```
#pragma config(UART_Usage, UART1, uartVEXLCD, baudRate19200, IOPins, None, None)
#pragma config(Sensor, in1,    cPot,           sensorPotentiometer)
#pragma config(Sensor, in3,    lPot,           sensorPotentiometer)
#pragma config(Sensor, in4,    gyro,           sensorGyro)
#pragma config(Sensor, in7,    status,         sensorAnalog)
#pragma config(Sensor, dgtl9,  dL,             sensorQuadEncoder)
#pragma config(Sensor, dgtl11, dR,             sensorQuadEncoder)
#pragma config(Motor,  port1,           cL,            tmotorVex393_HBridge, ope
#pragma config(Motor,  port2,           LF,            tmotorVex393_MC29, openLo
#pragma config(Motor,  port3,           LB,            tmotorVex393_MC29, openLo
#pragma config(Motor,  port4,           LLS,           tmotorVex393_MC29, openLo
#pragma config(Motor,  port5,           LLD,           tmotorVex393_MC29, openLo
#pragma config(Motor,  port6,           RLD,           tmotorVex393_MC29, openLo
#pragma config(Motor,  port7,           RLS,           tmotorVex393_MC29, openLo
#pragma config(Motor,  port8,           RB,            tmotorVex393_MC29, openLo
#pragma config(Motor,  port9,           RF,            tmotorVex393_MC29, openLo
#pragma config(Motor,  port10,          cR,            tmotorVex393_HBridge, ope
//*!!Code automatically generated by 'ROBOTC' configuration wizard

#pragma platform(VEX2)
#pragma competitionControl(Competition)
#include "Vex_Competition_Includes.c"

task autoSelect();
bool cmove = false;
bool side = false;
//All purpose autonomouse driving function
//Driving to distance and/or initiate scoring from a point (both on the feild ar
void Score(int y, bool score, int d, int h);
void setDrive(int x);
void stopAll();
void turn(int x, int holding);
void clawopn(int x);
void clawclamp();
void liftTo(int x, int spd, bool hold, int holding);
void Blocking(int height, int time, int spd);
int LCDbutton = 0;
int currBtn = 0;
int stage = 0;
void pre_auton()
{
  bLCDBacklight = true;                                    // Turn on LCD Backli
  startTask(autoSelect);
}

bool clamp = false;

task clawControl(){
  int CLOSED  = 500;
  int OPENED  = 1500;
  int signal = 0;
  int increm = 0;
  while(true){
    if(vexRT[Btn5U]){
      clamp = false;
      cmove = false;
      if(SensorValue[cPot] < OPENED){
        if(increm > 0)
          increm = 0;
```

```
        increm -= 5;
        signal = increm;
      }
      else{
        signal = 10;
      }
    }
    else if(vexRT[Btn5D]){
      clamp = false;
      cmove = false;
      if(SensorValue[cPot] > CLOSED){
        if(increm < 0)
          increm = 0;
        increm += 5;
        signal = increm;
      }
      else{
        signal = 0;
      }
    }
    else{
      increm = 0;
      if(SensorValue[cPot] > OPENED){
        signal = 20;
      }
      else{
        signal = 0;
        increm = 0;
      }
    }
    if(clamp == false){
      motor[cL] = signal;
      motor[cR] = signal;
    }
  }
}
bool userinput(){
  if(vexRT[Btn5D] || vexRT[Btn5U] || vexRT[Btn6D] || vexRT[Btn6U]){
    return true;
  }
  else{
    return false;
  }
}
int desiredLiftPosition;
task liftControl(){
  int liftpos = 0;
