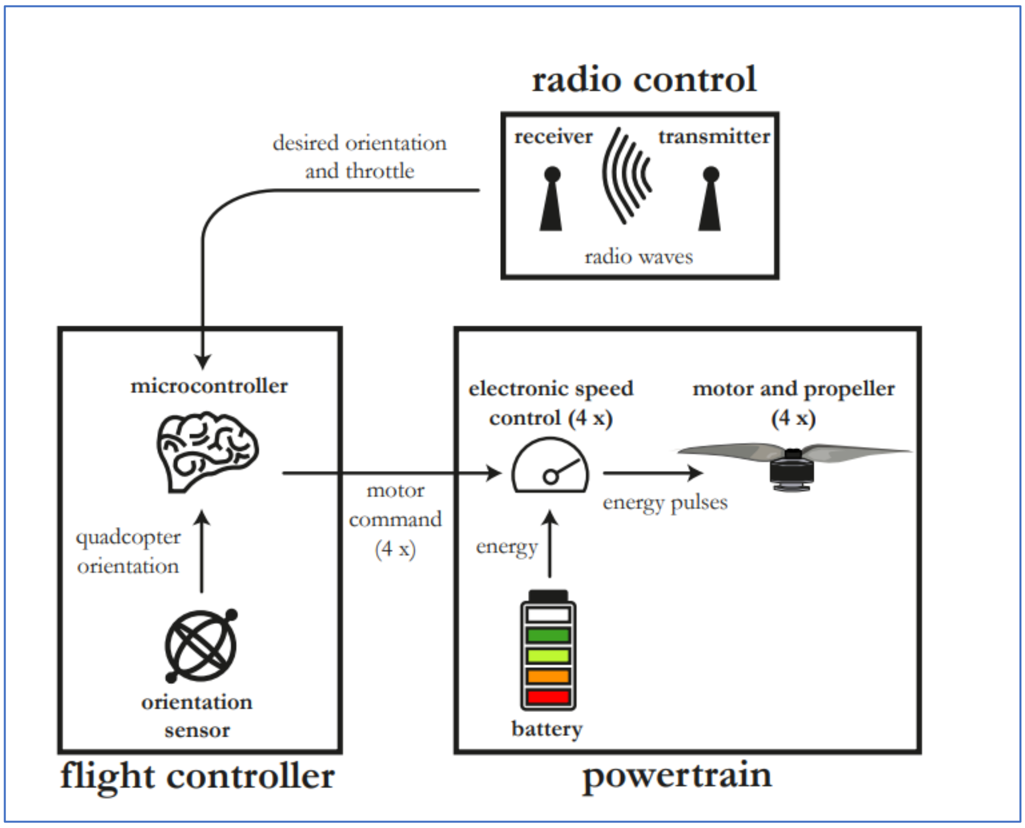
1) Introduction:

The creation of flying machines is a true engineering challenge and involves solving several problems, from aerodynamics to power systems. In the case of a quadcopter, you rely on four motors and propellers to provide enough thrust to start flying. Obviously, these are not the only necessary components. The following figure displays the basic overview of a quadcopter with three major active building blocks



The radio control system, which consists of a radio transmitter and a receiver. The position of the sticks on the radio transmitter are transformed into commands and subsequently sent to the receiver that is situated on your quadcopter.

The flight control system, which consist of a microcontroller and some sensors. The bare minimum you need to stabilize the quadcopter is an orientation sensor, you can add a module like MPU 6050 which consists of accelerometer and gyroscope to make flight more stable. The information of your sensor and the commands from your radio transmitter are then processed in the microcontroller, which is the brain of your quadcopter. The microcontroller calculates the optimal speed of each of the four motors to keep the quadcopter in the air.

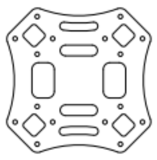
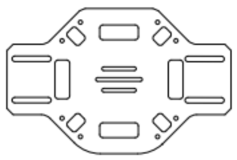
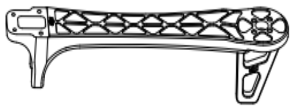
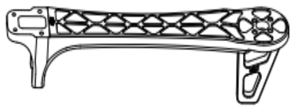
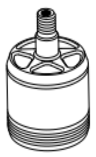
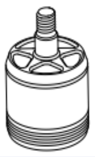
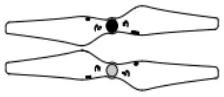
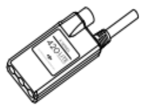
The third building block is the powertrain, which is the high current part of the quadcopter. The battery is the power source of the whole system and sends energy in the form of electrical current to four electronic speed controllers (ESCs) via power distribution board an ESCs converts the provided current into current pulses, with a pulse length proportional to the motor command sent from the microcontroller. This gives a motor speed proportional to the motor command and in turn, a certain thrust allowing you to take off.

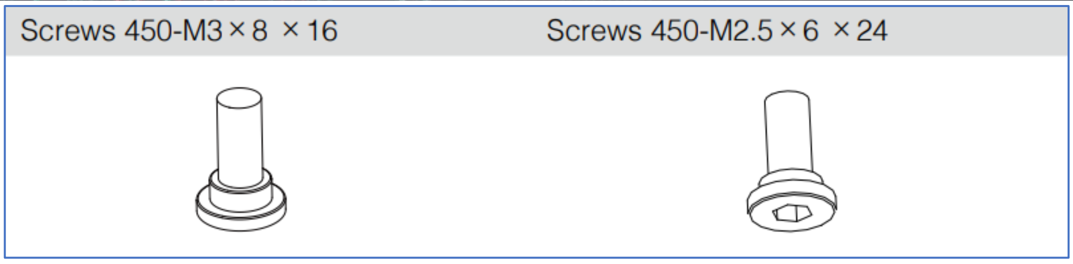
And basically, that's all there is to it, With the general idea behind our quadcopter clearly understood, let's have a look at all different physical parts that we will use. Our quadcopter consists of three active building blocks; a radio control system, flight controller and powertrain. Moreover, we also need a frame on which we can mount all these active components. Some auxiliary parts are also necessary, to charge the battery and test our microcontroller, sensors and powertrain before fixing them to the frame. All necessary components are listed below and divided into parts for the frame, flight controller, powertrain, battery and radio control.

