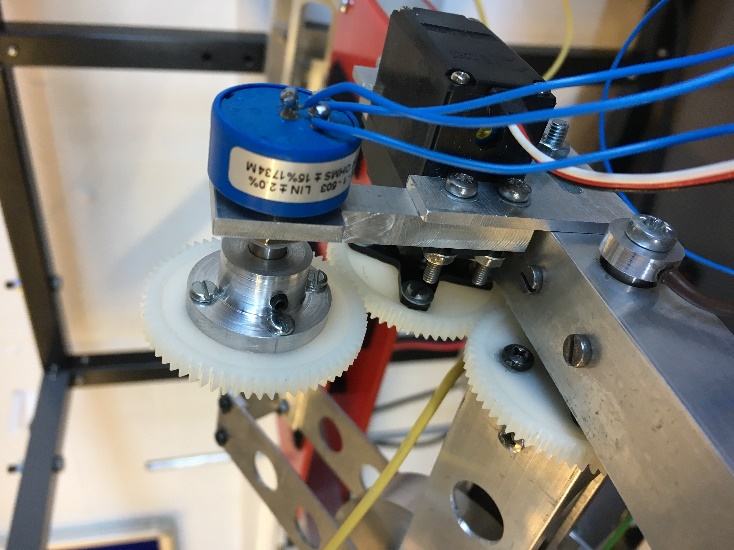
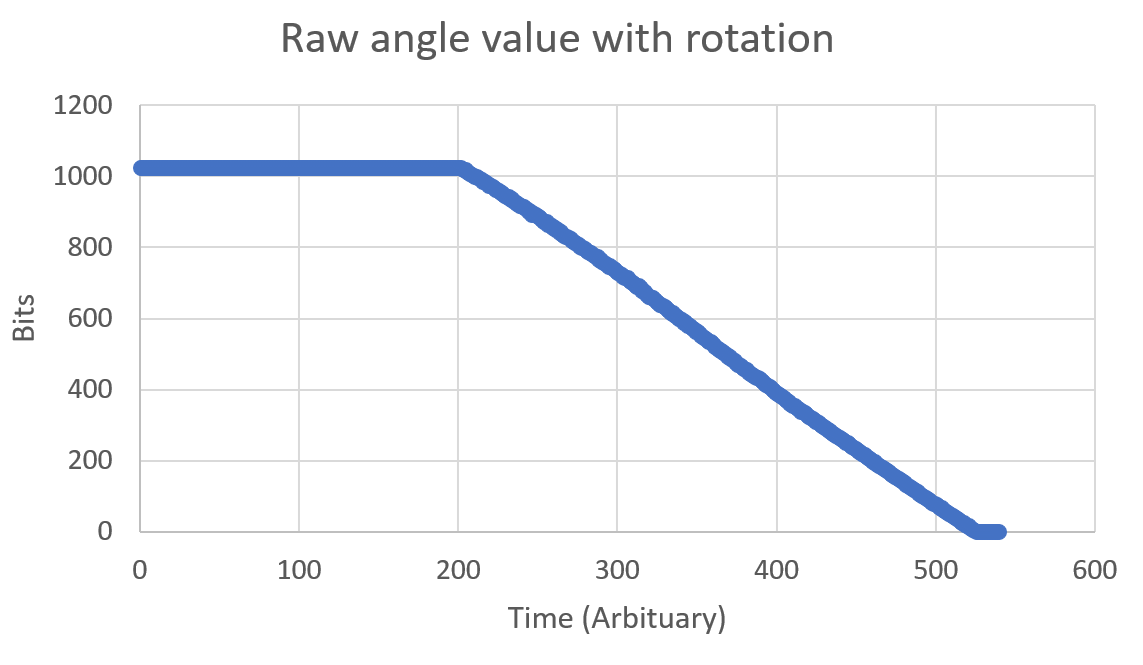
# Term 2 Week 10 summary – James Davies

## **Flywheel Orientation**

An important aspect for the code to produce correct RPM outputs to the servos is to know their current orientation due to the cos(θ) drop off.

