Documentation for the NRF24L01-based controller.

1. Reason for making a custom controller
2. nRF24L01+PA+LNA module
3. Reason for using the NRF module
4. About NRF, it’s features and ESB protocol.
5. SPI protocol and it’s Noise sensitivity
6. Code and documenation
7. Circuitry
8. Prototype 1

## Reason for making a custom controller

Till now the only way to move the rover was sending commands through the rocket m2, which was already saturated with the camera feed, and other communications.

This led to a (reported) 4-5 second delay in controlling the rover.

Hence we had to separate the manual control of the rover away from the rocket m2 so that we could control the rover properly even if the camera feed was fully saturated

There was also another reason the seniors mentioned where there was a lot of interference from other teams using the same 2.4ghz band. But that’s from the electrical seniors. The software seniors said there was no interference, just that the camera feed saturated the rocket m2, and also that there was some sort of bad connection. So there is some confusion around that.

Regardless, the main reason imo for making this controller is the first point, to separate the rover’s controls from the remaining communications.

Now, the reason we are not using a commercial controller despite it being better in every aspect is mostly because of the simplicity and modularity of a custom controller. A custom controller allows us to modify it based on how we decide to control the rover, its robotic arm and science kit, etc. while a commercial controller means we are limited to the controls already present in the controller.

## NRF24L01+PA+LNA module

### Reason for using the NRF module

This module was absolutely painful to work with. But once we got it working, it’s a charm. It’s almost amusing how many features this bad boy packs for 250rs

First off we had to decide on a radio communication module and we boiled it down to two - NRF24L01 and HC12. The reason why we chose these two can be separated into two - *electrical seniors are right* or *software seniors are right* (about the interference in the competition).

The HC12 module says electrical seniors are right. I mean, it solves the problem of separating control from communication yes. But it is a dumber module –with less features, and less range. It has only about 300 meters direct line of sight (LOS) range (which you’ll see is inferior soon). But the one problem it solves is that it communicates on the 433mhz rf band, which rarely any teams use, so you won't have any sort of interference with either the 2.4gz rocket m2 or other teams comms –*assuming* there’s no frequency jamming.

I'm not covering HC12 and anything related to it in this doc due to my lack of knowledge of it.

The NRF24L01 module says the software seniors are right. The delay in communication was due to the rocket m2 getting saturated with the camera feed, hence separating the control from the comms will fix the delay in control. There will also be no significant interference with other 2.4 GHz devices because the NRF module is quite smart and has many features (explained below). Also with the NRF24L01+PA+LNA module, you can achieve up to 1.1km with direct LOS (of which we’ve tested 600m before we ran out of road (also the nrf module was kinda underpowered in this test) (**Based on rough testing)** ). **Proper Testing needs to be done without direct LOS, but it could very possibly reach the same range in this case too. ??**

### About NRF, it’s features and ESB protocol.

For the best range, we’re using NRF24L01+PA+LNA, PA: (power amp) amplifies when transmitting. LNA: (low noise amp) amplifies when receiving. both sit between circuitry and antenna. Here’s a stack thread on PA LNA and duplexers for reference: [amplifier - What is a PA/LNA? - Electrical Engineering Stack Exchange](https://electronics.stackexchange.com/questions/237267/what-is-a-pa-lna)

### SPI protocol and it’s noise sensitivity

SPI (Serial peripheral interface) is another pain. Assuming we know about how SPI works. Here’s what i learnt with spi while working with it.

SPI communication is prone to noise so it’s best to isolate it from other electronics and use short wires.

Ive used a 10uF capacitor across VCC and ground in the circuit diagram. But there is a base module with several decoupling capacitors and resistors **(needs to be measured and circuitry figured out because i couldnt find the datasheets online).** We are trying to reduce the noise in the input voltage by decoupling vcc and ground. So a higher impedance between them and the smaller capacitor value, more decoupling.

Apart from that. There is not much to SPI. it is a very straight forward protocol. You choose which peripheral to communicate with using chip select pins, so there’s no slave address taking up half the data like i2c, and it’s synchronized.

### Code and documenation

The code for the nrf module was a mess to write. At first we werent able to find any resources (later turns out that was mostly skill issue).

