TUM Department of Electrical and Computer Engineering
Technical University of Munich

Practical Project

Real-time face tracking with a fully actuated robot head

Author: B.Sc. Primož Kočevar

Supervisor: M.Sc. Quentin Leboutet

February 24, 2018

Institute for Cognitive Systems
Technical University of Munich
Prof. Dr. Gordon Cheng

Contents

1. Introduction

The goal of this practical project is implementing real-time face tracking with a fully actuated robot head. Tracking a human face by a robot adds an important social dimension to the human computer interaction. It makes the interaction more natural and thus gives new ways thinking about interacting with a computer. However tracking a human face by a robot also has more technical implications. It provides better facial expression recognition and overall a better processing of a face. All of this is a consequence of an observation that humans usually move in the video stream. A robot that would demand for a person to be completely still to do facial expression recognition would be useless in many cases. Moving of the face can cause bad performance in recognition as the faces can move to the edge of the image or even completely out of the image frame. The proposed system tries to keep the face always in the centre of the image in real-time. This is achieved with face detection using haar features. Detection is using the video stream provided by a camera on the robot head. Processed information about the location of the face in the image is sent to the Arduino through serial communication. Arduino than converts this information to the actual movement of the robot head to different motors and gets feedback on this movement from rotational encoders.

In the first chapter more detailed architecture of the whole robot system is described. Face tracking and other functionalities implemented in Robot Operating System (ROS) [19] are given in chapter 3.1. Conversion of the tracking information to the actual movement of the robot head by motors is explained in chapters 3.2 and 3.3. The resulting performance of an implemented control loop is given in chapter 4.

This report is written as a documentation of work done during this project. Its main goal is to familiarize the reader with tools used and to guide him through the process of developing a working solution which contains many obstacles and challenges. Hopefully it will show the solutions to most of the problems and provide some basic knowledge on how systems and concepts that are used in this project actually work.

2. Design Goals

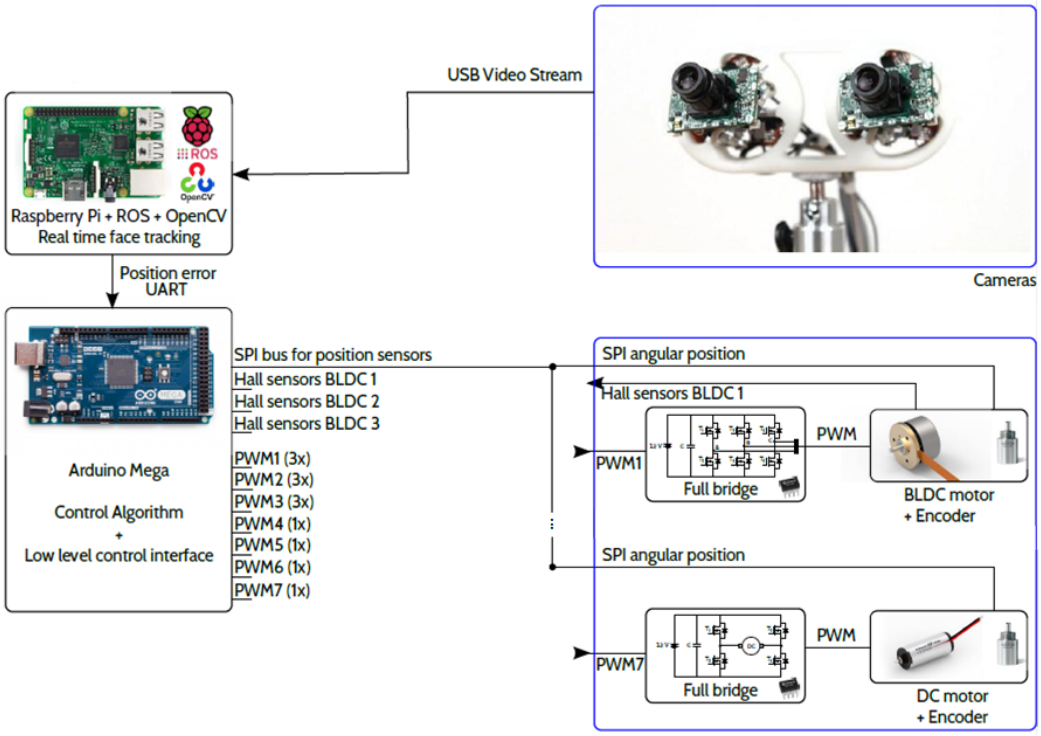

Figure 1 Architecture overview of design goals

The robot head used in this project has 7 actuated Degrees of Freedom (DoF): 3 for the neck and 2 for the eyes. Different kind of actuators are used for moving the robot head as Direct Current (DC) electric motors and Brushless Direct Current (BLDC) electric motors. Smaller DC motors are used to move the eyes of the robot head, which have less rotational inertia (J) compared to the whole head, therefore they require less torque. These motors are controlled using a Pulse Width Modulation (PWM) signal which is produced by an H-bridge. An H-bridge is used because even this small DC motors demand more current for their operation than an Arduino can provide. Bigger BLDC motors provide more torque for moving the whole head. BLDC control is more complicated as these motors also contain Hall sensors. Link angular positions are assessed using rotary encoders. These encoders communicate with the arduino low level control board using a custom 3-wire SPI interface.

As already mentioned, all this information is gathered by the Arduino board, more specifically Arduino Mega ADK [4] board using ATmega2560 microcontroller. On the Arduino a control loop is implemented which uses low level commands to control mentioned motors. This low level control algorithm takes as input the desired joint angle that needs to be achieved by the head in order to minimize the error between the current angle of the cameras and a desired angle. But to actually get information about a desired angle, another source is needed as an Arduino does not have great capabilities regarding face detection and tracking.

For this task a linux embedded computer [20] is used as it has hardware that is better suited for the task of real time face tracking. Also there are a lot of libraries already developed for face tracking that use OpenCV and ROS which can be installed on top of a Linux operating system that is recommended for a Raspberry Pi. This project uses an Ubuntu 16.04 LTS operating system and ROS Kinect on top as middleware. Image capture, image processing, face detection, error calculation, angle calculation are all parts of an architecture consisting of ROS nodes and topics. Implementation and functionalities of this architecture are further explained in section 3.1.4.

Linux embedded computer is connected to the Arduino through a Universal Serial Bus (USB) 2.0 interface. The same port is also used for debugging and programming the Arduino which brings some challenges. Alternatively universal asynchronous receiver-transmitter (UART) protocol could be used for communication between the Raspberry Pi and Arduino, which would leave serial port open for communication. In this case serial port is used for communication and blinking LEDs are used as a way of debugging.

For getting camera video stream to the Raspberry Pi, again a USB 2.0 connection is used as it is the most simple and common. To get camera stream information published onto a ROS topic, a special ROS node is required and implemented. Mentioning a small detail here is also important, that is our camera which is not a classic USB camera, but a specially developed camera. That brings some challenges which are described in more detail in section 3.1.4.

3. Implementation

In this chapter a more detailed view of the architecture is presented. All of different problems and obstacles that were discovered will be presented here, as well as most of the achieved solutions to these problems. It can also be noted that some problems were just too demanding and caused the development of this project to take a different approach in some areas. Therefore this chapter is the most important one if a reader wants to replicate the system or develop something similar, as it contains a lot of valuable hacks and solutions that had taken a lot of time to discover and were the most mind-boggling.

3.1. Face detection

3.1.1. Operating system of a Raspberry Pi

As programming a robot head contains a lot of inputs and outputs that have to work simultaneously a middleware ROS that was developed for similar projects is used. ROS is developed exactly this kind of scenarios when robots are programmed and operations are simultaneous. Choosing the correct operating system to work with Raspberry Pi and ROS is not as simple as it may seem. First choice when considering an operating system for the Raspberry Pi is naturally Raspbian [21] which is made for the Raspberry Pi. The recommended distribution of ROS to go with this operating system was ROS indigo. However, trying to install ROS on Raspbian proved more difficult than expected as a manual installation from the source was needed. Installing a version of ROS with only the basic packages can be achieved but to use the image view node for looking at the camera stream, which is crucial for this project, it is needed to have the full desktop version of ROS installed. The installation of the full desktop version of ROS proved very unstable and buggy. Because of these and many other different problems with ROS on , another approach was considered. As a few of the functionalities needed for face detection were already tested on the Ubuntu operating system on a laptop, installing Ubuntu on the Raspberry Pi was suggested. To support Indigo version of ROS, which was intended to be used at the beginning, Ubuntu 14.04 would have to be installed. As Raspberry Pi 3 does not support that version of Ubuntu, Ubuntu 16.04 LTS [22] is used. That also means it is not possible to use Indigo, but Kinetic version of ROS which supports different libraries for face detection. That brought some delay in this project as some of the face detection libraries working on ROS Indigo were already tested and adjusted.