  bool clicked8R = false, clicked7L = false;
  const int LIFT_DOWN = 200;
  int E_STOP = 1200;
  desiredLiftPosition = SensorValue[lPot];
  int liftSignal = 0;
  const int setPoint = 650;
  int div = 0;
  int scored = -1;
  int clawopen = 1600;
  while(true)
  {
    while(vexRT[Btn7U]){
```

```
      if(vexRT[Btn6U])
      {
        liftSignal = 127;
      }
      if(vexRT[Btn6D])
      {
        liftSignal = -127;
      }
      if(vexRT[Btn6U] == 0 && vexRT[Btn6D] == 0)
      {
        liftSignal = 0;
      }

      motor[LLS]  = liftSignal;
      motor[LLD]  = liftSignal;
      motor[RLS] = liftSignal;
      motor[RLD] = liftSignal;
      wait1Msec(30);
    }

    if(vexRT[Btn8R] == 1)
    {
      clicked8R = true;
    }
    if(vexRT[Btn8R] == 0 && clicked8R == true)
    {
      liftpos = 1;
      clamp = true;
      motor[cL] = 50;
      motor[cR] = 50;
      cmove = true;
      clicked8R = false;
    }
    if(vexRT[Btn6U])
    {
      scored = 0;
      liftpos = 0;
      cmove = true;
      int checkavr = (SensorValue[cPot]);
      clamp = true;
      while(vexRT[Btn6U] == 1 && SensorValue[lPot] < 1200 && scored == 0)
      {
        if(SensorValue[lPot] < 800 && checkavr < 600)
        {
          liftSignal = 120;
          motor[cL] = 100;
          motor[cR] = 100;
        }else if(SensorValue[lPot] < 600 && checkavr > 1100)
        {
          liftSignal = 120;
          motor[cL] = 100;
          motor[cR] = 100;
        }else if(SensorValue[lPot] < 400 + (checkavr / 3))
        {
          liftSignal = 120;
          motor[cL] = 100;
          motor[cR] = 100;
        }
        else
```

```
        {
          if(SensorValue[cPot] < clawopen){
            liftSignal = 120;
            motor[cL] = -100;
            motor[cR] = -100;
          }
          else{
            liftSignal = 120;
            motor[cR] = 10;
            motor[cL] = 10;
          }
          if(SensorValue[lPot] > 1100 && E_STOP == 1200){
            liftSignal = -120;
            scored = 1;
          }
        }
        motor[LLS] = liftSignal;
        motor[LLD] = liftSignal;
        motor[RLD] = liftSignal;
        motor[RLS] = liftSignal;
        if(scored == 1)
        {
          motor[cR] = 80;
          motor[cL] = 80;
          wait1Msec(250);
        }
      }
      if(scored == 1)
      {
        scored = -1;
      }
      clicked7L = false;
      clamp = false;
      desiredLiftPosition = SensorValue[lPot];
    }

    if(liftpos == 1)
    {
      if(!vexRT[Btn6U]){
        desiredLiftPosition = setPoint;
      }
    }
    if(vexRT[Btn6D] == 1 && SensorValue[lPot] > LIFT_DOWN){
      cmove = false;
      clamp = false;
      motor[cR] = 40;
      motor[cL] = 40;
      liftpos = 0;
      liftSignal = -60;
      desiredLiftPosition = SensorValue[lPot];
    }else if(liftpos != 1 && vexRT[Btn6D] == 0)
    {
      liftSignal = 0;
    }
    if(SensorValue[lPot] < LIFT_DOWN && desiredLiftPosition != setPoint && desi
      liftSignal = -8;
    }
    else {
```

```
      if(SensorValue[lPot] > E_STOP){
        desiredLiftPosition = E_STOP - 100;
      }
      if(abs(desiredLiftPosition - SensorValue[lPot]) > (float) 30 * (float) 11.
