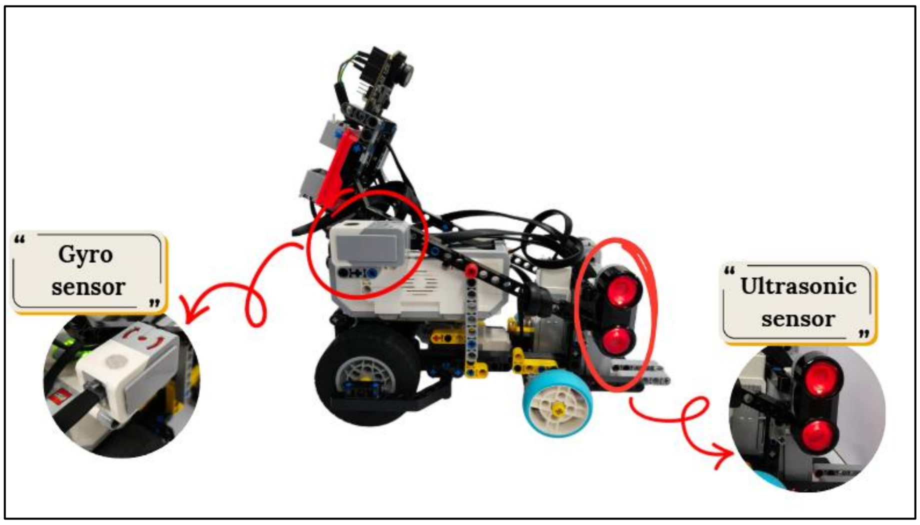
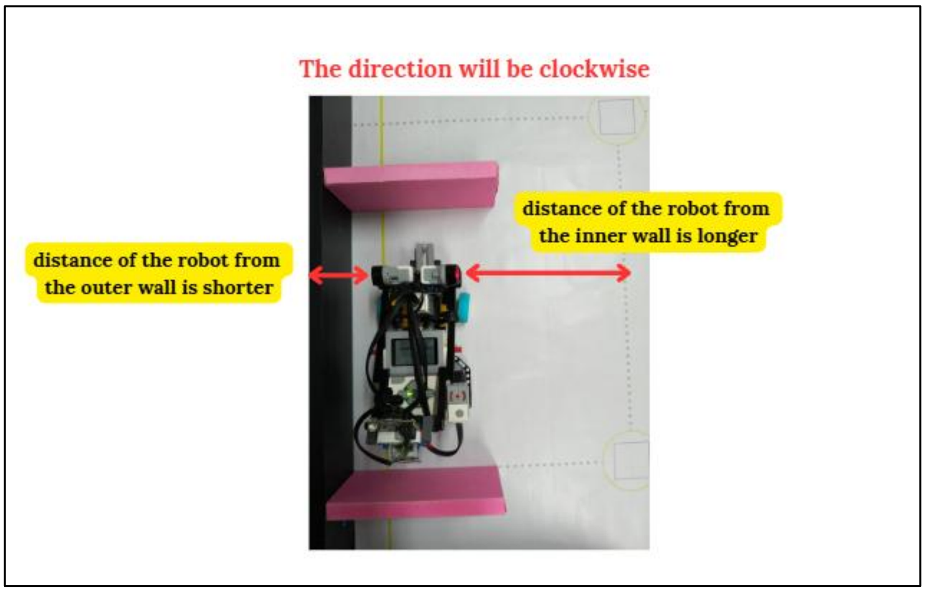
3.1 Open Challenge

To complete the open challenge, our robot uses two main sensors which is **an EV3 Gyro Sensor** and **two EV3 Ultrasonic Sensors** on both the left and right sides. Since both **EV3 Ultrasonic Sensors** share the same port, we use a **multiplexer** to allow the robot to read data from each one separately. The left ultrasonic sensor is connected to Channel 2 of the multiplexer, while the right ultrasonic sensor is connected to Channel 3.

