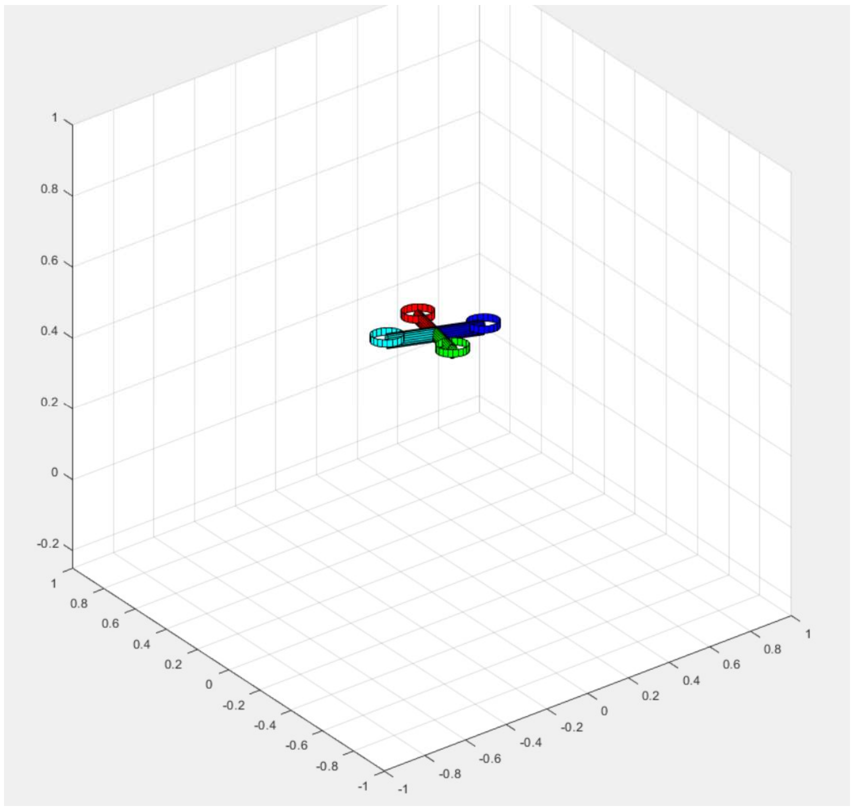
Project 1 – Drone Trajectory Planning

Jared Ticotin



Part A: Simulate the dynamics of the drone

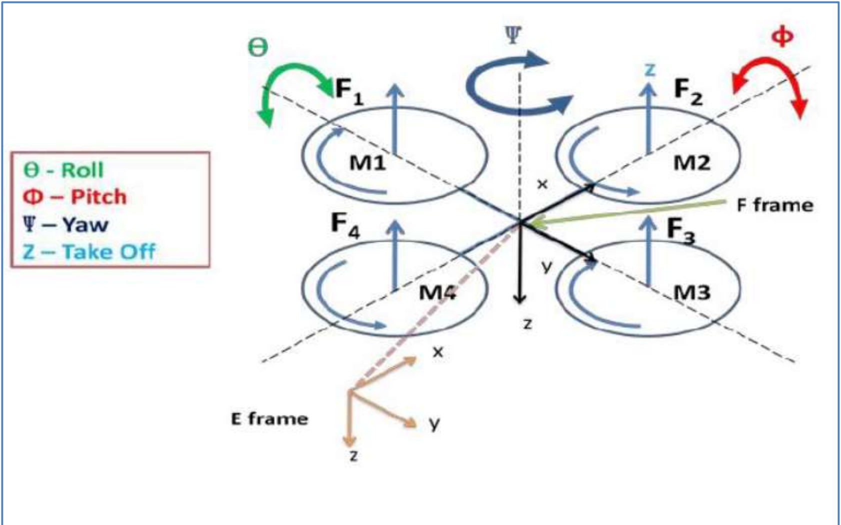


Figure 1. Quadcopter Control Frames

-see references [1]
-notice how F1,F3 spin CW (looking down) and F2,F4 spin CCW. This gives control to rotate about z
-note that z is pointing up for the drone reference.

