Model Rocket PID Tuner

Guido di Pasquo[1]

I. INTRODUCTION

Since there have been an increase in model rockets with active control systems, mainly in the form of TVC, the question of how to tune a PID has arisen. The fundamental problem in PID tuning for model rockets is the cost of in-plant tuning, since each iteration, or flight, needs a new motor, and the risk of destroying the model in the process is high. For that reason, a simple, semi-plug and play, Python based, tuner has been developed.

II. PID TUNER UNDERLIYNG MODEL

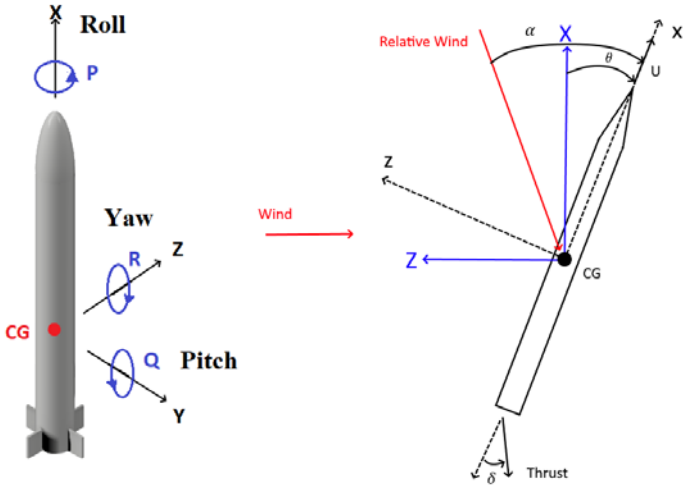

Figure 1 – Diagram of a model rocket, where Q is the pitch rate ($\dot{\theta}(t)$).

The first step was to find the equations of motion for pitch in a model rocket, which are the following [1][2]:

$$\begin{aligned} \sum F_x &= m\dot{U} \\ \sum F_z &= m\dot{W} \\ \sum M_x &= \dot{Q}\, I_y \end{aligned} \qquad (1)$$

Note there are not vector derivatives. This arises from the simulation method, where the velocity is considered a global magnitude; therefore, the local accelerations are transformed into global accelerations, and then integrated into the global velocity. This transformation is done in each timestep of the simulation, using a new θ each time.

---

[1] Aerospace Engineering student at "Universidad Tecnológica Nacional – Regional Haedo". Buenos Aires, Argentina. guidodipasquo@gmail.com

[2] In this project, positive elevator (TVC angle) induces positive pitch angle.

The rotational matrices are:

$$R_B^G = \begin{pmatrix} \mathrm{Cos}(\theta) & Sin(\theta) \\ -Sin(\theta) & \mathrm{Cos}(\theta) \end{pmatrix}$$
$$R_G^B = {R_B^G}^T = \begin{pmatrix} \mathrm{Cos}(\theta) & -Sin(\theta) \\ Sin(\theta) & \mathrm{Cos}(\theta) \end{pmatrix} \quad (2)$$

Where R_B^G transforms a vector from the body frame into the global frame, and R_G^B transforms a vector from the global frame into the body frame.

Expanding the left-hand side of Eq.1:

$$\sum F_x = T\ Cos(\delta_e) + m\ g_x - q\ S\ CA_\alpha$$
$$\sum F_z = T\ Sin(\delta_e) + m\ g_z + q\ S\ N_\alpha \quad (3)$$
$$\sum M_x = T\ Sin(\delta_e)\ (x_t - x_{cg}) + \ q\ S\ N_\alpha\ (x_a - x_{cg})$$

Where S is the cross-sectional area of the rocket, q the dynamic pressure, m its mass and Iy its moment of inertia. θ is the pitch angle, α the angle of attack. δ_e is the TVC angle (in radians). N_α is the normal force coefficient, CA_α is the axial force coefficient, and g the gravitational acceleration in the body frame. x_{cg} is the distance from the tip of the nose cone to the center of mass, x_a to the center of pressure and x_t to the TVC mount.

