```
#include <Servo.h>

// These constants won't change.  They're the pin numbers
// of the sensors' outputs:

// Sensor Pins --------------------------------------------------------
const int LpingPinS = 1;
const int LpingPinR = 0;
const int RpingPinS = 5;
const int RpingPinR = 4;
const int BpingPin = 14;
// --------------------------------------------------------------------

// Assign servos ------------------------------------------------------
Servo steer;      // Create servo object to control a servo
Servo wheelL;     // A maximum of eight servo objects can be created
Servo wheelR;
Servo wheelL2;
Servo wheelR2;
Servo Lssensor;   // Left side sensor
Servo Rssensor;   // Right side sensor
Servo Backsensor; // back up sensor servo
//--------------------------------------------------------------------

// Servo assignments/variables ----------------------------------------
int spos = 176;    // Variable to store the servo position
int bspos = 90;
int wdir = 90;     // The direction of the wheel
int dirturn;       // Direction/angle of turn
int Pos2adjst = 0; // Forward position 2 sensor adjustment.
```

```
int BLWR = 0;      // wheel ramping for navigating corners
int BRWR = 0;
// --------------------------------------------------------------------



// System status/startup operation ------------------------------------
int rstatus = 0; // The initial operation of the robot
int cycle = 0;   // Number of cycles applying to the sensor servo position
int updtrt = 90; // The rate of initial upate for the ultrasonic sensors
int fast = 60;   // Alert update rate
int Readfunction(int cycle);
int forward = 1;
void Calculate();
void ActiveCalculate();
void InitialBackReading();
void FwdLSen();
void FwdRSen();
void BSensor();
int running = 0;
//--------------------------------------------------------------------


//Navigation Cases---------------------------------------------------
void Navigation();
void Passive();
void SearchPath();
void Aggressive();
void Evasive();
void Large_Path();
void Finish_Previous();
int follow_up = 0;
```

```
int Direct_Obstacle = 0;

int Evade = 0;

int On_Course = 0;

int Course = 0;

int Left = 1;

int Right = 2;

int MRight_Back1 = 3;

int PRight_Back2 = 4;

int MLeft_Back1 = 5;

int PLeft_Back2 = 6;

int Back_Mleft = 7;

int Back_Mright = 8;

int Back_Straight = 9;

int TDuration = 0;

int TDO = 0;

int PW = 0;

int rst = 0;

//---------------------------------------------------------------------


//Sensor Reading Info Process----------------------------------------

int LSO = 0;//Direct output

int RSO = 0;//Direct output

int BS = 0;//Direct output of back sensor

int LS[6] = {0,0,0,0,0,0};// Left Sensor distance

int RS[6] = {0,0,0,0,0,0};// Right Sensor distance

int BSI[8] = {0,0,0,0,0,0,0,0};

int BR2 = 0;//90 degrees to the back right

int BR1 = 0;//45 degrees to the back right

int StrtBck = 0;//Directly Back distance

int BL2 = 0;//90 degrees to the back left
```

```
int BL1 = 0;//45 degrees to the back left
int counter = 0;

// Calculations
int FwdLavg = 0;//forward left average distance
int FwdRavg = 0;//forward right average distance
int FwdLfar = 0;//farthest forward left distance
int FwdRfar = 0;//farthest forward right distance
int FwdLneer = 0;//closest forward left distance
int FwdRneer = 0;//closest forward right distance
int LMin = 0;
int RMin = 0;
int LMax = 0;
int RMax = 0;
int Delta_side_avg = 0;
int LPossible_Path = 0;
int RPossible_Path = 0;
//--------------------------------------------------------------------

//Test modes---------------------------------------------------------
int satest = 92; //for servo angle test ONLY
//--------------------------------------------------------------------

//Servo Cailbration--------------------------------------------------
int Lssencal = -4;//left sensor servo calibration
int Rssencal = 3;//right sensor servo calibration
int Bsencal = 10;//Back sensor servo calibration
int steercal = -2;//steering calibration
int wheelcal = 0;//centers all wheel servos
//--------------------------------------------------------------------
```

```
long duration, inches, cm;


void setup() {
  // Initialize serial communication:
  Serial.begin(9600);
  steer.attach(18);                    // Attaches the servo to a pin
  wheelL.attach(17);
  wheelL2.attach(15);
  wheelR.attach(19);
  wheelR2.attach(16);
  Lssensor.attach(24);                 // Left side sensor servo
  Rssensor.attach(25);                 // Right side sensor servo
  Backsensor.attach(12);               // Back sensor servo

  //Startup Calibration
  pinMode(13, OUTPUT);                 // Status LED (this LED can be used to test
                                       // where the robot is in the code by setting
                                       // it to turn on under a certain condition.
