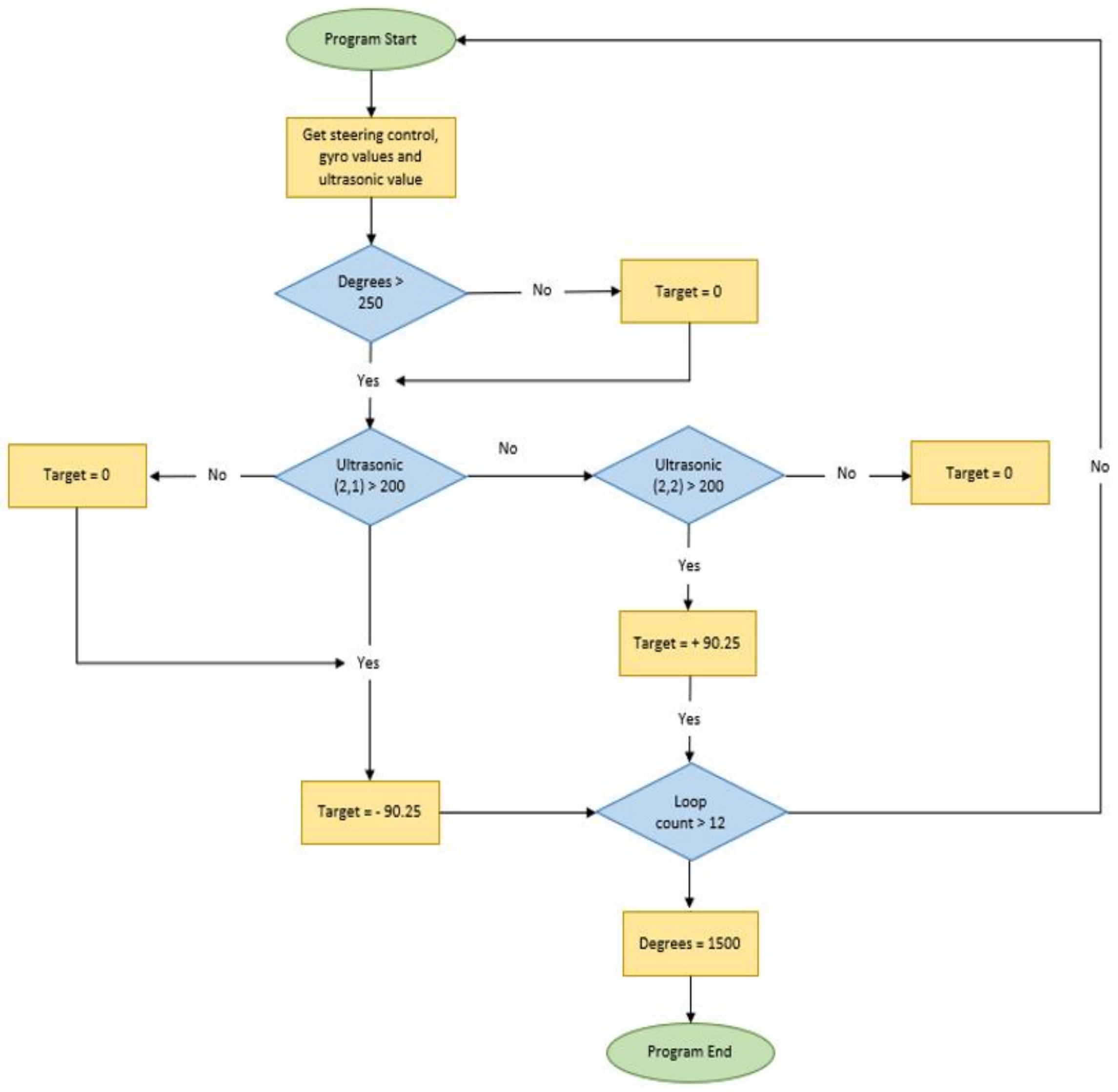
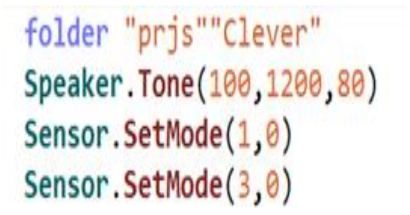
3.1 Open challenge

