# Horizontal Bearing rig

This documentation will attempt to provide a complete overview of information relevant to the Horizontal bearing setup and resources to access extra information. Blue text indicates less critical information, which is hopefully unnecessary.

Check other documentation if certain information is lacking ('How to' Guide - Test Motor, LabView & PLECS and PLECS Code and TI Hardware Documentation). However, the project has changed a reasonable amount since these were written.

## Operation

Setup:

Check the booster and Launchpad boards are connected.

Check the J5 jumper on the booster board is disconnected.

A hand holding a green circuit board

Description automatically generated

(J5 Jumper)

Check the connection of the computer to TP-Link and of the Link to NI chassis and Temperature Controller.

Connect the booster board to the motor.

Connect the computer and Launchpad and build the PLECS code onto the Launchpad.

Activate external mode and auto triggering.

Open the LabView code and check the TCP/IP connection socket is correct along with the chassis connections.

Check appropriate temperatures are reached.

Set the servo control to release the shaft.

Collecting data:

Run the LabView code. (and potentially the other LabVIEW code and the camera)

Ensure the PLECS speed is set to 0, and the rate limiter is set to a low increase (~1).

Connect the power supply to the booster board and turn it on.

A close-up of a circuit board

Description automatically generated

(Should look something like this at this stage)

Activate the relay in LabView.

Set the motor constant and voltage supply in simulation preferences/initialization.

The modulation amplitude should be controlled automatically, to regain manual control simple switch the nearby wire to the constant modulation amplitude block.

Start the motor with a low speed(~1Hz) then increase rate limiter (~20).

Increase speed. Critical speed should be in the ballpark of 50Hz. Try and move through the critical speed quickly.

Finishing collection:

Decrease the speed.

Reduce the rate limiter (only in the positive direction so it is low the next time the motor is started).

## Save the LabView file.

Disconnect relevant parts (probably turn off the power supply).

Set the servo to hold the shaft again.

Testing and debugging may be done by connecting the power supply, a small motor, the computer and the boards, then running the PLECS code with external mode. Also running PLECS simulations.

**DO NOT:**

Connect the J5 jumper while power supply is on and USB is connected.

Adjust the voltage or current provided to the booster board quickly (jumps of 20V are too fast).

Reverse the positive and negative terminals.

Allow a connected motor to start and stop or change speed rapidly (dependent on the motor). This can include running motor test code.

Reduce the carrier frequency to that used with commercial motors (10k) without understanding what you're doing. Low inductance motors can require much higher carrier frequency.

Run the board if any part of it has become too hot to touch (it is quite likely broken).

Leave the motor running at too low speeds or with too low a modulation amplitude that it does not spin.

I’m sure you can guess where this advice comes from.

## PLECS code

It is necessary to set the TSP, cgt and Uniflash in PLECS preferences. The TSP and aux software can be downloaded from C2000 ware.

It may be necessary to explicitly open the correct PLECS version before opening the files if multiple versions of PLECS are installed.

Rate limiting throughout the code is quite important to specify the accelerations of the motor.

To build the code, open the PLECS program, configure preferences, press ctrl+alt+b then press build with the USB connected.

External mode can be enabled in the target tab before building. Select serial then [Rx, Tx] = 43, 42 for the current Launchpad. Buffer size is how many points will be saved, if it is too large an error should appear at compile time. Then connecting to the board by USB, building, go to external mode, scan, select the right port, connect and activate auto-triggering to get feedback from the board (allowing use of scopes and displays).

Increasing the decimation to 10 cuts 9 out of 10 data points so allows for data to be gathered over a longer time frame.

The PWMs are activated by sending a 0 signal to GPIO 26.

The PWM protection block disables the PWMs by watching for certain things. There are a lot of negatives so follow carefully.

If the input En has NOT had a rising edge since start then it will DISABLE the PWMs.

The output of the block can indicate whether the PWM protection block has disable the PWMs or not (adjustable high or low).

If it detects a TZ signal (in my code it is set to TZ1 GPIO14 which is the Over temperature sensor) it will disable the PWMS. To restart the En signal must return to 0 then rise back up (this can be adjusted).

The modulation amplitude is given by the fraction of desired output voltage over input voltage.

See “Custom\_Motor\_Stats” for some measured data of the motor.

The main PLECS code should use Sensorless driving.

Sensorless

This changes often and I will not be able to completely document this. If it works:

It should run by activating a “button”. Changing this value will begin the motor.

It will run an offset calibration (checking the offsets of current ADCs).

It will run “startup” which should send dc to each motor winding to align them.

It will “closeloop” which may be unnecessary, it just switches to Sensorless control with a set speed.

It then switches to “run” which allows for proper control of the speed.

A block measures the input voltage, another controls the state and the rest reads the currents from the motor and uses this to predict the location of the rotor and run the motor at the desired speed.

