Controller-Observer implementation for Star-tracker Spaceship

Rohen A. Agarwal*
University of Illinois at Urbana-Champaign, Champaign-Urbana, Illinois, 61801

This report outlines my process of designing, implementing, and testing a controller I designed for a star-tracker spaceship. The aim of the controller is to be able to provide adequate torque to each of the wheels such that the spaceship is able to track the star (at random initial positions) throughout the simulation time period. To make this more challenging random "shooting stars" will interfere with the spaceship disturbing it's orientation but the controller must be able to compensate for the change. In my process of designing a controller for this purpose, along with checking for controllability using the controllability matrix, I also had to use the observability matrix to check for observability. Unlike previous projects where we knew the state of the body exactly at all times, we don't know the exact state solution this time and so we will have to work with state estimates from a sensor model.

I. Nomenclature

A	=	state matrix
B	=	input matrix
C	=	output matrix
L	=	injection matrix
K	=	gain matrix
x	=	state in state space model
$\hat{x}$	=	state estimate in state space model
u	=	input in state space model
$\hat{u}$	=	input estimate in state space model
e	=	error
Φ	=	roll angle (rad)
Θ	=	pitch angle (rad)
Ψ	=	yaw angle (rad)
ω_x	=	angular velocity about x-axis (rad/s)
ω_y	=	angular velocity about y-axis (rad/s)
ω_z	=	angular velocity about z-axis (rad/s)
τ_1	=	torque on wheel 1 (N.m)
τ_2	=	torque on wheel 2 (N.m)
τ_3	=	torque on wheel 3 (N.m)
τ_4	=	torque on wheel 4 (N.m)
α	=	right ascension of star (rad)
δ	=	declination of star (rad)

II. Introduction

The aim of my controller is to orient the spaceship (on which is mounted a camera) such that it is able to track the position of 3 stars in space over a period of time. There will be shooting stars following along random and unpredictable paths in space that may cause disturbance to the spaceship (and camera) when hit. As normally done, we would form a linearised model that can be represented as:

$$\dot{x} = Ax + Bu \tag{1}$$

$$y = Cx + Du \tag{2}$$

*Junior, Aerospace Engineering

However, in this case, we do not know what the exact state variables (x) are. To overcome this obstacle, we can use a one-step finite difference method to approximate the state variables in time. This then becomes and Initial Value Problem (IVP). One suitable one-step finite difference method we can use is the Forward Euler (FE) method. Implementing FE onto our problem we get:

$$\hat{x} = \hat{x}_0 + dt(\dot{\hat{x}}) \tag{3}$$

$$x = x_0 + dt(\dot{x}) \tag{4}$$

$$\dot{x} = Ax - BK\hat{x} \tag{5}$$

$$\dot{\hat{x}} = A\hat{x} + Bu - L(C\hat{x} - y) \tag{6}$$

As we can see in [6], we used the state estimates $\hat{x}$ and $\dot{\hat{x}}$ which we find using the FE method. dt is the small time step that we use to advance in time.

III. Requirements

I want my controller to

1) be able to track the stars for at least 8 seconds out of the 10
2) be able to stabilize the spaceship when hit by shooting stars
3) have a maximum mean squared error of 1 units for rpy

IV. Verification

I will design and implement my controller 3+1000 times. The first 3 times with GUI to see how the simulation works while the rest 1000 times to use the data to find and plot histograms. The 3 simulations will be able to verify requirements 1 and 2 while the 1000 simulations with the histograms will help in verifying requirement 3.

V. System

Figure 1 shows the system we start out with. The AE353 course website provides us with this system initialised. The blue body is the spaceship with 4 equally spaced out wheels on it represented in orange. These wheels spin when torque is applied causing the spaceship to change orientation using the principal of conservation of angular momentum. The three yellow balls represent the three stars that our spaceship camera (on one side of the spaceship) must track.