2) Components:

2.1) Frame:

The frame we are using in our project is F450, the parts which comes within the package are as follows:

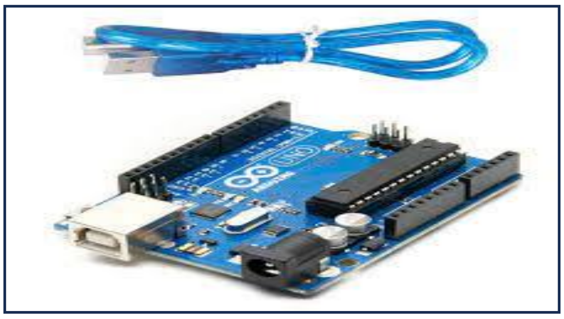
Top Board 450FBT ×1	Bottom Board 450FBB ×1
	
Frame Arms 450FAC ×2	Frame Arms 450FAW ×2
	
Motors CW ×2	Motors CCW ×2
	
Propeller Pairs	ESC ×4
	

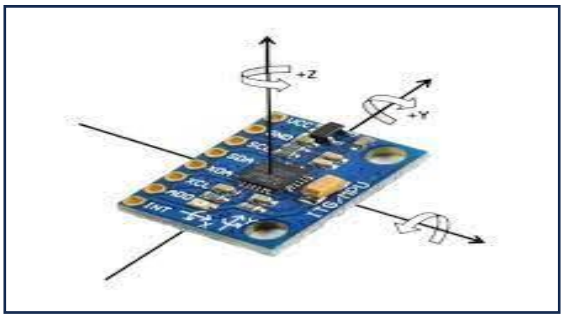


2.2) Flight Controller:

We can use a readymade flight controller for our project like KK2.1.5 , but in this project for learning experience we are going to make our own flight controller the components used for making flight controller are follows;

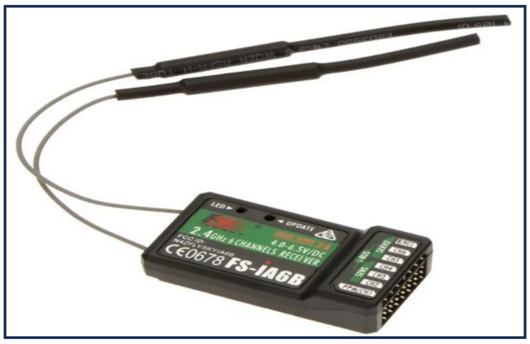
- Arduino UNO
- MPU 6050
- Jumper wires
- Arduino IDE (for uploading the code)


Arduino UNO


MPU 6050

2.3) Receiver:

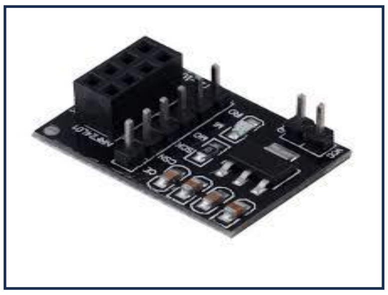
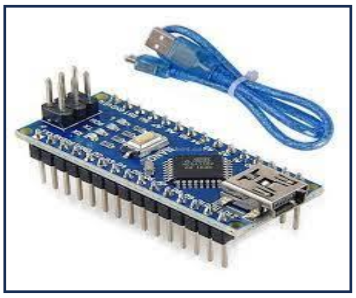
We can use FlySky receiver for our project, but for learning purpose we will be making our own receiver.



For making our own receiver the following components will be used

- Arduino NANO
- NRF Module

- NRF Power Adapter
- Jumper Wires
- Capacitor

2.4) <u>Transmitter:</u>

We can use FlySky remote as a transmitter

For making our own transmitter the following components will be required

- Arduino NANO
- NRF Module
- NRF Power Adapter
- Jumper Wires
- Toggle Switch
- ON/OFF Button
- Variable Resistor
- Red LED

2.5) <u>Battery:</u>

The battery for drone is LiPo battery Pack 2200mAh 3S

This battery will be used to power the receiver, Arduino UNO(flight Controller) and 4 brushless motors

To power the transmitter, we will be using 2 cells of 3.5v in series to achieve 7 volt supply to transmitter

2.6) Others:

The following components were also used to facilitate in the project

- Soldering Gun
- Soldering Wire
- Glue Gun
- Wire Stripper
- L-Key screw driver
- Veroboard
- Digital Multi Meter
- Header Pins

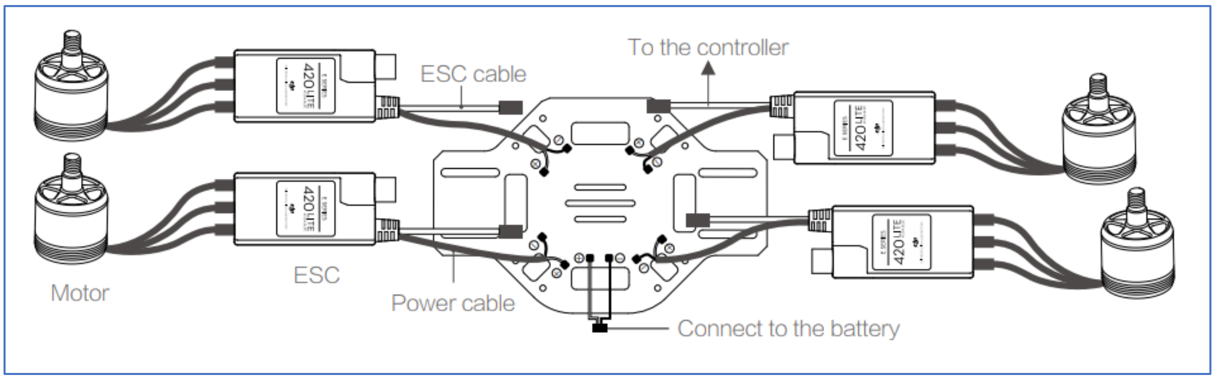
So, from the above discussion the parts used in this project were described, now lets move on to the project and have a look at the working principle of drone and how the above parts should be used in making of the Quadcopter.