The left and right ultrasonic sensors help the self-driving car determine its direction, whether it is moving clockwise or counterclockwise. These sensors detect the distance between the self-driving car and nearby walls, allowing the car to adjust its steering mechanism to avoid collisions. When an ultrasonic sensor reaches a certain threshold, indicating the absence of a wall, the car will turn accordingly. Additionally, the gyro sensor ensures that the car maintains a straight path and makes precise turns when necessary.

The diagram below shows the flowchart for the Open Challenge.

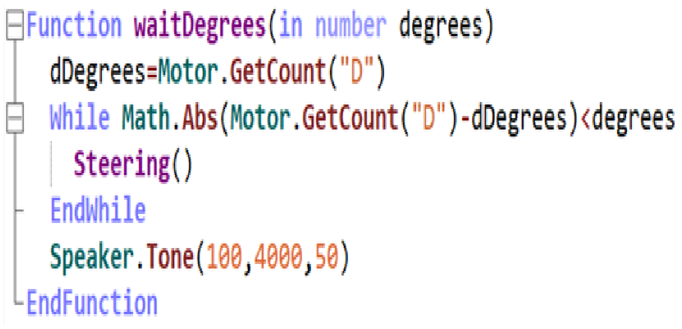


Below are the explanation for the Open Challenge programming:



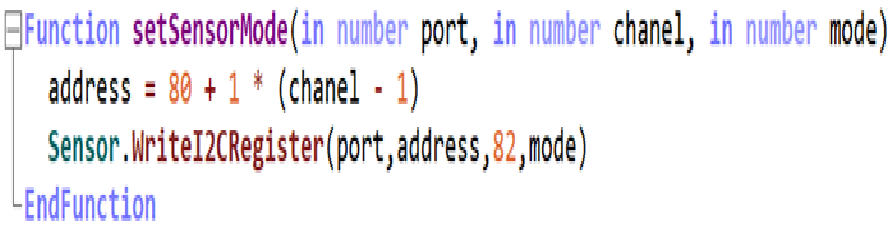
```
folder "prjs""Clever"
Speaker.Tone(100,1200,80)
Sensor.SetMode(1,0)
Sensor.SetMode(3,0)
```

The code snippet appears to set up a project in a folder named "prjs" with the title "Clever." Additionally, it sets the mode for two sensors (Sensor 1 and Sensor 3) to mode 0, which are preparing them for a basic operation and specific function.



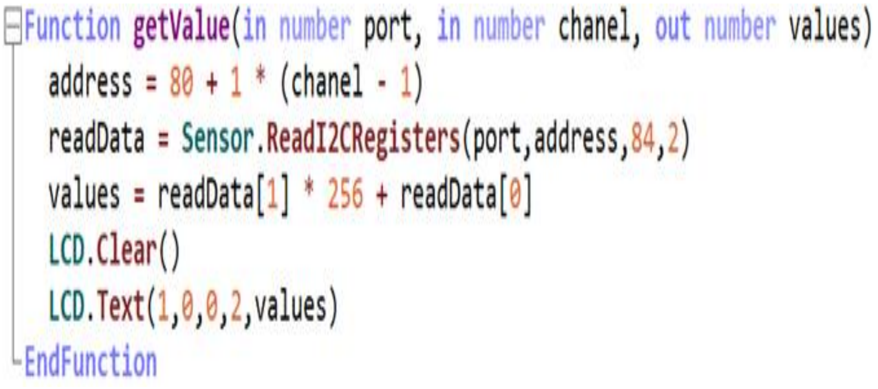
```
Function waitDegrees(in number degrees)
  dDegrees=Motor.GetCount("D")
  While Math.Abs(Motor.GetCount("D")-dDegrees)<degrees
    Steering()
  EndWhile
  Speaker.Tone(100,4000,50)
EndFunction
```

The **waitDegrees** function pauses the program until motor D rotates a specified number of degrees. It records the initial degree count of motor D and enters a loop that continues until the difference between the current count and the initial count reaches the desired degree threshold. During this wait, it repeatedly calls the Steering function to maintain the self-driving car’s steering.



