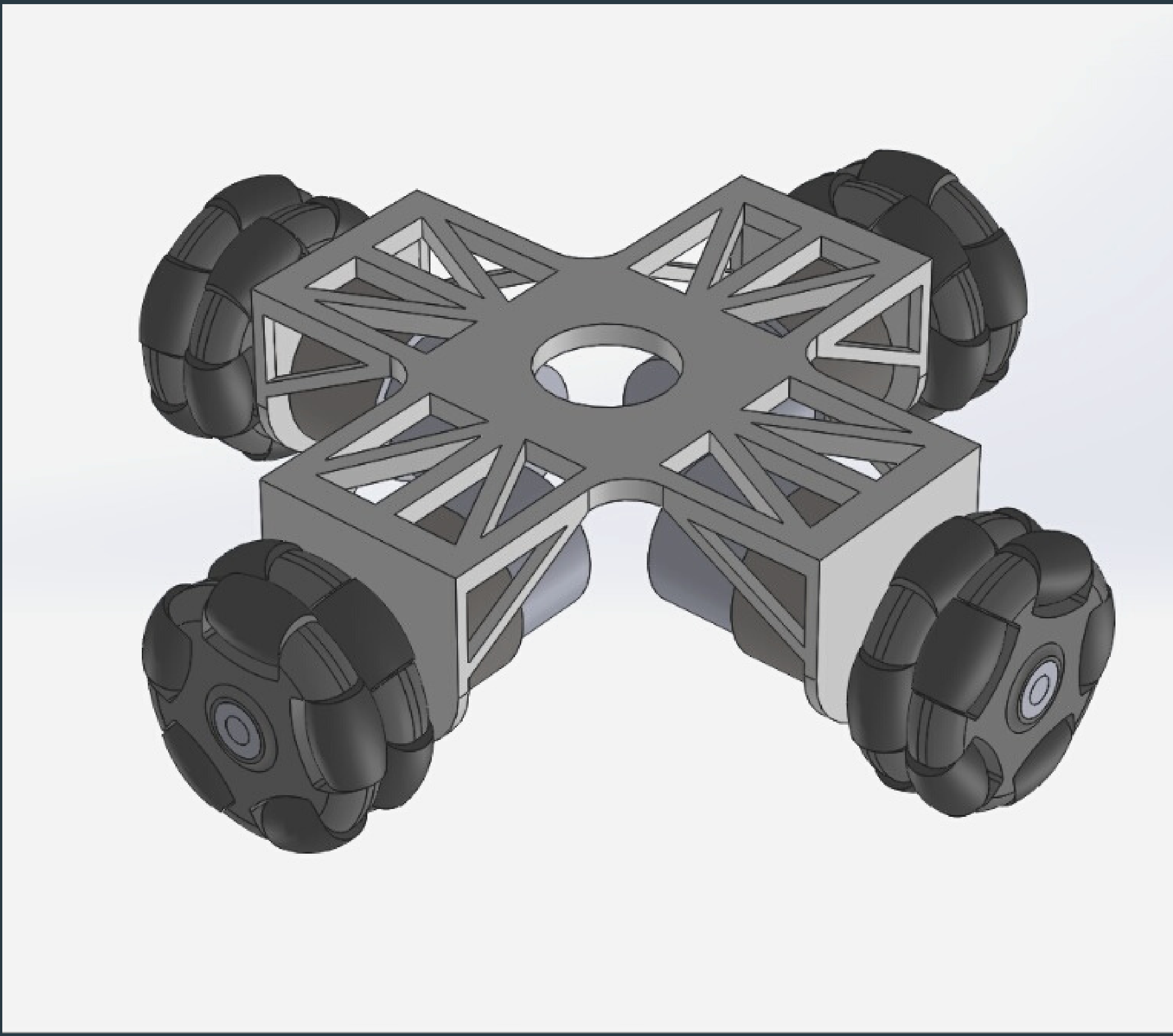
Maze solving Robot

Srishti 2020

Project Report



INDEX

Team Members:

1. Akshat Gupta
2. Sarthak Surolia
3. Shanmathi Selvaraj
4. Siddharth Gehlot
5. Sneha Jain
6. Suhani Jain
7. Susmit Walve
8. Yash Asati

Mentors:

1. Sanjeev Krishnan R.
2. Shubham Malviya

Abstract

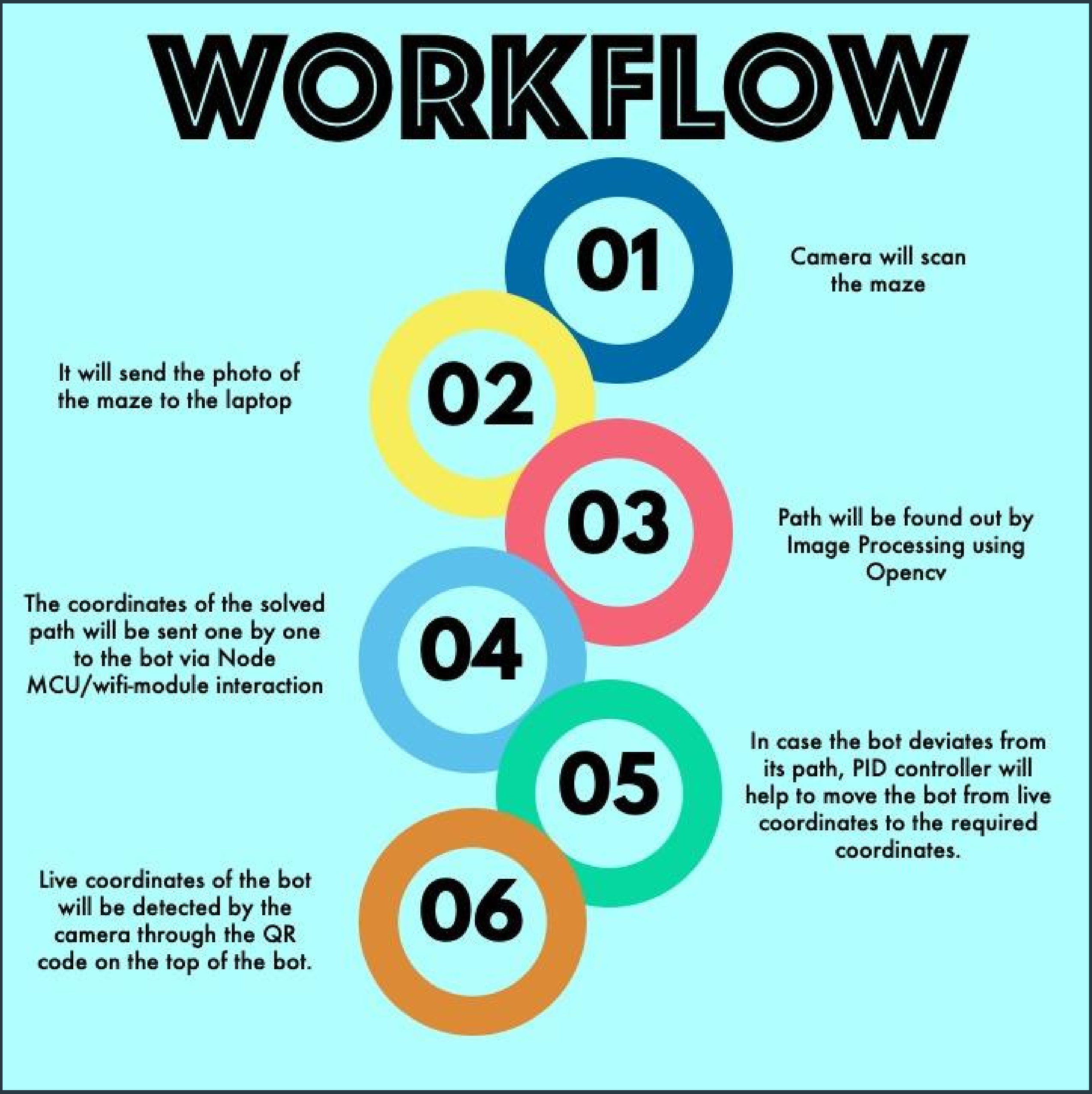
The Maze Solving Bot is a wireless bot. It is capable of solving any 2-D maze with the help of live video feed parsed by a python script. The speciality of this bot is that it can adapt itself to any maze and not one-kind in particular. Having a Four-wheel Omni drive, It aims to reach the ending point in the shortest time. If there are more than two ways out, then it works to solve the shortest path.