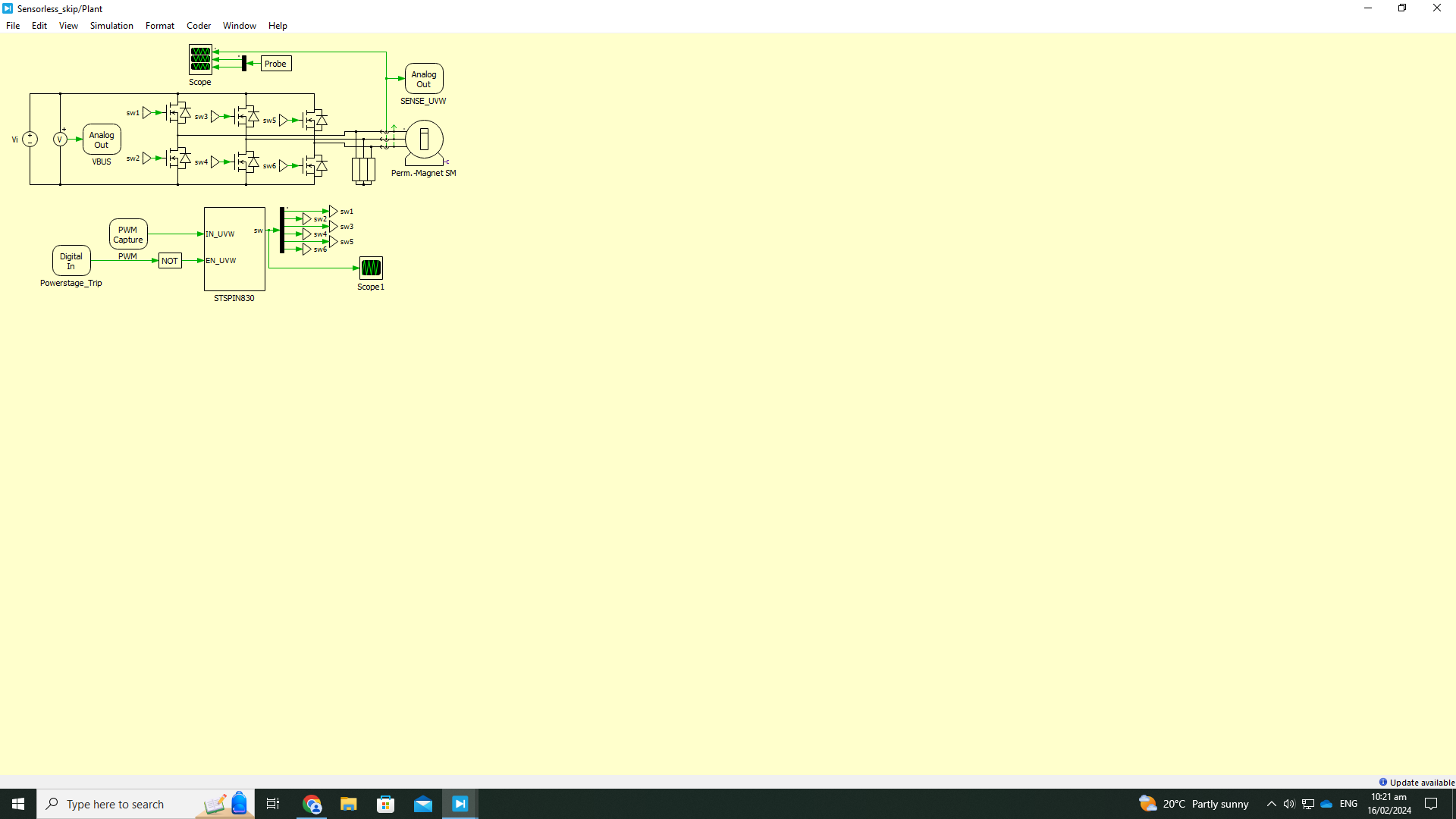
Tuning of the simulation parameters (motor parameters, initial speed and accelerations) will vastly change how well the motor runs. This was likely not completed during my time as an RA so I cannot speak to how exactly this should be done.

