Autonomous Vehicle Safety

Instructor: Dr. Marek Perkowski

Intelligent Robotics I - 2019

Aniruddha Shahapurkar

Amulya Huchachar

TABLE OF CONTENTS

OBJECTIVE ----------3
OVERVIEW----------3
VIKING BOT ----------4
COMPONENTS ----------10
TOOLS AND TECHNOLOGIES ----------16
MORPHOLOGICAL OPERATIONS ----------18
BIRD'S EYE VIEW TRANSFORMATION ----------22
FEATURES ----------24
CHALLENGES FACED----------25
POSSIBLE IMPROVEMENTS ----------25
CONCLUSION ----------26
REFERENCES ----------26

OBJECTIVE

Safety is the state of being "safe", the condition of being protected from harm or other non-desirable outcome. Safety is generally interpreted as implying a real and significant impact on risk of death, injury or damage to property. Road safety is one of the major issues.

Roads are essential to our everyday lives. We all use them in some way, by driving, riding, walking or travelling as a passenger, and we depend on them to obtain goods and services.

Unfortunately, this comes at a price, which includes people being killed and injured. However, road deaths and injuries are not inevitable. The last few decades have demonstrated that effective and comprehensive road safety strategies can reduce the number of people killed or injured on the road, despite increasing traffic levels.

According to the World Health Organization(WHO) more than 1 million people are killed on the world's roads each year. A report published by the WHO in 2004 estimated that some 1.2 million people were killed and 50 million injured in traffic collisions on the roads around the world each year.

SAFE was designed with the idea that the driver with a better sense of his/her surroundings is better equipped to drive defensively, hence reducing the probability of accidents. It provides the driver with the ability to track posterior vehicles by monitoring their relative speed, position, and direction.

OVERVIEW

This program was designed by Dr. Perkowski’s students as an affordable Situational Awareness Fault-Finder Extension device, which can be used on automobiles, motorcycles and bicycles. Much of their work is based on the system developed by Nieto [5]. It won the Second Prize at the Intel-Cornell Cup, a college-level embedded system competition.

The program is built on a viking bot which detects the obstacles by on it’s way from the given/set distance from the current position of the bot, and takes a right turn to the obstacle and go forward then again come back to the middle position and move forward. The program also perform homography to convert the frame into a birds-eye view display with boxes around the detected vehicles.

We have taken taken the idea of SAFE created by previous students, but the program is built from scratch in Python language (old program was built on C++).The main library using in building this program is OpenCV(Open Computer Vision).

Viking Bot

The viking bot is built from scratch for this project. We ordered the **Robot Car Kit for Raspberry Pi 3 Model B+/B/2B** from Amazon and assembled the car.
Here is the link for Robot kit from Amazon :

Package Contains the below items:

1 Set Acrylic Plates
1x Adeept Motor HAT V2.0
1x Raspberry Pi Camera(with Cable)
1x USB Microphone
1x Ultrasonic Sensor Module
2x Adeept RGB LED Module
4x Adeept WS2812 RGB LED Module
1x Adeept 3CH Line Tracking Module
3x Servo
1x Gear Motor
4x Wheels
1x Battery Holder
1x Cross Socket Wrench
2x Cross Screwdriver(Small and Large)
1x Winding Pipe
10x Bearing(6*F624ZZ + 4*F687ZZ)
2x Umbrella Gear Set
Other necessary accessories(Wires, Nuts, Screws, Copper Standoffs, Couplings)

Features of this Robot are :

- The Robot car kit based on Raspberry Pi(Compatible with RPi 3B/3B+/2B/2B+, Raspberry Pi is NOT included)
- Powered by 2x18650 batteries(NOT included). We need to arrange our own batteries.
- Equipped with 12x WS8212 serial RGB LEDs, these RGB LEDs can be controlled through only one GPIO pin, which can change a variety of colors and indicate the working state of the robot.
- Real-time Video Transmission - it can transfer the real-time images taken by the Raspberry Pi camera to a remote computer.
- Object Recognition and Tracking - based on openCV,can track objects of a specific shape or color

