Ergodesk
Automatic height adjustment of a desk
AXEL FYRESKÄR, OLIVIA EKMAN
ErgoDesk

Automatic height adjustment of a desk

AXEL FYRESKÄR, OLIVIA EKMAN

Bachelors’s Thesis at ITM, MDA
Supervisor: Nihad Subasic
Examiner: Nihad Subasic

TRITA ITM EX 2018:63
Abstract

The purpose of this project was to track the posture of a person sitting by a desk, and then use the information to adjust the desk to achieve an optimal height for the user.

The solution was to track the angle of the users forearm relative to the desk’s horizontal plane, which should be around zero degrees for an ergonomic posture. Two Light Emitting Diodes (LEDs) in different colours were attached to the user, one at the elbow and one at the wrist. The LEDs were tracked with a camera connected to a laptop running Open Source Computer Vision Library (OpenCV). Window tinting was placed on the camera lens so that only the LEDs and other strong light sources were visible. By using OpenCV, coordinates for the two LEDs were given. With the coordinates found, the angle of the users arm could be calculated and the height of the desk evaluated. Information about necessary height adjustments were sent from the laptop to an Arduino via Bluetooth. The Arduino then controlled a stepper motor by sending signals to a stepper driver, making the motor rotate, thus adjust the height of the desk.

The desk was a single leg laptop desk. The leg consists of two tubes. The lower, outer tube was connected to the foot while the upper inner tube, which ran freely in the outer tube, was connected to the tabletop. The height was adjusted with a lead screw positioned parallel to the leg. The nut on the lead screw was connected to the outer lower tube. When the motor rotated so did the lead screw, lifting the table top up or lowering it.

Keywords: mechatronics, automatic desk height, ergonomics, computer vision, colour tracking
Referat

Automatisk höjdjustering av ett skrivbord

Syftet med detta project var att undersöka om det är möjligt att i realtid spåra en persons hållning vid ett skrivbord, och använda den informationen för att justera skrivbordets höjd och nå en optimal höjd för användaren.


Nyckelord: mekatronik, automatisk skrivbordshöjd, ergonomi, datorseende, färgspärning
Acknowledgements

We would like to thank Nihad Subasic for supervising and feedback. We would also like to thank Staffan Qvarnström and Tomas Östberg for the help with the demonstrator. And finally we would like to thank all the lab assistants and classmates for the help and valuable 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 Ergonomics ............................................. 3  
2.2 Computer vision ........................................ 4  
2.3 Mechanics and electronics .............................. 6  

3 Demonstrator .............................................. 9  
3.1 OpenCV and programming ............................... 9  
3.2 Construction of height adjustable desk and LED units 12  

4 Results ..................................................... 15  
4.1 Testing rig and method ................................ 15  
4.2 Testing results ......................................... 16  

5 Discussion and conclusions .............................. 19  

6 Recommendations and future work ....................... 23  

Bibliography .................................................. 25  

Appendices .................................................... 26  

A Code .......................................................... 27  
A.1 Python code ........................................... 27  
A.2 Arduino code ........................................... 32  

B Wiring schematics ......................................... 35  

C Test results ................................................... 37
D Photos of desk and list of materials
## List of Figures

2.1 Different hand and wrist positions. .............................................. 3
2.2 A representation of the Hue, Saturation, Value (HSV) colour space. .... 4
2.3 Result of colour filtering a football. ........................................... 5
2.4 Geometric model of a person at a desk. ...................................... 5
2.5 Arduino Uno. .............................................................................. 6

3.1 Result of colour filtering after gaussian blur. ............................... 9
3.2 Image after use of erode and dilate functions. ................................. 10
3.3 Final image after processing. ....................................................... 10
3.4 Parts that connect the lead screw nut with the outer tube. ............... 12
3.5 Coupling for motor and lead screw .............................................. 13
3.6 Underside of desk ....................................................................... 14
3.7 LED units for tracking ............................................................... 14

4.1 Test rig for performance measuring .............................................. 15
4.2 Graph showing the error as a function of the testing rig’s angle ....... 16
4.3 Graph showing the error as a function of the camera’s distance ....... 16
4.4 Graph showing the error as a function of the cameras height ......... 17

5.1 The parallax effect .................................................................... 20

B.1 LED unit schematic ................................................................. 35
B.2 Arduino wiring schematic .......................................................... 36

D.1 Overview of the desk ............................................................... 39
D.2 Overview of the electronics ...................................................... 40
D.3 The desk’s foot with 3D-Printed adapters .................................. 40
## List of Tables

3.1 Conditions for desk adjustment ................................................. 11

C.1 Test data for 3D-Printed pucks, varying rig’s arm angle. ............... 37
C.2 Test data for LEDs, varying rig’s arm angle. ............................. 37
C.3 Test data for 3D-Printed pucks, varying camera distance. ................................................. 38
C.4 Test data for LEDs, varying camera distance ............................. 38
C.5 Test data for 3D-Printed pucks, varying camera height. ................ 38
C.6 Test data for LEDs, varying camera height ............................. 38

D.1 All material used in this project .............................................. 41
List of Abbreviations

AI  Artificial Intelligence
FOV  Field Of View
GPIO  General Purpose Input/Output
HSV  Hue, Saturation, Value
ISM  Industrial, Scientific and Medical
LED  Light Emitting Diode
OpenCV  Open Source Computer Vision Library
RGB  Red, Green, Blue
Chapter 1

Introduction

1.1 Background

As people spend more and more time of the day behind a desk, back and neck problems have become increasingly common [1]. This is often due to bad posture for a prolonged time and poor knowledge in ergonomics [2]. A system that can detect and react to bad posture could therefore aid the user in preventing future problems.

There are multiple ways to achieve this, two of which have been considered in this thesis - monitoring the pressure distribution in the users chair, and using a camera to monitor the angle of the users joints relative to the desk. Using 64 force sensitive resistors one would be able to detect nine different sitting postures [3]. Assuming that a combination of a computer and a camera can be used to track a user’s joints, one could theoretically detect an infinite number of postures. As such, this thesis will focus on joint and angle monitoring through computer vision.

1.2 Purpose

The aim of the project is to explore if it’s possible to live track the posture of a person sitting by a desk. The information will be used to adjust the height of the desk to achieve an optimal height, thus providing a comfortable and ergonomic posture. The user should be informed why the height of the desk is being adjusted in a non disturbing way to avoid confusion. The height adjustment will be motor driven.

