Thant Zin Htoo Paing
(2020009)

Design and implementation of a hexapod robot for search, rescue, and surveillance operations

BEng(Hons) Electronics System Engineering
Undergraduate Thesis Report
Department of Computer Science & Technology

Dr. Pyone Ei Ei Shwe
2021 – 2022

Abstract

This project's main goal is to develop a prototype version of the hexapod that can be implemented in search and rescue operations. Robotics in search and rescue operations has become a necessity in recent years due to the increasing casualties in emergency workers. The hexapod can serve as a replacement for wheeled robots since it has more flexibility and stability. In this project, a hexapod robot with 3 degrees of freedom per leg was developed. The hexapod also has a camera attached on the front; streaming video to the operator. The camera vision of the robot will allow the operator to search the survivors in a disaster site and also help the operator to control the hexapod from its point of view when it's out of the line of sight. The hexapod can also act as a surveillance robot by be setting it in a standby state in a location.

Keywords

Hexapod, multi-legged robot, search and rescue, tripod gait, surveillance, raspberry pi

Acknowledgments

This project could not be possible to finish in time without the help of the supervisor and the lectures. Supervisor Dr. Pyone Ei Ei Shwe gave a lot of advice in writing this thesis report, and how to approach and develop the project step by step. Moreover, I would like to thank lecturer Dr. Thu Htay Aung for pointing me in the right direction during the development process of this project. Also because of his lectures, I was able to write the report in a way that would simplify things for the readers. And lastly and most importantly, I would also like to thank my parents who gave financial support for this project and were mentally supportive during the hard times of the development process. Additionally, I would also like to show my gratitude to our English lecturer Daw Aye Thandar Oo for proofreading the thesis; without her help, this document would be filled with careless and overlooked mistakes.

Contents

Chapter 1: Introduction

1.1 Problem Statement and Background

As technology evolves, most of the tasks done by humans are being replaced by robots. Since the early 1960s, robotics has become a big part of the industrial processes, agriculture, military use, and even incrementally taking part in our everyday lives (Krishna, *et al*. 2014). Especially the tasks that require high precision or the ones that have a high-risk factor on the human operators. One particular task that has moderate to high danger levels for humans is search and rescue operations. So, in the case of the said operations in disaster sites, robots are needed in order to save time and reduce risks. Therefore, various types of robots are used in different operations depending on the disaster sites.

In general, disaster robots can be subdivided into 3 categories: wheeled type, airborne type, and legged type. However, each type has its own strengths and weaknesses. Wheeled-type robots such as RC cars and tanks are good robots for going over smooth or inclined surfaces but they cannot go over obstacles and need extra sensors and algorithms to avoid obstacles. Airborne types such as drones vary in sizes and shapes. The pros of it is that it can go over a site at very high altitudes but they cannot fly close to the ground nor they can go into small confined spaces. Legged robots such as biped, quadruped, hexapod, are good for going over obstacles and getting into small confined spaces; however, the more legs the robot has, the harder it is to control them. Still, the advantages of the legged robots can outweigh the disadvantages if the correct control methods and algorithms are applied to them.

Hexapod robots have 6 legs and they are more stable and can bear more load than the same sized quadrupeds. The hexapod robots can be more efficient in going over obstacles if they were to be programmed with position control and stabilization algorithms.

1.2 Introduction to the project

The overall purpose of the project is to develop a prototype hexapod robot that has a decent walking gait with added functions that are useful for search and rescue operations. The major part of this project is the hexapod which is an alternate option from other types of robots and acts as the project's fundamental. The added feature is the camera which provides visual feedback to the operator. However, to utilize the camera, a microprocessor is used. And to drive the servo motors, a microcontroller is utilized. Therefore, connection and data transmission between the microprocessor and microcontroller is necessary to develop this project as a whole.

1.3 Introduction to the artefact

This proposed project focuses on the core aspects of the hexapod robot which is being able to move with 6 legs and go over some obstacles. The gait algorithm used in the hexapod allows the hexapod to achieve fast, accurate, and precise movements. Adding a camera that can constantly stream video feedback is also a useful feature for controlling the hexapod while the robot is out of the user's sight. Since the robot has 6 legs (3 servo motors in each) algorithms for each leg were coded independently. The camera system was also integrated into the robot with minimal buffering and lags (depends on interferences and network strength). The chassis of the hexapod's body and legs are 3D printed with carbon fiber infused filament for better strength while being lightweight.

1.3.1 Working sequence of the artefact

The working process of the hexapod robot is simple. The main purpose of the robot is to navigate through areas, locate the remaining survivors and give information to the human operator. The working sequence of the robot in a disaster site is shown in the below figure.

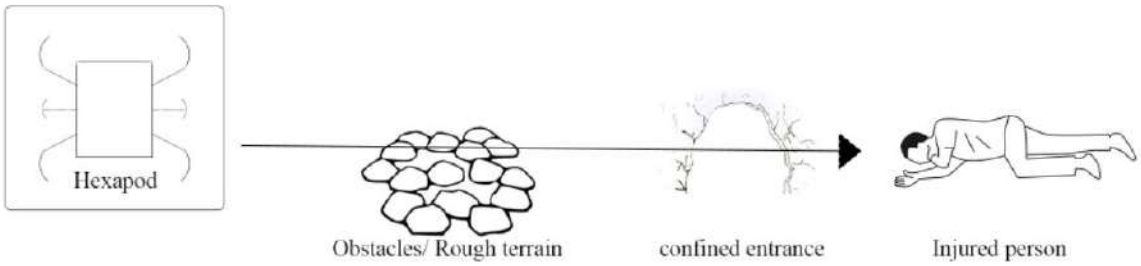


Figure (1.3.1.1) Working sequence of the hexapod robot in a disaster site

Firstly, the robot will be initiated by the operator and will stay on standby awaiting commands. The video streaming process will also be initiated as soon as the robot starts. The operator has to input directional commands for the robot. The robot can go in four directions: forward, reverse, left, and right. The robot's leg can go over small obstacles along the way and it can also fit into openings that are bigger than 6 inches in height and 15 inches in length. The robot is made to search survivors through the camera feedback which the driving operator can and will be able to see. If the robot was out of the operator's vision, there wouldn't be much of a problem because of the camera system.

1.4 Aim and objectives

The aim of this project is to aid the search and rescue workers in their operations. Reducing the risks upon them and saving precious time that would be used in searching for survivors. The objectives of this project are the tasks that were done throughout the development and are considered as mid to high tier level of impact.

The objectives are as follows;

- Literature review
- Developing a functional walking hexapod
- Incorporating the camera system
- Communication between RPi and Arduino
- Artifact testing and performance analysis

Chapter 2: Literature Review

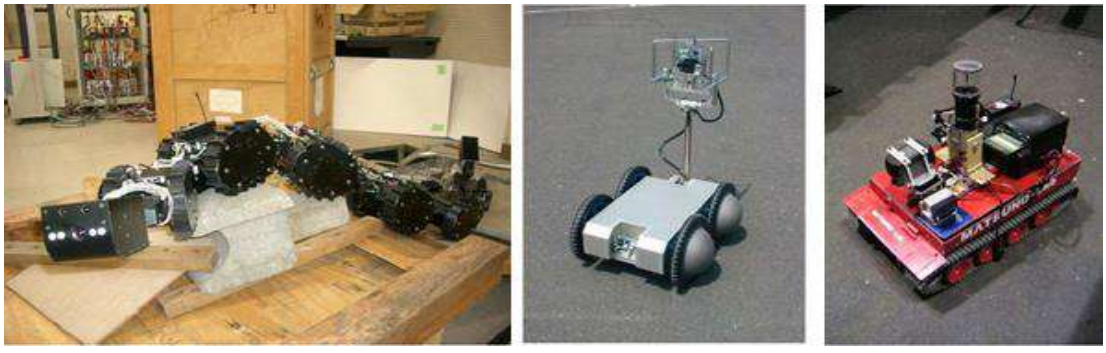
In this section, the literature review on different kinds of search and rescue robots and how hexapods fit into that category and different kinds of control methods and algorithms will be explained and how some of the findings and methods will be applied to this project.

2.1 Review on Search and Rescue Robots

Natural disasters have been an occurring problem since the beginning of history. Earthquakes, typhoons, cyclones, etc., are some of the many natural disasters that occur frequently throughout the world. Before the advancements of technology, disaster responses have been just policemen, firefighters, and medical workers being deployed. In doing so, the emergency workers have to risk their lives when they enter a disaster site (Marais, et al. 2016). According to Shah and Choset (2004), the first 48 hours are the most critical for finding victims with a higher chance of survivability. However, to be able to do that, either a large number of workers will be required or a lot of energy will be used from the existing workers.

To solve and minimize the problem of human labor and risk, robots that are dedicated to search and rescue operations were designed and developed. The robots can aid in reducing the risk for the human workers and speed up the process of searching for the victims. Robots can be valuable assets to the search and rescue operations; by going into collapsed and confined areas after an earthquake, withstanding chemical and biological hazards, etc. (Shah and Choset, 2004). Depending on the disaster situations, different variations of robots can be used. The types of disaster response robots can be generalized into 2 main groups; aerial robots and ground robots (Matsuno and Tadokoro, 2004).

Both groups have their unique abilities and features to aid in search and rescue processes. Aerial robots such as drones overviewing a disaster site and searching survivors from bird's-eye view. On-ground robots such as wheeled robots and track robots that are equipped with different kinds of sensors, crawler-type robots for going over obstacles, legged robots for more stability and speed than crawler robots.



(a) *(b)* *(c)*

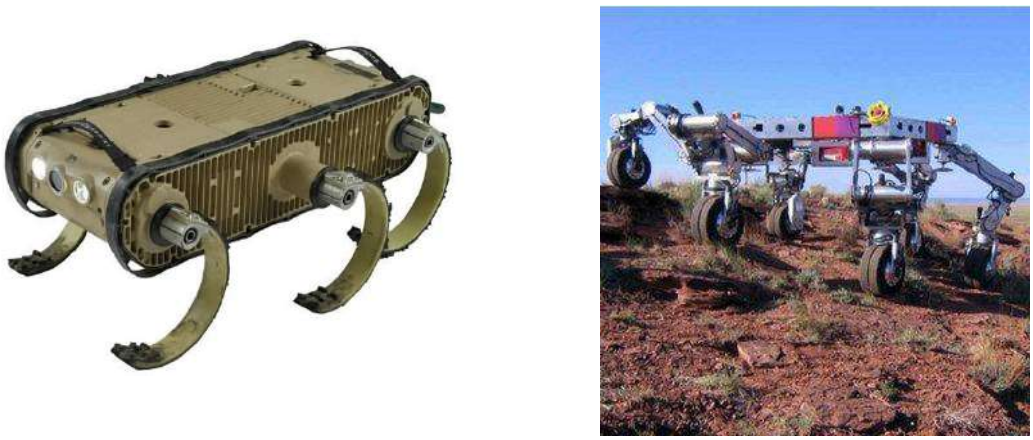
Figure (2.1) (a) Snake Type robot KOUGA (b) Wheel type robot FUMA (c) Tank/ Track type robot MA1

Although crawler and legged robots may have more potential than the wheel and track-based robots, they are less commonly used due to their complex development processes especially legged robots such as hexapods. Nevertheless, if the hexapods were to be equipped with simple and decent algorithms, they can be extended broadly into the search and rescue category.

2.2 Review on Hexapod Robot Types

Many tech companies have been following the trend of robotics for a few years now. Boston Dynamics, Intuitive, iRobot, Universal Robots, etc. are a few of the biggest robotics companies as of right now. Boston Dynamics, one of the top leading robotics companies, has been producing numerous robots for both commercial use and industrial uses. Tesla, which is one of the largest automobile companies very recently announced Tesla Bot, a humanoid robot that is run by the same AI used in their cars and intended to navigate traffic and plan routes.

There are various ways to classify different kinds of walking robots – by body shape, leg numbers, degrees of freedom in each leg, or locomotion technique (Moore and Buehler, 2001 cited in Zak and Rozman, 2015). Generally speaking, the hexapod robot is a 6-legged robot inspired by arachnids and insects such as spiders, scorpions, ants, etc. The hexapods are designed to further improve the stability of biped and quadruped robots (Wikipedia, 2021). There are multiple hexapods with different leg and body designs. Such as RHex hexapod with 6 C-shaped legs that rotate in a wheel-like motion (Saranli, et al. 2001 cited in Tedeschi and Carbone, 2014) and ATHLETE (Hauser, et al. 2008 cited in Tedeschi and Carbone, 2014) with wheels on the feet of each leg meant for steep and rough terrains. Different kinds of hexapods are designed and developed to suit specific needs in certain situations. E.g., space exploration, rough terrain exploration, etc.



(a) *(b)*

Figure (2.2) (a) RHex hexapod robot (b) ATHLETE hexapod robot

Hexapod robots can also be generally differentiated into 2 groups according to their body shapes: circular/hexagonal shapes and rectangular shapes. In circular or hexagonal shapes, the legs are evenly distributed. In a rectangular shape, two groups of 3 legs are distributed on

different sides of the rectangular shape. According to Čížek, Zoula, and Faigl (2021), circular/ hexagonal-shaped hexapods can better support the bodyweight due to the equal distribution of the legs however, the rectangular-shaped hexapods can move faster in terms of going forward. For a legged robot to walk properly, 2 degrees of freedom are needed at the minimum (Zak and Rozman, 2015). Most of the hexapods have 3 DOF and some even more. According to Čížek, et al. (2021), more DOF in the hexapod will improve its maneuverability but additional actuators/motors increase the power consumption and weight. In addition, more DOF means more complexity therefore, for keeping the complexity at a minimum and at the same time maintaining a smooth flow of walking gait, 3 DOF are mostly chosen as the ideal design for the hexapods.