      {
        div = 1;
      }
      else if(abs(desiredLiftPosition - SensorValue[lPot]) > (float) 10 * (float
      {
        div = 3;
      }
      else
      {
        div = 6;
      }
      if(vexRT[Btn6D] == 0)
        liftSignal = (desiredLiftPosition - SensorValue[lPot]) / div;
    }
  //  writeDebugStreamLine("CurrPos: %d, DesiredPos: %d", SensorValue[lPot], des

    //writeDebugStreamLine("ESTOP: %d", E_STOP);
    if(liftSignal > 0 && SensorValue[lPot] > E_STOP - 50)
    {
      liftSignal = liftSignal * -1 / 5;
    }
    motor[LLS]  = liftSignal;
    motor[LLD]  = liftSignal;
    motor[RLS] = liftSignal;
    motor[RLD] = liftSignal;
    wait1Msec(30);

  }
}

task autoSelect()
{
  bLCDBacklight = true;                                          // Turn on LCD Backli
  clearLCDLine(0);                                                 // Clear line 1 (0
  clearLCDLine(1);                                                 // Clear line 2 (1
  while(1)
  {
    displayLCDString(0, 0, "Autos: ");
    if(side == false)
    {
      displayLCDString(0, 10, "Left");
    }else
    {
      displayLCDString(0, 10, "Right");
    }
    LCDbutton = nLCDButtons;
    //writeDebugStreamLine("Btn Value: %d, Current Btn Vale %d, Auton: %d", LCDb
    if(LCDbutton != 0)
      currBtn = LCDbutton;
    if(LCDbutton == 0)
    {
      if(currBtn == 2)
      {
        side = !side;
      }
```

```
      if(currBtn == 4 && stage < 10)
      {
        stage++;
      }else if(currBtn == 4 && stage == 10)
      {
        stage = 0;
      }
      if(currBtn == 1 && stage > 0)
      {
        stage--;
      }else if(currBtn == 1 && stage == 0)
      {
        stage = 10;
      }
      switch(stage)
      {
        case 0: // no auton
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Open Claw");
          //writeDebugStreamLine("Open Claw");
            break;
        case 1: // Direct Cube
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Direct Cube");
          writeDebugStreamLine("Direct Cube");
            break;
        case 2: // 90 90 Cube
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "90 90 Cube");
          writeDebugStreamLine("90 90 Cube");
            break;
        case 3: // Blocking
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Blocking");
          writeDebugStreamLine("Blocking");
            break;
        case 4: // Stars
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Stars");
          writeDebugStreamLine("Stars");
            break;
        case 5: // Wall Stars / Cube
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Wall Stars/Cube");
          writeDebugStreamLine("Wall Stars/Cube");
            break;
        case 6: // Wall Stars / Cube
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Wall Stars/Stars");
          writeDebugStreamLine("Wall Stars/Stars");
            break;
        case 7: // no auton
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "No Auto");
            break;
        case 8: // no auton
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Straight/90cube");
            break;
```

```
        case 9: // no auton
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Empty Slot 2");
            break;
        case 10: // no auton
          clearLCDLine(1);                                                    // Clear l
          displayLCDCenteredString(1, "Empty Slot 3");
            break;
        default:
        break;
      }
      currBtn = 0;
    }
  }
}

task autonomous()
{
  stopTask(autoSelect);
  clearLCDLine(1);
  int s = 0;