ws2812 LED

Tracking Module

Raspberry Pi Camera

USB microphone

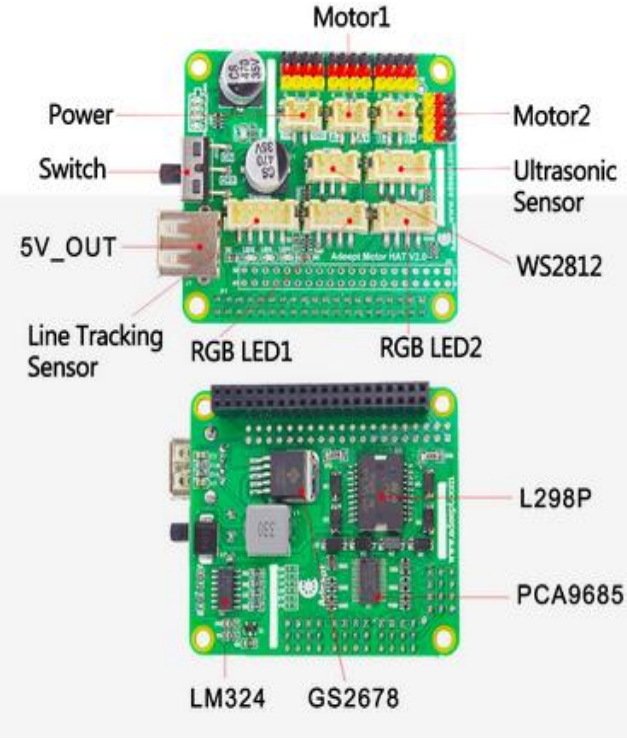
gear motor

coupler

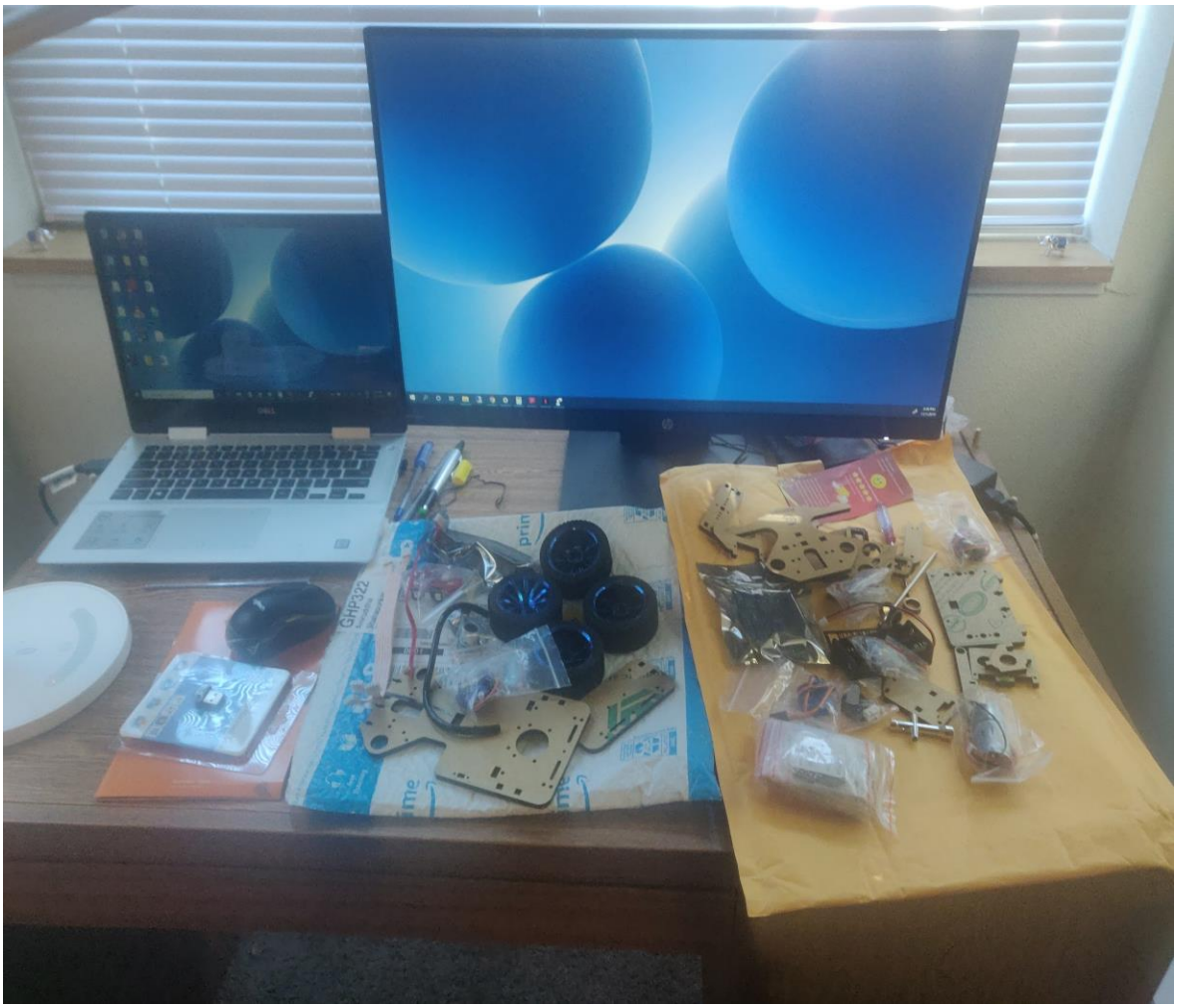
bevel gear

RGB LED

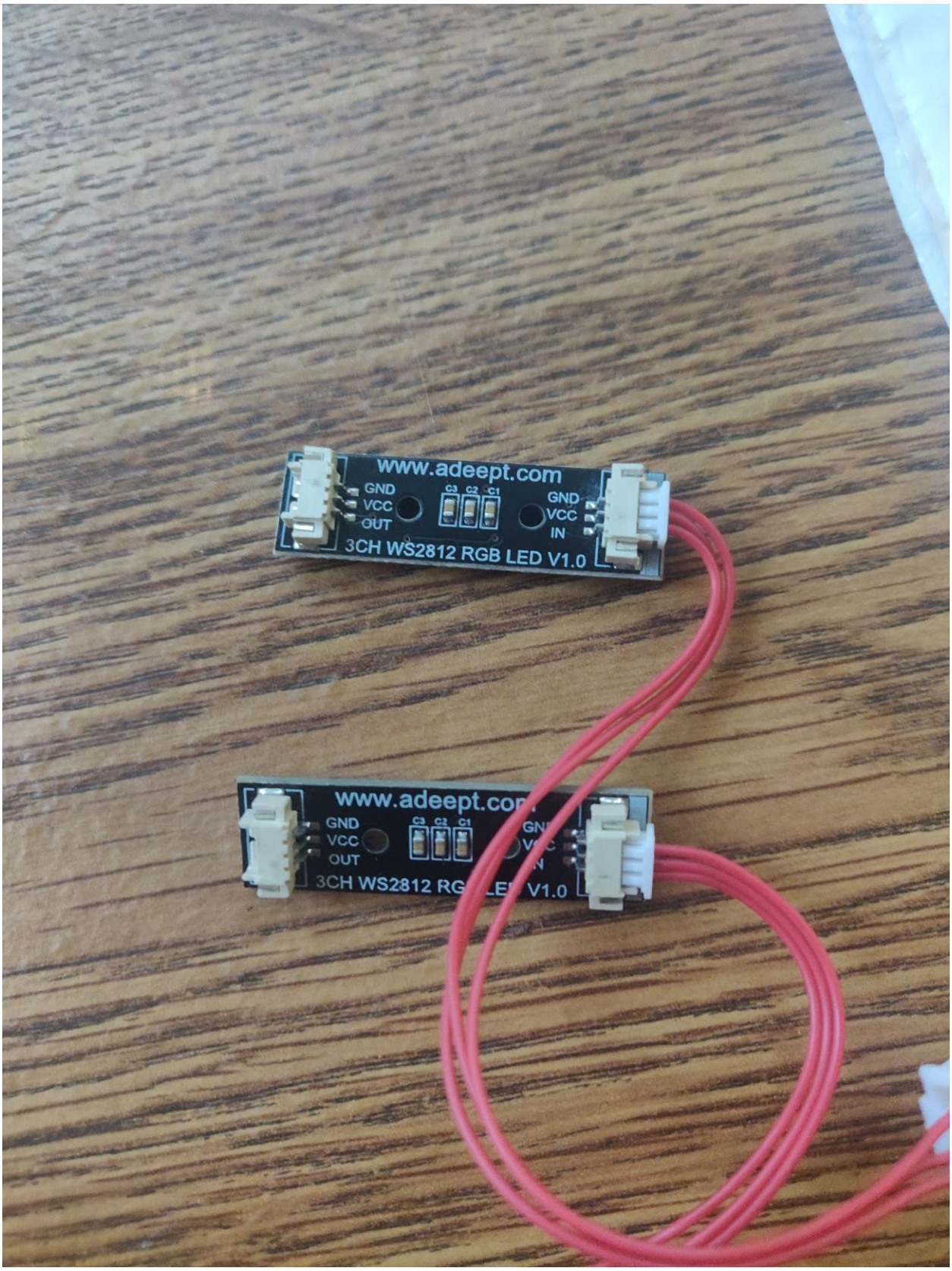
Ultrasonic Module