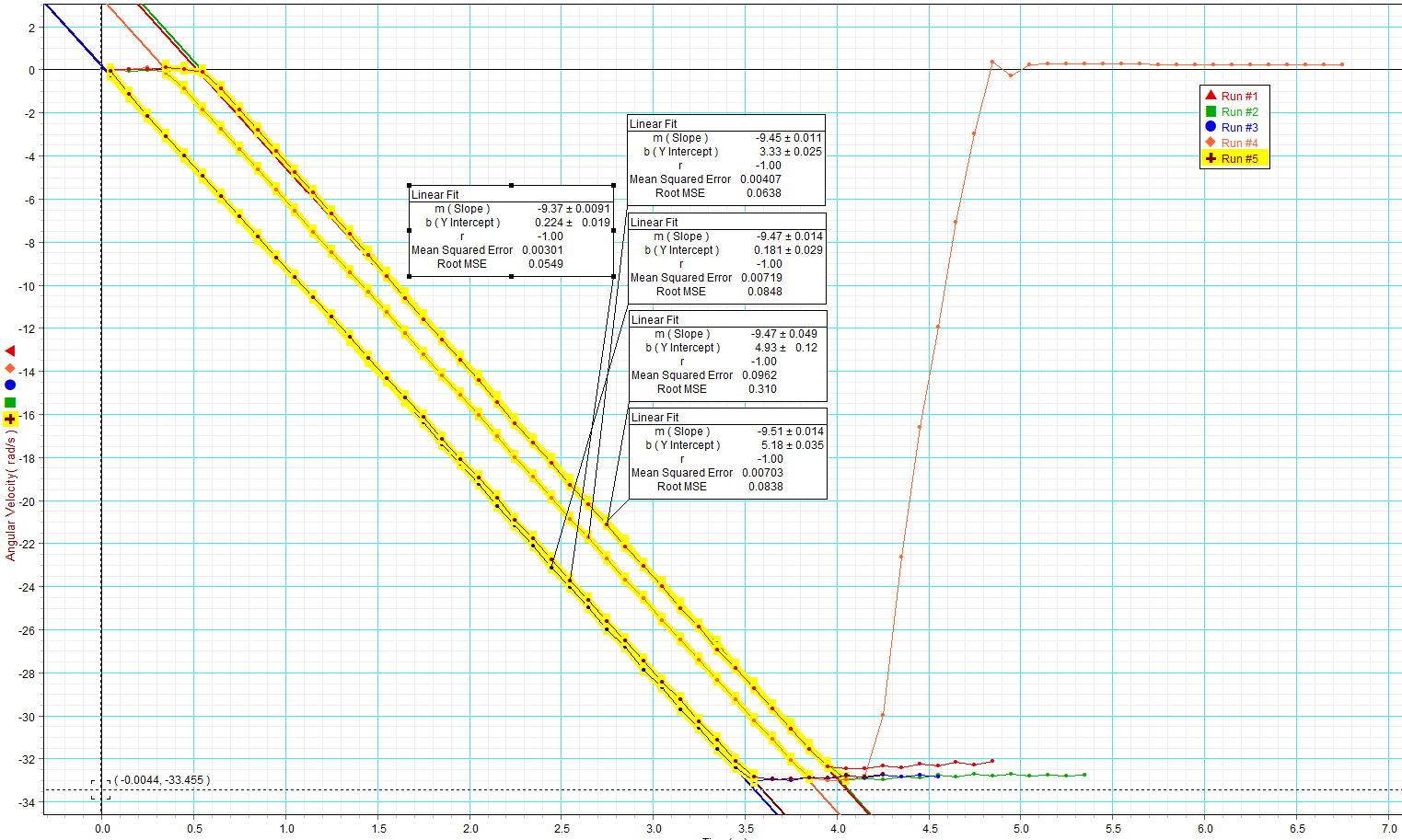
The code previously mentioned to do this was very intensive and highly inaccurate. This lead to the addition of the continuous potentiometer being attached and geared into the servo and flywheel housing, as show in the picture.

Using the 5V and analog pins on the Arduino can measure the change in voltage across the resistor as the servo rotates the flywheel. The potentiometer resets its resistance back to 0Ω every 360 degrees. Each gear was equal size so the ratio of turns of the flywheel to the potentiometer was 1:1.

There was a small issue of there being a “dead” in the measured voltage change across the resistor, as shown in the graph above. 0-1024 bits relates to a 5V signal. To get around this issue I moved the flat/zero point of the flywheel to the active region of the potentiometer. This shouldn’t be an issue as the voltage change is still apparent over the full working range of the flywheel.

## **Moment of Inertia of Large Flywheel**

The new larger flywheels have been manufactured, but for the code the moment of inertia of them need to be determined. To do this I performed an experiment that allows the flywheel to spin, as a mass falls, converting the potential energy of the falling weight to kinetic energy in the form of the rotation of the flywheel and the velocity of the falling mass.

Measuring the angular velocity of the flywheel as a function of time an plotting the data produced a value of 3x10-3KgM2, which is about 24x bigger than the small test flywheel.

