<https://github.com/joesphan/FFB-Steering-Wheel>

Design Goal:

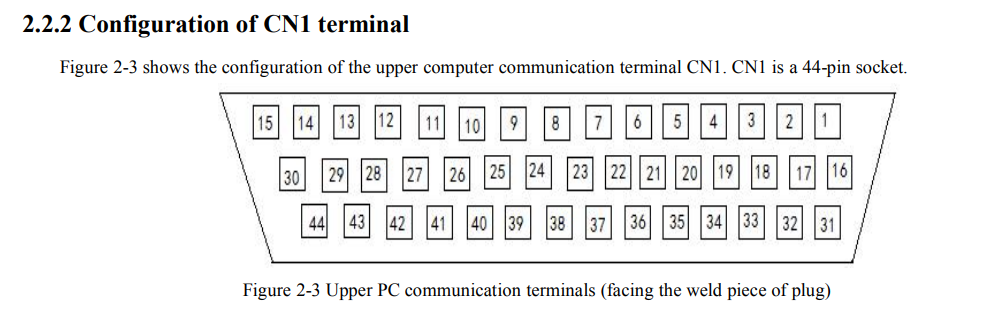
Inspired by the Marlin firmware for 3d printers, the goal of this project is to create something that everyone can use, using a familiar Arduino interface, and create compatibility across a range of parts

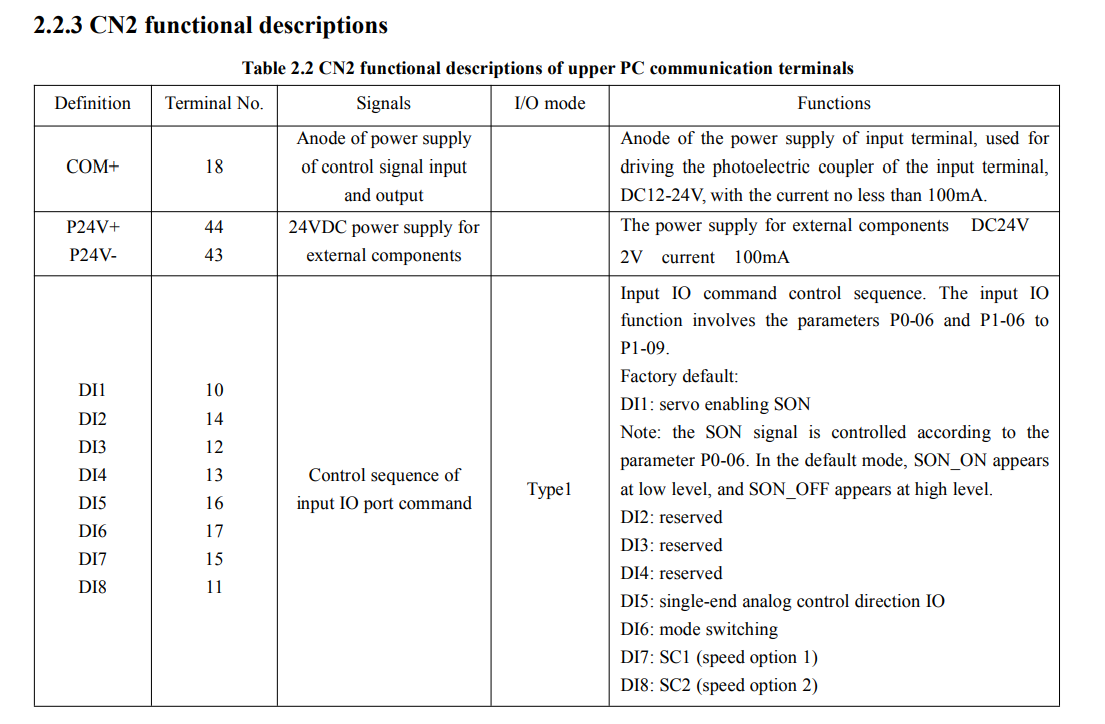
Main features:

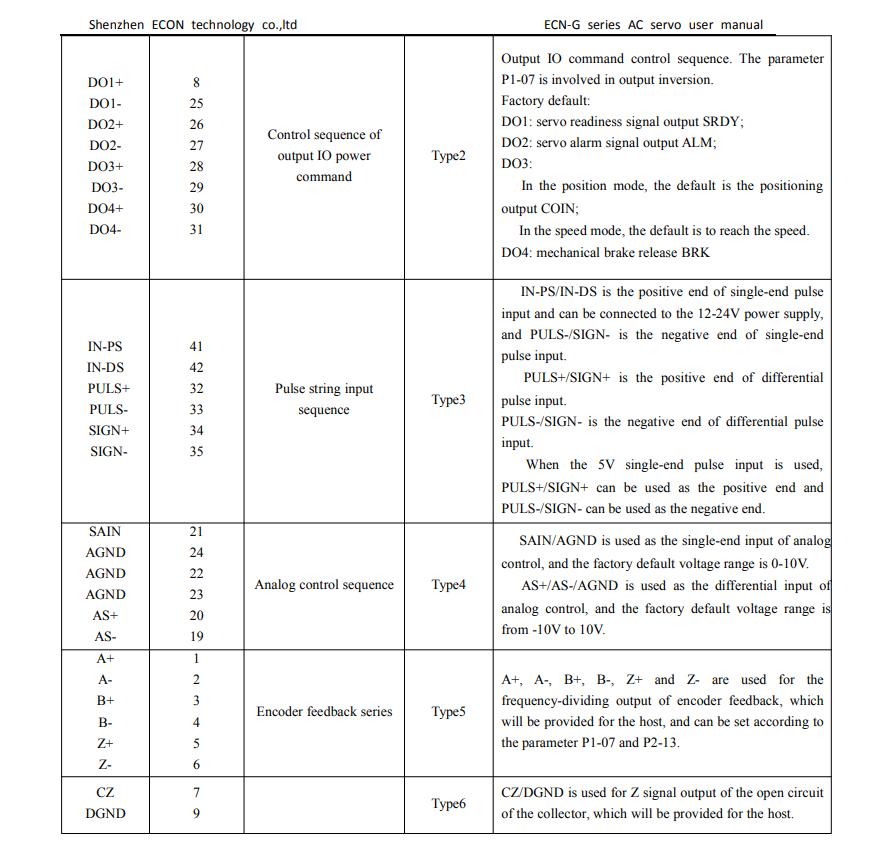
* Open source
  + Unlike MMOS, the source code can be viewed by everyone
  + Changes and improvements can be made/suggested by opening an issue on github
* Universal CNC servo drive control interface
  + The sim wheel controller is capable of driving servo drives with a +/- 10V signal
* Wireless peripherals
  + Pedals, yoke, shifters can all easily be added using one of the four auxiliary DB9 ports, or through the wireless ESP8266 UDP interface
  + The ESP8266 UDP interface has a latency of 20ms
* Easy configuration
  + All the parameters are set via the #defines tab

Basic Connection:

Pg 11 of the servo drive datasheet:







The naming conventions used in the code follows:

|  |  |  |
| --- | --- | --- |
| Name | Pin | Description |
| DI1 | 10 | -10V to EN motor, in reference to COM+ |
| COM+ | 18 | See above |
| A+ | 1 | Encoder phase A, ref to DGND |
| B+ | 3 | Encoder phase B, ref to DGND |
| DGND | 9 | Digital ground |
| AS+ | 20 | Differential + of control, +/-10V |
| AS- | 19 | Differential - of control, +/-10V |
| AGND | 22 | Differential ground reference |
|  |  |  |

Firmware Setup:

Everything is contained under the defines tab

1. //Safetys
2. #define MAX\_RPM 1
3. #define MOT\_EN\_PIN 7
5. //encoder parameters
6. #define MAX\_STEPS 15000                         //maximum number of steps one way
7. #define MAX\_FULL\_STEPS (MAX\_STEPS \* 2)        //max number of steps total
9. //encoder HW
10. #define ENCODER1\_PIN 3          //must support external interrupt
11. #define ENCODER2\_PIN 2          // ""    ""       ""       ""
13. //encoder RPM detect
14. #define ENCODER\_1\_REV 10000
15. #define RPM\_DIVIDEND (60000000 / ENCODER\_1\_REV)
17. //motor control HW
18. #define ADC\_PIN 5
20. //endstop damping settings: https://www.desmos.com/calculator/rhyetd6p3d
21. #define ENDSTOP\_FACTOR 2.5                  //dampening factor when turned past full lock
22. #define MAX\_OVERTRAVEL 100        //how much overtravel steps is allowed before full force is applied
23. #define BIT\_DEPTH 2048
25. //endstop rebound settings:
26. #define REBOUND\_FORCE 50            //rebound force proportion to speed
28. //endstop spring length
29. #define OVERTRAVEL\_SLOPE (BIT\_DEPTH / MAX\_OVERTRAVEL)   //internal calcs

[formatter](http://www.planetb.ca/syntax-highlight-word)

