EE 333 Project 2 Final Report

Austin Beinder and Jonah Frosch

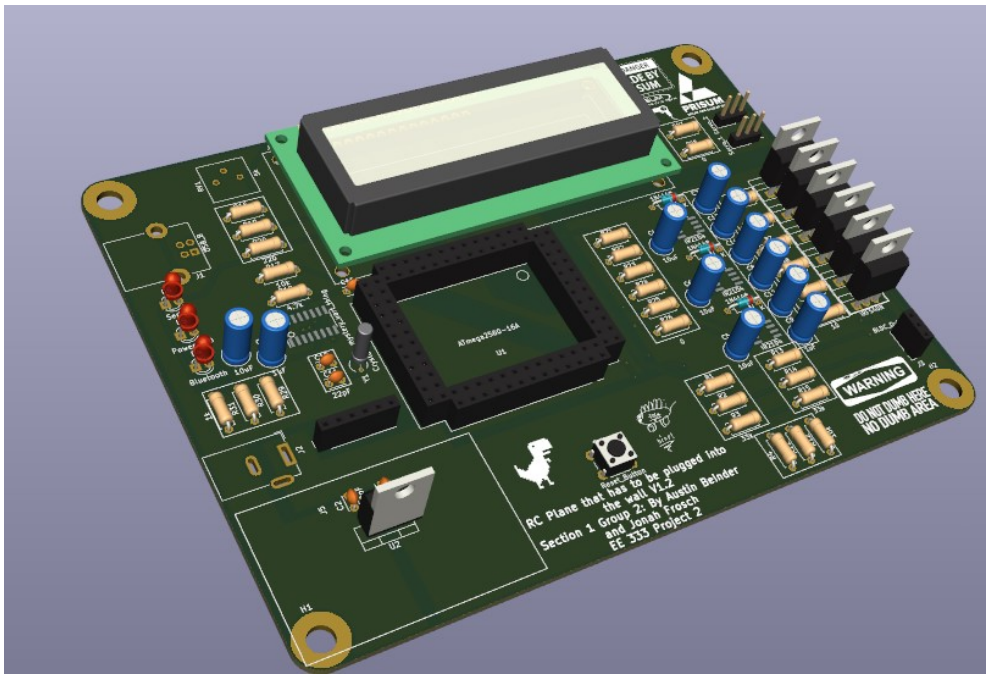

Figure 1: 3D View

Overview

The goal of this project is to create a PCB using an implementation of an Arduino or Atmega MCU which has a set of motor controllers that will be controlled remotely via a Bluetooth connection. The project will control a single sensorless BLDC motor's speed, and the rotations of two servo motors. It will also have the capability of flashing the Atmega chip via a serial port.

The Bluetooth connection is controlled by a Python app on a PC with a simple interface for controlling speed and rotation through a Logitech gamepad.

Initial Plan

The original plan for the project was largely the same, but we also wanted to measure the BLDC speed and display it on the LCD. We also wanted to make a GUI on the PC to display more information. However, in hindsight, this is simply too much data for the bluetooth to reliably transfer. As it currently stands, we are really only able to reliably transfer about 16 bits worth of data.

Breadboard

The breadboard version of the project was rather chaotic looking. But it did indeed demonstrate the idea. It used an Atmega 2560 connected through a CH340 USB to serial converter chip so that we could flash code directly to our project without the need for a UNO board middle man. The 2560 is then connected to a BLDC, two servos, bluetooth, and LCD and was able to get each piece to function in an overall test script. This software would later be optimized once we received the PCB. Using this setup along with a scale, we were able to determine our motor was capable of outputting 140 grams of thrust which is rather impressive in our opinion. Before optimizing our software, we could adjust the motor speed with buttons on the breadboard, and we could cause the servos to go from non rotating to sporadically rotating with the gamepad. We were also able to print some of the bluetooth messages on the LCD.