  float abspos = 0;
  if(side == false)
  {
    s = 1;
  }else
  {
    s = -1;
  }
  switch(stage)
  {
    case 0: // no auton
    clearLCDLine(1);                                                      // Clear line 2
    displayLCDCenteredString(1, "Open claw");
    clawopn(1800);
    break;
    case 1: // Direct Cube
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "Direct Cube");
      Score(38, false, 0, 0);//CUBE
      clawclamp();
      wait1Msec(800);
      liftTo(450, 70, true, 14);
      turn(-120 * s, 15);
      Score(-22, true, -18, 700);
      wait1Msec(300);
      liftTo(300, 90, false, 0);
      wait1Msec(300);
      clawopn(1400);
      Score(33, false, 0, 0);
      clawclamp();
      wait1Msec(200);
      Score(-5, false, 0, 0);
      liftTo(500, 70, true, 10);
      wait1Msec(200);
      Score(-33, true, 30, 800);
      //Score(-5, false, 0, 0);
      //wait1Msec(450);
      liftTo(300, 90, false, 0);
```

```
      clawopn(1100);
        break;
    case 2: // 90 90 Cube
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "90 90 Cube");
      Score(30, false, 0, 0);//90 90 CUBE
      turn(-90 * s, 0);
      clawopn(1200);
      Score(30, false, 0, 0);
      clawclamp();
      wait1Msec(400);
      liftTo(400, 70, true, 10);
      Score(-30, false, 0, 0);
      turn(-87 * s, 10);
      Score(-22, true, -19, 700);
      liftTo(300, 90, false, 0);
      clawopn(1400);
      Score(35, false, 0, 0);
      clawclamp();
      wait1Msec(200);
      liftTo(500, 70, true, 10);
      Score(-38, true, 34, 800);
      liftTo(300, 90, false, 0);
      clawopn(1100);
        break;
    case 3: // Blocking
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "Blocking");
      Score(-30, false, 0, 0);//BLOCKING
      turn(-20 * s, 0);
      Score(-30, false, 0, 0);
      turn(50 * s, 0);
      Blocking(1400, 3, 40);
      clawopn(1800);
      wait1Msec(3000);
      clawopn(1500);
      liftTo(300, 90, false, 0);
      Score(25, false, 0, 0);
      clawclamp();
      wait1Msec(300);
      liftTo(500, 90, true, 20);
      wait1Msec(300);
      Score(-15, true, 8, 700);
      wait1Msec(300);
      liftTo(300, 90, false, 0);
      //wait1Msec(300);
      clawopn(1400);
      Score(33, false, 0, 0);
      clawclamp();
      wait1Msec(200);
      Score(-5, false, 0, 0);
      liftTo(500, 70, true, 10);
      wait1Msec(200);
      Score(-33, true, 30, 800);
      liftTo(300, 90, false, 0);
      clawopn(1100);
        break;
    case 4: // Stars
      clearLCDLine(1);                                                    // Clear line
```

```
      displayLCDCenteredString(1, "Stars");
      clawclamp();//STAR
      liftTo(600, 50, false, 0);
      clawopn(1100);
      wait1Msec(300);
      turn(-43 * s, 0);
      wait1Msec(300);
      Score(-10, false, 0, 0);
      wait1Msec(300);
      liftTo(300, 90, false, 0);
      Score(60, false, 0, 0);
      clawclamp();
      liftTo(600, 90, true, 15);
      wait1Msec(200);
      Score(-45, false, 0, 0);
      turn(-90 * s, 10);
      abspos = SensorValue[gyro];
      Score(-45, true, 40, 750);
      liftTo(300, 90, false, 0);
      clawopn(1100);
      /*if(abs(SensorValue[gyro] - abspos) > 5)
      {
        Score(5, false, 0, 0);//90 90 CUBE
        turn(SensorValue[gyro] - abspos, 0);
      }*/
        break;
    case 5: // Wall Stars / Cube
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "Wall Stars / Cube");
      Score(-10, false, 0, 0);
      clawopn(1800);
      Score(-10, false, 0, 0);
      abspos = SensorValue[gyro];
      Blocking(1300, 2, 40);
