KNOWLEDGE FOR SERVICE

University
Of
Benin

Project
Proposal
Defense

PRESENTED BY OBOH EDWARD OSARETIN
ENG1503587
DEPARTMENT OF COMPUTER ENGINEERING

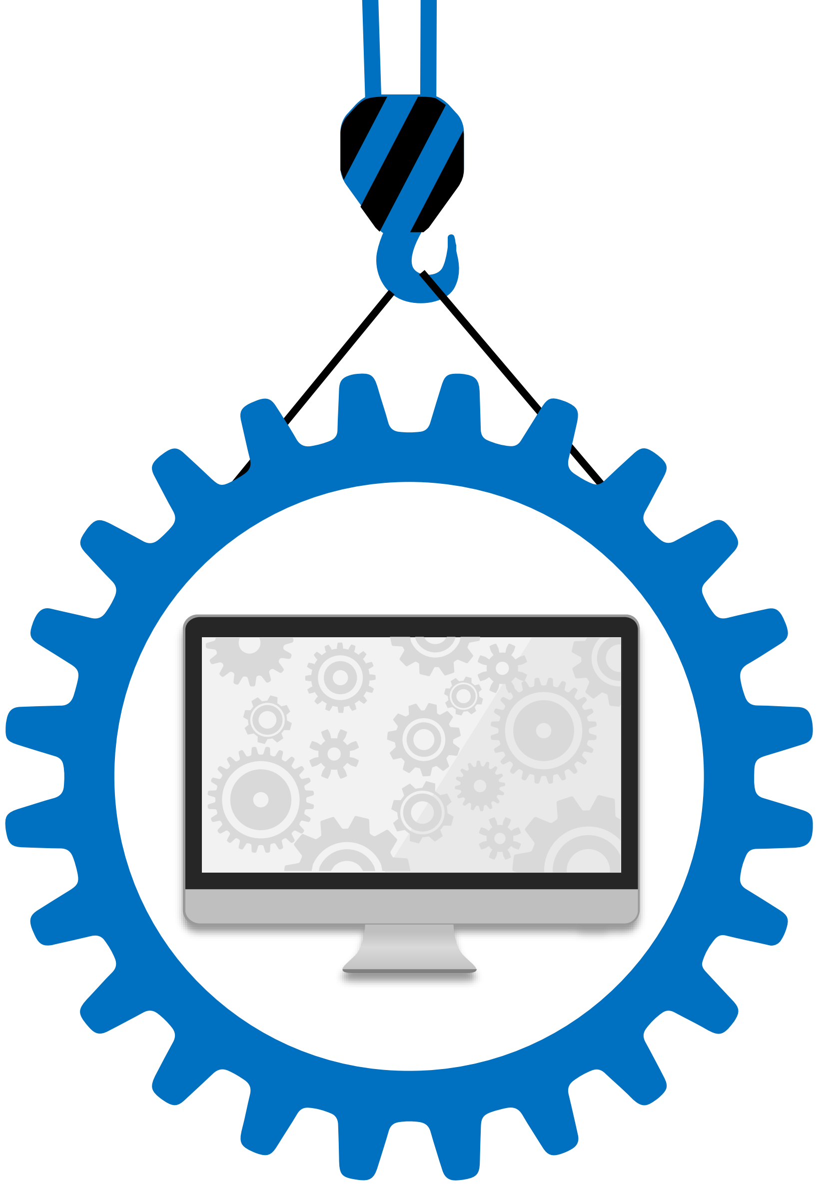
PROJECT ON
CONTROL DESIGN AND HARDWARE IMPLEMENTATION OF A MULTI-ROTOR SYSTEM
(JULY 2021)

BY

EDWARD OSARETIN OBOH
ENG1503587

UNDER

ENGR. J.A. IGIMOH,
DEPARTMENT OF COMPUTER ENGINEERING,
FACULTY OF ENGINEERING,
UNIVERSITY OF BENIN.



Presentation Outline

01 Introduction

02 Objective

03 Scope

04 Methodology

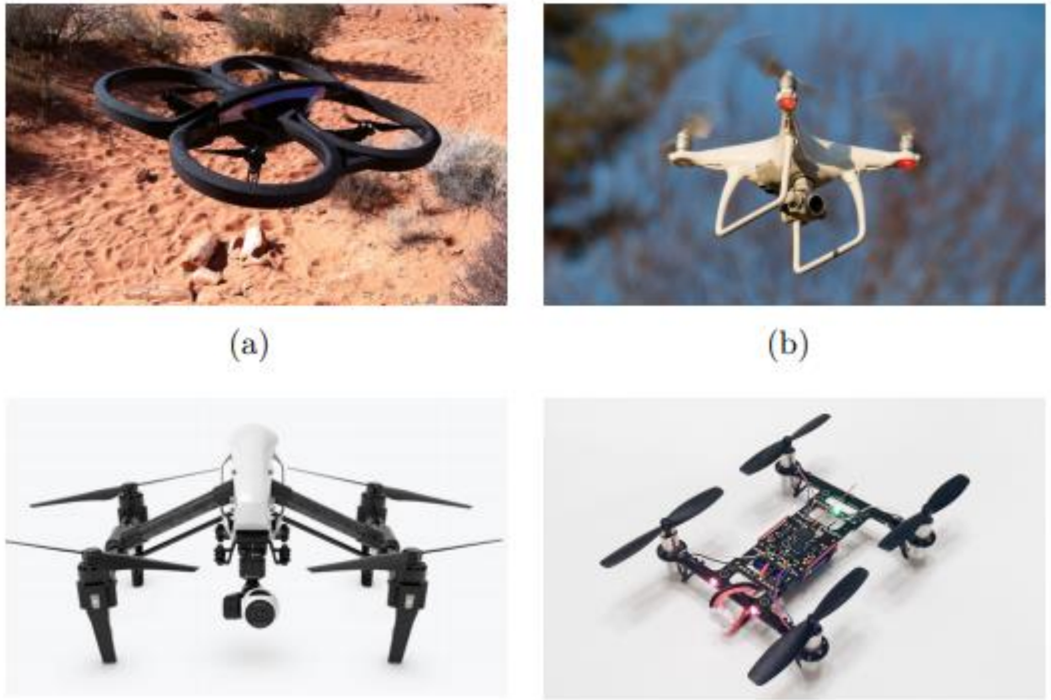
05 Result

06 Conclusion

07 Future Work

Introduction

About Multi-Rotor Systems (Quadcopters)



