Simulating Quadrotor Dynamics

Jett Vanbrocklin

February 3, 2026

1 Introduction

The motion of a quadrotor can be defined by a union of linear and attitude dynamics, kinematics, and their relationship. The directive equations for a simulator can be found in [1].

This Laboratory assignment breaks way for a high-fidelity quadrotor dynamics simulator in Matlab. This model is derived from Newtonian Dynamics and aims to determine the motion of a quadrotor given a set of parameters and motor-speeds over time. The report follows the theoretical analysis, specific implementation, and results of experimentation.

2 Theoretical Analysis

2.1 Relation of Total Rotation Angle and the Trace of a Rotation Matrix

Euler's formula expresses a rotation matrix in terms of an axis of rotation $\hat{a}$ and a rotation angle ϕ:

$$R(\hat{a}, \phi) = cos\phi I_{3x3} + (1 - cos\phi)\hat{a}\hat{a}^T - sin\phi[\hat{a}\times] \tag{1}$$

Let ϕ be the total rotation angle and the trace of R be written $tr(R)$. To show that

$$tr(R) = 1 + 2cos\phi \tag{2}$$

one can utilize properties of the trace function. First, the trace operator can be used on both sides of equation (1).

$$tr(R(\hat{a}, \phi)) = tr(cos\phi I_{3x3} + (1 - cos\phi)\hat{a}\hat{a}^T - sin\phi[\hat{a}\times])$$

Trace is a linear operator, therefore the right side can be separated.

$$tr(R(\hat{a},\phi)) = tr(cos\phi I_{3x3}) + tr(1-cos\phi)tr(\hat{a}\hat{a}^T) - tr(sin\phi)tr([\hat{a}\times])$$

By definition of trace,

$$tr(cos\phi I_{3x3}) = 3cos\phi$$

and

$$tr(1-cos\phi) = 1-cos\phi$$

then,

$$tr(R(\hat{a},\phi)) = 3cos\phi + (1-cos\phi)tr(\hat{a}\hat{a}^T) - tr(sin\phi)tr([\hat{a}\times])$$

Next, let $tr(\hat{a}\hat{a}^T) = \hat{a}^T\hat{a} = 1$ **and let** $tr([\hat{a}\times]) = 0$**. Then,**

$$\begin{aligned} tr(R(\hat{a},\phi)) &= 3cos\phi + (1-cos\phi)tr(\hat{a}\hat{a}^T) - tr(sin\phi)tr([\hat{a}\times]) \\ &= 3cos\phi + (1-cos\phi)tr(\hat{a}\hat{a}^T) - tr(sin\phi)tr([\hat{a}\times]) \\ &= 3cos\phi + (1-cos\phi)(1) - tr(sin\phi)(0) \\ &= 3cos\phi + (1-cos\phi) - 0 \\ &= 1 + 2cos\phi \end{aligned}$$

Therefore, the total rotation angle ϕ **can be related to the trace of** R **by**

$$tr(R(\hat{a},\phi)) = 1 + 2cos\phi$$

2.2 Time Derivative of a Rotation Matrix

Let $C(t) = R(\hat{a},\phi(t))$**,** $\mathbf{w} = w\hat{a}$**, and** $\phi(t) = wt$**. Through explicit differentiation of equation** (1)**, show that**

$$\frac{d}{dt}C(t) = -[w\times]C(t) \tag{3}$$

First, differentiate both sides of equation (1)

$$\begin{aligned} \frac{d}{dt}C(t) &= \frac{d}{dt}cos\phi I + \frac{d}{dt}(1-cos\phi)\hat{a}\hat{a}^T - \frac{d}{dt}sin\phi[\hat{a}\times] \\ &= (\frac{d}{dt}cos\phi)I + (\frac{d}{dt}(1-cos\phi))\hat{a}\hat{a}^T - (\frac{d}{dt}sin\phi)[\hat{a}\times] \\ &= -sin\phi(\frac{d\phi}{dt})I + sin\phi(\frac{d\phi}{dt})\hat{a}\hat{a}^T - cos\phi(\frac{d\phi}{dt})[\hat{a}\times] \\ &= (\frac{d\phi}{dt})(-sin\phi I + sin\phi\hat{a}\hat{a}^T - cos\phi[\hat{a}\times]) \end{aligned}$$

