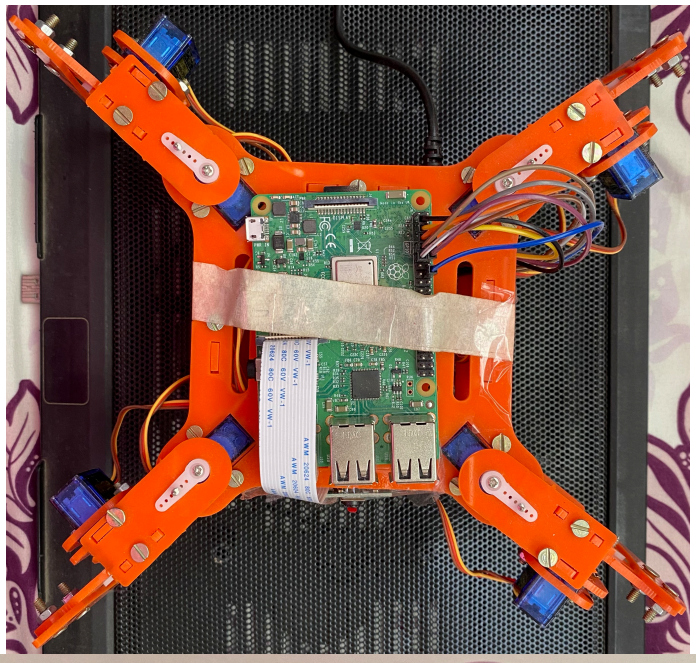
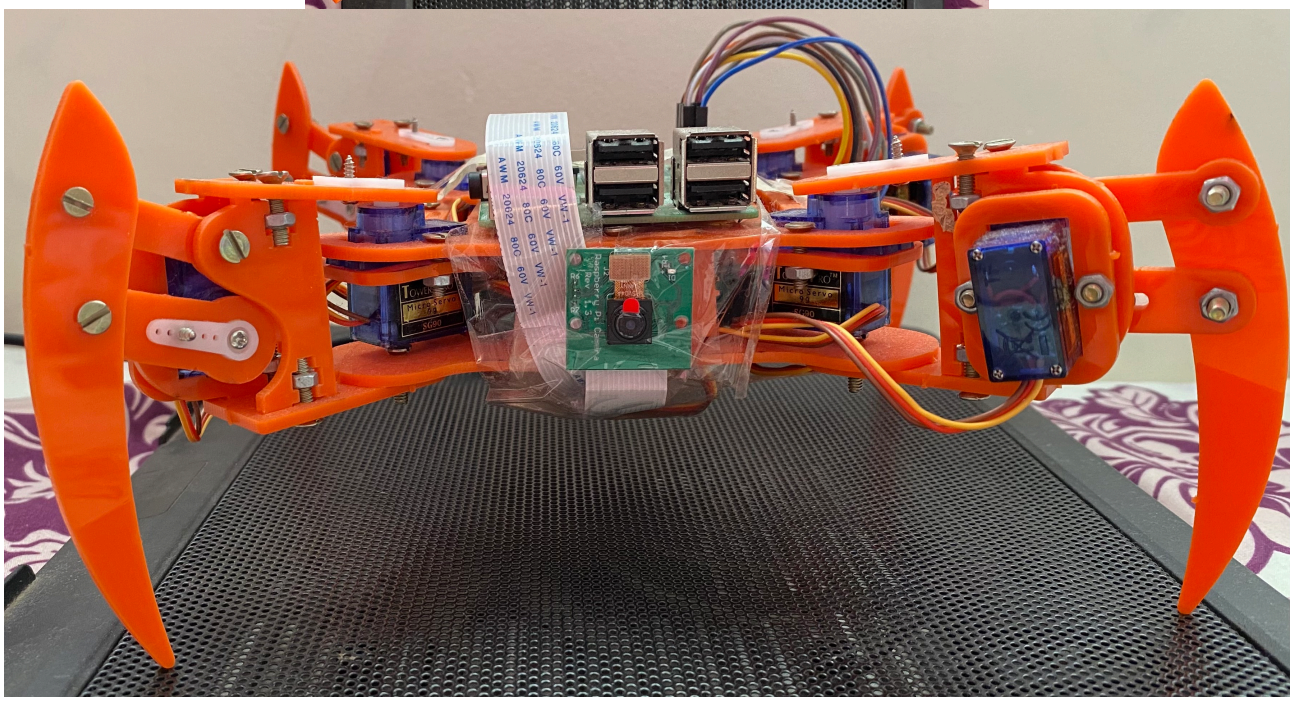
Hardwired Event