(a) (b) (c) (d)

Unmanned aerial vehicles (UAVs) are being increasingly used today than ever before in both military and civil applications. The rapid advancement in miniature sensors, actuators and processors has led to the development of powerful autopilot systems which play a major role in UAVs control by making the flight safer and more efficient. These gave a rise to small sized, interestingly featured commercial UAVs, one of which is the quadcopter.

The history of Quadcopters starts in the beginning of the 20th century. The first ever quadcopter built was "Gyroplane n: 01" in 1907. This quadcopter had many limitations. Its stabilization was achieved by control of people on the ground. During the 1920's, other quadcopters with much improved performance were built by engineers who targeted the vertical flight.

Objective

Problem Statement

When building a multi-rotor vehicle (drone), designers are faced with the choice of paying for a custom designed aerial vehicle, building their own vehicle from scratch, or sacrificing controllability for an inexpensive off-the-shelf system. While numerous inexpensive off-the-shelf multi-rotor platforms are available, they often consist of proprietary modules even when advertised as open-source. Common examples of these black-box modules are sensor-less brushless motor controllers, flight controllers, and radios. These modules are often proprietary and have limited hardware specifications (Clean Flight, “Clean Flight”, 2016), (Open Pilot, “Open Pilot”, 2016). Consequently, for a researcher, the control and the modifiability of these modules is limited.

Objective

About the Project

A main objective of this work is to provide researchers with a functional, fully specified, and stabilized quadcopter. This system will be specified from scratch hardware and software with the intent of eliminating as many black box components as possible. In addition, this flight system will have an emphasis on theoretical control as well as IMU data collection making it a prime candidate for future research.

Scope

About the Project

A

From the dynamic equations of the quadcopter system, a Proportional, Integral and Derivative (PID) based control system was designed to achieve stability of the system. The designed controller was able to control and stabilize the attitude (Roll, Pitch and Yaw). The designed controller was then be implemented on the hardware.

B

Other control strategies are not explored in this project. Altitude control and autonomous navigation are not part of the project, altitude and position of the vehicle in an inertial frame are controlled by the pilot commands.

C

Components manufacturing process, materials and detailed working are not concerned with the purpose of the project

Methodology

- Control Objective
- Hardware Components
- IMU Sensor Implementation
- Control System Implementation
- Software Architecture
- Hardware Implementation

Methodology

❖ Control Objective

A primary control objective for the quadcopter is to be able to control orientation. With this in mind, the first set of control goals can be summarized in the equation below where θ_c, φ_c, φ_c represent a user commanded rotation, where $\dot{\theta}$, $\dot{\varphi}$, $\dot{\psi}$ represent the rotational velocities of the quadcopter, and where T_c represents the user commanded throttle.

$$\dot{\theta} = 0 \quad \dot{\varphi} = 0 \quad \dot{\psi} = 0$$

$$\theta \rightarrow \theta_c \quad \varphi \rightarrow \varphi_c \quad \psi \rightarrow \psi_c, \quad T \rightarrow T_c$$

These control parameters, θ_c, φ_c, ψ_c, T_c, allow a user to maneuver the quadcopter to anywhere in three dimensional space. Consequently, many commercial systems give users these four degrees of freedom to operate a quadcopter. However, the user must also act as a control system in order to regulate the quadcopter's height to keep it above the ground. To achieve this, the user must observe the quadcopter's height and constantly adjust the throttle, T_c, such that the height with respect to ground, z, is roughly constant.

Methodology

Components

Frame	DJI F450 Quadcopter Frame
Propellers/Rotors	10x4.5 Propellers (4)
Motors	A2212 2200KV BLDC motors
ESCs	Hobby King 30A Brushless ESC
Battery	11.1V (3S) Li-po Battery
Transmitter	NRF24L10 PA (power amplifier) LNA (low noise amplifier)
Receiver	NRF24L10
Inertial Measurement Unit (IMU)	MPU 6050 6dof IMU
Arduino Boards	Arduino UNO (1), Arduino Nano (2)

Methodology

❖ IMU Implementation

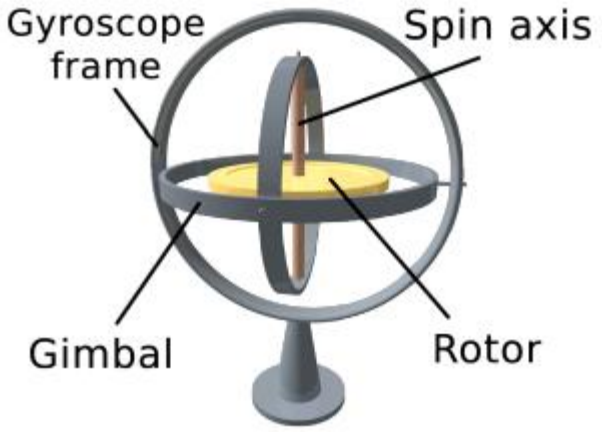
In order to meet the control goals, estimation of the quadcopter's orientation is required. With modern advancements in electronics, determining orientation can be done inexpensively, efficiently, and quickly with micro electrical mechanical system (MEMS) based sensors.

Unfortunately, currently, there is no single affordable MEMS sensor that directly measures θ, ϕ, ψ. Consequently, the combination of multiple MEMS sensors is required in order to accurately estimate orientation.

