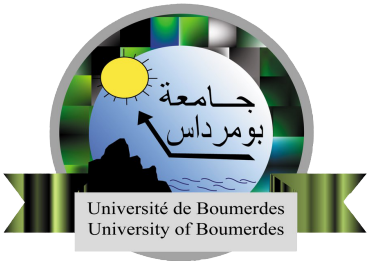


University M'Hamed BOUGARA - Boumerdes
Institute of Electrical and Electronics Engineering
Department of Electronics

Color Sorting Robot

"FALCON"

Prepared By:
Farouk Elkiouas
Lahlah Zehor
Kenza Mohammedi

Supervised By:
Pr.Razika Boushaki

Project Report Presented in Partial Fulfillment of the Requirements of the Degree of: 'LICENSE' In Electrical and Electronics Engineering

Boumerdes
May, 2022

Acknowledgement

In the name of Allah, The Most Beneficent and The Most Merciful, Alhamduli'Allah, all praises are to Allah for all blessings so that we could accomplish this project.

In addition, we would like to pay our special regards to our supervisor Dr.Boushaki Razika for her great efforts, guidance, and assistance in every step of this project made it an absolute privilege to work with her.

Finally, yet importantly, we would also like to express our sincere gratitude to our parents for their unending encouragement, prayers of day and night, patience, and motivation, without them, this project would not have seen the light, and we would like to send our greatest love and appreciation to, everyone that helped us for their constant support that makes us able to get such success and honor.

Abstract

This project aims to build an autonomous car arm robot with a smart approach to implementing the sorting of objects based on color. As the robotic car moves and displaces everywhere, a camera on the top of the arm scans through the area to detect any closest object and classifies it based on its color conditions, while the arm is taking care of picking it and dropping it in the specified location according to the color.

The color detection is based on an image processing method using the famous python library for computer vision and machine learning: "OpenCV" to write our code. The system is controlled by a Raspberry Pi 4 Model B board for the movements of the arm robot and to control the directions of the car's motion. This is a low-cost system with the simplest concepts to implement sorting effectively, saving working time, and it's one of the fastest systems in Industrial applications that provides fewer human mistakes when the manual system is undertaken.

Table of Contents

List of Figures

Introduction

In the era of robotics and automation, all industries are becoming automated for faster development and growth. A robot is an electro-mechanical machine that reduces human efforts and increases efficiency. It is a real-time machine that completes its tasks in a given time, with the help of computer programming. The project presents the design and development of a car arm robot with the application of color sorting using image processing. The color sorting robot is programmed to pick the object from one place and drop it accordingly into the respective colored box.

The intensity of the color measured by our camera based on the HSV color model which includes a range of colors is converted into a frequency which is given as input to the Raspberry Pie which then enables the motor driver circuit to drive the motors of the robot to grip the objects and drop them in the specified location according to the color. This report contains three main chapters: chapter one is about the history and description of our main theme "Robot, Object and color detection", the second chapter would dig into the hardware and software components needed with the definition of each one of them, as for the last chapter it would show our project implementation, our process, and our next goals.

Chapter 1

Basic Knowledge

Contents

1.1 Brief History

1.1.1 A robot is born!

One can think of no finer place to start a historical dissertation on robotics than the master of the Renaissance method, Leonardo da Vinci. In the early 1950s, investigators at the University of California scrutinized detailed drawings from da Vinci's notebooks, which form a tome exceeding 1,119 pages dating from 1480 to 1518 and, therefore, referred to, like the great Atlantic Ocean as the Codex Atlanticus. da Vinci was profoundly influenced by classical Greek thinkers in art and engineering. Modern investigations increasingly make it clear that he singularly pursued knowledge of everything known to these ancient scholars. He, in effect, was following in the footsteps of such figures as Hero of Alexandria, Philon, and Ctesibius who were all reported to be interested in mechanically simulating motion and human attributes. Da Vinci began a systematic method of devising and building the sophisticated mechanical device that was 500 years ahead of its time. His first robotic design was in December 1478, at the age of 26. It is a powerful mechanism that features a front-wheel-drive, rack-and-pinion automobile. Impressive as it is, it was also fully programmable, with the ability to control its motion and direction. It is now thought that this "base" would form the basis of his ultimate goal, a fully functional automaton. [1] To animate a humanoid machine, he was cognizant of his need to develop a more detailed database of human kinesiology. Leonardo grounded his knowledge further in drafting, anatomy, metalworking, toolmaking, and armor design, in addition to painting and sculpture. Leonardo was not content with a simple understanding of human anatomy, so he began to investigate and draw comparative anatomy, to better appreciate form and function. "You should make a discourse concerning the hands of each of the animals, to show in what way they vary." [2] In 1495, at about the time he was working on his method of painting on wet plaster and the Last Supper, da Vinci designed and probably built the first of several programmable humanoid robots. From research ongoing at the Florence-based Institute and Museum of the History of Science and work by Rosheim, it is now apparent his robot could open and close its anatomically correct jaw, sit up, wave its arms, and move its head.[3] This robot consisted of two independent systems The lower extremities had three degrees of freedom—legs, ankles, knees, and hips. The upper extremities had four degrees of freedom—arms with articulated shoulders, elbows, wrists, and hands. The orientation of the arms indicates it could only whole-arm grasp with the joints moving in unison. The device had an "onboard" programmable controller within the chest providing power and control over the arms. The legs were powered by an external crank arrangement. The Florence-based Institute and Museum of the History of Science have developed sophisticated computer models of this design with streaming video animation.[4]

1.1.2 Industrial revolution: the new era of robotic arm

Unimate introduced its first robotic arm in 1962. The arm was invented by George Devol and marketed by Joseph Engelberger. The first industrial arm was installed at the General Motors plant in Ternstedt, New Jersey, for automated diecasting. Ultimately, approximately 8,500 units were sold, from which the world knew this field was no longer the future of technology, but also the future of business.[5] Now let's talk a little about the types of robotic arms. It has five main types which are: rectangular coordinate, polar coordinate, cylindrical coordinate, revolver coordinate, and self-compliant automatic robot assembly (SCARA). Two more recent additions are called serpentine and anthropomorphic. [6] These arms can be subdivided by the types and complexity of each of their joints and control systems. The evolution of robotic arms is rapidly developing, however, and such schemes probably do more for organizing information than in defining the actual product. Applications to medicine, and surgery, in particular, are ripe for companies, because classic fields of application, for example, nuclear reactor work, have declined. In the past 40 years, radical improvements have been made and more degrees of freedom are now possible. Downsizing and cost reduction will follow. Hand technologies will rapidly advance as computer-control issues improve and work at universities will find fruitful applications in industry and medicine. "Haptics" and other sensory systems will be added to advanced surgical robotics as this technology evolves. [7]

1.1.3 Why did We choose a color sorting robot? (The ADVANTAGES!):

Reason 01: Let's talk first about humanity!
Our project is a color detector robotic arm placed on a vehicle for easy displacement. The robot detects the object according to the wanted color, grabs it, and places it in a certain place. This could help people with disabilities to interact more with the outer environment, for example, to place things when they can't.

Reason02: Us!
We as engineers are passionate to understand the world, how it functions, and how it works! And that applies to ourselves as well! How do we humans interact? How do we do things? Answering those questions is very interesting and important for us, therefore we choose our robot model based on two things: color detection: because we always wondered how humans can see and understand color, and before that: what is color? two: dynamic: placing and picking things according to order is similar to how our brains and bodies function all the time, and that's interesting to us!

Reason03: INDUSTRY!
Repetitive tasks and high accuracy have become the two contradictory needs of any industrial process. By introducing autonomous robotic applications, simple repetitive tasks can be accomplished while keeping the demands of accuracy and speed in mind. Nowadays in this fast-growing industrial age, every company needs speed in manufacturing to cope with the customer's requirements. In addition to that, Robots protect workers from repetitive, mundane, and dangerous tasks, while also creating more desirable jobs, like engineering, programming, management, and equipment maintenance they handle also, parts that are too small for human eyes and fingers and never make mistakes and improve the efficiency and productivity.

1.2 Object Detection

1.2.1 What do we mean by Object Detection?

Object detection is a computer vision technique that works to identify and locate objects within an image or video. Specifically, object detection draws bounding boxes around these detected objects, which allow us to locate where said objects are in (or how they move through) a given scene. Object detection is commonly confused with image recognition, so before we proceed, we must clarify the distinctions between them. Image recognition assigns a label to an image, a picture of a dog receives the label "dog", and a picture of two dogs still receives the label "dog". Object detection, on the other hand, draws a box around each dog and labels the box "dog". The model predicts where each object is and what label should be applied. In that way, object detection provides more information about an image than recognition. [7] Here's an example of how this distinction looks in practice:

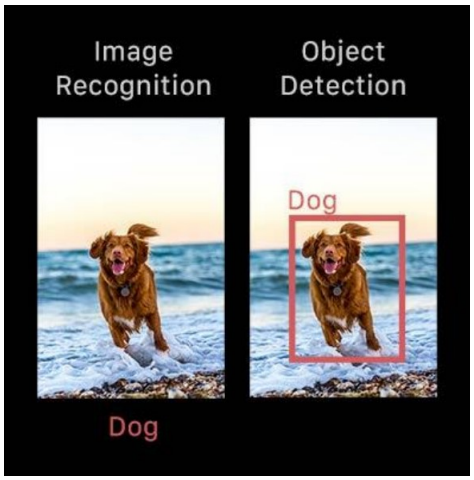


Figure 1.1: A picture shows the difference between image recognition and object detection. [7]

1.2.2 Models and types of object detection

Broadly speaking, object detection can be broken down into machine learning-based approaches and deep learning-based approaches. In more traditional ML-based approaches, computer vision techniques are used to look at various features of an image, such as the color histogram or edges, to identify groups of pixels that may belong to an object. These features are then fed into a regression model that predicts the location of the object along with its label. On the other hand, deep learning-based approaches employ convolutional neural networks (CNNs) to perform end-to-end, unsupervised object detection, in which features don't need to be defined and extracted separately. For a gentle introduction to CNNs, check out this overview. Because deep learning methods have become the state-of-the-art approaches to object detection, these are the techniques we'll be focusing on for this guide. [7]

1.2.3 Why is object detection important?

Object detection is inextricably linked to other similar computer vision techniques like image recognition and image segmentation, in that it helps us understand and analyze scenes in images or video. But there are important differences. Image recognition only outputs a class label for an identified object, and image segmentation creates a pixel-level understanding of a scene's elements. What separates object detection from these other tasks is its unique ability to locate objects within an image or video. This then allows us to count and then track those objects.[7]

1.2.4 Basic structure

Deep learning-based object detection models typically have two parts. An encoder takes an image as input and runs it through a series of blocks and layers that learn to extract statistical features used to locate and label objects. Outputs from the encoder are then passed to a decoder, which predicts bounding boxes and labels for each object. The simplest decoder is a pure regressor. The regressor is connected to the output of the encoder and predicts the location and size of each bounding box directly. The output of the model is the X, and Y coordinate pair for the object and its extent in the image. Though simple, this type of model is limited. You need to specify the number of boxes ahead of time. If your image has two dogs, but your model was only designed to detect a single object, one will go unlabeled. However, if you know the number of objects you need to predict in each image ahead of time, pure regressor-based models may be a good option. An extension of the regressor approach is a region proposal network. In this decoder, the model proposes regions of an image where it believes an object might reside. The pixels belonging to these regions are then fed into a classification subnetwork to determine a label (or reject the proposal). It then runs the pixels containing those regions through a classification network. The benefit of this method is a more accurate, flexible model that can propose arbitrary numbers of regions that may contain a bounding box. The added accuracy, though, comes at the cost of computational efficiency.

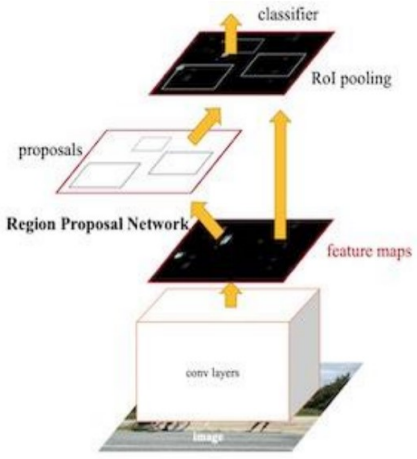


Figure 1.2: Object Detection Structure

Single shot detectors (SSDs) seek a middle ground. Rather than using a subnetwork to propose

regions, SSDs rely on a set of predetermined regions. A grid of anchor points is laid over the input image, and at each anchor point, boxes of multiple shapes and sizes serve as regions. For each box at each anchor point, the model outputs a prediction of whether or not an object exists within the region and modifications to the box's location and size to make it fit the object more closely. Because there are multiple boxes at each anchor point and anchor points may be close together, SSDs produce many potential detections that overlap. Post-processing must be applied to SSD outputs in order to prune away most of these predictions and pick the best one. The most popular post-processing technique is known as non-maximum suppression.