Research questions for the project:

1. Is it possible to track the posture of a person sitting by a desk using computer vision?
2. How precise can the tracking be?
3. What is a good posture while sitting and standing?
1.3 Scope

Since computer vision is a very broad subject, the scope of this project is limited according to the following:

- A set distance between the person and the desk so that the angles between the tracking points can be specified.
- A constant lighting which will make it easier to track the person’s posture.
- There should be clear tracking points with a strong contrast to the surroundings.
- A small desk will be used as the purpose of this project is to develop a prototype, not a finished product.

1.4 Method

A laptop was used to run OpenCV, a computer vision library for Python (the language of choice) and other programming languages. With the help of a camera a computer was used to track different-coloured LEDs attached to points of interest on the user’s body. By setting up vectors connecting each of these points, the angle between said vectors could be calculated. This gave information of the user’s posture which in turn could be used to determine whether the user’s desk needed to be raised or lowered. A command to raise or lower the desk was then sent via serial communication using Bluetooth to an Arduino. The Arduino would then control the stepper motor used to adjust the desk’s height.
Chapter 2

Theory

2.1 Ergonomics

To help define a good posture an introductory research in sitting ergonomics was carried out. The knowledge from this could then be used to set a proper desk height for the user. This section will be dedicated to that research.

A good posture is defined by many things. But since the project involves tracking points on the user’s arm, the posture will be evaluated by the position and angles of the wrist and elbow. Too high a level of the working desk leads to a strain on the shoulder muscles. It also increases the circulatory systems burden on the heart. The desk should therefore be at the height of the user’s elbow [4]. The absolute maximum height should be at the point between the user’s shoulder and elbow [5].

A good position of the arms includes an approximate angle of $90^\circ$ between the upper arm and forearm, the latter should be horizontal. This allows the shoulder muscles to be relaxed and the wrist to be in a neutral position. A neutral position for the wrist means that it follows a straight line along the forearm and is not bent sideways, as seen in the picture below [6].

![Different hand and wrist positions](image)

**Figure 2.1.** Different hand and wrist positions [7].
2.2 Computer vision

Computer vision is a field in data science which aims to allow computers to recognize, compare, and evaluate a video or photos. Today it is being used for a wide variety of tasks like automation in factories, face recognition, autonomous cars etc. OpenCV is a library available to most of the major programming languages and the Python version was to be used for this project [8]. In this project, OpenCV was used to track coloured LEDs and 3D-Printed pucks attached the users joints which gave real time coordinates of the joints, and could in turn be used to measure the angle between them.

For the computer to know what objects to track, a colour filter was used. By specifying the color range in Hue, Saturation, Value (HSV) values or Red, Green, Blue (RGB) values (two different colour spaces, for an illustration of HSV color space, see Figure 2.2), the computer could then isolate the wanted objects. The HSV colour space was used since the colour data (hue channel) is generally very consistent when the lighting/brightness (value channel) of a scene changes when compared to RGB and others [9].

![Figure 2.2. A representation of the HSV colour space.[10]](image)

The desired result of this procedure was an image with the objects represented in white, and the rest of the image in black as seen in Figure 2.2. To find the coordinates of the marked areas, edge detection was used. To identify an edge in an image, one differentiates the intensity, meaning the value channel of the HSV colour space, between adjacent points. In this project, that meant comparing pixels with it’s neighbours. Since most pixels in an image have different intensity, one has to specify a minimum change of intensity (threshold) that would be regarded as an edge [11]. By using the pixel grid as a coordinate system, and taking the average of all edge pixel’s coordinates in that grid, one can then get an approximation of the objects middle point and in this case the users joints.
2.2. COMPUTER VISION

Figure 2.3. Result of colour filtering a photo of a football. Made in Photoshop CS6.

A possible way to find the height adjustment needed was to find the difference of elbow’s and the wrist’s y-coordinates with:

\[ \Delta h = A_y - B_y \] (2.1)

A difference of zero would mean that the user’s wrist and shoulder are at the same height, and no adjustment would be needed. If wanted, the angle of the user’s forearm can be found using:

\[ \alpha = \arctan \frac{\Delta h}{\Delta x} \] (2.2)

This angle could then inform the user of why the desk height is adjusting to avoid any confusion. The geometric model used to derive these equations is shown in 2.4.

Figure 2.4. Geometric model of a person at a desk. Made in Adobe Illustrator CS6.
2.3 Mechanics and electronics

The computer that handles the Python program and OpenCV, communicates with an Arduino by serial communication via the internal Bluetooth module in the laptop and a HC-06 module for the Arduino. Serial communication is a way to transfer data, with its alternative known as parallel communication. With parallel communication multiple bits of data are sent at the same time. This requires multiple wires, one for every bit being transferred simultaneously and usually one or more with other purposes. With serial communication only one wire is required since the data is being sent one bit at a time [12].

Bluetooth is a wireless communication standard using radio waves with a frequency of 2.4 GHz (Industrial, Scientific and Medical (ISM) band) and a bandwidth of 1 MHz. The standard is commonly used in mobile devices due to its low power consumption and flexibility when compared to cables [13].

An Arduino (seen in Figure 2.3) is a single board microcontroller. By using the appurtenant programming language (a variation of C), instructions can be sent to the microcontroller. The Arduino can then, following the instructions, turn different kinds of input into outputs via General Purpose Input/Output (GPIO) pins [14].

A stepper motor is a type of brushless motor. It allows control of the rotation of the motor shaft in steps, where each step corresponds to a rotation of $\gamma$ degrees that is specified in the motor’s data sheet. This type of motor also allows for high accuracy, high torque at low rotational speeds and easy interface with a computer or micro controller. A typical stepper motor is constructed from permanent magnets (the rotor) and stationary coils/phases (the stator) surrounding. By energizing the phases and alternating the currents direction through the coils in a particular order, a magnetic field with alternating north and south poles is created around
2.3. MECHANICS AND ELECTRONICS

This repels or attracts the permanent magnets, causing the motor to step. By repeating this process in rapid succession, a rotating motion can be achieved. If the phases stays energized without changing the current direction after a step is taken, a holding torque is created, meaning that the motor can hold its position when an external torque is applied to the motor shaft [15].

