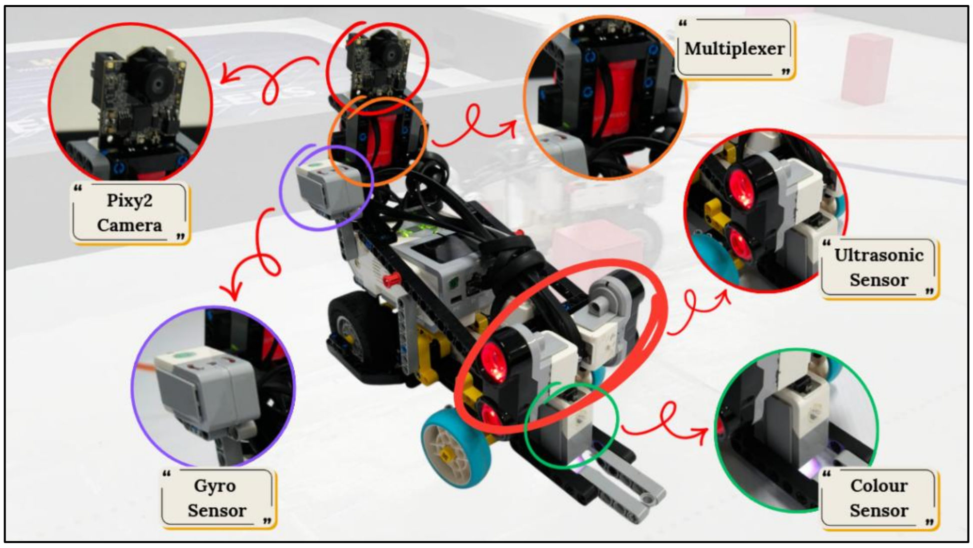
3.2 Obstacle Challenge

To complete the **Obstacle Challenge**, our robot utilizes four sensors which is **one Pixy2 Camera**, **one gyro sensor**, **two ultrasonic sensors on** both the left and right sides **and one colour sensor.** Since both ultrasonic sensors need to be connected to the same port, we use the **multiplexer** so that the robot can read data from both of the sensors.

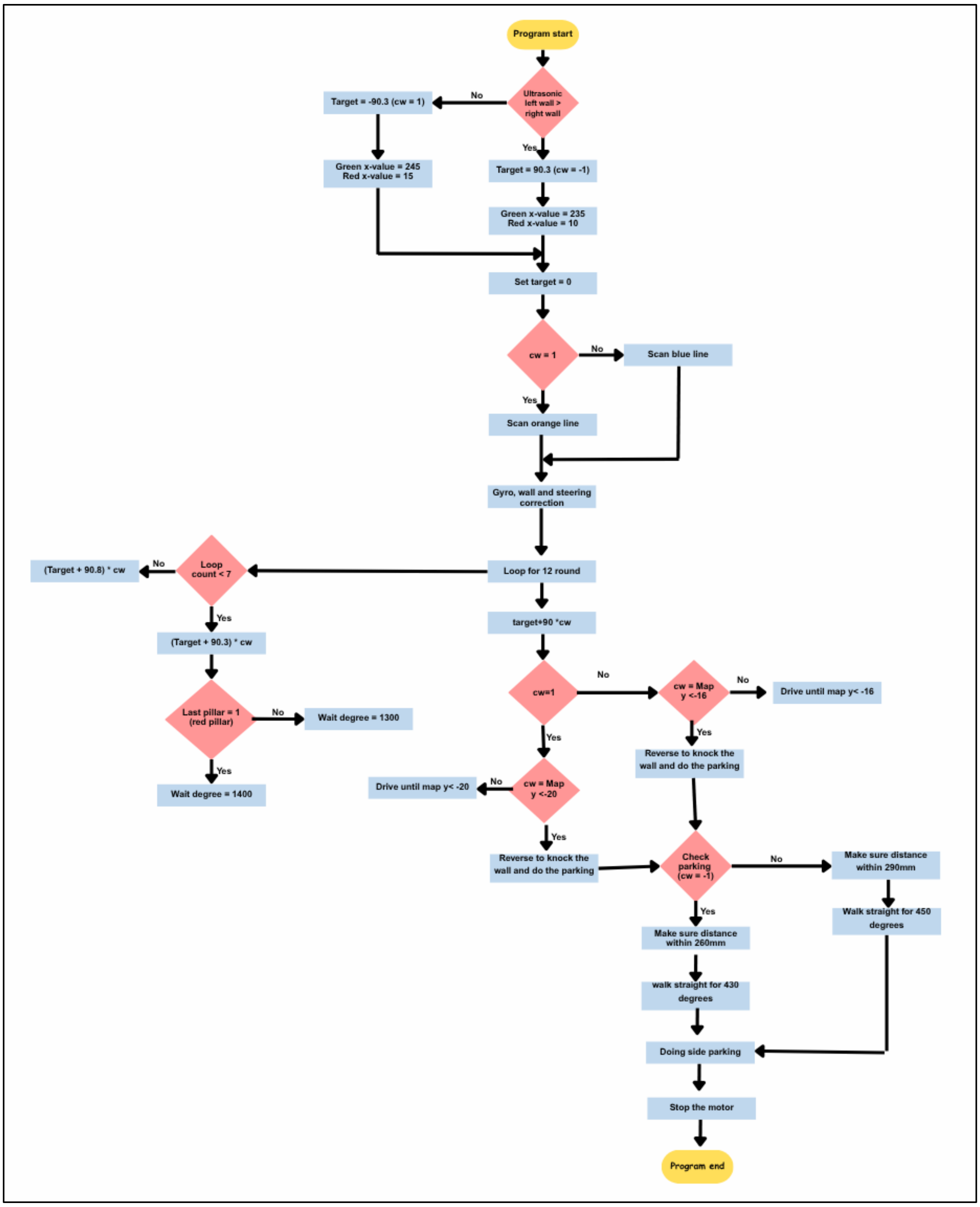


The **Pixy2 Camera is connected to port 1 and it helps to detect the presence of pillars and magenta parking lot during the round. Next, the EV3 Gyro Sensor that connected to port 2 helps to ensures that the robot does the accurate turns and keep the robot's heading straight during the round.** In addition, both of the **EV3 Ultrasonic Sensor** is connected to EV3 Brick port 3 via the multiplexer. They help to detect the presence of the inner and outer wall and prevent the robot from hitting into the wall. Lastly, we utilize the **EV3 Colour Sensor** to identify the coloured line on the map which is the orange and blue line. The identified colour allows the robot to know the direction accurately. For instance, if orange line is scanned first, the direction will be in clockwise direction.

Here is the quick overview about the program. When the robot begins, it first **resets its steering mechanism** to ensure it starts from a centred and accurate position. As the robot drives, a **PID steering control system** is used to maintain stable movement and the correction such as gyro correction and keeping optimum distances from the outer wall. During the round, the **Drive function** continuously combines inputs from pillar detection, wall tracking, and PID correction. The sensors such as Pixy 2 Camera allows the robot **to identify traffic**

sign accurately and choose the correct avoidance direction. After completing its last sections in the loop 12, the robot will **reverse to hit the wall** and aligns with the parking area and finally **performing a side parking neatly into the magenta parking zone**.