3) Making of 3 Building Blocks:

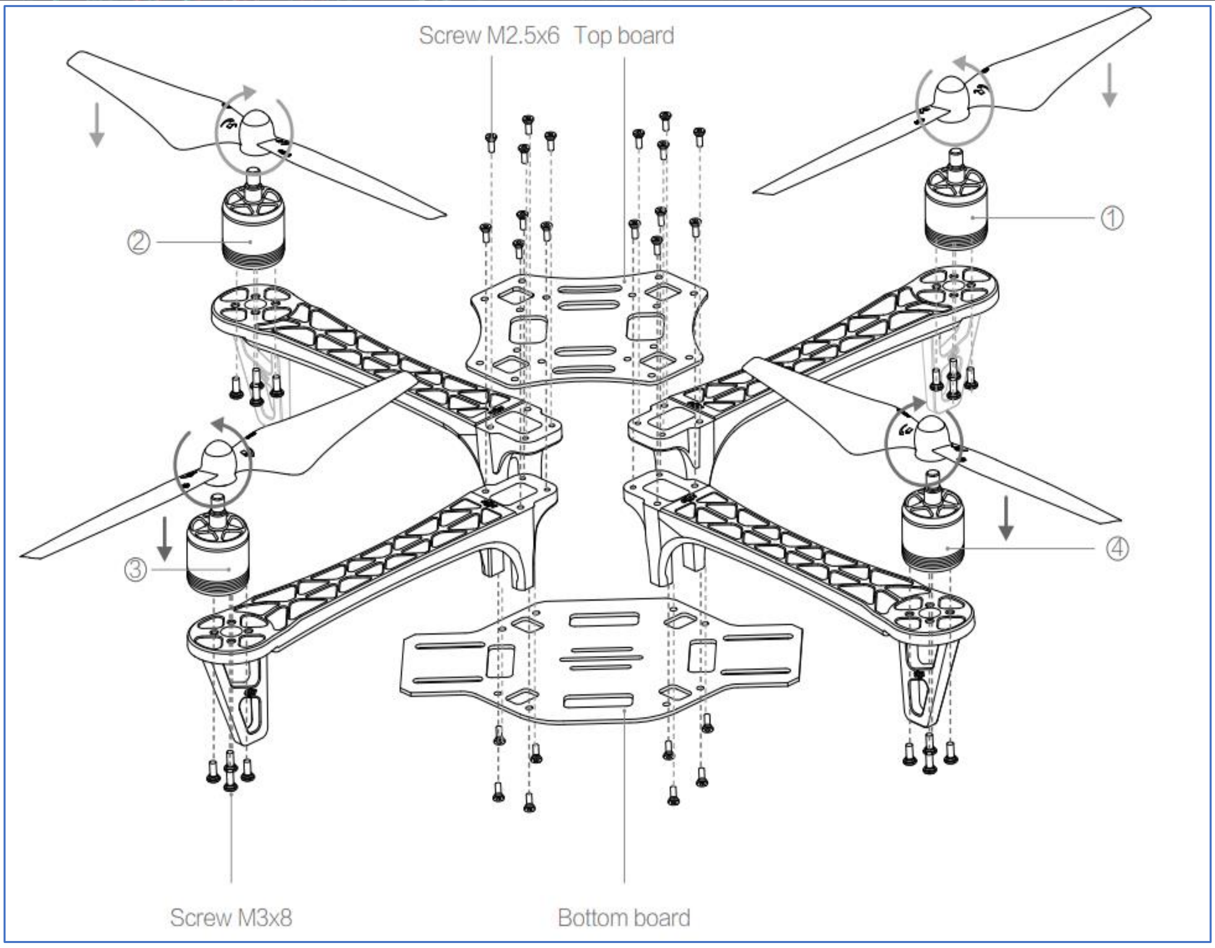
3.1) Assembling Frame:

Connection and Soldering:

The ESC controller is soldered to power distribution plate, then the brushless motor 3 wires are connected to ESC controller, after this the circuit should look like the following.



After the soldering then assemble the frame as shown in the following figure



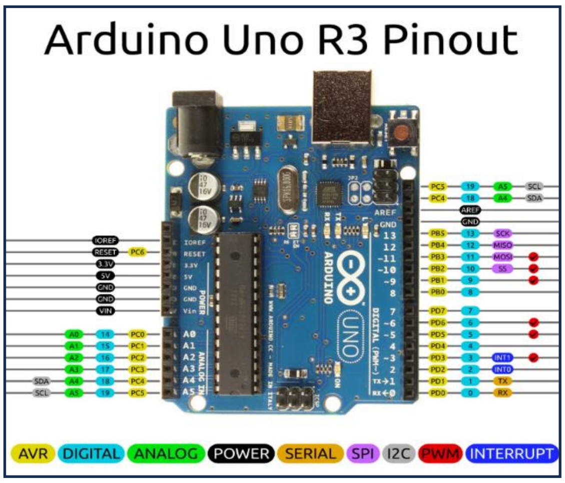
This frame will be used to host the flight controller, MPU 6050 ,4 brushless, motors and ESC controllers. The power supply will be given by the 2200mAH 3S LiPo battery and power to all ESC will be distributed via a distribution board that comes with the frame itself, the power to receiver will be delivered via ESC controllers and the power to Arduino UNO (flight controller) and lastly the MPU will be powered by Arduino UNO.