  wheelL.write(wheelcal+wdir);         // Set all drive servos
  wheelR.write(wheelcal+180-wdir);
  Backsensor.write(Bsencal+bspos);
  Lssensor.write(Lssencal+spos);       // Side sensors (sensor needs to be in
                                       // position before reading the sensor)
  Rssensor.write(Rssencal+180-spos); // Side sensor
}
```

```
void loop() {

  Readfunction();

  if (forward == 1) {
    ActiveCalculate();
  }

  Navigation();

  wheelL.write(wheelcal+wdir);  //set all drive servos
  wheelR.write(wheelcal+180-wdir);
  wheelL2.write(wheelcal+wdir-BLWR);
  wheelR2.write(wheelcal+180-wdir+BRWR);
  steer.write(dirturn + steercal);
  if (rst == true) {
    Backsensor.write(Bsencal+90);
    rst = false;
  }
}


long microsecondsToInches(long microseconds)
{
  // According to Parallax's datasheet for the PING))), sound travels at 1130
  // feet per second, or 1 inch per 73.746 microseconds.  This gives the
  // distance travelled by the ping, outbound and return, so we divide by 2
  // to get the distance to the obstacle.
  // See: http://www.parallax.com/dl/docs/prod/acc/28015-PING-v1.3.pdf
  return microseconds / 74 / 2; //Ping 74us/in
```

```
}



long microsecondsToCentimeters(long microseconds)
{
  // The speed of sound is 340 m/s or 1 centimetre per 29 microseconds.
  // The ping travels out and back, so to find the distance of the
  // object we take half of the distance travelled.
  return microseconds / 29 / 2;
}



void Readfunction(void) { // Reads Forward/back sensors and converts values
                          // to inches/displays reading
  if (cycle == 1) {       // Reset update rate to regular speed
    if (LMin >= 15 && RMin >= 15) {
      updtrt = 120;
    }
  }
  if (updtrt == fast)
    Serial.println("fast update");
  if (forward == true && cycle == 1)
    delay(20);
  else if (forward == false)
    updtrt = 100;
  if (running == 1 && cycle == 1 && forward == false)
    delay(120);
  delay(updtrt);
  if (cycle == 1) {
    if (updtrt == fast) { // If fast cycle speed, give extra time for Vservo
```

```
                        // to move to position 1
    delay(60);
  }
  delay(64); // Normal cycle 1 delay
}
FwdLSen(); // Read sensors(left)
delay(10);
FwdRSen(); // (right)
if (forward == false) {
  Bsensor(); // Read back sensor
}
switch(cycle) {
  case 0://start up/reset
    spos = 176;
    bspos = 155;
    running = 0;
  break;

  case 1: // First reading --------------------------
    LS[1] = LSO;
    Serial.print(LS[1]);
    Serial.print(" LS[1] in, ");
    RS[1] = RSO;
    Serial.print(RS[1]);
    Serial.print(" RS[1] in, ");
    if (forward == 0) {
      BR2 = BS;
      Serial.print("Back Sensor R2: ");
      Serial.println(BR2);
    }
```

```
  Serial.println();
  spos = 158;  // Positions are one position ahead
  bspos = 125; // due to the servo task being
  delay(20);   // in front of the cycles
break;

case 2: // Second reading -------------------------
  LS[2] = LSO;
  Serial.print(LS[2]);
  Serial.print(" LS[2] in, ");
  RS[2] = RSO;
  Serial.print(RS[2]);
  Serial.print(" RS[2] in, ");
  if (forward == false) {
    BR1 = BS;
    Serial.print("Back Sensor R1: ");
    Serial.println(BR1);
  }
  Serial.println();
  spos = 138;
  bspos = 90;
  delay(20);
break;

case 3: // Third reading --------------------------
  LS[3] = LSO;
  Serial.print(LS[3]);
  Serial.print(" LS[3] in, ");
  RS[3] = RSO;
  Serial.print(RS[3]);
```

```
  Serial.print(" RS[3] in, ");
  if (forward == 0) {
    StrtBck = BS;
    Serial.print("Back Sensor Straight: ");
    Serial.println(StrtBck);
  }
  Serial.println();
  spos = 115;
  bspos = 55;
  delay(20);
break;

case 4: // Fourth reading --------------------------
  LS[4] = LSO;