Team: Spidy

Ayush Singh Pal
Vidit Agarwal
Rishabh Singh

Team Mentor:
Surabhit Gupta





System Requirements:

- Raspberry Pi 3B (1GB RAM), Raspberry Pi 5MP Camera
- Python 3.7.3
- Opencv-contrib-python 3.4.4

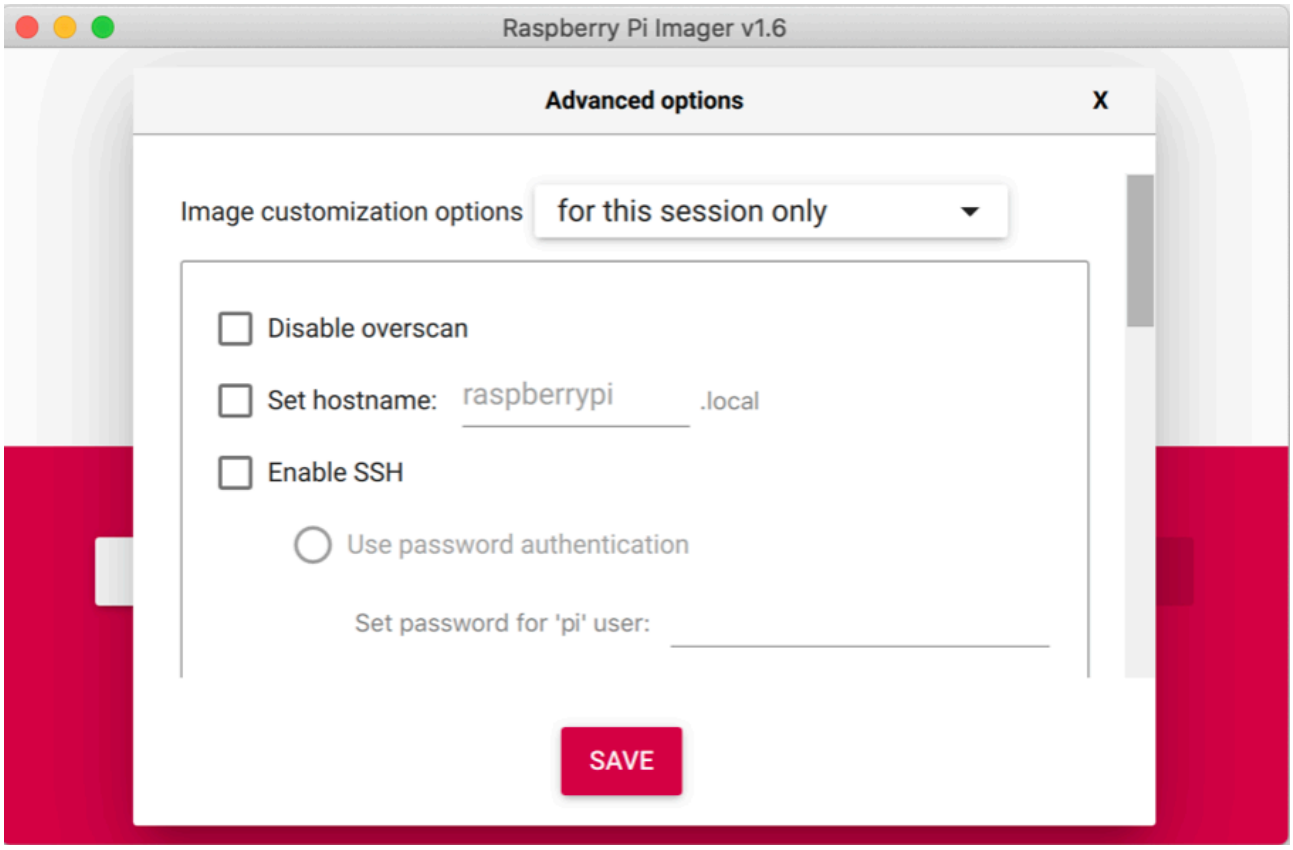
Installation of Raspberry Pi (Headless Start)

1. We require a memory card (Minimum 8GB). We goto official website and download and install the Raspberry Pi Imager (https://www.raspberrypi.com/software/). Now we connect the memory card to the computer and follow the on screen instructions to install Raspberry Pi OS on memory card. Follow the YouTube link (https://www.youtube.com/watch?v=ntaXWS8Lk34)

2. For Headless Start i.e. without using the external monitor and keyboard we need to goto Advanced menu. For Advanced menu we must push Ctrl-Shift-X and follow on screen instruction. For Headless start we need to enable SSH and set hostname and enter our Wi-Fi password.

