Arduino Based RC Plane

12.01.2024

Overview

Our team is trying to build an RC plane which is controlled by a sensor through Arduino microcontroller. The aerodynamic stability of the plane is handled by PID technique. However, due to limited resources, we were not able to test its stability. The majority of our work has been on creating the code for the plane and optimizing it for better performance.

Specifications

The devices used in the project are:

1. Arduino Microcontroller
2. 4 Servo motors
3. A BLDC motor
4. Controlling sensor (model is not finalized)

Two servo motors are connected to the wings as ailerons, one servo motor is connected to the rudder, and the last motor to the tail. Servo motors are controlled by sensors and further by the microcontroller.

Problems faced while framing the project:

1. Finding the right library for the code so that communication between the remote and the plane establishes.
2. We initially designed the model without PID technique, but due to poor stability of the plane, we had to establish PID technique.

Code

```
/*
Note ::
code of the PID functions are in development they're still yet to be tested
*/

#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Servo.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BNO055.h>
#include <utility/imumaths.h>

unsigned long lastInputAt = 0; // Stores the timestamp of the last input received
unsigned long timeNow = 0; // Stores the current timestamp
```

```
const byte ptr[6] = "00001"; // Address for the nRF24L01 module
const int ledPin = 13; // Pin for the onboard LED
RF24 channel(3, 2); // Create an instance of the RF24 class for communication

// PID constants
const double Kp = 0.5;
const double Ki = 0.2;
const double Kd = 0.1;

// PID variables
double error = 0;
double integral = 0;
double derivative = 0; // Add missing variable declaration
double output = 0; // Add missing variable declaration

Servo liftProvider; // Servo object for controlling lift
Servo leftAileron; // Servo object for controlling left aileron
Servo rightAileron; // Servo object for controlling right aileron
Servo directionChanger; // Servo object for controlling direction

Servo propellar; // Servo object for controlling propellar

int propelarRoatateValue, directionChangeValue, liftValue, angleShiftValue, aileronValue

// IMU variables
Adafruit_BNO055 BNOObject = Adafruit_BNO055(55);

struct FlightDetails
{
    byte directionChangeOffset; // Offset for adjusting direction change servo angle
    byte propellarRotationSpeed; // Speed of propellar rotation
    byte aileronServoAngle; // Angle of aileron servo
    byte liftServoAngle; // Angle of lift servo
    byte directionChangeStd; // Standard angle of direction change servo
    double directionError;
    double propellerError;
    double aileronError;
    double PIDError;
};

FlightDetails dataSet; // Structure for storing flight details

void setup()
{
    Serial.begin(9600); // Initialize serial communication
    channel.begin(); // Initialize the RF24 module
    channel.openReadingPipe(0, ptr); // Set the reading pipe address
    channel.setAutoAck(false); // Disable auto acknowledgment
    channel.setPALevel(RF24_PA_MAX); // Set power amplifier level
    channel.setDataRate(RF24_250KBPS); // Set data rate
    channel.startListening(); // Start listening for incoming data
    refactorData(); // Initialize the flight details
    directionChanger.attach(4); // Attach the direction change servo to pin 4
    liftProvider.attach(5); // Attach the lift servo to pin 5
    leftAileron.attach(6); // Attach the left aileron servo to pin 6
    RightAileron.attach(7); // Attach the right aileron servo to pin 7
    propellar.attach(10); // Attach the propellar servo to pin 10
```

```
    if (!BNOObject.begin())
    {
        while (1)
        {
            Serial.println("Failed to initialize BNO055! Check your wiring or I2C addres
            delay(1000);
        }
    }
}

void loop()
{
    timeNow = millis(); // Get the current timestamp

    if (timeNow - lastInputAt > 5000)
    {
        refactorData(); // If no input received for 1 second, reset the flight details
        ledGlow(); // Glow the LED
    }

    if (channel.available())
    {
        channel.read(&dataSet, sizeof(FlightDetails)); // If data available, read the fl
        lastInputAt = millis(); // Update the last input timestamp
    }