  Serial.print(LS[4]);
  Serial.print(" LS[4] in, ");
  RS[4] = RSO;
  Serial.print(RS[4]);
  Serial.print(" RS[4] in, ");
  if (forward == 0) {
    BL1 = BS;
    Serial.print("Back Sensor L1: ");
    Serial.println(BL1);
  }
  Serial.println();
  spos = 92;
  bspos = 25;
  delay(20);
break;
```

```
  case 5: // Fifth reading --------------------------

    LS[5] = LSO;

    Serial.print(LS[5]);

    Serial.print(" LS[5] in, ");

    RS[5] = RSO;

    Serial.print(RS[5]);

    Serial.print(" RS[5] in, ");

    if (forward == 0) {

      BL2 = BS;

      Serial.print("Back Sensor L2: ");

      Serial.println(BL2);

    }

    Serial.println();

    spos = 176;

    bspos = 155;

    cycle = 0;

    running = 1;

    delay(20);

  break;

}


if (LSO <= 10 | RSO <= 10) {

  updtrt = fast;

}

cycle ++;

if (forward == true) {

  bspos = 90;

}

Lssensor.write(Lssencal+spos);     // Side sensors (sensor needs to be in

                                   // position before reading the sensor)
```

```
  Rssensor.write(Rssencal+180-spos); // Side sensor
  if (forward == false)
    Backsensor.write(Bsencal+bspos);

  if (cycle == 1)
    Calculate();
}



void InitialBackReading(void) {
  int counter = 0;
  dirturn = 90;
  wheelL.write(wheelcal+90);  // Set all drive servos to stop while back
                              // sensor starts reading
  wheelR.write(wheelcal+180-90);
  wheelL2.write(wheelcal+90);
  wheelR2.write(wheelcal+180-90);
  steer.write(dirturn + steercal);
  while(counter <= 7) {
    if ((counter >= 1) && (counter <= 7)) {
      Bsensor();//read back sensor
      BSI[counter] = BS;
      Serial.print("Back Sensor R");
      Serial.print(counter);
      Serial.print(": ");
      Serial.println(BS);
      Serial.println();
    }

    switch(counter) {
```

```
case 0://start up/reset

  bspos = 155;

break;


case 1: // First reading --------------------------

  bspos = 140;

break;


case 2: // Second reading -------------------------

  bspos = 120;

break;


case 3:

  bspos = 90;

break;


case 4:

  bspos = 60;

break;


case 5:

  bspos = 40;

break;


case 6:

  bspos = 25;

break;


case 7: // Seventh reading --------------------------

  bspos = 90;
```

```
        cycle = 0;
      break;
    }

    Backsensor.write(Bsencal+bspos);
    if (counter == 0)
      delay(250);
    if ((counter == 1) | (counter == 6))
      delay(20);
    delay(200);
    counter ++;
  }
} // ----------------------------------------------------------------------------


void FwdLSen(void) { // Read forward Left sensor
  pinMode(LpingPinS, OUTPUT);
  digitalWrite(LpingPinS, LOW);
  delayMicroseconds(4);
  digitalWrite(LpingPinS, HIGH);
  delayMicroseconds(15); // Was 21
  digitalWrite(LpingPinS, LOW);

  pinMode(LpingPinR, INPUT);
  duration = pulseIn(LpingPinR, HIGH);
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  LSO = inches;
  delay(10);
}
```

```
void FwdRSen(void) { // Read forward Right sensor
  pinMode(RpingPinS, OUTPUT);
  digitalWrite(RpingPinS, LOW);
  delayMicroseconds(4);
  digitalWrite(RpingPinS, HIGH);
  delayMicroseconds(15);
  digitalWrite(RpingPinS, LOW);

  pinMode(RpingPinR, INPUT);
  duration = pulseIn(RpingPinR, HIGH);
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  RSO = inches;
}


void Bsensor(void) { // Read back sensor
  pinMode(BpingPin, OUTPUT);
  digitalWrite(BpingPin, LOW);
  delayMicroseconds(2);
  digitalWrite(BpingPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(BpingPin, LOW);

  pinMode(BpingPin, INPUT);
  duration = pulseIn(BpingPin, HIGH);
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
```

```
  BS = inches;
}



// The PING))) is triggered by a HIGH pulse of 2 or more microseconds.