3. To access SSH we need to type “ssh pi@hostname.local" on terminal and then enter the password. By this we can access the terminal of the Raspberry Pi through our local computer. The Computer and Raspberry Pi must be connected to the same Wi-Fi.

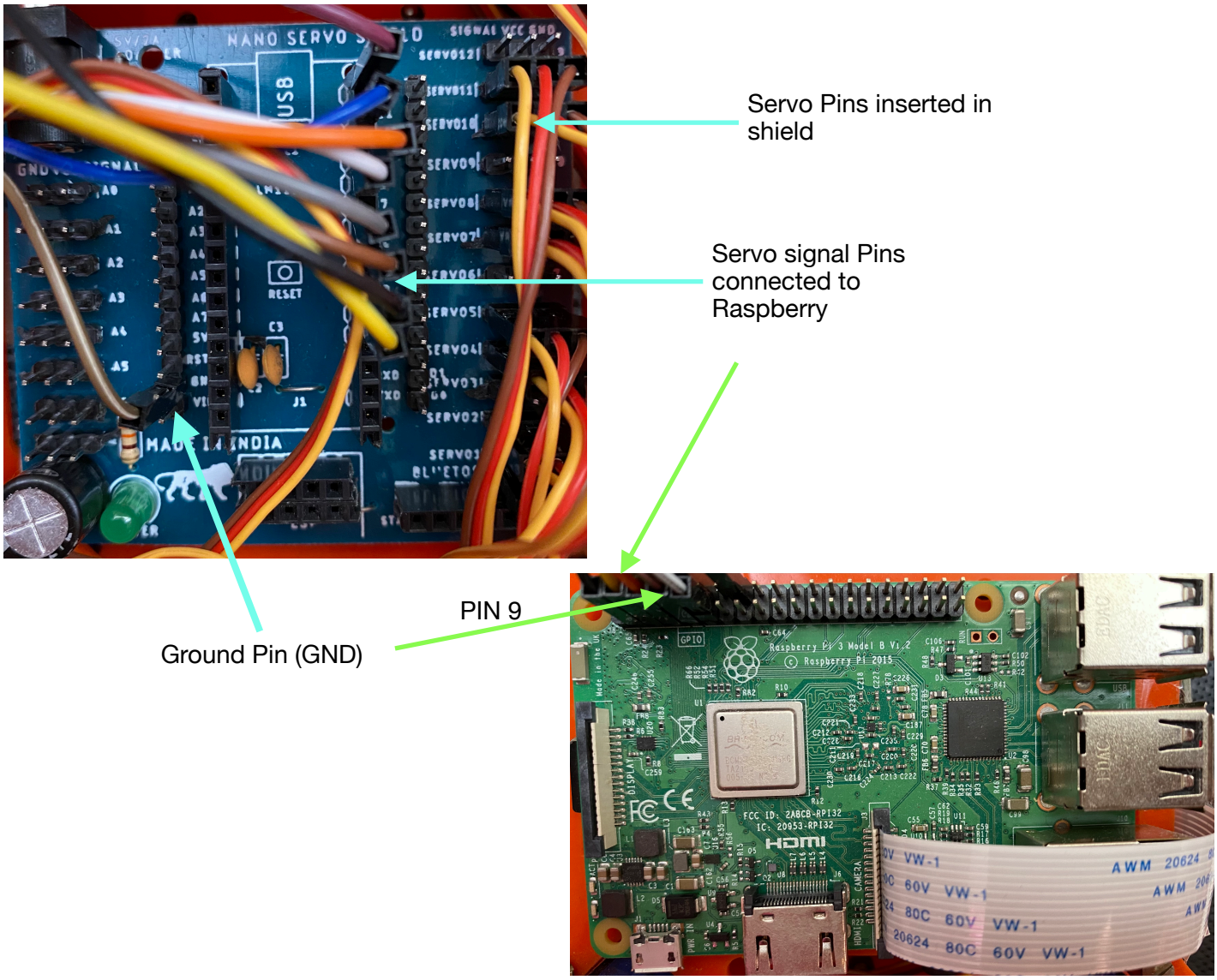
4. To get the GUI of Raspberry Pi we must enable VNC Server (Setup username and password). Now we can use this user name and password by entering it in Screen-sharing (Mac), Remote Desktop (Windows) to access the GUI.