Motivation

Maze solving bot is a simple and straightforward case where we can learn and experiment about path planning. The standard method of maze solving takes much time due to its trial and error methods. With the help of shortest path detection and robot live coordinates mapping, we looked forward to improvise the olden methods.

A maze in the real-world can represent roads and traffic. All of us want to know the best way to go to work, school etc. In the journey, we experience all sorts of obstructions like traffic, diversions, intersections and dead-ends. This project gives us an insight into how to solve this problem. It helps us find the best and the shortest path to avoid these barricades as much as possible

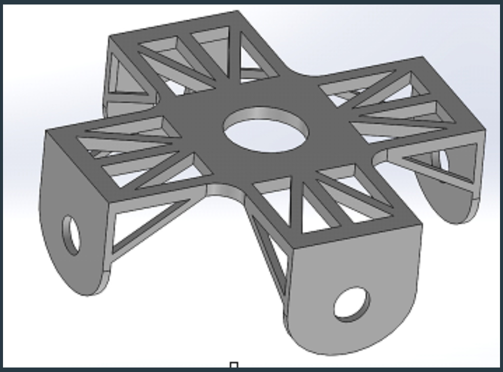
Workflow:


WORKFLOW
01
Camera will scan the maze
02
It will send the photo of the maze to the laptop
03
Path will be found out by Image Processing using Opencv
04
The coordinates of the solved path will be sent one by one to the bot via Node MCU/wifi-module interaction
05
In case the bot deviates from its path, PID controller will help to move the bot from live coordinates to the required coordinates.
06
Live coordinates of the bot will be detected by the camera through the QR code on the top of the bot.

Mechanical Aspects:

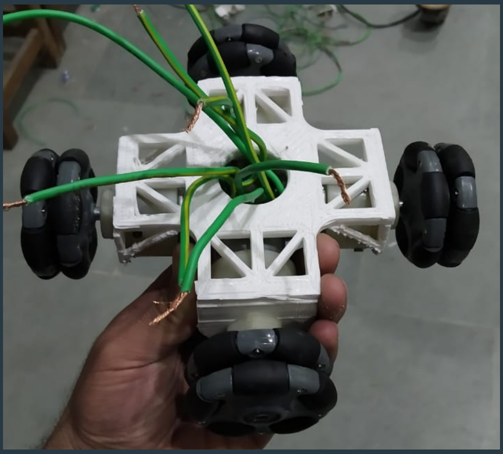
1. Chassis:

The main body of the robot is 3D printed using PLA(Polylactic Acid). It has the shape of a 'plus' sign. The dimensions of the chassis are about 14.5cm x 14.5cm x 5 cm. There are lightening holes present wherever possible to keep the weight of the chassis low.



2. Drive type

The maze solving bot uses a 4 wheeled Omni type of drive. 2 sets of Omni wheels are perpendicular to the other set. The reason for this design is that the maze only consists of perpendicular junctions. Hence the working of the robot is highly simplified if we use 4 WD Omni. 4 WD Omni in general has some limitations. It only takes 3 points to define a plane. Hence in most practical cases, the 4th wheel lies out of the plane. But in this project, we could dodge this limitation by 3D printing the chassis thus making it highly accurate.