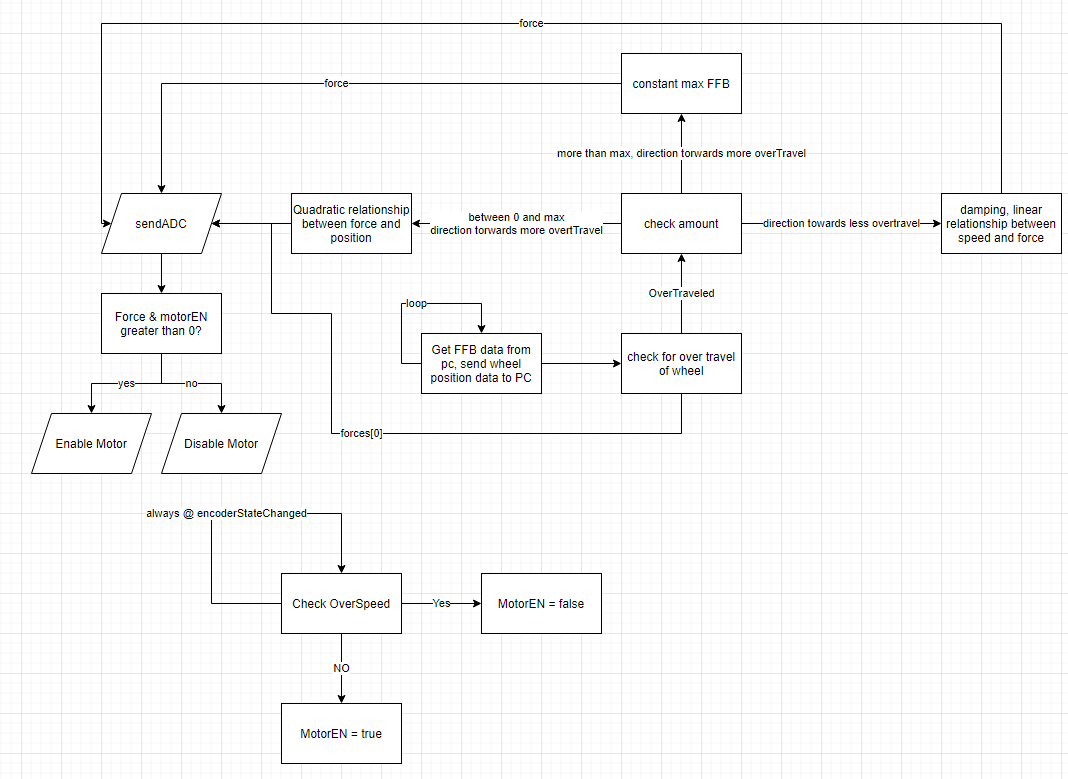
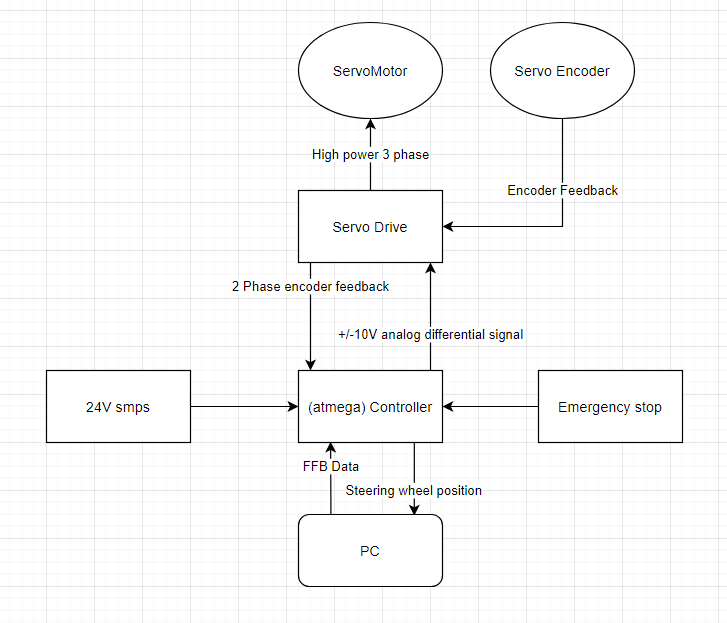
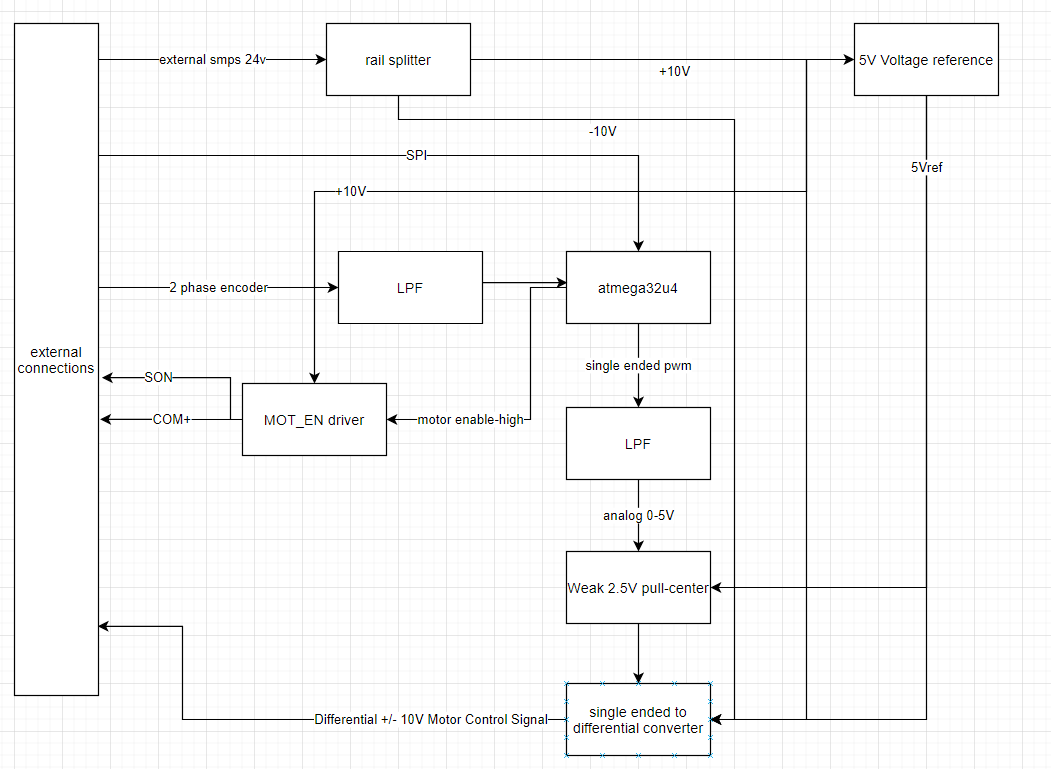
Some of the settings that require more thought are the rebound and dampening settings.