Connecting the Raspberry Pi and Servos

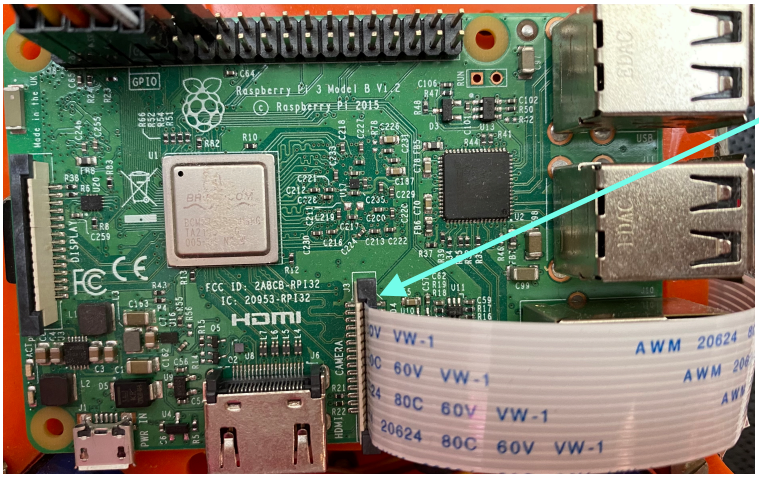
The outputs and inputs through the Raspberry Pi are accessed through GPIO pins.

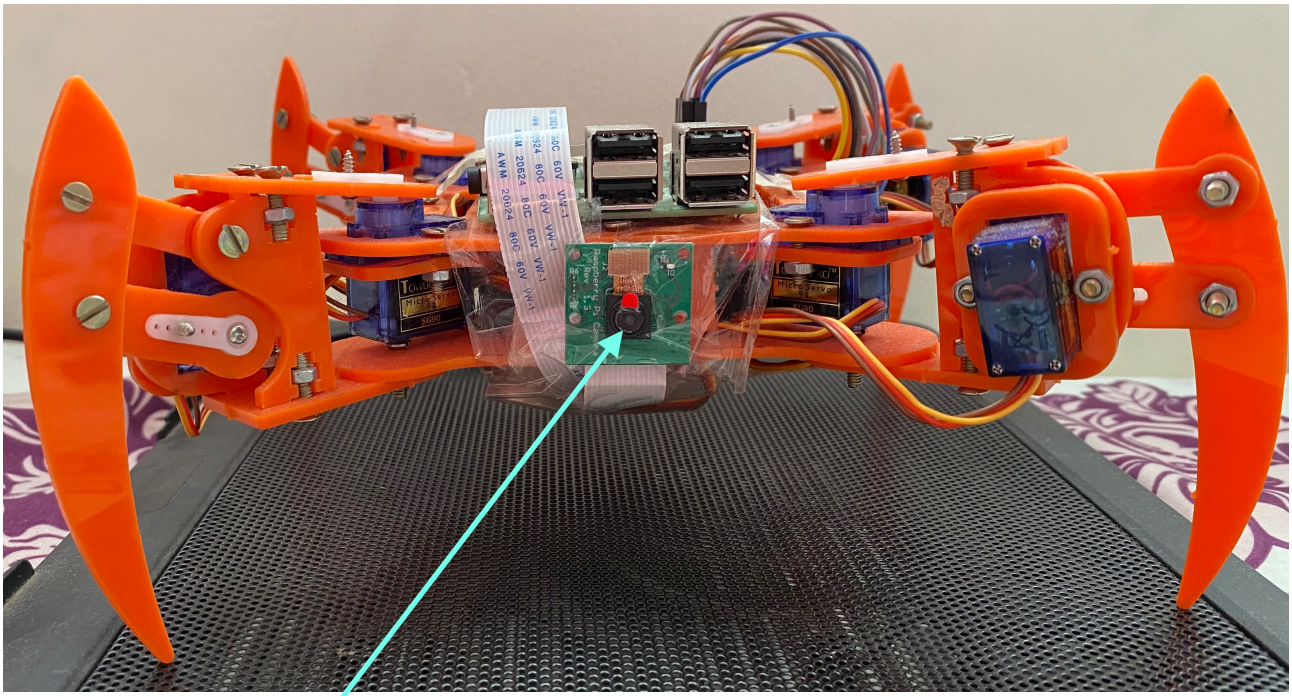
1. The servos are attached to the Arduino servo shield, and the pins they are connected to are noted. We use Arduino shelled as it has a dedicated power supply to fulfil the demand of the 8 servo motors. In servo SG90 the Brown wire is GND, Red wire is VCC, Yellow wire is for Signal.
2. Now we use the Raspberry Pi's GPIO pins for giving the PWM signals that would control the angles the servo has to move. We set the GPIO pins as PWM pins and use the numbering scheme as "board".
3. We connect the designated GPIO pin (as mentioned in movements.py file) to respective pin in Arduino shelled. We also connect a GND (Ground) pin form Raspberry to Arduino as the circuit needs to be complete.
4. The GPIO pins gives signals to the Yellow wire of servo. Now by changing the duty-cycle we can change the angle. Duty-cycle for 0 deg = 2, for 180 deg = 12.



