# Microcontrollers

there are a few things more than just finding new microcontrollers that I wanna report about here. Best to list it out:

1. Finding better microcontrollers by spec
2. Choosing a replacement to the ESP32 based on availability and pricing
3. Esp32 Dual core operations
4. Connecting external RAM to ESP32, and to the new microcontroller if we're switching to any
5. Using multiple MCUs, and the embedded control setup now that we have a rover to build upon.

Finding the better microcontroller by spec is pretty straightforward. Im looking at CoreMark scores and the amount of RAM available mostly. Im gonna ignore the DMIP score because there's a lot of talk online about how it's a flawed metric

https://www.eembc.org/coremark/scores.php

the above is a list of all the MCUs and MPUs and their scores on coremark

there are three primary microcontrollers with their own pros:

esp32 – cheap and easily available dev boards, pretty high performance for the price. Has dual cores, and we aren't utilizing both cores fully. MCU is available separately. Bad ADC, but we could use a different board for that.

stm32 – there's possible backwards compatibility, so we can test codes with a slower stm32 and upgrade to a faster one. there are chips with the same CPU but more GPIOs. The highest performing MCUs are available from ST. used heavily in the industry, learning how to use STMs would be a very handy tool. There's CAM communication protocol

teensy 4 – easily available dev boards, while also being faster than esp32. Overclockable upto 900mhz -just this I wanna try out.

(the following table is outdated, ill need to make a new one, example: there are a few esp32 models that offer extra benefits like upto 8MB of external RAM)

(I would recommend going through these links, they are the spec sheets:

ESP SoCs | Espressif Systems

ESP Modules | Espressif Systems

STM32 High Performance Microcontrollers (MCUs) - STMicroelectronics

Teensy USB Development Board )

| **MCU** | **CPU** | **RAM** | **Flash Memory** | **CoreMark Score** |
|---|---|---|---|---|
| **ESP32** | Dual-core Xtensa® 32-bit LX6, up to 240 MHz | 520 KB SRAM | 448 KB ROM, external flash support | 504 1 core/ 996 2 cores |
| **ESP32-S3** | Dual-core Xtensa® 32-bit LX7, up to 240 MHz | 512 KB SRAM | Supports external flash memory | 613 1 core/ 1181 2 cores |
| **STM32F4** | Arm® Cortex®-M4, single-core, up to 180 MHz | Up to 192 KB SRAM | Up to 1 MB Flash | 608 |
| **STM32F765** | Arm® Cortex®-M7, single-core, up to 216 MHz | 512 KB SRAM | 512 KB Flash | 1082 |
| **STM32H750** | Arm® Cortex®-M7, single-core, up to 480 MHz | 1 MB SRAM (192 KB TCM RAM, 512 KB user SRAM, 4 KB backup SRAM) | 128 KB Flash | 2424 |
| **STM32H7B0** | Arm® Cortex®-M7, single-core, up to 280 MHz | 1.4 MB SRAM | 128 KB Flash | 1414 |
| **STM32H747** | Dual-core: Arm® Cortex®-M7 up to 480 MHz and Cortex®-M4 up to 240 MHz | 1 MB SRAM | 2 MB Flash | 3224 2 cores |
| **STM32H753** | Arm® Cortex®-M7, single-core, up to 480 MHz | 1 MB SRAM | 2 MB Flash | Not specified |
| **Teensy 4.0** | Arm® Cortex®-M7, single-core, 600 MHz -overclockable upto 912mhz | 1024 KB RAM (512 KB tightly coupled) | 2048 KB Flash (64 KB reserved for recovery & EEPROM emulation) | 2314 (no oc) |
| **Teensy 4.1** | Arm® Cortex®-M7, single-core, 600 MHz -overclockable upto 912mhz | 1024 KB RAM (512 KB tightly coupled) | 7936 KB Flash (64 KB reserved for recovery & EEPROM emulation) | 2314 (no oc) |