      liftTo(300, 90, false, 0);
      writeDebugStreamLine("%d",abs(SensorValue[gyro] - abspos));
    /*  if(abs(SensorValue[gyro] - abspos) > 5)
      {
        Score(5, false, 0, 0);//90 90 CUBE
        turn(-(SensorValue[gyro] - abspos), 0);
        Score(25, false, 0, 0);//90 90 CUBE

      }else
      {*/
        Score(30, false, 0, 0);//90 90 CUBE
      //}
      turn(90 * s, 0);
      clawopn(1200);
      Score(30, false, 0, 0);
      clawclamp();
      wait1Msec(400);
      liftTo(400, 90, true, 20);
      Score(-30, false, 0, 0);
      turn(-87 * s, 10);
      Score(-22, true, -19, 700);
      liftTo(300, 90, false, 0);
      clawopn(1400);
      Score(32, false, 0, 0);
      clawclamp();
```

```
      wait1Msec(200);
      liftTo(400, 70, true, 5);
      Score(-35, true, 33, 800);
      liftTo(300, 90, false, 0);
      clawopn(1100);
        break;
    case 6: // Wall Stars / Stars
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "Wall Stars / Stars");
      Score(-10, false, 0, 0);
      clawopn(1800);
      Score(-10, false, 0, 0);
      abspos = SensorValue[gyro];
      Blocking(1300, 2, 40);
      liftTo(300, 90, false, 0);
    /*  if(abs(SensorValue[gyro] - abspos) > 5)
      {
        Score(5, false, 0, 0);//90 90 CUBE
        turn(SensorValue[gyro] - abspos, 0);
        Score(25, false, 0, 0);//90 90 CUBE
      }else
      {*/
        Score(30, false, 0, 0);//90 90 CUBE
      //}
      Blocking(300, 1, 70);
      liftTo(500, 30, true, 0);
      Score(60, false, 0, 0);
      clawopn(1000);
      wait1Msec(500);
      turn(90 * s, 0);
      wait1Msec(300);
      liftTo(300, 90, false, 0);
      Score(40, false, 0, 0);
      Score(40, false, 0, 0);
      clawclamp();
      wait1Msec(500);
      liftTo(500, 70, true, 10);
      wait1Msec(200);
      Score(15, false, 0, 0);
      turn(-90 * s, 10);
      Score(-45, true, 40, 750);
      liftTo(300, 90, false, 0);
      clawopn(1100);
        break;
    case 7: // no auton
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "No Auto");
        break;
    case 8: // no auton
      clearLCDLine(1);                                                    // Clear line
      Score(38, false, 0, 0);//CUBE
      clawclamp();
      wait1Msec(800);
      liftTo(500, 70, true, 14);
      turn(-37 * s, 15);
      Score(-30, false, 0, 0);
      turn(-87 * s, 15);
      Score(-22, true, -19, 750);
      liftTo(300, 90, false, 0);
```

```
      Score(5, false, 0, 0);
      clawopn(1500);
      turn(40 * s, 0);
      Score(45, false, 0, 0);//CUBE
      clawclamp();
      wait1Msec(500);
      Score(-5, false, 0, 0);
      liftTo(450, 70, true, 14);
      Score(-30, true, 28, 800);//CUBE
      liftTo(300, 90, false, 0);
        break;
    case 9: // no auton
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "Empty Slot 2");
        break;
    case 10: // no auton
      clearLCDLine(1);                                                    // Clear line
      displayLCDCenteredString(1, "Empty Slot 3");
        break;
    default:
        break;
  }
}

task usercontrol()
{
  bLCDBacklight = true;
  string powerExpander;
  string mainBattery;
  const int SIZE = 10;
  int oldL[SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  int oldR[SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
  int sumL = 0, sumR = 0;
  int driveMap[128] = {
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    22,23,24,25,26,27,28,28,29,29,
    30,30,31,31,32,32,33,33,34,34,
    35,35,36,36,37,37,38,38,39,39,
    40,40,41,41,42,42,43,43,44,44,