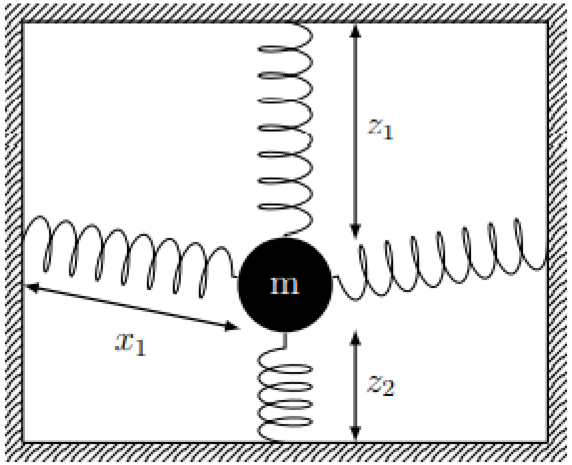
Methodology

❖ IMU Implementation

- Gyroscope



- Accelerometer



Arduino_drone_2.0_TWO_axis §

```
//////////////////////////////////////Gyro read/////////////////////////////////////
 Wire.beginTransmission(0x68);            //begin, Send the slave adress (in this case 68)
 Wire.write(0x43);                        //First adress of the Gyro data
 Wire.endTransmission(false);
 Wire.requestFrom(0x68,4,true);           //We ask for just 4 registers
 Gyr_rawX=Wire.read()<<8|Wire.read();     //Once again we shif and sum
 Gyr_rawY=Wire.read()<<8|Wire.read();
 /*Now in order to obtain the gyro data in degrees/seconds we have to divide first
 the raw value by 32.8 because that's the value that the datasheet gives us for a 1000dps range*/
 /*---X---*/
 Gyr_rawX = (Gyr_rawX/32.8) - Gyro_raw_error_x;
 /*---Y---*/
 Gyr_rawY = (Gyr_rawY/32.8) - Gyro_raw_error_y;
 /*Now we integrate the raw value in degrees per seconds in order to obtain the angle
 * If you multiply degrees/seconds by seconds you obtain degrees */
   /*---X---*/
 Gyro_angle_x = Gyr_rawX*elapsedTime;
 /*---X---*/
 Gyro_angle_y = Gyr_rawY*elapsedTime;
```

Gyroscope

Arduino_drone_2.0_TWO_axis §

```
  /////////////////////////////////////////Acc read/////////////////////////////////////////
  Wire.beginTransmission(0x68);     //begin, Send the slave adress (in this case 68)
  Wire.write(0x3B);                 //Ask for the 0x3B register- correspond to AcX
  Wire.endTransmission(false);      //keep the transmission and next
  Wire.requestFrom(0x68,6,true);    //We ask for next 6 registers starting withj the 3B
  /*We have asked for the 0x3B register. The IMU will send a brust of register.
  * The amount of register to read is specify in the requestFrom function.
  * In this case we request 6 registers. Each value of acceleration is made out of
  * two 8bits registers, low values and high values. For that we request the 6 of them
  * and just make then sum of each pair. For that we shift to the left the high values
  * register (<<) and make an or (|) operation to add the low values.
  If we read the datasheet, for a range of+-8g, we have to divide the raw values by 4096*/
  Acc_rawX=(Wire.read()<<8|Wire.read())/4096.0 ; //each value needs two registres
  Acc_rawY=(Wire.read()<<8|Wire.read())/4096.0 ;
  Acc_rawZ=(Wire.read()<<8|Wire.read())/4096.0 ;
 /*Now in order to obtain the Acc angles we use euler formula with acceleration values
 after that we substract the error value found before*/
 /*---X---*/
 Acc_angle_x = (atan((Acc_rawY)/sqrt(pow((Acc_rawX),2) + pow((Acc_rawZ),2)))*rad_to_deg) - Acc_angle_error_x;
 /*---Y---*/
 Acc_angle_y = (atan(-1*(Acc_rawX)/sqrt(pow((Acc_rawY),2) + pow((Acc_rawZ),2)))*rad_to_deg) - Acc_angle_error_y;
```

Accelerometer

Methodology

❖ IMU Implementation

- Sensor Fusion

Gyroscopes have good dynamic response and noise immunity, they have long term drift. In contrast, accelerometers have poor dynamic response but are not susceptible to drift in the same manner. Consequently, a high pass filter is used on the gyroscope data and a low pass filter is used on the accelerometer data.

$$\theta = atan\left(\frac{A_x}{\sqrt{A_y^2 + A_z^2}}\right) \cdot \frac{180}{\pi}$$

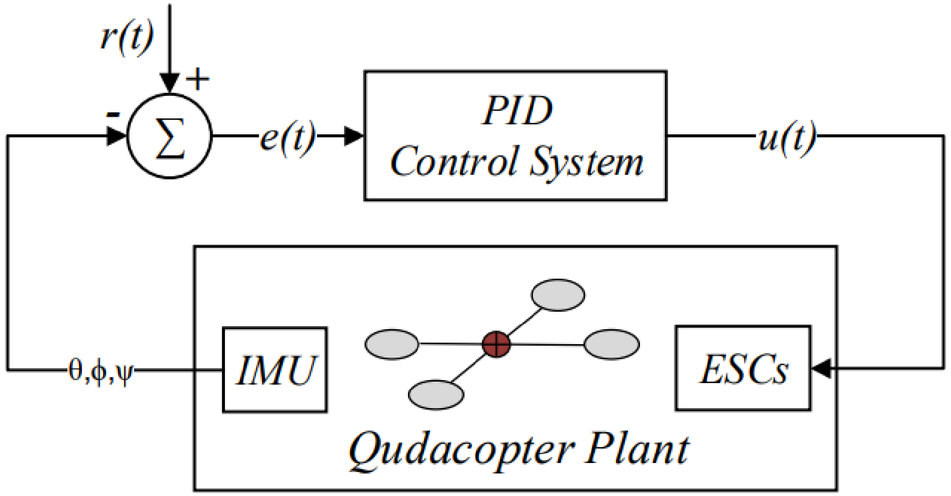
$$\phi = atan2\left(\frac{A_y}{A_z}\right) \cdot \frac{180}{\pi}$$