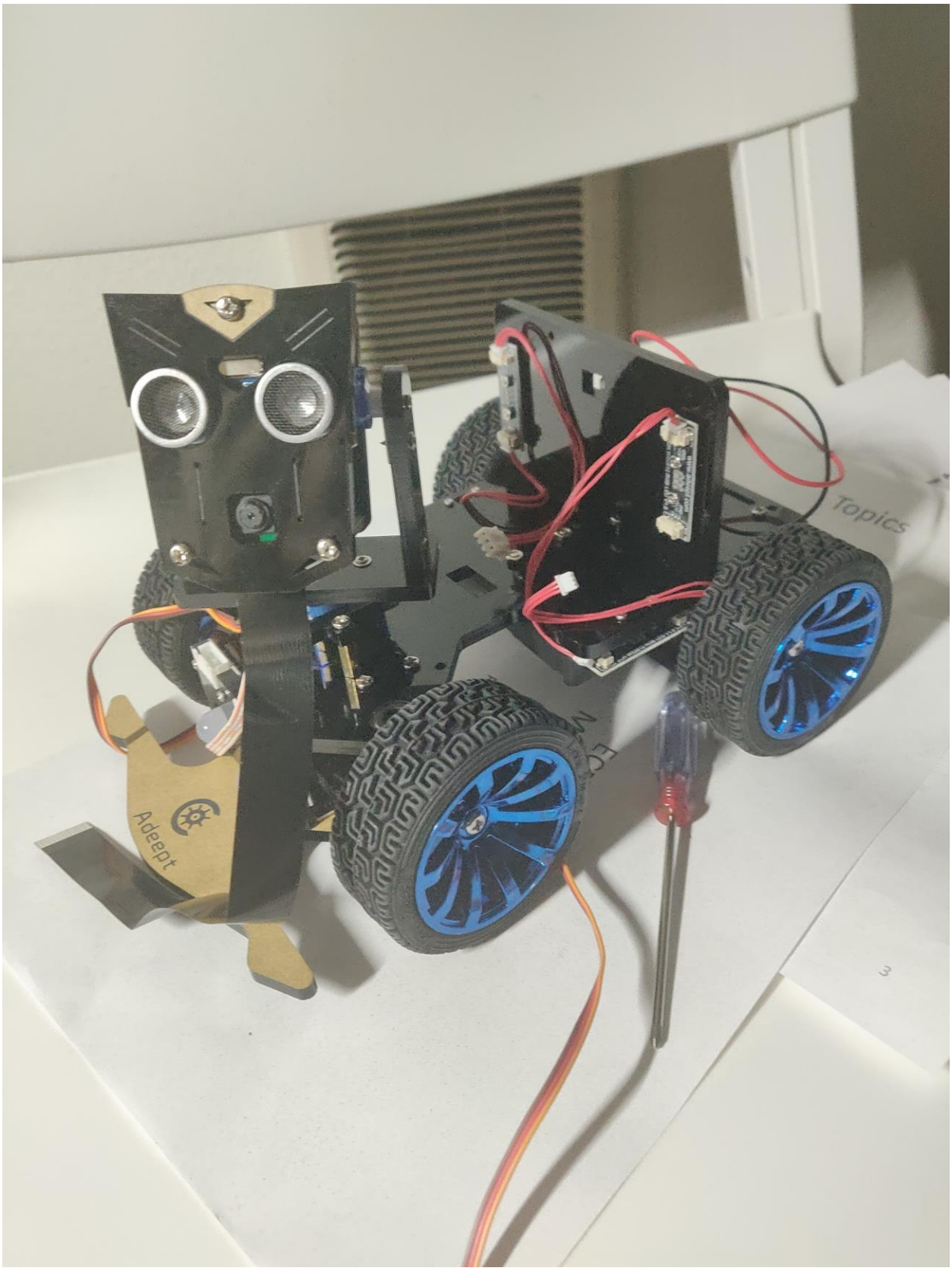


Before Assembly

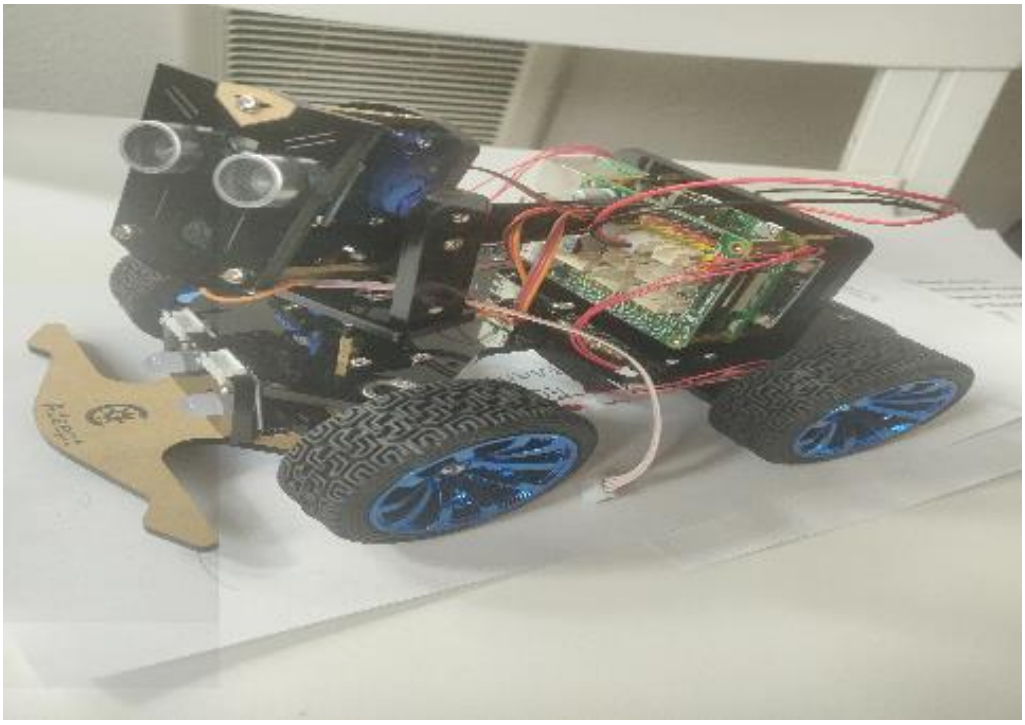



Gnd
VCC
IN
www.adeept.com
Sharing Perfects Innovation


www.adeept.com
GND
VCC
OUT
C3 C2 C1
GND
VCC
IN
3CH WS2812 RGB LED V1.0


Adeept
Topics

After Assembly :



