ACM 9000
Automated Camera Man
Automatiserad Kameraman

GUSTAV BURMAN

SIMON ERLANDSSON
Automated Camera Man
Automatiserad Kameraman

GUSTAV BURMAN
SIMON ERLANDSSON

Bachelor’s Thesis at ITM
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA-ITM-EX 2018:68
Abstract

Today’s digital society is changing the way we learn and educate drastically. Education is being digitalized with the use of online courses and digital lectures. This bachelor thesis solves the problem of how to be able to record a lecture without a camera operator, an Automated Camera Man (ACM), for easier production of high quality education material. It was achieved with a modularized design process, practical testing and a scientific approach. The Automated Camera Man can be placed in the rear of the lecture hall to record or stream the content while it actively adjusts itself and its direction towards the lecturer using image processing and analysis.

Keywords: mechatronics, robot, autonomous, tracking, filtering
Referat


Nyckelord: mekantronik, robot, automatiserad, spårning, filtrering
Acknowledgements

We would like to thank our supervisor Nihad Subasic for his support and Staffan Qvarnström for helping us with finding the right components. We would also like to send a huge thanks to the assistants and fellow students for all advice and constructive discussions.
# Contents

1 Introduction 1
   1.1 Background ................................................. 1
   1.2 Purpose .................................................. 1
   1.3 Scope .................................................... 2
   1.4 Method .................................................... 2

2 Theory 3
   2.1 Control system theory ...................................... 3
   2.2 DC Motors .................................................. 3
   2.3 H-Bridge ................................................... 4
   2.4 Microcontrollers .......................................... 4
   2.5 Image format .............................................. 5
   2.6 Filtering .................................................. 5

3 Demonstrator 7
   3.1 Choosing a tracking system ................................. 7
       3.1.1 Ultrasound ............................................ 7
       3.1.2 Electromagnetic waves ............................... 8
       3.1.3 Image processing .................................... 8
   3.2 Hardware .................................................. 8
       3.2.1 Microcontroller ....................................... 10
       3.2.2 Raspberry Pi Camera ................................ 10
       3.2.3 Stepper motor ....................................... 10
       3.2.4 Dual H-Bridge Circuit Board ........................ 10
   3.3 Software .................................................. 11
       3.3.1 Retrieving data from the camera .................... 11
       3.3.2 Color Filter .......................................... 11
       3.3.3 Position determining algorithm .................... 12
       3.3.4 Movement algorithm .................................. 12

4 Results 13
   4.1 Execution .................................................. 13
   4.2 Distance Study ............................................ 14
List of Figures

2.1 Illustration of how an H-bridge works [3] ........................................ 4
2.2 The Red-Green-Blue (RGB) matrix and its three layers [7] .............. 5

3.1 Picture of the complete system and the sun-and-planet gear, taken with
a Samsung Galaxy S8 ................................................................. 9
3.2 The hardware components and their connections with each other, created
with the online software draw.io ................................................ 9
3.3 Visualization of how the filtering algorithm works, created in the soft-
wares MATLAB and Microsoft Powerpoint ................................. 11

A.1 Flowchart of the main module, created with the online software draw.io 24
A.2 Flowchart of the image module, created with the online software draw.io 25
A.3 Flowchart of the motor module, created with the online software draw.io 26
List of Tables

3.1 Reference color intervals ........................................ 12
4.1 Distance and time study tests ................................... 14
4.2 Distance study test result extracts .............................. 15
4.3 Time study test result extracts ................................. 16
Abbreviations

ACM  Automated Camera Man.
CPU  Central Processing Unit.
DC   Direct Current.
FOV  Field of View.
GPIO General Purpose Input and Output Pins.
OS   Operating System.
RAM  Random Access Memory.
RGB  Red-Green-Blue.
Chapter 1

Introduction

This is a bachelor thesis report in mechatronics at KTH aiming to build a smart camera stand that can automatically readjust itself to aim at a person or an object. The project name is Automated Camera Man, ACM.

1.1 Background

Camera stand operators are expensive and in most video recording situations are not a feasible option. An automated camera stand could be the best economic option but could also allow users with a restricted budget and without skilled personnel to create high quality recordings. Imagine a lecturer holding a lecture while recording it. By changing from a stationary camera to one that can automatically adjust itself towards the lecturer, they could make the video more engaging. On a bigger scale an ACM could be implemented when recording concerts or sport events like soccer. Several cameras following each band member, or each soccer player, without any operating personnel. In fact, such a system could be used in a wide variety of implementations.

1.2 Purpose

The purpose of this project is to construct an ACM primary for university lectures. In doing so, exploring the possibilities of using relatively cheap components and limited programming skills to track and follow an object at a distance of 15 meters. The target of this thesis is to answer the following questions:

- What kind of tracking system is most suitable when building an ACM?

- How can a mechatronical construction and its control system be designed for an ACM?
1.3 Scope

On the condition of a workforce of two students working part time during one university semester. The main scope is to build a mechanical arm that uses an appropriate tracking system to turn itself towards a person or an object. The tracking system must fulfill as many of the following requirements as possible.

- Be able to tell the direction of a person or object 15 meters away with enough precision.
- Adjust the camera towards said direction in a smooth way.
- Should be discreet, no disturbing noises or lights.

1.4 Method

The project was divided into subproblems, or modules, which were worked on separately. The project started with a theoretical study and investigation on which tracking system theoretically fits the requirements best. The most suitable tracking system was then physically implemented by building a prototype.

The prototype consists of three independently developed modules. A control system module, that turns an arm according to given directions on a horizontal axis. A tracking module that finds the position of an object. Lastly, a mechanical construction module.
Chapter 2

Theory

2.1 Control system theory

A control system is a system designed to regulate a physical quantity based on a set-point. A system is stable if it sets to a position within a real time span and unstable if it never settles. An unstable system can escalate indefinitely (within physical limitations), which is undesirable. A set point is the desired state, while a real point is the system’s real state. The difference between the set point and the real point is the control error [1].