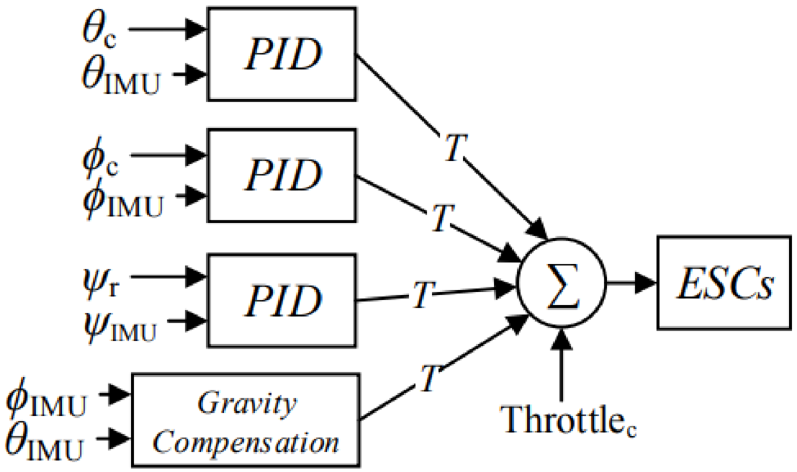
Methodology

❖ Control System Implementation

This control system was needed in order to handle real world disturbances and to account for unknown offsets. In order to drive the quadcopter's orientation (θ, ϕ, ψ) to desired values, a series of proportional integral derivative (PID) controllers were implemented.

$$u(t) = Kp \cdot e(t) + Ki \cdot Z\ t\ 0\ e(\tau)d\tau + Kd\ de(t)\ dt$$





Arduino_drone_2.0_TWO_axis §

```
/*///////////////////////////P I D///////////////////////////////////*/
roll_desired_angle = map(input_ROLL,1000,2000,-10,10);
pitch_desired_angle = map(input_PITCH,1000,2000,-10,10);

/*First calculate the error between the desired angle and *the real measured angle*/
roll_error = Total_angle_y - roll_desired_angle;
pitch_error = Total_angle_x - pitch_desired_angle;
/*Next the proportional value of the PID is just a proportional constant *multiplied by the error*/
roll_pid_p = roll_kp*roll_error;
pitch_pid_p = pitch_kp*pitch_error;
/*The integral part should only act if we are close to the desired position but we want to fine
tune the error. That's why I've made a if operation for an error between -2 and 2 degree.
To integrate we just sum the previous integral value with the error multiplied by  the integral
constant. This will integrate (increase) the value each loop till we reach the 0 point*/
if(-3 < roll_error <3)
{
  roll_pid_i = roll_pid_i+(roll_ki*roll_error);
}
if(-3 < pitch_error <3)
{
  pitch_pid_i = pitch_pid_i+(pitch_ki*pitch_error);
}
```

```
/*The last part is the derivate. The derivate acts upon the speed of the error. As we know the speed is the amount
of error that produced in a certain amount of time divided by that time. For taht we will use a variable called
previous_error. We substract that value from the actual error and divide all by the elapsed time. Finnaly we multiply
the result by the derivate constant*/
roll_pid_d = roll_kd*((roll_error - roll_previous_error)/elapsedTime);
pitch_pid_d = pitch_kd*((pitch_error - pitch_previous_error)/elapsedTime);
/*The final PID values is the sum of each of this 3 parts*/
roll_PID = roll_pid_p + roll_pid_i + roll_pid_d;
pitch_PID = pitch_pid_p + pitch_pid_i + pitch_pid_d;
/*We know taht the min value of PWM signal is 1000us and the max is 2000. So that tells us that the PID value can/s
oscilate more than -1000 and 1000 because when we have a value of 2000us the maximum value taht we could substract
is 1000 and when we have a value of 1000us for the PWM signal, the maximum value that we could add is 1000 to reach
the maximum 2000us. But we don't want to act over the entire range so -+400 should be enough*/
if(roll_PID < -400){roll_PID=-400;}
if(roll_PID > 400) {roll_PID=400; }
if(pitch_PID < -4000){pitch_PID=-400;}
if(pitch_PID > 400) {pitch_PID=400;}

/*Finnaly we calculate the PWM width. We sum the desired throttle and the PID value*/
pwm_R_F  = 115 + input_THROTTLE - roll_PID - pitch_PID;
pwm_R_B  = 115 + input_THROTTLE - roll_PID + pitch_PID;
pwm_L_B  = 115 + input_THROTTLE + roll_PID + pitch_PID;
pwm_L_F  = 115 + input_THROTTLE + roll_PID - pitch_PID;
```

