ROBOTIC CAMERA TRACKING

Supervisor – Dr. Phil Birch

DRK-1

UNIVERSITY OF SUSSEX

RICKY AMBROSE - 244144

Acknowledgments

I would like to thank the University of Sussex, Brighton for affording me the un-imaginable opportunity to complete my study here, despite all challenges which frustrated my efforts. I will not forget to thank Dr. Phil Birch, my project supervisor, whom for without his patience, guidance and understanding, I wouldn't have made it this far, especially with my dissertation. I also remain to thank Kevin Brady, the senior mechanical technician, for helping me with the printing of the components for my project, Lucy Uffindell-Saunderson for her patience with me, and providing access to the soldering lab. Both have been vital throughout the whole project.

Finally, a huge thank you to my parents and my sister for supporting me in this endeavour and encouraging me to always do better, and most importantly the almighty God, for his grace in me.

Abstract

Face tracking is the technology that detects facial features of humans and tracks them frame by frame. Although this technology is mostly applied to the security domain, it can also be used in other fields, such as mask detection, face following within a video frame, anomaly detection and expression detection. Connecting various computing devices via the internet to common everyday objects that are embedded with sensors and actuators, to improve connectivity and data collection is the basic idea behind internet of things (IoT). IoT is a relatively new technology that has a wide range of applications and will be an integral part of consumer products. This thesis will explore the paradigms of machine vision, face tracking, and IoT, in order to make a robot that can track and follow the face of a person.

Table of Contents

Table of Figures

Table of Equations

Table of Abbreviations

AI - Artificial Intelligence
BGR - Blue, Green, Red
CV - Computer Vision
DHCP - Dynamic Host Configuration Protocol
GUI - Graphical User Interface
I2C - Inter-Integrated Circuit
Id - Identification
IFP - Interface Profile
IoT - Internet of Things
iRDMI - Intel Remote Desktop Management Interface
IT – Information Technology
LoRa - Long Range
ML - Machine Learning
MV - Machine Vision
NN - Neural Network
OEM - Original Equipment Manufacturer
OOP - Object Oriented Programming
POM - Pouring Melted Resin
PWM - Pulse Width Modulation
RFID - Radio Frequency Identification
SOA - Service Oriented Architecture
SPI - Serial Peripheral Interface
TCP - Transmission Control Protocol
UDP - User Datagram Protocol
UI - User Interface
UPnP - Universal Plug and Play
USB - Universal Serial Bus
WEP - Wired Equivalent Privacy
Wi Fi - Wireless Fidelity
WAP - Wireless Access Point
WSGI - Web Server Gateway Interface
WSN - Wireless Sensor Networks

NOTE:
The code has also been uploaded to GitHub privately from: https://github.com/carbon-speed/DRK-1.

CHAPTER – 1

The Beginning

1 Introduction

Machine vision or computer vision is a field that is growing at a very fast pace. A machine vision system reads two dimensional projections and recovers useful information from the data provided by the sensor/camera. Although the information is gathered from a three-dimensional environment, the sensor in the camera converts that image into a two dimensional map. The computer should be able to perform the processing and tracking on its own, making the system artificially intelligent as there will be minimal human involvement. With the dawn of industry 4.0, which includes artificially intelligent systems, smart machine vision robots can be used to perceive in 3 dimensions and interact with humans to make sure perfect unison is maintained and the task is performed optimally. There are endless capabilities with MV and combining it with modern technologies such as IoT. This project combines MV with IoT, to get an understanding on the functioning of the basic concepts, and also to allow the robot to be able to take on updates in the future. The robot is called DRK-1 Dark Eye probe after the Sith probe droid from Star Wars (Fig 1).

Fig 1: DRK-1 Seeker Droid

1.1 Motivation

During Covid, the work from home culture was welcomed with open arms by many companies, and a lot of people started buying webcams in order to get better video clarity and audio as well. The idea of the project was to have a neatly designed mini robot that would sit beside the computer, and perform the tasks of face tracking, and following, by using 2 servo motors for motion. This will be paired with an IoT module so that it can transmit the data wirelessly and maybe be upgraded later on in the future. Binge watching Star Wars was also really helpful.

CHAPTER – 2

The Background Study

2 Literature Review

The literature review is divided into the respective topics as this is how the review had taken place. Although this is not a conventional method, it made the process of explaining what was learnt to make the project, in the manner it was learnt, quite clear.

2.1 Machine Vision

Machine vision(MV) is the technology that is used to provide analysis and image-based automation capabilities in an industry. MV enables the machine/robot to perform tasks such as automatic inspection, guidance, process control, tracking etc. just by viewing and analysing the image. This ability to be able to "see" what's going on, enables rapid decision making based on what the industrial equipment is seeing. In terms of functioning, artificial intelligence has three stages here: perception, cognition, and action (Jain R., et al., 1995). Psychophysics and cognitive science have studied human vision for quite some time now and the techniques of machine vision are related to what is known about human vision, including the concept of the neural network. But when considering overall AI structure, AI has three stages: machine learning, machine intelligence and machine consciousness (Fig 2). In order to achieve machine consciousness, the machine has to develop self-learning capabilities. One small step towards this is the use of machine vision and IoT. The primary use of MV are defect detection, tracking, sorting, and image based automatic inspection. MV can said to be one of the founding technologies of industrial automation, as incorporating it improved product quality, and production speeds. The dawn of edge computing and IoT devices has expanded the boundaries of what can and cannot be done using this remarkable technology. The components for image processing are usually the same, a camera(or other image capturing unit), lighting, software to evaluate, processor to run evaluation, and the output device. The image capturing unit doesn't have to be a part of the main image processing unit. When the whole unit , including the processing function is part of the same enclosure, the whole unit is referred to as *embedded processing*.

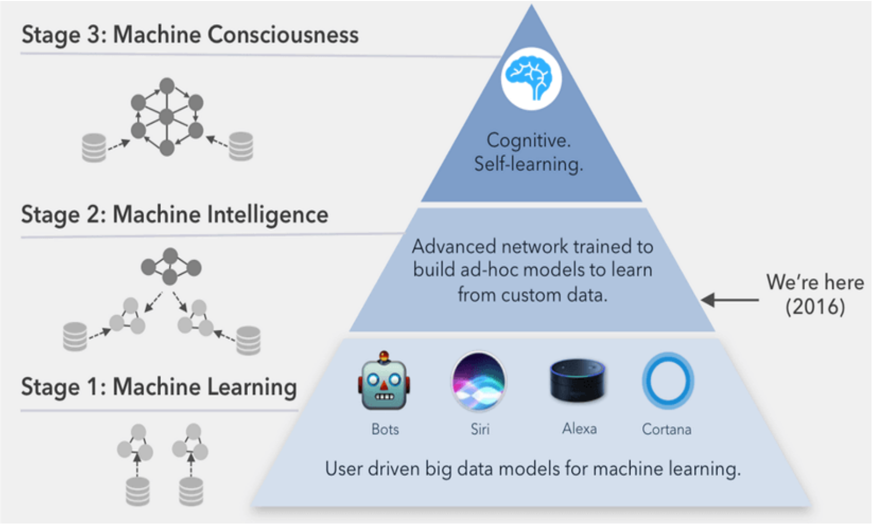

Stage 3: Machine Consciousness
Cognitive. Self-learning.
Stage 2: Machine Intelligence
Advanced network trained to build ad-hoc models to learn from custom data.
We're here (2016)
Stage 1: Machine Learning
Bots
Siri
Alexa
Cortana
User driven big data models for machine learning.

Fig 2: Stages of AI

2.1.1 Background Subtraction

As the camera location is fixed in most scenarios, separating the background is probably the most fundamental image processing operation, when it comes to security application with video. An overview and comparison of many techniques of *background subtraction* is given by Toyama, Krumm, Brumitt, and Meyers (Toyama K, et al., 1999). This method handles background model adaptation, it makes the initial judgement on whether a pixel is in the foreground or background and also address many classical problems such as issues with moving objects, time of the day, camouflages, and bootstrapping as shown in Fig 3.

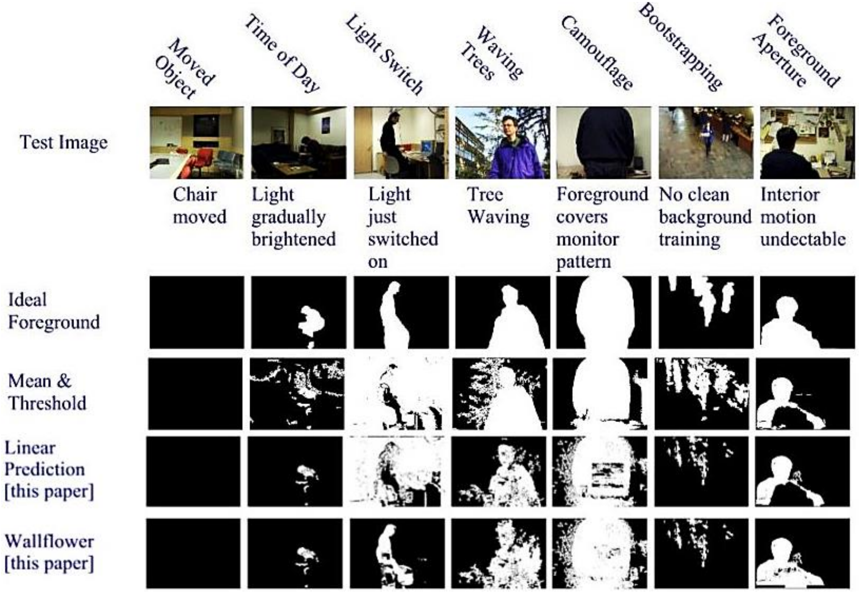


Fig 3: Wallflower subtraction results

The most important part of such a surveillance system is a module that differentiates between the background pixels, which have to be ignored, from the foreground pixels that have to be processed for identification and tracking. This differentiating, which is very vital, is not the most difficult task, but maintaining this model is very tough and is called *background maintenance*. Wallflower subtraction works at the pixel and handles foreground aperture problems when it performs regional level subtraction. It also takes care of the changes frame-by-frame and uses previous models to figure out what caused those changes. Compared to other background subtraction methods such as Gordon, et al. subtraction (Harville, et al., 2001), the Wallflower method doesn't require extra cameras to record depth, and also doesn't produce a "*halo*" around the foreground objects.

2.1.2 Face Detection & Haar Cascading

Tracking is done using a video source(sequence of images), rather than a still individual image. OpenCV has a built-in feature that can be used to detect faces, this can be used for virtually infinite applications, ranging from security to entertainment. In order to detect faces, some means of abstracting image detail is required to produce stable classification and tracking results. The abstractions are called *features* and are *extracted* from the image data. Haar-like features are one type of feature that is often used for real-time face tracking and were first used for this purpose in the paper by Paul Viola and Michael Jones (Viola P. & Jones M.J., 2004). A Haar classifier aka Haar Cascade Classifier is a ML, object recognition program that can be used to identify objects in images and videos. Integral images[1] are used to compute the image by using sub-rectangles and creating array references for every sub-rectangle made as shown in Fig 4. To determine the best features to represent an object from all of the available Haar features, we use AdaBoost.

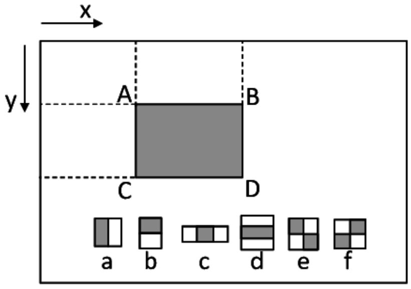


Fig 4: Working of integral image

[1] Integral Images are a matrix of numbers used to perform a certain function when overlayed on the image matrix

This is done by making a feature set of positive and negative images, which was run through a variant of AdaBoost to select the features and to train the classifier (Freund & Schapire, 1997). The AdaBoost classifier is used to boost classification performance of a learning algorithm by combining the weak classification functions to form a strong classifier. Compared to other feature selection mechanisms such as the wrapper method, AdaBoost has a really fast learning speed and in every round, the previously selected features are compactly and efficiently encoded using example weights. A large number of Haar features are needed for the model to be able to form a strong classifier by using the cascade classifiers.

The cascade classification works in stages, and in each stage, there are the weak learners. An accurate classifier is generated from the mean prediction of all the weak learners after training them using boosting. From these predictions, the classifier decides whether the object is found(positive) or moves onto the next region (negative) as shown in Fig 5.

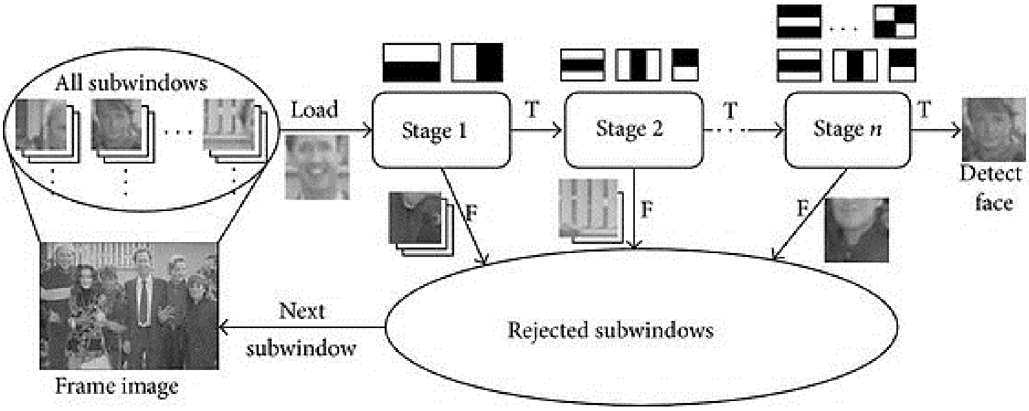


Fig 5: Cascade classifier flowchart

2.1.3 OpenCV

The OpenCV library is a free use open-source software under the Apache 2 License, and it can be used across various platforms. OpenCV is broadly structured into five main components, four of which are shown in Fig 6. The high-level computer vision algorithm and basic image processing is done in the CV component; The ML component includes statistical classifiers and clustering tools; HighGUI has the I/O routines and functions that stores and loads videos and images; and finally, the CXCore contains the basic data structures and content. The fifth component that is not mentioned in the figure is CvAux which contains the experimental algorithms, and the difunctional areas (Bradski & Kaehler, 2016). This includes eye and mouth tracking, texture description, 3D tracking, stereo vision etc.

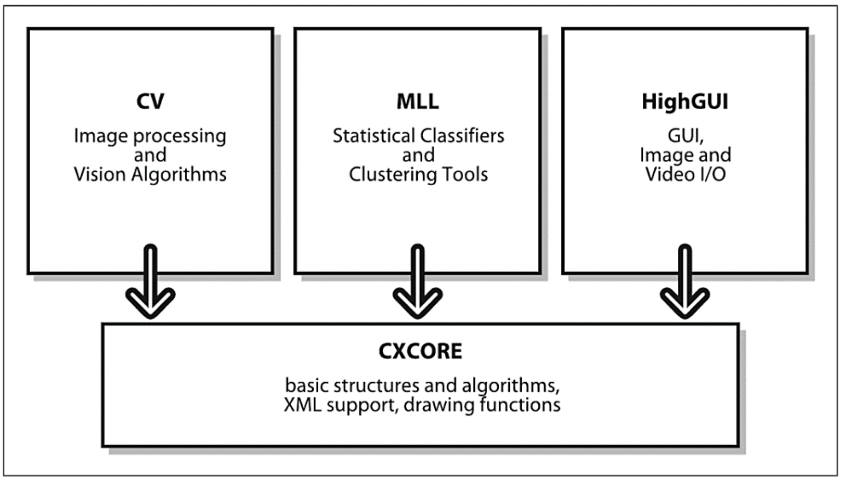

CV
Image processing and Vision Algorithms
MLL
Statistical Classifiers and Clustering Tools
HighGUI
GUI, Image and Video I/O
CXCORE
basic structures and algorithms, XML support, drawing functions

Fig 6: Structure of OpenCV

The most important reason why the OpenCV library was selected for the project is for its ability to track faces and also understand gestures. Various commands from the OpenCV library will be used to perform face tracking and then the data will be sent to the Arduino 33 IoT device. The OpenCV program will be run in python, in the laptop or computer that the webcam is connected to.

2.2 Internet of Things

One of the hottest paradigms in the IT world currently is *Internet of Things* (IoT). Today, most computers are wholly dependent on human beings for information, either by typing, pressing a button, taking a digital picture, or scanning a bar code. The issue with this is that people have very limited attention span, time to do things and the accuracy of performing the task. Computers need to be empowered with their own means of gathering information, so that they can hear and smell the world for themselves in all its glory. IoT has the potential to change the world, just as the internet did (Ashton, 2009). The wearable devices on our hand that track and monitor our heart rate, blood pressure, etc. are connected to our phones, and this data is stored on the cloud, there are also fridges that monitor the temperature and reminds the user if any item is low on storage and needs to be bought.

There are two important technologies in IoT are RFID and WSN. RFID, which stands for Radio Frequency Identification, is a technology that enables microchips to transmit the id of an object for the purpose of identifying, tracking, and monitoring to the reader wirelessly. Wireless sensor networks, or WSN's are mainly organised smart sensors that can be used to sense and monitor the environments and surroundings (Li, et al., 2012). For the end users to be able to access high-quality services, the technical standards of the IoT system need to be designed for smooth information exchange, processing, and communication between different devices. The success of IoT like all successful technologies, depends on standardization, which provides interoperability, compatibility, reliability, and effective operation on a global scale (D. & J., 2011).

