**Transmitter code:**

struct Signal {

  byte throttle;

  //byte pitch;

  //byte roll;

  byte yaw;

 // byte aux1;

//  byte aux2;

};

Signal data;

void ResetData() {

  data.throttle = 127;   // Motor stop

  //data.pitch = 1b 27;    // Center

  //data.roll = 127;     // Center

  data.yaw = 127;     //Center

 // data.aux1 = 127;

//  data.aux2 = 127;    //

}

void setup() {

  Serial.begin(9600);

}

// Joystick center and its borders |

int mapJoystickValues(int val, int lower, int middle, int upper, bool reverse) {

  val = constrain(val, lower, upper);

  if (val < middle)

    val = map(val, lower, middle, 0, 128);

  else

    val = map(val, middle, upper, 128, 255);

  return (reverse ? 255 - val : val);

}

void loop() {

  // Control Stick Calibration |

  // Setting may be required for the correct values of the control levers. | .

  data.throttle = mapJoystickValues(analogRead(A0), 12, 524, 1020, true);  // "true" or "false" for signal direction

  //data.roll = mapJoystickValues(analogRead(A3), 12, 524, 1020, true);

  //data.pitch = mapJoystickValues(analogRead(A2), 12, 524, 1020, false);     // "true" or "false" for servo direction

  data.yaw = mapJoystickValues(analogRead(A1), 12, 524, 1020, false);       // "true" or "false" for servo direction |

 // data.aux1 = mapJoystickValues(analogRead(A6), 12, 524, 1020, true);     // "true" or "false" for servo direction

  //data.aux2 = mapJoystickValues(analogRead(A7), 12, 524, 1020, true);     // "true" or "false" for servo direction

  Serial.write((byte\*)&data, sizeof(Signal));

  //Serial.println("throttle: ");

  //Serial.print(data.throttle);

  //Serial.println(" | pitch: ");

  //Serial.print(data.pitch);

  //Serial.println(" | roll: ");

  //Serial.print(data.roll);

  //Serial.println(" | yaw: ");

  //Serial.print(data.yaw);

  //Serial.print(" | aux1: ");

  //Serial.print(data.aux1);

  //Serial.print(" | aux2: ");

  //Serial.println(data.aux2);

  delay(100);  // Optional delay between readings

}

**Receiver code:**

#include <Servo.h>

Servo ch1;

Servo ch2;

struct Signal {

  byte throttle;

  byte yaw;

};

Signal data;

// PPM Signal Parameters

#define CHANNELS 2                // Number of channels

#define PPM\_PIN 5                 // PPM signal output pin

#define PPM\_FRAME\_LENGTH 22500    // PPM frame length in microseconds

#define PPM\_PULSE\_LENGTH 300      // PPM pulse length for each channel in microseconds

unsigned long lastPPMTime = 0;

void setup() {

  ch1.attach(2);

  ch2.attach(3);

  pinMode(PPM\_PIN, OUTPUT);

  Serial.begin(9600);

}

void loop() {

  if (Serial.available() >= sizeof(Signal)) {

    Serial.readBytes((char\*)&data, sizeof(Signal));  // Read the received data

    int ch\_width\_1 = map(data.throttle, 0, 255, 1000, 2000);  // Throttle channel (pin D2)

    int ch\_width\_2 = map(data.yaw, 0, 255, 1000, 2000);       // Yaw channel (pin D3)

    //int ch\_width\_1=2000;

    //int ch\_width\_2=1500;

    ch1.writeMicroseconds(ch\_width\_1);

    ch2.writeMicroseconds(ch\_width\_2);

    Serial.print("Throttle: ");

    Serial.print(ch\_width\_1);

    Serial.print(" | Yaw: ");

    Serial.println(ch\_width\_2);

    // Generate PPM signal

    GeneratePPMSignal(ch\_width\_1, ch\_width\_2);

  }

}

void GeneratePPMSignal(int throttleWidth, int yawWidth) {

  static int channelData[CHANNELS];

  static int currentIndex = 0;

  unsigned long currentTime = micros();

  if (currentTime - lastPPMTime >= PPM\_PULSE\_LENGTH) {

    lastPPMTime = currentTime;

    if (currentIndex >= CHANNELS) {

      currentIndex = 0;

      // Print the pulse lengths for each channel

      for (int i = 0; i < CHANNELS; i++) {

        Serial.print("Channel ");

        Serial.print(i + 1);

        Serial.print(" Pulse Length: ");

        Serial.print(channelData[i]);

        Serial.print(" | ");

      }

      Serial.println();

    }

    if (currentIndex == 0) {

      channelData[currentIndex] = throttleWidth;

    } else if (currentIndex == 1) {

      channelData[currentIndex] = yawWidth;

    }

    // Output PPM signal

    digitalWrite(PPM\_PIN, HIGH);

    delayMicroseconds(channelData[currentIndex]);

    digitalWrite(PPM\_PIN, LOW);

    currentIndex++;

  }

  // End of PPM frame

  if (currentTime - lastPPMTime >= PPM\_FRAME\_LENGTH) {

    delayMicroseconds(PPM\_PULSE\_LENGTH); // Delay for the remaining time until the frame completes

    lastPPMTime = micros();              // Reset lastPPMTime to start the next frame

  }

}