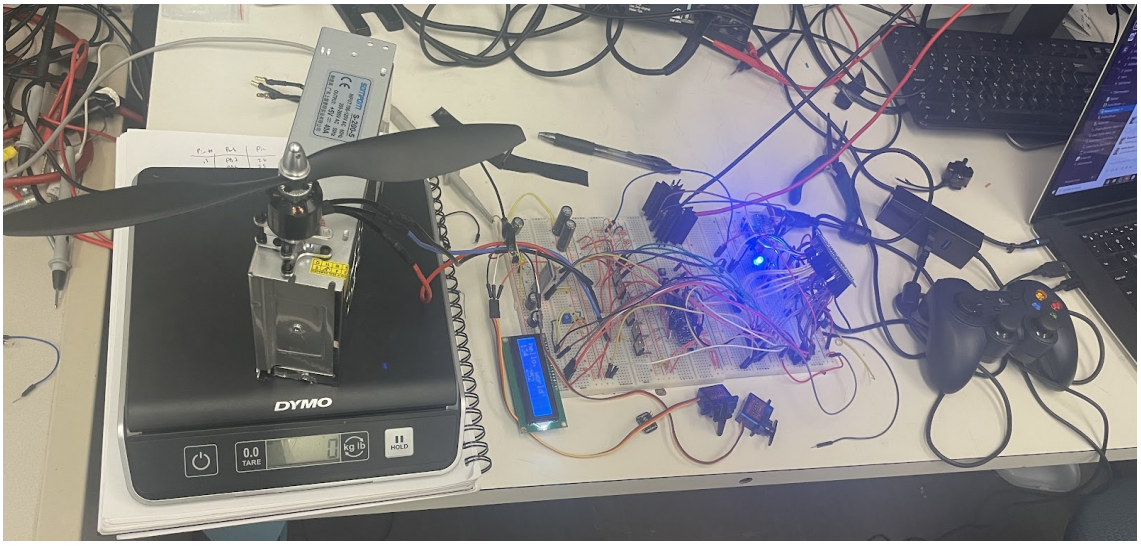

Figure 2: Breadboard project

Software Design

The software was split between the embedded (client) and python (server) side. We will discuss them briefly below.

Client

The embedded software works to interface with the various hardware components. The BLDC works by detecting the back emf of the motor generated by the rotating magnets within the coil. The microcontroller configures itself so that the positive input to the comparator is tied to the virtual neutral point between the motor lines, and the negative input is tied to one of the motor lines. The controller will switch which motor line the negative input is tied to as the motor spins. Whenever the comparator triggers, it triggers an interrupt. This ISR triggers a function that advances the motor position by driving the phases of the H bridge to the next location in the cycle. After this is when either the comparator is set to trigger on the rising or falling edge of the next comparator cycle, or to another motor line. The H bridge is driven by 6 PWM lines that turn

on and off the fet drivers which in turn turn on and off the fets. The startup sequence of the motor essentially just disables interrupts, and automatically advances position based on a delay that decreases in length over a short period of time. This runs prior to entering the main while (true) loop of the application.

The Bluetooth works by writing to or reading from the characteristic of the AT09 breakout board. The characteristic is essentially a 20 byte register that can be written to from either the master or the slave. However, we have had some issues whenever we try to reliably retrieve more than two characters from the characteristic and operating the motor. Also, sometimes the bluetooth will output random garbage values that we need to filter out, which means if those values are ever reached for real the system will seem like it is not responsive. We think that this is due to the interrupt from the comparator interrupting the serial transaction with the board. This could have been solved in one of two ways. Either we could have tried to optimize the software better to find a more reliable way to read from that serial port, like maybe another serial port had a larger hardware buffer so that this wouldn't have been an issue. The other option would have been to use two Atmega 328 chips. One could have been used for serial communication, bluetooth communication, and servo control, while the other could have been used to control the motor, and the communication between them could have just been a PWM line like on an actual ESC.