To control a stepper motor that uses a higher current than the pins of a micro controller can supply, a stepper motor driver with an external power supply can be used. The driver’s job is to interpret the input signal (from the Arduino) and power the different phases of the motor at the right time to make the motor spin accordingly.

In equation 2.1 the height adjustment needed would be given in pixels. Since the camera was at fixed distance from the user, a relationship, $x$, between pixels and length units could easily be found by measuring the pixels of an object with known length at said fixed distance from the camera.

$$x = \frac{\text{Width or height in pixels}}{\text{Known object’s length or height}}$$  \hspace{1cm} (2.3)

By using a coupling and a lead screw, the stepper motor’s rotational movement can be converted into a linear movement relative the nut attached to a fixed object, in this project the base of a desk.

This introduces a gearing which will have to be calculated in order to find a relation between the desk’s translation and the motors steps. This gearing will be dependent on the lead screw’s pitch, which affects the required torque to lift an object according to

$$T_{\text{raise}} = \frac{F d_m}{2} \left[ \frac{p \cdot \cos(\theta) + \mu \cdot \pi \cdot d_m}{\pi \cdot d_m \cdot \cos(\theta) - \mu \cdot p} \right]$$  \hspace{1cm} (2.4)

where $F$ is the force applied vertically on the lead screw, $d_m$ the mean diameter of the lead screw, $\mu$ the coefficient of friction and $l$ the lead of the screw. By choosing a fitting lead screw, $T_{\text{raise}}$ could then be lowered enough so as to meet the specification of the chosen stepper motor [16].

The LED units used as tracking points where made up out of 5mm LEDs, a CR2032 button cell battery (nominal voltage of 3V) and a resistor. A typical 5mm LED has a forward voltage of 2V and a forward current of 20 mA, meaning a voltage drop of 1V is required over the resistor [17]. With a current of 20 mA flowing through it, Ohm’s law can be applied to calculate the required resistance using:

$$R_{\text{res}} = \frac{U_{\text{res}}}{I_{\text{res}}} = \frac{1}{20 \cdot 10^{-3}} = 50\Omega$$  \hspace{1cm} (2.5)
The power dissipated by the resistor can be calculated with:

\[ P = U \cdot I = 1 \cdot 20 \cdot 10^{-3} = 20mW \]  \hspace{1cm} (2.6)

meaning any common film resistor with a 1/4W or 1/8W rating would suffice.
Chapter 3

Demonstrator

3.1 OpenCV and programming

As discussed in the theory chapter, OpenCV’s Python library was used to track the LEDs and 3D-printed pucks. This section will go more into detail on how the tracking was performed. The full code with comments is available in Appendix A.

The colour tracking was first done with 3D-printed pucks in the colours green and pink. Since the pucks were of very simple shapes, having a lot of detail in the image would have been a waste of computational resources later on. As such, a gaussian blur was used that averages the value of neighbouring pixels, making only larger areas of similarly valued pixels, stand out from the rest of the image.

Two color ranges (one matching each puck) in the HSV spectrum were defined using NumPy arrays. This would give a boolean mask, where every pixel that’s within the given spectrums would result in true (logical one) and be represented by a white pixel in the filtered image. The actual values for the ranges was experimentally found and often needed alteration to fit the current lighting in the room. The result of this procedure can be seen in Figure 3.1.

![Figure 3.1](image.png)

**Figure 3.1.** Result of colour filtering after gaussian blur. Mask to the left, webcam feed to the right.
To clean the image further, OpenCV’s functions erode and dilate were used. Erode removes small areas of white (logical ones) in the boolean mask, while dilate does the opposite (adds white), essentially making the white areas larger. By using erode before dilate, small areas of unwanted white in the mask could be removed, see Figure 3.1.

![Figure 3.2. Result after use of erode and dilate functions. Mask to the left, webcam feed to the right.](image)

With clean masks, showing only the pucks, a canny edge detector was used to find the contours of each puck. Using the contours, OpenCV could find what’s called ”Moments” (features of the contours), one of which was a contour’s centroid coordinates. All coordinates shown in Figure 2.4 were now known, and could be represented in the webcam feed, see Figure 3.1. Equation 2.2 would give the wanted angle. Using the method described with equation 2.3, the needed height adjustment of the desk could also be found.

![Figure 3.3. Final image after processing. Countours to the left, webcam feed with centroids marked and the triangle giving the angle \( \alpha \) drawn.](image)

The tracking of the pucks were very sensitive to changes in light and other disturbances such as similar colours in the surroundings. They were therefore replaced by 5 mm LEDs, whose colour is consistent even when the environment’s lighting changes. By placing a couple of layers of window tinting on the camera lens, only the LEDs and other strong light sources were visible to the camera.

In order for the desk to not constantly adjust height and to avoid unexpected
behaviour in case it loses tracking, a set of conditions had to be met before the laptop would send adjust commands to the Arduino. On top of this, a threshold where $\alpha$ would be considered within specifications were set. The conditions used can be found in Table 3.1 below.

<table>
<thead>
<tr>
<th>Reason</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avoid adjustment when wrist is slightly moved.</td>
<td>Adjust only if alpha has been out of specifications for 10 frames.</td>
</tr>
<tr>
<td>Avoid adjustment if user is not actively using the desk (phone call etc.)</td>
<td>Adjust only if wrist is close to the desk.</td>
</tr>
<tr>
<td>Robustness against lost tracking points (i.e. some walks in front of the camera)</td>
<td>Reuse last known coordinate of tracking point. If lost for more than 10 frames, stop adjustments.</td>
</tr>
</tbody>
</table>

If all conditions were met, a height adjust command would be sent from the laptop to the Arduino via the serial port (i.e. the bluetooth module).