These are the final resources we used the most:

* [Optimized high speed nRF24L01+ driver class documentation: RF24 Class Reference](https://nrf24.github.io/RF24/classRF24.html) - we used nRF24L01+ library by TMRh20.
* [Arduino + NRF24 simple tutorial and range test - YouTube](https://www.youtube.com/watch?v=57pdX6b0sfw) - this video gives the best (imo) overview on the module
* [nRF24L01 – Arduino Interfacing, Circuits, Codes, PA + LNA (electroniclinic.com)](https://www.electroniclinic.com/nrf24l01-arduino-interfacing-circuits-codes-pa-lna/) - an article from which i referred to about the decoupling capacitors.

Here’s the prototype 1 code:

//#################################

//#################################

//transmitter code

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

RF24 radio(4, 5); // CE, CSN

byte address[6] = "00001";

#define joy1\_x 14

#define joy1\_y 12

#define buttons 32

#define joy2\_x 26

#define joy2\_y 27

void setup() {

//initializing pins

// Joystick 1

pinMode(joy1\_x, INPUT);

pinMode(joy1\_y, INPUT);

//buttons

pinMode(buttons, INPUT);

// Joystick 2

pinMode(joy2\_x, INPUT);

pinMode(joy2\_y, INPUT);

// begining the module/library

if (!radio.begin()) {

Serial.println(F("radio hardware not responding!"));

while (1) {} // hold program in infinite loop to prevent subsequent errors

}

Serial.begin(115200);

Serial.println("start");

//the RF module's power level:

//Options: RF24\_PA\_MIN, RF24\_PA\_LOW, RF24\_PA\_HIGH, RF24\_PA\_MAX

radio.setPALevel(RF24\_PA\_MAX);

//setting this RF module to transmit mode:

radio.stopListening();

// Open the writing pipe with the defined address

// This is the address the data will be sent to

// use a different address if you wanna send the data to a different module.

radio.openWritingPipe(address);

}

struct controllerData{

int joystick1\_x;

int joystick1\_y;

int Button;

int joystick2\_x;

int joystick2\_y;

};

controllerData payload;

void loop() {

// Joystick 1

payload.joystick1\_x = analogRead(joy1\_x);

payload.joystick1\_y = analogRead(joy1\_y);

//buttons

payload.Button = analogRead(buttons);

// Joystick 2

payload.joystick2\_x = analogRead(joy2\_x);

payload.joystick2\_y = analogRead(joy2\_y);

Serial.print(payload.joystick1\_x);

Serial.print(",");

Serial.print(payload.joystick1\_y);

Serial.println(" ");

Serial.print(payload.Button);

Serial.println(",");

Serial.print(payload.joystick2\_x);

Serial.print(",");

Serial.print(payload.joystick2\_y);

Serial.println(" ");

Serial.println(" ");

Serial.println(" ");

Serial.println(" ");

radio.write(&payload, sizeof(payload));

delay(20);

}

//#################################

//#################################

//Reciever code

#include <SPI.h>

#include <nRF24L01.h>

#include <RF24.h>

RF24 radio(4, 5); // CE, CSN

byte address[6] = "00001";

void setup() {

//begining the module/library

if (!radio.begin()) {

Serial.println(F("radio hardware not responding!"));

while (1) {} // hold program in infinite loop to prevent subsequent errors

}

Serial.begin(115200);

Serial.println("start");

//the RF module's power level:

//Options: RF24\_PA\_MIN, RF24\_PA\_LOW, RF24\_PA\_HIGH, RF24\_PA\_MAX

radio.setPALevel(RF24\_PA\_MAX);

//opening reading "pipe" no.1 at the given address

radio.openReadingPipe(1, address);

//setting this RF module to recieve mode:

radio.startListening();

}

struct controllerpayload{

int joystick1\_x;

int joystick1\_y;

int Button;

int joystick2\_x;

int joystick2\_y;

};

void loop() {

controllerpayload payload;

uint8\_t pipe1; //create a buffer to which the 1st pipe writes to

if (radio.available(&pipe1)){ //see if there are any bytes available to read on pipe1

Serial.println("payload available:");

uint8\_t length = radio.getDynamicPayloadSize(); //get the length of the payload from RX FIFO pipe

Serial.println(length);

radio.read(&payload, length); //read the from the pipe the length of the payload from the above line

//printing the joystick values:

Serial.print("Joystick1 X: "); Serial.print(payload.joystick1\_x);

Serial.print(", Joystick1 Y: "); Serial.println(payload.joystick1\_y);

Serial.print("button value: "); Serial.print(payload.Button); Serial.println(", ");