No matter the types of body and degrees of freedom, if the hexapod has a bad and unsynchronized walking gait, it will not be able to perform very well. Therefore, even though hardware components and shapes matter, the software integration (coding) of the hexapod's walking gait and control algorithms are also important aspects.

2.3 Review on Hexapod Walking Gaits and Algorithms

Movement is one of the most important tasks for a mobile robot. It may seem basic and simple but in practice, there are a lot of aspects to consider such as which direction the legs are moving with which degrees and speed, etc. Controlling one leg is complex enough and may take some time but in the case of hexapods, there are six legs to be controlled. Therefore a properly calculated and synchronized walking gaits and methods are needed for making the hexapod's movement smooth and steady.

Zak and Rozman (2015) explain that a walking gait means the locomotion of the robot achieved by the movement of its legs. One of the methodologies for hexapod movement is to move according to a pre-planned gait (Priandana, et.al .2017). In hexapod's walking gait, 2 basic gaits can be classified: wave gait and tripod gait. There are also many other gaits but wave and tripod are the two main gaits that are most commonly used among hexapods. However, each gait has its own pros and cons.

Wave gait is one of the simplest gaits for legged robots. In wave gait, only one leg moves at a time in sequential order. The one advantage of the wave gait is that it has very high stability due to the hexapod's 5 legs being present and supporting the weight of the robot body at all times. However, the major disadvantage of the wave gait is that; due to the nature of only moving one leg at a time, the hexapod moves very slowly and the movement is not exactly straight either (Krishna, et al. 2014). Figure 2.3 below is the timing diagram presentation of the wave gait of the hexapod.

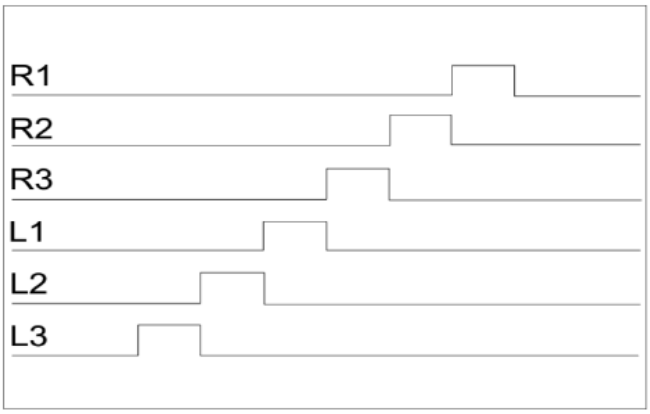


Figure (2.3) Timing Diagram for Wave gait

In tripod gait, the 6 legs of the hexapod are grouped into 2 groups; each group having 3 legs. In the first steps of the tripod gait, the first group of 3 legs will move forward while the other 3 remain stable. After the first step cycle, the second group of legs will move forward while the first group moves back to their initial positions. Both leg groups perform the same movement but the time period between the 2 groups is shifted by half a period (Zak and Rozman, 2015). Tripod gait provides faster movement but compared to the wave gait, it has less stability because only 3 legs can support the whole body weight during each movement cycle.

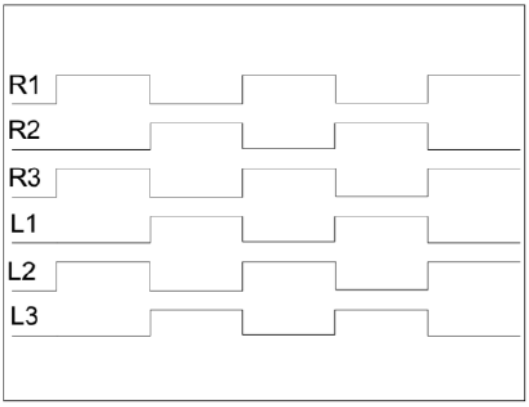


Figure (2.4) Timing Diagram for Tripod gait

Comparing the two gaits, although tripod gait is slightly less stable than the wave gait, most hexapods use tripod as their main choice of gait because of the benefit of going faster and doesn't sacrifice much of the stability. Some other hexapod models are equipped with both tripod and wave gait to be able to offer the choice depending on different surfaces.

As for the movement, a method called inverse kinematics is often used in hexapods. Priandana, et.al (2017) defines that inverse kinematics as the method of calculating each servo's angle by considering the mechanical design and dimensions of the hexapod. The hexapod's leg is composed of 3 individual parts: coxa, femur, and tibia; named after the exoskeleton parts of the arthropods. Each of those 3 individual parts contributes to calculating the inverse kinematics of the hexapod robot. By combining the inverse kinematics calculations and the walking gait, the hexapod will be able to move smoothly and efficiently.

Chapter 3: Hexapod Design and Implementation

This chapter contains the decisions and choices made after doing literature reviews; mainly about the hexapod's design and how the hardware was implemented. The designing part will explain about the choices made in components, body parts design in detail. The implementation part will explain the steps done in hardware assembly, changes made along the way, the programming methods used, and the final decisions for each functionality.

Before getting into details about the design and implementation steps of the project, a general understanding of the project is needed. Therefore, a flow chart of the hexapod will be provided to achieve an understanding of the hexapod's general working flow.

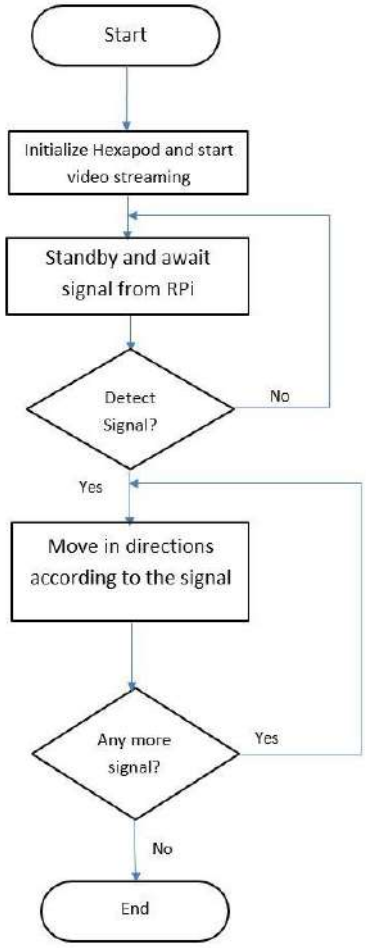


Figure (3.1) General Flow Chart of the Hexapod

As the flow chart goes, the hexapod will start and set the legs to their initial positions one by one. At the same time, the video streaming process from the camera will also be initiated. The legs will stay at the standby position as long as no directional keys are pressed by the user. When the Arduino in the hexapod receives the directional command from the RPi, it will drive the servo drivers according to the commands. After the hexapod has moved according to the user commands, the hexapod will continue to await further signals. If there are no further incoming signals, the hexapod will end its process.

On further explanation of how the hexapod moves according to which signals, the figure (3.2) below will show the byte data and which corresponds to which direction.

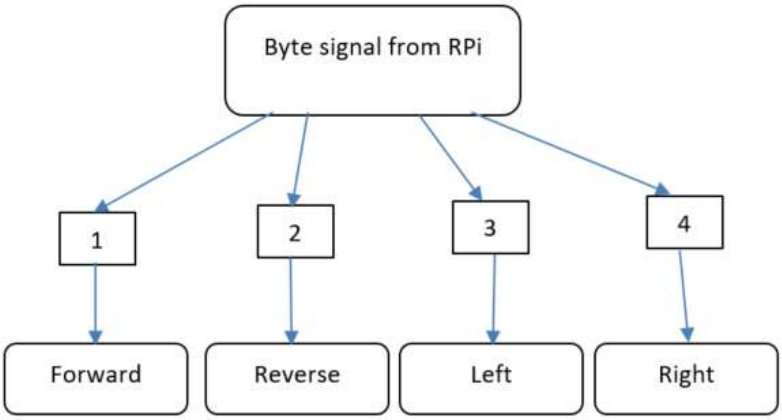


Figure (3.2) Block diagram of the corresponding signals

As seen in figure (3.2), according to the byte signal sent from the RPi to the hexapod, the movement and the direction of the hexapod change.

3.1: Design of the hexapod

After doing the literature reviews, the main decisions made can be divided into 2 categories: theoretical choices and hardware choices. Theoretical choices were DOF per leg and choice of walking gait. The hardware choices were microcontrollers, servo motors, camera, and body part materials, etc.

3.1.1: Theoretical Choices

For the theoretical aspect of the hexapod, the first thing to be considered is the degrees of freedom (DoF) per leg. As the hexapod has six legs, choosing DoF is one of the important aspects of the hexapod's behaviour. As discussed in the literature review section, at least 2 DoF must be present for the hexapod to have decent movement. More DoF means more fluidity in movement however, it would mean more complexity in controlling, algorithms, and the added weight of the extra servo motors. Therefore, 3 DoF is chosen as the ideal way of hexapod's movement.

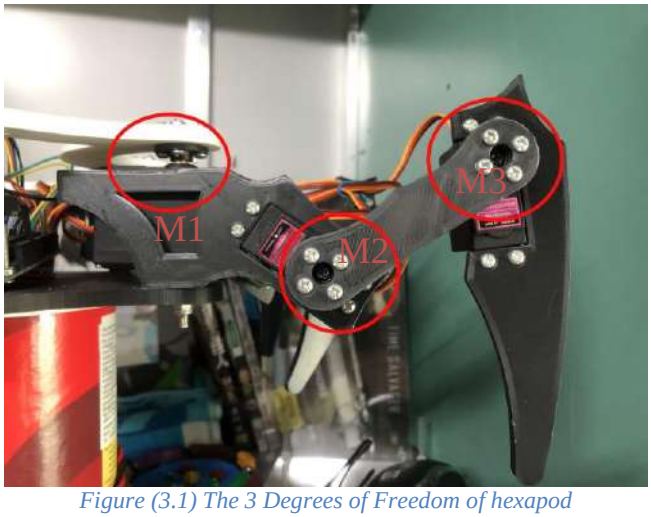


Figure (3.1) The 3 Degrees of Freedom of hexapod

For the second theoretical consideration, the walking gait is the highest of importance for the hexapod. As previously done in the literature review, there are 2 basic and general walking gaits: wave gait and tripod gait. To summarize the explanation from the literature review, the wave gait is the most stable; however, it is significantly slower. On the other hand, a tripod gait is less stable than a wave but it is faster and moves more accurately. By comparing the pros and cons of the two methods, sacrificing a bit of stability for a faster and more accurate walking gait is acceptable. Therefore, tripod gait is chosen as the main method of traversing for the hexapod.

3.1.2: Hardware Choices

As the hardware goes, the main components used in the hexapod are- the microcontroller, servo motors, servo motor drivers, camera, power supply, and lastly the body chassis. After doing some extensive research on the internet about hexapod projects, Arduino Mega was initially decided to be the main controller for the hexapod. However, the Arduino Mega cannot utilize camera vision therefore, Raspberry Pi was added to fully utilize the streaming process with the camera. Later on during the development, the microcontroller was changed from Arduino Mega to Arduino UNO due to a problem that will be explained in the implementation section. Since the hexapod has 3 DoF per leg and a total of 6 legs, the amount of servo motors required is 18. The amount of torque required for each servo can be calculated as follows.

$$T = \Sigma_{i=1}^{n} W_i D_i$$

Where,

$$T = Torque W_i = Weight\ of\ indvidual\ object D_i$$
$$= Distance\ of\ object\ from\ center\ of\ mass$$

For the calculation of servo motor torque, the second servo in the middle is chosen to be calculated because it is the one that will hold the most weight.

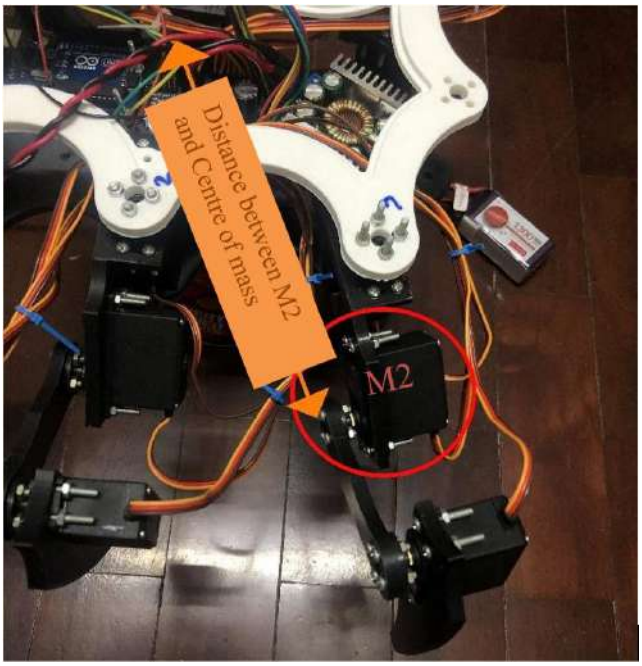


Figure (3.1.2.1) View of hexapod leg indicating the distance

$$Total\ hexapod\ weight = 2.28\ kg Weight\ acting\ on\ one\ leg = \frac{2.28}{6}$$

$$= 0.38\ kg\ Distance = 20cm$$

$$Torque = \ 0.38kg * 20cm = 7.6kg/cm$$

Therefore, the torque required for a servo is 7.6kg/cm which can also be applied to the other 2 servos on the different joints. The available servo in the market that can cover the requirement of 7.6kg/cm is MG996R which can hold from 9kg/cm with (4.8V) to 11kg/cm (6V).

For driving the chosen servo motors, the Arduino is capable of driving them individually. Arduino MEGA can drive up to 48 servo motors, however, the specific Arduino Shield required for MEGA to drive the servo motors wasn't available in the market and another reason for wanting to simplify the coding process, a PWM driver (PCA9685) is used. One driver can control 16 servos however, the drivers can link to another driver via the I2C method and control more than 16 servos.