Lets put down a few points

1. Just like there are so many variants of stm32s, there are a lot of variants of esp32 and sometimes they are a lot more development friendly
2. Esp32 is primarily a wifi/Bluetooth/rf focused microcontroller. we don't even use those features of the esp32, and we could instead get a different microcontroller for the same price that actually has features we might use (need a bit more research here)
   -like stm32f7's L cache and D cache that would speed up processing even if the total available memory is lesser than esp32
   -all stm32s and teensy microcontrollers have TCM (tightly coupled memory) -so basically RAM is slow to access but it's deterministic, meaning the access time is the same every time. Chache is very quick to access but it's indeterministic. So you have TCM, which is located very close to the cpu core, hence giving faster access times than SRAM, while also providing consistent access time -this is very useful for time critical applications, probably something like high speed data converters (HDMI to display port or something). Read more about it here: ARM Cortex-R Series Programmer's Guide

3. All of that is fun and games, but like, when does all of this even matter? Pretty much all 32 bit microcontrollers are advanced enough and have enough processing headroom for every other operation of ours. We wont in a million tenures use something like TCM, or ART accelerator (stm32's npu) in our rover. (we might use ART accel for opencv tho)

4. But there are a few glaring issues, like the esp32's noisy adc, which we found out about because we were working with potentiometers and joysticks. There's two fixes for that, one would be to replace the microcontroller with one that has a better adc

   we've hit the esp32's processing limits twice (and I personally have seen only one) – when running microros code, and when running the robotic arm code with controller integrated. Apparently with the microros code, it gets so bad that the microcontroller fries itself (im still skeptical of that, there must be some sort of safety precautions against running too hot)

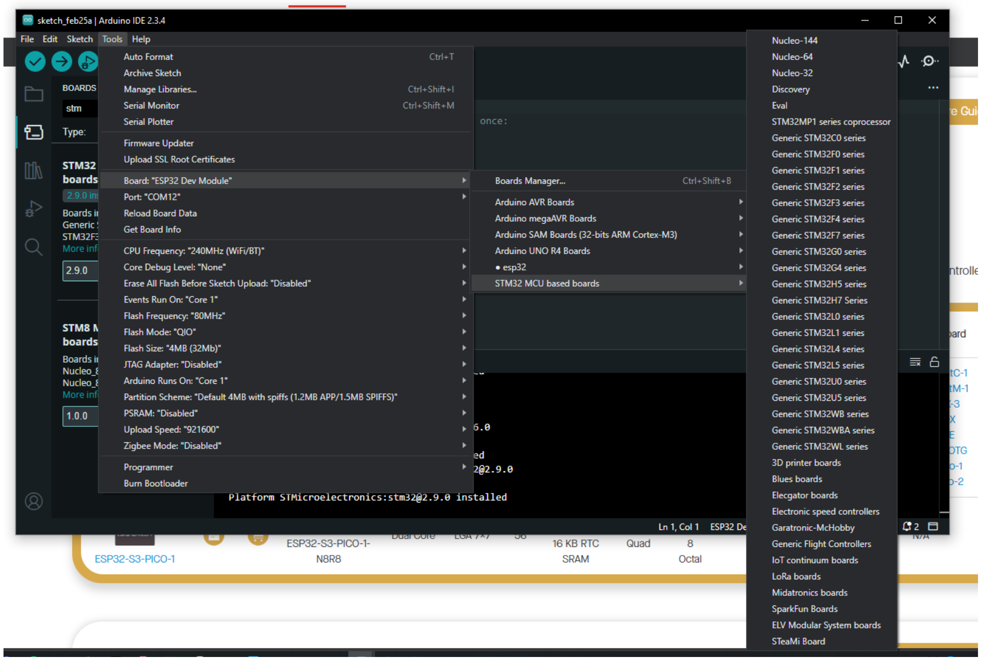
   all of these issues, while they could be fixed with changing the microcontroller -they could also very much be fixed with other changes.
   The controller wouldn't even push an Arduino nano to it's fullest limits -but then if it does, when I connect an lcd to it or something, we could always just use external ADCs for the few joystick inputs. If that's to costly, we can simply put one esp32 or stm32 for processing and a nano purely for reading joystick values. An atmega 328p costs barely anything

   The potentiometers for encoding -they are gonna replace it with magnetic encoders, and even if not magnetic encoders, we can use rotary encoders with custom gearboxes to get a pseudo high precision (multiple turns basically). Or if pots are the only way to go we could again, just use a nano for all the encoding and have it communicate with the esp32

   The robotic arm + controller code could use a LOT of optimizations, it is a 500 line code, yes, but there are also a lot of  optimizations we could do.

