Precision farming using an autonomous vehicle

Implementation by Group A1

Jasmeet Singh Matta
Electronic Engineering Department
Hochschule of Hamm-Lippstadt
Lippstadt, Germany
jasmeet-singh-ajit-singh.matta@stud.hshl.de

Jaouaher Belgacem
Electronic Engineering Department
Hochschule of Hamm-Lippstadt
Lippstadt, Germany
jaouaher.belgacem@stud.hshl.de

Arsany Girgis
Electronic Engineering Department
Hochschule of Hamm-Lippstadt
Lippstadt, Germany
arsany-samir-kamel.girgis@stud.hshl.de

***Abstract*—This paper presents an implementation of a fully autonomous vehicle for precision farming management. The goal of this project is to design a vehicle that collects objects by navigating autonomously on the testing field. The hardware part of the vehicle is designed with respect to the dedicated mission and to the available components. This project has been realized in three steps, the first part is the software modelling, the second is the hardware modelling by designing the vehicle chassis using Solid Works, and the final part is code implementation.**

I. Introduction (Jaouaher Belgacem)

As the world population is increasing, it is estimated in 28 years from now the need of the food be more than 70% of today's world need. Regarding the limited resources that we have, we to use technology to overcome the challenges of feeding more than 9.6 billion human beings [1]. The automatization of the farming and agriculture industries is a must to survive and avoid world famine. Nowadays, Artificial intelligence is the focus of the recent studies, and it is involved mainly in all industries it enables machine to recognize objects, make decisions, and more. The autonomous vehicles are one of the AI fields that has exponentially grown in the last years. They have so many benefits as they need less or no human interference to execute a task. The idea behind the autonomous vehicle that we are developing in this project is dedicated to collecting object such as bales of straw from farms by detecting them and then, drop these bales at a specific position.

II. Software Modelling

A. Use case (Jaouaher Belgacem)

The use case diagram is used to give an overview of the different functionalities of the autonomous vehicle. Our system should be able to detect the map which is the testing field as it includes vertical lines and horizontal lines. The lines can be distinguished through the colors. The second main use case of this vehicle is to move which means moving forward, backward or turning. The vehicle is required to make a 90 degree turn to switch directions. It can turn slightly to right or to left or steering on the line to keep moving on track. Finally, our autonomous car can stop once it reaches the destination, or if it detects an object.

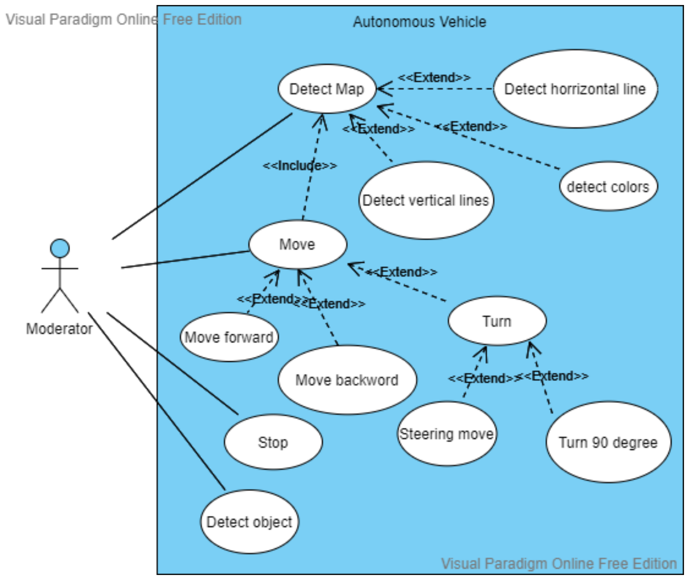


Fig. 1. Use Case Diagram

B. State Machine (Jaouaher Belgacem)

The state machine diagram is used to describe the abstract mission of the system which is moving from a starting point to a destination point through following the lines and defines its position on the map .

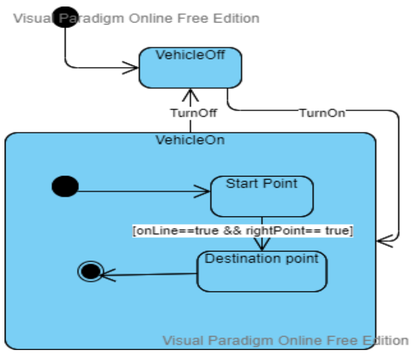

Visual Paradigm Online Free Edition
VehicleOff
TurnOff
TurnOn
VehicleOn
Start Point
[onLine==true && rightPoint== true]
Destination point
Visual Paradigm Online Free Edition

Fig. 2. State Machine Diagram

C. Sequence Diagram (Arsany Girgis)

Our sequence diagram describes the interaction between the entities of our car. First needs to start the system, where the Autonomous system (Arduino) will send a signal to get the readings of our 2 infrared sensors to send the needed directions for the motors in each specific scenario as discussed before. Where it is divided into 3 cases where the first case both sensors are on, the other is where both sensors are off and the last case is when only one sensor is on which means there is a direction adjustment.

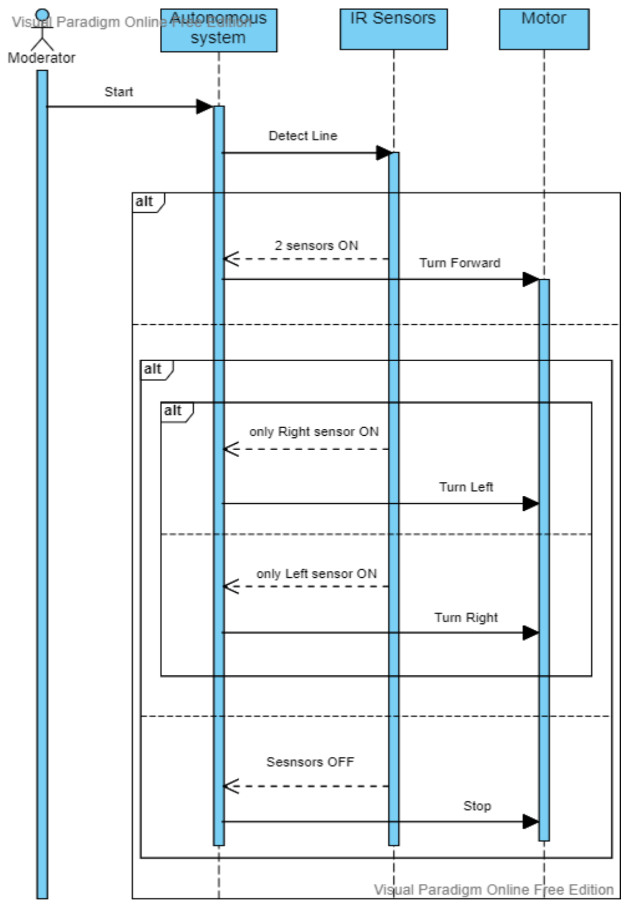


Fig. 3. Line Follower Sequence Diagram

D. Activity Diagram (Arsany Girgis)

The activity diagram describes the behavior of the autonomous system in all the cases to make the vehicle keep moving only on the colored line. Where the sensor detects the black so our logic is that if both sensors are detecting this means we are exactly on the line while if only one is detecting and the other is off, this means we need to adjust the position of our vehicles to the direction where the sensor is not detecting the black ground. If none of these cases is satisfied then this means that the car is passing a colored line

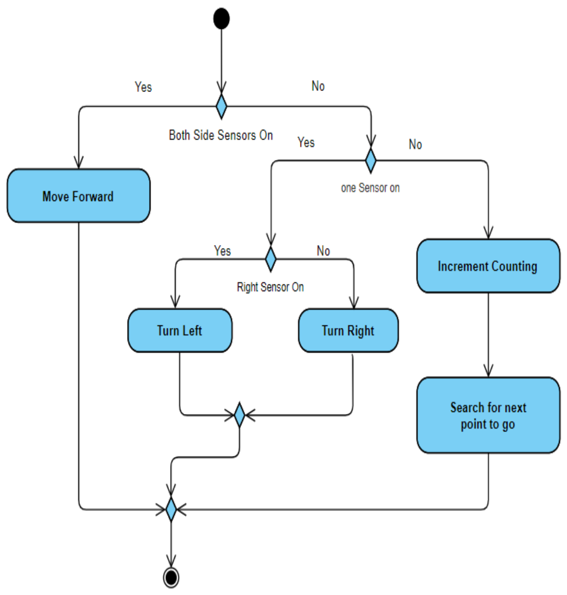


Fig. 4. Line Follower Activity Diagram

E. Context Diagram (Jaouaher Belgacem)

The context diagram describes all the external entities that is related to the behavior of the system. It highlights the main actors, the environment, and elements of system. We have one actor which acts as a stakeholder. The moderator is the main user of the system. The load represents the objects that the autonomous vehicle collects. This diagram includes environmental variables which are the light, that could affect the function of detecting colors because of the light reflection on the testing field. The last external element is the map, where the autonomous vehicle is supposed to navigate

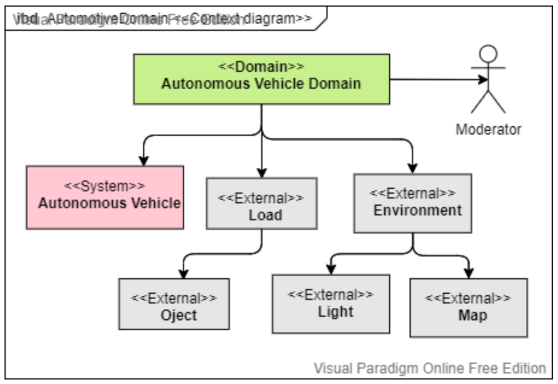

ibd AutomotiveDomain <<Context diagram>>
<<Domain>> Autonomous Vehicle Domain
Moderator
<<System>> Autonomous Vehicle
<<External>> Load
<<External>> Environment
<<External>> Oject
<<External>> Light
<<External>> Map
Visual Paradigm Online Free Edition

Fig. 5. Context Diagram

F. Parametric Diagram (Jasmeet Singh Matta)

Parametric diagrams are specialized Internal Block diagrams that help the modeler, to combine behavior and structure models. Parametric model also tells about the constraint in the environment in the form of constraint block. This constraint block can be used to create another constraint as we can see in Fig. 6 we use the weight constraint as value which affect the speed, whereas the weight constraint has all its input as real value.