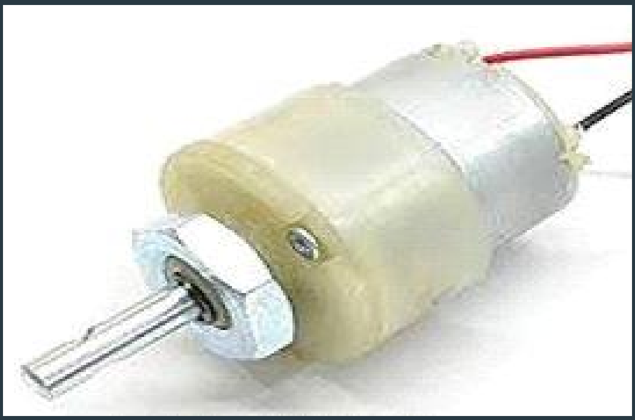


Electronic Aspects:

1. Actuator:

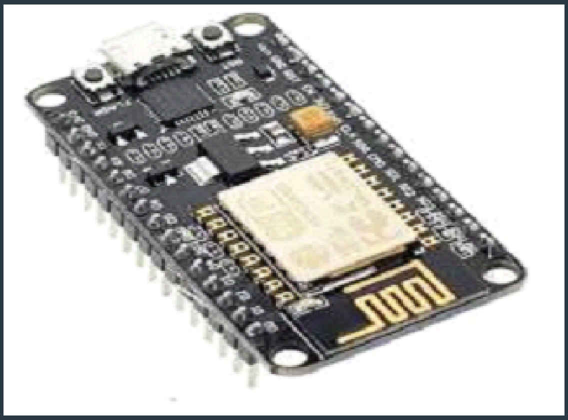
The robot uses four 12 volt- 100 rpm motors. These motors are relatively light, weighing a few hundred grams and overall have a small and compact size. Having a small body size is imperative for the robot to transverse through the narrow lanes of the maze.

There were some issues regarding the four motors not delivering equal amount of torque, resulting in the robot changing its orientation. This is where the PID control was employed to ensure the robot transverses the maze immaculately. Since the robot is designed to move on flat surfaces, none of the four wheels would lose contact with the surface, ensuring that the robot would not rotate about one of its wheels.

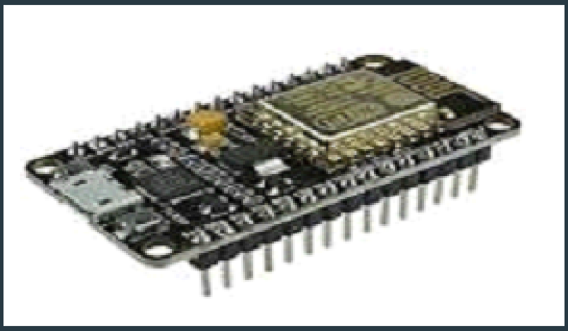


2. Microcontroller

A Node Microcontroller Unit a.k.a Node MCU was used as a microcontroller. It facilitates an open-source software for hardware development. It is small, lightweight, extremely efficient and is built on a “System on Chip” called ESP8266, which is relatively less costly.



One of the biggest benefits of the NodeMCU is that it allows communication over a wifi-network, hence there is no need to keep it pluged into the device on which the source code is, the instructions can be easily sent over a wifi-network, if neccesary, parsed with the help of a python script. The clock speed of the NodeMCU is close to 80 MHz, ensuring significant speed and preventing too much delay between the parsing of instructions and operation of the actuators. Also,this allows the execution of PID control equation to a significantly higher degree of precision.completing both the tasks of recieving coordinates and controlling motors is done by nodemcu. Coordinates are detected by scanning the QR code



3. PID Control

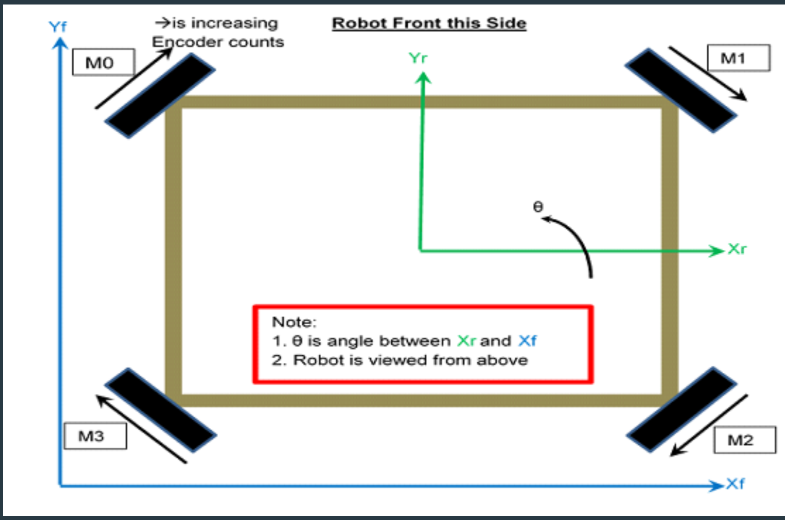
The PID control allows monitoring the progression of errors in the movements of the robot and making minute changes in the operation of the actuators to ensure a greater degree of precision in the robot's movements.