$$R_v^b(\Phi, \Theta, \Psi) = R_X(\Phi)R_Y(\Theta)R_Z(\Psi)$$

$$R_v^b(\Phi,\Theta,\Psi) = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi\end{bmatrix}\begin{bmatrix}\cos\Theta & 0 & -\sin\Theta\\ 0 & 1 & 0\\ \sin\Theta & 0 & \cos\Theta\end{bmatrix}\begin{bmatrix}\cos\Psi & \sin\Psi & 0\\ -\sin\Psi & \cos\Psi & 0\\ 0 & 0 & 1\end{bmatrix}$$

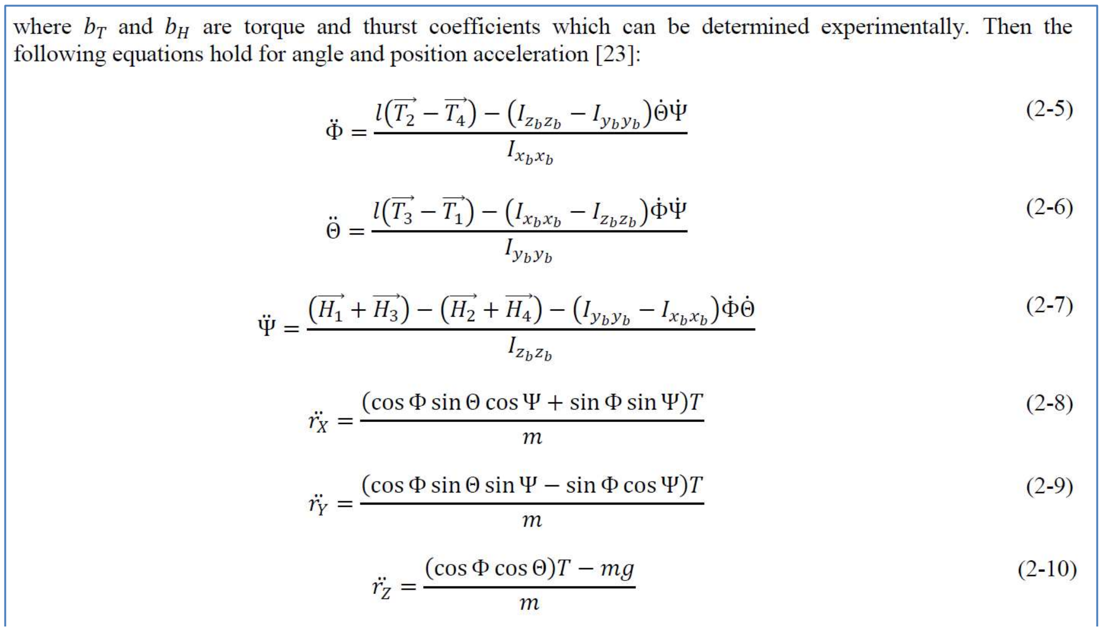
-rotation matrix [1]

Let us define $\vec{T}$ to be trust force and $\vec{H}$ is hub torque for every (1, 2, 3 and 4) DC Motor and Propeller system:

$$\vec{T} = b_T\left(\vec{\omega}_{propeller}\right)^2, \qquad \vec{H} = b_H\left(\vec{\omega}_{propeller}\right)^2 \tag{2-4}$$

-see references [1]
(hub torque = motor torque [2],[3])

Properties of the Drone:
$thrust = 37N\ per\ motor$
$m = mass\ =\ 10kg.l = 0.2m.$
$b_t\ =\ 8.5 * 10^{-6}.b_h\ =\ 2.15 * 10^{-7}$
$b = \frac{b_h}{b_t}\ =\ .0253$
Each of the 4 thrusters were used as the controlling variables.



where b_T and b_H are torque and thurst coefficients which can be determined experimentally. Then the following equations hold for angle and position acceleration [23]:

$$\ddot{\Phi} = \frac{l(\overrightarrow{T_2} - \overrightarrow{T_4}) - (I_{z_b z_b} - I_{y_b y_b})\dot{\Theta}\dot{\Psi}}{I_{x_b x_b}} \tag{2-5}$$

$$\ddot{\Theta} = \frac{l(\overrightarrow{T_3} - \overrightarrow{T_1}) - (I_{x_b x_b} - I_{z_b z_b})\dot{\Phi}\dot{\Psi}}{I_{y_b y_b}} \tag{2-6}$$

$$\ddot{\Psi} = \frac{(\overrightarrow{H_1} + \overrightarrow{H_3}) - (\overrightarrow{H_2} + \overrightarrow{H_4}) - (I_{y_b y_b} - I_{x_b x_b})\dot{\Phi}\dot{\Theta}}{I_{z_b z_b}} \tag{2-7}$$