2.2 DC Motors

Direct current (DC) motors are machines that transform energy from electrical direct current to mechanical motion. There exist many different types of DC motors, but they are mostly based on the same basic concepts. They consist of two parts, a rotating rotor and a stationary stator. A magnetic field is created when running an electrical current through a coil. The magnetic field attracts or repels against another magnetic field, usually originating from a permanent magnet. This generates a force that turns the rotor.

The stepper motor is a member of the DC motor family but compared to ordinary DC motors, stepper motors have a more complex method of movement. As the name suggests it turns in steps, each step is initiated with a special sequence of DC inputs.

A control system regulating a step motor will always be stable in control theory terms. That is because a stepper motor moves in defined steps. As different states require different inputs to step, a step-controller is necessary. The step controller consists of a regulating microcontroller, transistor circuits and H-bridges. The step count, that is steps per revolution, is usually in the 4 to 400 range and is based on each motors built in transmission. Step motors usually use a sun-and-planet gear
to acquire the required step-length [2].

2.3 H-Bridge

When regulating a DC motor the current should be able to switch direction, so that
the motor can rotate both clockwise and counterclockwise. This can be achieved
with an H-bridge. Figure 2.1 displays the different ways current can go through a
motor when using an H-bridge. The H-bridge consists of four switches which are
connected in pair to two logical gates. By activating the switches pair wise with
logical gates, the direction of the electricity can be controlled. However, there will
be a short-circuit if both gates are turned on simultaneously. This is countered by
some additional safety circuitry.

![H bridge topology](image)

**Figure 2.1.** Illustration of how an H-bridge works [3]

2.4 Microcontrollers

Microcontrollers are small computers designed for smaller and more simple tasks
than the ordinary desktop computers or smart phones. They consist of a central
processor unit (CPU), random access memory (RAM), general purpose input and
output pins (GPIO) and other electrical components. The GPIO-pins are used for
electrical input or output signals. An operating system (OS) is the interface be-
tween the human user and the computer. Depending on the type of microcontroller,
OS’s can be used to make them more user friendly [4].

Some microcontrollers supports multi-threaded programming. That is, a system
architecture that supports the assignment of threads to different tasks. If correctly
implemented it increases algorithm efficiency compared to single-threading algo-
rithms. In a way, multi-threading is equivalent to a computer multi-tasking [5].
2.5 Image format

A camera captures or records images. It is an optical instrument and is therefore susceptible to all kinds of optical phenomena such as difference in brightness. This difference will also affect the recorded data. Most cameras use an image sensor and an integrated circuit to pre-process the image into a manageable data format. The digital camera output is often in a Bayer pattern. The Bayer pattern is a matrix that consists of groups of two green, one blue and one red pixels. One of the most common raw image format is the Red-Green-Blue-matrix (RGB) shown in Figure 2.2. It consists of three matrix layers, one for each color, and can be created from the Bayer pattern. A pixel in the RGB matrix is created by merging a Bayer pattern group, and putting its color value in the corresponding layer in the matrix. The green value in the RGB matrix is the mean of the two corresponding green values in the Bayer pattern [6].

![Figure 2.2. The RGB matrix and its three layers](image)

2.6 Filtering

There are many kinds of filters used in computer science. The basic principle is to alter a set of data for further processing and data analysis. The process is similar to a physical filter, where only the wanted substance is let through. An example of a digital filter would be to only retrieve one color from an RGB matrix [8].
Chapter 3

Demonstrator

3.1 Choosing a tracking system

There exists a wide range of possible tracking systems that could in theory work. Sound or electromagnetic waves (light) can be used with some kind of transmitter and receiver. In Section 1.3 it was stated that there should be no disturbing noises or lights. This means that sound in the hearing range 20 Hz to 20 kHz and light in the visible length 380-750 nm should therefore be avoided [9]. Therefore, the tracker should only transmit in the ultrasound, infrasound or non-visible light spectrum. While both are plausible options they come with negative aspects which are discussed in the following sections.

3.1.1 Ultrasound

While ultrasound is not audible by human ears, many animals like dogs can hear frequencies up to 45 kHz. In an environment with dogs an ultrasound system is far from optimal. An earlier Bachelor Thesis [10] showed that ultrasound might not be able to reach the 15 meters target distance of this project. According to the thesis the maximum reach is approximately 250 cm, with the components available. Ultrasound is mostly used to detect objects on small distances up to 4 meters. However, in those implementations, the sound is transmitted and received from the same microchip.

The sound bounces on an object and thus loses some of its energy. A separated system for the transmitter and receiver could in theory reach longer distances. But is it enough to compute the angle of the transmitter? It is important to take into consideration how the transmitter and receiver are set up. The person wearing the speaker could turn around which would turn the transmitter away from the receiver and ruin the tracking. Its hard to build a transmitter that sends ultrasound in all directions because ultrasound is transmitted in a very narrow angle.
3.1.2 Electromagnetic waves

Just like ultrasound, light has to be emitted from something. But in contrast, light is omnidirectional. Think for example a helmet with a light bulb on top of it. Such a construction could definitely be implement, but at the cost of being rather inconvenient. Just imagine walking around with a large light-bulb-hat.

Using light also requires more sensors to get a sense of direction. Compared to a sonar tracking system, which could be implemented by using three microphones and then triangulating the signal. Photo-resistors can only register light from one direction and triangulation is not an option due to the speed of light. It would require accurate instruments far too expensive. Therefore, an array of photo-resistors in many different directions would be required to pinpoint the direction of the light source. Then, what is the difference from using a camera? A modern camera is just a complicated collection of densely packed photo-resistors in a Bayer pattern.