// Give a short LOW pulse beforehand to ensure a clean HIGH pulse:
// The same pin is used to read the signal from the PING))): a HIGH
// pulse whose duration is the time (in microseconds) from the sending
// of the ping to the reception of its echo off of an object.

void Calculate(void) { // Calculates data into usable information for later
  int i;               // navigation when repeating the read cycle (cycle == 1)
  FwdLavg = ((LS[2] + LS[3] + LS[4] + LS[5])/4);
  Serial.print("Left AVG: ");
  Serial.print(FwdLavg);
  FwdRavg = ((RS[2] + RS[3] + RS[4] + RS[5])/4);
  Serial.print(" Right AVG: ");
  Serial.print(FwdRavg);
  Serial.println();

  LMin = LMax = LS[2];
  RMin = RMax = RS[2];
  for (i = 3; i <= 5; ++i) {
    LMin = min(LMin, LS[i]);
    RMin = min(RMin, RS[i]);
    LMax = max(LMax, LS[i]);
    RMax = max(RMax, RS[i]);
  }

  Serial.print("LMin: ");
```

```
  Serial.print(LMin);

  Serial.print("  RMin: ");
  Serial.println(RMin);

  Serial.print("LMax: ");
  Serial.print(LMax);

  Serial.print("  RMax: ");
  Serial.println(RMax);

  Delta_side_avg = abs(FwdLavg - FwdRavg);
  Serial.print("Delta Side Average: ");
  Serial.println(Delta_side_avg);

  LPossible_Path = abs(LMax - FwdLavg);
  Serial.println("Left Possible Path: ");
  Serial.println(LPossible_Path);

  RPossible_Path = abs(RMax - FwdRavg);
  Serial.println("Right Possible Path: ");
  Serial.println(RPossible_Path);
  Serial.println();

  SearchPath();

}


void ActiveCalculate(void) { // Actively calculates the position (angle) of
```

```
                              // reading two in order to tell the exact position
                              // of an object by time difference, when possible
  if (cycle == 2) {
    int tmp = 0;
    int Delta_1 = 0;
    tmp = (LS[1] - RS[1]);
    Delta_1 = abs(tmp);
    Serial.print("delta-1: ");
    Serial.println(Delta_1);
    if ((0 < Delta_1) & (Delta_1 < 2)) {
      Serial.println("works");
      //spos = (spos + Pos2adjst);
    }
  }
  if ((cycle == 3) & ((LS[2] - LS[1]) >= 10) & ((RS[2] - RS[1]) >= 10)
    & (LS[1] <= 32) & (RS[1] <= 32)) {
    Direct_Obstacle = true;
  }
}


//-------------------------------------------------------------------------


void Navigation(void) { // Takes calculated values and makes the decision
                        // to plot a course accordingly.


// Decision of path ----------------------------
  BLWR = 0;
  BRWR = 0;
  if (BS <= 7 && running == true && forward == false && cycle > 1) {
```

```
  wheelL.write(wheelcal+90);  // Set all drive servos to stop while back
                             // sensor starts reading
  wheelR.write(wheelcal+180-90);
  wheelL2.write(wheelcal+90);
  wheelR2.write(wheelcal+180-90);
  steer.write(90 + steercal);
  cycle = 1;
  rst = true;
  forward = true;
}
else if ((LSO <= 6 || RSO <= 6) && (running == true) && (follow_up == false)
  && (cycle == 1 | cycle == 2)) {
  wheelL.write(wheelcal+90);  // Set all drive servos to stop while back
                               // sensor starts reading
  wheelR.write(wheelcal+180-90);
  wheelL2.write(wheelcal+90);
  wheelR2.write(wheelcal+180-90);
  steer.write(90 + steercal);
  wdir = 90;
}

if ((cycle == 1) && (running == true) && (forward == true)) {
  if (On_Course == true) { // The limitations of Evasive actions are defined:
    if ((LS[1] <= 9) | (RS[1] <= 9) | (LS[5] <= 4) | (RS[5] <= 4)
      | (LS[3] <= 5) | (RS[3] <=5)) {
      Evade = true; // On course is a future application where the robot has
    }               // a location that it is trying to get to and only avoids
  }                 // obstacle which would cause an imminent collision.