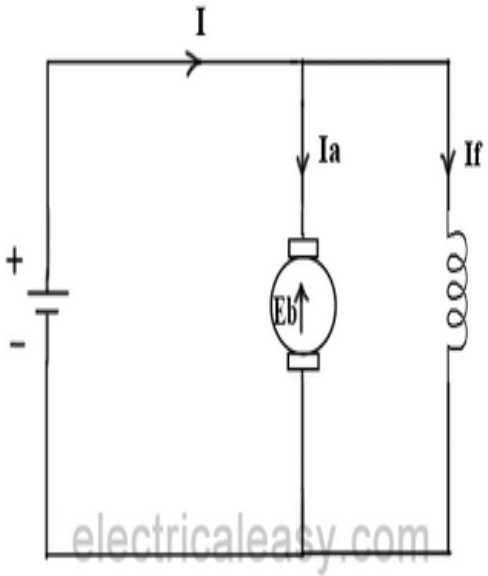
COMPONENTS

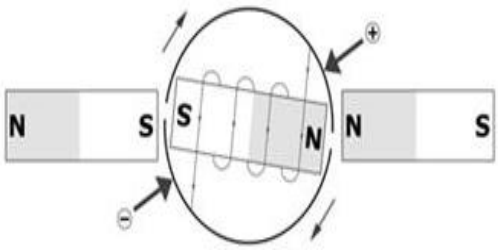
The components used in this project are :

- 1 DC Motor
- 3 Servo Motors
- Raspberry Pi 3
- Raspberry Pi Camera
- 2 RGB LEDs

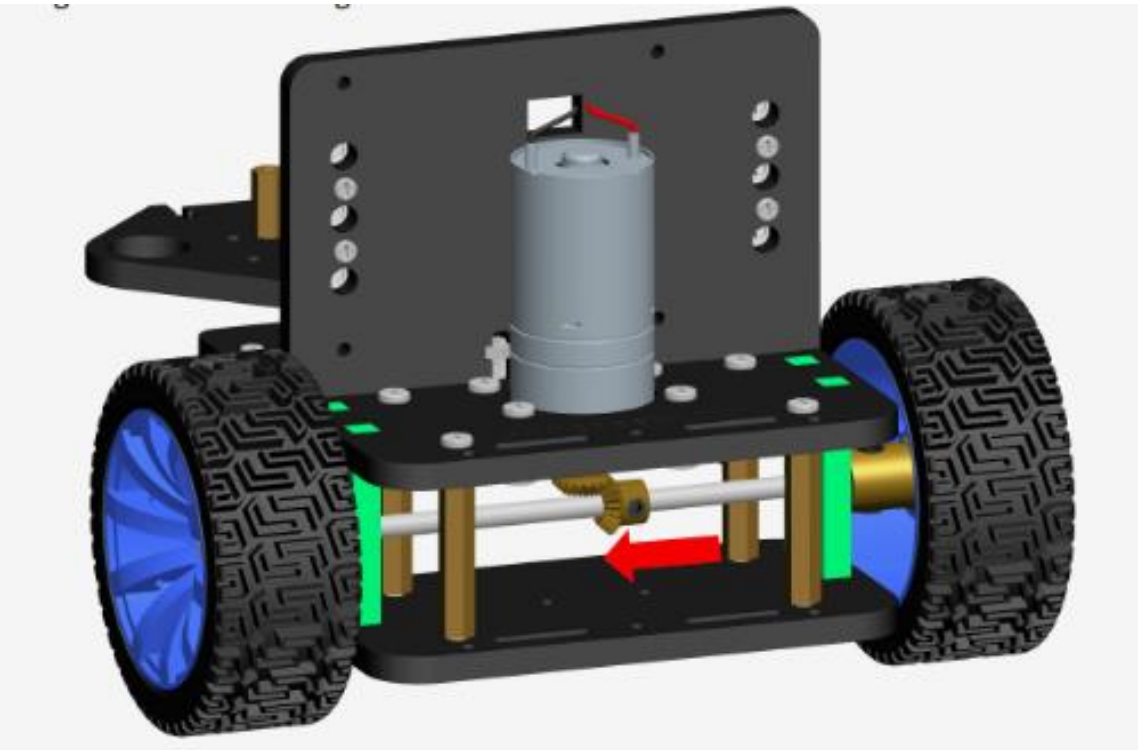
DC Motor

We have used 1 DC motor in our project. The DC motor is a machine that transforms electrical energy into mechanical energy in the form of rotation. Its movement is produced by the physical behavior of electromagnetism. DC motors have inductors inside, which produce the magnetic field used to generate movement.

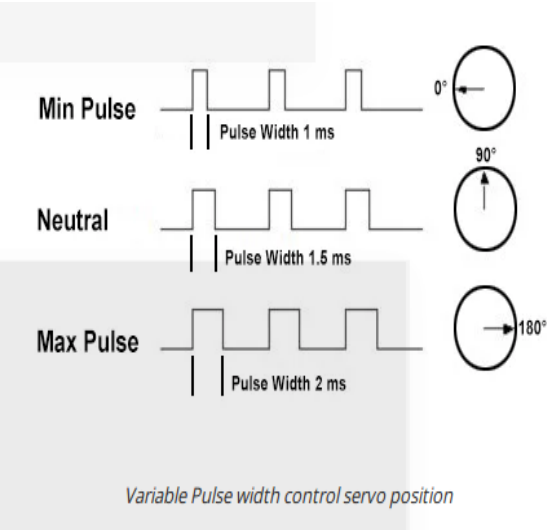




The DC Motor control on the viking bot looks like below.