    45,45,46,46,47,47,48,48,49,49,
    50,50,51,51,52,52,53,53,54,54,
    55,55,56,56,57,57,58,58,59,59,
    60,60,61,62,63,64,65,66,67,68,
    69,70,71,72,73,74,75,76,77,78,
    79,80,81,82,83,84,85,86,87,88,
    89,90,91,92,94,96,127,127};

  startTask(clawControl);
  startTask(liftControl);
  while (true)
  {
    oldL[9] = driveMap[abs(vexRT[Ch3])] * sgn(vexRT[Ch3]);
    oldR[9] = driveMap[abs(vexRT[Ch2])] * sgn(vexRT[Ch2]);
    sumL = 0;
    sumR = 0;
    for(int i = 0; i < SIZE - 1; i++){
      sumL += oldL[i];
      sumR += oldR[i];
```

```
      oldL[i] = oldL[i + 1];
      oldR[i] = oldR[i + 1];
    }
    sumL += oldL[9];
    sumR += oldR[9];
    motor[LF]  = sumL / SIZE;
    motor[LB]   = sumL / SIZE;
    motor[RF] = sumR / SIZE;
    motor[RB]  = sumR / SIZE;
    wait1Msec(25);
  }
}

void Blocking(int height, int time, int spd)
{
  while(SensorValue[lPot] < height)
  {
    motor[LLS] = (100);
    motor[LLD] = (100);
    motor[RLD] = (100);
    motor[RLS] = (100);
    motor[LF] = -spd;
    motor[LB] = -spd;
    motor[RF] = -spd;
    motor[RB] = -spd;
  }
  int signal = 0;
  clearTimer(T1);
  while(time1[T1] < time)
  {
    if(SensorValue[lPot] < height)
      signal = 100;
    if(SensorValue[lPot] > height)
      signal = -20;
    motor[LLS] = (signal);
    motor[LLD] = (signal);
    motor[RLD] = (signal);
    motor[RLS] = (signal);
    motor[LF] = -spd;
    motor[LB] = -spd;
    motor[RF] = -spd;
    motor[RB] = -spd;
  }
}

//All purpose autonomouse driving function
//Driving to distance and/or initiate scoring from a point (both on the field ar
void Score(int y, bool score, int d, int h)
{
  int signal;
  y = y * 28;
  d = d * 28;
  if(y > 0)
  {
    bool fop = true;
    bool reached = false;
    int Le = 0, Re = 0;
    int modR, modL;
    int acc = 20;
```

```
    float defspd = 100;
    SensorValue[dL] = 0;
    SensorValue[dR] = 0;
    while(reached == false)
    {
      Le = (y - SensorValue[dL]);
      Re = (y - SensorValue[dR]);
      if((SensorValue[dL] - SensorValue[dR]) > acc)
      {
        modL = SensorValue[dR] - SensorValue[dL];
        modR = SensorValue[dL] - SensorValue[dR];
      }
      else if((SensorValue[dR] - SensorValue[dL]) > acc)
      {
        modL = SensorValue[dR] - SensorValue[dL];
        modR = SensorValue[dL] - SensorValue[dR];
      }else
      {
        modL = 0;
        modR = 0;
      }
      if(fop)// Start motors LLSowly once
      {
        for(int p = 0; p < 5; p ++)
        {
          motor[LF] = defspd / (5 - p);
          motor[LB] = defspd / (5 - p);
          motor[RF] = defspd / (5 - p);
          motor[RB] = defspd / (5 - p);
          wait1Msec(75);
        }
        fop = false;
      }
//      writeDebugStreamLine("%d %d", Le, Re);
      motor[LF] = defspd + (modL);//FLe;
      motor[LB] = defspd + (modR);//FRe;
      motor[RF] = defspd + (modL);//BLe;
      motor[RB] = defspd + (modR);//BRe;
      if(abs(Le + Re)/2 < 500)
      {
        defspd = 70;
      }else if(abs(Le + Re)/2 < 300)
      {
        defspd = 30;
      }else if(abs(Le + Re)/2 < 100)
      {
        defspd = 10;
      }
      if(abs(Le + Re)/2 < 10)
      {
        reached = true;
      }else
      {
        reached = false;
      }
    }
    setDrive(defspd * 0.75);
    wait1Msec(5);
    setDrive(-1 * (defspd * 3));
```

```
    wait1Msec(100);
    setDrive(0);
  }else // Going backwards
  {
    bool fop = true;
    bool reached = false;
    int Le = 0, Re = 0;
    int modR, modL;
    int acc = -20;
    int shift = 0;
    float defspd = 100;
    SensorValue[dL] = 0;