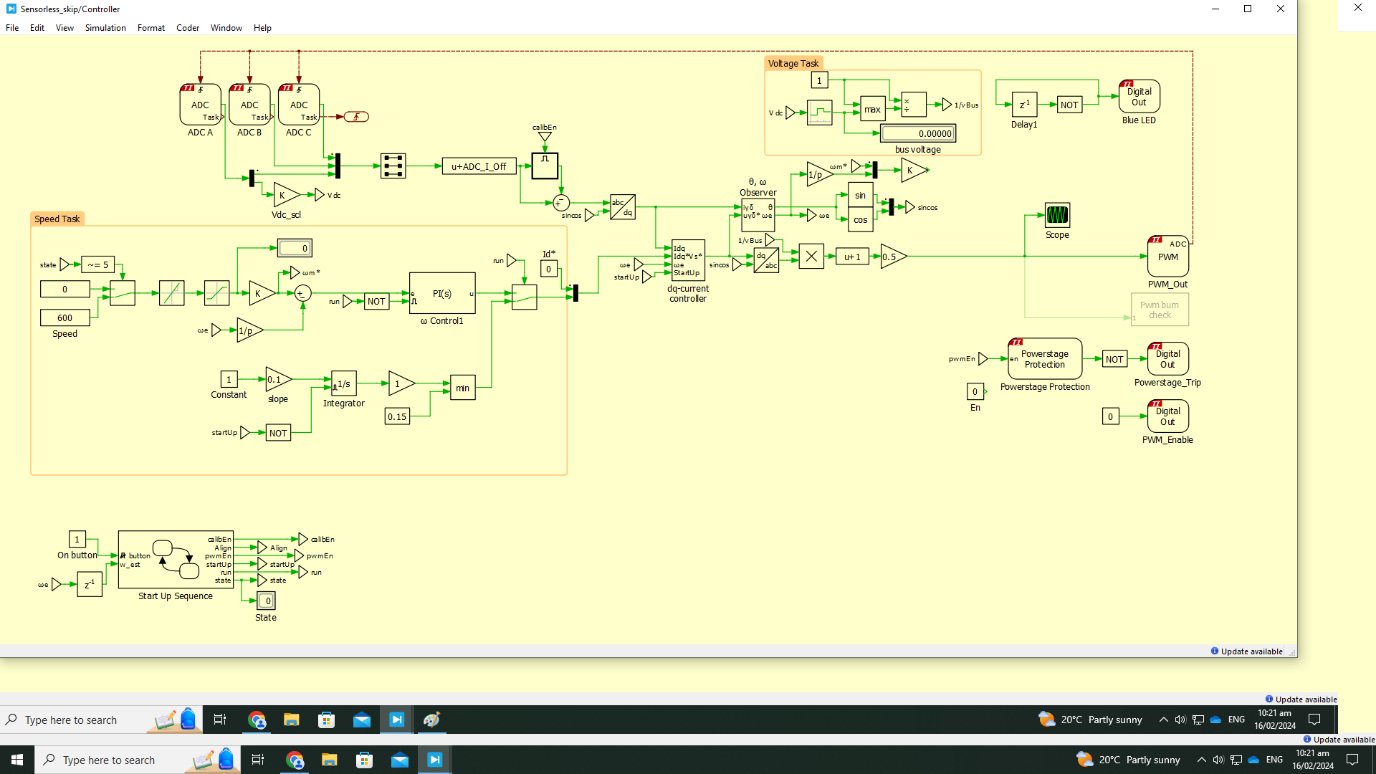
Specifically, to the version labelled Sensorless\_skip, this should run the test Flycat 2805 motor.

A screenshot of a computer

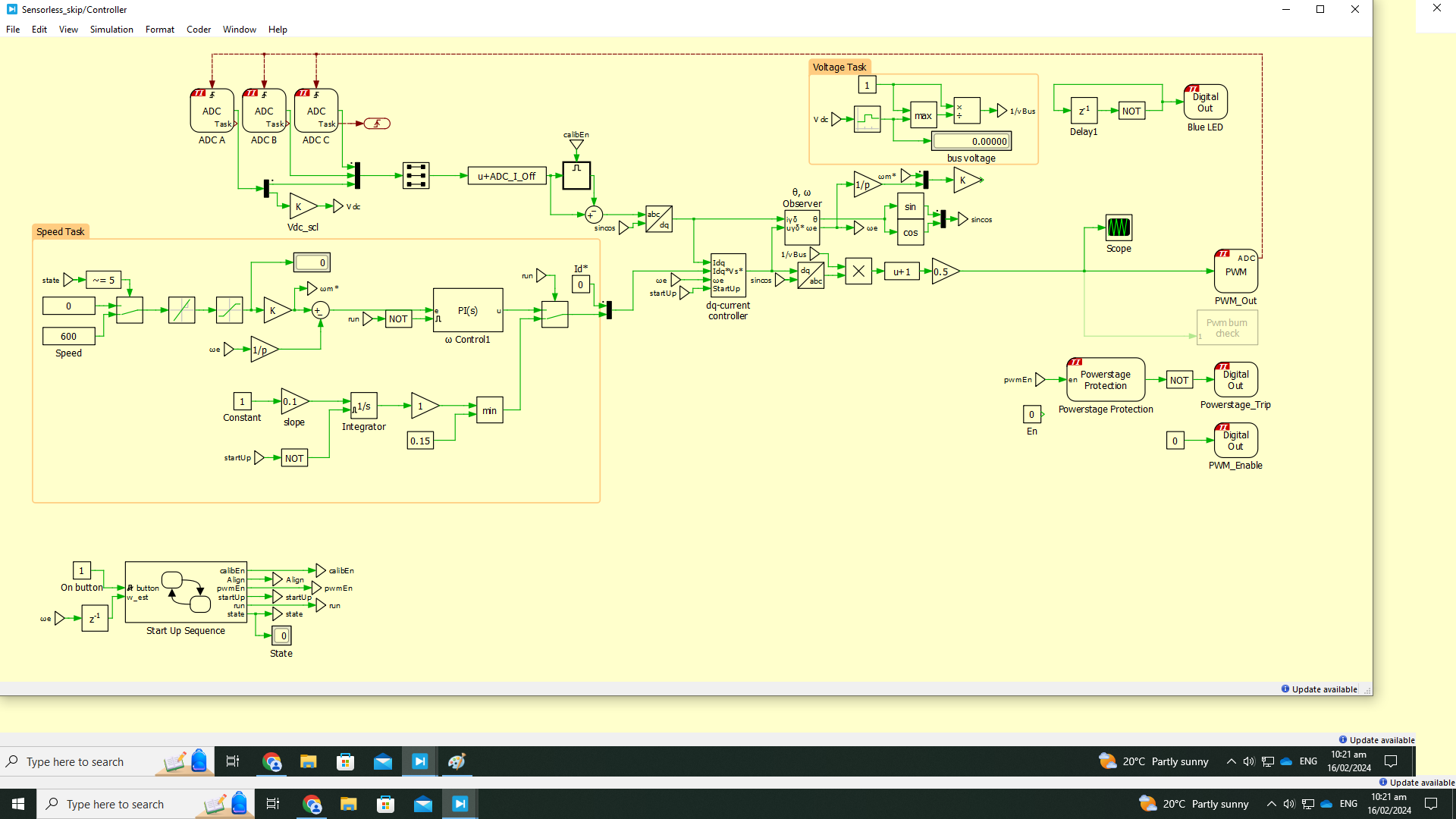
Description automatically generated

This is an overview of the PLECS code, showing the controller which is built onto the board and the Plant which can simulate a motor. This code is strongly based on the STM32 Sensorless FOC demo so additional details may be found in that documentation.

