```
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
//#include <HCSR04.h> //Sonar

// Pins for trigger and echo on SR04
int triggerPin = 4;
int echoPin = 5;
double duration, distance;

// Initialize all variables for PIDs
unsigned long prevMillis = 0;
unsigned long currentMillis = 0;
unsigned long dt = 0;
double setPoint = 0;
double currentDistance = 0;
double error = 0;
double lastError = 0;
double cumError = 0;
double P = 0;
double I = 0;
double D = 0;
int c = 0;
int counterAnomaly = 0;

// PID Tuning
double Kp = 0.06; // bang bang control and keep it
low to avoid overshoot
```

```
double Ki = 0.001; // remove steady state error
with this
double Kd = 3.5; // increase it to measure small
change in height and adjust the propellers

//Offsets to avoid drifting problems
int yawOffset = 0;
int pitchOffset = -1;
int rollOffset = 3;

////////////////////// PPM
CONFIGURATION//////////////////////////
#define channel_number 6  //set the number of
channels
#define sigPin 2  //set PPM signal output pin on
the arduino
#define PPM_FrLen 27000  //set the PPM frame length
in microseconds (1ms = 1000µs)
#define PPM_PulseLen 400  //set the pulse length
//////////////////////////////////////////////////////
//////////////

int ppm[channel_number];

const uint64_t pipeIn =  0xE8E8F0F0E1LL;

RF24 radio(9, 10);
```

```
// The sizeof this struct should not exceed 32 bytes
struct MyData {
  byte throttle;
  byte yaw;
  byte pitch;
  byte roll;
  byte AUX1;
  byte AUX2;
};

MyData data;

void resetData()
{
  // 'safe' values to use when no radio input is
detected
  data.throttle = 0;
  data.yaw = 127;
  data.pitch = 127;
  data.roll = 127;
  data.AUX1 = 0;
  data.AUX2= 0;

  setPPMValuesFromData();
}

void setPPMValuesFromData()
{
```

```
  ppm[0] = map(data.throttle, 0, 255, 1000, 1750);
//Using 5030 props instead of 1045
  ppm[1] = map(data.yaw + yawOffset,      0, 255,
1000, 2000);
  ppm[2] = map(data.pitch + pitchOffset,    0, 255,
1000, 2000);
  ppm[3] = map(data.roll + rollOffset,     0, 255,
1000, 2000);
  ppm[4] = map(data.AUX1,     0, 1, 1000, 2000);
  ppm[5] = map(data.AUX2,     0, 1, 1000, 2000);



  }

/**************************************************/


void setupPPM() {
  pinMode(sigPin, OUTPUT);
  digitalWrite(sigPin, 0);  //set the PPM signal
pin to the default state (off)


  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register (not very
important, sets the timeout for the first interrupt)
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
```

```
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5
microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare
interrupt
  sei();
}

void setup()
{
  Serial.begin(9600); //Serial communication
  resetData();
  setupPPM();

  //Setup sonar
//  HCSR04.begin(triggerPin, echoPin);
  pinMode(triggerPin, OUTPUT);
  pinMode(echoPin, INPUT);

  // Set up radio module
  radio.begin();
  radio.setDataRate(RF24_250KBPS); // Both
endpoints must have this set the same
  radio.setAutoAck(false);

  radio.openReadingPipe(1,pipeIn);
  radio.startListening();
}
```

```
/**************************************************/

unsigned long lastRecvTime = 0;

void recvData()
{
  while ( radio.available() ) {
    radio.read(&data, sizeof(MyData));
    lastRecvTime = millis();
  }
}

/**************************************************/

void loop()
{
  recvData();

  unsigned long now = millis();
  if ( now - lastRecvTime > 1000 ) {
    // signal lost?
    resetData();
  }
  //  set PIDs here with data from sonar
  int throttle = data.throttle;
  boolean aux1 = data.AUX1;
  if (aux1==0) {
    throttle = 0;
```

```
    c = 0;
    data.throttle = throttle;
    cumError = 0; //Set this to 0 to start integral
error from scratch
    Serial.println("Drone not started");
  } else {
    if (c<2) {
      throttle = 0;
      prevMillis = millis();
      c += 1;
      data.throttle = throttle;
      counterAnomaly = 0;
      Serial.println("Drone started and minimum
throttle given\n\n");
    } else {
//      Serial.println("Entered main body");
      currentMillis = millis();
      dt = currentMillis - prevMillis;
      setPoint = map(throttle, 0, 255, 5, 100);

      double distance = computeDistance();
      distance -= 8;
      if (distance>200 && counterAnomaly <1000) {
        distance = 0;
        counterAnomaly += 1;
      }

//      Serial.println(distance);
```

```
      while (distance<0 || distance>100) distance =
computeDistance();

//      Serial.println("got the distance");
      currentDistance = distance;
      error = setPoint - currentDistance;
      cumError += error;
      P = Kp*error;
      I = Ki*cumError*dt;
      D = Kd*(error-lastError)/dt;
      throttle = P+I+D;
      throttle = map(throttle, 5, 100, 0, 255);
      lastError = error;
      prevMillis = currentMillis;
//      Remove anomalies in throttle values
      Serial.print("Actual Throttle with PID: ");
      Serial.println(map(throttle, 0, 255, 1000,
1750));
      if(throttle<0){
        data.throttle = 0;
      } else if (throttle > 255){
        data.throttle = 255;
      } else {
        data.throttle = throttle;
      }
      Serial.print("Setpoint: ");
      Serial.println(setPoint);
      Serial.print("Current Distance: ");
```

```
      Serial.println(currentDistance);
//      Serial.print("Error: ");
//      Serial.println(error);
      Serial.print("Current Throttle: ");
      Serial.println(map(data.throttle, 0, 255,
1000, 1750));

      Serial.println("\n\n\n"); //For Testing
Purpose
    }
  }
  setPPMValuesFromData();
}

/**************************************************/

//#error Delete this line befor you cahnge the
value (clockMultiplier) below
#define clockMultiplier 2 // set this to 2 if you
are using a 16MHz arduino, leave as 1 for an 8MHz
arduino

ISR(TIMER1_COMPA_vect){
  static boolean state = true;

  TCNT1 = 0;

  if ( state ) {
```

```
    //end pulse
    PORTD = PORTD & ~B00000100; // turn pin 2 off.
Could also use: digitalWrite(sigPin,0)
    OCR1A = PPM_PulseLen * clockMultiplier;
    state = false;
  }
  else {
    //start pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    PORTD = PORTD | B00000100; // turn pin 2 on.
Could also use: digitalWrite(sigPin,1)
    state = true;

    if(cur_chan_numb >= channel_number) {
      cur_chan_numb = 0;
      calc_rest += PPM_PulseLen;
      OCR1A = (PPM_FrLen - calc_rest) *
clockMultiplier;
      calc_rest = 0;
    }
    else {
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) *
clockMultiplier;
      calc_rest += ppm[cur_chan_numb];
      cur_chan_numb++;
    }
```

```
  }
}

double computeDistance(){
//      Calculate 10 distances and its average
  distance = 0;
  for (int i = 0; i< 10; i++){
    digitalWrite(triggerPin, LOW);
    delayMicroseconds(2);
//      Sets the trigPin on HIGH state for 10 micro
seconds
    digitalWrite(triggerPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(triggerPin, LOW);
//      Reads the echoPin, returns the sound wave
travel time in microseconds
    duration = pulseIn(echoPin, HIGH);
//      Calculating the distance
    distance += duration*0.034/2;
  }
  return distance /= 10;
}
```