$$\ddot{r_X} = \frac{(\cos\Phi \sin\Theta \cos\Psi + \sin\Phi \sin\Psi)T}{m} \tag{2-8}$$

$$\ddot{r_Y} = \frac{(\cos\Phi \sin\Theta \sin\Psi - \sin\Phi \cos\Psi)T}{m} \tag{2-9}$$

$$\ddot{r_Z} = \frac{(\cos\Phi \cos\Theta)T - mg}{m} \tag{2-10}$$

-see reference [1]

Note: variable l (lower case L) is the length from 0 to the propeller.
variable $T = T_1 + T_2 + T_3 + T_4$

The H_n variables will be replaced with their relationship to T_n:

$H_n = T_n * \frac{b_h}{b_t} = T_n * b,\ where\ b = \frac{b_h}{b_t}$

$x_1 = \Phi; x_3 = \Theta\ ; x_5 = \Psi\ ; x_7 = r_x; x_9 = r_y; x_{11} = r_z$

The state equations:

$\dot{x}_1 = \dot{\Phi} = x_2$

$\dot{x}_2 = \ddot{\Phi} = \frac{l(T_2 - T_4) - (I_{zz} - I_{yy})x_4 x_6}{I_{xx}}$

$\dot{x}_3 = \dot{\Theta} = x_4$

$\dot{x}_4 = \ddot{\Theta} = \frac{l(T_3 - T_1) - (I_{xx} - I_{zz})x_2 x_6}{I_{yy}}$

$\dot{x}_5 = \dot{\Psi} = x_6$

$\dot{x}_6 = \ddot{\Psi} = \frac{b(T_1 + T_3 - T_2 - T_4) - (I_{yy} - I_{xx})x_2 x_4}{I_{zz}}$

$\dot{x}_7 = \dot{r}_x = x_8$

$\dot{x}_8 = \ddot{r}_x = \frac{(\cos(x_1)\sin(x_3)\cos(x_5) + \sin(x_1)\sin(x_5))T}{m}$

$\dot{x}_9 = \dot{r}_y = x_{10}$

$\dot{x}_{10} = \ddot{r}_y = \frac{(\cos(x_1)\sin(x_3)\sin(x_5) - \sin(x_1)\cos(x_5))T}{m}$

$\dot{x}_{11} = \dot{r}_z = x_{12}$

$\dot{x}_{12} = \ddot{r}_z = \frac{(\cos(x_1)\cos(x_3))T - mg}{m}$

The next step is to set up these state equations and test the dynamics using RK4.

Note: the full code is in the appendix

```
%variables for properties of the drone
Ixx=8e-4;Iyy=8e-4;Izz=6e-4;%double check these
%w=1000 rad/s. thrust = 8.5N. Torque = .215Nm
b=.0253;g=9.81;l=.2;m=10;
```

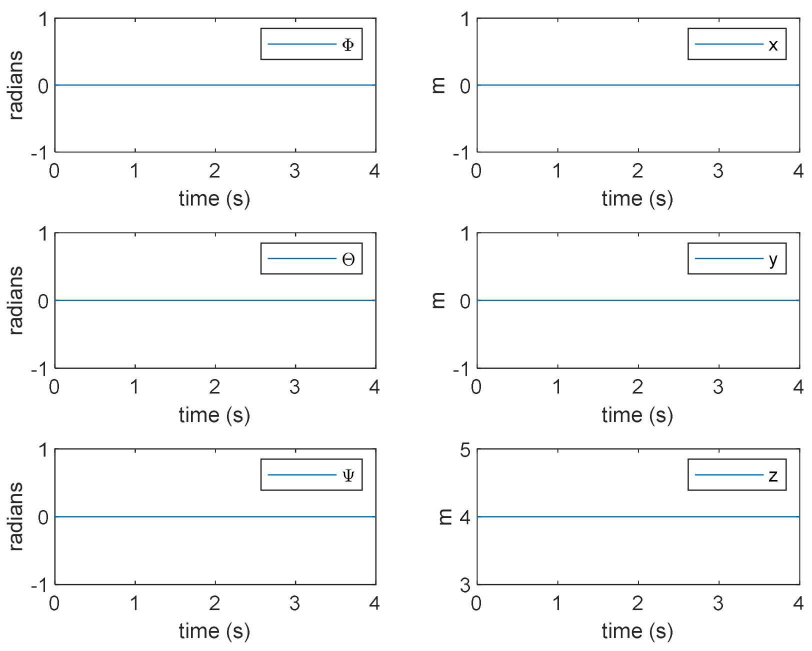
```
%initialize time vectors/parameters
dt=.01;
t0=0;
tt=t0:dt:t0+4;
L=length(tt);

%initial conditions
x=zeros(12,L);
x(11,1)=4;%initial height at 4m
```