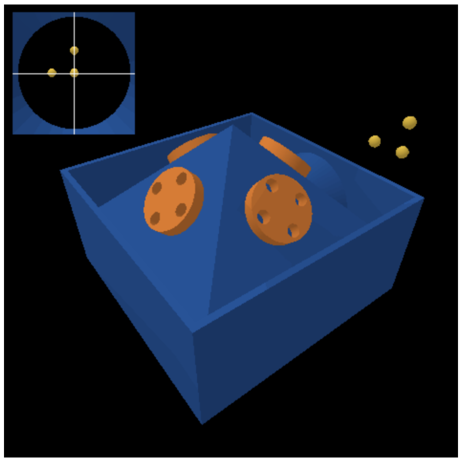


Fig. 1 System in pybullet

A. Equations of Motion

The governing equations of motions for our system were provided by Prof. Bretl and are shown in the "GenerateResults.ipynb" file as matrix f. I defined matrices f and g as:

$$f = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{\omega_x} \\ \dot{\omega_y} \\ \dot{\omega_z} \end{bmatrix} g = \begin{bmatrix} y_{star1} \\ z_{star1} \\ y_{star2} \\ z_{star2} \\ y_{star3} \\ z_{star3} \end{bmatrix} \tag{7}$$

B. Equilibrium values and A, B, and C matrices

I defined my equilibrium values such that

$$\phi_e = \theta_e = \psi_e = w_{x_e} = w_{y_e} = w_{z_e} = \tau_{1_e} = \tau_{2_e} = \tau_{3_e} = \tau_{4_e} = 0. \tag{8}$$

By taking the jacobian of matrix f with the angles and angular velocities I found the matrix A and by taking the jacobian of matrix f with the wheel torques I found the matrix B. To find matrix C, I computed the jacobian of matrix g with respect to the angles and angular velocities.

$$A = \frac{\partial f}{\partial x}\bigg|_{x_{equi}, u_{equi}} \quad B = \frac{\partial f}{\partial u}\bigg|_{x_{equi}, u_{equi}} \quad C = \frac{\partial g}{\partial x}\bigg|_{x_{equi}, u_{equi}} \tag{9}$$

My final matrices A, B, and C were:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -0.05241358 & 0.05241358 & 0 & 0 \\ 0 & 0 & -0.05241358 & 0.05241358 \\ 0 & 0 & -0.05241358 & 0.05241358 \end{bmatrix} \tag{10}$$

$$C = \begin{bmatrix} 0 & 0 & -2.625 & 0 & 0 & 0 \\ 0 & 2.625 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2.684 & 0 & 0 & 0 \\ -0.396 & 2.625 & 0 & 0 & 0 & 0 \\ 0.396 & 0 & -2.625 & 0 & 0 & 0 \\ 0 & 2.684 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{11}$$

C. Controllability

Now that we have the necessary matrices to define our state space model, It is critical to determine if the system is even controllable or not. To do this, we need to find the controllability matrix W that is defined as

$$W = \begin{bmatrix} B & AB & A^2B & A^3B & \dots & A^{n-1}B \end{bmatrix} \tag{12}$$

A system is controllable if the rank of the controllability matrix W is equal to the number of states in the system. In other words, the rank of matrix W must equal the number of rows in matrix A. From above, we can see that A is a (6x6) matrix and therefore has 6 states. Computing the rank of the controllability matrix W using python's np.linalg.matrix rank function, I found the rank of W to equal 6. The system is controllable! (Agarwal, DP2, 2021)

D. Observability

Since we are dealing with sensor data to give us approximates of the state of the spaceship in time, we need to check observability. The observability matrix is defined as:

$$O = \begin{bmatrix} C \\ CA \\ CA^2 \\ CA^3 \\ \vdots \\ CA^{n-1} \end{bmatrix} \tag{13}$$

A system is observable if the rank of the observability matrix O is equal to the number of states in the system. In other words, the rank of matrix O must equal the number of rows in matrix A. We know that A is a (6x6) matrix and therefore has 6 states. Computing the rank of the observability matrix O using python's np.linalg.matrix rank function, I found the rank of O to equal 6. The system is observable!

E. Using LQR to find stable gain matrix and injection matrix

The reason for choosing LQR as the method to find suitable gain matrix K and injector matrix L is because not only is it easy to implement, but is also efficient since the the entire idea of LQR is to reduce the "cost" and yet make the system attain equilibrium the fastest (Agarwal, DP2, 2021). The LQR cost function is defined as:

$$cost = \int_0 x(t)^T Qx(t) + u(t)^T Ru(t)\, dt \tag{14}$$

where Q and R are the weights associated with x(t) and u(t) respectively. Each entry in x(t) and u(t) are weighted by their corresponding entry in Q and R (Agarwal, DP2, 2021). In python I formed two LQR functions, one for the matrix K and one for L. My final matrices K and L were:

$$K = \begin{bmatrix} 7.071e-1 & 6.921e-16 & -5.000e-1 & -3.740 & 3.638e-15 & -2.635 \\ 7.071e-1 & 8.972e-16 & -5.000e-1 & 3.740 & 4.275e-15 & -2.635 \\ -8.784e-17 & -7.071e-1 & -5.000e-1 & -4.611e-17 & -3.740 & -2.635 \\ 1.582e-16 & 7.071e-1 & -5.000e-1 & 5.908e-16 & 3.740 & -2.635 \end{bmatrix} \tag{15}$$

$$L = \begin{bmatrix} -0.871 & 0.871 & -0.891 & -1.924 & 1.924 & 0.891 \\ -0.042 & 1.152 & -0.043 & 1.020 & 0.089 & 1.178 \\ -1.152 & 0.042 & -1.178 & -0.089 & -1.020 & 0.043 \\ -0.444 & 0.444 & -0.455 & -1.046 & 1.046 & 0.455 \\ -0.021 & 1.013 & -0.021 & 0.946 & 0.046 & 1.036 \\ -1.013 & 0.021 & -1.036 & -0.046 & -0.946 & 0.021 \end{bmatrix} \tag{16}$$

Using K and L, I defined my controller as:

$$F_K = A - BK \tag{17}$$

$$F_L = A - LC \tag{18}$$

F. Stability

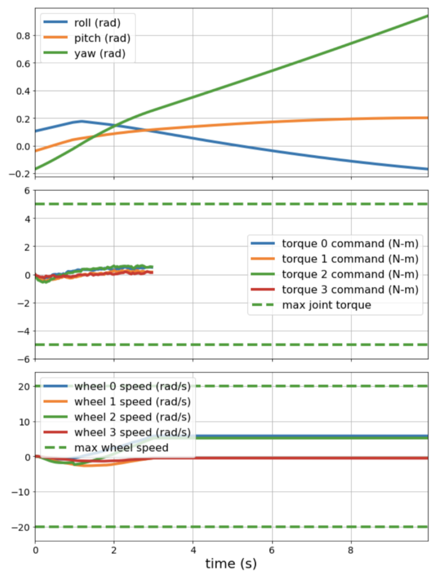
We ascertained that the system we have is controllable as well as observable but we still need to test stability. A stable system is one that, when given a definite, bounded input, returns a bounded output. Such a system is called "asymptotically stable". Mathematically, if all the real parts of the eigenvalues of a matrix are negative, the system is asymptotically stable. It turns out that all the eigenvalues for both K and L have negative real parts which is desired.

Now I have a stable control in theory, but I must implement it and analyse the results to see if my controller met the requirements or not.

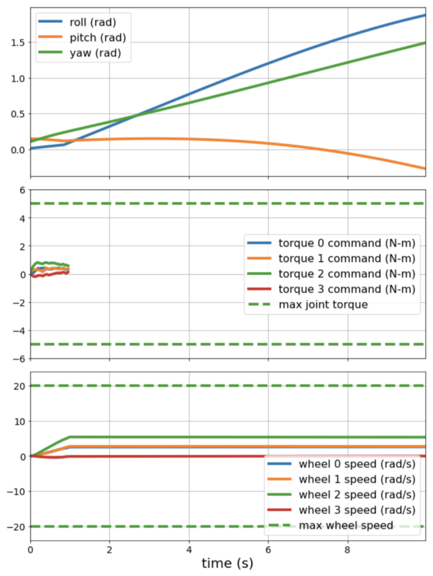
VI. Results

A. Implementation

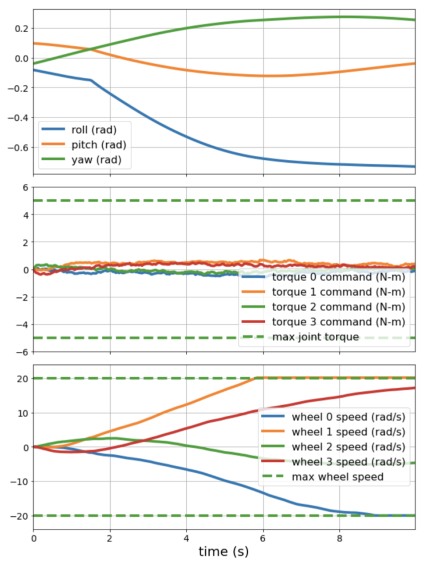
I implemented my controller three times separately and plotted the results. Each time, the random condition, that is the path and velocity of the shooting stars were different. My results for the three simulations are shown below.