    // Read IMU data
    sensors_event_t event;
    BNOObject.getEvent(&event);

    // Get orientation data
    imu::Quaternion quat = BNOObject.getQuat();
    imu::Vector<3> euler = quat.toEuler();

    // Adjust flight parameters based on IMU data
    adjustFlightParameters(euler);
}

void ledGlow(){
    pinMode(ledPin, OUTPUT); // Set the LED pin as output
    digitalWrite(ledPin, HIGH); // Turn on the LED
    delay(500); // Wait for 1 second
    digitalWrite(ledPin, LOW); // Turn off the LED
    delay(500); // Wait for 1 second
}

void adjustFlightParameters(imu::Vector<3> euler)
{
    // Adjust aileron servo angle based on roll angle
    dataSet.aileronServoAngle = map(euler.x(), -180, 180, 0, 180);

    // Adjust lift servo angle based on pitch angle
    dataSet.liftServoAngle = map(euler.y(), -90, 90, 0, 180);

    // Adjust direction change offset based on yaw angle
    dataSet.directionChangeOffset = map(euler.z(), -180, 180, 0, 180);
}
```

```
void calculatePID()
{
    error = dataSet.liftServoAngle - 120; // Calculate the error as the difference betwe
    integral += error;
    derivative = error - dataSet.PIDError;
    output = Kp * error + Ki * integral + Kd * derivative;

    // Update the lift servo angle based on the PID output
    liftValue = constrain(120 + output, 30, 100);
    liftProvider.write(liftValue);

    dataSet.PIDError = error;
}

void calculateAileronPID()
{
    // Read desired aileron angle from the flight controller
    byte desiredAileronAngle = dataSet.aileronServoAngle;

    // Calculate the error between desired and current aileron angle
    double aileronError = desiredAileronAngle - aileronValue;

    // Calculate PID output
    double pidOutput = Kp * aileronError + Ki * integral + Kd * (aileronError - dataSet.

    // Apply constraint on the integral term
    if (integral > 100) {
        integral = 100;
    } else if (integral < -100) {
        integral = -100;
    }

    // Update the aileron servo position based on the PID output
    leftAileron.write(aileronValue + pidOutput);
    rightAileron.write(aileronValue + pidOutput);

    // Update previous error and integral for next iteration
    dataSet.aileronError = aileronError;
    integral += aileronError;
}

void calculatePropellerPID()
{
    // Read desired propeller rotation speed from the flight controller
    byte desiredPropellerSpeed = dataSet.propellarRotationSpeed;

    // Calculate the error between desired and current propeller rotation speed
    double propellerError = desiredPropellerSpeed - propelarRoatateValue;

    // Calculate the integral term
    integral += propellerError;

    // Calculate the derivative term
    derivative = propellerError - dataSet.propellerError;

    // Calculate the output using PID formula
    output = propellerError * Kp + integral * Ki + derivative * Kd;
```

```
    // Update the propeller servo position based on the PID output
    propellar.write(output);

    // Store the current error for the next iteration
    dataSet.propellerError = propellerError;
}

void calculateDirectionPID()
{
    // Read desired direction angle from the flight controller
    byte desiredDirectionAngle = dataSet.directionChangeOffset;

    // Calculate the error between desired and current direction angle
    double directionError = desiredDirectionAngle - dataSet.directionChangeStd;

    // Calculate the integral term
    dataSet.directionIntegral += (directionError * Ki);

    // Calculate the derivative term
    double directionDerivative = (directionError - dataSet.directionLastError) * Kd;

    // Calculate the PID output
    double directionOutput = (directionError * Kp) +  directionDerivative;

    // Update the direction servo position based on the PID output
    int directionPosition = dataSet.directionChangeStd + directionOutput;
    directionChanger.write(directionPosition);

    // Update the last error for the next iteration
    dataSet.directionError = directionError;
}

void refactorData()
{
    dataSet.directionChangeOffset = 120; // Set the direction change offset to default
    dataSet.propellarRotationSpeed = 80; // Set the propellar rotation speed to default
    dataSet.aileronServoAngle = 120; // Set the aileron servo angle to default
    dataSet.liftServoAngle = 120; // Set the lift servo angle to default
    dataSet.directionChangeStd = 1; // Set the direction change standard angle
}
```