ANT-BOT

A PROJECT REPORT

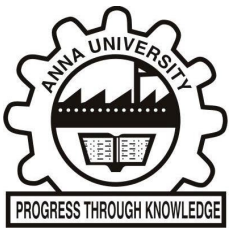
Submitted by

ABDUL RAHEEM	**2019504501**
ABHINAV R	**2019504502**
SANTHOSH S	**2019504578**
TARUN U	**2019504597**
VAIKUNTH GURUSWAMY	**2019504602**

in partial fulfillment for the award of the degree

IN

ELECTRONICS AND COMMUNICATION ENGINEERING



MADRAS INSTITUTE OF TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600 044

NOV 2021

ACKNOWLEDGEMENT

We wish to express our sincere gratitude to **Dr. M GANESH MADHAN**, Professor and Head, Department of Electronics Engineering, M.I.T campus,Anna University.

We wish to express our deep sense of gratitude to our guide **Mrs.V.Gowthami**, Teaching Fellow, Department of Electronics Engineering, for giving us the opportunity to undertake this project, for her valuable guidance, advice, motivation, encouragement, and support during the situations of problems we faced while doing the project. We gained a great amount of knowledge during the course of the project with her extensive vision, creative thinking and ideas have been a source of inspiration. We take immense pleasure in thanking her for her continuous guidance throughout this project.

We would like to extend our sincere gratitude to **Mrs.M.S.Vinotheni**, Teaching Fellow, Department of Electronics Engineering for her valuable suggestions and modifications during all the reviews which took our project to greater heights.

We would like to thank our project coordinators, all the teaching and non teaching staff members of the Department of Electronics Engineering, for their support in all aspects.

by:

ABDUL RAHEEM	**2019504501**
ABHINAV R	**2019504502**
SANTHOSH S	**2019504578**
TARUN U	**2019504597**
VAIKUNTH GURUSWAMY	**2019504602**

TABLE OF CONTENTS

1.ABSTRACT

With the rise in navigation technology, autonomous robots are needed for mapping regions that are otherwise dangerous for manual exploration. In this research project, a small-scale autonomous robot explores an environment using multiple experimental mapping and navigation algorithms. Ant-bot, a simplified rover that autonomously figures out the shortest distance and reaches the destination point without human interference and is controlled via Internet Of Things(IoT) devices. This thesis describes the entire process of its creation from hardware requirements and implementation, it will be compiled and simulated through a Windows application – VS Code in python language which will be connected through the ESP-32 Camera Module programmed in Arduino IDE. This project implements computing an escape route and traveling through to the escape point with the help of Image Processing which is implemented via Python code and the respective movement in real-time via hardware coding using Arduino.

2. INTRODUCTION

The goal of this research project is to map an unknown field and calculate the best path to travel from the start to the destination. The environment will be fed to the ant-bot. Using the python program the rover finds the most efficient route from the entrance to the exit.

The testing of different algorithms for mapping and navigating paths is backed by the desire to scale down real-world applications of affordable autonomous robots in order to gain a greater understanding of algorithms and robot development.

The Ant-bot will be trained on a separate machine and then transferred to an onboard computer that will control it. The rover will then be fully independent of other machines. Another goal of the thesis is to make this rover affordable and easy to build. A mainstream single board computer will be used with an inexpensive car chassis. The software will be written with standard machine learning libraries and easy to extend and reproduce.

3. HARDWARE

3.1.Esp32:

The ESP32-CAM is a very small camera module with the ESP32-S chip. Besides the OV2640 camera and several GPIOs to connect peripherals, it also features a microSD card slot that can be useful to store images taken with the camera or to store files to serve clients.

The ESP32-CAM doesn't come with a USB connector, so you need a USB to TTL UART programmer like CP2102 to upload code through the U0R and U0T pins (serial pins).

- The smallest 802.11b/g/n Wi-Fi BT SoC module
- Low power 32-bit CPU, can also serve the application processor
- Up to 160MHz clock speed, summary computing power up to 600 DMIPS
- Built-in 520 KB SRAM, external 4MB SRAM
- Supports UART/SPI/I2C/PWM/ADC/DAC
- Support OV2640 and OV7670 cameras, built-in flash lamp
- Support image WiFI upload
- Support TF card
- Supports multiple sleep modes

ESP32-CAM Pinout

The following figure shows the ESP32-CAM pinout (AI-Thinker module). There are three GND pins and two pins for power: either 3.3V or 5V.

GPIO 1 and GPIO 3 are serial pins. You need these pins to upload code to your board. Additionally, GPIO 0 also plays an important role, since it determines whether the

ESP32 is in flashing mode or not. When GPIO 0 is connected to GND, the ESP32 is in flashing mode.

The following pins are internally connected to the microSD card reader:

- GPIO 14: CLK
- GPIO 15: CMD
- GPIO 2: Data 0
- GPIO 4: Data 1 (also connected to the on-board LED)
- GPIO 12: Data 2
- GPIO 13: Data 3

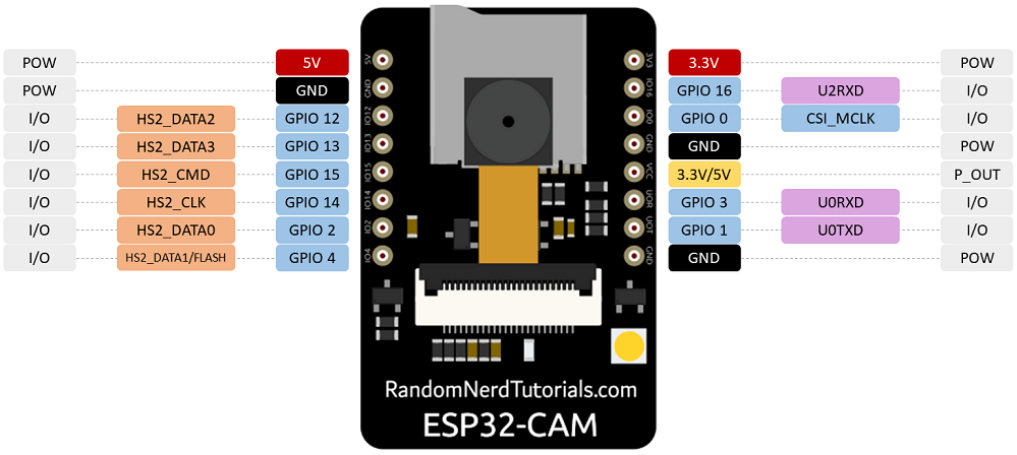


FIGURE 3.1.1: ESP 32 PIN DETAILS

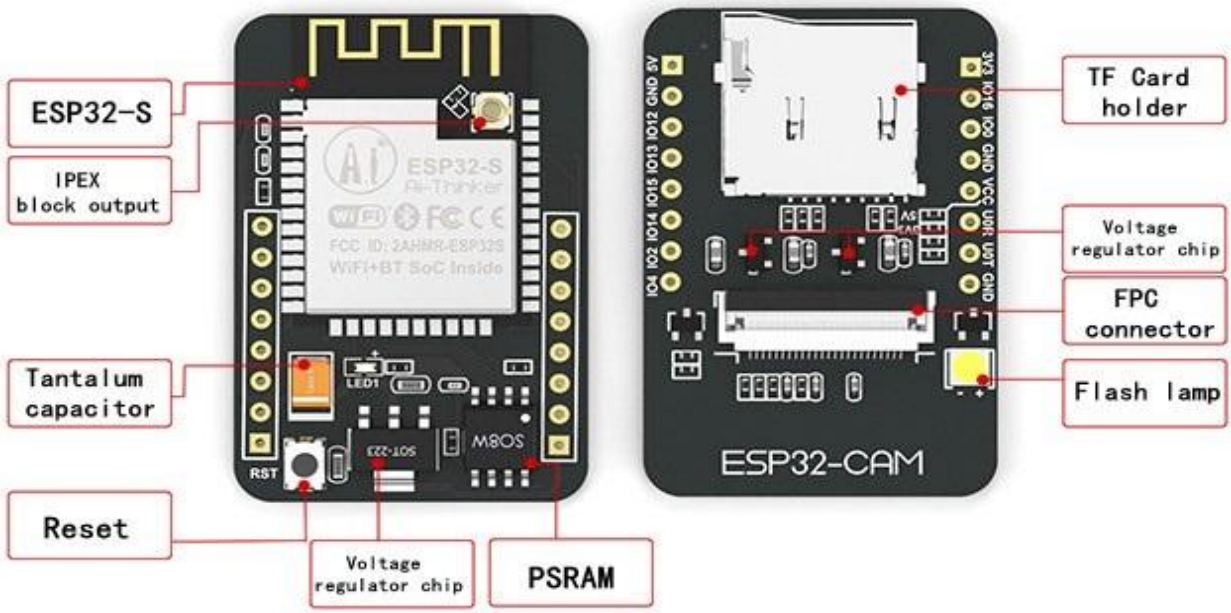


FIGURE 3.1.2: ESP 32 Camera Module

FIGURE 3.1.3: ESP32 Camera Module (Front and Back)

3.2.MB102 Breadboard Power Supply

MB102 Breadboard Power Supply module is one of the essential and low-cost components in the electronics labs. It powers the circuits and is also used for testing purposes. The small compact module is power efficient and can be operated using an input voltage range of 6.5 Volts to 12 Volts. The module has two voltage regulators which output 3.3 Volts and 5 Volts. Mb-102 also has an onboard capacitor for noise suppression and smoothing the input voltage.

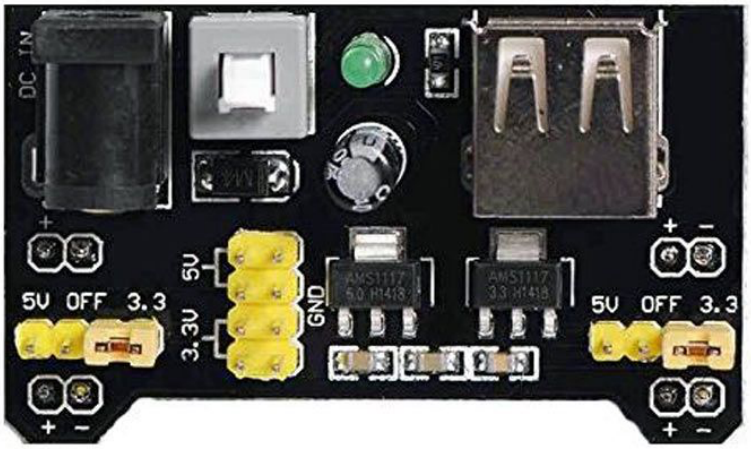


FIGURE 3.2.1: MB102 Power Supply Module

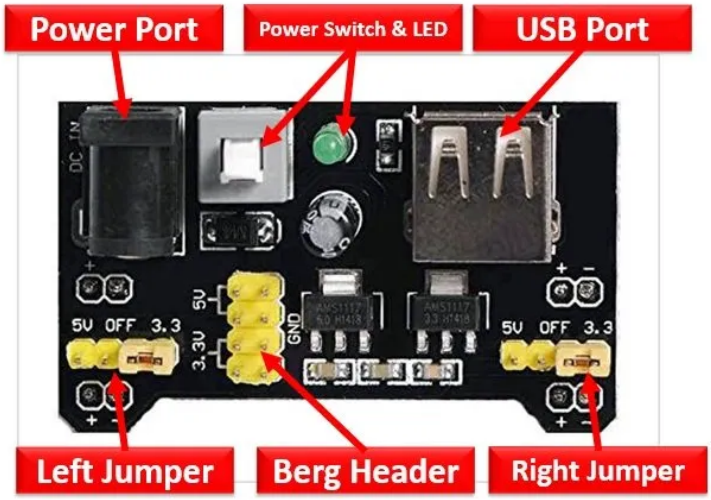


FIGURE 3.2.2: MB102 Power Supply Module Port Details

The breadboard power supply module consists of:

Power Port & USB Port: The DC power port and USB-A connector are provided to the module to power it up.

Power Switch & LED: A switch is embedded to provide extra control along with an LED to indicate the energizing of the module.

Jumpers: The mb102 breadboard supply module is capable of giving out 3.3 volts or 5 volts to breadboard rails. They can be operated individually.

Berg Headers: The berg headers can be used to output power to other devices as well.

Features and Specifications

- Operating Input voltage: 6.5 Volts – 12 Volts
- Output voltage: 3.3 Volts or 5 Volts
- Maximum Output Current: < 700 mA
- Module Dimensions: 5.3cm x 3.5cm
- The module has an on/off switch to control the external input switch.
- Along with the DC port, the Breadboard module has a USB port.
- The USB port provides input to the module to output power to the circuits.
- The module has selectable power rails that can be controlled independently.
- The module can switch between output voltages i.e 3.3V and 5V.
- Plug the BBPS module directly into the breadboard.
- For easiness, the BBPS module also has two pairs of onboard 3.3V and 5V DC output berg headers.

3.3.Servo Motor:

This SG90 servo motor controlled the direction of the robot. Here are some features:

- **Stall torque**: 1.8 kg/cm (4.8 V)
- **Gear type**: POM gear set
- **Operating speed:** 0.1 sec/60degree (4.8 V)
- **Operating voltage:** 4.8 V
- **Working frequency:** 50 Hz/1520 µs
- **Temperature range:** 0 ℃-55 ℃
- **Dead band width:** 1 µs
- **Power Supply:** Through External Adapter
- **Torque:** 2.2 kg
- Fits perfectly on LEGEND truck because it is an original accessory of LEGEND RC car SG90 servo inside the cavity of the LG-ZJO4 servo

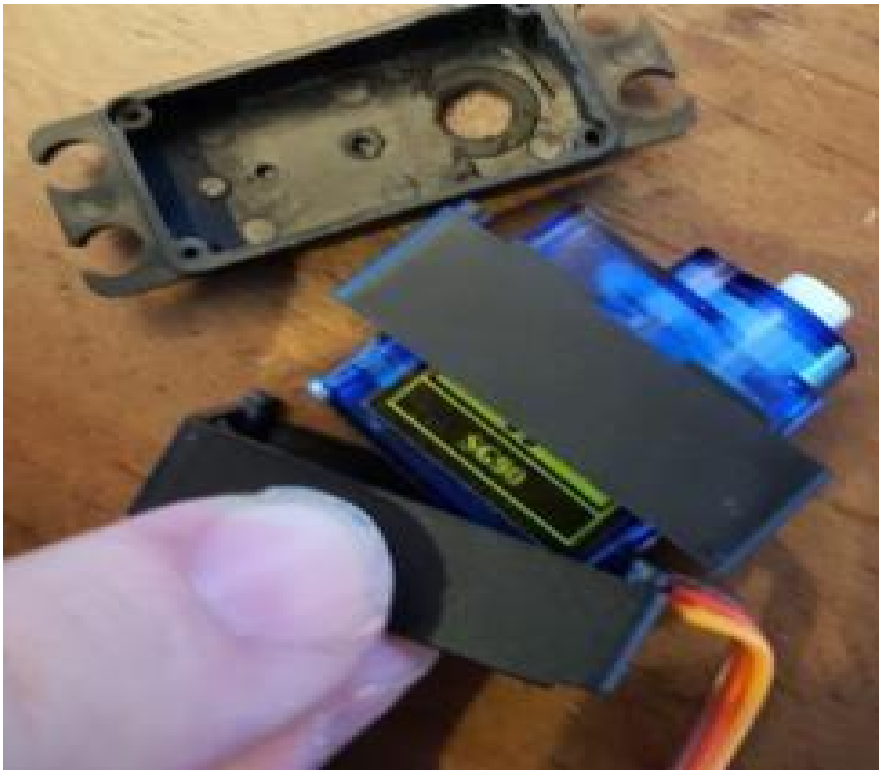


FIGURE 3.3.1: Servo Motor

The RC car had a specific cavity prepared to fit its original ‘LEGEND’ servo. But the program was designed to use servos of 3 wires (brown, red, yellow) and the original servo of the RC car has 5 (most RC cars use 5 wire servos). The LG-ZJ04 servo was the only one that fit the cavity and the SG90 micro servo was the only 3 wired servo that could be used. Thus, the case of the LG-ZJ04 was emptied and adapted in order to place the SG90 micro servo in.