3.2) Making of Flight Controller:

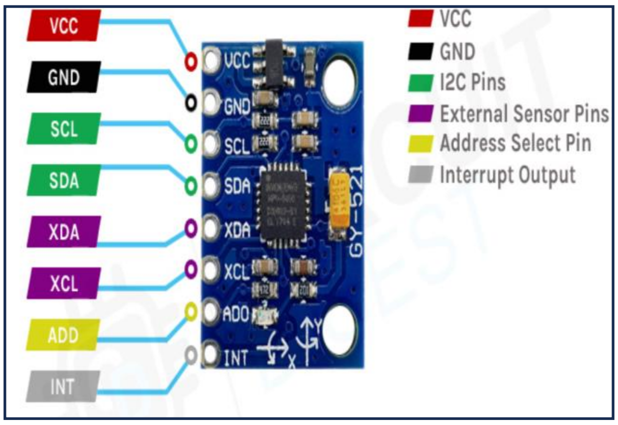
The flight controller consists of Arduino UNO and MPU 6050

3.2.1) Arduino UNO:

Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button.



3.2.2) MPU 6050:



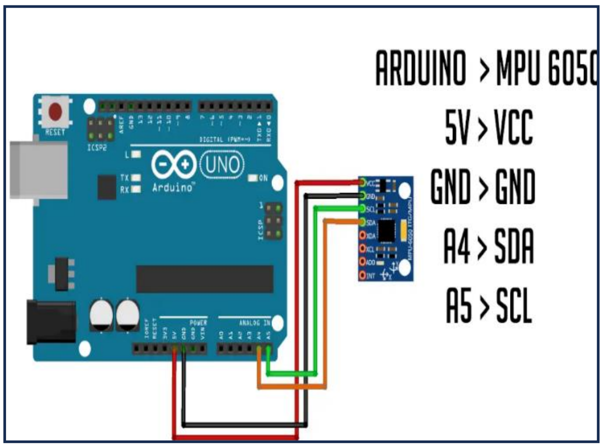
The MPU6050 is a Micro-Electro-Mechanical Systems (MEMS) that consists of a 3-axis Accelerometer and 3-axis Gyroscope inside it. This helps us to measure acceleration, velocity, orientation, displacement and many other motion-related parameters of a system or object.

3.2.3) Interfacing MPU with UNO:

The connectivity of the MPU is like the following, from Arduino to MPU , 5v , GND , A4 and A5 , to Vcc , GND , SDA and SCL respectively.

SCL=>Serial Clock Pin

SDA=>Serial Data Pin



The code which is burned on Arduino UNO have the following important attributes

3.2.3.1) Control Loop:

The main loop runs at a constant frequency, crucial for accurate control .PID controllers stabilize the quadcopter by adjusting motor signals based on sensor feedback.

3.2.3.2) PID Controller:

Computes control outputs (corrections) based on the error between desired and actual orientations. Consists of proportional, integral, and derivative terms to adjust control inputs. Output influences motor signals directly, maintaining stability.

3.2.3.3) Sensor Feedback:

Gyro sensors provide real-time angular rate measurements. Essential for accurate stabilization calculations by the PID controller. Receiver signals translate user input into desired orientations for stabilization.

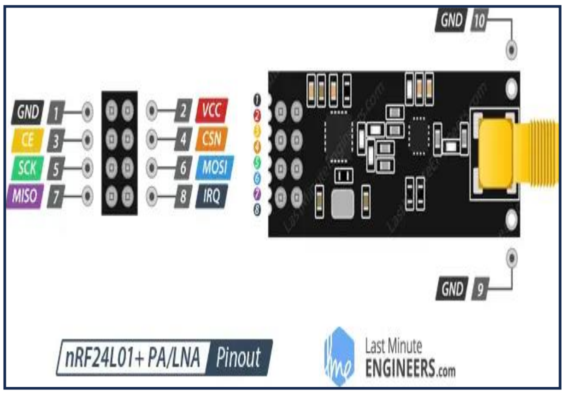
3.2.3.4) Interdependencies: PID controller relies on gyro sensor feedback and user input (receiver signals) to calculate corrections. Control outputs directly impact motor signals, affecting quadcopter orientation. Sensor readings are crucial for precise control adjustments and stable flight.

3.3) Making of Transmitter:

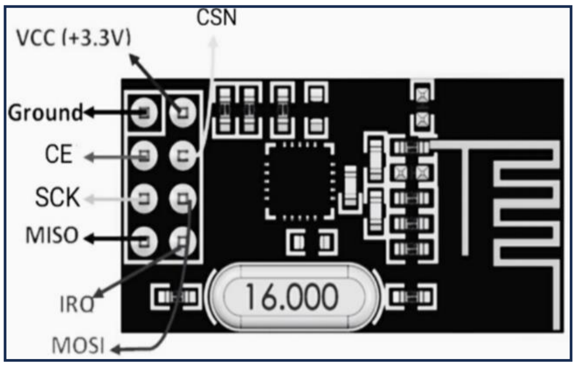
The transmitter used in the project is composed of the following components Arduino Nano, NRF Module, NRF Power Adapter, Joysticks, Buttons, Variable Resistor and 7v Battery.

3.3.1) NRF Antenna Module:

NRF24L01 RF Module. NRF24L01 is a radio transceiver module (SPI protocol) used to send and receive data at ISM operating frequency from 2.4 to 2.5 GHz. This transceiver module is composed of a frequency generator, beat controller, power amplifier, crystal oscillator modulator, and demodulator.