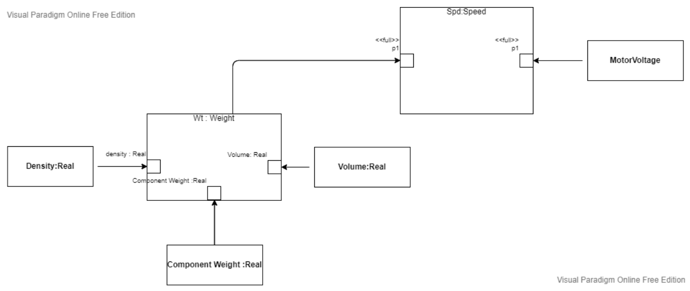


Fig. 6. parametric Diagram

G. Requirement Diagram (Jasmeet Singh Matta)

The Requirement are set of condition or capabilities that a system must satisfy or achieve. Requirement Diagram specify the requirements of the System that needs to be fulfilled. The requirement diagram contains relation such as deriveReq, refine and satisfy etc..
In the Fig. 7. Requirement are specified. We can see that System has defined weight, size and condition such as detecting line and object which it should satisfy.

III. DESIGNING

For the prototyping part, our vehicles design had couple of iterations to reach the final design. We had to make adjustment in our design regarding the required dimension and the different components that should be included.

A. Jaouaher Belgacem Design

Inspiration: The first design is inspired by the Jeep car as it is performant to be used in different terrains. As a results, a Jeep car would be a suitable vehicle for farming.

First Iteration

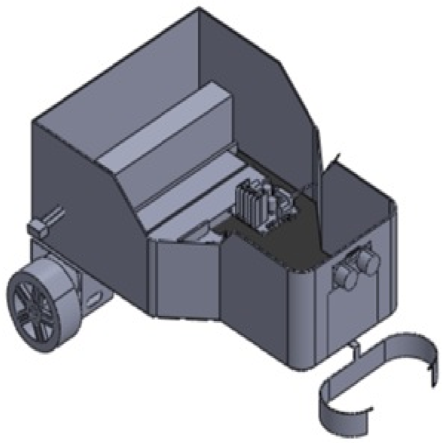


Fig. 8. First iteration of the autonomous vehicle

Second Iteration

The first iteration of vehicle design required some changes and some elements needed to be considered in the design such as the distance, the speed, and the gravity. We developed a second iteration of the prototype of the chassis. The chassis is the most important part of the design as it needs to fit all the different components. As we decided it to have a moonship vehicles, the chassis is inspired by the shape of the airship.

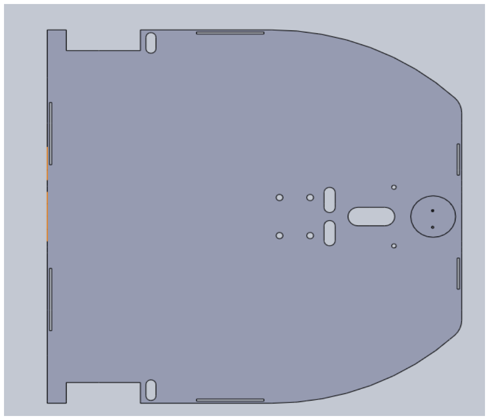


Fig. 9. Second iteration of the chassis

B. Design (Jasmeet)

Motivation: During the lab of prototyping a sample bot was shown, which consisted of finite polygons(squares). So to not go that way and to make a unique prototype which is also similar to spaceship or the moonship, it was decided to include circle. As circle have larger surface area it is easy to accomodate electronics component on it and to make design look cool and futuristic.

1) Concept: The motor is in front as it will help in 90° rotation

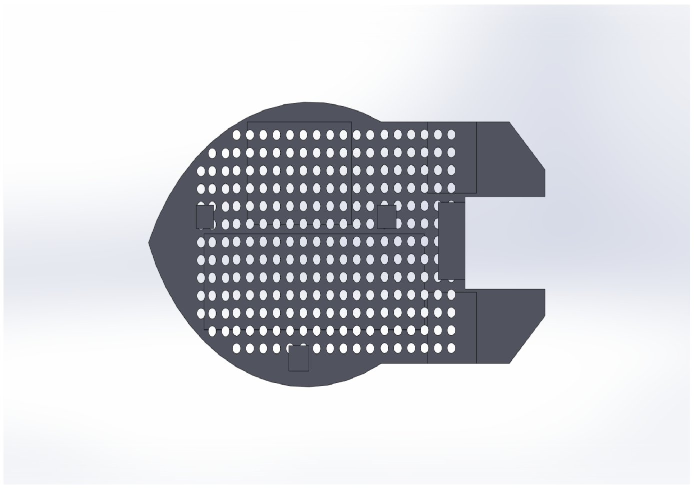


Fig. 10. First iteration of the Lower Chassis

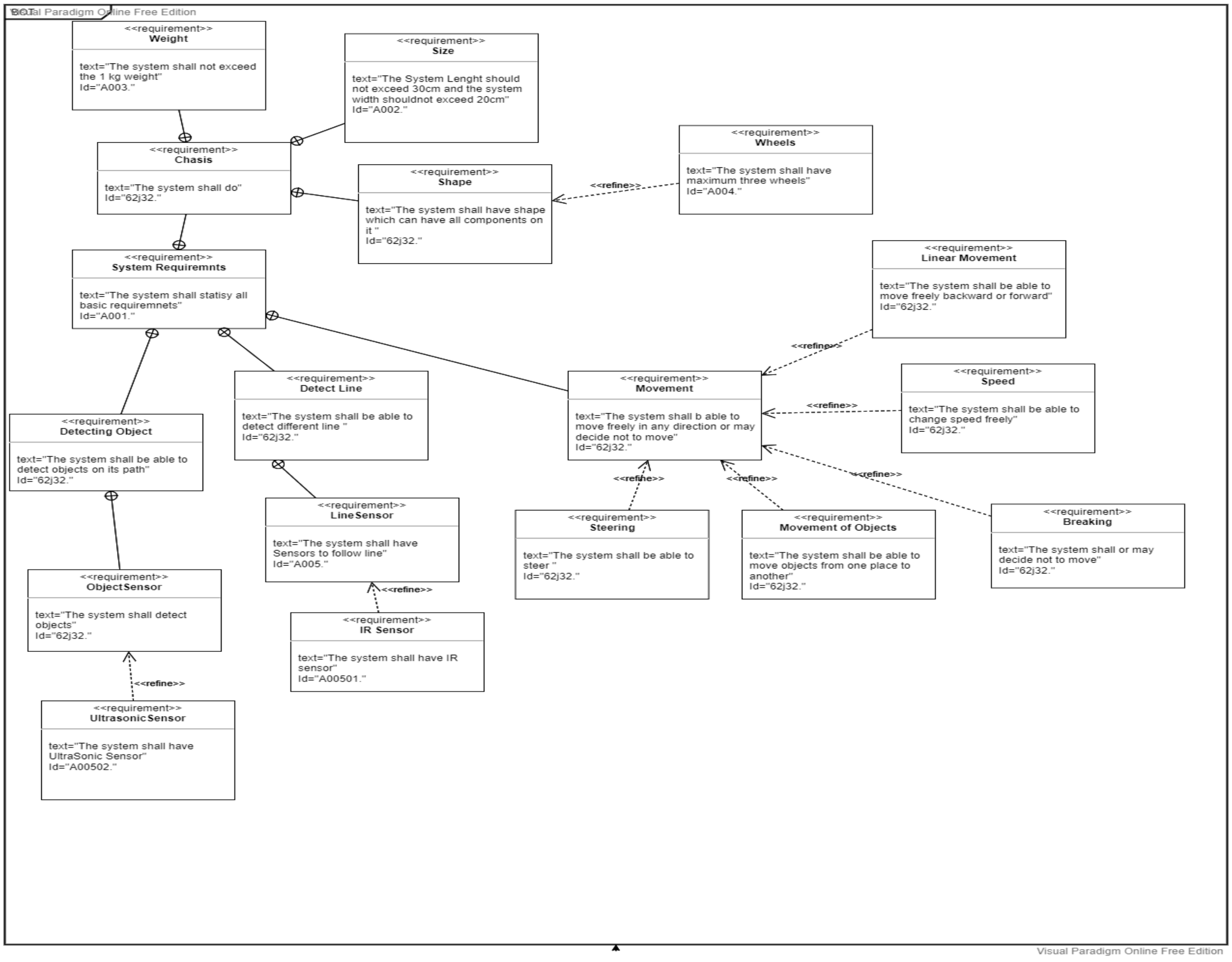

BOT
Visual Paradigm Online Free Edition
<<requirement>> Weight
text="The system shall not exceed the 1 kg weight" Id="A003."
<<requirement>> Size
text="The System Lenght should not exceed 30cm and the system width shouldnot exceed 20cm" Id="A002."
<<requirement>> Chasis
text="The system shall do" Id="62j32."
<<requirement>> Shape
text="The system shall have shape which can have all components on it " Id="62j32."
<<requirement>> Wheels
text="The system shall have maximum three wheels" Id="A004."
<<refine>>
<<requirement>> System Requiremnts
text="The system shall statisy all basic requiremnets" Id="A001."
<<requirement>> Linear Movement
text="The system shall be able to move freely backward or forward" Id="62j32."
<<refine>>
<<requirement>> Speed
text="The system shall be able to change speed freely" Id="62j32."
<<refine>>
<<requirement>> Detect Line
text="The system shall be able to detect different line " Id="62j32."
<<requirement>> Movement
text="The system shall b able to move freely in any direction or may decide not to move" Id="62j32."
<<requirement>> Detecting Object
text="The system shall be able to detect objects on its path" Id="62j32."
<<refine>>
<<refine>>
<<refine>>
<<requirement>> LineSensor
text="The system shall have Sensors to follow line" Id="A005."
<<requirement>> Steering
text="The system shall be able to steer " Id="62j32."
<<requirement>> Movement of Objects
text="The system shall be able to move objects from one place to another" Id="62j32."
<<requirement>> Breaking
text="The system shall or may decide not to move" Id="62j32."
<<refine>>
<<requirement>> ObjectSensor
text="The system shall detect objects" Id="62j32."
<<requirement>> IR Sensor
text="The system shall have IR sensor" Id="A00501."
<<refine>>
<<requirement>> UltrasonicSensor
text="The system shall have UltraSonic Sensor" Id="A00502."
Visual Paradigm Online Free Edition