3.4.DC Geared Motor and Chassis

The DC motor has a voltage supply range from 3-12 Volts.It has a speed of around 100 rpm. A 9 V battery is used to power the motors, and a power bank is used to power the Arduino Uno. The figure shows an example of the DC motor and chassis which are used in this project. The material for the chassis is acrylic with two wheels for steering and driving and one wheel for balancing the car.

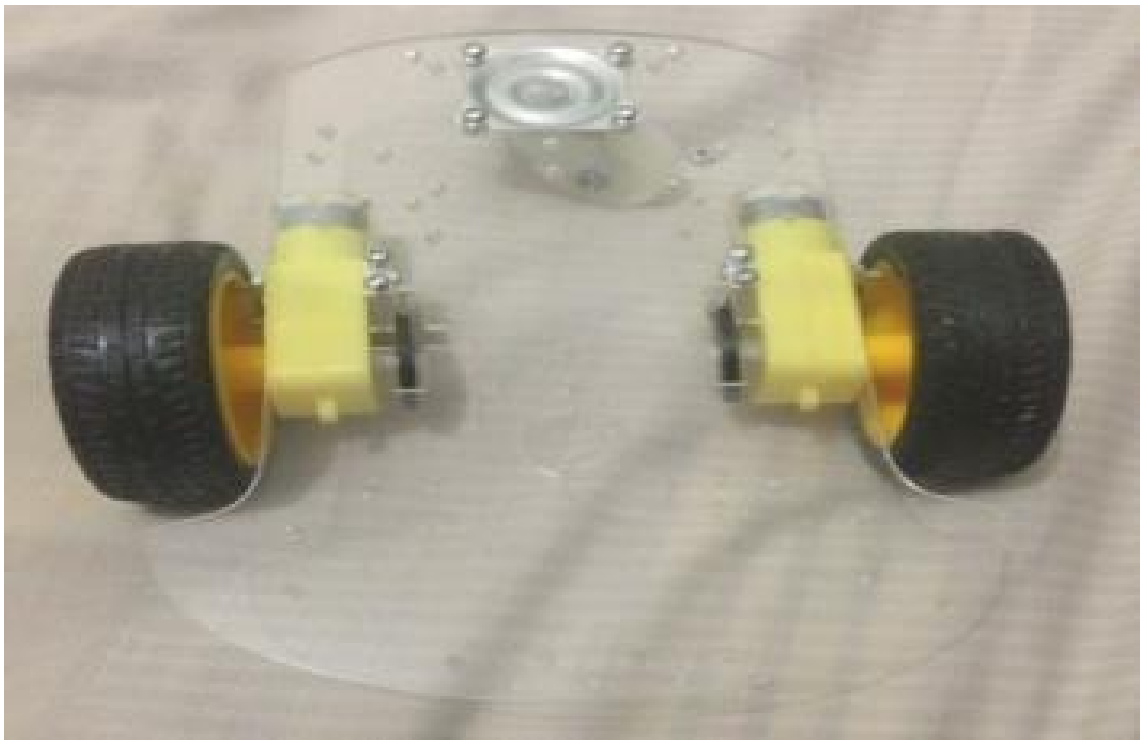


FIGURE 3.4.1: Robot Chassis

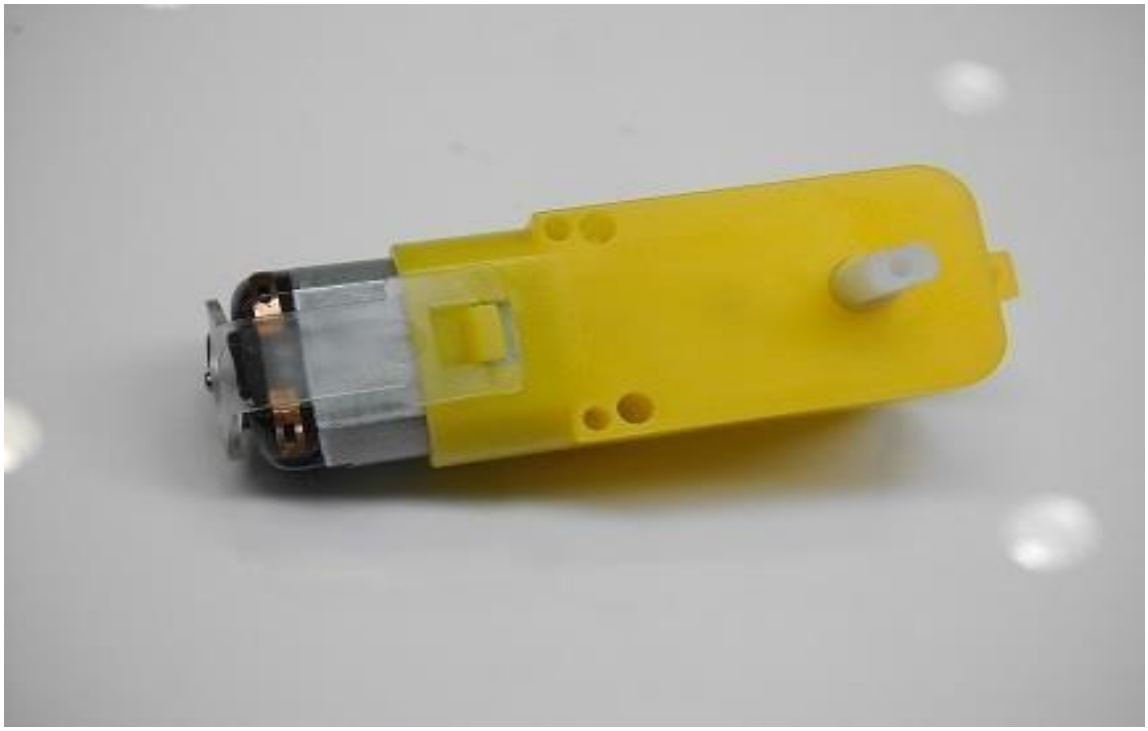


FIGURE 3.4.2: DC Motor

3.5.CP2102

The CP2102 is a highly-integrated USB-to-UART Bridge Controller providing a simple solution for updating RS-232 designs to USB using a minimum of components and PCB space. The CP2102 includes a USB 2.0 full-speed function controller, USB transceiver, oscillator, EEPROM, and asynchronous serial data bus (UART) with full modem control signals in a compact 5 x 5 mm MLP-28 package. No other external USB components are required.

USB Function Controller and Transceiver

The Universal Serial Bus function controller in the CP2102 is a USB 2.0 compliant full-speed device with an integrated transceiver and on-chip matching and pull-up resistors. The USB function controller manages all data transfers between the USB and the UART as well as command requests generated by the USB host controller and commands for controlling the function of the UART.

Asynchronous Serial Data Bus (UART) Interface

The CP2102 UART interface consists of the TX (transmit) and RX (receive) data signals as well as the RTS, CTS, DSR, DTR, DCD, and RI control signals. The UART supports RTS/CTS, DSR/DTR, and XOn/XOff handshaking.

The UART is programmable to support a variety of data formats and baud rates. The data format and baud rate programmed into the UART are set during COM port configuration on the PC.

Voltage Regulator

The CP2102 includes an on-chip 5 to 3 V voltage regulator. This allows the CP2102 to be configured as either a USB bus-powered device or a USB self-powered device. These configurations are shown in Figure 7 and Figure 8. When enabled, the 3 V voltage regulator output appears on the VDD pin and can be used to power external 3 V devices. See Table 8 for the voltage regulator electrical characteristics.

Alternatively, if 3 V power is supplied to the VDD pin, the CP2102 can function as a USB self-powered device with the voltage regulator disabled. For this configuration, it is recommended that the REGIN input be tied to the 3 V net to disable the voltage regulator. This configuration is shown in Figure 9. The USB max power and power attributes descriptor must match the device power usage and configuration. See application note “AN144: CP210x Customization Guide” for information on how to customize USB descriptors for the CP2102.

FIGURE 3.5.1: CP2102 Module

3.6.Basic NPN Transistor Switching Circuit

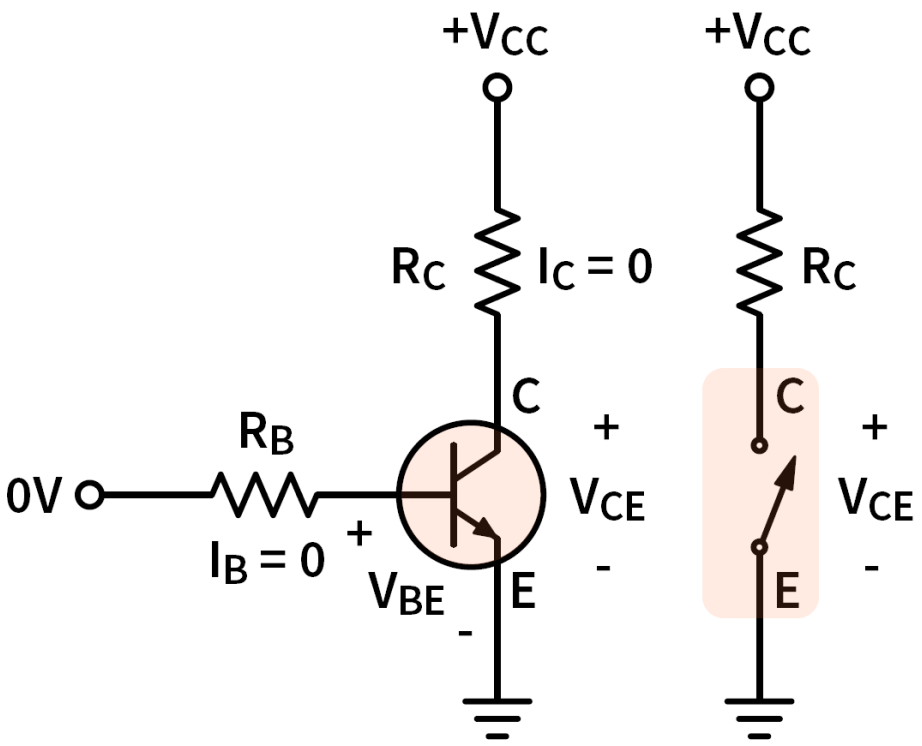


FIGURE 3.6.1: Transistor as a switch

- For the transistor as a switch, the transistor needs to be turned either fully “OFF” (cut-off) or fully “ON” (saturated). An ideal transistor switch would have infinite circuit resistance between the Collector and Emitter when turned “fully-OFF” resulting in zero current flowing through it and zero resistance between the Collector and Emitter when turned “fully-ON”, resulting in maximum current flow.

- In practice when the transistor is turned “OFF”, small leakage currents flow through the transistor, and when fully “ON” the device has a low resistance value causing a small saturation voltage (VCE) across it.

- Even though the transistor is not a perfect switch, in both the cut-off and saturation regions the power dissipated by the transistor is at its minimum.

- In order for the Base current to flow, the Base input terminal must be made more positive than the Emitter by increasing it above the 0.7 volts needed for a silicon device.

- By varying this Base-Emitter voltage VBE, the Base current is also altered and which in turn controls the amount of Collector current flowing through the transistor as previously discussed.

- When maximum Collector current flows the transistor is said to be Saturated. The value of the Base resistor determines how much input voltage is required and the corresponding Base current to switch the transistor fully “ON”.

4.SOFTWARE

4.1.Dijkstra Algorithm:

In this project, we have used the Dijkstra Algorithm to compute the shortest distance. The starting and destination point is given and by using this algorithm the shortest path is traced by the program.

Dijkstra's Algorithm is a popular graph theory algorithm. It is used to find the shortest path between detected points in the graph. It starts with a source point and known edge lengths between points.

First, assign each point with the initial value of ∞ except the source point which has the initial value of 0. Then the distance from each point to the closest adjacent point is found out. The initial adjacent point from the source, takes the values of distance from the source.

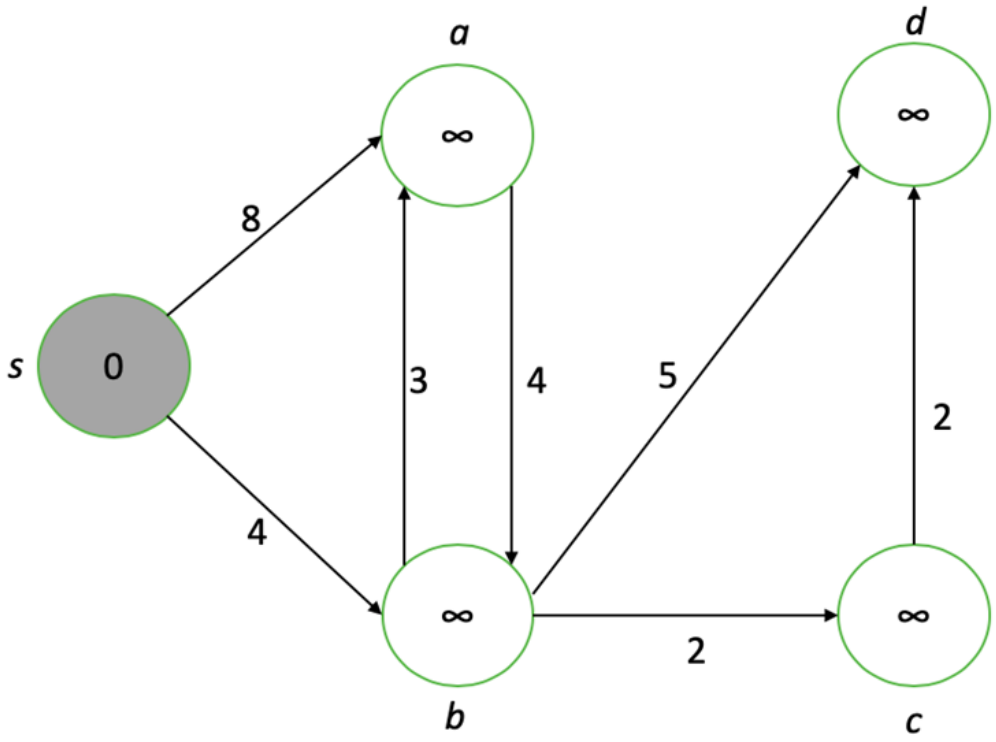


FIGURE 4.1.1: Representation of Dijkstra Algorithm

Once all the adjacent points to the source have taken value, then the focus shifts to one of the adjacent points, say point X which directs towards the next set of closest points. These points adjacent to the point of interest X take the value of the sum of the distance of the point from X and the distance of point X from the source