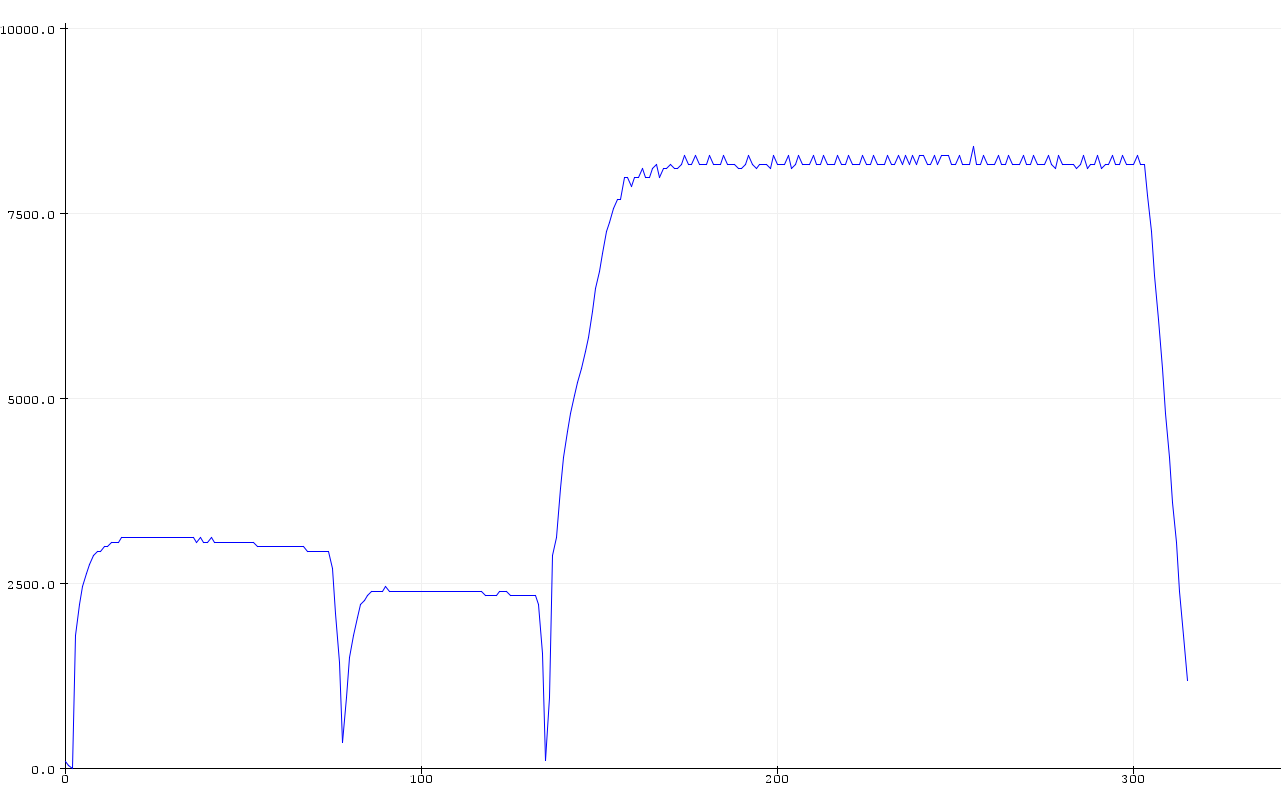
## **Safety Code**

With the larger flywheels being capable of outputting huge forces, I went about coding in some safety code. The three main functions:

1. RPM acceleration limiter – this stops the flywheel from changing directions to quickly damaging the mounts.
2. Smooth Servo output – this creates a rolling average of the output to smooth the movements, eliminating any “jidder”.
3. Failure Cut off – this causes the code to break and kill power to the motors if the system fails.

There was also the addition of a reset function, which returns the system back to the zero state on a Arduino reboot.