Serial.print("Joystick2 X: "); Serial.print(payload.joystick2\_x);

Serial.print(", Joystick2 Y: "); Serial.println(payload.joystick2\_y);

Serial.println(" ");

Serial.println(" ");

Serial.println(" ");

}

//else {

// Serial.println(-1);

//}

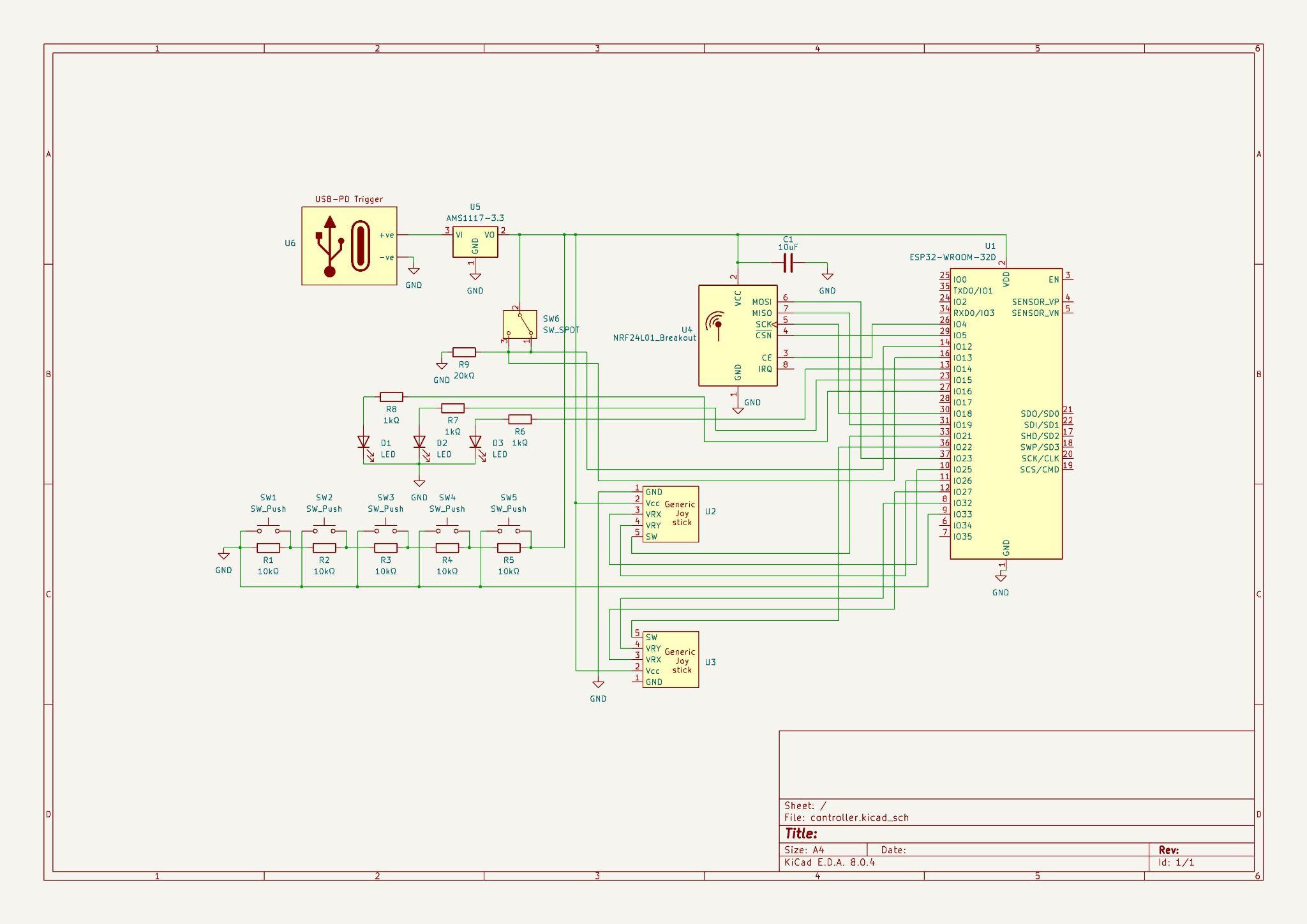
}

The reason for each function is explained in the comments.

It uses a few optional configurators like setPAlevel. Other than that it’s pretty basic.

## Circuitry

### Prototype 1:



Receiver circuit is the same except without all the inputs.