For this project, two PWM drivers were used to drive 18 servos. On driving the 18 servos, an external power is needed. Therefore, a 3 cell Li-Po battery of 11.1V 1300mAh was used. The Li-Po can power both the PWM drivers and the Arduino. To power the RPi, the same Li-Po can be used but as the Raspberry Pi 4 used in this project can only be powered by USB-C, it causes a lot of complications to diverge power from the Li-Po battery. However, a power bank (5V 3A) with a USB-C cable can be used to power the RPi with the drawback being an extra weight on the robot. On the topic of powering up from the Li-Po Battery, the PWM driver cannot be given power from the battery directly. Since the battery is 11.1V it needs to be stepped down to 5V to supply power. As for the case of Arduino, the supply voltage can be from (5V-

12V) but driving the servo motors can draw up amperes depending on the load of the servos. Therefore, a DC-DC voltage step-down module that can supply 12A was used to supply both the Arduino and PCA9685.

As for the body parts of the hexapod, there were 3 considerations according to the literature reviews and the researches. The options were aluminum, acrylic, and lastly, 3D printed parts. To briefly explain the 3 options, the first one, aluminum is light in weight and strong but hard to work with and not readily available in the market. The second one, acrylic, is dense and strong although heavier and also hard to work and not easily accessible. The last option is 3D printed parts which can be strong depending on the print settings, lightweight, and easily accessible, if anything happened or doesn't work perfectly, it can be reworked and printed again easily. How the 3D printed parts are taken and assembled will be further explained in the implementation section.

3.2 Implementation of the Hexapod

The implementation stage of the hexapod will be explained in this section step-by-step. First of all, to make it simple to understand, this section will be broken down into sub-sections. The sub-sections are as follows; Body parts, Arduino, The Raspberry Pi, and The Final Integration. The pre-planned implementation for this project is to start by writing code for the main walking mechanisms on Arduino for the hexapod. After that, the camera, which will be used accordingly with the raspberry pi will be tested and worked. After that, sending data from the RPi to the Arduino is done. However, before code writing is started on the Arduino, the legs need to be assembled first to the body so that the servo motors can be set and calibrated accordingly. By doing that, we can easily test the leg movements and change the code during the Arduino code writing process.

3.2.1 The body parts

To use the 3D printed parts for the hexapod, firstly the 3D models are needed. Initially, it was planned to draw the 3D models from the ground up but doing so would require a lot of time and skills and time for this project is very limited and tight to draw complex 3D models. Therefore, the 3D models were taken from an open-sourced hexapod project which luckily had the same servo motor slots and threads used in this project. So, the 3D models were taken from that open source project and then commissioned to a 3D printing service shop. It took over a week for the parts to arrive due to the current political conditions in this country. Assembling the legs with the servo motors were started as soon as the parts arrived. Unfortunately, one of the joints broke off due to an error during assembly and another set of joints had to be printed again. Therefore, it took a while to get all the parts and assemble them to grasp the initial condition of the hexapod.

3.2.2 Arduino

To make this project easier to work on, the first approach was writing the walking gait on the Arduino. The Arduino code was written in C++ language. During the initial considerations for the hexapod, the walking gait was going to be coded with inverse kinematics for determining the degrees of the leg joint angles. However, to write a code with inverse kinematics would be time-consuming and would require a lot of troubleshooting since it involves complex mathematical equations and computing. Also, the servo motors used in this project are average cheap servo motors that don't always have the right torque and don't always land on the exact position so, using inverse kinematics would be unfit for both software-wise and hardware-wise. Therefore, to make the hexapod move in simple ways, a pre-fixed movement value set was used.

To explain how the movement of the hexapod in this project works, one leg of the hexapod will be taken as an example. The leg of the hexapod has to go through 2 phases: The swing phase and the stance phase. The swing phase is when the leg is lifted from the ground and rotates in the direction the hexapod wants to move. For e.g., if the hexapod wants to move forward, the leg will rise up and rotate forward. And then the leg will drop down to touch with the ground, completing the stance phase.

The next phase which is the stance phase is when after the hexapod is done lifting, rotating, and touching to the ground, the leg will rotate back to the initial position and drag the body to the desired direction. In this way, the hexapod moves in the desired direction.

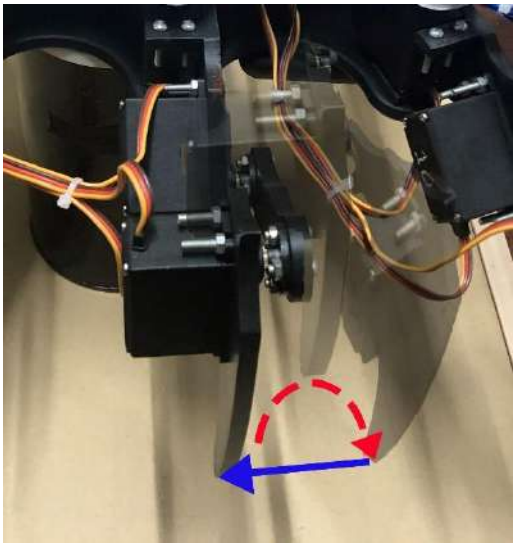


Figure (3.2.2.1) The two phases (Red = swing phase, Blue = stance phase)

Due to the nature of the servo motors used for this project, each servo motor's positions are not the same. Therefore, before writing code for the tripod gait, calibrations for each servo were done to fine-tune the positions and find the legs' initial positions.

Now, the walking code of the leg will be explained in detail. First, we create some initial values. These are the values of stating leg conditions. In the code, the value names are given according to i(n)L(n). i(n) indicate the number of stage and L1, L2 indicates the leg group.

```
int i1L1 = 0;
int i2L1 = 0;
int i3L1 = 0;
int i4L1 = 0;

int i1L2 = 0;
int i2L2 = 0;
int i3L2 = 0;
int i4L2 = 0;

int m = 0;

boolean l1status = LOW;
boolean l2status = LOW;
byte inputData = 0;
```

(a)

```
void moveLeg1() {
  // Swign phase - move leg though air - from initial to final position
  // Rise the leg
  if (i1L1 <= 10)
  {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(55 + i1L1 * 2));
    driver.setChannelPWM(2, pwmServo.pwmForAngle(15 + i1L1 * 3));
    i1L1++;
  }
  // Rotate the leg
  if (i2L1 <= 30) {
    driver.setChannelPWM(0, pwmServo.pwmForAngle(60+i2L1));
    i2L1++;
  }
  // Move back to touch the ground
  if (i2L1 > 20 & i3L1 <= 10) {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(75 - i3L1 * 2));
    driver.setChannelPWM(2, pwmServo.pwmForAngle(45 - i3L1 * 3));
    i3L1++;
  }
```

(b)

Figure (3.2.2.2) (a) Initial values (b) Swing phase code

As shown in figure (b), this portion of the code is the swing phase which was explained briefly earlier. Now the limit for i1L1 was set to 10 to raise the leg for 10 extra degrees from the initial position. After that rotating the leg is done by setting the limit of i2L1 to 30. Finally, for the swing phase, the leg has to move back to touch the ground. For that, the leg will start going down until the limit of i2L1 reaches 30 (20 in this case) and i3L1 is set to 10 for it to match with the i1L1's limit.

```
// Stance phase - move leg while touching the ground
// Rotate back to initial position
if (i2L1 >= 30) {
  driver.setChannelPWM(0, pwmServo.pwmForAngle(90 - i4L1));
  i4L1++;
  l1status = HIGH;
}
// Reset the counters for repeating the process
if (i4L1 >= 30) {
  i1L1 = 0;
  i2L1 = 0;
  i3L1 = 0;
  i4L1 = 0;
}
```

Figure (3.2.2.3) Stance Phase code

For the stance phase, which happens after the leg has touched the ground, the leg will start to rotate back to the initial position. For that, the limit for i2L1 was used again, subtracting 30 degrees from the rotated position. After the leg has finished rotating, the status for leg group 1 will be set as HIGH and the leg conditions will be reset back to 0. In this way, the leg moves by utilizing 2 phases. The other legs were coded in a similar fashion making all 6 legs work uniformly.

Now, after explaining each leg's functionality, the next to be explained is how the tripod gait works. Tripod gait earned its name because it separates the 6 legs into 2 groups which contain 3 legs. Therefore, when the hexapod moves, 3 legs hold the body weight while the other 3 moves. Hence, the tripod gait was named.

```
//Forward
if (m == 1) {
  moveLeg1();
  moveLeg3();
  moveLeg5();
  if (l1status == HIGH) {
    moveLeg2();
    moveLeg4();
    moveLeg6();
  }
}
```

(a)

```
//Reverse
if (m == 2) {
  moveLeg1Rev();
  moveLeg3Rev();
  moveLeg5Rev();
  if (l1status == HIGH) {
    moveLeg2Rev();
    moveLeg4Rev();
    moveLeg6Rev();
  }
}
```

(b)

```
//Left
if ( m == 4) {
  moveLeg1();
  moveLeg3();
  moveLeg5Left();
  if (l1status == HIGH){
    moveLeg2();
    moveLeg4Left();
    moveLeg6Left();
  }
}
```

(c)

```
//Right
if ( m == 3) {
  moveLeg1Right();
  moveLeg3Right();
  moveLeg5();
  if (l1status == HIGH){
    moveLeg2Right();
    moveLeg4();
    moveLeg6();
  }
}
```

(d)

Figure (3.2.2.4) (a) Tripod Forward (b) Tripod reverse (c) Tripod Left (d) Tripod Right

The codes shown in figure () indicate the tripod gait for four directions of the hexapod. They all work in 2 groups of 3 legs. After moving the first 3 legs, the code will check the status of the first group and if that passes, it will cycle to the second group. The altering between 2 leg groups is what makes the tripod gait fast yet stable at the same time.

The coding process of the legs was done by coding and testing the algorithm on each leg. Also, the initial tests were done on a heightened platform so that the legs wouldn't bear the body's load and can move freely.

Figure (3.2.2.5) One leg testing

After testing each leg, one side of the hexapod was tested with one PCA9685 driver. Afterward, a second PCA9685 was linked with the first one via the I2C method to drive the remaining side of the hexapod. Finally, both sides were tested for their leg movements.

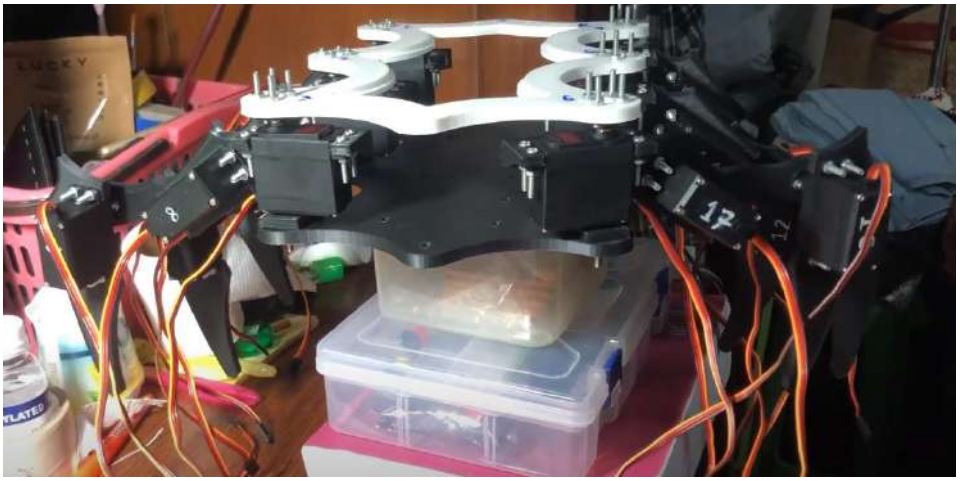
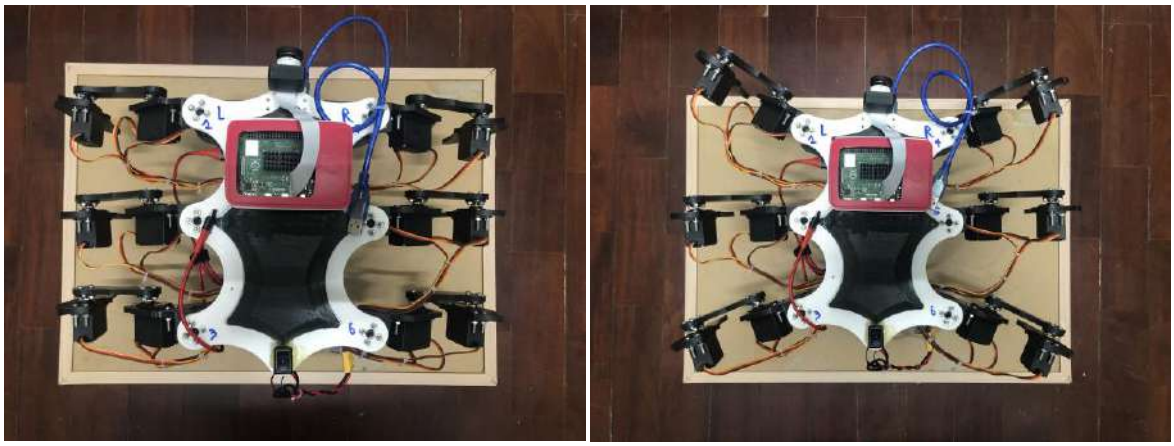


Figure (3.2.2.6) Both sides test

Although the initial tests were done fine on the heightened platform, it needs to be able to walk on the ground. Therefore, on-ground tests were needed to be conducted. Unfortunately, one error occurred during on-ground testing. The error was that the hind legs weren't able to carry the weight. This was due to the legs not being distributed evenly and causing the backside of the hexapod to topple. Hence, the initial positions of the hexapod's legs are one of the important aspects of the hexapod. They need to be distributed evenly so that they can carry the body's weight evenly.