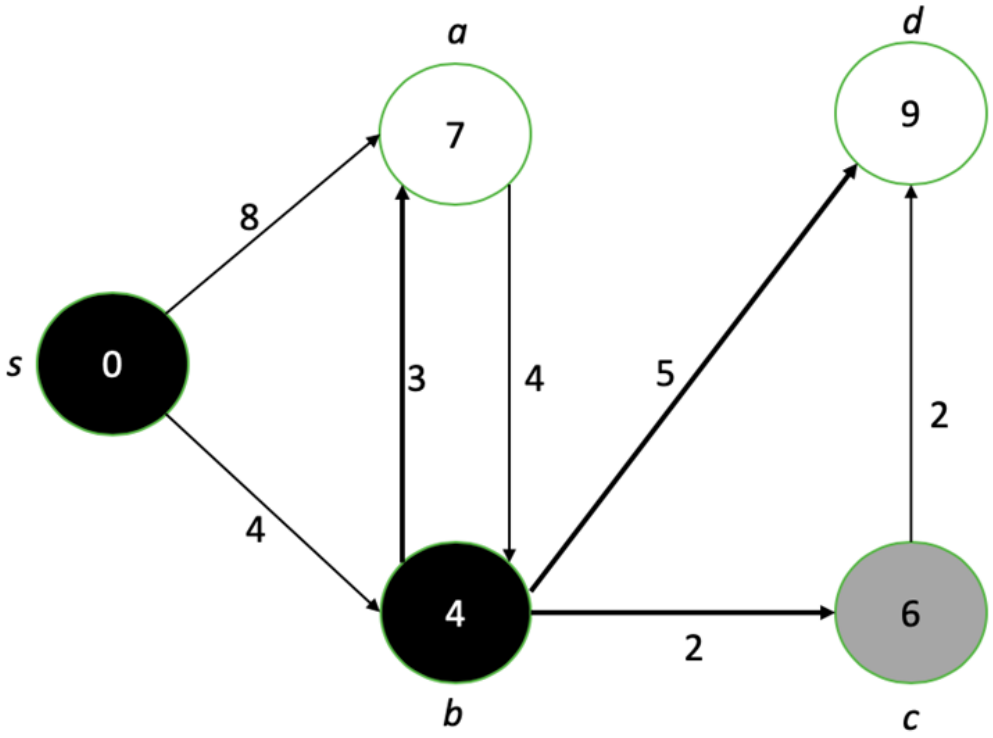


FIGURE 4.1.2: Implementation of Dijkstra Algorithm

So, this process continued until all of the points detected assumed a value. In the end, the shortest path length is detected by connecting the points of the adjacent points with the shortest length successively.

In order to detect the path, we visualize the maze to be a collection of black and white points, where black has an RBG value of (0,0,0) and white has an RBG value of (255,255,255). In order to accomplish the required task, we find the distance

between the adjacent points using the Euclidean Squared Distance formula and summing with 0.1 so that we do not end up with a 0 distance between points.

Euclidean Distance Formula

$$= 0.1 + (R_1 - R_2)^2 + (B_1 - B_2)^2 + (G_1 - G_2)^2$$

Our objective here is being able to reach the destination from the source point without crossing over any of the black points, assuming we follow along the white path to reach the destination.

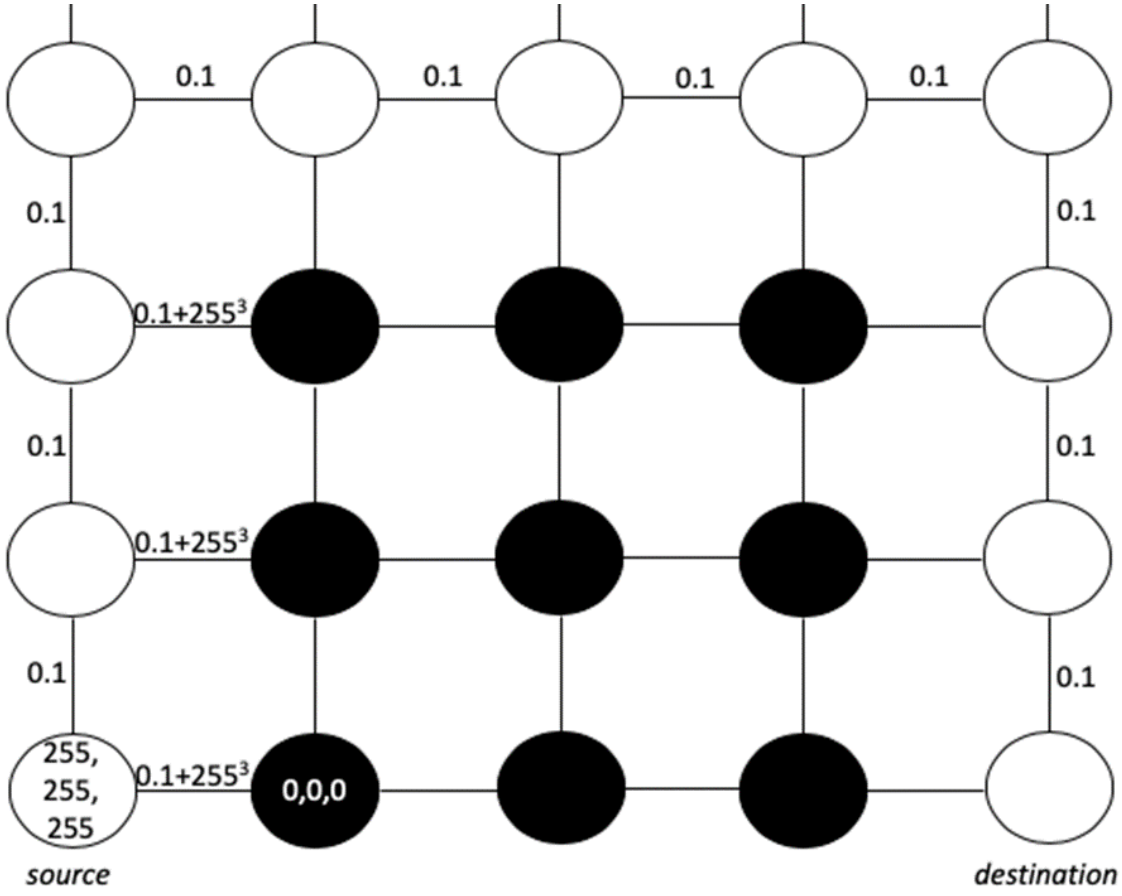


FIGURE 4.1.3: Working of Dijkstra Algorithm

Reference:

https://towardsdatascience.com/solving-mazes-with-python-f7a412f2493f

4.2.Maze solving

A maze is a puzzling way that consists of different branches of passages where the aim of the micro mouse is to reach the destination by finding the most efficient route within the shortest possible time.

Ant bot is a small wheeled robot that consists of an ESP32 camera Module,motors, and controller and acts. Artificial Intelligence plays a vital role in defining the best possible way of solving any maze effectively.

Graph theory appears as an efficient tool while designing proficient maze solving techniques. The graph is a representation or collection of sets of nodes and edges. This concept is deployed in solving unknown mazes consisting of multiple cells. Depending on the number of cells, the maze dimension may be 8x8, 16x16, or 32x32. The IEEE standard maze has 16×16 square cells, and each cell‟s length and width are both 18 centimeters. Each cell can be considered as a node that is isolated by walls or edges

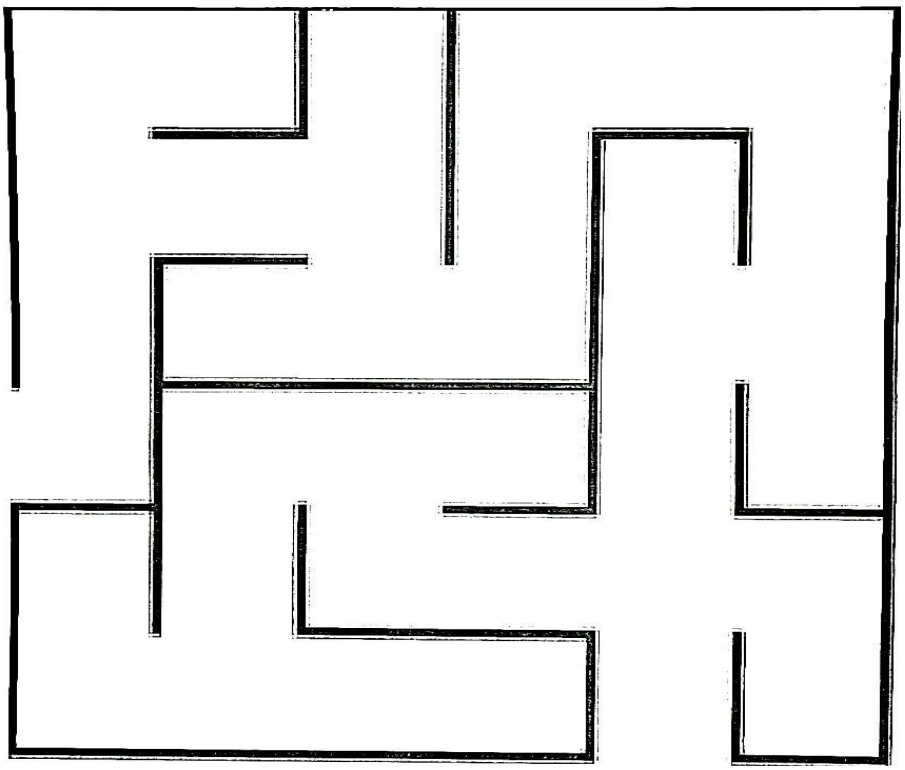


FIGURE 4.2.1: Image of Maze

Maze solving is a very old problem, but still, now it is considered an important field of robotics. This field is based on decision-making algorithms. The main aim of this project is to make an autonomous maze solver robot. In this project Hardware development, software development and maze construction had been done. For performance testing, the robot will implement to solve any given maze. The capability of finding the shortest path is also verified

Image processing software determines the location of the walls and uses them to solve the maze given a start and stop location on the mazes that are programmed in. The maze is solved for every location the rover can be in on the maze. The directions for how the balls should move are then sent to ESP 32 that tells the motors how to move to navigate the ball through the maze.

4.3. Edge detection:

Edge Detection is a technique of image processing used to identify points in a digital image with discontinuities, simply to say, sharp changes in the image brightness. These points where the image brightness varies sharply are called the edges (or boundaries) of the image.

It is one of the basic steps in image processing, pattern recognition in images, and computer vision. When we process very high-resolution digital images, convolution techniques come to our rescue.

Edge Detection Operators are of two types:

- Gradient-based operator which computes first-order derivations in a digital image like, Sobel operator, Prewitt operator, Robert operator

- Gaussian–based operator which computes second-order derivations in a digital image like, Canny edge detector, Laplacian of Gaussian

Sobel Operator:

It is a discrete differentiation operator. It computes the gradient approximation of the image intensity function for image edge detection. At the pixels of an image, the Sobel operator produces either the normal to a vector or the corresponding gradient vector. It uses two 3 x 3 kernels or masks which are convolved with the input image to calculate the vertical and horizontal derivative approximations respectively –

Advantages of sobel operation:

1. Simple and time-efficient computation
2. Very easy at searching for smooth edges

Limitations of sobel operation:

1. Diagonal direction points are not preserved always
2. Highly sensitive to noise
3. Not very accurate in edge detection
4. Detecting with thick and rough edges does not give appropriate results.

Prewitt Operator:

This operator is almost similar to the Sobel operator. It also detects the vertical and horizontal edges of an image. It is one of the best ways to detect the orientation and magnitude of an image. It uses the kernels or masks

Advantages of prewitt operation*:*

1. Good performance on detecting vertical and horizontal edges
2. Best operator to detect the orientation of an image

Limitations of prewitt operation:

1. The magnitude of the coefficient is fixed and cannot be changed
2. Diagonal direction points are not preserved always

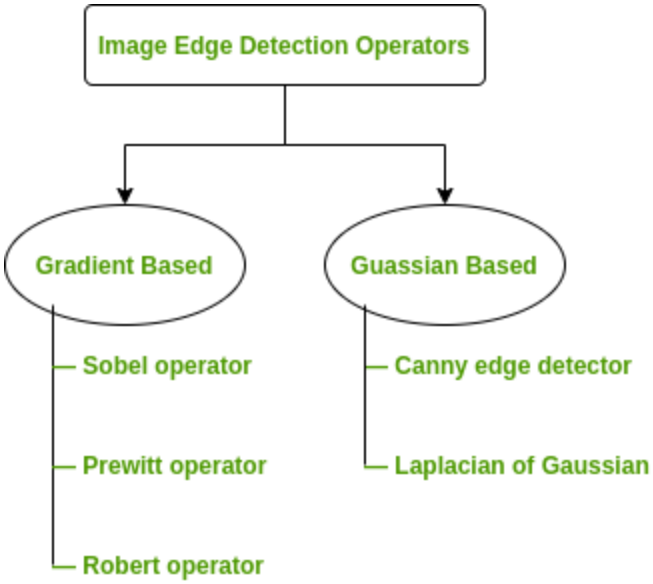


FIGURE 4.3.1: Classifications

Robert Operator: This gradient-based operator computes the sum of squares of the differences between diagonally adjacent pixels in an image through discrete differentiation. Then the gradient approximation is made. It uses the following 2 x 2 kernels or masks –

Advantages of robert operator:

1. Detection of edges and orientation is very easy
2. Diagonal direction points are preserved

Limitations of robert operator:

1. Very sensitive to noise
2. Not very accurate in edge detection

4.4. Instruction Set Generation:

The output from the Dijkstra algorithm generates pixelated values for the path to be traveled from the starting to the destination point mentioned by the user. The pixelated values need to be processed in order to pass to the ESP 32 camera module. In order to process the generated values, the maze dimensions (length and width in centimeters) are to be fed as an input to the program.

Edge detection protocol is performed for the pixelated values generated using the Dijkstra Algorithm. The coordinates for the edges of the path are stored as the output to the Edge detection algorithm.