The Arduino was programmed independently of the Python program on the laptop. Upon power up the Arduino would lower the table until an endstop (a momentary switch) mounted on the base of the desk was hit. This zeroed the coordinate system of the desk. A simple counter variable could now be used to keep track of the desk’s position, ensuring it could not exceed its physical limit. After zeroing the coordinates, a loop was constructed to check for available data on the serial port and read that data (height adjustment commands from the laptop). Depending on the received command, the direction pin was set either low or high to make the motor rotate clockwise or counter clockwise. The Arduino would then control the stepper motor by writing alternating logical ones and zeroes (effectively a PWM signal) to the stepper drivers “step”-pin, making the motor rotate and adjust the height of the desk. Each logical one made the motor take a single step ($1/200$ of a revolution when configured for full-stepping with the chosen motor). Since the precision acquired from $1/200$ of a revolution per step was deemed plentiful, the stepper driver was configured in full-stepping mode to allow for maximum torque from the motor. This meant leaving the MS1, MS2 and MS3 pins unconnected since they are pulled low internally with the chosen stepper driver. Due to the constant gearing of the lead screw, each step corresponded to a fixed height adjustment. Using this information one could know how many steps the motor had to take for the angle $\alpha$ to be within the allowed range (and for $\Delta h$ to be close to zero).
3.2 Construction of height adjustable desk and LED units

The desk was made of a table top and foot from an IKEA laptop stand, *Svartåsen*. Both original inner and outer tubes were replaced with longer ones to allow for the larger height adjustments required to reach both standing and sitting positions. The tubes were attached to the foot and table top by 3D-printed adapters that allowed for the new tube’s diameters.

The height was adjusted with a lead screw which was positioned parallel to the pedestal. The lead screw nut was attached to the outer (lower) tube with parts shown in Figure 3.4, which were cut in aluminium with a water jet. To ensure that the lead screw was parallel to the tubes, the two parts allowed some adjustments of the distance between the lead screw and the tubes through the use of slots.

![Figure 3.4. Parts that connects the lead screw nut with the outer tube.](image)

The lower part was glued to a washer that was welded to the outer, stationary tube. The stepper motor (model Nema 17) was attached to the underside of the table top with a 3D-printed case. The lead screw was connected to the motor with a sleeve coupling made in aluminium, see figure 3.5 below.
3.2. CONSTRUCTION OF HEIGHT ADJUSTABLE DESK AND LED UNITS

The coupling has holes with different diameters, one for the motor axle and one for the lead screw. They are attached by two grub screws perpendicular to the motor axle and lead screw. To prevent the axle from rotating inside the coupling, the motor axle was sanded down to get a flat surface which the screw could be tightened against. This was not required for the lead screw since it’s threads allow the grub screw to have a sturdy grip.

A blue LED strip was installed on the side of the desk top with double sided tape to allow for tracking of the table. The Arduino, stepper driver (DRV8825), and bluetooth module (HC-06) was mounted to the underside of the desk. A momentary switch that was used as an endstop was attached to the motor casing using a 3D-printed bracket which allowed for some adjustment of the endstop’s distance from the casing. This endstop (see Figure 3.2) would be triggered by hitting the parts in figure 3.4 when the desk was at it’s lowest position. The endstop was connected to ground and a digital pin configured with an internal pull-up resistor to avoid leaving the pin floating. All electronics were powered through a 12V power supply rated for 2000mA. More pictures of the full construction and a list of materials is available in Appendix D.
The LED unit housing was designed in Solid Edge and 3D-Printed. A short piece of resistance wire (with negligible resistance) was wound into a coil to act as a spring pushing against one of the button cell’s poles. One end of the coil was soldered to a resistor with value $a$ of 56 $\Omega$ (calculated value from equation 2.5 rounded up to closest standard resistor) and then to the LED. A wire was soldered from the other leg of the LED and then attached to the top of the other half of the casing directly above the battery’s other pole. Upon pushing the two halves together, the circuit would be closed and the LED would light up. A photo of the LED units is shown below in Figure 3.2 A schematic of all wiring for both the LEDs and the Arduino is available in Appendix B.
Chapter 4

Results

4.1 Testing rig and method

To test the precision and accuracy of the demonstrator, a test rig simulating a persons arm was built. Two laser cut wooden arms held together with a single screw and bolt allowed the angle of the upper arm to be adjusted. See Figure 4.1 for a photo of the test rig. This rig was then used to measure the performance of the Python program when using LED units and 3D-printed pucks.

![Figure 4.1. Test rig with 3D-printed pucks attached and the web camera.](image)

Each test had a duration of 10 seconds after which the maximum, minimum, and average angle registered by the computer vision program was found. The average angle was calculated by summing the angle for every frame within the set time span and dividing by the total number of frames during that same time. After each test one of three parameters was altered - camera’s distance to the rig, camera’s height relative the rig or the angle of the rig’s arm. The same tests was carried out for both types of tracking points. Both types of tracking points were tested in different lighting conditions throughout the project.
4.2 Testing results

The data acquired from testing was plotted in graphs (see Figure 4.2 to 4.4) to compare the performance of the LED units and 3D-printed pucks. As the Python program only commands the Arduino to adjust the desk height if the angle $\alpha$ has been out of specification for a set amount of frames in a row, the average registered angle was determined to be the best way to measure performance. In the graphs below, the difference between this angle and the actual angle of the rig is presented as an error. Full testing data including minimum and maximum angles is available in Appendix C.

![Graph showing the error as a function of the testing rig's angle. Made in Microsoft Excel.](image)

**Figure 4.2.** Graph showing the error as a function of the testing rig's angle. Made in Microsoft Excel.

![Graph showing the error as a function of the camera's distance to the testing rig. The testing rig's arm angle was fixed at 30 degrees. Made in Microsoft Excel.](image)

**Figure 4.3.** Graph showing the error as a function of the camera’s distance to the testing rig. The testing rig’s arm angle was fixed at 30 degrees. Made in Microsoft Excel.
4.2. TESTING RESULTS

Figure 4.4. Graph showing the error as a function of the cameras height relative the desk that the testing rig was placed on. The testing rig’s arm angle was fixed at 30 degrees. Made in Microsoft Excel.

The LED units proved superior in terms of consistency in varying lighting. For the 3D-printed pucks to be tracked well, they relied on the environment to give a diffused, even lighting. Sharp lights and shadows caused reflections on the pucks, making them appear white to the camera. As a result, the computer vision program often lost track of the pucks. The LED units provided their own source of light, while efficiently blocking everything but other strong light sources using tinting film. This made the LED units work well in all tested environments with the exception of outdoors.
Chapter 5

Discussion and conclusions

The testing showed high accuracy for both the LED units and 3D-printed pucks, with a worst case average error of 2.26 and 0.89 degrees for the LED units and 3D-printed pucks respectively. All tests also showed consistent results, and only differed roughly one degree when altering testing parameters mentioned in section 4.1. Due to the small result changes when altering test parameters, the effects of human error and slight environmental changes (less light through windows due to clouds, person opening a door and changing lighting etc.) was high. As such, it’s hard to see any reoccurring patterns and to distinguish effects of altered test parameters and said sources of error.