3.1.3 Image processing

A camera collects a lot of data that has to be handled in some way. This means that the microcontroller processing the pictures has to be powerful enough for the computations. By using a camera, it would be possible to track objects using only a receiver. For instance, by using algorithms for color or movement detection. The transmitter would in this case be a distinct color or shape. This has been done before and requires a larger focus on computer software and algorithms. However, it decreases the number of hardware components needed. In addition, the tracking range of such a system should only be limited by the resolution of the camera and how far it can see.

Following this analysis, it was decided that a camera and the image processing approach was the most suitable tracking system.

3.2 Hardware

The final construction, shown in Figure 3.1, is an assembly of all the modules, brought together with 3D-printed plastic parts. To acquire smaller steps, smoother motion and larger torque, a sun-and-planet gear was used. The transmission has a gear ratio of 5.

To avoid tangling the cables when the system rotates, it was decided that all the systems components, camera, Raspberry Pi, H-bridge and motor, should be rotating as well. By such an arrangement, there was only two cables that had to be connected outside of the rotating part.

In addition, a small base was built to hold together and organize all of the rotating parts.
3.2. HARDWARE

Figure 3.1. Picture of the complete system and the sun-and-planet gear, taken with a Samsung Galaxy S8

The prototype can be divided into multiple hardware modules, each performing a different task. The brain of the whole operation is the Raspberry Pi, which does all the computing. The computing includes both image processing and giving orders to the motor. A camera is connected to the Raspberry Pi and a dual H-Bridge circuit works as a link between the Raspberry Pi’s GPIO-pins and a stepper motor. The stepper motor acts on the transmission and turns the camera stand according to instructions. Two different voltage supplies are necessary to run the system. All of this is displayed in Figure 3.2.

Figure 3.2. The hardware components and their connections with each other, created with the online software draw.io
3.2.1 Microcontroller

When choosing the right microcontroller two types were compared against each other, the Raspberry Pi and the Arduino UNO. The Raspberry Pi was deemed most suitable for the intended task due to the amount of processing power needed for capturing and analyzing pictures. The Raspberry Pi 3 has superior computational power compared to the Arduino UNO. The Raspberry Pi 3 uses a Quad Core CPU, which allows multiple threads to be run simultaneously [11]. This makes it possible for multiple tasks to run at the same time, which means that some calculations can be computed faster and in parallel. Using the Raspberry Pi, motor regulating and image processing modules could be programmed to run on different threads, running in practice simultaneously. It should also be possible to increase the output signal refresh rate and therefore, increase the potential accuracy of the ACM. In addition, the smoothness of the camera movement also increases with higher refresh rate.

3.2.2 Raspberry Pi Camera

The Raspberry Camera V2 is specifically designed to operate with the Raspberry Pi and is fully supported by Raspbian which is the creators recommended OS. Therefore, the Raspberry Camera V2 was a natural choice together with the Raspberry Pi. The camera consists of a Sony IMX219 image sensor with an attached focus lens. In total it only weights 3 grams which makes it perfect for light weight applications [12].

3.2.3 Stepper motor

For this project a generic step motor was chosen with a power rating of roughly 3 watts. The control system becomes much simpler when having a step motor compared to an ordinary DC motor. This is because, according to control system theory, there can be no instability since stepper motors turn in exact known steps. The movement is exactly decided by the microcontroller. The step motor has a step count of 200. This means that each step is equal to 1.8 degrees. Such a large step would be noticeable and disturb the recording. To address this problem, a transmission was designed with a gear ratio of 5. Thus, a total step count of 1000 (0.36 degrees per step) was achieved.

3.2.4 Dual H-Bridge Circuit Board

To run the step motor a L298N-microchip which contains two H-bridges was used. The circuitry is based on a schematic circuit found in the L298N product sheet [13]. It was modified to fit a microcontroller with four steering GPIO-pins as regulators and was printed/milled out on a circuit board and soldered together.
3.3 Software

The software was divided into three modules, which run in parallel with multithreading. These modules are a main module, a motor module and an image module. The primary purpose of the main module is to initiate the other modules and connect them together. The image module takes an image and filters it with respect to the color green. The filtered image is then passed through a function which calculates the position of the green object. This function determines how many pixels the green object is from the center of the picture frame. The pixel value is then used to calculate a control error angle based on the field of view (FOV) of the camera. The error angle is then used by the motor module to move the step motor. The flowcharts of all modules can be found in Appendix A, and the complete code in Appendix B.

3.3.1 Retrieving data from the camera

When connecting the camera to the Raspberry Pi, images can be taken using Python 3’s Picamera library. It has a wide range of camera settings that has to be dialed to the correct setting state. The aimed state is one with fast image processing, an adequate image resolution and a wide camera angle. Since a color tracing algorithm is used, the images have to be color-images. The camera output must be an RGB matrix because the algorithms are designed for the RGB format. The values in the RGB-matrices are in this case 8 bit integers, that is a number between 0 and 255. A value of 0 equals no intensity and a value of 255 equals full intensity.

3.3.2 Color Filter

The green filter checks every pixel in the image and compares the pixel to the reference RGB-values. For the pixel to pass through the filter, it first of all has to be above a certain magnitude in the green spectrum. Secondly, green has to be the most dominant color by a certain factor. This process is illustrated in Figure 3.3.

Figure 3.3. Visualization of how the filtering algorithm works, created in the softwares MATLAB and Microsoft Powerpoint
The reference value or reference intervals were chosen by looking at a color spectrum and selecting the intervals which subjectively looked green in combination with testing and adjusting. The testing resulted in intervals of red, green and blue which serves as a reference for the computer to know which colors are green.

<table>
<thead>
<tr>
<th>Color</th>
<th>Comparison</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>&gt;</td>
<td>120</td>
</tr>
<tr>
<td>Green</td>
<td>&gt;</td>
<td>Red</td>
</tr>
<tr>
<td>Green</td>
<td>&gt;</td>
<td>Blue</td>
</tr>
<tr>
<td>Red</td>
<td>&lt;</td>
<td>90</td>
</tr>
<tr>
<td>Blue</td>
<td>&lt;</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 3.1. Reference color intervals

Python 3’s built-in operators and loops are in most cases really slow. Logical operators, boolean matrix algebra and boolean masks were used to dramatically speed up the filtering algorithm. It was implemented using the open source Python extension pack NumPy. The logical operators in NumPy are much closer to the machine than ordinary Python [14] and therefore significantly shorten the processing time.