Next, lets expand $-[\mathbf{w}\times]C(t)$.

$$\begin{aligned}
-[\mathbf{w}\times]C(t) &= -[w\hat{a}\times]C(t) \\
&= -w[\hat{a}\times]C(t) \\
&= -\frac{d\phi}{dt}[\hat{a}\times]C(t) \\
&= -\frac{d\phi}{dt}[\hat{a}\times](cos\phi I + (1-cos\phi)\hat{a}\hat{a}^T - sin\phi[\hat{a}\times]) \\
&= -\frac{d\phi}{dt}(cos\phi[\hat{a}\times] + (1-cos\phi)\hat{a}\hat{a}^T[\hat{a}\times] - sin\phi[\hat{a}\times][\hat{a}\times]) \\
&= -\frac{d\phi}{dt}(cos\phi[\hat{a}\times] + -sin\phi(\hat{a}\hat{a}^T - I)) \\
&= \frac{d\phi}{dt}(-sin\phi I + sin\phi\hat{a}\hat{a}^T - cos\phi[\hat{a}\times])
\end{aligned}$$

Therefore, $\frac{d}{dt}C(t) = -[w\times]C(t)$

3 Implementation

The equations described in section one were translated into Matlab to achieve a reliable simulation for quadrotor dynamics. The following subsections aim to describe relevant segments of code that support the simulator.

3.1 Cross-Product Equivalent

The function *crossProductEquivalent(u)*, where *u* is a 3-by-1 vector, outputs *uCross*, a 3-by-3 skew-symmetric cross-product equivalent matrix. Eg. *cross(u,v) = uCross∗v* .

This function was implemented using the general equation for the skew-symmetric cross-product equivalent matrix from a 3-by-1 vector.

$$uCross = \begin{bmatrix} 0 & \text{-u(3)} & \text{u(2)} \\ \text{u(3)} & 0 & \text{-u(1)} \\ \text{-u(2)} & \text{u(1)} & 0 \end{bmatrix} \tag{4}$$

3.2 Rotation Matrix Generator

The function *rotationMatrix(*$\hat{a}$*,*ϕ) generates a rotation matrix R from a given axis of rotation $\hat{a}$ and angle ϕ, where $\hat{a}$ is a 3-by-1 unit vector.**

This function was implemented using equation (1) above.

3.3 Euler to DCM Converter

The function *euler2dcm(e)* takes a 3-by-1 vector of euler angles e and converts them to a 3-by-3 direction cosine matrix R_B^W using a 3-1-2 order.

Input vector e takes the form

$$e = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

First, the inputs must be normalized and checked for a singularity. For the 3-1-2 order implementation, the ranges are as follows.

$$\phi = [\frac{-\pi}{2}, \frac{\pi}{2}]$$

$$\theta = (-\pi, \pi]$$

$$\psi = (-\pi, \pi]$$

And under 3-1-2, the singularity exist when $\phi \geq \frac{\pi}{2}$.

Finally, using the previously described *rotationMatrix* function, rotation matrices can be defined for the inputed euler angles, where

$$R1 = rotationMatrix(\hat{a}_x, \phi)$$

$$R2 = rotationMatrix(\hat{a}_y, \theta)$$

$$R3 = rotationMatrix(\hat{a}_z, \psi)$$

and the ouput R_B^W can be found with the equation

$$R_B^W = R2 * R1 * R3$$

3.4 DCM to Euler Converter

The function *dcm2euler(R_B^W)* converts a direction cosine matrix R_B^W to Euler angle vector e where

$$e = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$$

To implement this function, equation (1) above was evaluated at arbitrary Euler angles ϕ, θ, and ψ and unit vectors e_1. e_2, and e_3 where