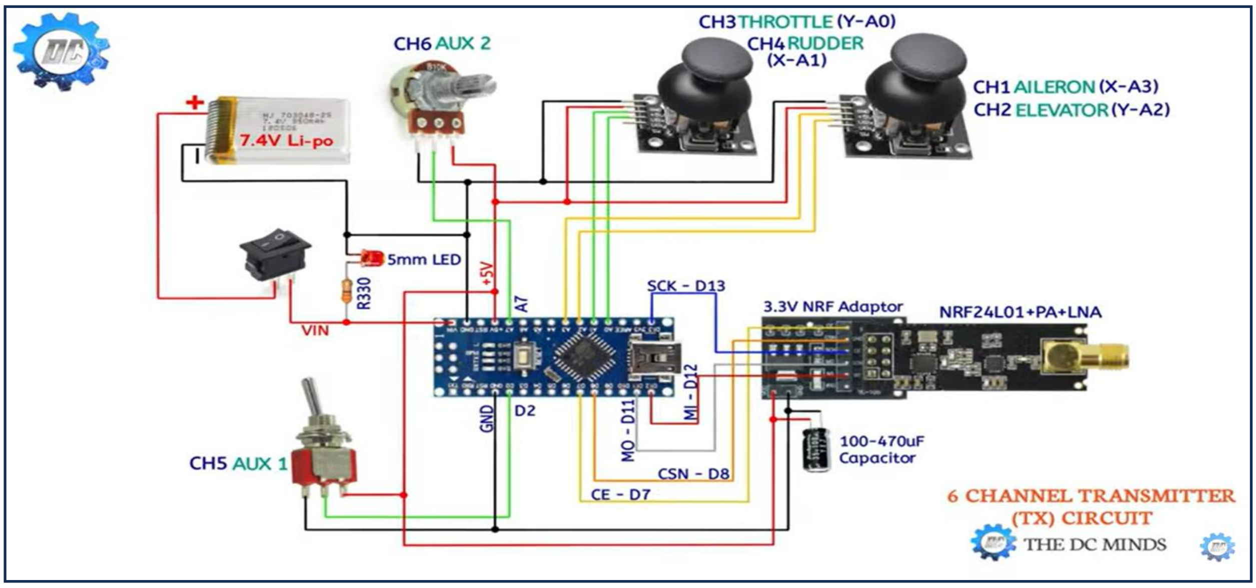
3.3.2) NRF Power Adapter:



It regulates the 3.3V input to 1.9~3.6V DC and incorporates bypass capacitors for reliable operation. The NRF24L01 utilizes Enhanced ShockBurst (ESB) protocol to support two-way data packet communication with packet buffering, packet acknowledgement and automatic re-transmission of lost packets

3.3.3) Circuit Diagram:

The following circuit diagram was followed in making of a transmitter



Following attributes are contained by transmitter's code

3.3.3.1) Control Data Structure:

The Signal struct defines the data packet structure containing throttle, yaw, pitch, roll, and auxiliary control values.

3.3.3.2) Joystick Mapping:

mapJoystickValues() function scales raw joystick input values to fit within a defined range (0-255) for each control axis.Joystick values are mapped to the range 0-128-255 to fit within the byte data type range.

3.3.3.3) Data Acquisition:

Raw joystick values are read from analog and digital pins using analogRead() and digitalRead() functions.

3.3.3.4) Wireless Communication:

NRF24L01 module is initialized for wireless communication using the RF24 library.The transmitter sends control data packets wirelessly to a receiver using the radio.write() function.

3.3.3.5) Serial Debugging:

Serial output is used for debugging, displaying raw and mapped joystick values.

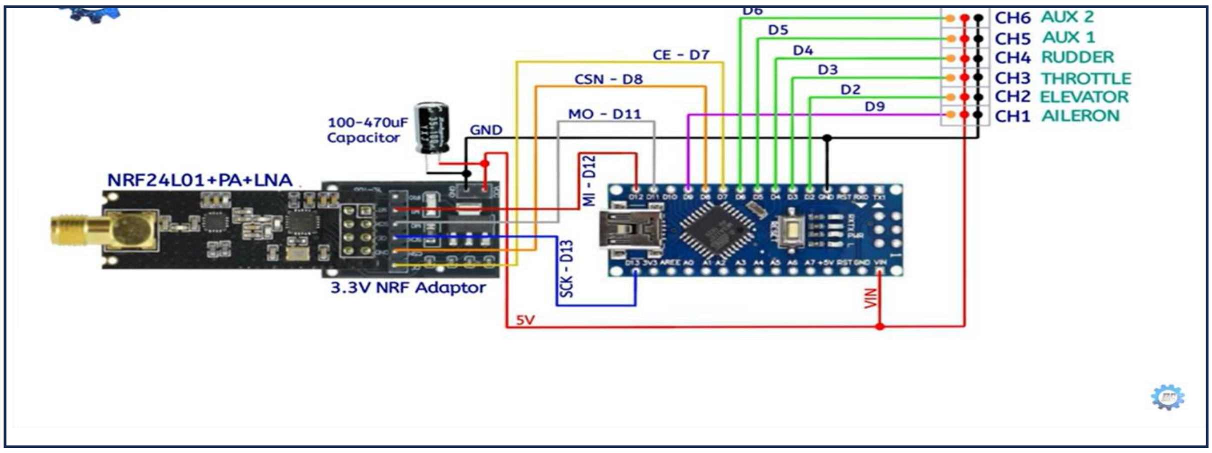
3.3.3.6) Interdependency:

The mapping of joystick values ensures that control signals are within a standardized range suitable for transmission. Reliable wireless communication is crucial for transmitting control data to the receiver for actuation.

<u>3.4) Making of receiver:</u>

The receiver consists of the Arduino nano, NRF Antenna Module, NRF Power Adapter and capacitor

<u>3.4.1) Circuit diagram:</u>



The following attributes are contained in the code of receiver

3.4.1.1) Control Signal Processing:

Received control signals (throttle, yaw, pitch, roll, aux1, aux2) are stored in the Signal struct. Control signal values are mapped from 0-255 range to servo pulse width range (1000-2000 microseconds) for each channel.

3.4.1.2) Servo Control:

Six servos (ch1-ch6) are initialized and attached to respective pins. Servo positions are set based on the mapped control signal values using writeMicroseconds().

3.4.1.3) Wireless Communication:

NRF24L01 module is configured to receive data wirelessly from the transmitter. The receiver constantly listens for incoming data packets and updates control signals accordingly.

3.4.1.4) Data Reception:

recvData() function continuously reads incoming data packets from the transmitter and updates the data struct.

3.4.1.5) Signal Reset:

If no data is received for more than 1 second, control signals are reset to default values using the ResetData() function.