(a) *(b)*

Figure (3.2.2.7) (a) Initial leg distributions (b) Final leg distributions

3.2.3 Raspberry Pi

The Raspberry Pi used in this project is the Raspberry Pi 4 Model B with 2GB of RAM. RPi 4 is powerful enough for this project since the only requirement is to use the camera and give commands to the Arduino. RPi 3 might have been the optimal usage for it but at the time of buying components, RPi 4 was the only model available in the market.

RPi is a microprocessor which is basically a small computer the size of a credit card. For it to be fully utilized, a micro SD card with Raspbian OS installed is needed. Raspbian is a Debian-based OS for RPi. It changes its OS version around every 2-year mark. During the time of working for this project's RPi section, the Raspbian OS was changed to the newest OS called

Bullseye. The new OS changed the major library for the pi-camera which impacted majorly on Pi-Camera functions. Since the OS update was released on 30th October 2021, tutorials and references needed for the new library were almost to none. Therefore, adapting to the new OS system and its camera library would be extremely tedious and hard to do. As a solution, an old Raspbian OS called Buster had to be installed for it to work properly with the reference codes and the camera.

The camera chosen for this project was the Pi Camera Ver1.3 with 5 megapixels. To work with the pi camera, an open-sourced camera streaming code called pi-camera-flask was used. The way video streaming works in this project is by utilizing motion JPEG. Motion JPEG works by streaming individual JPEG images to the HTML page. But to use the HTML page with the python code, a proper bridge is needed. Therefore, Flask, which acts as a medium between Python and HTML page and also supports motion JPEG. Since the streaming is via Flask, it can be viewed on any devices that is connected to the same network. For e.g., if the RPi's IP address is '192.168.1.10', the address for viewing the live stream is '192.168.1.10:5000'. The address can be put on any device that has an internet browser and view from that browser. Hence, making it convenient and easily accessible. The slight drawback of viewing the camera is that depending on the internet strength of the RPi the lag may differ across multiple devices during the stream.

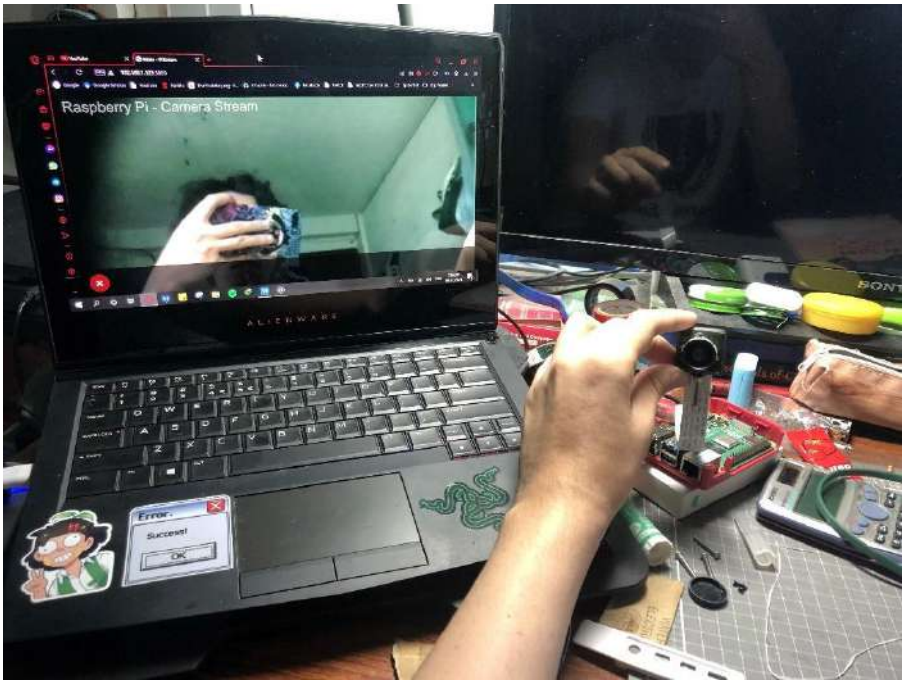


Figure (3.2.3.1) Camera stream test

Another important task for the RPi is to send directional commends to the Arduino which then will move the hexapod according to the directional commends given by RPi. There are 2 main methods of sending commands from RPi to Arduino. The first method is through the I2C method and the second method is through the serial connection. Both methods were tried during the process of this project.

For testing out the I2C method, an Arduino UNO was used with 4 LEDs on a breadboard as stand-in substitutes for the 4 directional movements. The I2C method connects the Arduino and RPi through GPIO pins. In this case, RPi is the master and Arduino is the slave hence, no logic converters were required. The stand-alone testing was successful and can get the Arduino UNO to light up the LEDs accordingly. However, one problem occurred during integrating the system to the hexapod. In the hexapod, the Arduino MEGA was already using an I2C port for driving the motors where the Arduino is the master and the drivers are the slaves. For working with the RPi, the Arduino needs to be the slave and the RPi the master. After extensive research about this case, it is nearly impossible for the Arduino to act as the master and slave simultaneously. Therefore, the I2C connection method for communicating with the RPi was dismissed.

Since the previous method was rejected for its overlapping master-slave problem, a second method of using the serial connection had to be implemented. Serial communication is done by using the USB connecter of the Arduino and hooking it up to the RPi. The reason serial connection wasn't the first chosen method is that there is a problem where if the Arduino was unplugged from the RPi, the USB port name might change (e.g. ttyACM0 or ttyACM1). But since it is the only method remaining, the problem had to be accepted. As long as the Arduino is connected to the RPi, said problem won't occur.

```
if __name__ == '__main__':
    ser = serial.Serial('/dev/ttyACM0', 9600, timeout=1)
    ser.reset_input_buffer()
    while True:
        main()
```

Figure (3.2.3.2) Portion of the python code where the said problem could occur

The serial testing was also successful with a few minor changes in ways of sending byte data from the I2C method. The one advantage of a serial connection is that it is more stable than an I2C connection. In the I2C method, spamming commands would make the Arduino force disconnect from the RPi. However, the serial method will not be disconnected no matter how many commands were spammed.

Before trying to send commands from RPi to Arduino, giving commands from a user to RPi is needed to be considered first. Giving directional commands from the keyboard of a laptop to the RPi wirelessly was one of the optimal choices. The other choice is using a Bluetooth controller to send commands, but it shortens the range compared to the Wi-Fi option. To define the keypress of a keyboard and send it to RPi as a command, a keypress module code was needed to be written.

```
import pygame

def init():
    pygame.init()
    win = pygame.display.set_mode((100,100))

def getKey(keyName):
    ans = False
    for eve in pygame.event.get():pass
    keyInput = pygame.key.get_pressed()
    myKey = getattr(pygame,'K_{}'.format(keyName))
    if keyInput [myKey]:
        ans = True
    pygame.display.update()

    return ans
```

Figure (3.2.3.3) KeyPressModule python code snippet

For writing the code, a pre-installed library called “pygame” was imported and used. The function names used in the main definition were taken from the pygame library. The main definition getKey was written as shown in figure (3.2.3.3). Simply put, the function will get the keypresses from the keyboard and returns them to the code. This getKey function will be the main one that will be used in the RPi to Arduino serial python code.

```
import serial
import time
import KeyPressModule as kp

time.sleep(1)
kp.init()

def main():
    if kp.getKey('UP'):
        ser.write (b'1')
        time.sleep(0.1)
    elif kp.getKey('DOWN'):
        ser.write(b'2')
        time.sleep(0.1)
    elif kp.getKey('LEFT'):
        ser.write(b'3')
        time.sleep(0.1)
    elif kp.getKey('RIGHT'):
        ser.write(b'4')
        time.sleep(0.1)
    else:
        ser.write(b'0')
        time.sleep(0.1)
```

Figure (3.2.3.4) Serial connection code snippet

In the code for serial connection with the Arduino, firstly 3 libraries were imported. The 3 libraries are serial, time, and the previously written KeyPressModule. The Serial library is for the serial connection, time is for setting delays between so that it won’t overload the system, and the kp is the KeyPressModule for the keyboard commands. For the main definition, it is as simple as using the previously written getKey function and sending byte commands according to what key was pressed. In this project, the directional key commands were set as up, down, left, right directional keys. It can be set to any key such as WASD or any other keys the user wants. However, to keep the coding process simple and easy to understand, directional keys

were used. As seen in figure (3.2.3.4) the code was written in IF, ELSE condition loop. Therefore, if no keys are currently being pressed, it will keep sending the '0' byte to the Arduino which corresponds to the standby state of the hexapod. If any of the written keys were pressed, it will act according to the keyboard press and send the corresponding byte to the Arduino which will then drive the servo motors as the RPi command.

Before integrating the serial connection code to the main hexapod with Arduino, several adjustments were needed so that the RPi can be used in headless mode. Headless mode is where no external monitor, keyboard, or mouse is needed for using the RPi. For that, 2 conditions are necessary. Setting a static IP and setting up SSH. The raspberry pi's IP address changes every time the pi is re-booted. Therefore, setting a static IP will make things easier for using RPi in headless mode. The static IP chosen for this project is '192.168.1.123'. SSH is short for Secure Shell which basically works to control the RPi through its terminal from a host computer.As for the case of SSH, it was as easy as enabling it in the RPi configuration menu.

There are 2 popular ways to use the RPi in headless mode. The first way is with the VNC viewer. VNC is built-in with the RPi which only needs to be turned on through the RPi configuration menu. It enables the user to view and control the RPi's desktop remotely from a different computer that has VNC viewer installed. The second way is through PuTTY. PuTTY can only connect to the RPi's terminal and control through it. Setting up a static IP makes connecting with both the VNC and PuTTY easier since they have to connect with RPi through its IP address.

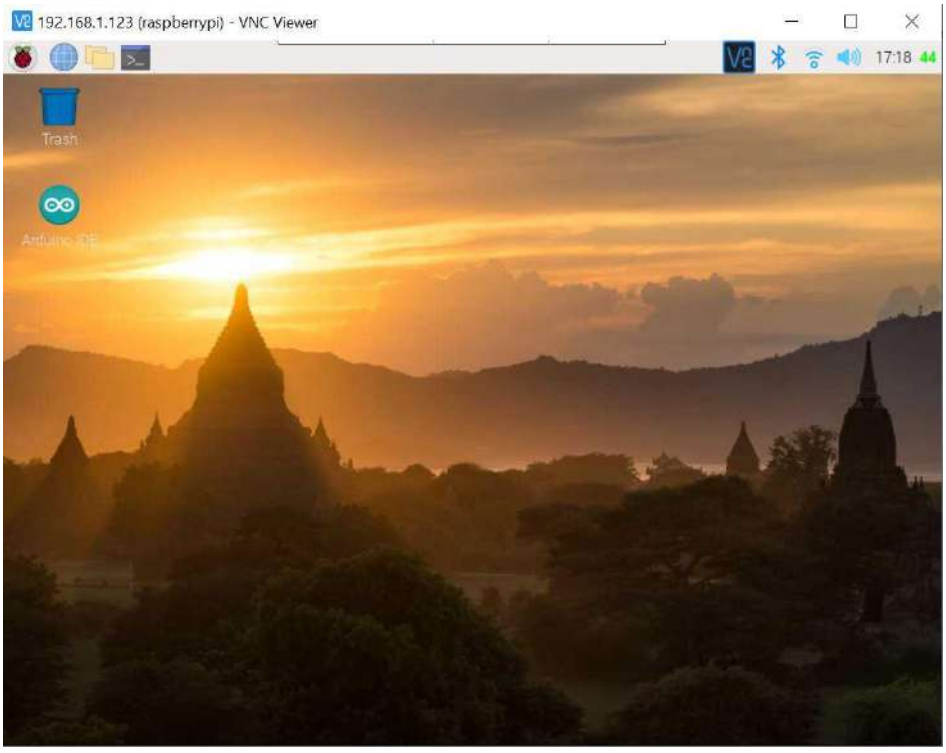


Figure (3.2.3.5) Desktop view of RPi through VNC viewer

3.2.4 The Integration (Final Assembly)

After finishing the respective steps for this project, namely the body parts, Arduino, and the Raspberry Pi, the last thing to do was combine them all and make the hexapod whole.

Before placing the electronic components on the hexapod body, the RPi needs to be connected to the hexapod's Arduino MEGA. Unfortunately during the assembly, an error occurred. That being Arduino MEGA could not get the data from RPi. After testing between the Arduino MEGA and UNO back and forth, the said error was only happening with Arduino MEGA. Searching about it on the internet through forums didn't give satisfying answers. Since the Arduino MEGA on the hexapod only uses the I2C port, exchanging it to UNO was an easy fix for the current error. Therefore, the main Arduino used in this project changed from MEGA to UNO with no changes in the code.

The servo motors were all attached with the 3D printed parts. All that's left to do is place the Arduino, PWM drivers, RPi, camera, buck converter, Li-Po battery, and power bank accordingly and neatly on the body. Since the hexapod model used in this project was taken from an open-source project, the insides of the hexapod were small to fit all the necessary components. Therefore, placing and weight distribution of the components were necessary. The Arduino, the 2 drivers, and the buck converter were placed inside the hexapod. The battery and the power bank were attached to the bottom side of the hexapod with velcros for easy removal when recharging them. However, the RPi did not fit inside the hexapod since the insides were all crammed with jumpers and wires. Therefore, the RPi was put on the top of the hexapod over the cover plate. The camera was put in a case made for the pi camera V1.3. A slot for screwing on the smartphone camera lens was also put. The purpose of the smartphone camera lens is for enhancing the field-of-view of the pi camera. The hexapod must be suspended on a heightened platform when it's not in use so that the legs won't stress out and wear the servo motors.