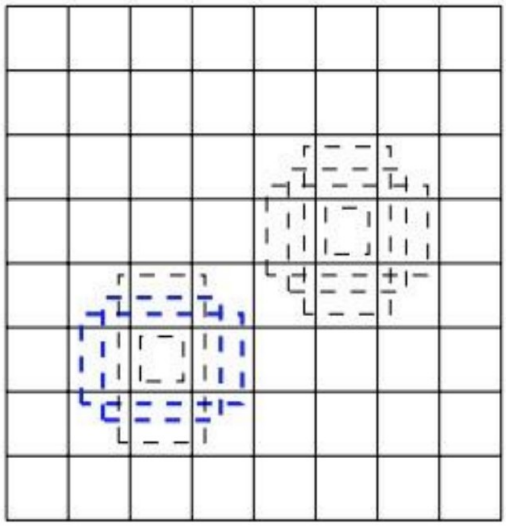


Figure 1.3: Intersection Area

Finally, a note on accuracy. Object detectors output the location and label for each object, but how do we know how well the model is doing? For an object's location, the most commonly-used metric is intersection-over-union (IOU). Given two bounding boxes, we compute the area of the intersection and divide it by the area of the union. This value ranges from 0 (no interaction) to 1 (perfectly overlapping). For labels, a simple "percent correct" can be used. [7]

1.3 HSV Model and Color Detection

1.3.1 What is color detection?

Color detection is the process of detecting the name of any color. Simple, isn't it? Well, for humans this is an extremely easy task but for computers, it is not straightforward. Human eyes and brains work together to translate light into color. Light receptors that are present in our eyes transmit the signal to the brain. Our brain then recognizes the color. [8]

1.3.2 HSV Model

Monochromatic color means light of a single wavelength. We will use the video, captured using a webcam as input and try to detect objects of a single color. To achieve that it's highly recommended to convert the colors from RGB format to HSV. HSV stands for hue, saturation, and value. Hue signifies the color itself, technically it denotes the angle in the HSV color model. To understand the HSV color model more. Imagine a cylinder, it has a height, a radius, and a curved surface area. Take the initial position of the radius as a reference and move the radius anticlockwise to an angle , i.e. is the angle between the current position of the radius with its initial position. If is 0 degrees then it denotes the color red, if is 120 degrees then it denotes green, and so on. The length of the radius is the saturation of the color and the height signifies the value or V in HSV.

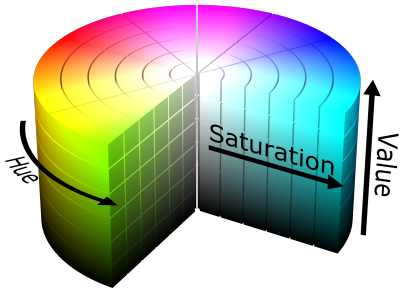


Figure 1.4: HSV Color Space

The main idea is to convert the input RGB image (BGR in the case of OpenCV because that's how images are formatted in this module) to HSV format, which will make it easier for us to mask the specific color out of the frame. That is, whatever color in the provided HSV range will be given a value of 255 and others will be simply 0, and as a result, every object with color in the specified range will change to white leaving the rest of the image i.e., background black. [9]

1.3.3 Summary

To summarize, the concept of a robot system has been around for a long time to replace humans in performing repetitive and dangerous tasks and the world knew that this field is no longer the future of technology, but also the future of business. Our color sorting robot is a low-cost system with the simplest concepts to implement sorting effectively, saving working time, and it's one of the fastest systems in Industrial applications that provides workers safety, fewer human mistakes, and improved efficiency and productivity, as in the other hand our system can help people with disabilities to interact more with the outer environment. In the next chapter, we are going to talk about the theoretical background and hardware overview of our project. Let's jump in!

Chapter 2

Theoretical background & Hardware overview

Contents

2.1 Introduction

The system consists of the microcomputer unit, with the Raspberry pi camera, servo motors (MG996R, MG90s), PCA9685 16 Channel 12 Bit PWM Servo Driver, Ultrasonic distance sensor, L293N motor driver, DC Motor.Li-Po battery and 1 8 Cell Lithium Battery Level Indicator Module. The color sorting robot is running with a Raspberry pi 4 model B board as a principal microcomputer unit to control the whole system from the movements of the arm robot to the control the directions of the car's motion. The raspberry pi works based on a Linux OS that uses Debian which is a Linux distribution and OpenCV-Python designed to solve computer vision problems. In the body of this chapter, all the types of hardware and how they function all together with the different softwares used will be presented.

2.2 Hardware part

2.2.1 Raspberry pi 4 model B

Raspberry Pi is a series of small single-board computers (SBCs) using the Linux operating system developed in the United Kingdom by Upton. The Raspberry Pi project originally leaned towards the promotion of teaching basic computer science in schools and was commonly used to run larger and smart programs to achieve output quickly meanly in robotics. The name Raspberry Pi is derived from the fruit pie, raspberry pie. This is because many companies in the computer neighborhood where Raspberry Pi was based used fruit names such as Apple and apricot as names for their companies and products whereas "Pi" is associated with the programming language "Python." [10] Raspberry Pi 4 Model B shown in figure 3.7 is the latest product in the popular Raspberry Pi range of computers. It offers ground-breaking increases in processor speed, multimedia performance, memory, and connectivity. This product's key features include a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a pair of micro-HDMI ports, hardware video decode at up to 4Kp60, up to 8GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, and PoE capability (via a separate PoE HAT add-on), more information figure 2.2. [11] A Raspberry Pi 4 board has 40 pins on it, as can be seen in figure 2.3. Among these pins, there are four power pins, two of which are 5v pins and another two are 3V3 pins. The 5V is directly connected to the USB port but 3V3 is connected to the regulator which gives the stable 3 volts output. Raspberry Pi 4 has 8 ground pins that are connected internally and any ground pin can be used by the power supply or external device to make the common ground. The remaining 28 GPIO pins are labeled starting from GPIO 0 and going up to GPIO 27. The GPIO pins can be programmed to be output pins or input pins. The GPIO pins can be digitally

programmed so that they can be turned ON or OFF. The output of any GPIO pin is 3.3v and can be used to control output components like an LED or a motor. These ON/OFF conditions can also be interpreted as a Boolean True/False, 1/0, or HIGH/LOW. Some of these pins also have a dual function. For example, pin 3 or GPIO 2 also acts as an I2C pin. [12]

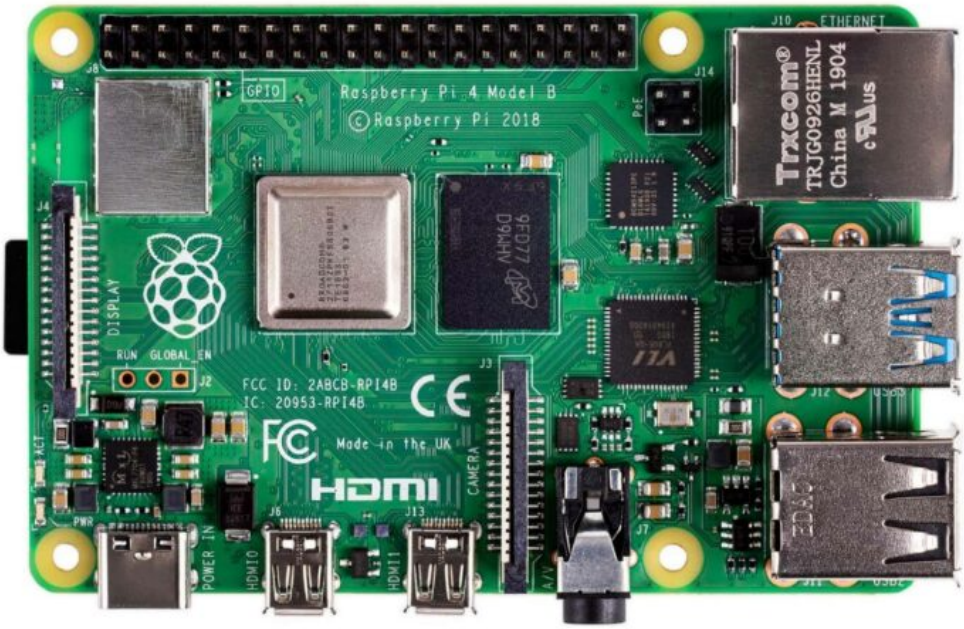


Figure 2.1: Raspberry pi 4 model B board. [11]

Processor:	Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
Memory:	1GB, 2GB, 4GB or 8GB LPDDR4 (depending on model) with on-die ECC
Connectivity:	2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless LAN, Bluetooth 5.0, BLE Gigabit Ethernet 2 × USB 3.0 ports 2 × USB 2.0 ports.
GPIO:	Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
Video & sound:	2 × micro HDMI ports (up to 4Kp60 supported) 2-lane MIPI DSI display port 2-lane MIPI CSI camera port 4-pole stereo audio and composite video port
Multimedia:	H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
SD card support:	Micro SD card slot for loading operating system and data storage
Input power:	5V DC via USB-C connector (minimum 3A[1]) 5V DC via GPIO header (minimum 3A[1]) Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature 0–50°C

Figure 2.2: Specification of a Raspberry pi 4 model B board. [11]

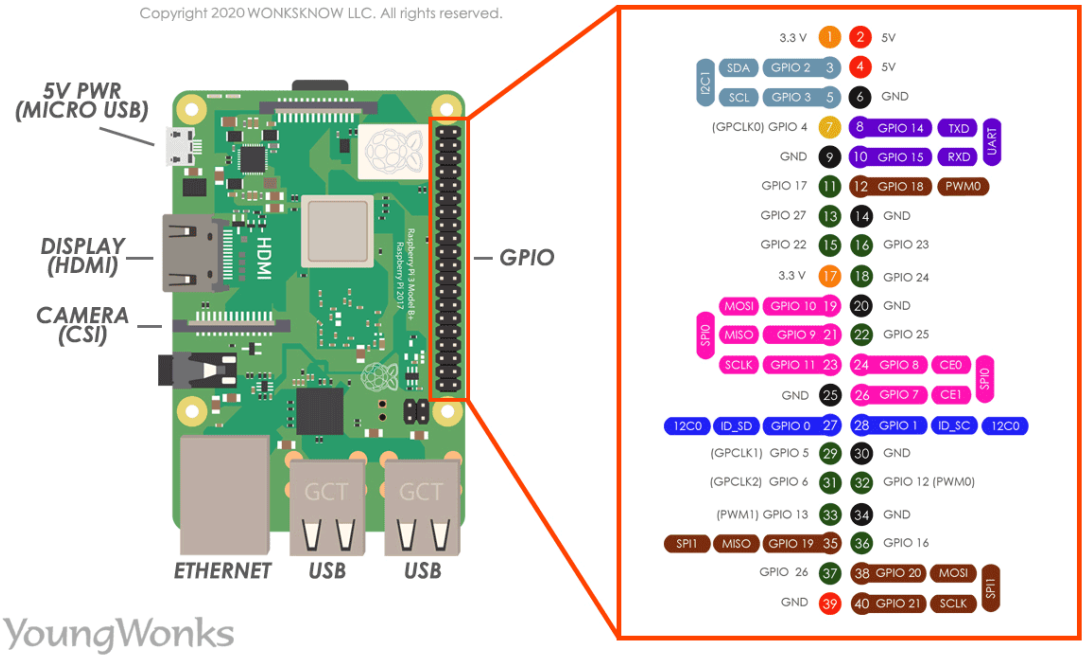


Figure 2.3: pins on a Raspberry Pi 4 Model B board.[13]

2.2.2 Raspberry Pi Camera v2

The Raspberry Pi Camera v2 is a high quality 8-megapixel Sony IMX219 image sensor custom designed add-on board for Raspberry Pi. It attaches to Pi by way of one of the small sockets on the board upper surface by a short ribbon cable. The board itself is tiny, at around 25mm x 23mm x 9mm and its weighs just over 3g, making it perfect for robot or other applications where size and weight are important. In terms of still images, the camera is capable of 3280 x 2464-pixel static images, and also supports 1080p30, 720p60 and 640x480p90 video.