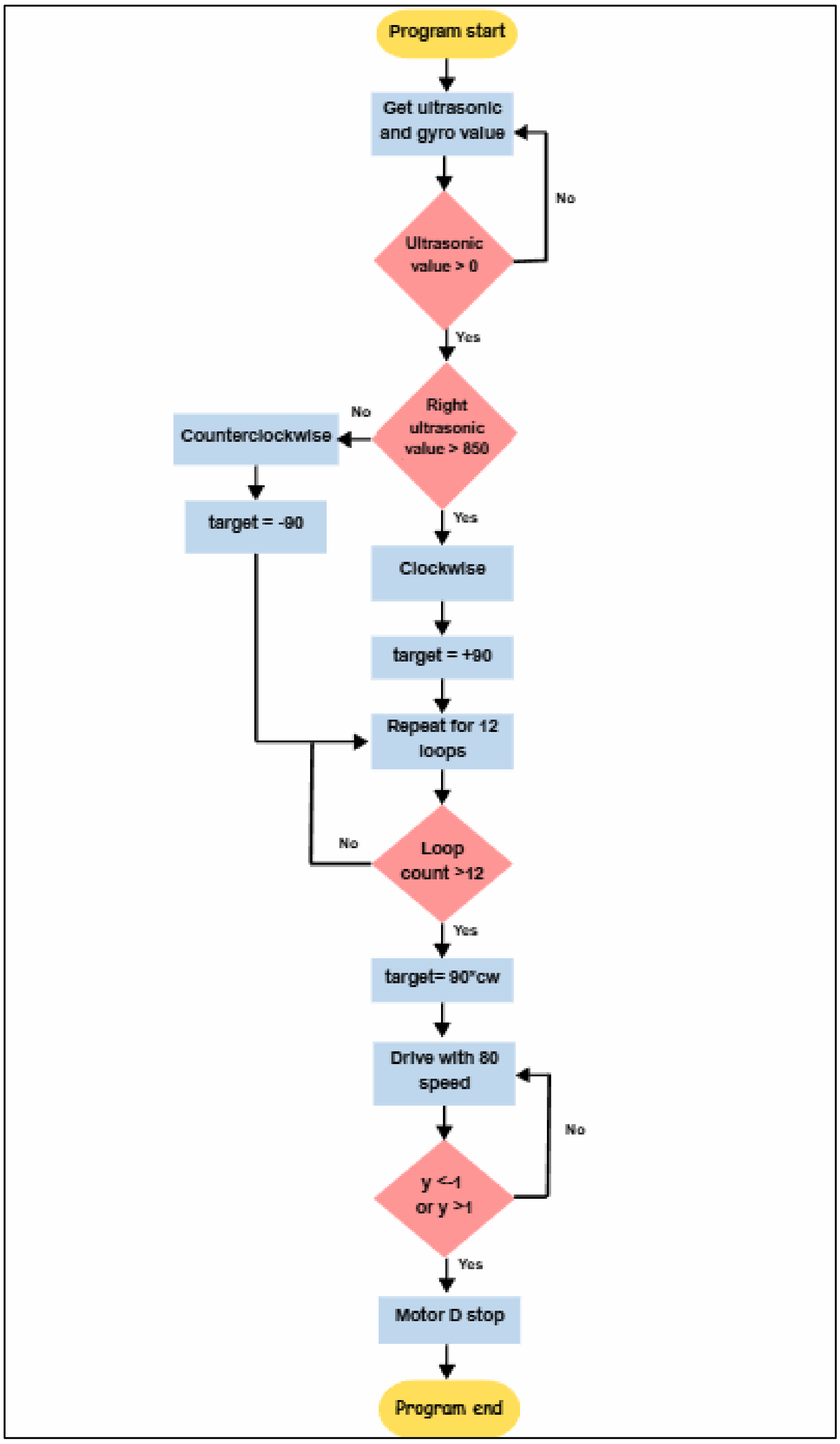


The **EV3 Ultrasonic Sensors** helps the car to measure the distance between the robot and the walls. By comparing readings from the left and right sensors, the robot can determine whether it should move **clockwise** or **counter clockwise**. These readings also help the robot adjust its steering to avoid collisions and stay at a safe distance from inner and the outer walls. Additionally, the **EV3 Gyro Sensors** monitor the robot's heading, ensuring it maintains a straight path and performs accurate turns at the correct angle when turning.