(a) Attempt 1

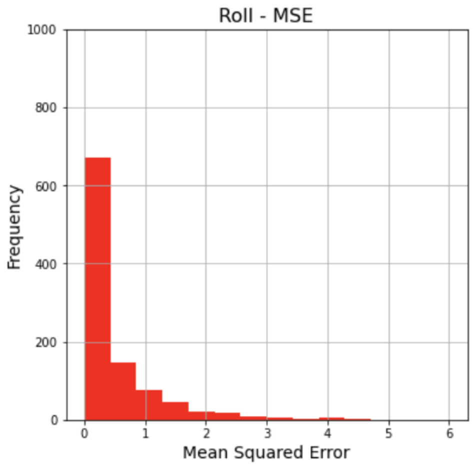


(b) Attempt 2

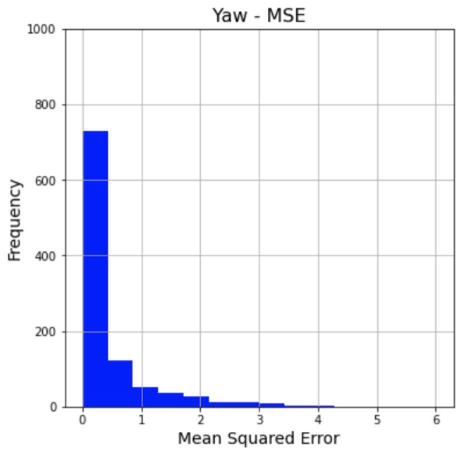


(c) Attempt 3

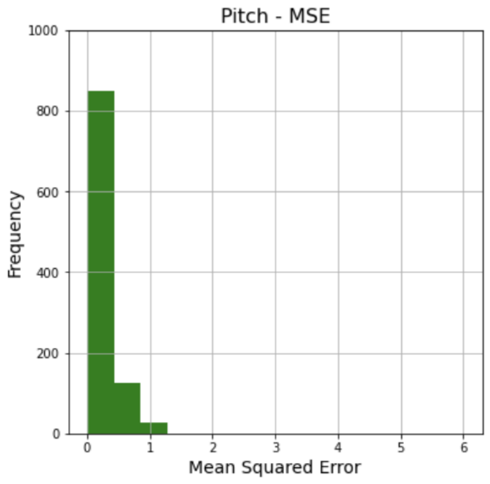
Fig. 2 Attempts 1, 2, 3



(a) Roll - MSE



(b) Yaw - MSE



(c) Pitch - MSE

Fig. 3 Roll, Pitch, Yaw mean squared error

As seen, each time the controller acts differently. My controller is unable to track the stars accurately in the given time duration. Therefore, my controller is not robust. To graphically represent my results, I performed the simulation thousand times in a loop and plotted the Mean Squared Errors for each of the angles: Roll, Pitch, and Yaw.
The mean mse values for rpy were:

Angle	Mean (rad)	Median (rad)	Std (rad)
Roll	0.472	0.207	0.662
Pitch	0.188	0.083	0.237
Yaw	0.391	0.081	0.656

VII. Conclusion

I began my process by linearising my system to get a state space model, and found matrices A, B, and C, after which I checked for controllability, observability. I then found gain matrix K and injection matrix L using the very efficient LQR method. I checked for stability and found that both K and L matrices are stable. Next I implemented my control several times and plotted the mean squared error for the roll, pitch and yaw for each iteration. I came to the conclusion that my controller does not work perfectly since it is unable to track the stars during the simulation period. While it is able to meet requirement 3 (mean errors are all < 1), it is unable to meet requirements 1 and 2 consistently.

Acknowledgments

I would like to thank professor Timothy Bretl for the very helpful lectures, example in-class codes as well as the homework that helped me in coding my controller. I also want to thank professor Timothy Bretl, for the AE353 course website off which I used the EOMs pre derived for me, as well as the images and description of the system. Additionally, his SpacecraftDemo file as well as the DeriveEOM file were extremely helpful in my own implementation of my controller. I also want to credit Jushan Chen, and Harry Zhao because I took inspiration from their code to use in my own. Their code helped me find stable gain matrix K and injection matrix L in a really efficient way. I also want to thank all my peers for the continuous fruitful discussions we had on the topics as well as the project on Campuswire or Discord which helped me understand the content and do the project better.

References

[1] Bretl, Timothy.
`https://tbretl.github.io/ae353-sp21/projectsdesign-project-3-spacecraft-with-star-tracker`

[2] Agarwal, Rohen (2021).
Design Project 2 - Differential Drive Robot