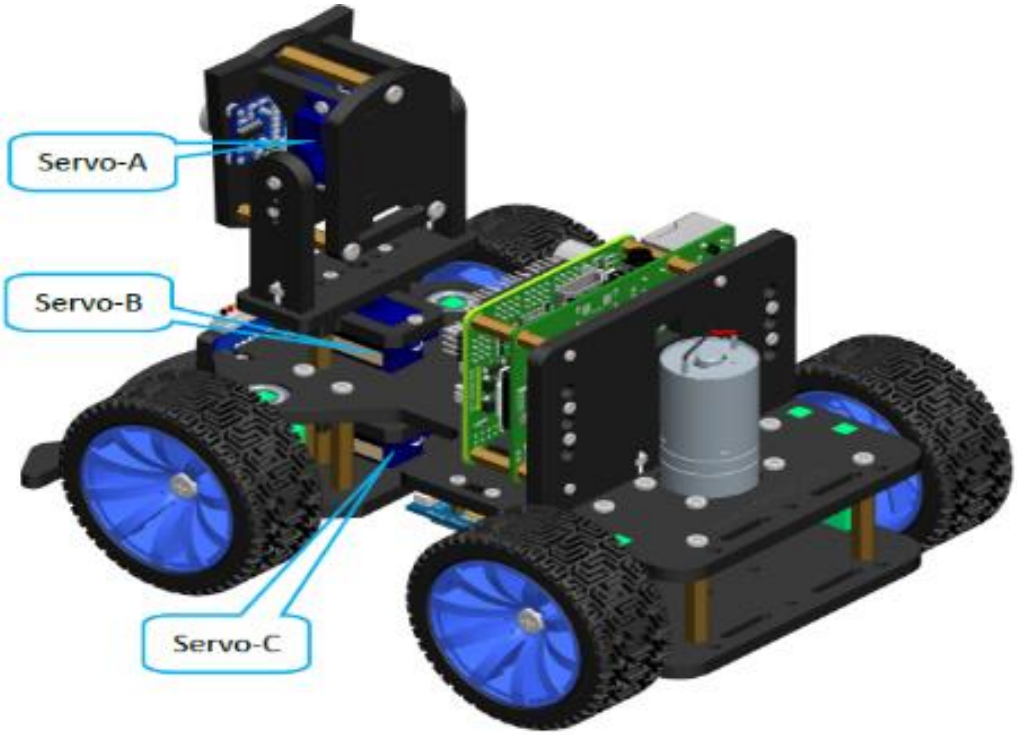


Servo Motor :



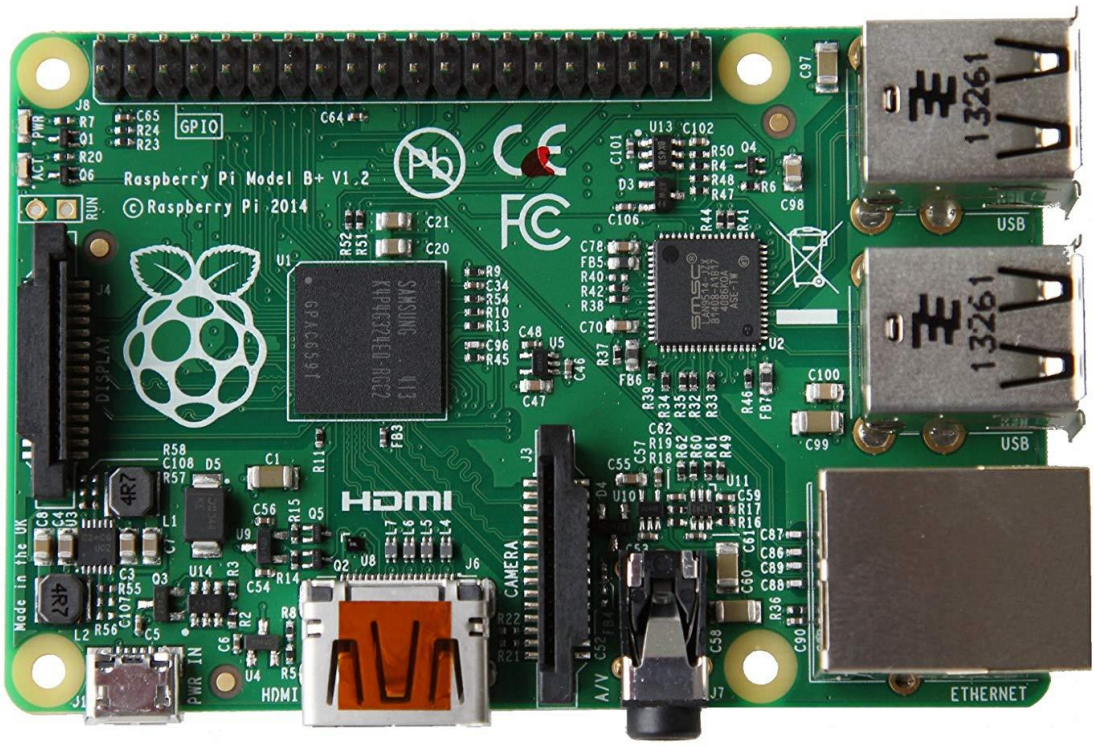
We have used 3 Servo Motors in our project. A servo motor is a rotary actuator that allows for precise control of angular position. It consists of a motor coupled to a sensor for position feedback. It also requires a servo drive to complete the system. The drive uses the feedback sensor to precisely control the rotary position of the motor.

The Servo Motor controls on the viking bot looks like below.



Raspberry Pi 3 :

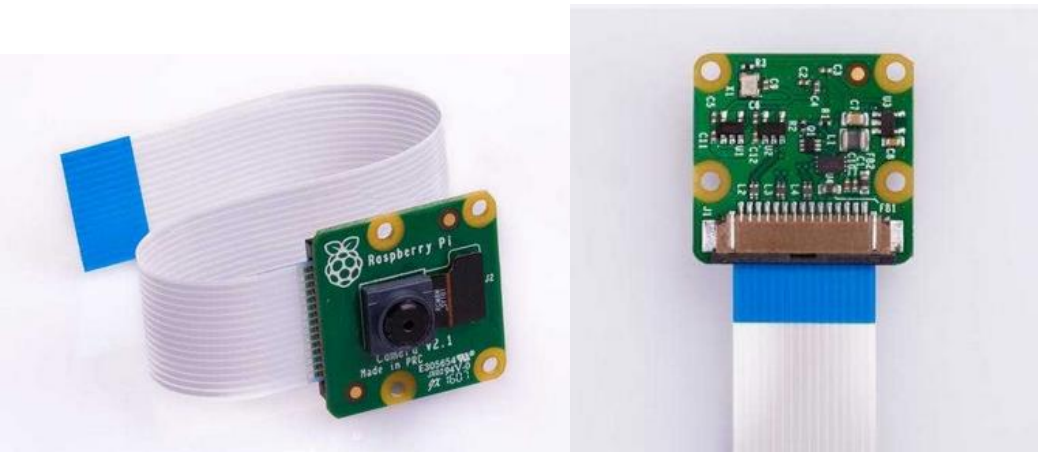
The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor. It is a capable little device that enables people to explore computing, and to learn how to program in languages like Scratch and Python. It has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting bird houses with infra-red cameras and many more Robotic projects.