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

In a quick overview, when the robot starts, it **first checks and sets up all its sensors**. Next, it **resets its steering** before moving. The robot then uses a **PID control system** to ensure smooth and stable turns. A drive function controls its speed and distance based on sensor data. Finally, the **main program** combines all these functions, enabling the robot to **follow walls**, **make precise turns**, and **complete all laps** efficiently and accurately. Further explanations of each function and programming part are provided in the **3.1.1 section below**.

Diagram below shows the flowchart for Open Challenge:



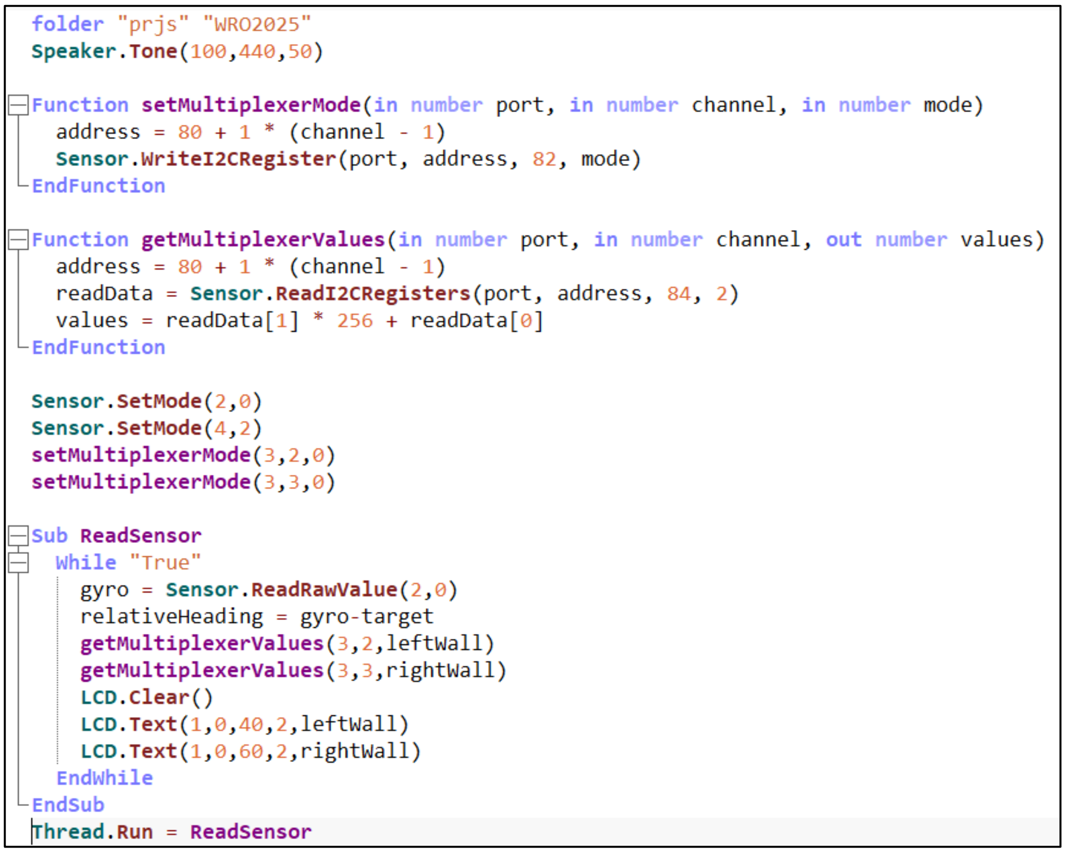
Appendix 2 contains the complete programming code for the **Open Challenge.**