Actuator Dynamics

The actuator dynamics can have a detrimental effect in stability, so they must be modeled. The transfer function of the servo is extracted from experimental data from [4]. Because the actuator is non-linear, the transfer function proposed is:

$$\frac{K}{s^2 + J.s + K} \quad (4)$$

Where K and J vary with the relative magnitude of the inputs.

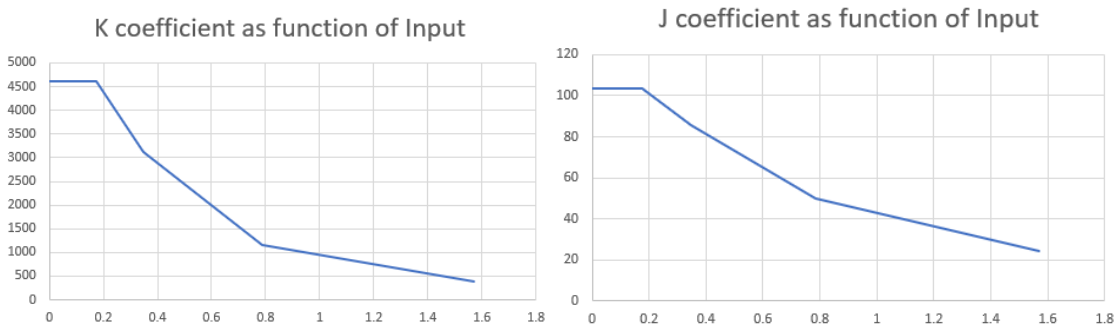

Figure 2 – K and J as a function of relative input (radians).

The relative magnitude of the inputs refers to the difference between an input and the current position of the servo, is easy to note that an input from 40º to 45º is equivalent an a 5º input (or "*u_delta*") in the model.

The resulting State Space Model is:

$$\begin{pmatrix}\dot{\delta}\\ \ddot{\delta}\end{pmatrix} = \begin{pmatrix}0 & 1\\ -K & -J\end{pmatrix}\begin{pmatrix}\delta\\ \dot{\delta}\end{pmatrix} + \begin{pmatrix}0\\ K\end{pmatrix}u$$

where C is assumed to be the Identity matrix, D is zero and K and J are calculated with a trend line that spans from 10º to 90º. The discretization is done in the program. The simulated response was slightly faster than the experimental, so it was adjusted by multiplying *"u_delta"* by a factor of 1.3 to simulate the servo alone, or by 1.75 to simulate the TVC mount inertia and its effect on the actuator dynamics.

III. PID TUNER CODE

The details of the code will not be shown, instead the overall behavior is analyzed, followed by key points that must be taken into account when translating the gains obtained to a flight computer.

First, the block diagram of the controller used is analyzed:

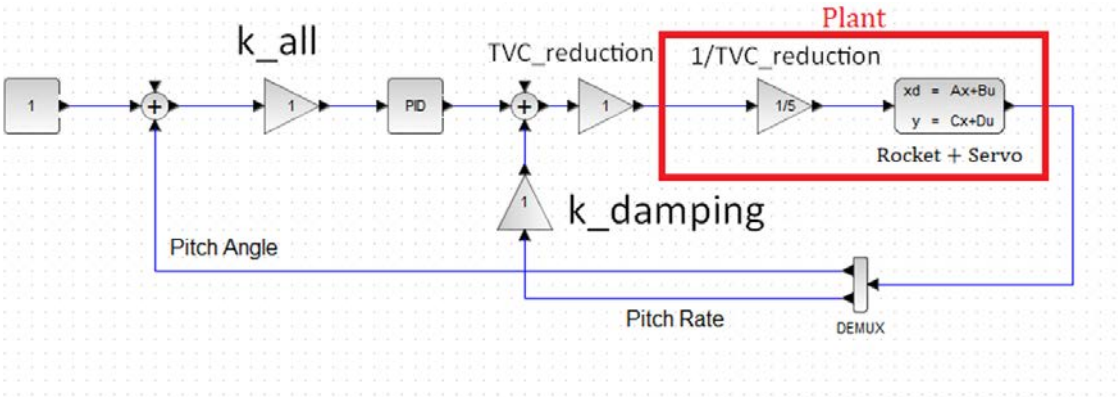

Figure 3 – Block diagram of the controller and model.