3.3.3 Position determining algorithm

In order to acquire a single point value or coordinate from an array, a weighted mean was used. This weighted mean represents the middle of the green area, or the center of green color. Again, usage of Python’s loops took too much time. Instead, the weighted mean was calculated algebraically using NumPy’s built in matrix summation functions.

3.3.4 Movement algorithm

As pointed out in the Section 1.3, the necessary preference of movement in an ACM is smooth and with no sudden motions. The recordings by a camera mounted upon the ACM should not be shaky. The most basic way to turn a stepper motor is to step the number of steps at a constant speed. The positive with this is that it is easy to implement. The negative is it can be shaky, especially when jumping back and forth on small angles. A solution to the shaking is smart software, no movement should occur if the angle is smaller than ten steps. It is not a perfect solution, but it stops the motor from twitching.
Chapter 4

Results

In this chapter the results from studies made on the demonstrator will be presented. A study was made on different distances and camera resolutions, as well as measuring the processing time for the image module on different resolutions. These studies were conducted to answer if the demonstrator had reached the scope of the project.

- Be able to tell the direction of a person or object 15 meters away with enough precision.
- Adjust the camera towards said direction in a smooth way.

The next three sections will display and discuss the results of the study and how it was conducted.

4.1 Execution

Tests were performed in three different environments and different light conditions with the following tools and settings:

- Green polyester cloth with two sizes: 24x27 cm and 54x79 cm
- Resolutions: 112x80, 224x160, 448x320 and one test at 1920x1088
- Distances ranging from 2 to 35 m

In whole, 17 tests were conducted under different circumstances as seen in Table 4.1. Only the later tests (11-17) will be presented in the following two sections. Tests 1 to 3 were performed outside in sunlight and did not find the position at all. Tests 4 to 9 were performed indoors with a longest distance of 12 m and uneven light. On two of the tests the cloth was lit up with a relatively dim flashlight, but the light was still to uneven. Because of this, it was not possible to measure a connection between green color detection and distance reliably. Test 10 to 17 was performed in the same room, with a distance much longer than 15 m and a strong lamp. On test 10 the lamp was too close to the cloth and it appeared white. In test 10, no position
was found.

The tests were conducted by taking images at a distance of 2 meters, 4 m and then increased by 1 m until the ACM could not detect anything. To make sure it was not a temporary thing, the light source was moved a bit and the cloth shaken, followed by a new trial. If the ACM once again found a position the distance was increased. However, if the ACM did not register a new position after a number of trials the test was ended and the distance noted. To make sure there were no false detections, a couple of purposefully empty pictures were taken. No green interference was detected during tests 11 to 17.

As all the images were taken and processed, the program timed the image processing module.

<table>
<thead>
<tr>
<th>Number [-]</th>
<th>Resolution [Pixels]</th>
<th>Environment [-]</th>
<th>Cloth size [-]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>112x80</td>
<td>Outside</td>
<td>Small</td>
</tr>
<tr>
<td>2</td>
<td>112x80</td>
<td>Outside</td>
<td>Large</td>
</tr>
<tr>
<td>3</td>
<td>224x160</td>
<td>Outside</td>
<td>Large</td>
</tr>
<tr>
<td>4</td>
<td>112x80</td>
<td>Inside</td>
<td>Small</td>
</tr>
<tr>
<td>5</td>
<td>224x160</td>
<td>Inside</td>
<td>Small</td>
</tr>
<tr>
<td>6</td>
<td>448x320</td>
<td>Inside</td>
<td>Small</td>
</tr>
<tr>
<td>7</td>
<td>448x320</td>
<td>Inside</td>
<td>Large</td>
</tr>
<tr>
<td>8</td>
<td>448x320</td>
<td>Inside with extra light</td>
<td>Large</td>
</tr>
<tr>
<td>9</td>
<td>448x320</td>
<td>Inside with extra light</td>
<td>Small</td>
</tr>
<tr>
<td>10</td>
<td>112x80</td>
<td>Inside with extra light</td>
<td>Small</td>
</tr>
<tr>
<td>11</td>
<td>112x80</td>
<td>Inside with extra light</td>
<td>Small</td>
</tr>
<tr>
<td>12</td>
<td>224x160</td>
<td>Inside with extra light</td>
<td>Small</td>
</tr>
<tr>
<td>13</td>
<td>448x320</td>
<td>Inside with extra light</td>
<td>Small</td>
</tr>
<tr>
<td>14</td>
<td>112x80</td>
<td>Inside with extra light</td>
<td>Large</td>
</tr>
<tr>
<td>15</td>
<td>224x160</td>
<td>Inside with extra light</td>
<td>Large</td>
</tr>
<tr>
<td>16</td>
<td>448x320</td>
<td>Inside with extra light</td>
<td>Large</td>
</tr>
<tr>
<td>17</td>
<td>1920x1088</td>
<td>Inside with extra light</td>
<td>Large</td>
</tr>
</tbody>
</table>

Table 4.1. Distance and time study tests

### 4.2 Distance Study

Many of the tests failed to find any position. The problem was the light combined with the camera settings. If the light levels were wrong, green would not be perceived as green by the camera. This was a problem both in very lit locations (in
4.3. TIME STUDY

<table>
<thead>
<tr>
<th>Resolution [Pixels]</th>
<th>Cloth size [-]</th>
<th>Max distance[m]</th>
</tr>
</thead>
<tbody>
<tr>
<td>112x80</td>
<td>Small</td>
<td>5</td>
</tr>
<tr>
<td>224x160</td>
<td>Small</td>
<td>11</td>
</tr>
<tr>
<td>448x320</td>
<td>Small</td>
<td>10</td>
</tr>
<tr>
<td>112x80</td>
<td>Large</td>
<td>13</td>
</tr>
<tr>
<td>224x160</td>
<td>Large</td>
<td>21</td>
</tr>
<tr>
<td>448x320</td>
<td>Large</td>
<td>22</td>
</tr>
<tr>
<td>1920x1088</td>
<td>Large</td>
<td>35</td>
</tr>
</tbody>
</table>

Table 4.2. Distance study test result extracts

outdoors sunlight) and in not very lit places (indoors) or rooms with large light contrast. For the ACM to be able to detect an object of the correct green, the object had to be adequately lit.