Connecting Camera

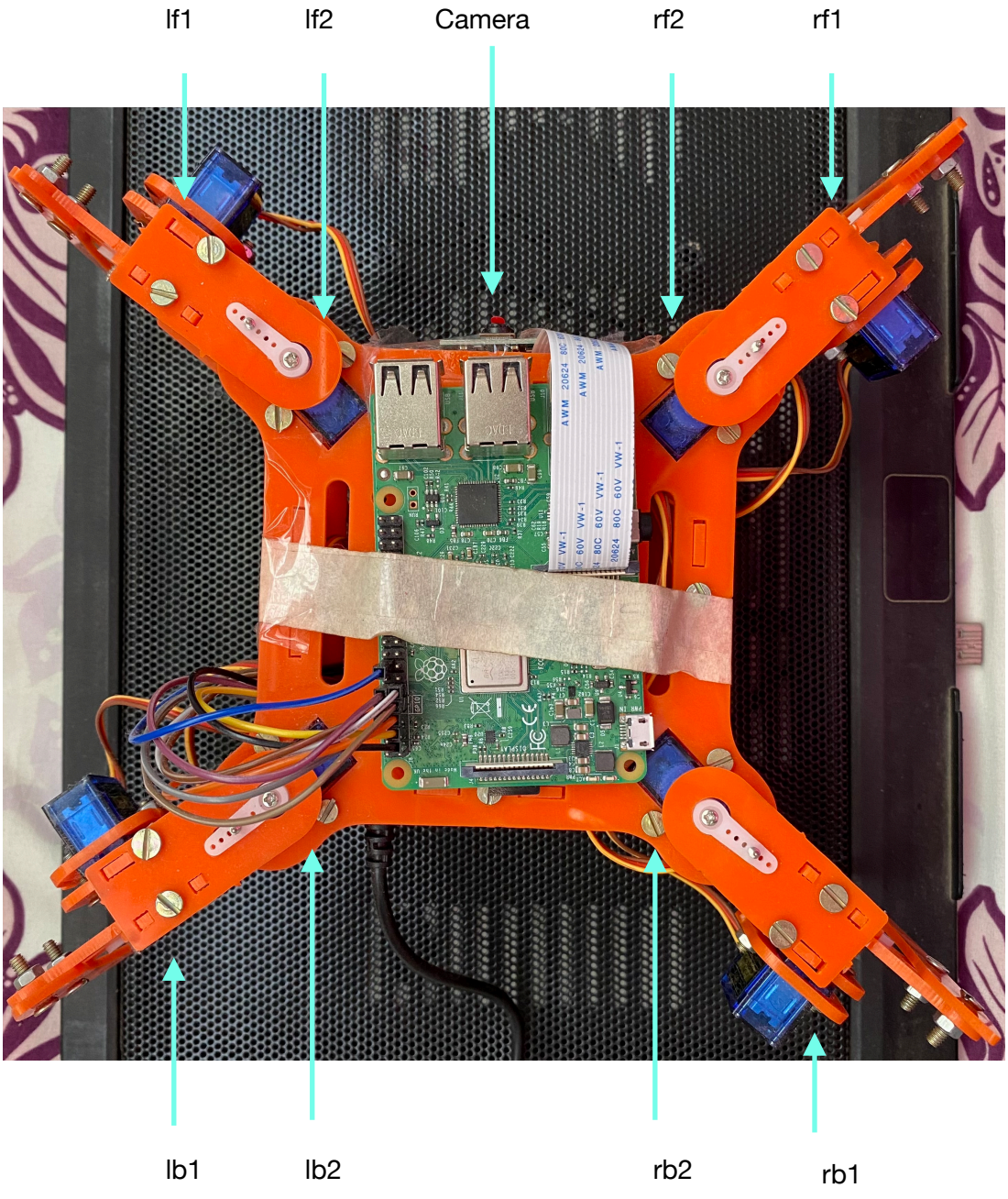
1. Enter following command in terminal “sudo raspi-config”, then select the interference options there we must enable camera. Then by clicking 2 times “Tab” button clicking on finish.
2. Now to attach the camera we must locate the connector and pull the grey holder up and insert the ribbon of camera, by keeping the blue coloured part towards the ethernet port. Then moving the grey holder down.
3. Then we position the camera and run the codes that would access the camera.
4. The video was streamed to our device through the window produced by “cv2.imshow(<Window_Name>, <img_numpy_array>) in the GUI interface of Raspberry Pi.