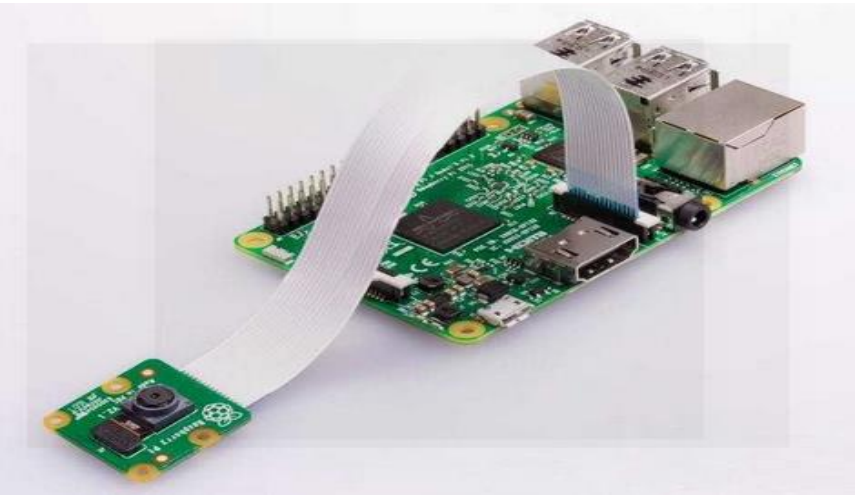


Raspberry Pi Camera :

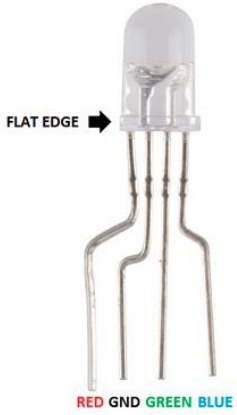
The Raspberry Pi camera module is capable of taking full HD 1080p photo and video and can be controlled programmatically. We have used 1 Pi camera in our project.

Connecting the camera to Raspberry Pi board : The flex cable inserts into the connector situated between the Ethernet and HDMI ports, with the silver connectors facing the HDMI port. The flex cable connector should be opened by pulling the tabs on the top of the connector upwards then towards the Ethernet port. The flex cable should be inserted firmly into the connector, with care taken not to bend the flex at too acute an angle. The top part of the connector should then be pushed towards the HDMI connector and down, while the flex cable is held in place.





RGB LEDs : We have used 2 RGB LEDs in our project. RGB LED means red, blue and green LEDs. RGB LED products combine these three colors to produce over 16 million hues of light. Note that not all colors are possible. Some colors are “outside” the triangle formed by the RGB LED. Also, pigment colors such as brown or pink are difficult, or impossible, to achieve.



Features

We could able to achieve below mentioned features as part of this project.
Movements : Forward , Backward, Right, Left
Image Detection
Obstacle Avoidance
LED Indicator

Tools and Technologies

GPIO (General-purpose input/output)

A GPIO is a signal pin on an integrated circuit or board that can be used to perform digital input or output functions. By design it has no predefined purpose and can be used by the hardware or software developer to perform the functions they choose. Typical applications include controlling LEDs, reading switches and controlling various types of sensors.
Raspberry Pi, have a 40-pin GPIO connector that provides access to about 25 GPIO lines.

The most common functions of GPIO pins include:

- Being configurable in software to be input or output
- Being enabled or disabled
- Setting the value of a digital output
- Reading the value of a digital output
- Generating an interrupt when the input changes value

GPIO can be used in three modes:

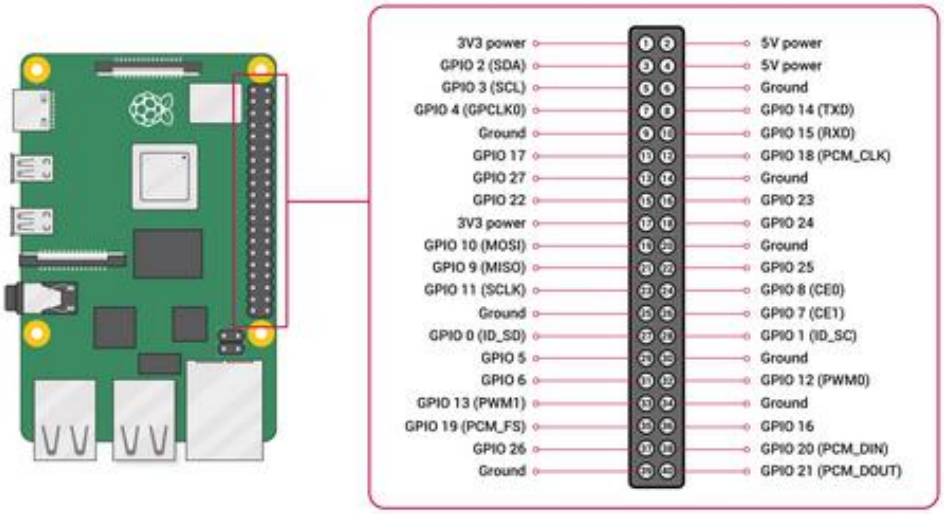
- input
- output
- UART interface

GPIO input

This is the default mode, in which the beacon receives input from the connected device via GPIO. You can imagine a button broadcasting its status (on/off) through the beacon. In this configuration, the beacon will broadcast received data two 0/1 values

GPIO output

In the output mode, beacon delivers data to the connected device via GPIO. You could for example switch a LED lamp on or off with the beacon controlled from a mobile app.In this configuration, the beacon will deliver data from two pins about their binary states to the connected device.



We have imported the Raspberry Pi GPIO in to the python code and used it for LED pin configuration for ON and OFF of the lights, monitor pin configuration for movement of forward, backward, right and left. We also use GPIO for side and colour configuration of LEDs.

Adafruit - PCA9685

Servo motors are often driven using the PWM outputs available on most embedded MCUs. But while the Pi does have native HW support for PWM, there is only one PWM channel available to users at GPIO18. That kind of limits your options if you need to drive more than one servo or if you also want to dim an LED or do some sort of other PWMgoodness as well. Thankfully ... the PI does have HW I2C available, which we can use to communicate with a PWMdriver like the PCA9685, used on Adafruit's 16-channel 12-bit PWM/Servo Driver!

We imported Adafruit_PCA9685 into python code using this we can communicate with servo motors and able to run our 3 servo motors. By controlling servo, the camera can move up and down, left and right and ultrasonic wave can move to left and right.

Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. ... Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse.

OpenCV

Open Source Computer Vision is a library of programming functions _mainly aimed at real-time computer vision. In our program OpenCV is mainly used for image processing.

Image processing is used for getting a live- stream input frames. It takes a stream of image frames from the camera as input and outputs the tracking boxes of all detected objects. An extension of this algorithm would be these boxes and their associated vectors will be fed to the final danger detection and potential user visualization stages.

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python.

