{I}}\boldsymbol{y}^{\mathrm{T}} = \begin{pmatrix} y_1 & y_1 & \cdots & y_1 \\ y_2 & y_2 & \cdots & y_2 \\ \vdots & \ddots & \vdots & \vdots \\ y_n & \cdots & \cdots & y_n \end{pmatrix} = \mathsf{y_n} \tag{8.17}$$

and

$$\boldsymbol{I}\hat{\boldsymbol{x}}^{\mathrm{T}} = \mathsf{x_{n+1}} \tag{8.18}$$

$$\boldsymbol{I}\hat{\boldsymbol{y}}^{\mathrm{T}} = \mathsf{y_{n+1}} \tag{8.19}$$

the matrices $\mathsf{x_n}$, $\mathsf{y_n}$, $\mathsf{x_{n+1}}$ and $\mathsf{x_{y+1}}$ are calculated. The result is a uniform length (Equation 8.20) of all necessary matrices for further calculations like the proximity matrix.

$$dim(\mathsf{x_n}) = dim(\mathsf{y_n}) = dim(\mathsf{x_{n+1}}) = dim(\mathsf{y_{n+1}}) \tag{8.20}$$

To this, the calculation is done in Python by:

```
1
2  # Extract x,y values of frame n and frame n+1
3  #xO,yO ... x-old, y-old
4  #xN, yN ... x-new, ynew
5  xO,yO = KoordinatenFrameN0.T
6  xN, yN = KoordiantenFrameN1.T
7
8  #Create ones-matrices of the coordinates
9  #of frame n and n+1
10 onesArrA = np.ones((KoordinatenFrameN0.shape[0],1))
11 onesArrB = np.ones((KoordiantenFrameN1.T.shape[0],1))
12
13 #calculate the outer products of ones and x0,xN
14 xO = np.outer(xO,onesArrB)
15 xN = np.outer(onesArrA,xN)
16
17 #calculate the outer products of ones and y0,yN
18 yO = np.outer(yO,onesArrB)
19 yN = np.outer(onesArrA,yN)
20
21 #calculate the deviations of all
22 #possible coordinates of frame n
23 #and n+1
24 dx = np.power(xO-xN,2)
25 dy = np.power(yO-yN,2)
26 R = np.sqrt(dx+dy)
```

```
27
28 #Probibility function declaration
29 sigma = 5
30 G = np.exp(-(R*R)/(2*sigma*sigma))
```

Correlation Matrix

The probability matrix G shows with its rows and columns the correlation between objects of frame n and frame $n+1$. An example is shown in the following:

$$\hat{\mathsf{G}} = round(G) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{8.21}$$

G shows, that every object of frame n and frame $n+1$ is in a single row. The row indicates the position of the corresponding object n and $n+1$ in the following frame $n+1$. For example, the object 1 of frame n corresponds with object 1 of frame $n+1$ and object 3 of frame n corresponds to object 4 of frame $n+1$. So the aim of the probability matrix evaluation is the classification of objects resulting into a dynamic and easy to use correlation matrix, which assigns ongoing objects into the persistent object database. The evaluated probability matrix result for the given example leads to the correlation matrix

$$\mathsf{CorrMatrix} = \begin{pmatrix} 1 & 1 \\ 2 & 2 \\ 3 & 4 \\ 4 & 3 \end{pmatrix}. \tag{8.22}$$

This matrix is computed in Python via two loops, which reads the position of ones, evaluated for their positions and assigns the values to a new matrix CorrMatrix. With the new information of the correlation matrix, the updated object database can be calculated:

```
1  i=0
2  j=0
3
4  while(i<G.shape[0]):
5   while(j<G.shape[1]):
6    if(G[i,j]==1):
7     CorrMatrix = np.append(CorrMatrix,np.array([[i,j]]),axis=0)
8    j+=1
9   j=0
10  i+=1
```

```
11 for C in CorrMatrix:
12  c1,c2 = C
13  starnumber,x,y,online = StarTable[c1]
14  xnew,ynew = CurrentCoordinates[c2]
15  StarTable[c1]=starnumber,xnew,ynew,1
16 return StarTable
```

8.3.7 Camera Calibration

The position of the camera in the guidescope has no fixed orientation and can be rotated in the eyepiece of the scope, which makes a calibration of the position essential. The kinematics must also be considered, so its behaviour also influences the result of the camera calibration, a essential step of relative and further reverse kinematics calculations. To do the calibration process, the given kinematics is moved into both possible directions, while a single object is tracked. The information of the tracking process, more precisely the x and y data of the movement, is logged into two arrays. One array for each axis. These coordinates are saved in RAlog and DECLog. Depending on the focal length, the number of measurement points are set regarding to the distance of the movement of the objects while calibration. (RA = right ascension axis, DEC = declination axis)

$$\mathsf{RALog} = \begin{pmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}, \mathsf{DECLog} = \begin{pmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_n & y_n \end{pmatrix}. \tag{8.23}$$

The calibration of the right ascension axis is done by:

```
1  global RACalibIndicator
2  global RALog
3  if(len(RALog) <= (CalibrationSteps-1)):
4   #move mount in RA
5   GPIO.output(20,1)
6   #save x and y values to array
7   RALog = np.append(RALog,np.array ([[StarTable[TrackedStar,1],StarTable
        [TrackedStar,2]]])
8   ,axis=0)
9  if(len(RALog) == CalibrationSteps):
10  #Calibration done, set indicator to 1
11  #and disable mount movement in RA
12  RACalibIndicator = 1
13  GPIO.output(20,0)
14  cv2.putText(frame,'RA Calibration on: '+str(len(RALog))+' Steps'
        ,(10,20),1,1,(255,255,255),1,0)
```

And the calibration of the declination axis by:

```
1  global DECCalibIndicator
2  global DECLog
3  if(len(DECLog) <= (CalibrationSteps-1) and RACalibIndicator == 1):
4   #move mount in DEC
5   GPIO.output(21,1)
6   #save x and y values to array
7   DECLog = np.append(DECLog,np.array
8   ([[StarTable[TrackedStar,1],StarTable[TrackedStar,2]]])
9   ,axis=0)
10 if(len(DECLog) == CalibrationSteps):
11  #Calibration done, set indicator to 1
12  #and disable mount movement in DEC
13  GPIO.output(21,0)
14  DECCalibIndicator = 1
15  cv2.putText(frame,'DEC Calibration on: '+str(len(DECLog))+' Steps'
        ,(10,32),1,1,(255,255,255),1,0)
```

8.3.8 Fitting

After successfully creating the log files of right ascension and declination axes, a linear function is fitted through the data points of each axis (RALog and DECLog). Used is a least square algorithm provided by the open source numpy library of Python [23]:

```
1 #calculate slope of fitted function through xRA, yRA,
2 #degree of function = 1
3 slopeRA, zRA = np.polyfit(xRA,yRA,1)
4 #calculate slope of fitted function through xDEC, yDEC,
5 #degree of function = 1
6 slopeDEC, zDEC = np.polyfit(xDEC,yDEC,1)
7 #convert slope from radians to degree
8 RAangle = np.degrees(np.arctan(slopeRA))
9 DECangle = np.degrees(np.arctan(slopeDEC))
```

The variables *slopeRA* and *slopeDEC* are the slopes of the two fitted linear functions. The argument of the fitting function requires the data points (x,y) and degree of the polynomial - in this case a first order degree. The slope, calculated as the quotient of y and x values, can converted to radians through the evaluation of its tangent value. So the result can be expressed in radians or converted to degrees with the function $np.degrees(radiansValue)$ for easy legibility of the user.