Fig. 7. Requirements Diagram

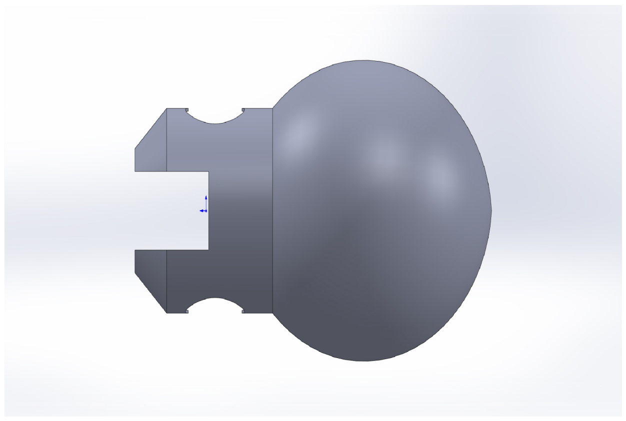


Fig. 11. First iteration Top View

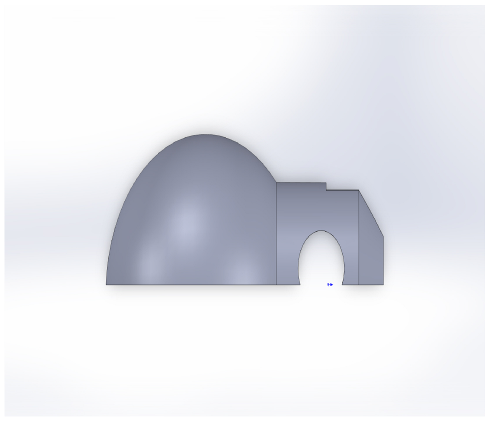


Fig. 12. First iteration Side View

2) Assembly of First Design: Refer to Fig. 13.

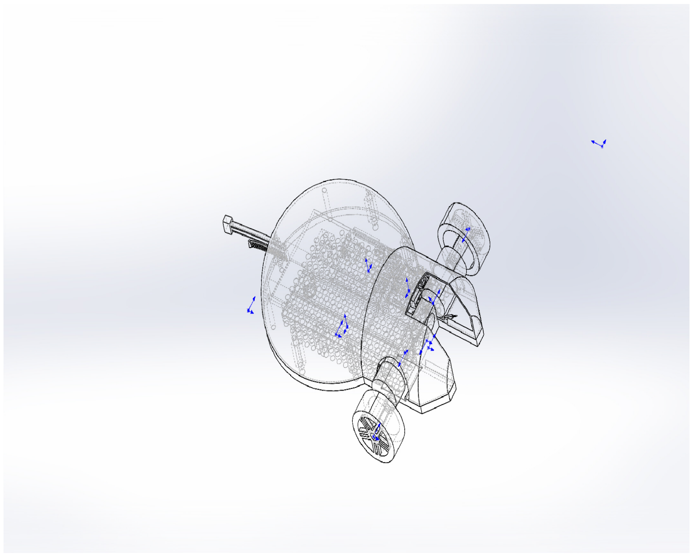


Fig. 13. Assembly of First Iteration

3) Final Iteration: As it was a little bit challenging to fit all component on First Iteration Design, we decided to increase the surface area of our bot and to also include specific space for color sensor and line sensor

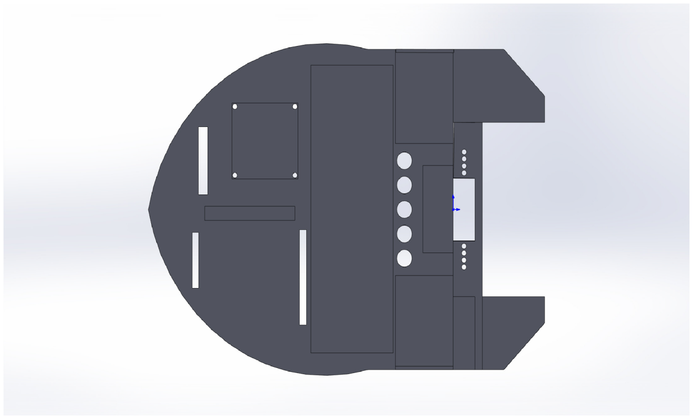


Fig. 14. Lower Chassis Final Design

C. Final Design (Jaouaher Belgacem)

For the final design, we decided to keep the lower chassis of the final iteration of the second design with making some changes in it, to fit all the components. After the updates, all the components fit in the chassis.

We inspired this shape from the falcon spaceship, and we update it according to the dimensional requirements and the hardware components. The rounded shape is a good fit for all the wires laying under it. We need some height at the back part because the breadboard is placed on top of the battery with the help of holders.
The rounded part of the design fits the battery, Arduino card, the motor drive and the breadboard. However, the front part contains the motors and the ultrasonic sensor

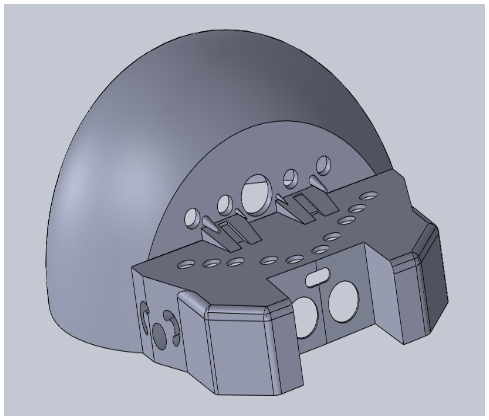


Fig. 15. the Upper body of the final design

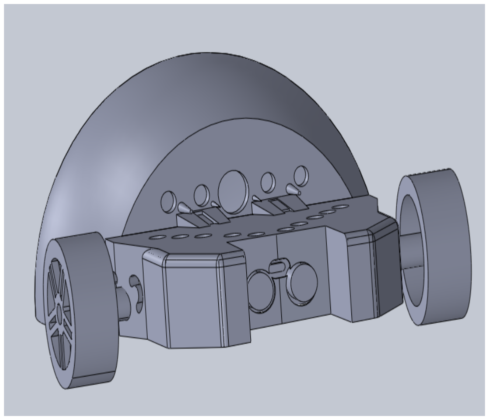


Fig. 16. Assembly of Final Design

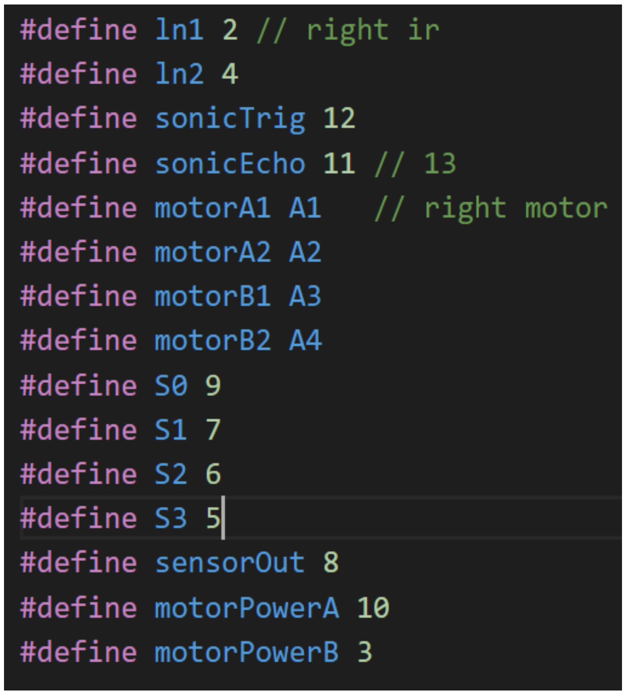
IV. Algorithm and Code

A. Definitions (Arsany Girgis)

We defined all the Arduino pins by names that indicate their function in the hardware implementation so that if any of the pins have changed or the cabling changed it would be easy to change it here only and not in the entire code, and by using their names defined here it would automatically change the pins in the whole code wherever it's mentioned.

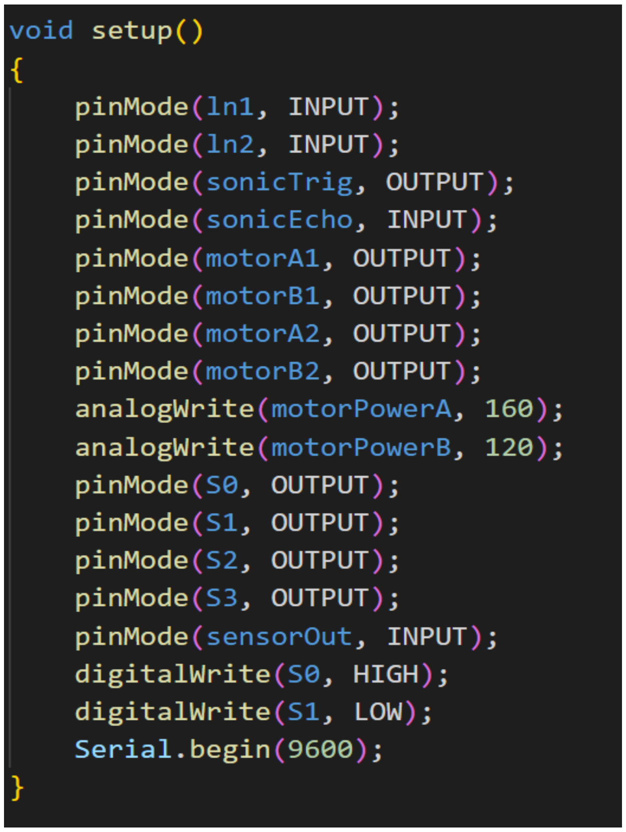
B. Setup (Arsany Girgis)