2.2.1 SOA for IoT

In this section, the various components of IoT layer architecture are described in brief. This will form the basis of the project and will explain in brief the importance each layer has to make the whole project work. Service Oriented Architecture (SOA) is an architectural using which , application make use of the services available to the user. The services are published in an easy to access and work on method so that it's easier for the developers to assemble their apps or services (Garg, 2021). In (Zhang, et al., 2017) an event-based distributed IoT service platform is proposed that aimed to enable IoT domain service creation and minimising the complexity of service accessibility (B, et al., 2018). Like every SOA, even the SOA for IoT has layers. SOA for IoT has 4 layers (Fig 7) (Sikder, et al., 2021) and they are explained below:

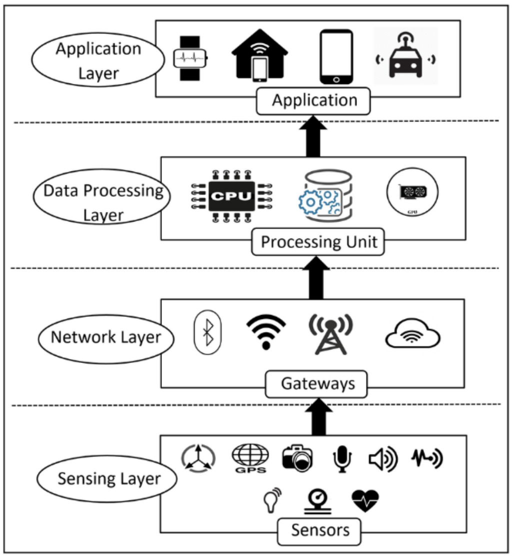


Fig 7: IoT architecture layer and components

2.2.1.1 Sensing Layer

This layer integrates all the existing hardware, which includes RFID's, sensors, actuators etc. to sense/control in the physical world (Xu, et al., 2014). All the sensors in the IoT devices are integrated through sensor hubs (Perera, et al., 2013). The hub accumulates all the sensor data and forwards it for processing. There are several transport mechanisms such as I2C and SPI that are used for data flow between application. The mechanism depends on the IoT device and there are generally 3 types of sensors, motion, environmental and position. In certain industrial sectors, intelligent service development schemes and a Universal Unique Identifiers (UUID) are used to be able to identify and retrieve information, rendering UUID's to be critical for optimal service deployment of huge IoT networks (Wu, et al., 2013) (Ilie-Zudor, et al., 2011).

2.2.1.2 Network Layer

The network layer connects all things together and permit the sharing collected data in the sensing layer to other connected devices. The network layer is usually implemented in IoT devices using diverse communication technologies such as Wi-Fi, Bluetooth, LoRa, cellular network etc. These communication technologies are vital for designing the networking layer in IoT, and issues relating to the network management need to be addressed for heterogenous networks, QoS requirements, service discovery, data and signal processing, security, and privacy (Guinard, et al., 2010). The network layer is highly susceptible to *Man-in-the-Middle* attacks (M & G.M., 2014). The key exchange mechanism in IoT must be secure enough to prevent any intruder from eavesdropping or committing identity theft (Mahmoud, et al., 2015).

2.2.1.3 Data Processing Layer

The data from the sensing layer is collected and analysed in this layer, in order to be able to make decisions. Certain devices such as smart watches, home hubs, smart fridges etc. stores the result from the previous data analysis in this layer to improve user experience. The analysed data and the results are shared to the devices after processing via the network layer. The middleware exists between the technological and the application levels and hides the details that are not to the programmers focus (Atzori, et al., 2010).

2.2.1.4 Application Layer

The application layer is where all the results of the processed data is presented or implemented. This layer is focused on the user experience, and it is used by the user to execute various tasks. There is a diverse range of applications, such as personal care, health care, smart transportation, smart home appliances etc. This layer also bears the responsibility of simplifying the management of the interconnected devices. An interface profile (IFP) could be said to be a service standard to support the interaction of a deployed application on the network. A facility to facilitate the interaction of various services to various devices via a common protocol for Universal Plug and Play (UPnP) can be said to be a requirement of a good IFP (Xu, et al., 2014).

2.2.2 Security and Privacy in IoT

There will be resistance and mistrust from people in IoT as long as the public confidence is not built that there is no serious threat to privacy with such devices. IoT devices are especially extremely vulnerable to attacks for several reasons. Due to the device mostly staying unattended/ unsupervised, it can be attacked physically. Wireless communication based attacks can be used to eavesdrop into someone's lives easily, and because most of the previous IoT devices are characterized as "*low capability*" devices, they are not capable of performing complex security functions or implementing such schemes (Atzori, et al., 2010). All solutions to solve security issues use some form of cryptography, and encryption takes a large amount of energy and bandwidth resources, both at the source and at the destination. These solutions cannot be applied for IoT devices due to the involvement of sensor nodes and tags, which are constrained in terms of energy, communication, and computational capabilities.

Privacy is a vital part of our civilization, which is so important, that it is in the legislations of various countries. It is a legal right that all the private data of an individual has to be kept private, and not public. Privacy concerns with regards to IoT devices are well justified in that sense, especially in the current times when smart devices have made it into our household, and continuously collect data.

CHAPTER – 3

The Parts of DRK-1

3 Components

The DRK-1 requires certain components that work in unison to enable the robot to work and complete the expected tasks. Certain components of the robot have been purchased from the market, while some had to be designed to suit the requirements based on certain criterial. The purchased components are mostly OEM parts and have been selected after understanding and reading their respective datasheets, or any available information regarding the component. Other alternatives like the Raspberry Pi, ESP32 CAM couldn't be used mostly because of the lack of availability.

3.1 Electronic Components

The electrical components of the project were selected after careful consideration. Since most of the components are OEM, their datasheets of the components were read in order to be selected for the project.

3.1.1 Camera

The camera sensor used in the project is the Nulea C902 HD Webcam. It is a cheap webcam that resolution of up to 1080p and frame rate of 30fps. The webcam comes in a tiny package of 25mm x 27mm x 70mm, as shown in Fig 8, which is really great for the project as it will suit the look of the robot. This was one of the smallest and cheapest cameras that is available in the market, and it has been disassembled from its OEM packaging to be able to sit with the robot using a 3D printed mount. It has a really great field of view as well, at 98°. The camera has a mechanical privacy shutter that can be closed or opened when used for online security, it has automatic light correction that consists of a low light boost, exposure adjustment, and colour boosting. The camera is universally compatible, meaning it is a plug and play device, and doesn't require any extra software or drivers to run or capture any video. It is compatible with a wide range of PC's and operating systems such as Windows 7/8/10/2000/XP, Mac OS, Android or above. To know when the camera is being accessed by the laptop, a green light is also present. When the camera is just being powered, a red light is activated, but when the sensor is turned on, the green light is also turned on.

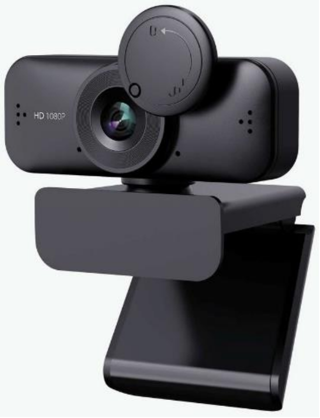


Fig 8: Nulea C902 HD camera

3.1.2 Arduino 33 IoT

The Arduino 33 IoT (Fig 9) has the form factor of the Arduino nano board paired with an ARM Cortex-M0 32 bit SAM D21 processor, an ECC608A crypto chip and the powerful NINA-W102 multi-radio module, which means it is extremely secure, and has Wi-Fi, and Bluetooth. This board will be responsible for controlling the two servo motors after receiving the angle values from the python program. The board can give has an I/O voltage of 3.3V's but can take an input voltage from 5-18V's. In order to be able to receive a voltage of 5V from the board, a solder bridge needs to be made. But since the robot has an external 9V power source, that won't be necessary.

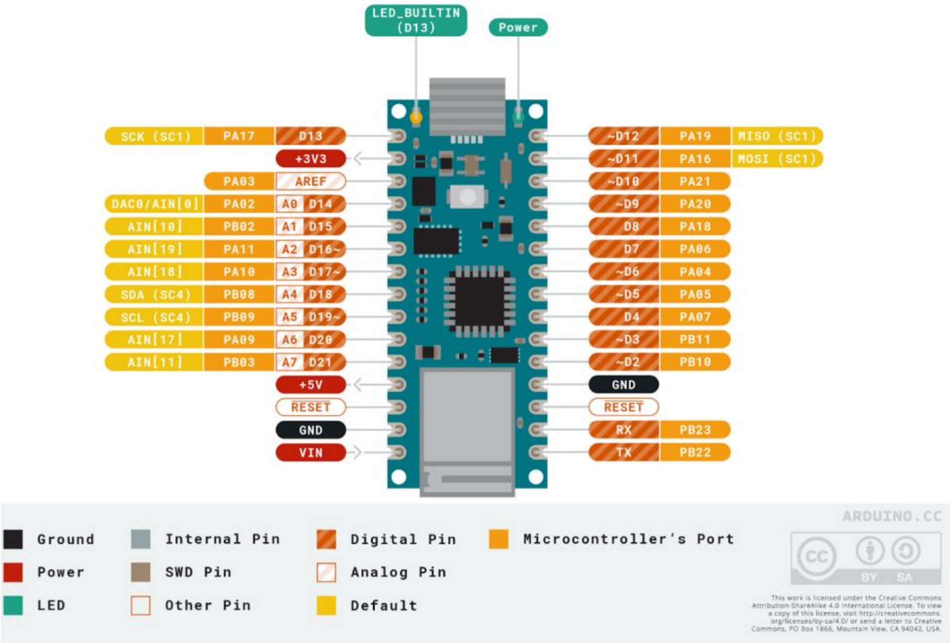


Fig 9: Arduino 33 IoT pin outs

3.1.3 L7805CV

The robot is supplied with 9V and 1Amp of current, while all the components except for the Arduino have to be powered with 5V input voltage. It is a three terminal regulator that comes in the industry standard TO-220 packaging. The L7805CV (Fig 10) voltage regulator is a cheap accessory that performs well, and also has a safety shut-off, when the temperature of the module becomes too much to prevent damage of the component.

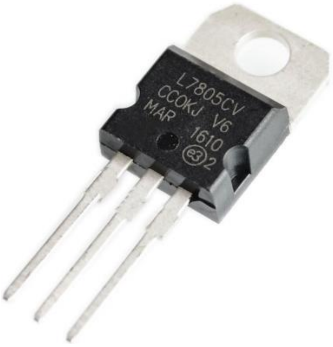


Fig 10: L7805CV Voltage regulator

The voltage regulator plays the pivotal role of supplying the components with regulated and stable 5V's of electricity. But during this process, the extra voltage is dissipated as heat, and the regulator can get really hot. From the datasheet it is understood that adding capacitors of 0.33µF to the input & ground pins, and 0.1µF to the ground & output (5V) pins, helps reduce the amount of heat dissipation required as explained in Fig 11.

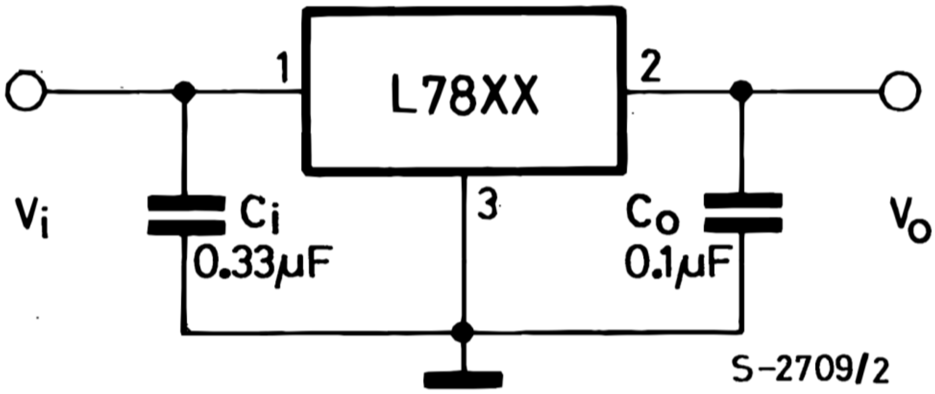


Fig 11: L7805CV Application Circuit

If not this circuit, the other option for dissipating heat was to use a heat sink. But given the small size of the whole robot, the plastic shell, and the closed structural design of the robot, using a heat sink wasn't feasible. So, to tackle this issue and to have some safety, each component that requires 5V's of supply is essentially provided with its own regulator.

3.1.4 Neopixel Stick 5050

The Neopixel Stick 5050 will be used in the project to provide a visual aid regarding the processing stage. This LED stick is really small and has really bright SK6812 RGB LED's (see Fig 12). The stick consists of 8 individually addressable LED's that are mounted on a PCB with mounting holes. The LEDs are addressable as there is a ~18mA constant current driver chip inside each LED which will keep the colour of the LED consistent even if the voltage varies. The LED stick requires a power input of 5V's (4-7V also does it), and the power and ground ports are provided on the top and bottom of the LED strip for ease of use. The LED strip is very time sensitive and hence requires timing-specific protocol from a real-time microcontroller such as an Arduino (Adafruit, n.d.).

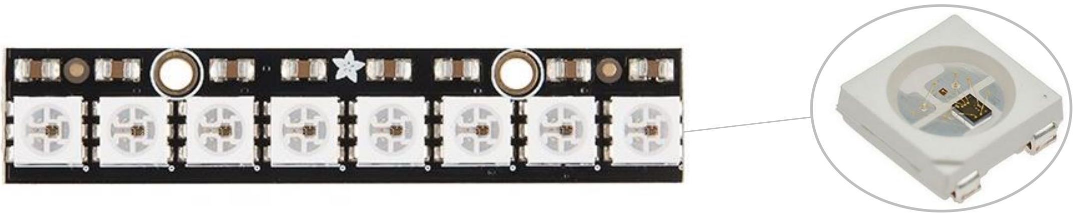


Fig 12: Neopixel 8 LED Stick & SK6812 LED

3.1.5 MG90D/SG90D Servo Motors

The MG90D and SG90D (see Fig 13) are very popular digital servo motors that are manufactured by TowerPro. The SG90 is the most popular motor that weighs 9gms, while the MG90D, weighs 13gms. The MG stands for metal gear, and it is a very robust servo motor with double ball bearings and 6061-T6 aluminium shaft, because of which it can produce up to 2.4kgs/cm of torque at 6.6V's and has a dead band width of 1µs. The SG90D servo motor produces a stall torque of 1.8kgs/cm at 4.8V's, and the gears are made with POM[2]. The servo motors are connected to the other components using *horns*. They are a very common type of servo motors and are extremely easy to use.

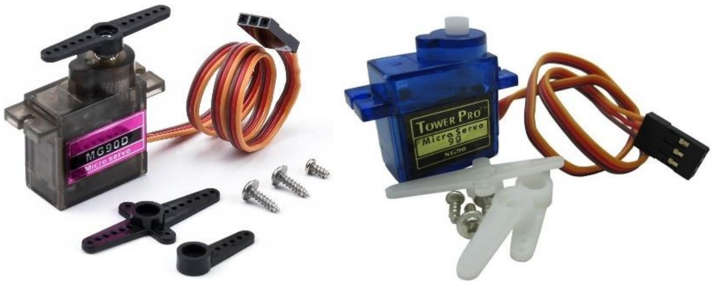


Fig 13: MG90D & SG90D with their respective horns (from left to right)

[2] Pouring Melted Resin

3.1.6 Piezo Electric Buzzer

An OEM piezoelectric buzzer (see Fig 14) is also packaged inside the robot to provide audio feedback for any changes or updates in the processing of the robot. A piezoelectric buzzer uses reverse piezoelectric effect to generate sound. The piezoelectric effect is generated in crystalline materials that possess non-centrosymmetry[3]. This affect is the preferred method of choice for energy harvesting in meso-to-micro scale devices (APC International Ltd., 2015). The buzzer consists of a piezoelectric crystal that is placed in between two conductors. When a potential is applied across these crystals they push or pull the crystal, and the resultant oscillation generates sound waves ranging from 2 to 4KHz (Bhatt, n.d.).

Fig 14: Piezoelectric Buzzer

3.1.7 JST PH connectors

In order to be able to divide the whole robot into separate units, for easy access, debugging and assembly, the JST PH connectors are used (see Fig 15). These connectors are really tiny and are easy to connect and disconnect whenever required. They have a pitch of 2mm, current rating of 2Amps and can take voltages of up to 100V's (JST, n.d.). The connections are soldered to the pins and a male-female type connection are made.

Fig 15: JST PH-2 Connectors

3.1.8 Barrel Jack Connector

The robot is powered by an external 9V's adapter. This adapter is connected to the robot using a barrel jack type connector. The connector has 3 pins for the output (see Fig 16) where 1 is the positive terminal (aka centre-positive pin barrel jack[4]), 2 is the shunt[5], and 3 is the negative terminal. The barrel jack connector is a very common OEM part that is used in many electronic devices as it is very easy and simple to use and operate.

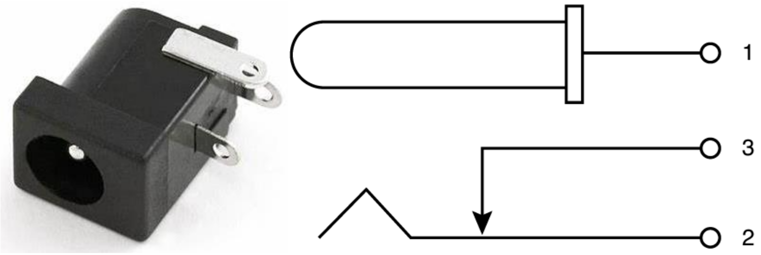


Fig 16: Barrel Jack connector and it's circuit diagram

[3] Structure with no centre of symmetry

[4] 2.1mm pin

[5] Shunt is a low-resistance path for the electrical current to flow from

3.1.9 1N4007 Diode

An 1N4007 diode (see Fig 17) is PN junction that belongs to the 1N400x series family. It is a very small and cheap component that is well-known to be ideal for a wide variety of applications. In this robot, the diode is used to make sure that the current flows in the correct direction. This issue is partially resolved by using a modern standardised barrel-jack connector, but when the wrong barrel jack is connected in which the polarity of the terminals is reversed the whole line up of electronic devices might burn out. Although these types of jacks are not very common, it is always better to be prepared for such issues, hence, the 1N4007 diode is used to prevent negative voltage supply problems. The 1NXXXX devices are based on the American standard numbering system in which the 1 means it has 1 junction; N signifies that it is a semiconductor diode (eTechnophiles, n.d.).