3.1.2. ROS overview

Unlike what is suggested by its name, the ROS is not, strictly speaking, an "*operating system*" but rather a *powerful open-source, cross-platform and cross-language middleware environment* which is specifically designed and optimized for robotics applications. In practice, ROS provides a set of software libraries and tools which *considerably* accelerate the development of robot based applications. But its greatest asset lies without any doubt in its powerful gener-

Figure 2 Ubuntu 16.04 LTS operating system containing ROS Kinect middleware

alized TCP/IP *communication framework* allowing to easily and efficiently handle the variety of information transiting in a robot (sensor data, video, control orders and so on). ROS is built around two main components, namely:

1. the core software [16], developed and maintained by the open-source foundation WILLOW GARAGE which provides the both communication network and the general system architecture.
2. a huge database of task-specific libraries and packages, freely accessible on the ROS website and maintained by a strong community made of different academic, industrial or even individual actors.

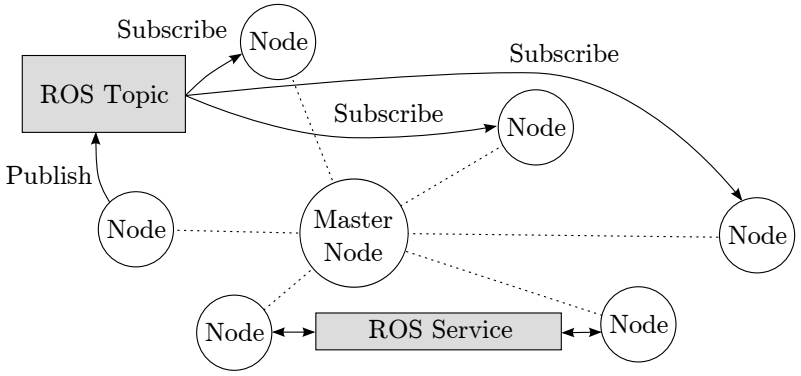

Figure 3 Some components of the ROS architecture.

The P2P-like ROS communication network is structured around a few key concepts, allowing both flexibility and ease of use:

- **Nodes** are the building blocks of any ROS architecture: they can be defined as a set of "*process performing computation*" [16] for a given ROS session. A robot generally contains many nodes, each one performing a set of various tasks such as path planning, inverse dynamics computation or sensor data processing. Nodes get to know one another via the "*master*". As depicted in Figure 3, they can communicate with each other by transmitting or receiving "*messages*" via a set of so called "*topics*" or "*services*". Dedicated software libraries are made available to the user so new nodes can be easily developed in C++ or in PYTHON.

- **Messages** can be defined as "*strictly typed data structures*" [16] that are transmitted between the different nodes. These structures may for example contain arrays of signed and/or unsigned integers (8/16/32 bits), floats (32/64 bits), boolean, strings and so on. Additional application-oriented types can be found in many packages: these new types being in fact a combination of several standard types.

- **Topics** can be defined as *asynchronous communication protocols* between the different nodes. An interesting characteristic of topics is that they "*decouple the production of information from its consumption*" [16] which is particularly useful in robotics – especially for computer vision purpose – when data cannot be obtained "*on request*", but in stead, comes as a continuous stream. A node can either connect with a topic as a "*publisher*" (in order broadcast a data stream), or as a "*subscriber*" (in order to receive data from other topics). An arbitrary number of publishers and subscribers can be generated with a single topic.

- **Services** are the *synchronous alternative to topics*, taking the form of a remote procedure call ("*request/answer*") instead of a "*publish/subscribe*" model. By using a service, a given node will only receive data on request. It is interesting to notice that the received data may change with the request, allowing a good flexibility.

- **ROS Master** is a special node which keeps track of the different publishers and subscriber to a topic as well as the different service requests in order to allow nodes to locate one another. It has to be launched before any other ROS process by simply entering the "`roscore`" command in a terminal.

3.1.3. Face detection overview

Face detection in this project is achieved using the OpenCV library. OpenCV initially comes with each ROS distribution and can therefore be easily integrated to any robot vision problem. Face detection is achieved using a Cascade Classifier with Haar-like features is used. Object Detection using Haar feature-based cascade classifiers is an effective object detection method [23]. It is a machine learning based approach where a cascade function is trained from a lot of positive and negative images. It is then used to detect objects in other images.

Initially, the algorithm needs a lot of positive images (images of faces) and negative images (images without faces) to train the classifier. A set of features are extracted from the picture. For this, haar features shown in Figure 4 are used. They work just like the convolution kernel, which is applied to a certain number of neighbour pixels to get the value of one pixel. Each feature is a single value obtained by subtracting sum of pixels under white rectangle from sum of pixels under black rectangle[7].

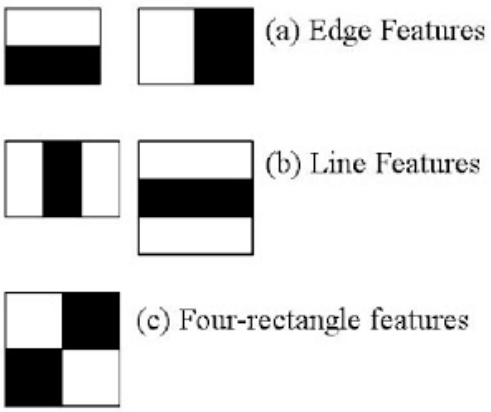

Figure 4 Different haar features [7].

Features are provided in different XML files which are added in the project beside the algorithm code as they initialize cascade classifiers. These XML files define the objects we want to detect and are different for different shape of faces, profiles and different facial features. These files were created by training machine learning algorithms on hundreds or even thousands of images that contain a face or not.

3.1.4. Implemented ROS nodes and topics

In this section an implemented architecture of ROS nodes and topics are presented and later functions of each of these nodes are defined. A final architecture of nodes and topics in ROS can be seen in Figure 5.

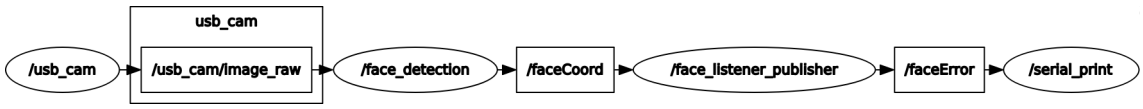

Figure 5 Architecture of ROS nodes and topics

Starting with the first node, which is responsible for getting the image stream from the USB camera. This node can be implemented in two ways, corresponding to the camera used. At first a regular USB camera was used for testing and this camera worked with the most common and developed ROS node for USB cameras [17]. However when trying to use the same approach for a camera that is provided on the robot, the classic USB node does not work. Camera used on the robot is a Pointgrey Research Firefly MV FMVU-03MTC with a Micron MT9V022177ATC sensor and an uncommon resolution of 752x480 [9]. The result of this unusual resolution is also a bit lower FPS when doing face tracking from the video stream. For this camera a custom node was implemented [18] and recommendations from [10] were used to get this camera working as it did not work by automatically installing this driver as recommended here [15]. At this point the camera was working on a ROS environment on a linux PC which was used for testing and development. When trying to port this functionality to an ARM-based Raspberry Pi big problems were encountered. It turns out that this camera is not made to work on the Raspberry Pi as its ARM drivers only support boards such as the BeagleBone and not the Raspberry Pi [8].

Now that a video stream is published on an `/usb_cam/image_raw` ROS topic it can be subscribed by a node that performs the face detection routine. This `face_detection` node contains all of the face detection implementation in this system. Firstly a python library for face detection in OpenCV was tested on a PC using ROS indigo. Face detection was achieved at 30 FPS with camera resolution set to 320x480 as recommended. But as discussed in section 3.1.1, when the distribution of ROS changed to Kinect because of problems on the Raspberry Pi, this python library no longer worked. The reason for this is, that it is using an older version of OpenCV v2, but ROS Kinect only supports OpenCV v3. As rewriting a whole python library was not an efficient solution, a C++ implementation of face detection in OpenCV is used. It turns out this implementation is even more efficient than the python one and gives better results concerning FPS at a given resolution, which is very important for a relatively low power computer as a Raspberry Pi.

After `/face_detection` node detects faces in the video stream it publishes the results of this process to the `/faceCoord` topic in the form of coordinates and sizes of detected faces. A problem of multiple faces detected is also solved here. For the purposes of this project only one face has to be detected as a robot head can only track one face at a time. Taking only

the face that is detected first and with the most probability is a way of ensuring always having only one face to track. This is done by only taking the first few parameters which correspond to the first face from the detection vector and forgetting all others.All of this is happening in the `/face_listener_publisher` node which also publishes error vectors to a `/faceError` topic. This error vectors are calculated from face coordinates provided by a `/faceCoord` topic as shown in Figure 6.

Figure 6 A sketch of error vector calculation from a video stream.

Coordinates colored blue, dimensions of the detected face colored green and image resolution colored red all present an output of a `/faceCoord` topic. This information is then easily calculated and an error vector is given as a result in the following way:

$$xError = faceXcoord + W/2 - ImgResX/2 \tag{3.1}$$

$$yError = faceYcoord + H/2 - ImgResY/2 \tag{3.2}$$

This error vector (`[xError,yError]`) presents a length in pixels for which a detected face is shifted from the center of the screen in each axis. Naturally minimizing this error to `[0,0]` is intended as at this error value face is exactly in the middle of the video stream which means the cameras are exactly aligned and thus tracking the face successfully.