In our case the robot had to move parallel to the co-ordinate axis only on a flat surface with minimal skidding of the wheels. This simplified our job many-fold, since now we focus only on the "proportional errors", without considering the derivative or the integral terms. Also, the orientation of the robot had to be taken into account, since it was always possible that one of the parallel sets of wheels moves faster than its counterpart, hence resulting in a rotation of the robot's chassis to rotate and become disoriented.

We made use of a python script to determine the robot's current co-ordinates as well as its orientation with respect to the co-ordinate axis. This allowed us to operate the actuators to bring the robot to its correct orientation and position.To change the orientation, all four wheels rotate in clockwise or counter-clockwise manner , until the robot gets as close to the desired orientation as possible.

For handling errors in the position, we handle movement along the x and y-axis separately (the robot always moves along a path parallel to the co-ordinate axis).The speed of the robot can be altered in accordance with the equation v=kp*(x-x') where- x-current position of the robotx'-desired position of the robotkp- error constant, determined through trials.

As one can note, the robot's speed gets reduced as we get closer to the desired position, the rate of decrease depending significantly on the constant kp. Similar arguments can be applied to the movements along the y-axis. The constant kp depends on the material of the wheels, the traction available off the surface, the amount of skidding and some inevitable manufacturing defects. The value remains somewhere between 0.03- 0.5 but can vary greatly, thus requiring a number of trials to determine.


Yf
→is increasing Encoder counts
Robot Front this Side
M0
M1
Yr
θ
Xr
Note:
1. θ is angle between Xr and Xf
2. Robot is viewed from above
M3
M2
Xf

Cost Structure:

COMPONENTS	COST(INR)
2X MOTOR DRIVER L2899	250
4X DC () RPM MOTORS	700
NODEMCU ESP8266	350
12V BATTERY	1000
3D PRINTED CHASSIS	800
4X OMNI WHEELS	2200
TOTAL	5300

Applications:

- Our maze solving bot is a four-wheel chassis. It is planer even in uneven terrains. It can be used as a food and other things delivery system for high altitude army posts. Usually, a drone will not get a signal in such high altitudes. Moreover, this bot uses image processing before starting to solve the maze, which comes in handy for areas having low signals catching capacities.

- Future of driving is self-driving cars or fully automated cars. While parking in a parking area. They are a sophisticated version of our maze solving bot. Wherein obstacles are other cars and pillars. There can be a series of cameras on the wall which takes a whole 2D image of the parking area and uses image processing to solve the maze (or parking area) and tell the best possible coordinate to know where to park the car. So these can be seen as daily applications of our bot.

Limitations:

- The Camera that we are using for image processing is attached to our laptop with wires, so that might become a bit hectic to handle.

- There has to be a minimum height above which Camera should be mounted, so for our maze solving bot show we need a spacious room, and we have to deal with a bunch of extra wires.

Further Improvements

- Sensors can be added to the bot to avoid further immediate obstacles.

Reference Links:

1.QR code Scanner
https://www.learnopencv.com/opencv-qr-code-scanner-c-and-python/

2. Bot Traversal
https://youtu.be/0u78hx-66Xk

3. DIC-Terrace_farming_documentation
https://github.com/marsiitr/DIC_Terrace_Farming

4. controlling Arduino with Python based API
https://www.instructables.com/id/Controlling-Arduino-with-python-based-web-API-No-p/

5. Solidworks
https://www.solidworks.com/