Due to the issue of having to only use two characters reliably, we encoded all the interactions we needed within 16 bits. We used two bits for increasing or decreasing motor speed, and 6 bits for each servo position. We had two blank bits that were at the beginning of each byte so that we could write them as characters in python which are only 7 bits. The bluetooth is configured to only be retrieved every so many iterations of the main while loop, and while it is currently configured to be retrieved every cycle, we have sometimes found it advantageous to decrease its frequency of retrieval.

Similar to how the bluetooth is only retrieved every so many cycles, the LCD is only updated every certain number of cycles. This allows the microcontroller to have as much possible time to spend on the BLDC. Currently the LCD is only setup to printout the decimal value of the first byte in the characteristic of the bluetooth along with "hello world!".

The servos are implemented using a hardware controlled PWM timer operating at 50 Hz that is from a servo library for the 2560 we found online. When trying to use the traditional arduino servo library, we discovered the issue that it changes its duty cycle whenever the CPU is busy working on something. For this reason, a hardware based PWM servo needed to be used. However the motor was already using timers 1 and 2 for triggering the comparator, and so that is why we needed to change the pins the servos were attached to in the final changes to the project, and we opted to use pins 44 and 45 of the arduino or 39 and 40 of the mega.

Server

The server side of the project works by using the python packages PyGame and Bleak. PyGame interfaces with a Logitech gamepad. It reads events from the controller, and depending on the axis that the event happens on we can know if it is the left or right joysticks, or the back triggers. Besides the axis, each event also has a value corresponding with the rotation of the

joystick or trigger. We encode values to the servos or motor based upon these values. Bleak interfaces with the bluetooth by using asynchronous functions that write to or read the characteristic. We currently have it setup to have one servo controlled by the up/down of the left joystick, and the other servo by the right joystick, and the motor speed controlled by the back triggers. Hold the triggers down and the speed will increase or decrease depending on the side.