For the setup, the "ln1", and "ln2" of the infrared sensor are taken as input to be able to find the line, as well as the "sonicEcho" of the Ultrasonic sensor, which receives the signal sent by "sonicTrig" to detect any object while the motor pins are set as outputs to be able to move the wheels and setting the motor power with the calibrated speeds to run smoothly.
For the color sensor, 4 pins are output pins as they act as filters to simulate the value of the red, green, and blue colors with each color having its own filter configuration, and the pin "sensorOut" acts



```
#define ln1 2 // right ir
#define ln2 4
#define sonicTrig 12
#define sonicEcho 11 // 13
#define motorA1 A1   // right motor
#define motorA2 A2
#define motorB1 A3
#define motorB2 A4
#define S0 9
#define S1 7
#define S2 6
#define S3 5
#define sensorOut 8
#define motorPowerA 10
#define motorPowerB 3
```

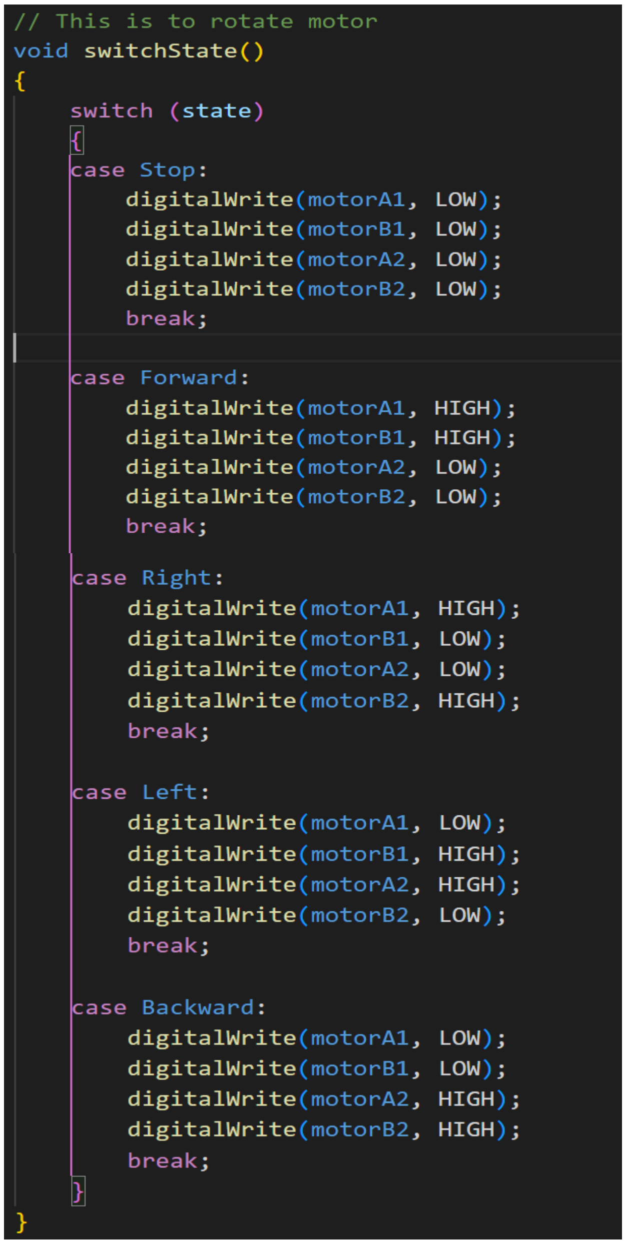
Fig. 17. Hash Defines



```
void setup()
{
    pinMode(ln1, INPUT);
    pinMode(ln2, INPUT);
    pinMode(sonicTrig, OUTPUT);
    pinMode(sonicEcho, INPUT);
    pinMode(motorA1, OUTPUT);
    pinMode(motorB1, OUTPUT);
    pinMode(motorA2, OUTPUT);
    pinMode(motorB2, OUTPUT);
    analogWrite(motorPowerA, 160);
    analogWrite(motorPowerB, 120);
    pinMode(S0, OUTPUT);
    pinMode(S1, OUTPUT);
    pinMode(S2, OUTPUT);
    pinMode(S3, OUTPUT);
    pinMode(sensorOut, INPUT);
    digitalWrite(S0, HIGH);
    digitalWrite(S1, LOW);
    Serial.begin(9600);
}
```

Fig. 18. setup

1) The switchState Function (Arsany Girgis): In the switchState function a switch case is implemented to check for



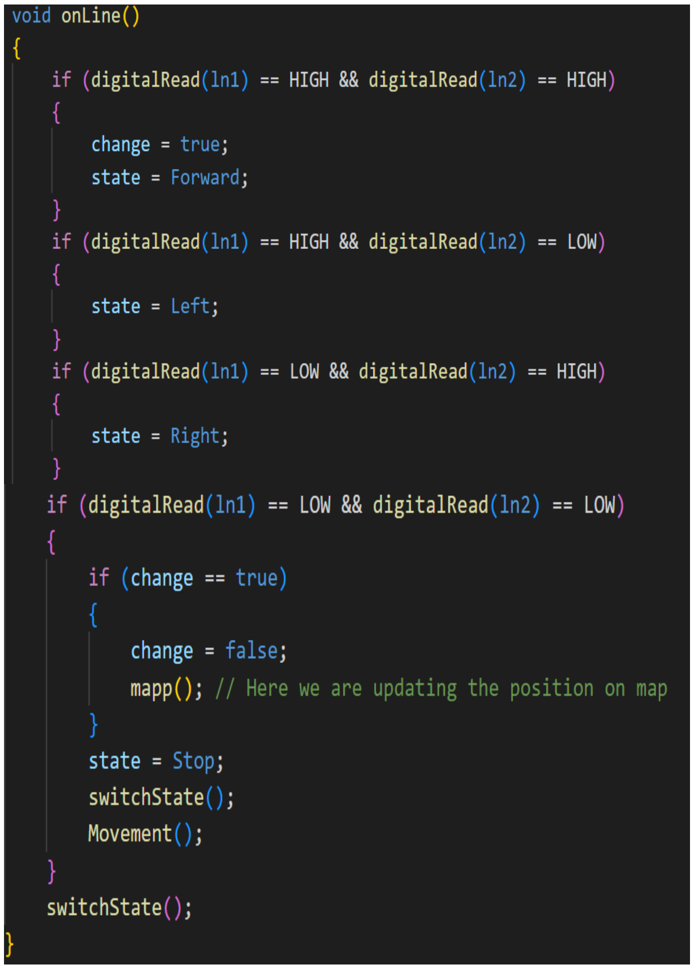
```
// This is to rotate motor
void switchState()
{
    switch (state)
    {
    case Stop:
        digitalWrite(motorA1, LOW);
        digitalWrite(motorB1, LOW);
        digitalWrite(motorA2, LOW);
        digitalWrite(motorB2, LOW);
        break;

    case Forward:
        digitalWrite(motorA1, HIGH);
        digitalWrite(motorB1, HIGH);
        digitalWrite(motorA2, LOW);
        digitalWrite(motorB2, LOW);
        break;

    case Right:
        digitalWrite(motorA1, HIGH);
        digitalWrite(motorB1, LOW);
        digitalWrite(motorA2, LOW);
        digitalWrite(motorB2, HIGH);
        break;

    case Left:
        digitalWrite(motorA1, LOW);
        digitalWrite(motorB1, HIGH);
        digitalWrite(motorA2, HIGH);
        digitalWrite(motorB2, LOW);
        break;

    case Backward:
        digitalWrite(motorA1, LOW);
        digitalWrite(motorB1, LOW);
        digitalWrite(motorA2, HIGH);
        digitalWrite(motorB2, HIGH);
        break;
    }
}
```

Fig. 19. Function to configure the motor pins

each situation if the state is Forward, Stop, Left, Right or backward. The variable state is then updated in other functions like the Movement function, and online function each time the vehicle needs to turn or stop or move where the switchState function then sets the pin configurations for the vehicle to move accordingly.

C. onLine Function (Arsany Girgis)

The online function reads the inputs of the infrared sensors and then determines the required behavior so when both sensors detect black (High), this means that the colored line is between the vehicle which is the case for the car to move forward, while if only the left sensor detects (low) which means it's on a colored line this means the car should move to the left to calibrate itself to be on the line. And it's the same case if only the right sensor detects (low) then the state variable should be equal to Right



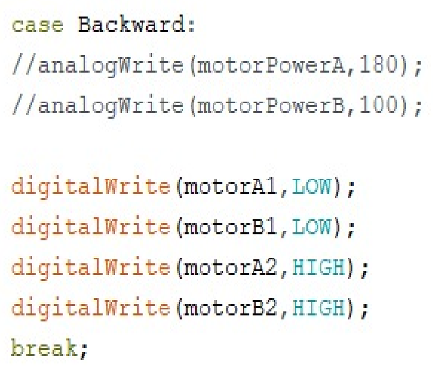
```
void onLine()
{
    if (digitalRead(ln1) == HIGH && digitalRead(ln2) == HIGH)
    {
        change = true;
        state = Forward;
    }
    if (digitalRead(ln1) == HIGH && digitalRead(ln2) == LOW)
    {
        state = Left;
    }
    if (digitalRead(ln1) == LOW && digitalRead(ln2) == HIGH)
    {
        state = Right;
    }
    if (digitalRead(ln1) == LOW && digitalRead(ln2) == LOW)
    {
        if (change == true)
        {
            change = false;
            mapp(); // Here we are updating the position on map
        }
        state = Stop;
        switchState();
        Movement();
    }
    switchState();
}
```

Fig. 20. Function to follow the line

For the case where both sensors are detecting low (the car is on a colored line), we use this information to count where we are on the map given that both sensors were previously detecting HIGH and this is implemented in the flag change. At this point, it's needed to call the movement function to actually know where to go then.

