ADJUSTABLE SURVEILLANCE CAMERA ROBOT

Final Year Project Report

By

Muhammad Hamza (173088)

Abdul Hadi Zahid (174442)

In Partial Fulfillment

Of the Requirements for the degree

Bachelors of Electrical Engineering (BEE)

School of Electrical Engineering and Computer Science

National University of Sciences and Technology

Islamabad, Pakistan

(2020)

DECLARATION

We hereby declare that this project report entitled "**ADJUSTABLE SURVEILLANCE CAMERA ROBOT**" submitted to the "Department of Electrical Engineering", is a record of an original work done by us under the guidance of Supervisor "**DR. FARID GUL**" and Co-Supervisor "**Engr. Neelma Naz**" and that no part has been plagiarized without citations. Also, this project work is submitted in the partial fulfillment of the requirements for the degree of Bachelor of Electrical Engineering.

Team Members:

Muhammad Hamza (173088)

Abdul Hadi Zahid (174442)

Supervisor: Dr. Farid Gul

Co-Supervisor: Miss Neelma Naz

Date:

15th June. 2020

DEDICATION

To Allah the Almighty,

To our Parents and Faculty

&

To Everyone who we met along the way

ACKNOWLEDGMENTS

We are immensely grateful to our respected advisor Dr. Farid Gul and co-advisor Miss Neelma Naz for guiding us throughout in our final year project. Your guidance, skills, advice, motivation and support enabled us in achieving our goals.

We greatly appreciate everyone else who was involved in this project which includes but not limited to parents, faculty, friends who helped us in making us succeed. We are also very grateful for the staff of SMME Workshop for helping us in the construction of our robot frame.

Table of Contents

List of Figures

Abstract

Throughout the world, most of the industries have designated health-hazardous areas where there is a risk or danger to life. This may be a radioactive chamber in a nuclear plant or a gas pipeline in a fertilizer plant or a confined-space in some chamber. Constant surveillance for that area is not necessary but occasionally humans have to visit that place and for that in order to survive, they need special gear and equipment.

This project presents the study and design of a Surveillance Robot. A user can remotely control this robotic vehicle and navigate it to the required location. The robot will have a camera mounted on it which will provide live streaming to the user with the ability to have a couple of degrees of freedom including circle azimuth motion, vertical motion and tilt motion. The robot will also be equipped with multiple environment sensors which will provide real-time data to the user of the conditions of the hazardous area.

Chapter 1

Introduction

With the beginning of industrialization throughout the world from the beginning of the 20th century, humans have been excessively creating machines to cater for their needs. These machines in industries in order to improve efficiency and productions use many different kinds of materials which are toxic and hazardous to human health. Constant maintenance and repair are part of the machines and even though many of this process may be automated, they still require human insights. With the ever-increasing industries, bio-hazards areas are also forming at an alarming rate and they too require occasional checkup. Maintaining a constant surveillance system using cameras sometimes is too expensive and requires an extensive network.

Another application of this remote-surveillance project is in security fields. This robot can be used by security agencies to look up threats and analyze the structure of a field or a building at a safe distance. Alternatively, this can also be used at home to keep an eye on young children's or pets and in general the whole house to ensure no one enters without you knowing about it.

This project will cater to this problem with a robotic vehicle which a technical user can access remotely in a hazard-free zone. User can control the vehicle movement in the hazardous area easily. The robot will be providing a constant live video stream using camera mounted on it to the user interface. In order to provide better viewing angles to the user, additional platforms are also made onto this vehicle which can be controlled by the user. First there is a rotatory platform which provides a 360° view of the surrounding. Then a vertical elevation system is implemented on top of that platform which will help change the degrees of vision the robot can stream to user. Then in the end a separate mechanism is placed onto the top which will provide tilt motion so user can look from straight ahead to exactly 90° above or below and all the angles in between.

Multiple environmental data collection sensors including (temperature, humidity, CO2, CH4, CO concentration sensors etc.) are also mounted onto the robot which provides the user a live stream of the data of the surroundings of the robot.

Even though user can control and navigate the robot remotely, multiple safety measures are also implemented on the robot which includes edge detection and object detection which actively makes sure it does not accidently hit the robot with nearby surroundings and getting damaged..

Motivation:

Every industry throughout the world is trying to reduce work-related injuries and fatalities but many of the time accidents happen because of some very simple reason which could easily have been avoided. These incidents are many a time due to improper safety regulations and standards or their implementations. Robots have been making our lives simpler for the past few decades and cost of a human life far outweighs the cost of robot. Our robot being very inexpensive further enlarges the gaps and provides a way for humans to keep away from these areas as much as possible. During routine inspections etc., this robot can easily be utilized to collect data and observe the habitat without putting human life in danger.

Objectives:

The objectives of the project were modified and added throughout the year with the consult of supervisors but the core ideas remain the same

- To develop mathematical models of the systems that govern the different motions of the robot
- To create a user-friendly web-based interface for technicians to use remotely and monitor the data
- To create an enclosed four-wheeled robotic chassis that can navigate all sorts of terrains that may be required in an industry
- To create the platforms that can provide the rotatory, vertical and title motion to the live-streaming camera as multiple degrees of freedom.

Chapter 2

Literature Review

This project has been around as a DIY build experiment across the internet and universities from a long time with different scopes especially with the arrival of the first Raspberry Pi micro-computer with camera ports. But most of the projects were constrained with their degrees of freedoms or types of data outputs that they could provide. We are going to mention a few projects that have been done that resembles ours.

S.A Joshi, Aparna Tondarkar, Krishna Solanke, Rohit Jagtap [1] created a surveillance robot for military applications. Their prototype used a Raspberry Pi 3B, Servo/DC Motors for tilt and chassis movement, Ultrasonic sensor for object detection and camera. User can control the robot remotely via internet on a web-based interface.

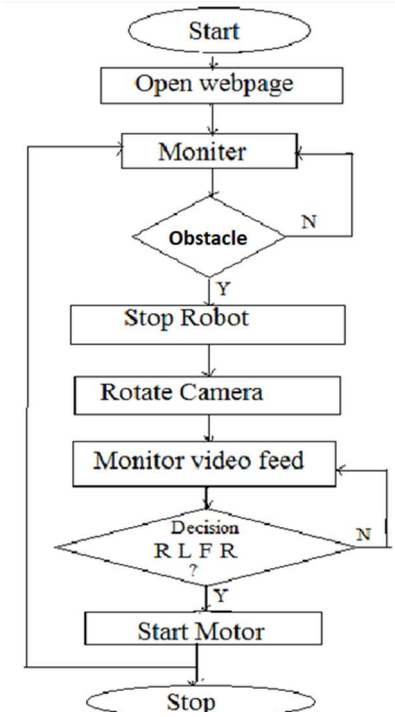

Figure 2.1 - Joshi, Tondarkar, Solanke, Jagtap's Robot Flowchart

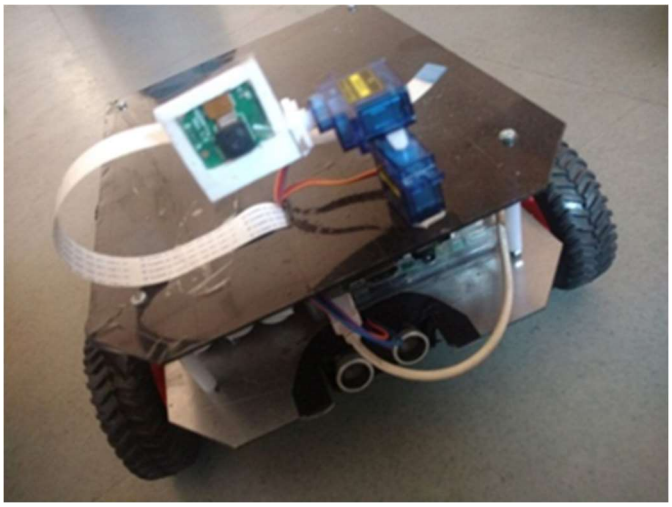

Figure 2.2 - Joshi, Tondarkar, Solanke, Jagtap's Robot

The web interface provided some functionalities which included the motion of the robot in all the four directions and the tilt motion of the camera with some very basic controls for the movement of the robot.

Neha Joisher, Sonal Jain, Tejas Malviya, and Vaishal Gaikwad [2] also implemented this surveillance robot but with even more reduced capacity then the earlier one we mentioned. They also used a Raspberry Pi 3B, DC motors with drivers and a camera. Web-interface connect the car to the user online. User can control all of the four motions of the car but not of the camera and there is no live data coming from the robot about the environment.

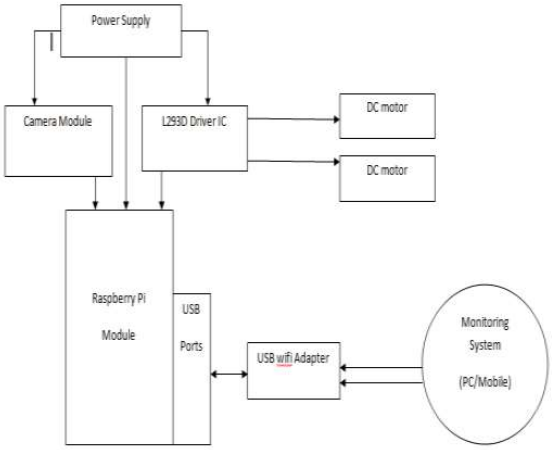

Figure 2.3 - Joshier, Jain, Malviya, Gaikwad Robot Flowchart

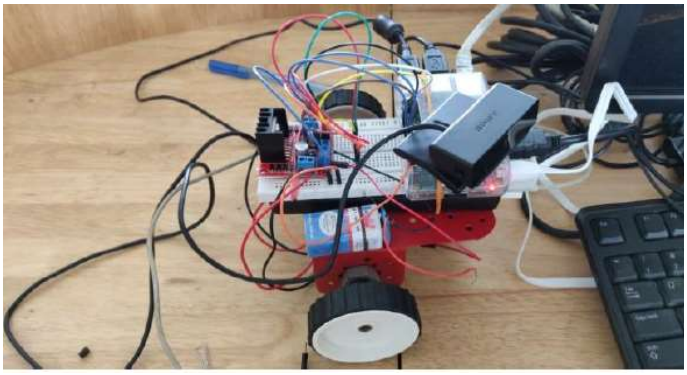

Figure 2.4 - Joshier, Jain, Malviya, Gaikwad Robot

Saddam [3] also made a very similar robot like that of Neha, Sonal, Tejas, Vaishal [2] with nearly all of the same features i.e. user can control the motion of the robot remotely on a web-interface. It uses nearly all the identical components as to that of Neha, Sonal, Tejas, Vaishal [2] robot.

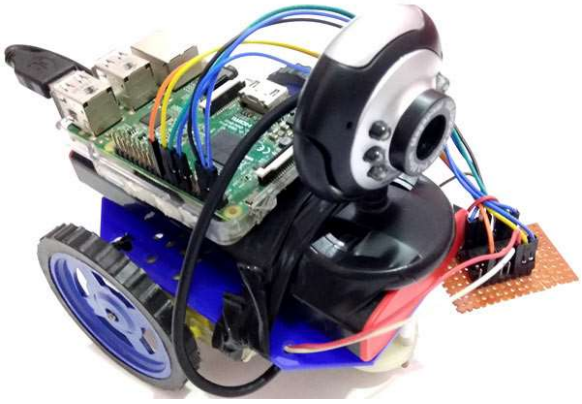

Figure 2.5 - Saddam Robot

Chapter 3

Detailed System Design and Architecture

Design Methodology

Moving forward we will start our discussion with the overall architecture of the robot. Then we will move onto the theoretical aspect of the robot i.e. mathematically modelling our robot and its components. We will design the robot in Simulink in MATLAB and see how it reacts to different inputs. Then we will move onto electrical hardware portion of the project where we will discuss all the components used, their working and their interfacing with each other. Then we will work onwards and design the mechanical hardware portion where we will design each of the platforms for different kinds of motion and then assemble the whole robot in one final design. Then we move forward to the software portion where we describe how the components are coded and how the user-interface is created and how this all merges both the electrical and mechanical portion into one final product.

System Overview

The major part of this project revolved around the different kinds of motions that this robot can provide and for that we needed to figure out the materials and parts that we would be using. The robot has 4 distinct types of independent motions each and requires their own special components. Then they also need a brain i.e. a micro-controller which can coordinate and control all the movements whose input would be provided by the user using a micro-computer.

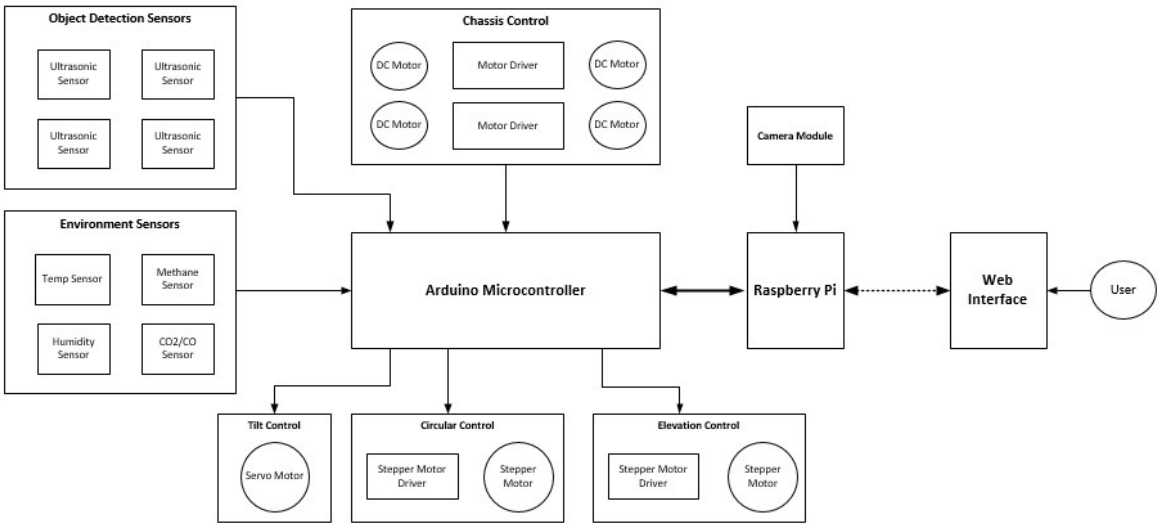

Figure 3.1 System Architecture Overview

The first task is to model the main parts of the system mathematically. These main parts are the three types of motors which controls all the motions of the robot. To model this, we first use the known mathematical techniques of control systems and then implement these in Simulink, MATLAB to test it out.

Mathematical Modelling of Motors

Diving deeply into the robot characteristics, it is revealed that it comprises of four separate open-looped control systems which are completely independent from each other. The first part is the chassis with its DC Motors, then comes the second's rotatory motion and third's elevation motion using exactly the same Stepper Motor system but completely independent to one another. The fourth and last is the tilt motion, made possible using servo motor which too is independent. So, in this topic, we discuss all of the three types of motors, mathematically model them and then simulate them in Simulink.