8.3.9 Coordinate Transformation

The calculated slopes of the data points can now be used to determine a relative coordinate system, which includes the camera position in the guidescope and also the behaviour of the used kinematics. That circumstance makes the whole tracking algorithm independent of the kinematics, the observation site and the camera orientation, which guarantees a maximum of flexibility. The slope of the declination axis gives the rotation of the coordinate system, assuming orthogonality. Additionally a single object can be set as origin of the new coordinate system, so the relative coordinate system is rotated and shifted. The advantage of the additional shift is the easy way to evaluate deviations within a target-actual comparison. Every value unequal to zero gives the deviation of the object in pixels in the relative coordinate system.
The transformation of the object database into the relative coordinate system in Python is done with:

```
1  StarTableTrans = np.empty((0,4),float)
2  for Star in StarTable:
3   #extract object number, coordinates and online
4   #of each object of the database
5   number, x, y, online = Star
6   #shift
7   x = x-xreference
8   y = y-yreference
9   #rotation by RA-angle
10  xtrans = x*np.cos(slopeRA)+y*np.sin(slopeRA)
11  ytrans = y*np.cos(slopeRA)-x*np.sin(slopeRA)
12  #write transformed database
13  StarTableTrans = np.append(StarTableTrans,np.array
14  ([[number,xtrans,ytrans,1]]),axis=0)
```

The transformation process includes the object database values and sets up a new transformed database, while maintaining the order of objects. A successful transformation can be recognised by the indication, that the value of the desired tracked object in the transformed object database at time of transformation is equal to zero or very close to zero.

8.4 Tracking

The tracking functions uses the calculated transformed object database, to track objects by minimisation of the deviation of the object to the origin of the relative coordinate system. The minimisation is done by a kinematics movement corresponding to the object deviation in a closed loop function [24]. The Raspberry Pi creates pulses, send them

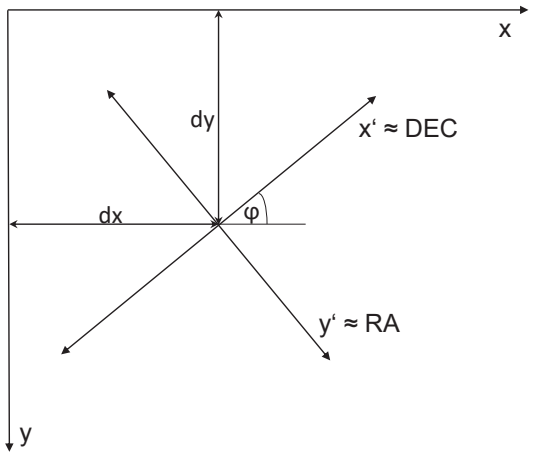

Figure 8.2: Transformation: Rotation by φ und shift by dx and dy

to the electrical interface, which translates them to the ST-4 standard input signal of the telescope mount. The behaviour of the switching time depends, as already mentioned, on the used type of interface, such as relays or transistors. Relays need a minimum on-time cause of their inertia. The speed of the resulting mount speed is set on the EQ-6 hand controller between 0.25 and 1x absolute sidereal speed. A certain inertia of the mount has to be considered, and also atmospheric induced distortions, which affect the apparent object position. The deviation is split into four cases: $dx > 0$, $dx < 0$, $dy > 0$,$dy < 0$. Programmed in Python, considering the minimum on-time of relays:

```
1  if(dx > 0 ):
2  GPIO.output(20,1) #RA+
3  GPIO.output(26,0) #RA-
4  time.sleep(sleeptime)
5  if(dx < 0 ):
6  GPIO.output(26,1) #RA-
7  GPIO.output(20,0) #RA+
8  time.sleep(sleeptime)
9  if(dy < 0):
10 GPIO.output(19,1) #DEC+
11 GPIO.output(21,0) #DEC-
12 time.sleep(sleeptime)
13 if(dy > 0 ):
14 GPIO.output(21,1) #DEC-
15 GPIO.output(19,0) #DEC+
16 time.sleep(sleeptime)
```

The correction movement takes place as long as there is a deviation, until $dx = 0$, $dy = 0$.

8.5 PyGame

The whole GUI is programmed as a combination of openCV and PyGame and the requirement to convert image and video data between the libraries considering their compatibility. To use the PyGame library, the initialisation has to be done by:

8.6 Initialise PyGame

```
1 import pygame
2 from pygame.locals import *
```

A new window with a resolution of 800 by 600 pixels is created by:

```
1 pygame.display.set_caption("Window Name")
2 screen = pygame.display.set_mode([800,600])
```

For additional program outputs via textual overlays in the PyGame window, a font and its size must be initialised by:

```
1 pygame.font.init()
2 #...SysFont('FontName',TextSize)
3 myfont2 = pygame.font.SysFont('Arial',15)
```

8.6.1 User Inputs: Buttons

An easy way to implement buttons in PyGame is to load pre created button images and place them to a specific area in the PyGame window.

```
1 #with path/filename.png
2 button_name = pygame.image.load("buttons/button_name.png")
3 #display button at x,y in the window
4 screen.blit(button_name,(651,5))
```

To connect a called function and the button, the position of the mouse cursor is checked when pressed and evaluated by its coordinates in the window as following:

```
1
2 #define function
3 def CheckMousePos():
4
5 #read mouse cursor position
6 xmouse,ymouse = pygame.mouse.get_pos()
```

```
7
8  #check an area, where x is betweet
9  #x1 and x2
10 #and y is between
11 #y1 and y2
12 if(ymouse > y1Value and ymouse < y2Value and xmouse > x1Value and
       xmouse < x2Value):
13  #execute the desired button function
14  ButtonFunction()
```

The called function is executed by a transition-function. For clarity in the program code these functions are outsourced and called by:

```
1  #define the button function
2
3  def ButtonFunction():
4   doSomething()
5   callOtherFunctions()
6
7  #Other used functions
8  #as example:
9
10 def Search():
11  Search()
12  global StarTable
13  print("Select")
14  StarTable = getInitialStarTable()
15  print(StarTable)
16
17 #Do starmatching an track objects visually
18 #without mount movement
19 def TrackButton():
20  print("Track")
21  global StarTable
22  StarTable = StarMatching(StarTable)
23  print(StarTable)
24
25 #capture image and save it to a file
26 def CapImgButton():
27  print('CapImg')
28  im = Image.fromarray(tframe)
29  im.save(strftime("%Y-%m-%d-%H:%M:%S",gmtime())+"img.jpeg")
30  print("CapImg")
```

8.7 Threading

Due to image acquisition, depending on the exposure time of the camera, the program is blocked and allows no user inputs and no program outputs. This circumstance causes a difficult operability of the program, because the user inputs are only recognised, when the image acquisition is inactive. The same applies to the output such as image visualisation and calculations. For this reason a parallel operation of the image acquisition and the rest of the program is required. In Python, the available threading library provides that kind of desired parallelism.
In the first step, the library must be imported by:

```
1 import threading
```

