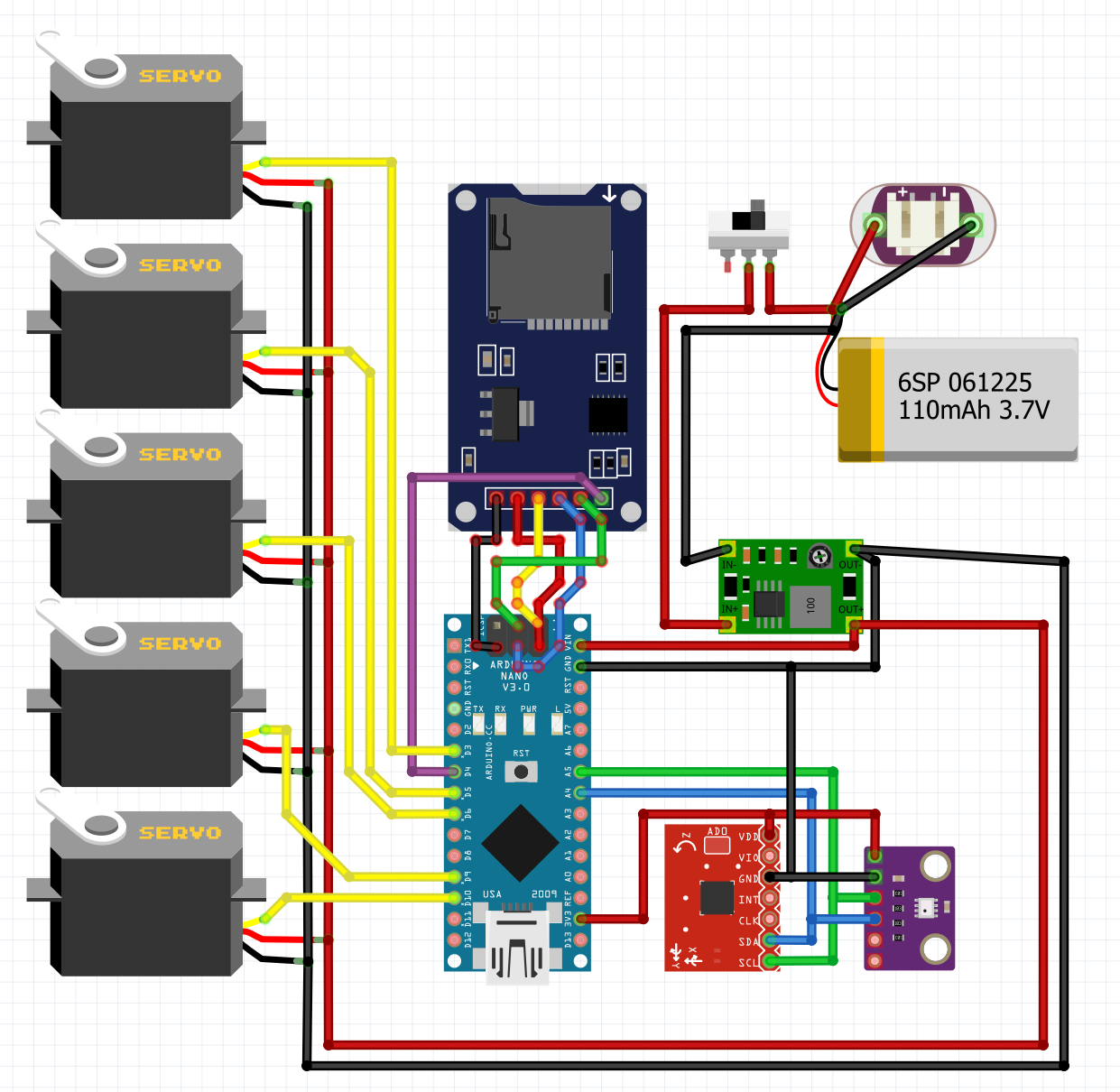
Is it possible to create a self-stabilizing rocket?

Well yes of course, nowadays almost all missiles have a guidance system that allows for trajectories to be calculated and finely controlled. Despite the widespread use of this technology, it is mainly restricted to large corporations with huge RnD departments however, the rise of consumer electronics recently had given the ability for individuals to create complex circuitry and controllers. This will be the focus of this endeavour, is it possible to create a self-stabilizing model scale rocket?

The Control System

The brainbox of this operation will be the Arduino, a small programmable micro-controller (essentially a very small computer) which will read the pitch, yaw and roll as well as the altitude of the rocket and use that to control the angle of four fins mounted to the top of the rocket.

Secondary aims include to calculate the altitude of the rocket from atmospheric pressure readouts and to hopefully extrapolate velocity from it. Another aim is to use a SD card to record the data from the rocket and be able to act as a datalogger for analysis after launches.