- **DC Motor**

Consider steady-state model for armature-controlled DC motor

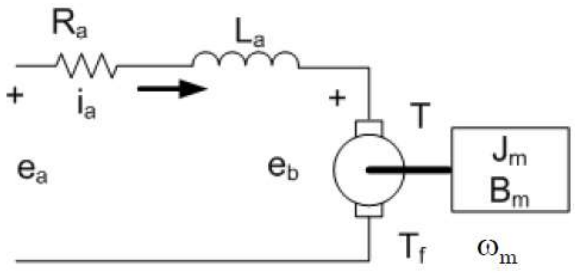

Figure 3.2 DC Motor Circuit Diagram

Here i_a = armature current, e_b = back-emf, e_a= armature terminal voltage, w_m = motor speed (rad/sec), T = motor torque, T_f = static friction torque, R_a = armature resistance, L_a = armature inductance, J_m = rotational inertia, B_m = viscous friction

Consider $e_a(t)$ and $e_b(t)$ as inputs and $i_a(t)$ as output. Applying KVL we get,

$$e_a(t) = R_a i_a(t) + L_a \frac{di_a(t)}{dt} + e_b(t)$$

Torque of DC motor is proportional to the product of flux and current. Since flux is constant in this system, the torque is proportional to $i_a(t)$ alone.

$$T \propto i_a$$

$$T = K_t i_a$$

The differential equation governing the mechanical system of motor is given by

$$J_m \frac{d^2\theta}{dt^2} + B_m \frac{d\theta}{dt} = T$$

The back emf of DC machine is proportional to speed (angular velocity) of shaft.

$$e_b \propto \frac{d\theta}{dt}$$

$$e_b = K_b \frac{d\theta}{dt}$$

Taking Laplace transformation of the above equations with zero initial conditions we get,

$$E_a(s) = R_a I_a(s) + sL_a I_a(s) + E_b(s)$$

$$T(s) = K_t I_a(s)$$

$$s^2 J_m \theta(s) + sB_m \theta(s) = T(s)$$

$$E_b(s) = sK_b \theta(s)$$

Equating above equations we get,

$$K_t I_a(s) = s^2 J_m \theta(s) + sB_m \theta(s)$$

$$I_a(s) = \frac{(s^2 J_m \theta(s) + sB_m \theta(s))}{K_t}$$

Thus, differential equation

$$(R_a + sL_a) I_a(s) + E_b(s) = E_a(s)$$

Becomes

$$(R_a + sL_a) \frac{(s^2 J_m + sB_m)}{K_t} \theta(s) + sK_b \theta(s) = E_a(s)$$

$$\frac{(R_a + sL_a)(s^2 J_m + sB_m) + sK_b K_t}{K_t} \theta(s) = E_a(s)$$

The required transfer function of armature-controlled dc motor is

$$\frac{\theta(s)}{E_a(s)} = \frac{K_t}{(R_a + sL_a)(s^2 J_m + sB_m) + sK_b K_t}$$

$$\frac{\theta(s)}{E_a(s)} = \frac{K_t}{s[J_m L_a s^2 + (J_m R_a + B_m L_a)s + (B_m R_{a+} K_b K_t)]}$$

$$\frac{\theta(s)}{E_a(s)} = \frac{\frac{K_t}{J_m L_a}}{s\left[s^2 + \frac{(J_m R_a + B_m L_a)}{J_m L_a} s + \frac{(B_m R_{a+} K_b K_t)}{J_m L_a}\right]}$$

Thus, angular velocity would become

$$\frac{w(s)}{E_a(s)} = \frac{\frac{K_t}{J_m L_a}}{\left[s^2 + \frac{(J_m R_a + B_m L_a)}{J_m L_a} s + \frac{(B_m R_{a+} K_b K_t)}{J_m L_a}\right]}$$

The Simulink model of the DC-motor is

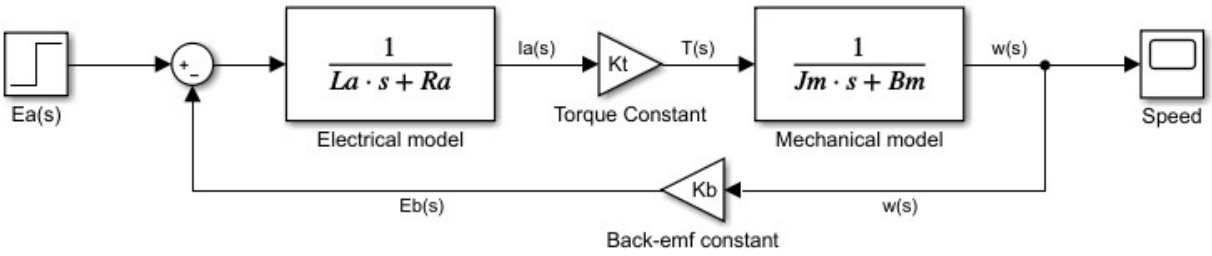

Figure 3.3 DC Motor Simulink Control Diagram

The motor has parameter values

$J_m = 0.02\ kgm^2/s^2, B_m = 0.1\ Nms, K_b = K_t = 0.1\ Nm/Amp, R_a = 2\ ohm, L_a = 0.5\ H$

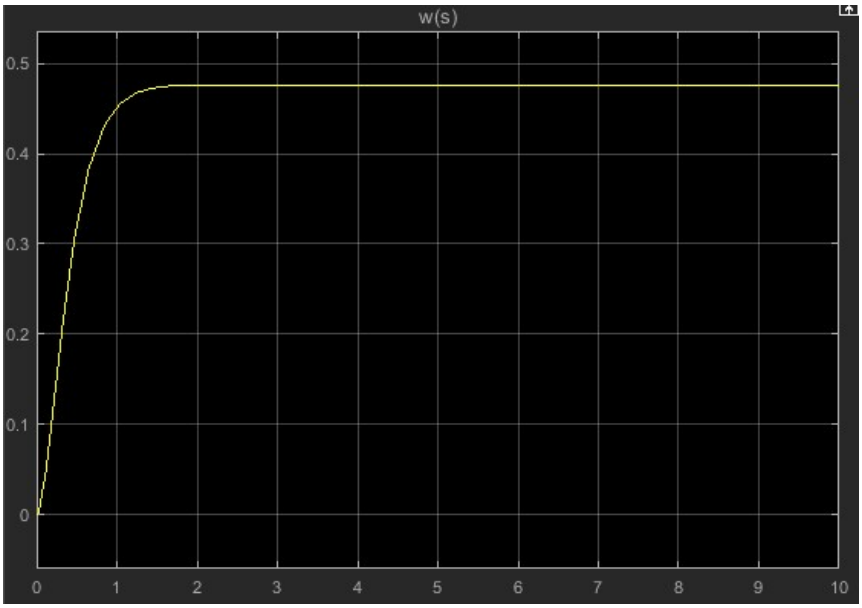

Figure 3.4 DC Motor Simulink Control Output between Angular Speed w ($\frac{rad}{s}$) w.r.t Time t(s)

From this graph we observe that because of the initial torque required to turn the motor from static position, it takes some time to reach its final value but it rises exponentially after that.

- **Servo Motor**

A servo motor can basically be divided into an error detector (potentiometer), an amplifier and a dc-motor all in a feedback loop as shown in the diagram. This system enters a reference angle θref signal and outputs an angle θ

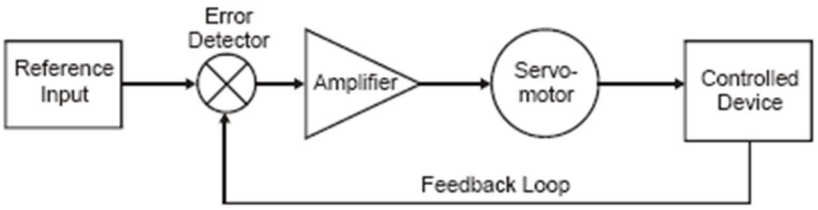

Figure 3.5 Servo Motor Control Loop

We have to find the transfer function of this system. The amplifier is simply a gain of magnitude K. We have already modeled the DC motor which we can simply add here with few modifications as follow

Now with the addition of gear box, we now have

$$J_m = J_a + \frac{{N_1}^2}{N_2} J_L$$

Where J_a= motor inertia, J_L =load inertia, and N_1 and N_2 are just the number of teeth's of input and output teeth

$$B_m = B_a + \frac{{N_1}^2}{N_2} B_L$$

Where B_a= motor damping constant, B_L= load damping constant, and N_1 and N_2 are just the number of teeth of the gear which will help us find the input-output gear-ratio.

Also because of the gears we have

$$\theta'(s) = \frac{N_1}{N_2}\,\theta(s)$$

where $\theta'(s)$ is the $\theta(s)$ in above transfer function, Thus

$$\frac{\theta'(s)}{E_a(s)} = \frac{\frac{K_t}{J_m L_a}.\frac{N_1}{N_2}}{s[s^2 + \frac{(J_m R_a + B_m L_a)}{J_m L_a} s + \frac{(B_m R_{a+} K_b K_t)}{J_m L_a}]}$$

The potentiometer at the end of the system which will convert the θ(s) into voltage e_{pot}(s) which will be fed to amplifier

$$E_{pot}(s) = K_{pot}\theta'(s)$$

where K_{pot} is the potentiometer constant.

The Simulink model of the Servo Motor System is

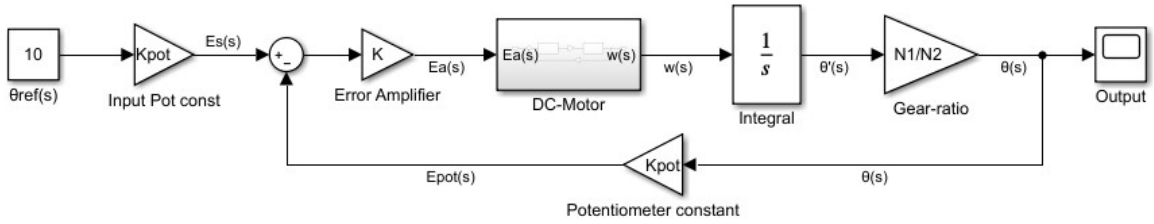

Figure 3.6 Servo Motor Simulink Control Diagram

Where the DC-motor module is

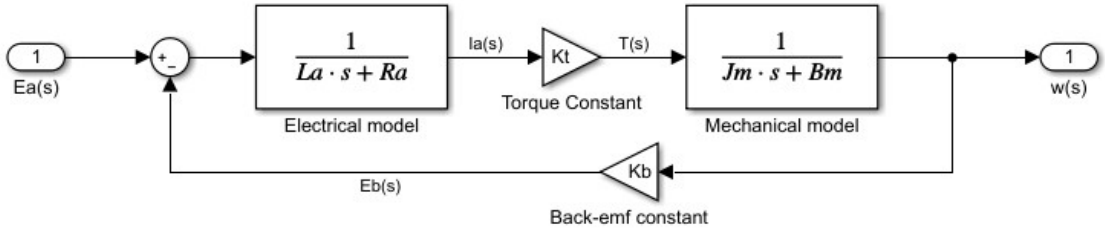

Figure 3.7 DC Motor in Servo Simulink Control Diagram

The given motor has parameter values

$J_m = 0.02\ kgm^2/s^2$, $B_m = 0.1\ Nms$, $K_b = K_t = 0.01\ Nm/Amp$, $R_a = 2\ ohm$, $L_a = 0.5\ H$, N1/N2 = 1/10, $K_{pot} = 5/\pi$ and K = 20

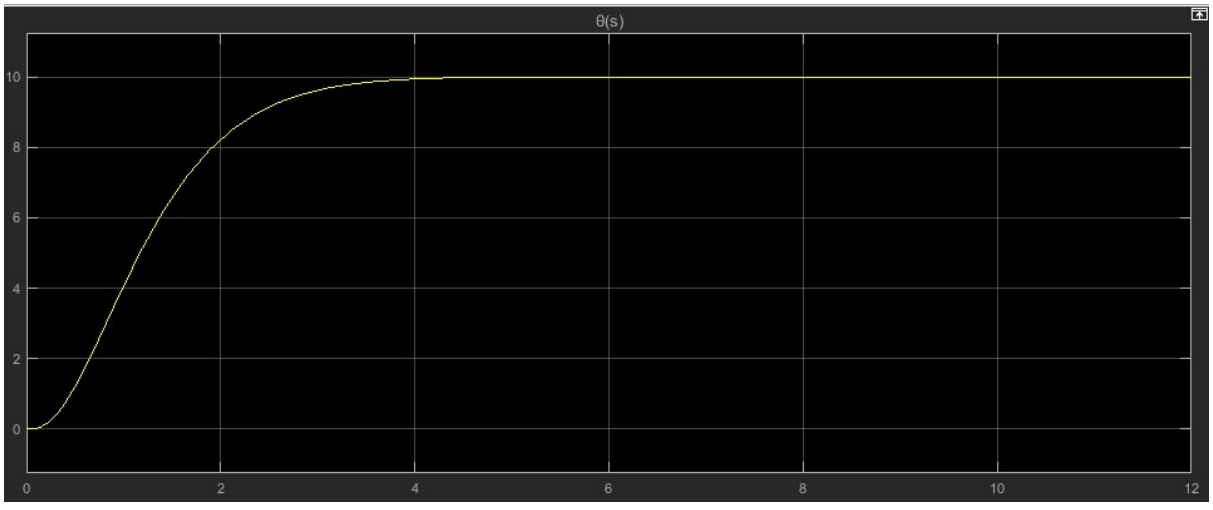

Figure 3.8 Servo Motor Simulink Control Output between Output Angle θ (deg) vs w.r.t Time t(s)

From the graph we observe that because of constant and rigorous feedback control, it takes a few seconds to reach its final value of the desired angle. The delay isn't that large that it would cause a catastrophic problem for out system so we will stick with this motor for now.

- **Stepper Motor**

*This modelling draws inspiration and works of **Ahmet, Malik, Peter [4]** and **Ricardo [5].***

The mathematical model of a stepper model could be divided into two sub-models, electrical model and mechanical model. We first start with the electrical model, where each phase of the stepper motor phases could be modelled as an RL circuit with a back electromotive force (emf).

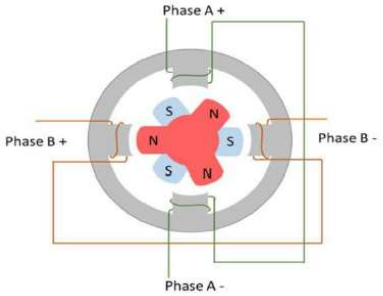

Figure 3.9 Stepper Motor Internal Wiring

The differential equations both the phase a and phase b are given by (1) and (2), respectively,

$$L_a \frac{di_a(t)}{dt} = -R_a i_a - e_a(t) + u_a(t) \qquad (1)$$

$$L_b \frac{di_b(t)}{dt} = -R_b i_b - e_b(t) + u_b(t) \qquad (2)$$

Where $e_a(t) = K_m w_m \sin(p\theta_m)$ and $e_b(t) = K_m w_m \cos(p\theta_m)$

In equation (1) and (2), the phase resistances and inductances are assumed to be equal, $R_a = R_b = R$ [Ω] and $L_a = L_b = L$ [H]. Moreover, $u_a(t)$ and $u_b(t)$ are the terminal voltages [V], $e_a(t)$ and $e_b(t)$ are the back emf [V], K_m is the motor constant, p is the number or motor poles pairs (teeth), w_m is the rotor (mechanical) angular speed [rad/s], and θ_m is the rotor (mechanical) angular position [rad].

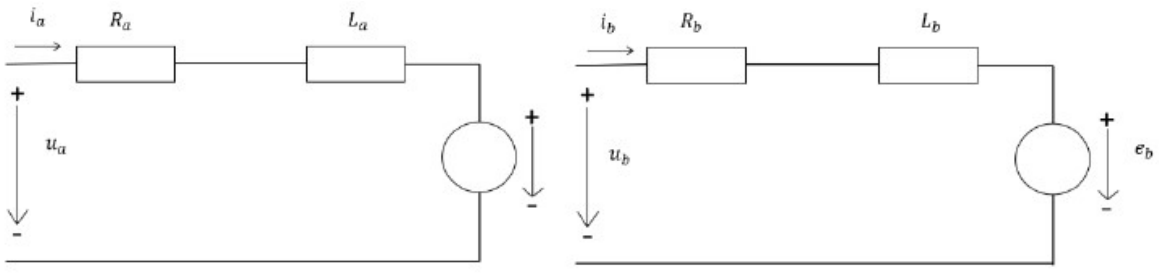

Figure 3.10 Stepper Motor Circuit Diagram

The shaft of the hybrid stepper motor, which represents the mechanical part of the system, is modelled as a rigid body subjected to different torques. The differential equation of the mechanical sub-system is given by (3),

$$J_m \frac{dw_m}{dt} = \tau_{em} - Bw_m - \tau_{dm} - \tau_l \qquad (3)$$

Where $\tau_{em} = K_m(-i_a \sin(p\theta_m) + i_b \cos(p\theta_m))$ and $\tau_{dm} = T_{dm} \sin(2p\theta_m + \alpha)$

In the mechanical model, J_m [kg.m^2] is the motor moment of inertia, B [kg/s m] is the motor viscous friction coefficient, τ_{em} [N.m] is the electromagnetic torque, T_{dm} [N.m] is the detent torque applied, α is the phase shift related to , and τ_l [N.m] is the applied load torque.

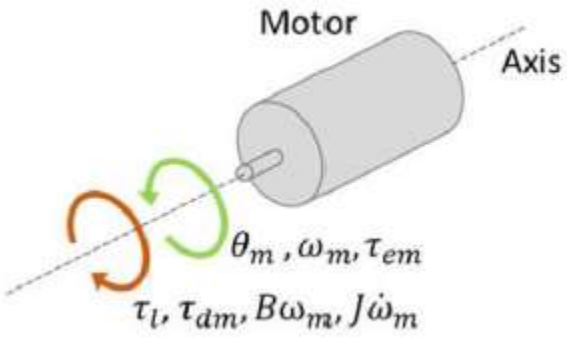

Figure 3.11 Stepper Motor Rotational Diagram

Use of FOC for Decoupling

In order to control position of the stepper motor, a concept of *Field Oriented Control (FOC)* is used. The main idea of FOC is to convert is to change electrical signals from one coordinate to other. The transformation is done to transform the reference frame from stator to rotor, to control the produced torque in the mechanical system. The main advantages of FOC are fast response and smooth operation. Moreover, this controlling method is used for stepper motors and synchronous motors. Park's transformation is used to transform the electrical model from a and b to d and q frame.