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad e_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

After evaluation, the determined general matrix to convert a 3-by-3 direction cosine matrix for a 3-1-2 order is the following. Let

$$C(\psi, \phi, \theta) = R(e_2, \theta) * R(e_1, \phi) * R(e_3, \psi)$$

then,

$$C(\psi, \phi, \theta) = \begin{bmatrix} c\theta c\psi + s\theta s\phi s\psi & c\theta s\psi + s\theta s\phi c\psi & -s\theta c\phi \\ -c\phi s\psi & c\phi c\psi & s\phi \\ s\theta c\psi + c\theta s\phi s\psi & s\theta s\psi - c\theta s\phi c\psi & c\theta c\phi \end{bmatrix} \tag{5}$$

where $cx = cosx$ and $sx = sinx$.

Once equation (5) was found, the inputted direction cosine matrix R_B^W (specifically, its indices) could be related to Euler angles.

But first, a singularity can be detected through $R_B^W(2,3)$. It was stated previously that a singularity exists in a 3-1-2 order if $\phi \geq \frac{\pi}{2}$. Since

$$R_B^W(2,3) = sin\phi$$

Then a singularity exists when $R_B^W(2,3) = 1$ by transition.

Next, the Euler angles are found through manipulation of the indeces of R_B^W. Derive

$$\phi = \arcsin(R_B^W(2,3))$$

$$\theta = \text{atan2}(-R_B^W(1,3), R_B^W(3,3))$$

$$\psi = \text{atan2}(-R_B^W(2,1), R_B^W(2,2))$$

Using the above equations, the Euler angles are found and structured as the output vector e.

3.5 ODE Model for Quadrotor Dynamics

The function *quadOdeFunction(t,X,omegaVec,destVec,P)* takes in numerous inputs to model quadrotor dynamics through Ordinary Differential Equations. The function is intended to be called by Matlab's ODE solver *ode45*.

The inputs can be described as follows.

t - scalar time input

X - Nx-by-1 quad state where

$$X = [r_I^T, v_I^T, R_B^I(1,1), R_B^I(2,1), ..., R_B^I(2,3), R_B^I(3,3), \omega_B^T]^T$$

with:

r_I - 3x1 position vector in I in meters

v_I - 3x1 velocity vector in I in meters/sec

RBI - 3x3 attitude matrix from I to B frame

$omegaB$ - 3x1 angular rate vector of body wrt I, expressed in B in rad/sec

omegaVec - 4x1 vector of rotor angular rates, in rad/sec

distVec - 3x1 vector of constant disturbance forces, expressed in Newtons in I

P - Structure containing relevant quad parameters and constants

And output,

Xdot - Nx by 1 time derivative of the input vector X

The implemented differential equations are sourced from [1].

After unpacking the input state X, the governing equations are used to find the derivative of position, velocity, angular velocity, and the direction cosine matrix. The determined derivatives are then structured into the output vector described above.

3.6 Quadrotor Dynamics Simulator

The function *simulateQUadrotorDynamics(S)* is the parent function of the simulator. An input S is comprised of the following elements.

tVec - Nx1 vector of uniformly-sampled time offsets in seconds

oversampFact - Oversampling factor

omegaMat - (N-1)x4 matrix of rotor speed inputs.

state0 - Initial state of the quad. with:

r - 3x1 position vector in I in meters

e - 3x1 vector of Euler angles

v - 3x1 velocity vector in I in meters/sec

$omegaB$ - 3x1 angular rate vector of body expressed in B in rad/sec

distMat - (N-1)x3 matrix of disturbance forces, expressed in Newtons in I

quadParams - Structure containing relevant quad parameters

constants - Structure containing relevant constants

The output structure P is as follows.

tVec - Mx1 vector of output sample points, in seconds

state0 - State of the quad at times in P.tVec with:

$rMat$ - Mx3 matrix of quad position at time steps in P.tVec, in meters

$eMat$ - Mx3 matrix of Euler angles at time steps in P.tVec, in radians

$vMat$ - Mx3 matrix of Euler angles at time steps in P.tVec, in meters/second

$omegaBMat$ - 3x1 Mx3 matrix of angular velocities at time steps in P.tVec, in radians/second

where N is the number of time steps and $M = (N-1)*oversampFact+1$.