A global variable, which stores the image data is initialised and preallocated:

```
1 global tframe
2 #x,y,z, wehre z is the image depth
3 #z = 3 = RGB
4 #z= 1 = grey level
5 tframe = np.zeros((480,640,3))
```

The actual function is packed into a threading class:

```
1 #tframe ... threaded frame
2
3 class getFrame(threading.Thread):
4 def __init__ (self):
5  threading.Thread.__init__(self)
6  self.stopThread = False
7
8  def run(self):
9   while True:
10    global tframe
11    global ExpTime
12    #set camera controls
13    camera.set_control_value(asi.ASI_EXPOSURE, ExpTime)
14    camera.set_control_value(asi.ASI_GAIN, CamGain)
15    #save image data to tframe
16    tframe = camera.capture_video_frame()
17
18    #stop the thread
19    if self.stopThread == True:
20    break
21  def stopThread(self, stopThread):
```

```
22    self.stopThread = stopThread
```

To start the function of the thread class, it is called by:

```
1  t1 = getFrame()
2  t1.start()
```

Now the camera starts a continuous video capture and updates the global variable *tframe* with actual data. This data is read parallel to this by the main program for calculation and presentation.

Chapter 9

Program Explanation

The GUI, programmed with the PyGame library, makes several adjustments possible. To obtain an optimal result regarding exposure time and gain some settings must be made. These settings are done by the user input part of the program, while the result can be estimated by the program outputs. Thus it is possible to adjust all settings till the desired result is visible, such as sharply edged objects, the number of objects and a dark background with the least possible amount of noise.

9.1 User Inputs

All necessary options are summarised as user input functions, which are connected to buttons. General settings, such as the number of data points of the calibration process are fixed global values set in the source code due initialisation.

9.1.1 Exit (Button 1)

The exit button closes the PyGame window, and the GPIO ports and also stops the video stream of the ZWO Asi camera, to avoid problems in case of a restart of the program.

9.1.2 Select (Button 2)

The select button starts an initial search for objects and shows the calculated boundary pixels of each object and the number of recognised objects, while a first temporary object database is generated. The visualisation of the boundary pixels is useful to evaluate the correct settings of exposure time a gain for accurate object detection. In case the gain value is too high, the distinguishable of objects is not given through image data noise. So, a compromise between exposure time and the gain must be considered. With the select button, that condition can be checked every time as desired. High camera noise also causes flickering moments of the objects.

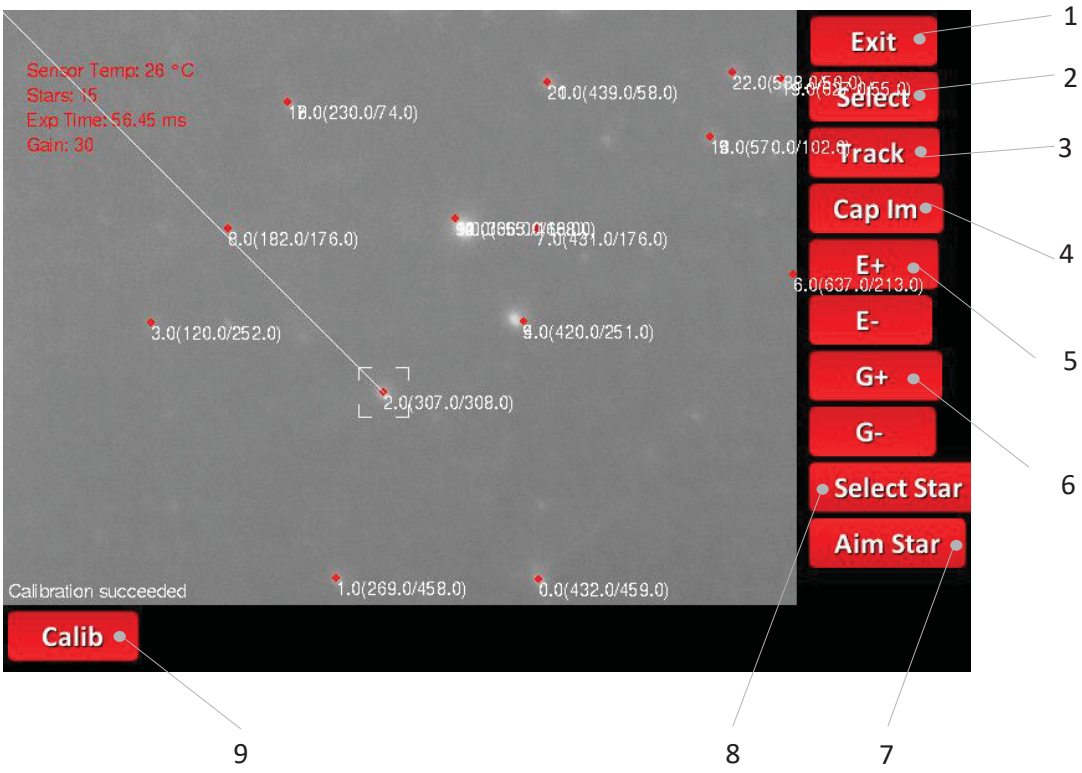

Figure 9.1: Program screenshot

9.1.3 Track (Button 3)

The track button tracks all recognised objects, excluding new objects, and refreshes the object database every program cycle. The tracked objects are visually marked. This option can be deactivated at any point of the process and without any movement of the mount.

9.1.4 Capture Image (Button 4)

The cap image button acquires an image at any time the user presses this button. It stores the actual original image data of the camera in a jpeg file with a time stamp as file name. This makes a parallel use of guiding and image photography possible with a single device.

9.1.5 Exposure +/- (Button 5-2)

This button controls the exposure time of the acquired image. It starts with an initial value of 1 ms and can be increased and decreased live in the whole process for easy adjustment.

9.1.6 Gain +/- (Button 6-2)

This button controls the gain value, with an initial value of 50. As already mentioned, a compromise between gain and exposure time must be set. If the gain is too high, a lot of noise covers the acquired image, which makes the object detection inaccurate and gives distortions of the boundary pixels. This leads to inappropriate moments calculations. Considering, the camera is very light sensitive, there is no need to run the program at high gain values. An exception might be ultra fast object tracking under low light conditions with the knowledge of inaccurate object tracking.

9.1.7 Aim Object (Button 7)

This function aims an recognised object with the unique number of the compared and persistent object database, which guarantees, except limit cases, the tracking of one desired object. This object is specially visually marked in the GUI and starts the movement of the mount.

9.1.8 Select Star (Button 8)

This function selects the desired object for tracking and controlling the kinematics.

9.1.9 Calibration (Button 9)

This function starts the calibration process, consisting of the right ascension and declination axis calibration, the linear fitting and the coordinate transformation and gives the transformed object database in order to the reverse kinematics. The number of calibration steps is set to an initial value of 100 data points per axis.

9.2 Program Outputs

Any data the program generates can be presented in the GUI. Object are marked as an overlay of the video stream, consisting the centre point, the object number and the absolute image coordinates in x and y. Through the camera communication protocol, a value representing the camera sensor temperature, is read and converted to $^{\circ}C$. The absolute number of stars recognised in the initial frame is also shown in the GUI. To estimate a nicely working compromise between exposure time and gain, that values are also shown in the GUI. The exposure time is calculated by one second divided by the number of captured frames per second (fps).

$$Exp = \frac{1s}{fps} \tag{9.1}$$

Chapter 10

Testings

To evaluate each step in the software development process, a step by step testing is applied. This is a very important point in the development of a comprehensive and diverse software, in order to avoid errors of partial program functions.