Fig 17: IN4007 Diode

3.2 CAD Designed Components

The robot is designed in such a way that the assembly and disassembly can be done with ease, and all the components have to be accessible. The designing process took the most amount of time as a lot of thought had to go into not only accommodating all of the components in a small packaging, but also making sure that it looked simple and beautiful from the outside. The robot is split into 3 basic sections, the top, mid and bottom. Each of these sections is to house a plethora of parts, and their wiring as well. There is a fourth piece, that is the base cover, to cover and hold the wiring of the power unit and the camera. The design of the components had to be done by keeping in mind that the parts are going to be 3D printed. Below, the final iteration of the components will be explained:

3.2.1 Base

The *Base* is the simplest structure in the whole design. It houses the barrel jack and guide the camera wires towards the outside. There are only 2 groups of wires that live in the base area, those are the two main power wires (positive and negative), and the four wires from the camera (positive, negative, Din and Dout). Along with the power lines, the 1N4007 diode is also connected in series for the above mentioned reasons in 3.1.9 1N4007 Diode. The other features that are part of the design are the two servo horn holes, the big hole in the centre for the servo motor itself, and a guide pillar to push the wires up and guide them. The base is 12mm thick and is designed such the all the extrudes are vertical (Fig 18).

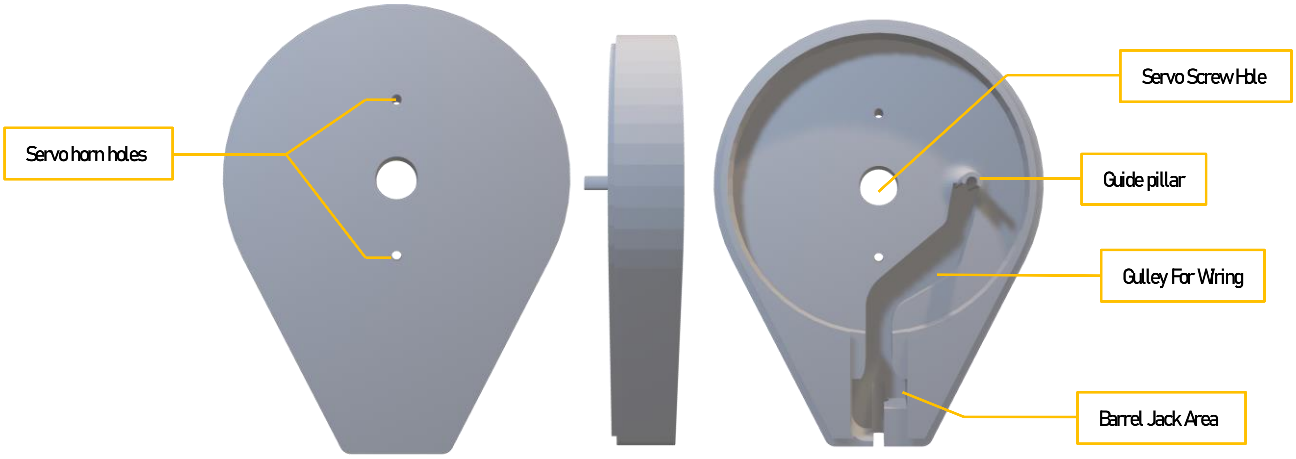


Fig 18: Different views of Base

3.2.2 Mid-Section

The middle section is a little more complex when compared to the *Base*. The design had to be simple and be able to house most of the mid-level components, and hence a cylindrical shape was selected. The cylindrical shape cannot be followed on the inside of the part as well, as most of the electronic components have flat surfaces. The plan is to fit the Neopixel light stick, two voltage regulators, the MG90D servo motor and the other wiring inside this part. Another important consideration was that this part would also serve as the housing for the top lid (3.2.3 Top Lid) along with the parts it houses. Since the servo motor is connected to this component, it will spin, and the base would be stationary. The wiring that comes from the base (power) and the wiring of the camera should have some *slots* to flow from, unobstructed.

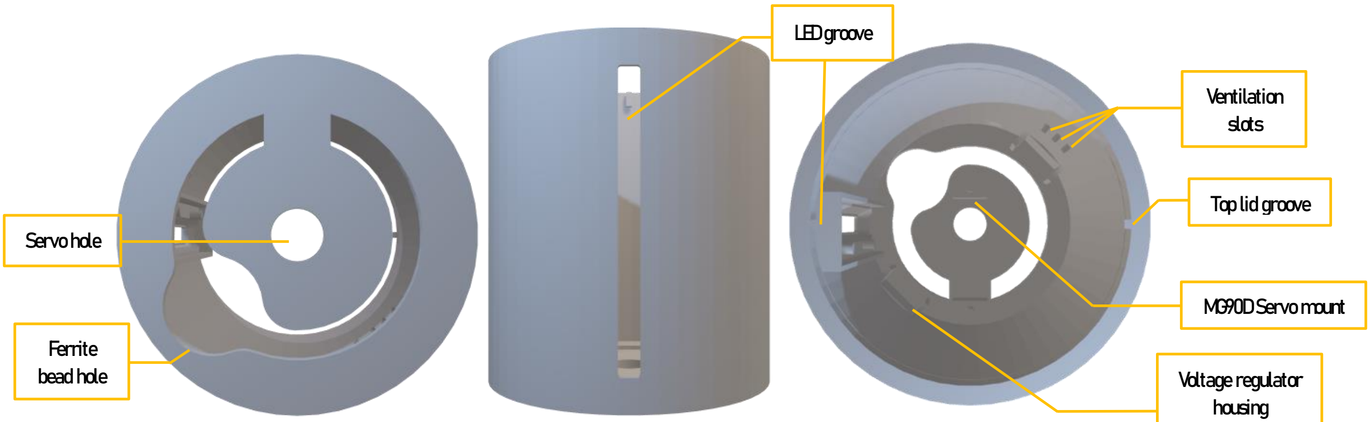


Fig 19: Different views of Mid-Section

From Fig 19, in the top and bottom views, the slots can be seen with a sudden big hole. The purpose of this hole is to allow the USB wire and the *ferrite bead* to also pass through. There are also some holes in the side of the cylinder for some ventilation, as at that place, two voltage regulators are present at close proximity, and finally a central hole for the shaft of the servo motor to be connected to the horn at the base. The mid-section also has male-extruded grooves, to have proper alignment of the top lid and also for achieving perfect fit of both the components.

3.2.3 Top Lid

The top lid is seemingly the most complex of all the components. This is mostly due to its unusual shape. The tolerances of this part had to be on point or else it would mean that the component will either be too loose or not fit at all. The top lid houses one voltage regulator, one piezoelectric speaker, the Arduino 33 IoT board, the SG90D tilt servo motor and the camera mount. The camera wires are also guided through a hole that is placed on the lid. The *top lid* is designed to mate properly with the *mid-section.*

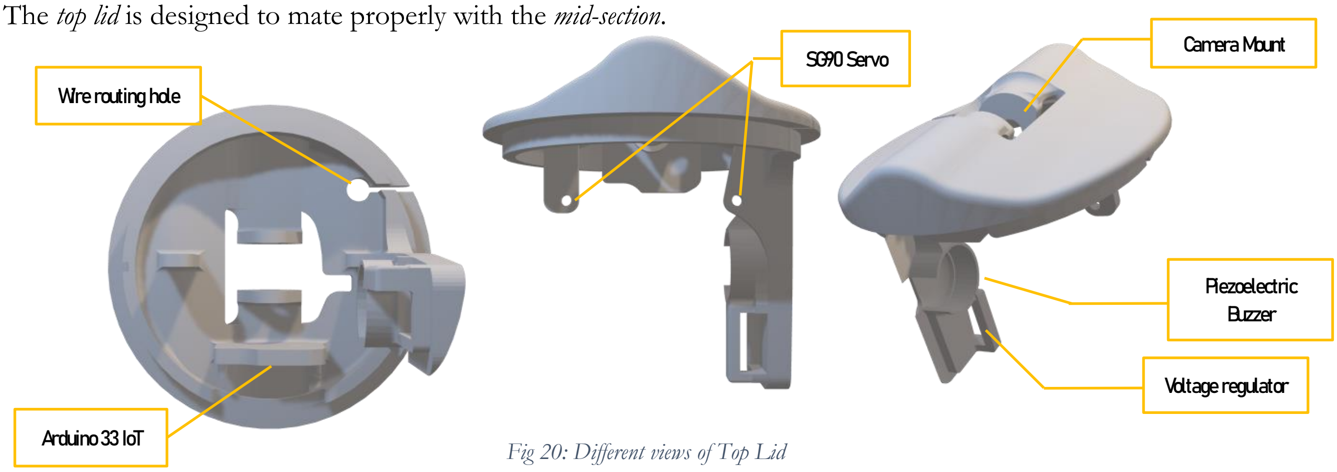


Fig 20: Different views of Top Lid

Fig 20 clearly shows the location of the mounts. The top lid had various iterations to accommodate the electronic components perfectly, and also many pieces were 3D printed. This piece was the only one that has an odd shape and took a lot of supports to print. Every hole, slot, mount, cut has a significant purpose on the part and plays a vital role, either for ease of printing or ease of assembly.

3.2.4 Camera Mount & Pin

The camera mount is used to hold the camera. It is a rather simple design, with the addition of a pin hole for rotation and another hole for the pivoting wire from the tilt servo motor. The mount is required as the camera has no proper mounting piece that is suitable for mounting it directly to the robot. The camera and the mount are mated using a male and female type joint (Fig 21).

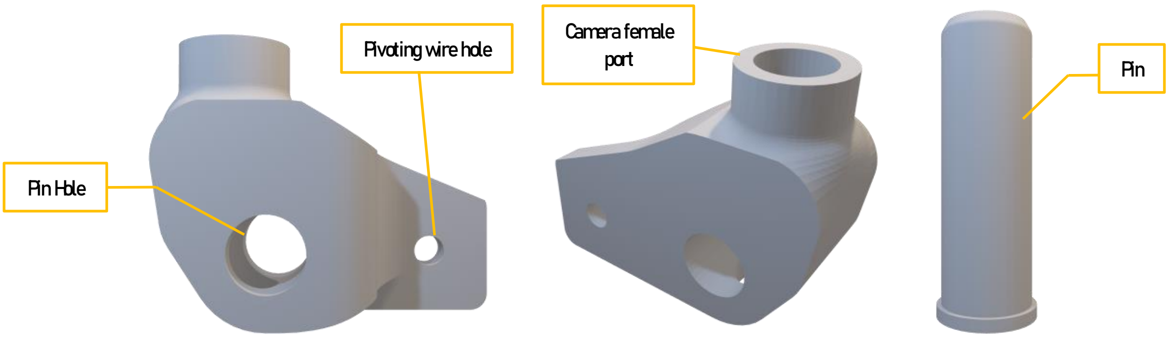


Fig 21: Different views of Camera mount and Pin (from left to right)

3.2.5 Base Cover

The *Base Cover* is used to cover the exposed wiring that exists in the *Base* section. It is present for aesthetic purposes and is made of a different material and manufacturing process. The Base cover is made of a 4mm flat white nylon sheet, that has been laser cut in the shape given in Fig 22. They are grooves for the barrel jack, and a big hole in the centre for the mid-section to sit.

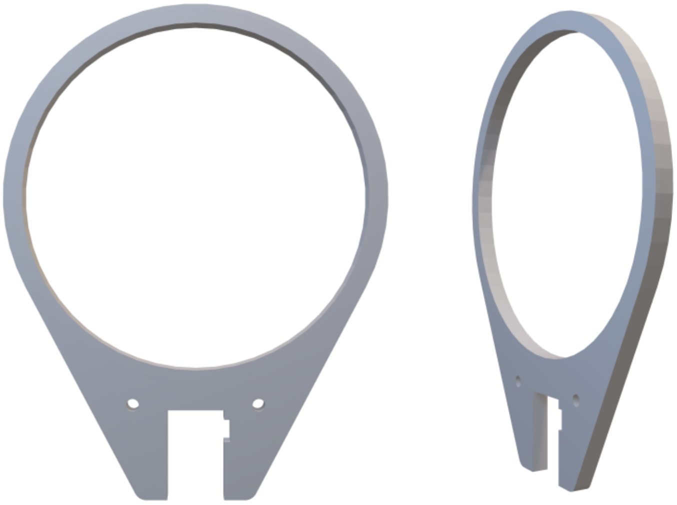


Fig 22: Different views of Base Cover

CHAPTER – 4

The Functioning and Assembly of DRK-1

4 Assembly

4.1 Electrical Connections

The circuit was made using Fritzing as it was extremely easy to use, and it has a large part library. Many OEM parts release their component's digital models that are ready to be used in Fritzing directly, which made the process of using this software more fluid and easier. The base, mid and top sections of the robot are all soldered separately and connected using the JST PH2 male-female pins. This is to be able to make assembly and disassembly of the components possible, and also for the purpose of easy debugging, if necessary.

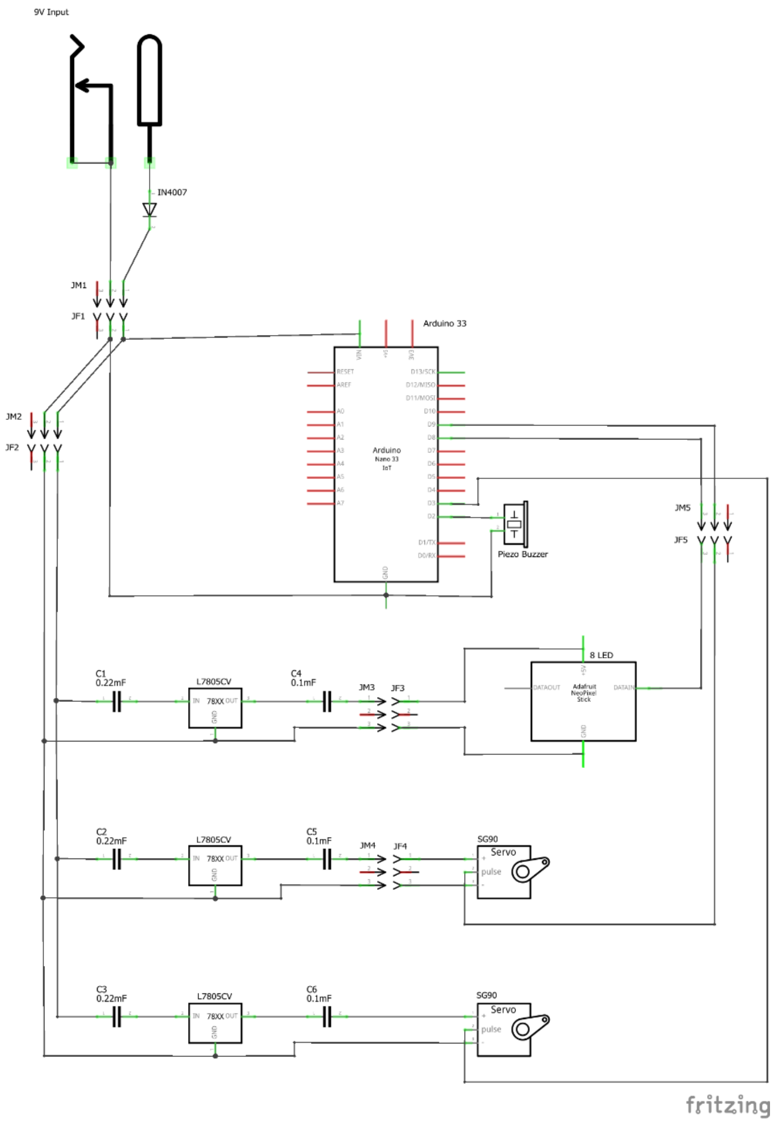


Fig 23: Circuit diagram of DRK-1 from Fritzing

Fig 23 shows the circuit diagram designed for the DRK-1 robot. It contains all the components necessary for the proper functioning of the robot. The circuit can be explained much more easily when the flow of current is followed step-by-step:

1. The 9V, 1Amp adapter is connected to the barrel jack that is mounted to the robot.
2. The power flows through the 1N4007 diode if the polarity is correct, or else no power flow is observed as the light doesn't turn on and no sound is generated.
3. The power is then split into two. One going directly to the V_{in} of the Arduino, and the other goes to the three voltage regulators. These connections split and made using the JST pins.
4. The voltage regulators take in the 9V's input voltage and drops it down to 5V's to supply it to the two servo motors and the Neopixel stick. The power going from these regulators are only used to power the components.
5. The 9V's that goes directly to the Arduino, is used to power up the micro-controller. The two servo motors and the Neopixel stick have one DATAIN pin that is connected to the Arduino at various digital pins.
6. The piezoelectric buzzer is the only component that is connected directly to the Arduino.
7. The ground from the power source is common for all the devices, and it is important to have all the grounds common.
8. It can be observed that all the digital pins have one JST PH connector and the power pins have another.

4.2 Assembly

This section will explain the assembly of the robot. It will include the type of mating and it will also attempt to provide a better explanation of the design decisions made.

4.2.1 Top Lid Assembly

The assembly of the Top Lid CAD model can be seen in Fig 24, along with all the components used for attaching them together.

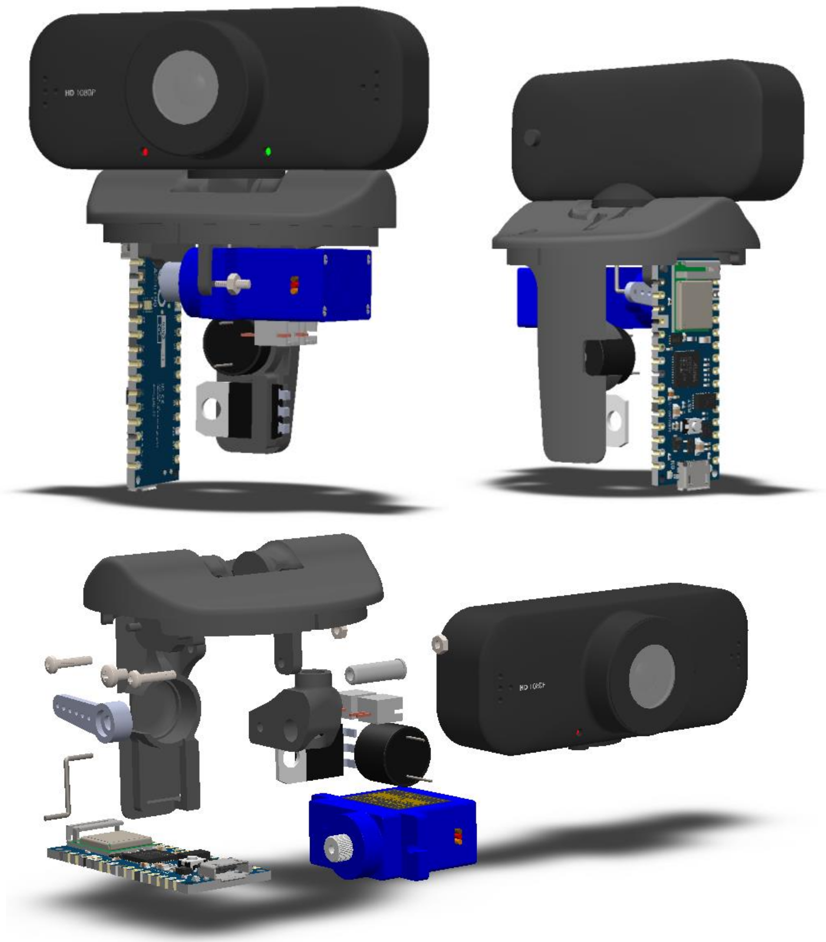


Fig 24: Top Lid Full CAD Assembly and Exploded view

The fun part about the Top Lid assembly is making the most of available space, which mean a lot of the stuff has to be cramped up in there. This process although easy on paper, it isn't very easy to do in practise. A lot of practical forward thinking is needed to be able to design a dense assembly component like this. The Top lid has to assembled in steps to ensure that all the components are assembled as designed and there are no issues later on.