Applying Park's transformation to eq (1) and (2), we get,

$$L \frac{di_d}{dt} = -Ri_d - pLi_q w_m + u_d \qquad (4)$$

$$L \frac{di_q}{dt} = -Ri_q - pLi_d w_m + u_q + K_m w_m \qquad (5)$$

This transformation is commonly used in three-phase electric machine models, where it is known as a Park transformation. It allows you to eliminate time-varying inductances by referring the stator and rotor quantities to a fixed or rotating reference frame. In the case of a synchronous machine, the stator quantities are referred to the rotor. I_d *and* I_q *represent the two DC currents flowing in the two equivalent rotor windings (d winding directly on the same axis as the field winding, and q winding on the quadratic axis), producing the same flux as the stator.* I_a*,* I_b*, and* I_c *currents.*

As can be seen from (4) and (5) the currents $i_d$and $i_q$are coupled, [4] proposed a method to decouple them, the new electrical decoupled and linear system is given by (6) and (7).

$$L\frac{di_d}{dt} = -Ri_d + u_d^{lin} \qquad (6)$$

$$L\frac{di_q}{dt} = -Ri_q + u_q^{lin} \qquad (7)$$

In order to achieve this decoupling, the electrical drive system adds the quantities u_d^{dec} and u_q^{dec} to the desired u_d^{lin} and u_q^{lin}.

$$u_d^{dec} = -Lpw_m i_q \qquad (8)$$

$$u_q^{dec} = Lpw_m i_d + K_m w_m \qquad (9)$$

The block diagram of the decoupled system with the electrical drive subsystem, electrical subsystem in d and q frame, and the mechanical subsystem's Simulink model becomes

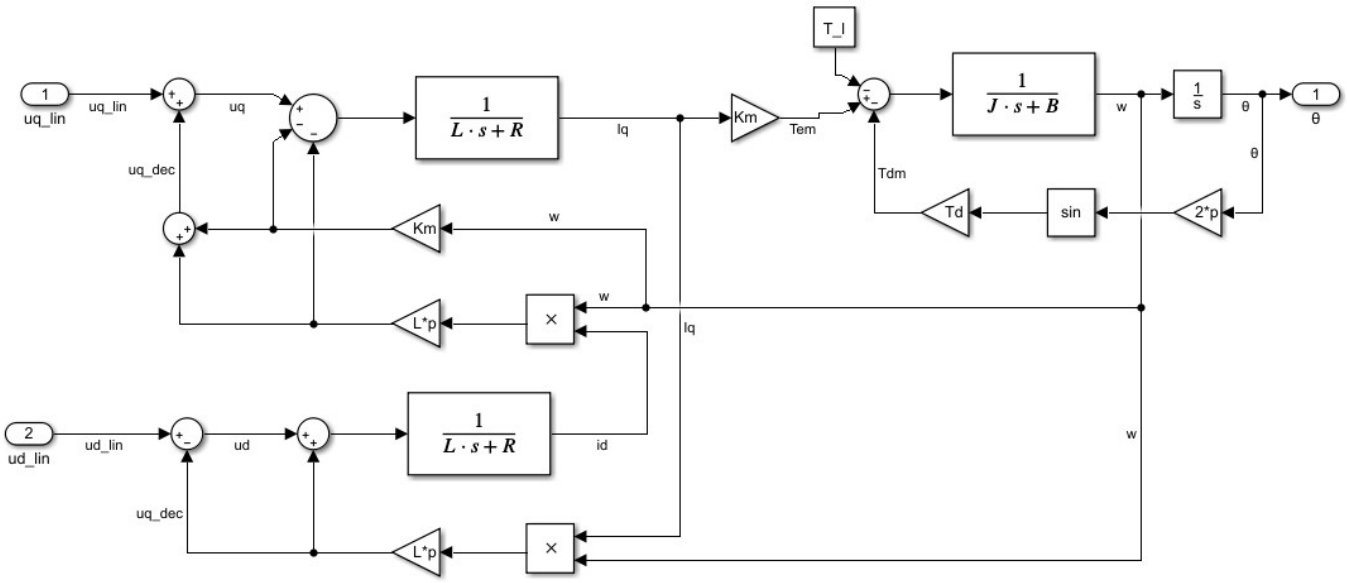

Figure 3.12 Decoupled Stepper Motor Simulink Diagram

The parameters of this model used were $L = 3x10^{-3}, R = 3, K_m = 3, T_d = 6, B = 4, p = 50,\ J = 0.08$. This system was then placed inside a closed loop

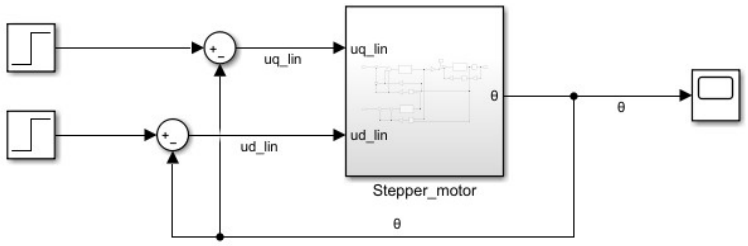

Figure 3.13 Stepper Motor Mechanical System Simulink Diagram

This system is actually the motor's mechanical model of the system and will be used later to simulate the whole motor with all the feedback controllers. To simulate this, we provided step inputs of values 8V and 5V respectively to this given system. The output obtained is

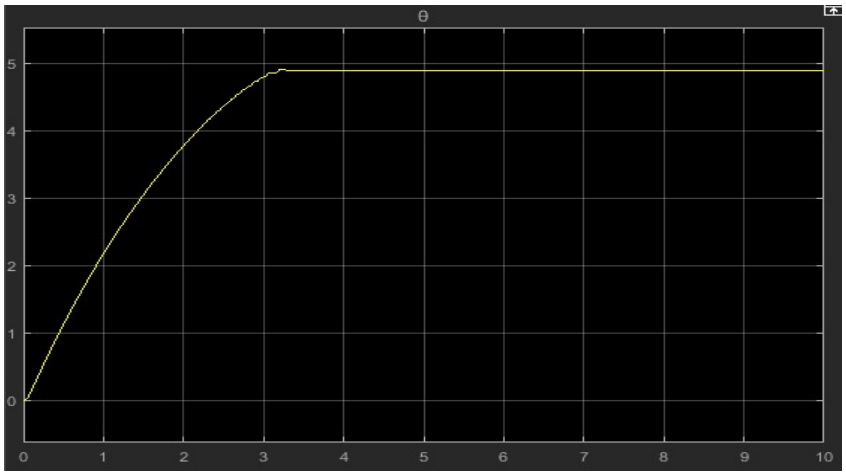

Figure 3.14 Stepper Motor Mechanical System Simulink Output between Output Angle θ (deg) w.r.t Time t(s)

Now, the configuration of the motor closed-loop drive system mentioned in [6] is considered in this research. In this configuration, three controllers are required, one for the position, the other for the speed and the third one for the current. The most inner loop is the current controller loop (which includes motor model which we just designed) and it has the largest bandwidth. The most outer loop is the position control loop. Figure shows the block diagram of this control configuration.

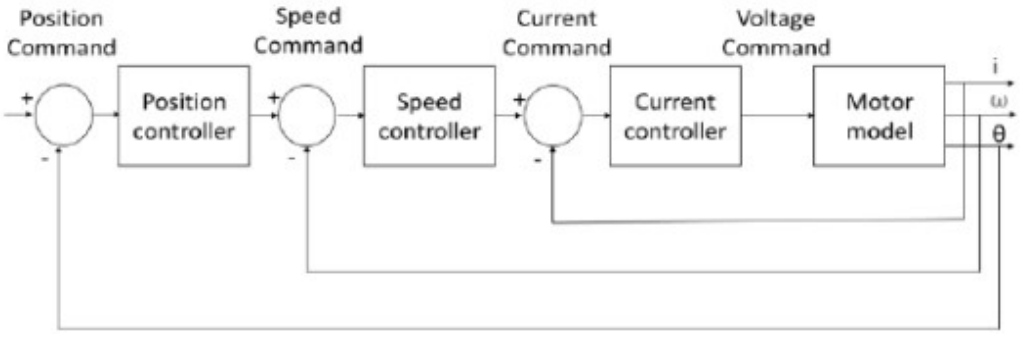

Figure 3.15 Stepper Motor Control Loop

Once this is achieved the electrical subsystem is reduced to the linear system:

$$L_w i_d = -R_w i_d + u_d^{lin} \qquad (10)$$

$$L_w i_q = -R_w i_q + u_q^{lin} \qquad (11)$$

Now the loop on both currents, i_q and i_d, is closed with a controller as shown where the blocks corresponding to the parts decoupled has been omitted in order to simplify the representation. This simplification is realistic if the decoupling between w_m (back emf), i_q and i_d is ideal.

Current Loop

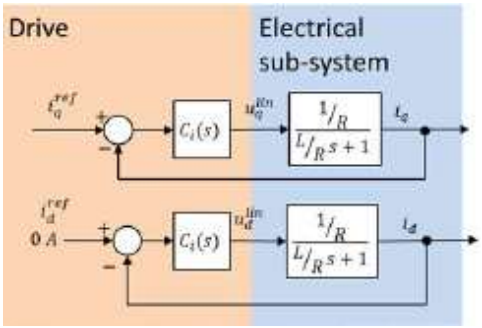

Figure 3.16 Stepper Motor Current Loop

For the current loop in ideal decoupling we find $I_T(s)$

$$I_T(s) = \frac{\frac{C_i(s)}{R}}{\frac{L}{R}s + 1 + \frac{C_i(s)}{R}}$$

The next step is to close the loop on w_m. For the speed controller calculation, the detent torque (amount of torque needed to overcome the static equilibrium of motor) has been neglected, since for this loop it is seen as a non-linear disturbance to be compensated and is not taken into account in the linear controller design

Speed Loop

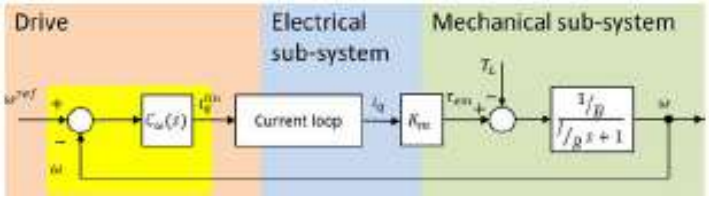

Figure 3.17 Stepper Motor Speed Loop

Now simplifying, the rotor speed loop

$$S(s) = \frac{\frac{1}{B}[K_m C_w(s) I_T(s) - T_L]}{\frac{J}{B}s + 1 + K_m C_w(s) I_T(s) - T_L}$$

One more loop is added on the position, on top of the speed loop as shown as

Position Loop

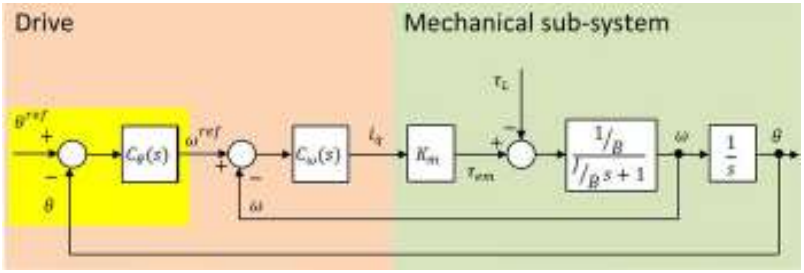

Figure 3.18 Stepper Motor Position Loop

Simplifying for the position loop we get

$$\frac{\theta}{\theta_{ref}} = \frac{C_\theta(s)S(s)}{s + C_\theta(s)S(s)}$$

Where

$$S(s) = \frac{\frac{1}{B}[K_m C_w(s) I_T(s) - T_L]}{\frac{J}{B}s + 1 + K_m C_w(s) I_T(s) - T_L}$$

And

$$I_T(s) = \frac{\frac{C_i(s)}{R}}{\frac{L}{R}s + 1 + \frac{C_i(s)}{R}}$$

Now we need to find the values of $C_\theta(s), C_i(s)$ and $C_w(s)$. Now as we know the PID equation is

$$C(s) = K_p + \frac{K_i}{s} + \frac{K_d N}{1 + \frac{N}{s}}$$

For $C_i(s), K_p = 4.01, K_i = 8.02, K_d = -0.14, N = 4$ we get

$$C_i(s) = 4.01 + \frac{8.02}{s} + \frac{0.56}{1 + \frac{4}{s}}$$

For $C_w(s), K_p = 1.97, K_i = 4.33, K_d = 0.12, N = 13$ we get

$$C_d(s) = 1.97 + \frac{4.33}{s} + \frac{1.56}{1 + \frac{13}{s}}$$

For $C_\theta(s), K_p = 6.99, K_i = 6.09, K_d = 1.95, N = 68$ we get

$$C_\theta(s) = 6.99 + \frac{6.09}{s} + \frac{132.6}{1 + \frac{68}{s}}$$

Putting the forms of $I_T(s), S(s), C_\theta(s), C_i(s), C_w(s)$ and other parameters, we can find the final transfer function. The system was then implemented in Simulink

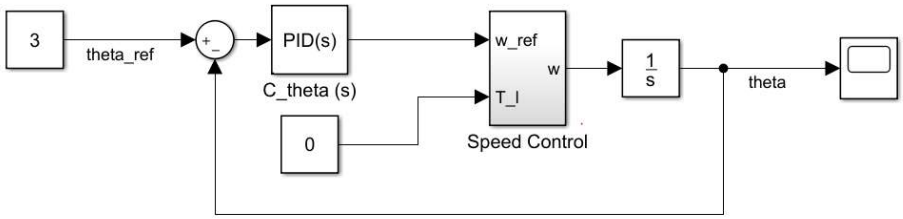

Figure 3.19 Stepper Motor Simulink Control Loop

Where the speed control module was

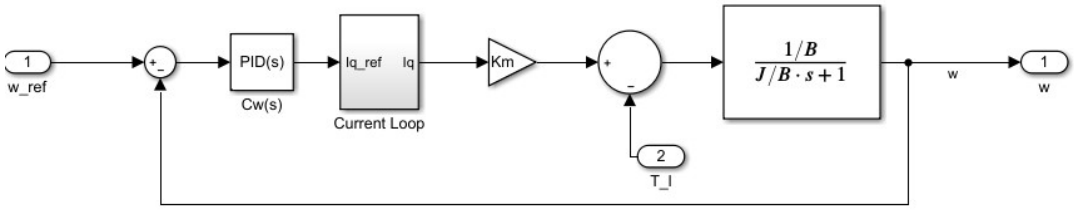

Figure 3.20 Stepper Motor Simulink Speed Control Loop

Now the current module will have the decoupled stepper motor system that we made earlier.

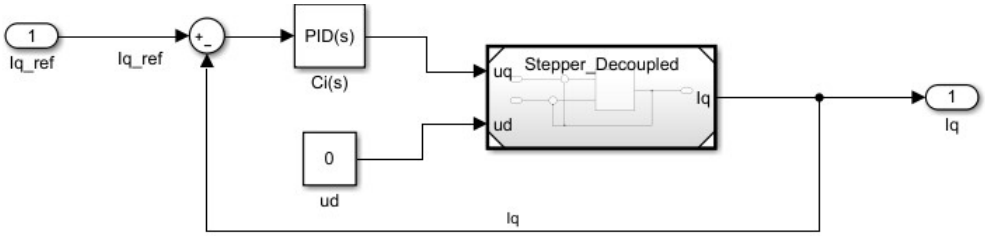

Figure 3.21 Stepper Motor Simulink Current Control Loop

The parameters of this model used were $L = 3x10^{-3}, R = 3, K_m = 3, T_d = 6, B = 4, p = 50, J = 0.08$. The input to the system was 3(degrees). The output obtained is as shown which closely resembles to that of the input

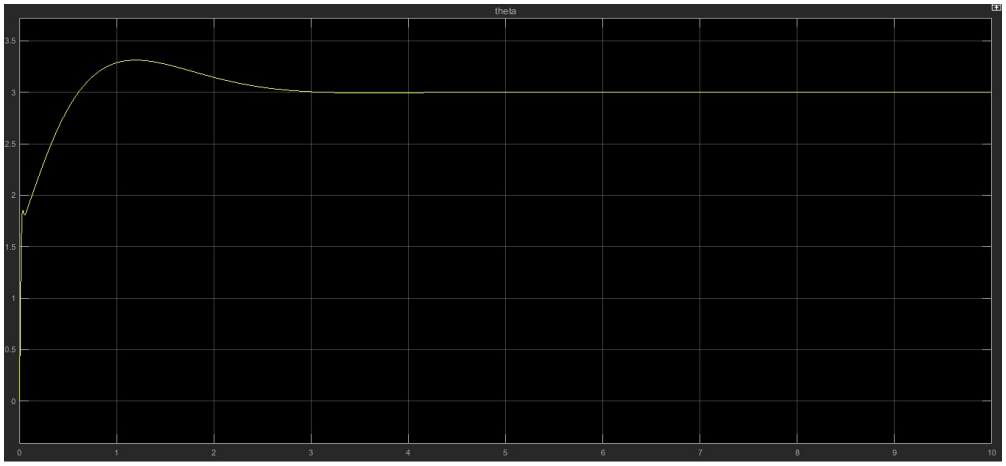

Figure 3.22 Stepper Motor Simulink Control Loop Output between Output Angle θ (deg) w.r.t Time t(s)

From the graph we observe that, it takes a second (or two) to exponentially reach its final but very precise final output. This means that it is very fast and responsive to be used in our system.

Electrical Hardware Design:

In this topic, we will discuss each electrical sub-division of the robot and see what parts they are made up of and how do they integrate with each other to provide the user knowledge and information to act on. The robot is electrically sub-divided into Robot-Control, Chassis-Motion, Elevation/Circular-Motion, Tilt-Motion, Environmental-Sensors, Object-Detection Sensors, Power-Circuit and Camera System. Schematics of each sub-division with final electrical schematic of the whole robot is provided in the end.

- **Robot Control:**

In order to incorporate all the sensors and motors and to provide a central command we need a micro-controller and then to relay this information and camera video stream to user via network we need a mini-computer capable of handling all the loads.