10.1 Simulated Test - Video Stream Input

The simulation includes a video capture of an astronomic software suite 'Stellarium', which shows the two-dimensional movement of heavenly objects. That includes the change of number of recognised objects. That test was used, to avoid the influence of camera noise and exposure time, thus the video capture gives perfectly image data for further processing. The source code is adapted to a static video input stream instead of reading data from the camera. Especially at the beginning of the software development phase, the functionality of the object matching algorithm is tested.

10.2 Static Testing

The next step in simulation is testing under fixed conditions, by aiming the tracking camera on the mount to a star constellation on a distant image. This case includes testing the gain and exposure time of the tracking camera and also testing the calibration and the two-axis tracking behaviour of the mount. For test purpose, the system is setted up, such as exposure time, threshold and gain, calibrated and coordinate transformed. After that, the position of the tracked star is moved out of the coordinate origin to calculate the deviations of the object. In the next step, the tracking function is activated to test the mount kinematics and take back the star to the origin of the coordinate system. A peculiarity of the mount is the tracking speed in the negative right ascension axis - in real world testing, this speed is a stop of the kinematics, so the rotation of the earth moves the star in this particular direction. For this reasons only deviations in the positive right ascension and

positive and negative declination axes can be tested.

10.3 Dynamic Testing

In order to move a picture of an object constellation like the stars move in the sky, the star image is fixed on a rotating plate with an angular speed equal to the earth. The plate is rotated by a precision gear motor controlled via a *B&R* PLC (Programmable Logic Controller). The telescope mount is set up with the pole finder right to the centre of the rotating plate. The distance of the telescope is determined by the adjustable latitude of the mount location. This angle must be respected, otherwise no approximate orthogonal coordinate system results during the calibration process.

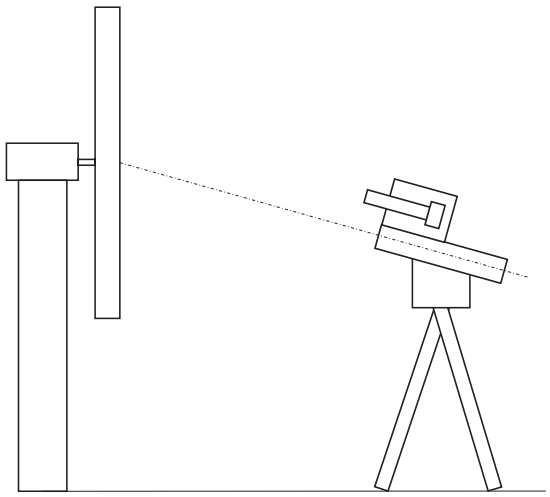

Figure 10.1: Rotating plate

10.4 Real World Testing

The first real world tests are done on 8/1/2017 at $47°7'41.31"N$, $15°21'32.62"E$ pointing to the southern hemisphere with the first version of the program without the PyGame GUI. The tracking mount is positioned with a deliberate alignment error in pile high and azimuth to provoke permanent correction movements. While the running calibration process, an image with the observing camera is taken to visualize the procedure: The lower straight line in figure 10.2 is the result of the continuous capture of the star while right ascension calibration, costing of 100 data points, while the upper line is the capture during the declination calibration. In this position, the tracked object reached its end point of the calibration process and will be centred again after the tracking is enabled. Image 10.3 presents the user interface of the non GUI version of the program in the first test. The program shows the successful calibration with its steps taken and the selected star

Figure 10.2: Calibration procedure

number of the object database. The blue and white lines indicates the calculated relative coordinate system, with dRA and dDEC representing the deviation of the object in the given coordinate system. At this test, the processed orthogonality error is $-11.08°$.

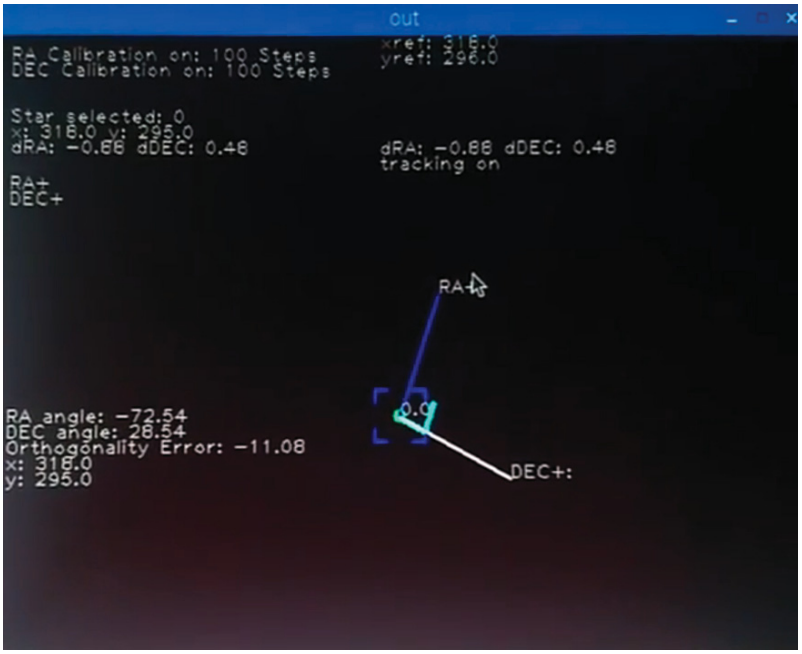

Figure 10.3: Guiding

10.5 Results

As a result, a comparison is shown (Fig. 10.4), which represents a long exposure with (b) and without (a) activated tracking. Both pictures show the same picture detail with an

Figure 10.4: Comparison

(a) unguided

(b) guided

exposure time of 180 s, ISO 500 and a focal length of the telescope of 700 mm, captured with a Canon EOS 50D DSLR and a crop factor C of 1.6, which leads to an effective focal length of 1120 mm regarding to equation 10.1.

$$f_{effektiv} = C \cdot f_{scope} = 1.6 \cdot 700\,mm = 1120\,mm \tag{10.1}$$

Each of the images in figure 10.4 (a) and (b) are shot with a statical movement of the mount of the right ascension axis. Image (a) is unguided, with that already mentioned static movement. Image (b) is the guided image, with the static movement and additional calculated correction pulses of the program in both axes. The objects in image (b) are sharp and not blurred, based on the successful guidance through the developed software.

10.6 More Shots

Figure 10.5 is a summarised long time exposure with about 1.5 hours in total of the Orion Nebula. The used telescope is a newton refractor (130mm opening, 650 mm focal length) in combination with a DSLR (digital single lens reflex) camera (Canon EOS 50D). Image

Figure 10.5: Orion Nebula

10.6 shows (a) the Andromeda Nebula with a total exposure time of one hour and a short exposure of the Plejades (b). A single shot of the Milky Way (Fig. 10.7) with a total

(a) Andromeda

(b) Plejaden

Figure 10.6: Andromeda and Plejades

exposure time of only 340 seconds at a low camera ISO setting (ISO 500) in combination with a wide angle lens (Tokina 11-16 mm f/2.8). Figure 10.8 is a guided wide angle long exposure image of the Milky Way in the southern hemisphere. The relative movement of the mount in direction of the tracked objects is shown by the blurriness of the trees in front of the camera, while the tracked objects are sharp.