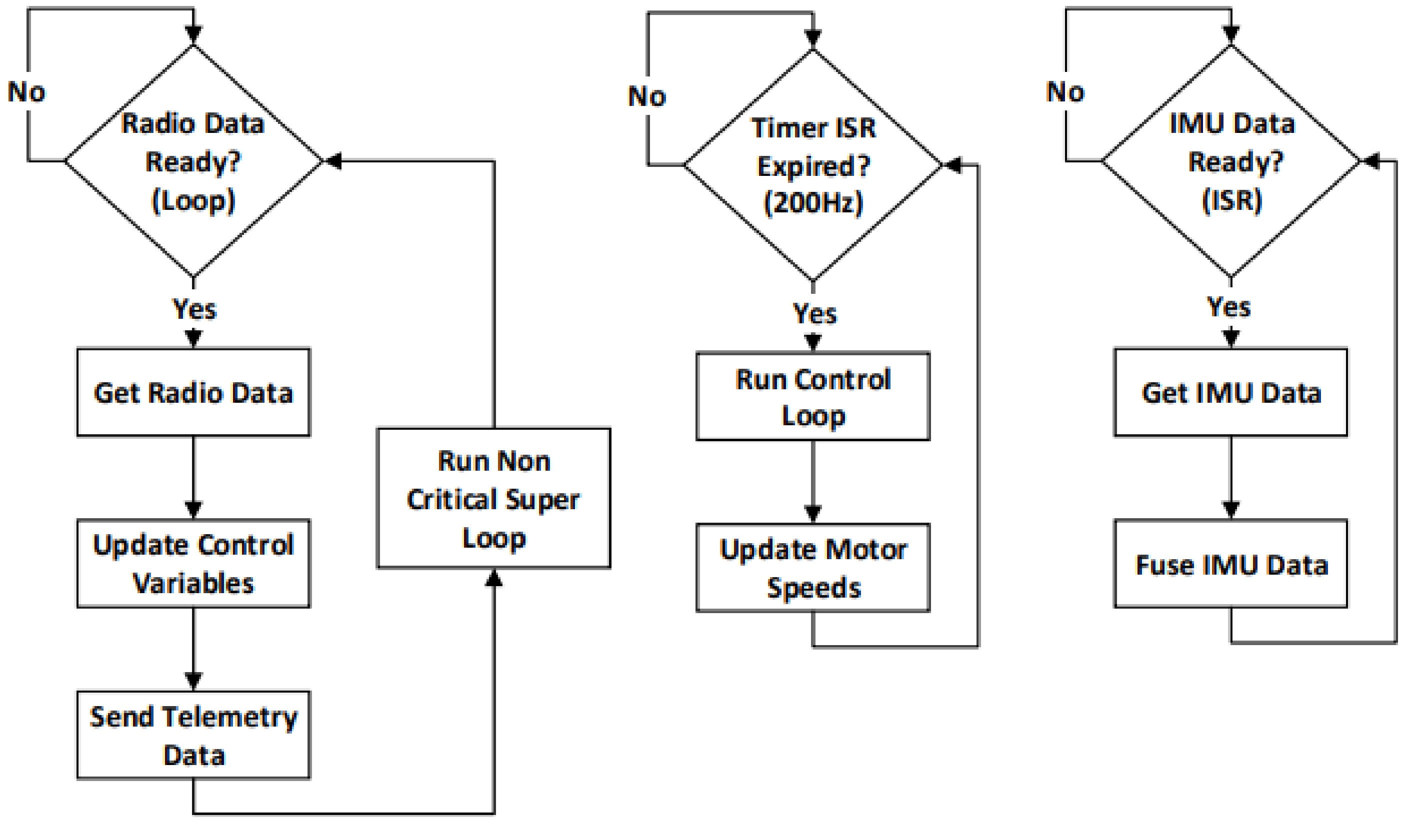
Methodology

❖ Software Architecture

The control loop along with the IMU code account for the majority of flight critical code running on the quadcopter. To keep control timing and IMU collection constant, the software was implemented in three separate threads. These threads primarily manage IMU filter updates, control system updates, and radio link updates.