For endstop dampening, one can refer to [this desmos graph](https://www.desmos.com/calculator/v05nhapfn8) when the wheel is moving towards more overtravel.

Endstop rebound dampening of is just a linear relationship between speed towards wheel center and REBOUND\_FORCE

A library is used for communication with the PC. Be sure to use [my fork](https://github.com/joesphan/ArduinoJoystickWithFFBLibrary), as it offers 12 bit (adjustable) FFB resolution, rather than fixed 8 bit.

Theory:

* PID Loop:
  + 
* Hardware Diagram:
  + 
* Controller internal diagram:
  + 

Wired Peripherals:

* All the remaining pins on the Arduino micro have been broken out to the DB9
* The pinouts can be found under fullschematic.dch/pdf

Wireless Peripherals:

This method requires building a ESP8266 DB9 wireless adapter and plugging into DSUB-1. The ESP Wi-Fi receiver will communicate to the ATMega32u4 via UART. A custom packet is used to communicate information between peripheral and controller, through Wi-Fi UDP. The UDP packet is as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Function | HEADER | Byte 1 | | | | | | | | Byte 2 | | | | | | | | Byte 3 | Byte 4 | Byte 5 | Byte 6 | Byte n | |
| Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 | Bit7 | Bit6 | Bit5 | Bit4 | Bit3 | Bit2 | Bit1 | Bit0 |  |
| Button input 1 | ‘A’ |  |  |  |  |  |  | Left shift paddle  (Button0) | Right shift paddle  (Button 1) |  | | | | | | | |  |  |  |  |  |
| Axis input 1 | ‘B’ | Throttle, high 8 bits of the 16 | | | | | | | | Throttle, low 8 bits of the 16 | | | | | | | | Brake, high 8 bits of the 16 | Brake, low 8 bits of the 16 | Clutch, high 8 bits of the 16 | Clutch, low 8 bits of the 16 |  | |
| Button input 2 | ‘C’ | Reverse gear  (Button2) | Gear 1  (Button3) | Gear 2  (Button4) | Gear 3  (Button5) | Gear 4  (Button6) | Gear 5  (Button7) | Gear 6  (Button8) | Gear 7  (Button9) | Gear 8  (Button10) | Gear 9  (Button11) | Gear 10  (Button12) | Shifter console button 1  (Button13) | Shifter console button 2  (Button14) | Shifter console button 3  (Button15) | Shifter console button 4  (Button16) | Shifter console button 5  (Button17) |  |  |  |  |  | |
| Axis input 2 | ‘D’ | Handbrake, high 8 bits of the 16 | | | | | | | | Handbrake, low 8 bits of the 16 | | | | | | | |  |  |  |  |  | |

Coding conventions:

0x01 is recognized as HIGH for buttons, therefore it is active HIGH.

0x00 is recognized as LOW for buttons

The peripheral will automatically connect to the host Wi-Fi access point and send the packet to the host ESP8266 via UDP.

AP: SimInput

Key: 12345678

IP to send to: 192.168.4.1

Port to send to: 4210

**Official Peripheral Documentation**

Please open an issue and provide documentation, and Github page link to get the peripheral officially listed.

Shifter Paddles

Github: <https://github.com/joesphan/Wireless-shifter-pedals>

Header used: ‘A’