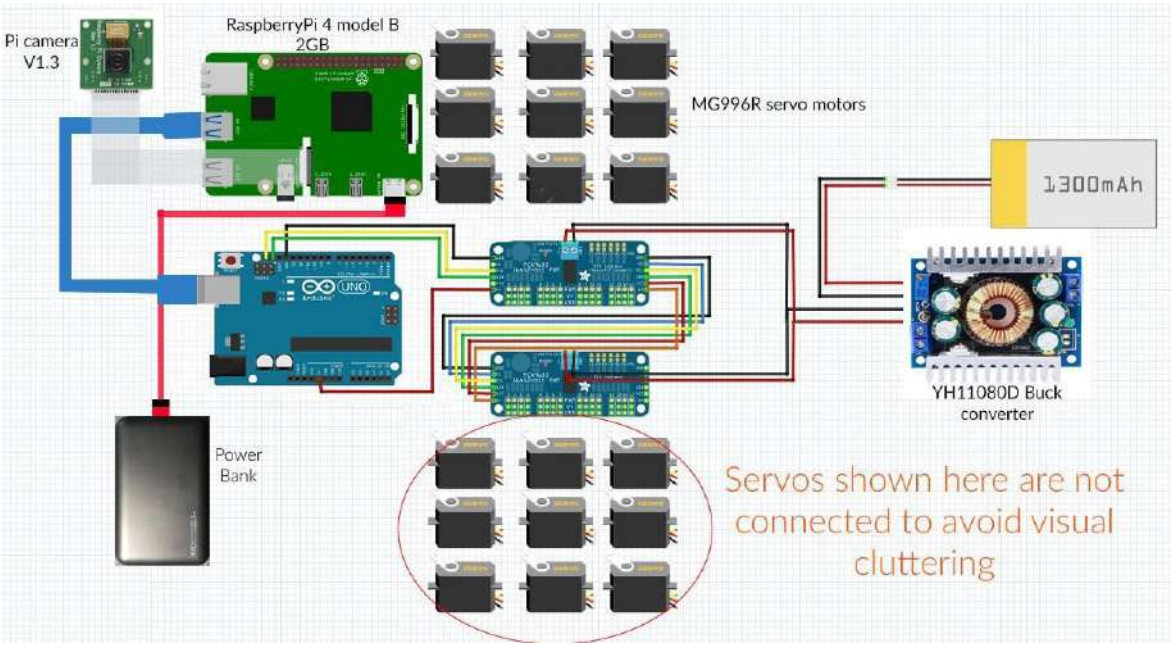


Figure (3.2.4.1) Overall schematic of the project

Figure (3.2.4.1) visualizes the overall graphical schematic of this project after all the implementations had been done.

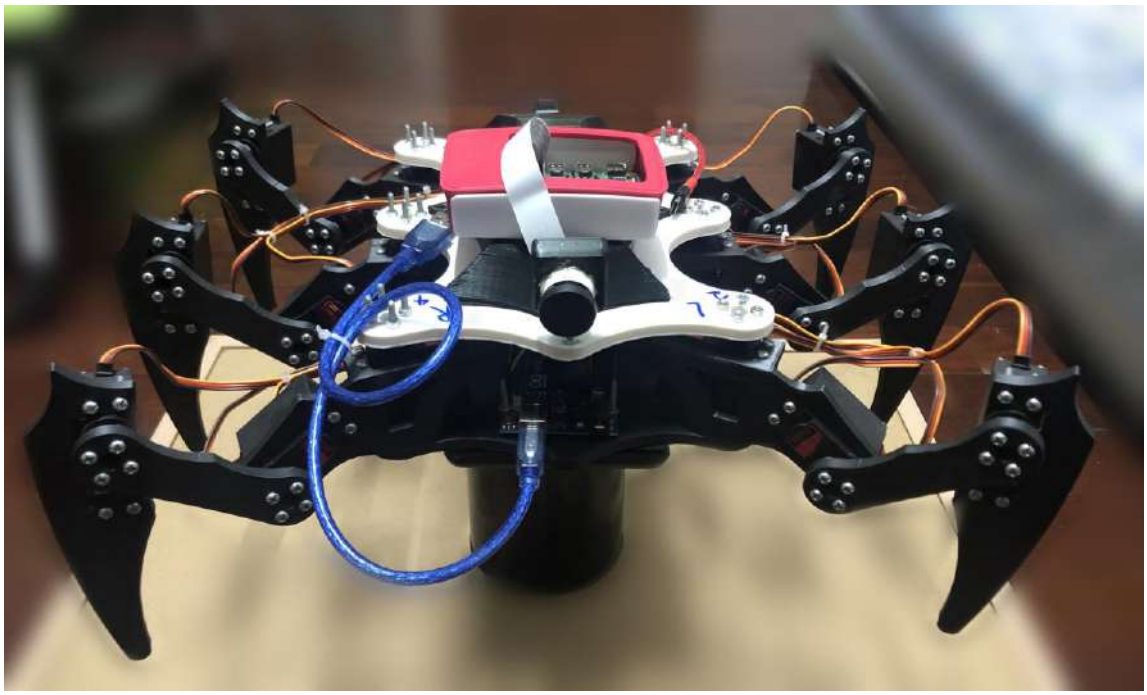


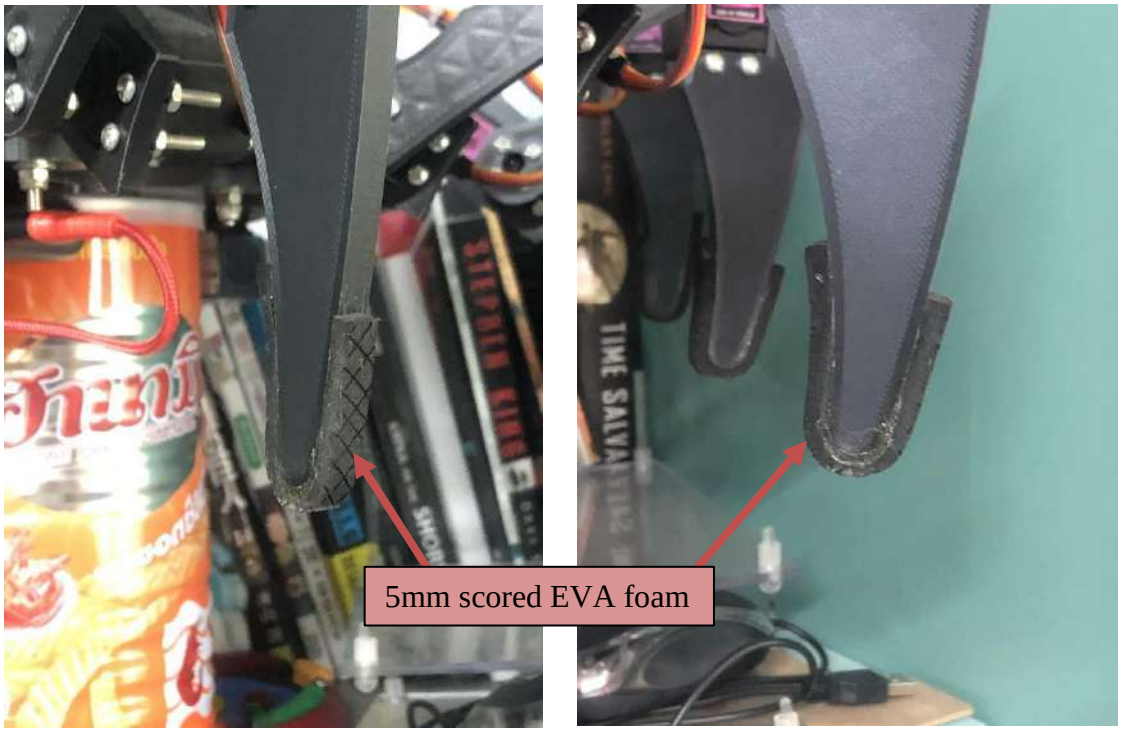
Figure (3.2.4.2) The final hexapod

Figure (3.2.4.2) shows the finalized look of the hexapod after all the implementations said in this chapter were completed.

Chapter 4: Testing and Performance Analysis

This chapter will continue after implementing the choices made in the previous chapter. The hexapod will be tested and analyzed for its performance in different situations, surfaces and discussions on the performances will be made along the way.

To analyze the performance and find out the potential of the hexapod, multiple tests were conducted. Firstly, the hexapod was tested for its fundamental function which is walking. The hexapod's body parts were 3D printed with PLA material therefore, its feet are also plastic. Hence, testing the hexapod for its movement on the ground would sometimes lose its grip due to the smooth plastic nature of its feet. To accommodate that problem, a set of foam paddings were glued onto the hexapod's feet. The material for the padding was 5mm EVA foam with grooves cut for improved grip.



(a) *(b)*

Figure (4.1) (a), (b) View of the foam padding on the hexapod's feet

The added foam padding improved the stability of the hexapod during movement. The video link of the walking test with foam paddings will be provided below.

Walking test with foam feet:
https://drive.google.com/file/d/1wR6Lzv52OkazWARKRbm84YAcFHDJ8bgn/view?usp=sharing

However, a problem was found during the test. The problem being the hexapod has a slight sway when it goes in a straight direction. The deviation is roughly about 10~15% from

the original path. The sway in direction is not consistent every time the test was conducted. Troubleshooting for the problem and calibrating the servo degrees several times, it is found that the problem was due to the servo motors. As the servo motors are cheap copy versions, their degrees may vary every time they move. Fortunately, the problem is a small inconvenience that can be overlooked, and also the direction can be adjusted manually by the user with directional keypresses.

As the second test for the hexapod's performance, the test of time will be conducted. The test of time is that the time that the hexapod will take during traversing from point A to point B. The time taken to travel between points A and B is directly proportional to the speed of the hexapod. In the Arduino code of the hexapod, the default walking speed for the hexapod is 30. However, the speed can be adjusted according to the user's preferences. One thing to keep in mind is that with more speed the hexapod traverses, the more stability it will have to sacrifice.

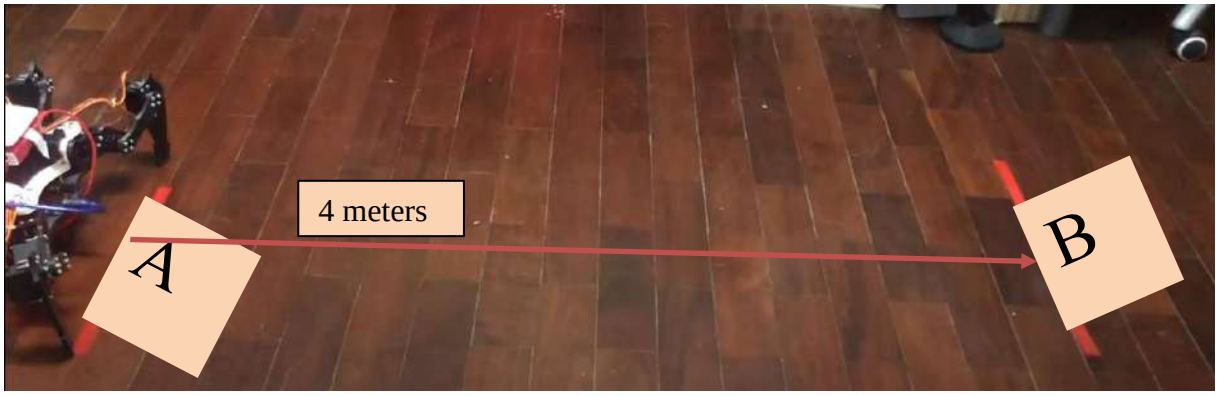


Figure (4.2) Walking speed test from point A to B

Video link for the figure (4.2) - https://drive.google.com/file/d/1MDqMiCexPQIirrkc-TkcsXsN8lMQTqPz/view?usp=sharing

From the video, point A and point B are 4 meters apart from each other. And the time the hexapod took to travel from A to B was 45 seconds. Therefore, by calculation, the hexapod was moving with a speed of 0.089m/s. The speed can be adjusted however, as said above, more speed means less stability.

Another important aspect of this project is the camera vision. The camera needs to have the least lag as much as possible. Since the camera stream is via Flask with motion JPEG, there is always going to be a delay. The delay depends on the strength of the Wi-Fi and its range of it. As far as the tests go, the delay goes around 1~2 seconds depending on the Wi-Fi strength.

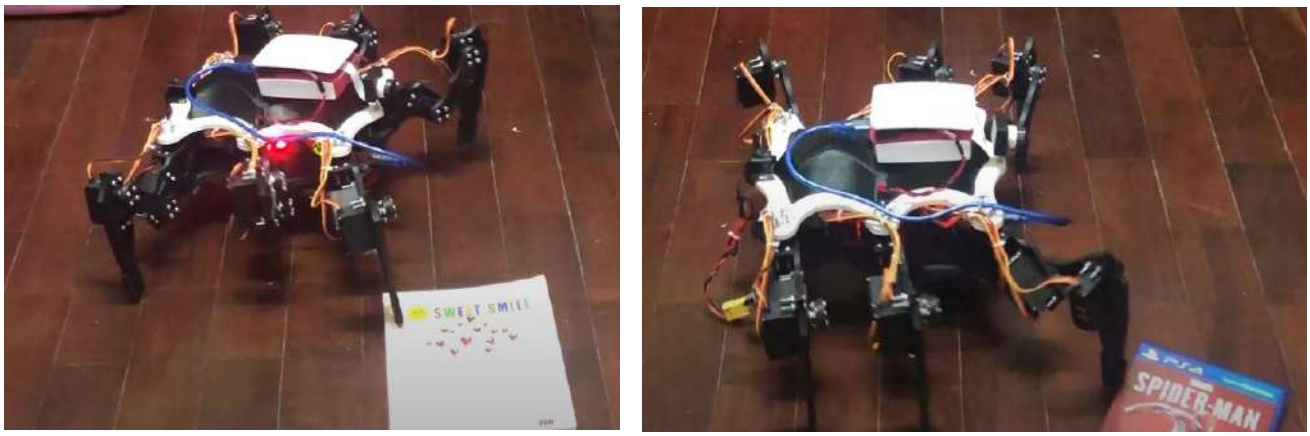
The last and one of the most important aspects of the hexapod is its battery life. There are 2 different batteries used in the hexapod; one is for the servo motor drivers and another is for the RPi. The Li-Po battery used for the servos is an 11.1V 1300 mAh battery. The power bank used is a 10000 mAh power bank that can provide 5V 3A. Before conducting the test,

both the power bank and the Li-Po batteries were charged to their full capacity. The power bank has 4 LED indicators representing 25% charge each and the full charged voltage of the Li-Po battery is 12.5V.

