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

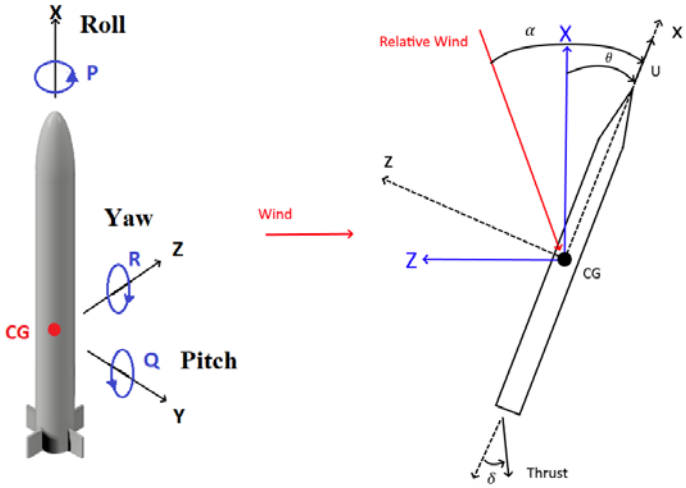

Figure 1 – Diagram of a model rocket, where Q is the pitch rate ($\dot{\theta}(t)$).

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

$$\frac{mU}{Sq}\dot{u}(t) - C_{x_u}u(t) - C_{x_\alpha}\alpha(t) - C_w\mathrm{Cos}(\Theta)\theta(t) = 0$$

$$-C_{z_u}u(t) + \left(\frac{mU}{Sq} - \frac{d}{2U}C_{z_{\dot{\alpha}}}\right)\dot{\alpha}(t) - C_{z_\alpha}\alpha(t) + \left(-\frac{mU}{Sq} - \frac{d}{2U}C_{z_q}\right)\dot{\theta}(t) - C_w\mathrm{Sin}(\Theta)\theta(t) = C_{z_{\delta_e}}\delta_e(t) \quad (1)$$

$$-\frac{d}{2U}C_{m_{\dot{\alpha}}}\dot{\alpha}(t) - C_{m_\alpha}\alpha(t) + \frac{I_y}{Sqd}\ddot{\theta}(t) - \frac{d}{2U}C_{m_q}\theta(t) = C_{m_{\delta_e}}\delta_e(t)$$

[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.

And are be approximated to [1]:

$$-C_{z_\alpha}\alpha(t) + \left(-\frac{mU}{Sq}\right)\dot{\theta}(t) - C_w \text{Sin}[\Theta]\theta(t) = C_{z_{\delta_e}}\delta_e(t)$$
$$-C_{m_\alpha}\alpha[s] + \frac{I_y}{Sqd}\ddot{\theta}(t) = C_{m_{\delta_e}}\delta_e(t) \quad (2)$$

The State-Space model is obtained:

$$\begin{pmatrix}\dot{\theta}\\ \ddot{\theta}\\ \dot{\alpha}\end{pmatrix} = \begin{pmatrix}0 & 1 & 0\\ 0 & 0 & \frac{C_{m_\alpha} * Sqd}{I_y}\\ \frac{C_w \text{Sin}[\Theta] * Sq}{mU} & 1 & \frac{C_{z_\alpha}}{(\frac{mU}{Sq})}\end{pmatrix} * \begin{pmatrix}\theta\\ \dot{\theta}\\ \alpha\end{pmatrix} + \begin{pmatrix}0\\ \frac{C_{m_{\delta_e}} * Sqd}{I_y}\\ \frac{C_{z_{\delta_e}} * Sq}{mU}\end{pmatrix} * \delta_e \quad (3)$$

Where S is the cross-sectional area of the rocket, q the dynamic pressure, d its diameter, U the longitudinal velocity, m its mass and Iy its moment of inertia. θ is the pitch angle and its temporal derivatives, Θ is the initial pitch angle, α the angle of attack. δ_e is the TVC angle (in radians). The rest of parameters are summed up in Table 1.

Parameter	Definition	Formula
C_w	Gravity.	$-\frac{mg}{Sq}$
C_{m_α}	Moment variation with angle of attack.	$\frac{C_{N_\alpha}(x_{cg} - x_a)}{Sqd}$
C_{z_α}	Normal Force variation with alpha.	$-C_{N_\alpha}$
$C_{m_{\delta_e}}$	Moment variation with input angle.	$\frac{Thrust * (x_t - x_{cg})}{Sqd}$
$C_{z_{\delta_e}}$	Normal Force variation with input angle.	$\left(\frac{d}{(x_t - x_{cg})}\right) * C_{m_{\delta_e}}$

Table 1 – Aerodynamic and inertial parameters of a model rocket.