In the first test the rig’s arm angle was altered. No correlation between the angle and the error could be seen, meaning that the computer vision program is capable of properly calculating the angle $\alpha$ regardless of the user’s current posture. Likewise no correlation was found between the error and the camera’s distance to the test rig. At a distance of above 2 meters, the LEDs were not strong enough to be seen by the camera, therefore no further testing was meaningful.

For the third test where the height of the camera was gradually increased while keeping the testing rig’s arm fixed at 30 degrees, a decrease in registered angle was expected due to a parallax effect, see Figure 5.1. No such result was shown in the test data. This is probably a result caused by the camera not able to be moved high up enough before the test rig was out of the camera’s Field Of View (FOV).
Worth noting is that the LED units consistently had a larger difference between maximal and minimal registered angle, see Appendix C. This is due to the glare in the webcam caused by pointing light sources directly at the camera sensor. By adjusting the angle of the LEDs relative the camera sensor slightly, the glare effect could change drastically. This could partially explain why the LED unit’s test results fluctuated more than the 3D-printed puck’s. In this project’s application this was not an issue since the registered angle was averaged over more frames than in the test. The angle also had to be out of specification (which were larger than the fluctuations caused by glare) for multiple frames before a height adjustment command was sent to the Arduino. There are multiple other sources of error present, all of which the testing ruled out as a problem. Examples of these are camera lens distortion and nonoptimal colour filtering ranges. As a summary of the testing, no inconsistencies that could cause problems reading a persons posture at the desk were found.

With regards to the construction of the demonstrator, there are multiple problems caused by the scope of this project - to build a prototype and not a finished project. The stepper motor is just strong enough to adjust the height of the desk, and common desk products like a computer monitor can not be placed on top. The bearings in the motor are not designed for the axial load they are currently exposed to. As a result, the expected life time is significantly decreased. The desk is also a bit unstable, especially when in a high position for standing use. Another issue is that only the wrist and elbow are currently tracked. This means that there is no information available of the elbows position relative the user’s shoulder. As a result, a user could hold his or her arms straight out, at a 90 degree angle, and the computer vision program would regard it as a good posture and not lower the desk as needed to reach the wanted 90 degree angle between upper arm and forearm. Using a third tracking point on the user’s shoulder to monitor this would introduce problems - a
user placed to far from or to close to the desk would have to be prompted to adjust its distance to the desk. This is not possible using only mechanical solutions through the desk and a second unit controlling the user’s distance from the desk would have to be constructed. Another solution would be to simply let the user know that he or she is at an improper distance from the desk through a visual or audible notification.

As a conclusion, we have successfully been able to track parts of a persons posture at a desk in real time, and automatically adjust the height of a desk to reach the optimal, horizontal posture of the elbow relative the wrist for the user. The precision of the computer vision algorithm proved to be more than enough for this application. We consider the purpose of this project fulfilled.
Chapter 6

Recommendations and future work

There are a number of things that could be improved to make the product more user friendly. One of them is how the user interacts with the computer running the program. In this project a laptop was used to run OpenCV. This task could be moved to a single board computer mounted under the table top or by the camera. Another thing that would be needed to replace the laptop is an on/off button for the height adjustment. A small display could inform the user of why the height is changing or not.

Making it possible to save settings for different users, as with the drivers seat in some cars, could make it possible for a user to save a good standard height and not have to calibrate the desk for every use, even if another person has been using the desk lately.

Wearing LEDs or pucks is not optimal for every day work, therefore a solution where these are not needed would be preferred. One solution would be to use stereo vision. Stereo vision is using two cameras at different positions and comparing their pictures to get 3D-information, a disparity map, showing the depth in a picture. That information would make it possible to distinguish the users arm from his torso, which is needed to be able to track the angle of the arm without any worn tracking points.

As mentioned in the discussion, tracking more than two points is necessary to get a good height even if the user is not sitting or standing right behind the desk. A third point at the users shoulder would give the information needed. Another alternative to consider would be to identify bad postures is with Artificial Intelligence (AI). This could make it possible to adjust the height not only relative to the angle of the users arm but also to the users back and neck posture.

The desk built in this project works for demonstration but is to unstable for actual use as a desk. For future work this could be solved by having two or more legs and
two motors, much like many manually adjusted desks are constructed. If the height is adjusted with lead screws the motors needs to be relieved from any axial loads. An option to lead screws could be an hydraulic system. Having multiple table legs makes it possible to have a larger table top, which also is needed if the desk is going to be used for more than a laptop.
Bibliography


Appendix A

Code

A.1 Python code

```python
# By Axel Fyreskar and Olivia Ekman
# Course MF133X - Examensarbete inom mekantronik, May 2018
#
# This program uses OpenCV to track LEDs or 3D-Printed pucks depending on the defined color ranges. The coordinates of the tracked objects are then used to calculate the angle between the two objects and a horizontal plane. It then sends serial commands to the Arduino to raise or lower the desk, thus adjusting the angle.

import cv2
import numpy as np
import serial

device = cv2.VideoCapture(1)  # Opens the webcam device

# Initial setup of centroid coordinates to avoid zero division if no tracking points are identified at startup

cxGreenLast = 1
cyGreenLast = 1
cxPinkLast = 1
cyPinkLast = 1
cyBlue = 0
ydiff = 0
pinkLostFrames = 0
outOfSpecCount = 16
BT = serial.Serial('COM6', 9600)  # Open serial port to arduino (baudrate 9600), set according to device manager in Windows

print("Serial Port: %s" % BT.name)  # Print port name for debugging
```