    SensorValue[dR] = 0;
    while(reached == false && bIfiRobotDisabled == false)
    {
      if(SensorValue[lPot] > 350)
      {
        signal = 10;
        //writeDebugStreamLine("%d", signal);
        motor[LLS] = (signal);
        motor[LLD] = (signal);
        motor[RLD] = (signal);
        motor[RLS] = (signal);
      }
      //Calculating error
      Le = (y - SensorValue[dL]);
      Re = (y - SensorValue[dR]);
      if((SensorValue[dL] - SensorValue[dR]) < acc)
      {
        modR = SensorValue[dR] - SensorValue[dL];
        modL = SensorValue[dL] - SensorValue[dR];
      }
      else if((SensorValue[dR] - SensorValue[dL]) < acc)
      {
        modR = SensorValue[dR] - SensorValue[dL];
        modL = SensorValue[dL] - SensorValue[dR];
      }else
      {
        modL = 0;
        modR = 0;
      }
      if(fop) // Start motors LLSowly once
      {
        for(int p = 0; p < 5; p ++)
        {
          motor[LF] = -(defspd / (5 - p));
          motor[LB] = -(defspd / (5 - p));
          motor[RB] = -(defspd / (5 - p));
          motor[RF] = -(defspd / (5 - p));
          wait1Msec(75);
        }
        fop = false;
      }
      if(score)//If scoring while going backwards
      {
        bool fp = false; // Only loop once
        bool keepup = true;
        if(fp == false && abs(SensorValue[dL]) < abs(y) && abs(SensorValue[dR])
        {
```

```
          //clamp cPot closed
          motor[cL] = 100;
          motor[cR] = 100;
          while(SensorValue[lPot] < 1400 && bIfiRobotDisabled == false)
          {
            //writeDebugStreamLine("%d", signal);
            if(keepup)
            {
              motor[LLS] = (10);
              motor[LLD] = (10);
              motor[RLD] = (10);
              motor[RLS] = (10);
            }
            if(SensorValue[lPot] < 1200 && (abs(SensorValue[dL]) > abs(d) && abs
            { // Lift after reaching setpoint
              if((SensorValue[cPot] < 1500 && SensorValue[lPot] > h))
              {
                //Open cPot after lift has reached a point
                motor[cL] = -127;
                motor[cR] = -127;
              }else if(SensorValue[cPot] > 1500){
                motor[cL] = 20;
                motor[cR] = 20;
              }

              //keep lifting lift
              keepup = false;
              motor[LLS] = (100);
              motor[LLD] = (100);
              motor[RLD] = (100);
              motor[RLS] = (100);
              wait1Msec(25);
            }
            if(SensorValue[dR] > y && SensorValue[dL] > y)//If needed keep going
            {
              motor[LF] = -(defspd + (modL ) - (float)shift);//FLe;
              motor[LB] = -(defspd + (modR));//FRe;
              motor[RF] = -(defspd + (modL) - (float)shift);//BLe;
              motor[RB] = -(defspd + (modR));//BRe;
            }else
            {
              motor[LF] = (0);//FLe;
              motor[LB] = (0);//FRe;
              motor[RF] = (0);//BLe;
              motor[RB] = (0);//BRe;
            }
            if(SensorValue[dL] < y && SensorValue[dR] < y && SensorValue[lPot] >
            {
              motor[cL] = 40;
              motor[cR] = 40;
              motor[LLS] = -(40);
              motor[LLD] = -(40);
              motor[RLD] = -(40);
              motor[RLS] = -(40);
              wait1Msec(1000);
              //Break if setpoint is reached
              reached = true;
              break;
            }
```

```
          }
          if(/*SensorValue[dL] < (y+100) && */SensorValue[dR] < y && SensorValue