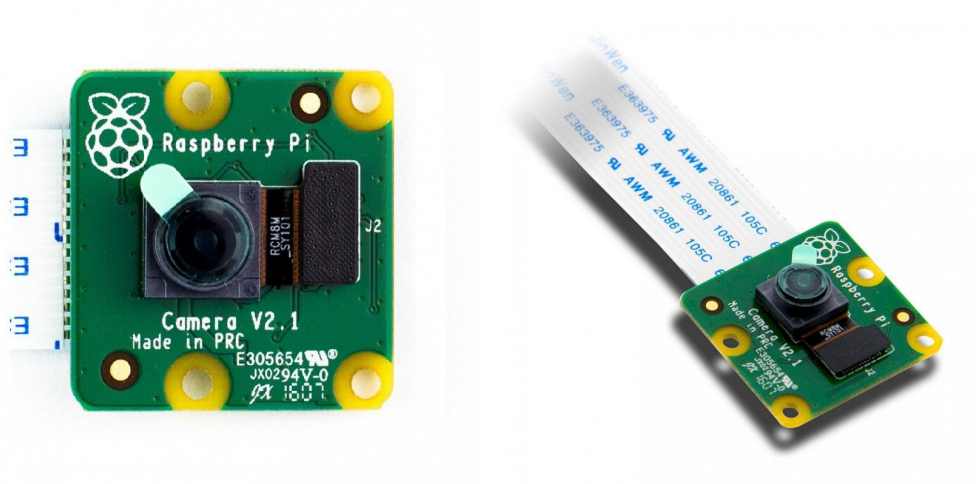


Figure 2.4: Camera R Pi module. [14]

2.2.3 lithium polymer battery (LiPo)

Lipo is a rechargeable battery of lithium-ion technology using a polymer electrolyte instead of a liquid one. High conductivity semisolid polymers form this electrolyte. These lipo batteries provide a higher specific energy than other lithium-battery types. It is a newer type of battery now used in many consumer electronics devices.[15]

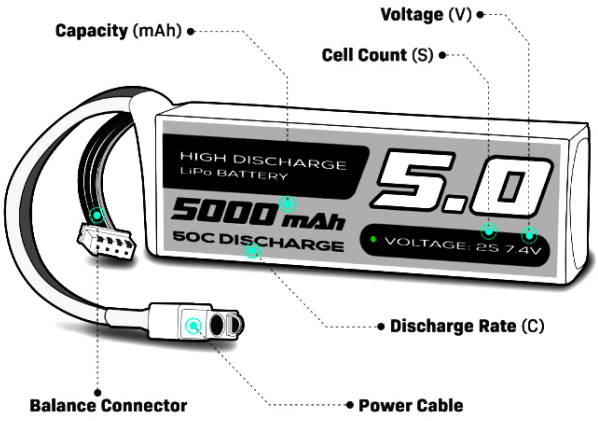


Figure 2.5: Li-po battery specifications

2.2.4 1-8 Cell Lithium Battery Level Indicator Module

Lithium battery capacity level indicator for 1 to 8 cells provides a quick visual reference and easy-to-read voltage status and charge status (capacity) of Lithium-ion batteries with a maximum voltage of 4.2 volts per cell. The Lithium Battery Capacity Indicator color is red, and the display bars are blue.

- Display Electricity Quantity Parameter:

Note: N represents battery quantity
When the battery voltage is over N*3.3V, it will illuminate 1 block electricity quantity.
When the battery voltage is over N*3.5V, it will illuminate 2 blocks of electricity quantity.
When the battery voltage is over N*3.7V, it will illuminate 3 blocks of electricity quantity.
When the battery voltage is over N*3.9V, it will illuminate 4 blocks of electricity quantity.
When the battery voltage is less than N*3.3V,4 blocks display will be off; it represents the battery is less than 3.3V, and you can charge the battery. [16]

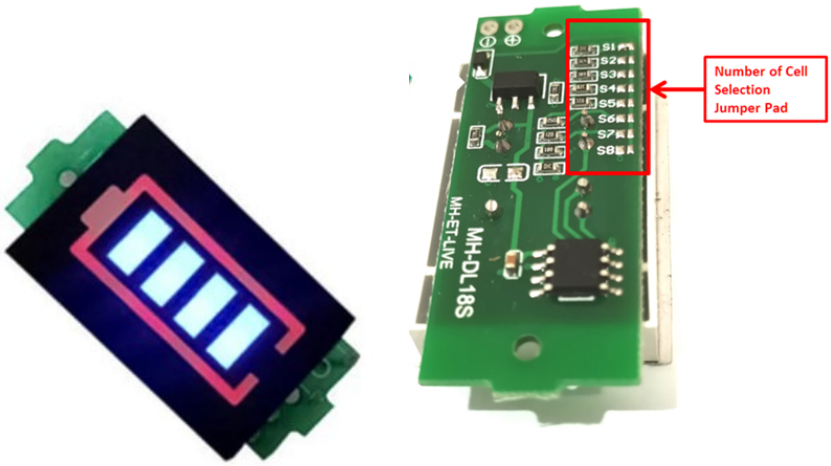
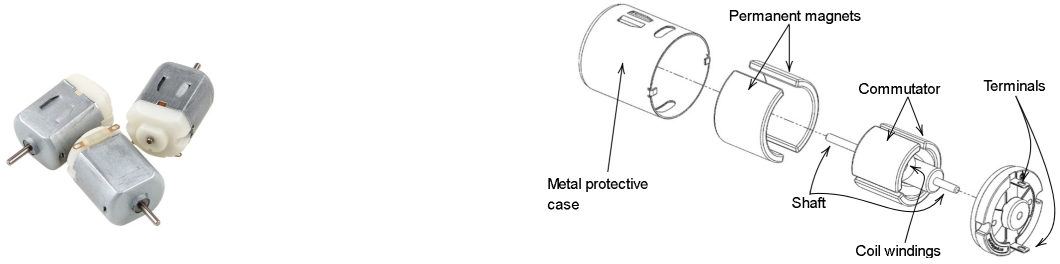


Figure 2.6: 1 8-Cell-Lithium Battery Level Indicator Module and the user configuration. [17]

2.2.5 12V DC Motor

A direct current (DC) motor is a type of electric machine that takes electrical power through direct current and converts this energy into mechanical rotation. DC motors include two key components: a stator and an armature. The stator is the stationary part of a motor that provides a rotating magnetic field, while the armature rotates. A simple DC motor uses a stationary set of magnets in the stator, and a coil of wire with a current running through it to generate an electromagnetic field aligned with the center of the coil. One or more windings of insulated wire are wrapped around the core of the motor to concentrate the magnetic field and they are connected to a commutator (a rotary electrical switch), that applies an electrical current to the windings. The commutator allows each armature coil to be energized in turn, creating a steady rotating force (known as torque).[18]



(a) **(b)**

Figure 2.7: (a) DC Motor Module. (b) Internal parts of a DC motor. [19]

2.2.6 L298N motor driver

This L298N Motor Driver Module is a high-power motor driver module used in robotics projects that involves driving DC and Stepper Motors. The L298N module can control up to 4 DC motors, or 2 DC motors with directional and speed control. This module consists of an L298 motor driver IC, a 78M05 5V regulator, resistors, capacitor, Power LED, and 5V jumper in an integrated circuit. [20]

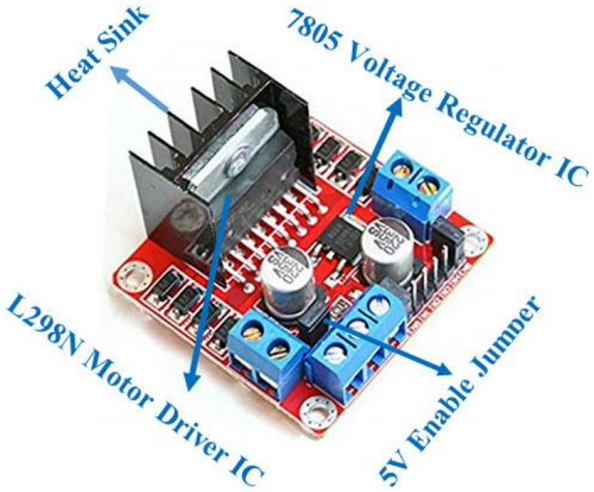


Figure 2.8: L298N Motor Driver Module component. [20]

- L298N Module Pinout Configuration:

IN1 & IN2: Motor A input pins. Used to control the spinning direction of Motor A.
IN3 & IN4: Motor B input pins. Used to control the spinning direction of Motor B.
ENA: Enables PWM signal for Motor A.
ENB: Enables PWM signal for Motor B.

OUT1 & OUT2: Output pins of Motor A.
OUT3 & OUT4: Output pins of Motor B.
12V: 12V input from DC power Source.
5V: Supplies power for the switching logic circuitry inside L298N IC.
GND: Ground pin.[20]

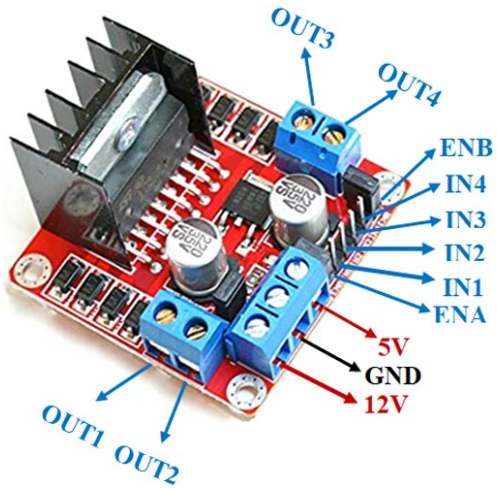


Figure 2.9: L298N Motor Driver Module Pinout. [20]

For the use of this module, a python code is proposed as **GPIO** library which is a Python library to use the L298N motor driver with a Raspberry Pi, and the code is as follows:

L298N Motor Driver with GPIO Library code example

```
import RPi.GPIO as gpio
import time


def init():
    gpio.setmode(gpio.BCM)
    gpio.setup(17, gpio.OUT)
    gpio.setup(22, gpio.OUT)
    gpio.setup(23, gpio.OUT)
    gpio.setup(24, gpio.OUT)


def forward(sec):
    init()
    gpio.output(17, False)
    gpio.output(22, True)
    gpio.output(23, True)
    gpio.output(24, False)
    time.sleep(sec)
    gpio.cleanup()
```

2.2.7 Servo motor

A servo motor is an electromechanical device, consisting of three key elements – a motor, a feedback device, and control electronics, that produces torque and velocity based on the supplied current and voltage. A servo motor works as part of a closed-loop system providing torque and velocity as commanded by a servo controller utilizing a feedback device to close the loop. The feedback device supplies information

such as current, velocity, or position to the servo controller, which adjusts the motor action depending on the commanded parameters.[21]

MG996R

The MG996R is a metal gear servo motor with a maximum stall torque of 11 kg/cm and a weight of 55g. The motor rotates from 0 to 180 degrees based on the duty cycle of the PWM wave supplied to its signal pin with a running Current between 500 mA – 900 mA (6V).[22]

Figure 2.10: MG996R servo motor module.[23]

MG90s

MG90S is a micro servo motor with metal gear, tiny and lightweight with high output power, thus ideal for Helicopters, Quadcopter, or Robot arms. durability. The Servo can rotate approximately 180 degrees (90 in each direction) with a maximum torque of 2.2kg/cm.[24]

Figure 2.11: MG90S servo motor module. [25]

- Wire configuration of servo motor:

  Brown: Ground wire connected to the ground of the system.
  Red: Powers the motor typically +5V is used.
  Orange: PWM signal is given in through this wire to drive the motor.[26]

- How a servo motor works:

You can control the servo motor by sending a series of pulses to the signal line, a conventional analog servo motor expects to receive a pulse roughly every 20 milliseconds (i.e., the signal should be 50Hz). The length of the pulse determines the position of the servo motor as shown in the figure below:

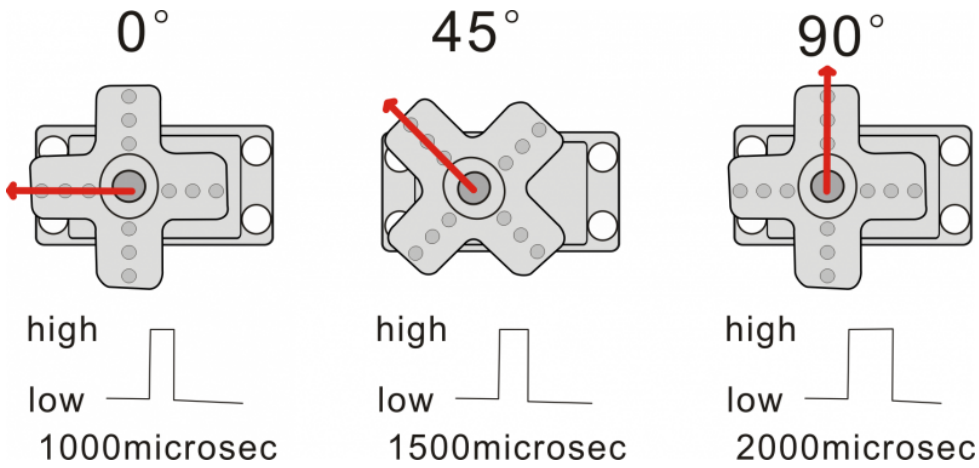


Figure 2.12: servo motor pulse modulation.[27]

If the pulse is high for 1ms, then the servo angle will be zero.10
If the pulse is high for 1.5ms, then the servo will be at its center position.
If the pulse is high for 2ms, then the servo will be at 180 degrees.
Pulses ranging between 1ms and 2ms will move the servo shaft through the full 180 degrees of its travel.[28]

2.2.8 PCA9685 16 Channel 12 Bit PWM Servo Driver

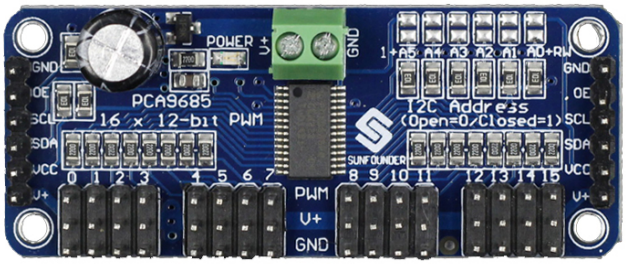


Figure 2.13: PCA9685 16 channel 12bit PWM servo controller module.[29]

The PCA9685 16-channel 12-bit PWM servo motor driver is used to control up to 16 servo motors using I2C communication with microcontrollers like Raspberry Pi or Arduino. The PCA9685 is a 12-bit, 16-channel, I2C bus controller that delivers its output independently at 4096 steps (12-bit resolution), but all with the same frequency. This frequency is programmable from 24Hz to 1526Hz, and the duty cycle is adjustable from 0-100% it's a 5V compliant, which means it could be controlled from a 3.3V microcontroller, which is good in controlling white or blue LEDs with a 3.4V+ forward voltage.[29]

For the use of this module,**Adafruit PCA9685** which is a Python library to use the PCA9685 PWM servo/LED controller with a Raspberry Pi.[30]

Adafruit PCA9685 code example

```
import time
from board import SCL, SDA
import busio
from adafruit_motor import servo
from adafruit_pca9685 import PCA9685

i2c = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class
    instance.
pca = PCA9685(i2c)
pca.frequency = 50
servo7 = servo.Servo(pca.channels[7])

# We sleep in the loops to give the
    servo time to move into position.
for i in range(180):
    servo7.angle = i
    time.sleep(0.03)
for i in range(180):
    servo7.angle = 180 - i
    time.sleep(0.03)
pca.deinit()
```

We used PCA also in our project in order to control the leds that are placed on the edges of the vehicle .

2.2.9 Ultrasonic distance sensor

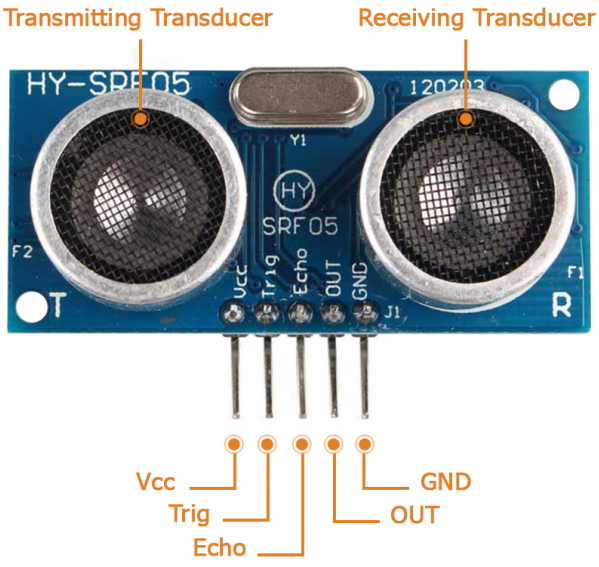


Figure 2.14: Ultrasonic Distance Sensor Pinout. [31]

The Ultrasonic Distance Sensor is a sensor used in obstacle avoidance systems that uses non-contact ultrasound sonar consisting of two ultrasonic transducers (transmitter, receiver). Ultrasonic measures the distance to an object using ultrasonic sound waves by measuring the time it took for the reflected waves

to return to the receiver and then will convert the reflected sound into an electrical signal. The sensor is small, easy to use in any robotics project, and offers excellent non-contact range detection between 2 cm to 400 cm with an accuracy of 3mm. Since it operates on 5 volts, it can be hooked directly to a Raspberry Pi or any other 5V logic microcontrollers.[32]

ultrasonic distance sensor code example

```
from gpiozero import DistanceSensor

ultrasonic = DistanceSensor(echo=17,
    trigger=4)
while True:
    print(ultrasonic.distance)
```

]

2.3 Software part

2.3.1 Solidworks

The SOLIDWORKS software is a mechanical design automation application that lets designers develop mechatronic systems from beginning to end, experiment with features and dimensions, and produce models and detailed drawings. SOLIDWORKS was developed by MIT graduate Jon Hirschtick and was bought by Dassault Systems in 1997. The software now encompasses several programs that can be used for both 2D and 3D design. SOLIDWORKS, at the initial stage, is used for planning, visual ideation, modeling, feasibility assessment, prototyping, and project management. The software is then used for the design and building of mechanical, electrical, and software elements. Finally, the software can be used for management, including device management, analytics, data automation, and cloud services, and be used by mechanical, electrical, and electronics engineers to form a connected design. [33]

Figure 2.15: SOLIDWORKS' logo.[34]

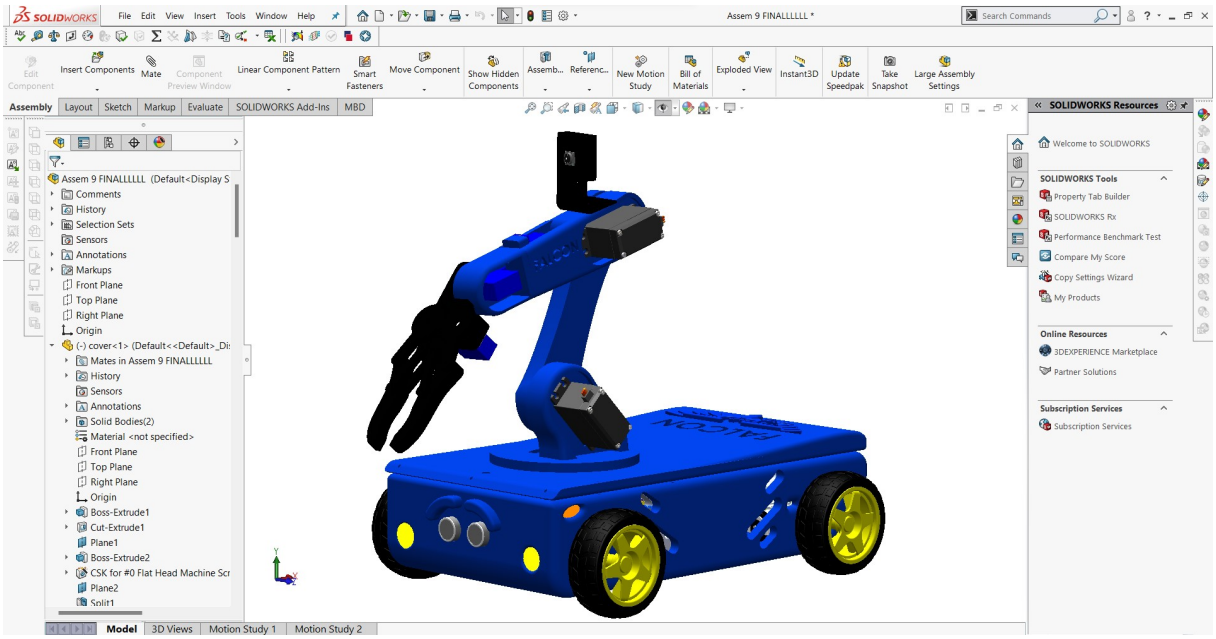


Figure 2.16: FALCON inside SOLIDWORKS!

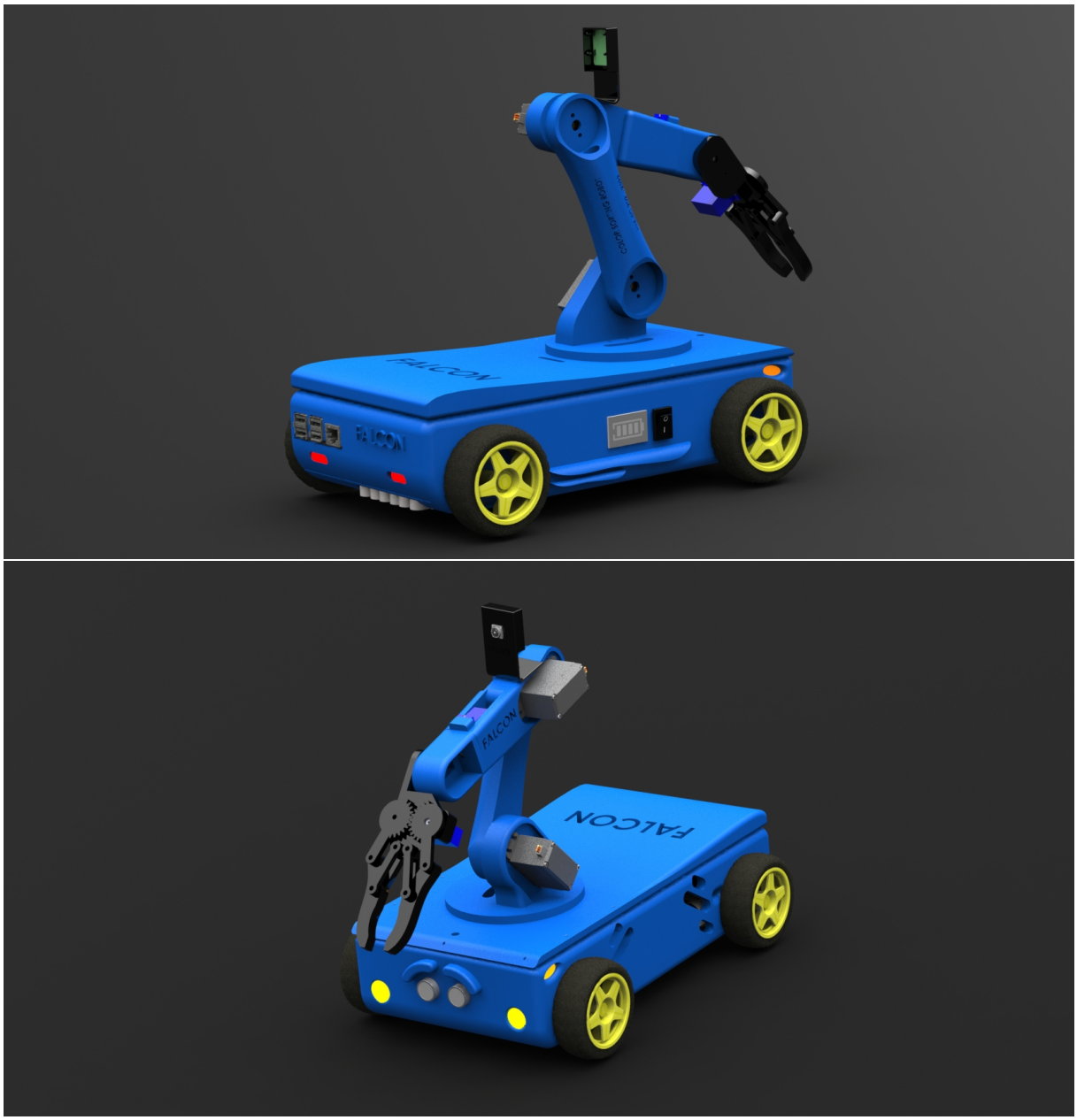


Figure 2.17: Renders of the Color Sorting Robot "FALCON" using SOLIDWORKS.