The main portion of the function performs a for loop to iterate through each time step and solve the ODE at the oversampled intervals. The Matlab ode solver *ode45* numerically integrates the ODE solution and returns it back to the loop, where the outputs are stored in the numerous state-over-time matrices described in P above.

After iterating through every time step, the final value of the quad state is stored in the output structure, and the simulator returns the state-over-time matrix.

4 Results and Analysis

4.1 Euler and DCM Conversion Analysis

To test the supporting functions that convert between Euler and DCM rotation representations, one could initially show that these composition of these functions returns the input. E.g. If one were to plug in an arbitrary e where $e = [\phi, \theta, \psi]^T$ into *euler2dcm(e)*. It should be expected that inputting its return value into *dcm2euler*(R_B^W) would return the original e.

```
>> e = [0.5;1.2;0.75]

e =

    0.5000
    1.2000
    0.7500

>> euler2dcm(e)

ans =

   -0.0395    0.5739   -0.8179
   -0.5982    0.6421    0.4794
    0.8004    0.5082    0.3180

>> dcm2euler(ans)

ans =

    0.5000
    1.2000
    0.7500
```

Figure 1: Euler to DCM Test

Figure 1 shows an example chain of outputs from the Matlab command line that demonstrates this.

A second method for testing the supporting function *euler2dcm(e)*, after confirming the function *rotationMatrix*($\hat{a}, \phi$ is valid, is to compare outputs given corresponding axis-angle and Euler angles.

For example, consider axis $\hat{a} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$ **and angle** $\phi = 0.5$ **as inputs to** ***rotationMatrix***$(\hat{a}, \phi$**. Then consider Euler angle vector** $e = \begin{bmatrix} 0.5 \\ 0 \\ 0 \end{bmatrix}$ **as input to** ***euler2dcm(e)*** .

```
e1 =

     1
     0
     0

>> phi1

phi1 =

    0.5000

>> euler1

euler1 =

    0.5000
         0
         0

>> rotationMatrix(e1,phi1), euler2dcm(euler1)

ans =

    1.0000         0         0
         0    0.8776    0.4794
         0   -0.4794    0.8776


ans =

    1.0000         0         0
         0    0.8776    0.4794
         0   -0.4794    0.8776

>>
```

Figure 2: Euler to DCM related to Rotation Matrix Output

The return values of these functions should be the same DCM matrix. This is shown in Figure 2.

4.2 Results

The implementation of the numerous supporting functions described above, as well as the top level simulating loop and the ODE model,

make up the Quadrotor Dynamics Simulator. Running the *topSimulate* script, which in turn calls a function *visualizeQuad*, creates the state-over-time simulation for a given set of initial conditions and parameters and displays the motion of a quad in a 3D space over time.

		TIME 2			TIME 3996			TIME 7991		
Output	r (m)	-0.00040	0.00000	0.00000	-0.64246	0.19370	-0.06777	0.62616	0.77479	-0.27109
	e (rad)	0.00002	0.05000	0.20040	0.04998	-0.00136	1.79803	-0.00272	-0.04993	-2.88725
	v (m/s)	-0.79976	0.00005	-0.00002	0.15674	0.19394	-0.06786	1.11347	0.38788	-0.13571
	omegab (rad/s)	0.00000	0.00000	0.80000	0.00000	0.00000	0.80000	0.00000	0.00000	0.80000
Expected	r (m)	-0.00040	0.00000	0.00000	-0.64246	0.19370	-0.06777	0.62616	0.77479	-0.27109
	e (rad)	0.00002	0.05000	0.20040	0.04998	-0.00136	1.79803	-0.00272	-0.04993	-2.88725
	v (m/s)	-0.79976	0.00005	-0.00002	0.15674	0.19394	-0.06786	1.11347	0.38788	-0.13571
	omegab (rad/s)	0.00000	0.00000	0.80000	0.00000	0.00000	0.80000	0.00000	0.00000	0.80000

Figure 3: Example State Instances in Output Compared to *Ptest.mat*

In the output of the simulateQuadrotorDynamics given a set input S, the store of the quad state over time matches the values found in Ptest.mat at corresponding time steps. Some example state values are listed in Figure 3.