Stage-1: The camera mount is fit in place and the camera pin is slid in.
Stage-2: The wire link for pivoting the camera is attached to the camera mount hole and the servo horn.
Stage-3: The program is run, and it is ensured that the servo motor is at the resting 90° position, before attaching the horn to the motor.
Stage-4: The SG90D servo motor is attached to the mounting points on the Top Lid.
Stage-5: The voltage regulator and the piezoelectric buzzer are slid into their positions, after which the Arduino board is also placed on its mounts.
Stage-6: The wires from the camera are slid into the guide slot provided.

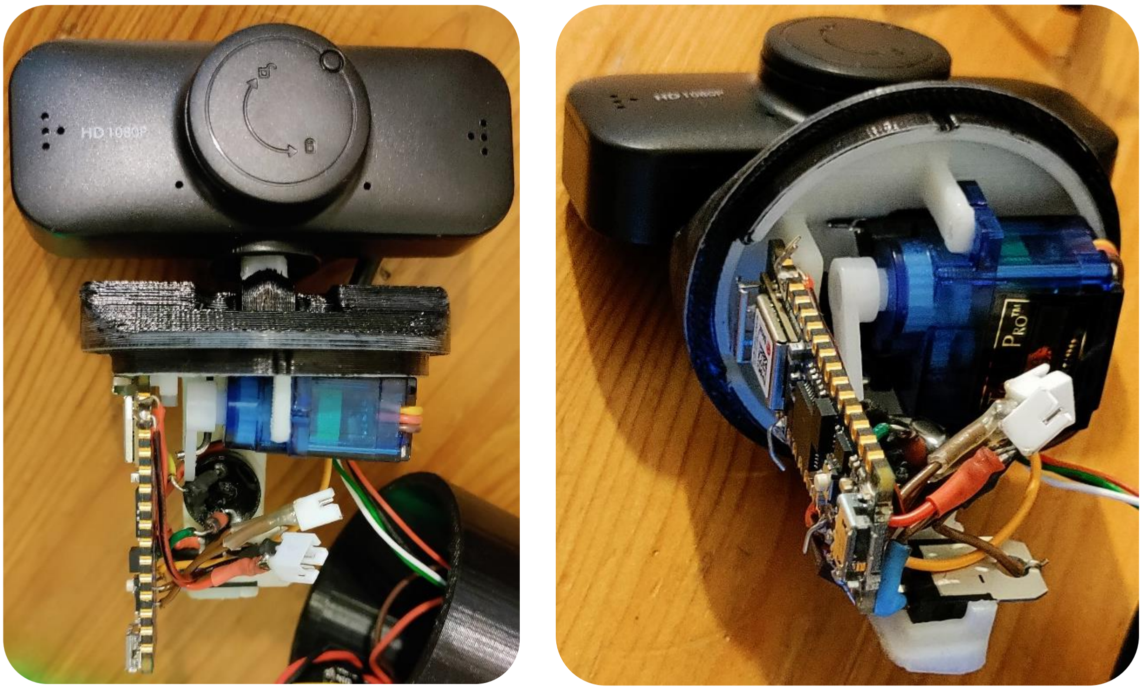


Fig 25: Top lid real assembly

From the Top Lid assembly, there are 2 pairs of wires that are used to connect and disconnect it from the base. One pair is the power lines from the Arduino and voltage regulator for both the V_{in} and GND, while the other are the data pins for the rotate servo motor and the light stick. Fig 25 shows how the model is after the soldering is performed and completed.

4.2.2 Mid-Section Assembly

The Mid-Section's assembly (Fig 26) is populated with a lot of wiring. There is some excessive wiring provided so that the Top Lid could be lifted up in order to upload a new program, debug and make any required changes. The Mid section assembly contains the MG90D servo motor that is used for providing rotational motion for the robot. It also has two voltage regulators for supplying the required power to the LED stick and the servo motor. They are affixed at the two corners in such a way that they are close to the component they are powering.

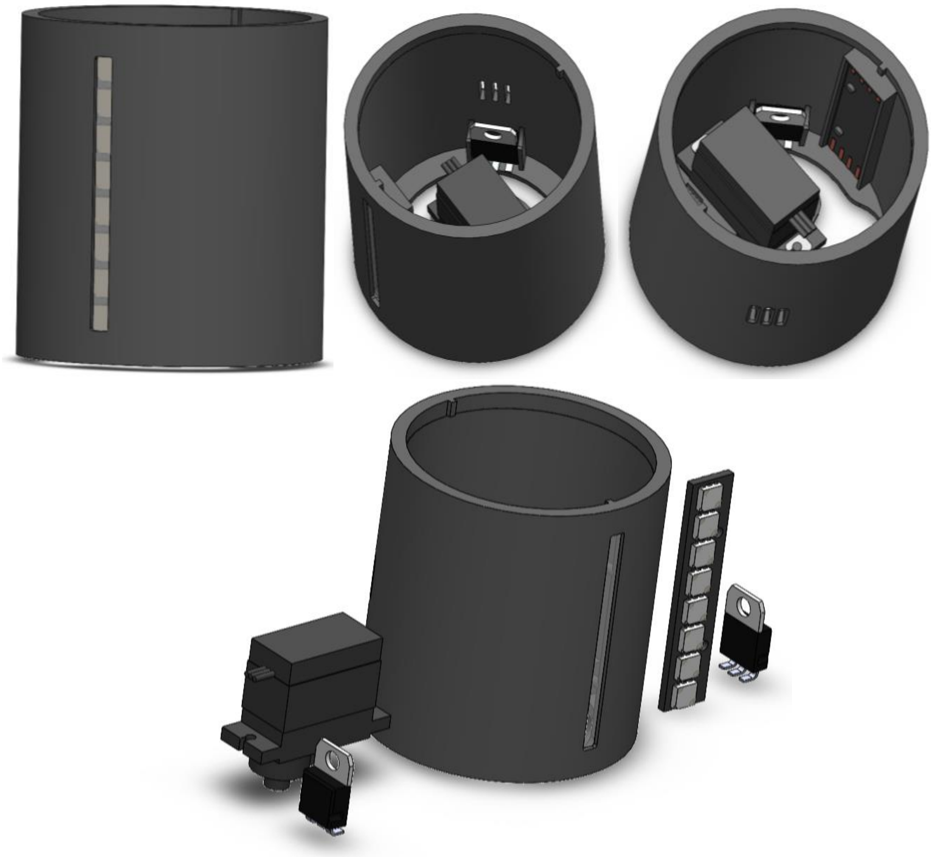


Fig 26: Mid-Section CAD Assembly and Exploded views

The Neopixel stick has the option of being powered from both ends of the stick, but data can be sent from only one end of the stick. Hence the orientation of the stick is done in such a way that both the 5V's and Ground connections are made at the bottom and the D_{in} pin is facing the top, so that it is closer to the Arduino for data transmission.

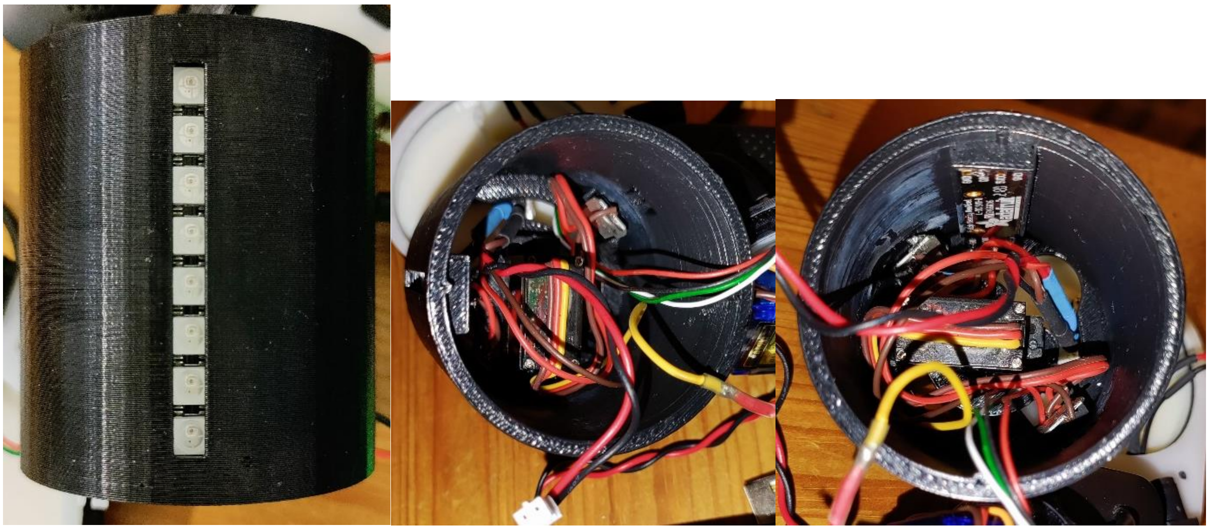


Fig 27: Mid-Section assembly

4.2.3 Base Section Assembly

The Base section assembly has to hold just one electrical component, that is the Barrel Jack Connector. The design of the base is done to keep up with the Star Wars theme (Tried to make it look like the Millennium Falcon, without the cabin) (see Fig 28).

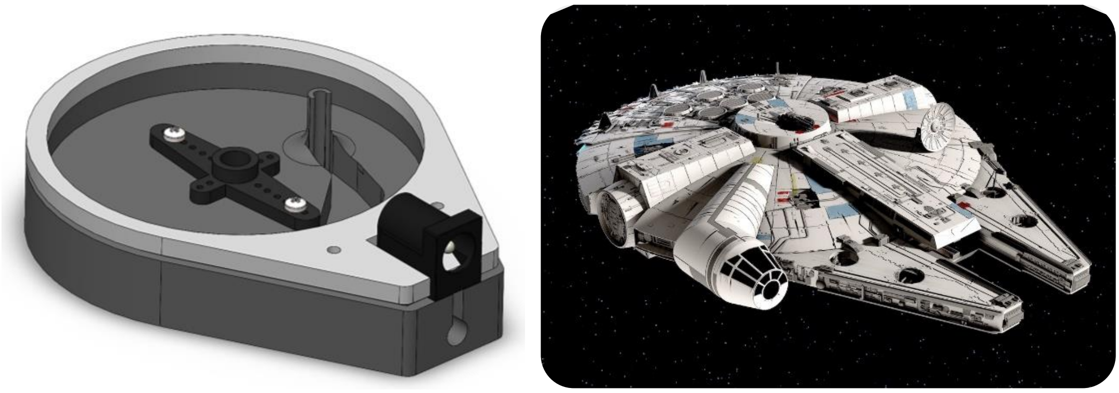


Fig 28: Comparison of Millennium Falcon and Base Assembly

The Base Assembly has gulley's that are for providing some space for the wires to pass through without obstructing the movement of the Mid-Section. A few iterations of the Base Assembly had to be made and tested for fitment and practicality, and the final result is in Fig 29. There is a hole in the rear of the Base part, which is provided to accommodate the camera wires, and it allows for a neat exit of the camera wires.

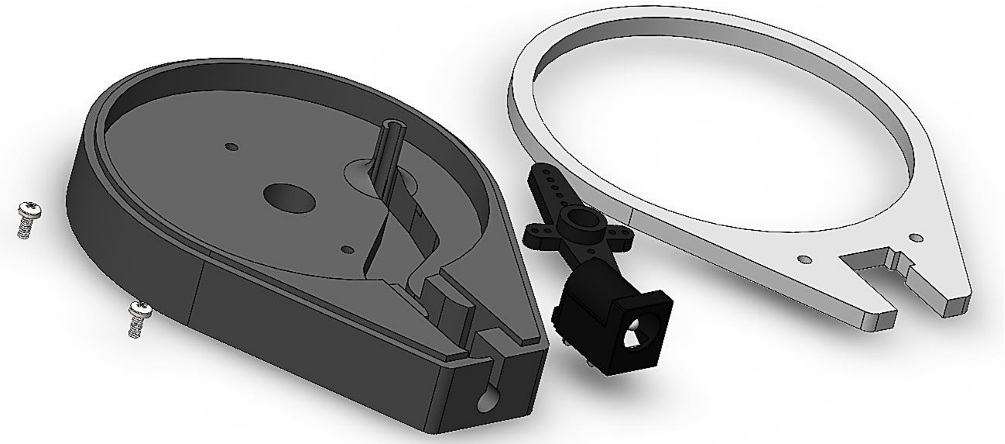


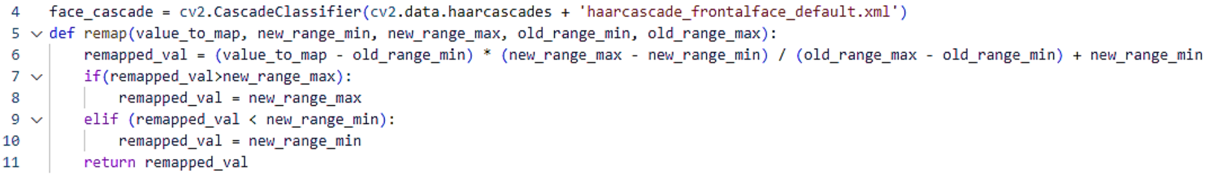
Fig 29: Base Section Exploded view

4.3 Programming

There are multiple programming files to be explained. Object Oriented Programming (OOP) was used for the Python code. The whole code can be found in the 7.2 Code, in this section, only the important parts of the code will be explained.

4.3.1 Face Tracking Python (opr.py)

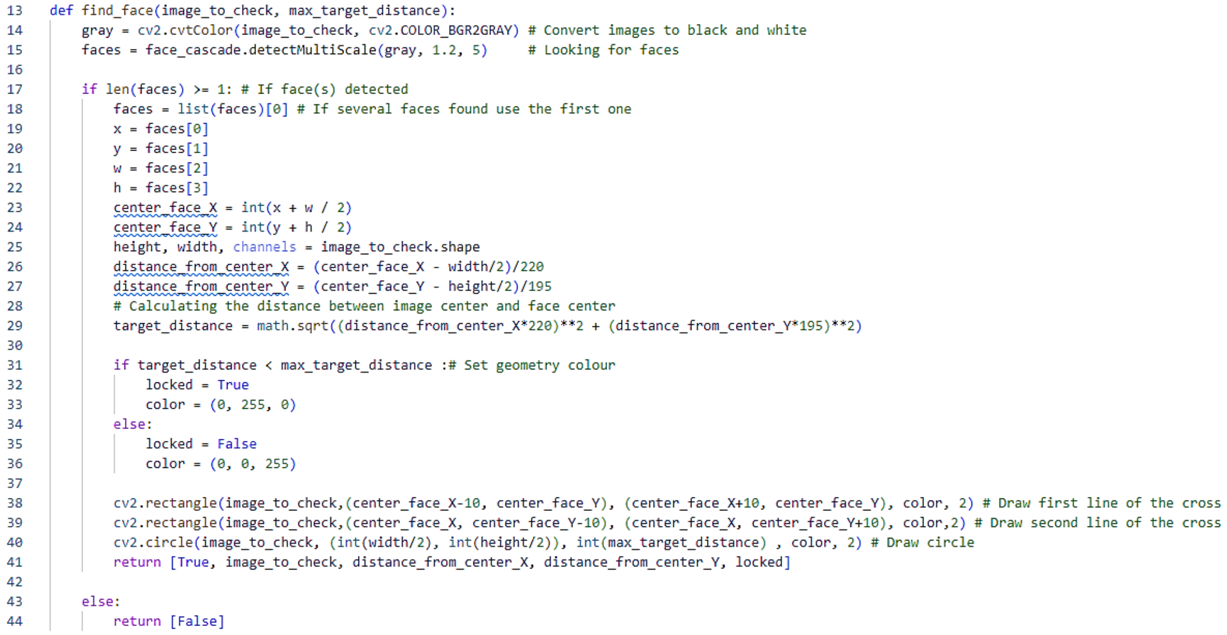
The face-tracking program uses the cv2 library. The file is called *opr.py* and it contains the pre-sets for face-recognition. There are two functions, that are defined in the program, remap, and find_face.



```python
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
def remap(value_to_map, new_range_min, new_range_max, old_range_min, old_range_max):
    remapped_val = (value_to_map - old_range_min) * (new_range_max - new_range_min) / (old_range_max - old_range_min) + new_range_min
    if(remapped_val>new_range_max):
        remapped_val = new_range_max
    elif (remapped_val < new_range_min):
        remapped_val = new_range_min
    return remapped_val
```

Fig 30: remap function in opr.py

Fig 30 shows the first part of the program. Initially the cascade classifier file is run, and since the robot is to detect and track faces, the classifier file that will be used is haarcascade_frontalface_default.xml. This file is pre-trained to detect faces, provide points for drawing shapes, detecting emotions, race, and various other things (Kumar, 2021). The remap function is used for remapping and defining new values for the viewing field of the robot's motion. Since OOP is used, the use of this function will be made clearer in the main code.



```python
def find_face(image_to_check, max_target_distance):
    gray = cv2.cvtColor(image_to_check, cv2.COLOR_BGR2GRAY) # Convert images to black and white
    faces = face_cascade.detectMultiScale(gray, 1.2, 5)     # Looking for faces

    if len(faces) >= 1: # If face(s) detected
        faces = list(faces)[0] # If several faces found use the first one
        x = faces[0]
        y = faces[1]
        w = faces[2]
        h = faces[3]
        center_face_X = int(x + w / 2)
        center_face_Y = int(y + h / 2)
        height, width, channels = image_to_check.shape
        distance_from_center_X = (center_face_X - width/2)/220
        distance_from_center_Y = (center_face_Y - height/2)/195
        # Calculating the distance between image center and face center
        target_distance = math.sqrt((distance_from_center_X*220)**2 + (distance_from_center_Y*195)**2)

        if target_distance < max_target_distance :# Set geometry colour
            locked = True
            color = (0, 255, 0)
        else:
            locked = False
            color = (0, 0, 255)

        cv2.rectangle(image_to_check,(center_face_X-10, center_face_Y), (center_face_X+10, center_face_Y), color, 2) # Draw first line of the cross
        cv2.rectangle(image_to_check,(center_face_X, center_face_Y-10), (center_face_X, center_face_Y+10), color,2) # Draw second line of the cross
        cv2.circle(image_to_check, (int(width/2), int(height/2)), int(max_target_distance) , color, 2) # Draw circle
        return [True, image_to_check, distance_from_center_X, distance_from_center_Y, locked]

    else:
        return [False]
```

Fig 31: find_face function in opr,py

Then comes the find_face function. This function performs various functions such as colour conversion (from colour to black and white), using the cvtColor command. This command exists in the cv2 library and then to convert from colour(BGR) to black and white, the command cv2.COLOR_BGR2GRAY is used. In case there are multiple faces in the frame, there is a command faces=list(faces)[0] that selects only the first face to track. Then the x-coordinates, y-coordinates, width, and height are set for drawing the shapes on the face. In order to keep the camera centred, the distance between the image centre and the face centre is calculate in the target_distance variable, and is defined as:

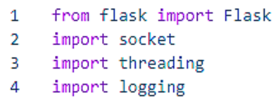
$$\sqrt{}(distance\_from\_center\_X * 220) ** 2 + (distance\_from\_center\_Y * 195) ** 2$$

Equation 1: target_distance equation

Based on the distance of the face from the centre of the image, the colour of the shape changes to indicate whether the face has been detected or not. This is a rather simple piece of code and is one of three python codes.