## **RPM measurement**

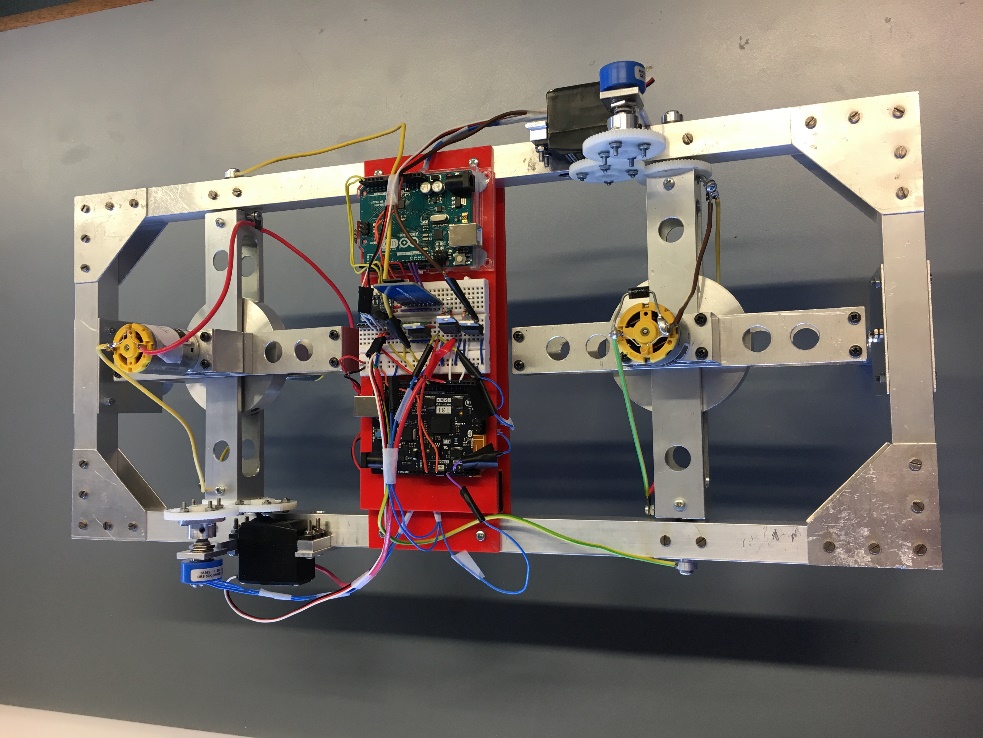
Again, for the equation shown on the first page, the flywheel angular velocity is important, so I went about designing a system that can measure the RPM live. The basic setup of the system is a photodiode and a IR LED in parallel that can reflect light from the flywheel. The flywheels have been painted black and reflective tape attached on the circumference. This is done to get a better more obvious signal when the tape passes the photodiode.

In the code, the signal is zeroed by taking an average measurement first to remove noise. The code then goes through a state -based logic when it finds a signal which is above a threshold. This then counts the time to find 10 peaks then works out the average RPM value.

The graph shows the rpm output from the code for three different velocities of the test flywheel spinning.

## **Rewiring and LIPO**

The whole frame has been tidied to remove dangling wires to reduce the risk of shorts. This was for in preparation of using the LIPO to power the system, which can power all 4 motors and 2 servos at once. This makes the system fully isolated now.



## **Dissertation Plan**

1.0 Introduction

Introduce the topic, and give some background on the general topic.

2.0 Theory

* Gyroscopic Theory
  + What is gyroscopic force?
  + How can we use gyroscopic precession for stability?
* System Model
  + Equations of motion
  + Moment of inertia equations
* Microcontrollers and Code
  + Arduino Uno for measurement
  + Arduino Genuino for control system
  + Code and libraries
* Circuits
  + Control circuit components
  + Measurement system
* PID
  + What is PID?
  + Application for control system

3.0 Methodology

* Test frame, and frame evolution.
  + Basic frame
  + Servo Types
  + Angle measurement system
  + Motor placement
* Controller and code approaches
  + Motivation for the two controllers
  + Filtering and smoothing
  + Accuracy assumptions
    - How precise does the system need to measure?
  + Functions
* Physics in code approach
  + Why apply the physics? (not just PID)
  + Limits and assumptions in coded physics
* Flywheel design and moment of inertia measurement.
  + Flywheel requirements
  + Design thoughts
    - Ease of production
    - MOI/Mass ideas
  + How to find MOI experimentally
  + Implementation into code.
* RPM measurement system
  + Photodiode and LED.
* PID application
  + Used for system tunings, and physics correction
  + Less dependence on correct values for any system changes (like rider mass)
* Code manipulation and PID changes
  + Code Structure, and optimisation
  + Method of PID tuning
* Safety Precautions, both code and physically.
  + Safety code
    - Servo Limits
    - System Failure procedure
  + Physical protection

4.0 Results

* New wheel MOI measurements
* Working with small wheel, and limitations (Need to go back and get some data for flat system)
* Raised system, and failures
* System optimisation, and PID changes.
* Issues!

5.0 Discussion

* Results
* Issues
* Further development

6.0 Conclusions

Conclude here.