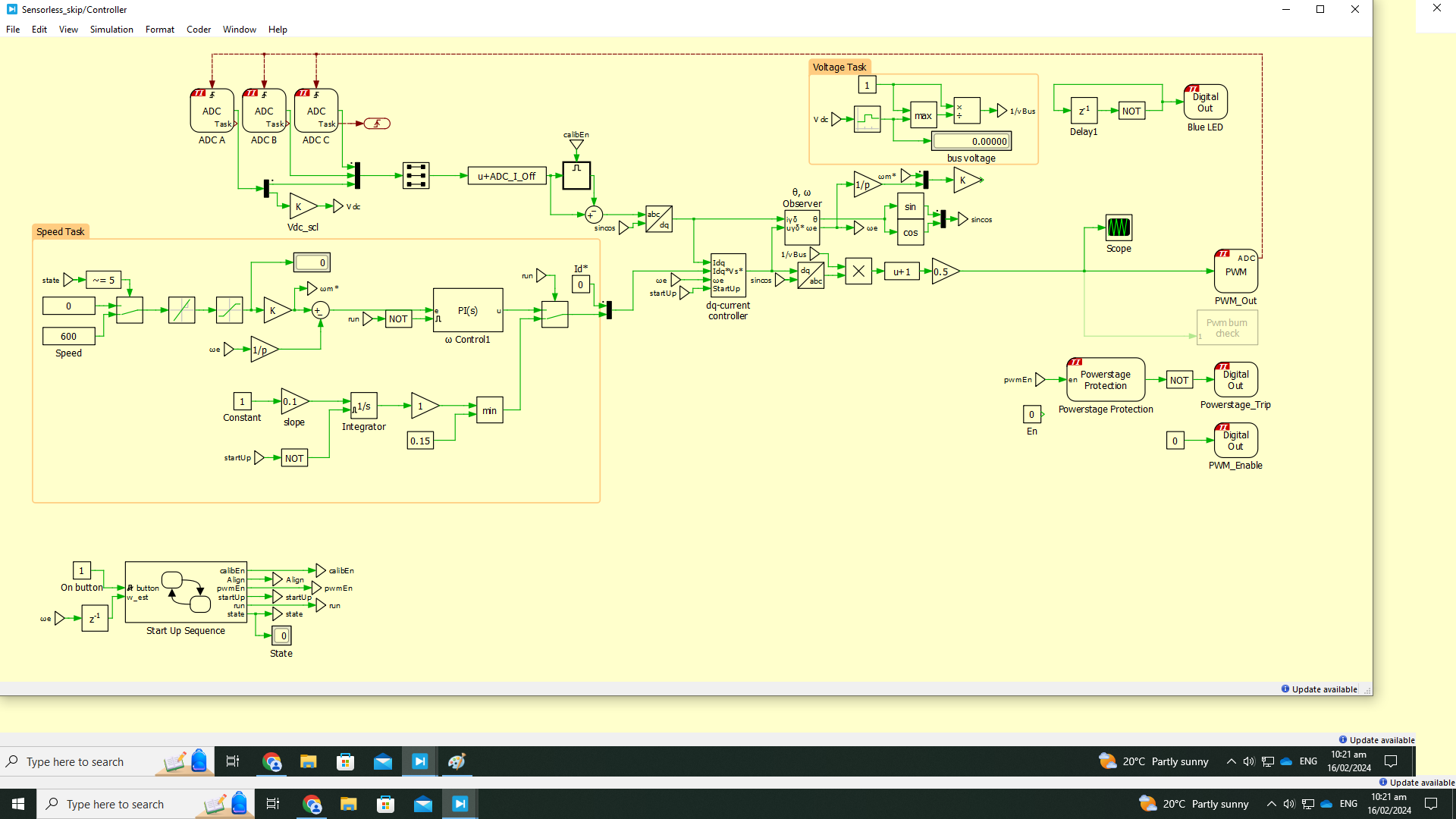
 The plant runs a simple motor with parameters specified in the simulation initialization parameters. It is not always accurate to real life even when parameters are tuned perfectly. There are likely other parameters/ interactions not accounted for.