4.3.2 Python Arduino Code (arduino.py)

Since the Arduino isn't capable of directly understanding the data sent from the webcam directly, and neither is it capable of performing MV tasks on its own. To facilitate this task, the image processing is done in the laptop, in python. All the Arduino has to do now, is get the angles for the servo motor to move to from the python code after the image processing is done. In this robot, the data is transmitted wirelessly via the internet, hence certain new libraries have to be used.

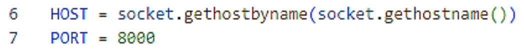


```python
from flask import Flask
import socket
import threading
import logging
```

Fig 32: Libraries imported for talking with Arduino

Flask is one such library. It is a module in python that enable the use of a web application framework without worrying too much about the low-level details like thread-management and protocol. It allows the making of localhosts easy and is very easy to get started with because of its Pythonic framework.

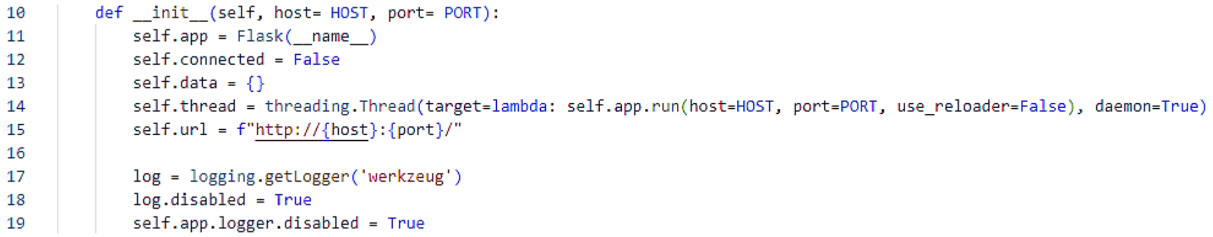
Socket is used to create a specific *socket node* to listen to a specific port and IP address while another socket is used to request a connection (Learn Python 101, 2022). The gethostbyname function returns one IP address. PORT 8000 uses a defined protocol (iRDMI) to TCP Protocol to communicate depending on the application. It is one of the main protocols in TCP/IP networks and deals with two hosts, to establish a connection and exchange streams of data, which in our case is the servo motor angles. TCP is used as it guarantees the transfer of data in the same order it is sent (Audit My PC, n.d.).



```python
HOST = socket.gethostbyname(socket.gethostname())
PORT = 8000
```

Fig 33: socket function in arduino.py

Threading is a library that allows two things to happen at once. It is used to get tasks to run simultaneously and doesn't necessarily mean that the task will run faster, or it'll run on a different processor. Logging is a command used for tracking events that happens when a software/code is run.



```python
    def __init__(self, host= HOST, port= PORT):
        self.app = Flask(__name__)
        self.connected = False
        self.data = {}
        self.thread = threading.Thread(target=lambda: self.app.run(host=HOST, port=PORT, use_reloader=False), daemon=True)
        self.url = f"http://{host}:{port}/"

        log = logging.getLogger('werkzeug')
        log.disabled = True
        self.app.logger.disabled = True
```

Fig 34: __init__ function in arduino.py

Werkzeug is a comprehensive WSGI web application library. It is used in this program to wrap other WSGI applications and handle the data stream. The WSGI is vital for Flask applications and describes the communication between the python application and web servers. Our python application provides *callable* objects which contain certain functionalities that invoke the WSGI application (Stud, 2020). This is explained in Fig 35, where the user is the Arduino 33 IoT board.

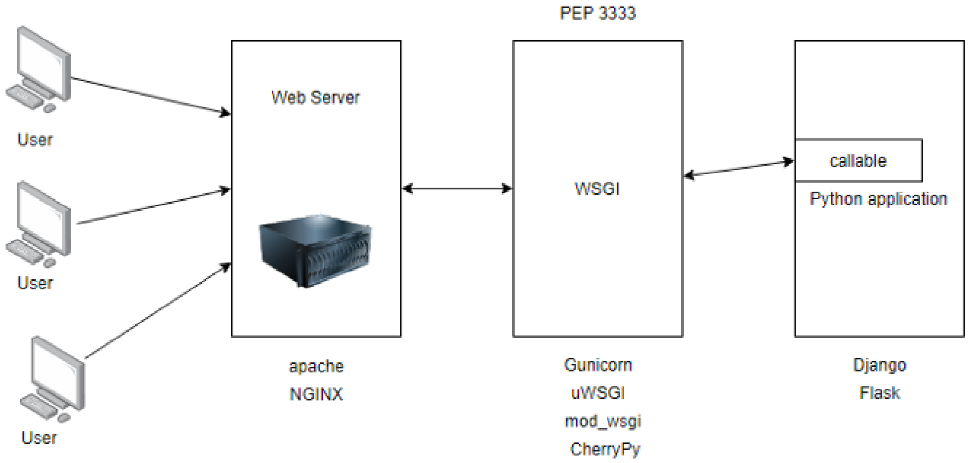
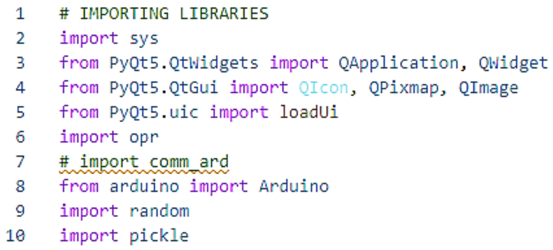


Fig 35: WSGI working

4.3.3 Main Program (Camera_tracker_main.py)

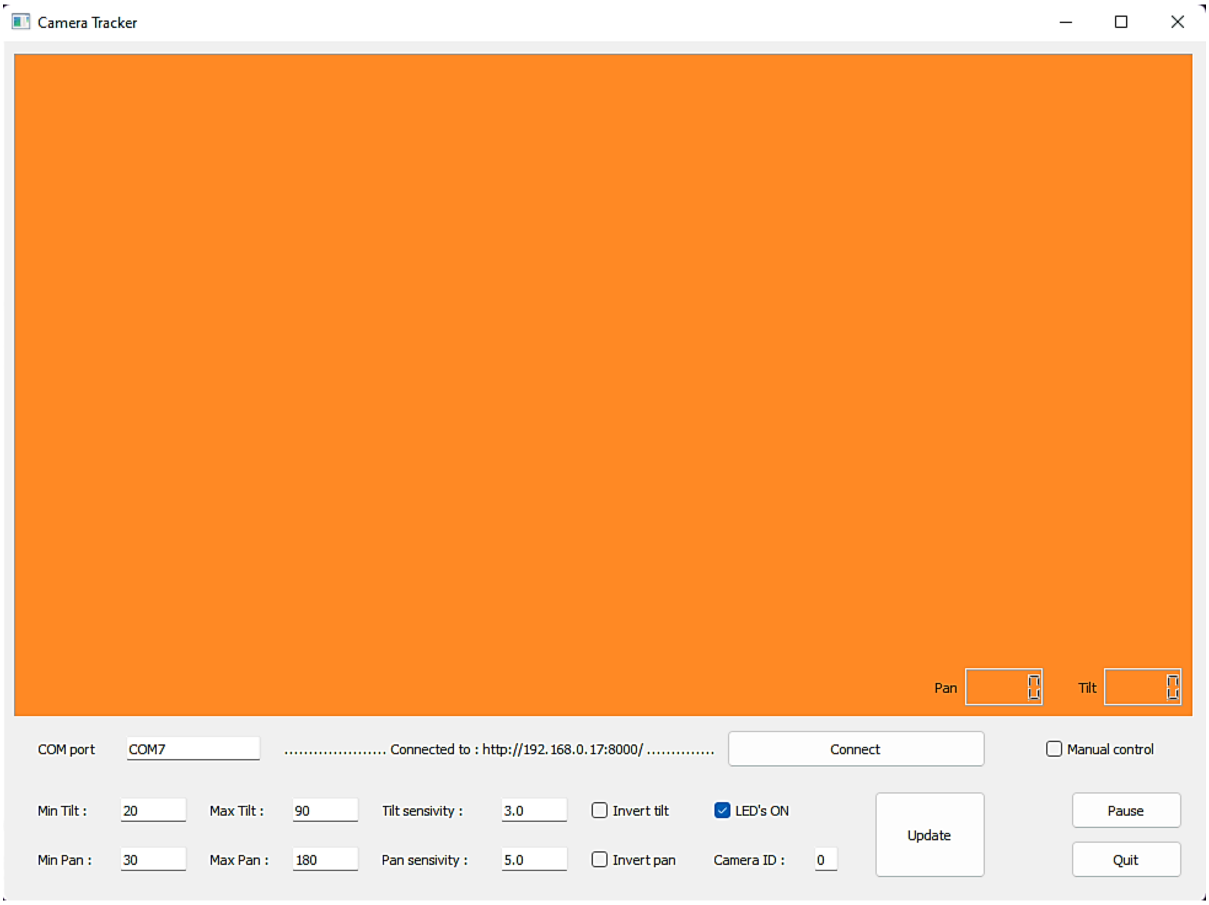
The main program is where all the above created programs are run. This section won't cover the code in detail, but it will try to explain some important parts of the program.



```python
# IMPORTING LIBRARIES
import sys
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon, QPixmap, QImage
from PyQt5.uic import loadUi
import opr
# import comm_ard
from arduino import Arduino
import random
import pickle
```

Fig 36: Libraries imported in Camera_tracker_main.py

The PyQt5 is a GUI toolkit that is capable of cross-platform usage. The toolkit can be used for front-end and back-end applications. It has a lot of features such as QtWidgets where UI elements are created, and user input events are handles. QtGUI module provides classes for OpenGL, windowing systems integration, event handling, font, text, 2D graphics etc. The tracker.ui file contains information regarding the windowpane, where all the widget, buttons, and other functions are defined. These functions are for the UI, in which the user first interacts with the robot to setup the values for the max and min tilt angles and also the sensitivity of the tilt. There is also the option of inverting the tilt of the robot's motion. The values entered here will be transmitted to the python code and updated to the Arduino. The coolest function is "manual control" in which the robot can be controlled with the mouse itself. The mouse behaves like a joystick and the updated movements are sent to the robot (see Fig 37: Camera Tracker UI). One of the imported libraries is the pickle library, which enables the serializing and de-serializing of python object structure. This library will allow the transfer of data easily from one server/system to another, and then store that information in a file or database.



Camera Tracker

Pan

Tilt

COM port COM7

.... Connected to : http://192.168.0.17:8000/ ....

□ Manual control

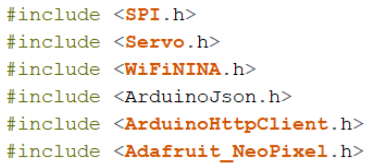
Min Tilt : 20 Max Tilt : 90 Tilt sensivity : 3.0 □ Invert tilt ☑ LED's ON

Min Pan : 30 Max Pan : 180 Pan sensivity : 5.0 □ Invert pan Camera ID : 0

Fig 37: Camera Tracker UI

4.3.4 Arduino Code (Camera_Tracking_Neo_Pixel.ino)

Since the Arduino uses the NINA-W102 module for internet, it needs the WiFiNINA.h library to connect to the internet. The Arduino communicates with the Wi-Fi using the SPI bus. The module can be used to connect to an encrypted or an open network(WEP, WPA) and the IP address can be assigned through a DHCP or statically. The WiFiNINA.h library can be used to instate servers, clients and send/receive UDP packets through Wi-Fi (Arduino, 2022). This library can also manage DNS.

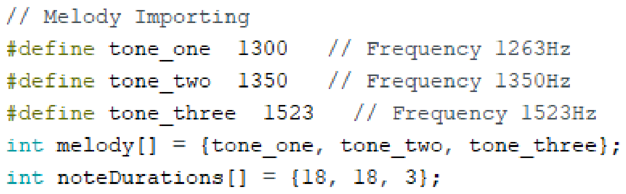


```
#include <SPI.h>
#include <Servo.h>
#include <WiFiNINA.h>
#include <ArduinoJson.h>
#include <ArduinoHttpClient.h>
#include <Adafruit_NeoPixel.h>
```

Fig 38: Packages imported in Arduino Program

Along with the Wi-Fi related libraries (SPI.h, WiFiNINA.h), there are other network related libraries that are installed. The ArduinoJson.h is a very popular, well known for its self-contained capabilities, amazing security (integrated with OSS-Fuzzing), is open-source and is extremely efficient among many other advantages of this library. The ArduinoHttpClient.h is a class that performs HTTP requests such as GET, POST and PUT.

Then there are the libraries that are used to control the electronics. For controlling the servo's, the Servo.h library is used. It provide the servos with information such as angle and speed of rotation, via the digital pins, using PWM signals. The Adafruit_NeoPixel.h library is used to control the Neopixel Stick. The stick is given with commands to make it produce certain animated effects, to make the robot feel more "*alive*" and vibrant. It should be able to respond and provide some visual feedback.



```
// Melody Importing
#define tone_one  1300   // Frequency 1263Hz
#define tone_two  1350   // Frequency 1350Hz
#define tone_three  1523   // Frequency 1523Hz
int melody[] = {tone_one, tone_two, tone_three};
int noteDurations[] = {18, 18, 3};
```

Fig 39: Audio melody imported files

As mentioned above, the robot is also equipped with a piezoelectric buzzer, that is used to produce audio feedback. But this audio isn't just a simple single note response every time an update is given, the buzzer plays a melody. This melody is played by tuning the frequency at which the signals are sent to the buzzer. A fast high frequency sound can be achieved by giving a high frequency signal to the microphone, and the vice versa for a low frequency output. This is performed by using PWM signals.

CHAPTER – 5

The Final Thoughts

5 Conclusion

5.1 Possible Improvements

The DRK-1 has been designed to be future ready and accommodate a lot of interesting upgrades. There are a lot of possible upgrades that can be done to improve the functionality and performance of the robot, to make it more suitable for its intended purpose. This section will explore the possible improvements that can be made, and also try to provide concrete reasons as to how these upgrades would improve the robot.

- One main issue with the robot is that the wire lengths were not cut as per measurement, which led to the problem of excessive wiring. For a robot of this scale, even milli-meter (mm) matters, and if there is too much wiring living in the mid-section, the mating of the Top Lid and the Mid-Section becomes cumbersome.
- Since the robot is going to be at the desktop side at all times and is already capable of facing the user. Adding some extra features such as gesture control, to control the PC would improve the utility of this robot. OpenCV already has gesture recognition capabilities, and this can be used to control some functionality of the PC such as volume, brightness etc.
- The overall design of the robot can be improved to make it more appealing. If it were to be released to the consumer market, a much more appealing design would be required, something that is competent enough to sit on a tabletop and look amazing. The RGB lighting needs a diffuser so that it doesn't sting the eyes when turner on. The design could be a modern type of look, or it could be a cartoony type, where the robot can make sounds and move in an *animated* fashion.
- The issue with the robot currently is that it is not remote, and needs a power connection, and the wire from the webcam to travel from the robot to the PC for image processing. But if these were to be performed locally, it would be amazing. To make the robot independent of the external power supply, li-ion or li-po batteries could be used, with a magnetic charging port. In order to eliminate the camera wire, an ESP32 CAM can be used, and paired with a powerful microcontroller like the Raspberry Pi 2040. There are many machine vision kits that are available in the market, and they can be used to make the profile of the robot smaller.
- Adding the feature of receiving a live feed of the room from anywhere in the world would prove to be such an asset. Along with this, if the robot is able to send names, and photos, in case a face is detected whenever somebody enters a room when they are not supposed to, the robot can send a photo, as well as the name of the person, if they are registered in the database.

Peng Zhihui has made an amazing robot with most of the above improvements. It is an extremely complex robot with some really cool capabilities and functionalities. Definitely worth a look. Information can be found on (Rowntree, 2022), and details about the robot are available on the GitHub page (Zhihui, 2022)(Fig 40).

Fig 40: Peng Zhihui's ElectronBot

5.2 Concluding Statement

Computer vision has been along with us for quite some time now. There are a lot of advanced functionalities and research that is being done in this field almost every single day. The dawn of Industry 4.0 has also brought in interconnected devices, that have made their place in our homes and our personal lives in the form of smart home accessories, or activity trackers. I explored throughout this thesis report, the use of a MV robot to perform the face tracking task using the OpenCV library and some features of IoT. Although this device isn't very advanced in its functionality, it is "*tomorrow* ready", meaning that it has the resources for further improvement and can be worked upon to make it even more advanced and complex without any additional hardware required. There are a plethora of advantages when having a robot that sits beside the PC, with a webcam on top, for following your face everywhere. The possibility of adding certain advanced features is very high because of how easy it is to use the OpenCV library, and this robot would become a must have accessory. The robot will be worked upon and will be made more and more advanced with time.

