ESP32 Drone with web-based remote

Introduction

The goal of this project is to have a stable, flying drone controlled via any web browser interface in the same local network as the drone. I started this project last summer as a learning exercise but it proved more difficult than anticipated and I couldn't manage to finish the drone up until now. The project is mostly DIY as the batteries, the body and the software being designed by me.

Descriere generală

The drone's flight controller functions are executed by a ESP32 C6 super-mini paired with an MPU6500 Inertial Measuring Unit, which provides the raw data necessary for stabilization. To determine the drone’s precise orientation, a complementary filter is used and it blends high-frequency gyroscope data with stable accelerometer readings to achieve a balance between short-term accuracy and long-term stability. This orientation data feeds into a continuous control loop where a Proportional-Integral-Derivative (PID) controller calculates the required throttle adjustments based on the error between the desired and actual state, with these commands then being transmitted to the motors as Pulse Width Modulation (PWM) signals. Instead of a traditional radio transmitter, the drone is managed via a dedicated web interface stored in the microcontroller's flash memory and accessed through a local area network, providing a streamlined, browser-based control platform for the operator. The physical frame is made of 4 3d printed arms attached to a 3mm polycarbonate sheet.

Hardware Design

Component	Quantity
ESP32-C6-Super-Mini	1
MPU6500 (IMU)	1
A2208 1400KV 14T BLDC Motor	4
30A ESC	4
Mateksys Mini Power Hub	1
3S 12.6V 20A Li-ion BMS	1
Samsung 35E 18650 3500mAh 8A	12

ESP32-c6-super-mini: This is the microcontroller and it will receive the input from the remote interface and use it for the calculations that will result in the output needed by the motors. Almost all the pins on the microcontroller can do the same things (PWM, SPI), so I chose them in the most convenient

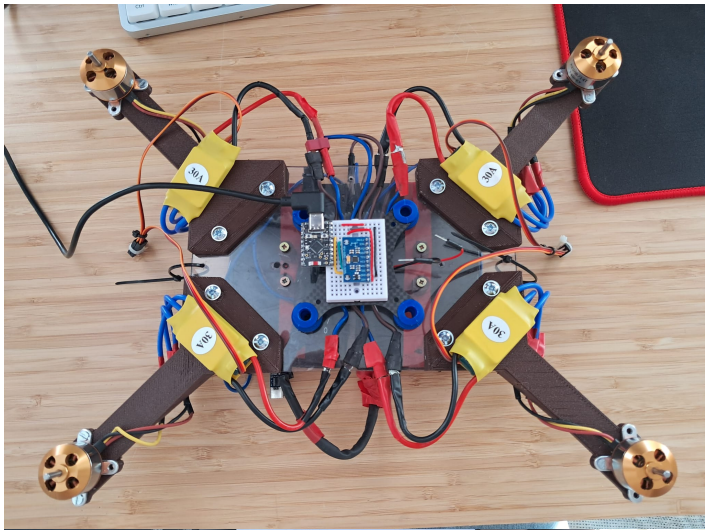
way possible for the circuit layout on the breadboard. I chose pins 4, 5, 6, 7 for SPI, leaving them to be defined in the software as SCLK, MOSI, MISO, and CS. The pins chosen for the motors must be capable of PWM, and since all pins on the microcontroller are capable of producing a PWM signal, I chose pins 20, 19, 18, 15 for the 4 motors.