As seen in Table 4.2, the maximum distance the ACM was able to track depended greatly upon the size of the green cloth, strength of the source and camera resolution. The test results showed that the ACM system is indeed capable of tracking objects at a distance of 15 meters given a strong enough source in a suitable light environment.

4.3 Time Study

The smoothness of the systems movement can be tested in two ways, either by testing the motor module or the image module. In the study, only the image modules calculation time was tested. The bottleneck of the system is not the motor but how fast new position values can be found. The motor works as fast as physically possible, and with the transmission its movement was designed to work evenly. No empirical study was conducted to prove that statement. However, it was practically observed. The faster new and reliable position values can be found the more gently and precisely the ACM can turn.

The time studies that were conducted, with the result presented in Table 4.3, showed that with higher resolution the image module processing time increases. There were not enough data points to create any particular model.
### Table 4.3. Time study test result extracts

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Cloth size</th>
<th>Mean time [s]</th>
<th>Mean variance [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td>112x80</td>
<td>Small</td>
<td>0.0711</td>
<td>7.8208e-05</td>
</tr>
<tr>
<td>224x160</td>
<td>Small</td>
<td>0.0874</td>
<td>8.3442e-05</td>
</tr>
<tr>
<td>448x320</td>
<td>Small</td>
<td>0.1607</td>
<td>3.8387e-04</td>
</tr>
<tr>
<td>112x80</td>
<td>Large</td>
<td>0.0700</td>
<td>7.4560e-05</td>
</tr>
<tr>
<td>224x160</td>
<td>Large</td>
<td>0.0882</td>
<td>1.0984e-04</td>
</tr>
<tr>
<td>448x320</td>
<td>Large</td>
<td>0.1478</td>
<td>1.4775e-04</td>
</tr>
<tr>
<td>1920x1088</td>
<td>Large</td>
<td>1.4600</td>
<td>0.0055</td>
</tr>
</tbody>
</table>
Chapter 5

Conclusion and Discussion

5.1 Discussion

Image processing is the only discussed tracking system that satisfies the research questions. By using image processing there are multiple ways to detect an object or an image. That is, through contours, through movement and through color. All of these options come with advantages and disadvantages.

Identifying a specific contour is only usable when having a distinct and static contour to track. In addition, the contour has to be identifiable regardless of where the person being tracked is facing. This means that contour tracking should not be implemented as a sole tracking method.

The same goes for motion tracking. It should not be used as the primary tracking system, but rather as a complementary to another tracking system. This is because of two main reasons. Firstly, motion is not distinct and there is no guarantee that the lecturer will move at all times. Secondly, a moving ACM system introduces new problems mainly regarding precision and image blurriness.

Image color processing was found to be the most suitable object tracking method because it could be used standalone. There was a couple of ways it could have been implemented, through more advanced recognition software, pixel clusters or filters.

Filters are easy to build, implement and tweak and were therefore the ideal option. But there are some inherent problems using this method. One of these is the problem of choosing a single point to track out of the many pixels of the green source. Another complication could also arise if two or more separate green sources appear.

Both of these problems were solved using a weighted mean which ensured that the point to track always ends up in the middle of the green object. In the case of
multiple separate green objects, the point tracked will end up somewhere in between
the sources. This is perfectly acceptable and with intent. What if there are two
lecturers? Is the ACM only supposed to track one of them? It would be preferred to
have both of the lecturers in the picture. Although, this introduces other problems.
Between the pictures, the filtering algorithm filtered the sources slightly different.
The result of the filtering depended on the light hitting the respective source. A
slight difference in light could disturb the position algorithm. This disturbance
could induce twitchy movement while tracking multiple sources.

One possible way to solve the problem resulting from multiple green objects could
be to combine all image tracking methods above to complement each other.

As seen in the time test Table 4.3, using a camera resolution of 224x160 was most
suitable with respect to the refresh rate of the system and the amount of acquired
data. Using this resolution adds 400\% more usable data while only increasing the
calculation time with a mere 25\%. As seen in the distance test Table 4.2, a resolu-
tion of 224x160 was plenty enough to detect objects on a distance of 15 m, given a
strong enough source of green color and suitable light.

Through practical tests it was discovered that the refresh rate of the ACM sys-
tem is more important than using the highest resolution possible. The refresh rate
is the how many calculation cycles, from taking a picture to moving the motor, the
system does per second. The refresh rate is more important because the chance of
capturing good data in a certain time frame is much larger with a larger refresh
rate. This could be explained by small differences in the pictures taken, such as
movement of the camera, movement of the cloth or a small difference in light. The
algorithm only needs one correct successful cycle to be able to adjust towards the
tracked object.

For example, lets compare the smallest resolution of 112x80 with the largest reso-
lution of 1920x1088. According to the times shown in Table 4.3, the ACM was able
to make about 20 calculation cycles at the lowest resolution in the same time as
one calculation at the highest resolution. The larger picture offers more data, which
increases the chance for success. However, failure to identify the source has to be
taken into account. A failure at a low resolution would be far less critical because
of less calculation time. If the resolution is lower, the algorithm gets more chances
to find the object, especially if the object is moving slightly.