27
while True:
    _, frame = device.read() # reads frames from device (webcam)
    frame = cv2.GaussianBlur(frame, (11, 11), 0) # Blurs the frames to reduce noise
    frame = cv2.flip(frame, 1) # Flips the frames horizontally
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) # Converts frames from BGR to HSV
    lowergreen = np.array([65, 70, 70]) #np.array([40, 85, 50]) # Defines lower boundary of the green marker (elbow)
    uppergreen = np.array([75, 255, 255]) #np.array([60, 255, 240]) # Defines upper boundary of the green marker (elbow)
    lowerpink = np.array([-20, 85, 100]) #np.array([140, 100, 50]) # Defines lower boundary of the red marker (wrist)
    upperpink = np.array([10, 255, 255]) #np.array([180, 255, 255]) # Defines upper boundary of the red marker (wrist)
    lowerblue = np.array([100, 80, 80])
    upperblue = np.array([125, 255, 255]) # Defines upper boundary of the blue marker (desk)
    maskPink = cv2.inRange(hsv, lowerpink, upperpink) # Creates boolean mask according to red boundaries
    maskGreen = cv2.inRange(hsv, lowergreen, uppergreen) # Creates boolean mask according to green boundaries
    maskBlue = cv2.inRange(hsv, lowerblue, upperblue) # Creates boolean mask according to blue boundaries
    # Erodes and dilates (remove false positives, add false negatives)
    kernel = None
    maskPink = cv2.erode(maskPink, kernel, iterations=1)
    maskPink = cv2.dilate(maskPink, kernel, iterations=3)
    #maskGreen = cv2.erode(maskGreen, kernel, iterations=3)
    #maskGreen = cv2.dilate(maskGreen, kernel, iterations=3)
    #maskOrange = cv2.erode(maskOrange, kernel, iterations=3)
    #maskOrange = cv2.dilate(maskOrange, kernel, iterations=3)
    # Finds contours of the masks (binary)
    edgesPink = cv2.Canny(maskPink, 100, 200)
    edgesGreen = cv2.Canny(maskGreen, 100, 200)
    edgesBlue = cv2.Canny(maskBlue, 100, 200)
    contoursPink = cv2.findContours(edgesPink, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contoursGreen = cv2.findContours(maskGreen, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    contoursBlue = cv2.findContours(maskBlue, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
A.1. PYTHON CODE

```python
# Extracts features of the contours (in a dictionary)
MPink = cv2.moments(contoursPink[0])
MGreen = cv2.moments(contoursGreen[0])
MBlue = cv2.moments(contoursBlue[0])

if int(MPink['m00']) != 0:  # If contour’s area is existing (atleast 1 true pixel)
    pinkLostFrames = 0  # Resets counter of number of lost frames for the red marker

    # Find centroid’s coordinates of contour
    cxPink = int(MPink['m10'] / MPink['m00'])
    cyPink = int(MPink['m01'] / MPink['m00'])

    if int(MBlue['m00']) != 0:  # If blue marker is found
        cyBlue = (int(MBlue['m01'] / MBlue['m00']))  # Calculate y-coordinate of blue marker
    else:  # If blue marker is not found
        cyBlue = cyPinkLast + 100  # Set blue marker’s y-coordinate to 100 pixels below the last known pink marker’s coordinate to avoid desk adjustment

        ydiff = abs(cyPink - cyBlue)  # Calculate height difference of red marker and blue marker

        # Save coordinates for reuse if next frames coordinates are not found
        cxPinkLast = cxPink
        cyPinkLast = cyPink

else:  # If red marker has not been found
    pinkLostFrames += 1  # Add one to red markers lost frame counter

    cxPink = cxPinkLast  # Set red markers x-coordinate to last known coordinate

    if pinkLostFrames > 10:  # If red marker has not been identified for 10 or more frames in a row
        cyPink = cyGreenLast  # Set red markers y-coordinate to last known y-coordinate in order to avoid desk adjustments

    else:
        cyPink = cyPinkLast

if int(MGreen['m00']) != 0:  # If green marker has been identified
    cxGreen = int(MGreen['m10'] / MGreen['m00'])  # Calculate its x-coordinate
    cyGreen = int(MGreen['m01'] / MGreen['m00'])  # Calculate its y-coordinate
```

cxGreenLast = cxGreen  # Save coordinates for reuse if next frames coordinates are not found

cyGreenLast = cyGreen  # Save coordinates for reuse if next frames coordinates are not found

else:  # If green marker has not been identified
    cxGreen = cxGreenLast  # Set green markers x-coordinate to last known coordinate
    cyGreen = cyGreenLast  # Set green markers y-coordinate to last known coordinate

# Calculate the vertical (height) and horizontal differences of centroids
# Later used to know needed desk adjustment
deltah = cyGreen - cyPink
deltax = cxGreen - cxPink
# print("DeltaH:", deltax)  # Debugging
# print("DeltaX:", deltax)  # Debugging

# Find angle between forearm and horizontal line extending from wrist/desk
if deltax == 0:
    armangle = 90  # Avoid division by zero, if deltax == 0, green marked is directly above pink marker
else:
    armangle = np.rad2deg(np.arctan(deltah / deltax))

# Draw circles marking the centroids
cv2.circle(frame, (cxPink, cyPink), 7, (0, 0, 255), -1)
cv2.circle(frame, (cxGreen, cyGreen), 7, (0, 0, 255), -1)

# Draw triangle connecting centroids
cv2.line(frame, (cxPink, cyPink), (cxGreen, cyGreen), (0, 0, 255), thickness=1)
cv2.line(frame, (cxPink, cyPink), (cxGreen, cyPink), (0, 0, 255), thickness=1)
cv2.line(frame, (cxGreen, cyPink), (cxGreen, cyGreen), (0, 0, 255), thickness=1)
cv2.line(frame, (0, cyBlue), (640, cyBlue), (255, 0, 0), thickness=1)
cv2.ellipse(frame, (cxPink, cyPink), (35, 35), 0, 0, armangle, (0, 255, 0), 2)
armangletext = 'alpha=' + str(round(armangle, 1))
cv2.putText(frame, armangletext, (cxPink - 100, cyPink - 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)

# Show windows with wanted masks
totMask = maskGreen + maskPink + maskBlue  # Add masks to show both in one window
# cv2.imshow("Mask", totMask)
A.1. PYTHON CODE

```python
import cv2

# Rules for desk adjustments
if armangle < -10 or armangle > 0:  # If the angle if out of spec, add one to out of spec count
    outOfSpecCount += 1
if armangle < -10 and ydiff < 65 and outOfSpecCount > 20:  # If angle if less then 10 deg, ydiff is less than 65 pixels (wrist is close to desk), and arm angle been out of spec for more than 20 frames
    BT.write(b'L')  # Send serial command to adjust desk height
elif armangle > 0 and ydiff < 65 and outOfSpecCount > 20:  # If angle if larger then 10 deg, ydiff is less than 65 pixels (wrist is close to desk), and arm angle been out of spec for more than 20 frames
    BT.write(b'H')  # Send serial command to adjust desk height
elif armangle > -10 and armangle < 0:  # If armangle is withing spec (wrist is horizontal to elbow, or slightly lower)
    outOfSpecCount = 0  # Reset outofspec counter
    BT.write(b'O')  # Send angle OK command to arduino
else:
    BT.write(b'O')  # If none of the above rules apply, send OK to arduino in order to avoid unexpected behaviour

if cv2.waitKey(1) == 27:  # If user press ESC key, break out from loop and send quit instruction to Arduino
    BT.write(b'O')
break
```