3.4.1.6) Serial Output:

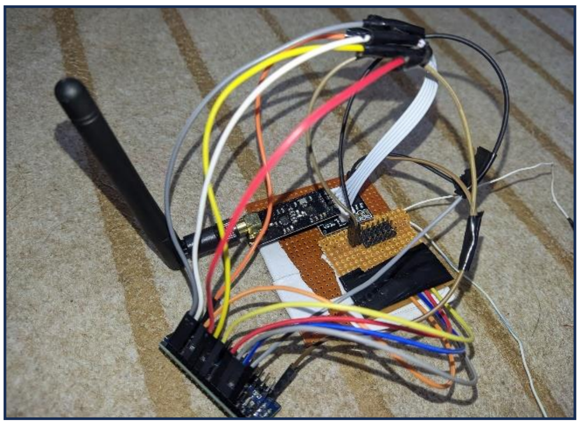
Serial output is used for debugging, displaying received control signal values, mapped servo pulse widths, and channel widths.

3.4.1.7) Interdependency:

The receiver's functionality depends on continuous and reliable wireless communication with the transmitter to receive control signals. Servo control depends on accurately mapping received control signal values to appropriate servo pulse widths.

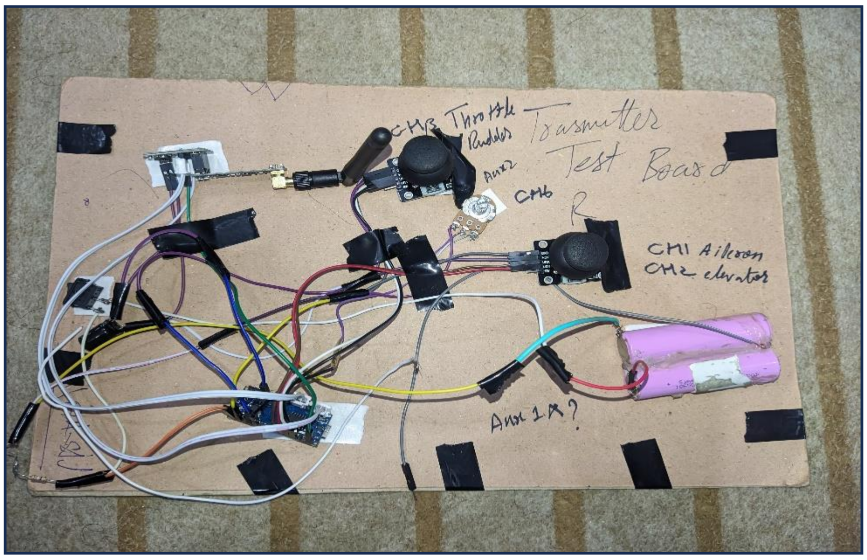
4) <u>Our Project:</u>

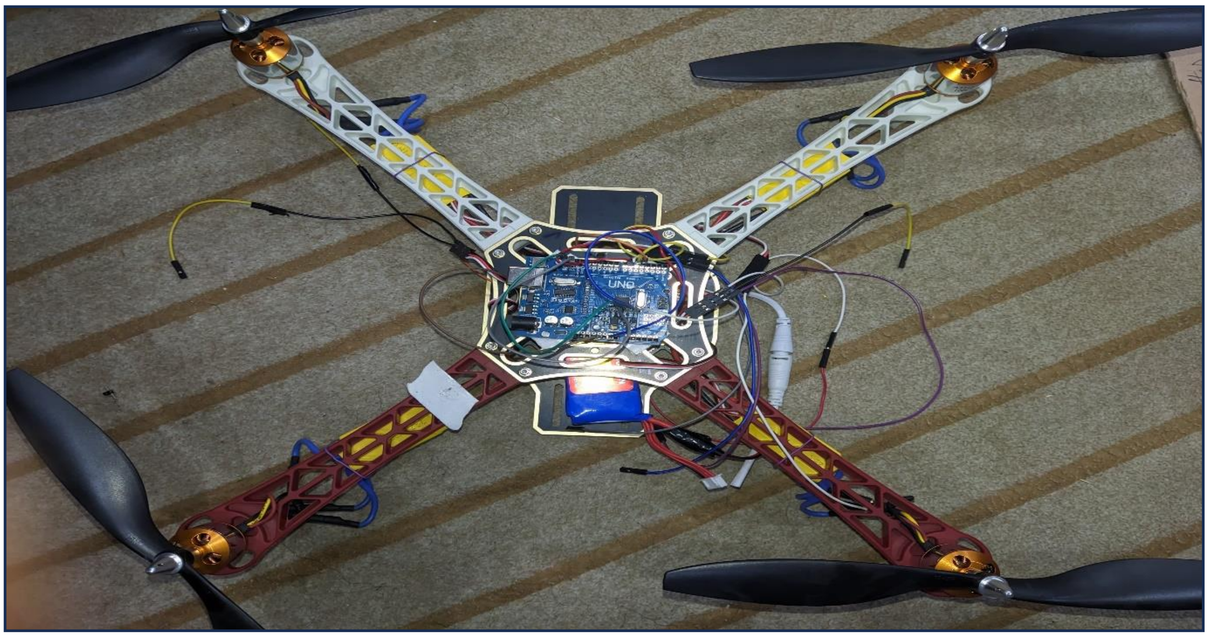
After making the above components our project snapshots are following