Boost converter

Li-po battery

Power switch

Charging port

Gyroscope

Altimeter

SD card reader

Arduino

**Circuit Diagram**

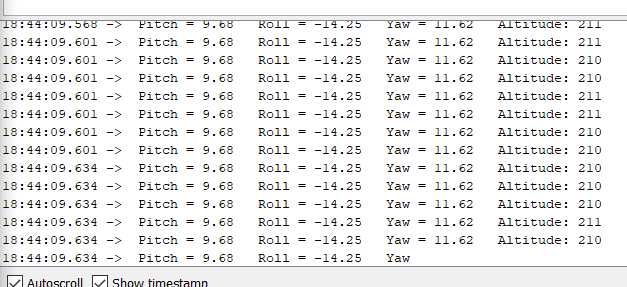
Data input and outputs

Firstly, powering the components:

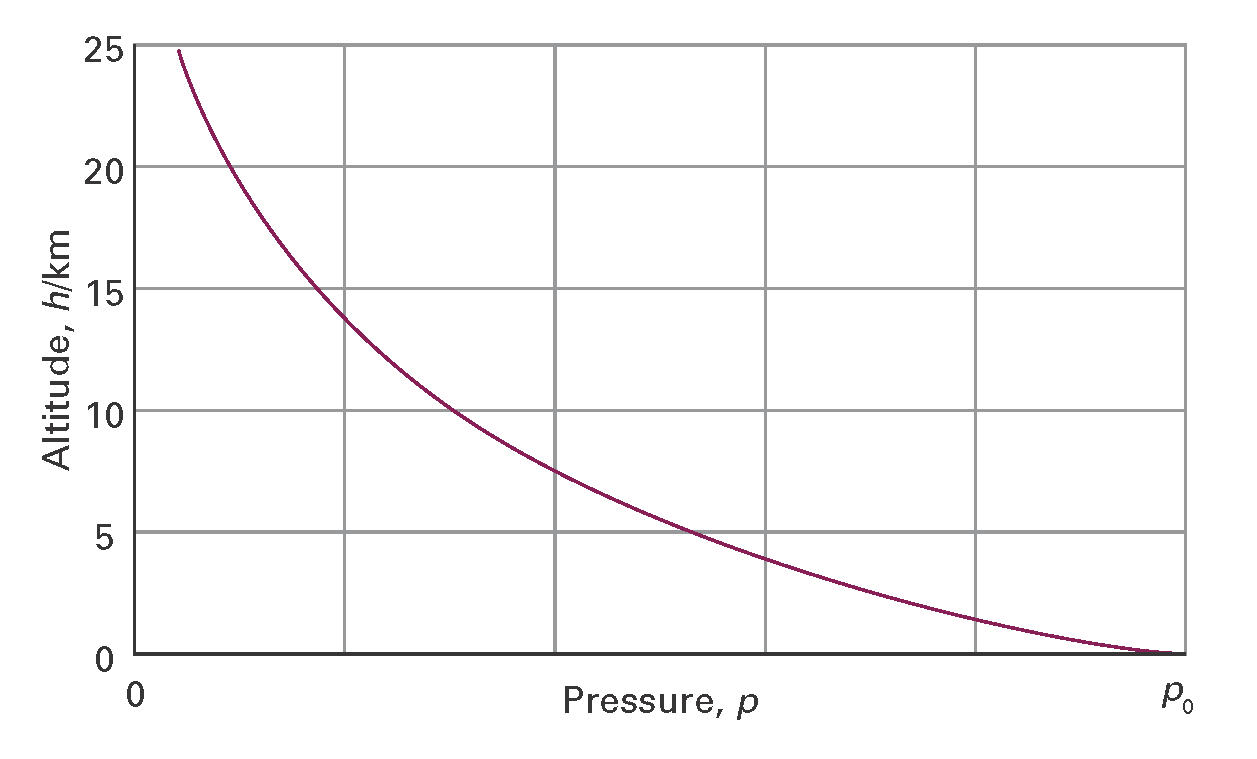
1. The voltage from the lithium polymer (li-po) battery is boosted from 3.7v (nominal cell voltage) to 6.0v since the other electronics run on 5v logic. The additional 1v is needed since there will be a small 0.7v voltage drop from the power regulator on the Arduino.
2. The Arduino then powers the gyroscope and altimeter at 3.3v since they run on 3.3v logic and will most likely become damaged when powered at 5v.
3. The servos are connected directly to the boost converter’s 6v as it means that no current needs to run through the Arduino to get to the servos. This is important as the servos will be drawing about 100mA under no load, and up to 500mA at full which the Arduino cannot supply without burning up.

Secondly, the controls:

1. The Arduino communicates with the modules and reads the angles off the gyroscope and the pressure off the altimeter, receiving data in the format of:



1. Since the altimeter outputs pressure in pascals*,* it needs to be converted into a altitude above sea level in



1. From the angles received, the Arduino will then send a signal to the servos along the yellow wires, instructing them to move to a different angle if necessary.

At this point, emerges the main problem of control, how much should the servos change angle?

PID controllers

If the angle change is too much, it will cause the rocket to over-correct, leading to the servos swinging back to correct the over-correction. This leads to a system that rapidly loses stability essentially balancing on a knife’s edge. What is needed is a system that actively measures the change in angle and corrects for it. Systems like this are called PID controllers and are used in inherently unstable scenarios such as self-balancing robots, temperature regulators or in this case, a model rocket.