Diagram below shows the flowchart for **Obstacle Challenge:**



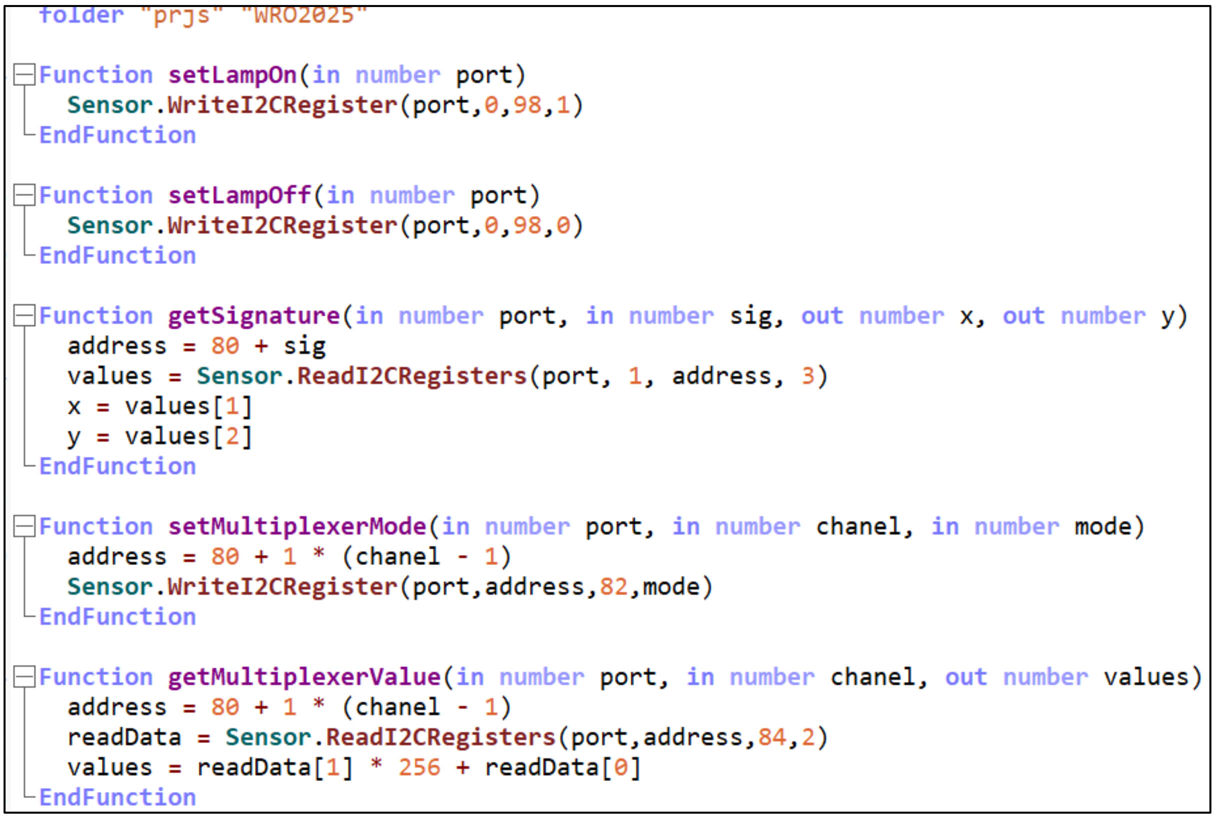
Appendix 3 contains the complete programming code for the **Obstacle Challenge.**

3.2.2 Coding explanation

Below is a brief explanation of the key parts of the code used to control the robot during the challenge.

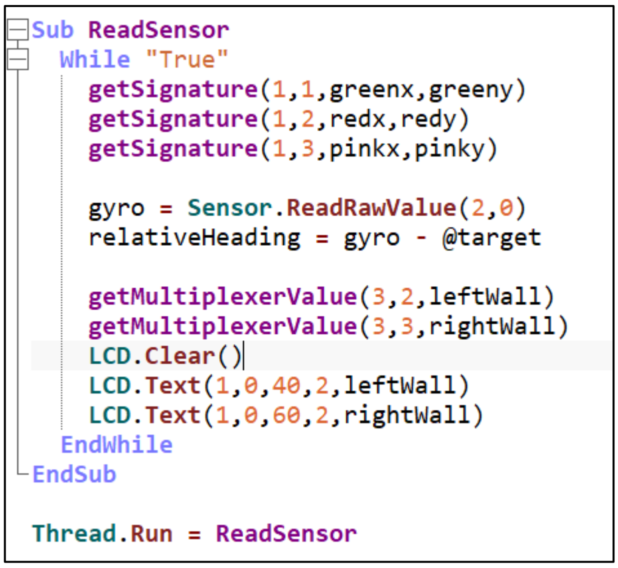
(a) Initialization and Set Up

This part is aim to set up the **Pixy2 Camera** and **multiplexer** before running. The **setLampOn** and **setLampOff** function turns the lamp of the sensor on and off while the **getSignature** function allow the **Pixy2Camera** to read the position of the green and red pillars through their XY coordinate shown in the **Pixy Mon V2** apps. The **setMultiplexerMode function** allows the EV3 to connect with both ultrasonic sensors in the same port while the **getMultiplexerValues function** allows the robot to read the distance values from the left and right ultrasonic sensors.



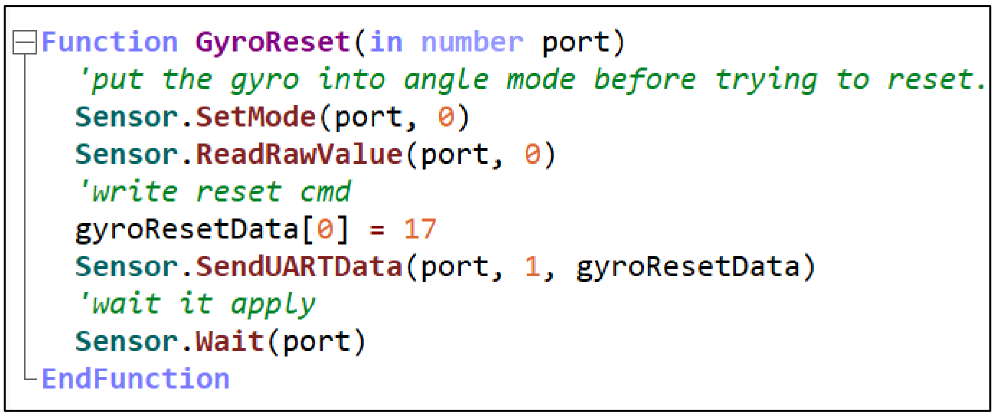
```
folder "prjs" "WRO2025"

Function setLampOn(in number port)
  Sensor.WriteI2CRegister(port,0,98,1)
EndFunction

Function setLampOff(in number port)
  Sensor.WriteI2CRegister(port,0,98,0)
EndFunction

Function getSignature(in number port, in number sig, out number x, out number y)
  address = 80 + sig
  values = Sensor.ReadI2CRegisters(port, 1, address, 3)
  x = values[1]
  y = values[2]
EndFunction

Function setMultiplexerMode(in number port, in number chanel, in number mode)
  address = 80 + 1 * (chanel - 1)
  Sensor.WriteI2CRegister(port,address,82,mode)
EndFunction

Function getMultiplexerValue(in number port, in number chanel, out number values)
  address = 80 + 1 * (chanel - 1)
  readData = Sensor.ReadI2CRegisters(port,address,84,2)
  values = readData[1] * 256 + readData[0]
EndFunction
```