  else if ((LS[1] <= 12) | (RS[1] <= 12) | (LMin <= 9) | (RMin <= 9)) {
    Evade = true;
```

```
  }


  if (Evade == true) {
    Evasive();
  }
  else if (PW == true) {
    Large_Path();
  }
  else if (follow_up == true) {
    Finish_Previous();
  }
  else if ((On_Course == true) && (Direct_Obstacle == true)) {
    Aggressive();
  }
  else {
    Passive();
  }
// Path/action is planned in order of relativity -----------------------------
}
else if ((cycle == 1) && (running == true) && (forward == false)) {
  if ((BL1 <= 3) | (BL2 <= 2) | (BR1 <= 3) | (BR2 <= 2) | (StrtBck <= 4)) {
    Evasive();
  }
  else if (follow_up == true) {
    Finish_Previous();
  }
}
else if (follow_up == true) {
  Finish_Previous();
}
```

```
}



void Evasive(void) {
  follow_up = true;
  TDuration = 0;
  wdir = 180;
  int choice = 0;
  int tmp = 0;
  int avgdiff = 0;
  int BIS[8] = {0,0,0,0,0,0};// Back Sensor Initial distance
  choice = random(101);
  tmp = 0;
  tmp = (FwdLavg - FwdRavg);
  avgdiff = abs(tmp);
  if ((LS[1] <= 17) | (RS[1] <= 17)) { // If obstacle is sensed directly ahead:
    if ((LS[3] >= 20) && (LS[4] >=15) && (RS[3] >=20) && (RS[4] >= 15)
      && (LS[5] >= 6) && (RS[5] >= 6) && (LS[2] >= 7) && (RS[2] >= 7)) {
                          // If both sides have enough space to turn
      if (avgdiff <= 30) { // Checks for similar distances
        if (choice < 51) { // Turn by probability
          Course = Left;
          TDuration = 15;
        }
        else {
          Course = Right;
          TDuration = 15;
        }
      }
      else if (FwdLavg > FwdRavg) {
```

```
    Course = Left;
    TDuration = 20;
  }
  else {
    Course = Right;
    TDuration = 20;
  }
  forward = true;
}
else if ((LS[3] >= 20) && (LS[4] >=15) && (LS[5] >= 6) && (LS[2] >= 7)) {
                                     // Left side has enough space
  Course = Left;
  TDuration = 20;
  forward = true;
}
else if ((RS[3] >=20) && (RS[4] >= 20) && (RS[5] >= 6) && (RS[2] >= 7)) {
                                     // Right side has enough space
  Course = Right;
  TDuration = 20;
  forward = true;
}
else { // Attempts to go in reverse
  wdir = 0;
  running = false;
  forward = false;
  cycle = 0;
  digitalWrite(13, LOW);   // Set the LED on
  InitialBackReading();    // Takes an initial detailed reading
  if ((BSI[1] >= 0) && (BSI[7] >= 0) && (BSI[2] >= 28) && (BSI[6] >= 28)
    && (BSI[3] >= 28) && (BSI[5] >= 28) && (BSI[4] >= 20) && (LS[5] > 6)
```

```
  && (RS[5] > 6)) {   // If plenty of space
  if (choice <= 50) { // Probability left or right
    Course = Back_Mleft;
    TDuration = 20;
  }
  else {
    Course = Back_Mright;
    TDuration;
  }
}
else if ((BSI[1] >= 0) && (BSI[7] >= 0) && (BSI[2] >= 24)
  && (BSI[3] >= 20) && (BSI[4] >= 10) && (RS[5] > 6)) {
  Course = Back_Mleft;
  TDuration = 20;
}
else if ((BSI[1] >= 0) && (BSI[7] >= 0) && (BSI[6] >= 24)
  && (BSI[5] >= 20) && (BSI[4] >= 10) && (LS[5] > 6)) {
  Course = Back_Mright;
  TDuration = 20;
}
else if ((BSI[1] >= 0) && (BSI[7] >= 0) && (BSI[4] >= 35)) {
  Course = Back_Straight;
  //no limit on duration
}
else {
  digitalWrite(13, HIGH);  // Set the LED on
  delay(400);
  digitalWrite(13, LOW);   // Set the LED on
  follow_up = false;
  cycle = 1;
```

```
        forward = true;
        wdir = 90;
      }
    }
  }
  if (follow_up == true)
    Finish_Previous();
}



void Large_Path(void) {

  if ((LPossible_Path > 60) && (LPossible_Path > LS[1] + 20)
    && (FwdLavg < 36) & (LMin >= 12)) {