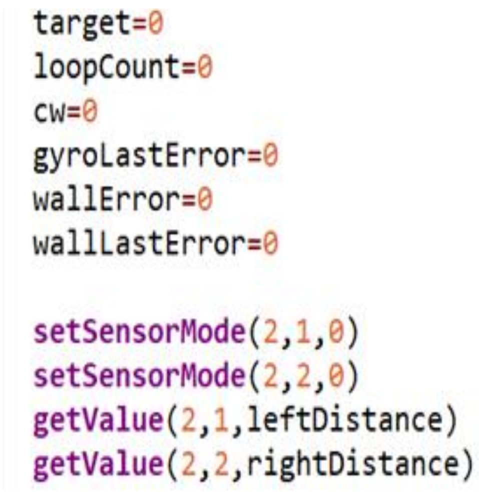
```
Function setSensorMode(in number port, in number chanel, in number mode)
  address = 80 + 1 * (chanel - 1)
  Sensor.WriteI2CRegister(port,address,82,mode)
EndFunction
```

The **setSensorMode** function sets up a multiplexer sensor by choosing the port, channel and mode. It calculates the I2C address for the channel and sends a command to set the sensor’s mode. This allows the self-driving car to adapt the sensor's functionality according to the requirements of the task.



```
Function getValue(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]
  LCD.Clear()
  LCD.Text(1,0,0,2,values)
EndFunction
```

The **function getValue** takes three parameters: port, channel, and values. It calculates the I2C address for the specified channel by adding 80 to the channel number (adjusted by subtracting 1). The function then reads two registers from the sensor using the Sensor.ReadI2CRegisters, starting at the calculated address. It combines the two bytes of data to compute a single value, which is assigned to the values output parameter. Finally, it clears the LCD display and shows the values at the specified position on the screen.



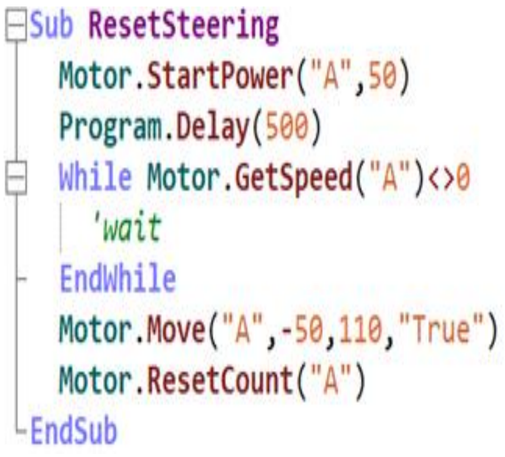
```
target=0
loopCount=0
cw=0
gyroLastError=0
wallError=0
wallLastError=0

setSensorMode(2,1,0)
setSensorMode(2,2,0)
getValue(2,1,leftDistance)
getValue(2,2,rightDistance)
```

This code snippet initializes several variables and configures sensors for a self-driving car’s movement. It sets values such as **target, loopCount, cw, gyroLastError, and**