Morphological Operations

- In short: A set of operations that process images based on shapes. Morphological operations apply a structuring element to an input image and generate an output image.
- The most basic morphological operations are: Erosion and Dilation. They have a wide array of uses, i.e. :
    - Removing noise
    - Isolation of individual elements and joining disparate elements in an image.
    - Finding of intensity bumps or holes in an image
- We will explain dilation and erosion briefly, using the following image as an example:

Dilation

- This operations consists of convolving an image A with some kernel (B), which can have any shape or size, usually a square or circle.
- The kernel B has a defined anchor point, usually being the center of the kernel.
- As the kernel B is scanned over the image, we compute the maximal pixel value overlapped by B and replace the image pixel in the anchor point position with that maximal value. As you can deduce, this maximizing operation causes bright regions within an image to "grow" (therefore the name dilation).
- The dilatation operation is:
  $dst(x,y)=max(x',y'):element(x',y')\neq 0 src(x+x',y+y')$

- Take the above image as an example. Applying dilation we can get:

- The bright area of the letter dilates around the black regions of the background.
  Erosion
- This operation is the sister of dilation. It computes a local minimum over the area of given kernel.
- As the kernel B is scanned over the image, we compute the minimal pixel value overlapped by B and replace the image pixel under the anchor point with that minimal value.
- The erosion operation is:
  `dst(x,y)=min(x′,y′):element(x′,y′)≠0src(x+x′,y+y′)`
- Analogously to the example for dilation, we can apply the erosion operator to the original image (shown above). You can see in the results below that the bright areas of the image get thinner, whereas the dark zones gets bigger.

RGB ↔ GRAY

Transformations within RGB space like adding/removing the alpha channel, reversing the channel order, conversion to/from 16-bit RGB color (R5:G6:B5 or R5:G5:B5), as well as conversion to/from grayscale using:

RGB[A] to Gray:Y←0.299·R+0.587·G+0.114·B

And

Gray to RGB[A]:R←Y,G←Y,B←Y,A←max(ChannelRange)

The conversion from an RGB image to gray is done with:

cvtColor(src, bwsrc, cv::COLOR_RGB2GRAY);

More advanced channel reordering can also be done with **cv::mixChannels**.

See also

cv::COLOR_BGR2GRAY, **cv::COLOR_RGB2GRAY**, **cv::COLOR_GRAY2BGR**, **cv::COLOR_GRAY2RGB**

RGB ↔ CIE XYZ.Rec 709 with D65 white point

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \leftarrow \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} \leftarrow \begin{bmatrix} 3.240479 & -1.53715 & -0.498535 \\ -0.969256 & 1.875991 & 0.041556 \\ 0.055648 & -0.204043 & 1.057311 \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

X, Y and Z cover the whole value range (in case of floating-point images, Z may exceed 1).

See also

cv::COLOR_BGR2XYZ, **cv::COLOR_RGB2XYZ**, **cv::COLOR_XYZ2BGR**, **cv::COLOR_XYZ2RGB**

RGB ↔ YCrCb JPEG (or YCC)

$$Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$
$$Cr \leftarrow (R - Y) \cdot 0.713 + delta$$
$$Cb \leftarrow (B - Y) \cdot 0.564 + delta$$
$$R \leftarrow Y + 1.403 \cdot (Cr - delta)$$
$$G \leftarrow Y - 0.714 \cdot (Cr - delta) - 0.344 \cdot (Cb - delta)$$
$$B \leftarrow Y + 1.773 \cdot (Cb - delta)$$

where

$$delta = \begin{cases} 128 & \text{for 8-bit images} \\ 32768 & \text{for 16-bit images} \\ 0.5 & \text{for floating-point images} \end{cases}$$

Y, Cr, and Cb cover the whole value range.

See also

cv::COLOR_BGR2YCrCb, cv::COLOR_RGB2YCrCb, cv::COLOR_YCrCb2BGR, cv::COLOR_YCrCb2RGB

RGB ↔ HSV

$$V \leftarrow max(R, G, B)$$
$$S \leftarrow \begin{cases} \frac{V - min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$
$$H \leftarrow \begin{cases} 60(G - B)/(V - min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R)/(V - min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G)/(V - min(R, G, B)) & \text{if } V = B \end{cases}$$

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

If H<0 then H←H+360 . On output 0≤V≤1, 0≤S≤1, 0≤H≤360 .

The values are then converted to the destination data type:

- 8-bit images: V←255V,S←255S,H←H/2(to fit to 0 to 255)
- 16-bit images: (currently not supported) V<−65535V,S<−65535S,H<−H
- 32-bit images: H, S, and V are left as is

See also

cv::COLOR_BGR2HSV, cv::COLOR_RGB2HSV, cv::COLOR_HSV2BGR, cv::COLOR_HSV2RGB

RGB ↔ HLS

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V_{max} \leftarrow max(R, G, B)$$

$$V_{min} \leftarrow min(R, G, B)$$

$$L \leftarrow \frac{V_{max} + V_{min}}{2}$$

$$S \leftarrow \begin{cases} \frac{V_{max} - V_{min}}{V_{max} + V_{min}} & \text{if } L < 0.5 \\ \frac{V_{max} - V_{min}}{2 - (V_{max} + V_{min})} & \text{if } L \geq 0.5 \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/(V_{max} - V_{min}) & \text{if } V_{max} = R \\ 120 + 60(B - R)/(V_{max} - V_{min}) & \text{if } V_{max} = G \\ 240 + 60(R - G)/(V_{max} - V_{min}) & \text{if } V_{max} = B \end{cases}$$

If $H<0$ then $H \leftarrow H+360$. On output $0 \leq L \leq 1$, $0 \leq S \leq 1$, $0 \leq H \leq 360$.

The values are then converted to the destination data type:

- 8-bit images: $V \leftarrow 255 \cdot V, S \leftarrow 255 \cdot S, H \leftarrow H/2$(to fit to 0 to 255)
- 16-bit images: (currently not supported)
  $V<-65535 \cdot V, S<-65535 \cdot S, H<-H$
- 32-bit images: H, S, V are left as is

Bird's-Eye View Transformation

- We want to measure the curvature of the lines and to do that, we need to transform the road image to a top-down view. To Compute the perspective transform, M, given the source and destination points we use cv2.getPerspectiveTransform(src,dst). To compute the inverse perspective transform we use cv2.getPerspectiveTransform(dst,src). Finally, we can Warp the image using the perspective transform cv2.warpPerspective(img, M, img_size, flags=cv2.INTER_LINEAR).

- We need to identify four (4) source coordinates points for the perspective transform. In this case, I assumed that road is a flat plane. This isn't strictly true, but it can serve as an approximation for this project. We need to pick four points in a trapezoidal shape (similar to region masking) that would represent a rectangle when looking down on the

road from above.

- There are many ways to select it. For example, many perspective transform algorithms will programmatically detect four source points in an image based on the edge or corner detection, and analyze attributes like color and surrounding pixels. We have selected a trapezoid by using image dimensions ratios as an input. We found it to be a smart way to manually calibrate the pipeline and make sure it generalizes for different roads.