4.3 Experimentation: Circular Path

As part of this Laboratory, we were tasked with defining the initial and constant parameters for the quadrotor that resulted in a circular path. While I was unable to determine the proper values that resulted in success, the supporting math is as follows.

To begin, a free-body diagram of the drone in an initial position can be drawn to determine the pitch angle allowing for supplying proper force to counteract gravity and the centrifugal force. From this diagram, these values were drawn.

Let radius of circle $r = 1$ and period $T = 5$ seconds.

$$\begin{aligned} C_f &= m * w^2 * r \\ &= 0.78 * (\frac{2\pi}{5}) * 1 \\ &= 1.232 \end{aligned}$$

and,

$$\begin{aligned} mg &= 0.78 * 9.81 \\ &= 7.6518 \end{aligned}$$

Using those values,

$$\begin{aligned}\phi &= \arctan(\frac{1.232}{7.6518}) \\ &= 0.16\end{aligned}$$

$$\begin{aligned}F_T &= \sqrt{(1.232)^2 + (7.6518)^2} \\ &= 7.75\end{aligned}$$

Then, using the Euler Kinematics Equation for a 3-1-2 ordered rotation, one can find w_B.

$$\begin{aligned}w_B &= \begin{bmatrix} c\theta & 0 & -s\theta c\phi \\ 0 & 1 & s\phi \\ s\theta & 0 & c\theta c\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0.159 \\ 0 & 0 & 0.987 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1.2566 \end{bmatrix} \\ &= \begin{bmatrix} 0 \\ 0.201 \\ 1.240 \end{bmatrix}\end{aligned}$$

Next, N_B can be found. We know that w_B is constant. We also know

$$\dot{w_B} = J^{-1}(N_B - [w_B\times]Jw_B) = 0$$

since w_B is constant. Therefore,

$$\begin{aligned}N_B &= [w_B\times]Jw_B \\ &= \begin{bmatrix} 0 & -1.24 & 0.201 \\ 1.24 & 0 & 0 \\ -0.201 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.0018 & 0 & 0 \\ 0 & 0.0036 & 0 \\ 0 & 0 & 0.0047 \end{bmatrix} \begin{bmatrix} 0 \\ 0.201 \\ 1.24 \end{bmatrix} \\ &= \begin{bmatrix} 0.2844x10^{-3} \\ 0 \\ 0 \end{bmatrix}\end{aligned}$$

Then, we can substitute in and solve for rotor speeds.

$$N_B = \sum N_{iB} + r_{iB} \times F_{iB}$$

where $N_{iB} = s_i N_i z_B$, $N_i = k_N w_i^2$ and $F_{iB} = k_F w_i^2$.

Substitute,

$$N_B = \sum s_i k_N w_i^2 z_B + r_{iB} \times k_F w_i^2 z_B$$

However, once this equation is derived, a circular path is not generated after plugging values in. There is a mistake or invalid assumption somewhere in these steps, though I can not identify where it is.

5 Conclusion

A quadrotor dynamics simulator has been implemented in Matlab code through this Laboratory. To ensure the simulator could support necessary calculations, testing was done over the converting functions between DCM and Euler angles. Furthermore, testing was done over the general simulated values produced as a function of quad-state over time. A final experimentation was done with attempts of generating a circular path for the quadrotor using initial values quad parameters, however the goal was not achieved.

References

1 Humphreys, Todd. "Main." Jan. 2026.