Methodology

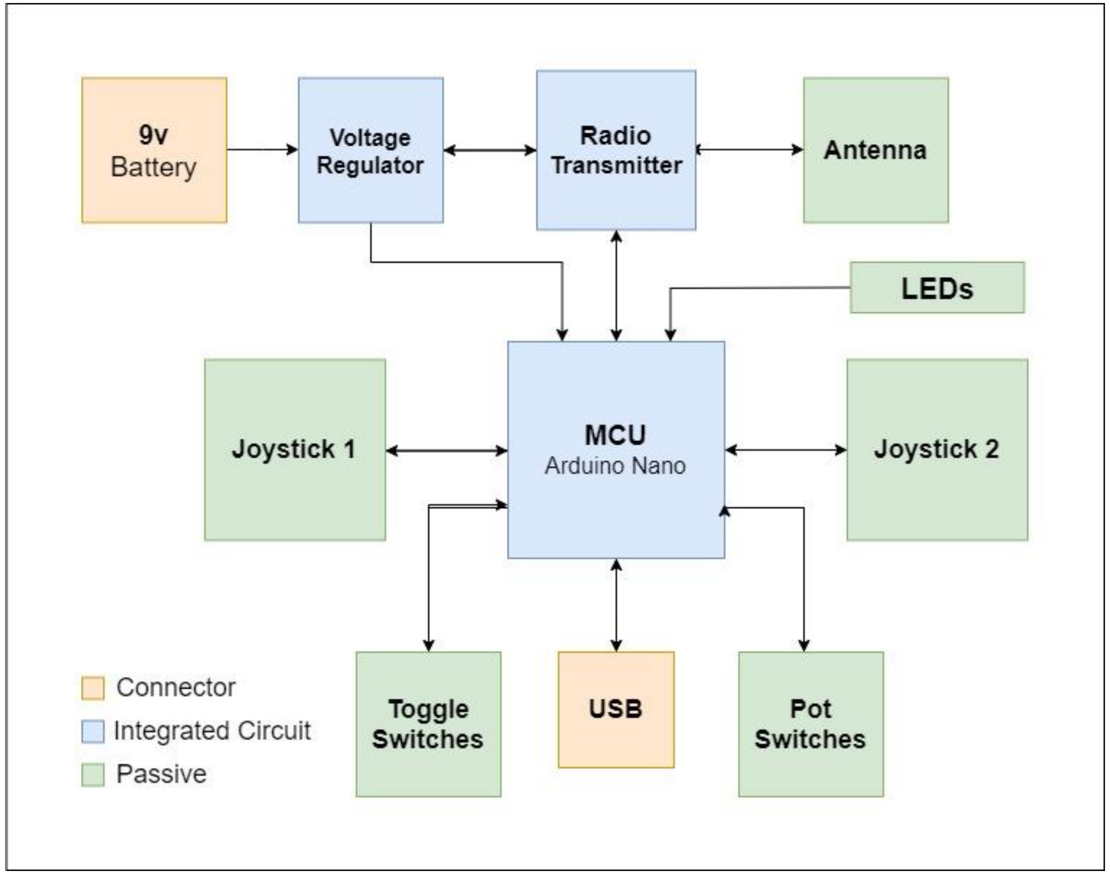
❖ Software Architecture - Flowchart

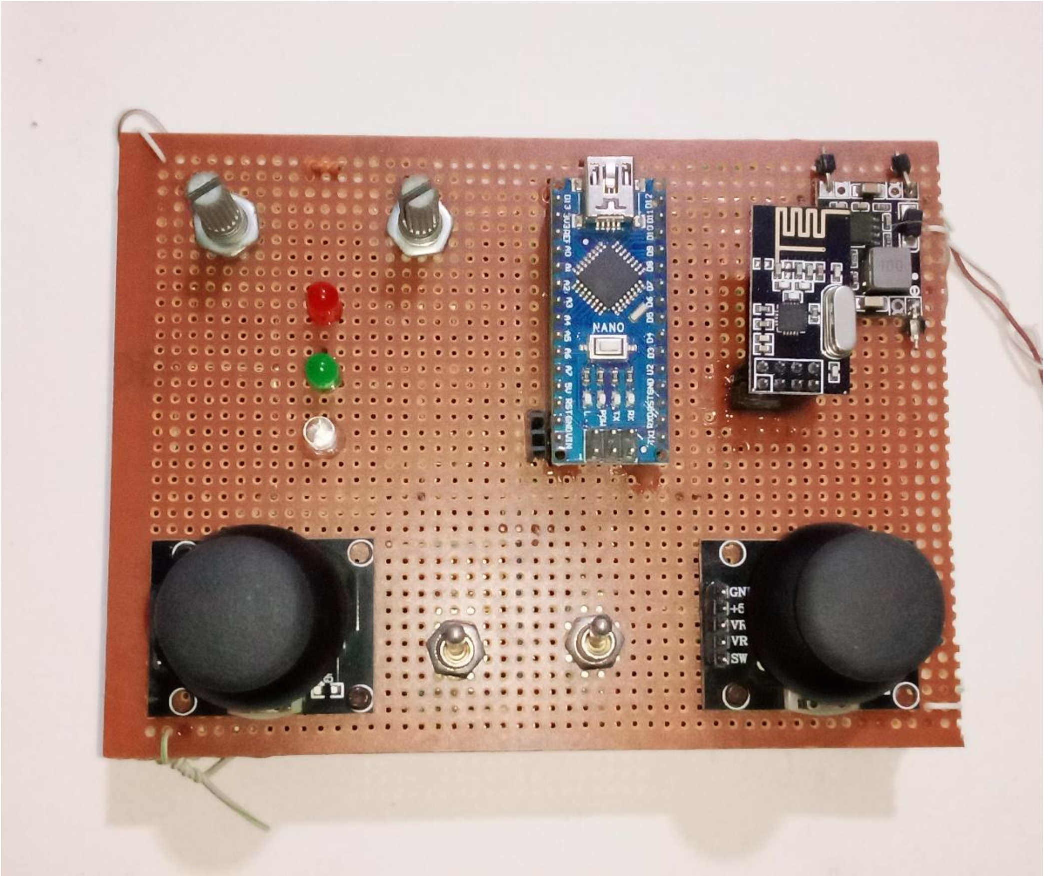


Methodology

❖ Hardware Implementation

Remote Controller

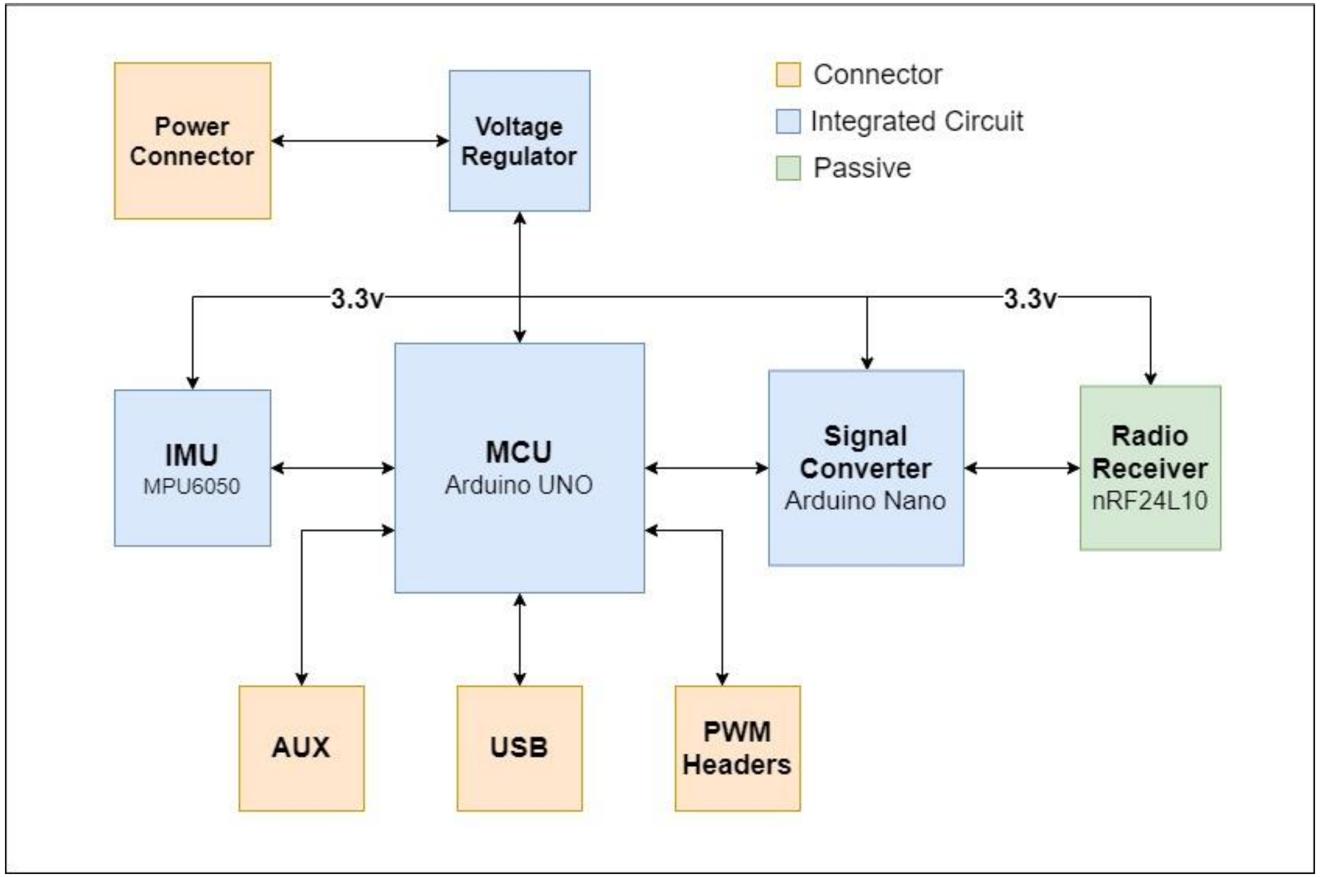


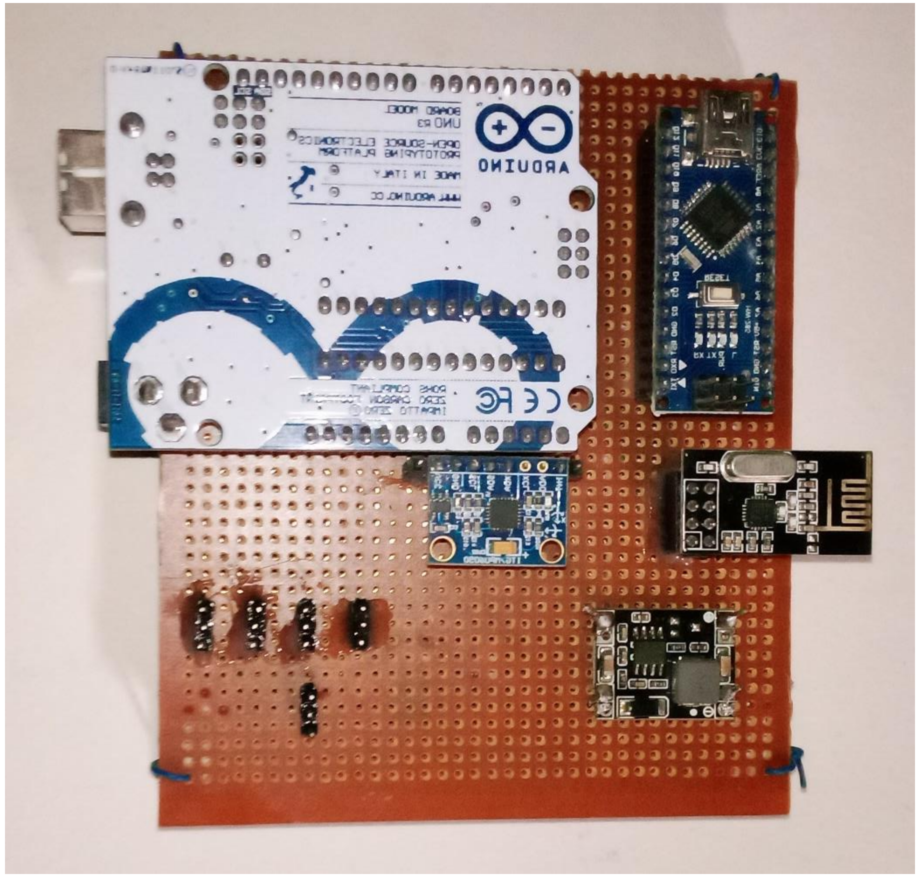


Methodology

❖ Hardware Implementation

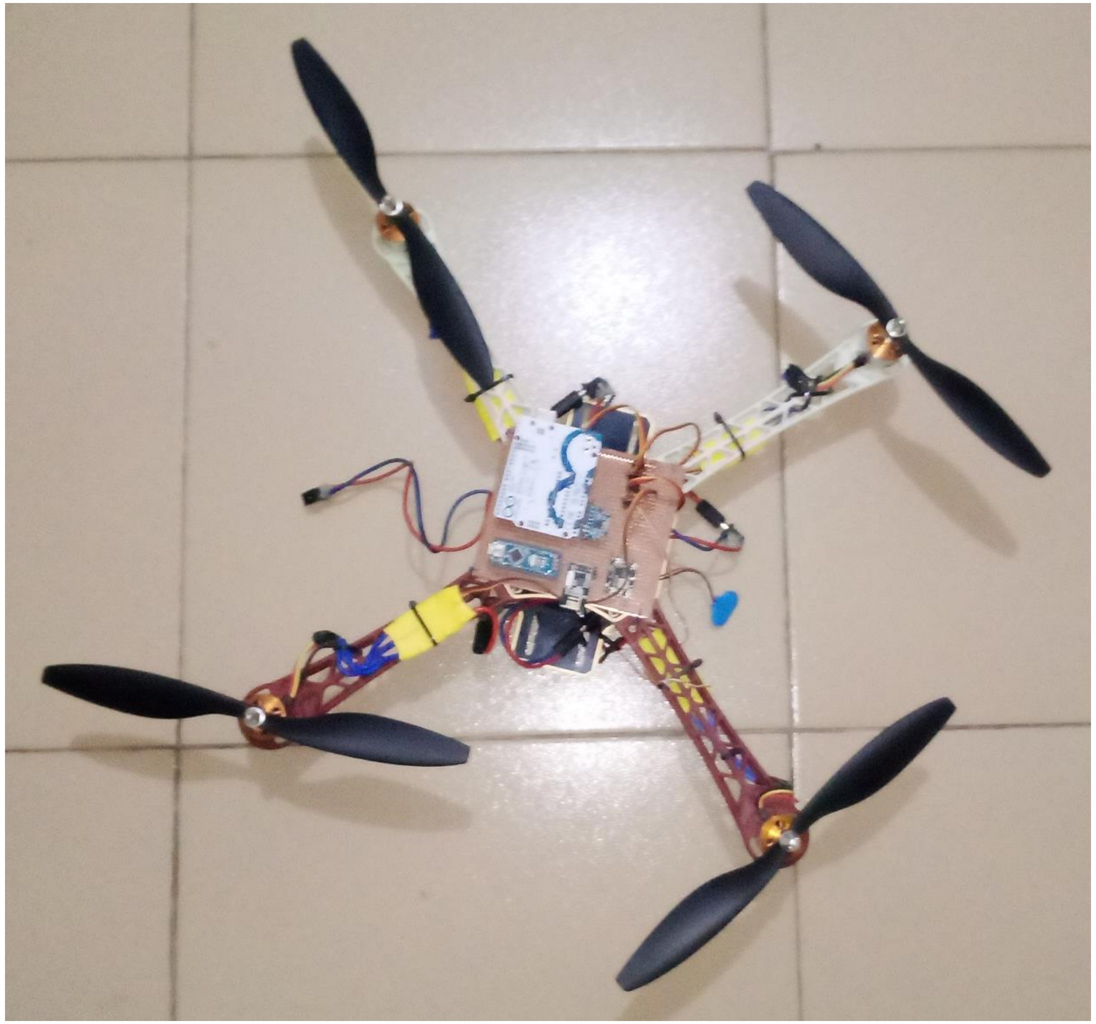
Flight Controller





Result

- Drone Build



Result

For safety and verification purposes, experimental setup is designed to examine the controller response before applying it to a real flight test. An experimental testbench was used. On the test bench, the quadcopter is held firmly from the top and the bottom allowing only rotational motion, i.e., three degrees of freedom. Various reference signals are sent via RC to explore the system reference tracking

At the end of testing, as expected, the multirotor system (Quadcopter in this case) was controlled by the electronic Remote controller using RF signal. The flight controller was able to control roll, pitch, yaw, altitude and motion in x or y direction. As expected, the multirotor system balanced itself using commands in the control software written to its microcontroller, when there were no changes to inputs sent from the remote controller.