Node `/face_listener_publisher` also publishes a desired angle to be achieved by the actuators for a separate coordinates X and Y to the `/faceError` topic. Desired angle is for now simply calculated from the error in each direction by scaling the values of the errors to angle values that can be achieved by the joint controlling the angle of the camera. However,

sent desired angle is calculated as a relative angle according to the current position of the camera. As our control loop takes in an angle that is absolute and has a position zero always in the same place, transforming the relative angle is needed. This absolute desired angle is later used as an input in a control loop controlling the position of the cameras presented in section 3.2.6.

A more advanced solution for calculating a desired angle using width and height of the face as a quasi third dimension was also proposed, but it did not bring better results when tracking a face therefore a simpler solution is used. At this point it should be mentioned that also a more advanced method of calculation and control could be used in this node. This is the node that could contain a more smart way of moving the motor by calculating Jacobian matrices and making the movement of the robot head more distributed as described in section 3.3.2. But this way of implementing things will be left for future work.

The last node implemented in this architecture is a `/serial_print` node which is responsible for sending the processed data through a serial connection from the Raspberry Pi to the Arduino. This node is separately implemented in a python package as a contrast to all other nodes which are written in C++ and all a part of the same face_detection ROS package. A reason for this is that a serial connection is much easier achieved in python with the help of a library called pyserial. Here a communication speed must be set high enough as it can bottleneck the whole system of nodes if the FPS of the video stream and the face detection becomes faster than this communication. For example a baud rate of 9600 can be a bit slow if face detection works really well and FPS are high 30 in which case the whole system could start failing, as was observed during this project. A baud rate of 115200 should be set just to be sure the bottlenecks do not arise from a serial communication between the Raspberry Pi and the Arduino.

In this last node data is also packed in JSON format and sent through serial communication. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C, Java, JavaScript, Perl, Python, and many others [13]. More specifically, data is sent as a JSON object whose structure can be seen in Figure 7. This JSON object is then easily parsed on the Arduino and data can be extracted.

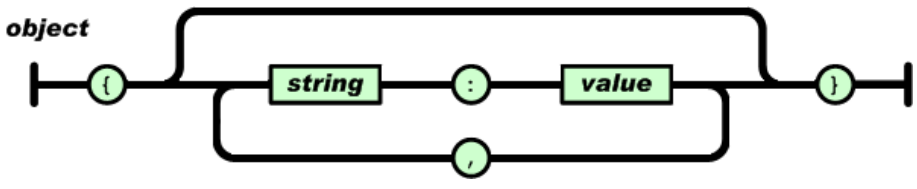

Figure 7 A structure of a JSON object [13].

A few notes can be taken here regarding Raspberry Pi and ROS. When compiling catkin packages in ROS with `catkin_make` command it is obligatory to assign more swap space on the Raspberry Pi as the 1 GB of RAM it has will quickly be used up by the demanding compilations of face detection libraries. Also very important to note is that a command for compiling should be `catkin_make -j2` where a `-j2` option is added for specifying the maximal number of jobs at the same time to 2. As the default number is 4 and a Raspberry Pi 3 has 4 cores it crashes when demanding compilations are needed and the default option is enabled.

Implementing this ROS system of nodes and topics on the Raspberry Pi brought mentioned problems like camera incompatibility with the Raspberry Pi, instability of ROS and Ubuntu combination on the Raspberry Pi (producing Kernel errors), constant problems with compilation on the Raspberry Pi and problems with the first implementation of python face detection library. All of this pushed further development to be only on a PC containing ROS and Ubuntu 16.04 which was meant only for testing but things worked perfectly on it. This decision was made only after many days/weeks spent solving mentioned issues which were produced only by porting already working solutions from a PC to the Raspberry Pi, where the same solutions did not work.

3.2. Control of the eyes movement

In this section a more detailed view of the whole system for controlling the movement of the eyes is given. As already mentioned in chapter 2 DC motors in this project are used for controlling eye movements of the robot head. Eyes have much smaller rotational inertia and so require less torque which is provided by smaller DC motors. DC motors are controlled by Pulse-width modulation (PWM) signals sent from the Arduino PWM enabled ports through an H-bridge.

3.2.1. Arduino overview

Arduino is an inexpensive, commercially available electronic board with a microcontroller and some I/O capabilities. It exists in various versions, that share the same, simple programming language. The huge success of Arduino, with respect to other microcontroller boards, was due to the fact that both hardware and software were released as Open Source projects: you can read, study and even expand its capabilities both in terms of software as well as in terms of hardware. All the information are shared under the Creative Commons Attribution-ShareAlike 3.0 License [11].

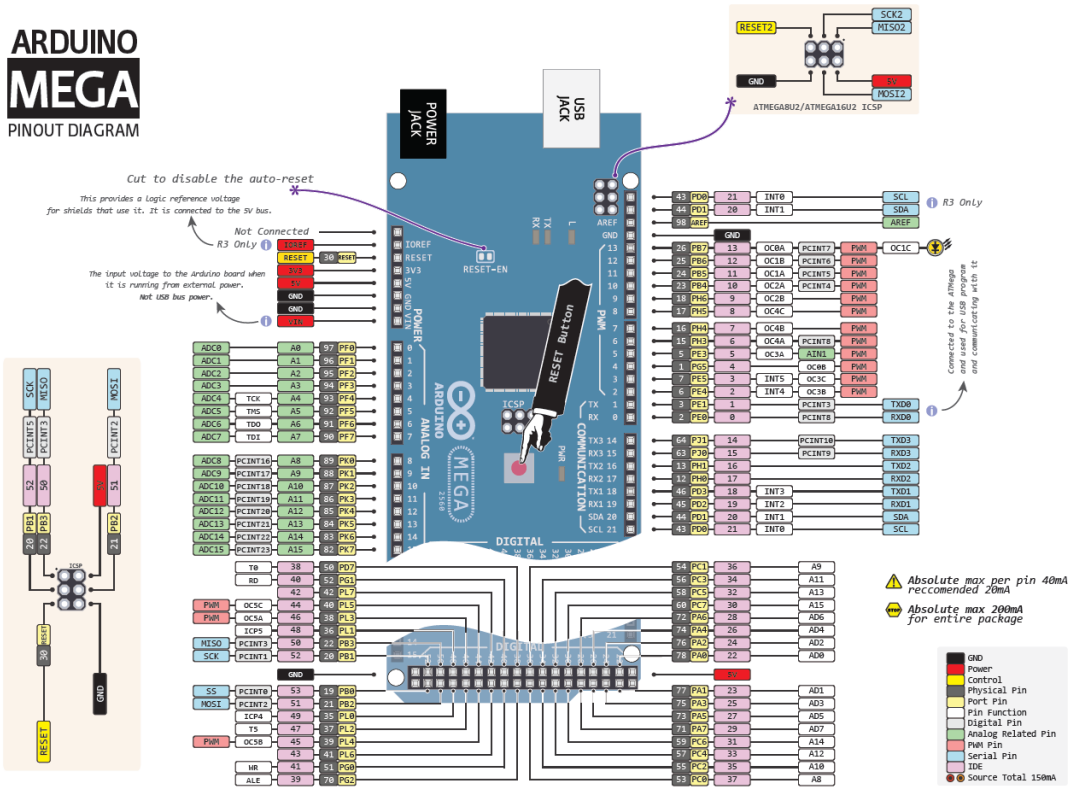

Figure 8 Arduino mega pinout diagram [2].

In this project a Mega ADK version of the Arduino is used. The Arduino MEGA ADK is a microcontroller board based on the ATmega2560. It has a USB host interface to connect with Android based phones, based on the MAX3421e IC. It has 54 digital input/output pins (of

which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs(hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button [1]. To program and use pins on the Arduino with C++ using registers, understanding of the architecture of ports and lines is crucial and can be seen in Figure 8.

3.2.2. DC motors

Electrical motors can be presented as energy converters that convert electrical power $P_{el} = U \cdot I$ to mechanical power $P_{mech} = \tau\omega$ plus some produced heat by Joule power losses $P_{heat} = R_r {i_r}^2$ of the winding (resistance R_r)[14]. Mentioned power balance can be presented in the mathematical form:

$$P_{elec} = P_{heat} + P_{mech} = R_r {i_r}^2 + \tau\omega \tag{3.3}$$

$$V_r = R_r i_r + L_r \frac{di_r}{dt} \tag{3.4}$$

$$e = \frac{n}{K_v} \tag{3.5}$$

$$\tau = K_t i_r \tag{3.6}$$

This formulas are illustrated in Figure 9. Generally it can be observed from (3.5) and (3.6) that torque depends linearly on current and rotational speed depends linearly on voltage.