PCB Design

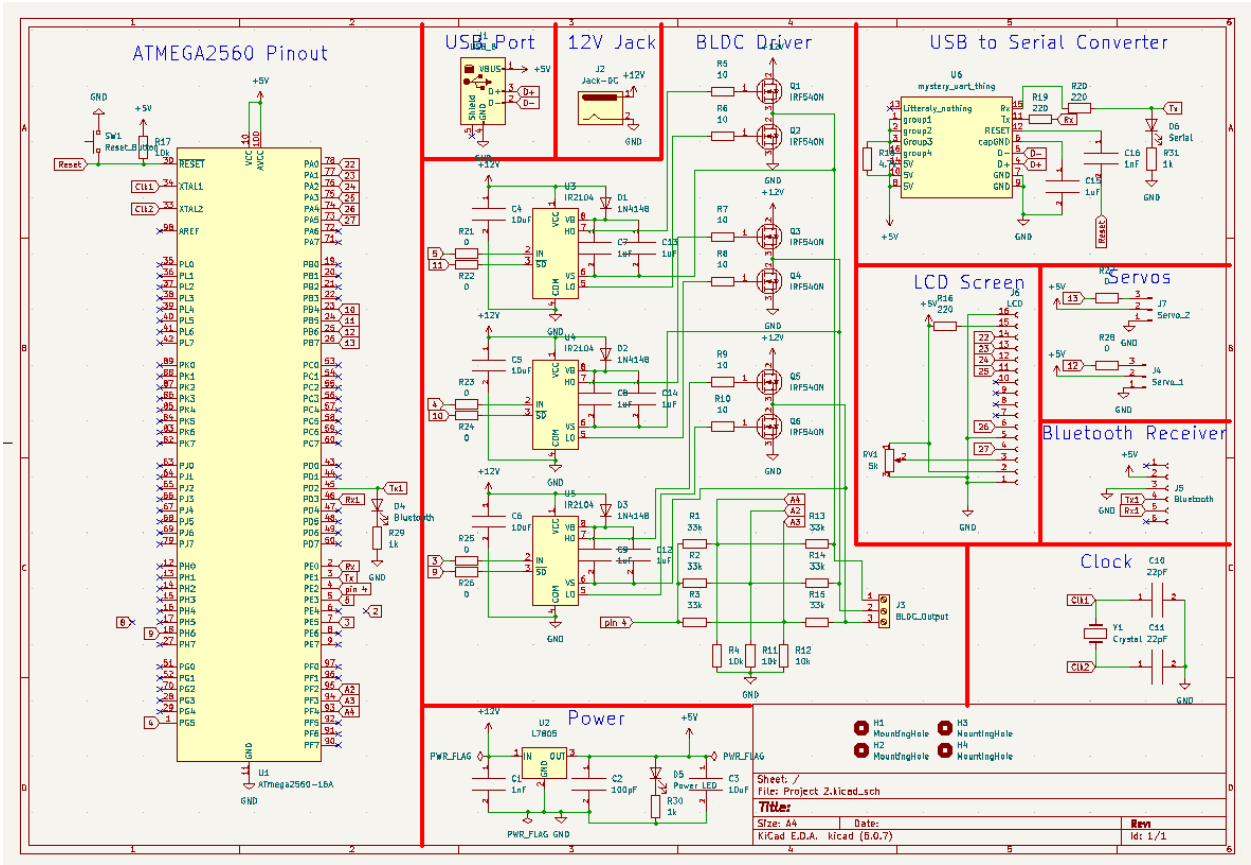

Figure 3: Schematic

Our overall PCB design process was fairly straightforward and similar to how we made our project 1 PCB, save for a few exceptions. We built each subsystem in our schematic, and used tags to connect each subsystem together so that the schematic was easier to look at and overall be more readable. When we went to build the PCB, though, we tried a different approach from project 1, where this time around we tried to place the components of each subsystem in groups on the board. The thought with this was that it would ensure that traces wouldn't have to go all over the board to connect two components, but instead connect close components with a few select traces connecting subsystems running the length of the board. This method ended up working out great, with the only major issue being that initially this process led to some wasted space, so we ended up running 3 iterations of the board, tweaking the placement of components and rotating the 2560 so that traces would not cross over each other as much.

While I didn't save any screenshots of the initial designs, the final one we came up with is pasted below:

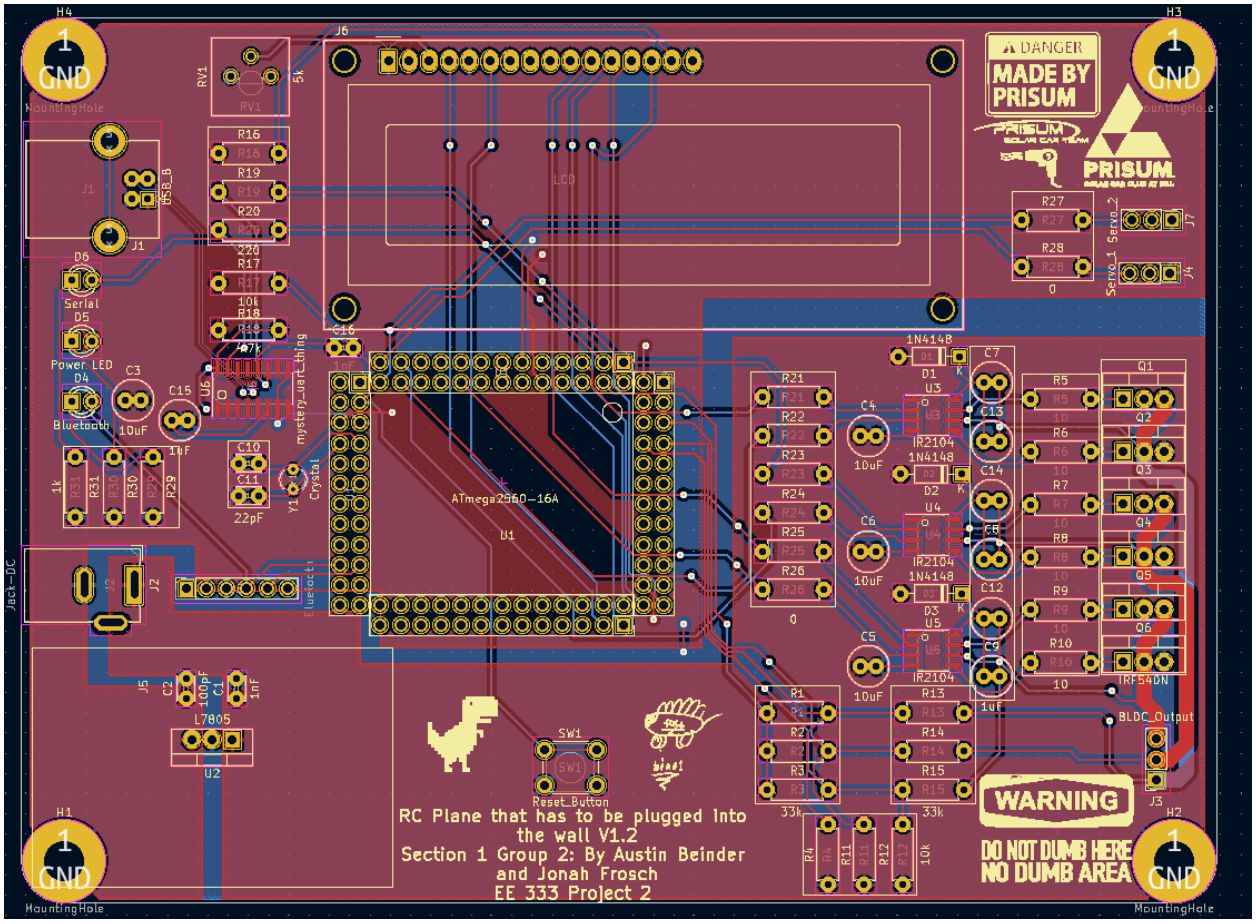

Figure 4: Final PCB Layout

In addition to the placement, We also went for power and ground planes on our board, as it is expected to push up to 2 amps and simple traces might not be enough. Where we could not use a plane to carry our power, we calculated the required trace width and used that, which you can see on the far right of our board.

Lastly, you can notice a strange 100 pin footprint in the center of the PCB. That is a custom Breakout board we made for the ATMEGA2560, since it is an SMD microcontroller. Being that this project is extremely experimental, we wanted to plan ahead in the event that the IC breaks, so that we wouldn't have to deal with the hassle of desoldering a 100 pin SMD component. Instead we found a 100 pin breakout board, and made a footprint to go along with it, so that we could hotswap 2560s if needed. Both the breakout board and footprint are seen below.

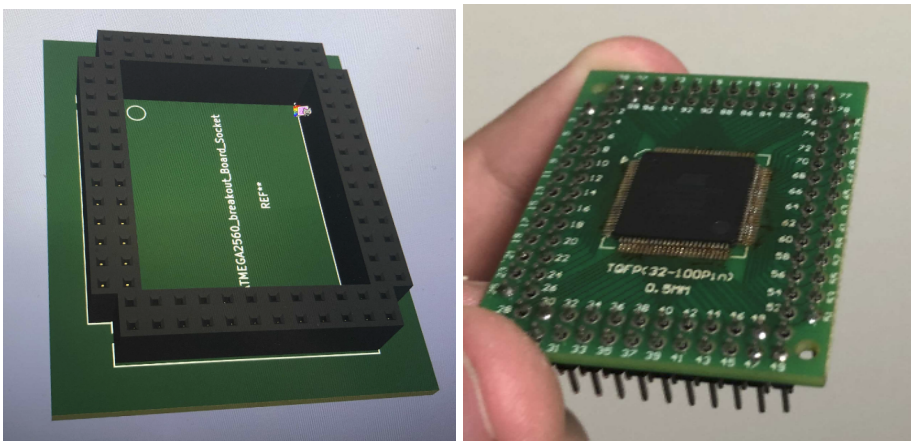

Figure 5: Microcontroller footprint and breakout Board

Obstacles

The largest obstacles of this project were probably both the software portions of the project, in conjunction with getting the serial communication to work correctly. We also tried and failed to get our own version of a PWM breakout board to work.