CHAPTER – 6

The References

6 References

Adafruit, n.d. *1426 - Neopixel Stick.* [Online], Available at: https://www.adafruit.com/product/1426 [Accessed 24 Aug 2022].

APC International Ltd., 2015. *Principles of Piezoelectric Energy Harvesting.* [Online], Available at: https://www.americanpiezo.com/blog/piezoelectric-energy-harvesting/#:~:text=Principles%20Piezoelectricity%20is%20found%20in%20crystalline%20materials%20that,to%20an%20applied%20electric%20field%20%28converse%20piezoelectric%20effect%29. [Accessed 24 8 2022].

Arduino, 2022. *Arduino - WiFiNINA.* [Online], Available at: https://www.arduino.cc/reference/en/libraries/wifinina/, [Accessed 27 Aug 2022].

Ashton, K., 2009. That 'Internet of Things' Thing. *RFID Journal,* 22(7), pp. 97-114.

Atzori, L., Iera, A. & Morabito, G., 2010. The Internet of Things: A survey. *Computer Networks,* 54(15), pp. 2787-2805.

Audit My PC, n.d. *Audit My PC - TCP 8000.* [Online] , Available at: https://www.auditmypc.com/tcp-port-8000.asp#:~:text=Side%20note%3A%20TCP%20port%208000%20uses%20the%20Transmission,establish%20a%20connection%20and%20exchange%20streams%20of%20data. [Accessed 25 Aug 2022].

Bhatt, A., n.d. *Engineers Garage - Piezo Buzzer.* [Online], Available at: https://www.engineersgarage.com/piezo-buzzer/, [Accessed 24 Aug 2022].

Bradski, G. & Kaehler, A., 2016. *Learning OpenCV 3: Computer vision with the OpenCV library.* s.l.:O'Reilly Media Inc..

B, S. B., Mohamed, A. S. & Al-Atroshi, C., 2018. Adaptability of SOA in IoT Services – An Empirical Survey. *International Journal of Computer Applications (0975 – 8887),* 182(31), p. 8887.

D., B. & J., S., 2011. Internet of Things: Applications and Challenges in Technology and Standardization. *Wireless Pers Commun,* 58(1), pp. 49-69.

eTechnophiles, n.d. *Beginners Guide to 1N4007 Diode- Specs, Pinout, Equivalent.* [Online] Available at: https://www.etechnophiles.com/1n4007-diode-specs-pinout-equivalent/#:~:text=%201N4007%20Diode%20Specifications%20%201%201N4007%20diode,square%20waves%20of%20greater%20than%2015...%20More%20, [Accessed 24 Aug 2022].

Freund, Y. & Schapire, R., 1997. A Decision-Theoretic Generalization of Online Learning and an Application to Boosting. *Journal of Computer and System Sciences,* 55(1), pp. 119-139.

Garg, S., 2021. *Geek for Geeks - Service Oriented Architecture.* [Online], Available at: https://www.geeksforgeeks.org/service-oriented-architecture/, [Accessed 24 Aug 2022].

Guinard, D. et al., 2010. Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services. *IEEE Transactions on Services Computing,* 3(3), pp. 223-235.

Harville, M., Gordon, G. & Woodfill, J., 2001. Foreground Segmentation Using Adaptive Mixture Models in Color and Depth. *Proceedings IEEE Workshop on Detection and Recognition of Events in Video,* pp. 3-11.

Ilie-Zudor, E. et al., 2011. A Survey of Applications and Requirements of Unique Identification Systems and RFID Techniques. *Computers in Industry,* 62(3), pp. 227-252.

J. R., K. R. & S. B., 1995. *Machine Vision.* (Vol. 5, pp. 309-364) ed. New York: McGraw-hill.

JST, n.d. *PH Connector.* [Online], Available at: https://www.jst.com/products/crimp-style-connectors-wire-to-board-type/ph-connector/, [Accessed 24 Aug 2022].

Kumar, R., 2021. *Medium.com - Human Face, emotion and race detection with python.* [Online] Available at: https://medium.com/analytics-vidhya/human-face-emotion-and-race-detection-with-python-86ca573e0c45#:~:text=What%20is%20the%20haarcascade_frontalface_default.xml%20file%3F%20It%20is%20a,squares%2C%20or%20any%20shapes%20on%20the%20face.%20Now, [Accessed 25 Aug 2022].

Learn Python 101, 2022. *Learn Python 101 - Sockets in Python.* [Online], Available at: https://learnpython101.com/sockets-in-python#:~:text=Python%20creates%20an%20object%20of%20the%20socket%20using,the%20other%20communicating%20node%20to%20request%20the%20connection., [Accessed 25 Aug 2022].

Li, S., Xu, L. D. & Wang, X., 2012. Compressed Sensing Signal and Data Acquisition in Wireless Sensor Networks and Internet of Things. *IEEE Transactions on Industrial Informatics,* 9(4), pp. 2177-2186.

M, A. & G.M., K., 2014. Security and privacy in the Internet of Things: Current Status and Open Issues. *International Conference on Privacy and Security in Mobile Systems (PRISMS),* pp. 1-8.

Mahmoud, R., Yousuf, T., Aloul, F. & Zualkernan, I., 2015. *Internet of things (IoT) security: Current Status, Challenges and Prospective Measures.* s.l., IEEE, pp. 336-341.

Perera, C. et al., 2013. Dynamic Configuration of Sensors using Mobile Sensor Hub in Internet of Things Paradigm. *IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing,* pp. 473-478.

Rowntree, D., 2022. *HackADay - ELECTRONBOT: A SWEET MINI DESKTOP ROBOT THAT TICKS ALL THE BOXES.* [Online], Available at: https://hackaday.com/2022/03/20/__trashed-6/, [Accessed 25 Aug 2022].

Sikder, A. K. et al., 2021. A Survey on Sensor-Based Threats and Attacks to Smart Devices and Applications. *IEEE Communications Surveys & Tutorials,* 23(2), pp. 1125-1159.

Stud, P., 2020. *Medium - What is WSGI (Web Server Gateway Interface)?.* [Online], Available at: https://medium.com/analytics-vidhya/what-is-wsgi-web-server-gateway-interface-ed2d290449e [Accessed 25 Aug 2022].

T. K., K. J., B. B. & M. B., 1999. *Wallflower: Principles and Practice of Background Maintenance.* s.l., IEEE, pp. 255-261.

V. P. & J. M., 2004. Robust Real-Time Face Detection. *International Journal of Computer Vision,* 57(2), pp. 137-154.

Wu, Y., Sheng, Q. Z. & Zeadally, S., 2013. *RFID: Opportunities and Challenges,* London: Springer.

Xu, L. D., He, W. & Li, S., 2014. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics,* 10(4), pp. 2233-2243.

Zhang, Y., Chen, J.-L. & Cheng, B., 2017. Integrating Events into SOA for IoT Services. *IEEE Communications Magazine,* 55(9), pp. 180-186.

Zhihui, P., 2022. *GitHub - ElectronBot: Mini Desktop Robot.* [Online], Available at: https://github.com/peng-zhihui/ElectronBot, [Accessed 25 Aug 2022].