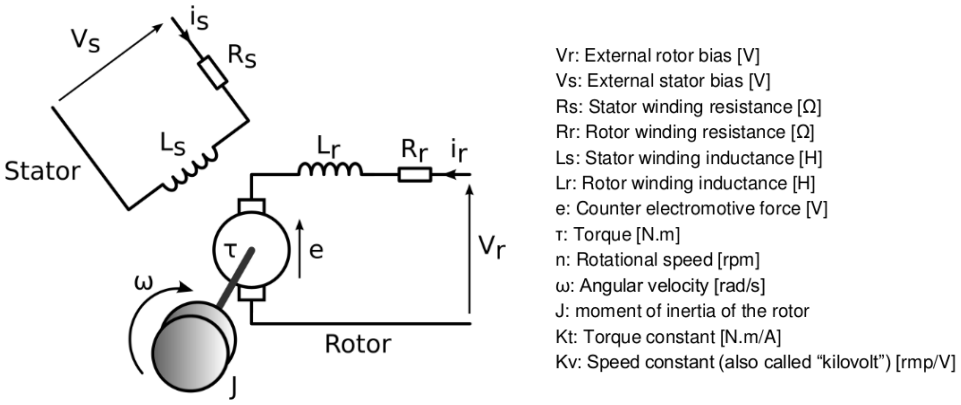

Figure 9 Theoretical schematic of a DC motor.

Induced voltage e that is a result of rotational speed n also brings some problems when trying to control the motors with switching the voltage ON and then OFF. When voltage in turned off on the motor pins, which is done by a transistor T in Figure 10, motor because of its rotational inertia J retains some rotational speed n which produces some induced voltage e in the opposite direction. Current that is produced by this voltage in the opposite direction could seriously damage other parts of this circuit (e.g. a power supply, switching semiconductors...). That is why a free-wheeling diode D is added that limits the current in the opposite direction. Effect of this diode D can be seen in Figure 11.

This is exactly what is done with PWM signals only in a faster manner, however a more

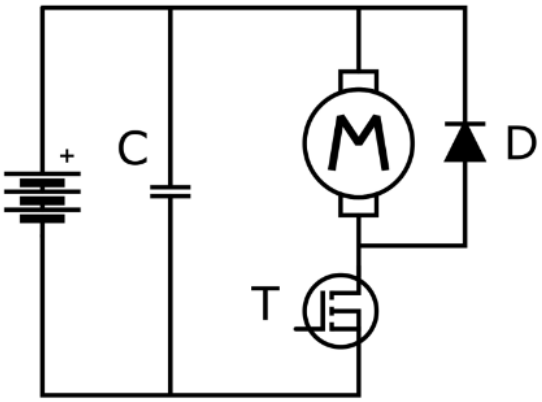

Figure 10 Model circuit for controlling a DC motor.

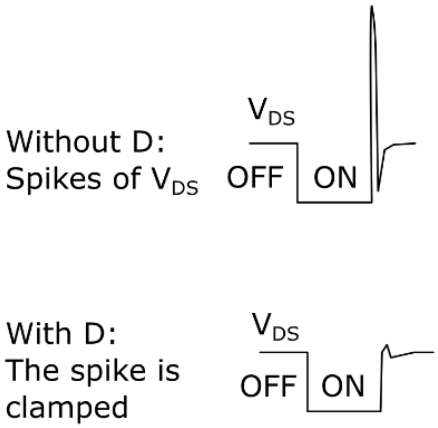

Figure 11 Difference in voltage with and without a diode during switching.

detailed view of PWM control will be given in section 3.2.3.

3.2.3. PWM generation for DC motor control

Considering an equation (3.6) that says torque is directly related to the amount of voltage provided to the motor, if a mean of voltage provided can be changed than also the amount of torque produced by the motor can be controlled. When setting a PWM signal duty-cycle from 0-100% as seen on Figures 12, 13, 14 and 15 mean of the voltage signal produced is changed proportionally. That is the reason a PWM signal can control an amount of torque a DC motor produces with its duty cycle setting.

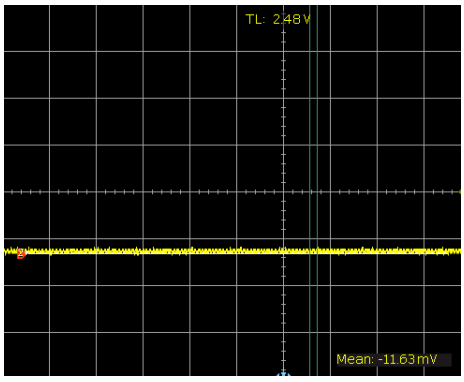

Figure 12 PWM with 0% duty cycle.

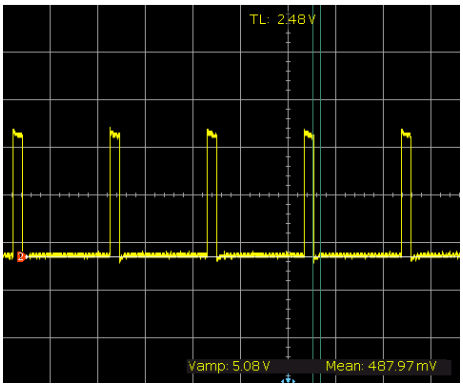

Figure 13 PWM with 10% duty cycle.

When generating PWM signals for controlling a DC motor with the Arduino built in function analogWrite(), it generates a PWM signal of 500 Hz. This frequency does not work well with DC electrical motors as it produces harsh motions and noise because the motor inductance works as a low-pass filter. That is why a period of a PWM signal should be much smaller than the rise time of the voltage on the motor because of the motor inductance as seen in Figure 16. Here it can be seen that voltage applied to the DC motor is much smoother when a period of a PWM signal is much smaller than the rise time ($T >> T_{PWM}$).

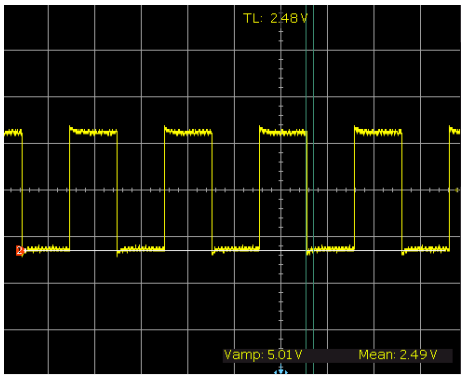

Figure 14 PWM with 50% duty cycle.

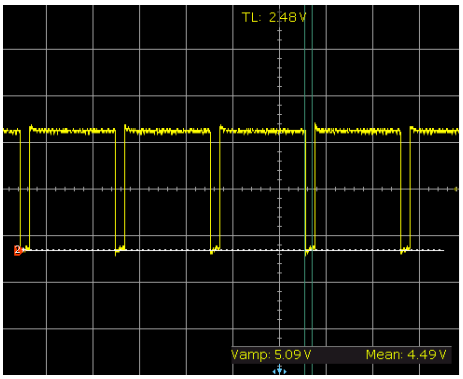

Figure 15 PWM with 90% duty cycle.

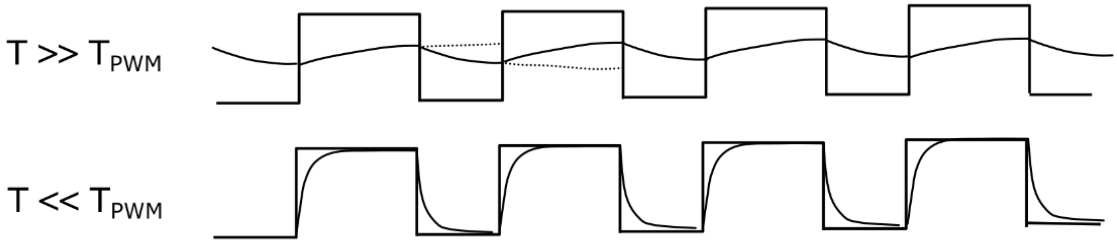

Figure 16 PWM switching effecting the DC motor voltage according to the period of the switching.

Equivalently this means the frequency of a PWM signal should be much higher, preferably greater than $16kHz$. However, if a PWM signal frequency is high, that also produces higher losses in the MOSFET that is responsible for the switching. A reason for this losses can be described with Figure 17. Switching losses come from the initial energy required to charge the MOSFET gate at each switch. As seen in Figure 17, when the gate charge grows from 5-8 nC no gate source voltage increase is seen. This presents a loss in energy at each ON and OFF switch. If these switches occur commonly as in the case of high frequency then this losses add up to a considerable amount. Therefore a compromise between losses and smoothness must be made regarding the choice of PWM frequency.

Frequency control of a PWM signal for smoother torque of the motors with Arduino Mega 2560 is explained in the following forum post [3]. Frequency of PWM in the case of this project is changed to $31kHz$, which provides much more linear torque supply and does not make any noise. Also a switching MOSFET does not get too hot which indicates switching losses are not too high and the choice of frequency was correct.