   The microros code -software tells they've been planning to switch away from microros, but maybe this is the only subsystem that requires a change

5. STM32s support the Arduino framework. ((1) #345 ESP32 vs STM32: Which one is better (Bluepill)? - YouTube). Teensy supports the Arduino framework. Esp32 supports the Arduino framework. Hell, 8051 probably supports Arduino framework. Everything supports Arduino framework, we don't need to fear bare metal coding. We don't even need to fear embedded C. Sure it is a very useful skill to have, but we are not done for if we switch to some other microcontroller and we don't know how to do the bare metal coding. (How to program and debug the STM32 using the Ardui... - STMicroelectronics Community)



```
Board: "ESP32 Dev Module"
Port: "COM12"
CPU Frequency: "240MHz (WiFi/BT)"
Core Debug Level: "None"
Erase All Flash Before Sketch Upload: "Disabled"
Events Run On: "Core 1"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
JTAG Adapter: "Disabled"
Arduino Runs On: "Core 1"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
PSRAM: "Disabled"
Upload Speed: "921600"
Zigbee Mode: "Disabled"
```

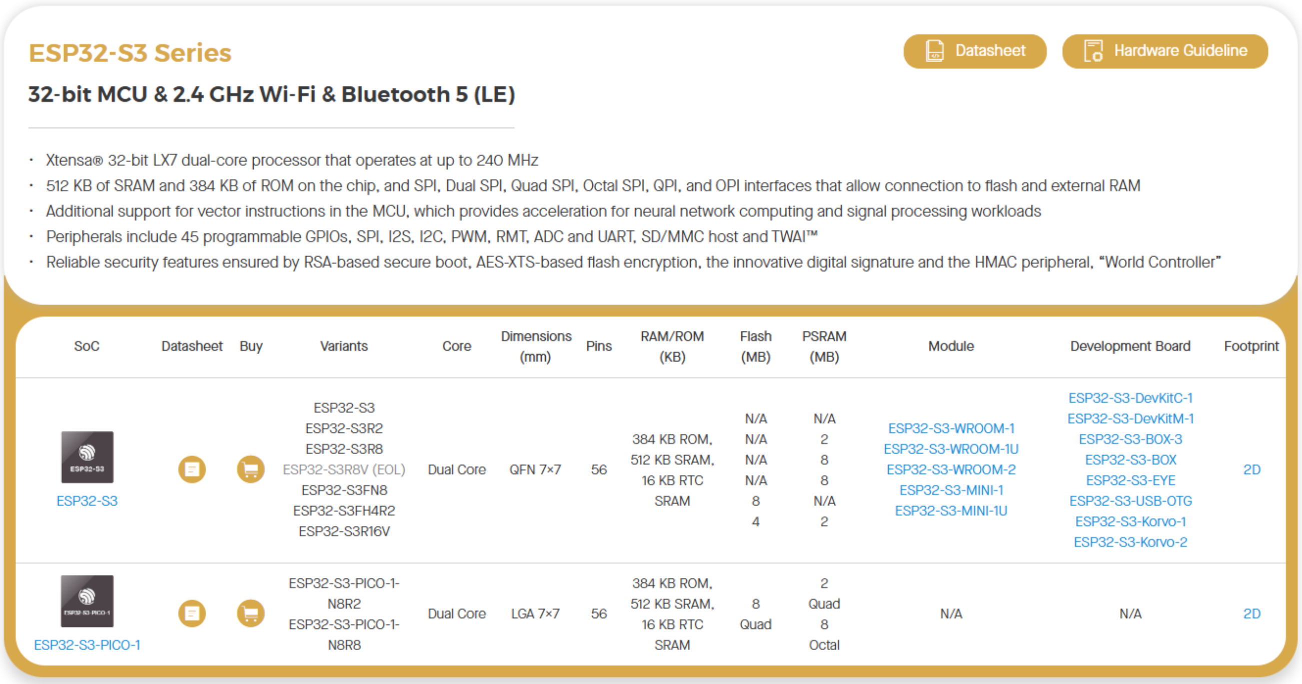
```
Platform STMicroelectronics:stm32@2.9.0 installed
```