Result

Bill of Engineering Measurement and Evaluation

Component	Specification	Number	Cost
Frame	DJI F450 Quadcopter Frame	1	8000
Propellers/Rotors	10x4.5 Propellers	4	4 x 1500
Motors	A2212 BLDC motors	4	4 x 3500
ESCs	Hobby King 30A Brushless ESC	4	4 x 3000
Battery	11.1 V Li-po Battery	1	8000
Transmitter	NRF24L10 PA(power amplifier) LNA(low noise amplifier)	1	3500

Result

Receiver	**NRF24L10**	**1**	**3000**
Inertial Measurement Unit (IMU)	MPU 6050 6dof IMU	1	2400
Arduino UNO	Arduino UNO Artmega MCU	1	8000
Arduino Nano	Arduino Nano, Atmega MCU	2	4200
Vero Board	Dotted Copper Vero Board	2	2 x 400
Joystick	2 Axis Joystick Potentiometer	2	2 x 3500
Other Components for soldering, connections, testing and binding	Switches, LEDs, Potentiometers, header connectors, connecting wires, Glue, solder, Battery cap	-	3500
Shipping and Delivery fee	-	-	7000
	TOTAL		87,400

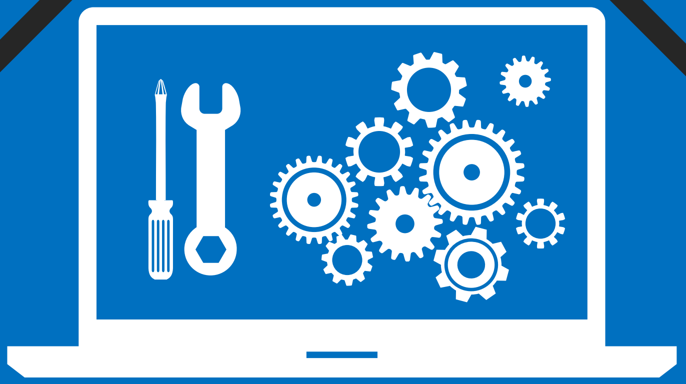
Conclusion

In this Project we,

- developed a control update laws as well as control objectives
- developed a 6 degrees of freedom fusion filter to provide accurate estimations of orientation
- demonstration a control system to achieve stable auto leveling flight using a series of PID controllers
- overviewed the RF hardware involved in creating the transcieving link between the remote and the quadcopter
- specified other hardware involved in the system such as the flight controller and electronic speed controllers

Future Work

- Currently, the IMU filter operates on Euler angles when ideally it should be based upon quaternions to avoid singularities.
- With regards to the control system, the first thing that would be improved is the PID control system performance. The current balancing action is somewhat noisy and the yaw control is lacking.
- In addition to PID control, other control techniques could be implemented such as linear quadratic regulator (LQR) as well as model predictive control (MPC).
- A final improvement would be the addition of an onboard GPS sensor and a Lidar sensor to facilitate autonomous navigation in both indoor and outdoor environments
- Addition of a camera to enable the system to be used for aerial surveillance or for gathering data.



Thank You

For Listening To This Presentation