```
/*****************************************************************************************************
define statements
*****************************************************************************************************/
#define LED_PIN 13          //onboard LED
#define MOTOR1_ENABLE 2     //Motor driver pins
#define MOTOR1_IN1 4
#define MOTOR1_IN2 3
#define MOTOR2_ENABLE 5
#define MOTOR2_IN1 6
#define MOTOR2_IN2 7

#define SPEAKEROUT 31       //speaker output pin

#define comp_filter_const 0.992    //complimentary filter constant (0<=const<=1)
#define DT 25    //sample time (ms)
#define motorStart 25       //motor PWM functions will receive this as min value
#define LeftMotorTrim 0     //adjust these to get robot to drive in straight lines
#define RightMotorTrim 1

#define CE_PIN   12       //wireless module CE pin
#define CSN_PIN 13       //wireless module CSN pin

#define angleAverageNum 1

#define remoteYaxisNatural 120    //resting state of Y (vertical) axis on remote control.
#define remoteXaxisNatural 122    //resting state of X (Horizontal) axis on remote control.

#define TrayLEDPin 28
/*****************************************************************************************************
Libraries
*****************************************************************************************************/
#include "Wire.h"  // Arduino Wire library is required if I2Cdev I2CDEV_ARDUINO_WIRE implementation is used in I2Cdev.h
#include "I2Cdev.h"  // I2Cdev and MPU6050 must be installed as libraries, or else the .cpp/.h files for both classes must be in the
include path of your project
#include "MPU6050.h"
#include "Math.h"
#include <LiquidCrystal.h>
#include <SPI.h>
#include <RF24.h>
#include <stdint.h>

LiquidCrystal lcd(43, 45, 47, 49, 51, 53);  //(RS, En, D4, D5, D6, D7)    // initialize the library with the numbers of the interface
pins
#include <PID_v1.h>

/*-----( Declare objects )-----*/
MPU6050 accelgyro;    //define MPU6050 as accelgyro object
RF24 radio(CE_PIN, CSN_PIN); // Create a Radio object
```

```
/*-----( Declare Variables )-----*/
const uint64_t pipe = 0xE8E8F0F0E1LL; // Radio: Define the transmit pipe
uint8_t  dataBuffer[32];
int lastData[2] = {0,0};

float gyro_scale = 131.0;  //value = [(2^16)-1]/(gyro scale (degrees/second))
float accelerometer_scale = 50.0;    //scale value for raw accelerometer data from IMU

float idealAngle= 93.0;
float myKp = 154.0;  //154.0;    values that work reasonably well on carpet
float myKi = 105.0;  //225.0;
float myKd = 0.90;  //0.00;

float Setpoint, Input, Output;    //PID variables
PID BalancePID(&Input, &Output, &Setpoint,myKi, myKp, myKd, DIRECT);    //Specify the links and initial tuning parameters

float currentAngle, accelAngle, gyroAngle, gyroRateY;  //angles and gyro data
float lastAngle = idealAngle;
float angleArray[angleAverageNum];
int avgAngleIndex = 0;     //create index for average angle array at beginning of array

int LeftMotorDrive = 0;//128;
int RightMotorDrive = 0;//-128;

int beginningTime, endTime, loopTime, computeBeginTime, computeLastTime;    //debug variables for timing

bool blinkState = false;    //blink state variable (Debug, but useful for program running indicator)
bool TrayLEDState = 1;

unsigned int loopCount = 0;    //used so remote vals are only read every <x> control loops

signed int remote_yaxisSetpoint;
/****************************************************************************************************
Initialization
****************************************************************************************************/
void setup() {
    Serial.begin(115200);
    Wire.begin();  // join I2C bus (I2Cdev library doesn't do this automatically)