The battery testing strategy is to run the hexapod until the battery indicator on the power bank goes down by one light and measure the voltage of the Li-Po battery. After 1 hr of running the hexapod, one of the LED indicators on the power bank goes down. Therefore, the RPi used in the hexapod will be able to run about 4 hours since the power bank has 4 LED indicators. Measuring the voltage of the Li-Po battery with the voltmeter, it was found that after 1 hr of running time, the voltage went down to 11.79V. Therefore, calculating the capacity used during an hour, the result was that the hexapod can run for about 2.8 hours until it reaches the minimum capacity of 10.5V. Comparing both the battery capacities of the 2 power sources, the RPi can run for nearly 4 hrs and the servo motors can run for 2.8 hrs. However, even though the RPi can run for 4 hrs, the hexapod cannot fully function without the servo motors. Therefore, the battery life of the hexapod can be set to 2.8 hrs.

The current prototype of the hexapod doesn't feature environment and surface adaptability. Therefore, unlike high-level hexapods that are specifically designed to go over obstacles, this hexapod can only go over linear surfaces. However, to know the true limit of the hexapod, multiple tests will be conducted on a semi-rough surface with minimal obstacles.

As the testing goes, it is found that the hexapod can go over obstacles that are less than 10mm. Also, due to the hexapod's swing phase when it walks, it will likely to move the obstacles with its swing motion if they are light enough.



(a) *(b)*

Figure (4.3)(a) Hexapod going over 9mm book (b) Hexapod pushing away the 15mm thick CD box

Video links for figure (4.3)-

(a)https://drive.google.com/file/d/1HJT6lWQjoWJELvm4de80RKWqiiOLaqID/view?usp=sharing

(b)https://drive.google.com/file/d/1qmBQ0X_rQbevO5fl5T1k34YUmYGGGigj/view?usp=sharing

The hexapod was also tested with a stair-like obstacle (9mm thick books stacked together). It was seen that as long as the middle leg of the hexapod gives friction for the first leg, it will climb over the set of stairs. However, as seen in the video, the books were light enough for the hexapod to move them away from its path.

(a) *(b)*

Figure (4.4) (a) Hexapod going over first step (b) Hexapod going over second step with the help of the middle leg

Video link for the figure (4.4) -
https://drive.google.com/file/d/1N7odQxP3klZLluHVjK74kKH_NFTTxgYr/view?usp=sharing

Chapter 5: Conclusions and Future Improvements

5.1 Closing of the Hexapod Project

Looking back at the project, during the development of the hexapod, there were a lot of errors, setbacks, and changes. Also, schedule changes, tight working hours on the project, and the political issues in the country affected majorly on the project. Therefore, some expected features or original aims of the project were trimmed down to suit the new conditions yet give satisfaction to the original aims and objective of the project. Despite all the development problems and technical problems, the project was finished in time and achieved about 90% of the original goals.

5.2 Discussion

There are some slight drawbacks in this project. Firstly the weight of the hexapod. Since it has 2 separate microcontroller and a microprocessor on the hexapod, it adds more to the hexapod's weight. Since the chassis of the hexapod was sourced from an open source project, there weren't enough room for both Arduino and RPi.

The results from hexapod's walking algorithm and the camera installed on the hexapod were satisfactory although they can be improved. However, there is a small problem in using the RPi in headless mode. When trying to view RPi desktop with VNC viewer, it will sometimes show a black screen depending on the Wi-Fi signal. Therefore, RPi needs to be rebooted again and again until the VNC no longer shows a black screen. The camera streaming also has a similar problem. Sometimes the camera stream will freeze on a frame and the RPi needs to be rebooted back to work again.

On the other hand, the direction accuracy was not enough to go in a straight line. Although the problem can be worked around by adjusting the hexapod to go in its desired direction.

Therefore, the above said problems are small and can be worked around but it is somewhat of a nuisance in this project.

All the hardware and software aspects of the project were discussed above but in terms of developing this whole project, budget is the most important aspect. The budget for this project was initially aimed to be 300,000 MMKs which is nearly 168$ with current exchange rates. Because of this project was developed during the political crisis of this country, the component prices went up double and some shipping routes were blocked. Therefore, the components were considered solely on their availability in the market. Hence, the final budget for this project cost 438,000 MMKs (around 245$).

5.3 Future Improvements

Since the goal of this project is to develop a prototype of the hexapod with decent functions, some aspects of the hexapod can be improved to provide more usability and functionality in search and rescue operations.

The first thing to be improved is the body design of the hexapod. If given more time, 3D modelling each body part to fit the schematics of the hexapod can improve greatly in performance.

Another thing to improve is the camera. Due to the market availability and budget of this project, a relatively cheap camera version was used. The camera can be improved with more enhanced models or night vision cameras which will also add a night vision feature to the hexapod.

The next improvement is one of the bigger improvements. The walking algorithm can be improved with kinematics and more complex mathematics to adapt the surface of the ground.

The RPi is a powerful microprocessor however, in this project, to make things simple, driving the motors was assigned to the Arduino. The RPi can handle in driving the servo motors but it will complicate things more slightly in driving both the servos and handling the camera stream. Not to mention the load the RPi will have to take doing all the processes. However, by using just the RPi, the hexapod will become more of a compact form factor; saving space and reducing weight.

Coupled with the previous improvement, the battery output can be made to have a USB C output so that it can supply both the servo drivers and the RPi. However, a battery with more mAh would be needed for giving enough power and having better battery life.

References

Čížek, P. Zoula, M. and Faigl, J. (2021) 'Design, Construction, and Rough-Terrain Locomotion Control of Novel Hexapod Walking Robot With Four Degrees of Freedom Per Leg', - *IEEE Access*, 9, pp. 17866-17881.

Krishna, A. *et al*. (2014) 'Design and fabrication of a hexapod robot'. International Conference on Embedded Systems (ICES). pp. 225.

Marais, S.T. Nel, A.L. and Robinson, P.E. (2016) 'Reflex assisted walking for a hexapod robot'. *Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech*), pp. 1.

Matsuno, F and Tadokoro, S. (2004) 'Rescue Robots and Systems in Japan'. pp. 12.

Priandana, K., Buono, A. and Wulandari (2017) 'Hexapod leg coordination using simple geometrical tripod-gait and inverse kinematics approach*'. International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pp. 35.

Shah, B. and Choset, H. (2004) 'Survey on Urban Search and Rescue Robots', *日本ロボット学会誌*, 22(5), pp. 582-586.

Tedeschi, F. Carbone, G. (2014). 'Design Issues for Hexapod Walking Robots'. *Robotics*. [Online]. 3 (2). p.pp. 181–206.

Wikipedia (2021) *Legged robot.* Available at: https://en.wikipedia.org/wiki/Legged_robot (Accessed: 5.11.2021).

Zak, M. and Rozman, J. (2015) 'Design, construction and control of hexapod walking robot'. 2015 IEEE 13th International Scientific Conference on Informatics. pp. 302.

Zhang, H., Liu, Y., Zhao, J., Chen, J. and Yan, J. (2014) 'Development of a bionic hexapod robot for walking on unstructured terrain', *Journal of Bionic Engineering*, 11(2), pp. 176-187.

Zhang, L. Li, D. Yang, F. and Liu, C. (2016) 'Development and attitude control of a Hexapod bionic-Robot'. 2016 IEEE International Conference on Robotics and Biomimetics (ROBIO) pp. 77.