The user can choose between a step, ramp or constant zero input, based on the stability requirements for the system. This is done because non linearities like the actuator or maximum deflection angle can destabilize a stable system based on the input.

After the program begins, the local accelerations are computed. The global velocity is transformed into the local frame, the accelerations are integrated, and the local velocity is once again transformed into the global frame. Once the new states and outputs are calculated, they are treated as readings from sensors and are manipulated by the PID, and a new input is computed. This process is repeated at each step of the simulation, ensuring that the flight is simulated from liftoff to burnout.

The reader should be careful when using the gains obtained by this method, since the flight computer code must match that of the simulator. Key points that should not be overlooked are:

- All angles are in radians, if not they are immediately converted.
- The control scheme is found in the function *control_theta* and the PID scheme in *PID*.
- The output of the controller (*u_controler*) is multiplied by the TVC reduction ratio before being sent to the actuator, this must be implemented in the user's flight computer.

- If the user decides to implement anti-windup, its structure must be carefully emulated to ensure the same effects, mainly the saturation at the PID output, as well as after the *k_damping* feedback.
- As for V0.4 of the simulator, where mass and moment of inertia are kept constant, the user should verify the gains obtained against the extremes of these parameters to ensure stability.
- The Torque Based Controller implemented simply multiplies the output of the controller by: $\frac{Reference\ Thrust}{Actual\ Thrust}$, and it is followed by another saturation to prevent over-deflection of the TVC mount. Is easy to see that if the Actual Thrust is lower than the reference thrust, the gain is greater than the unity, to produce the same torque output that the reference thrust would have produced.

The thrust curve implemented by default is:

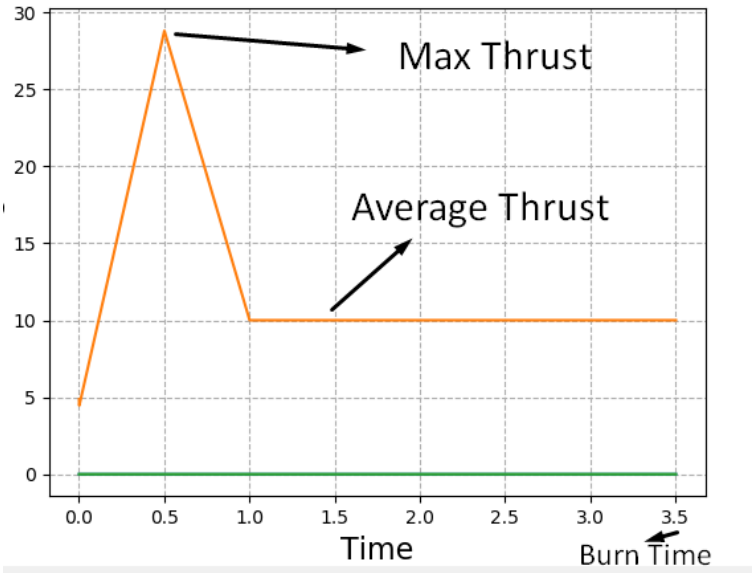

Figure 4 – Thrust Curve.

- The $x_{cp} = f(\alpha)$ default is:

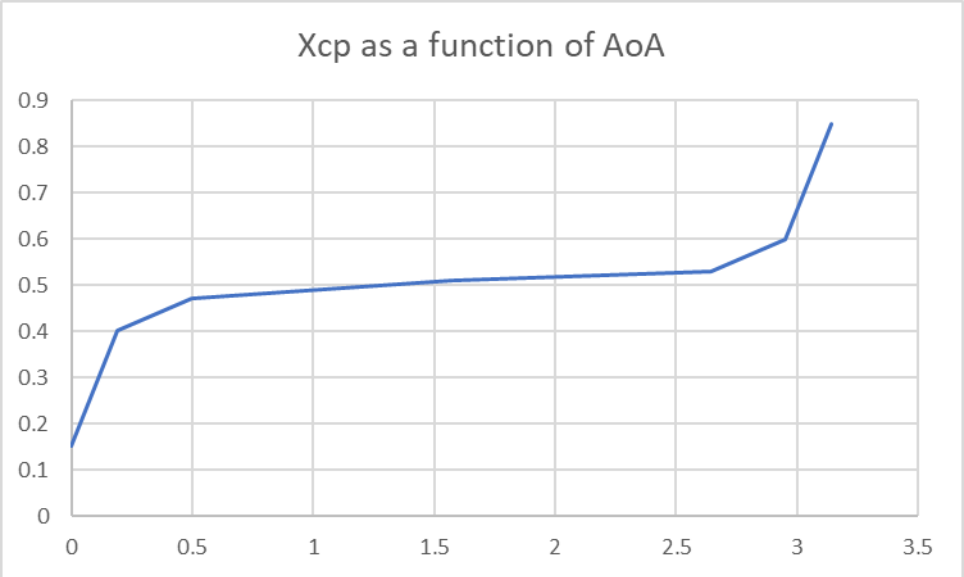

Figure 5 – Default movement of the CP, based on a 1m long, finless rocket.

IV. VERIFICATION

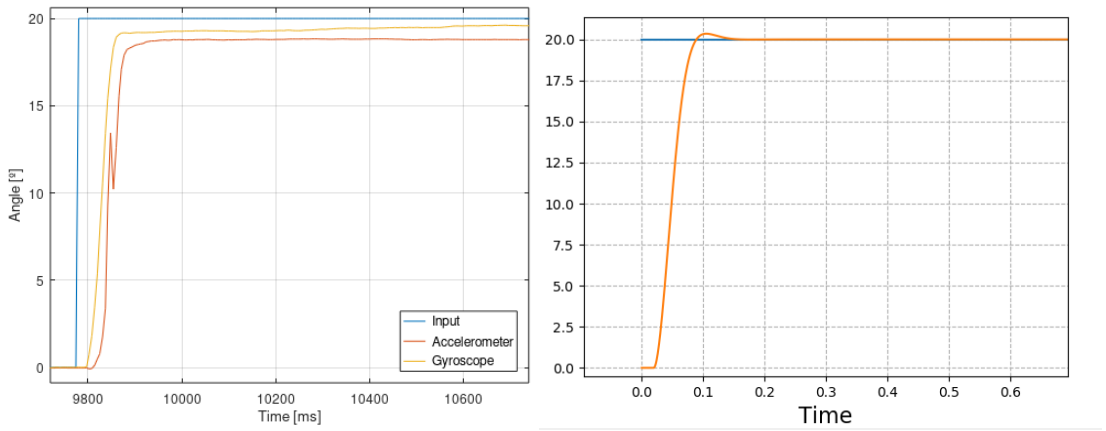

Figure 6 – Verification of the servo. Left is the experimental data from [4] and Right is the simulation of the program.

The error was not modeled due to its random nature [4], the rise time in both cases is 0.069 seconds. The sample time delay can be seen in both plots, the main difference in the responses is that the simulation slightly overshoots.

Due to the author not having real flight data of TVC controlled model rockets, the simulation could not be verified.

V. CONCLUTIONS

The mathematical model of a model rocket has been developed and introduced into a simulator. The plug and play methodology has been achieved, due to the user only having to plug his rocket's parameters and play with the gains.
The 3D animations facilitate the analysis of the flight and the tuning of the PID. Consequently, they are expected to improve the usability of the simulator.

VI. REFERENCES

[1] - Automatic Control of Aircraft and Missiles - John H. Blakelock

[2] - OpenRocket technical documentation - Sampo Niskanen

[3] - https://dsp.stackexchange.com/questions/45042/bilinear-transformation-of-continuous-time-state-space-system

[4] – SG90 Characterization. Guido di Pasquo, Alexis M. Caratozzolo, Tomás Ziroldo
https://www.youtube.com/watch?v=fnq0u8TrlA4

[5] – Aerodynamic Roll Control in Model Rockets – Me and the boys again
https://www.youtube.com/watch?v=PzdltECLooE

[6] – Everyone that posted tutorials on internet.