    pinMode(LED_PIN, OUTPUT);        // configure Arduino LED alive blinker

    //Assign Motor Output Pins, and set them Low (off):
    pinMode(MOTOR1_ENABLE, OUTPUT);
    pinMode(MOTOR1_IN1, OUTPUT);
    pinMode(MOTOR1_IN2, OUTPUT);
    pinMode(MOTOR2_ENABLE, OUTPUT);
    pinMode(MOTOR2_IN1, OUTPUT);
    pinMode(MOTOR2_IN2, OUTPUT);

    digitalWrite(MOTOR1_ENABLE, LOW);
    digitalWrite(MOTOR1_IN1, LOW);
```

```
    digitalWrite(MOTOR1_IN2, LOW);
    digitalWrite(MOTOR2_ENABLE, LOW);
    digitalWrite(MOTOR2_IN1, LOW);
    digitalWrite(MOTOR2_IN2, LOW);

    //Assign Speaker Output Pin and set it Low (off):
    pinMode(SPEAKEROUT, OUTPUT);
    digitalWrite(SPEAKEROUT, LOW);

    //Assign Tray Led Signal Pin and set it High (on):
    pinMode(TrayLEDPin, OUTPUT);
    digitalWrite(TrayLEDPin, LOW);

    delay(500);  //delay to allow power to reach a stable state
    // set up the LCD's number of columns and rows, and set curson to beginning:
    lcd.begin(20, 4);
    lcd.setCursor(0, 0);
    lcd.print("Angle: ");

    accelgyro.initialize();        // initialize MPU-6050
    accelgyro.setFullScaleAccelRange(MPU6050_ACCEL_FS_8);    //set accelerometer scale to 8G
    accelgyro.setDLPFMode(2);    //set the Digital LPF to: Accel - 94 Hz, Gyro - 98 Hz

    radio.begin();
    radio.openReadingPipe(1,pipe);
    radio.startListening();

    getMotionData();    //get initial data for variables
    getCurrentAngle(accelAngle);      //initialize other values
    currentAngle = accelAngle;

    BalancePID.SetSampleTime(DT);
    BalancePID.SetOutputLimits(-255+motorStart,255-motorStart);

    //initialize the PID variables we're linked to
    Input = accelAngle;
    Setpoint = idealAngle;

    while(abs(accelAngle - idealAngle) > 1){  //wait until robot is vertical to turn on motors
      getMotionData();
    }

    currentAngle = accelAngle;
    SpeakerBeep();  //500Hz for ~500ms
    BalancePID.SetMode(AUTOMATIC);  //turn on PID

    computeLastTime = millis();    //initialize computLastTime var for PID timing
  }

/****************************************************************************************************
Main Program
```

```
****************************************************************************************************/
void loop() {

        if(loopCount == 5){
          loopCount = 0;
          if(checkRadio(dataBuffer)){
          //debug lines: print wireless data buffer to serial terminal
//            for(int i=0; i<sizeof(dataBuffer)-5; i++){
//            Serial.print(i); Serial.print(": "); Serial.print(dataBuffer[i]); Serial.print("  ");
//            }
//          Serial.println();
          }
        }

            if(millis() - computeLastTime >= (DT)){
            loopCount++;
            computeLastTime = millis();
            getMotionData();
            getCurrentAngle(UpdateAverageAngle(accelAngle));
            Input =  currentAngle;
            BalancePID.Compute();
            MotorLogic(Output);
            lcd.setCursor(7, 0);        //display current angle
            lcd.print(currentAngle);    // ^ ^ ^ ^ ^ ^ ^ ^ ^ ^
            getSerialConstant();
            UpdateTrimAndRemote();
          }

    // blink LED to indicate activity
    blinkState = !blinkState;
    digitalWrite(LED_PIN, blinkState);
    delay(2);    //short delay for stability
}

/****************************************************************************************************
Functions
****************************************************************************************************/
//UpdateTrimAndRemote : updates the setpoint and L/R motor values based on the Trim potentiometer mounted on the robot