Plus, each sensor such as **EV3 Gyro Sensor, EV3 Color Sensor, EV3 Ultrasonic Sensor** and **Pixy2 Camera** must be set up to their own port before the robot starts. **ReadSensor** continuously updates the **green (Signature 1)** and **red (Signature 2)** pillars positions on the camera views. The **magenta parking lot position (Signature 3)** also updated from time to time. Moreover, the robot also reads the **gyro angle** and **wall distance** in real time so that the robot will not off from its target.



```
Sub ReadSensor
  While "True"
    getSignature(1,1,greenx,greeny)
    getSignature(1,2,redx,redy)
    getSignature(1,3,pinkx,pinky)

    gyro = Sensor.ReadRawValue(2,0)
    relativeHeading = gyro - @target

    getMultiplexerValue(3,2,leftWall)
    getMultiplexerValue(3,3,rightWall)
    LCD.Clear()
    LCD.Text(1,0,40,2,leftWall)
    LCD.Text(1,0,60,2,rightWall)
  EndWhile
EndSub

Thread.Run = ReadSensor
```

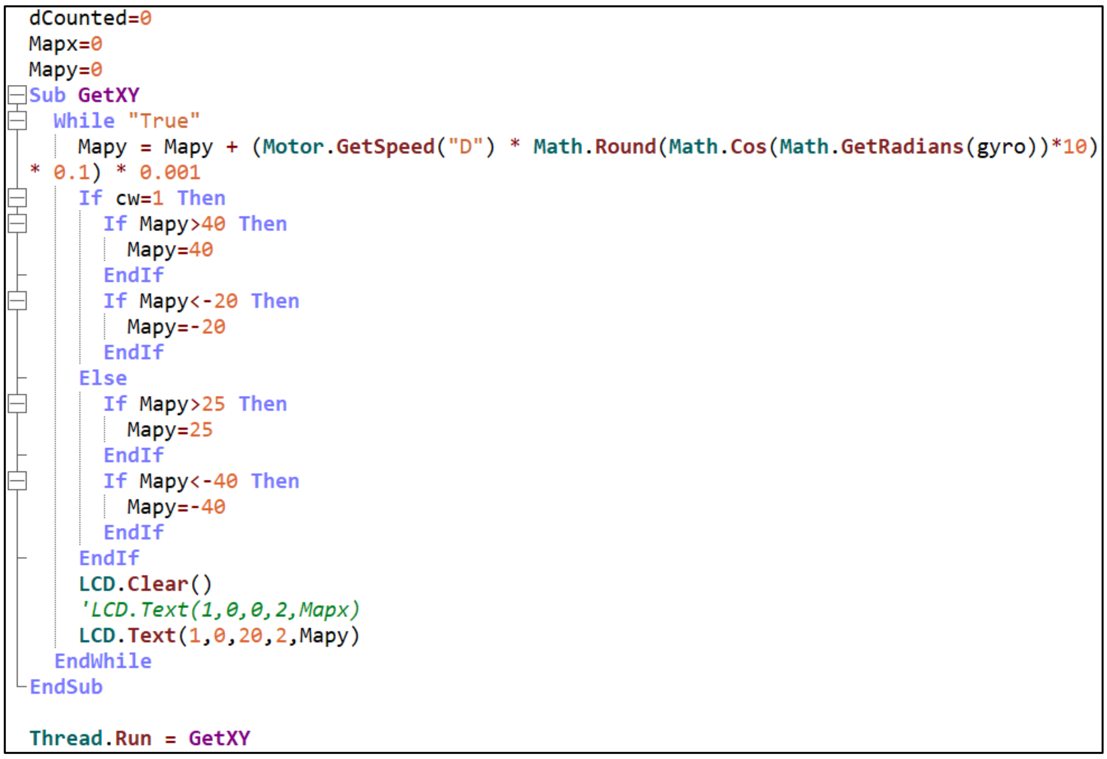
The **GyroReset** allows the robot to reset the gyro sensor so that the gyro will measure the angles from zero again. It is like pressing the “reset” button on the **EV3 Gyro Sensor** so that it starts measure the heading from 0 again. This is important because it avoids cumulative errors that might affect the robot.



```
Function GyroReset(in number port)
  'put the gyro into angle mode before trying to reset.
  Sensor.SetMode(port, 0)
  Sensor.ReadRawValue(port, 0)
  'write reset cmd
  gyroResetData[0] = 17
  Sensor.SendUARTData(port, 1, gyroResetData)
  'wait it apply
  Sensor.Wait(port)
EndFunction
```

(b) Car Position Coordinate Tracking

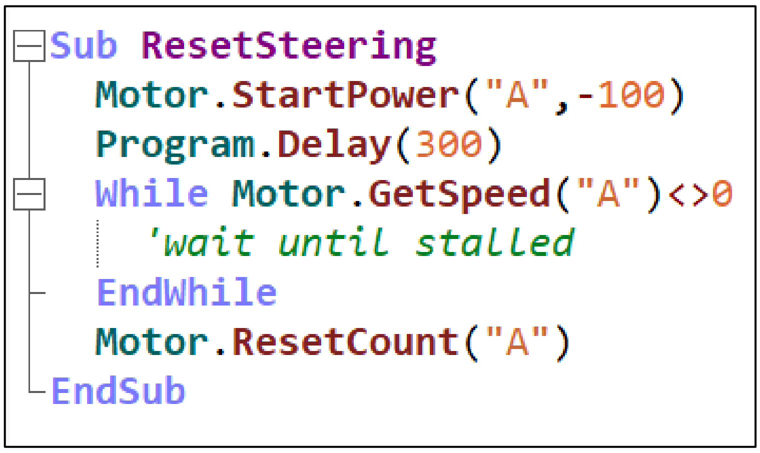
This part helps to **track the robot’s position** by calculating its movement along the **Y-axis.** The robot estimates its position based on two main factors which is the **speed of the drive motor** and **the angle measured by the gyro sensor**. By applying the cosine of the gyro angle, the program determines how much of the robot’s movement contributes to vertical travel, which is the y-axis and the value will be **stored in MapY.** To maintain accuracy, the program continuously recalculates MapY in real time as the robot moves. The updated value is displayed on the LCD screen, allowing users to monitor the robot’s position throughout the run. To avoid unrealistic or drifting values, MapY is also **restricted within a specific range.** When the robot is in a clockwise direction, MapY is allowed to vary between –20 and 40. In contrast, during a counter-clockwise run, the allowed range is shifted to between –40 and 25.



```
dCounted=0
Mapx=0
Mapy=0
Sub GetXY
  While "True"
    Mapy = Mapy + (Motor.GetSpeed("D") * Math.Round(Math.Cos(Math.GetRadians(gyro))*10)
* 0.1) * 0.001
    If cw=1 Then
      If Mapy>40 Then
        Mapy=40
      EndIf
      If Mapy<-20 Then
        Mapy=-20
      EndIf
    Else
      If Mapy>25 Then
        Mapy=25
      EndIf
      If Mapy<-40 Then
        Mapy=-40
      EndIf
    EndIf
    LCD.Clear()
    'LCD.Text(1,0,0,2,Mapx)
    LCD.Text(1,0,20,2,Mapy)
  EndWhile
EndSub

Thread.Run = GetXY
```

