ROCKET CONTROL SYSTEM

The rocket system has a double integral transfer function. This is because the damping of the system is only produced by air friction which is negligible compared to the torque applied by the motor gimballing. The system has a constant phase of 180 degrees, so is inherently unstable. A lead controller will be designed to control the rocket pitch and yaw angle.

The lead controller will be designed to have a low frequency zero and a high frequency pole around the 0dB frequency of the system transfer function. A maximum phase lead of 30 degrees will be added at the 0dB frequency of the system.

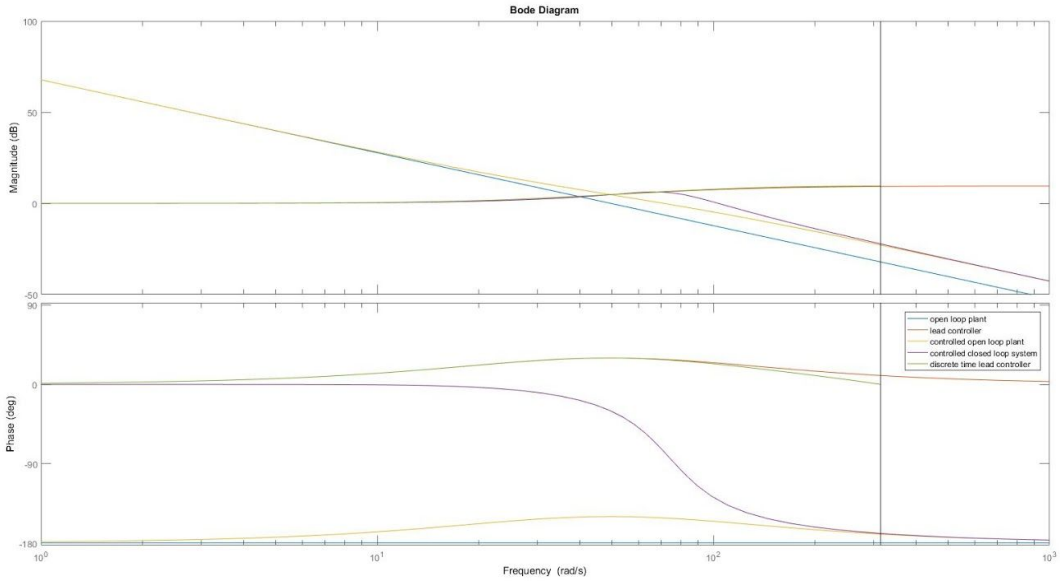
The system transfer function is given by (Ft*Dm)/Is^2, where Ft is the thrust force of the motor, Dm is the distance from the motor to the centre of gravity and I is the rotational inertia. The lead controller implemented will have the form 1/alpha((s-Wb)/(s-Wb/alpha)), where frequencies Wb and Wb/alpha straddle the 0dB frequency of the system.

When transfer function is converted to discrete time using a first order bilinear transformation, a discrete time digital filter can be created using the current values of the input, and the values of the input and output 1 sample ago.

controlSystem.m matlab code in control system folder in repository calculates the continuous time lead controller transfer function and discrete time transfer function. The inputs are the values of rotational inertia, distance to centre of gravity, motor thrust force and sampling frequency. These can be altered for other shapes and weights of rocket and matlab will output an applicable controller.

A bode plot of each of the system transfer functions is shown here.
Definitions are as follows:

* open loop plant
  - Uncontrolled system with motor angle as input and rocket angle as output
* lead controller
  - Lead controller transfer function on its own
* controlled open loop plant
  - Open loop plant multiplied with lead controller, open loop (no feedback)
* controlled closed loop system
  - Controlled open loop plant with feedback. This represents the final rocket system as a whole.
* discrete time lead controller
  - Transfer function of lead controller by itself when sampled at 100Hz, rather than continuously. The higher the sampling rate, the closer this will match the continuous controller. There is no need to sample higher than 100Hz.




```
>> controlSystem

Cd =

  2.4 z - 1.801
  -------------
   z - 0.4004

Sample time: 0.01 seconds
Discrete-time transfer function.

Warning: The closed-loop system is unstable.
> In ctrlMsgUtils.warning (line 25)
  In DynamicSystem/margin (line 65)
  In controlSystem (line 30)

PhaseMarin =

   48.4042
```

The output of the Matlab program shows that the controlled system has a phase margin of about 48 degrees which means it is very stable and should have lots of room to move if any of the constants defined at the top change. This is actually very likely, as the motor thrust force has been modelled as a constant 11N, which in reality is not the case at all.
Ignore the error message that it is unstable, matlab will always say this when plotting a discrete time transfer function.

The Z transform equation that matlab outputs can be rearranged to give a discrete time controller implementable in software.

$Y(n) = 2.4*X(n) - 1.801*X(n-1) + 0.4004Y(n-1)$

Y(n) is the current output. X(n) is the current input. Y(n-1) is the output one sample ago. X(n-1) is the input one sample ago.

CONTROL SOFTWARE

The control software calculates the current angle of the rocket by reading values from an IMU (Inertial Measurement Unit). It uses these values to calculate the desired motor gimbal angle using the lead controller.
The control code was written using PlatformIO, and can be easily used by importing the controlCode folder as a platformIO project. It can also be run using the arduino IDE, but the MPU9250 (the IMU) library must be copied into your arduino libraries.
I2C communication is used to commuicate with the IMU using the Teensy3.2 and outputs voltages to the servo motors that control the motor gimballing angle.
There are a few variables that must be altered to run this code for different rocket systems. First of all, Xcentre and Ycentre must contain the values that set the servo angles to their actual centre. Next, servoXratio and servoYratio must be altered to represent the ratio between the servo angle and the actual angle the motor moves to. In our small scale rocket system this represents the ratio between the distance from the point the motor is moved from to the pivot point, and the servo arm length. Multiplying the servo angle by this ratio gives the angle the motor sits at.
Finallly servoOffsetRatio represents the inherent servo offset due to the servo actually moving a little less than the angle you set it to.
To use this code for a different system, the lead controller values in the main loop can be altered.