PID controllers work by taking the difference in desired angle and current angle(called the “error”) and computing a needed angle change (called the “solution”) which is sent the servos to achieve that while monitoring the error constantly.

The 3 components of a PID controller are:

1. Proportional –

The controller takes the error and sets the solution to be proportional to that difference. If this is set too high, it will cause the problem as mentioned above and cause rapidly growing oscillations but set too low there will be little change in the error. Thus, something is needed to dampen the oscillations.

1. Derivative –

This considers the magnitude of the solution computed by the proportional and decreases it according to how small the error is. It is effectively a damper to the proportional part of the controller and stops the oscillations by lowering the solution according to how small the error it. However, another problem arises, if this is set too high, it will dampen the solution too much that the error will never be 0 (the system will never reach its desired value) as the dampening effect will become greater than the solution, leading to an equilibrium achieved just below the desired value. Thus, the last part of the PID controller is needed.

1. Integral –

This finds the integral between the current angle and the desired angle. The smaller the error, the larger this component becomes. It essentially pushes the solution just enough that the solution will become zero when the error is zero, thus achieving the desired angle.

|  |  |  |
| --- | --- | --- |
| Component | Too high | Too low |
| Proportional | Rapid oscillations | Changes very slowly |
| Derivative | Equilibrium achieved below desired angle | Rapid oscillations |
| Integral | Small, rapid oscillations | Never reaches desired angle |

The main problem with controllers like this is that it requires extensive testing and tuning the constants for each of the components. Additionally, they constants would most likely need to change according to the relative wind speed, creating further problems.

Coupled with the need for complex calculations on the Arduino’s part just to compute the solution, as well as the resulting decreased refresh rate of this systems means that it is not very suitable for small scale rockets. It is possible to get around this hardware problem by using better micro-controllers however the prices for a control board rises very rapidly.

The Alternate Solution

The method of stabilization used in the end is a relatively simple system which has many advantages over a PID system. The process is:

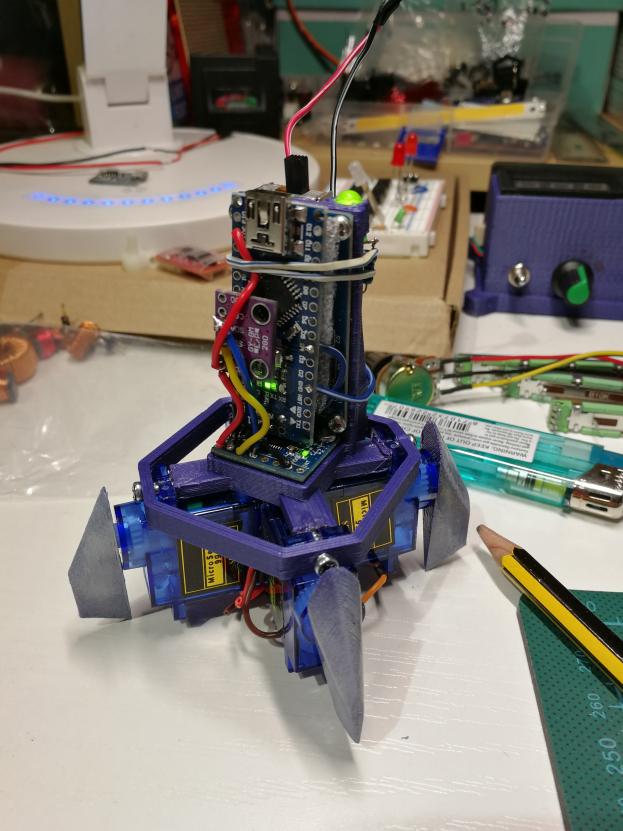
1. Determining the angles of pitch, yaw and roll of the rocket.
2. Moving the servo motors so that they always are perpendicular to the ground, resulting in almost no further calculation needed as well as negating the opportunity for over-correcting to occur.

Working in 2 dimensions, if the rocket is angled slightly to the left, the fins pitch to the right (relative to the rocket) but remain perpendicular to the ground. This results in the rocket experiencing a force which pushes it to zero degrees or vertical.

In the end, we opted for this solution since it make the code a lot easier as well as being less demanding on the Arduino. This also meant that we did not need to perform any wind tunnel experiments to adjust the controller, also a highly time-consuming task. As stated above, the settings would most likely need to change according to wind speed and with the rocket reaching up to 70ms-1, it would be impossible to simulate this in the wind tunnel setup we had.

Results

Following the launch on the 4th May, we saw that the correction system functioned perfectly with the rocket flying straight up, overshooting the desired height as it was our first launch. Unfortunately, due to the rocket reaching a high altitude of 371m, the rocket drifted on the parachutes for a very long distance away from the launch site, resulting in the bottom part of the rocket being lost, preventing us from performing further launches. Despite top half of the rocket containing the electronics being recovered, we collected no data due to either a programming error or other hardware errors.



Left to right:

Nosecone, electronics bay, middle tube section, bottom stage, motor mounts, fin support structure.