Micro-controller: Because of the vast number of sensors and drivers we are using with different types of interface, the robot uses the **Arduino Mega 2560 [16]**. It operates at 5VDC and has 54 digital I/O pins (15 PWM) and 16 analogue input pins. It has a clock speed of 16MHz with a large Flash memory of 256KB and SRAM of 8KB.

Figure 3.23 Arduino Mega 2560

Micro-Computer: The robot uses a **Raspberry Pi 4B (4GB) [17]** micro-computer to serve as the link between the robot and the user. It has a Quad-Core ARM processor running at 1.5GHz with 4GB of DDR4-RAM with on board wireless, Bluetooth and ethernet connections. It runs Linux-based kernel (Raspbian) and can connect with Camera's with CSI ports and monitors with HDMI ports. It has 4x USB ports to interface with serial devices.

Figure 3.24 Raspberry Pi

- **Chassis Motion:**

The chassis of the robot is responsible for the motion of the whole structure in all four directions and is essentially made up of two main components i.e. Geared DC Motor and Motor Driver

Geared DC Motor: These are small dc motors with operating voltage of 5-9V and current rating of 110-150mA with a geared box attached to it. Its load speed is somewhere between 95-175 rev/min with it covering a distance of 25-48m/min if a wheel is attached to it.

Figure 3.25 DC Geared Motor with Wheels

Motor Driver: To control the geared motor, we used the motor driver **L293D IC [6].** It is a digital input IC and can be used to run two DC motors with same IC and it can control both their direction and speed. It also provides the motor necessary voltage between 4.5-36V with maximum current of 1.2A.

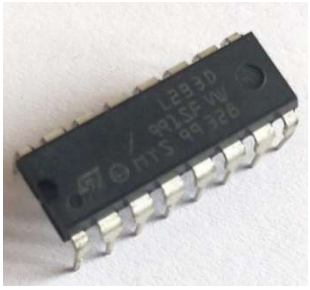

Figure 3.26 L293D Motor Driver IC

The electrical schematic on how the chassis electrical components are configured in the robot is explained by this diagram

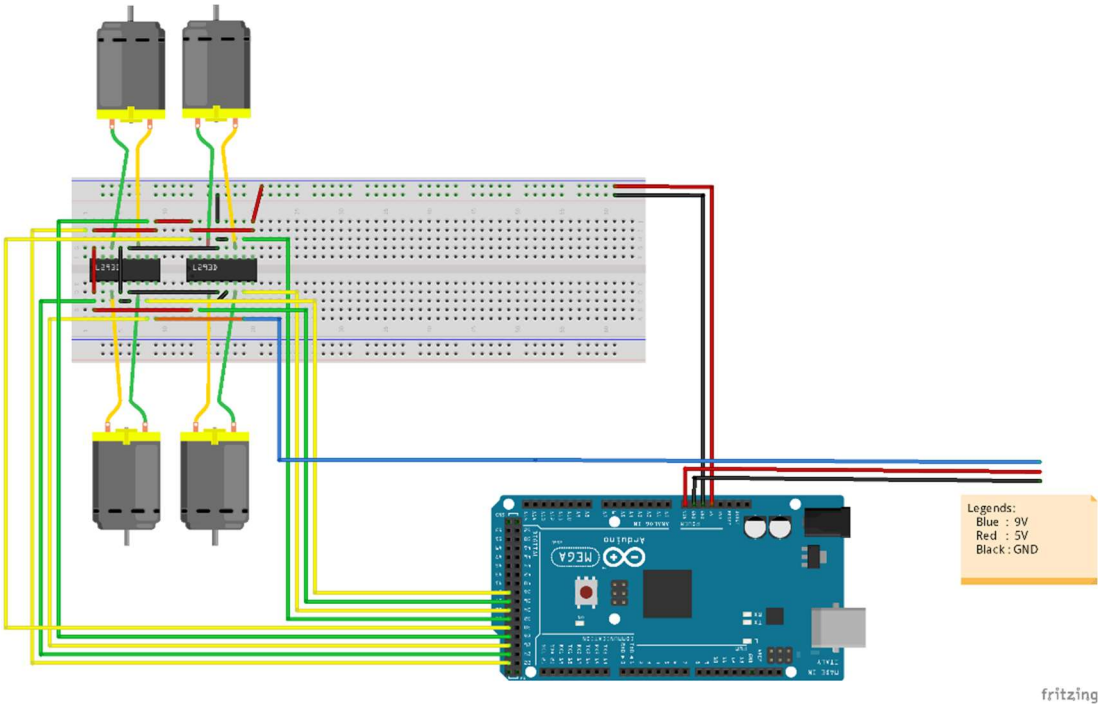

Figure 3.27 Robot Chassis Electrical Schematic

- **Circular/Elevation Motion:**

Both the circular motion and the vertical (elevation) motion of the camera platform uses the same two components i.e. a Stepper Motor and a Stepper Motor Driver

Stepper Motor: For this project, we used a bidirectional **Nema17 Stepper Motor [7]** which operates at 12VDC. It is a two-phase motor with each phase drawing a maximum of 1.2A with a high torque of 3.2kg-cm. This motor step angle is 1.8º i.e. 200 steps/rev.

Figure 3.28 Nema17 Stepper Motor

Stepper Motor Driver: To drive the Nema17 Stepper Motor we use the **A4988 Stepper Motor** Driver Module [8]. It is digital module designed to operate bipolar stepper motors in full-, half-, quarter-, eighth-, and sixteenth-step modes, with an output drive capacity of up to 35 V and ±2 A. It can control both the direction and steps of the respective motor.

Figure 3.29 A4988 Stepper Motor Driver

The electrical schematic on how the elevation/circular motion electrical components are configured in the robot is explained by this diagram

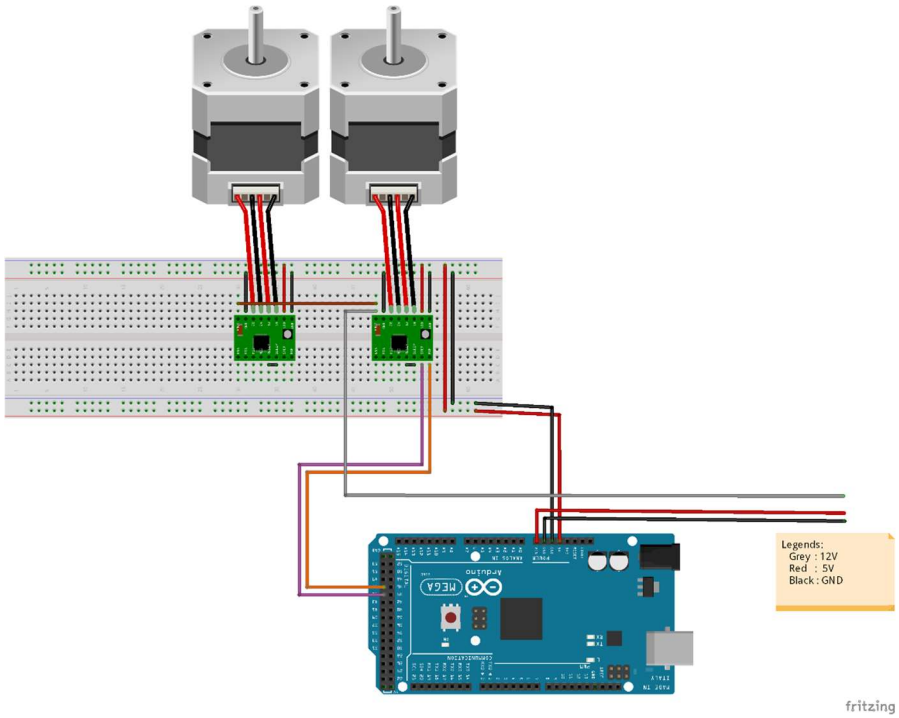

Figure 3.30 Elevation/Circular Motion Electrical Schematic

- **Tilt Motion:**

The tilt motion of the camera module is solely controlled by the servo motor and doesn't require any additional driver circuits.

Servo Motor: To provide tilt motion to the camera, we used an **SG-90 Servo Motor [9].** This motor operates at 5V and requires a PWM signal to operate. It can move 90º in each direction essentially making a 180º movement. It has a built-in feedback mechanism as explained in mathematical modelling that corrects its angle.

Figure 3.31 SG-90 Servo Motor

The electrical schematic on how the tilt motion electrical components are configured in the robot is explained by this diagram. It uses the analogue port of Arduino to get the signal and move accordingly.

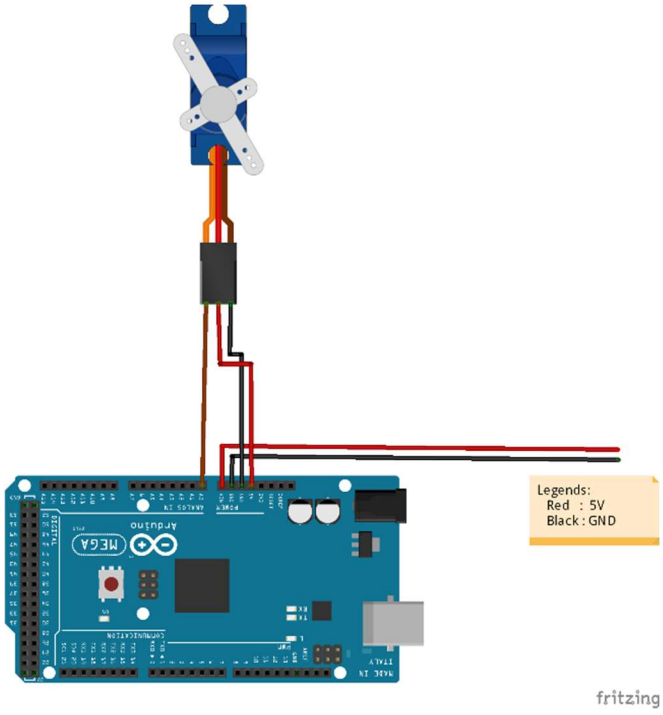

Figure 3.32 Tilt Motion Electrical Schematic

- **Environmental Sensors:**

The robot holds many sensors that can measure the temperature, humidity, CH4, CO2, CO concentration in its environment and relay it back to the users. To achieve this, following components are used.

Temperature & Humidity Sensor: In order to measure the temperature and humidity, we use the **DHT-11 sensor [10].** It operates at 5VDC and outputs a digital serial data to the micro-controller. It can measure temperature of the range 0ºC to 50ºC and relative humidity of range 20% to 90% with an accuracy of ±1°C and ±1% respectively.

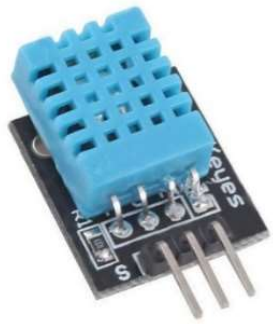

Figure 3.33 DHT-11 Sensor

Methane Sensor: In order to detect methane gas (CH4) in environment, **MQ4 sensor [11]** is used. It operates at 5V and has a high sensitivity to methane and can give both digital output and analogue output. It can detect concentration of methane between 200-10000ppm. Here we are using analogue output because of improved accuracy.

Figure 3.34 MQ-4 CH4 Gas Sensor

Carbon Mono-oxide Sensor: To detect CO in the environment, the robot is equipped with an **MQ7 sensor [12].** It is very similar to that of MQ4 sensor discussed previously i.e. it operates at 5V and can give both digital output and analogue output of which we again use analogue output because of better accuracy. It can detect concentration of CO between 20-2000ppm.

Figure 3.35 MQ-7 CO Gas Sensor

Carbon Dioxide Sensor: We used **MQ135 gas sensor [13]** to detect CO2 level in environment. The specifications of this sensor are again very similar to both MQ7 and MQ4 sensor with having analogue output interface that we use here. It can detect concentration of CO2 between 100-200ppm.

Figure 3.36 MQ-135 CO2 Gas Sensor

The electrical schematic on how the environmental sensors components are configured in the robot is explained by this diagram

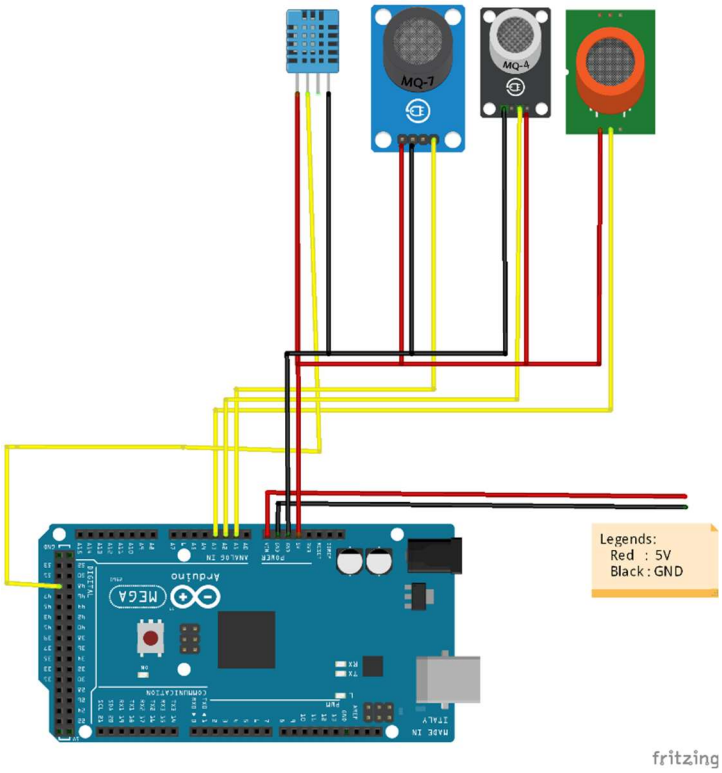

Figure 3.37 Environmental Sensors Electrical Schematic

**The green colored MQ-135 also requires a ground wire but the fritzing software library that supports this sensor grounds it by default, so its black wire isn't shown here.*

- **Object Detection Sensor:**

In order to detect incoming objects from all the four sides of the chassis and take actions to avoid them we need a sensor which can tell us how close we are to them.

Ultrasonic Sensor: To detect distance from objects, we use **HCSR-04 Sensor [14].** It's has a range between 2cm and 400cm with an accuracy of 3mm and a viewing angle of 15º with a refresh rate of 40Hz and gives a digital output to the controller. It uses Arduino built in library function ***pulseIn()*** to detect the time the digital output pin "Echo" remains HIGH and then from this distance is calculated by using the formula

Figure 3.38 HC-SR04 Ultrasonic Sensor

$$distance\ (cm) = time * \frac{0.034}{2}$$

The electrical schematic on how the object detection sensors components are configured in the robot is explained by this diagram

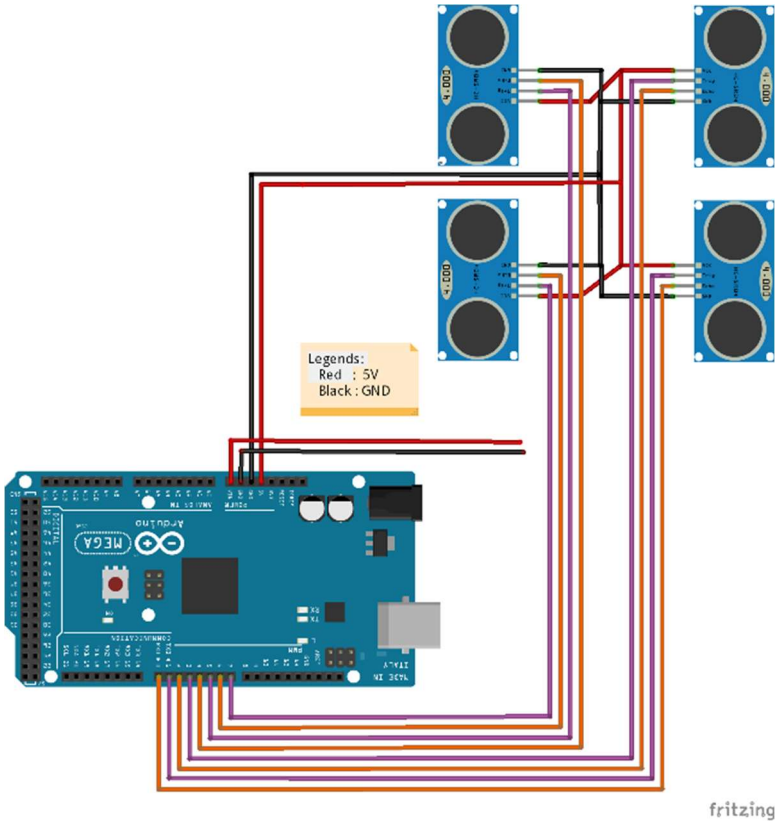

Figure 3.39 Object-Detection Sensors Electrical Schematic

- **Camera**

In order to live-stream the location to the user remotely, robot uses a **No-IR Raspberry Pi v2 Camera Module [15].** This is an 8-megapixel camera capable of taking infrared photographs of 3280 x 2464 pixels and can capture video at 1080p30, 720p60 and 640x480p90 resolutions.