In addition, a high refresh rate makes it possible to create a naturally turning
ACM. Although, in Table 4.2, higher resolution was shown to increase the range
of the ACM, which means that the range has to be balanced against the refresh rate.

One of the requirements in section 1.3 stated that the ACM should not produce
disturbing noises or lights. While there are no apparent loud noises, the motor is
5.2. CONCLUSION

not totally silent. The ACM could be somewhat disturbing for an audience sitting next to it or for a camera mounted on it. The sound could leak into the video if the camera-microphone is used to record sound.

Disturbing lights are a bit more subjective. The ACM does not in itself have any disturbing lights. Although, the ACM requires the source to wear a distinct green cloth. It could subjectively be distracting to watch a lecturer that wears distinct green clothes.

5.2 Conclusion

**What kind of tracking system is most suitable when building an ACM?**

A theoretical study and discussion found image processing analysis to be the most applicable tracking system. That is, a system that takes an image and puts it through an algorithm which searches for the intended object. The system only requires a receiver (the camera) and no separate transmitter unit. The ACM system was capable of tracking objects at a distance of 15 meters given a strong enough source in a suitable light environment. The demonstrator showed that it works in practice.

**How can a mechatronical construction and its control system be designed for an ACM?**

A mechatronic construction could be built using a Raspberry Pi, a camera, a step motor and a dual H-bridge circuit. The components could be put together using 3D-printed plastic parts. By using a step motor the control system becomes inherently stable, so no advanced control system has to be designed.

5.3 Recommendations for future work

The image detection algorithms could be further refined, using more advanced methods of object detection and combining different autonomous tracking solutions. How the algorithm handles the detection of two green objects could be further worked on. Also, the color filter could be redialed and tested with another color model instead of RGB.

To make the movement act more natural, like a person steering the camera, a more advanced velocity deciding algorithm for the motor could be designed. On top of that another degree of freedom could be implemented, so the camera can follow the lecturer both horizontally and vertically.
Bibliography


Appendix A

Flowcharts

This chapter contains generalized flowcharts for the three software modules in the project.
Figure A.1. Flowchart of the main module, created with the online software draw.io
**Figure A.2.** Flowchart of the image module, created with the online software *draw.io*.

The image module takes images and outputs position values that can be used by the motor for regulating. It runs on a thread which will end if any thread ends.

The whole algorithm is set up between a try and an exception. If the algorithm crashes it sends the message 'end' to main.
Figure A.3. Flowchart of the motor module, created with the online software draw.io
Appendix B

Python Code

B.1 main.py

```python
import threading
from imageshooter import *
from motorstyrning import *
from imageprocessing import *
import queue

class positionlog():
    def __init__(self):
        # CONSTANTS
        self.gearratio = 5

        # Transmission constant
        self.motor_steprevolution = 200

        # Number of steps per revolution for the motor only
        self.steprevolution = self.motor_steprevolution * self.gearratio

        # Number of steps per revolution
        self.camerawidth = 112
        self.cameraheight = 80
```
self.cameraFOV=62

# Camera Field Of View (degrees)
self.cameraFOV_steps=round(self.cameraFOV/360*self.steprevolution) # Camera Field Of View measured in steps

# POSITION VALUES#
self.COV = self.steprevolution//4

# Current Center Of View (Position from left border position to center of view)

self.errorvalue=0

# Where the measured value is compared to the last image taken (image position)

self.imageposition = self.steprevolution//4

# Where the latest images was taken

###
### OTHER#
###

self.textlog = queue.Queue()

# Used to communicate messages to the main thread

def PixelsToSteps(self, pixels):
    """ Transforms image pixels to step motor steps """
    steps = round(pixels/self.camerawidth*self.cameraFOV_steps)
    return int(steps)

def get_realerror(self):
    """ Returns the error value of the current position """
    return self.imageposition+self.errorvalue-self.COV

def __str__(self):
    """ Return the current position values """
    text='COV: ' + str(self.COV) + '
      errorvalue: ' + str(self.errorvalue) + '
      imageposition: ' + str(self.imageposition) + '
      realerror: ' + str(self.get_realerror())
def init_threaded_modules():
    """This method starts two threads: the motor thread and the image
    processing thread"""
    # New motor module thread
    motorThread = threading.Thread(target=motor_module, args=(pl,True))
    motorThread.daemon=True # Will terminate when main-thread ends
    motorThread.start()
    # New image module thread
    imageThread = threading.Thread(target=image_module, args=(pl,))
    imageThread.daemon=True # Will terminate when main-thread ends
    imageThread.start()

def motor_module(positionlog, loop=True):
    """Motor module that runs the motor on a thread""
    try:
        positionlog.textlog.put('Initializing motor module')
        H=Hbrygga()
        while loop:
            PosX = positionlog.get_realerror() # In steps
            if PosX>10:
                positionlog
                H.onestep(0.018,True)
                positionlog.COV+=1
            elif PosX<-10:
                H.onestep(0.018,False)
                positionlog.COV-=1
            else:
                H.setToIdle() # Let the motor rest so it doesn't get
            hot
        except Exception as e:
            positionlog.textlog.put(e)
            positionlog.textlog.put('end')
    def image_module(positionlog):
        """Image module that takes care of taking images with the camera
        and processing it""
        try:
            positionlogg.textlog.put('Initializing image module')
            camera=picamera.PiCamera()
            implementsettings(camera)

