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

1. **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" "NRC"

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.

```
Sensor.SetMode(2,0)
Sensor.SetMode(4,2)
setMultiplexerMode(3,1,0) 'leftside
setMultiplexerMode(3,3,0) 'rightside

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

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

    getMultiplexerValue(3,1,leftWall)
    getMultiplexerValue(3,3,rightWall)
  EndWhile
EndSub
```

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
```

2. 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
```

3. 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
```

4. 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 help 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.

```
'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
```

If the **green pillar is the closest**, it sets **@lastPillar = 1** and lights the **green LED**. Additionally, the y-coordinate (150) allow 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.

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
```

5. Others movement block

(i) WaitDegrees

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.

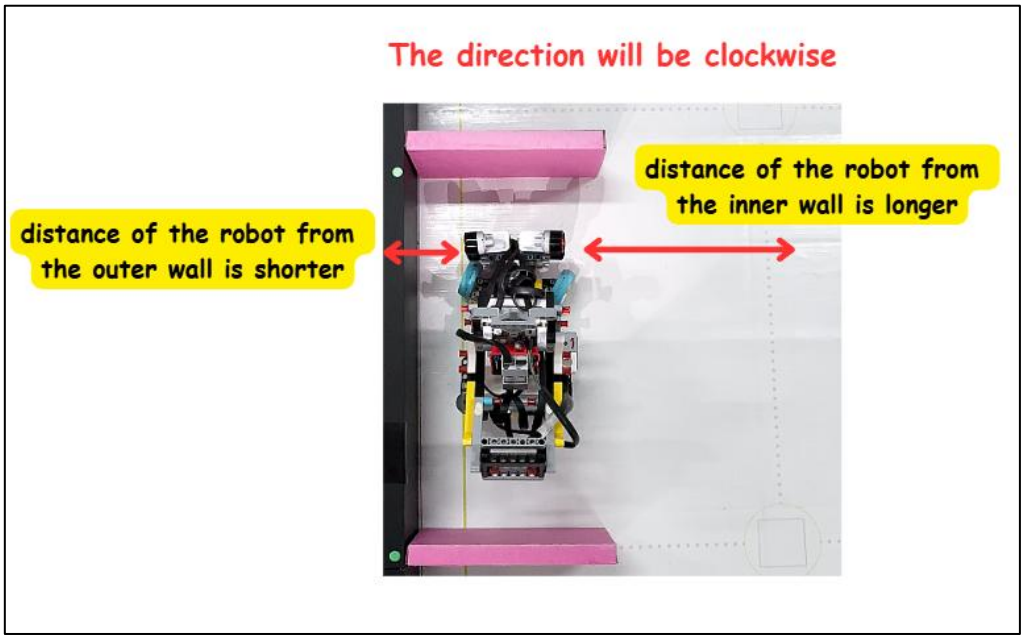
```
Function SteeringDrive(in number steering, in number speed, in number degrees, in number stop)
  Motor.ResetCount("D")
  While Math.Abs(Motor.GetCount("D"))<degrees
    If Sensor.ReadRawValue(4,0)=0 Or @leftWall=0 Or @rightWall=0 Then
      Motor.StartPower("D",0)
    Else
      Motor.StartPower("D",speed)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

    PID(steering,3,0.1,2)

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

6. 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 from the left and right sides.

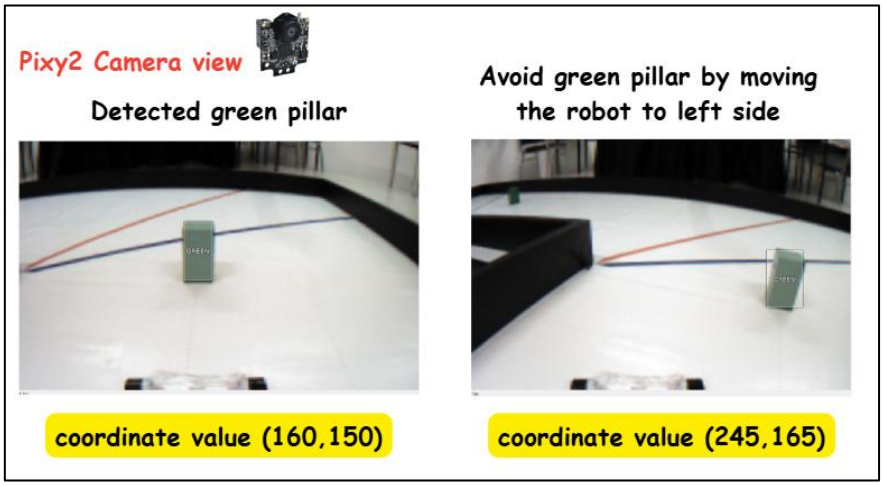
```
If leftWall>rightWall Then  '-----------ccw-------------'
   Speaker.Tone(100,1000,50)
   target=90.3
   cw=-1
```

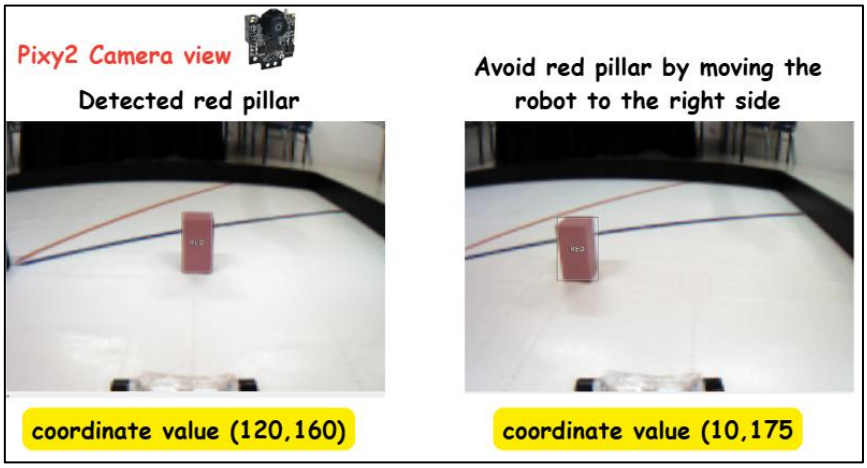
As shown in the diagram below, **if the left wall distance is greater than the right wall distance (leftWall>rightWall)**, the robot will know that it is counterclockwise. However, **if the right wall is greater than left wall (rightWall>leftWall),** the robot will know that it is in clockwise direction.



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=235
    greenFollow=135

    redAvoid=10
    redFollow=120

Else                          '--------------CW-------------'
    Speaker.Tone(100,200,50)
    target=-90.3
    CW=1

    'Pillar X value
    greenAvoid=245
    greenFollow=160

    redAvoid=10
    redFollow=145
EndIf
```