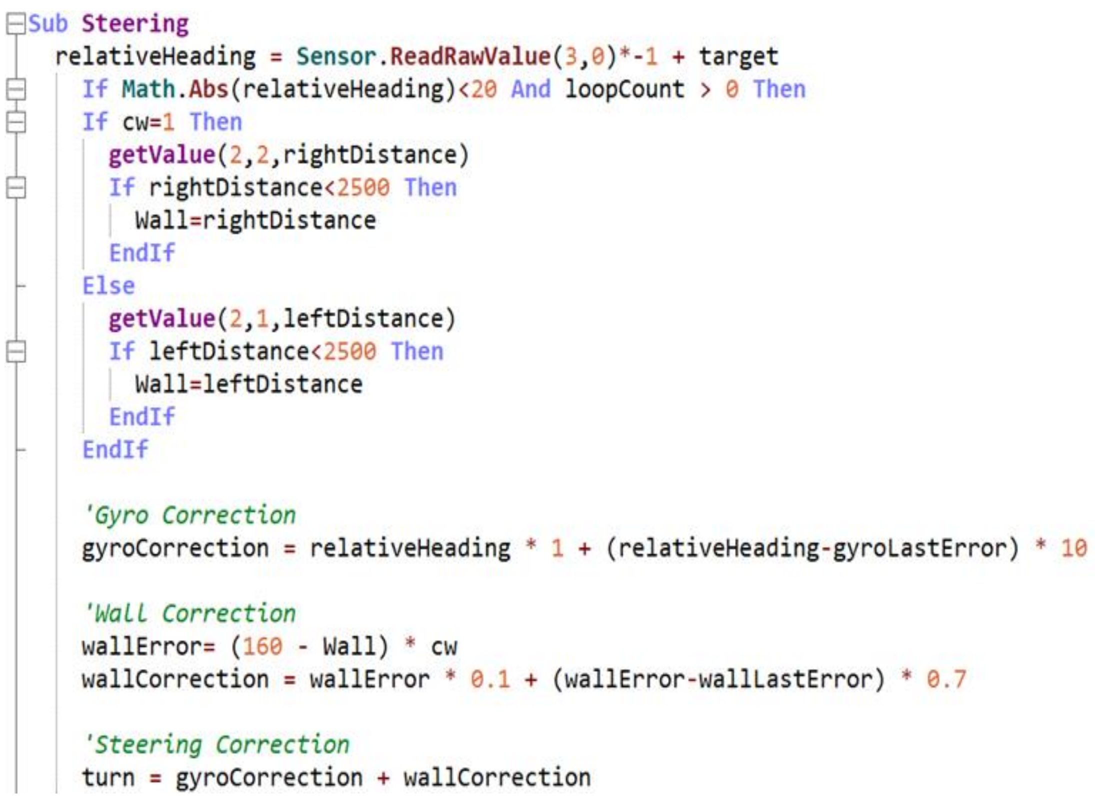
wallError to zero, which likely represents the target value, loop iteration count, a clockwise direction indicator, and error values for gyro and wall sensors, respectively.

The **setSensorMode** function is called to configure two sensors on port 2, with channels 1 and 2, both set to mode 0, possibly for basic functionality. After that, the getValue function retrieves distance readings from the left and right sensors (channels 1 and 2) and assigns these values to the variables leftDistance and rightDistance. This setup is likely part of a control loop for a robot to navigate based on distance measurements from its surroundings.



```
Sub ResetSteering
  Motor.StartPower("A",50)
  Program.Delay(500)
  While Motor.GetSpeed("A")<>0
    'wait
  EndWhile
  Motor.Move("A",-50,110,"True")
  Motor.ResetCount("A")
EndSub
```

The **ResetSteering** subroutine reset motor A. It starts the motor at 50% power for 500 milliseconds, then waits until the motor stops. Afterwards, it moves the motor backwards at -50% speed for 110 degrees and finally resets the motor's position count. The Motor. The move function specifies the motor port, speed, rotation degrees, and brake mode.