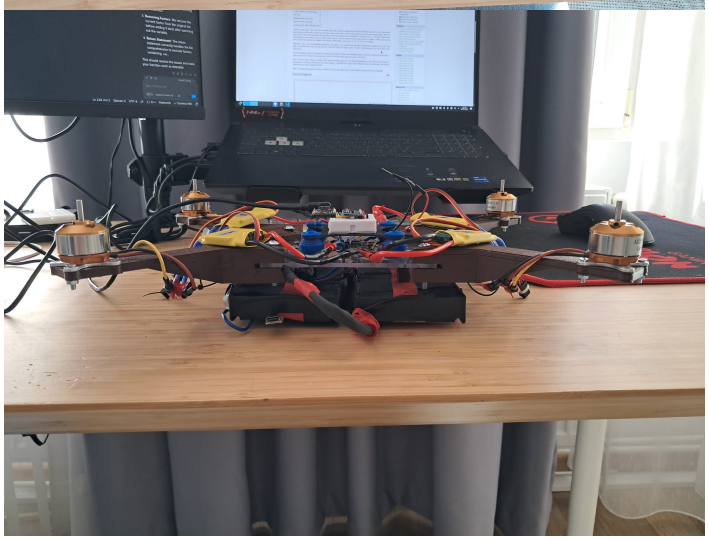
MPU6500: This is the most important component of the drone, as it provides the information regarding the attitude of the UAV. This IMU is compatible with both I2C and SPI, but I chose SPI due to its higher speed. For SPI, the following pins are used: SCLK, SDI, SDO, NCS.

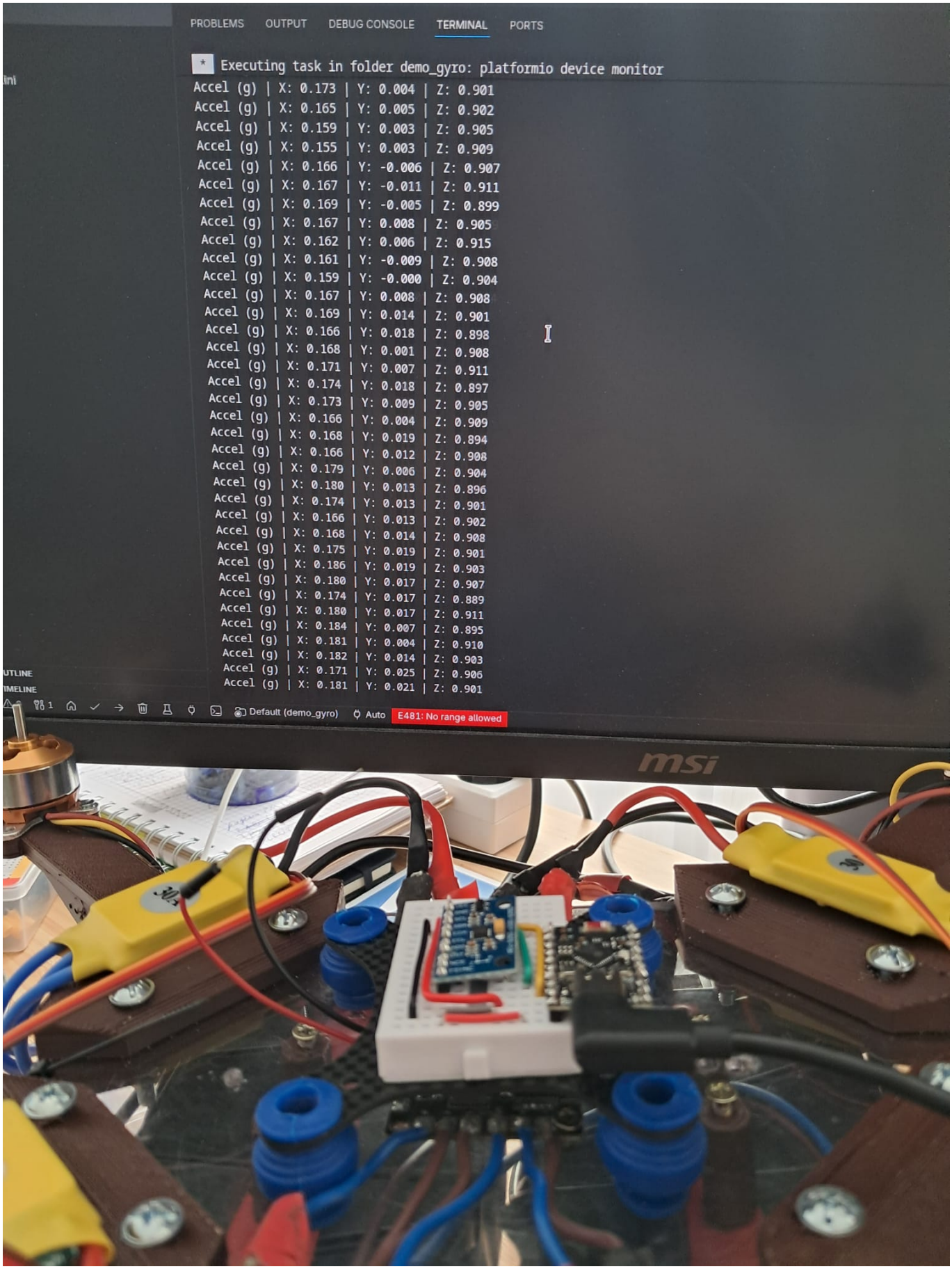
Power Distribution Board: This module is connected to the drone's battery and distributes power to the ESCs and the microcontroller. I connected the 5V terminals to the power pins of the ESP and the motor power terminals to the ESCs.