2.3.2 Printing and Manufacturing the Robot

3D printing used to create three-dimensional objects through a layering method.Sometimes referred to as additive manufacturing.

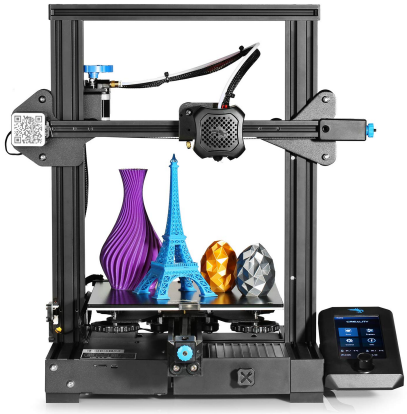


Figure 2.18: Creality Ender 3 V2 3D Printer [35]

Thus after printing the parts, the next phases are Refinements,Painting and Assembly:

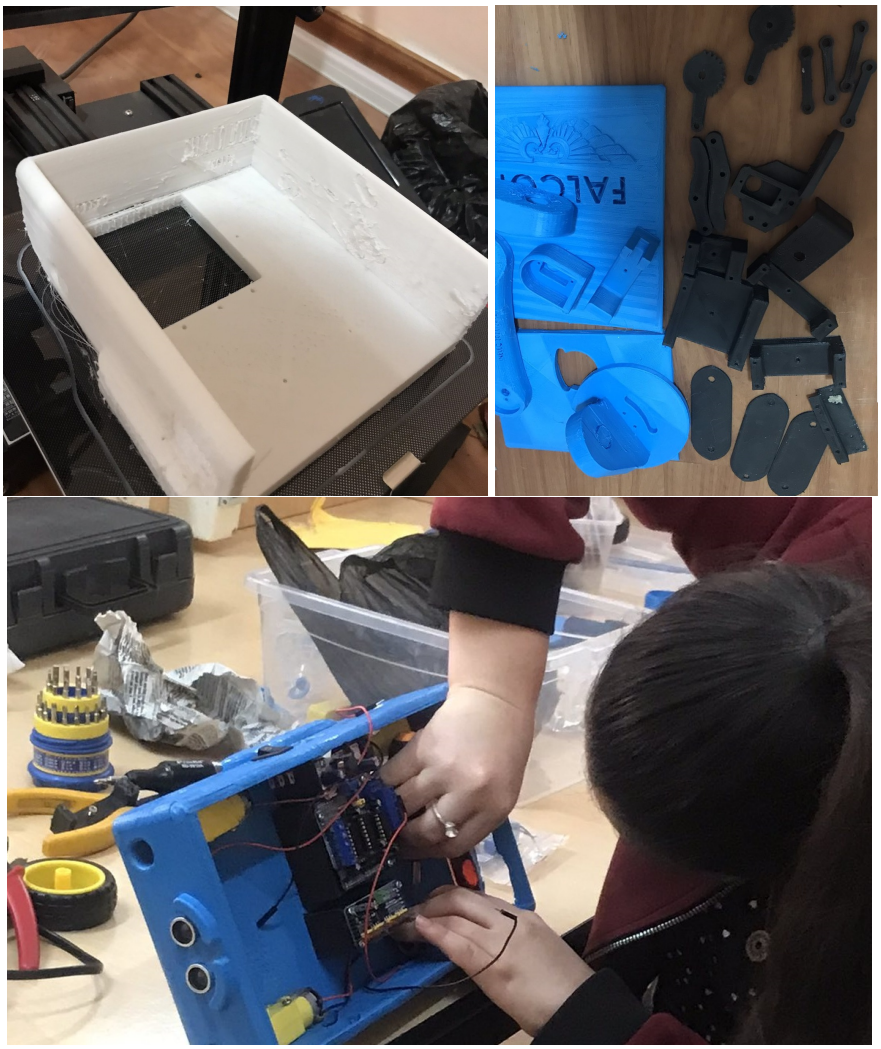


Figure 2.19: FALCON Manufacturing Process

2.3.3 OpenCV with python for color detection

OpenCV was started at Intel in 1999 by Gary Brodsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Brodsky to manage Intel's Russian OpenCV team. OpenCV supports a wide variety of programming languages such as C++, Python, Java, etc., and is available on different platforms including Windows, Linux, OS X, Android, and iOS. The raspberry pi works based on a Linux OS that uses Debian which is a Linux distribution and OpenCV-Python which is a library of Python bindings designed to solve computer vision problems mainly color detection. Python is a high-level, interpreted, general-purpose programming language, started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability. Color detection is important to recognize objects, it is also utilized as a tool in various image editing and drawing applications and is mostly in demand in computer vision. A color detection algorithm identifies pixels in an image that match a specified color or color range, the color of detected pixels can then be changed to distinguish them from the rest of the image. [36]

Figure 2.20: OpenCV with python's logo.[37]

2.4 Summary

In this chapter, the different components that have been used in this color sorting robot had been represented and given a general overview of their working principle. The next chapter is going to deal with the design and development of the system.

Chapter 3

Coding & Features

Contents

3.1 System Architecture

The electronic part is one of the vital parts in the development of the robot. It includes several components that can be mentioned as servo motors, microcontrollers, DC motors with wheels, servo and motor drivers, camera R pi, and a battery level indicator. Figure 3.1 shows the block diagram of the robot operation, which consists of the camera R pi as input of the system which is on the top of the arm robot to scan through the area in order to detect any closest object and classify it based on its color conditions. Raspberry pi 4 model B is used as a microcontroller that is connected with other components: motor driver (L293N) is used to activate the moving of the gear motors on the other hand the PCA9685 16 Channel 12 Bit PWM Servo Driver is used to activate the servo motors that are placed in the arm robot. The robot when detects the object's color, moves on its own. The coding is been done in Python language in the Raspbian software and is stored upon an SD card. Raspberry PI has GPIO (General Purpose Input/Output) pins that can be commanded to turn high or low in the program, and it is responsible for controlling the motors through the motor driver circuit. To rotate to either right or left direction, one motor will remain off and the other one will move, thus resulting in rotation of the body, for the movement of each of the servo motors, the duty cycle value needs to be adjusted so that the required rotation is obtained. Whereas for each torque corresponding to the wheel, 0 or 1 needs to be specified. As the object is detected by the camera, it scans it and detects what color is it, the robot moves forward to the object slowly, stops at a certain limit to avoid running over it, then the arm is taken care of picking it up and dropping it in the specified location according to the color.

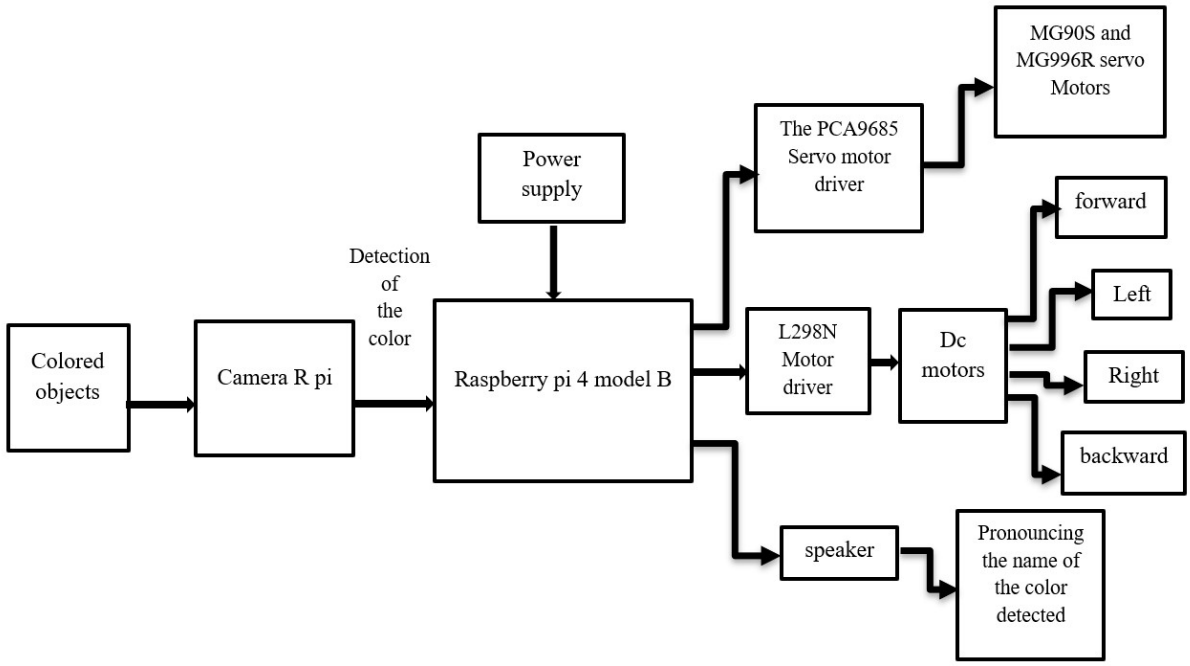


Figure 3.1: Block diagram of the robot

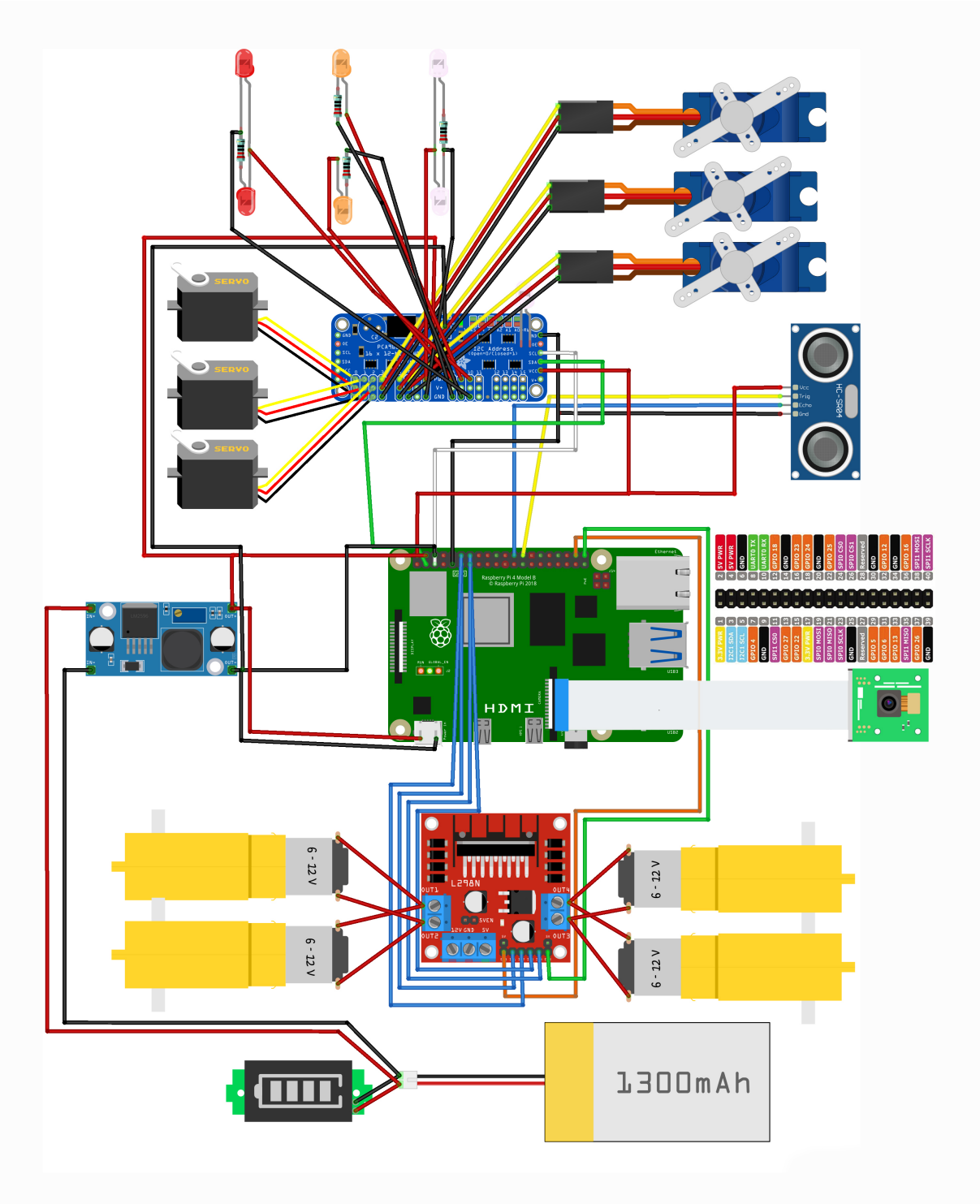


Figure 3.2: Overall Circuit

3.1.1 Inverse kinematics (IK) algorithm

Inverse kinematics is the use of kinematic equations to determine the motion of a robot to reach a desired position. For example, to perform automated bin picking, a robotic arm used in a manufacturing line needs precise motion from an initial position to a desired position between bins and manufacturing machines. The grasping end of a robot arm is designated as the end-effector. The robot configuration is a list of joint positions that are within the position limits of the robot model and do not violate any constraints the robot has.[38]