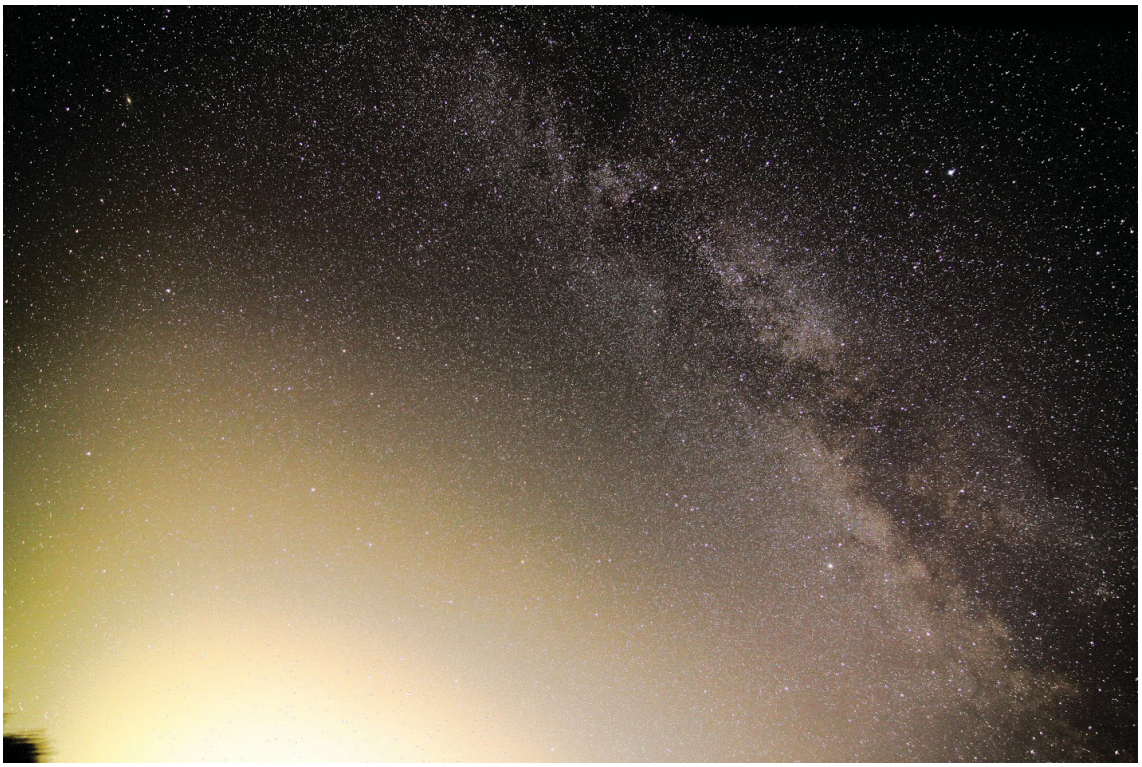

Figure 10.7: Milky Way guided

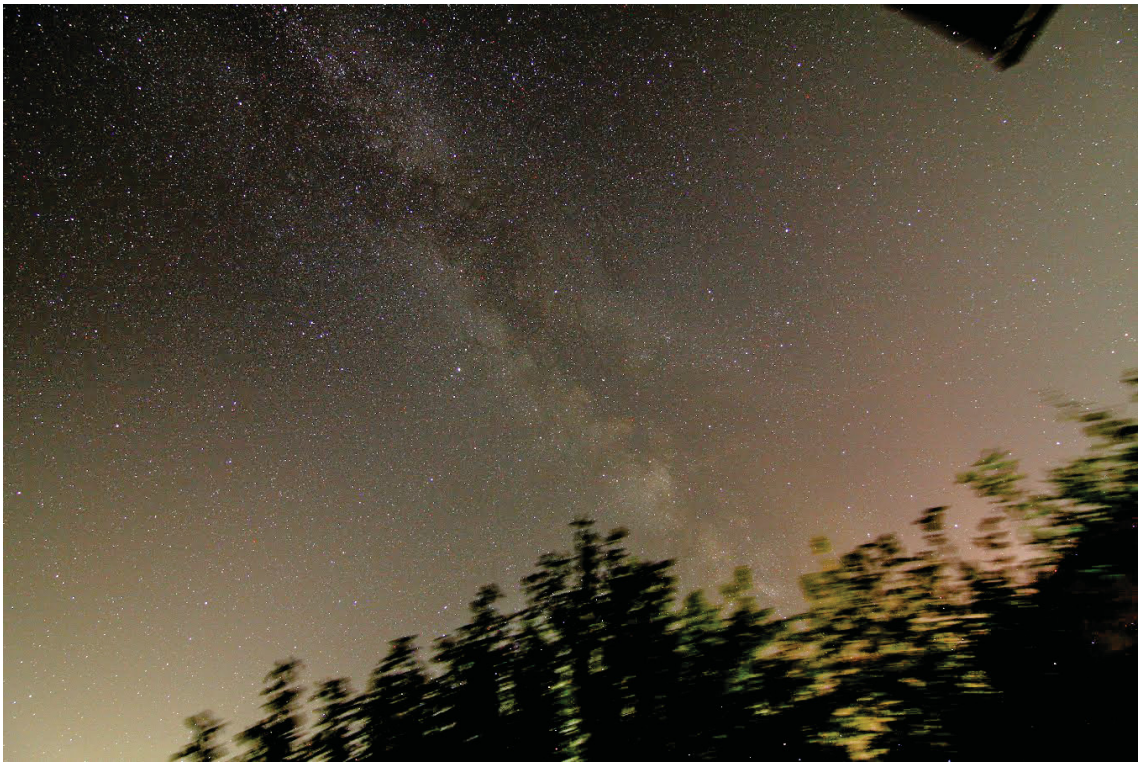

Figure 10.8: Blurred earth objects

Chapter 11

Conclusion and Outlook

As tested, the software worked very accurate for long time exposures. Short exposures are characterised by a little overshoot of the closed loop controlling part of the software. To avoid those overshoots, some adjustments and correlations between minimal pulse duration of the guiding signal and the tracking speed have to set up in the mount control for further must be tests. So far, the closed loop part is programmed as a kind of proportional controller, where the deviation error is used for the target-actual comparison with a static pulse time, independently of the distance to the origin. An improvement would be a proportional component controlling the minimal pulse time. This advantage would probably may make noticeable difference especially at small deviations. Another way to improve the accuracy in terms of the calculated moments would be an evaluation of the deviation regarding an in-axis image histogram (Fig. 11.1). Changes distribution of the histogram would give the actual deviation of the object. This would make the program independent