3.1.1 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 Setup

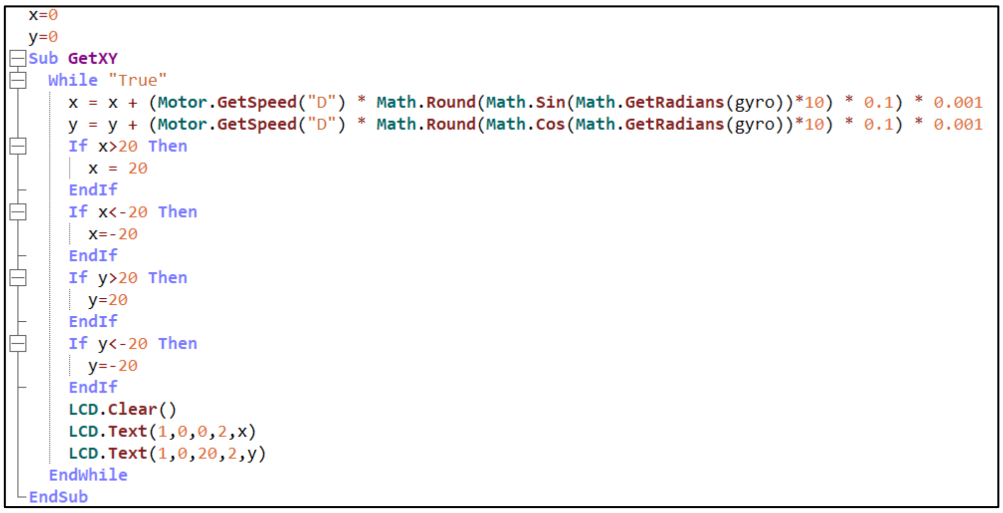
This part is aim to set up the sensors and multiplexer before running. The **setMultiplexerMode function** allows the EV3 to connect with both ultrasonic sensors in the same port while the **getMultiplexerValues function** allows the robot to read the distance values from the left and right ultrasonic sensors. Additionally, the **ReadSensor** continuously updates the robot’s heading and wall distances in real time.



```
folder "prjs" "WRO2025"
Speaker.Tone(100,440,50)

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

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

Sensor.SetMode(2,0)
Sensor.SetMode(4,2)
setMultiplexerMode(3,2,0)
setMultiplexerMode(3,3,0)

Sub ReadSensor
  While "True"
    gyro = Sensor.ReadRawValue(2,0)
    relativeHeading = gyro-target
    getMultiplexerValues(3,2,leftWall)
    getMultiplexerValues(3,3,rightWall)
    LCD.Clear()
    LCD.Text(1,0,40,2,leftWall)
    LCD.Text(1,0,60,2,rightWall)
  EndWhile
EndSub
Thread.Run = ReadSensor
```

(b) Car Position Coordinate Tracking

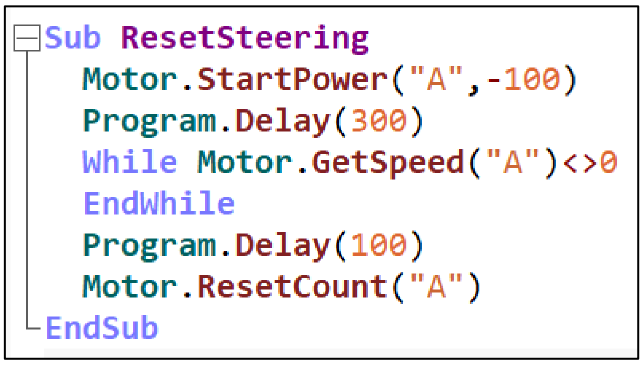
This part of the program is about **position tracking using a gyro sensor and motor speed**. It continuously calculates the robot’s position on the X and Y axis based on how fast the motor is spinning and the angle detected by the gyro sensor. The **Math.Sin** and **Math.Cos** functions are used to determine the direction of movement, converting the robot’s rotation into **horizontal (x) and vertical (y)** changes. The program keeps updating these values in real time, showing them on the LCD screen. To prevent errors, it also limits both the X and Y positions to a **range between -20 and 20**. In simple terms, this part allows the robot to “know” where it is moving by combining speed and direction data, then displays its current position on the screen.



```
x=0
y=0
Sub GetXY
  While "True"
    x = x + (Motor.GetSpeed("D") * Math.Round(Math.Sin(Math.GetRadians(gyro))*10) * 0.1) * 0.001
    y = y + (Motor.GetSpeed("D") * Math.Round(Math.Cos(Math.GetRadians(gyro))*10) * 0.1) * 0.001
    If x>20 Then
      x = 20
    EndIf
    If x<-20 Then
      x=-20
    EndIf
    If y>20 Then
      y=20
    EndIf
    If y<-20 Then
      y=-20
    EndIf
    LCD.Clear()
    LCD.Text(1,0,0,2,x)
    LCD.Text(1,0,20,2,y)
  EndWhile
EndSub
```