    dirturn = 120;
    BLWR = 86;
  }
  else {
    dirturn = 60;
    BRWR = 86;
  }
}



void Finish_Previous(void) {
  switch(Course) {
    case 1:
      if (TDuration > 0) {
        dirturn = 140;
        BLWR = 84;
```

```
  }
  else {
    dirturn = 90;
    follow_up = false;
    BLWR = 0;
  }
break;

case 2:
  if (TDuration >= 1) {
    dirturn = 50;
    BRWR = 84;
  }
  else if (TDuration == 0) {
    dirturn = 90;
    follow_up = false;
    BRWR = 0;
  }
break;

case 3:
break;

case 4:
  if (TDuration > (TDO - 4)) {
    dirturn = 50;
    BRWR = 84;
  }
  else if (TDuration > 4) {
    dirturn = 90;
```

```
      BRWR = 0;
    }
    else if (TDuration > 0) {
      dirturn = 140;
      BLWR = 84;
    }
    else {
      dirturn = 90;
      BLWR = 0;
    }
  break;

  case 5:
  break;

  case 6:
    if (TDuration > (TDO - 4)) {
      dirturn = 140;
      BLWR = 84;
    }
    else if (TDuration > 4) {
      dirturn = 90;
      BLWR = 0;
    }
    else if (TDuration > 0) {
      dirturn = 50;
      BRWR = 84;
    }
    else {
      dirturn = 90;
```

```
    BRWR = 0;
  }
break;

case 7:
  if (TDuration >= 16)
    dirturn = 90;
  else if (TDuration > 0)
    dirturn = 140;
  else if (TDuration == 0) {
    dirturn = 90;
    follow_up = false;
    forward = true;
    rst = true;
  }
break;

case 8:
  if (TDuration >= 16) {
    dirturn = 90;
    digitalWrite(13, HIGH);
  }
  else if (TDuration > 0)
    dirturn = 50;
  else if (TDuration == 0) {
    dirturn = 90;
    follow_up = false;
    forward = true;
    rst = true;
  }
```

```
    break;


    case 9:
      if (cycle == 1 && (((LS[3] >= 20) && (LS[4] >=20) && (LS[5] >= 10)
        && (LS[1] >= 10)) | ((RS[3] >=20) && (RS[4] >= 20)) && (RS[5] >= 8)
        && (RS[1] >= 8))) { // If both sides have enough space to turn
        Evasive();
        forward == true;
        rst = true;
      }
    break;
  }


  if (TDuration > 0)
    TDuration = (TDuration - 1);
}



void Aggressive(void) {


}




void Passive(void) { // Robot acts in a passive manner evading objects only as
                     // needed
      wdir = 180;
      Serial.println("not allert");
      if (Delta_side_avg >= 20) { // If difference between two sides is greater
                                  // than 40 inches
        if ((FwdLavg > FwdRavg)) {
```

```
    dirturn = 140;
    BLWR = 84;
  }
  else if ((FwdLavg < FwdRavg)) {
    dirturn = 50;
    BRWR = 84;
  }
}
else if ((LMin >= (RMin + 15)) && LMin >= 10 && LMin <= 40) {
  if (LS[1] > 24 || RS[1] > 24) {
    Course = 6;
    TDuration = 11;
  }
  else {
    dirturn = 140;
    BLWR = 84;
  }
}
else if ((RMin >= (LMin + 15)) && RMin >= 10 && RMin <= 34) {
  if (LS[1] > 24 || RS[1] > 24) {
    Course = 4;
    TDuration = 11;
  }
  else {
    dirturn = 50;
    BRWR = 84;
  }
}
else {
  dirturn = 90;
```

```
        BLWR = 0;
        BRWR = 0;
        Serial.println("straight");
        Serial.println();
      }
      TDO = TDuration;
}


void SearchPath(void) { // Searches for outliers which could be potential paths
                        // to follow
  int tmp = 0;
  tmp = (LMax - FwdLavg);
  int Lpath = abs(tmp);
  tmp = 0;
  tmp = (RMax - FwdRavg);
  int Rpath = abs(tmp);

  PW = false;
  if ((LPossible_Path > 60) && (LPossible_Path > LS[1] + 20)
    && (FwdLavg < 36) & (LMin >= 12))
    PW = true;
  else if ((RPossible_Path > 60) && (RPossible_Path > RS[1] + 20)
    && (FwdRavg < 36) && (RMin >= 12))
    PW = true;
}


void Agressive(void) {
```

```
}
```