D. 90 Degree Turn (Jaouaher Belgacem)

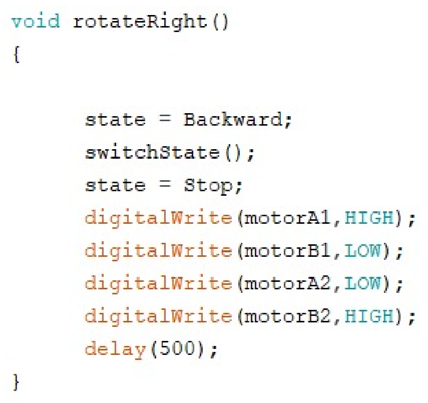
For the function of 90 degrees turn, we need to add another case in the switch case, which is the backward, as the vehicle behaves unsalable at the crossing lines and sometimes it turns after the cross. By moving the car backwards, it has enough space to turn until it detects the lines again with IR sensors.



```
case Backward:
//analogWrite(motorPowerA,180);
//analogWrite(motorPowerB,100);

digitalWrite(motorA1,LOW);
digitalWrite(motorB1,LOW);
digitalWrite(motorA2,HIGH);
digitalWrite(motorB2,HIGH);
break;
```

Fig. 21. Backward switch case

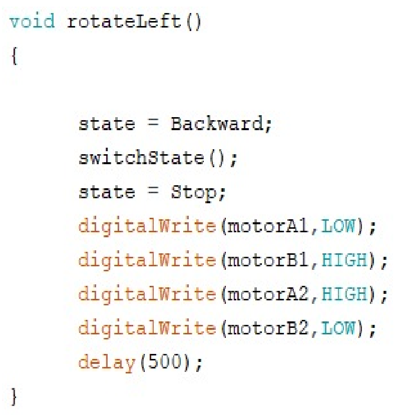
For achieving the 90 degrees turn the vehicle moves backwards, stops, and then turns to the right in this case



```
void rotateRight()
{

     state = Backward;
     switchState();
     state = Stop;
     digitalWrite(motorA1,HIGH);
     digitalWrite(motorB1,LOW);
     digitalWrite(motorA2,LOW);
     digitalWrite(motorB2,HIGH);
     delay(500);
}
```

Fig. 22. Turning right function

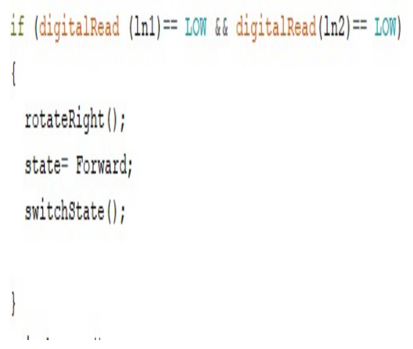
Turning to the left has the same concept as the right but we only reversed the activated parts of the motors.



```
void rotateLeft()
{

     state = Backward;
     switchState();
     state = Stop;
     digitalWrite(motorA1,LOW);
     digitalWrite(motorB1,HIGH);
     digitalWrite(motorA2,HIGH);
     digitalWrite(motorB2,LOW);
     delay(500);
}
```

Fig. 23. Turning Left function

The vehicle keeps calling the turning function until the IR sensors detect one of the lines or both. Once the car turns a 90-degree it steers on the line to follow the track. For this function, we used the IR sensors only. We did limit the color sensor to reading colors only. As we noticed that if the vehicle stops or moves slower, it may detect a color more than one time which messed up the counters



```
if (digitalRead (ln1)== LOW && digitalRead(ln2)== LOW)
{
 rotateRight();
 state= Forward;
 switchState();


}
```

Fig. 24. calling the turning function to the right in the main loop

1) Object Detection (Jaouaher Belgacem): For object detection, ultrasonic is the responsible sensor as it can detect up to 4m. The vehicle keeps moving by following the lines and when it detects an object, there are two options:

- The distance is bigger than the minimum distance for stopping so the car keeps moving
- The distance is equal to or less than the minimum distance. As a result, the vehicle stops.

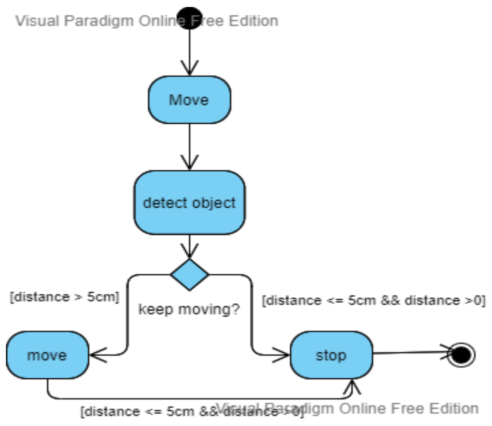
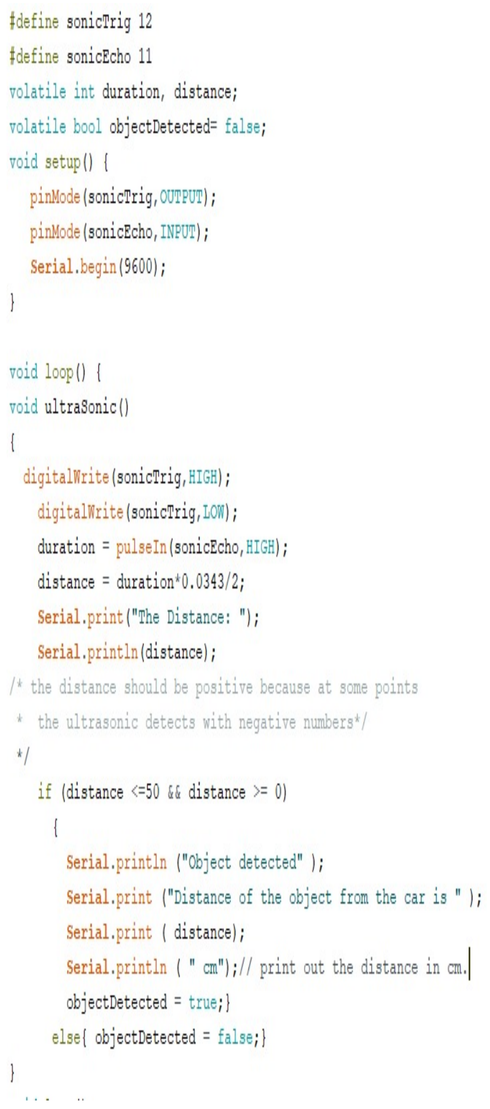

Visual Paradigm Online Free Edition
Move
detect object
keep moving?
[distance > 5cm]
[distance <= 5cm && distance >0]
move
stop
[distance <= 5cm && distance >0]
Visual Paradigm Online Free Edition

Fig. 25. Activity diagram of object detection

The ultrasonic detects large distances and displays it on the serial monitor as a positive number and sometimes as a negative number. However, when the distance is short it displays only positive numbers. To ensure a stable behavior we required that the distance should be less than a specific number and at the same time greater than zero.

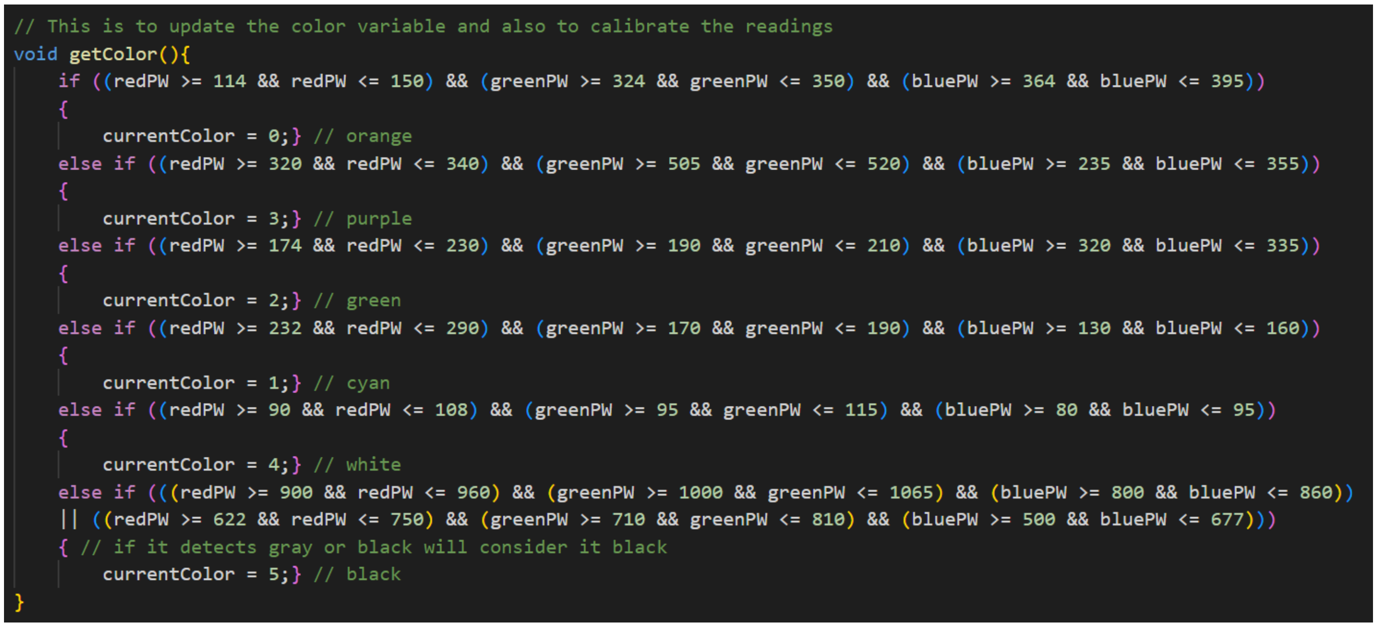


```
#define sonicTrig 12
#define sonicEcho 11
volatile int duration, distance;
volatile bool objectDetected= false;
void setup() {
  pinMode(sonicTrig,OUTPUT);
  pinMode(sonicEcho,INPUT);
  Serial.begin(9600);
}

void loop() {
void ultraSonic()
{
 digitalWrite(sonicTrig,HIGH);
   digitalWrite(sonicTrig,LOW);
   duration = pulseIn(sonicEcho,HIGH);
   distance = duration*0.0343/2;
   Serial.print("The Distance: ");
   Serial.println(distance);
/* the distance should be positive because at some points
 * the ultrasonic detects with negative numbers*/
 */
   if (distance <=50 && distance >= 0)
     {
      Serial.println ("Object detected" );
      Serial.print ("Distance of the object from the car is " );
      Serial.print ( distance);
      Serial.println ( " cm");// print out the distance in cm.
      objectDetected = true;}
     else{ objectDetected = false;}
}
```

Fig. 26. Implementation of object detection

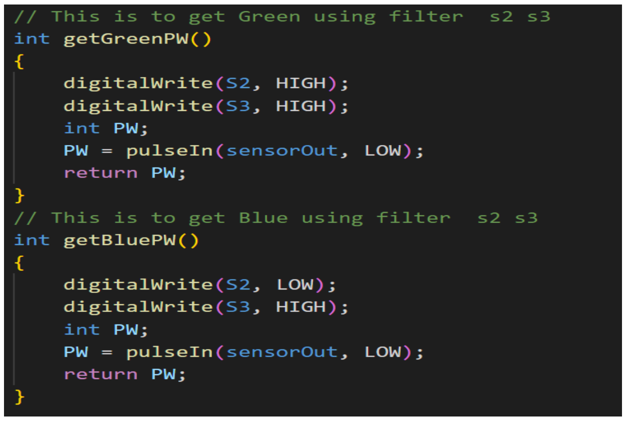
E. Color Detection (Arsany Girgis)

In the getColor function, the red, green, and blue pulse widths are checked to be able to differentiate between the ranges of different colors. Where each color has its own range of frequencies. The if conditions check to find any case and update the currentColor which is the color where the color is currently passing over.