Appendix A – Arduino Sketch for the hexapod

```
#include "PCA9685.h"
#include <Wire.h>

PCA9685 driver(B000000); //address pin for first driver
PCA9685 driver1(B000001);//address pin for second driver

//Servo Mappings
//Left Side
//First Leg
PCA9685_ServoEval pwmServo(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo1(82, 481); // (-90deg,+90deg)
PCA9685_ServoEval pwmServo2(82, 481); // (-90deg,+90deg)
//Second Leg
PCA9685_ServoEval pwmServo3(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo4(82, 481); // (-90deg,+90deg)
PCA9685_ServoEval pwmServo5(82, 481); // (-90deg,+90deg)
//Third Leg
PCA9685_ServoEval pwmServo6(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo7(82, 481); // (-90deg,+90deg)
PCA9685_ServoEval pwmServo8(82, 481); // (-90deg,+90deg)

//Right Side
//Fourth Leg
PCA9685_ServoEval pwmServo9(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo10(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo11(82, 481); // (-90deg, +90deg)
//Fifth Leg
PCA9685_ServoEval pwmServo12(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo13(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo14(82, 481); // (-90deg, +90deg)
//Sixth Leg
PCA9685_ServoEval pwmServo15(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo16(82, 481); // (-90deg, +90deg)
PCA9685_ServoEval pwmServo17(82, 481); // (-90deg, +90deg)

int i1L1 = 0;
int i2L1 = 0;
int i3L1 = 0;
int i4L1 = 0;

int i1L2 = 0;
int i2L2 = 0;
int i3L2 = 0;
int i4L2 = 0;

int m = 0;

boolean l1status = LOW;
boolean l2status = LOW;
byte inputData = 0;

void setup() {
```

```
  Wire.begin();                // Wire must be started first
  Wire.setClock(400000);       // Supported baud rates are 100kHz, 400kHz, and 1000kHz
  Serial.begin(9600);
  delay(20);

  driver.resetDevices();       // Software resets all PCA9685 devices on Wire line
  driver1.resetDevices();

  driver.init ();
  driver1.init();
  driver.setPWMFrequency(50);   // Set frequency to 50Hz first driver
  driver1.setPWMFrequency(50);  // Ser frequency to 50Hz second driver

//Move to initial positions
//Leg1
//driver.setChannelPWM(servo number on the driver, pwmServo.pwmForAngle(0));
  driver.setChannelPWM(0, pwmServo.pwmForAngle(60));
  driver.setChannelPWM(1, pwmServo.pwmForAngle(50));
  driver.setChannelPWM(2, pwmServo.pwmForAngle(10));
  delay(500);
//Leg2
  driver.setChannelPWM(3, pwmServo.pwmForAngle(45));
  driver.setChannelPWM(4, pwmServo.pwmForAngle(45));
  driver.setChannelPWM(5, pwmServo.pwmForAngle(6));
  delay(500);
//Leg3
  driver.setChannelPWM(6, pwmServo.pwmForAngle(30));
  driver.setChannelPWM(7, pwmServo.pwmForAngle(50));
  driver.setChannelPWM(8, pwmServo.pwmForAngle(13));
  delay(500);
//Leg4
  driver1.setChannelPWM(0, pwmServo.pwmForAngle(-20));
  driver1.setChannelPWM(1, pwmServo.pwmForAngle(-10));
  driver1.setChannelPWM(2, pwmServo.pwmForAngle(25));
  delay(500);
//Leg5
  driver1.setChannelPWM(3, pwmServo.pwmForAngle(-11));
  driver1.setChannelPWM(4, pwmServo.pwmForAngle(-22));
  driver1.setChannelPWM(5, pwmServo.pwmForAngle(25));
  delay(500);
//Leg6
  driver1.setChannelPWM(6, pwmServo.pwmForAngle(8));
  driver1.setChannelPWM(7, pwmServo.pwmForAngle(-20));
  driver1.setChannelPWM(8, pwmServo.pwmForAngle(10));
  delay (3000);
}

void loop() {
  while (Serial.available())
  {
    int inputData= (Serial.read()-'0');
    if (inputData == 1) {
      m = 1;
  }
    if (inputData == 2) {
```

```
    m =2;
  }
   if (inputData == 3) {
    m = 3;
   }
   if (inputData == 4) {
    m = 4;
   }
   if (inputData ==0) {
    initialPosition();
   }
  }
  //Forward
  if (m == 1) {
   moveLeg1();
   moveLeg3();
   moveLeg5();
   if (l1status == HIGH) {
    moveLeg2();
    moveLeg4();
    moveLeg6();
   }
  }
  //Reverse
  if (m == 2) {
   moveLeg1Rev();
   moveLeg3Rev();
   moveLeg5Rev();
   if (l1status == HIGH) {
    moveLeg2Rev();
    moveLeg4Rev();
    moveLeg6Rev();
   }
  }
  //Left
  if ( m == 4) {
   moveLeg1();
   moveLeg3();
   moveLeg5Left();
   if (l1status == HIGH){
    moveLeg2();
    moveLeg4Left();
    moveLeg6Left();
   }
  }
  //Right
  if ( m == 3) {
   moveLeg1Right();
   moveLeg3Right();
   moveLeg5();
   if (l1status == HIGH){
    moveLeg2Right();
    moveLeg4();
    moveLeg6();
   }
```

```
  }
  delay(30); // Walking speed(Recommended==30)
}


void moveLeg1() {
  // Swign phase - move leg though air - from initial to final position
  // Rise the leg
  if (i1L1 <= 10)
  {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(50 + i1L1 * 2));
    driver.setChannelPWM(2, pwmServo.pwmForAngle(10 + i1L1 * 3));
    i1L1++;
  }
  // Rotate the leg
  if (i2L1 <= 30) {
    driver.setChannelPWM(0, pwmServo.pwmForAngle(60+i2L1));
    i2L1++;
  }
  // Move back to touch the ground
  if (i2L1 > 20 & i3L1 <= 10) {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(70 - i3L1 * 2));        //(the degree here
is the result after adding (i1L1*2 and i1L1*3) from initial positions)
    driver.setChannelPWM(2, pwmServo.pwmForAngle(40 - i3L1 * 3));
    i3L1++;
  }

  // Stance phase - move leg while touching the ground
  // Rotate back to initial position
  if (i2L1 >= 30) {
    driver.setChannelPWM(0, pwmServo.pwmForAngle(90 - i4L1));
    i4L1++;
    l1status = HIGH;
  }
  // Reset the counters for repeating the process
  if (i4L1 >= 30) {
    i1L1 = 0;
    i2L1 = 0;
    i3L1 = 0;
    i4L1 = 0;
  }
}


void moveLeg2(){
   if (i1L2 <= 10) {
    driver.setChannelPWM(4, pwmServo.pwmForAngle(45 + i1L2 * 2));
    driver.setChannelPWM(5, pwmServo.pwmForAngle(6 + i1L2 * 3));
    i1L2++;
  }
  if (i2L2 <= 30) {
    driver.setChannelPWM(3, pwmServo.pwmForAngle(45+i2L2));
    i2L2++;
  }
  if (i2L2 > 20 & i3L2 <= 10) {
```

```
    driver.setChannelPWM(4, pwmServo.pwmForAngle(65 - i3L2 * 2));
    driver.setChannelPWM(5, pwmServo.pwmForAngle(36 - i3L2 * 3));
    i3L2++;
   }
   if (i2L2 >= 30) {
    driver.setChannelPWM(3, pwmServo.pwmForAngle(75 - i4L2));
    i4L2++;
   }
   if (i4L2 >= 30) {
    i1L2 = 0;
    i2L2 = 0;
    i3L2 = 0;
    i4L2 = 0;
 }
 }

 void moveLeg3() {
   if (i1L1 <= 10) {
    driver.setChannelPWM(7, pwmServo.pwmForAngle(50 + i1L1 * 2));
    driver.setChannelPWM(8, pwmServo.pwmForAngle(13 + i1L1 * 3));
   }
   if (i2L1 <= 30) {
    driver.setChannelPWM(6, pwmServo.pwmForAngle(30+i2L1));
   }
   if (i2L1 > 20 & i3L1 <= 10) {
    driver.setChannelPWM(7, pwmServo.pwmForAngle(70 - i3L1 * 2));
    driver.setChannelPWM(8, pwmServo.pwmForAngle(43 - i3L1 * 3));
   }
   if (i2L1 >= 30) {
    driver.setChannelPWM(6, pwmServo.pwmForAngle(60 - i4L1));
   }
 }


 void moveLeg4() {
   if (i1L2 <= 10) {
    driver1.setChannelPWM(1, pwmServo.pwmForAngle(-10 - i1L2 *2));
    driver1.setChannelPWM(2, pwmServo.pwmForAngle(25 - i1L2 *3));
   }
   if (i2L2 <= 30) {
    driver1.setChannelPWM(0, pwmServo.pwmForAngle(-20 - i2L2));
   }
   if (i2L2 > 20 & i3L2 <= 30) {
    driver1.setChannelPWM(1, pwmServo.pwmForAngle(-30 + i3L2 *2));
    driver1.setChannelPWM(2, pwmServo.pwmForAngle(-5 + i3L2 *3));
   }
   if (i2L2 >= 30) {
    driver1.setChannelPWM(0, pwmServo.pwmForAngle(-50 + i4L2));
   }
 }
 void moveLeg5() {
   if (i1L1 <= 10) {
    driver1.setChannelPWM(4, pwmServo.pwmForAngle(-22 - i1L1 *2));
    driver1.setChannelPWM(5, pwmServo.pwmForAngle(25 - i1L1 *3));
   }
```

```
  if (i2L1 <= 30) {
    driver1.setChannelPWM(3, pwmServo.pwmForAngle(-11 - i2L1));
  }
  if (i2L1 > 20 & i3L1 <= 30) {
    driver1.setChannelPWM(4, pwmServo.pwmForAngle(-42 + i3L1 *2));
    driver1.setChannelPWM(5, pwmServo.pwmForAngle(-5 + i3L1 *3));
  }
  if (i2L1 >= 30) {
    driver1.setChannelPWM(3, pwmServo.pwmForAngle(-41 + i4L1));
  }
}
void moveLeg6() {
  if (i1L2 <= 10) {
    driver1.setChannelPWM(7, pwmServo.pwmForAngle(-20 - i1L2 *2));
    driver1.setChannelPWM(8, pwmServo.pwmForAngle(10 - i1L2 *3));
  }
  if (i2L2 <= 30) {
    driver1.setChannelPWM(6, pwmServo.pwmForAngle(8 - i2L2));
  }
  if (i2L2 > 20 & i3L2 <= 10) {
    driver1.setChannelPWM(7, pwmServo.pwmForAngle(-40 + i3L2 *2));
    driver1.setChannelPWM(8, pwmServo.pwmForAngle(-20 + i3L2 *3));
  }
  if (i2L2 >= 30) {
    driver1.setChannelPWM(6, pwmServo.pwmForAngle(-22 + i4L2));
  }
}
void moveLeg1Rev(){
  if (i1L1 <= 10)
  {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(50 + i1L1 * 2));
    driver.setChannelPWM(2, pwmServo.pwmForAngle(10 + i1L1 * 3));
    i1L1++;
  }
  if (i2L1 <= 30) {
    driver.setChannelPWM(0, pwmServo.pwmForAngle(60-i2L1));
    i2L1++;
  }

  if (i2L1 > 20 & i3L1 <= 10) {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(70 - i3L1 * 2)); //(the degree here is
the result after adding (i1L1*2 and i1L1*3) from initial positions)
    driver.setChannelPWM(2, pwmServo.pwmForAngle(40 - i3L1 * 3));
    i3L1++;
  }
  if (i2L1 >= 30) {
    driver.setChannelPWM(0, pwmServo.pwmForAngle(30 + i4L1));
    i4L1++;
    l1status = HIGH;
  }
  if (i4L1 >= 30) {
    i1L1 = 0;
    i2L1 = 0;
    i3L1 = 0;
    i4L1 = 0;
```

```
  }
}
void moveLeg2Rev(){
  if (i1L2 <= 10) {
    driver.setChannelPWM(4, pwmServo.pwmForAngle(45 + i1L2 * 2));
    driver.setChannelPWM(5, pwmServo.pwmForAngle(6 + i1L2 * 3));
    i1L2++;
  }
  if (i2L2 <= 30) {
    driver.setChannelPWM(3, pwmServo.pwmForAngle(45-i2L2));
    i2L2++;
  }
  if (i2L2 > 20 & i3L2 <= 10) {
    driver.setChannelPWM(4, pwmServo.pwmForAngle(65 - i3L2 * 2));
    driver.setChannelPWM(5, pwmServo.pwmForAngle(36 - i3L2 * 3));
    i3L2++;
  }
  if (i2L2 >= 30) {
    driver.setChannelPWM(3, pwmServo.pwmForAngle(15+i4L2));
    i4L2++;
  }
  if (i4L2 >= 30) {
    i1L2 = 0;
    i2L2 = 0;
    i3L2 = 0;
    i4L2 = 0;
}
}
void moveLeg3Rev(){
  if (i1L1 <= 10) {
    driver.setChannelPWM(7, pwmServo.pwmForAngle(50 + i1L1 * 2));
    driver.setChannelPWM(8, pwmServo.pwmForAngle(13 + i1L1 * 3));
  }
  if (i2L1 <= 30) {
    driver.setChannelPWM(6, pwmServo.pwmForAngle(30-i2L1));
  }
  if (i2L1 > 20 & i3L1 <= 10) {
    driver.setChannelPWM(7, pwmServo.pwmForAngle(70 - i3L1 * 2));
    driver.setChannelPWM(8, pwmServo.pwmForAngle(43 - i3L1 * 3));
  }
  if (i2L1 >= 30) {
    driver.setChannelPWM(6, pwmServo.pwmForAngle(0 + i4L1));
  }
}
void moveLeg4Rev(){
  if (i1L2 <= 10) {
    driver1.setChannelPWM(1, pwmServo.pwmForAngle(-10 - i1L2 *2));
    driver1.setChannelPWM(2, pwmServo.pwmForAngle(25 - i1L2 *3));

  }
  if (i2L2 <= 30) {
    driver1.setChannelPWM(0, pwmServo.pwmForAngle(-20 + i2L2));

  }
  if (i2L2 > 20 & i3L2 <= 30) {
```

```
    driver1.setChannelPWM(1, pwmServo.pwmForAngle(-30 + i3L2 *2));
    driver1.setChannelPWM(2, pwmServo.pwmForAngle(-5 + i3L2 *3));
   }
   if (i2L2 >= 30) {
    driver1.setChannelPWM(0, pwmServo.pwmForAngle(10 - i4L2));
   }
}
void moveLeg5Rev(){
   if (i1L1 <= 10) {
    driver1.setChannelPWM(4, pwmServo.pwmForAngle(-22 - i1L1 *2));
    driver1.setChannelPWM(5, pwmServo.pwmForAngle(25 - i1L1 *3));
   }
   if (i2L1 <= 30) {
    driver1.setChannelPWM(3, pwmServo.pwmForAngle(-11 + i2L1));

   }
   if (i2L1 > 20 & i3L1 <= 30) {
    driver1.setChannelPWM(4, pwmServo.pwmForAngle(-42 + i3L1 *2));
    driver1.setChannelPWM(5, pwmServo.pwmForAngle(-5 + i3L1 *3));
   }
   if (i2L1 >= 30) {
    driver1.setChannelPWM(3, pwmServo.pwmForAngle(19 - i4L1));
   }
}
void moveLeg6Rev(){
   if (i1L2 <= 10) {
    driver1.setChannelPWM(7, pwmServo.pwmForAngle(-20 - i1L2 *2));
    driver1.setChannelPWM(8, pwmServo.pwmForAngle(10 - i1L2 *3));
   }
   if (i2L2 <= 30) {
    driver1.setChannelPWM(6, pwmServo.pwmForAngle(8 + i2L2));
   }
   if (i2L2 > 20 & i3L2 <= 10) {
    driver1.setChannelPWM(7, pwmServo.pwmForAngle(-40 + i3L2 *2));
    driver1.setChannelPWM(8, pwmServo.pwmForAngle(-20 + i3L2 *3));
   }
   if (i2L2 >= 30) {
    driver1.setChannelPWM(6, pwmServo.pwmForAngle(38 - i4L2));
   }
}
void moveLeg1Right() {
   if (i1L1 <= 10)
   {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(50 + i1L1 * 2));
    driver.setChannelPWM(2, pwmServo.pwmForAngle(10 + i1L1 * 3));
    i1L1++;
   }
   if (i2L1 <= 30) {
    driver.setChannelPWM(0, pwmServo.pwmForAngle(60-i2L1));
    i2L1++;
   }

   if (i2L1 > 20 & i3L1 <= 10) {
    driver.setChannelPWM(1, pwmServo.pwmForAngle(70 - i3L1 * 2)); //(the degree here is
the result after adding (i1L1*2 and i1L1*3) from initial positions)
```

```
    driver.setChannelPWM(2, pwmServo.pwmForAngle(40 - i3L1 * 3));
    i3L1++;
   }
   if (i2L1 >= 30) {
    driver.setChannelPWM(0, pwmServo.pwmForAngle(30 + i4L1));
    i4L1++;
    l1status = HIGH;
   }
   if (i4L1 >= 30) {
    i1L1 = 0;
    i2L1 = 0;
    i3L1 = 0;
    i4L1 = 0;
   }
  }
  void moveLeg2Right(){
   if (i1L2 <= 10) {
    driver.setChannelPWM(4, pwmServo.pwmForAngle(45 + i1L2 * 2));
    driver.setChannelPWM(5, pwmServo.pwmForAngle(6 + i1L2 * 3));
    i1L2++;
   }
   if (i2L2 <= 30) {
    driver.setChannelPWM(3, pwmServo.pwmForAngle(45-i2L2));
    i2L2++;
   }
   if (i2L2 > 20 & i3L2 <= 10) {
    driver.setChannelPWM(4, pwmServo.pwmForAngle(65 - i3L2 * 2));
    driver.setChannelPWM(5, pwmServo.pwmForAngle(36 - i3L2 * 3));
    i3L2++;
   }
   if (i2L2 >= 30) {
    driver.setChannelPWM(3, pwmServo.pwmForAngle(15+i4L2));
    i4L2++;
   }
   if (i4L2 >= 30) {
    i1L2 = 0;
    i2L2 = 0;
    i3L2 = 0;
    i4L2 = 0;
  }
  }
  void moveLeg3Right() {
   if (i1L1 <= 10) {
    driver.setChannelPWM(7, pwmServo.pwmForAngle(50 + i1L1 * 2));
    driver.setChannelPWM(8, pwmServo.pwmForAngle(13 + i1L1 * 3));
   }
   if (i2L1 <= 30) {
    driver.setChannelPWM(6, pwmServo.pwmForAngle(30-i2L1));
   }
   if (i2L1 > 20 & i3L1 <= 10) {
    driver.setChannelPWM(7, pwmServo.pwmForAngle(70 - i3L1 * 2));
    driver.setChannelPWM(8, pwmServo.pwmForAngle(43 - i3L1 * 3));
   }
   if (i2L1 >= 30) {
    driver.setChannelPWM(6, pwmServo.pwmForAngle(0 + i4L1));
```

```
  }
}
void moveLeg4Left() {
  if (i1L2 <= 10) {
    driver1.setChannelPWM(1, pwmServo.pwmForAngle(-10 - i1L2 *2));
    driver1.setChannelPWM(2, pwmServo.pwmForAngle(25 - i1L2 *3));
  }
  if (i2L2 <= 30) {
    driver1.setChannelPWM(0, pwmServo.pwmForAngle(-20 + i2L2));
  }
  if (i2L2 > 20 & i3L2 <= 30) {
    driver1.setChannelPWM(1, pwmServo.pwmForAngle(-30 + i3L2 *2));
    driver1.setChannelPWM(2, pwmServo.pwmForAngle(-5 + i3L2 *3));
  }
  if (i2L2 >= 30) {
    driver1.setChannelPWM(0, pwmServo.pwmForAngle(10 - i4L2));
  }
}
void moveLeg5Left() {
  if (i1L1 <= 10) {
    driver1.setChannelPWM(4, pwmServo.pwmForAngle(-22 - i1L1 *2));
    driver1.setChannelPWM(5, pwmServo.pwmForAngle(25 - i1L1 *3));
  }
  if (i2L1 <= 30) {
    driver1.setChannelPWM(3, pwmServo.pwmForAngle(-11 + i2L1));
  }
  if (i2L1 > 20 & i3L1 <= 30) {
    driver1.setChannelPWM(4, pwmServo.pwmForAngle(-42 + i3L1 *2));
    driver1.setChannelPWM(5, pwmServo.pwmForAngle(-5 + i3L1 *3));
  }
  if (i2L1 >= 30) {
    driver1.setChannelPWM(3, pwmServo.pwmForAngle(19 - i4L1));
  }
}
void moveLeg6Left() {
  if (i1L2 <= 10) {
    driver1.setChannelPWM(7, pwmServo.pwmForAngle(-20 - i1L2 *2));
    driver1.setChannelPWM(8, pwmServo.pwmForAngle(10 - i1L2 *3));
  }
  if (i2L2 <= 30) {
    driver1.setChannelPWM(6, pwmServo.pwmForAngle(8 + i2L2));
  }
  if (i2L2 > 20 & i3L2 <= 10) {
    driver1.setChannelPWM(7, pwmServo.pwmForAngle(-40 + i3L2 *2));
    driver1.setChannelPWM(8, pwmServo.pwmForAngle(-20 + i3L2 *3));
  }
  if (i2L2 >= 30) {
    driver1.setChannelPWM(6, pwmServo.pwmForAngle(38 - i4L2));
  }
}
void initialPosition() {
  m = 0;
  l1status = LOW;
  l2status = LOW;
  //delay(30);
```

```
//Leg1
  driver.setChannelPWM(0, pwmServo.pwmForAngle(60));
  driver.setChannelPWM(1, pwmServo.pwmForAngle(50));
  driver.setChannelPWM(2, pwmServo.pwmForAngle(10));
  delay(10);
//Leg2
  driver.setChannelPWM(3, pwmServo.pwmForAngle(45));
  driver.setChannelPWM(4, pwmServo.pwmForAngle(45));
  driver.setChannelPWM(5, pwmServo.pwmForAngle(6));
  delay(10);
//Leg3
  driver.setChannelPWM(6, pwmServo.pwmForAngle(30));
  driver.setChannelPWM(7, pwmServo.pwmForAngle(50));
  driver.setChannelPWM(8, pwmServo.pwmForAngle(13));
  delay(10);
//Leg4
  driver1.setChannelPWM(0, pwmServo.pwmForAngle(-20));
  driver1.setChannelPWM(1, pwmServo.pwmForAngle(-10));
  driver1.setChannelPWM(2, pwmServo.pwmForAngle(25));
  delay(10);
//Leg5
  driver1.setChannelPWM(3, pwmServo.pwmForAngle(-11));
  driver1.setChannelPWM(4, pwmServo.pwmForAngle(-22));
  driver1.setChannelPWM(5, pwmServo.pwmForAngle(25));
  delay(10);
//Leg6
  driver1.setChannelPWM(6, pwmServo.pwmForAngle(8));
  driver1.setChannelPWM(7, pwmServo.pwmForAngle(-20));
  driver1.setChannelPWM(8, pwmServo.pwmForAngle(10));
  delay(10);

  i1L1 = 0;
  i2L1 = 0;
  i3L1 = 0;
  i4L1 = 0;

  i1L2 = 0;
  i2L2 = 0;
  i3L2 = 0;
  i4L2 = 0;
}
```