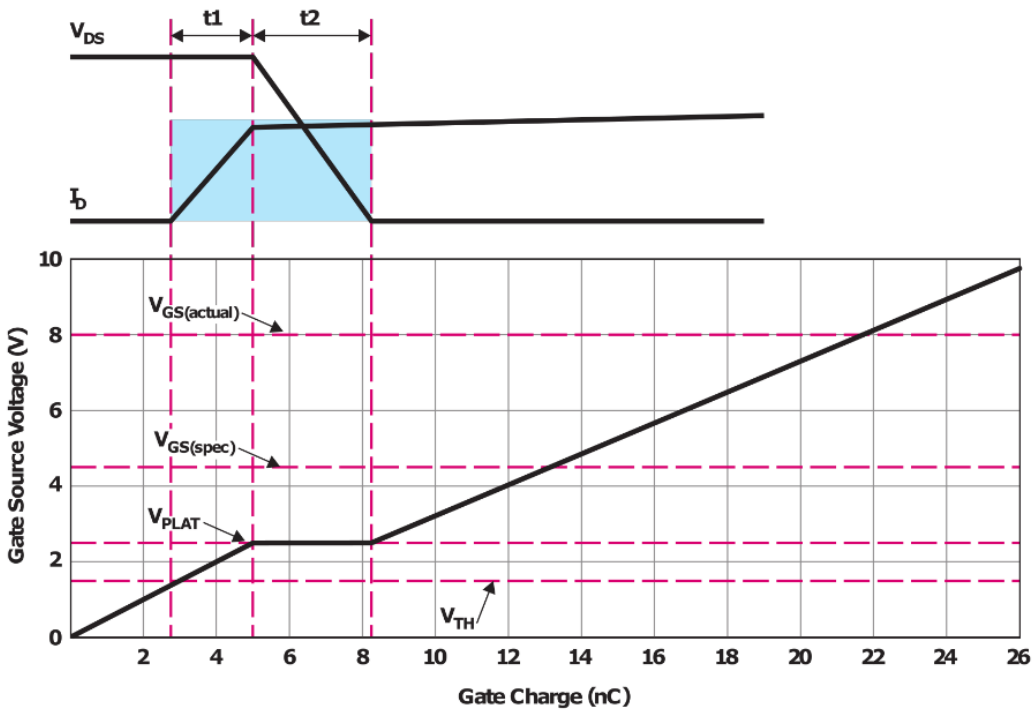

Figure 17 Losses in a transistor caused by a switch.

3.2.4. Controlling the motors with Arduino through an H-bridge

As already discussed in previous chapters, to control DC motors, transistors for switching and diodes for leveling the voltage spikes are needed. As in most cases where electrical motors must be controlled, for this project too a bidirectional movement is needed. To move the motor in both directions a more complex but still simple circuit of 4 transistors and 4 corresponding diodes is implemented. This kind of circuit can be observed in Figure 18. One direction of movement in this case can be achieved by enabling transistor T_1 and T_4. Alternatively a direction of movement opposite to the first one can be achieved by enabling transistors T_2 and T_3. In this case it can be observed that the current flow through the motor is now reversed compared to the first movement.

This circuit is easily implemented with the L298N Dual H-Bridge Motor Driver [12], which already contains needed transistors and diodes. This motor driver is capable of outputting 5-35V with peaks up to 2A which is more than sufficient for used DC motors. It also contains a big heat-sink for cooling down the H-bridge that is doing the switching. An even more important reason to use this motor driver is the dual H-bridge it contains [6]. This H-bridge allows the Arduino to control the motor safely with a 5 V PWM signal that is than replicated by the H-bridge on the output with a 5-35 V amplitude. This voltage is supplied to the H-bridge by an external source and it allows for the motor to use more current than the Arduino can supply. This is very helpful as an Arduino can only supply approximately 20 mA of current at 5 V which is not enough for most motors and therefore should not be used directly to control the motors.

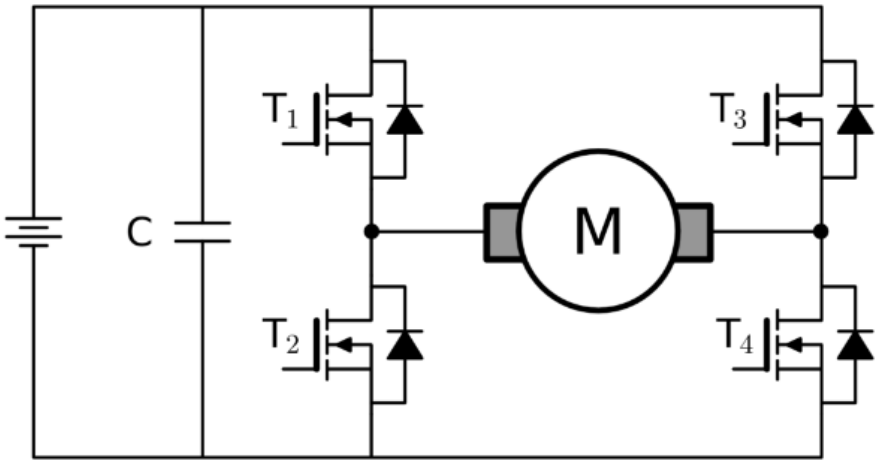

Figure 18 Schematic of an H-bridge controlling the directions of a DC motor

An important remark must be made here regarding the power supply and the Arduino connection to the motor driver. It is very important for the functioning of the motor driver that the ground line of the external power supply and the Arduino be connected so that they work on the same electrical potential.

Another remark must be made regarding the actual implementation of rotary limbs and motors on the robot head. When first running the DC motors at full torque problems accured in the joints responsible for rotating the eyes in a horizontal manner. Too much friction was produced in these joints by an old rubber between the main frame and the moving camera frame. This problem was resolved after applying a lot of WD-40 to the joints and completely removing the old rubber by disassembling the whole robot eye. Also observed was that the friction could be adjusted by tightening a screw which could prove useful later when setting the PID parameters.

3.2.5. SPI communication with encoders

Providing feedback on the effect of DC motors are encoders that act as angle sensors. They provide information on the effect of DC motors on angles of different joints on the robot head. In this project high precision Contelec Vert-X 13 encoders [5] are used. Communication with these encoders is done by a special 3-wire Serial Peripheral Interface Bus (SPI) as shown in Figure 19. This interface is different from a classic SPI interface, such that it only has one Data line for communication between master and slaves which makes this a half-duplex communication. As observed from the datasheet and Figure 19 data in this communication is captured on the falling edge (CPOL=0) and SCLK line is LOW when idle and HIGH when active (CPHA=1). A combination of these two parameters defines a SPI_MODE1 way of communication. Another important fact for communication is that data bits from the slave are sent Most Significant Bit (MSB) first. All of these parameters are necessary for making an

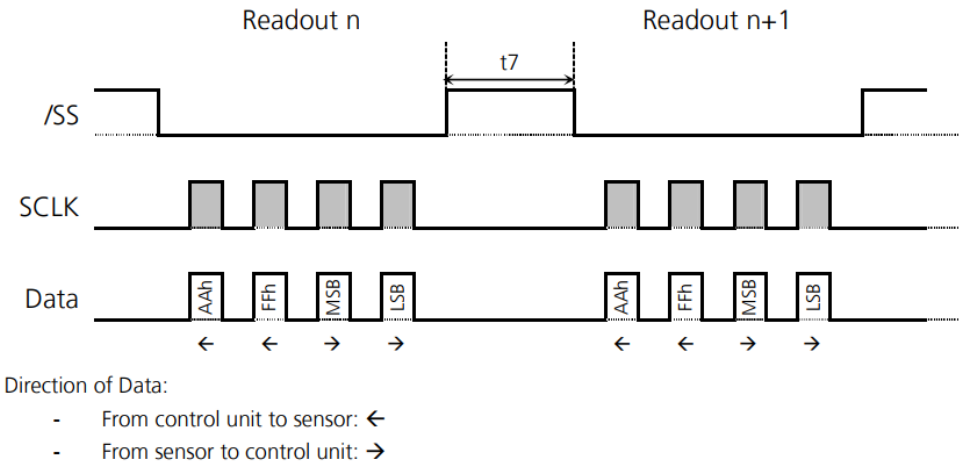

Figure 19 3-wire SPI communication procedure.

SPI connection work.