(c) Reset Steering

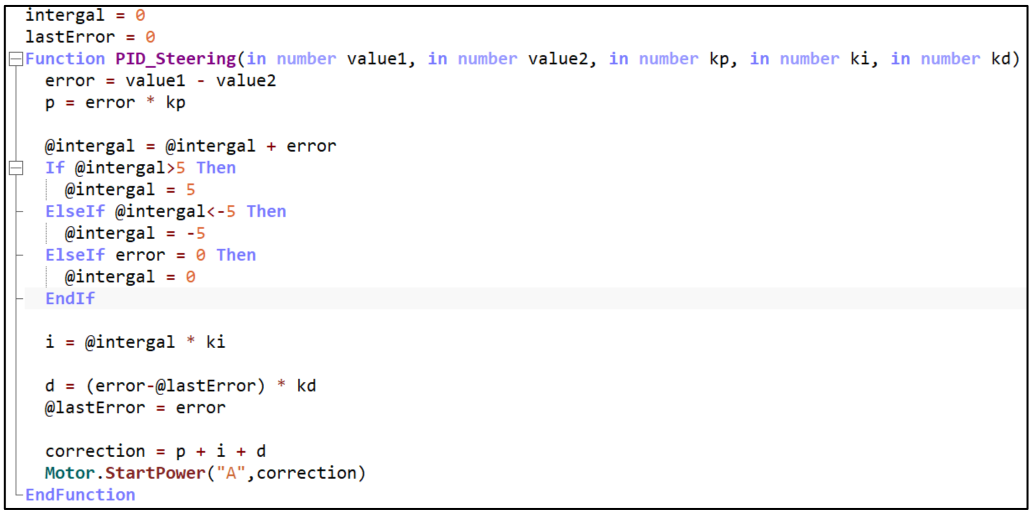
Reset Steering make sure the steering motor starts at a fixed desired position before driving. The robot turns the steering motor all the way to the left for **300 milliseconds** by using the **medium motor in the port A**. Hence, it waits until the motor completely stops moving, which means it has hit the limit. After that, the motor’s rotation is reset to zero. This step is important because it gives the information to the robot exactly where the “straight ahead” position is.



```
Sub ResetSteering
  Motor.StartPower("A",-100)
  Program.Delay(300)
  While Motor.GetSpeed("A")<>0
  EndWhile
  Program.Delay(100)
  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. 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 correction 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.

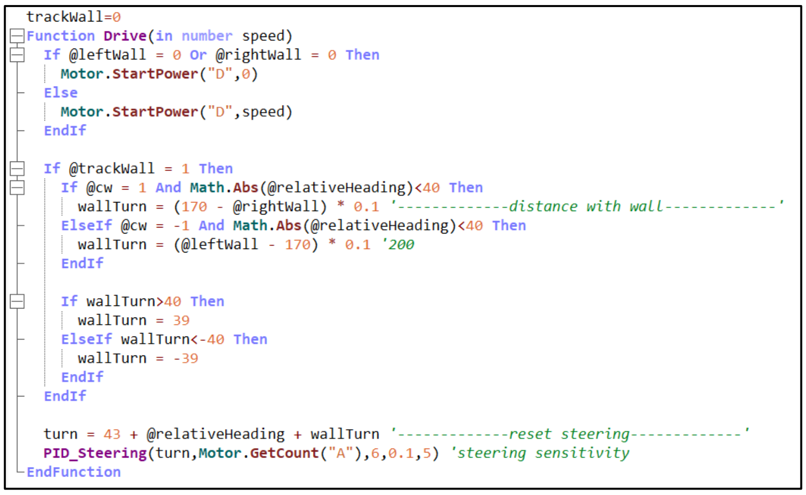