//                        and the data from the remote control
void UpdateTrimAndRemote(){
  signed int potVal, remote_yaxis, remote_xaxis;
       potVal = analogRead(A0);
       potVal -= 512;
       potVal /= 6;

       if(dataBuffer[0] != 0){
       remote_yaxis = ((dataBuffer[0]+lastData[0])/2-remoteYaxisNatural);    //average two values of remote yaxis data and scale to -
127<x<128
       lastData[0] = dataBuffer[0];    //maintain value used in average

         if(remote_yaxisSetpoint < remote_yaxis){
```

```
        remote_yaxisSetpoint += (remote_yaxis - remote_yaxisSetpoint)/2;
        }
        else if(remote_yaxisSetpoint > remote_yaxis){
        remote_yaxisSetpoint -= (remote_yaxisSetpoint - remote_yaxis)/2;
        }


      remote_xaxis = ((dataBuffer[1]+lastData[1])/2-remoteXaxisNatural)/2;    //average two values of remote xaxis data and scale to
(-127<x<128)/2
      lastData[1] = dataBuffer[1];    //maintain value used in average
        LeftMotorDrive = remote_xaxis;
        RightMotorDrive = -remote_xaxis;
      }
      else{
       remote_yaxis = 0;
      LeftMotorDrive = 0;
       RightMotorDrive = 0;
      }

      Setpoint  = idealAngle + (float)potVal*0.01 + (float)remote_yaxisSetpoint*0.02;

      lcd.setCursor(7, 2);         //display current angle
      lcd.print(Setpoint);    // ^ ^ ^ ^ ^ ^ ^ ^ ^ ^

      if(dataBuffer[3] == 1){
       ToggleTrayLED();
      }
}


//getMotionData requests data from the IMU and modifies the values in accelAngle and gyroRateY
void getMotionData(){
      signed int ax, az;
      ax = 0; az = 0; gyroRateY = 0;
      int i;

    // read raw accel/gyro measurements from device:
    for (i=0; i<10; i++){
          ax += accelgyro.getAccelerationX();
          az += accelgyro.getAccelerationZ();
          gyroRateY += accelgyro.getRotationY();
    }

          ax = ax/(float)i;
          az = az/(float)i;

          gyroRateY = -gyroRateY/((float)gyro_scale*(float)i);

          ax/=(float)accelerometer_scale;
          az/=(float)accelerometer_scale;
```

```
          accelAngle = (float)(atan2(ax,az))*RAD_TO_DEG;    //calculate angle from accelerometer data

}

//getCurrentAngle updates the current angle variable based on IMU data and the complementary filter
void getCurrentAngle(float accel_Angle){
  currentAngle = (float)(comp_filter_const)*(currentAngle+(gyroRateY*((float)DT/1000.0)))+(1-comp_filter_const)*accel_Angle;
}

//UpdateAverageAngle averages out a number of previous sets of IMU data. Averaging resulted in too much delay, so this is implemented
as the average of a single reading
//Returns: (float) average angle
float UpdateAverageAngle(float newAngle){
  float averageAngle = 0;

  angleArray[avgAngleIndex] = newAngle;    //insert new angle into array

  if(avgAngleIndex == (angleAverageNum - 1)){  //increment pointer; wrap around if at end
    avgAngleIndex = 0;
  }
    else{
     avgAngleIndex++;    //move pointer
    }

    //compute average
    int j;
    for(j=0; j<angleAverageNum; j++){
      averageAngle += angleArray[j];
    }
    averageAngle = averageAngle/(float)j;

  return averageAngle;
}

//translate PID output and remote input into valid PWM and direction data for the motors
void MotorLogic(int value){
  signed int LeftValue = value + LeftMotorDrive;
  signed int RightValue = value + RightMotorDrive;

 if((LeftValue) < 0){
      Motor1Control(0, (abs(LeftValue))+motorStart);
 }
 else if((LeftValue) >= 0) {
   Motor1Control(1, (LeftValue)+motorStart);
 }

  if(RightValue < 0){
      Motor2Control(0, (abs(RightValue))+motorStart);
 }
 else if(RightValue >= 0) {