(c) Reset Steering

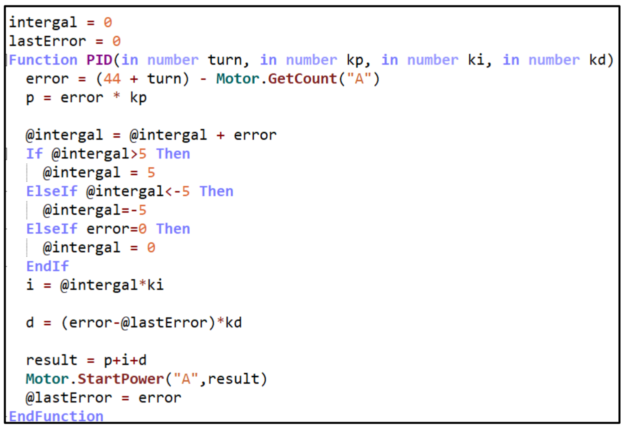
The **ResetSteering** helps to reset the steering motor before the robot starts running. It will turn the steering wheel all the way to one side and let the robot know the starting point. This is important because it will allow the robot steering always aligned in middle before it starts to move.



```
Sub ResetSteering
   Motor.StartPower("A",-100)
   Program.Delay(300)
   While Motor.GetSpeed("A")<>0
     'wait until stalled
   EndWhile
   Motor.ResetCount("A")
EndSub
```

(d) PID Steering Control

The **PID Steering function** helps the robot to adjust the steering motor whenever it starts to drift off from the target. This function is crucial in the **Obstacle Challenge** because it allows the robot to maintain in the straight heading after avoiding the pillars. The **P** part fixes most of the error right away, the **I** part makes small adjustments over time and the **D** part make sure that the correction is done smoothly. These three adjustments are added together to get one correct value, which is sent to motor A to turn the steering. This keeps the robot moving smoothly, avoiding zig-zagging, and makes its turns more accurate.



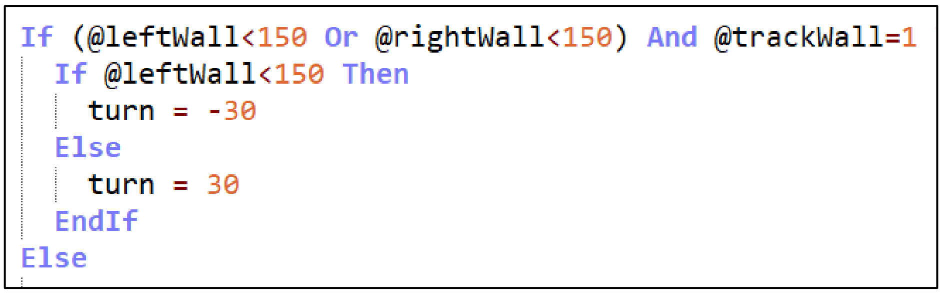
```
intergal = 0
lastError = 0
Function PID(in number turn, in number kp, in number ki, in number kd)
  error = (44 + turn) - Motor.GetCount("A")
  p = error * kp

  @intergal = @intergal + error
  If @intergal>5 Then
    @intergal = 5
  ElseIf @intergal<-5 Then
    @intergal=-5
  ElseIf error=0 Then
    @intergal = 0
  EndIf
  i = @intergal*ki

  d = (error-@lastError)*kd

  result = p+i+d
  Motor.StartPower("A",result)
  @lastError = error
EndFunction
```

(e) Drive

The **Drive** function combined four main parts which including the **TrackWall**, **TrackPillar** and **PID Steering**. In sum, these values help the robot constantly adjust its steering and make sure the robot move smoothly and accurately.

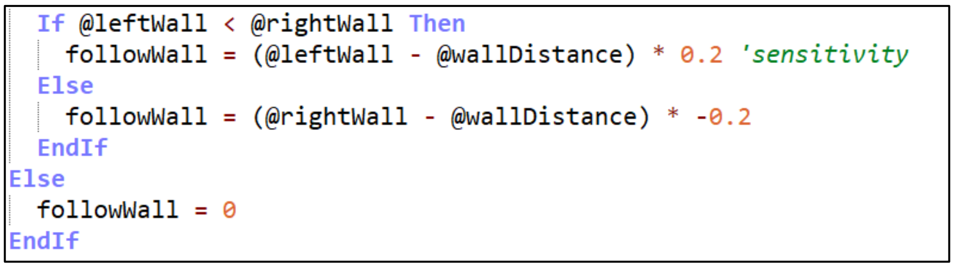
(i) TrackWall

The robot checks its **distances from the walls** within the range of 150mm. If it’s too close to one of the sides, it will steer away. Plus, the robot will **check which wall is closer by comparing the distance values from both sides**. If the left wall is closer, the robot will adjust its steering to follow the left side. If the right wall is closer, it will follow the right side instead, but with a negative steering value to turn in the opposite direction.



```
If (@leftWall<150 Or @rightWall<150) And @trackWall=1
  If @leftWall<150 Then
    turn = -30
  Else
    turn = 30
  EndIf
Else
```

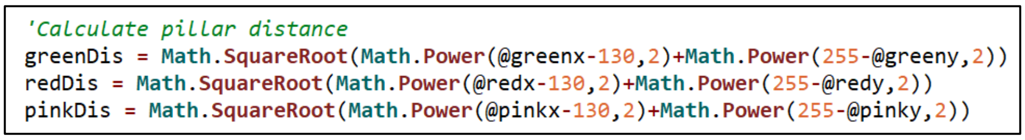
If **neither wall is detected** within the set range, the robot will not make any wall-following adjustments. This allows it to **continue moving straight** without unnecessary corrections.



```
  If @leftWall < @rightWall Then
    followWall = (@leftWall - @wallDistance) * 0.2 'sensitivity
  Else
    followWall = (@rightWall - @wallDistance) * -0.2
  EndIf
Else
  followWall = 0
EndIf
```