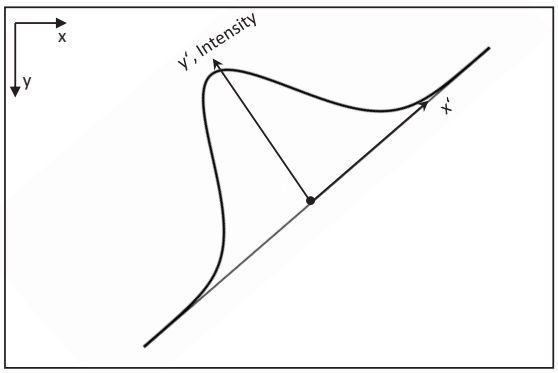

Figure 11.1: Histogram

of the calculated moments of the object and less susceptible of distortions through the atmosphere, causing a little twitching of the objects. In order to the distortions, the kinematics tries to correct its position regarding to this image information. To minimise these 'false movements' a short mean distance should be calculated, to react only when there is actually a physical deviation of the tracked object. Also, it would be interesting to test

the controller on another mount, or even to build a functional and fast kinematics, which is lightweight and transportable. Another idea is the possibility of remote control of the software. So far, this is very limited because of the data rates of the USB 3.0 camera. The difficulty is the remote video access to the Raspberry Pi including the visualisation of openCV and PyGame. For even easier handling, it might be possible to compile the software for Android-devices and connect the electrical control unit via bluetooth to a mobile device. In this case, the Android-device would process the image data. A big challenge would be the connection of the camera via USB to the mobile phone or tablet.