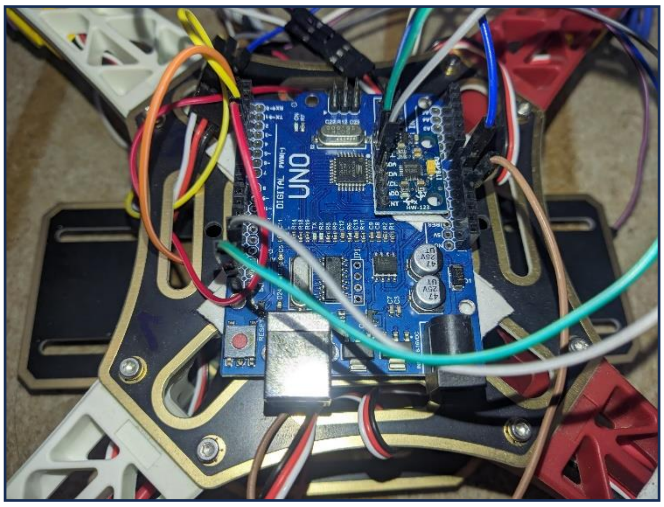
This is the receiver for our project

This is the prototype of our transmitter





This is the the frame of quadcopter with all the essential components on it

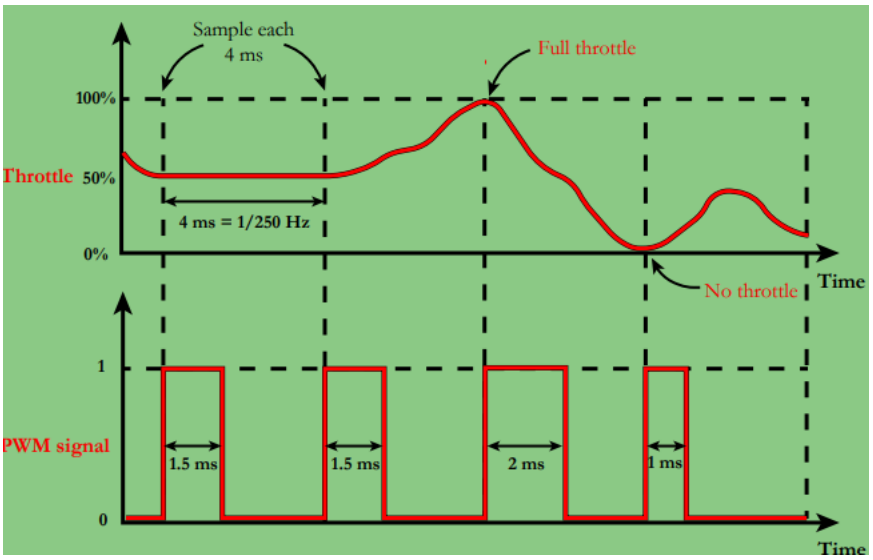


This is the flight controller

5) Working:

5.1) ESC Control through PWM:

We can control the speed of your motor through the ESC, which in turn gets its commands from channel 3 (the throttle channel) of the receiver. The receiver sends Pulse Width Modulated (PWM) signals to the ESC to the channel, indicating a desired throttle value between 0 and 100%. But what is a PWM signal, really? In essence, a PWM signal is just a signal that varies between a HIGH voltage (for example 5V) and a LOW voltage (for example 0V), or between 1 and 0 to keep the concept simple. It is the time length during which the signal stays HIGH that is used to transfer information. Suppose for example that a time length of 1 ms HIGH corresponds with no throttle (0%) and a time length of 2 ms HIGH corresponds with full throttle (100%). The PWM frequency of most receivers and ESCs is 4 ms (or 1/0.004=250 Hz), meaning that chosen signal repeats itself every 4 ms with a length that corresponds with throttle command. This concept is illustrated in the figure below and will be used later on to send commands to your ESCs using Teensy instead of directly from the receiver.



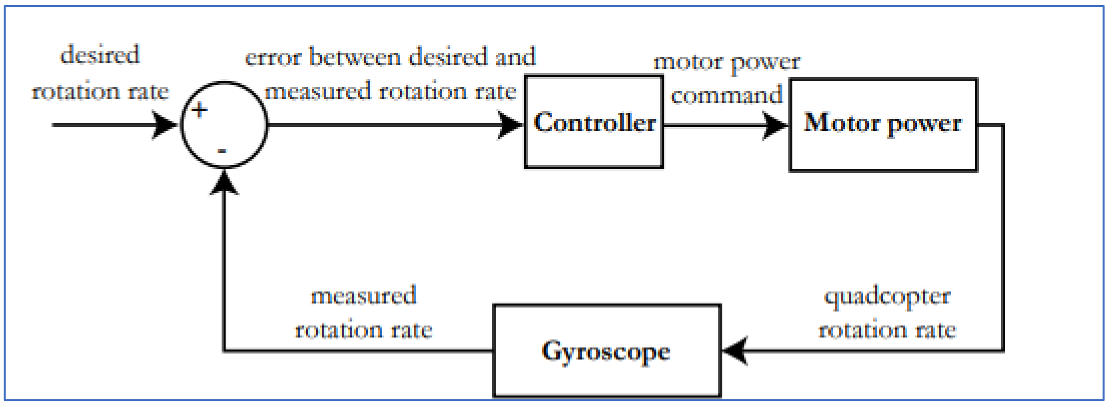
The commands that we give through our radio controller are transmitted via radio waves and picked up by our receiver. The receiver then converts the radio waves to a signal which can be read by microcontroller. We will now see how to convert these signals from the receiver to variables that can be used further in the flight code. There are multiple methods to transfer information through digital signals, one of which is Pulse Width Modulation (PWM). PWM is an easy method to send information of one radio channel from the receiver to the microcontroller. However, receiving information from multiple radio channels would require one signal cable to the microcontroller for each channel.

5.2) Quadcopter Control:

The system consists of the following components:

- **Gyroscope:** This sensor measures the angular velocity of the quadcopter around its roll, pitch, and yaw axes.

- **Controller:** This compares the desired rotation rate of the quadcopter to the measured rotation rate from the gyroscope. The difference between these two values is called the error. The controller uses this error signal to calculate the appropriate motor power commands.
- **Motor power:** The controller sends commands to the quadcopter's motors, which adjust their speed to create the desired rotation.