Where 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, and C_{N_α} is obtained from [2].

Knowing the State Space model, where C is assumed to be the Identity matrix, and D zero, it is discretized by Tustin's method [3]:

$$\mathbf{A_d} = \left(\frac{2}{T}\mathbf{I} - A\right)^{-1} \cdot \left(\frac{2}{T}\mathbf{I} + A\right)$$

$$\mathbf{B_d} = \left(\frac{2}{T}\mathbf{I} - A\right)^{-1} \cdot \mathbf{B} \qquad (4)$$

$$\mathbf{C_d} = \mathbf{C} \cdot (\mathbf{A_d} + \mathbf{I})$$

$$\mathbf{D_d} = \mathbf{C} \cdot \mathbf{B_d} + \mathbf{D}$$

The discretization is done in the program using Numpy.

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:

$$\frac{K}{s^2 + J.s + K} \qquad (5)$$

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

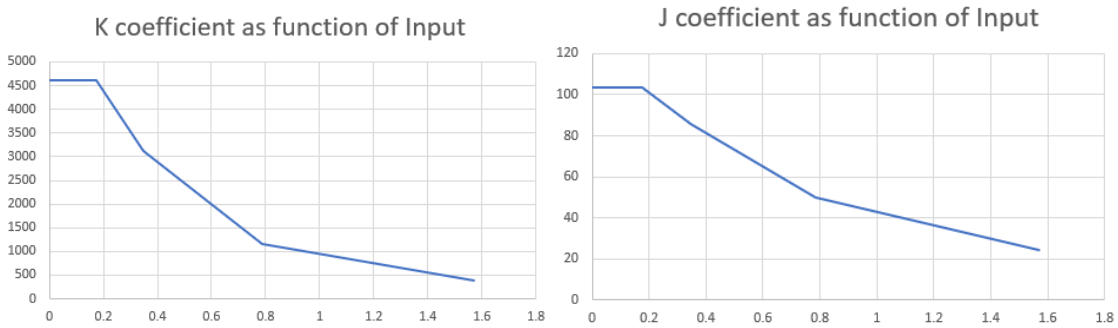

Figure 2 – K and J as a function of relative input (radians).

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. As before, 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 have to be taken into account when translating the gains obtained to a flight computer.

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

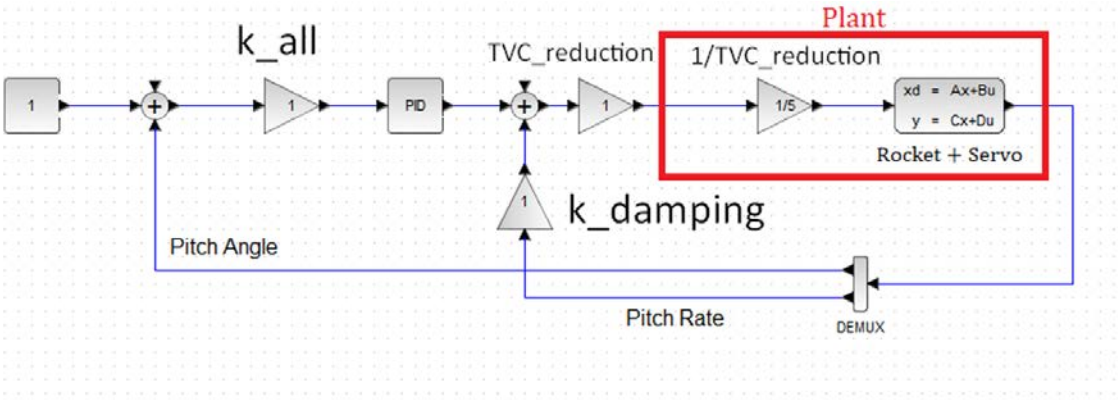

Figure 3 – Block diagram of the controller and model.

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, it updates parameters like U, q, C_{N_α} based on the data from the rocket (as thrust and mass) and recalculates the model's state space matrices. Disturbances as wind are computed, and their effects introduced as an equivalent input bypassing the actuator. Once the new states and outputs are calculated, they are treated as readings from sensors and are manipulated by the gains and 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 divided by 57.3.
- The control scheme is found in the function *control_tita* 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 V0.3.2 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 thrust curve implemented by default is:

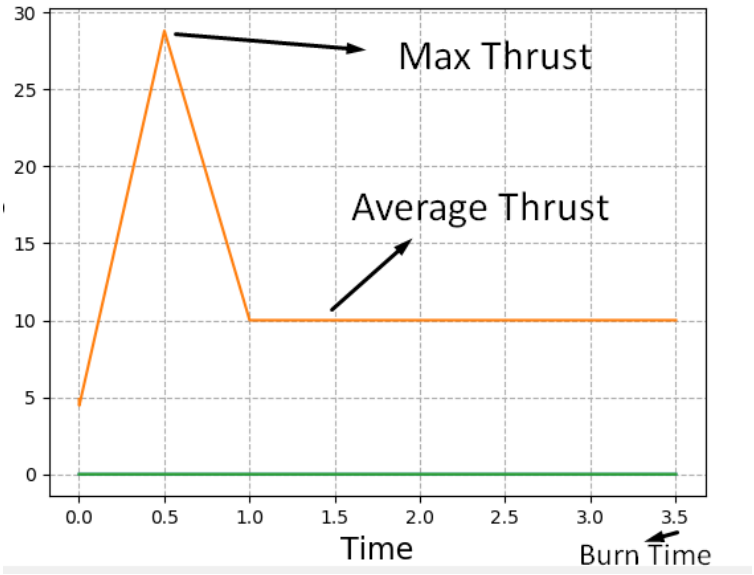

Figure 4 – Thrust Curve.

IV. VERIFICATION

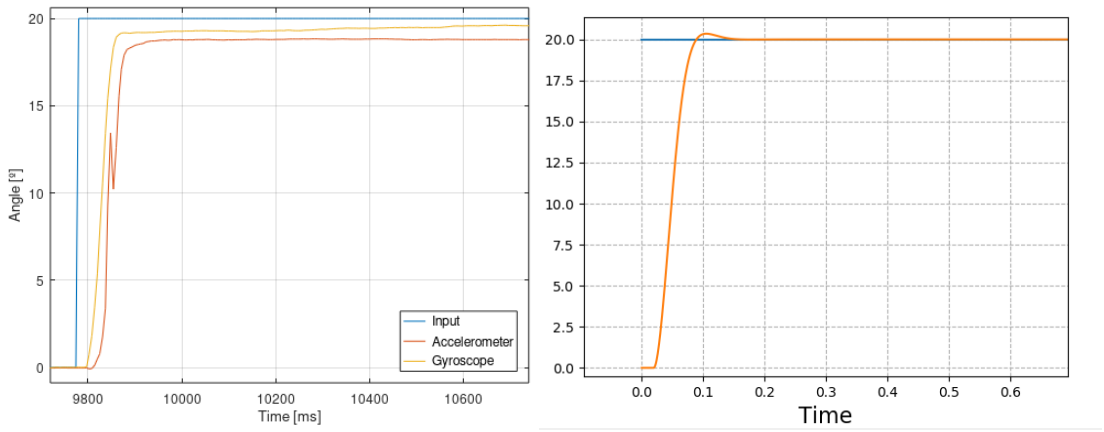

Figure 5 – Verification of the servo. Left is the experimental data from [4] and Right is the simulation of the program.

The error was not modeled due to its random nature ([4]), the rise time in both cases is 0.069 seconds. The sample time 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 verification of the simulation is done against roll control tests extracted from [5].

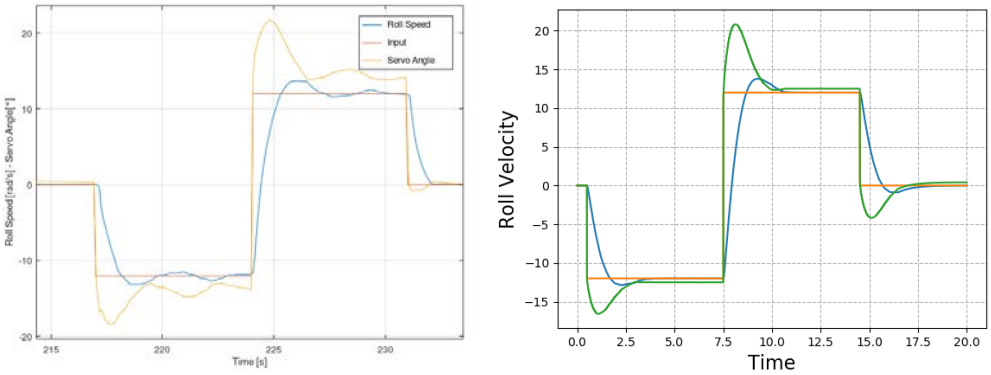

Figure 6 – Simulation procedure verification. Left is real wind tunnel data; Right is the simulation.

The response is similar to the real one, other industry software like X-Cos give results that closely match our program. Comparison against real world data not only tests the simulation procedure, but mainly the modeling precision, therefore, the simulation is considered successful.

V. CONCLUTIONS

The mathematical model of a model rocket has been developed, discretized 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.

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 me bois again
https://www.youtube.com/watch?v=PzdltECLooE

[6] – Everyone that posted tutorials on internet.