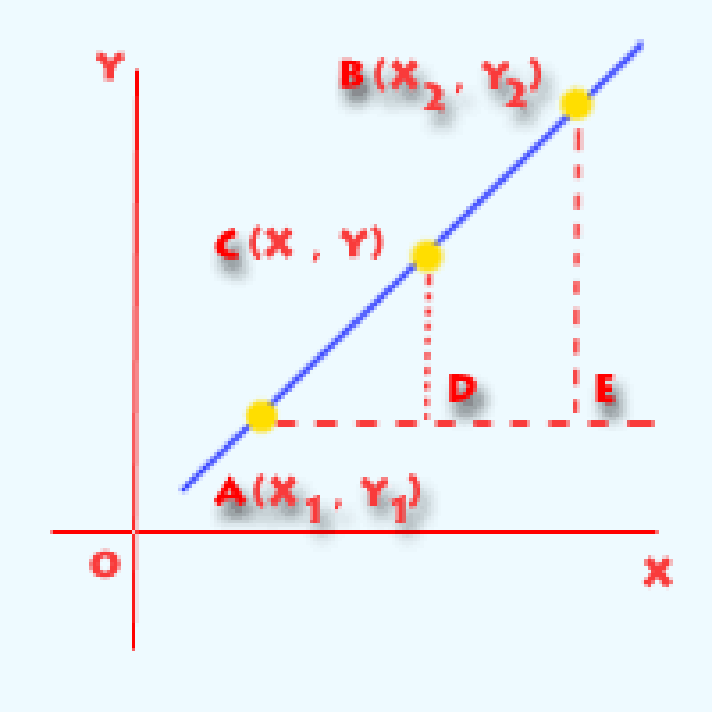


FIGURE 4.4.1: Two Point-Slope Form

The slope of the line joining the consequent coordinates from the Edge detection output set is to be calculated. The slope is calculated with reference to the above image and the below-mentioned equation.

$$M=(Y_2 - Y_1)/(X_2 - X_1)$$

Where,

M is the slope.

(X1, Y1) is the cartesian coordinates of the first pixel.

(X2, Y2) is the cartesian coordinates of the second pixel.

The consequent slopes are determined and the difference in the slopes are used to generate the,

Directions :

Left- If the difference of the slopes so obtained is positive.

Right-If the difference of the slopes so obtained is negative.

Straight-If the difference of the slopes so obtained is zero.

In an exceptional case, where the change in the orientation is greater than 90 degrees consecutive, two same instructions are passed depending upon the sign of difference of the consecutive slopes.

Degree in a change of orientation:

The difference of the values obtained from the inverse tangent function of the consecutive slopes for the edge detection output set coordinates is used to find the difference in the angle of approach.

If the value so obtained:

If greater than 90 degrees then multiple sets of instruction are generated such that two instruction sets are passed such that each has a value of 90 degrees.

Distance:

Depending upon the maze dimensions the user gives as input the distance to be covered per instruction is calculated. The distance between the consecutive points in the Edge detection output set is calculated using the Euclidean distance formula.

Euclidean Distance Formula: $\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

In an exceptional case, if the slope difference of the consecutive coordinates in the Edge detection output set is greater than 90 degrees
then two instructions are passed into the instruction set.

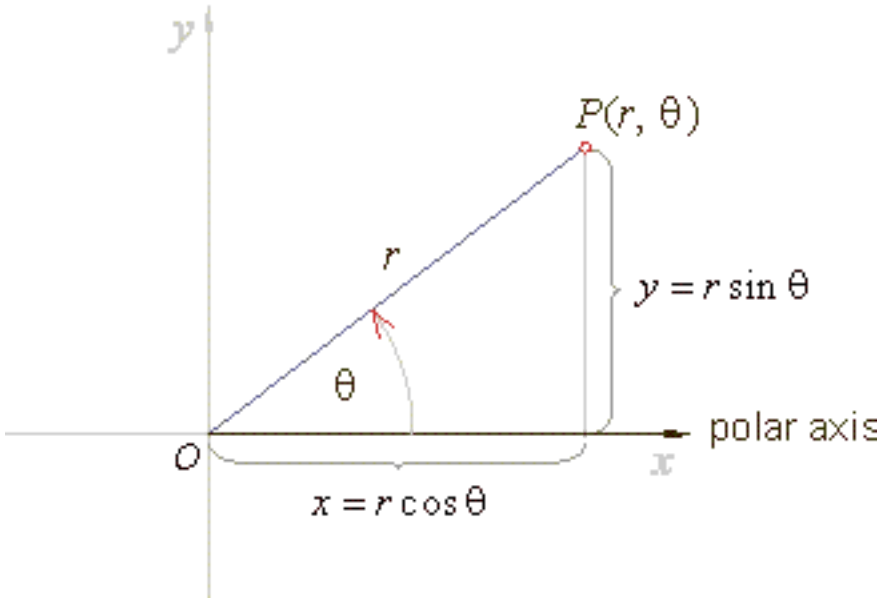


FIGURE 4.4.2 : Polar Coordinate Form

The above image depicts the manner in which the ANTBOT should navigate in this case. The ‘r’ in the image represents the shortest distance between the two points, since the angle is greater than 90 degrees the bot will travel along the polar axes, ‘x’ and ‘y’.

The image size, height x width (in pixels), and the user given maze dimension are compared and a proportional value is obtained for extrapolating the instruction set with respect to the physical maze to be solved.

4.5. Pygame Simulation:

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language.

This platform provides an efficient way to simulate various environments. In ANTBOT, Pygame is used to execute the instruction set generated in a virtual environment.

The maze image is passed as an input to the program to get the instruction set and as an environment to the Pygame. The instruction set is generated as a sequence depending upon the input car speed. The edges of the maze image input are detected and the car travels in the detected path following the instruction set so generated above.

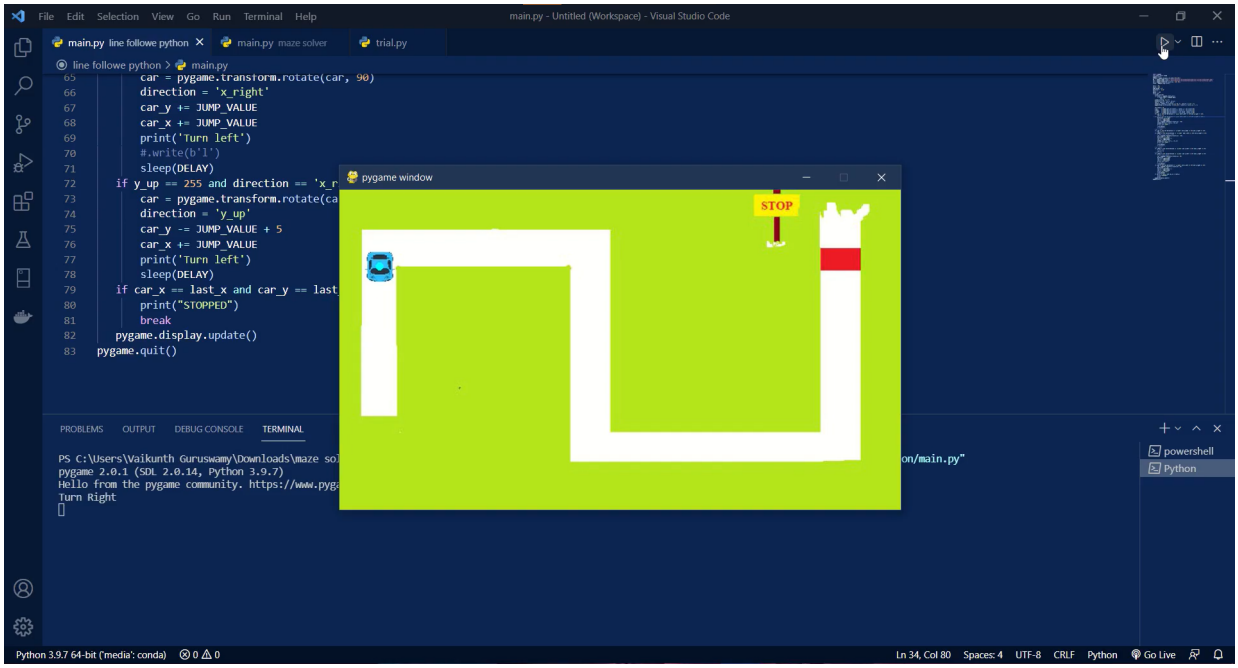


FIGURE 4.5.1: Pygame simulation

4.6. ARDUINO IDE

Arduino IDE is open-source software, designed by Arduino.cc and mainly used for writing, compiling & uploading code to all Arduino Modules and other modules too. It is based on Processing. Arduino code is written in C, C++ with an addition of special methods and functions and is based on wiring.

Arduino IDE has 2 parts:

i) Editor for writing the required code.

ii) Compiler for compiling and uploading the code.

Arduino IDE environment has 3 sections:

i) Menu and toolbar containing options like Verify, Upload, File, Sketch, Serial monitor, etc.

ii) Text editor where the text is written.

iii) Output pane where compilation status of running code and errors are shown and status bar showing the board and the port used

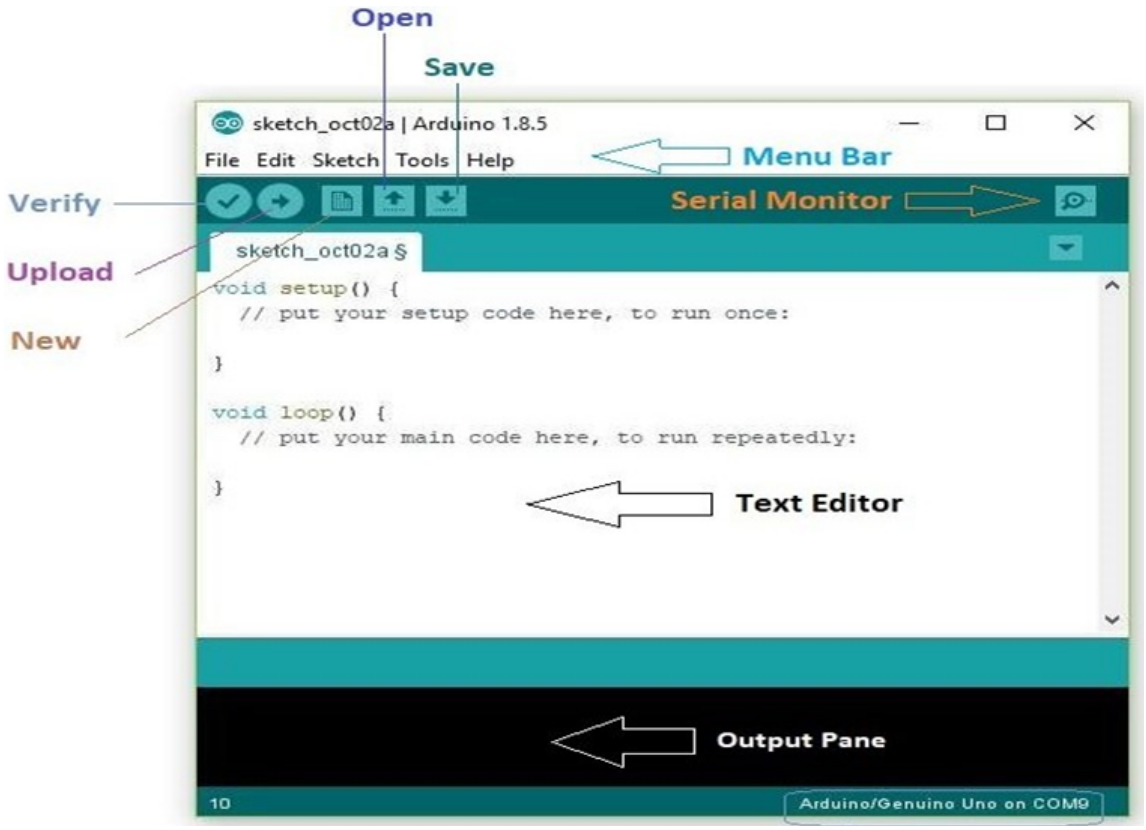


FIGURE 4.6.1: ARDUINO IDE

The main code, also known as a sketch, created on the IDE platform will ultimately generate a Hex File which is then transferred and uploaded in the controller on the board. Arduino files are saved as .uno file extensions.
Arduino IDE is preferred because it is cross-platform (can run on Windows, macOS, Linux), is easy to use for beginners, and flexible for advanced programmers too as advanced programming can be done too and can be extended too, can be extended to boards other than Arduino boards, has lots of libraries which are very helpful and has a serial monitor which helps to easily visualize the error.

Arduino IDE is used in our project to upload code into the ESP32 camera module. The ESP32 camera module can be added by going to File> Preferences and then entering
https://dl.espressif.com/dl/package_esp32_index.json
into the "Additional Board Manager URLs" field and then, clicking "OK". Now, the board will be added to IDE and can be downloaded using Tools > Board > Boards Manager and pressing the install button on the "ESP32 by Espressif Systems". After being installed, again go to Tools > Board to select the required ESP32 board. Additional libraries can be downloaded if needed and added to the IDE.

Coding is done using Arduino code with some differences for the ESP32 camera module. Code is verified and uploaded into the ESP32 camera module and if any error is detected, it is then corrected. The serial monitor is useful for debugging to find the errors in the code. It is found on the right-hand side of the toolbar and can be opened only when the board is connected to the system. Before uploading, the correct port where the device is inserted is to be mentioned or errors will arise. Input can be given to and output can be taken from the serial monitor which acts as an interface between the board and user to find the problems in the code and debug them.

4.7. Tinkercad Simulation:

The instruction set generated is tested using Tinkercad simulating software and Arduino UNO board in replacement of ESP32.