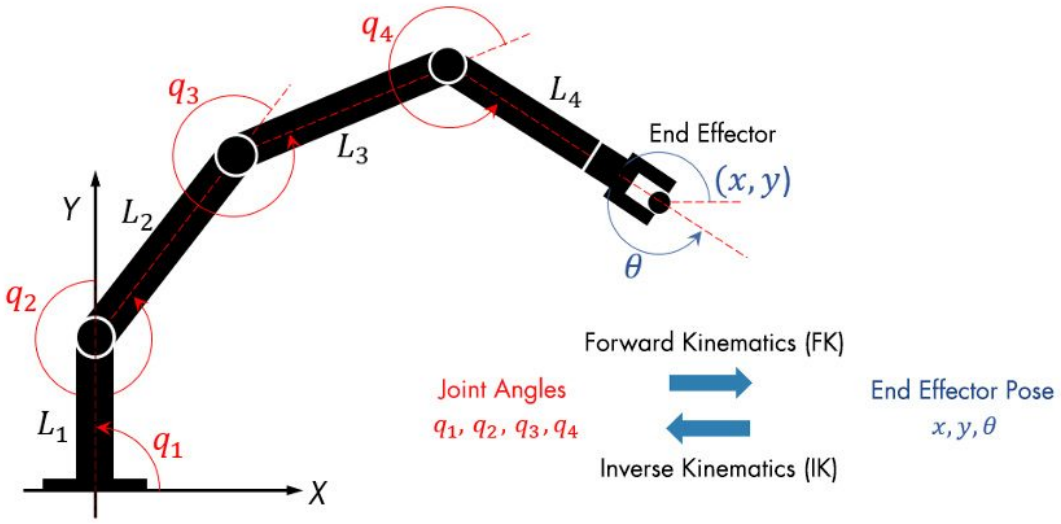


Figure 3.3: Configuring the joint positions of a robot using forward or inverse kinematics [38]

Given the desired robot's end-effector positions, inverse kinematics (IK) can determine an appropriate joint configuration for which the end-effectors move to the target pose. with Solidworks we can measure precisely the joints:

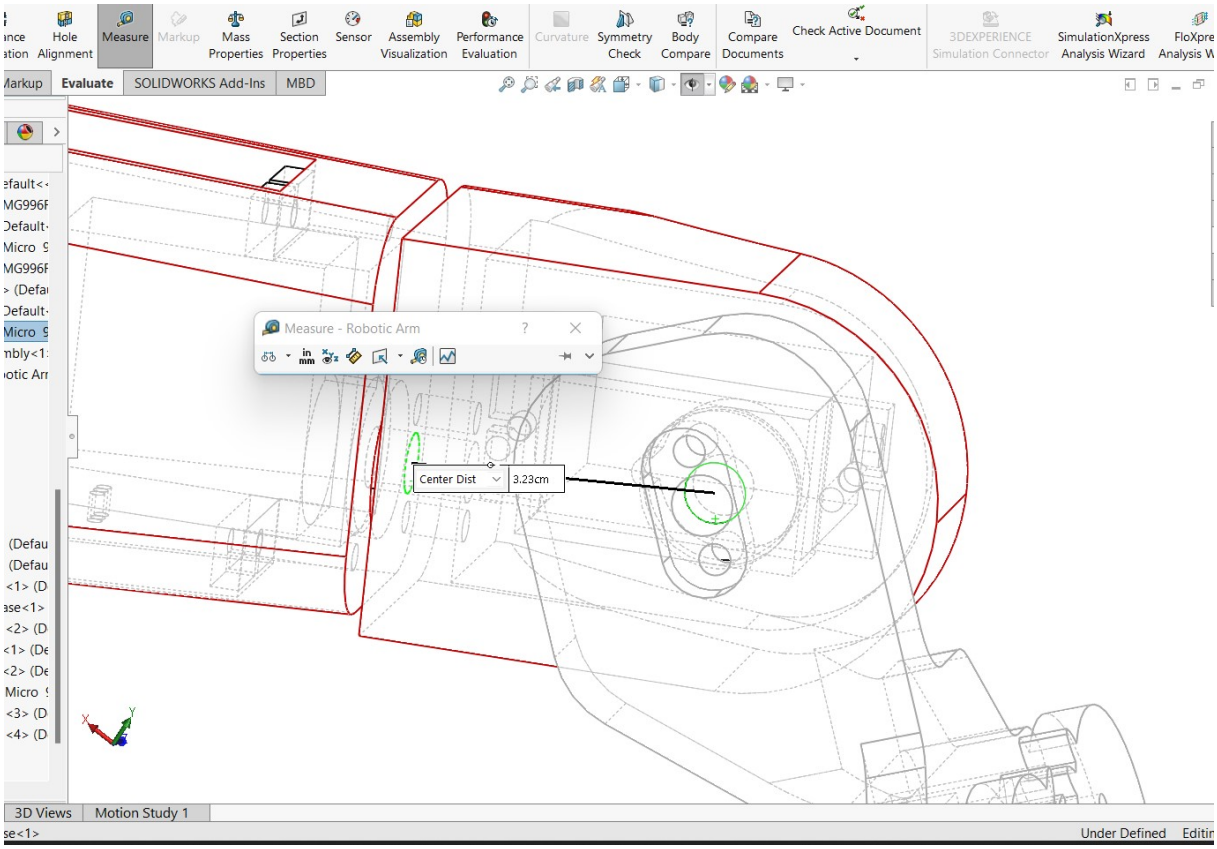


Figure 3.4: Measuring the joints lengths

using **tinyik** library to calculate the joint angles from x and y,z values then grab the objects:

tinyik library example code

```
import tinyik
import numpy as np

Falcon = tinyik.Actuator(  # set the joint lenghts with axes of
    rotation
    [
        "z",
        [0.0, 0.0, 0.25],
        "y",
        [12, 0.0, 0.0],
        "y",
        [9.05, 0.0, 0.0],
        "x",
        [3.23, 0.0, 0.0],
        "y",
        [4.15, 0.0, 0.0],
    ]
)

# Since the joint angles are zero by default
print(Falcon.angles)

# Set a position of the end-effector to calculate the joint
    angles:
Falcon.ee = [10, 10, 90]
Falcon.angles = np.round(np.rad2deg(Falcon.angles))
print(Falcon.angles)
```

Terminal

```
[0. 0. 0. 0. 0.]
[ 45. -81. -0. -207. -0.]
```

3.1.2 Color Detection algorithm

The Algorithm:

a color object detected by reading its coordinates and measuring the x and y distances from the center of the Frame.Hence, we can know the deviation of the object to generate motion commands to minimize this deviation with respect to the Tolerance zone. also the Speed of the robot is adjusted during runtime using this following formula:

$$new.speed = min.speed + \frac{(max.speed - min.speed) \times |x.distance|}{hW}$$

» where hW is the half width of the Frame

So the farther the object from the center lines the more is the speed and the larger the Tolerance the sooner the Robot will acquire the object but with reduced accuracy. see Figure 3.5 bellow :

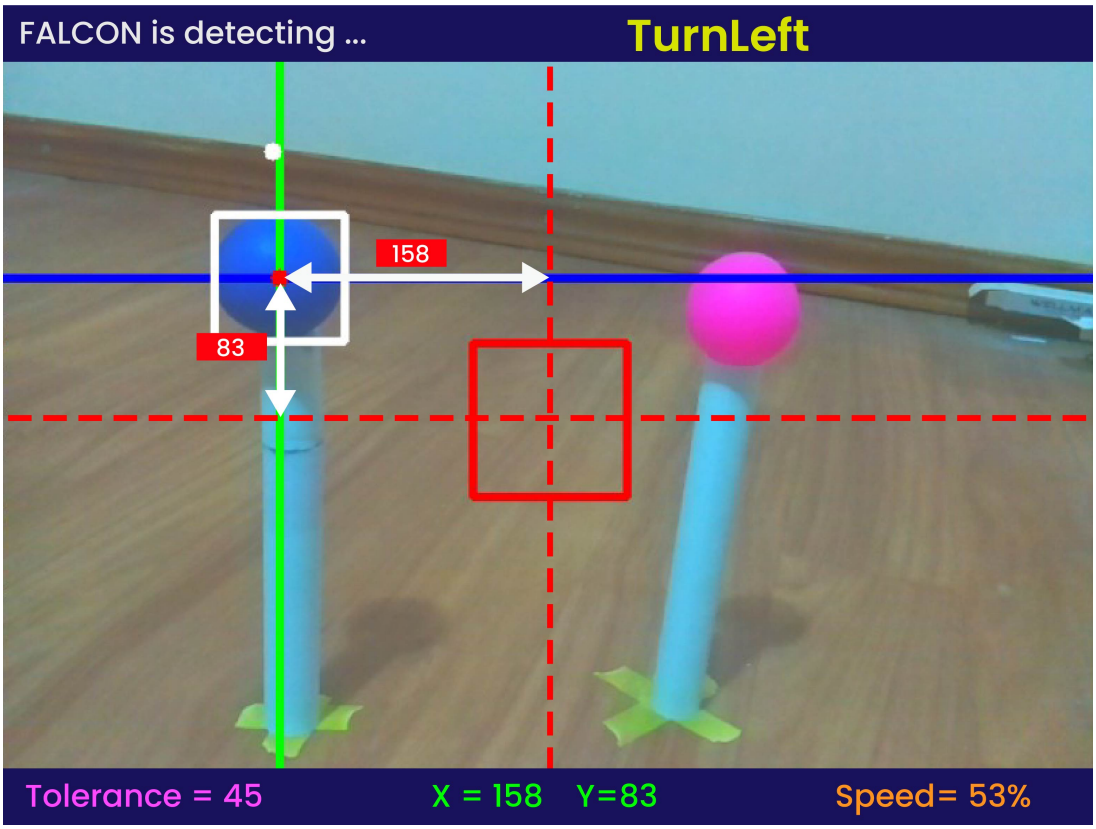


Figure 3.5: Colored Object Detection Algorithm

3.1.3 Codes

Here are the Codes done to achieve the aim mentioned above:
All libraries and necessary codes are gathered in this file as functions to be called in the main code:

FALCON file for calling functions

```
### Import Libraries:
import time
import tinyik
import numpy as np
import RPi.GPIO as gpio
import pyttsx3  # Text to Speech Library

engine = pyttsx3.init()  # object creation

### Import the PCA9685 module:
import Adafruit_PCA9685
from board import SCL, SDA
import busio
from adafruit_motor import servo
from adafruit_pca9685 import PCA9685

i2c = busio.I2C(SCL, SDA)  # Initialise the PCA9685 using the default address (0x40
    ).
pwm = Adafruit_PCA9685.PCA9685()  # Create a simple PCA9685 class instance.
pca = PCA9685(i2c)
pca.frequency = 60
```

```
21
22 # Servo Channels:
23 servo0 = servo.Servo(pca.channels[0])
24 servo1 = servo.Servo(pca.channels[1])
25 servo2 = servo.Servo(pca.channels[2])
26 servo3 = servo.Servo(pca.channels[3])
27 servo4 = servo.Servo(pca.channels[4])
28 servo5 = servo.Servo(pca.channels[5])
29
30 # LED Channels:
31 LedR = servo.Servo(pca.channels[8])
32 LedL = servo.Servo(pca.channels[9])
33 LedF = servo.Servo(pca.channels[10])
34 LedB = servo.Servo(pca.channels[11])
35 # Setting Channels:
36 Set0 = servo.Servo(pca.channels[13])
37 Set90 = servo.Servo(pca.channels[14])
38 Set180 = servo.Servo(pca.channels[15])
39
40 ###Variables:
41 # time:
42 tsll = 0.2
43 tsl = 0.7
44 tsh = 1
45 t1 = 3
46 # L298n:
47 speed = 100
48 en1 = 20
49 en2 = 21
50 in3 = 17
51 in4 = 22
52 in1 = 23
53 in2 = 24
54 # Setting DC Motors Controllers:
55 gpio.setmode(gpio.BCM)
56 gpio.setup(in1, gpio.OUT)
57 gpio.setup(in2, gpio.OUT)
58 gpio.setup(in3, gpio.OUT)
59 gpio.setup(in4, gpio.OUT)
60 gpio.setup(en1, gpio.OUT)
61 gpio.setup(en2, gpio.OUT)
62 pwm1 = gpio.PWM(en1, 100)
63 pwm2 = gpio.PWM(en2, 100)
64
65 # Speech Variables:
66 """ RATE"""
67 rate = engine.getProperty("rate")  # getting details of current speaking rate
68 print(rate)  # printing current voice rate
69 engine.setProperty("rate", 150)  # setting up new voice rate
70
71
72 """VOLUME"""
73 volume = engine.getProperty(
74     "volume"
75 )  # getting to know current volume level (min=0 and max=1)
76 print(volume)  # printing current volume level
77 engine.setProperty("volume", 10.0)  # setting up volume level  between 0 and 1
```

```

"""VOICE"""
voices = engine.getProperty("voices")  # getting details of current voice
# engine.setProperty('voice', voices[0].id)  #changing index, changes voices. o for
    male
engine.setProperty(
    "voice", voices[2].id
)  # changing index, changes voices. 1 for female

Falcon = tinyik.Actuator(  # set the joint lenghts with axes of rotation
    [
        "z",
        [0.0, 0.0, 0.25],
        "y",
        [12, 0.0, 0.0],
        "y",
        [9.05, 0.0, 0.0],
        "x",
        [3.23, 0.0, 0.0],
        "y",
        [4.15, 0.0, 0.0],
    ]
)
###Defining Functions:
# define Inverse Kinematics function
def IK(x, y, z):
    Falcon.ee = [x, y, z]
    Falcon.angles = np.round(np.rad2deg(Falcon.angles))
    return Falcon.angles


# Robotic Arm IK function
def ArmIk(a, b, c, d, e, f):
    # joint 0:
    servo0.angle = a
    # time.sleep(tsll)
    # joint 2:
    servo2.angle = c
    time.sleep(tsl)
    # joint 1:
    servo1.angle = b
    # time.sleep(tsll)
    # joint 3:
    servo3.angle = d
    # joint 4:
    servo4.angle = e
    # time.sleep(tsll)
    # joint 5:
    servo5.angle = f


# Set Camera view
def ArmCameraStart():
    print("CameraOne")
    # joint 0:
    print("Channel0")
    servo0.angle = 75
```

```
    time.sleep(tsll)
    # joint 1:
    print("Channel1")
    servo1.angle = 110
    time.sleep(tsll)
    # joint 2:
    print("Channel2")
    servo2.angle = 180
    time.sleep(tsl)
    # joint 3:
    print("Channel3")
    servo3.angle = 90
    # joint 4:
    print("Channel4")
    servo4.angle = 20
    time.sleep(tsll)
    # joint 5:
    print("Channel5")
    servo5.angle = 0


def Forward(speed):
    gpio.setmode(gpio.BCM)
    gpio.setup(in1, gpio.OUT)
    gpio.setup(in2, gpio.OUT)
    gpio.setup(in3, gpio.OUT)
    gpio.setup(in4, gpio.OUT)
    gpio.setup(en1, gpio.OUT)
    gpio.setup(en2, gpio.OUT)
    pwm1.start(speed)
    pwm2.start(speed)

    gpio.output(in1, False)
    gpio.output(in2, True)
    gpio.output(in3, False)
    gpio.output(in4, True)
    print("FORWARD")


def Backward(speed):
    gpio.setmode(gpio.BCM)
    gpio.setup(in1, gpio.OUT)
    gpio.setup(in2, gpio.OUT)
    gpio.setup(in3, gpio.OUT)
    gpio.setup(in4, gpio.OUT)
    gpio.setup(en1, gpio.OUT)
    gpio.setup(en2, gpio.OUT)
    pwm1.start(speed)
    pwm2.start(speed)

    gpio.output(in1, True)
    gpio.output(in2, False)
    gpio.output(in3, True)
    gpio.output(in4, False)
    print("BACKWARD")


```

```
191 def TurnRight(speed):
192     gpio.setmode(gpio.BCM)
193     gpio.setup(in1, gpio.OUT)
194     gpio.setup(in2, gpio.OUT)
195     gpio.setup(in3, gpio.OUT)
196     gpio.setup(in4, gpio.OUT)
197     gpio.setup(en1, gpio.OUT)
198     gpio.setup(en2, gpio.OUT)
199 
200     pwm1.start(speed)
201     pwm2.start(speed)
202 
203     gpio.output(in1, True)
204     gpio.output(in2, False)
205     gpio.output(in3, False)
206     gpio.output(in4, True)
207     print("TURNRIGHT")
208 
209 
210 def TurnLeft(speed):
211     gpio.setmode(gpio.BCM)
212     gpio.setup(in1, gpio.OUT)
213     gpio.setup(in2, gpio.OUT)
214     gpio.setup(in3, gpio.OUT)
215     gpio.setup(in4, gpio.OUT)
216     gpio.setup(en1, gpio.OUT)
217     gpio.setup(en2, gpio.OUT)
218     pwm1.start(speed)
219     pwm2.start(speed)
220 
221     gpio.output(in1, False)
222     gpio.output(in2, True)
223     gpio.output(in3, True)
224     gpio.output(in4, False)
225     print("TURNLEFT")
226 
227 
228 def Stop():
229     gpio.setmode(gpio.BCM)
230     gpio.setup(in1, gpio.OUT)
231     gpio.setup(in2, gpio.OUT)
232     gpio.setup(in3, gpio.OUT)
233     gpio.setup(in4, gpio.OUT)
234 
235     gpio.output(in1, False)
236     gpio.output(in2, False)
237     gpio.output(in3, False)
238     gpio.output(in4, False)
239     gpio.cleanup()
240 
241 
242 # Lights:
243 def FrontLight():
244     print("FRONT LED")
245     pwm.set_pwm(8, 6000, 1500)
246     pwm.set_pwm(9, 0, 0)
247     pwm.set_pwm(10, 0, 0)
```

```
248     pwm.set_pwm(11, 0, 0)
249
250
251 def BackLight():
252     print("STOP")
253     pwm.set_pwm(8, 0, 0)
254     pwm.set_pwm(9, 0, 0)
255     pwm.set_pwm(10, 0, 0)
256     pwm.set_pwm(11, 6000, 1500)
257
258
259 def RightLight():
260     print("RIGHT LED")
261     pwm.set_pwm(8, 0, 0)
262     pwm.set_pwm(9, 6000, 1500)
263     time.sleep(0.4)
264     pwm.set_pwm(9, 0, 0)
265     time.sleep(0.4)
266     pwm.set_pwm(10, 0, 0)
267     pwm.set_pwm(11, 0, 0)
268
269
270 def LeftLight():
271     print("LEFT LED")
272     pwm.set_pwm(8, 0, 0)
273     pwm.set_pwm(9, 0, 0)
274     pwm.set_pwm(10, 6000, 1500)
275     time.sleep(0.4)
276     pwm.set_pwm(10, 0, 0)
277     time.sleep(0.4)
278     pwm.set_pwm(11, 0, 0)
279
280
281 def Homing():
282     ArmHoming()
283     time.sleep(0.1)
284     ArmHoming()
285     time.sleep(0.1)
286     ArmHoming()
287     time.sleep(0.1)
288
289
290 def SpeakBegin():  # WELCOMING VOICE
291     engine.say("Step Back.. Falcon is detecting!")
292     engine.runAndWait()
293     engine.stop()
294
295
296 def Speak(text):  # TEXT TO SPEECH CONVERSION FUNCTION
297     str(text)
298     engine.say(text)
```

the following is the main code:

FALCON Main Code

```
### Import NECESSARY Libraries:
import RPi.GPIO as gpio
import cv2
from matplotlib.pyplot import text
import numpy as np
import Falcon
import time

## RESET Variables:
en1 = 20
en2 = 21
in1 = 17
in2 = 22
in3 = 23
in4 = 24
gpio.setmode(gpio.BCM)
gpio.setup(in1, gpio.OUT)
gpio.setup(in2, gpio.OUT)
gpio.setup(in3, gpio.OUT)
gpio.setup(in4, gpio.OUT)
gpio.setup(en1, gpio.OUT)
gpio.setup(en2, gpio.OUT)

T = 45  # Tracking-box Tolerance:
CDT = 5  # Color Detection Tolerance
SpeakBlue = 0
SpeakRed = 0
SpeakGreen = 0

cap = cv2.VideoCapture(0)
W = 640
H = 480
hW = W / 2
hH = H / 2
cap.set(cv2.CAP_PROP_FRAME_WIDTH, W)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, H)
cap.set(cv2.CAP_PROP_BRIGHTNESS, 64.0)
cap.set(cv2.CAP_PROP_EXPOSURE, 126.0)
cap.set(cv2.CAP_PROP_SATURATION, 55.0)
cap.set(cv2.CAP_PROP_CONTRAST, 4.0)
cap.set(cv2.CAP_PROP_GAIN, -1.0)

success, frame = cap.read()
rows, cols, _ = frame.shape
x_medium = int(cols / 2)
y_medium = int(rows / 2)
center = int(cols / 2)

print("Detecting,,")
Falcon.SpeakBegin()
Falcon.Homing()
Falcon.ArmCameraStart()

while True:
```

```
    success, frame = cap.read()
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Blue =(LH_B, LS_B, LV_B, HH_B, HS_B, HV_B)
    Blue = (104, 100, 112, 122, 246, 255)
    # Red =(LH_R, LS_R, LV_R, HH_R, HS_R, HV_R)
    Red = (149, 97, 0, 173, 230, 255)
    # Green =(LH_G, LS_G, LV_G ,HH_G, HS_G, HV_G)
    Green = (33, 152, 157, 77, 255, 255)

    # blue color
    low_blue = np.array([Blue[0], Blue[1], Blue[2]])
    high_blue = np.array([Blue[3], Blue[4], Blue[5]])
    blue_mask = cv2.inRange(hsv_frame, low_blue, high_blue)
    # red color
    low_red = np.array([Red[0], Red[1], Red[2]])
    high_red = np.array([Red[3], Red[4], Red[5]])
    red_mask = cv2.inRange(hsv_frame, low_red, high_red)
    # green color
    low_green = np.array([Green[0], Green[1], Green[2]])
    high_green = np.array([Green[3], Green[4], Green[5]])
    green_mask = cv2.inRange(hsv_frame, low_green, high_green)

    contours, hierarchy = cv2.findContours(
        blue_mask + red_mask + green_mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE
    )

    contours = sorted(contours, key=lambda x: cv2.contourArea(x), reverse=True)
    for cnt in contours:
        (x, y, w, h) = cv2.boundingRect(cnt)

        x_medium = int((x + x + w) / 2)
        y_medium = int((y + y + h) / 2)
        break

    X1 = x
    Y1 = y
    X2 = x + w
    Y2 = y + h

    x11 = W - (hW + T)
    y11 = H - (hH + T)
    x22 = W - (hW - T)
    y22 = H - (hH - T)

    x_distance = hW - x_medium
    y_distance = hH - y_medium
    min_speed = 40
    max_speed = 90
    new_speed = int(min_speed + ((max_speed - min_speed) * abs(x_distance) / hW))
    speed = int(((new_speed - min_speed) / (max_speed - min_speed)) * 100)

    print("xdis is: ", x_distance)
    print("ydis is: ", y_distance)
    print("New speed is: ", new_speed)
    print("Speed is: ",str(speed)+ "%")