```
intergal = 0
lastError = 0
Function PID_Steering(in number value1, in number value2, in number kp, in number ki, in number kd)
  error = value1 - value2
  p = error * kp

  @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
  @lastError = error

  correction = p + i + d
  Motor.StartPower("A",correction)
EndFunction
```

(e) Drive

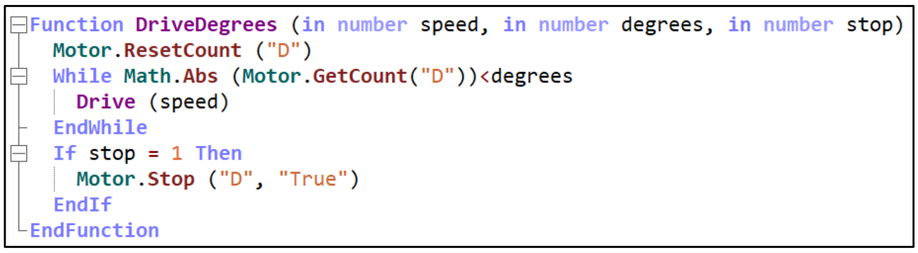
Function **Drive** controls how the robot **drives and follows a wall**. First, it checks whether the ultrasonic sensors on both sides are working. If either sensor is not detecting a wall (value = 0), the robot **stops moving**. This acts as a **safety feature**, ensuring the robot doesn't move blindly when there is no sensor data. If both sensors are working, the robot moves forward at the given speed.

Next, when wall tracking (trackWall) is turned on, the robot adjusts its direction to stay at the correct distance from the wall. If the robot is moving **clockwise**, it uses the right sensor; if moving **counterclockwise**, it uses the left sensor. It calculates how much the robot **needs to turn** to maintain the proper distance and making sure it's not too close or too far. If the error is **more than or less than 40** based on the robot's distance and the inner wall, it will perform a correction to the steering. Finally, a **PID steering control** ensures smooth and precise turns based on this calculation, keeping the robot aligned while following the wall safely.



```
trackWall=0
Function Drive(in number speed)
  If @leftWall = 0 Or @rightWall = 0 Then
    Motor.StartPower("D",0)
  Else
    Motor.StartPower("D",speed)
  EndIf

  If @trackWall = 1 Then
    If @cw = 1 And Math.Abs(@relativeHeading)<40 Then
      wallTurn = (170 - @rightWall) * 0.1 '-------------distance with wall-------------'
    ElseIf @cw = -1 And Math.Abs(@relativeHeading)<40 Then
      wallTurn = (@leftWall - 170) * 0.1 '200
    EndIf

    If wallTurn>40 Then
      wallTurn = 39
    ElseIf wallTurn<-40 Then
      wallTurn = -39
    EndIf
  EndIf