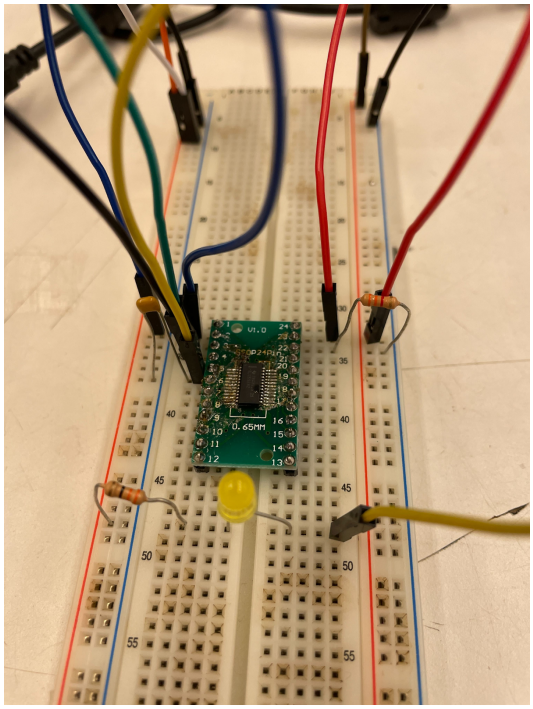

Figure 6: PWM Breakout Board

Final Outcome

The final outcome of the project came out rather well. The board for one looks awesome, furthermore it works essentially as intended. We have honestly run out of time, but if we had more time we could definitely increase the bandwidth of communication with the bluetooth, and

do fancier stuff like determining the RPM of the motor and either printing that to the LCD, or feeding it back to the user PC. We could also do a rev 2 of the board and make it better. However as it stands, I think that this project is nearing 150 engineering hours of effort going into it, and it is the end of the semester. Other than these possible improvements, it works as intended which is awesome.