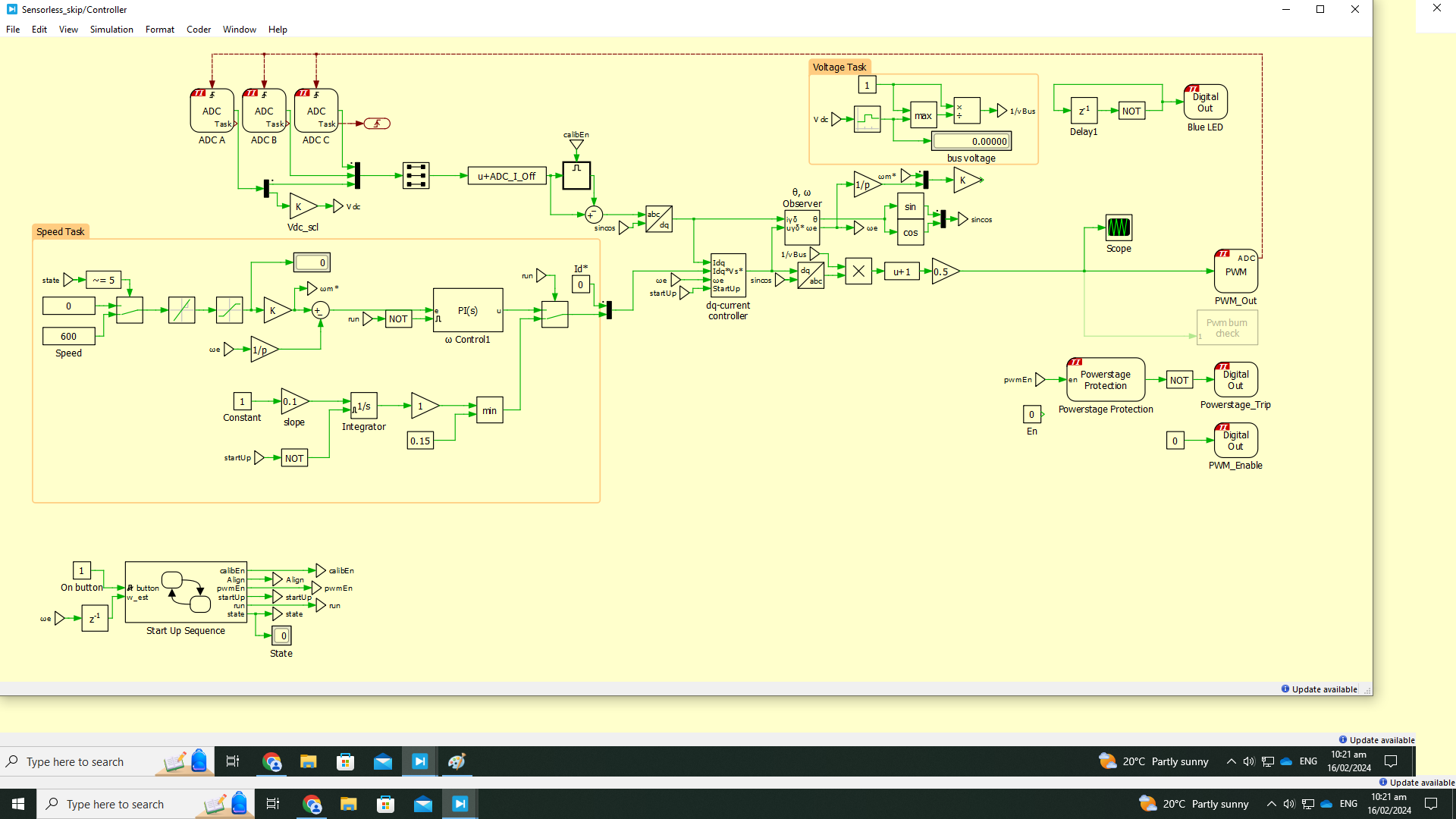
This is a view of the entire controller which will be broken down.



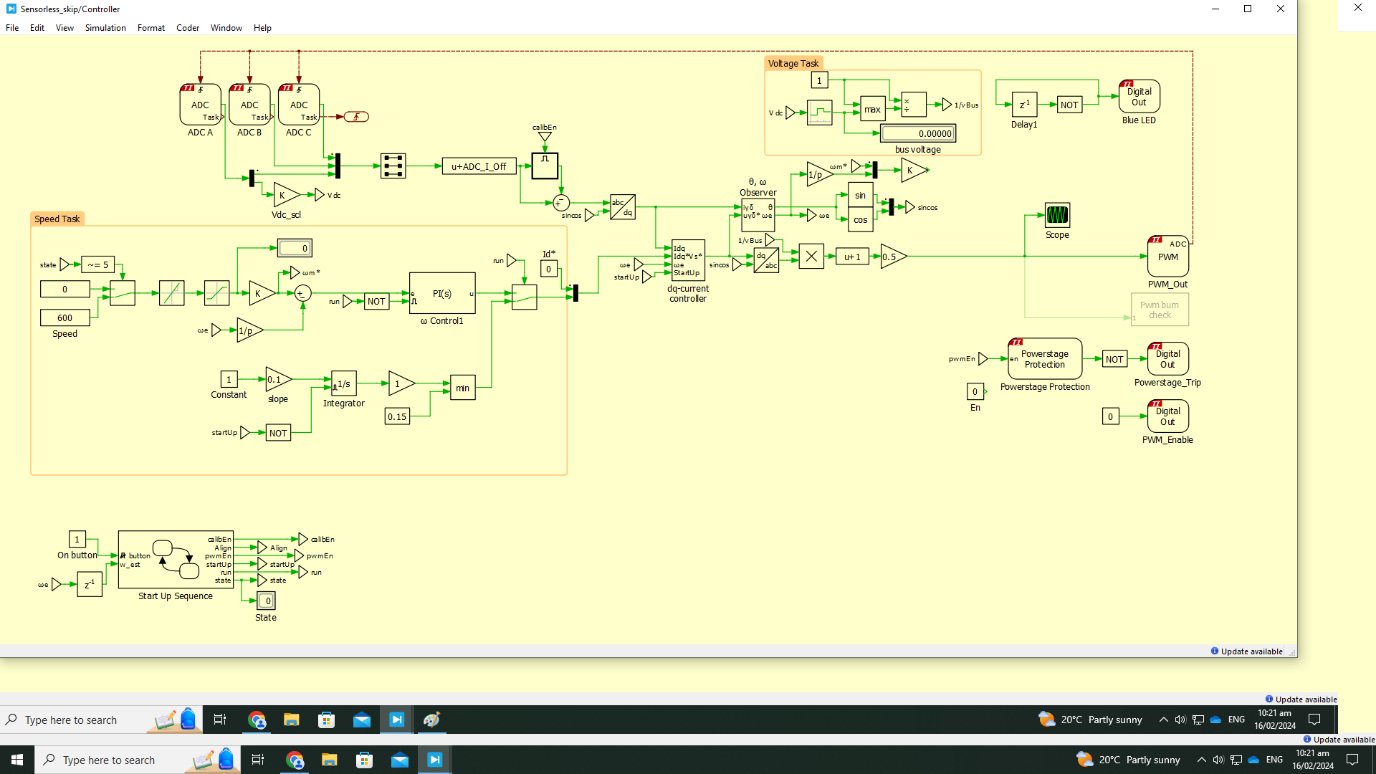
This part is intended to take a speed signal then convert it to the dq plane with the PI controller to be used for the speed control.