Full Python code with comments. Used on the laptop to run OpenCV and communicate with the Arduino attached to the desk.
A.2 Arduino code

// By Axel Fyreskar and Olivia Ekman
// Course MF133X - Examensarbete inom mekatronik, May 2018
//
// This program receives serial commands from the laptop via a
bluetooth module and raises or
lowers the desk as needed. A
stepper motor is controlled through
a DRV8825 stepper driver.

#include <SoftwareSerial.h> // Import SoftSerial library
SoftwareSerial BT(10, 11); // Defines softserial port called BT. Sets
RX and TX pin of BT port to 10
and 11
const int ledPins[] = {2, 3, 4}; // Defining pins of LEDs for
debugging (in order RGB)
const int endStopPin = 7; // Defining pin for the endstop (Connect
one pin to 7, the other to GND)
const int stepPin = 5; // Defining stepper pin for stepper driver
const int dirPin = 6; // Defining direction pin for stepper driver
const int sleep_resetPin = 8; // Defining sleep and reset pin for
stepper driver
const int maxSteps = 17000; // Maximum steps from zero position
int i = 0; // Used for counting (loops)
int currSteps = 0; // Variable for counting steps from bottom
char serial_rx = ' '; // Variable used to save incoming commands

void setup()
{
  // Serial.begin(9600); // Initializing serial port, baudrate 9600
  BT.begin(9600); // Initializing BT soft serial port, baudrate 9600
  int pin = 2;
  while(pin<=6){ // Initializing pins to output and defaulting them
to logical low.
    pinMode(i, OUTPUT);
    digitalWrite(i, LOW);
    pin++;
  }
  pinMode(sleep_resetPin, OUTPUT); // Setting pin used for sleep
  and reset to output
  pinMode(endStopPin, INPUT_PULLUP); // Setting pin used for the
  endstop to a input through a
  internal pull up resistor
  pinMode(LED_BUILTIN, OUTPUT); // Setting the internal LED to a
  output
  digitalWrite(LED_BUILTIN, LOW); // Turning internal LED off
  digitalWrite(dirPin, LOW); // Setting direction pin to LOW in
  order for the desk to lower it self
  digitalWrite(sleep_resetPin, HIGH); // Setting sleep and reset pin
to HIGH in order to enable the
A.2. ARDUINO CODE

```cpp
stepper driver
while (digitalRead(endStopPin) == 1) { // Homing coordinates (finding end stop)
    digitalWrite(stepPin, HIGH);
    delayMicroseconds(800);
    digitalWrite(stepPin, LOW);
    delayMicroseconds(800);
}
currSteps = 0; // When at endstop, set current steps to 0
digitalWrite(sleep_resetPin, LOW); // Set stepper driver to sleep mode
}
void loop() {
    if (BT.available() > 0) { // Checking if there is any serial data available
        serial_rx = BT.read(); // Reading a string from the serial stream
        if (serial_rx == 'H') { // If the angle is to High
            if (currSteps != 0) { // If desk is not at bottom
                digitalWrite(dirPin, LOW); // Set direction to low
                digitalWrite(sleep_resetPin, HIGH); // Enable stepper driver
                i = 0;
                while (i < 1000 && currSteps != 0) { // Step 1000 times, check if bottom is reached for each step
                    digitalWrite(stepPin, HIGH);
                    currSteps--;
                    delayMicroseconds(800);
                    digitalWrite(stepPin, LOW);
                    delayMicroseconds(800);
                    BT.read();
                    i++;
                }
            }
        } else { // If the angle is Ok
            digitalWrite(sleep_resetPin, LOW); // Set stepper driver to sleep
        }
    } else if (serial_rx == 'O') { // If the angle is to Low
        if (currSteps != maxSteps) { // Check if desk is at max height
            digitalWrite(dirPin, HIGH); // Set direction pin to high
        }
    } else if (serial_rx == 'L') { // If the angle is to Low
        if (currSteps != maxSteps) { // Check if desk is at max height
            digitalWrite(dirPin, HIGH); // Set direction pin to high
        }
    }
}
```
APPENDIX A. CODE

digitalWrite(sleep_resetPin, HIGH); // Enable stepper driver
i = 0;
while(i<1000 && currSteps < maxSteps){ // Step 1000 times, check if max height is reached for each step
    digitalWrite(stepPin, HIGH);
currSteps++;
delayMicroseconds(1100);
digitalWrite(stepPin, LOW);
delayMicroseconds(1100);
BT.read();
i++;
}
else{
digitalWrite(sleep_resetPin, LOW); // If max height is reached, set stepper driver to sleep
}

else if(serial_rx == '0') // If communication stops (OpenCV/Python quits)
digitalWrite(sleep_resetPin, LOW); // Set stepper driver to sleep

digitalWrite(sleep_resetPin, HIGH); // Enable stepper driver
i = 0;
while(i<1000 && currSteps < maxSteps){ // Step 1000 times, check if max height is reached for each step
    digitalWrite(stepPin, HIGH);
currSteps++;
delayMicroseconds(1100);
digitalWrite(stepPin, LOW);
delayMicroseconds(1100);
BT.read();
i++;
}
else{
digitalWrite(sleep_resetPin, LOW); // If max height is reached, set stepper driver to sleep
}

else if(serial_rx == '0') // If communication stops (OpenCV/Python quits)
digitalWrite(sleep_resetPin, LOW); // Set stepper driver to sleep

Full code used on the Arduino to receive commands from the laptop and control the stepper motors.
Appendix B

Wiring schematics

Figure B.1. Schematic of the LED units. Made using Schematics.com online editor
Figure B.2. Schematic of the Arduino wiring. Made using Fritzing.
Appendix C

Test results

Actual angle was the test rig’s set angle. Camera distance was measured from the pucks/LEDs to the camera sensor. Camera height was measured from the desk where the test rig was placed to the camera sensor. Average, minimum and maximum angles were the result of the Python code.