Appendix A

Source Code

```
1  ############################################################
2  # capturing video via threading, convert it, make surface
3  # and represent it via pygame
4  # pip3 install zwoasi
5  # pip install zwoasi (thanks to Steve Marple for the
6  # python wrapper)
7  # Download asi-sdk and copy the armv7 dynamic lib
8  #to /lib/zwoasi/armv7/
9
10
11 import cv2
12 import numpy as np
13 import os
14 import threading
15 import pygame
16 from pygame.locals import *
17 import sys
18 import zwoasi as asi
19 import time
20 from PIL import Image
21 from time import gmtime, strftime
22
23 # Imported from previous program #
24 TrackedStar = 0
25
26 CalibrationSteps = 100
```

```
27 StartCalibration = 0
28 RACalibIndicator = 0
29 RALog = np.empty((0,2),float)
30
31 DECCalibIndicator = 0
32 DECLog = np.empty((0,2),float)
33
34 slopeRA = 0
35 slopeDEC = 0
36 slopeIndicator = 0
37
38 StarTableTrans = np.empty((0,4),float)
39
40 dx = 0
41 dy = 0
42
43 TrackingIndicator = 0
44
45 ShowLog = 1
46
47 RAangle = 0
48 DECangle = 0
49
50 def getCoordinates():
51    frame = tframe
52    TempCoordinates_ = np.empty((0,2),float)
53    Threshold_ = 120
54    ret, thresh_ = cv2.threshold(tframe,Threshold_,255,0)
55    image, cnts, hierarchy = cv2.findContours(thresh_, cv2.
         ↪ RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
56    for c in cnts:
57       M = cv2.moments(c)
58       if(M['m10']!=0):
59          cX = int(M['m10']/M['m00'])
60          cY = int(M['m01']/M['m00'])
61          TempCoordinates_ = np.append(TempCoordinates_,np.
               ↪ array([[cX,cY]]),axis=0)
62    return TempCoordinates_
63
```

```
64 def getInitialStarTable():
65     Coordinates = getCoordinates()
66     TempStarTable = np.empty((0,4),float)
67     StarNumber = 0
68     print(Coordinates)
69     for x,y in Coordinates:
70         TempStarTable = np.append(TempStarTable,np.array([[
               ↪ StarNumber,x,y,1]]),axis=0)
71         StarNumber+=1
72     return TempStarTable
73 
74 def StarMatching(StarTable):
75     CorrMatrix = np.empty((0,2),int)
76 
77     CurrentCoordinates = getCoordinates()
78     i,x0,y0,onlineTable = StarTable.T
79     xN,yN = CurrentCoordinates.T
80     onesArrA = np.ones((StarTable.shape[0],1))
81     onesArrB = np.ones((CurrentCoordinates.shape[0],1))
82     x0 = np.outer(x0,onesArrB)
83     xN = np.outer(onesArrA,xN)
84     y0 = np.outer(y0,onesArrB)
85     yN = np.outer(onesArrA,yN)
86     dx = np.power(x0-xN,2)
87     dy = np.power(y0-yN,2)
88     R = np.sqrt(dx+dy)
89     sigma = 20
90     G=np.exp(-(R*R)/(2*sigma*sigma))
91     G=np.round(G,0)
92 
93     i=0
94     j=0
95 
96     while(i<G.shape[0]):
97         while(j<G.shape[1]):
98             if(G[i,j]==1):
99                 CorrMatrix = np.append(CorrMatrix,np.array([[i,j
                       ↪ ]]),axis=0)
100             j+=1
```

```
101         j=0
102         i+=1
103     for C in CorrMatrix:
104         c1,c2 = C
105         starnumber,x,y,online = StarTable[c1]
106         xnew,ynew = CurrentCoordinates[c2]
107         StarTable[c1]=starnumber,xnew,ynew,1
108     print(CorrMatrix)
109     return StarTable
110
111 def getSlope():
112     global slopeRA
113     global slopeDEC
114     global RAangle
115     global DECangle
116
117     xRA, yRA = RALog.T
118     xDEC, yDEC = DECLog.T
119
120     slopeRA, zRA = np.polyfit(xRA,yRA,1)
121     slopeDEC, zDEC = np.polyfit(xDEC,yDEC,1)
122     RAangle = np.degrees(np.arctan(slopeRA))
123     DECangle = np.degrees(np.arctan(slopeDEC))
124
125 def RACalibration(StarTable,TrackedStar):
126
127     global RACalibIndicator
128     global RALog
129     if(len(RALog) <= (CalibrationSteps-1)):
130         GPIO.output(20,1)
131         RALog = np.append(RALog,np.array([[StarTable[
              ↪ TrackedStar,1],StarTable[TrackedStar,2]]]),axis
              ↪ =0)
132     if(len(RALog) == CalibrationSteps):
133         RACalibIndicator = 1
134         GPIO.output(20,0)
135     cv2.putText(frame,'RA Calibration on: '+str(len(RALog))+
          ↪ ' Steps',(10,20),1,1,(255,255,255),1,0)
136
```

```
137 def DECCalibration(StarTable,TrackedStar):
138     global DECCalibIndicator
139     global DECLog
140     if(len(DECLog) <= (CalibrationSteps-1) and
          ↪ RACalibIndicator == 1):
141         GPIO.output(19,1)
142         DECLog = np.append(DECLog,np.array([[StarTable[
              ↪ TrackedStar,1],StarTable[TrackedStar,2]]]),axis
              ↪ =0)
143     if(len(DECLog) == CalibrationSteps):
144         GPIO.output(19,0)
145         DECCalibIndicator = 1
146     cv2.putText(frame,'DEC Calibration on: '+str(len(DECLog)
          ↪ )+' Steps',(10,32),1,1,(255,255,255),1,0)
147
148 def CoordinatesTransformation(StarTable,TrackedStar):
149     global StarTableTrans
150     global xreference
151     global yreference
152
153     StarTableTrans = np.empty((0,4),float)
154     for Star in StarTable:
155         number, x, y, online = Star
156         x = x-xreference
157         y = y-yreference
158         motionAnglex = abs(DECangle)
159         motionAngley = abs(RAangle)
160         xtrans = x*np.cos(motionAnglex * np.pi/180)-y*np.sin(
              ↪ motionAnglex * np.pi/180)
161         ytrans = x*np.sin(motionAnglex * np.pi/180)+y*np.cos(
              ↪ motionAnglex * np.pi/180)
162         #xtrans = x*np.cos(slopeDEC)+y*np.sin(slopeDEC)
163         #ytrans = y*np.cos(slopeDEC)-x*np.sin(slopeDEC)
164         StarTableTrans = np.append(StarTableTrans,np.array([[
              ↪ number,xtrans,ytrans,1]]),axis=0)
165
166 # Init everything #
167
168 asi.init("/lib/zwoasi/armv7/libASICamera2.so")
```

```
169 camera = asi.Camera(0)
170 global ExpTime
171 global CamGain
172 global tframe
173 global tresh
174 global TempCoordinates
175 global StarTable
176 TempCoordinates = np.empty((0,2),float)
177 ExpTime = 100
178 CamGain = 50
179 tframe = np.zeros((480,640,3))
180 thresh = np.zeros((480,640,1))
181 pygame.init()
182 # GPIOs #
183 import RPi.GPIO as GPIO
184
185 GPIO.setmode(GPIO.BCM)
186 GPIO.setup(26,GPIO.OUT)
187 GPIO.setup(19,GPIO.OUT)
188 GPIO.setup(21,GPIO.OUT)
189 GPIO.setup(20,GPIO.OUT)
190
191 GPIO.output(26,0) #RA#
192 GPIO.output(19,0) #DEC-
193 GPIO.output(20,0) #RA+
194 GPIO.output(21,0) #DEC+
195
196 def closeGPIO():
197    GPIO.output(26,0) #RA-
198    GPIO.output(19,0) #DEC-
199    GPIO.output(20,0) #RA+
200    GPIO.output(21,0) #DEC+
201    GPIO.cleanup()
202
203 # camera initial settings #
204
205 camera.set_control_value(asi.ASI_GAIN, CamGain)
206 camera.set_control_value(asi.ASI_EXPOSURE, 100)
207 camera.set_control_value(asi.ASI_WB_B, 99)
```

```
208 camera.set_control_value(asi.ASI_WB_R, 75)
209 camera.set_control_value(asi.ASI_GAMMA, 50)
210 camera.set_control_value(asi.ASI_BRIGHTNESS, 150)
211 camera.set_control_value(asi.ASI_FLIP, 0)
212 camera.set_control_value(asi.ASI_BANDWIDTHOVERLOAD, camera.
    ↪ get_controls()['BandWidth']['MinValue'])
213 camera.set_roi_format(640,480,1,0)
214 camera.auto_wb()
215 camera.start_video_capture()
216
217 ###############################################
218
219 pygame.display.set_caption("Freded window")
220 screen = pygame.display.set_mode([800,600])
221 pygame.font.init()
222 myfont2 = pygame.font.SysFont('Arial',15)
223 #Buttons#
224
225 button_calib = pygame.image.load("buttons/button_calib.png"
    ↪ )
226 button_capimg = pygame.image.load("buttons/button_capim.png
    ↪ ")
227 button_expminus = pygame.image.load("buttons/button_emin.
    ↪ png")
228 button_expplus = pygame.image.load("buttons/button_eplus.
    ↪ png")
229 button_exit = pygame.image.load("buttons/button_exit.png")
230 button_gminus = pygame.image.load("buttons/button_gmin.png"
    ↪ )
231 button_gplus = pygame.image.load("buttons/button_gplus.png"
    ↪ )
232 button_search = pygame.image.load("buttons/button_search.
    ↪ png")
233 button_select = pygame.image.load("buttons/button_select.
    ↪ png")
234 button_track = pygame.image.load("buttons/button_track.png"
    ↪ )
235
236 # Display Text and Buttons Func #
```

```
237 def ShowText():
238     screen.blit(SensorTempText,(20,40))
239     screen.blit(StarsFoundText,(20,60))
240     screen.blit(ExpTimeText,(20,80))
241     screen.blit(CamGainText,(20,100))
242 
243 
244     screen.blit(button_exit,(651,5))
245     screen.blit(button_select,(650,50))
246     screen.blit(button_track,(650,95))
247     screen.blit(button_capimg,(650,140))
248     screen.blit(button_expplus,(650,185))
249     screen.blit(button_expminus,(650,230))
250     screen.blit(button_gplus,(650,275))
251     screen.blit(button_gminus,(650,320))
252 # Check Mouse Position for Button #
253 def CheckMousePos():
254 
255     global SearchIndikator
256     xmouse,ymouse = pygame.mouse.get_pos()
257 
258     if(ymouse < 50):
259         ExitButton()
260 
261     if(ymouse > 51 and ymouse < 91 and xmouse > 650 and
        ↪ xmouse < 800):
262         SelectButton()
263 
264 
265     if(ymouse > 96 and ymouse < 135 and xmouse > 650 and
        ↪ xmouse < 800):
266         TrackButton()
267 
268     if(ymouse >141 and ymouse < 181 and xmouse > 650 and
        ↪ xmouse < 800):
269         CapImgButton()
270 
271     if(ymouse >185 and ymouse < 225 and xmouse > 650 and
        ↪ xmouse < 800):
```

```
272         ExpPlusButton()
273
274     if(ymouse >231 and ymouse < 269 and xmouse > 650 and
            ↪ xmouse < 800):
275         ExpMinusButton()
276
277     if(ymouse >277 and ymouse < 313 and xmouse > 650 and
            ↪ xmouse < 800):
278         GainPlusButton()
279
280     if(ymouse >320 and ymouse < 360 and xmouse > 650 and
            ↪ xmouse < 800):
281         GainMinusButton()
282
283 # Button Control Functions #
284
285 def ExitButton():
286     print('Exit')
287     print(pygame.mouse.get_pos())
288     pygame.quit();
289     closeGPIO()
290
291 def SelectButton():
292     Search()
293     global StarTable
294     print("Select")
295     StarTable = getInitialStarTable()
296     print(StarTable)
297
298 def TrackButton():
299     print("Track")
300     global StarTable
301     StarTable = StarMatching(StarTable)
302     print(StarTable)
303
304 def CapImgButton():
305     print('CapImg')
306     im = Image.fromarray(tframe)
```

```
307     im.save(strftime("%Y-%m-%d-%H:%M:%S",gmtime())+"img.jpeg
            ↪ ")
308     print("CapImg")
309
310 def ExpPlusButton():
311     global ExpTime
312     ExpTime = int(ExpTime * 1.3)
313     print("E+")
314
315 def ExpMinusButton():
316     global ExpTime
317     ExpTime = int(ExpTime * 0.75)
318     print("E-")
319
320 def GainPlusButton():
321     global CamGain
322     if(CamGain == 150):
323         CamGain = 150
324     else:
325         CamGain+=10
326     print("G+")
327
328 def GainMinusButton():
329     global CamGain
330     if(CamGain == 10):
331         CamGain = 10
332     else:
333         CamGain-=10
334     print("G-")
335
336
337 # Init Controls #
338 SearchIndikator = 0
339
340 ############################
341
342 def Search():
343     global thresh
344     global TempCoordinates
```

```
345     global SearchIndikator
346     SearchIndikator=(SearchIndikator+1)%2
347 
348     if(SearchIndikator == 1):
349        TempCoordinates = np.empty((0,2),float)
350        Threshold = 120
351        blurred = cv2.blur(tframe,(2,2))
352        #blurred = cv2.bilateralFilter(tframe,11,17,17)
353        reth,thresh = cv2.threshold(blurred,Threshold,255,0)
354        image, cnts, hierarchy = cv2.findContours(thresh, cv2
              ↪ .RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
355 
356        for c in cnts:
357           M = cv2.moments(c)
358           if(M['m10']!=0):
359              cX = int(M['m10']/M['m00'])
360              cY = int(M['m01']/M['m00'])
361              TempCoordinates = np.append(TempCoordinates,np.
                    ↪ array([[cX,cY]]),axis=0)
362 
363 
364 # Draw Stars #
365 
366 def drawStars():
367     if(len(TempCoordinates) > 0):
368        for x,y in TempCoordinates:
369           pygame.draw.circle(screen,(255,0,0),(int(x),int(y)
                 ↪ ),10)
370 
371 # Capture Fred #
372 
373 class getFrame(threading.Thread):
374     def __init__ (self):
375        threading.Thread.__init__(self)
376        self.stopThread = False
377 
378     def run(self):
379        while True:
380           global tframe
```

```
381             global ExpTime
382             camera.set_control_value(asi.ASI_EXPOSURE, ExpTime
                    ↪ )
383             camera.set_control_value(asi.ASI_GAIN, CamGain)
384             tframe = camera.capture_video_frame()
385             if self.stopThread == True:
386                 break
387     def stopThread(self, stopThread):
388         self.stopThread = stopThread
389
390 # Convert frame to gray for surface#
391 # pygame want's it this way #
392
393 def gray (im):
394     im = 255* (im/im.max())
395     w,h=im.shape
396     ret = np.empty((w,h,3),dtype=np.uint8)
397     ret[:,:,2] = ret[:,:,1] = ret[:,:,0] = im
398     return ret
399
400 # Start Fred #
401
402 t1 = getFrame()
403 t1.start()
404
405 ##############################
406
407 # warm up camera before start #
408
409 time.sleep(1)
410 i=0
411 while(i==5):
412     tframe = camera.capture()
413     i+=1
414
415 ############################
416 # here begins the loop of the main program #
417
418 try:
```

```
419     while True:
420         SensorTemp = (asi.ASI_TEMPERATURE * 10 - 32)/1.8 #
              ↪ convert fahrenheit to real unit
421         SensorTemp = int(SensorTemp)
422         SensorTempText = myfont2.render("Sensor Temp: " + str
              ↪ (SensorTemp) + " C",False,(255,0,0))
423         ExpTimems = ExpTime / 1000
424         ExpTimeText = myfont2.render("Exp Time: " + str(
              ↪ ExpTimems) + " ms",False,(255,0,0))
425         StarsFoundText = myfont2.render("Stars: " + str(len(
              ↪ TempCoordinates)),False,(255,0,0))
426         CamGainText = myfont2.render("Gain: " + str(CamGain),
              ↪ False,(255,0,0))
427         screen.fill([0,0,0])
428
429         if(SearchIndikator == 0):
430             frame = gray(tframe)
431         if(SearchIndikator == 1):
432             Search()
433             frame = gray(thresh)
434
435         frame = np.rot90(frame)
436         frame = pygame.surfarray.make_surface(frame)
437         screen.blit(frame,(5,5))
438
439
440         #print(pygame.mouse.get_pos())
441
442         #Always draw button before text ->overlay
443         #DrawButtons()
444         ShowText()
445         if(SearchIndikator == 1):
446             drawStars()
447
448         pygame.display.update()
449
450         for event in pygame.event.get():
451             if event.type == KEYDOWN:
452                 sys.exit(0)
```

```
453         if event.type == pygame.MOUSEBUTTONDOWN:
454             CheckMousePos()
455 
456 except (KeyboardInterrupt,SystemExit):
457     pygame.quit()
458     cv2.destroyAllWindows()
```