            # Setup for position function
            columns=int(positionlog.camerawidth)
            rows=int(positionlog.cameraheight)
            MultMatrix=np.transpose(np.zeros(columns))
            b=0
            while b <= columns:
                MultMatrix[b]=b-columns/2+1;
                b+=1
            while True:
                image=takeRGBimage(camera).array
currentPos = positionlog.COVP  # So we know where the image was taken
im2 = image.copy()
FiltIm = GreenFilt(im2)
PosX = GreenPos(FiltIm[:,:,:], MultMatrix, rows, columns)
step_PosX = positionlog.PixelsToSteps(PosX)

if abs(step_PosX) > 10:  # If position is too small don’t save the data
    positionlog.errorvalue = step_PosX
    positionlog.imageposition = currentPos

positionlog.textlog.put('Step position: ' + str(step_PosX))

except Exception as e:
    positionlog.textlog.put(e)
    positionlog.textlog.put('end')

if __name__ == '__main__':
    print('Setting up log')
    pl = positionlog()
    print('Setting up modules')
    init_threaded_modules()
    while True:
        if pl.textlog:
            msg = pl.textlog.get()  # Message from other threads
            print(msg)
            if msg == 'end':
                print('A thread crashed. Shutting down...')
                # end the program
                break;

H = Hbrygga()
H.setToIdle()

B.2 imageprocessing.py

# Filtering and image processing  #
# functions for ACM9000 project.  #
# By Simon Erlandsson & Gustav Burman  #
#  #
# Version 2.4:2018-05-14  #
#  #
#  #
import numpy as np
from scipy import misc
import time

def GreenFilt(RGB):
B.3. IMAGESHOOTER.PY

""" Filters out everything but green. Returns a black and white (boolean) matrix. """
range1=np.logical_and(RGB[:,:,1] >= 121,RGB[:,:,1]>RGB[:,:,0])
range2=np.logical_and(RGB[:,:,1]>RGB[:,:,2],RGB[:,:,0]<90,RGB[:,:,0]<80)
valid_range=np.logical_and(range1,range2)
RGB[valid_range] = 255  # Output color value if true (all channels)
RGB[np.logical_not(valid_range)] = 0  # Black if false
return RGB

def GreenPos(FiltIm, MultMatrix, rows, columns):
    """ Finds the green mean-value of the image. X axis only. """
    a=np.sum(FiltIm*MultMatrix)
    b=np.sum(FiltIm)
    if b == 0:
        ans=0
    else:
        ans=round(a/b)
    return ans

if __name__=="__main__":
    # For testing purposes
    import PIL as Image
    from imageshooter import *
    camera=picamera.PiCamera()
    implementsettings(camera)

    arr = takeRGBimage(camera).array
    RGB=arr.copy()
    Gim=GreenFilt(RGB)
    misc.imsave("test.jpeg",Gim)

B.3 imageshooter.py

#----------------------------------------------------------------------------------
# RGB image shooting program for a Raspberry Pi 3 and a Raspberry Pi camera v.2
# By Simon Erlandsson & Gustav Burman
# CMAST
# Version 1.0:2018-04-17
#----------------------------------------------------------------------------------
import picamera
import picamera.array
import time
import numpy
def implementsettings(camera):
    """Will implement the settings below for a picamera object"""
    camera.sensor_mode=0  #Automatiskt av resolution och framerate
    camera.resolution= (112, 80)
    camera.sharpness = 0
    camera.contrast = 0
    camera.brightness = 50
    camera.saturation = 0
    camera.IS0 = 50
    camera.video_stabilization = False
    camera.exposure_compensation = 0
    camera.exposure_mode = 'sports'
    camera.meter_mode = 'average'
    camera.awb_mode = 'auto'
    camera.image_effect = 'none'
    camera.color_effects = None
    camera.rotation = 0
    camera.hflip = True
    camera.vflip = True
    camera.crop = (0.0, 0.0, 1.0, 1.0)
    camera.image_denoise=False

def takejpgimage(name,camera):
    """Takes a jpeg image""
    #Specific for camera.capture:
    uvp = True  # use_video_port
    camera.capture(name + '.jpeg')

def takeRGBimage(camera):
    """Takes an images and returns an RGB matrix in the form of a picamera.array.PiRGBArray""
    output=picamera.array.PiRGBArray(camera)
    output.truncate(0)
    #Specific for camera.capture:
    uvp = True  # use_video_port
    camera.capture(output, 'rgb', use_video_port=uvp)
    return output

def camerastatus(camera):
    """Prints information about a picamera""
    print('Resolution ' + str(camera.resolution))
    print('Exposure mode ' + str(camera.exposure_mode))
    print('Horizontal flip ' + str(camera.hflip))
    print('Vertical flip ' + str(camera.vflip))
    print('Current exposure speed ' + str(camera.exposure_speed) + 'us')
    print('Image denoise: ' + str(camera.image_denoise))
    print('Image effect: ' + str(camera.image_effect))
    start=time.time()
    takeRGBimage(camera)
    end=time.time()
    taken=(time.time()-start)
    print('Time to shoot RGB image ' + str(taken) + 's')
B.4. IMAGETEST.PY

```python
if __name__ == '__main__':
camera = picamera.PiCamera()
implementsettings(camera)
takejpgimage('test', camera) # Takes a jpeg image
output = takeRGBimage(camera) # Takes a rgb image
camerastatus(camera)
print('Captured %dx%d image' % (output.array.shape[1], output.array.shape[0]))
start = time.time()
output = takeRGBimage(camera)
end = time.time()
taken = (time.time() - start)
print('Time to shoot RGB image ' + str(taken) + 's')
print('Captured %dx%d image' % (output.array.shape[1], output.array.shape[0]))
```