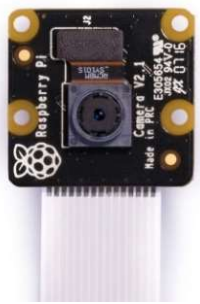

Figure 3.40 No-IR Raspberry Pi Camera

- **Power Circuit Hardware:**

Because there are many different types of components attached in a robot, they all have different power requirements both in terms of voltage and current. To centralize this, we use one rechargeable battery and use converters to supply power throughout the structure.

Power Supply: To power up all the components we used a **Lithium-Polymer (LiPo) 3S Battery [18].** This is a rechargeable battery and capable of providing 11.1VDC with a maximum current rating of 20A, continuous discharge rate of 25C and having capacity of 2200mAh.

Figure 3.41 Li-Po Battery

Buck Converters: In order to decrease the voltage provided by the battery to be used in digital circuits, we used buck converters. Even here we used two different kinds of converters because of current ratings.

- **For Raspberry Pi:** Because the current rating of Raspberry Pi is 2A and requires a USB interface to be powered up, we used a **LM2596 Buck Converter** Module. It can provide output of 5V with current up to 3A.

Figure 3.42 LM-2596 Buck Converter

- **For Arduino/Drivers & Motors:** Arduino and all the sensors/drivers requires only 5V with at maximum 1A drawn current. We also don't need a USB output interface so here we used an **XL4015 Buck Converter.** This converter inputs 8V-36V and can provide a maximum current up to 5A.
  For DC Motors, we use a separate XL4015 Buck Converter with its output set to 9V constant.

Figure 3.43 XL-4015 Buck Converter

Full Schematic:

Combining all the components and schematic above, the final diagram is presented below which explains the full electrical diagram of the surveillance robot.

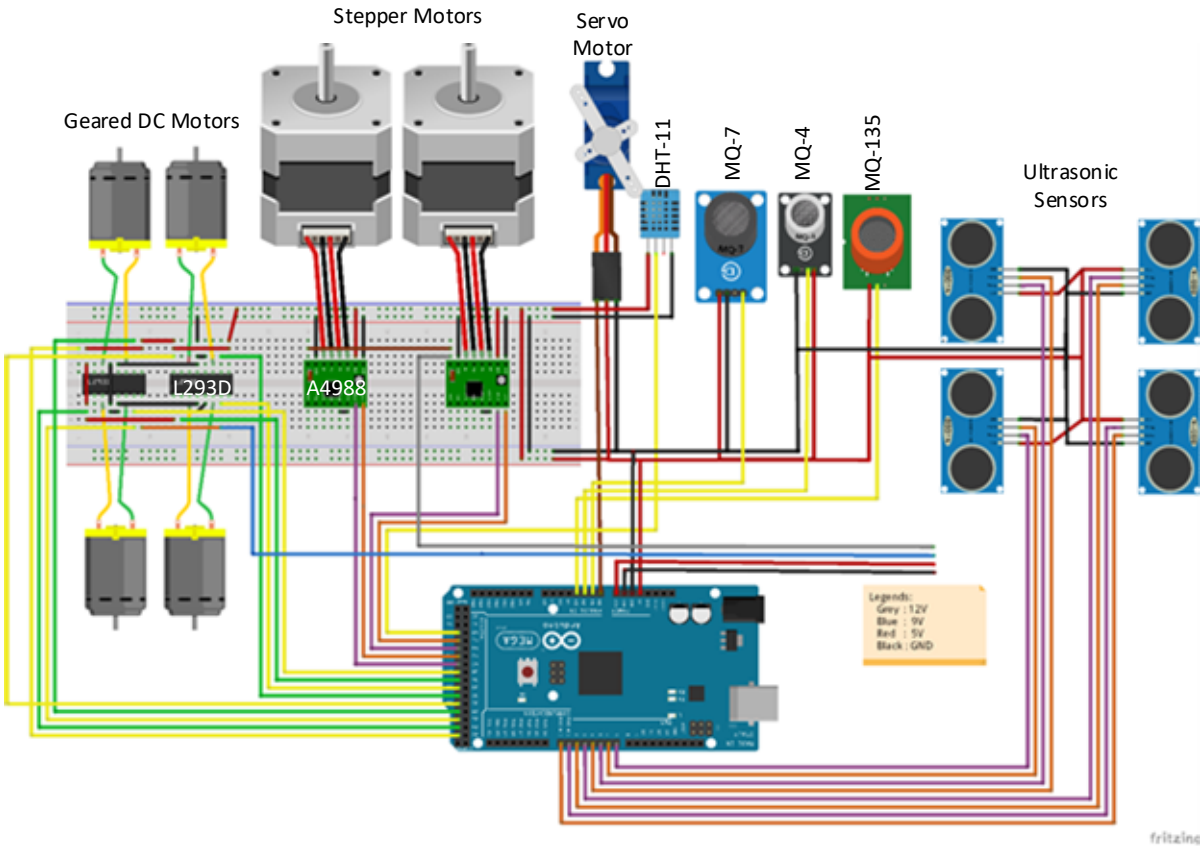

Figure 3.44 Complete Electrical Schematic of Robot

**The schematic doesn't include Raspberry Pi, Camera and Buck Converters because of the lack of its availability in the schematic software Fritzing.*

The grey wire is the 12V power line that is used to power the Stepper Motor's and is directly connected to the Li-Po Battery while for the 9V Power line which is used to power chassis DC motors, XL4015 Buck Converter is used which decreases the voltage from 12V at the battery. Then comes the 5V line which is used to power all the drivers, servo, Arduino, sensors and it too uses the XL4015 Buck Converter but Raspberry Pi uses the LM2596 Buck Converter because of its high ampere rating and requiring a USB output. At full load the robot draws about 3-4A at 12V including the stepper motors and Raspberry Pi.

The yellow colored wires from Sensors are the Analogue Input Data Lines while the red & maroon colored lines from ultrasonic sensors are Digital Input Data Lines. Also, yellow & green from L293D drivers, orange & purple from A4988 drivers and yellow line to Servo Motor are Digital Output Signals sent to motor

Mechanical Hardware Design:

The first major design challenge is to design the platform that will perform the circular 360º motion, the structure that will move the camera vertically up or down or tilt the camera in positive or negative direction. Then we will design the final robot in its final shape.

- **Circular Motion Platform**:

The circular motion to the camera is provided by the Nema17 Stepper Motor. But the platform also has to bear weight to the whole structure of the elevation system and the tilt system so the camera cannot be simply mounted on top. For this, a ball-bearing system is devised which uses the concept of flat-surface bearing to easily balance and rotate the whole elevation structure on it without creating too much stress on the motor. 15 ball bearings of size 10mm are used in the structure. The 3D CAD drawing of the platform is shown below

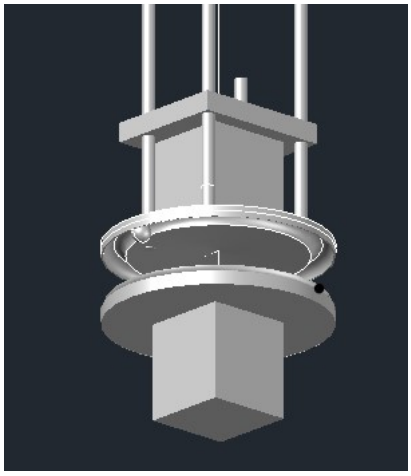

Figure 3.45 3D CAD Drawing of the Rotatory Platform

The following diagram shows the lower and upper disc of the bearing. It has the groves that contains that contains the ball bearing required to move.

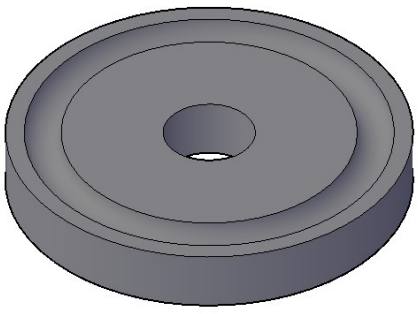

Figure 3.46 2D view of the Bottom plate of Rotatory Platform

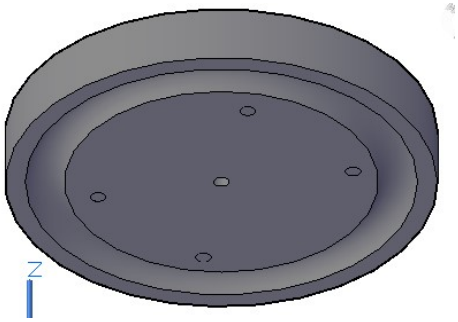

Figure 3.47 2D view of the Upper plate of Rotatory Platform

- **Elevation Motion Platform**:

To move the camera up or down vertically, again a stepper motor is used a structure is made that converts circular motion of the motor's rotor to linear motion. It uses a stainless-steel 300mm thread attached to the stepper motor using a stain-less steel nut. It then uses a aluminum slider with gear and ball bearings inside it attached to an immovable aluminum back-plate using rollers and moves in a translational motion when the thread turns because of the rotation of motor.

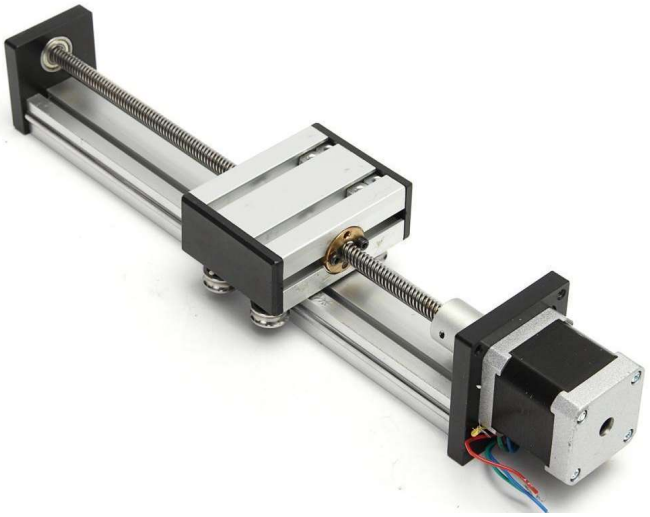

Figure 3.48 Elevation Platform Structure

This assembly was ordered from China [25] and wasn't made specially for the project.

This assembly which will sit on rotatory platform is made in such a way that it can be easily removable and user can use the robot with only rotation and tilt motion. This is usually feasible where there is very low height adjustment and 300mm long elevation structure will definitely hit it. The 3D Diagram of that said platform that will sit on top of rotatory platform is

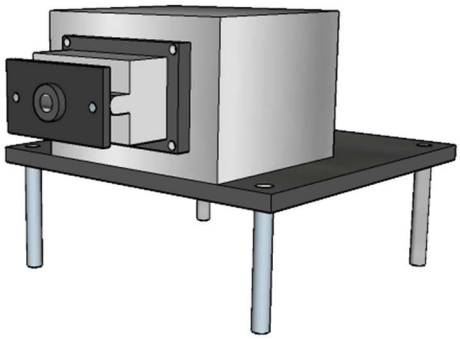

Figure 3.49 Elevation-Less Platform Structure

- **Tilt Motion Platform:**

As discussed earlier, tilt motion is done by the use of Servo Motor. For this, a plastic assembly is used (as it only needs to handle the load of camera, so doesn't need to be metallic). The servo's output rotor is connected to the plastic back of the camera and helps it turn 90° in both directions. This structure also has the ability to pan (i.e. rotate at its axis) but we achieve this through the rotational platform so this functionality wasn't used. This pan functionality conflicts with that of elevation platform because of how it is designed by the manufacturer. If we place it on the elevation platform and try to use this functionality, it will just rotate the live feed in a circular motion like a spiral. The only way to use this is to remove both Elevation and Circular motion platform and place it directly on the chassis but still then it can rotate only 90 degrees in 2 directions so 360-degree freedom won't be attained.

Figure 3.50 Tilt Platform Structure

This assembly was ordered from Lahore [26] and wasn't made specially for the project.

Designing the Final Robot:

The robot chassis is designed using 3D printing techniques which will enclose all the electrical components and hardware in it. The final structure too can be described and separated in terms of its types of motion. Also, the robot is so designed that its elevation platform is very easily removed with the help of four long screws and replaced by the flat-panel onto which the Tilt-motion platform rests.

The 3D designed structure (was under progress before university closing & subsequent lockdown) is made from PLA (Polylactic Acid) filament which is insoluble and is also water proof and can gives strong rigidity to the robot outer shell. Because the design is not supposed to house very small parts, layer width of 0.2mm is used which gives more support to the rest of the hardware.

The final robot approximately **weighs 4-5kg** (estimate) and have a height of 0.4m with chassis area of 0.23m x 0.31m (estimate). The detail dimensions are given further on.

The design of the robot with its parts displayed are shown below. Two separate diagrams are used in order to ease the displaying of components.

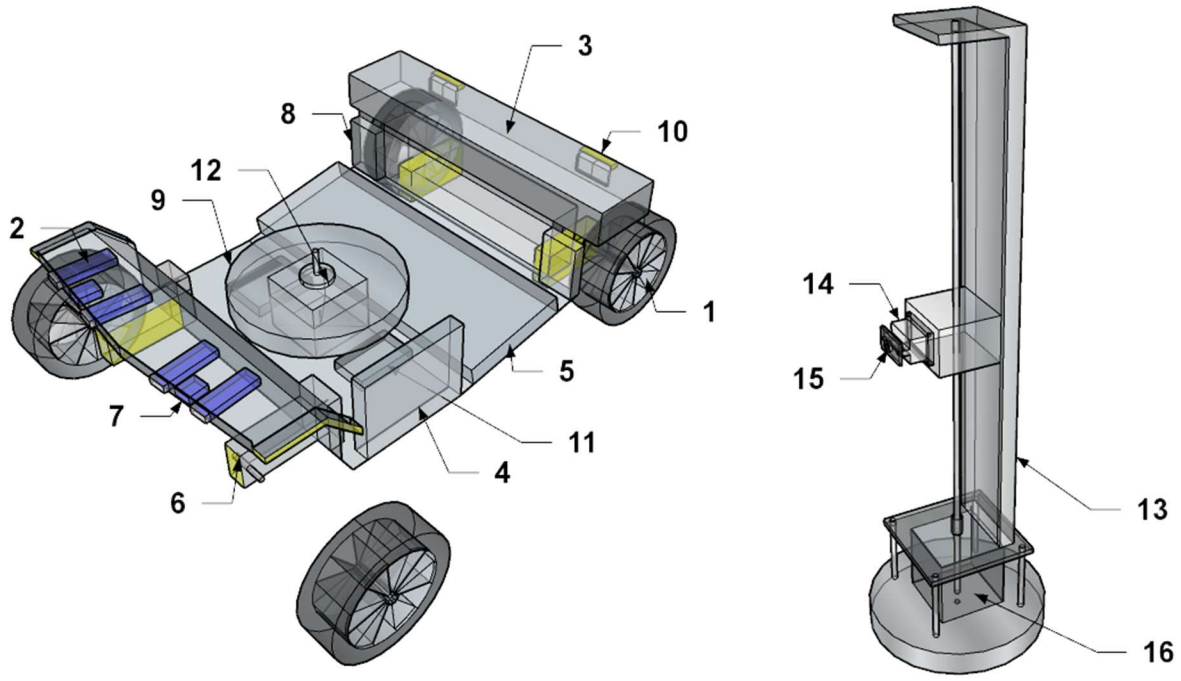

Figure 3.51 Disassembled Structure of Robot

1. Plastic-Rubber Tyres
2. Ultrasonic Sensor (for Object Detection)
3. Rechargeable LiPo battery
4. Arduino Mega Micro-controller
5. Raspberry Pi 4B Micro-computer
6. DC Geared Motors
7. Head Lights
8. DC Motor Driver (L293D IC, mirrored locations at all four DC motors)
9. Flat Surface Bearing (for Rotational Motion)
10. Environmental Sensors (including Temp/Humidity, CO2, CO, CH4)
11. Stepper Motor Driver (A4988 Module)
12. Stepper Motor (Nema17 for Rotational Motion)
13. Linear Actuator (for Vertical Motion)
14. Servo Motor Assembly (for Tilt Motion)
15. Camera Module
16. Stepper Motor (Nema17 for Vertical Motion)

The final structure of robot with its dimensions (units are in Imperial System) which are estimates and may differ from final product are given below.