- We have also implemented a code to properly sort the four source points for the perspective transformation. Just in case we change the way we come up with those points in the future. It is VERY IMPORTANT to feed it correctly, a wrong step here will mess everything up. The points need to be sorted “clockwise”, starting from top-left. The methodology consists in normalizing the input into the [0, 2pi] space, which naturally will sort it “counter-clockwise”. Then we invert the order before output the function.

- How to make sure we have a good transformation?
  The easiest way to do this is to investigate an image where the lane lines are straight and find four points lying along the lines that, after perspective transform, make the lines look straight and vertical from a bird’s-eye view perspective.

- We applied undistortion and then bird’s-eye transformation on a straight image of the road, and played with the trapezoid dimensions until getting this result:

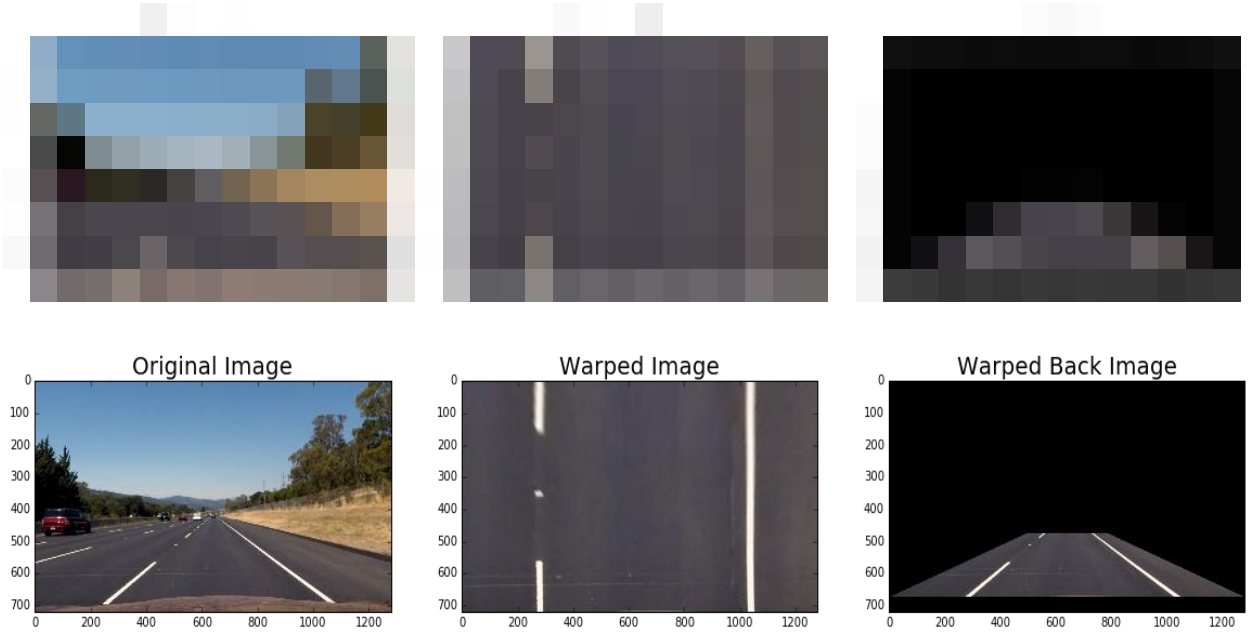
- Final trapezoid ratios and car’s hood cropping:

- bottom_width=0.4, percentage of image width

- top_width=0.092, percentage of image width

- height=0.4, percentage of image height

- car_hood=45, number of pixels to be cropped from bottom meant to get rid of car’s hood



Features

Vehicle Motion and LED Control

- All three servo motors have been controlled.
- Controlled DC motor for car's forward and backward movement
- Controlled the 2 RGB LEDs.

Object Detection

Successfully detected a yellow ball using OpenCV and Picamera.

Obstacle Avoidance

When the object is detected at a set distance with the car, the car takes the right turn to avoid the obstacle

Challenges Faced

Robot : Built the robot(Viking Bot) from scratch, since we could not find/get the viking bot from Lab we had to order for it online. We ordered through Amazon for Car Robot which works on Raspberry Pi. Assembly of robot with proper configuration took some time.

Here is the link from Amazon for Robot : https://www.amazon.com/Adeept-Recognition-Transmission-Educational-Instructions/dp/B07KM3YCYP

Software Installation : Since the previous code was in C++, we decided to build the project in high level language Python and to use OpenCV as opencv has many inbuilt libraries. Our project is built on Raspberry Pi board and installation of required software look time. Below are the software we installed for project.

18650 Batteries : Our robot need 2 18650 Batteries to run. We could not get these batteries in the lab and finding these batteries on shop or online was a hard part. Because of not having batteries our project start date was delayed.

Here is the link to order online : https://www.18650batterystore.com/18650-Batteries-s/106.htm

Color Masks : Since we both does not have any prior experience in Robotics and we are very new to python and OpenCV, We took some days to explore the features of OpenCv and hands on experience with python and Raspberry Pi. implementation of color mask using OpenCv took sometime.

Possible Improvements

- Building a better platform for the car using various obstacles to check the performance.
- Using the bird-eye-view image for better understanding of obstacles or potholes.
- Providing auditory alerts to users about potentially hazardous vehicles.
- Letting the car to decide to take turns or deviations based on the space availability when it encounters obstacles.
- Detecting the traffic signals and taking appropriate decisions accordingly.
- Implementation of vehicle to vehicle communication.

Conclusion

It was a great learning experience as we learnt motion control using DC and Servo motors and object detection via image processing where we explored OpenCV tool.

Safe cars combine a variety of sensors to perceive their surroundings, such as radar, lidar, sonar, PGS, odometry and inertial measurement units. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage.

References

https://community.nxp.com/docs/DOC-1067
https://www.electricaleasy.com/2014/01/basic-working-of-dc-motor.html
https://www.raspberrypi.org/documentation/usage/camera/
https://www.raspberrypi.org/products/camera-module-v2/
https://www.ics.com/blog/introduction-gpio-programming
https://community.estimote.com/hc/en-us/articles/217429867-What-is-GPIO-
https://www.raspberrypi.org/documentation/hardware/raspberrypi/gpio/README.md
https://cdn-learn.adafruit.com/downloads/pdf/adafruit-16-channel-servo-driver-with-raspberry-pi.pdf
https://en.wikipedia.org/wiki/Self-driving_car
https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html
https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html
https://chatbotslife.com/self-driving-cars-advanced-computer-vision-with-opencv-finding-lane-lines-488a411b2c3d