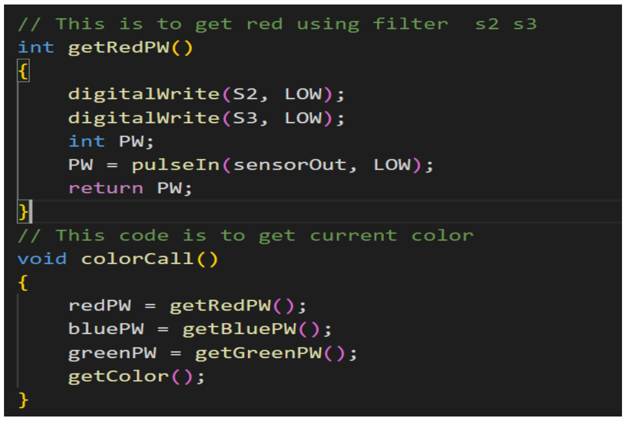
```
// This is to update the color variable and also to calibrate the readings
void getColor(){
    if ((redPW >= 114 && redPW <= 150) && (greenPW >= 324 && greenPW <= 350) && (bluePW >= 364 && bluePW <= 395))
    {
        currentColor = 0;} // orange
    else if ((redPW >= 320 && redPW <= 340) && (greenPW >= 505 && greenPW <= 520) && (bluePW >= 235 && bluePW <= 355))
    {
        currentColor = 3;} // purple
    else if ((redPW >= 174 && redPW <= 230) && (greenPW >= 190 && greenPW <= 210) && (bluePW >= 320 && bluePW <= 335))
    {
        currentColor = 2;} // green
    else if ((redPW >= 232 && redPW <= 290) && (greenPW >= 170 && greenPW <= 190) && (bluePW >= 130 && bluePW <= 160))
    {
        currentColor = 1;} // cyan
    else if ((redPW >= 90 && redPW <= 108) && (greenPW >= 95 && greenPW <= 115) && (bluePW >= 80 && bluePW <= 95))
    {
        currentColor = 4;} // white
    else if (((redPW >= 900 && redPW <= 960) && (greenPW >= 1000 && greenPW <= 1065) && (bluePW >= 800 && bluePW <= 860))
    || ((redPW >= 622 && redPW <= 750) && (greenPW >= 710 && greenPW <= 810) && (bluePW >= 500 && bluePW <= 677)))
    { // if it detects gray or black will consider it black
        currentColor = 5;} // black
}
```

Fig. 27. Function to determine which color the car is passing on



```
// This is to get Green using filter  s2 s3
int getGreenPW()
{
    digitalWrite(S2, HIGH);
    digitalWrite(S3, HIGH);
    int PW;
    PW = pulseIn(sensorOut, LOW);
    return PW;
}
// This is to get Blue using filter  s2 s3
int getBluePW()
{
    digitalWrite(S2, LOW);
    digitalWrite(S3, HIGH);
    int PW;
    PW = pulseIn(sensorOut, LOW);
    return PW;
}
```

Fig. 28. filter to receive each color



```
// This is to get red using filter  s2 s3
int getRedPW()
{
    digitalWrite(S2, LOW);
    digitalWrite(S3, LOW);
    int PW;
    PW = pulseIn(sensorOut, LOW);
    return PW;
}
// This code is to get current color
void colorCall()
{
    redPW = getRedPW();
    bluePW = getBluePW();
    greenPW = getGreenPW();
    getColor();
}
```

Fig. 29. calling all functions to work together

For the getGreenPW function the filter is set for the s2, and s3 to HIGH and the sensor gets the pulse width using the built-in function pulse-in. it's also the same for getBluePW abd getRedPW functions but each with its own filter. Where the filter for the blue color is s2 set to low and s3 set to HIGH, while for red, both s2, and s3 pins are set to LOW.

The colorCall function gets all the value of redPW, greenPW, and bluePw so they could be checked in the getColor function which frequency they are satisfying to set the currentColor.

F. Mapping (Jasmeet Singh Matta)

For knowing the position of the bot on the map one needs to find which direction it is facing?. How does it know where it is? and how will it know where to go next ?.

"Which direction is the bot facing?.": Even though this sound complicated the answer to this is very simple. The bot is initially facing north and there are only 90° turns available. Therefore circular mapping coordinates is implemented, where if bot rotate left it add 1 to the counter and once it reaches the max or min (depending on turn) it will reset counter with respect to the next turn. For example facing east it will be 3 therefore while turning left it will become 4 but it is out of bound, therefore it is reset back to zero. Same applies when it is facing north and turns to right. It will be -1 and it is out of bound so counter becomes max.

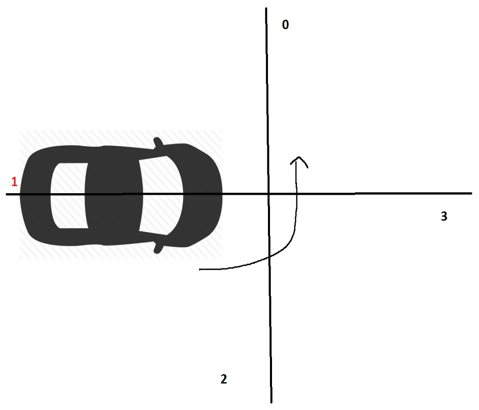
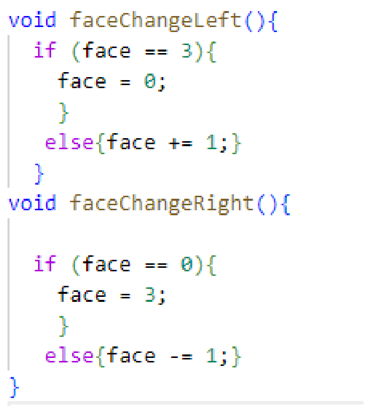


Fig. 31. Rotation of Car from East to North



```
void faceChangeLeft(){
  if (face == 3){
    face = 0;
    }
   else{face += 1;}
  }
void faceChangeRight(){

  if (face == 0){
    face = 3;
    }
   else{face -= 1;}
}
```

Fig. 32. How face are updated

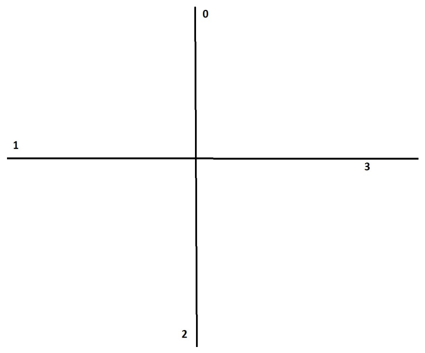
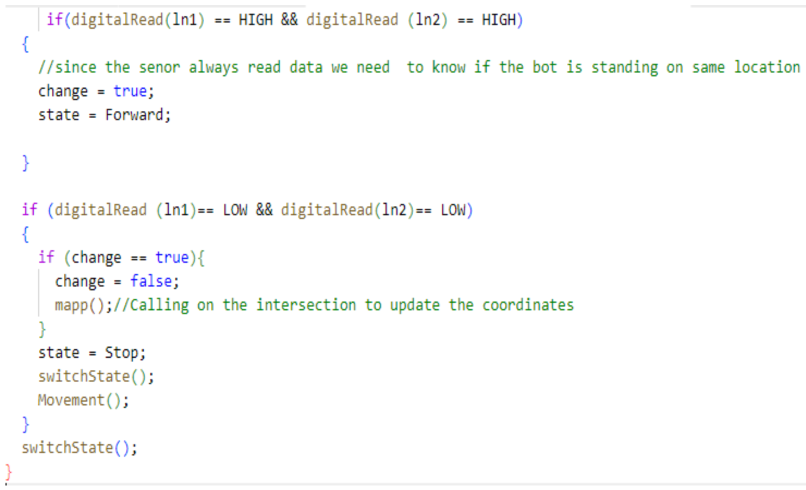


Fig. 30. General Idea of Faces

How does the bot know where it is?: For this the initial position of the bot is known i.e. (1,1) 1 in X and 1 in Y. For this project 2D coordinate system is used where the coordinate (1,1) is the origin and only 1st quadrant.
For every time the intersection is reached, the position has to be updated. Therefore using the face of bot one can add or subtract from the coordinate system.