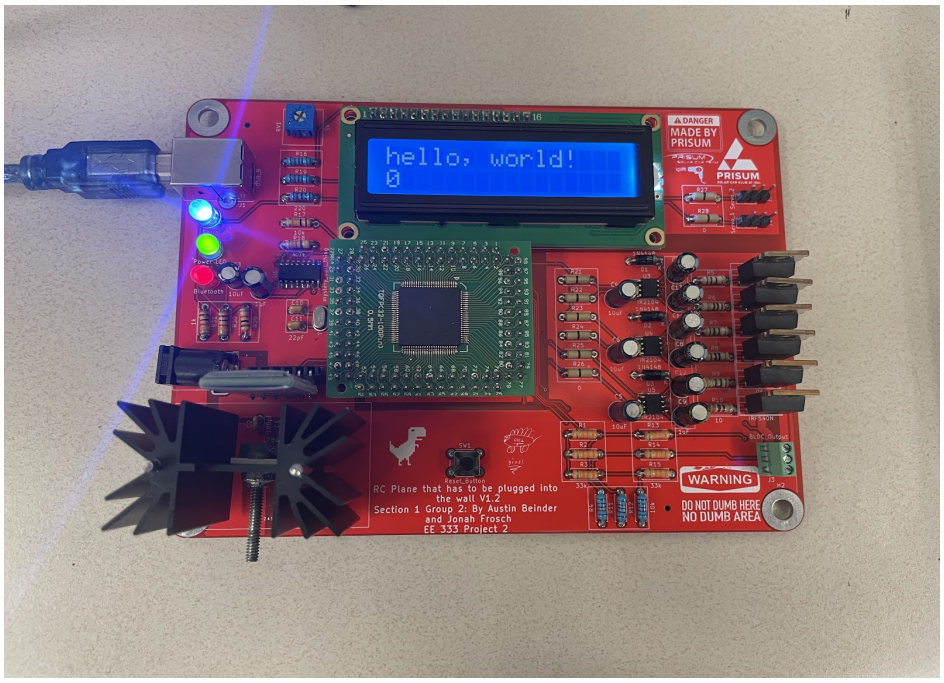

Figure 7: Final PCB

Conclusion

In summary, we *successfully* created a BLDC and Servo controller capable of bluetooth and serial communication and using an LCD screen.

References

LCD Display

https://docs.arduino.cc/learn/electronics/lcd-displays

BLDC

https://simple-circuit.com/arduino-sensorless-bldc-motor-controller-esc/

https://www.amazon.com/dp/B01MAUFDHW?psc=1&ref=ppx_yo2ov_dt_b_product_details

Servo

https://docs.arduino.cc/learn/electronics/servo-motors

https://www.amazon.com/Sipytoph-Helicopter-Airplane-Walking-Control/dp/B09185SC1W/ref=sr_1_8?crid=GRIXSPPNSYFY&keywords=sg90%2Bservo&qid=1665756797&qu=eyJxc2MiOiIzLjU3IiwicXNhIjoiMy41NCIsInFzcCI6IjMuNTEifQ%3D%3D&s=toys-and-games&sprefix=sg90%2Bservo%2Ctoys-and-games%2C104&sr=1-8&th=1

Slave Side Bluetooth

https://medium.com/@yostane/using-the-at-09-ble-module-with-the-arduino-3bc7d5cb0ac2

https://www.amazon.com/dp/B00WGPKZ8Y?psc=1&ref=ppx_yo2ov_dt_b_product_details

Master Side Bluetooth

https://docs.pycom.io/tutorials/networks/ble/

PWM Breakout

https://howtomechatronics.com/tutorials/arduino/how-to-extend-arduino-pwm-outputs-tlc5940-tutorial/

Gamepad

https://www.amazon.com/dp/B003VAHYQY?psc=1&ref=ppx_yo2ov_dt_b_product_details

https://github.com/denilsonsa/pygame-joystick-test

Atmega 328
https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Atmega 2560
https://ww1.microchip.com/downloads/en/devicedoc/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf

Appendix

BOM

Item	Qty	Reference(s)	Value
1	2	C1, C16	1nF
2	1	C2	100pF
3	4	C3, C4, C5, C6	10uF
4	7	C7, C8, C9, C12, C13, C14, C15	1uF
5	2	C10, C11	22pF
6	3	D1, D2, D3	1N4148
7	1	D4	Bluetooth
8	1	D5	Power LED
9	1	D6	Serial
10	4	H1, H2, H3, H4	MountingHole
11	1	J1	USB_B
12	1	J2	Jack-DC
13	1	J3	BLDC_Output
14	1	J4	Servo_1
15	1	J5	Bluetooth
16	1	J6	LCD
17	1	J7	Servo_2
18	6	Q1, Q2, Q3, Q4, Q5, Q6	IRF540N
19	6	R1, R2, R3, R13, R14, R15	33k
20	4	R4, R11, R12, R17	10k
21	6	R5, R6, R7, R8, R9, R10	10
22	3	R16, R19, R20	220
23	1	R18	4.7k
24	8	R21, R22, R23, R24, R25, R26, R27, R28	0
25	3	R29, R30, R31	1k
26	1	RV1	5k
27	1	SW1	Reset_Button
28	1	U1	ATmega2560-16A
29	1	U2	L7805
30	3	U3, U4, U5	IR2104
31	1	U6	mystery_uart_thing
32	1	Y1	Crystal

Figure 7: BOM