  turn = 43 + @relativeHeading + wallTurn '-------------reset steering-------------'
  PID_Steering(turn,Motor.GetCount("A"),6,0.1,5) 'steering sensitivity
EndFunction
```

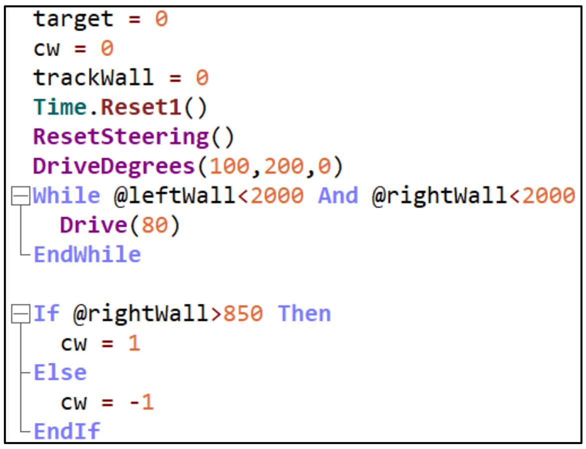
Moreover, the **DriveDegrees function** moves the robot forward in rotation degrees. He robot will keep driving at the given speed until the target degrees are reached, and stops the motor if stop is set to 1.



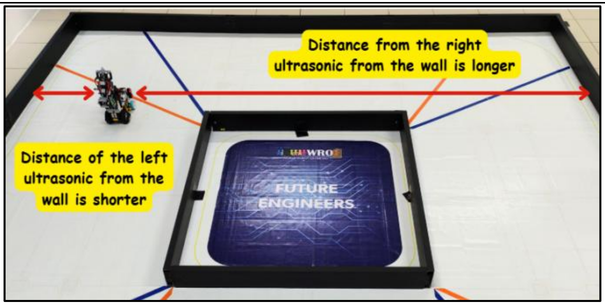
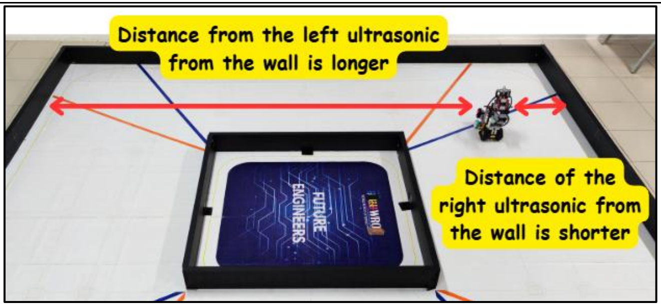
```
Function DriveDegrees (in number speed, in number degrees, in number stop)
  Motor.ResetCount ("D")
  While Math.Abs (Motor.GetCount("D"))<degrees
    Drive (speed)
  EndWhile
  If stop = 1 Then
    Motor.Stop ("D", "True")
  EndIf
EndFunction
```

(f) Main program

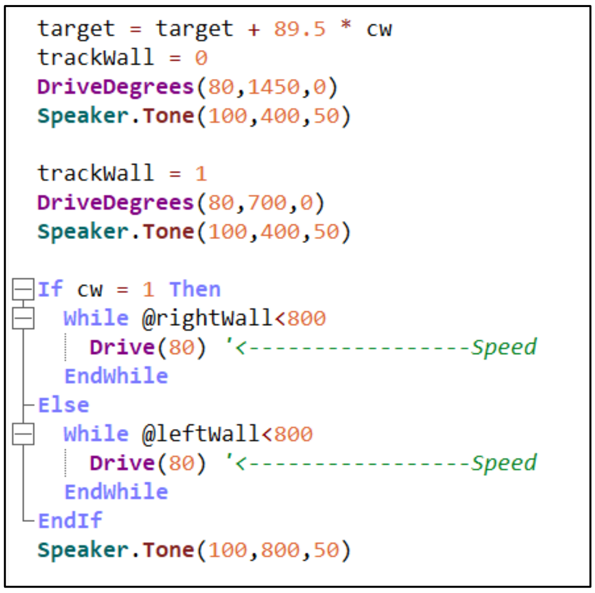
This program resets timers and steering, then drives forward until it detects a wall on either side. It determines whether to follow the right or left wall based on sensor readings.