- While facing North(0): add 1 to Y coordinate
- While facing East(3) : add 1 to X coordinate
- While facing South(2): subtract 1 from Y coordinate
- While facing West(1) : subtract 1 from X coordinate



```
  if(digitalRead(ln1) == HIGH && digitalRead (ln2) == HIGH)
{
  //since the senor always read data we need  to know if the bot is standing on same location
  change = true;
  state = Forward;

}

if (digitalRead (ln1)== LOW && digitalRead(ln2)== LOW)
{
  if (change == true){
    change = false;
    mapp();//Calling on the intersection to update the coordinates
  }
  state = Stop;
  switchState();
  Movement();
}
switchState();
}
```

Fig. 33. Intersection and Calling for updates

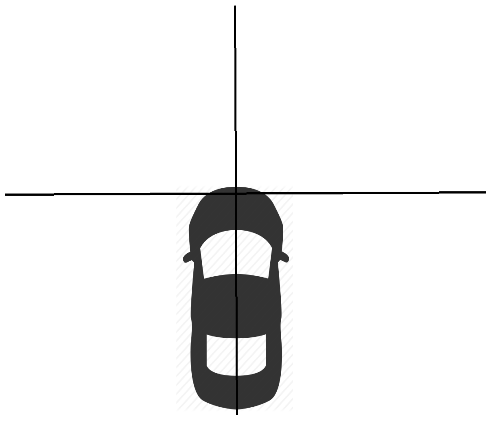
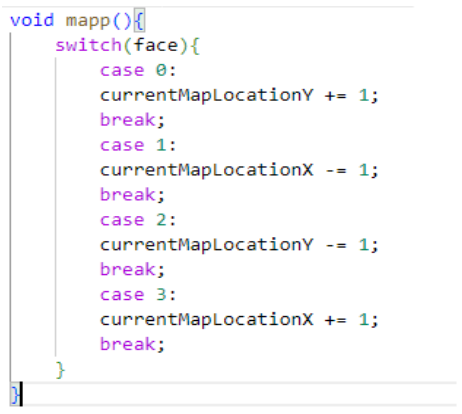


Fig. 34. car on Intersection

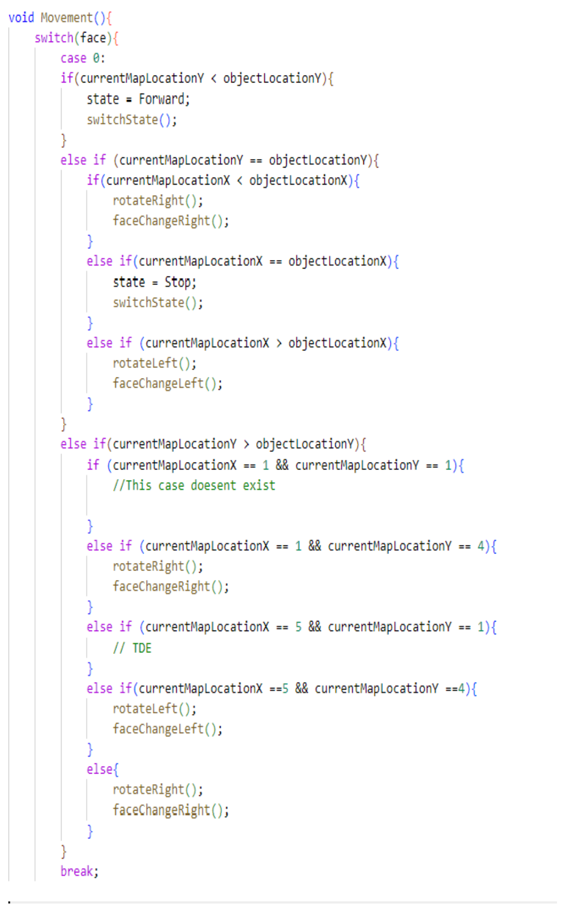


```
void mapp(){
    switch(face){
        case 0:
        currentMapLocationY += 1;
        break;
        case 1:
        currentMapLocationX -= 1;
        break;
        case 2:
        currentMapLocationY -= 1;
        break;
        case 3:
        currentMapLocationX += 1;
        break;
    }
}
```

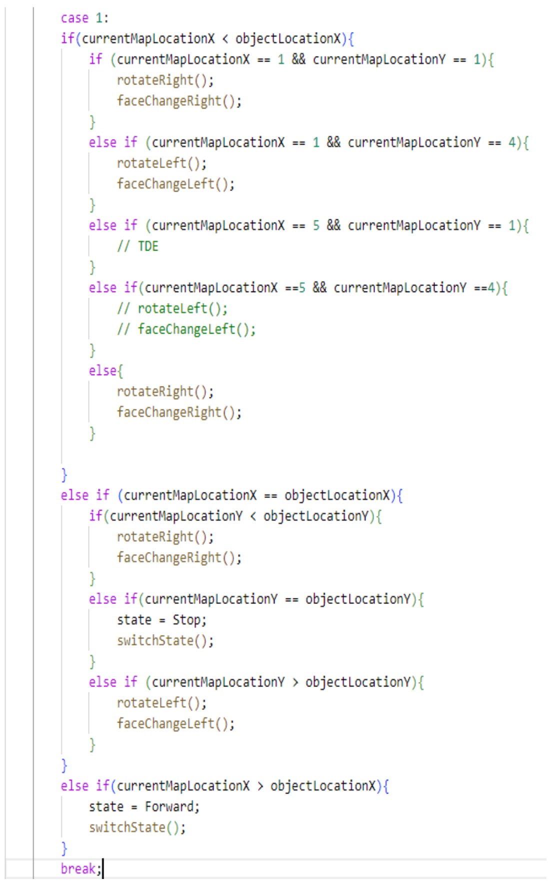
Fig. 35. Update of Location

G. Movement Algorithm (Jasmeet Singh Matta)

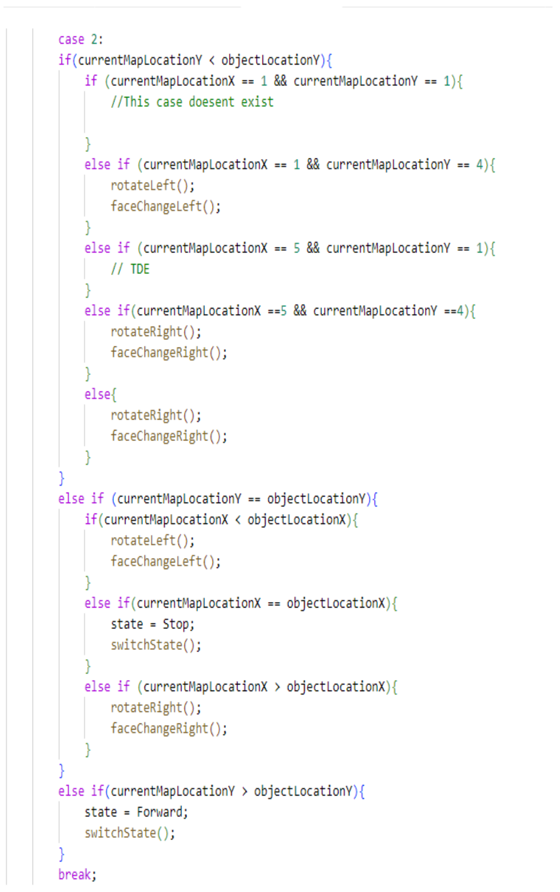
how will it know where to go next ?. To answer this question, it is very simple math of comparing location and face. Initially the bot is facing North and is at origin(1,1).We have the second coordinates as (2,2) so as the bot is facing north and its currentY location is less than objectiveY location by 1. So the bot has to move one intersection up. Since the bot is still facing north and currentY Coordinates matches with objectiveY coordinates. Now currentX and objectiveX coordinates are compared so as to know which way to turn. Also since the bot is at the border of 1st quadrant, it will always turn right. And this logic is applied on all the faces as the bot switch through them The coordinates given to the bot should be non negative.