          {
          //  writeDebugStreamLine("2");
            reached = true;
          }
        }
        if(SensorValue[cPot] > 1500){
          motor[cL] = 20;
          motor[cR] = 20;
        }
      }else // If going backwards but NOT scoring
      {
        if(SensorValue[dL] > y && SensorValue[dR] > y)//Keeping backing up till
        {
          motor[LF] = -(defspd + (modL ) - (float)shift);//FLe;
          motor[LB] = -(defspd + (modR));//FRe;
          motor[RF] = -(defspd + (modL) - (float)shift);//BLe;
          motor[RB] = -(defspd + (modR));//BRe;
        }else
        {
          motor[LF] = (0);//FLe;
          motor[LB] = (0);//FRe;
          motor[RF] = (0);//BLe;
          motor[RB] = (0);//BRe;
        }
      }
      if((abs(Le) + abs(Re))/2 < 50)
      {
        setDrive(0);
        reached = true;
      }
    } // end of reached loop
    setDrive(0);
  }
//  writeDebugStreamLine("Done");
  stopAll();
}

void turn(int x, int holding)
{
  // - is right
  SensorValue[gyro] = 0;
  int error = abs(x * 10);
  float signal = 0;
  motor[LLS] = (holding);
  motor[LLD] = (holding);
  motor[RLD] = (holding);
  motor[RLS] = (holding);
  clearTimer(T1);
  while(abs(error) > 10)
  {
    error = abs(x * 10) - abs(SensorValue[gyro]);
    signal = (error / 450.0) * 120;
    if(signal < 40)
      signal = 40;

    if(x > 0)
    {//right fwd, left back
```

```
      motor[LF] = -(signal);
      motor[LB] = -(signal);
      motor[RF] = (signal);
      motor[RB] = (signal);
    }else
    {//left fwd, right back
      motor[LF] = (signal);
      motor[LB] = (signal);
      motor[RF] = -(signal);
      motor[RB] = -(signal);
    }
  }
  if(x > 0)
    {//right fwd, left back
      motor[LF] = (80);
      motor[LB] = (80);
      motor[RF] = -(80);
      motor[RB] = -(80);
      wait1Msec(100);
    }else
    {//left fwd, right back
      motor[LF] = -(80);
      motor[LB] = -(80);
      motor[RF] = (80);
      motor[RB] = (80);
      wait1Msec(100);
    }
    setDrive(0);
}

void setDrive(int x)
{
  motor[LF] = (x);//FLe;
  motor[LB] = (x);//FRe;
  motor[RF] = (x);//BLe;
  motor[RB] = (x);//BRe;
}
void stopAll()
{
  motor[LF] = (0);//FLe;
  motor[LB] = (0);//FRe;
  motor[RF] = (0);//BLe;
  motor[RB] = (0);//BRe;
  motor[LLS] = (0);
  motor[LLD] = (0);
  motor[RLD] = (0);
  motor[RLS] = (0);
}


void clawopn(int x)
{
  bool closing = false;
  while(SensorValue[cPot] < x)
  {
    motor[cR] = -70;
    motor[cL] = -70;
    closing = false;
  }
```

```
  while(SensorValue[cPot] > x)
  {
    motor[cR] = 50;
    motor[cL] = 50;
    closing = true;
  }
  if(closing == false)
  {
    motor[cR] = 70;
    motor[cL] = 70;
    wait1Msec(60);
  }
  if(closing)
  {
    motor[cR] = -50;
    motor[cL] = -50;
    wait1Msec(60);
  }
    motor[cR] = 0;
    motor[cL] = 0;
}
void clawclamp(){
    motor[cR] = 100;
    motor[cL] = 100;
    wait1Msec(200);
}
void liftTo(int x, int spd, bool hold, int holding)
{
  if(SensorValue[lPot] < x)
  {
    while(SensorValue[lPot] < x)
    {
      motor[LLS] = (spd);
      motor[LLD] = (spd);
      motor[RLD] = (spd);
      motor[RLS] = (spd);
    }
  }
  if(SensorValue[lPot] > x)
  {
    while(SensorValue[lPot] > x)
    {
      motor[LLS] = -(spd);
      motor[LLD] = -(spd);
      motor[RLD] = -(spd);
      motor[RLS] = -(spd);
    }
  }
  if(hold)
  {
    motor[LLS] = (holding);
    motor[LLD] = (holding);
    motor[RLD] = (holding);
    motor[RLS] = (holding);
  }else
  {
    motor[LLS] = (0);
    motor[LLD] = (0);
    motor[RLD] = (0);
```

```
    motor[RLS] = (0);
  }
}
```