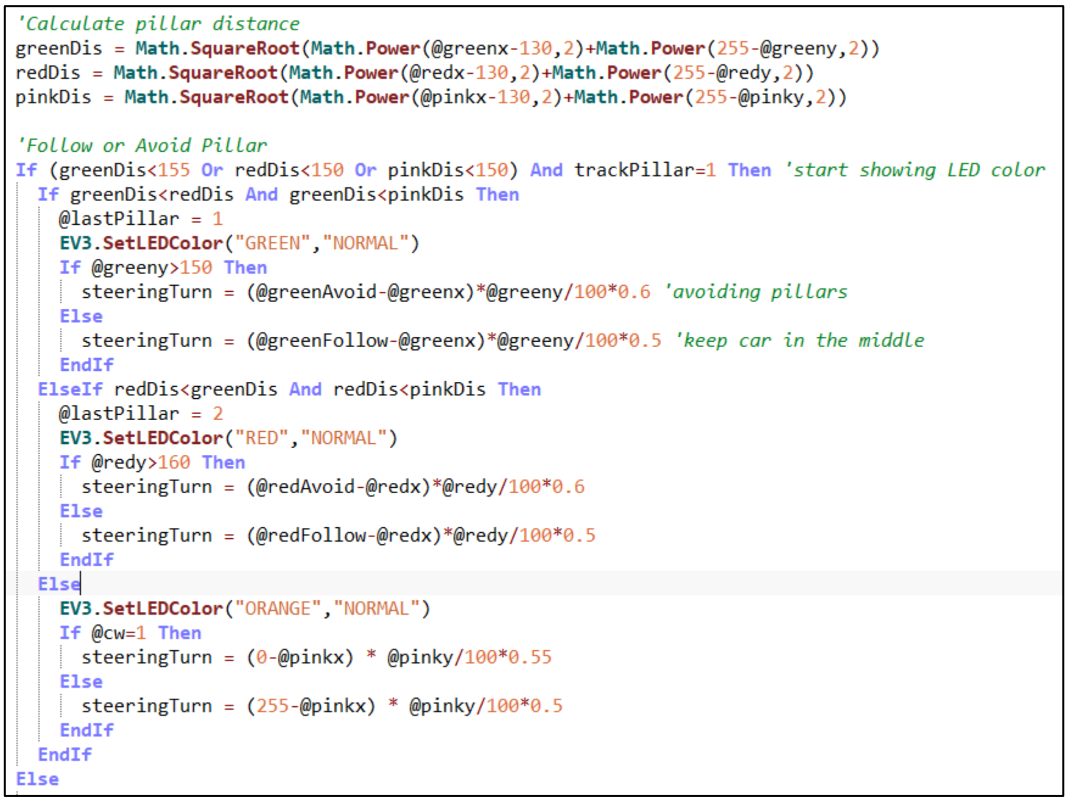
(ii) TrackPillar

This part helps to calculate how far each pillar from the **Pixy2 Camera's view.** It uses the **Pythagorean theorem** to figure out the straight-line distance based on the difference in horizontal (x) and vertical (y) positions.



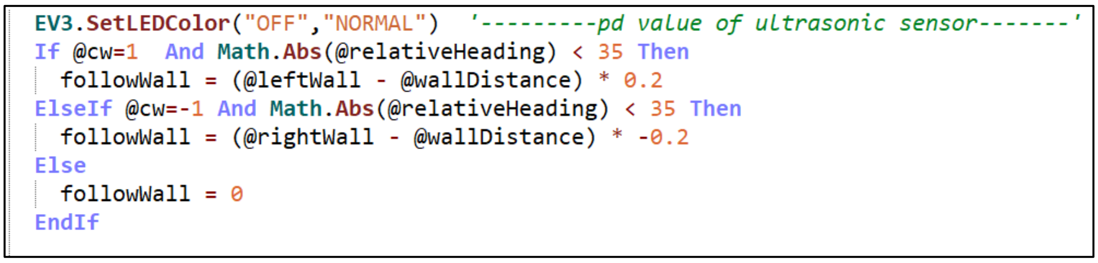
```
'Calculate pillar distance
greenDis = Math.SquareRoot(Math.Power(@greenx-130,2)+Math.Power(255-@greeny,2))
redDis = Math.SquareRoot(Math.Power(@redx-130,2)+Math.Power(255-@redy,2))
pinkDis = Math.SquareRoot(Math.Power(@pinkx-130,2)+Math.Power(255-@pinky,2))
```

This part helps the robot to check the presence of **red, green pillars** and **magenta** parking lot. If the pillars are near to the robot, the robot will start reacting by showing the LED colours. If the **green pillar is the closest**, it sets **@lastPillar = 1** and lights the **green LED**. Additionally, the y-coordinate (150) allows the robot to know the distance of the pillar from the Pixy2 Camera's view. However, if the **red pillar** is the closest, it sets **@lastPillar = 2** and light **red LED**. The steering will start to do the adjustment once the red pillars is located at the y-coordinate value (160) from the Pixy2 Camera's view. If the **magenta parking lot** is near, the **orange LED** lights up. The robot will steer to the **right** if it is moving clockwise, or to the **left** if it is moving counter clockwise to avoid hitting the magenta parking lot.



```
'Calculate pillar distance
greenDis = Math.SquareRoot(Math.Power(@greenx-130,2)+Math.Power(255-@greeny,2))
redDis = Math.SquareRoot(Math.Power(@redx-130,2)+Math.Power(255-@redy,2))
pinkDis = Math.SquareRoot(Math.Power(@pinkx-130,2)+Math.Power(255-@pinky,2))

'Follow or Avoid Pillar
If (greenDis<155 Or redDis<150 Or pinkDis<150) And trackPillar=1 Then 'start showing LED color
  If greenDis<redDis And greenDis<pinkDis Then
    @lastPillar = 1
    EV3.SetLEDColor("GREEN","NORMAL")
    If @greeny>150 Then
      steeringTurn = (@greenAvoid-@greenx)*@greeny/100*0.6 'avoiding pillars
    Else
      steeringTurn = (@greenFollow-@greenx)*@greeny/100*0.5 'keep car in the middle
    EndIf
  ElseIf redDis<greenDis And redDis<pinkDis Then
    @lastPillar = 2
    EV3.SetLEDColor("RED","NORMAL")
    If @redy>160 Then
      steeringTurn = (@redAvoid-@redx)*@redy/100*0.6
    Else
      steeringTurn = (@redFollow-@redx)*@redy/100*0.5
    EndIf
  Else
    EV3.SetLEDColor("ORANGE","NORMAL")
    If @cw=1 Then
      steeringTurn = (0-@pinkx) * @pinky/100*0.55
    Else
      steeringTurn = (255-@pinkx) * @pinky/100*0.5
    EndIf
  EndIf
Else
```

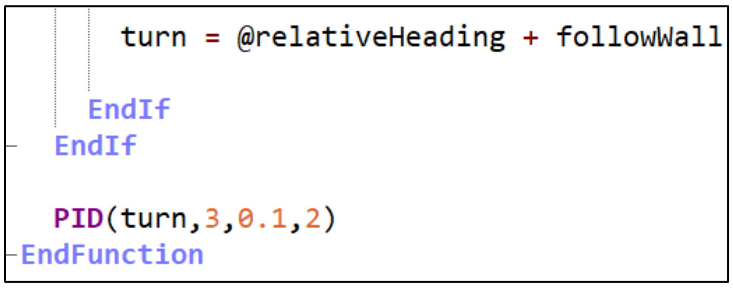
When no pillar is nearby, the robot turns off its **LED light** and uses the **ultrasonic sensors** to follow the wall. If it is moving clockwise, it will check the left wall. Conversely, if it is moving counter clockwise, it will check the right wall. If it is too far off its intended direction (relative heading), it will not make any wall adjustments to avoid over-correcting.



```
EV3.SetLEDColor("OFF","NORMAL")  '---------pd value of ultrasonic sensor-------'
If @cw=1  And Math.Abs(@relativeHeading) < 35 Then
  followWall = (@leftWall - @wallDistance) * 0.2
ElseIf @cw=-1 And Math.Abs(@relativeHeading) < 35 Then
  followWall = (@rightWall - @wallDistance) * -0.2
Else
  followWall = 0
EndIf
```

(iii) Steering Error

The robot’s steering correction is based on a combination of **pillar tracking and wall tracking**. The robot then applies this correction to the **PID steering motor**. At the same time, it powers the **drive motor** which is the Motor D to move forward at **runSpeed.**



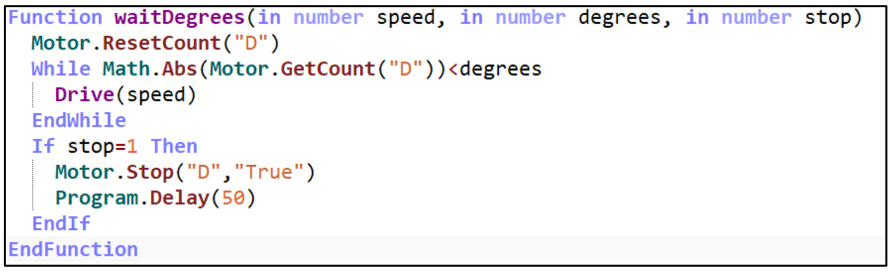
```
      turn = @relativeHeading + followWall

    EndIf
  EndIf

  PID(turn,3,0.1,2)
EndFunction
```

(f) Others movement block

Below shows the others movement block that help the robot to perform better.