CHAPTER – 7

The Appendix

7 Appendix

7.1 Drawings

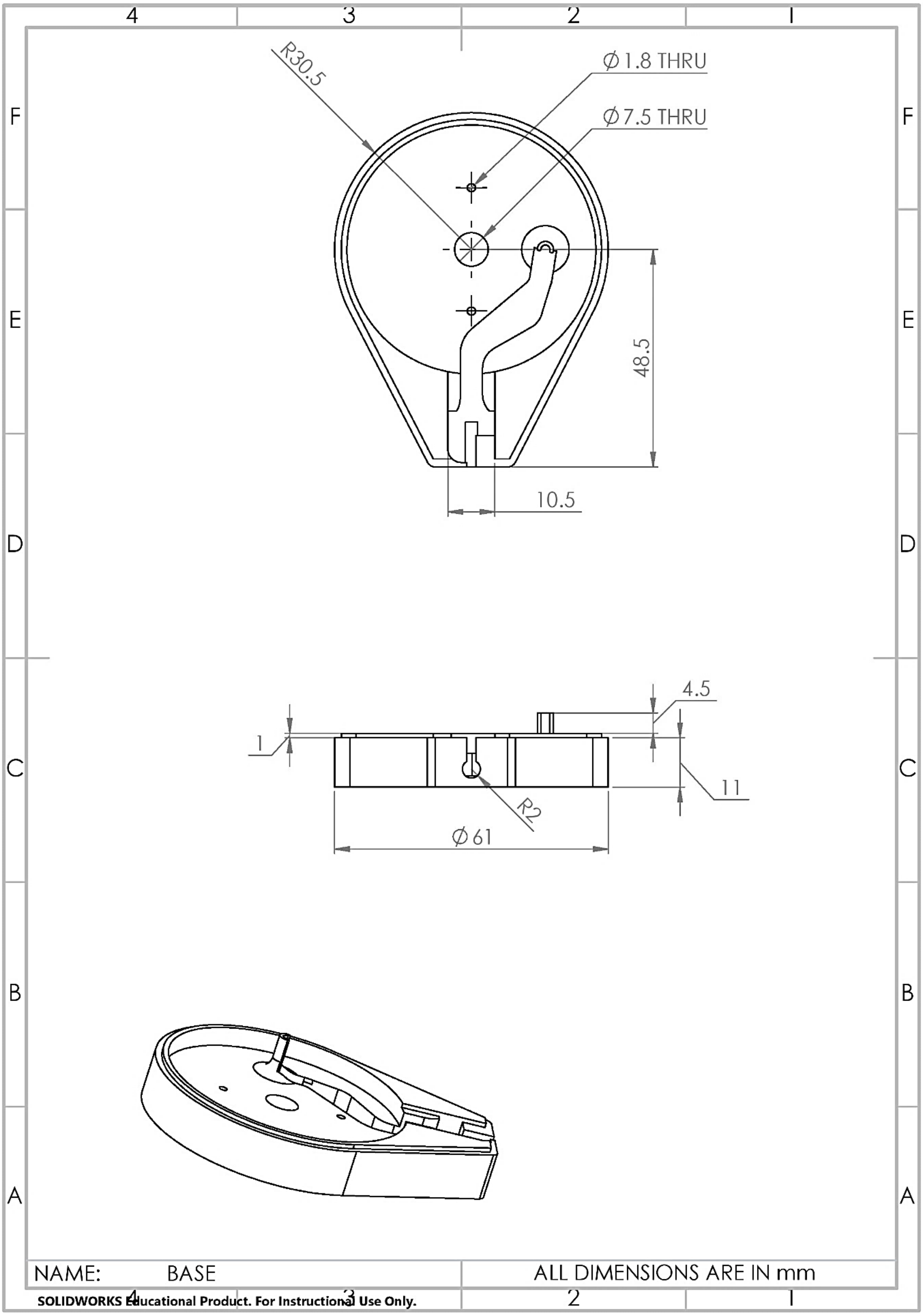


Fig 41: Base Part Diagram

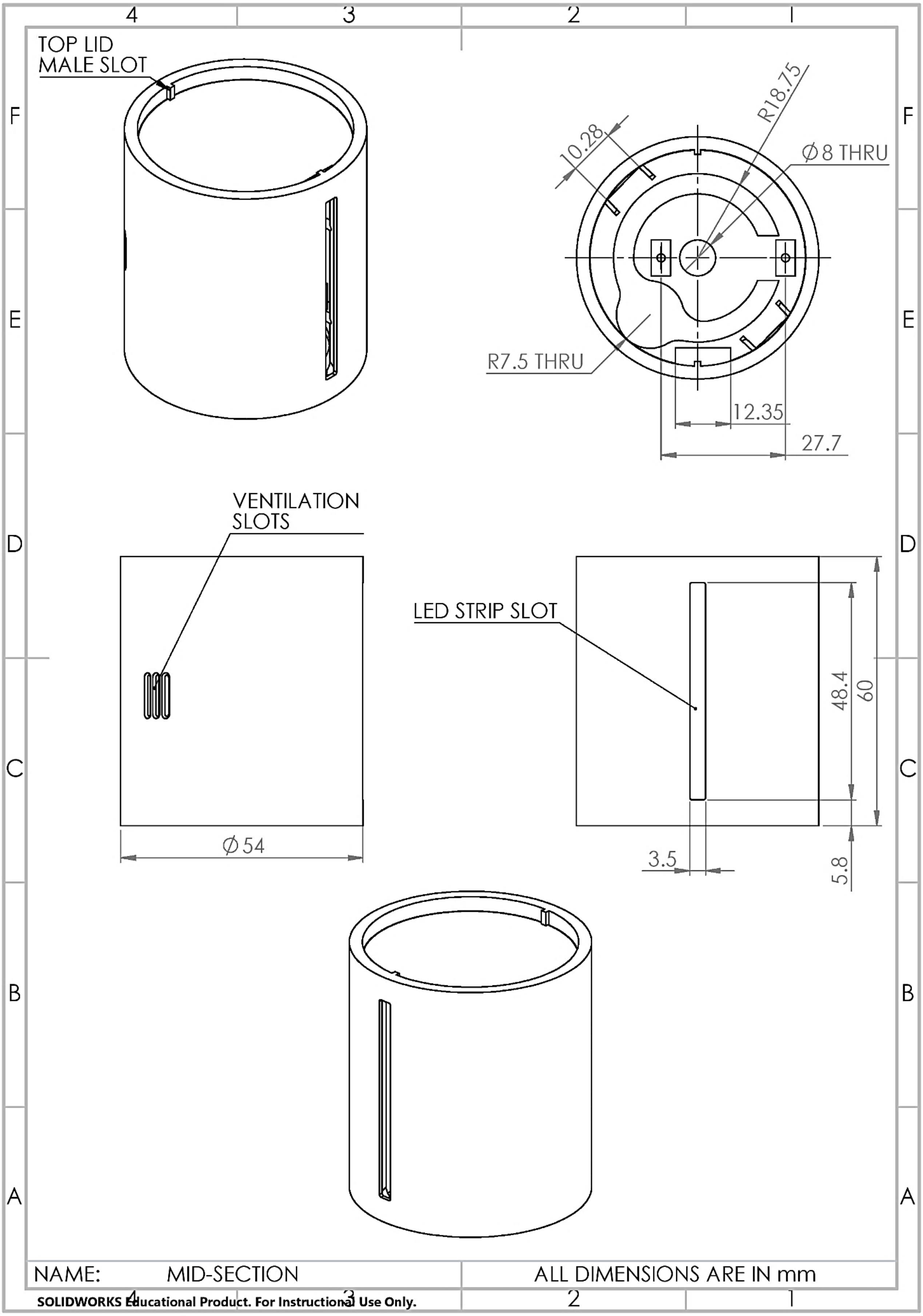


Fig 42: Mid-Section Drawing

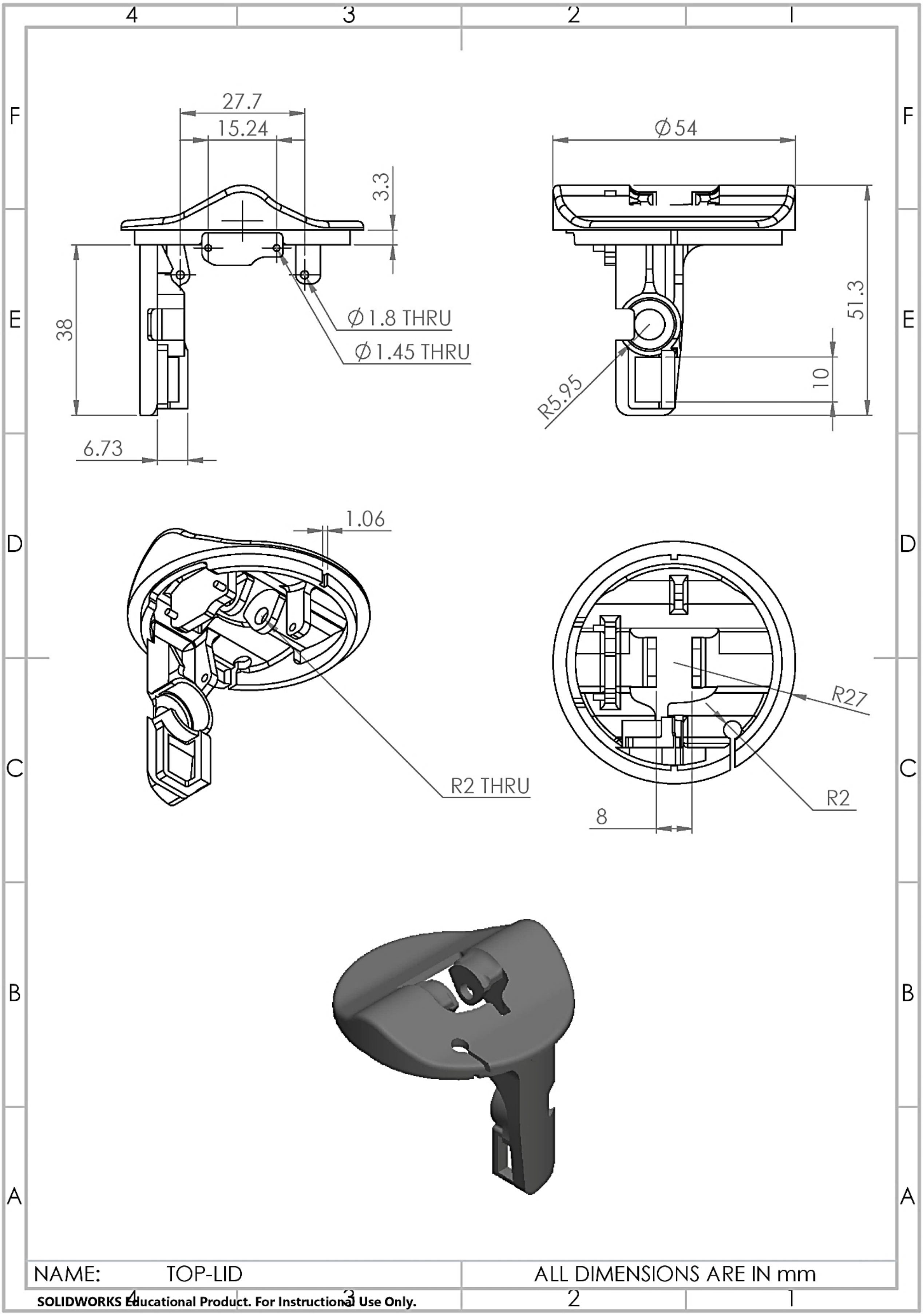


Fig 43: Top Lid Drawing

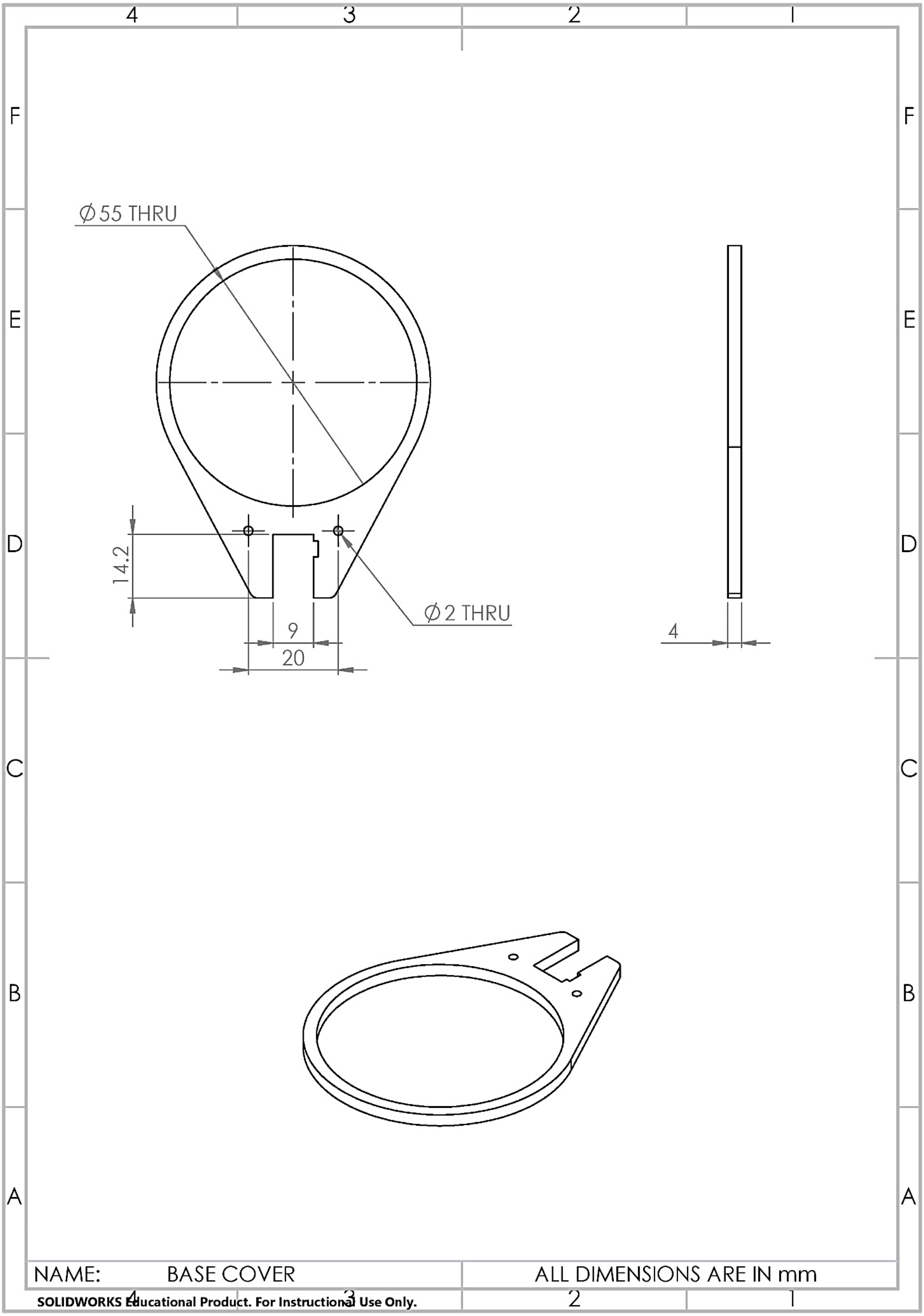


Fig 44: Base Cover Drawing

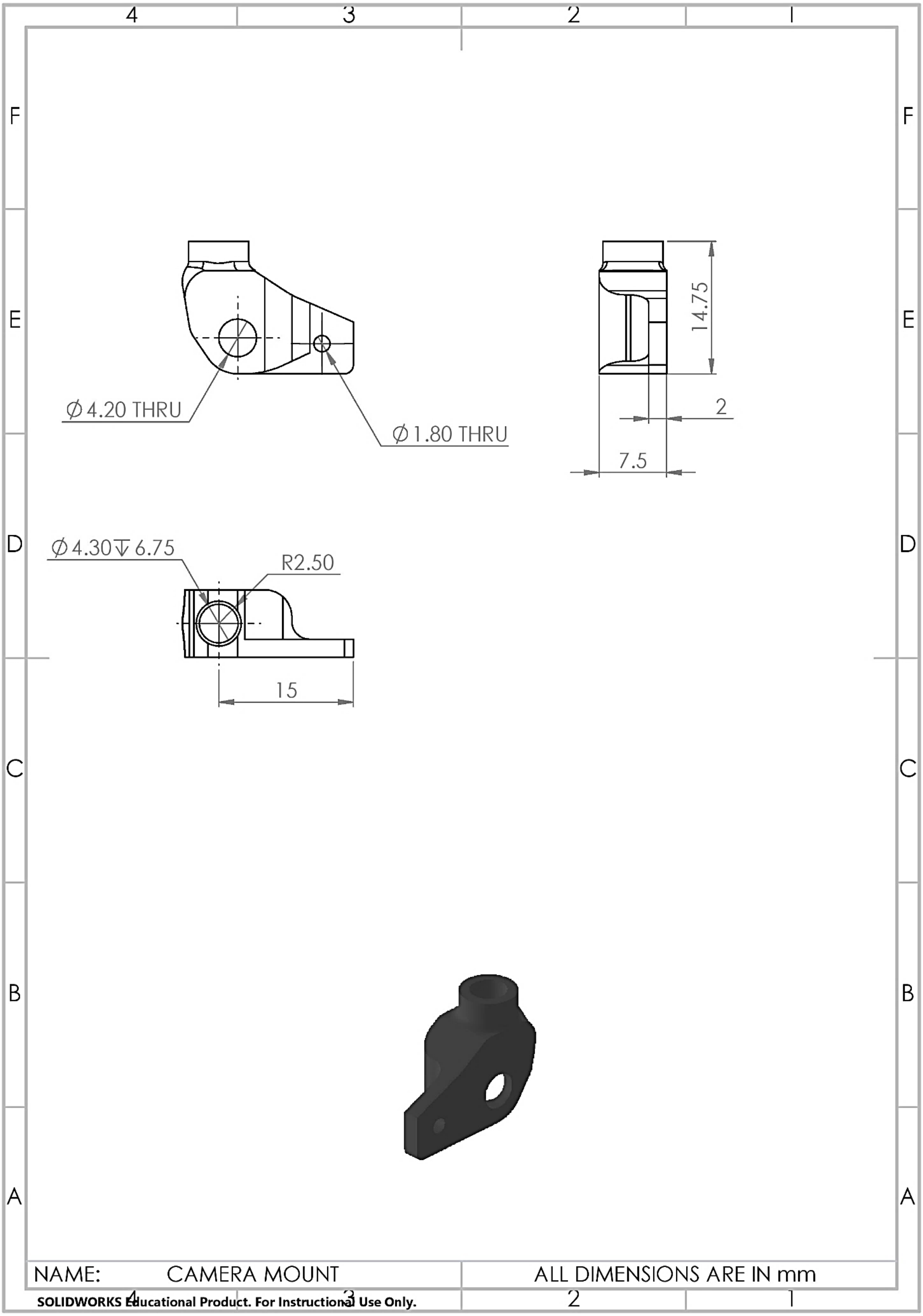


Fig 45: Camera Mount Drawing

7.2 Code

7.2.1 Face Detection Code (opr.py)

```
import cv2
import math

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
def remap(value_to_map, new_range_min, new_range_max, old_range_min, old_range_max):
    remapped_val = (value_to_map - old_range_min) * (new_range_max - new_range_min) / (old_range_max - old_range_min) + new_range_min
    if(remapped_val>new_range_max):
        remapped_val = new_range_max
    elif (remapped_val < new_range_min):
        remapped_val = new_range_min
    return remapped_val

def find_face(image_to_check, max_target_distance):
    gray = cv2.cvtColor(image_to_check, cv2.COLOR_BGR2GRAY) # Convert images to black and white
    faces = face_cascade.detectMultiScale(gray, 1.2, 5)     # Looking for faces

    if len(faces) >= 1: # If face(s) detected
        faces = list(faces)[0] # If several faces found use the first one
        x = faces[0]
        y = faces[1]
        w = faces[2]
        h = faces[3]
        center_face_X = int(x + w / 2)
        center_face_Y = int(y + h / 2)
        height, width, channels = image_to_check.shape
        distance_from_center_X = (center_face_X - width/2)/220
        distance_from_center_Y = (center_face_Y - height/2)/195
        target_distance = math.sqrt((distance_from_center_X*220)**2 + (distance_from_center_Y*195)**2) # Calculating the distance between image
center and face center

        if target_distance < max_target_distance :# Set geometry colour
            locked = True
            color = (0, 255, 0)
        else:
            locked = False
            color = (0, 0, 255)

        cv2.rectangle(image_to_check,(center_face_X-10, center_face_Y), (center_face_X+10, center_face_Y), color, 2) # Draw first line of the cross
        cv2.rectangle(image_to_check,(center_face_X, center_face_Y-10), (center_face_X, center_face_Y+10), color,2) # Draw second line of the cross
        cv2.circle(image_to_check, (int(width/2), int(height/2)), int(max_target_distance) , color, 2) # Draw circle
        return [True, image_to_check, distance_from_center_X, distance_from_center_Y, locked]

    else:
        return [False]
```

7.2.2 Python Arduino Code (arduino.py)

```python
from flask import Flask
import socket
import threading
import logging

HOST = socket.gethostbyname(socket.gethostname())
PORT = 8000

class Arduino:
    def __init__(self, host= HOST, port= PORT):
        self.app = Flask(__name__)
        self.connected = False
        self.data = {}
        self.thread = threading.Thread(target=lambda: self.app.run(host=HOST, port=PORT, use_reloader=False), daemon=True)
        self.url = f"http://{host}:{port}/"

        log = logging.getLogger('werkzeug')
        log.disabled = True
        self.app.logger.disabled = True

    def connect(self):
        if not self.connected:
            try:
                @self.app.route('/')
                def home_page():
                    return self.data

                self.thread.start()
                print(f"Streaming data to url: http://{HOST}:{PORT}/")
                self.connected = True
                return True
            except:
                pass
        return False

    def sendData(self, data: dict):
        self.data = data

if __name__=='__main__':
    a = Arduino()
    a.connect()
```