(this is stm32 boards installed on my laptop)

The only thing we might need to fear is unsupported libraries. But most basic libraries are always supported. The exotic libraries, we could use other microcontrollers, or we could even write them ourselves because hear me out:

6. What are we trying to do here ? We are all in a team, and like, we're here to make this rover and go to the competition. But then we're also here to learn stuff. So rather than seeing something like making our own libraries as a waste of time, we could see it as a learning opportunity. Our rover is no longer in infancy. It's up and running, and we can assemble it and get it running for another IRC in less than a day.
So in a position like ours (I am only refering to the electrical domain, but this could apply to other domains very shortly), in a position like ours, maybe it's fine if I have to spend a week writing my own lora library for the stm32.

7. so overall. We don't have to use the same microcontroller. if software needs a lot of memory in their microcontroller, we could use an esp32-s3r8 -it has 8MB of PSRAM (external SRAM for esp32s).


Datasheet
Hardware Guideline

**ESP32-S3 Series**

**32-bit MCU & 2.4 GHz Wi-Fi & Bluetooth 5 (LE)**

- Xtensa® 32-bit LX7 dual-core processor that operates at up to 240 MHz
- 512 KB of SRAM and 384 KB of ROM on the chip, and SPI, Dual SPI, Quad SPI, Octal SPI, QPI, and OPI interfaces that allow connection to flash and external RAM
- Additional support for vector instructions in the MCU, which provides acceleration for neural network computing and signal processing workloads
- Peripherals include 45 programmable GPIOs, SPI, I2S, I2C, PWM, RMT, ADC and UART, SD/MMC host and TWAI™
- Reliable security features ensured by RSA-based secure boot, AES-XTS-based flash encryption, the innovative digital signature and the HMAC peripheral, "World Controller"