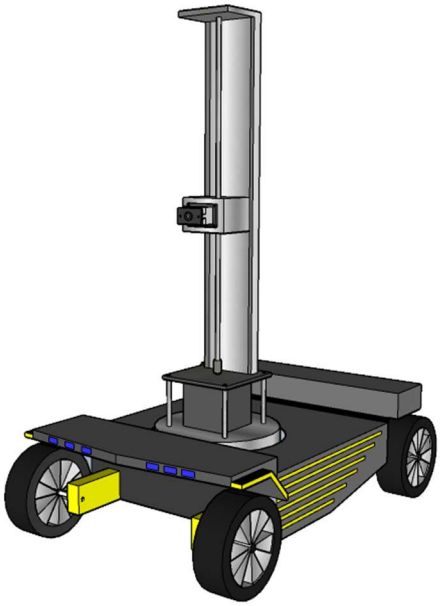

Figure 3.52 Complete Structure of Robot

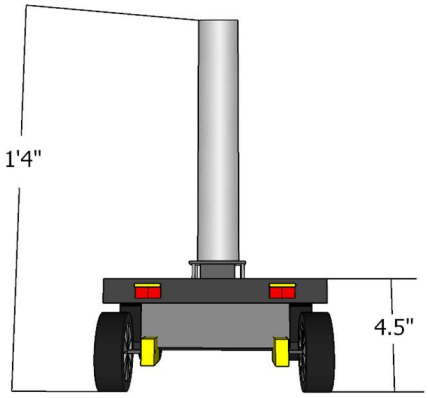

Figure 3.53 Back View of Robot

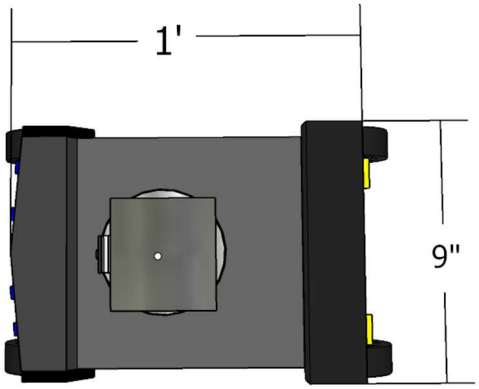

Figure 3.54 Top View of Robot

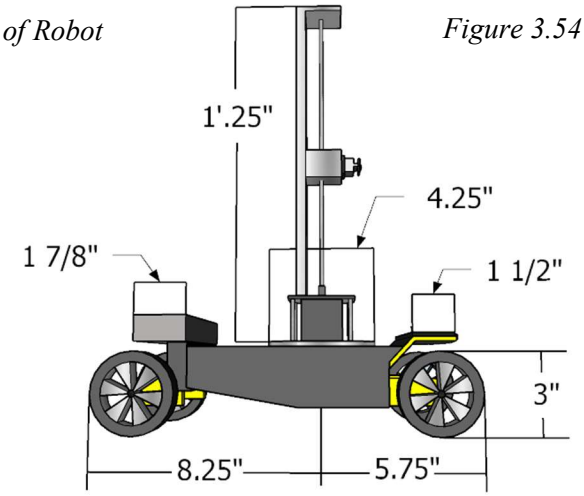

Figure 3.55 Side View of Robot

The model was made in SketchIt software and it may differ from the actual hardware product

Software Design:

In order for the robot to be controlled by a user remotely we need an interface and a way to reach through the internet to robot securely and take data and video from robot to user and transmits commands from user to robot. Here we will deal with step by step how to establish this communication link.

Controlling the Input to Motor's:

The robot motors are controlled using Arduino Mega Microcontroller using a programming language based in C. The code is written with several conditions as explained in **Annex A.**

- **Chassis Control**

The chassis movement is controlled by user input but before moving it always check certain conditions. It takes input from the all of the four ultrasonic sensors and see if there is anything that it can collide within the next 10cm of its movement and if there isn't, it moves 10cm in that specific direction.

- **Circular Motion Control**

The circular motion stepper motor is coded so that with a single user input, Arduino sends 5 pulses of signals to its driver, rotating the camera above it only by 9º. The controller also keeps count as not to exceed full two circular rotations counts in one single direction as it would put strain on the wirings of both motor and camera.

- **Elevation Control**

The elevation motion stepper motor is coded so that with a single user input, Arduino sends 1000 pulses (200x5) of signals to its driver, lowering or elevating the camera about 1cm. (The pitch of the screw on which the camera resides is 2mm, so in order to move 1cm, 5 full rotations of screw are required and each full rotation needs 200 pulses to the driver). The controller also makes sure that it keeps count the distance travelled in both directions so as not to hit either end accidently.

- **Tilt Control**

The tilt motion servo motor is coded so that with a single user input, Arduino sends a PWM signal to the motor to move the camera in positive/negative direction by 15º. Arduino also makes sure servo doesn't get more then 6 pulses in one direction as it cannot go beyond 90-degrees and if tried, it resists and stresses the gear assembly.

Remote Connection to Robot:

The first task is to establish a network communication between user and robot via network. There are two ways that this is achieved.

- **Local-Area Network Connection**: When the Raspberry Pi 4B connects with the local Wi-Fi (which is pre-configured onto the system by user), it is assigned an IP address by the router. This IP address is unique when compared to all the devices connected to the same router and any other device which is connected to the same router can communicate it with Raspberry Pi with this IP address and for that no internet at the back-end is required. This connection is possible if you are in close proximity to the robot which is often not the case so we need a way to connect via the internet
- **Internet Connection**: The most common way to connect via internet is to buy a static IP address from internet but this robot uses an alternate way. It uses global-area networking service called **Zero Tier [19].** Essentially it virtualizes the concept of Local-Area Networking over the cloud. We formed a network on their service (essentially making Zero Tier a router) and connect and authenticated both robot and user system on that network. Both the user and robot are assigned an IP address which can be used in the same way as two devices in a local-area network connected are used.

Live-Video Streaming across Network

In order to stream video, Raspberry Pi uses the No-IR 5MP camera. It uses **crtmpserver [20]** (a high-speed streaming server) to transmit the video-data captured using **ffmpeg [21]** (set of libraries used to record, store and transmit video-data). The live-video is streamed onto the webpage on the user-side (webpage URL is the IP address of robot) using a Flash-based **Strobe Player [22]** library. The Player code is implemented on the client-side. In order to conserve band-width and to stream at as minimum latency as possible the resolution of the video-stream was set to 600x400.

Live-Data Transmission across Network

The sensors transmit the data to the Arduino Micro-controller which needs to be displayed to the user and the user commands the movement of the robot which is conveyed by Arduino to the drivers that controls the motors. The transmission is done using a library called **php_serial.class.php [23]** which is implemented in PHP and a Linux-based app called **Minicom [24]**. This takes uses the serial communication interface of the Arduino and Raspberry Pi and sends and receives data continuously in a loop both from the user and the sensors. This is also

implemented on the client-side. The baud rate of the data communication was set as default to 9600 by the Raspberry Pi client.

Client-side Web-Interface:

In order to be the friendliest and requiring the least amount of computer resources, the user-interface was designed on web which can accessed by any authenticated user by just typing the IP of the robot. (Many browsers block flash-based players by default, so user may need to allow flash-player access in order to get the video-stream). When the user opens up the interface, the script looks for a device named '***ttyACM0'*** (Linux name's for Arduino) and establish connection with it before starting the rest of communication. It uses the Raspbian time to show the live date & time on the interface and also logs the connection in its memory. All of interface and connection files are placed in the /var/www/html/ location in Raspbian which makes opening them up much simpler by just typing in the IP in the URL bar on a browser.

The interface was designed in HTML5 **[Annex B]** and styled in CSS3 **[Annex-C]** with the user-control elements (like control buttons) made in PHP7. The player-code was written in JavaScript.

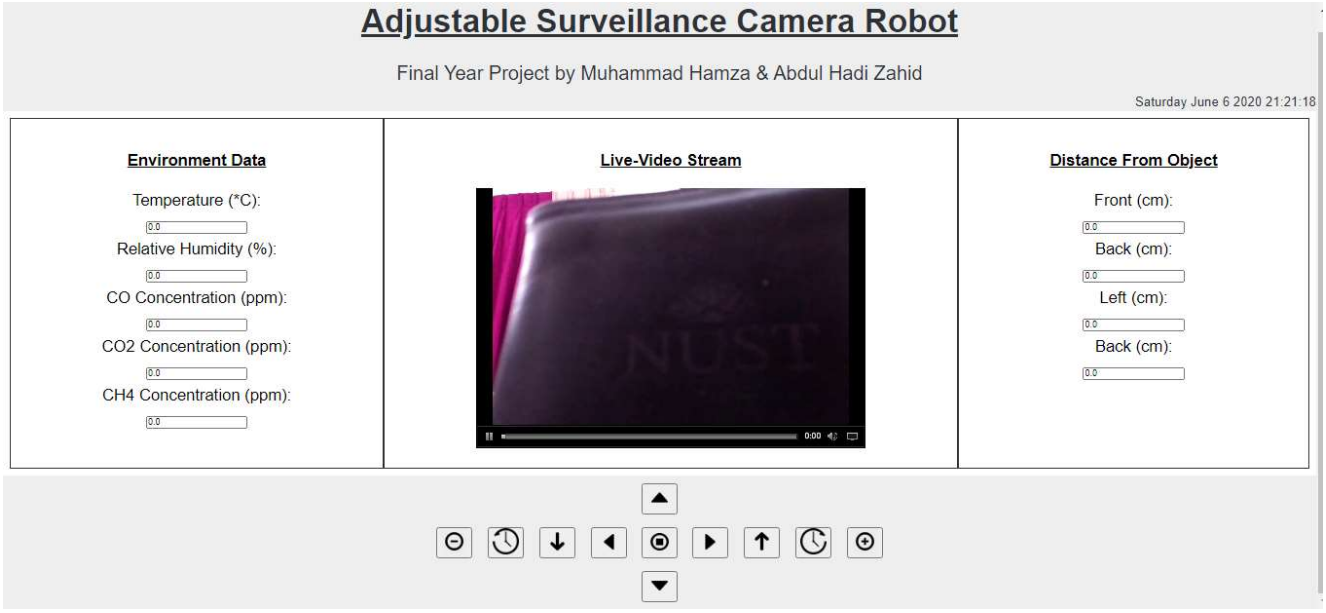

Figure 3.56 User Web-Interface

As seen from the interface screenshot above, user can see real-time environment and proximity data from sensors on the screen on both left and right sides of video-stream which is in the middle (the values of all these sensors are zero, because of the lack of final-interface testing with sensors due to COVID-19, the sensors were previously tested on old-interface).

The user can control the robot using the buttons available at the bottom of the screen. The options available are

Chassis Control

- Steer Forward: Making the whole robot chassis move forward
- Steer Reverse: Making the whole robot chassis go in reverse
- Steer Left: Making the whole robot chassis turn in left direction
- Steer Right: Making the whole robot chassis turn in right direction

Elevation Control

- Up: Elevate the camera in upwards direction
- Down: Lower the camera in downwards direction

Rotatory Control

- Clockwise: Rotate the camera in clockwise direction
- Anti-Clockwise: Rotate the camera in anti-clockwise direction

Tilt Control

- Positive: Tilt the camera in positive direction w.r.t x-axis
- Negative: Tilt the camera in negative direction w.r.t x-axis

Stop

- Stop: Stop all the movements of the robot instantaneously

Flowchart of Robot Control:

The following flowchart shows the sequence of how a robot functions when it is given an input by the user to move in a certain way and what kinds of conditions it pass through before performing the said action.

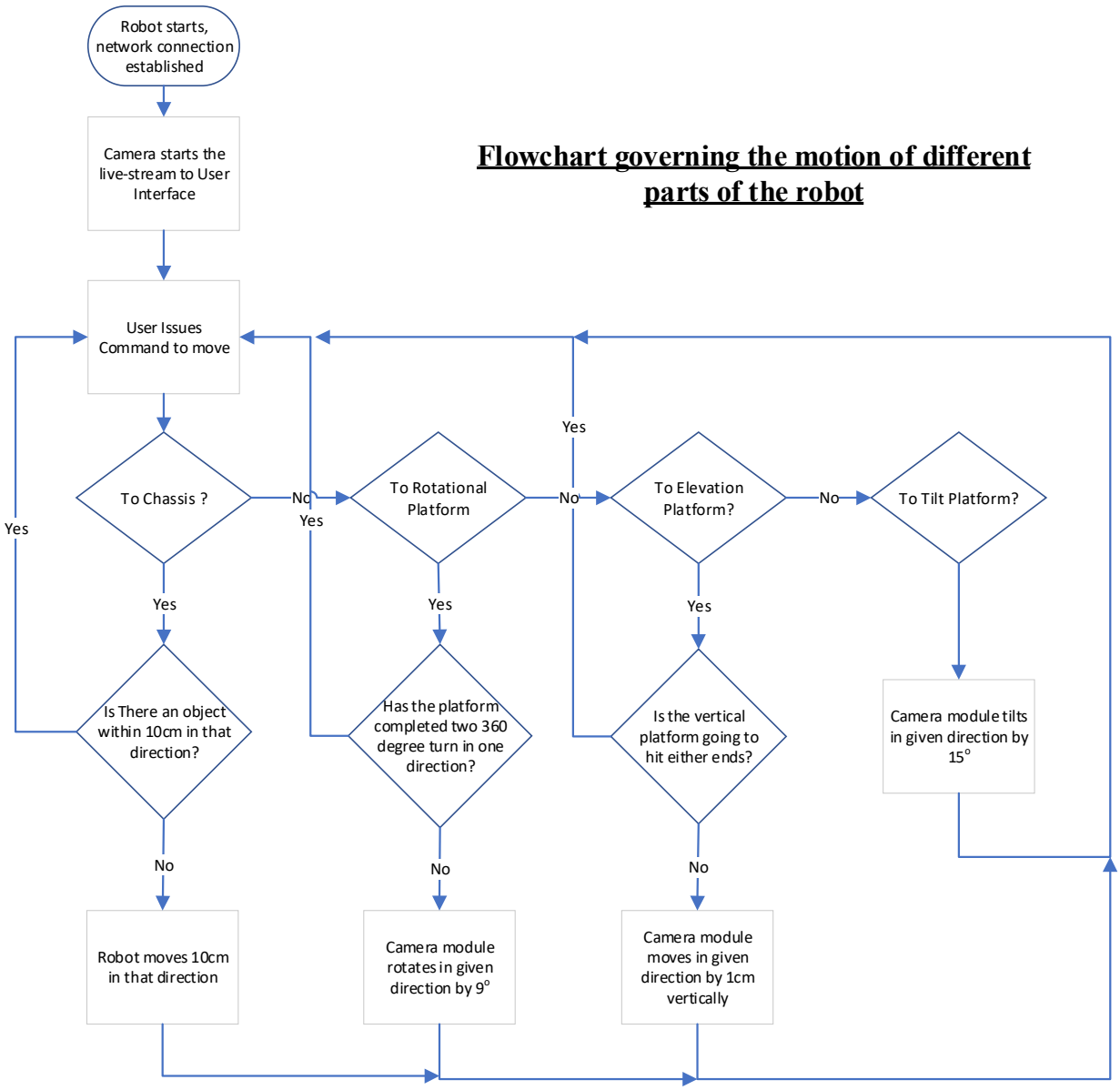

Figure 3.57 Flow-Chart of the motion of Robot

Chapter 4

Testing and Results

Pre-COVID'19

The first prototype of the robot was made using wood as structural material and was more focused on solving all the problems related to motions of the robot. This was done by mid-defense of the project. At that time, it had a very simple interface compared to the new one and then the robot chassis could move, the camera could elevate, tilt and rotate according to user-inputs. At that time, environmental sensors and object detection wasn't as of yet implemented.

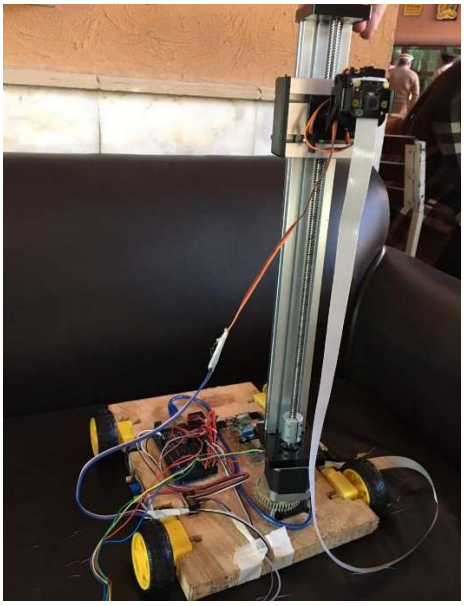

Figure 4.1 Pre-COVID Prototype

Post-COVID'19:

As seen by the CAD models and electrical diagrams in architecture chapter, all of the environmental and object detection sensors were checked and interfaced with robot but the rotatory platform (made from Aluminum in SMME Workshop) and the 3D printed robot chassis (made from plastic in SEECS 3D Printing Lab) was designed in software and was under or due to be manufactured. But because of the COVID-19 and the closure of campus and all of its facilities on 13th March, 2020, the finalized hardware was never manufactured hence it couldn't be tested.

Because of the lack of equipment, the hardware part remained incomplete, but the software interface was completed and a animation was made in Autodesk Maya software which shows how the robot would have worked in its final form.

Cost Analysis

One of the biggest selling point of this project is it being very inexpensive in nature compare to its counterparts which many times more than ours. Some of the examples of robots with some similar features in market are given below

SMP Security provides **S5.2 Series Security Bots [27].** They too have wireless control, 360-degree view, IR sensor with some additional features like face detection and mesh network capabilities for patrolling and their costs starting from 39,900$ (6.2 Million PKR) up to 45900$ (7.5 Million PKR).

Figure 5.1 S5.2 Series Security Robot

Inspector Robots provides their **Pan-Tilt Mini-Robot [28]** for a cost of 4590$ (0.9 Million PKR). This robot has similar wireless functionality with pan-tilt features of our robot with a professional grade camera.

Figure 5.2 Pan-Tilt Mini-Robot

Appbot provides their **Riley Homebot [29]** for a cost of 168$ (28000 PKR) which is even though less then ours but it doesn't have any of the rotatory/elevation/tilt motions that our robot provides.