```

```
    pt1 = (X1, Y1)
    pt2 = (X2, Y2)
    pt11 = (x11, y11)
    pt22 = (x22, y22)

    cv2.line(frame, (x_medium, 0), (x_medium, 10 * W), (0, 255, 0), 4)
    cv2.line(frame, (0, y_medium), (10 * H, y_medium), (255, 0, 0), 4)

    # pick pixel Center color
    pixel_center = frame[int(x_distance), int(y_distance)]
    cv2.circle(frame, (int(x_distance), int(y_distance)), 5, (255, 255, 255), -1)
    print(pixel_center[0], pixel_center[1], pixel_center[0])
    cv2.circle(frame, (x_medium, y_medium), 5, (0, 0, 255), -1)

    cv2.rectangle(
        frame,
        (int(pt1[0]), int(pt1[1])),
        (int(pt2[0]), int(pt2[1])),
        color=(255, 255, 255),
        thickness=3,
    )
    cv2.rectangle(
        frame,
        (int(pt11[0]), int(pt11[1])),
        (int(pt22[0]), int(pt22[1])),
        color=(0, 0, 255),
        thickness=3,
    )

    if (-T < x_distance < T) and (y_distance > T):
        Falcon.Forward(new_speed)
        print("FORWARD")

    elif (-T < x_distance < T) and (y_distance < -T):
        Falcon.Backward(new_speed)
        print("BACKWARD")

    elif (x_distance > T) and ((y_distance > T) or (y_distance < T)):
        Falcon.TurnLeft(new_speed)
        print("LEFT")

    elif (X1 > x11) and (x_distance < T) and ((y_distance > T) or (y_distance < T))
    :
        Falcon.TurnRight(new_speed)
        print("RIGHT")

    elif (x_distance < T) and (y_distance < T):
        print("Center Pixel is: ", pixel_center)
        if (
            (Blue[0] - CDT < pixel_center[0] < Blue[3] + CDT)
            and (Blue[1] - CDT < pixel_center[1] < Blue[4] + CDT)
            and (Blue[2] - CDT < pixel_center[2] < Blue[5] + CDT)
        ):
            print("Blue")
            SpeakBlue = 1

        elif (
```

```
            (Red[0] - CDT < pixel_center[0] < Red[3] + CDT)
            and (Red[1] - CDT < pixel_center[1] < Red[4] + CDT)
            and (Red[2] - CDT < pixel_center[2] < Red[5] + CDT)
        ):
            print("Red")
            SpeakRed = 1

        elif (
            (Green[0] - CDT < pixel_center[0] < Green[3] + CDT)
            and (Green[1] - CDT < pixel_center[1] < Green[4] + CDT)
            and (Green[2] - CDT < pixel_center[2] < Green[5] + CDT)
        ):
            print("Green")
            SpeakGreen = 1

        Falcon.Stop()
        print("STOP")
        time.sleep(1)
        break

    cv2.imshow("Frame", frame)
    cv2.imshow("Mask", blue_mask + red_mask + green_mask)
    cv2.resizeWindow("Frame", W, H)
    cv2.resizeWindow("Mask", W, H)
    key = cv2.waitKey(1)

    if key == 27:
        break

if SpeakBlue == 1:
    Falcon.Speak("The Color is Blue")
    print("Blue")
if SpeakRed == 1:
    print("Red")
if SpeakGreen == 1:
    Falcon.Speak("The Color is Green")
    print("Green")

Falcon.angles = Falcon.IK(x=x_distance, y=y_distance, z=20)
print(Falcon.angles)
Falcon.ArmIk(a=Falcon.angles[0]+80,b=abs(Falcon.angles[1]) ,c=abs(Falcon.angles[2])
     ,d=abs(Falcon.angles[3]) ,e=abs(Falcon.angles[4]), f=90)
time.sleep(2)
Falcon.ArmIk(a=Falcon.angles[0]+80,b=abs(Falcon.angles[1]) ,c=abs(Falcon.angles[2])
     ,d=abs(Falcon.angles[3]) ,e=abs(Falcon.angles[4]), f=0)time.sleep(2)
print("PICKED")
Falcon.ArmIk(a=Falcon.angles[0]+80,b=abs(Falcon.angles[1]) ,c=abs(Falcon.angles[2])
     ,d=abs(Falcon.angles[3]) ,e=abs(Falcon.angles[4]), f=0)
print("DONE")
Falcon.ArmCameraStart()
cv2.destroyAllWindows()
```

3.2 Additional Features

3.2.1 Speaker

By connecting the speaker's USB jack to any USB port on Raspberry Pi board and the standard 3.5mm audio jack, as well as changing the setting of Audio's output to Analog instead of HDMI on Raspberry Pi OS [39]:

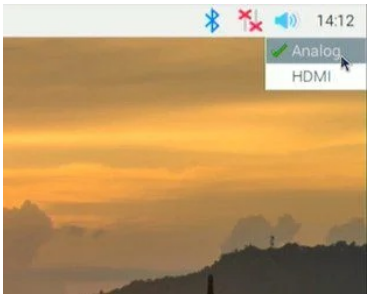


Figure 3.6: setting Audio's output to Analog

Figure 3.7: USB Powered 3.5mm Jack Speaker [39]

using **pyttsx3** (text-to-speech conversion library in Python) Sounds can be heard from FALCON like pronouncing the color detected after reading the pixel information of the grabbed object and compare its HSV values to the saved ones in order to play its specific voice color. The code to perform the reading of the color is included below:

pyttsx3 code example

```
import pyttsx3

engine = pyttsx3.init()
engine.say("I will speak this text")
engine.runAndWait()
```

]

Changing Voice tone ,Rate and Volume is possible !

Discussion and Result

After completing the programming part and the compilation of the design we need to verify the correctness of our Color Sorting Robot functionality. This can be done by placing colored objects at various locations and distances across its path. We can see that when FALCON detects the nearest colored object,it moves towards it ,grabs it,pronounce its color then reads its location to set the right joint angles (using IK) then the Arm grabs it and Sorts it to the specified location according to the color.

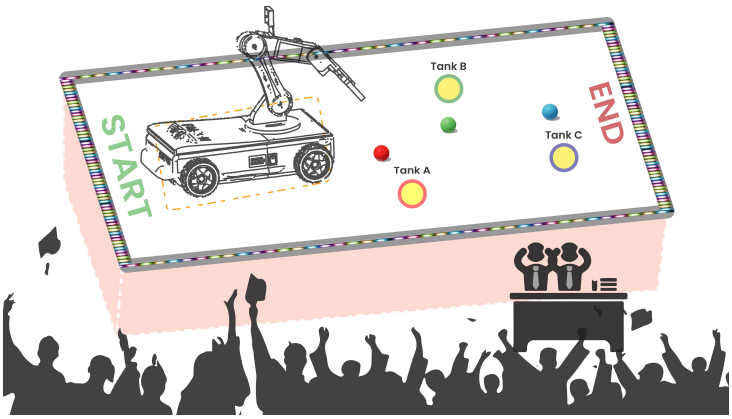


Figure 3.8: FALCON main aim

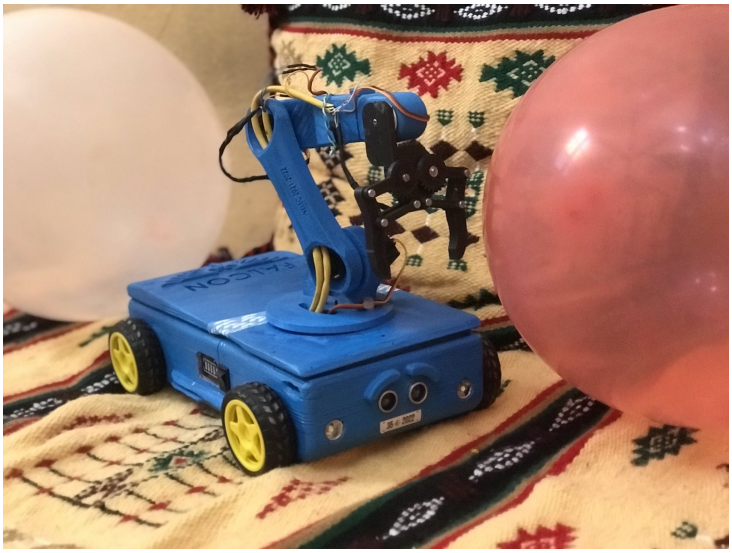


Figure 3.9: FALCON is alive!

Conclusion and Future Work

Overall, a sorting-colored robot has been successfully developed. It has advantageous features such as the ability to detect the location of colored object automatically. It also has the ability to avoid hitting any obstacle or surrounding objects due to its provision of an ultrasonic sensor. There were some problems encountered during the implementation, especially from the hardware part; the dc-dc converter step down gave random values when connected to the PCA9685 resulting in the damaging of some servo motors. Working on this project was a rich learning experience that expanded our knowledge from theory to practice. We now acquire skills in color sorting robots (arm and car) using the OpenCV-python library for color detection, even 3D design and implementation of the circuit even which helped to get an extra understanding of the different features of the components we used.

For future work, a PCB design of a board connecting all the single components of the robot will be a great thing rather than the use of jumper wires which results in many connection failures,an application can be added to use the robot manually , gesture-control its movements to perform various missions in the way wanted. Also, this system can be developed on a much larger scale to pick and place items that are more difficult for the man type.

Bibliography

[1] *Rosheim ME.Leonardo's programmable automaton. A reconstruction.* URL: http://www.anthrobot.com/press/article_leo_programmable.html.

[2] *Nuland SB.Leonardo da Vinci. New York: Penguin Books.2000 [Google Scholar].* URL: https://scholar.google.com/scholar_lookup?title=Leonardo+da+Vinci&author=SB+Nuland&publication_year=2000&.

[3] *Capello S, Moran ME, Belarmino J, Firoozi F, Kolios E, Perrotti M (2005) The da Vinci robot. 23rd world congress on endourology. J Endourol 19(1): A133 [Google Scholar].*

[4] *J Robot Surg. 2007; 1(2): 103–111.* URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4247431/.

[5] *Engelberger J. Robots in service. Cambridge: MIT Press; 1989. [Google Scholar].* URL: https://scholar.google.com/scholar_lookup?title=Robots+in+service&author=J+Engelberger&publication_year=1989&.

[6] *main-types-of-robots.*

[7] URL: https://www.fritz.ai/object-detection/.

[8] URL: https://data-flair.training/blogs/project-in-python-colour-detection/.

[9] URL: https://www.geeksforgeeks.org/real-time-object-color-detection-using-opencv/.

[10] URL: https://books.google.dz.

[11] URL: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-product-brief.pdf.

[12] URL: raspberry-pi-4-gpio-pinout-specifications-and-schematic.

[13] URL: https://www.youngwonks.com/blog/Raspberry-Pi-4-Pinout.

[14] URL: https://www.cytron.io/c-ai-dl/c-nvidia-jetson/c-nvidia-jetson-camera/p-raspberry-pi-8mp-camera-module-v2.

[15] URL: https://www.genstattu.com/bw/.

[16] URL: https://lastminuteengineers.com/.

[17] URL: https://handsontec.com/index.php/product/18-cell-lithium-battery-level-indicator-module-user-configurable/.

[18] URL: https://www.iqsdirectory.com/articles/electric-motor/dc-motors.html.

[19] URL: https://www.circuitbasics.com/introduction-to-dc-motors/.

[20] URL: https://components101.com/modules/l293n-motor-driver-module.

[21] URL: https://circuitdigest.com/article/servo-motor-working-and-basics.

[22] URL: https://www.electronicoscaldas.com/datasheet/MG996R.

[23] URL: MG996R-Servo-High-Torque.

[24] URL: https://pdf1.alldatasheet.com/datasheet-pdf/MG90S.

[25] URL: mg90s.

[26] URL: https://components101.com/motors/mg996r-servo-motor-datasheet.

[27] URL: https://wiki.keyestudio.com/.

[28] URL: https://lastminuteengineers.com/.

[29] URL: http://wiki.sunfounder.cc/PCA9685_16_Channel_12_Bit_PWM.

[30] URL: Adafruit%20Python%20PCA9685.

[31] URL: https://www.theengineeringprojects.com/introduction-to-hc-sr04-ultrasonic-sensor.

[32] URL: https://lastminuteengineers.com/-sr04-ultrasonic-sensor.

[33] URL: SOLIDWORKS_Introduction_EN.pdf.

[34] URL: solidworks-logo.

[35] URL: https://megadeals.ma/boutique/impression3daumaroc/ender-3-st-pro-v2.

[36] URL: opencv.org.

[37] URL: opencv%20/python%20logo.

[38] URL: https://www.mathworks.com/discovery/inverse-kinematics.html.

[39] URL: https://www.cytron.io/p-6w-stereo-usb-powered-3-5mm-jack-speaker-black.