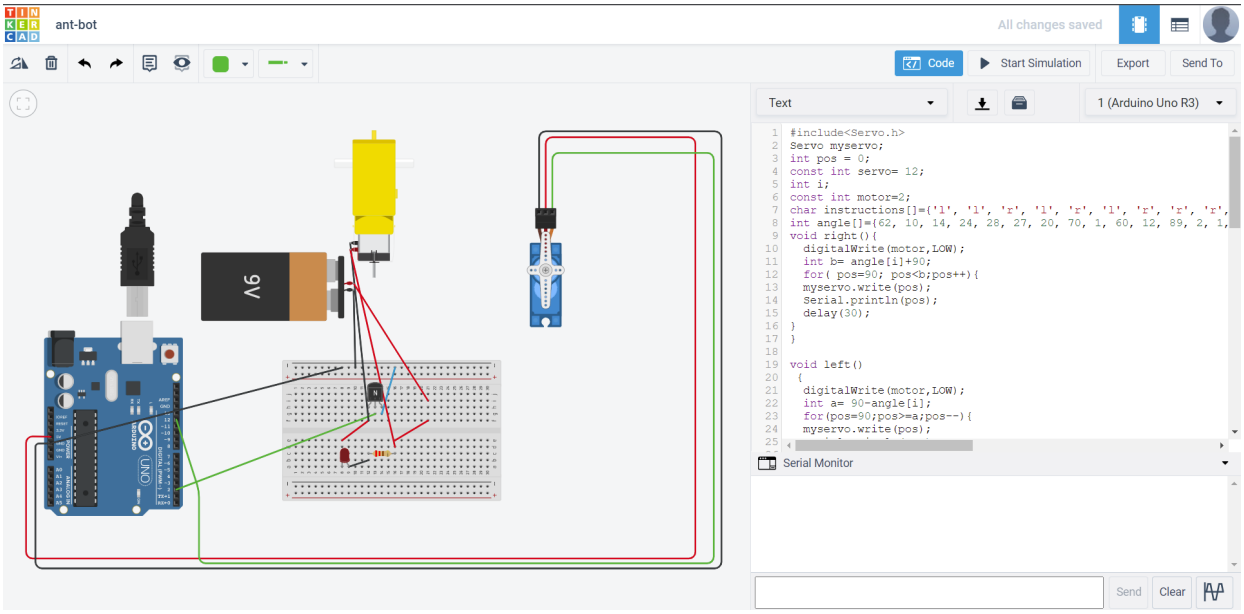


FIGURE 4.7.1: Tinkercad Simulation

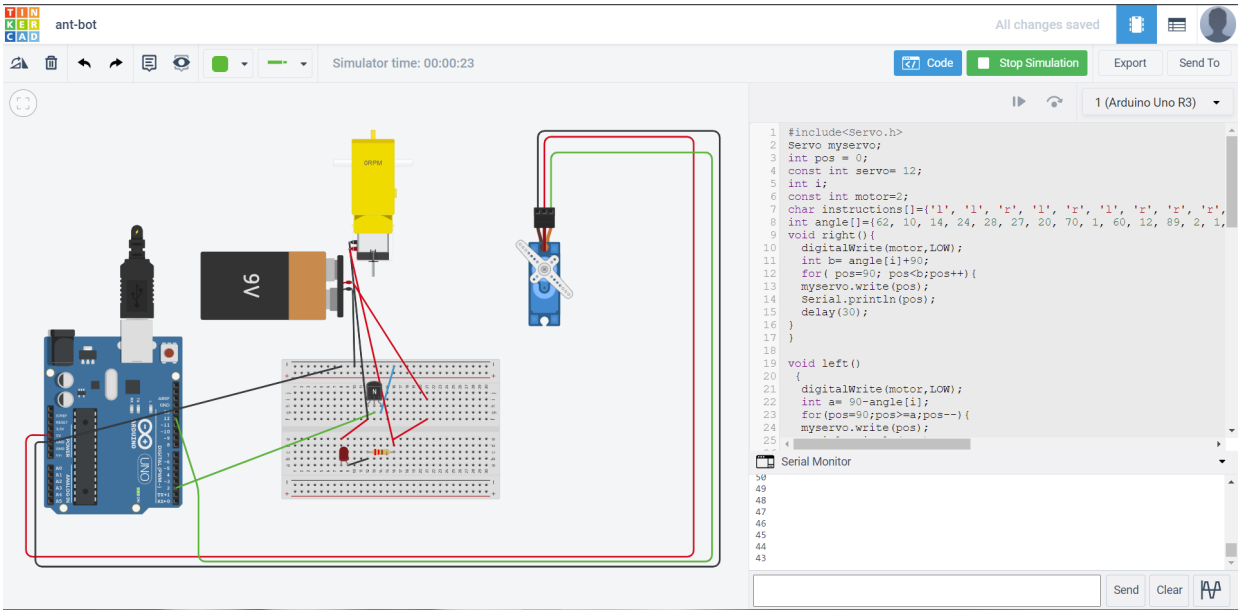


FIGURE 4.7.2: Tinkercad Simulation (with serial monitor output)

Displacement in the servo motor orientation and the DC motor functionalities are observed.

5.BLOCK DIAGRAM

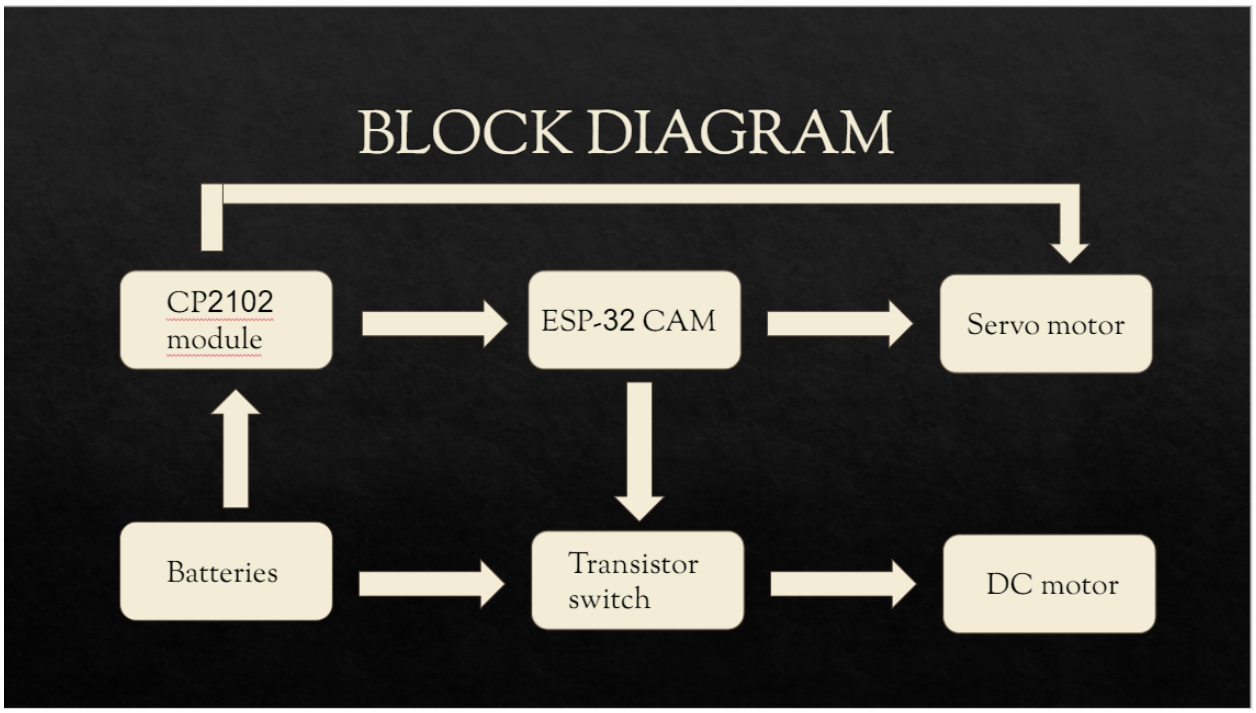


FIGURE 5.1: Block Diagram

The battery is the main power source for the ant-bot. The power is passed on through the BJT switch and FTDI module. The regulated power supply is further passed on to the ESP-32 for processing and generation of the instruction set and sends a control signal to the BJT switch and servo motor for its operations.

The output from the BJT switch is passed to the DC motor for movements of the bot.

6. ANT-BOT WORKING

6.1.UPLOADING CODE TO ESP32:

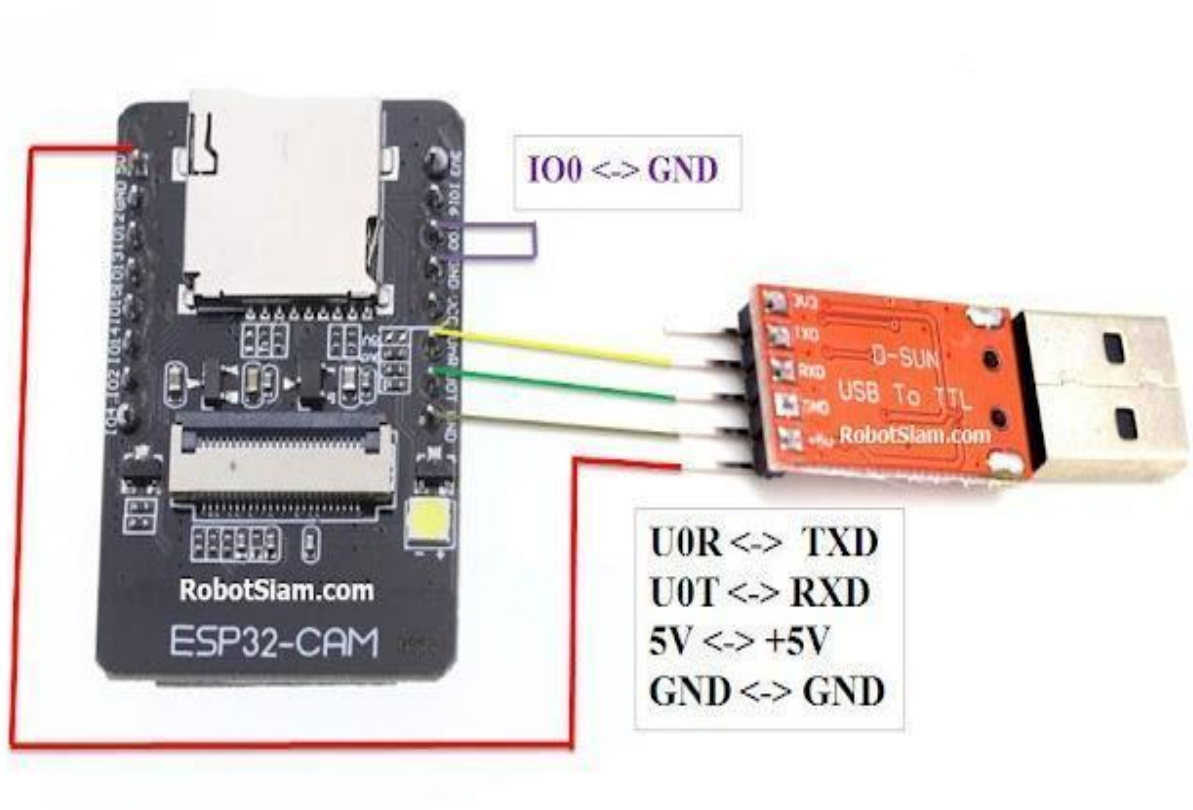


FIGURE 6.1.1: Hardware Connection

First, Arduino code needs to be uploaded into the ESP 32 camera module in order to be executed. For this, we connect the ESP 32 camera module with CP2102 USB to the TTL UART module. It is done so because the ESP 32 camera module doesn't have any way to communicate with the system by itself. CP2102 is connected to ESP 32 camera module first and this setup is then connected to the system to upload the code into ESP 32 camera module.

Connections between the ESP32 camera module and CP 2102 are as follows:

U0R<->TXD

UOT<->RXD

These connections are due to the fact that communication used here is UART communication where U0RXD (GPIO3) of ESP32 camera is connected to TXD of CP2102 and TXD (GPIO1) of ESP 32 camera module is connected to RXD of CP2102

5V<->+5V

5V of the ESP32 camera module is powered by +5V pin of CP2102 module

GND<->GND3

The third ground of the ESP32 camera module and ground of CP2102 are connected together

GPIO0<->GND2 (ESP32)

There is also this connection between two pins of the ESP32 camera module. This is done to keep the ESP 32 camera module in flashing mode. Flashing mode is the mode in which code can be uploaded into the ESP32 camera module. When this connection is removed, the ESP32 camera module will go into the normal execution mode and the code can’t be uploaded anymore.

6.2.CIRCUIT DIAGRAM

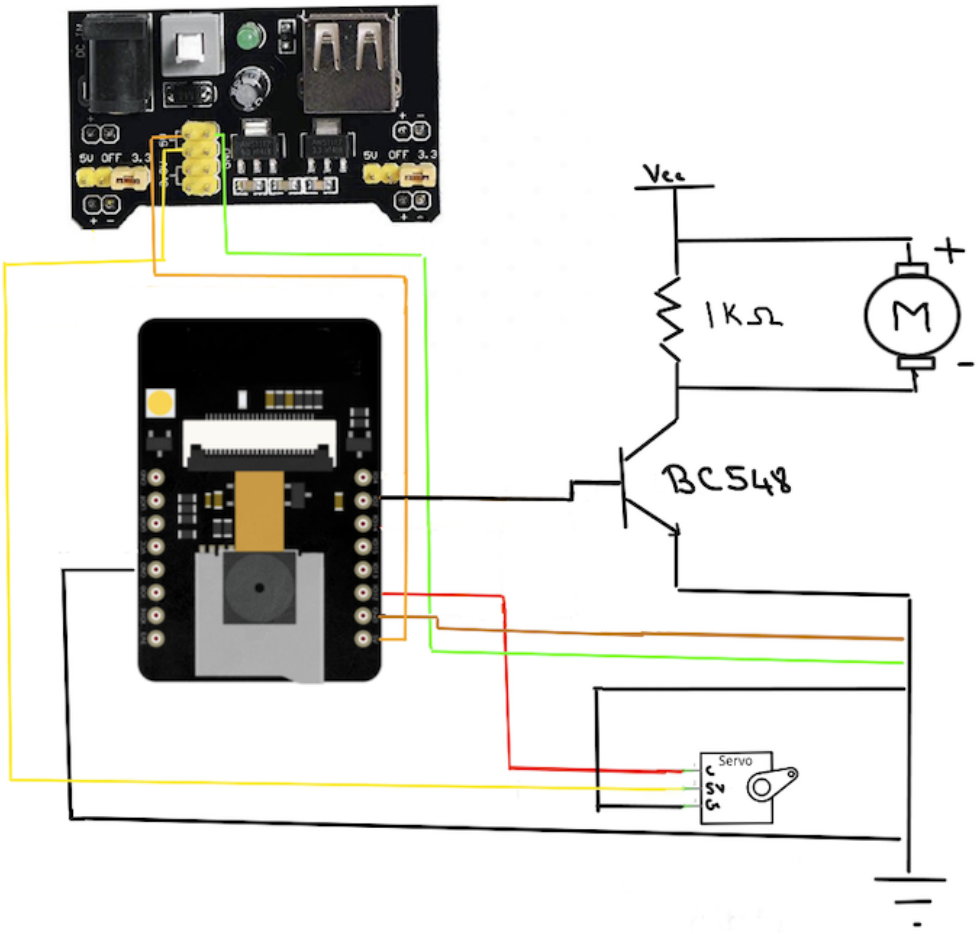


FIGURE 6.2.1: Circuit Diagram

Connections from MB102 5V, 3.3 V power supply module are as follows:

(5V) 1<-> 5V (ESP 32 camera module)

The first 5V voltage regulated power supply from MB102 is given to 5V of the ESP32 camera module to power it up.

(GND) 1<-> Common GND

The first ground of MB102 is connected to the common ground.

(5V) 2<-> 5V (Servo motor)

The second 5V voltage regulated power supply from MB102 is given to 5V of Servo motor to power it up

Connections from the ESP32 camera module are as follows:

5V <-> (5V) 1 (MB102)

The 5V ESP32 camera module powers up from the first 5V voltage regulated power supply from MB102.

(GND) 1<-> Common GND

The first ground of the ESP 32 camera module is connected to the common ground

(GND) 2<-> Common GND

The second ground of the ESP32 camera module is connected to the common ground

GPIO 12<-> Control (Servo motor)

GPIO 12 of ESP32 camera module gives control signal to the servo motor and is connected to control pin of servo motor

GPIO 2<-> Base (BC 548)

GPIO 2 of the ESP32 camera module gives a control signal which is used in the switch circuit to control the motor to the base of the BC548 transistor.

Connections from BC 548 are as follows:

Base<-> GPIO2 (ESP32 camera module)

The base of the BC548 transistor is given a control signal for the switch circuit to control the motor by GPIO 2 of the ESP32 camera module.

Emitter<-> Common GND

The emitter of the BC 548 transistor in the switch circuit is connected to the common ground.

Collector<-> 1kΩ resistor

Collector of BC548 transistor in the switch circuit is connected to 1kΩ resistor

Collector<-> Negative terminal (DC motor)

The collector of the BC 548 transistor in the switch circuit is connected to the negative DC motor terminal.