Bibliography

[1] Gregory D Hager, Markus Vincze, and IEEE Robotics and Automation Society. *Robust vision for vision-based control of motion*. 2000.

[2] Kari Pulli, Anatoly Baksheev, Kirill Kornyakov, and Victor Eruhimov. Real-time computer vision with OpenCV. *Communications of the ACM*, 55(6):61, jun 2012.

[3] Richard Szeliski. *Computer Vision*. Texts in Computer Science. Springer London, London, 2011.

[4] Hannu Karttunen, Pekka Kröger, Heikki Oja, Markku Poutanen, and Karl Johan Donner, editors. *Fundamental Astronomy*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[5] W.M.Smart. *Textbook on Spherical Astronomy*. Press Syndicate of the University of Cambridge, 1931.

[6] TS Optics. PhotoLine 102mm Duplet Apo Manual.

[7] SkyWatcher. EQ6 Mount Manual, 2016.

[8] Alex Zelinsky. Learning OpenCV—Computer Vision with the OpenCV Library (Bradski, G.R. et al.; 2008)[On the Shelf]. *IEEE Robotics & Automation Magazine*, 16(3):549, sep 2009.

[9] Michael Kofler. *Linux 2013*. Addison-Wesley Verlag, 2012.

[10] ZWO Asi. 120MM-S Mono Software Developement Kit Manual. 2017.

[11] Mark Pilgrim. *Python 3 - Intensivkurs*. Xpert.press. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[12] Al Sweigart. *Making Games with Python & Pygame*. 1 edition, 2012.

[13] Ulrich Tietze and Christoph Schenk. *Halbleiter-Schaltungstechnik*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1993.

[14] Simon Monk. *Make Your Own PCBs with Eagle: From Schematic Designs to Finished Boards*. Mcgraw Hill Book Co, 2017.

[15] Went Tzu Lin Hsieh Tsung Han. *DesignSpark PCB Guidebook*.

[16] Toshimi Taki. *Matrix Method for Coordinates Transformation*. 2002.

[17] Manfred Husty, Adolf Karger, Hans Sachs, and Waldemar Steinhilper. *Kinematik und Robotik*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[18] Linda G. Haralick, Robert M., Shapiro. *Computer and Robot Vision*. Prentice Hall, 2002.

[19] Gary Bradski, Adrian Kaehler, and Gary Bradski. *Learning OpenCV*, volume 53. 2013.

[20] Robert F. Fischer. *Optical System Design*. Mcgraw Hill Book Co, 2 edition, 2008.

[21] M. Bass and Optical Society of America. *Handbook of Optics: Fundamentals, techniques, and design*. McGraw-Hill, 1994.

[22] Joseph Howse. *OpenCV Computer Vision with Python*. Packt Publishing, 2013.

[23] Travis E. Oliphant. Python for Scientific Computing. *Computing in Science & Engineering*, 9(3):10–20, 2007.

[24] F. Chaumette, P. Rives, and B. Espiau. Positioning of a robot with respect to an object, tracking it and estimating its velocity by visual servoing. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2248–2253. IEEE Comput. Soc. Press.