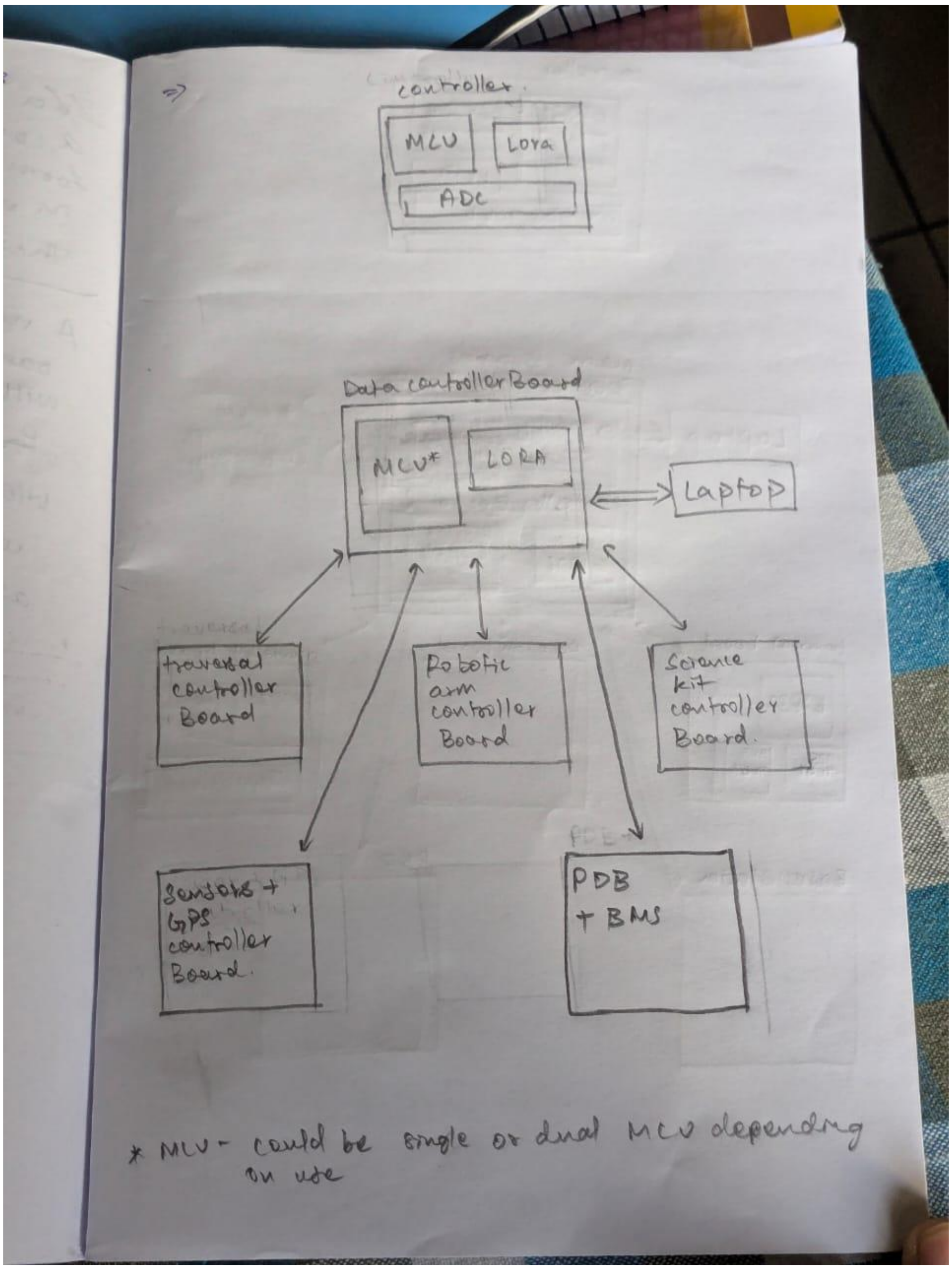
| SoC | Datasheet | Buy | Variants | Core | Dimensions (mm) | Pins | RAM/ROM (KB) | Flash (MB) | PSRAM (MB) | Module | Development Board | Footprint |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ESP32-S3 | | | ESP32-S3 ESP32-S3R2 ESP32-S3R8 ESP32-S3R8V (EOL) ESP32-S3FN8 ESP32-S3FH4R2 ESP32-S3R16V | Dual Core | QFN 7×7 | 56 | 384 KB ROM, 512 KB SRAM, 16 KB RTC SRAM | N/A N/A N/A N/A 8 4 | N/A 2 8 8 N/A 2 | ESP32-S3-WROOM-1 ESP32-S3-WROOM-1U ESP32-S3-WROOM-2 ESP32-S3-MINI-1 ESP32-S3-MINI-1U | ESP32-S3-DevKitC-1 ESP32-S3-DevKitM-1 ESP32-S3-BOX-3 ESP32-S3-BOX ESP32-S3-EYE ESP32-S3-USB-OTG ESP32-S3-Korvo-1 ESP32-S3-Korvo-2 | 2D |
| ESP32-S3-PICO-1 | | | ESP32-S3-PICO-1-N8R2 ESP32-S3-PICO-1-N8R8 | Dual Core | LGA 7×7 | 56 | 384 KB ROM, 512 KB SRAM, 16 KB RTC SRAM | 8 Quad | 2 Quad 8 Octal | N/A | N/A | 2D |