```
target = 0
cw = 0
trackWall = 0
Time.Reset1()
ResetSteering()
DriveDegrees(100,200,0)
While @leftWall<2000 And @rightWall<2000
  Drive(80)
EndWhile

If @rightWall>850 Then
  cw = 1
Else
  cw = -1
EndIf
```

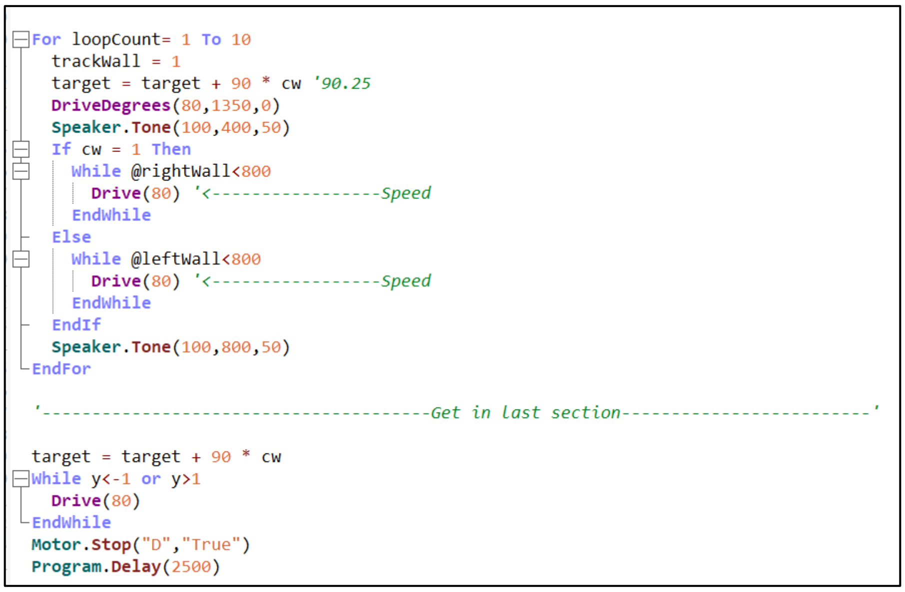
 Distance from the right ultrasonic from the wall is longer Distance of the left ultrasonic from the wall is shorter WRO FUTURE ENGINEERS	 Distance from the left ultrasonic from the wall is longer Distance of the right ultrasonic from the wall is shorter FUTURE ENGINEERS WRO
The robot will move in clockwise direction.	**The robot will move in counter clockwise direction.**

When the robot enters the **second section**, it does **not immediately start wall tracking**. Instead, it first drives a certain distance in degrees before activating wall tracking. This is done to **prevent sudden corrections**. If the robot starts tracking the wall when the sensor values are far from the desired distance, it could make a sharp turn and accidentally hit the wall. After driving the initial distance, **wall tracking (trackWall)** is turned on, and the robot begins adjusting its direction to follow the wall smoothly. While tracking, the robot moves at a **controlled speed of 80**, which ensures stable motion and reduces the risk of collisions as it approaches the wall.



```
target = target + 89.5 * cw
trackWall = 0
DriveDegrees(80,1450,0)
Speaker.Tone(100,400,50)

trackWall = 1
DriveDegrees(80,700,0)
Speaker.Tone(100,400,50)

If cw = 1 Then
  While @rightWall<800
    Drive(80) '<-----------------Speed
  EndWhile
Else
  While @leftWall<800
    Drive(80) '<-----------------Speed
  EndWhile
EndIf
Speaker.Tone(100,800,50)
```

Then, it loops the same program for **10 times, turning 90°** in the chosen direction in every loops. After completing the 10 loops, the robot enters the **last section**. It updates the target from time to time and drives forward while checking the **Y-coordinate** until it is roughly within the range -1 to 1, which represent that the robot will stop at the **(0,0) position.** Once there, the robot stops all motors and pauses for 2.5 seconds to complete the task.



```
For loopCount= 1 To 10
  trackWall = 1
  target = target + 90 * cw '90.25
  DriveDegrees(80,1350,0)
  Speaker.Tone(100,400,50)
  If cw = 1 Then
    While @rightWall<800
      Drive(80) '<-----------------Speed
    EndWhile
  Else
    While @leftWall<800
      Drive(80) '<-----------------Speed
    EndWhile
  EndIf
  Speaker.Tone(100,800,50)
EndFor

'---------------------------------------Get in last section------------------------'

target = target + 90 * cw
While y<-1 or y>1
  Drive(80)
EndWhile
Motor.Stop("D","True")
Program.Delay(2500)
```