B.4 imagetest.py

```python
# ####################################################################
# Test program for camera and image processing.
# By Simon Erlandsson & Gustav Burman
# # Version 2.1: 2018-05-14
# ####################################################################
from imageshooter import *
from imageprocessing import *
from PIL import *
import time

# Functions used for data writing to file
def writelines(name, matrix):
    """ Adds a matrix of values to a file by the name of '(name).dat' """
    with open(name + '.dat', 'a') as file:
        for row in matrix:
            rowtext = str(row[0])
            for i in range(1, len(row)):
                rowtext += ',' + str(row[i])
            file.write(rowtext + '
')

def emptyfile(name):
    """ Empties the file stated as '(name).dat' """
    open(name + '.dat', 'w').close()

if __name__ == '__main__':
camera = picamera.PiCamera()
implementsettings(camera)
emptyfile('testdata') # Remember to save file after every re-run
num = 1

# Setup for position function
print('Setup for GreenPos')
```
APPENDIX B. PYTHON CODE

```python
columns = int(input('Columns (112|224): '))
rows = int(input('Rows (80|160): '))
print(str(columns), str(rows))
camera.resolution = (columns, rows)
MultMatrix = np.transpose(np.zeros(columns))
b = 0
while b <= columns:
    MultMatrix[b] = b - columns / 2 + 1;
    b += 1
print(str(MultMatrix))

while True:
    avstand = input('At what distance is the object?')
t0 = time.time()
image = takeRGBimage(camera).array
t1 = time.time()
im2 = image.copy()
t3 = time.time()
FiltIm = SuperGreenFilt(im2)
t4 = time.time()
# [PosX, PosY] = PosFunOneD(FiltIm[:, :, 1])  "Not interested in this one
# PosY2 = 0;
PosX = GreenPos(FiltIm[:, :, 1], MultMatrix, rows, columns)  "Rename
t5 = time.time()
if abs(PosX) > 10:
    print('WITHIN 10 PIXELS. NO NEW POSITION ASSIGNED')
t6 = time.time()

# Saving images:
FiltIm[:, int(PosX + columns / 2 - 1)] = 127  "Draws a line at PosX
print('PosX is ' + str(PosX))
misc.imsave('/media/pi/USB DISK/TESTBILDER/testi' + str(num) + '.jpg', image)
misc.imsave('/media/pi/USB DISK/TESTBILDER/testg' + str(num) + '.jpg', FiltIm)

# Printing the time it took to do each step
print('\ntakeRgbimage(camera).array: ' + str(t1 - t0))
print('image.copy(): ' + str(t3 - t1))
print('GreenFilt: ' + str(t4 - t3))
print('PosFunOneD: ' + str(t5 - t4))
print('if abs(PosX) > 10: ' + str(t5 - t6))
write_lines('testdata', [[num, avstand,PosX, t1 - t0, t3 - t1, t4 - t3, t5 - t4, t6 - t5]])

num += 1
```

34
B.5. MOTORSTYRNING.PY

B.5 motorstyrning.py

```python
# Code for driving a motor with a circuitry containing L298N and a Raspberry Pi's GPIO-pins.
# By: Simon Erlandsson
# Version: 2.3:2018-05-14

import RPi.GPIO as GPIO
import time

class Hbrygga:
    def __init__(self):
        GPIO.setmode(GPIO.BOARD)

        # Controller GPIO-pins: 31 33 35 37
        # Enabling pins: 36 38
        # Setting up outputs:
        self.ctrlpins_list = [31, 33, 35, 37]
        GPIO.setup(self.ctrlpins_list, GPIO.OUT)

        self.state0 = (GPIO.LOW, GPIO.LOW, GPIO.LOW, GPIO.LOW)
        self.state1 = (GPIO.HIGH, GPIO.LOW, GPIO.HIGH, GPIO.LOW)
        self.state2 = (GPIO.HIGH, GPIO.LOW, GPIO.LOW, GPIO.HIGH)
        self.state3 = (GPIO.LOW, GPIO.HIGH, GPIO.LOW, GPIO.HIGH)
        self.state4 = (GPIO.LOW, GPIO.HIGH, GPIO.HIGH, GPIO.LOW)
        self.setupState()

    def loop(self):
        # Main-loop for stepping
        GPIO.output(self.ctrlpins_list, GPIO.LOW)  # sets all to GPIO.LOW
        GPIO.output((GPIO.HIGH, GPIO.LOW))  # sets first HIGH and second LOW

        try:
            while True:
                self.velocityFunction(100, True)
                self.velocityFunction(100, False)
                self.step(100, 0.005, True)
                self.step(100, 0.1, False)
                GPIO.output(self.ctrlpins_list, self.state0)
        except KeyboardInterrupt:
            GPIO.cleanup()  # resets the status of any GPIO-pins (run before end)

    def step(self, steps, s_delay, turnclockwise):
        # Takes a number of steps, and sleeps for s_delay seconds, clockwise or anti-clockwise
```

35
APPENDIX B. PYTHON CODE

```python
# Steps: number of steps
# s_delay: How many second between each step
for i in range(steps):
    self.nextState(turnclockwise)
    time.sleep(s_delay)

def onestep(self, s_delay, turnclockwise):
    """Takes a step, and sleeps for s_delay seconds""
    self.nextState(turnclockwise)
    time.sleep(s_delay)

def nextState(self, turnclockwise):
    """Sets the next state on the motor based on if it's moving
clockwise or anti-clockwise""
    if self.state==0:
        GPIO.output(self.ctrlpins_list, self.state1)
    elif self.state==1:
        GPIO.output(self.ctrlpins_list, self.state2)
    elif self.state==2:
        GPIO.output(self.ctrlpins_list, self.state3)
    else:
        # self.state ==3
        GPIO.output(self.ctrlpins_list, self.state4)
    if turnclockwise == True:
        # clockwise
        self.state = (self.state + 1) % 4
    else:
        # anti-clockwise
        self.state = (self.state - 1) % 4

def setToIdle(self):
    """Lets the motor rest so it doesn't get to hot""
    GPIO.output(self.ctrlpins_list, self.state0)

def setupState(self):
    """ Goes through all states on the motor and sets it to the
starting state""
    speed = 0.01
    GPIO.output(self.ctrlpins_list, self.state1)
    time.sleep(speed)
    GPIO.output(self.ctrlpins_list, self.state2)
    time.sleep(speed)
    GPIO.output(self.ctrlpins_list, self.state3)
    time.sleep(speed)
    GPIO.output(self.ctrlpins_list, self.state4)
    time.sleep(speed)
    self.state = 0

if __name__ == '__main__':
    Hb = Hbrygga()
    Hb.loop()
```

36