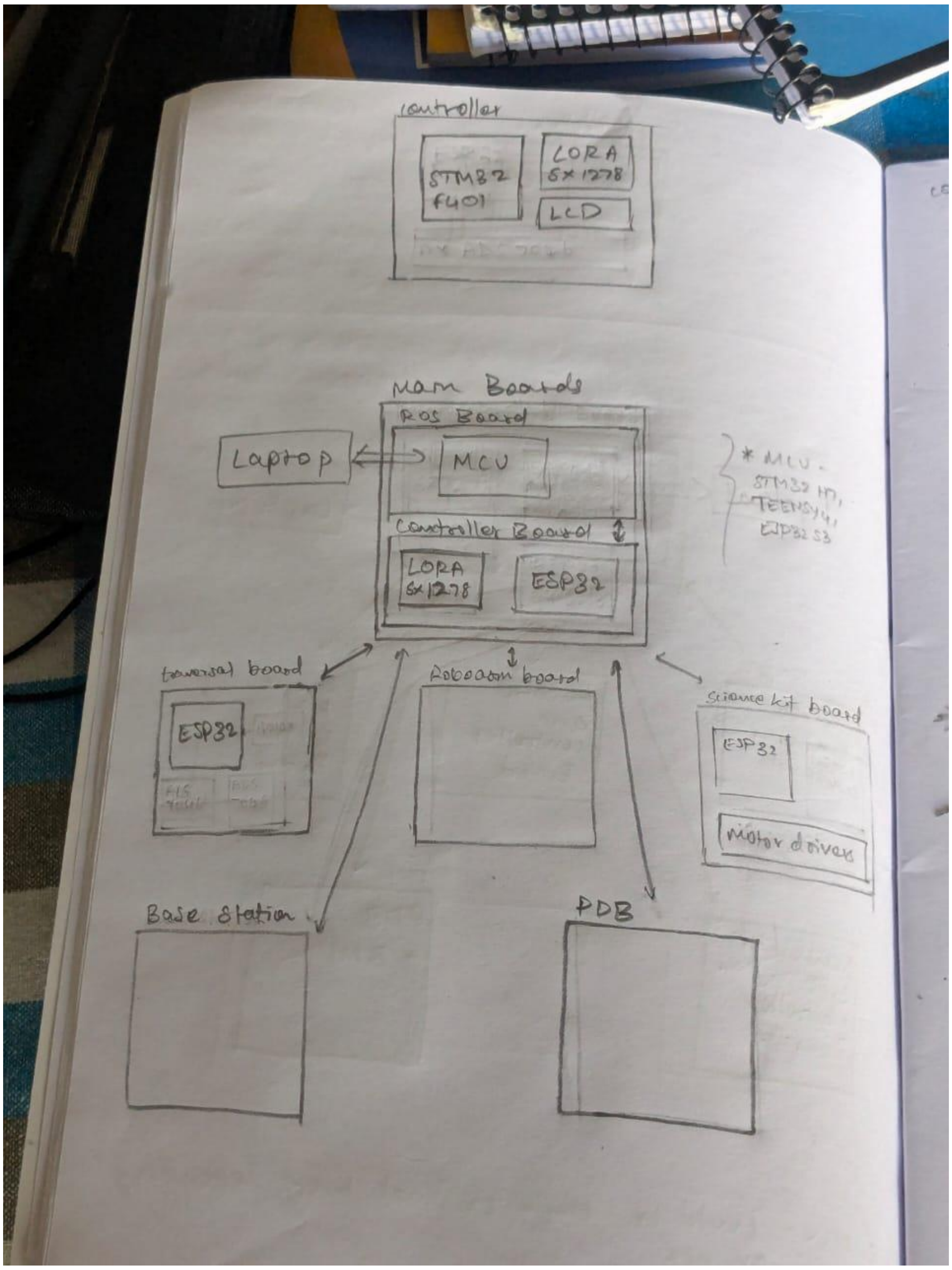
if robotic arm needs high processing power and a lot of gpio with stable adc, they can use an stm32h7. It reaches coremark scores upto 3000 on a single core (you get only about 1000 points with esp32 using both of it's cores)

if we need library support for sensors, we could use an Arduino nano, library support for lora -esp32

8. since the arduino framework is supported on pretty much every microcontroller now, we can develop the same code and know we can replace the microcontroller with minimal changes to the code. And I insist we move to an embedded board design (instead of using dev boards and headers and header pins). There are several good reasons. But isn't the fact that we get an opportunity to learn how to do these enough?

9. I also propose this new microcontroller network for our rover. it will be easier to work on different modules at the same time. you can develop the receiver board, the ros board, the traversal board separately and parallely. It's easier to replace parts of it instead of having to keep track of codes and have to reupload it every time. it's easier to make mini bots for smaller competitions (which we should hit to keep ourselves busy)


controller
MCU
Lora
ADC
Data controller Board
MCU*
LORA
Laptop
traversal controller Board
Robotic arm controller Board
Science kit controller Board
Sensors + GPS controller Board
PDB + BMS
* MCU - could be single or dual MCU depending on use


controller
STM32 F401
LORA SX1278
LCD
Main Boards
ROS Board
Laptop
MCU
Controller Board
LORA SX1278
ESP32
* MCU - STM32 H7, TEENSY4.1, ESP32 S3
traversal board
ESP32
Robo arm board
science kit board
ESP32
motor driver
Base station
PDB