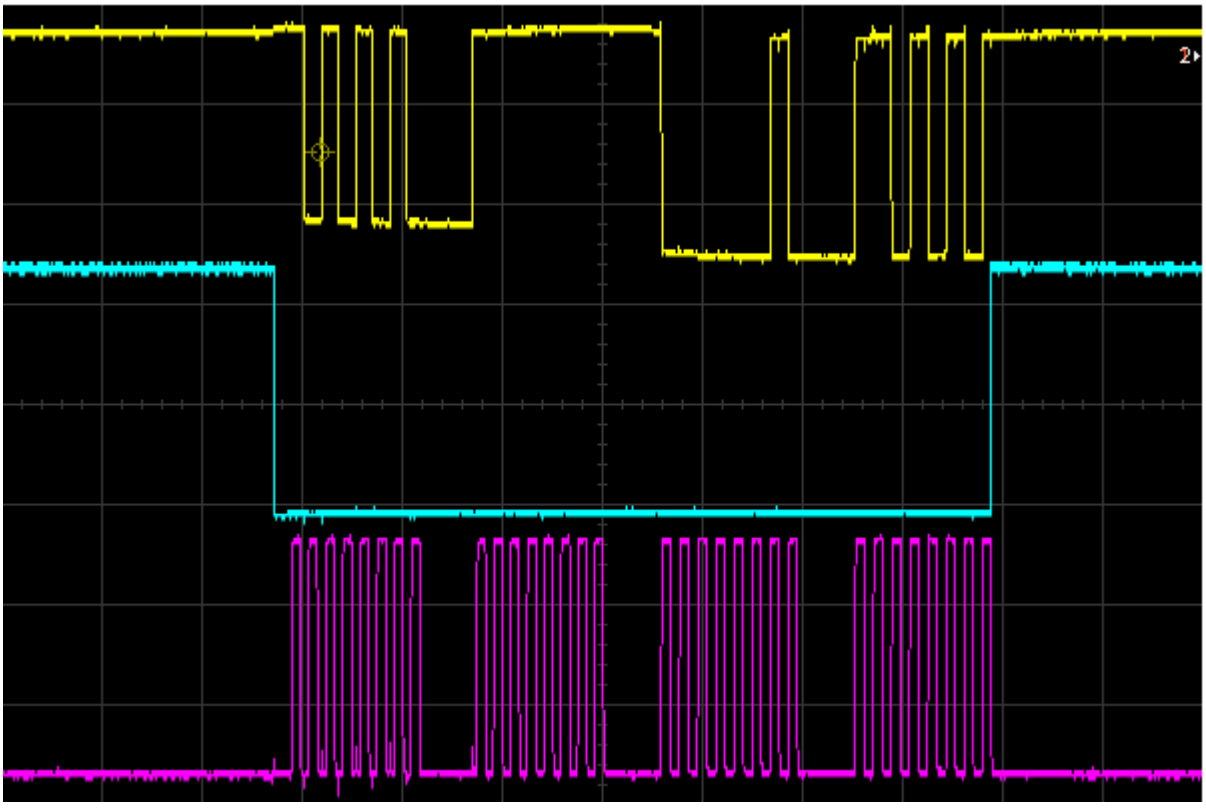

Figure 20 Timeline of a working 3-wire SPI connection. Yellow - data, green - slave select, pink - clock

In the case of a 3-wire SPI communication used by Contelec encoders, setting up a working communication is a bigger challenge than expected. There are no already developed libraries working with this kind of sensors through this odd interface. Therefore a custom 3-wire, half-duplex SPI library had to be developed which produced a lot of headaches as the documentation is really basic. Library is written in C++ using registers of pins to set the states and delays to time the communication correctly.

All of this development had to be debugged with an oscilloscope until it finally worked as shown in Figure 20. Observing the yellow data line, in the first part synchronizing bits sent from the Master to the Slave can be seen. Alternating bits present 0xAA and constant HIGH presents 0xFF in hexadecimal format. In the second part the actual data bits are sent from the Slave, which has Slave Select line set to LOW, to the Master where they are then decoded in to degrees by C++ code. Speed of this communication is set by the clock line in pink.

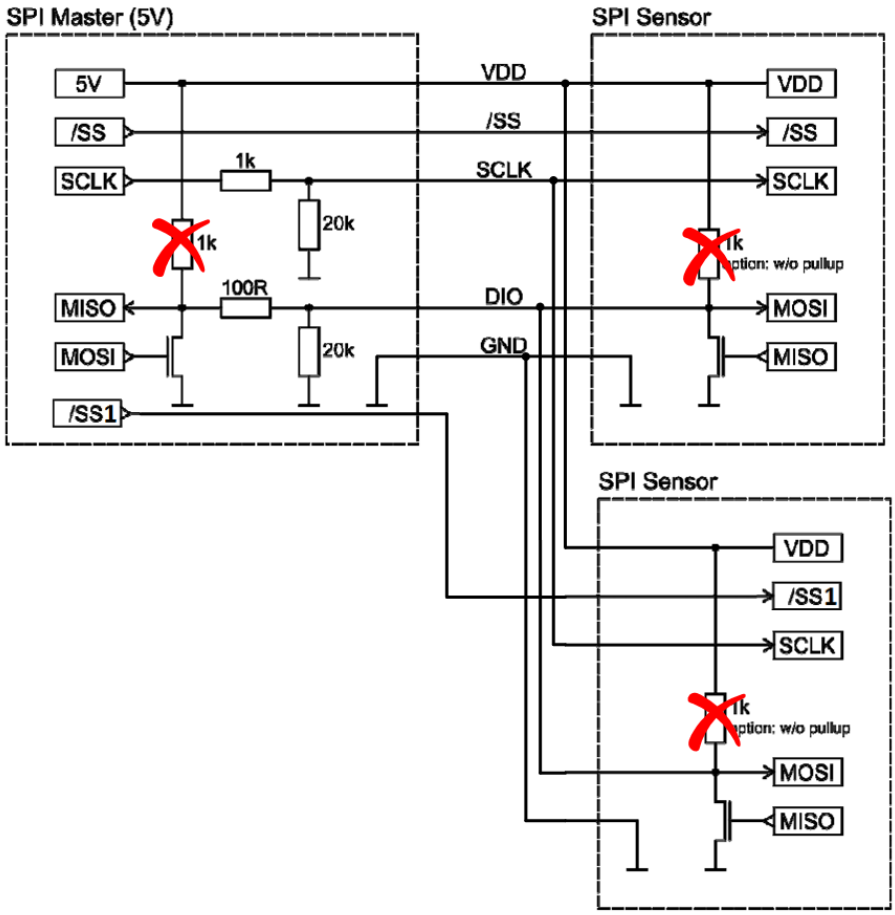

Figure 21 Implemented circuit for 3-wire SPI communication. Some pull-up resistors are removed from the data sheet implementation for multiple slaves communication.

Applying this custom written library to a single master and single slave architecture worked perfectly. However after testing it with multiple slaves some problems arised as communication started failing when adding another slave. This issue took some time to figure out as problem was searched for in the code, but was actually in the hardware. When a multiple slave architecture was implemented as shown in Figure 21 everything worked perfectly without any change to the code. A crucial change in the implementation was removing the pull-up resistors indicated by red crosses.

3.2.6. Control loop

Firstly a crucial part of a control loop, a PID controller, will be more generally described. A PID controller in a control loop as in Figure 23 can be presented more specifically by equations:

$$P_{component} = K_p e(t). \tag{3.7}$$

$$I_{component} = K_i \int_0^t e(\tau) d\tau. \tag{3.8}$$

$$D_{component} = K_d \frac{de(t)}{dt}. \tag{3.9}$$

$$u(t) = P_{component} + I_{component} + D_{component}. \tag{3.10}$$

Here it can be seen a PID controller uses 3 different parameters for adjusting the effect usually an error has on the output of the system (denoted by $u(t)$. All three of these parameters have a different meaning and are used for completely different corrections of the response. Firstly, P parameter is very simple as it only multiplies the error by a certain amount. This can be good enough for some use-cases but for others it can be too slow in reaching the desired state. That is where I parameter is used for reaching the desired state faster. But that can also have its side effects as it can produce some oscillations and instability. This is where D parameter finds its place as it acts as a damper and dampens the oscillations and instability of the response. However, too much dampening can produce an offset in the response which is again removed by increasing the I parameter. As can be observed this could be a lengthy process with many iterations as a great setting for one parameter could ruin the setting of the other. That is why a good choice of these parameters is the most important.

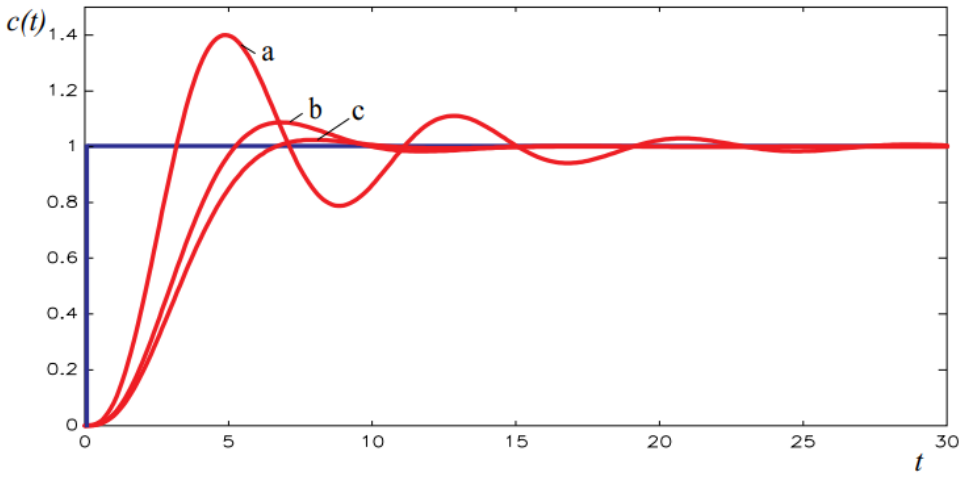

Figure 22 Effect of different P components to the response $c(t)$. a) $K_P = 0.7$, b) $K_P = 0.39$, c) $K_P = 0.32$.[24]

In Figure 22 it can be seen that a good response could be achieved by only setting the P parameter correctly, however this can not be applied to all cases. Keeping in mind that friction often acts as a damper and thus as a D component in real-life systems which suggests usage of an I parameter to solve the problem of offset. As later discussed, that kind of natural D

parameter is actually present in this project.

In this section all parts of the hardware system are going to be put together in to a control loop as seen on Figure 23. Input of this control loop is a desired angle which is provided through serial connection in a JSON form as already discussed in section 3.1.4. Desired angle is given separately for each axis. Therefore every axis and limb of the robot head has its own unique control loop as frictions and rotational inertia's of all limbs are very different. This is also the reason every control loop has its own unique PID parameters settings, which is a bit of a nuisance when trying to make this kind of system work.