Table C.1. Test data for 3D-Printed pucks, varying rig’s arm angle.

<table>
<thead>
<tr>
<th>Actual angle [°]</th>
<th>Distance [m]</th>
<th>Height [cm]</th>
<th>Avg. angle [°]</th>
<th>Min. angle [°]</th>
<th>Max. angle [°]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.5</td>
<td>10</td>
<td>0.32</td>
<td>-0.63</td>
<td>0.64</td>
</tr>
<tr>
<td>15</td>
<td>1.5</td>
<td>10</td>
<td>15.07</td>
<td>14.50</td>
<td>15.64</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>10</td>
<td>30.65</td>
<td>30.26</td>
<td>31.23</td>
</tr>
<tr>
<td>45</td>
<td>1.5</td>
<td>10</td>
<td>44.28</td>
<td>43.87</td>
<td>44.78</td>
</tr>
</tbody>
</table>

Table C.2. Test data for LEDs, varying rig’s arm angle.

<table>
<thead>
<tr>
<th>Actual angle [°]</th>
<th>Distance [m]</th>
<th>Height [cm]</th>
<th>Avg. angle [°]</th>
<th>Min. angle [°]</th>
<th>Max. angle [°]</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1.5</td>
<td>10</td>
<td>0.85</td>
<td>-1.75</td>
<td>3.58</td>
</tr>
<tr>
<td>15</td>
<td>1.5</td>
<td>10</td>
<td>16.39</td>
<td>15.36</td>
<td>16.88</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>10</td>
<td>30.19</td>
<td>29.78</td>
<td>32.36</td>
</tr>
<tr>
<td>45</td>
<td>1.5</td>
<td>10</td>
<td>44.15</td>
<td>43.51</td>
<td>45.43</td>
</tr>
</tbody>
</table>
## APPENDIX C. TEST RESULTS

### Table C.3. Test data for 3D-Printed pucks, varying camera distance.

<table>
<thead>
<tr>
<th>Actual angle [°]</th>
<th>Distance [m]</th>
<th>Height [cm]</th>
<th>Avg. angle [°]</th>
<th>Min. angle [°]</th>
<th>Max. angle [°]</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1.0</td>
<td>10</td>
<td>29.11</td>
<td>27.55</td>
<td>29.45</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>10</td>
<td>30.65</td>
<td>30.26</td>
<td>31.23</td>
</tr>
<tr>
<td>30</td>
<td>2.0</td>
<td>10</td>
<td>29.48</td>
<td>28.64</td>
<td>30.17</td>
</tr>
</tbody>
</table>

### Table C.4. Test data for LEDs, varying camera distance.

<table>
<thead>
<tr>
<th>Actual angle [°]</th>
<th>Distance [m]</th>
<th>Height [cm]</th>
<th>Avg. angle [°]</th>
<th>Min. angle [°]</th>
<th>Max. angle [°]</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1.0</td>
<td>10</td>
<td>31.14</td>
<td>30.47</td>
<td>31.68</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>10</td>
<td>30.19</td>
<td>29.78</td>
<td>32.36</td>
</tr>
<tr>
<td>30</td>
<td>2.0</td>
<td>10</td>
<td>31.26</td>
<td>30.00</td>
<td>33.13</td>
</tr>
</tbody>
</table>

### Table C.5. Test data for 3D-Printed pucks, varying camera height.

<table>
<thead>
<tr>
<th>Actual angle [°]</th>
<th>Distance [m]</th>
<th>Height [cm]</th>
<th>Avg. angle [°]</th>
<th>Min. angle [°]</th>
<th>Max. angle [°]</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1.5</td>
<td>0</td>
<td>29.86</td>
<td>29.27</td>
<td>30.26</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>10</td>
<td>30.65</td>
<td>30.26</td>
<td>31.23</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>20</td>
<td>29.91</td>
<td>29.27</td>
<td>30.42</td>
</tr>
</tbody>
</table>

### Table C.6. Test data for LEDs, varying camera height.

<table>
<thead>
<tr>
<th>Actual angle [°]</th>
<th>Distance [m]</th>
<th>Height [cm]</th>
<th>Avg. angle [°]</th>
<th>Min. angle [°]</th>
<th>Max. angle [°]</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>1.5</td>
<td>0</td>
<td>29.53</td>
<td>28.32</td>
<td>31.07</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>10</td>
<td>30.19</td>
<td>29.78</td>
<td>32.36</td>
</tr>
<tr>
<td>30</td>
<td>1.5</td>
<td>20</td>
<td>32.26</td>
<td>31.23</td>
<td>33.52</td>
</tr>
</tbody>
</table>
Appendix D

Photos of desk and list of materials

Figure D.1. An overview of the desk in full.
APPENDIX D. PHOTOS OF DESK AND LIST OF MATERIALS

Figure D.2. An overview of the electronics mounted to the underside of the desk.

Figure D.3. The desk’s foot with 3D-Printed adapters.
<table>
<thead>
<tr>
<th>Table D.1. All material used in this project</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ikea Svartåsen (desk)</td>
</tr>
<tr>
<td>Laptop (Dell XPS 9560 was used in this project)</td>
</tr>
<tr>
<td>Arduino Uno</td>
</tr>
<tr>
<td>Electrolytic capacitor, 20V 100 $\mu$F</td>
</tr>
<tr>
<td>Nema 17 stepper motor</td>
</tr>
<tr>
<td>DRV8825 stepper driver</td>
</tr>
<tr>
<td>Momentary switch used as endstop</td>
</tr>
<tr>
<td>Various 5 mm LEDs for tracking points</td>
</tr>
<tr>
<td>12V LED strip for tracking of table</td>
</tr>
<tr>
<td>HC-06 Bluetooth module</td>
</tr>
<tr>
<td>2 pieces of metal tubing, one fitting into the other used for the desk’s leg</td>
</tr>
<tr>
<td>Lead screw</td>
</tr>
<tr>
<td>Metal plates used for attaching lead screw nut to desk leg</td>
</tr>
<tr>
<td>Metal rod for making coupling</td>
</tr>
<tr>
<td>3D-Printer for making adapters, housings and other various parts</td>
</tr>
<tr>
<td>12V 2000mAh power supply</td>
</tr>
</tbody>
</table>