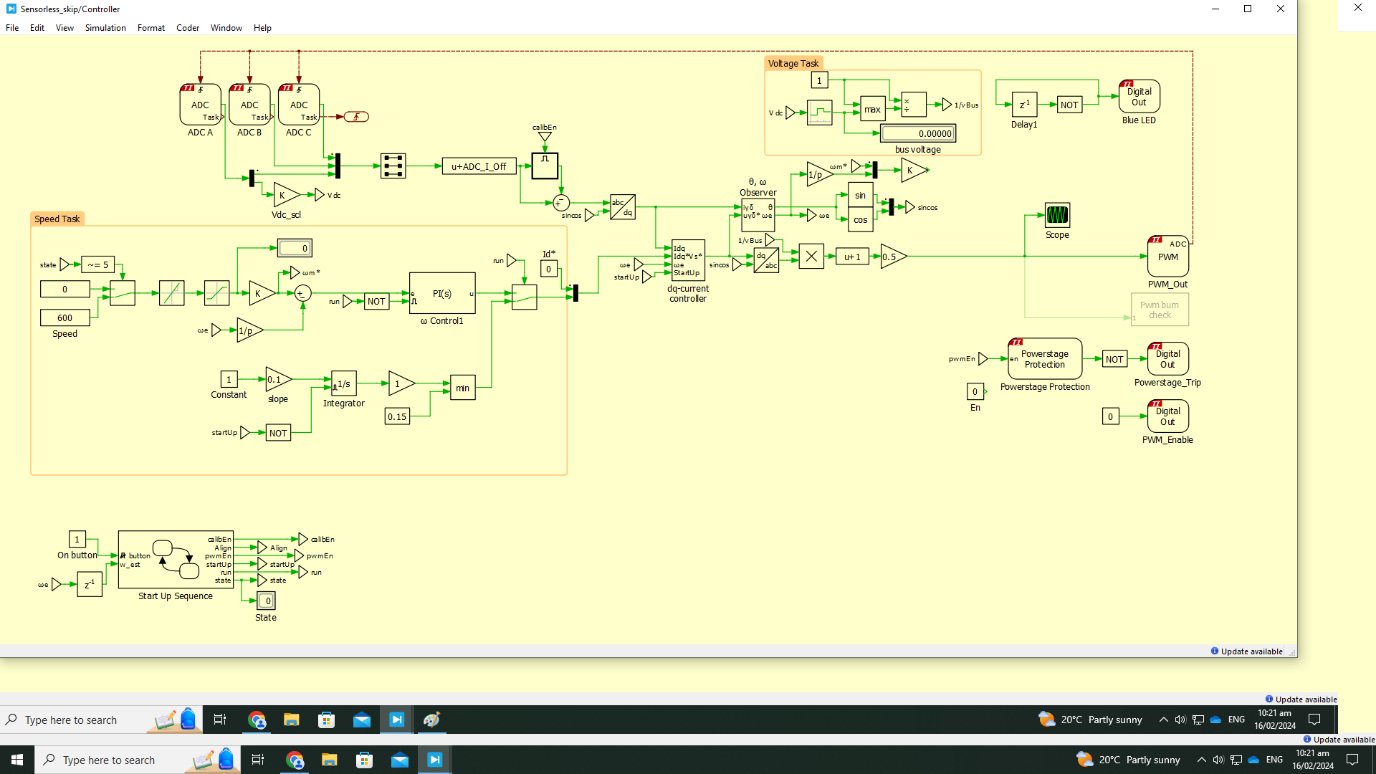
This sends a slope to during the startup sequence to align the rotor to a known position.