(i) WaitDegrees

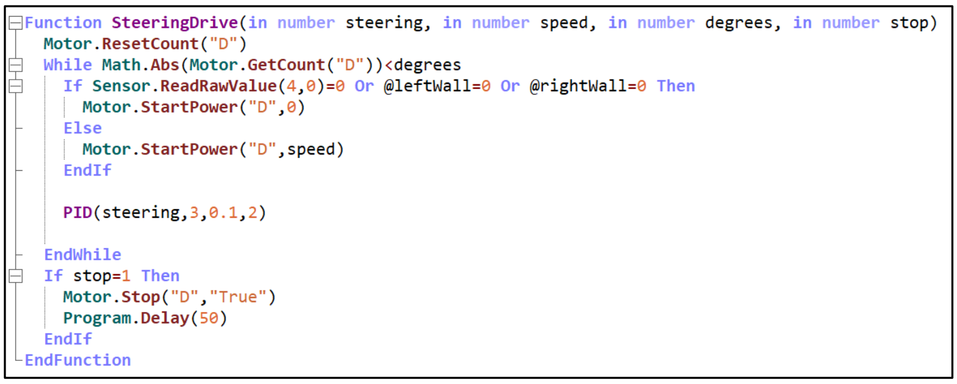
The **WaitDegrees** function makes the robot to move a certain distance based on the rotation of its drive motor **(Motor D**). The stop=1 means the robot will stops the motor and waits for a 50ms after finishing the movement.



```
Function waitDegrees(in number speed, in number degrees, in number stop)
  Motor.ResetCount("D")
  While Math.Abs(Motor.GetCount("D"))<degrees
    Drive(speed)
  EndWhile
  If stop=1 Then
    Motor.Stop("D","True")
    Program.Delay(50)
  EndIf
EndFunction
```

(ii) SteeringDrive

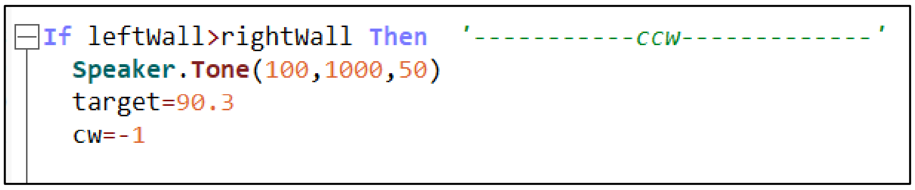
The **SteeringDrive** function makes the robot move forward a certain distance while controlling the steering. Therefore, it will combine the current steering motor position with any additional steering adjustment during both of the challenge rounds.



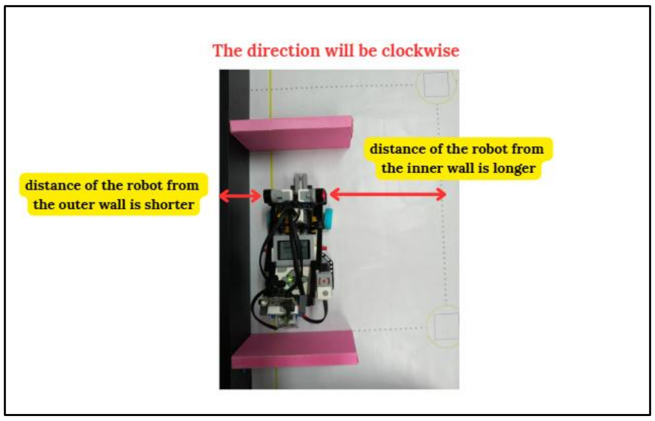
```
Function SteeringDrive(in number steering, in number speed, in number degrees, in number stop)
  Motor.ResetCount("D")
  While Math.Abs(Motor.GetCount("D"))<degrees
    If Sensor.ReadRawValue(4,0)=0 Or @leftWall=0 Or @rightWall=0 Then
      Motor.StartPower("D",0)
    Else
      Motor.StartPower("D",speed)
    EndIf

    PID(steering,3,0.1,2)

  EndWhile
  If stop=1 Then
    Motor.Stop("D","True")
    Program.Delay(50)
  EndIf
EndFunction
```

(g) Main program

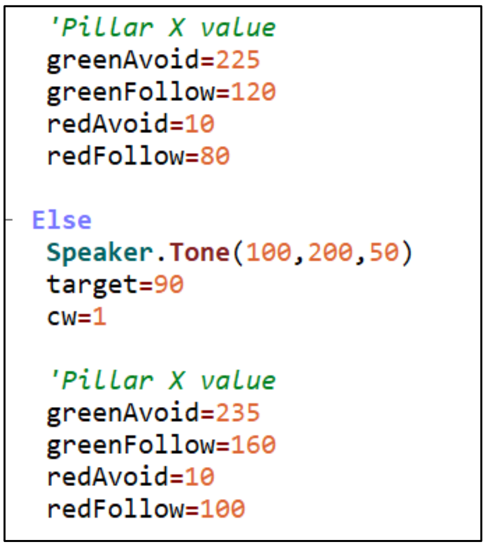
When the robot starts, it first decides whether it will move in a **counterclockwise** or **clockwise** direction based on the ultrasonic sensor readings. As shown in the diagram below, **if the left wall distance is greater than the right wall distance (leftWall>rightWall)**, the robot will know that it is counterclockwise. However, **if the right wall is greater than left wall (rightWall>leftWall),** the robot will know that it is in clockwise direction.



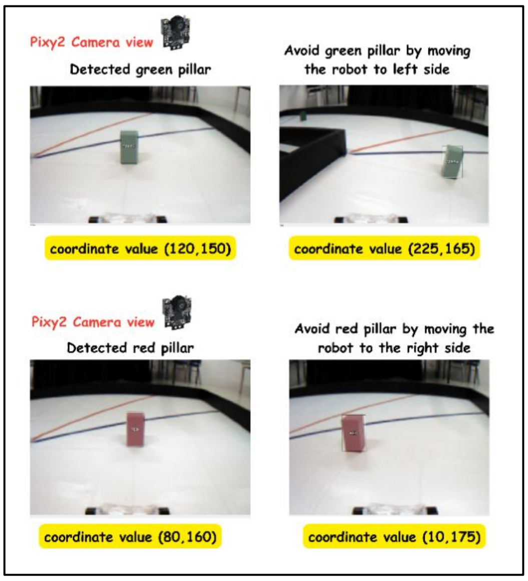
```
If leftWall>rightWall Then  '-----------CCW-------------'
  Speaker.Tone(100,1000,50)
  target=90.3
  cw=-1
```


The direction will be clockwise
distance of the robot from the outer wall is shorter
distance of the robot from the inner wall is longer

Next, the pillar X value tells the robot where the pillar should appear in the camera’s field of view. By using this value, the robot will avoid the pillar or adjust its path to stay centred when following it.