Appendix B- Python Codes for the Raspberry Pi

Keypress Module

```
import pygame

def init():
        pygame.init()
        win = pygame.display.set_mode((100,100))
def getKey (keyName):
        ans = False
        for eve in pygame.event.get():pass
        keyInput = pygame.key.get_pressed()
        myKey = getattr(pygame, 'K_{}' .format(keyName))
        if keyInput [myKey]:
                ans = True
        pygame.display.update()

        return ans

def main():
        if getKey('LEFT'):
                print( 'Key Left was pressed')
        if getKey('RIGHT'):
                print('Key Right was pressed')

if __name__== '__main__':
        init()
        while True:
                main()
```

RPi to Arduino Serial

```
import serial
import time
import KeyPressModule as kp

time.sleep(1)
kp.init()

def main():
        if kp.getKey('UP'):
                ser.write(b '1' )
                time.sleep(0.1)
        elif kp.getKey('DOWN'):
                ser.write(b '2' )
                time.sleep(0.1)
        elif kp.getKey('LEFT'):
                ser.write(b '3' )
                time.sleep(0.1)
        elif kp.getKey('RIGHT'):
                ser.write(b '4' )
                time.sleep(0.1)
        else:
```

```
                ser.write(b ‘0’ )
                time.sleep(0.1)

if __name__== ‘__main__’ :
        ser = serial.Serial(‘/dev/ttyACM0’, 9600, timeout =1)
        ser.reset_input_buffer()
        while True:
                main()
```

Camera Module

```
import cv2
from imutils.video.pivideostream import PiVideoStream
import imutils
import time
import numpy as np

class VideoCamera(object):
    def __init__(self, flip = False):
        self.vs = PiVideoStream().start()
        self.flip = flip
        time.sleep(2.0)

    def __del__(self):
        self.vs.stop()

    def flip_if_needed(self, frame):
        if self.flip:
            return np.flip(frame, 0)
        return frame

    def get_frame(self):
        frame = self.flip_if_needed(self.vs.read())
        ret, jpeg = cv2.imencode('.jpg', frame)
        return jpeg.tobytes()
```

Main Camera Code

```
from flask import Flask, render_template, Response, request
from camera import VideoCamera
import time
import threading
import os

pi_camera = VideoCamera(flip=False) # flip pi camera if upside down.

# App Globals (do not edit)
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html') #you can customze index.html here

def gen(camera):
```

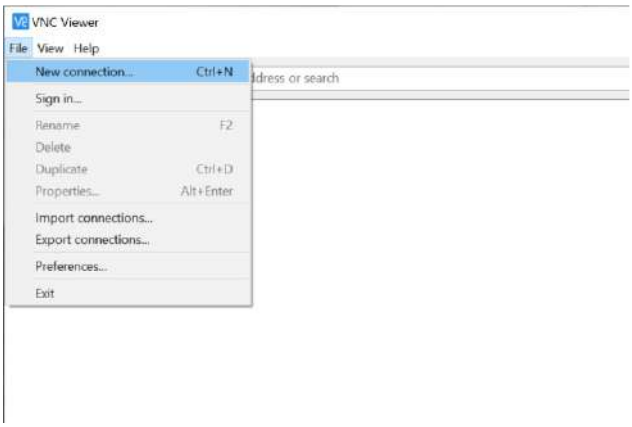
```
    #get camera frame
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')

@app.route('/video_feed')
def video_feed():
    return Response(gen(pi_camera),
                    mimetype='multipart/x-mixed-replace; boundary=frame')

if __name__ == '__main__':

    app.run(host='0.0.0.0', debug=False)
```

The codes necessary to run the hexapod (Both Arduino and Python codes) will be available to view and download in this following link https://drive.google.com/drive/folders/1eDkhwJmfLM_P0MWthb4GC6Vi_cFXZeON?usp=sharing

Appendix C – User Manual

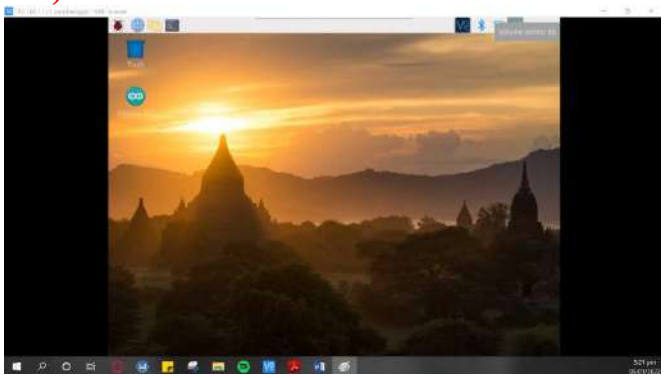
Step 1: Plug the Power Bank into the RPi.

Step 2: From the computer, open VNC viewer and select add new connection (For the first time user)

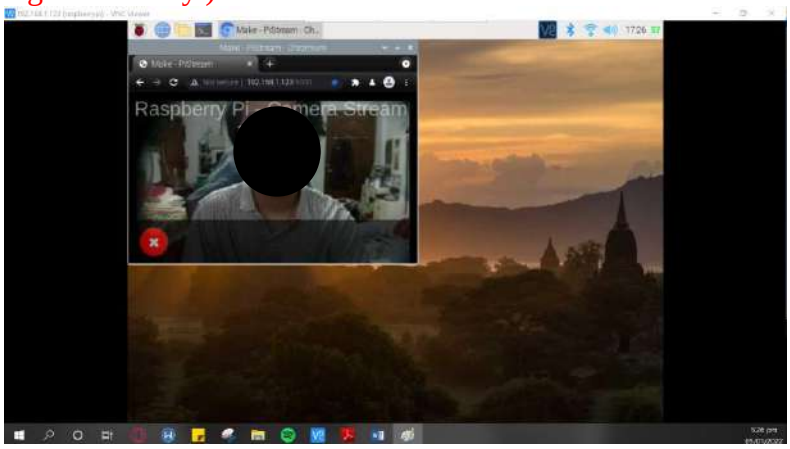


Step 3: Enter the static IP address for the RPi.
Enter the following username and password.
(Check remember password so that u won't have to enter next time)

Step 4: After a successful connection, the RPi Desktop view will show up.
(Note: sometimes, depending on the Wi-Fi connection, the screen will show black. The only current fix for that is to unplug and re-plug the power bank to the RPi; essentially rebooting the RPi).

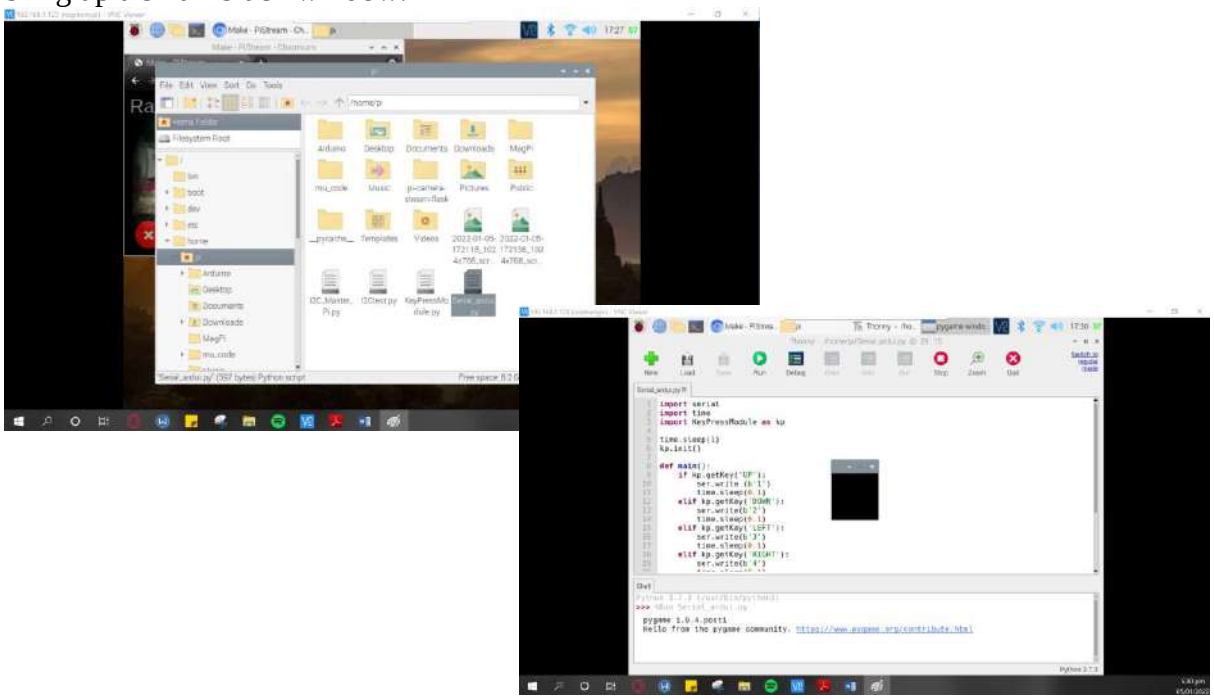


Step 5: Open the browser of your choice and type in the following address “192.168.1.123:5000” to show up camera stream.
(Note: It is recommended to put the window size of the browser, to ¼ of the screen to reduce lag and latency.)



Step 6: Flip the switch at the back of the hexapod and plug in the Arduino cable into the top USB 3.0 port of the RPi.

Step 7: From the location “/home/pi” open the file “Serial_ardui.py”. And pressing Run will bring up a small black window.



Step 8: Now u can minimize the other windows except the small black window and the browser window for the camera stream and put them side by side. You can now give commends to the hexapod by pressing the arrow keys from the keyboard.
(Note: Before giving commends make sure to click inside the small black window so that the commends are registered)