Connections from Servo motor are as follows:

Control<-> GPIO12 (ESP32 camera module)

The control pin of the servo motor is connected to GPIO 12 of the ESP32 camera module and receives a control signal from it

GND<-> Common GND

The ground of the servo motor is connected to the common ground

5V<-> (5V) 2 (MB102)

5V of Servo motor is powers up from the second 5V voltage regulated power supply from MB102

Other connections are:

(Vcc)1<-> MB102

First External power supply powers up the MB102 module which in turn gives the voltage regulated power of 5V and 3.3 V as output

(Vcc)2<->Positive terminal(DC motor)

The second power supply is connected to a positive DC motor terminal which powers up the DC motor.

(Vcc)2<->1kΩ resistor

The second power supply is connected to a 1kΩ resistor

6.3.CONNECTION FLOW

MB102 is supplied with power supply (Vcc)1. Voltage regulated power is supplied from MB102 to Servo motor and ESP 32 camera module. ESP 32 camera module gives control voltage to Servo motor and base of BC548 in the switch circuit. Control voltage from the ESP32 camera module is used by the servo motor to change the angle of the servo motor from 0 degrees to 180 degrees.

Control voltage from the ESP32 camera module is used as base voltage in the switch circuit done using BC548. The Collector of BC 548 in the switch circuit is connected to the negative terminal of the DC motor and 1kΩ resistor. (Vcc)2 is connected to 1 kΩ resistor and positive terminal of DC motor to power up the DC motor. The emitter of BC548 in the switch circuit is connected to the common ground. When the base voltage is less than 0.7 V, it becomes an open circuit where there is no connection between the negative DC motor terminal connected to the collector and the common ground switching OFF the motor. Otherwise, it is switched ON.

The first and second grounds of the ESP 32 camera module, the ground of Servo motor and MB102 are connected to the common ground.

7.OUTPUT

Output from Dijkstra algorithm:

After solving the input image from the user using the Dijkstra algorithm and overlapping the resultant over the input image the following image is obtained:

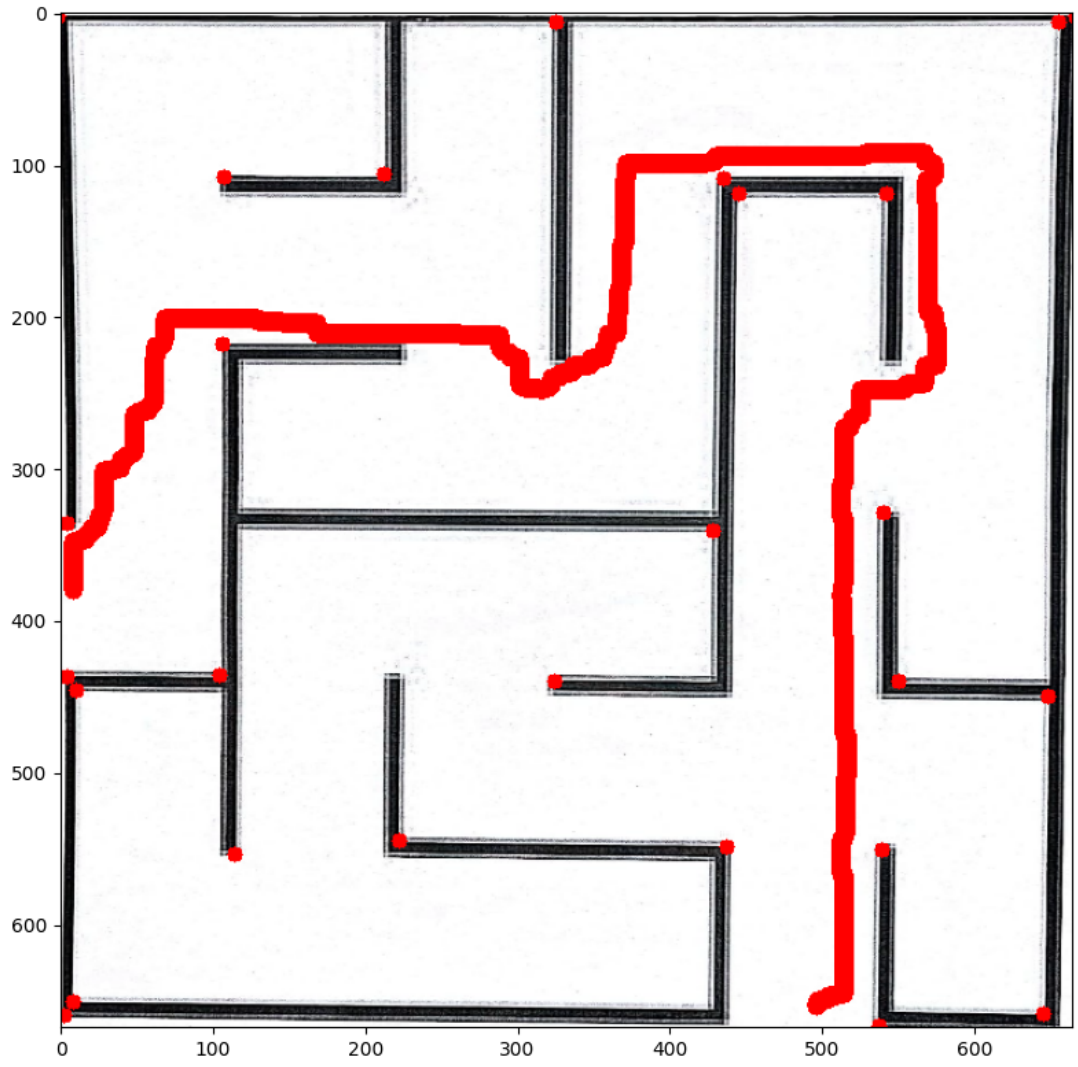


FIGURE 7.1: Python Turtle Simulation

The red color path is the required path to reach the destination from the starting point.

The red dots denote the edges of the input image.

The resultant path is further extracted from the above image and preprocessed before performing edge detection.

On performing edge detection to the resultant image corner points of the path and the output so obtained is presented below:

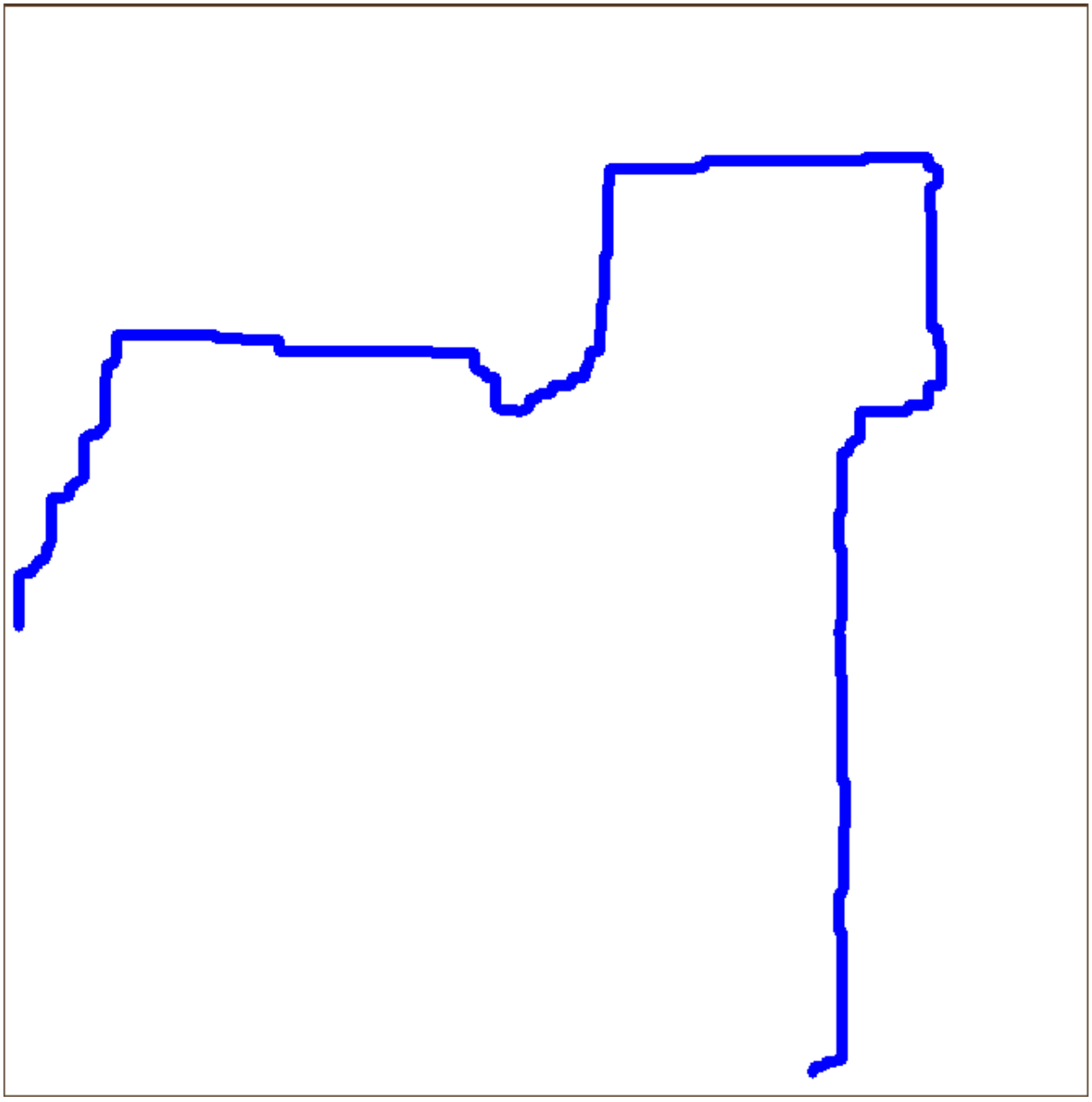


FIGURE 7.1: Path Extraction

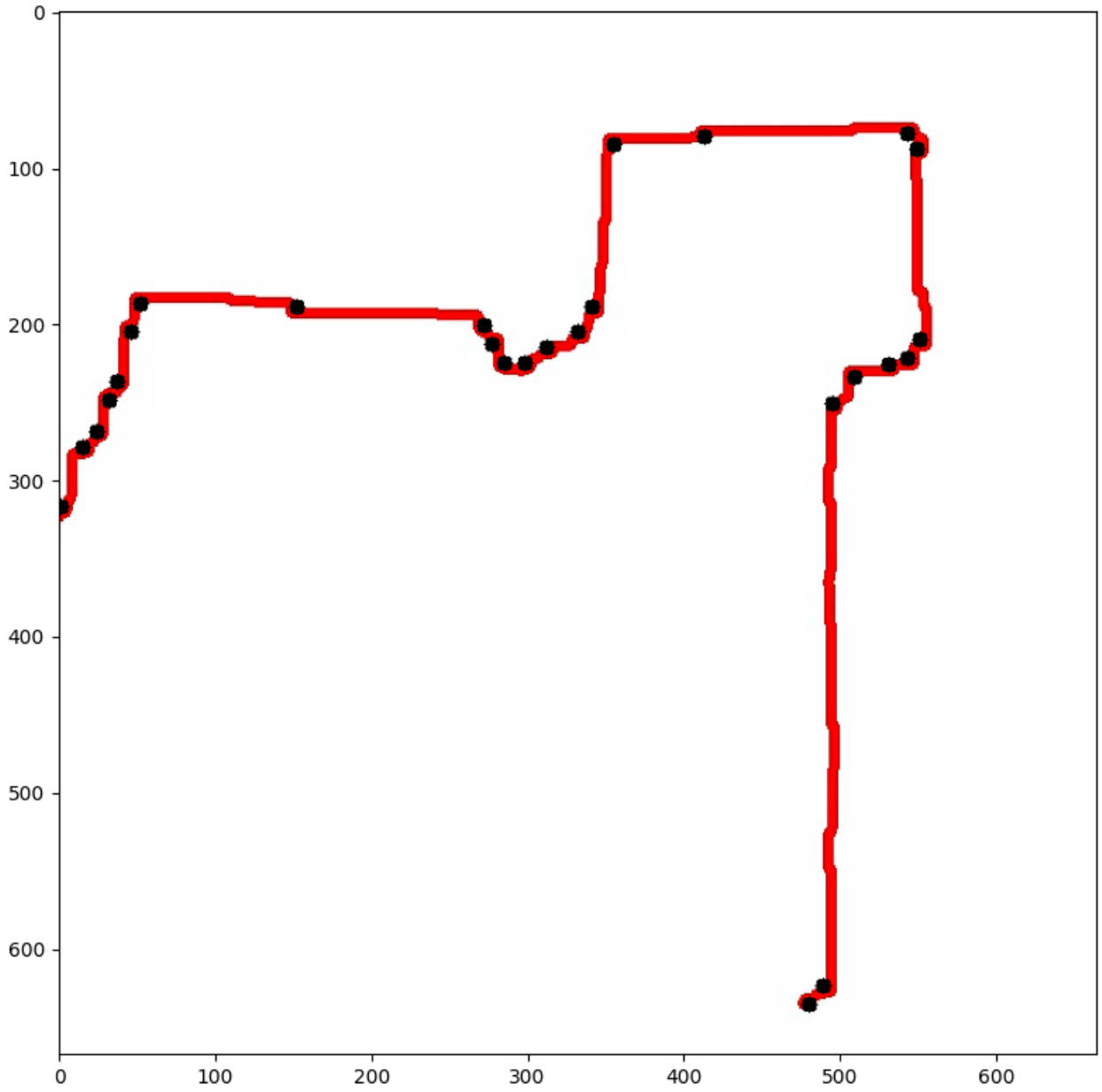


FIGURE 7.2: Python Turtle Simulation

The obtained edge points are further used to generate the instruction set by following the methodology mentioned above.

After generating the instruction set, it is implemented in hardware.