   Motor2Control(1, (RightValue)+motorStart);
```

```
 }
}


//Motor*Control modifies the pins that control the direction, and also sets the PWM duty cycle of a motor
void Motor1Control(bool rot_direction, int rot_speed){
//Motor Control: 0-->  backwards    1--> forwards
  digitalWrite(MOTOR1_ENABLE, LOW);

  if(rot_direction == 0){
   digitalWrite(MOTOR1_IN1, HIGH);
   digitalWrite(MOTOR1_IN2, LOW);
  }
  else {
   digitalWrite(MOTOR1_IN1, LOW);
   digitalWrite(MOTOR1_IN2, HIGH);
  }

  analogWrite(MOTOR1_ENABLE, rot_speed + LeftMotorTrim);
}

//Motor*Control modifies the pins that control the direction, and also sets the PWM duty cycle of a motor
void Motor2Control(bool rot_direction, int rot_speed){
//Motor Control: 0-->  backwards    1--> forwards
  digitalWrite(MOTOR2_ENABLE, LOW);

  if(rot_direction == 0){
   digitalWrite(MOTOR2_IN1, HIGH);
   digitalWrite(MOTOR2_IN2, LOW);
  }
  else {
   digitalWrite(MOTOR2_IN1, LOW);
   digitalWrite(MOTOR2_IN2, HIGH);
  }

  analogWrite(MOTOR2_ENABLE, rot_speed + RightMotorTrim);
}

//checkRadio checks if the radio is present and operating, and if so checks to see if data is available and reads it into the passed
array if it is
//Returns: 1 if radio is operating, 0 else
bool checkRadio(uint8_t* array){
   if ( radio.available() )
  {

    // Read the data payload until we've received everything
    bool done = false;
    while (!done)
    {
      // Fetch the data payload
      done = radio.read(array, sizeof(array));
```

```
    }
    return 1;
  }
  else
  {
      return 0;
  }
}

//SpeakerBeep generates a square wave at a set frequency for a set duration
void SpeakerBeep(){

    for(int c = 0; c<350; c++){
        digitalWrite(SPEAKEROUT, HIGH);
        delayMicroseconds(1000);
        digitalWrite(SPEAKEROUT, LOW);
        delayMicroseconds(1000);
    }
    digitalWrite(SPEAKEROUT, LOW);
}

//ToggleTrayLED toggles the output pin controlling the tray LED sequence
void ToggleTrayLED(){
  if(TrayLEDState == 1){
    TrayLEDState = 0;
    digitalWrite(TrayLEDPin, TrayLEDState);
  }
  else{
    TrayLEDState = 1;
    digitalWrite(TrayLEDPin, TrayLEDState);
  }
}

//getSerialConstant allows us to use the arduino serial monitor to view and modify PID, angle, and other parameters while the device
is running.
//Key: <parameter>: <increase>/<decrease>    //any combination of the following letters followed by "Send" in the serial monitor
window.
//Proportional: q/a
//Integral: w/s
//Derivative: e/d
//Setpoint angle: r/f
void getSerialConstant(){
 while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();

      switch (inChar) {
    case 'q':
        myKp += 1.0;
      break;
    case 'a':
```

```
        myKp -= 1.0;
        if(myKp<0){
         myKp = 0;
        }
      break;
    case 'w':
        myKi += 1.0;
      break;
    case 's':
        myKi -= 1.0;
        if(myKi<0){
         myKi = 0;
        }
      break;
    case 'e':
        myKd += 0.05;
      break;
    case 'd':
        myKd -= 0.05;
        if(myKd<0){
         myKd = 0;
        }
      break;
    case 'r':
        idealAngle += 0.01;
      break;
    case 'f':
        idealAngle -= 0.01;
      break;

    default: break;

    }
  }
  Serial.print(myKp); Serial.print("  ");
  Serial.print(myKi); Serial.print("  ");
  Serial.print(myKd); Serial.print("  ");
  Serial.println(Output); Serial.print("  ");
//  Serial.print(loopTime); Serial.print("  ");
//  Serial.print(currentAngle); Serial.print("  ");
//  Serial.print(accelAngle); Serial.print("  ");
//  Serial.print(Input); Serial.print("  ");
//  Serial.println(idealAngle);

//  Setpoint = idealAngle;
  BalancePID.SetTunings(myKp, myKi, myKd);
}
```