<u>5.3) Motor and Sensor Control:</u>

The system entails these primary components:

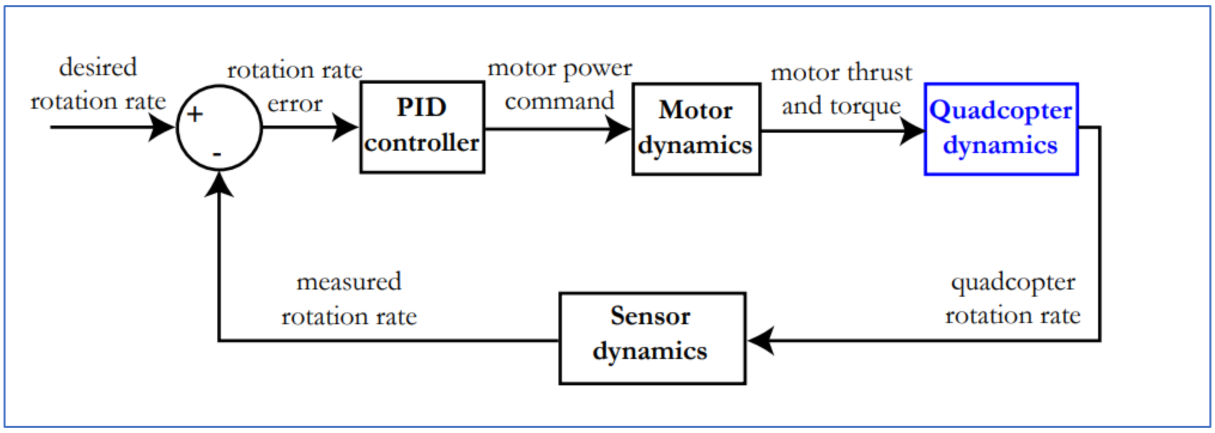
- **Desired Rotation Rate:** This block signifies the pilot's desired rotational speed for the quadcopter.
- **Sensor Dynamics:** This block represents the sensors that measure the quadcopter's actual rotation rate. These sensors can include gyroscopes and accelerometers.
- **Measured Rotation Rate:** This block indicates the actual rotation rate of the quadcopter as picked up by the sensors.
- **Error:** This block calculates the difference between the desired rotation rate and the measured rotation rate.
- **PID Controller:** This block stands for proportional-integral-derivative controller. It uses the error signal to calculate the necessary motor commands to minimize the error.
- **Motor Dynamics:** This block signifies the mathematical model that simulates the quadcopter's motors.
- **Motor Power and Torque:** This block signifies the motor commands sent by the PID controller to the quadcopter’s motors. These commands determine the amount of power supplied to the motors, which in turn affects the amount of torque generated by the rotors.
- **Quadcopter Dynamics:** This block represents the mathematical model that simulates the physical characteristics of the quadcopter that affect its motion, such as its mass, inertia, and blade aerodynamics.

The feedback loop functions in the following manner:

The pilot inputs a desired rotation rate. The sensor dynamics block simulates the sensors that measure the helicopter's actual rotation rate. The error block calculates the difference between the desired and actual rotation rates. The PID controller uses the error signal to compute motor commands that attempt to reduce the error. The motor dynamics block simulates the quadcopter's motors and the torque they generate based on the motor commands. The quadcopter dynamics block simulates the quadcopter's response to the changes in torque, including its new rotation rate.

The sensor dynamics and quadcopter dynamics blocks introduce a time delay between when the motor commands are sent and when the quadcopter's rotation rate changes. This time delay needs to be accounted for by the PID controller to maintain stability in the simulated environment.

5.4) Quadcopter Dynamics Simulation:



- **Desired Rotation Rate:** This block signifies the pilot's desired rotational speed for the quadcopter on each axis (roll, pitch, yaw).
- **Sensor Dynamics:** This block represents the sensors that measure the quadcopter's actual rotation rates. These sensors can include gyroscopes and accelerometers.
- **Measured Rotation Rate:** This block indicates the actual rotation rate of the quadcopter on each axis as picked up by the sensors.
- **Error:** This block calculates the difference between the desired rotation rate and the measured rotation rate for each axis.
- **PID Controller:** This block stands for proportional-integral-derivative controller. It uses the error signal for each axis to calculate the necessary motor commands to minimize the error.
- **Motor Power and Torque:** This block signifies the motor commands sent by the PID controller to the quadcopter's motors. These commands determine the amount of power supplied to the motors, which in turn affects the amount of torque generated by each rotor.
- **Quadcopter Dynamics:** This block represents the mathematical model that simulates the physical characteristics of the quadcopter that affect its motion, such as its mass, inertia, and blade aerodynamics.

The feedback loop works as follows:

1. The pilot inputs a desired rotation rate for each axis (roll, pitch, yaw).
2. The sensor dynamics block simulates the sensors that measure the helicopter's actual rotation rates.
3. The error block calculates the difference between the desired and actual rotation rates for each axis.
4. The PID controller uses the error signal for each axis to compute motor commands that attempt to reduce the error.
5. The motor power and torque block translate the motor commands into actual power delivered to the rotors, resulting in a change in torque.
6. The quadcopter dynamics block simulates how the quadcopter responds to the changes in torque, including its new rotation rates.

The sensor dynamics and quadcopter dynamics blocks introduce a time delay between when the motor commands are sent and when the quadcopter's rotation rates change. This time delay needs to be accounted for by the PID controller to maintain stability.

6) <u>Conclusion:</u>

This project details the comprehensive development of a quadcopter controlled by an Arduino-based flight system. Utilizing an Arduino UNO paired with an MPU 6050 sensor for the flight controller, and Arduino Nano with NRF24L01+ modules for both transmitting and receiving signals, we have established a robust control architecture. The transmitter system captures joystick inputs and transmits them wirelessly. The receiver processes these signals and communicates with the flight controller. The flight controller uses the MPU 6050 data to stabilize the quadcopter and adjust motor speeds through ESCs. Mathematical modeling, including Newton-Euler equations, supports the theoretical foundation for quadcopter dynamics and stabilization using PID control. This setup demonstrates a practical and scalable approach to building and controlling quadcopters, emphasizing the integration of sensors, wireless communication, and control algorithms.

By focusing on the use of renewable energy sources and efficient flight control systems, this project aligns with several United Nations Sustainable Development Goals (SDGs), including SDG 7 (Affordable and Clean Energy) and SDG 9 (Industry, Innovation, and Infrastructure). The quadcopter's design and implementation promote innovation in drone technology and support sustainable development initiatives.