ESC: They are used to keep a constant motor speed by taking power from the PDB and providing it to the motors and the power is proportional to the input received from the microcontroller. I connected their signal to the PWM pins of the microcontroller, and they are powered by the power distribution board.

BMS: The battery management system is needed to have a safe battery pack so they charge uniformly and don't overdeplete.






PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
Executing task in folder demo_gyro: platformio device monitor
Accel (g) | X: 0.173 | Y: 0.004 | Z: 0.901
Accel (g) | X: 0.165 | Y: 0.005 | Z: 0.902
Accel (g) | X: 0.159 | Y: 0.003 | Z: 0.905
Accel (g) | X: 0.155 | Y: 0.003 | Z: 0.909
Accel (g) | X: 0.166 | Y: -0.006 | Z: 0.907
Accel (g) | X: 0.167 | Y: -0.011 | Z: 0.911
Accel (g) | X: 0.169 | Y: -0.005 | Z: 0.899
Accel (g) | X: 0.167 | Y: 0.008 | Z: 0.905
Accel (g) | X: 0.162 | Y: 0.006 | Z: 0.915
Accel (g) | X: 0.161 | Y: -0.009 | Z: 0.908
Accel (g) | X: 0.159 | Y: -0.000 | Z: 0.904
Accel (g) | X: 0.167 | Y: 0.008 | Z: 0.908
Accel (g) | X: 0.169 | Y: 0.014 | Z: 0.901
Accel (g) | X: 0.166 | Y: 0.018 | Z: 0.898
Accel (g) | X: 0.168 | Y: 0.001 | Z: 0.908
Accel (g) | X: 0.171 | Y: 0.007 | Z: 0.911
Accel (g) | X: 0.174 | Y: 0.018 | Z: 0.897
Accel (g) | X: 0.173 | Y: 0.009 | Z: 0.905
Accel (g) | X: 0.166 | Y: 0.004 | Z: 0.909
Accel (g) | X: 0.168 | Y: 0.019 | Z: 0.894
Accel (g) | X: 0.166 | Y: 0.012 | Z: 0.908
Accel (g) | X: 0.179 | Y: 0.006 | Z: 0.904
Accel (g) | X: 0.180 | Y: 0.013 | Z: 0.896
Accel (g) | X: 0.174 | Y: 0.013 | Z: 0.901
Accel (g) | X: 0.166 | Y: 0.013 | Z: 0.902
Accel (g) | X: 0.168 | Y: 0.014 | Z: 0.908
Accel (g) | X: 0.175 | Y: 0.019 | Z: 0.901
Accel (g) | X: 0.186 | Y: 0.019 | Z: 0.903
Accel (g) | X: 0.180 | Y: 0.017 | Z: 0.907
Accel (g) | X: 0.174 | Y: 0.017 | Z: 0.889
Accel (g) | X: 0.180 | Y: 0.017 | Z: 0.911
Accel (g) | X: 0.184 | Y: 0.007 | Z: 0.895
Accel (g) | X: 0.181 | Y: 0.004 | Z: 0.910
Accel (g) | X: 0.182 | Y: 0.014 | Z: 0.903
Accel (g) | X: 0.171 | Y: 0.025 | Z: 0.906
Accel (g) | X: 0.181 | Y: 0.021 | Z: 0.901
Default (demo_gyro)
Auto
E481: No range allowed
msi