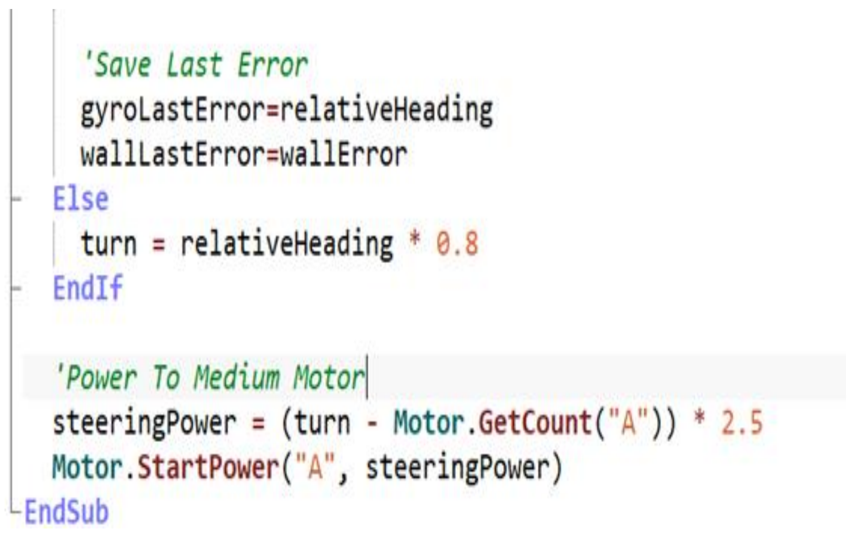


```
Sub Steering
  relativeHeading = Sensor.ReadRawValue(3,0)*-1 + target
    If Math.Abs(relativeHeading)<20 And loopCount > 0 Then
    If cw=1 Then
      getValue(2,2,rightDistance)
      If rightDistance<2500 Then
        Wall=rightDistance
      EndIf
    Else
      getValue(2,1,leftDistance)
      If leftDistance<2500 Then
        Wall=leftDistance
      EndIf
    EndIf

    'Gyro Correction
    gyroCorrection = relativeHeading * 1 + (relativeHeading-gyroLastError) * 10

    'Wall Correction
    wallError= (160 - Wall) * cw
    wallCorrection = wallError * 0.1 + (wallError-wallLastError) * 0.7

    'Steering Correction
    turn = gyroCorrection + wallCorrection
```

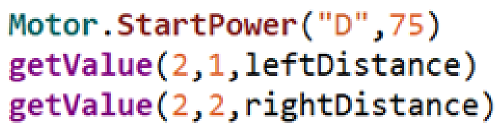


```
    'Save Last Error
    gyroLastError=relativeHeading
    wallLastError=wallError
  Else
    turn = relativeHeading * 0.8
  EndIf

  'Power To Medium Motor
  steeringPower = (turn - Motor.GetCount("A")) * 2.5
  Motor.StartPower("A", steeringPower)
EndSub
```

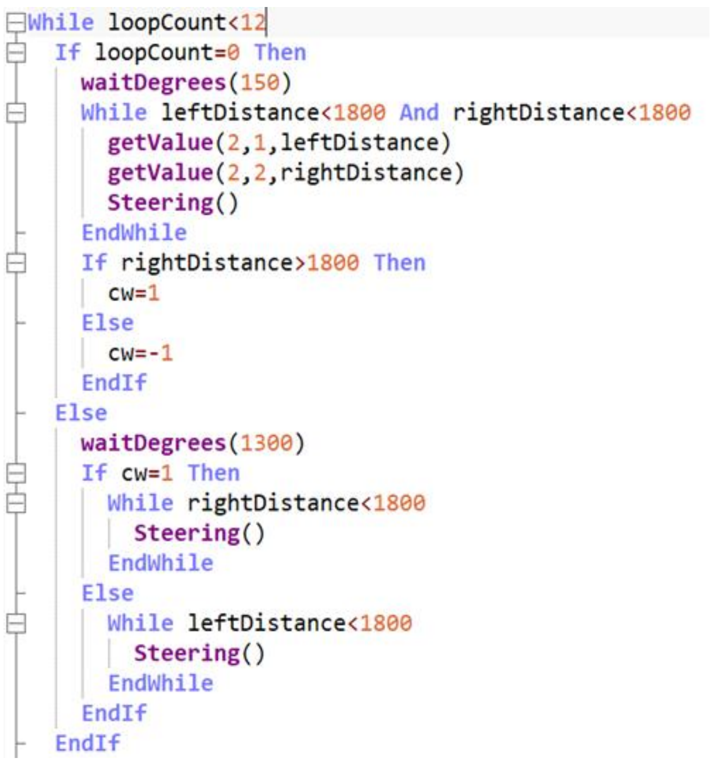
The subroutine **Steering** calculates the steering adjustments for a self-driving car based on its relative heading and wall distances. It first reads the raw value from sensor 3, adjusts it by negating it and adding a target value to get **relativeHeading**. If the absolute value of **relativeHeading** is less than 20 and loopCount is greater than 0, it checks if the self-driving car is moving clockwise (cw = 1). Depending on the direction, it retrieves the right or left distance using getValue, updating the Wall variable if the distance is less than 2500.