Figure 5.3 Riley Homebot

Now we will discuss in detail about the total cost of the final robot. (*Shipment costs not included*)

S/No	Product	Quantity	Price (PKR)	Reference
1	Raspberry Pi 3B with Accessories	1	7000	[30]
2	30cm Linear Actuator + Stepper Motor	1	10500	[31]
3	Raspberry Pi Camera with Cable	1	1350	[32][33]
4	Pan-Tilt Servo-Bracket	1	250	[34]
5	LM2596 Buck Converters	1	150	[35]
6	XL4015 Buck Converters	2	560	[36]
7	DC Geared Motor with Wheels	4	600	[37]
8	L293D DC Motor Drivers	2	80	[38]
9	DHT-11 Sensor	1	140	[39]
10	MQ-4 Methane Sensor	1	290	[40]
11	MQ-7 CO Sensor	1	230	[41]
12	MQ-135 CO2 Sensor	1	220	[42]
13	HC-SR04 Ultrasonic Sensor	4	560	[43]
14	Arduino Mega 2560	1	1250	[44]
15	SG90 Servo Motor	1	220	[45]
16	Nema17 Stepper Motor	1	600	[46]
17	A4988 Stepper Motor Driver	2	280	[47]
18	Wires	1	200	-
19	10mm Ball Bearings	12	240	-
20	LiPo Battery (2200mAh)	1	3000	[48]
21	Rotatory Platform Manufacturing	1	2000	-
22	3D Printing (*Estimated Quote*)	1	3000	-
23	Miscellaneous	1	500	-
	GRAND TOTAL		**33220 PKR (202$)**	

(*Actual cost may vary from the reference due to changing in price in markets especially change in value of Rupee compare to Dollar in international shipments*)

Chapter 6

Future Works

Current Problems & Possible Solutions:

The robot designed even though very robust has a few shortcomings that still need to be addressed. Some of these problems are mentioned below

- In order for camera to start streaming, user has to SSH into the Raspberry Pi and uses its terminal window to input the command to capture video. This is due to security reasons and locked by Raspbian OS but it stills adds an additional layer of technicality.
- In order for serial communications between Raspberry Pi and Arduino, user again has to SSH into the Pi and open minicom in the background. Because this software has to run continuously in the background, it cannot be added to startup list of files as it starts running in infinite loop and never finish completing the boot-up.
- The elevation platform raises the height of the whole structure by 300cm which makes it impossible for the robot to pass through areas with less height.
- The camera module isn't suitable for capturing video at dark places which makes capturing it difficult in confined areas if headlights are not working.
- The on-board Wi-Fi adaptor of Raspberry Pi is small so if a router is far away from the robot, the robot will lose connection and stop working and for it to be recovered, workers have to physically find it at the location.

These problems can be modified by more research or updating or replacing a component for different piece of hardware

- Learning Linux Kernel at more basic level to auto-start the camera and the Minicom app on startup by bypassing the security restrictions.
- Modifying the current actuator of the elevation platform with piston-based Actuator in order to reduce its height significantly, so it only extends as needed by the user.
- Providing real-time updates and warnings to the user on network connectivity so user can take necessary actions before it becomes out of range.
- The current camera is capable of infra-red night vision but it isn't implemented. It could be done so to solve the dark video issues in confined spaces

Future Improvements & Recommendation:

This robot is just a prototype and it can be improved much farther for it to be more appreciated by the industrial community. Some of our recommendations are

- In industry, many more types of sensor data are supposed to be collected which can be added to the current capability of the robot.
- The current chassis of robot is only capable of moving on straight or slight uneven surfaces. Different types of wheels setup could be used to improve this and make it so it can move on many other types of rough terrains.
- The current user-interfaced is open on the network which means anyone can access it. It can be closed down with User-ID and Password for security reasons.
- The player used to stream video is Flash-based which has known security-vulnerabilities. It has to be changed to HTML players before it could be deployed to commercial use.
- Mechanical arm can be added to increase the functionality of this robot as currently it serves only to observe and note.
- Functionality can be added so in case of a catastrophic connection failure with user, the robot follows a pre-specified path back to safety and doesn't remain stranded there.

Chapter 7

Conclusion

Working in an industrial environment is always a risky job especially when we add working in confined spaces and hazardous areas. Many of the time, the job is only to assess the area and our robot completely takes the risk-factor out of it. By using this robot, getting data about conditions of the hazardous area and seeing it has never been easier. This robot being inexpensive especially considering the rest of machinery of an industry can easily be made part of the daily inspections. With easy-to-use user-interface and much easier way to modify the working conditions of robot in the code, this robot could be a solution to many problems faced by the industries including reducing cost to implement procedures and equipment to protect human lives.

Work Distribution

This whole project can be divided into three distinct portions on practical side i.e. mechanical hardware, electrical hardware and software development and on theoretical side it can be divided into two parts i.e. project report and demonstration video. Both of us team members contributed in every scope of the project and the following data highlights this

	Muhammad Hamza	***Abdul Hadi Zahid***
Practical Side	Mechanical Hardware Development (Tilt Platform)	Mechanical Hardware Development (Chassis, Rotatory, Elevation Platform)
	Electrical Hardware Development (System Control, Chassis, Rotatory Elevation, Tilt Motion, Camera Circuits)	Electrical Hardware Development (Power system, Environmental & Object Detection Sensor)
	Software Development (User Interface, Remote Network Connections)	Software Development (Electrical Components Integrations to MCU)
Theoretical Side	Project Report (Literature Review, Mathematical Modeling, Design Methodology & Overview, Electrical Hardware Design, Software Design)	Project Report (Mechanical Hardware Design, Introduction & Conclusion, Robot 3D Graphics Design, Future Recommendations)
	Demonstration Video (User-Interface)	Demonstration Video (Robot Motion Animation, Robot Parts Explanation, Introduction & Conclusion)

References

[1] Joshi, S.A., Tondarkar, A., Solanke, K. and Jagtap, R. 2018. Surveillance Robot for Military Application. *International Journal of Engineering and Computer Science*. 7, 05 (May 2018), 23939-23944

[2] Joisher, Neha and Jain, Sonal and Malviya, Tejas and Gaikwad (Mohite), Vaishali, Surveillance System Using Robotic Car (October 30, 2018)

[3] https://bit.ly/3cVKlLs

[4] Karadeniz, Ahmet & Alkayyali, Malek & Szemes, Péter. (2018). Modelling and Simulation of Stepper Motor For Position Control Using LabVIEW. Recent Innovations in Mechatronics. 5. 10.17667/riim.2018.1/7.

[5] http://oa.upm.es/32464/1/tesis-Ricardo-Picatoste.pdf

[6] http://www.ti.com/lit/ds/symlink/l293.pdf

[7] https://www.cuidevices.com/product/resource/nema17-amt112s.pdf

[8] https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf

[9] http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

[10] https://bit.ly/3fflP9J

[11] https://bit.ly/2MVye6x

[12] https://www.sparkfun.com/datasheets/Sensors/Biometric/MQ-7.pdf

[13] https://bit.ly/3cUcId1

[14] https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

[15] http://www.farnell.com/datasheets/2056180.pdf

[16] https://docs.rs-online.com/9ab3/0900766b80e8ba22.pdf

[17] https://bit.ly/2Ajgfoh

[18] https://download.mikroe.com/documents/datasheets/battery-datasheet.pdf

[19] https://www.zerotier.com/

[20] http://manpages.ubuntu.com/manpages/bionic/man1/crtmpserver.1.html

[21] https://ffmpeg.org/

[22] https://sourceforge.net/adobe/smp/home/Strobe%20Media%20Playback/

[23] https://gist.github.com/gravataLonga/6c89821b845d15e939a0

[24] https://help.ubuntu.com/community/Minicom

[25] https://bit.ly/2zabWea

[26] https://bit.ly/37mmvY7

[27] https://smpsecurityrobot.com/products/

[28] https://www.inspectorbots.com/Inspector_Bots/Pan_Tilt_Minibot.html

[29] https://amzn.to/2B012rY

[30] https://bit.ly/2BYogPQ

[31] https://bit.ly/2AuSWI3

[32] https://bit.ly/3dTHUKJ

[33] https://bit.ly/3cVxQ2L

[34] https://bit.ly/37mmvY7

[35] https://bit.ly/30A3jVM

[36] https://bit.ly/2BYq5wa

[37] https://bit.ly/2BPIWcz

[38] https://bit.ly/3hls2Tq

[39] https://bit.ly/2UCR2Mp

[40] https://bit.ly/3hdVCdH

[41] https://bit.ly/3fgxo0x

[42] https://bit.ly/2YxgsMG

[43] https://bit.ly/3frYPET

[44] https://bit.ly/2MMfzKu

[45] https://bit.ly/37mAFse

[46] https://bit.ly/2MSbR25

[47] https://bit.ly/3cWd5Ux

[48] https://bit.ly/2XSk6kO

Annexures

Annex A:

Arduino MCU Code for Robot

```
1.  //***** ARDUINO CODE FOR SURVEILLANCE CAMERA ROBOT****//
2.  #include "Servo.h"         //Servo Motor Library
3.  #include "DHT.h"           //DHT Sensors Library
4.  #define DHTTYPE DHT11     //defining the type of dht used in robot
5.  Servo tiltControl;         //Tilt Motion Conrol Object
6.
7.  char receivedChar;
8.  boolean newData = false;
9.  //defining the digital pins of dc geared motors
10. const int dcMotor1pin1 = 23;
11. const int dcMotor1pin2 = 25;
12. const int dcMotor2pin1 = 27;
13. const int dcMotor2pin2 = 29;
14. const int dcMotor3pin1 = 31;
15. const int dcMotor3pin2 = 33;
16. const int dcMotor4pin1 = 35;
17. const int dcMotor4pin2 = 37;
18. //defining the digital pins of elevation motion stepper motor
19. const int elevationStep = 39;
20. const int elevationDirt = 41;
21. //defining the digital pins of circular motion stepper motor
22. const int circularStep = 43;
23. const int circularDirt = 45;
24. //defining the digital pins of the object detection ultrasonic sensors
25. const int ultraSonic1Trig = 2;
26. const int ultraSonic1Echo = 3;
27. const int ultraSonic2Trig = 4;
28. const int ultraSonic2Echo = 5;
29. const int ultraSonic3Trig = 6;
30. const int ultraSonic3Echo = 7;
31. const int ultraSonic4Trig = 8;
32. const int ultraSonic4Echo = 9;
33. //defining the digital pins of environmental sensors
34. const int dht11pin = 47;
35. const int mq4pin = A0;
36. const int mq7pin = A1;
37. const int mq135pin = A2;
38. const int Servopin = A0;
39.
40. int tiltangle = 0;      //tilt motion angle set to zero on start
41. int elevDist = 0;       //elevation motion distance from bottom set to zero on start
42. int circDist = 0;       //circualr motion angle set to zero on startup
43. int dist1object = 0;    //distance of the nearby object to ultrasonic sensor 1
44. int dist2object = 0;    //distance of the nearby object to ultrasonic sensor 2
45. int dist3object = 0;    //distance of the nearby object to ultrasonic sensor 3
46. int dist4object = 0;    //distance of the nearby object to ultrasonic sensor 4
47.
48. void setup() {                          //initial setup of MCU
49.   Serial.begin(9600);                  //serial communication begin at baud rate of 9600 bps
50.   //setting all the motor driving pins to OUTPUT mode and sensors pins to INPUT mode
51.   pinMode(dcMotor1pin1, OUTPUT);
52.   pinMode(dcMotor1pin2, OUTPUT);
53.   pinMode(dcMotor2pin1, OUTPUT);
54.   pinMode(dcMotor2pin2, OUTPUT);
55.   pinMode(dcMotor3pin1, OUTPUT);
56.   pinMode(dcMotor3pin2, OUTPUT);
```

```
  pinMode(dcMotor4pin1, OUTPUT);
  pinMode(dcMotor4pin2, OUTPUT);
  pinMode(elevationStep, OUTPUT);
  pinMode(elevationDirt, OUTPUT);
  pinMode(circularStep, OUTPUT);
  pinMode(circularDirt, OUTPUT);
  pinMode(ultraSonic1Echo, OUTPUT);
  pinMode(ultraSonic1Trig, OUTPUT);
  pinMode(ultraSonic2Echo, OUTPUT);
  pinMode(ultraSonic2Trig, OUTPUT);
  pinMode(ultraSonic3Echo, OUTPUT);
  pinMode(ultraSonic3Trig, OUTPUT);
  pinMode(ultraSonic4Echo, OUTPUT);
  pinMode(ultraSonic4Trig, OUTPUT);

  tiltControl.attach(Servopin);
  tiltControl.write(0);               //making sure servo is at zero angle on startup

  DHT dht(dht11pin, DHTTYPE) ;
  dht.begin()                         //ask dht 11 sensor to start transmission of fata

  pinMode(mq4pin, INPUT);
  pinMode(mq7pin, INPUT);
  pinMode(mq135pin, INPUT);
}

void loop() {                          //infinite loop of MCU
  //functions to control different aspect of robot
  objectDetection();
  Motion();
  temp_humid();
  EnvSensor();
}

void objectDetection()                  //object detection sensor mechanis
{
  //making all the trigger pins of all sensors LOW simultaneously
  digitalWrite(ultraSonic1Trig, LOW);
  digitalWrite(ultraSonic2Trig, LOW);
  digitalWrite(ultraSonic3Trig, LOW);
  digitalWrite(ultraSonic4Trig, LOW);
  Delay(2);
  //making all the trig pins HIGH simultaneously and then with a 10ms delay make the LOW again
      digitalWrite(ultraSonic1Trig, HIGH);
      digitalWrite(ultraSonic2Trig, HIGH);
      digitalWrite(ultraSonic3Trig, HIGH);
      digitalWrite(ultraSonic4Trig, HIGH);
      Delay(10);
      digitalWrite(ultraSonic1Trig, LOW);
      digitalWrite(ultraSonic2Trig, LOW);
      digitalWrite(ultraSonic3Trig, LOW);
      digitalWrite(ultraSonic4Trig, LOW);
      //time the pulses received from all the Echo pins and convert them into distances (cm)
      dist1object = pulseIn(ultraSonic1Echo, HIGH)*0.034/2;
      dist2object = pulseIn(ultraSonic2Echo, HIGH)*0.034/2;
      dist3object = pulseIn(ultraSonic3Echo, HIGH)*0.034/2;
      dist4object = pulseIn(ultraSonic4Echo, HIGH)*0.034/2;
      //transmits all the data serially to the Rasp Pi
      Serial.write(dist1object);
      Serial.write(dist2object);
      Serial.write(dist3object);
      Serial.write(dist4object);
    }