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
trackPillar=1
trackWall=1
mainSpeed=45
If cw=1 Then
  While Sensor.ReadRawValue(4,0)<>4 And Sensor.ReadRawValue(4,0)<>5
    Drive(mainSpeed)'scan map color & no pillar speed
  EndWhile
Else
  While Sensor.ReadRawValue(4,0)<>2
    Drive(mainSpeed)
  EndWhile
EndIf
Speaker.Tone(100,600,100)
```

During the Obstacle Round, the robot adjusts its steering by correcting any error measured from three sources which include **TrackPillar** and **TrackWall**. 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 12 loops.

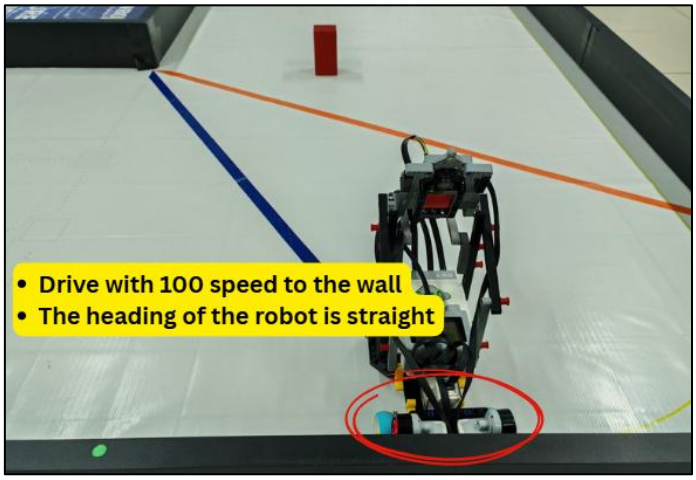
```
For loopCount = 1 To 12
  If cw = 1 Then
    If loopCount < 7 Then
      target = target + 90.3 * cw
    Else
      target = target + 90.8 * cw
    EndIf
  Else
    If loopCount < 7 Then
      target = target + 90.2 * cw
    Else
      target = target + 90.8 * cw
    EndIf
  EndIf
```

This 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 'cw green is last
    degrees = 1400
  Else
    degrees = 1300 'cw red is last
  EndIf
Else '-----------------anticw--------------------'
  If lastPillar = 2 Then
    degrees = 1300 'ccw red is last
  Else
    degrees = 1250 'ccw green is last
  EndIf
EndIf
waitDegrees(mainSpeed,degrees,0) 'above value will put inside here
Speaker.Tone(100,1500,100)
If cw=1 Then
  While Sensor.ReadRawValue(4,0)<>4 And Sensor.ReadRawValue(4,0)<>5
    Drive(mainSpeed*0.9) 'speed when avoid pillar
  EndWhile
Else
  While Sensor.ReadRawValue(4,0)<>2
    Drive(mainSpeed*0.9)
  EndWhile
EndIf
```