The subroutine then computes a gyro correction based on the **relativeHeading** and its previous error and a wall correction based on the distance to the wall. These corrections are combined to determine the steering adjustment and turn. If the condition is not met, it turns to a scaled version of relativeHeading. Finally, it calculates the power for the medium motor by adjusting the steering power based on the difference between the calculated turn and the motor count and activates the motor with Motor.StartPower("A", steeringPower).

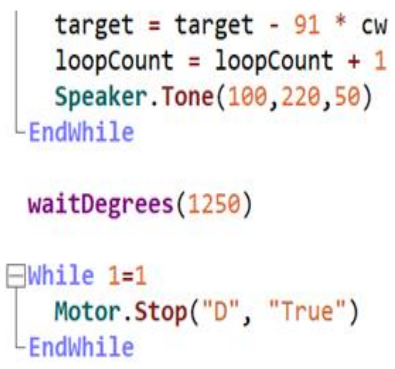


```
Motor.StartPower("D",75)
getValue(2,1,leftDistance)
getValue(2,2,rightDistance)
```

The code snippet starts by activating motor "D" at 75% power using **Motor.StartPower**("D", 75). It then retrieves distance measurements from two sensors: it calls getValue(2, 1, leftDistance) to get the left distance and getValue(2, 2, rightDistance) to get the right distance. This setup is likely part of a control mechanism for navigating or avoiding obstacles based on the distances measured by the sensors.



```
While loopCount<12
  If loopCount=0 Then
    waitDegrees(150)
    While leftDistance<1800 And rightDistance<1800
      getValue(2,1,leftDistance)
      getValue(2,2,rightDistance)
      Steering()
    EndWhile
    If rightDistance>1800 Then
      cw=1
    Else
      cw=-1
    EndIf
  Else
    waitDegrees(1300)
    If cw=1 Then
      While rightDistance<1800
        Steering()
      EndWhile
    Else
      While leftDistance<1800
        Steering()
      EndWhile
    EndIf
  EndIf
```



```
  target = target - 91 * cw
  loopCount = loopCount + 1
  Speaker.Tone(100,220,50)
EndWhile

waitDegrees(1250)

While 1=1
  Motor.Stop("D", "True")
EndWhile
```

The code snippet initiates a loop that runs while loopCount is less than 12. In the first iteration (loopCount = 0), it waits for the motor to rotate 150 degrees with waitDegrees(150) and enters a nested loop that continuously retrieves left and right distance measurements until both distances are less than 1800. Within this loop, it calls the Steering() function to adjust the self-driving car’s direction. Afterwards checks if the right distance exceeds 1800; if so, it sets cw to 1 (indicating a clockwise direction), otherwise, it sets cw to -1.

For subsequent iterations, it waits for the motor to rotate 1300 degrees, and based on the current value of cw, it enters another nested loop that keeps calling Steering() while either the left or right distance is less than 1800. After each iteration, it adjusts the target by subtracting 91 multiplied by cw and increments loopCount. It also plays a tone using the speaker.

After the loop concludes, the code waits for the motor to rotate 1250 degrees, and then it enters an infinite loop that stops motor "D" with Motor.Stop("D", "True").