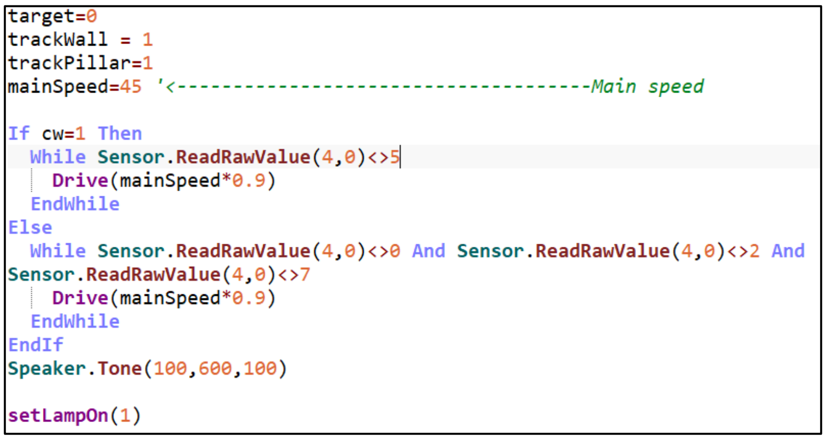


```
   'Pillar X value
   greenAvoid=225
   greenFollow=120
   redAvoid=10
   redFollow=80

 Else
   Speaker.Tone(100,200,50)
   target=90
   cw=1

   'Pillar X value
   greenAvoid=235
   greenFollow=160
   redAvoid=10
   redFollow=100
```



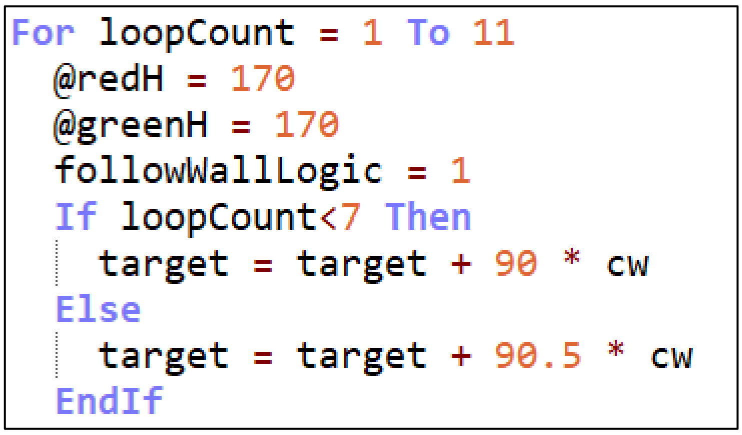
Pixy2 Camera view
Detected green pillar
coordinate value (120,150)
Avoid green pillar by moving the robot to left side
coordinate value (225,165)
Pixy2 Camera view
Detected red pillar
coordinate value (80,160)
Avoid red pillar by moving the robot to the right side
coordinate value (10,175)

After exiting the parking lot, the robot will drive forward at a speed of 45 until its colour sensor detects either a blue line or an orange line on the ground. Upon detecting a target line, the robot will immediately play a sound to indicate that the line has been found.



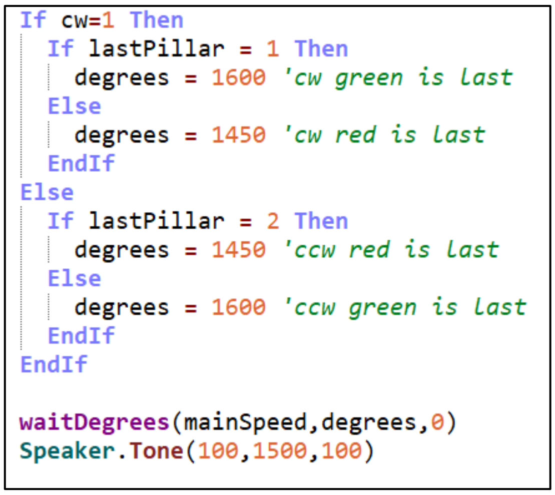
```
target=0
trackWall = 1
trackPillar=1
mainSpeed=45 '<-------------------------------------Main speed

If cw=1 Then
  While Sensor.ReadRawValue(4,0)<>5
    Drive(mainSpeed*0.9)
  EndWhile
Else
  While Sensor.ReadRawValue(4,0)<>0 And Sensor.ReadRawValue(4,0)<>2 And
Sensor.ReadRawValue(4,0)<>7
    Drive(mainSpeed*0.9)
  EndWhile
EndIf
Speaker.Tone(100,600,100)

setLampOn(1)
```

During the Obstacle Round, the robot adjusts its steering by correcting any error measured from three sources which include **TrackPillar** and **TrackWall**. The redH and greenH refer to the distances of pillar from the camera view. It allows the robot to know when to do the tracking of the pillars. The steering target is slightly different depending on whether the robot is moving clockwise or counterclockwise so a different adjustment value is applied for each direction. This steering correction process is repeated for 11 loops.



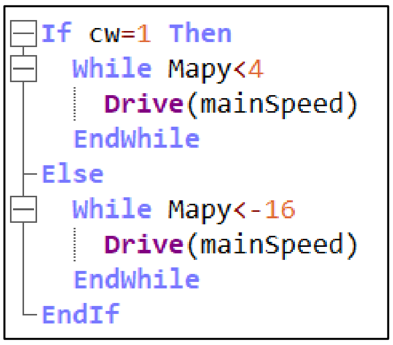
```
For loopCount = 1 To 11
  @redH = 170
  @greenH = 170
  followWallLogic = 1
  If loopCount<7 Then
    target = target + 90 * cw
  Else
    target = target + 90.5 * cw
  EndIf
```

The below part of the program ensures the robot does not overturn and end up facing the wrong direction while avoiding a pillar. When the robot avoids a pillar on its left side, which mean it passes a green pillar in a clockwise run or a red pillar in a counterclockwise run. It needs to turn a greater number of degrees. This extra turning ensures it does not accidentally scan the wrong-coloured line, which could cause it to head in the wrong direction.



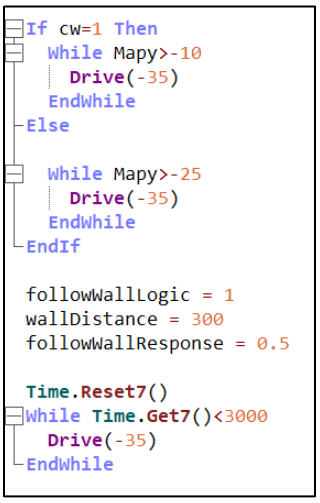
```
If cw=1 Then
  If lastPillar = 1 Then
    degrees = 1600 'cw green is last
  Else
    degrees = 1450 'cw red is last
  EndIf
Else
  If lastPillar = 2 Then
    degrees = 1450 'ccw red is last
  Else
    degrees = 1600 'ccw green is last
  EndIf
EndIf

waitDegrees(mainSpeed,degrees,0)
Speaker.Tone(100,1500,100)
```

The program below indicates the robot had completed 11 loops. If the robot is moving in **clockwise direction** (cw=1), it will continue driving forward as long as its MapY value is less than **4.** This means the robot will keep moving until it reaches the target Y-coordinate, ensuring it get into the last section completely before proceed to the reverse movement. Conversely, when the robot is moving in a **counter-clockwise direction (cw= -1),** the target range is different. In this case, the robot will continue driving forward while MapY is still less than **–16.** Ultimately, this part of the program ensures the robot get into its initial position in the last section to make sure the round is completed.