Circuit Diagram

Software Design

Up until now I've implemented a driver for the SPI data transfer of the accelerometer and gyroscope data from the MPU6500. That data would be further processed by multiplying it by the scalar for the chosen resolution. The scaled data would further be polled at a bigger frequency than the control loop, counting how many readings would happen between each loop and then dividing the summed data by that number, achieving an oversampling effect to remove noise from the data. A lowpass filter would be applied over both accelerometer and gyroscope data to remove the low frequency noise create by mechanical movements. What remains is clean data that is fused into a single attitude measurement with a complementary filter. At startup, the sensor goes through a calibration process in which the average of multiple stationary readings is taken and saved as a stationary error which is later subtracted from each reading.

UAVs should handle unpredictable disruptions, the most unpredictable of which can be the pilot so I wrote a function to interpolate the current throttle to the desired throttle to soften quick changes made by the pilot. Controlling the drone is easy; the remote is made of a direction knob which tilts the drone in the desired direction and a throttle knob which tells the motors how fast they should spin. The remote is a web interface served by the microcontroller if the browser is connected to the microcontroller's local network. Websockets transmit the data from the remote to the microcontroller.

The attitude measurement is used to compute the input for the PID controller. The input for the controller is the error given by the difference in the attitude of the UAV and the desired attitude which is set by the pilot. The flow of the program is based on a 30ms period flow of inputs from the remote, a 250Hz control loop, and the program loop in which they are processed and used. As a safety measure, a neutral attitude and a low throttle is set if there have been more than 35ms from the last input, being activated by a watchdog timer. Absence of input data isn't the only feature using a timer; the control loop, being as important as it is, is entered via a flag set by an interrupt triggered when 4000 microseconds pass.

Motor mixing is done after all the PID calculations are done, resulting in the motor output which is based on the frequency at which the control loop operates. The motor output is delivered via PWM with the ESP32 LED Control functionality.

The libraries I used for this project are the following:
-Arduino.h - I used it for generic functions and to keep the code simple as I wanted to avoid the RTOS of the ESP32
-SPI.h - the sensor readings are done using this library, I used it for simplicity and ease of use
-WiFi.h - used for the WiFi stack necessary for the web remote
-ESPAsyncWebServer.h - used to host the web remote interface
-ESPmDNS.h - used to define a text address for the web remote so I wouldn't have to type the ip of the ESP32 each time

The structure of the project:

-main.cpp - web server functionalities, control loop
-PID.cpp - separate PID class and functionalities
–PID.h
-MPU6500.cpp - separate sensor SPI readings from the main code
–MPU6500.h

Rezultate Obţinute

Video demonstrativ: https://www.youtube.com/watch?v=J08_xca9AOc

Concluzii

Download

O arhivă (sau mai multe dacă este cazul) cu fişierele obţinute în urma realizării proiectului: surse, scheme, etc. Un fişier README, un ChangeLog, un script de compilare şi copiere automată pe uC crează întotdeauna o impresie bună .

Fişierele se încarcă pe wiki folosind facilitatea **Add Images or other files**. Namespace-ul în care se încarcă fişierele este de tipul **:pm:prj20??:c?** sau **:pm:prj20??:c?:nume_student** (dacă este cazul). **Exemplu:** Dumitru Alin, 331CC → **:pm:prj2009:cc:dumitru_alin**.

Jurnal

Puteți avea și o secțiune de jurnal în care să poată urmări asistentul de proiect progresul proiectului.

Bibliografie/Resurse

Listă cu documente, datasheet-uri, resurse Internet folosite, eventual grupate pe **Resurse Software** şi **Resurse Hardware**.

Export to PDF