7.2.3 Main Python Code (Camera_tracker_main.py)

```python
# IMPORTING LIBRARIES
import sys
from PyQt5.QtWidgets import QApplication, QWidget
from PyQt5.QtGui import QIcon, QPixmap, QImage
from PyQt5.uic import loadUi
import opr
from arduino import Arduino
import random
import pickle
import cv2

class App(QWidget):

    def __init__(self):
        super().__init__()
        self.ui = loadUi('tracking.ui', self)#Load .ui file that define the window layout(buttons, tick boxoex)

        self.setMouseTracking(True)    # Allows mouse tracking(use during manual mode)
        self.manual_mode = False       # Set manual mode to false to start in tracking face mode
        self.LED_ON = True             # Set LED on mode
        self.CameraID = 0              # Define camera ID, first camera = ID 0, second ID 1 and so on

        self.rec = True                # Allows to start camera recording
        self.cap = cv2.VideoCapture(self.CameraID)  # Define OpenCV camera recording
        self.cap.set(3, 960)                        # Set capture width
        self.cap.set(4, 540)                        # Set capture height

        self.min_tilt = 22          # Minimum tilt angle in degree (up/down angle)
        self.max_tilt = 80          # Maximum tilt angle in degree
        self.current_tilt = 0       # Current tilt (info received from arduino and displayed in LCD numbers)
        self.target_tilt = 90       # Tilt angle to reach

        self.min_pan = 80           # Minimum pan angle in degree(left/ right angle)
        self.max_pan = 100          # Maximum pan angle in degree
        self.current_pan = 80       # Current pan (info received from arduino and displayed in LCD numbers)
        self.target_pan = 90        # Pan angle to reach

        self.roam_target_pan = 90
        self.roam_target_tilt = 90
        self.roam_pause = 40# Amount of frame the camera is going to pause for, when roam tilt or pan target 
reached
        self.roam_pause_count = self.roam_pause   # Current pause frame count

        self.is_connected = False   # Boolean defining if Arduino is connected

        self.InvertPan = False      # Inverting Pan
        self.InvertTilt = False     # Inverting Tilt

        self.face_detected = False  # Face detection boolean
        self.target_locked = False  # Face detection and close to center boolean
        self.max_target_distance = 40  # Minimum distance between face/center of image for setting target locked
        self.max_empty_frame = 50   # Number of empty frame (no face detected) detected before starting roaming
        self.empty_frame_number = self.max_empty_frame   # Current empty frame count
        self.ard = Arduino()     # Create object allowing communicationn with arduino
        self.initUI()    # Set-up UI(given below)

    def initUI(self):     # UI related stuff
        self.setWindowTitle('Camera Tracker')              # Set window title
        self.label = self.ui.label                  # Set label (it will be used to display the captured images)
        self.QuitButton = self.ui.QuitButton            # Set quit button
        self.PauseButton = self.ui.PauseButton          # Set pause button
        self.Pan_LCD = self.ui.Pan_LCD                  # And so on...
        self.Tilt_LCD = self.ui.Tilt_LCD                # ...
        self.Manual_checkbox = self.ui.Manual_checkbox  # ...
        self.ConnectButton = self.ui.ConnectButton
        self.COMlineEdit = self.ui.COMlineEdit
        self.COMConnectLabel = self.ui.COMConnectLabel
        self.UpdateButton = self.ui.UpdateButton
        self.MinTiltlineEdit = self.ui.MinTiltlineEdit
        self.MaxTiltlineEdit = self.ui.MaxTiltlineEdit
        self.InvertTilt_checkbox = self.ui.InvertTilt_checkbox
        self.InvertTilt = self.InvertTilt_checkbox.isChecked()
        self.MinPanlineEdit = self.ui.MinPanlineEdit
        self.MaxPanlineEdit = self.ui.MaxPanlineEdit
        self.InvertPan_checkbox = self.ui.InvertPan_checkbox
        self.InvertPan = self.InvertPan_checkbox.isChecked()
        self.TiltSensivityEdit = self.ui.TiltSensivityEdit
        self.TiltSensivity = 1
        self.PanSensivityEdit = self.ui.PanSensivityEdit
        self.PanSensivity = 1
        self.LED_checkbox = self.ui.LED_checkbox
        self.CameraIDEdit = self.ui.CameraIDEdit

        self.QuitButton.clicked.connect(self.quit)              # Binding quit button to quit method
        self.PauseButton.clicked.connect(self.toggle_recording) # Binding pause button to pause method
        self.Manual_checkbox.stateChanged.connect(self.set_manual_mode)
        self.ConnectButton.clicked.connect(self.connect)
        self.UpdateButton.clicked.connect(self.update_angles)
        self.load_init_file()
        self.update_angles() # Updating angle method
        self.record()        # Start recording

    def load_init_file(self):
    # To  allow to reload the latest values entered in text boxes even after closing the software
        try:        # Tries to open init file if existing
            with open('init.pkl', 'rb') as init_file:
                var = pickle.load(init_file)  # Load all variable and update text boxes
                self.COMlineEdit.setText(var[0])
                if(var[4]):
                    self.MinTiltlineEdit.setText(str(var[2]))
                    self.MaxTiltlineEdit.setText(str(var[1]))
                else:
                    self.MinTiltlineEdit.setText(str(var[1]))
                    self.MaxTiltlineEdit.setText(str(var[2]))
                self.TiltSensivityEdit.setText(str(var[3]))
                self.InvertTilt_checkbox.setChecked(var[4])
                if (var[8]):
                    self.MinPanlineEdit.setText(str(var[6]))
                    self.MaxPanlineEdit.setText(str(var[5]))
                else:
                    self.MinPanlineEdit.setText(str(var[5]))
                    self.MaxPanlineEdit.setText(str(var[6]))
                self.PanSensivityEdit.setText(str(var[7]))
                self.InvertPan_checkbox.setChecked(var[8])
                # self.CameraIDEdit.setText(str(var[9]))
                self.LED_checkbox.setChecked(var[10])
            print(var)
            # Set variables
        except:
            pass

    def save_init_file(self):
        init_settings = [self.COMlineEdit.text(),
        self.min_tilt, self.max_tilt, self.TiltSensivity, self.InvertTilt,
        self.min_pan, self.max_pan, self.PanSensivity, self.InvertPan,
        self.CameraID, self.LED_ON]
        with open('init.pkl', 'wb') as init_file:
            pickle.dump(init_settings, init_file)
    def connect(self):    # Set COM port from text box if arduino not already connected
        if(not self.is_connected):
            if (self.ard.connect()):    #set port label message
                self.COMConnectLabel.setText("..................... Connected to : " + self.ard.url +
" ......................")
                self.is_connected = True
            else:
                self.COMConnectLabel.setText(".................... Cant connect to : " + self.ard.url
+ " ......................")
    def update_angles(self):  # Update variables from text boxes
        try:
            self.InvertTilt = self.InvertTilt_checkbox.isChecked()
            self.InvertPan = self.InvertPan_checkbox.isChecked()
            self.TiltSensivity = float(self.TiltSensivityEdit.text())
            self.PanSensivity = float(self.PanSensivityEdit.text())
            self.LED_ON = self.LED_checkbox.isChecked()
            self.cap.release()# Camera needs to be released to update the camera ID (if changed)
            self.CameraID = int(self.CameraIDEdit.text())
            self.cap = cv2.VideoCapture(self.CameraID)
            if(self.InvertPan):
                self.max_pan = int(self.MinPanlineEdit.text())
                self.min_pan = int(self.MaxPanlineEdit.text())
            else:
                self.min_pan = int(self.MinPanlineEdit.text())
                self.max_pan = int(self.MaxPanlineEdit.text())

            if(self.InvertTilt):
                self.max_tilt = int(self.MinTiltlineEdit.text())
                self.min_tilt = int(self.MaxTiltlineEdit.text())
            else:
                self.min_tilt = int(self.MinTiltlineEdit.text())
                self.max_tilt = int(self.MaxTiltlineEdit.text())
            self.save_init_file()
            print("Values updated")
        except:
            print("Can't update values")

    def mouseMoveEvent(self, event):
        # The position of the mouse is tracked and converted to a pan and tilt amount
        # For example if mouse completely to the left-> pan_target = 0(or whatever minimum pan_target 
value is)
        # If completely to the right-> pan_target = 180(or whatever maximum pan_target value is)
        # Same principal for tilt
        if(self.manual_mode): # If manual mode selected
            if(35<event.y()<470 and 70<event.x()<910):
                if(self.InvertTilt):
                    self.target_tilt = opr.remap(event.y(), self.max_tilt, self.min_tilt, 470, 35)
                    # (470,35 allows the mouse to be tracked only over the image)
                else:
                    self.target_tilt =  opr.remap(event.y(), self.min_tilt, self.max_tilt, 35, 470)

                if (self.InvertPan):
                    self.target_pan = opr.remap(event.x(), self.max_pan, self.min_pan, 910, 70)
                else:
                    self.target_pan = opr.remap(event.x(), self.min_pan, self.max_pan, 70, 910)

    def update_LCD_display(self):
        # Update servo angle values sent by the arduino
        # Actually pretty useless since the arduino return its target value and not its actual real 
world position
        self.Pan_LCD.display(self.current_pan)
        self.Tilt_LCD.display(self.current_tilt)

    def quit(self):
        print('Quit')
        self.rec = False
        sys.exit()

    def closeEvent(self, event):
        self.quit() # Call quit method when cross pressed

    def set_manual_mode(self):
        self.manual_mode = self.Manual_checkbox.isChecked()
        if(not self.manual_mode):          # If not in manual mode
            self.random_servos_position()  # Select a random pan and tilt target
        print(self.manual_mode)

    def random_servos_position(self):
        self.target_tilt = random.uniform(self.min_tilt, self.max_tilt)
        self.target_pan = random.uniform(self.min_pan, self.max_pan)

    def toggle_recording(self):
        if(self.rec):
            self.rec = False                   # Stop recording
            self.PauseButton.setText("Resume") # Change pause button text
        else:
            self.rec = True
            self.PauseButton.setText("Pause")
            self.record()
    def record(self):  # Video recording
        while(self.rec):
            ret, img = self.cap.read() # CAPTURE IMAGE
            if(self.is_connected):            # If arduino connected
                if(self.manual_mode):         # And manual mode on
                    processed_img = img       # Don't process image
                else:
                    processed_img = self.image_process(img
            else:                             # If arduino  not connected
                processed_img = img           # Don't process image

            self.update_GUI(processed_img)    # Update image in window
            cv2.waitKey(0)                    # No delay between frames

            self.move_servos() # Move servos

            if (not self.rec):     # Allows while loop to stop if pause button pressed
                break

    def update_GUI(self, openCV_img): # Convert OpenCV image and update PyQt label
        try: # If this doesn't work check camera ID
            openCV_img = cv2.resize(openCV_img, (960, 540))  # This is stretching the image a bit but
if not done it won't fit the UI
            height, width, channel = openCV_img.shape
            bytesPerLine = 3 * width
            qImg = QImage(openCV_img.data, width, height, bytesPerLine,
QImage.Format_RGB888).rgbSwapped()

            pixmap = QPixmap(qImg)
            self.label.setPixmap(pixmap)

        except:
            self.label.setText("check camera ID")

        self.show()

    def move_servos(self):
        if (self.is_connected):

            if self.LED_ON and not self.manual_mode:
                if not self.face_detected: # Set led mode (0:red, 1:yellow 2:green)
                    led_mode = 0
                else:
                    if self.target_locked:
                        led_mode = 1
                    else :
                        led_mode = 2

            elif self.LED_ON and self.manual_mode:
                led_mode = 3 #turn all led's on
            else:
                led_mode = 4 #turn led's off

            to_send = {
                "target_pan": float(self.target_pan),
                "target_tilt": float(self.target_tilt),
                "led_mode": led_mode
            }

            self.ard.sendData(to_send)

            # data_to_send = "<" + str(int(self.target_pan)) + "," + str(int(self.target_tilt)) + ","
+ str(led_mode) + ">"
            # self.ard.runTest(data_to_send)

            # The data sent to the arduino will look something like the this (<154, 23, 0>)
            # The arduino will look for the start character "<"
            # Then save everything following until it finds the end character ">"
            # At that point the arduino will have saved a message looking like this "154, 23, 0"
            # It will then split the message at every coma and use the pieces of data to move the
servos acordingly
```

```
    def roam(self):
        if(self.roam_pause_count < 0 ):      # If roam count inferior to 0
            self.roam_pause_count = self.roam_pause             # Reset roam count
            self.roam_target_pan = int(random.uniform(self.min_pan, self.max_pan))
            self.roam_target_tilt = int(random.uniform(self.min_tilt, self.max_tilt))
        else:        # If roam count > 1
                     # Increment pan target toward roam target
            if (int(self.target_pan) > self.roam_target_pan):
                self.target_pan -= 1
            elif (int(self.target_pan) < self.roam_target_pan):
                self.target_pan += 1
            else:    # If roam target reached decrease roam pause count
                self.roam_pause_count -= 1

            if (int(self.target_tilt) > self.roam_target_tilt):
                self.target_tilt -= 1
            elif (int(self.target_tilt) < self.roam_target_tilt):
                self.target_tilt += 1
            else:
                self.roam_pause_count -= 1

    def image_process(self, img):  # Handle the image processing
        processed_img = opr.find_face(img, self.max_target_distance)

        if(processed_img[0]):              # If face found
            self.face_detected = True
            self.empty_frame_number = self.max_empty_frame  # Reset empty frame count
            self.target_locked = processed_img[4]
            self.calculate_camera_move(processed_img[2], processed_img[3])
            return processed_img[1]
        else:
            self.face_detected = False
            self.target_locked = False
            if(self.empty_frame_number> 0):
                self.empty_frame_number -= 1  #Decrease frame count until it equal 0
            else:
                self.roam()              # Then roam
            return img

    def calculate_camera_move(self, distance_X, distance_Y):
        #self.target_pan += distance_X * self.PanSensivity
        if(self.InvertPan): # Handle inverted pan
            self.target_pan -= distance_X * self.PanSensivity
            if(self.target_pan>self.min_pan):
                self.target_pan = self.min_pan
            elif (self.target_pan < self.max_pan):
                self.target_pan = self.max_pan

        else:
            self.target_pan += distance_X * self.PanSensivity
            if(self.target_pan>self.max_pan):
                self.target_pan = self.max_pan
            elif (self.target_pan < self.min_pan):
                self.target_pan = self.min_pan
        #self.target_tilt += distance_Y * self.TiltSensivity
        if(self.InvertTilt): #handle inverted tilt
            self.target_tilt -= distance_Y * self.TiltSensivity
            if(self.target_tilt>self.min_tilt):
                self.target_tilt = self.min_tilt
            elif (self.target_tilt < self.max_tilt):
                self.target_tilt = self.max_tilt
        else:
            self.target_tilt += distance_Y * self.TiltSensivity
            if(self.target_tilt>self.max_tilt):
                self.target_tilt = self.max_tilt
            elif (self.target_tilt < self.min_tilt):
                self.target_tilt = self.min_tilt

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = App()
    app.exec_()
```

7.2.4 Arduino Code (DRK_1_Arduino_Code.ino)

```
#include <SPI.h>
#include <Servo.h>
#include <WiFiNINA.h>
#include <ArduinoJson.h>
#include <ArduinoHttpClient.h>
#include <Adafruit_NeoPixel.h>

// Melody Importing
#define tone_one  1300   // Frequency 1263Hz
#define tone_two  1350   // Frequency 1350Hz
#define tone_three  1523   // Frequency 1523Hz
int melody[] = {tone_one, tone_two, tone_three};
int noteDurations[] = {18, 18, 3};

char ssid[] = "JAI_BALAYYA";
char pass[] = "48GOODWOOD";
int status = WL_IDLE_STATUS;     // the Wifi radio's status

char serverAddress[] = "192.168.0.17"; // server address
int port = 8000;

WiFiClient wifi;
HttpClient client = HttpClient(wifi, serverAddress, port);

StaticJsonDocument<200> doc;

Servo panServo;
Servo tiltServo;

// Neopixel Setup---------------------------------------------------------------------------------------------
Adafruit_NeoPixel pixels = Adafruit_NeoPixel(8, 11, NEO_GRB + NEO_KHZ800);
class Strip {
public:
  uint8_t   effect;
  uint8_t   effects;
  uint16_t  effStep;
  unsigned long effStart;
  Adafruit_NeoPixel strip;
  Strip(uint16_t leds, uint8_t pin, uint8_t toteffects, uint16_t striptype) : strip(leds, pin, striptype) {
    effect = -1;
    effects = toteffects;
    Reset();
  }
  void Reset(){
    effStep = 0;
    effect = (effect + 1) % effects;
    effStart = millis();
  }
};

struct Loop {
  uint8_t currentChild;
  uint8_t childs;
  bool timeBased;
  uint16_t cycles;
  uint16_t currentTime;
  Loop(uint8_t totchilds, bool timebased, uint16_t tottime) {
    currentTime=0;
    currentChild=0;
    childs=totchilds;
    timeBased=timebased;
    cycles=tottime;
    }
};

Strip strip_0(8, 11, 8, NEO_GRB + NEO_KHZ800);
struct Loop strip0loop0(1, false, 1);
```

```
//---------------------------------------------------------------------------------------------------------------
const byte buffSize = 40;
char inputBuffer[buffSize];
const char startMarker = '<';
const char endMarker = '>';
byte bytesRecvd = 0;
boolean readInProgress = false;
boolean newDataFromPC = false;

float panServoAngle = 90.0;
float tiltServoAngle = 90.0;
// int LED_state = 11;

String url = "http://192.168.1.168:8000/";

void setup()
{
  Serial.begin(9600);
  panServo.attach(10);
  tiltServo.attach(9);

  strip_0.strip.begin();

  // moveServo();
  start_sequence();

  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB port only
  }

  // check for the presence of the shield:
  if (WiFi.status() == WL_NO_SHIELD) {
    Serial.println("WiFi shield not present");
    // don't continue:
    while (true);
  }

  String fv = WiFi.firmwareVersion();
  if (fv != "1.1.0") {
    Serial.println("Please upgrade the firmware");
  }

  // attempt to connect to Wifi network:
  while (status != WL_CONNECTED) {
    Serial.print("Attempting to connect to WPA SSID: ");
    Serial.println(ssid);
    // Connect to WPA/WPA2 network:
    status = WiFi.begin(ssid, pass);

    // wait 10 seconds for connection:
    delay(10000);
  }

  // you're connected now, so print out the data:
  Serial.print("You're connected to the network");
  printCurrentNet();
  printWifiData();
}

void loop()
{
    strips_loop_rainbow();
    getDataFromPC();
    replyToPC();
    moveServo();
    setLED();
}
```

```
void getDataFromPC()
{
   // receive data from PC and save it into inputBuffer

   if (wifi.status() == WL_CONNECTED)
   {
      client.get("/");

      // read the status code and body of the response
      int statusCode = client.responseStatusCode();
      String json = client.responseBody();

      if (statusCode == 200)
      {
         DeserializationError error = deserializeJson(doc, json);
         //Serial.println(data);

         if (error)
         {
            Serial.print(F("deserializeJson() failed: "));
            Serial.println(error.f_str());
            return;
         }

         panServoAngle = doc["target_pan"];
         panServoAngle = doc["target_tilt"];
         LED_state = doc["led_mode"];
      }
      else
      {
         Serial.println("Error: " + statusCode);
      }
   }
}

void processData() // for data type "<float, float, int>"
{
   char *strtokIndx; // this is used by strtok() as an index

   strtokIndx = strtok(inputBuffer, ","); // get the first part
   panServoAngle = atof(strtokIndx);      // convert this part to a float

   strtokIndx = strtok(NULL, ",");   // get the second part(this continues where the previous call left off)
   tiltServoAngle = atof(strtokIndx); // convert this part to a float

   strtokIndx = strtok(NULL, ","); // get the last part
   LED_state = atoi(strtokIndx);   // convert this part to an integer (string to int)
}
void replyToPC()
{

   if (newDataFromPC)
   {
      newDataFromPC = false;
      Serial.print("<");
      Serial.print(panServo.read());
      Serial.print(",");
      Serial.print(tiltServo.read());
      Serial.println(">");
   }
}

void moveServo()
{
   panServo.write(panServoAngle);
   tiltServo.write(tiltServoAngle);
}
```

```
// -----------------------------------------RAINBOW-------------------------------------------------------------------------
void strips_loop_rainbow() {
  if(strip0_loop0() & 0x01)
    strip_0.strip.show();
}

uint8_t strip0_loop0() {
  uint8_t ret = 0x00;
  switch(strip0loop0.currentChild) {
    case 0:
           ret = strip0_loop0_eff0();break;
  }
  if(ret & 0x02) {
    ret &= 0xfd;
    if(strip0loop0.currentChild + 1 >= strip0loop0.childs) {
      strip0loop0.currentChild = 0;
      if(++strip0loop0.currentTime >= strip0loop0.cycles) {strip0loop0.currentTime = 0; ret |= 0x02;}
    }
    else {
      strip0loop0.currentChild++;
    }
  };
  return ret;
}

uint8_t strip0_loop0_eff0() {
    // Strip ID: 0 - Effect: Rainbow - LEDS: 8
    // Steps: 103 - Delay: 52
    // Colors: 3 (255.0.0, 0.255.0, 0.0.255)
    // Options: rainbowlen=168, toLeft=true,
  if(millis() - strip_0.effStart < 52 * (strip_0.effStep)) return 0x00;
  float factor1, factor2;
  uint16_t ind;
  for(uint16_t j=0;j<8;j++) {
    ind = strip_0.effStep + j * 0.6130952380952381;
    switch((int)((ind % 103) / 34.333333333333336)) {
      case 0: factor1 = 1.0 - ((float)(ind % 103 - 0 * 34.333333333333336) / 34.333333333333336);
              factor2 = (float)((int)(ind - 0) % 103) / 34.333333333333336;
              strip_0.strip.setPixelColor(j, 255 * factor1 + 0 * factor2, 0 * factor1 + 255 * factor2, 0 * factor1 + 0 * factor2);
              break;
      case 1: factor1 = 1.0 - ((float)(ind % 103 - 1 * 34.333333333333336) / 34.333333333333336);
              factor2 = (float)((int)(ind - 34.333333333333336) % 103) / 34.333333333333336;
              strip_0.strip.setPixelColor(j, 0 * factor1 + 0 * factor2, 255 * factor1 + 0 * factor2, 0 * factor1 + 255 * factor2);
              break;
      case 2: factor1 = 1.0 - ((float)(ind % 103 - 2 * 34.333333333333336) / 34.333333333333336);
              factor2 = (float)((int)(ind - 68.66666666666667) % 103) / 34.333333333333336;
              strip_0.strip.setPixelColor(j, 0 * factor1 + 255 * factor2, 0 * factor1 + 0 * factor2, 255 * factor1 + 0 * factor2);
              break;
    }
  }
  if(strip_0.effStep >= 103) {strip_0.Reset(); return 0x03; }
  else strip_0.effStep++;
  return 0x01;
}

// ---------------------------------------------------------------------------------------------------------------
void setLED() {
    if (LED_state == 2) { // Yellow ON

    }
    else if (LED_state == 1) { // Face Found, Green ON

    }
    else if (LED_state == 0) { // No Face, Red ON

    }
    else if (LED_state == 3) { // All ON
        strips_loop_rainbow();
    }
```

```
else { // All OFF

    }
}

// -------------------------------------------------------------------------------------------------------------------
void startup_sound() {
  for (int thisNote = 0; thisNote < 3; thisNote++) {
    int noteDuration = 1000 / noteDurations[thisNote];  // Play note
    tone(8, melody[thisNote], noteDuration*0.8);

    int pauseBetweenNotes = noteDuration * 1.30;  // Pause note
    delay(pauseBetweenNotes);

    noTone(8); // Stop sound
  }
}

void printWifiData() {
  // print your WiFi shield's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);
  Serial.println(ip);

  // print your MAC address:
  byte mac[6];
  WiFi.macAddress(mac);
  Serial.print("MAC address: ");
  Serial.print(mac[5], HEX);
  Serial.print(":");
  Serial.print(mac[4], HEX);
  Serial.print(":");
  Serial.print(mac[3], HEX);
  Serial.print(":");
  Serial.print(mac[2], HEX);
  Serial.print(":");
  Serial.print(mac[1], HEX);
  Serial.print(":");
  Serial.println(mac[0], HEX);

}

void printCurrentNet() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print the MAC address of the router you're attached to:
  byte bssid[6];
  WiFi.BSSID(bssid);
  Serial.print("BSSID: ");
  Serial.print(bssid[5], HEX);
  Serial.print(":");
  Serial.print(bssid[4], HEX);
  Serial.print(":");
  Serial.print(bssid[3], HEX);
  Serial.print(":");
  Serial.print(bssid[2], HEX);
  Serial.print(":");
  Serial.print(bssid[1], HEX);
  Serial.print(":");
  Serial.println(bssid[0], HEX);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI):");
  Serial.println(rssi);
  // print the encryption type:
  byte encryption = WiFi.encryptionType();
  Serial.print("Encryption Type:");
  Serial.println(encryption, HEX);
  Serial.println();
}

// LED

void start_sequence()
{
    startup_sound();
    for(int i=8; i>=0; i--){
      pixels.setPixelColor(i, pixels.Color(0,255,35));
      pixels.show();
      delay(100);
    }
    panServo.write(90);
    tiltServo.write(90);
    delay(300);
}
```