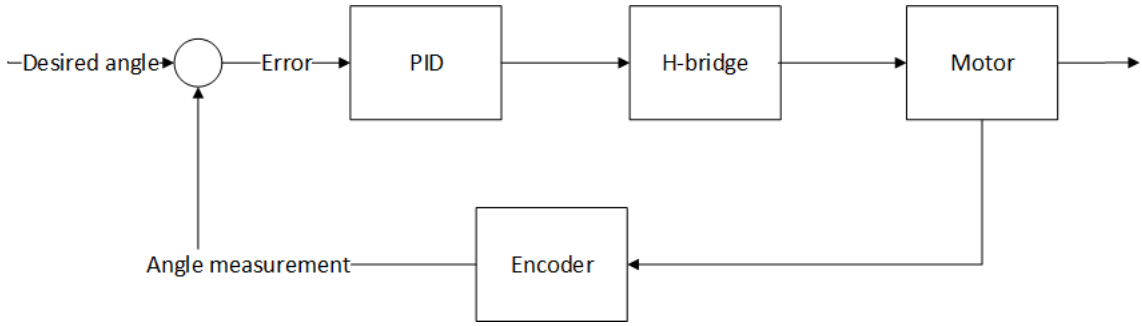

Figure 23 Feedback loop

A control loop in this project is implemented very simply as a single loop of consecutive operations. Firstly as mentioned, serial data is acquired from the USB bus and a desired angle is parsed from a JSON object. However, as already mentioned, this angle is relative to the position of the camera and control loop takes in an absolute value as current angles are in an absolute form. That is simply done by summing a desired relative angle up with the current angle from the encoder. But there is another trick here. As frequency of new frames is very slow (8 FPS at 752x480), fresh feedback in between this frames is needed from the encoders for the system to work optimally. This is done by setting a global variable desired angle that is constant during one frame and the motors are working faster during this one frame to reach this absolute angle. Then when a new frame arrives, a desired angle is updated and the process repeats. This is a way to partially achieve functioning almost like two different loops with different frequencies working simultaneously in one actual loop, which gives much better results.

Next this desired angle is compared to an actual measured angle and an error is calculated. This error is than saved in a vector of errors for calculating I and D parameters of the controller. Depending on the sign of the error a direction of the motor torque is set by the H-bridge as already mentioned in section 3.2.4. According to the set PID parameters a PWM signal is produced for controlling the motors. This PWM signal is then applied to a DC motor through an H-bridge and converted into torque. The effect of this torque is then observed by the encoder that measures a new angle. This angle is than again compared to a desired angle, which could be already updated or not, and therefore a new error is produced and thus a control loop is repeated. Here it should be mentioned that in the case of this project a PI

controller is used as D component of the controller is already provided by the friction between the joints on the robot head.

Very important parameter with this kind of control loops is a frequency or speed of these loops. Implementing a fast control loop in Arduino is sometimes challenging as functions that provide analog output capabilities are simple to use, but sacrifice performance. That is why writing the outputs manually with registers of ports and lines is sometimes needed if high speed is to be achieved. A big performance bottleneck are also different serial write functions that are commonly used in Arduino for debugging. Getting rid of all this could provide great improvements in performance, but can also be much more challenging to actually program than just using simple libraries.

However, in the case of this project a control frequency of 770 Hz was achieved without making too much changes to the original code and doing a lot of optimization. This is certainly not a high frequency, but for this control loop it works perfectly and higher frequencies are not needed. The baud rate of the serial communication used for the input to the control loop was 115200 bps. As this frequency is high enough, there is no need for a complex architecture of multiple loops and everything can be achieved by a single loop. However, if better performance would be needed the same functionality could be achieved by implementing two loops. One would be slower and would be responsible for serial communication and at each new desired angle it would update a common variable. The second loop would only be for controlling the angle of the joint and would not have to wait for any communication to pass. This loop could than be really fast and this would be most efficiently implemented by interrupts.

3.3. Control of the neck motions

3.3.1. BLDC Overview

Brushless Direct Current (BLDC) motors are electrical DC motors that do not include brushes for applying current to the rotor. They are electronically commutated so that a AC electric current can be achieved for running the motor. Instead of applying brushes to the motor, they include permanent magnets on the rotor which provide for a synchronous running or they use an inductance rotors that run asynchronously. But usually the permanent magnets on the rotor are used and this kind of motors will be described. A construction of this kind of a motor can be seen in Figure 24.

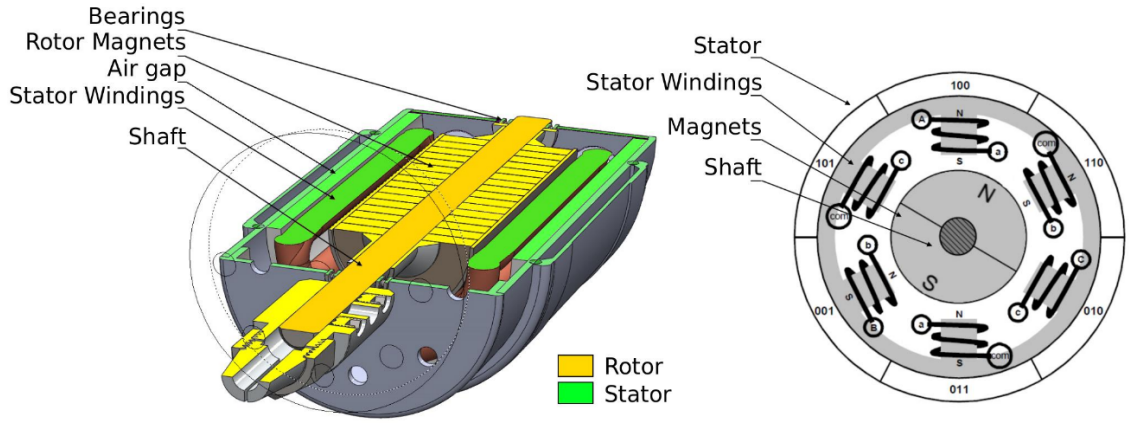

Figure 24 Shematic of a BLDC motor.

Because of this kind of construction that does not include brushes which can be worn out, these motors are more resilient to aging and thus more reliable. They also provide reduced Electro Magnetic Interferance and produce low noise. BLDC motors are also more power efficient and thus provide more torque and achieve higher speeds than regular DC motors that use brushes. Because their design is more complex and often also uses Hall sensors for control, these motors have more control over position and velocity. Sensors and electric controllers that control the stator coils currents are presented in Figure 25.

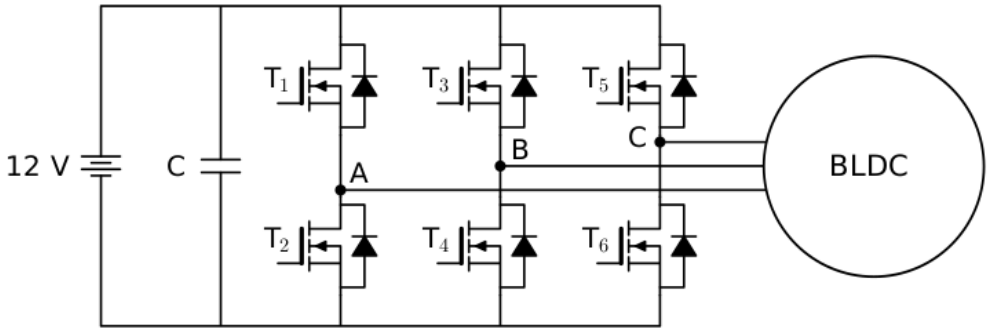

Figure 25 bLDC Motor connected to a three-phase integrated half-H-bridge driver

Even if Hall sensor is not included in the motor, information about the rotor position can be

gathered from the inducted current in the stator coils. These motors are then called Sensorless BLDC motors. It should be mentioned that this sensors are usually not controlled only by PWM signals but by different protocols that receive codes from the controller. A code only determines which transistors are going to be enabled and thus which coils are going to be active. Each code stands for a unique position of a rotor and thus by repeatedly sending different codes we can control the motor.

BLDC electrical motors are used for moving the neck and the eyes joints that contain DC motors. These are bigger and stronger motors that are meant for slower movements of the whole system.

3.3.2. Distribution of eyes and neck motions

DC motors in these project would be responsible for faster motions and BLDC motors would move slower. This is again connected to rotational inertia of different parts of the robot head and naturally a desired angle would maximally be achieved with faster and weaker DC motors and only a slow change of the angle would be added by BLDC motors. This would be implemented by a matrix that would map the desired movements to the different joints and different motors depending on the rotational inertia of the joints. Rotational inertia would act as a parameter in optimizing the movement so that the required angle is achieved as efficiently as possible and thus moving the joints with the most rotational inertia the least.

However as there was no more time for implementing the functioning of BLDC motors, and thus the distribution of motions upon different motors was not possible. Unfortunately implementation of this will have to be left for future work.

4. Results

In this chapter the results of a designed system are presented. Only the performance of a control loop is presented as this is the only thing easily measurable. Performance of the face tracking algorithm is already partially presented in 6. Also important is that frame rate of approximately 8 FPS at 752x480 resolution could be achieved by face recognition in ROS on a laptop computer running an i7 4600U Intel processor.