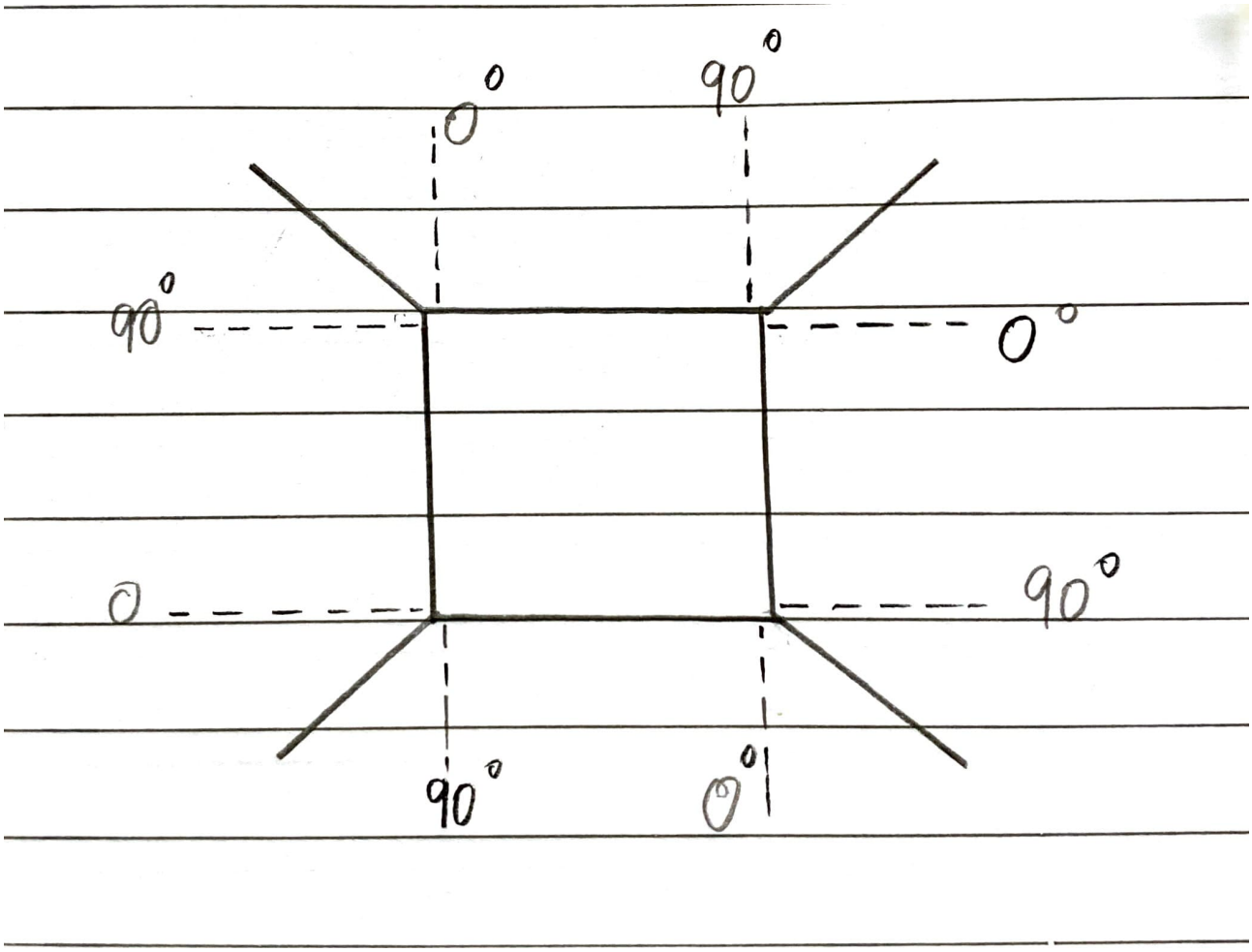


Movements of Robot

1. The servos are labeled in the program movement.py as shown in diagram. Their angle are being changes by function ChangeDutyCycle(). The position of leg based on angle can be determined by the digram given in next page.
2. The movements.py program is formed by following a algorithm to move robot in different directions.
3. The robot is controlled using the w, a, s, d keys. It is programmed that by pressing w in terminal it will move robot in forward direction. Similar procedure for other directions.



Angles to be changed on legs



Method to leg 2 (Reference to above figure)

Forward movement:

1. We move the lb2 to 0 deg (Duty-cycle = 2).
2. Then we move lf2 to 90 deg (Duty-cycle = 7)
3. Then we simultaneously move lf2 to 45 deg (Duty-cycle = 4.5), rf2 to 0 deg (Duty-cycle = 2), lb2 to 45 deg (Duty-cycle = 4.5), rb2 to 90 deg (Duty-cycle = 7)
4. Then we move rb2 to 45 deg (Duty-cycle = 4.5)
5. Then we move rf2 to 45 deg (Duty-cycle = 4.5)

Reverse movement:

1. We move the rf2 to 0 deg (Duty-cycle = 2).
2. Then we move rb2 to 90 deg (Duty-cycle = 7)
3. Then we simultaneously move rb2 to 45 deg (Duty-cycle = 4.5), lb2 to 0 deg (Duty-cycle = 2), rf2 to 45 deg (Duty-cycle = 4.5), lf2 to 90 deg (Duty-cycle = 7)
4. Then we move lb2 to 45 deg (Duty-cycle = 4.5)
5. Then we move lf2 to 45 deg (Duty-cycle = 4.5)

Pitch towards Left:

1. We move the rf2 to 90 deg (Duty-cycle = 7).
2. Then we move lb2 to 90 deg (Duty-cycle = 7)
3. Then we simultaneously move lb2 to 45 deg (Duty-cycle = 4.5), lf2 to 0 deg (Duty-cycle = 2), rf2 to 45 deg (Duty-cycle = 4.5), rb2 to 0 deg (Duty-cycle = 2)
4. Then we move lf2 to 45 deg (Duty-cycle = 4.5)
5. Then we move rb2 to 45 deg (Duty-cycle = 4.5)

Pitch towards Right:

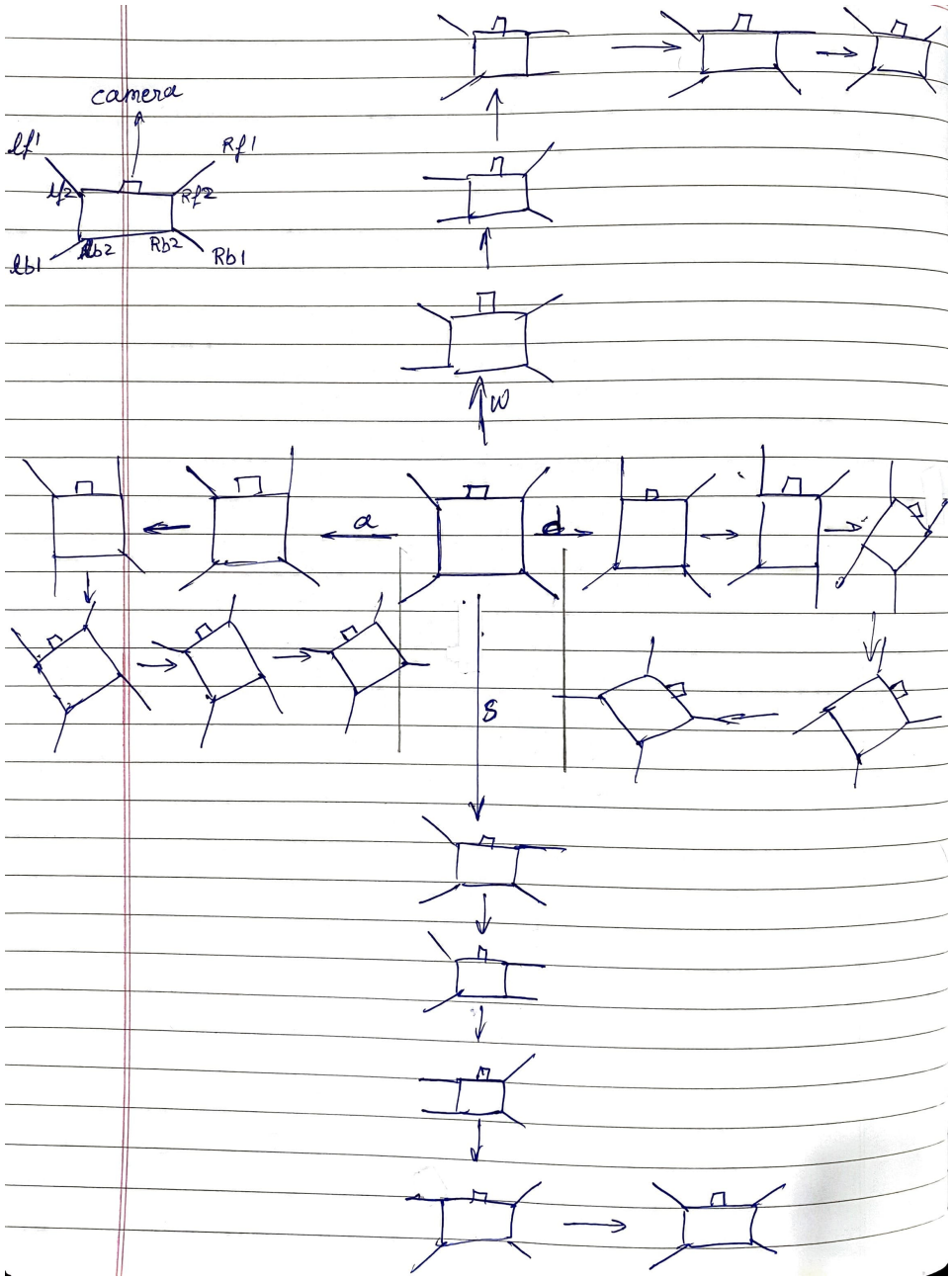
1. We move the lf2 to 0 deg (Duty-cycle = 2).
2. Then we move rb2 to 0 deg (Duty-cycle = 2)
3. Then we simultaneously move rb2 to 45 deg (Duty-cycle = 4.5), lf2 to 90 deg (Duty-cycle = 7), lf2 to 45 deg (Duty-cycle = 4.5), rb2 to 90 deg (Duty-cycle = 7)
4. Then we move rf2 to 45 deg (Duty-cycle = 4.5)
5. Then we move lb2 to 45 deg (Duty-cycle = 4.5)

**** Before any of step occurs we will lift the leg by changing duty cycle of leg numbered 1 for 4.5 to 0 i.e 45 deg to 0 deg.**

This algorithm is explained in below picture.

Algorithm for Robot movements

It is shown in following diagram.



Object Tracking

In this, we took the help of the opecv-contrib python library. Using the MEDIANFLOW tracker present in this library we tracked the region which we select at the beginning where we get the option to drag and select the region of interest which has to be tracked in further frames. We also tried KCF, MIL, BOOSTING, MOOSE, CSRT tracker but were not able to give decent frame rates along with proper tracking in spite of the object moving out of the frame and coming back, occlusion, object moving too fast in the frame, as compared to MEDIANFLOW tracker.

Tracking the moving object

(Did not work as expected thus it's video not included)

From tracker.update(<image_numpy_array>) we got various attributes of the tracked frame like coordinates of corner, width and height of the box. Using this data we found out the centre of the rectangle formed to bound our region of interest. Then we found the shape of the frames captured by Raspberry PI using frame.shape which came out to be (640, 480, 3) so if the centre of the tracked frame is greater than 340 then it will move right else if less than 300 it will move left and if between 300 to 340 then our bot will standstill. But this idea did not work well the tracking and video streaming was very laggy and the time the bot took to respond to change in the coordinate of the tracked frame was very slow thus at times the bot continued to move left when it had to stop and when it stopped it was too late it had turned more than 90^{o} and object had moved out of the frame and thus bot had stopped. We thought it might be due to much computation begin performed on pi so we resized the frame to half its size to reduce computation load but it did not improve the performance of the bot.