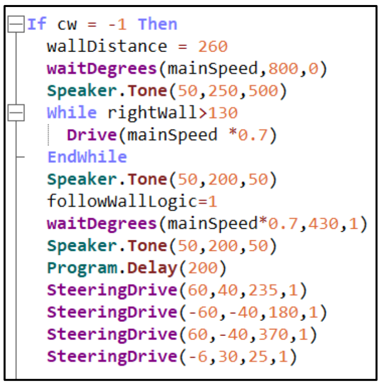
```
If cw=1 Then
  While Mapy<4
    Drive(mainSpeed)
  EndWhile
Else
  While Mapy<-16
    Drive(mainSpeed)
  EndWhile
EndIf
```

The below part of the program makes the robot **reverse until it reaches the correct position**. When the robot is **moving clockwise**, it keeps reversing until its y-coordinate becomes **less than –10.** When it is moving **counterclockwise**, it reverses further until the **y-coordinate reaches –25.** This helps the robot gently touch the outer wall so it can straighten itself and follow the wall more easily during the parking phase. After that, the program **turns on wall-following mode**, sets the wall distance to 30 cm, and uses a response value of 0.5 to help the robot correct its steering whenever the distance changes. Finally, the robot continues reversing for 3 seconds to make sure it is fully in position before starting its final parking movement.

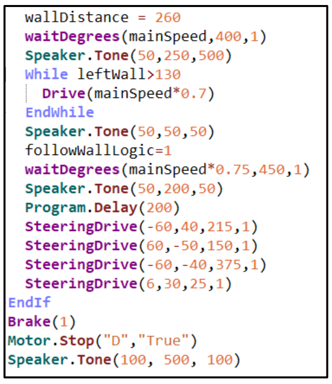


```
If cw=1 Then
  While Mapy>-10
    Drive(-35)
  EndWhile
Else

  While Mapy>-25
    Drive(-35)
  EndWhile
EndIf

followWallLogic = 1
wallDistance = 300
followWallResponse = 0.5

Time.Reset7()
While Time.Get7()<3000
  Drive(-35)
EndWhile
```

Both of the picture below shows the detailed program **during the side parking phase**. The robot will move forward while maintaining a fixed distance from the parking lot which is **260 mm** for both directions. As it moves, it continuously checks this distance. If the distance goes out of range, the robot will **adjust its steering to bring it back to the correct value**. Once it passes the first magenta parking lot, the robot will reverse into the parking space and ensure it is parked parallel to the parking lot.

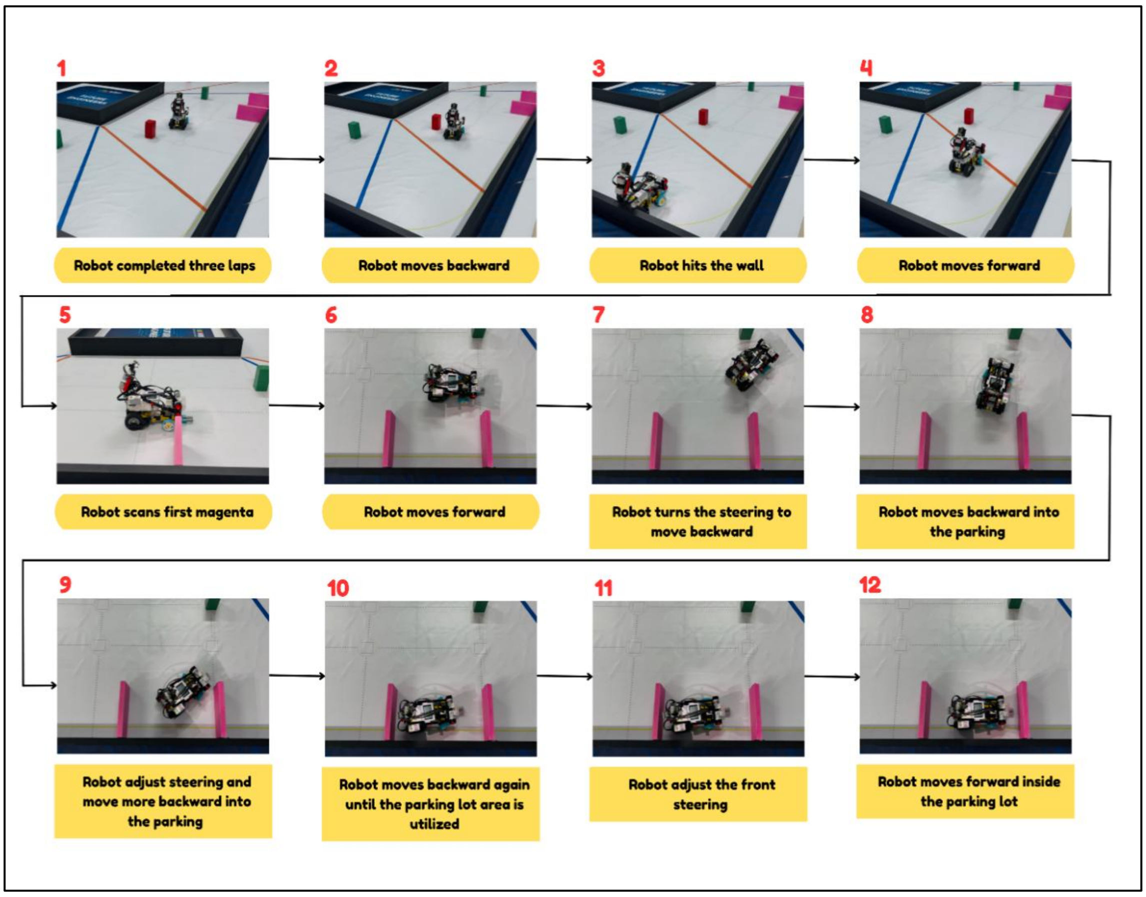


```
If cw = -1 Then
  wallDistance = 260
  waitDegrees(mainSpeed,800,0)
  Speaker.Tone(50,250,500)
  While rightWall>130
    Drive(mainSpeed *0.7)
  EndWhile
  Speaker.Tone(50,200,50)
  followWallLogic=1
  waitDegrees(mainSpeed*0.7,430,1)
  Speaker.Tone(50,200,50)
  Program.Delay(200)
  SteeringDrive(60,40,235,1)
  SteeringDrive(-60,-40,180,1)
  SteeringDrive(60,-40,370,1)
  SteeringDrive(-6,30,25,1)
```



```
  wallDistance = 260
  waitDegrees(mainSpeed,400,1)
  Speaker.Tone(50,250,500)
  While leftWall>130
    Drive(mainSpeed*0.7)
  EndWhile
  Speaker.Tone(50,50,50)
  followWallLogic=1
  waitDegrees(mainSpeed*0.75,450,1)
  Speaker.Tone(50,200,50)
  Program.Delay(200)
  SteeringDrive(-60,40,215,1)
  SteeringDrive(60,-50,150,1)
  SteeringDrive(-60,-40,375,1)
  SteeringDrive(6,30,25,1)
EndIf
Brake(1)
Motor.Stop("D","True")
Speaker.Tone(100, 500, 100)
```

The flow diagram below illustrates the detailed process of side parking.


1
Robot completed three laps
2
Robot moves backward
3
Robot hits the wall
4
Robot moves forward
5
Robot scans first magenta
6
Robot moves forward
7
Robot turns the steering to move backward
8
Robot moves backward into the parking
9
Robot adjust steering and move more backward into the parking
10
Robot moves backward again until the parking lot area is utilized
11
Robot adjust the front steering
12
Robot moves forward inside the parking lot