In Figure 26 a measurement of an angle camera joint through time is given. At time 0 seconds, torque is applied through the motor to the camera joint in a horizontal axis. An exact amount of applied torque which is proportional to the duty cycle of a PWM signal is calculated through an PI controller loop. As already mentioned D component of the controller is provided through the actual friction in the camera joint and can be adjusted by a screw. The results of this quasi PID controller are satisfying as a desired angle value is achieved in approximately 0.5 seconds. However this time was much faster when there was no cable limiting the movement. However, this is still fast enough for the desired use case, as following a moving person is not a task that would require any faster movement. It can also be seen that the control loop frequency of 770 HZ which has a period of 0.0013 seconds is fast enough as good results are achieved.

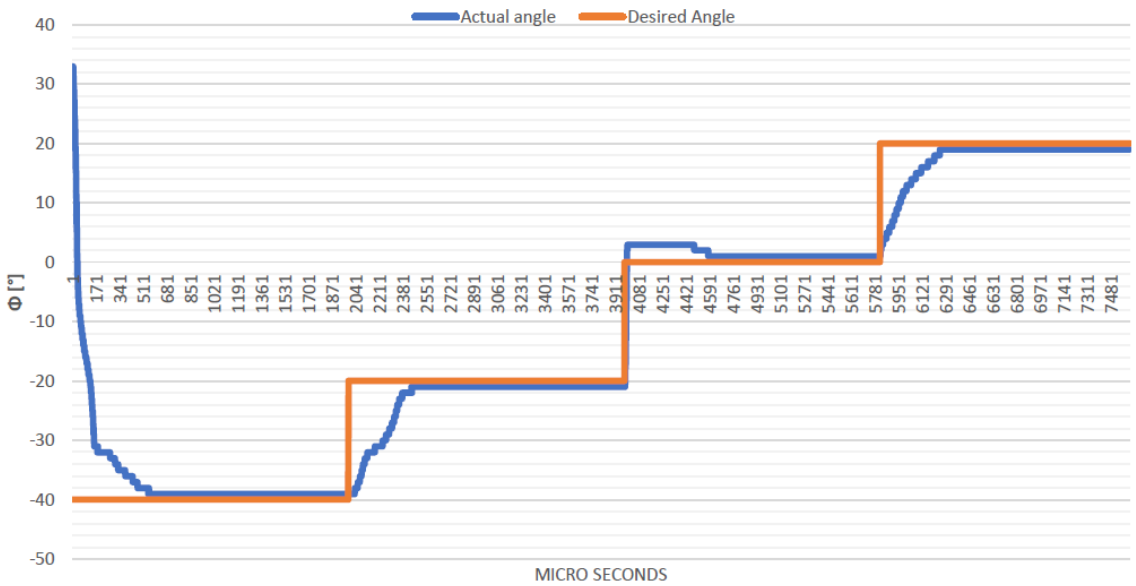

Figure 26 Response of a camera joint angle in blue to a desired angle command in red. Movement of the camera joint is horizontal.

Also mentioned should be that this result was only achieved after a lot of tweaking of the P and I parameters. At first a setup with only the P parameter was implemented and it could be easily seen that the offset in the angle of the camera joint was still noticeable after some time. This is the consequence of the inherent D component in the design of the camera joint.

This was the reason to add also the I component which helped immensely but also brought oscillations at higher angles and had to be limited and tweaked to only lower angles (<20 degrees). It is also possible that better results could be achieved with different settings but in favor of reliability the parameters were set to move more slowly.

The same approach tried to be implemented for a camera joint that moves in a vertical manner. At first this task seemed easier as there was less friction in the joint itself. However when applying the same control loop to work in the vertical manner there was a problem. In the vertical way there is an addition to the torque needed to rotate the joint because of gravitational forces and also the cables of the camera are pretty stiff and provide a lot of friction. Both of this factors combine to such resistance in the vertical way which is not able to be overcomed by the DC motors installed even provided with 100% PWM signal. The result is that the implemented control loop works in the vertical way only for very limited angles that can be achieved by the motors and the tracking is thus not even implemented in the y axis on the camera.

5. Conclusion

Implementing real time tracking of the face with a robot head proved as a very multidisciplinary task that required expertise in many different areas of engineering and programming. The workload stretched from very low-level tasks like soldering and implementing custom libraries, to the very high-level tasks like installing the right ROS distribution and programming face detection. On both sides there were also some unexpected challenges such as implementing a very low-level custom 3-wire SPI library for communication with encoders, getting ROS, face detection and camera to work on a Raspberry Pi.

When testing the system everything was implemented on a linux computer where the system worked seamlessly. However, trying to make the same components work on a Raspberry Pi was problematic due to issues with ROS versions and mentioned camera connectivity problems. Thus, development was continued on a linux computer where everything works and the developed code can easily be applied to any computer running Ubuntu 16.04 which makes it more portable. There were still many challenges encountered that will not be stated here but are described in this report extensively.

There is still some future work in the case of this project and there are things that could still be improved. Regarding the movement in Y-axis the torque needed is higher and thus thermal problems with the motors occur. Therefore some stronger motors or lighter cameras should be supplied for the y-axis movement of the cameras in which case already developed control loop could be used. With a few adjusted parameters it should easily work as it is not much different than the control loop for the x-axis apart from gravity. Next task would be to make BLDC motors working and trying to make the tracking of the eyes more distributed through the robot by solving the optimization problem using weighted inverse kinematics. This could provide much smoother and more natural movement of the robot.

However, because of the challenges encountered and the multidisciplinary nature of this project although pretty intense and demanding at times it has been a great learning experience.

List of Figures

Bibliography

[1] *Arduino ADK*. https://store.arduino.cc/arduino-mega-adk-rev3. Accessed: 2018-20-01.

[2] *Arduino Mega 2560 Diagram and Pinouts*. https://arduino-info.wikispaces.com/MegaQuickRef. Accessed: 2018-20-01.

[3] *Arduino Mega 2560 PWM frequency*. https://forum.arduino.cc/index.php?topic=72092.0. Accessed: 2018-20-01.

[4] *Arduino fundation*. https://www.arduino.cc/en/Guide/Introduction. Accessed: 2018-20-01.

[5] *Contelec Vert-X 13 - 5V / SPI*. http://www.contelec.ch/fileadmin/user_upload/contelec/Downloads/Datenblaetter/Englisch/Vert-X/Vert-X%2013/Vert-X_13_5V_SPI_e.pdf. Accessed: 2018-20-01.

[6] *Dual Full Bridge Driver L298*. http://www.st.com/en/motor-drivers/l298.html. Accessed: 2018-20-01.

[7] *Face Detection using Haar Cascades*. https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html. Accessed: 2018-20-01.

[8] *FlyCapture 2.x and ARM*. https://www.ptgrey.com/tan/10357. Accessed: 2018-20-01.

[9] *FlyCapture 2.x and Linux*. https://eu.ptgrey.com/tan/10548. Accessed: 2018-20-01.

[10] *FlyCapture2::ErrorType 33*. https://github.com/ros-drivers/pointgrey_camera_driver/issues/151. Accessed: 2018-20-01.

[11] Organtini Giovanni. “SCIENTIFIC ARDUINO PROGRAMMING”. In: *A free addendum to ”Scientific Programming”*. Sapienza Università di Roma INFN–Sez. di Roma. 2016.

[12] *How to use the L298N Dual H-Bridge Motor Driver*. https://www.bananarobotics.com/shop/How-to-use-the-L298N-Dual-H-Bridge-Motor-Driver. Accessed: 2018-20-01.

[13] *Introducing JSON*. https://www.json.org/. Accessed: 2018-20-01.

[14] *Key information on maxon DC motors*. http://hades.mech.northwestern.edu/images/e/ee/Maxon_Motor_Guide.pdf. Accessed: 2018-20-01.

[15] *Point Grey camera driver based on libflycapture2.* http://wiki.ros.org/pointgrey_camera_driver. Accessed: 2018-20-01.

[16] Morgan Quigley et al. “ROS: an open-source Robot Operating System”. In: *ICRA workshop on open source software*. 2009, p. 5.

[17] *ROS Driver for USB Cameras*. http://wiki.ros.org/usb_cam. Accessed: 2018-20-01.

[18] *ROS driver for Pt. Grey cameras, based on the official FlyCapture2 SDK.* https://github.com/ros-drivers/pointgrey_camera_driver. Accessed: 2018-20-01.

[19] *ROS wiki*. http://wiki.ros.org/. Accessed: 2018-20-01.

[20] *Raspberry Pi 3 Model B*. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/. Accessed: 2018-20-01.

[21] *Raspbian*. https://www.raspbian.org/. Accessed: 2018-20-01.

[22] *Ubuntu 16.04.3 LTS (Xenial Xerus)*. http://releases.ubuntu.com/16.04/. Accessed: 2018-20-01.

[23] Paul Viola and Michael Jones. "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1. IEEE. 2001.

[24] Borut Zupančič. *Vodenje sistemov*. Tržaška cesta 125, 1000 Ljubljana, Slovenia: Založba FE, Univerza v Ljubljani.