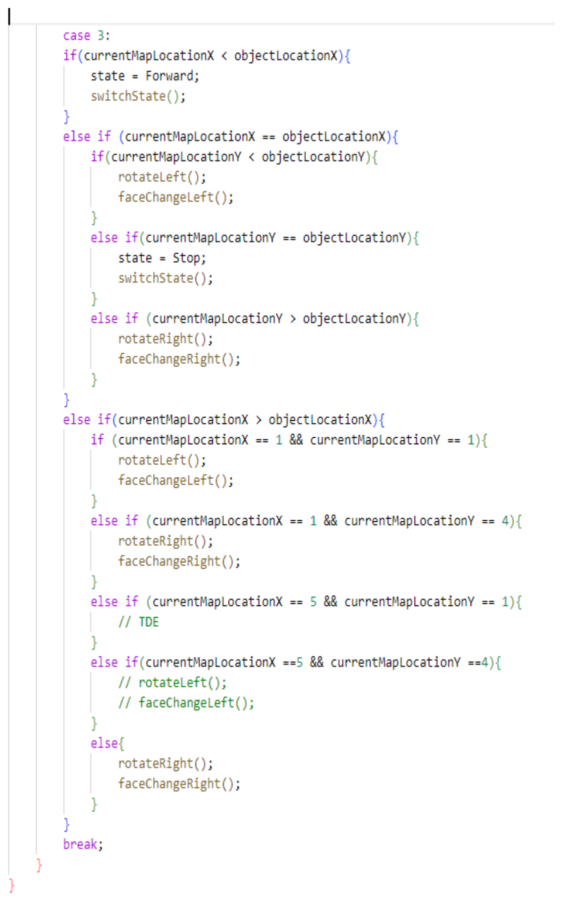
```
void Movement(){
  switch(face){
    case 0:
    if(currentMapLocationY < objectLocationY){
      state = Forward;
      switchState();
    }
    else if (currentMapLocationY == objectLocationY){
      if(currentMapLocationX < objectLocationX){
        rotateRight();
        faceChangeRight();
      }
      else if(currentMapLocationX == objectLocationX){
        state = Stop;
        switchState();
      }
      else if (currentMapLocationX > objectLocationX){
        rotateLeft();
        faceChangeLeft();
      }
    }
    else if(currentMapLocationY > objectLocationY){
      if (currentMapLocationX == 1 && currentMapLocationY == 1){
        //This case doesent exist

      }
      else if (currentMapLocationX == 1 && currentMapLocationY == 4){
        rotateRight();
        faceChangeRight();
      }
      else if (currentMapLocationX == 5 && currentMapLocationY == 1){
        // TDE
      }
      else if(currentMapLocationX ==5 && currentMapLocationY ==4){
        rotateLeft();
        faceChangeLeft();
      }
      else{
        rotateRight();
        faceChangeRight();
      }
    }
    break;
```



```
case 1:
if(currentMapLocationX < objectLocationX){
  if (currentMapLocationX == 1 && currentMapLocationY == 1){
    rotateRight();
    faceChangeRight();
  }
  else if (currentMapLocationX == 1 && currentMapLocationY == 4){
    rotateLeft();
    faceChangeLeft();
  }
  else if (currentMapLocationX == 5 && currentMapLocationY == 1){
    // TDE
  }
  else if(currentMapLocationX ==5 && currentMapLocationY ==4){
    // rotateLeft();
    // faceChangeLeft();
  }
  else{
    rotateRight();
    faceChangeRight();
  }

}
else if (currentMapLocationX == objectLocationX){
  if(currentMapLocationY < objectLocationY){
    rotateRight();
    faceChangeRight();
  }
  else if(currentMapLocationY == objectLocationY){
    state = Stop;
    switchState();
  }
  else if (currentMapLocationY > objectLocationY){
    rotateLeft();
    faceChangeLeft();
  }
}
else if(currentMapLocationX > objectLocationX){
  state = Forward;
  switchState();
}
break;
```



```
case 2:
if(currentMapLocationY < objectLocationY){
  if (currentMapLocationX == 1 && currentMapLocationY == 1){
    //This case doesent exist

  }
  else if (currentMapLocationX == 1 && currentMapLocationY == 4){
    rotateLeft();
    faceChangeLeft();
  }
  else if (currentMapLocationX == 5 && currentMapLocationY == 1){
    // TDE
  }
  else if(currentMapLocationX ==5 && currentMapLocationY ==4){
    rotateRight();
    faceChangeRight();
  }
  else{
    rotateRight();
    faceChangeRight();
  }
}
else if (currentMapLocationY == objectLocationY){
  if(currentMapLocationX < objectLocationX){
    rotateLeft();
    faceChangeLeft();
  }
  else if(currentMapLocationX == objectLocationX){
    state = Stop;
    switchState();
  }
  else if (currentMapLocationX > objectLocationX){
    rotateRight();
    faceChangeRight();
  }
}
else if(currentMapLocationY > objectLocationY){
  state = Forward;
  switchState();
}
break;
```



```
        case 3:
        if(currentMapLocationX < objectLocationX){
            state = Forward;
            switchState();
        }
        else if (currentMapLocationX == objectLocationX){
            if(currentMapLocationY < objectLocationY){
                rotateLeft();
                faceChangeLeft();
            }
            else if(currentMapLocationY == objectLocationY){
                state = Stop;
                switchState();
            }
            else if (currentMapLocationY > objectLocationY){
                rotateRight();
                faceChangeRight();
            }
        }
        else if(currentMapLocationX > objectLocationX){
            if (currentMapLocationX == 1 && currentMapLocationY == 1){
                rotateLeft();
                faceChangeLeft();
            }
            else if (currentMapLocationX == 1 && currentMapLocationY == 4){
                rotateRight();
                faceChangeRight();
            }
            else if (currentMapLocationX == 5 && currentMapLocationY == 1){
                // TDE
            }
            else if(currentMapLocationX ==5 && currentMapLocationY ==4){
                // rotateLeft();
                // faceChangeLeft();
            }
            else{
                rotateRight();
                faceChangeRight();
            }
        }
        break;
    }
}
```

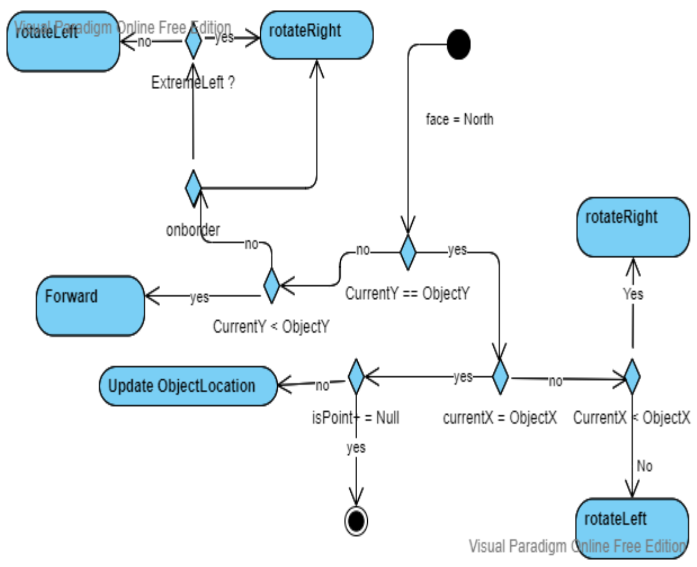


Fig. 38. Algorithm's Activity While facing North

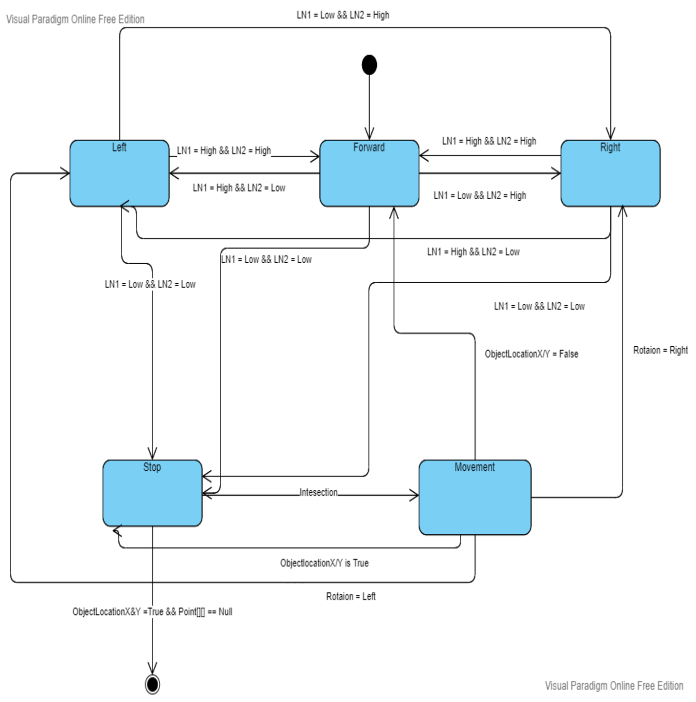
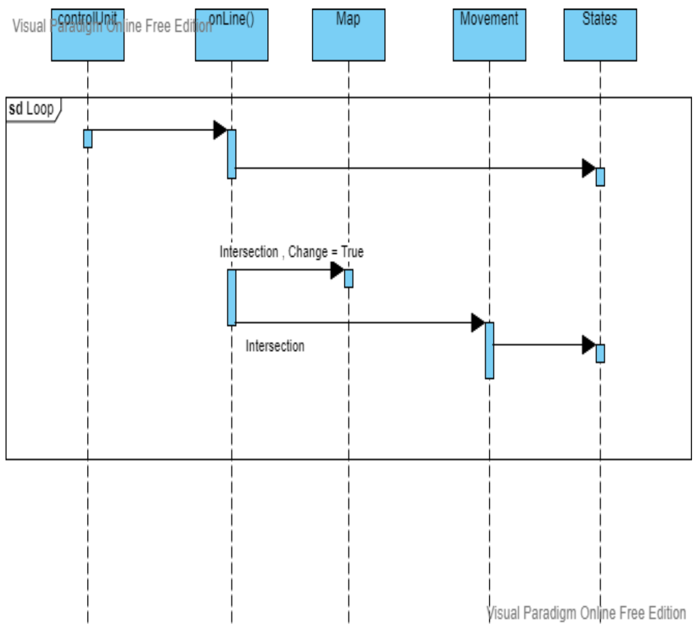


Fig. 36. Overview States



controlUnit
onLine()
Map
Movement
States
Visual Paradigm Online Free Edition
sd Loop
Intersection , Change = True
Intersection
Visual Paradigm Online Free Edition

Fig. 37. Overview Activity

Future Improvement (Jasmeet Singh Matta)

In future with same environment one can add obstacle avoidance algorithm in this section a scenario of how to implement is shared. While moving forward the bot scan for object in front of it and if there is a object save its distance in a variable. When the bot reach intersection compare if the value is less than the total distance between two intersection if yes then turn left or right one has to declare it at borders and then can continue using movement function

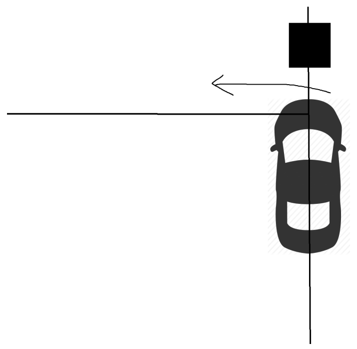


Fig. 39. Right Border Condition

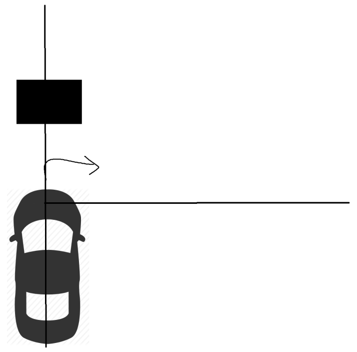


Fig. 40. Left Border Condition

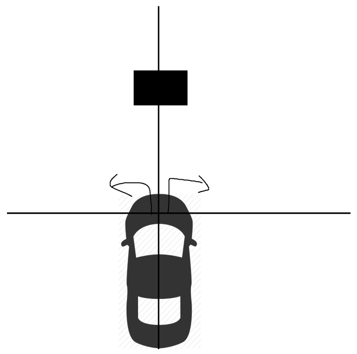


Fig. 41. Middle Condition

CONCLUSION (JASMEET SINGH MATTA)

During the development, we realize that it is important to reorganize your thought and requirement into a model as it will help in the future and why designing is also an important part of engineering. Such as how a simple error in design or a slightly different design can have non relative output, For example, placement of an IR sensor on or off the line. Placement of Motor in front or in back etc. After completion of the project, we realise how the concept can be used in a real-life system such as an advanced driver-assistance system.