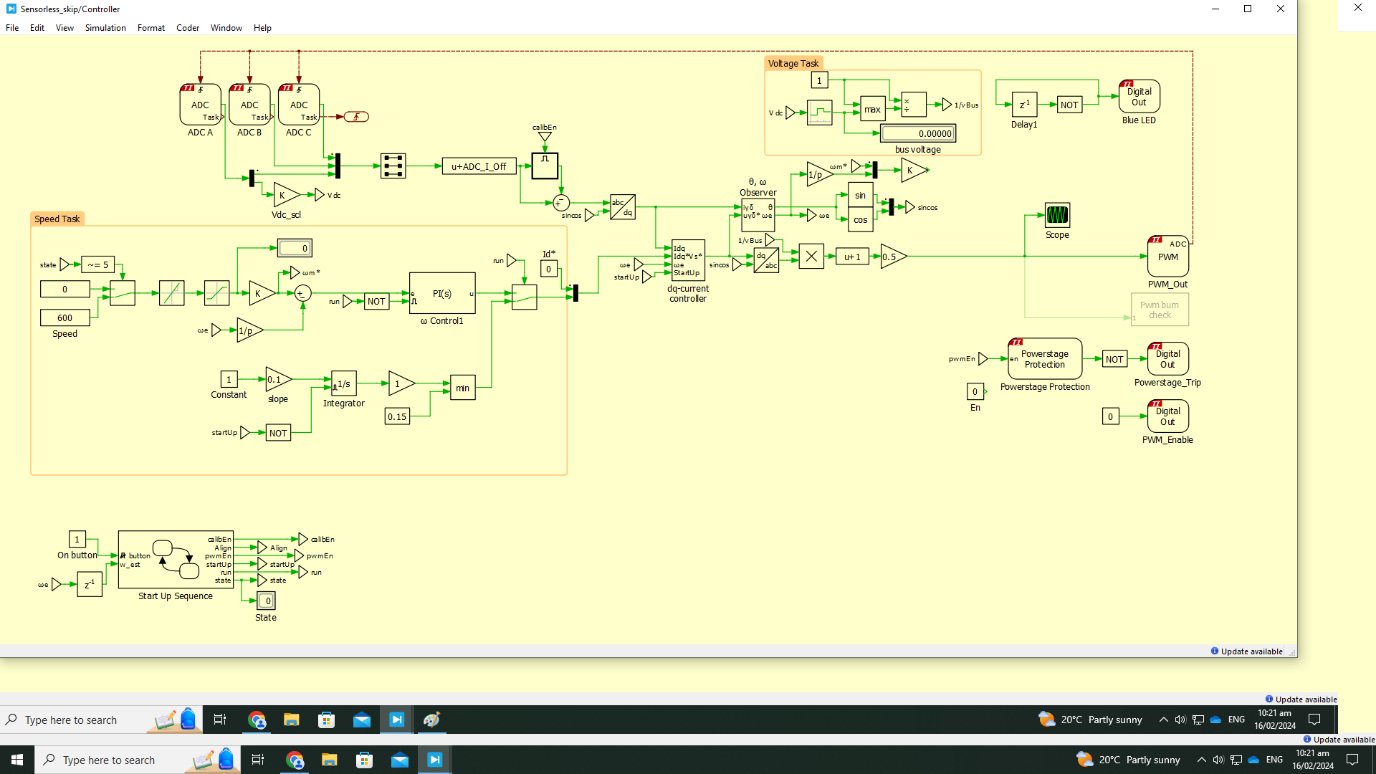
The set to 0 part is the initial speed that the motor will try to jump to, it then transitions to the number in the speed box with the rate limiter.



This controls the state of the program. And runs through set sequences.



This PWM protection block and enable block ensure the PWMs are running when they should be.



This block reads the ADCs connected to the booster board, reading the currents in the wires to the motor and the dc voltage.

A screenshot of a computer

Description automatically generated

This block averages the voltage and returns 1/Vdc. It also blinks the blue LED.

A screenshot of a computer

Description automatically generated

Here is the most complex part of the code and part I understand least. To the best of my knowledge the observer estimates the rotor position from measured currents and uses it to convert in and out of the dq plane and estimates the back-emf. The dq current controller takes the intended dq and measured dq, and back emf and outputs an adjusted dq which is then converted back to 3 phase sinusoids which are offset and scaled to be in between 0 and 1.

Blind

A computer screen shot of a computer

Description automatically generated

A driving frequency (speed) is put through a rate limiter which is then used to generate three sine waves which are moved above 0 and scaled to max at 1 then scaled by the modulation amplitude. The modulation amplitude may or may not be controlled by the BEMF constant and a minimum required to run the motor.

When driving with a sine wave and no sensing, the simple equation sin(f\*t) will cause improper waveforms while smoothly increasing. This is fixed by working in the phase space, the easiest way to do this is use the integral of frequency rather than time \* frequency.

## Debugging

Connecting an oscilloscope to PWM outputs (backwards) will kill it (yeah that ones embarrassing).

Take care of the reset button, resets connections.

To copy-paste between PLECS projects you may need to open one from inside the other(file->open).

Voltage readings are out of 80V scaled to 3.3V.

There are different offsets on current and voltage ADCs.

If using the base TI C2000 PWM block, complementary outputs should be used.

Unplug it and plug it back in again fixes several build errors.