After completing 12 loops, the robot will move straight for 300 for counter clockwise to ensures the robot do not hit any pillars while moving into the parking area. After moving for certain degrees, the robot will adjust its heading based on the direction. The robot will drive into the wall with a speed of 100 to make sure its heading is perfectly straight. This is important because it allows the robot to move backward in a straight way.

```
If cw = 1 Then          '----------------cw----------------'
  target = target - 95
  Speaker.Tone(100,500,1000)
Else                     '-----------------anti-cw------------
  WaitDegrees(mainSpeed,300,0)
  target = target + 95
  Speaker.Tone(100,600,100)
EndIf

Time.Reset5()
While Time.Get5()<3000
  Drive(100)
EndWhile
Brake(1)
Motor.Stop("D","True")
Speaker.Tone(100,500,100)
```



Next, the robot will perform a reverse action and readjust the heading target to ensure it is perfectly straight and aligned parallel to the magenta parking lot.

```
'-----------------reverse------------'
If cw = 1 Then
  target = target + 70
  waitDegrees(mainSpeed*-1,340,0)
Else
  target = target - 60
  WaitDegrees(mainSpeed*-1,340,0)
EndIf
Brake(1)
```

The robot will move forward while maintaining a fixed distance from the parking lot which is **300 mm** for a counter clockwise heading and **240 mm** for a clockwise heading. 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 second 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 = 300 '275 ori
  waitDegrees(mainSpeed,600,0) 'before magenta
  Speaker.Tone(50,250,1000) 'sound 1
  While rightWall>140
    Drive(mainSpeed)
  EndWhile
  Speaker.Tone(50,200,50)  'sound 2
  waitDegrees(mainSpeed,270,0) 'cross 1st magenta
  Speaker.Tone(50,200,50) 'sound 3
  While rightWall>140
    Drive(mainSpeed)
  EndWhile
  waitDegrees(mainSpeed,300,1) 'cross 2nd magenta
  Speaker.Tone(50,400,50) 'sound 4
  Program.Delay(300)
  SteeringDrive(-70,-50,335,1) 'first reverse
  SteeringDrive(60,-50,305,1) 'second reveerse
  SteeringDrive(-40,40,15,1)
```

```
Else '.....clockwise....
  wallDistance = 240
  waitDegrees(mainSpeed,350,0)
  Speaker.Tone(50,250,1000)
  While leftWall>140
    Drive(mainSpeed)
  EndWhile
  Speaker.Tone(50,50,50)
  waitDegrees(mainSpeed,270,0)
  Speaker.Tone(50,200,50)
  While leftWall>140
    Drive(mainSpeed)
  EndWhile
  waitDegrees(mainSpeed,275,1)
  Speaker.Tone(50,400,50)
  Program.Delay(300)
  SteeringDrive(80,-50,355,1)
  SteeringDrive(-60,-50,310,1)
  SteeringDrive(40,80,25,1)
EndIf
```

The flow diagram below illustrates the detailed process of parking in the magenta parking lot.