```

```
121.    void Motion()                           //all of the robotic motions function
122.    {
123.      if (Serial.available() > 0) {        //check to see if user asked any function to perform
124.        receivedChar = Serial.read();
125.        newData = true;
126.      }
127.      int cmd = (receivedChar - '0');
128.
129.      while(newData == true)
130.      {
131.        if(cmd == 1)                          //forward chassis motion
132.        {
133.          if (dist1object > 10)               //making sure object is 10cm away from Sensor 1
134.          {
135.            digitalWrite(dcMotor1pin1, HIGH);
136.            digitalWrite(dcMotor1pin2, LOW);
137.            digitalWrite(dcMotor2pin1, HIGH);
138.            digitalWrite(dcMotor2pin2, LOW);
139.            digitalWrite(dcMotor3pin1, HIGH);
140.            digitalWrite(dcMotor3pin2, LOW);
141.            digitalWrite(dcMotor4pin1, HIGH);
142.            digitalWrite(dcMotor4pin2, LOW);
143.            Delay(5000); // 5sec delay
144.          }
145.        }
146.        else if(cmd == 2)                     //reverse chassis motion
147.        {
148.          if (dist2object > 10)               //making sure object is 10cm away from Sensor 2
149.          {
150.            digitalWrite(dcMotor1pin1, LOW);
151.            digitalWrite(dcMotor1pin2, HIGH);
152.            digitalWrite(dcMotor2pin1, LOW);
153.            digitalWrite(dcMotor2pin2, HIGH);
154.            digitalWrite(dcMotor3pin1, LOW);
155.            digitalWrite(dcMotor3pin2, HIGH);
156.            digitalWrite(dcMotor4pin1, LOW);
157.            digitalWrite(dcMotor4pin2, HIGH);
158.            Delay(5000); // 5sec delay
159.          }
160.        }
161.        else if(cmd == 3)                     //left chassis motion
162.        {
163.          if(dist3object > 10)                //making sure object is 10cm away from Sensor 3
164.          {
165.            digitalWrite(dcMotor1pin1, LOW);
166.            digitalWrite(dcMotor1pin2, LOW);
167.            digitalWrite(dcMotor2pin1, HIGH);
168.            digitalWrite(dcMotor2pin2, LOW);
169.            digitalWrite(dcMotor3pin1, LOW);
170.            digitalWrite(dcMotor3pin2, HIGH);
171.            digitalWrite(dcMotor4pin1, LOW);
172.            digitalWrite(dcMotor4pin2, LOW);
173.            Delay(5000); // 5sec delay
174.          }
175.        }
176.        else if(cmd == 4)                     //right chassis motion
177.        {
178.          if(dist4object > 10)                //making sure object is 10cm away from Sensor 4
179.          {
180.            digitalWrite(dcMotor1pin1, HIGH);
181.            digitalWrite(dcMotor1pin2, LOW);
182.            digitalWrite(dcMotor2pin1, LOW);
183.            digitalWrite(dcMotor2pin2, LOW);
184.            digitalWrite(dcMotor3pin1, LOW);
```

```
            digitalWrite(dcMotor3pin2, LOW);
            digitalWrite(dcMotor4pin1, LOW);
            digitalWrite(dcMotor4pin2, HIGH);
            Delay(5000); // 5sec delay
          }
        }
        else if(cmd==5)                        //stop all the movements of robot
        {
          digitalWrite(dcMotor1pin1, LOW);
          digitalWrite(dcMotor1pin2, LOW);
          digitalWrite(dcMotor2pin1, LOW);
          digitalWrite(dcMotor2pin2, LOW);
          digitalWrite(dcMotor3pin1, LOW);
          digitalWrite(dcMotor3pin2, LOW);
          digitalWrite(dcMotor4pin1, LOW);
          digitalWrite(dcMotor4pin2, LOW);
          digitalWrite(elevationDirt, LOW);
          digitalWrite(elevationStep, LOW);
          digitalWrite(circularStep, LOW);
          digitalWrite(circularDirt, LOW);
        }
        else if (cmd ==6)                      //elevation vertically up
        {
          if (elevDist != 300)                 //making sure the Al block hasnt reached the top
          {
            digitalWrite(elevationDirt, HIGH);  //stepper motor dir so to make camera move up
            for (int x =0; x< 1000; x++)        //1000 pulses (= 1cm up dist)
            {
              digitalWrite(elevationStep, HIGH);
              delayMicroseconds(800);
              digitalWrite(elevationStep, LOW);
              delayMicroseconds(800);
            }
            elevDist++;                          //vertical distance couter inc
          }
        }
        else if (cmd ==7)                       //elevation vertically down
        {
          if (elevDist != 0)                   //making sure block hasn't reached the bottom
          {
            digitalWrite(elevationDirt, LOW);   //stepper motor dir so to make camera move down
            for (int x =0; x<1000; x++)
            {
              digitalWrite(elevationStep, HIGH);
              delayMicroseconds(800);
              digitalWrite(elevationStep, LOW);
              delayMicroseconds(800);
            }
            elevDist--;
          }
        }
        else if (cmd ==8)                       //circular motion clockwise
        {
          if (circDist ! = 720)                 //making sure robot dont make 2 rot in +ve dir
          {
            digitalWrite(circularDir, HIGH);     //stepper motor clockwise direction
            for (int x =0; x < 5; x++)           //5 pulse (=9 degrees)
            {
              digitalWrite(circularStep, HIGH);
              delayMicroseconds(800);
              digitalWrite(circularStep, LOW);
              delayMicroseconds(800);
            }
            circDist = circDist + 5;              //circular angle counter
```

```
249.          }
250.        }
251.        else if (cmd ==9)                          //circular motion anticlockwise
252.        {
253.          if(circDist != -720)                     //so robot dont make 2 rot in -ve dir
254.          {
255.            digitalWrite(circularDir, LOW);        //stepper motor anti clockwise direction
256.            for (int x =0; x < 5; x++)
257.            {
258.              digitalWrite(circularStep, HIGH);
259.              delayMicroseconds(800);
260.              digitalWrite(circularStep, LOW);
261.              delayMicroseconds(800);
262.            }
263.            circDist = circDist - 5;
264.          }
265.        }
266.        else if (cmd == 10)                        //servo motion positive
267.        {
268.          if (tiltangle != 90)                     //if angle hasnt crossed the 90 degree limit
269.          {
270.            for (int angleT = tiltangle; angleT < tiltangle + 15; angleT++) //15 deg +ve turn
271.            {
272.              tiltControl.write(angleT);
273.              delay(15);
274.            }
275.            tiltangle = tiltangle + 15;            //tilt counter
276.          }
277.        }
278.        else if (cmd ==11)                         //servo motion negative
279.        {
280.          if (tiltangle != -90)                    //if angle hasnt crossed the -90 deg limit
281.          {
282.            for (int angleT = tiltangle; angleT > tiltangle - 15; angleT--) //15 deg -ve turn
283.            {
284.              tiltControl.write(angleT);
285.              delay(15);
286.            }
287.            tiltangle = tiltangle - 15;
288.          }
289.        }
290.        else     //if no user input is detected, make sure no movement is done
291.        {
292.          digitalWrite(dcMotor1pin1, LOW);
293.          digitalWrite(dcMotor1pin2, LOW);
294.          digitalWrite(dcMotor2pin2, LOW);
295.          digitalWrite(dcMotor2pin1, LOW);
296.          digitalWrite(dcMotor3pin1, LOW);
297.          digitalWrite(dcMotor3pin2, LOW);
298.          digitalWrite(dcMotor4pin1, LOW);
299.          digitalWrite(dcMotor4pin2, LOW);
300.          digitalWrite(elevationDirt, LOW);
301.          digitalWrite(elevationStep, LOW);
302.          digitalWrite(circularDirt, LOW);
303.          digitalWrite(circularStep, LOW);
304.        }
305.          newData = false;
306.      }
307.    }
308.
309.    void temp_humid()                          //tep & humidity sensor function
310.    {
311.      float humidity = dht.readHumidity() ;  //reading humidity value from dht11
312.      float temp = dht.readTemperature() ;   //reading temp value from dht11
```

```
313.      serial.write(humidity);
314.      serial.write(temp);
315.    }
316.
317.    void EnvSensor()                          //Environmental sensor function
318.    {
319.      int methaneVal = analogRead(mq4pin);    //reading analogue val from MQ4 sensor
320.      int COval= analogRead(mq7pin);          //reading analogue val from MQ7 sensor
321.      int CO2Val = analogRead(mq135pin);      //reading analogue val from MQ135 sensor
322.      //trasnmitting these values to raspberry pi
323.      serial.write(methaneVal);
324.      serial.write(COval);
325.      serial.write(CO2val);
326.    }
```

Annex: B

HTML Interface Code for Raspberry Pi

```
1.  <!---HTML CODE FOR USER INTERFACE IN RASPBERRY PI-->
2.  <!DOCTYPE html PUBLIC "-
    //W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
3.  <html xmlns="http://www.w3.org/1999/xhtml">//
4.  <head>
5.   <title>Surveillance Robot</title>          <!---Browser Tab Title-->
6.   <!---importing strobe media libraries-->
7.   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></scrip
    t>
8.   <script type="text/javascript" src="assets/js/date_time.js"></script>
9.   <script type="text/javascript" src="assets/js/reload.js"></script>
10.  <script type="text/javascript" src="strobe/lib/swfobject.js"></script>
11.  <link rel="stylesheet" type="text/css" href="assets/css/style.css">
12.  <script type="text/javascript">
13.         //flash media layer
14.        // Create a StrobeMediaPlayback configuration
15.        var parameters = {
16.            src: "rtmp://" + window.location.hostname + "/flvplayback/myStream",
17.            autoPlay: true,
18.            controlBarAutoHide: false,
19.            playButtonOverlay: true,
20.            showVideoInfoOverlayOnStartUp: false,
21.            optimizeBuffering : false,
22.            initialBufferTime : 0.1,
23.            expandedBufferTime : 0.1,
24.            minContinuousPlayback : 0.1,
25.            poster: "strobe/images/poster.png"
26.        };
27.
28.       // Embed the player SWF:
29.       swfobject.embedSWF
30.       ( "strobe/StrobeMediaPlayback.swf"
31.        , "strobeMediaPlayback"
32.        , 600        //resolution of video player
33.        , 400
34.        , "10.1.0"
35.        , {}
36.        , parameters
37.        , { allowFullScreen: "true"}
38.        , { name: "strobeMediaPlayback" }
39.        );
40.
41.    </script>
```

```
42.
43. </head>
44.
45. <body>
46.    <!---PHP Serial Communication Setup-->
47.   <?php
48.   include '/var/www/html/php_serial.class.php';
49.   $serial = new phpSerial();
50.   $serial->deviceSet("/dev/ttyACM0");       <!---Arduino Serial Name-->
51.   $serial->confBaudRate(9600);              <!---baud rate-->
52.   $serial->deviceOpen();
53.   ?>
54.
55.
56. <!----------------------------------------->
57.
58. <div  align="Center">   <!---Title of Project-->
59.   <div class="title"><span style="color:#2e3238;font-
    size40:px; "><u>Adjustable Surveillance Camera Robot</u></span></div>
60. </div>
61. <div  align="center">   <!---Team members of Project-->
62.     <div class="developed-by"><span style="color:#2e3238;font-
    size:30px; ">Final Year Project by Muhammad Hamza & Abdul Hadi Zahid<br></span></div>
63. </div>
64.
65. <!---div for display time Start-->
66. <div align="right">
67.  <div class="date-time"> <span id="date_time"></span></div>
68.             <script type="text/javascript">window.onload = date_time('date_time');</scr
    ipt>
69. </div>
70. <!---div for display time End-->
71.
72. <!---div for display environment data starts-->
73. <div class="my-grid-container fix">
74.   <!--div-1 open-->
75.   <div class="my-grid-item">
76.    <p><u><b>Environment Data</b></u></p>
77.         <label for="fname">Temperature (*C):</label><br>            <!---temp-->
78.         <input type="text" id="fname" name="fname"  value=$serial->readMessage()><br>
79.         <label for="fname">Relative Humidity (%):</label><br>       <!---humidity-->
80.         <input type="text" id="fname" name="fname"  value=$serial->readMessage()><br>
81.         <label for="fname">CO Concentration (ppm):</label><br>      <!---CO-->
82.         <input type="text" id="fname" name="fname"  value=$serial->readMessage()><br>
83.         <label for="fname">CO2 Concentration (ppm):</label><br>     <!---CO2-->
84.         <input type="text" id="fname" name="fname"  value=$serial->readMessage()><br>
85.         <label for="fname">CH4 Concentration (ppm):</label><br>      <!---CH4-->
86.         <input type="text" id="fname" name="fname"  value=$serial->readMessage()><br>
87.         </div>
88.     <!--div-1 Close-->
89.
90.     <!--div-2 open [VIDEO]-->
91.   <div class="my-grid-item">
92.         <p><u><b>Live-Video Stream</b></u></p>
93.         <div id="strobeMediaPlayback">
94.   </div>
95.   </div>
96.     <!--div-2 Close-->
97.
98.     <!--div-3 open [Distance from Obj]-->
99.   <div class="my-grid-item">
100.         <p><u><b>Distance From Object</b></u></p>
101.               <label for="fname">Front (cm):</label><br>                    <!---
    Front dist-->
```

```
                <input type="text" id="fname" name="fname"  value=$serial-
    >readMessage()><br>
                <label for="fname">Back (cm):</label><br>                        <!---
    Back dist-->
                <input type="text" id="fname" name="fname"  value=$serial-
    >readMessage()><br>
                <label for="fname">Left (cm):  </label><br>                      <!---
    Left dist-->
                <input type="text" id="fname" name="fname"  value=$serial-
    >readMessage()><br>
                <label for="fname">Right (cm):</label><br>                       <!---
    Right dist-->
                <input type="text" id="fname" name="fname"  value=$serial-
    >readMessage()><br>
                </div>
            <!--div-3 Close-->
        </div>

        <!--Placing Buttons on Interface-->
        <table class="table table-bordered table-dark" align="center">
         <tbody>
            <tr>
              <th scope="col">
                  <div id="buttons">
                  <center>
                    <form method="get" action="index.html">         <!--
    Negative Tilt Button-->
                          <button type="submit" name="Negative" value="Negative" style="m
    argin:10px"><img src="assets/images/negative.png" width="40" height="40"></button>
                    </form>
                    </center>
              </div>
              </th>

              <th scope="col">
                  <div id="buttons">
                  <center>
                    <form method="get" action="index.html">             <!--
    Anticlockwise movement Button-->
                          <button type="submit" name="Anticlockwise" value="Anticlockwise"
     style="margin:10px" ><img src="assets/images/anticlock.png" width="40" height="40"></b
    utton>
                    </form>
                    </center>
              </div>
              </th>

              <th scope="col">
                  <div id="buttons">
                  <center>
                    <form method="get" action="index.html">             <!--
    Down movement Button-->
                          <button type="submit" name="Down" value="Down" style="margin:10p
    x" ><img src="assets/images/downTail.png" width="40" height="40"></button>
                    </form>
                    </center>
              </div>
              </th>

              <th scope="col">
                <div id="buttons">
                  <center>
                    <form method="get" action="index.html" >                <!--
    Chassis Buttons-->
```

```
                    <button type="submit" name="Forward" value="Forward" style="marg
in:10px" ><img src="assets/images/forward.png" width="40" height="40"></button>
                    <br>
                    <button type="submit" name="Left" value="Left" style="margin:10p
x" ><img src="assets/images/left.png" width="40" height="40"></button>
                    <button type="submit" name="StopC" value="Stop" style="margin:10
px" ><img src="assets/images/stop.png" width="40" height="40"></button>
                    <button type="submit" name="Right" value="Right" style="margin:1
0px" ><img src="assets/images/right.png" width="40" height="40"></button>
                    <br>
                    <button type="submit" name="Reverse" value="Reverse" style="marg
in:10px" ><img src="assets/images/backword.png" width="40" height="40"></button>
                  <br>
              </center>
                  </form>
            </div>
            </th>

            <th scope="col">
                <div id="buttons">
                <center>
                  <form method="get" action="index.html">         <!--
up movement Button-->
                    <button type="submit" name="Up" value="Up" style="margin:10px"
><img src="assets/images/upTail.png" width="40" height="40"></button>
                  </form>
                  </center>
            </div>
            </th>

            <th scope="col"><div id="buttons">
                <center>
                  <form method="get" action="index.html">         <!--
clockwise movement Button-->
                    <button type="submit" name="Clockwise" value="Clockwise" style="
margin:10px" ><img src="assets/images/clock.png" width="40" height="40"></button>
                  </form>
                  </center>
            </div>
            </th>

            <th scope="col">
                <div id="buttons">
                <center>
                  <form method="get" action="index.html">         <!--
Neg Tilt movement Button-->
                    <button type="submit" name="Positive" value="Positive" style="ma
rgin:10px" ><img src="assets/images/positive.png" width="40" height="40"></button>
                  </form>
                  </center>
            </div>
            </th>
          </tr>
        </tbody>
       </table>

      <?php                                    <!--PHP Data Transmission-->
          if(isset($_GET['Forward'])){          <!--Forward: 1-->
              $serial->sendMessage("1");
          }
          else if(isset($_GET['Reverse'])){     <!--Reverse: 2-->
              $serial->sendMessage("2");
          }
          else if(isset($_GET['Left'])){        <!--Left: 3-->
```

```
                $serial->sendMessage("3");
            }
            else if(isset($_GET['Right'])){         <!--Right: 4-->
                $serial->sendMessage("4");
            }
            else if(isset($_GET['Stop'])){          <!--Stop: 5-->
                $serial->sendMessage("5");
            }
            else if(isset($_GET['Up'])){            <!--Up: 6-->
                $serial->sendMessage("6");
            }
            else if(isset($_GET['Down'])){          <!--Down: 7-->
                $serial->sendMessage("7");
            }
            else if(isset($_GET['Clockwise'])){     <!--Clockwise: 8-->
                $serial->sendMessage("8");
            }
            else if(isset($_GET['Anticlockwise'])){     <!--Anti clockwise: 9-->
                $serial->sendMessage("9");
            }
            else if(isset($_GET['Positive'])){          <!--+ve tilt: 10-->
                $serial->sendMessage("10");
            }
            else if(isset($_GET['Negative'])){          <!---ve tilt: 11-->
                $serial->sendMessage("11");
            }
        ?>
        </body>
        </html>
```

Annex: C

CSS Style Code for Raspberry Pi

```
/* STYLE CSS CODE FOR USER INTERFACE ON RASPBERRY PI*/
html,body {
  font: 400 0.875rem/1.5 "Open Sans", sans-serif;
  margin: 0;
  padding: 0;
  background-color: #eee;
}
.title {            /*setting the style for the Project Title*/
  margin: 0;
  vertical-align: top;
  text-align: center;
  font-weight: bold;
  font-size: 50px;
  color: #2e3238;
  text-transform: none;
  display: inline-block;
}
.developed-by {     /*setting the style for the Project Team Members names*/
  margin: 0;
  vertical-align: top;
  text-align: center;
  font-size: 50px;
  color: #2e3238;
  text-transform: none;
  display: inline-block;
}
.date-time {      /*setting the style for the date-time shown*/
  margin: 0;
```

```
  vertical-align: top;
  text-align: center;
  font-size: 20px;
  color: #2e3238;
  text-transform: none;
  display: inline-block;
}
.my-grid-
container { /*setting the style for the grid container that holds the data & video*/
  display: grid;
  grid-template-columns: auto auto auto;
  background-color: #FFFFFF;
  padding: 10px;
}
.my-grid-item {          /*setting the style for the grid container data */
  background-color: rgba(255, 255, 255, 0.8);
  border: 1px solid rgba(0, 0, 0, 0.8);
  padding: 20px;
  font-size: 25px;
  text-align: center;
}
```