The hardware implementation of the project is displayed below:

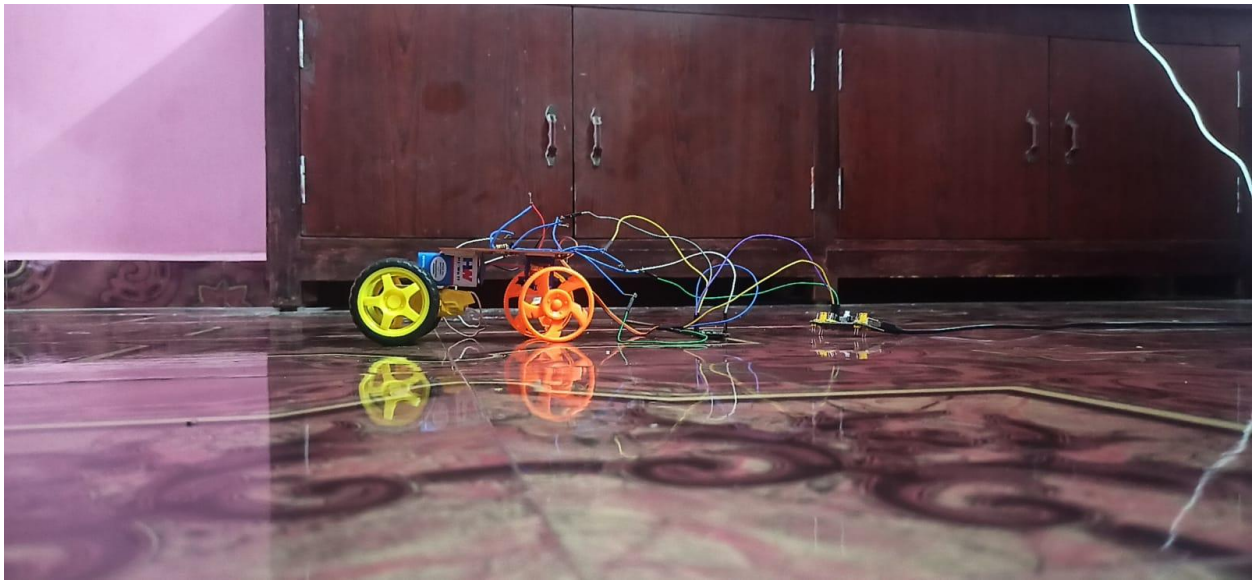


FIGURE 7.3: Hardware Implementation of ANT-BOT

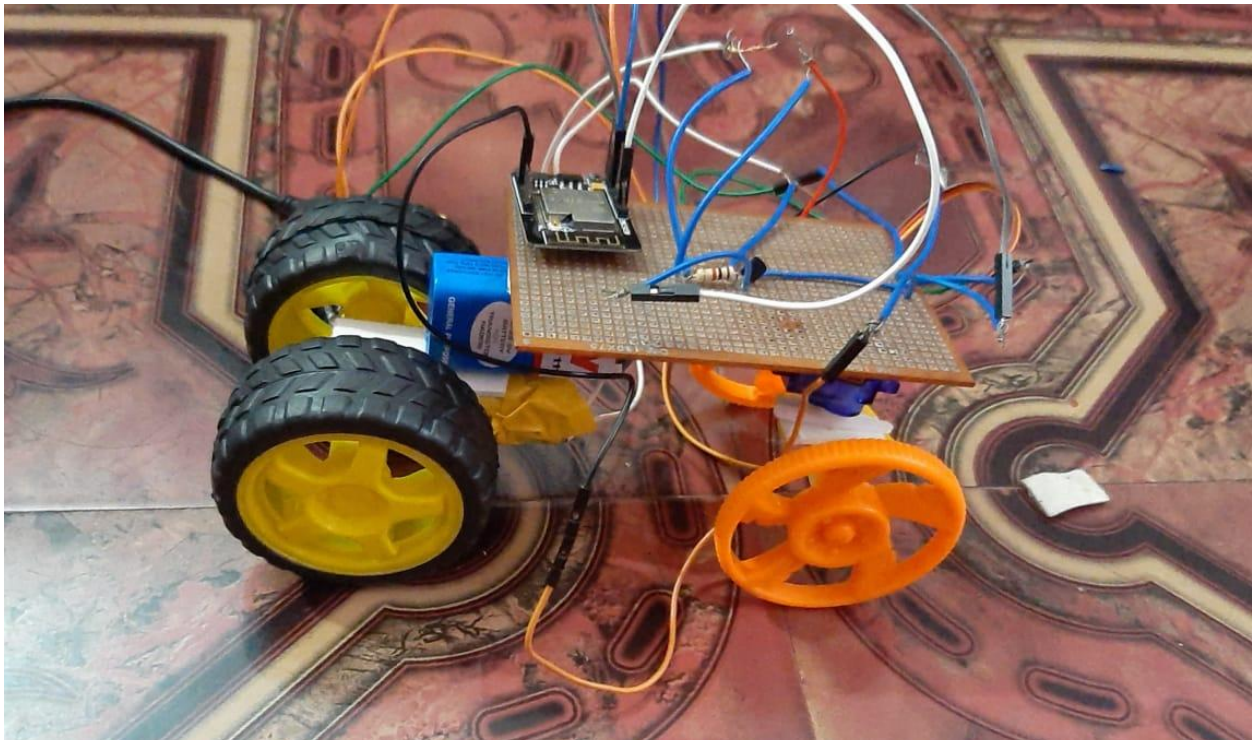


FIGURE 7.4: Hardware Implementation of ANT-BOT

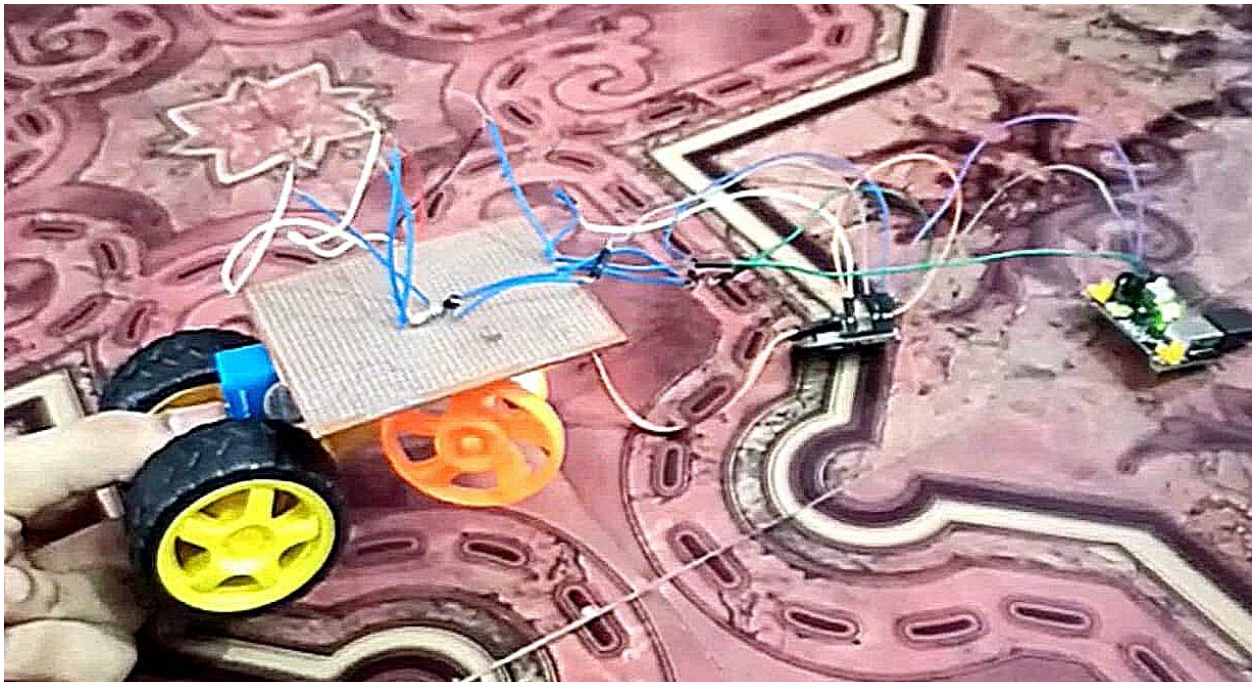


FIGURE 7.5: Hardware Implementation of ANT-BOT

8.APPENDIX

8.1.Python code:

```
#importing necessary libraries to the python file


#library for operating with the image

import cv2

#library to operate with pixel value matrix

import numpy as np

#library to display proceed image

import matplotlib.pyplot as plt


### start of Dijkstra algorithm


# class to assign parameters to be processed for each pixel value

class Vertex:

    def __init__(self,x_coord,y_coord):

        self.x=x_coord

        self.y=y_coord

        self.d=float('inf')

        self.parent_x=None

        self.parent_y=None

        self.processed=False

        self.index_in_queue=None


# function to get neighbour pixel values for a given pixel value

def get_neighbors(mat,r,c):
```

```
    shape=mat.shape
    neighbors=[]
    if r > 0 and not mat[r-1][c].processed:
         neighbors.append(mat[r-1][c])
    if r < shape[0] - 1 and not mat[r+1][c].processed:
            neighbors.append(mat[r+1][c])
    if c > 0 and not mat[r][c-1].processed:
        neighbors.append(mat[r][c-1])
    if c < shape[1] - 1 and not mat[r][c+1].processed:
            neighbors.append(mat[r][c+1])
    return neighbors


# populating the maze in the forward direction or upward direction and
returning the corresponding values
def bubble_up(queue, index):
    if index <= 0:
        return queue
    p_index=(index-1)//2
    if queue[index].d < queue[p_index].d:
            queue[index], queue[p_index]=queue[p_index], queue[index]
            queue[index].index_in_queue=index
            queue[p_index].index_in_queue=p_index
            quque = bubble_up(queue, p_index)
    return queue
```

```
# populating the maze in the downwards direction or upward direction and
returning the corresponding values
def bubble_down(queue, index):
    length=len(queue)
    lc_index=2*index+1
    rc_index=lc_index+1
    if lc_index >= length:
        return queue
    if lc_index < length and rc_index >= length:
        if queue[index].d > queue[lc_index].d:
            queue[index], queue[lc_index]=queue[lc_index], queue[index]
            queue[index].index_in_queue=index
            queue[lc_index].index_in_queue=lc_index
            queue = bubble_down(queue, lc_index)
    else:
        small = lc_index
        if queue[lc_index].d > queue[rc_index].d:
            small = rc_index
        if queue[small].d < queue[index].d:
            queue[index],queue[small]=queue[small],queue[index]
            queue[index].index_in_queue=index
            queue[small].index_in_queue=small
            queue = bubble_down(queue, small)
    return queue


#get pixel distance between two given point in the image
```

```
def get_distance(img,u,v):
    return 0.1 +
(float(img[v][0])-float(img[u][0]))**2+(float(img[v][1])-float(img[u][1]))
**2+(float(img[v][2])-float(img[u][2]))**2


#function to draw the resultant path on the input image
def drawPath(img,path, thickness=12):
    x0,y0=path[0]
    for vertex in path[1:]:
        x1,y1=vertex
        cv2.line(img,(x0,y0),(x1,y1),(255,0,0),thickness)
        x0,y0=vertex


#the main function to activate the above mentioned functions
def find_shortest_path(img,src,dst):
    pq=[]
    #src --> starting point of the input image
    #dst --> destination point of the input image


    source_x=src[0]
    source_y=src[1]
    dest_x=dst[0]
    dest_y=dst[1]
    imagerows,imagecols=img.shape[0],img.shape[1]
    matrix = np.full((imagerows, imagecols), None)
    #for loop to initialize the class vertex objects to the image
```

```
    for r in range(imagerows):
        for c in range(imagecols):
            matrix[r][c]=Vertex(c,r)
            matrix[r][c].index_in_queue=len(pq)
            pq.append(matrix[r][c])
    matrix[source_y][source_x].d=0
    pq=bubble_up(pq, matrix[source_y][source_x].index_in_queue)
    #while loop to populate the image
    while len(pq) > 0:
        u=pq[0]
        u.processed=True
        pq[0]=pq[-1]
        pq[0].index_in_queue=0
        pq.pop()
        pq=bubble_down(pq,0)
        neighbors = get_neighbors(matrix,u.y,u.x)
        for v in neighbors:
            dist=get_distance(img,(u.y,u.x),(v.y,v.x))
            if u.d + dist < v.d:
                v.d = u.d+dist
                v.parent_x=u.x
                v.parent_y=u.y
                idx=v.index_in_queue
                pq=bubble_down(pq,idx)
                pq=bubble_up(pq,idx)
```

```
    path=[]
    iter_v=matrix[dest_y][dest_x]
    path.append((dest_x,dest_y))
    #while loop to find the shortest distance to reach the destination
    while(iter_v.y!=source_y or iter_v.x!=source_x):
        path.append((iter_v.x,iter_v.y))
        iter_v=matrix[iter_v.parent_y][iter_v.parent_x]

    path.append((source_x,source_y))
    return path

#input image
img = cv2.imread(r'C:\Users\Vaikunth Guruswamy\Downloads\ant-bot
final\tarun.jpeg')
#cv2.imshow('oo',img)
img = cv2.resize(img,(700, 700))
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

############## detect corners with the goodFeaturesToTrack function.
#########
corners = cv2.goodFeaturesToTrack(gray, 27, 0.01, 10)
corners = np.int0(corners)
l=[]
for i in corners:
    #print(i.ravel())
    x, y = i.ravel()
```

```
    l.append([x,y])
    cv2.circle(img, (x, y), 5, 255, -1)


x_max=y_max=0
x_min=y_min=800
for index in l:
    if x_min>index[0]:
        x_min=index[0]
    if x_max<index[0]:
        x_max=index[0]
    if y_min>index[1]:
        y_min=index[1]
    if y_max<index[1]:
        y_max=index[1]
end_points=[[x_min,y_min],[x_min,y_max],[x_max,y_min],[x_max,y_max]]
#print(end_points)
##########################################################################
############
i = img[x_min:x_max,y_min:y_max]
cv2.imshow('original image',i)
img =i
p = find_shortest_path(img,(8,380),(496,654))
drawPath(img,p)
#print(p)
plt.figure(figsize=(7,7))
plt.imshow(img)
```

```
############################path extraction ###############


bg = np.ones(img.shape, dtype="uint8")

bg*=255


def drawPath(image,path, thickness=5):

    '''path is a list of (x,y) tuples'''

    x0,y0=path[0]

    for vertex in path[1:]:

        x1,y1=vertex

        cv2.line(image,(x0,y0),(x1,y1),(255,0,0),thickness)

        x0,y0=vertex


for i in range(len(p)-1):

    drawPath(bg, (p[i], p[i+1]))


cv2.imshow("result", bg)

cv2.waitKey(0)

cv2.destroyAllWindows()


cv2.imwrite(r"C:\Users\Vaikunth Guruswamy\Downloads\ant-bot
final\res.jpg", bg)

##############################edge detect #################



# read the image
```

```
img = cv2.imread(r'C:\Users\Vaikunth Guruswamy\Downloads\maze
solver\res.jpg')
img = cv2.resize(img,(700,700))


# convert image to grayscale image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)


# detect corners with the goodFeaturesToTrack function.
corners = cv2.goodFeaturesToTrack(gray, 27, 0.01, 10)
corners = np.int0(corners)
l=[]
for i in corners:
    #print(i.ravel())
    x, y = i.ravel()
    l.append([x,y])
    cv2.circle(img, (x, y), 5, 0, -1)


x_max=y_max=0
x_min=y_min=800
# for loop to find the edgepoints of the given maze
for index in l:
    if x_min>index[0]:
        x_min=index[0]
    if x_max<index[0]:
        x_max=index[0]
    if y_min>index[1]:
```

```
        y_min=index[1]
    if y_max<index[1]:
        y_max=index[1]
end_points=[[x_min,y_min],[x_min,y_max],[x_max,y_min],[x_max,y_max]]
#print(end_points)
i = img[18:686,20:685]


plt.imshow(i)
o=p
ind =[]
import math
res={}
for i in l:
    for j in o:
        if((i[0] in range(j[0]-5,j[0]+5)) and (i[1] in
range(j[1]-5,j[1]+5))):
            ind.append(j)
            res[o.index(j)] = j
            break



#print(res)
list_keys = list(res.keys())
list_keys.sort()
list_empty=[]
#print(list_keys)
```

```
for i in list_keys:
    list_empty.append(list(res[i]))
#print(list_empty)
#print(l)
l=list_empty
print('####################################final
output##########################################')
################################inst
angle=0
import math
list_coordinates=l
difference=0
instruction=[]
y_len=float(input("enter the length of map along y:"))
x_len=float(input("enter the length of map along x:"))
#rpm=1000
x_ratio=x_len/700
y_ratio=y_len/700
for index in range(len(list_coordinates)-2,0,-1):

y_diff=-(list_coordinates[index][1]-list_coordinates[index+1][1])*y_ratio

x_diff=(list_coordinates[index][0]-list_coordinates[index+1][0])*x_ratio
    if x_diff!=0:
        slope=(math.atan(y_diff/x_diff)*180)/math.pi
    else:
        slope=90
```

```
    difference=angle-slope
    angle=slope
    distance=math.sqrt(y_diff**2+x_diff**2)
    if difference<0:
        instruction.append(["l",round(abs(difference)),distance])
    elif difference>0:
        instruction.append(["r", round(abs(difference)), distance])
    else:
        instruction.append(["s", round(abs(difference)), distance])


print('\nfinal instruction\n')
for i in instruction:
    print(i)
turning=[]
angles=[]
length=[]
for index in instruction:
    turning.append(index[0])
    angles.append(index[1])
    length.append(index[2])


for index in range(0,len(angles)):
    rev_index=len(list_coordinates)-1-index
    if angles[index]>90:
        angles.insert(index+1,90)
        turning.insert(index+1,turning[index])
```

```
        angles[index]=90
        length.insert(index+1,math.cos(length[index]))
        length[index]=math.sin(length[index])
print('\ndirection to be turned\n',turning)
print('\nangle to be turned\n',angles)
print('\ndistance to be covered after each instruction\n',length)


plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Provided the input image and the dimensions of the required path the above program will generate the instruction set as mentioned below:

##########final output ##########

enter the length of map along y:30

enter the length of map along x:25

final instruction

['l', 62, 0.7731462358705604]

['l', 10, 1.662675493269317]

['r', 14, 0.551250896616421]

['l', 24, 0.8638133301748447]

['r', 28, 0.6261316285946036]

['l', 27, 1.383005452495634]

['r', 20, 0.8657013200349206]

['r', 70, 3.5970509462897633]

['r', 1, 4.250954274618578]

['r', 60, 0.557142857142857]

['l', 12, 0.5705797266726149]

['l', 89, 0.5527759261127386]

['l', 2, 0.6062598621120842]

['r', 1, 0.8558323314714376]

['l', 36, 0.8226388599364554]

['l', 13, 4.489221786111481]

['r', 77, 2.082482819587607]

['r', 8, 4.646020900090918]

['l', 92, 0.5571428571428572]

['r', 0, 5.228693402168227]

['r', 31, 0.551250896616421]

['r', 37, 0.4615855548972432]

['l', 4, 0.875284210402308]

['l', 36, 0.8277397303648591]

['l', 27, 15.988945479643753]

direction to be turned

['l', 'l', 'r', 'l', 'r', 'l', 'r', 'r', 'r', 'r', 'l', 'l', 'l', 'r', 'l', 'l', 'r', 'r', 'l', 'l', 'r', 'r', 'r', 'l', 'l', 'l']

angle to be turned

[62, 10, 14, 24, 28, 27, 20, 70, 1, 60, 12, 89, 2, 1, 36, 13, 77, 8, 90, 90, 0, 31, 37, 4, 36, 27]

distance to be covered after each instruction

[0.7731462358705604, 1.662675493269317, 0.551250896616421, 0.8638133301748447, 0.6261316285946036, 1.383005452495634, 0.8657013200349206, 3.5970509462897633, 4.250954274618578, 0.557142857142857, 0.5705797266726149, 0.5527759261127386, 0.6062598621120842, 0.8558323314714376, 0.8226388599364554, 4.489221786111481, 2.082482819587607, 4.646020900090918, 0.5287633042203256, 0.8487693256179817, 5.228693402168227, 0.551250896616421, 0.4615855548972432, 0.875284210402308, 0.8277397303648591, 15.988945479643753]

8.2.Arduino code:

```
#include<ESP32Servo.h>//ESP32Servo header file is included


Servo myservo;//Servo object is created

int pos = 0; //Variable to store position of servo motor

const int servo= 12;//Variable to store pin at which servo motor is connected to ESP32 camera module

int i;//Iteration variable for executing the instructions given

const int motor=2;//Variable to store pin at which motor is connected to ESP32 camera module

//Instructions for direction in which the vehicle should move and when it should stop after completing the task is in obtained from the python code
```

```
char instructions[]={'l', 'l', 'r', 'l', 'r', 'l', 'r', 'r', 'r', 'r',
'l', 'l', 'l', 'r', 'l', 'l', 'r', 'r', 'l', 'l', 'r', 'r', 'r', 'l', 'l',
'l','X'};

//Angle needed to be turned obtained from the python code

int angle[]={62, 10, 14, 24, 28, 27, 20, 70, 1, 60, 12, 89, 2, 1, 36, 13,
77, 8, 90, 90, 0, 31, 37, 4, 36, 27};

//Distance to be covered after each instruction in cm

float distance[]=
{0.77,1.66,0.55,0.86,0.62,1.38,0.86,3.59,4.25,0.55,0.57,0.55,0.60,0.85,
0.82,4.48,2.08,4.64,0.52,0.84,5.22,0.55,0.46,0.87,0.82,15.98};



// Function to be used when the vehicle is turning right

void right()

{

  digitalWrite(motor,LOW);//motor is turned LOW

  int b= angle[i]+90;// As 90 degrees is considered straight, angle needed
to be turned towards right will be 90 plus the angle given

  for( pos=90; pos<b;pos++)// goes from 90 degrees to b degrees which is
more than 90 in incremental steps of 1 degree

  {

  myservo.write(pos);//Position of servo motor is written to pos whose
value is upgraded in every iteration

  Serial.println(pos);//Current position of servo motor is displayed on
serial monitor followed by a new line

  delay(30);//Delay of 30 ms is given for servo to reach the position
given and maintain it

  }

}
```

```
//Function to be used when the vehicle is turning left
void left()
 {
  digitalWrite(motor,LOW);//motor is turned LOW
  int a= 90-angle[i];// As 90 degrees is considered straight, angle needed to be turned towards right will be 90 plus the angle given
  for(pos=90;pos>=a;pos--)// goes from 90 degrees to a degrees which is less than 90 degrees in decremental steps of 1 degree
  {
  myservo.write(pos);//Position of servo motor is written to pos whose value is upgraded in every iteration
  Serial.println(pos); //Current position of servo motor is displayed on serial monitor followed by a new line
  delay(30);//Delay of 30 ms is given for servo to reach the position given and maintain it
  }
}



//Function to be used when the vehicle is moving forward
void straight(float d1)
{
  digitalWrite(motor,HIGH);//motor is turned HIGH
  myservo.write(90);//Position of servo motor is written to 90 degrees which is the straight position
```

```
  long t= d1/0.03401;//Time for which vehicle must go straight is calculated by t= d1/(speed of motor(1.667rpms)*distance covered in 1 rotation by wheel(20.41cm))
  delay(t);//Delay of t ms is given to reach and maintain the position of servo and state of motor for the distance given
}

//Setup is used for initialization which is executed only once
void setup()//
{
  // allocation of all the timers
  ESP32PWM::allocateTimer(0);
  ESP32PWM::allocateTimer(1);
  ESP32PWM::allocateTimer(2);
  ESP32PWM::allocateTimer(3);
  pinMode(motor,OUTPUT);//Assigning motor pin to OUTPUT
  myservo.setPeriodHertz(50);//Servo used has frequency of 50 Hz, so the object myservo is set to the same value
  myservo.attach(servo, 544, 2400); //servo pin is attached to myservo with minimum and maximum pulse width of 544us and 2400us corresponding to 0 degree and 180 degrees respectively
  Serial.begin(115200);//Serial monitor is initialised at a baud rate of 115200 bps
}

//Loop will be executed continuously
void loop()
{
```

```
Serial.println("Started");//Print in the serial monitor that program has "Started" followed by a new line
for (i =0; i<27; i++)//Loop to execute the instructions given
{
    if(instructions[i]=='r')//If the instruction is 'r' following statements are executed
   {
   right();//right function is called
   Serial.println("Right");//Print in the serial monitor that instruction has been given to turn "Right" followed by a new line
   straight(distance[i]);//straight function is called which has distance as parameter
   Serial.println("Forward");//Print in the serial monitor that instruction has been given to go "Forward" followed by a new line
   }
  if(instructions[i]=='l')//If the instruction is 'l' following statements are executed
   {
   left();//left function is called
   Serial.println("Left");//Print in the serial monitor that instruction has been given to turn "Left" followed by a new line
   straight(distance[i]);//straight function is called which has distance as parameter
   Serial.println("Forward");//Print in the serial monitor that instruction has been given to go "Forward" followed by a new line
   }
   if(instructions[i]=='X')//If the instruction is 'X' following statements are executed
   {
```

```
  Serial.println("End");//Print in the serial monitor that instruction has
been given to "End" the execution followed by a new line
   digitalWrite(motor,LOW);//motor is turned LOW
   myservo.write(90);//Position of servo motor is written to 90 degrees
which is the straight position
   delay(100);//Delay of 100 ms is given to reach and maintain the
position of servo and state of motor
   while(1)//Infinite loop to make it a single execution program
   {
    Serial.println("Execution over");//Print in the serial monitor that
execution is over
   }
   }
}
}
```

9.CONCLUSION

In conclusion, the path detection algorithm has successfully been implemented in the rover and the objectives of the project have been achieved.

It finds out the shortest path in a given environment by using Dijkstra’s algorithm. Several tests have been done to determine the proficiency of the rover.

The comprehensive planning and design of the ANT-BOT at an earlier stage made the construction process easy. Even using low-cost material, the rover is still able to perform well. Dijkstra's algorithm is implemented and shows a good result in the pathfinding case.

Besides, this project provides a good platform for academic learning. We can apply some of the theories learned during class and practice in this project. During the practical time, we are able to understand the actual condition to carry out the project and try to solve all the problems faced. Hence, critical thinking and problem solving are the major issues for us to bring out this Mini Project successfully.

10.FUTURE WORK

- The map was web scrapped to find a real-world path to travel to the destination as per user needs.

  As sample input starting point as “Madras institute of technology” and destination point as “College of engineering, Guindy” was given and the result so obtained below:

```
Madras Institute of Technology, Chennai

College of engineering, Guindy

connected

Gas

from Madras Institute of Technology, Anna University, MIT Rd, Radha
Nagar, Chromepet, Chennai, Tamil Nadu 600044

to College of Engineering, Guindy, 12, Sardar Patel Rd, Anna
University, Guindy, Chennai, Tamil Nadu 600025

24 min (16.3 km)

via Chennai - Theni Hwy/Chennai - Trichy Hwy

Fastest route, the usual traffic

Madras Institute of Technology, Anna University

MIT Rd, Radha Nagar, Chromepet, Chennai, Tamil Nadu 600044

Take Chitlapakkam Main Rd and MIT Flyover to Chennai - Nagapattinam
Hwy/Chennai - Theni Hwy/Chennai - Trichy Hwy/Grand Southern Trunk Rd
in New Colony

4 min (1.5 km)

Head southwest on MIT Rd toward Hastinapuram Main Rd
```

```
230 m

Turn right at Hastinapuram Main Rd

270 m

Continue straight

97 m

Turn right onto Dr Rajendra Prasad Rd

9 m

Turn right onto Chitlapakkam Main Rd

450 m

At MIT Roundabout, take the 2nd exit onto MIT Flyover

400 m

Follow Chennai - Theni Hwy/Chennai - Trichy Hwy to Guindy National
Park

24 min (14.1 km)

Use any lane to turn slightly left onto Chennai - Nagapattinam
Hwy/Chennai - Theni Hwy/Chennai - Trichy Hwy/Grand Southern Trunk Rd

Pass by Citibank ATM (on the left in 700 m)

2.0 km

Slight right onto Chennai - Theni Hwy/Chennai - Trichy
Hwy/Pallavaram Flyover

Continue to follow Chennai - Theni Hwy/Chennai - Trichy Hwy

1.8 km
```

```
Keep right to continue on Airport Flyover/Chennai - Nagapattinam
Hwy/Chennai - Theni Hwy/Chennai - Trichy Hwy/Grand Southern Trunk Rd

Continue to follow Chennai - Nagapattinam Hwy/Chennai - Theni
Hwy/Chennai - Trichy Hwy/Grand Southern Trunk Rd

Pass by the petrol pump (on the right in 3.8 km)

6.2 km

Use the left 2 lanes to turn slightly left to merge with Chennai -
Nagapattinam Hwy/Chennai - Trichy Hwy

2.2 km

Continue straight onto Anna Salai/Chennai - Nagapattinam Hwy/Chennai
- Trichy Hwy/Mount Rd

35 m

Keep left to stay on Anna Salai/Chennai - Nagapattinam Hwy/Chennai -
Trichy Hwy/Mount Rd

Continue to follow Chennai - Trichy Hwy

Pass by Metropolitan Magistrate Courts (on the left in 850 m)

1.2 km

Use the left lane to continue on Taluk Office Rd

95 m

Taluk Office Rd turns left and becomes Sardar Patel Rd

500 m

Take Guest House Road to your destination

3 min (700 m)

Turn left toward Guest House Road
```

```
26 m

Sharp right onto Guest House Road

350 m

Turn left

62 m

Turn right

110 m

Turn left

53 m

Turn right

Destination will be on the left

94 m

College of Engineering, Guindy

12, Sardar Patel Rd, Anna University, Guindy, Chennai, Tamil Nadu
600025

These directions are for planning purposes only. You may find that
construction projects, traffic, weather, or other events may cause
conditions to differ from the map results, and you should plan your
route accordingly. You must obey all signs or notices regarding your
route.

Sign in

Live traffic

FastSlow

Layers
```

```
Map data ©2021 IndiaTermsPrivacy

Send feedback

2 km
```

- The wifi module will be implemented while manufacturing on a large scale
- Lane detection and navigation was also referred.
    - Lane Detection with Steer Assist and Lane Departure Monitoring(GitHub:https://github.com/vaikunth-coder27/lane-detection-with-steer-and-departure)

11.REFERENCE

1. Alfian Ma'arif, Aninditya Anggari Nuryono, Iswanto, "Vision-Based Line Following Robot in Webots", IEEE2021.

2. Malkit Singh, Rajnish Kumar, Vaibhav Giradkar, Pallavi Bhole, Minu Kumari, "ARTIFICIALLY INTELLIGENT MAZE SOLVER ROBOT", International Research Journal of Engineering and Technology (IRJET) Vol. 03, pp. 325-329, Apr-2016

3. Mohammad O.A. Aqel, Ahmed Issa, Mohammed Khdair, Majde ElHabbash, Mohammed AbuBaker, Mohammed Massoud, "Intelligent Maze Solving Robot Based on Image Processing and Graph Theory Algorithms ", 2017 IEEE

4. Xinglin Yu , Yuhu Wu , Xi-Ming Sun, "A Navigation Scheme for a Random Maze using Reinforcement Learning with Quadrotor Vision".