Disconnects where the code stops running are possibly due to errors (you won’t get an error message).

Apply simulation parameters.

Having a blinking LED running constantly lets you know if it crashed.

Set a CPU load block to not overload it.

Check TZ1 pin is GPIO 14 (OT pin on booster).

#### Unsolved issues:

Board disconnecting. This may be during build or in external mode, when turning on power supply, running parts of code or when activating auto-triggering. My best guesses of factors which may be involved include buffer size, number of tuneable parameters, scopes, rate of auto-triggering, boot mode, run mode, lack of power or some form of hardware bug or corruption. Various fixes were done by removing scopes, parameters, reducing triggering rate, resetting the board, replugging the USB, changing computers and boards and overwriting the current program. This issue was possibly related to a series of BSODs when connected.

-Most recent update: appears to fix after updating PLECS, codegen tools and Uniflash as much as possible (TSP cannot currently run PWMs, PLECS support knows about this).

### Links

Launchpad

Increasing length. All are separately relevant.

[TMS320F28379D LaunchPad Quick Start Guide (Rev. A) (digikey.com)](https://www.digikey.com/htmldatasheets/production/2368159/0/0/1/launchxl-f28379d.pdf)

[LAUNCHXL-F28379D Overview User's Guide (Rev. C) (ti.com)](https://www.ti.com/lit/ug/sprui77c/sprui77c.pdf?ts=1705459313333&ref_url=https%253A%252F%252Fwww.bing.com%252F)

[TMS320F2837xS Real-Time Microcontrollers Technical Reference Manual (Rev. H)](https://www.ti.com/lit/ug/spruhx5h/spruhx5h.pdf?ts=1704766915996&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FTMS320F28377S)

Booster schematic

[BOOSTXL-3PhGaNInv Schematic (Rev. A)](https://www.ti.com/lit/df/sluray0a/sluray0a.pdf?ts=1704752569661)

Inverter stats

[LMG5200 80-V, 10-A GaN Half-Bridge Power Stage datasheet (Rev. E) (ti.com)](https://www.ti.com/lit/ds/symlink/lmg5200.pdf?ts=1705465988619&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLMG5200)

Motor types (BLDC vs PMSM)

[[FAQ] Trapezoidal Motors vs. Sinusoidal Motors - Motor drivers forum - Motor drivers - TI E2E support forums](https://e2e.ti.com/support/motor-drivers-group/motor-drivers/f/motor-drivers-forum/909911/faq-trapezoidal-motors-vs-sinusoidal-motors)

[Electric Motor Windings Comparison (oswos.com)](https://oswos.com/motor-windings/)

More information on the hardware used does exist online, but I found these to be the most useful.

## Design

One overview is that a signal is sent from the computer to define a frequency on the Launchpad, creating sine waves to modulate a PWM signal sent to the motor. Data is recorded with a DAQ and LabView.

### History

Throughout its use, the method of transferring a signal to the Launchpad has been adjusted.

The initial design sent a signal from the computer to an analog output device connected to analog pins on the Launchpad, which would read the voltage, then be converted to a PWM and sent to the motor.

This design eventually ran into signal noise problems. While the produced signal was very clean, the analog pins receiving the signal observed fluctuations around 0.1mV. While this is not huge, it must be cleaned to drive the motor smoothly. There was also a distinct offset in the average value of the voltage read by the pins, which was found to vary significantly over time (up to 100%), voltage and pin. Several solutions were explored in conjunction with each other. Very rough rounding helped make the signal clean enough but reduced the resolution at which the motor speed could be controlled. Fast changes in speed **did** damage the booster board multiple times. If readdressed, this problem could be alleviated using the rate limiter block in PLECS. Furthermore, no matter how harsh the rounding, there would always be a line which, if the motor were to oscillate around, would cause unpleasant speed changes. For example, a voltage of 0.5, when rounded to the nearest integer, can oscillate between 0 and 1. Very long periodic averages slowed the response time of the motor with much less benefit than was expected. Calibration was also addressed. Passing both 1V and 0V to pins on the Launchpad to correct for scale and offset was found to be almost useless due to the noisy readings being specific to each pin, only exacerbating the problem by adding more sources of noise.

A possible solution to avoid this noise was to send a pulse train to GPIO pins on the Launchpad instead. While this would be perfectly possible, it would require some non-trivial interpretation on the Launchpad, so it was not implemented due to there being a more accessible option.

The Launchpad is currently controlled through external mode with parameter tunability. The computer is directly connected to the Launchpad with a USB cable, allowing for real-time motor speed adjustment and visualisation of the response. The noise in this connection is considered negligible. The booster board power supply must be isolated from the Launchpad while connected to the computer to prevent overloading the Launchpad. More details will be in the operation section of the documentation.

## LabView code

This reads and saves temperature and rpm data to a file, activating the relay connecting the driver to the motor. Most relevant information can be inferred from the front panel. Further specifics should be inferred from the code itself, as this documentation will not be able to keep up with changes.

In the past, when this was used to output an analog signal, it was essential to reset that signal to 0 before aborting the program.

Mohammad has made adjustments to the LabView since I had access to it so the version here is not the most recent.

### Further work

If this project is readdressed, one way to increase the stability and likely the speed is to address rotor dynamics. A brief outline of what I thought may be relevant can be found in the MagMapper documentation.