The above properties are used for the different test cases for part 1A. In all tests, I've kept all 4 thrusters at some constant value and checked the resulting output (position/orientation).
I used the RK4 algorithm to solve this ODE (shown in the appendix)

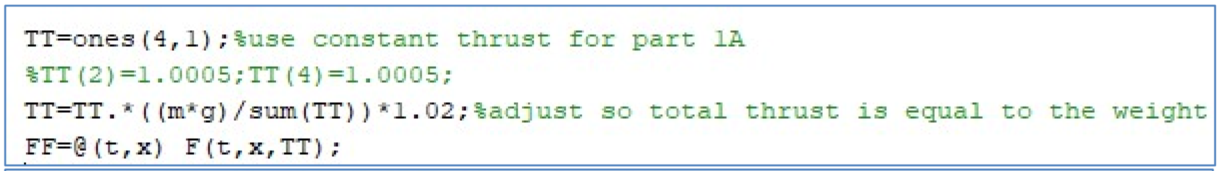
Test 1:

```
TT=ones(4,1);%use constant thrust for part 1A
%TT(2)=1.0005;TT(4)=1.0005;
TT=TT.*((m*g)/sum(TT));%adjust so total thrust is equal to the weight
FF=@(t,x) F(t,x,TT);
```

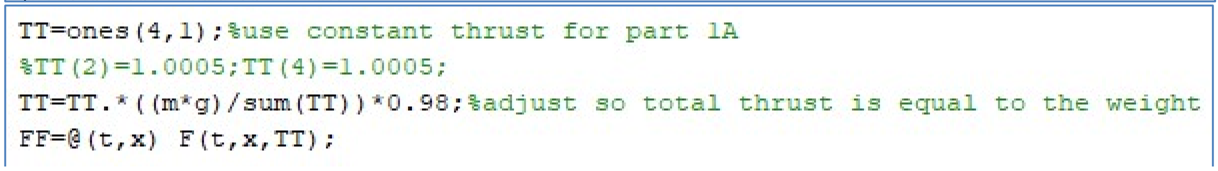
All thrusters are set equal and the total thrust force is the weight of the drone. Result:



As expected, there's no rotation and no change in position.

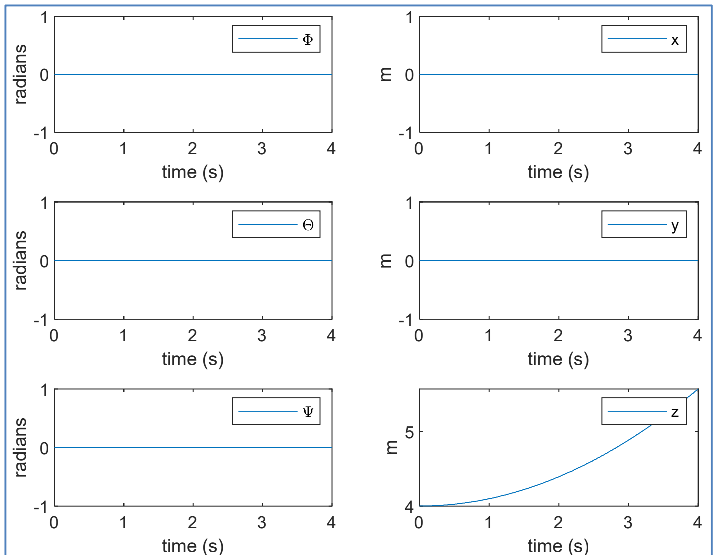


```
TT=ones(4,1);%use constant thrust for part 1A
%TT(2)=1.0005;TT(4)=1.0005;
TT=TT.*((m*g)/sum(TT))*1.02;%adjust so total thrust is equal to the weight
FF=@(t,x) F(t,x,TT);
```

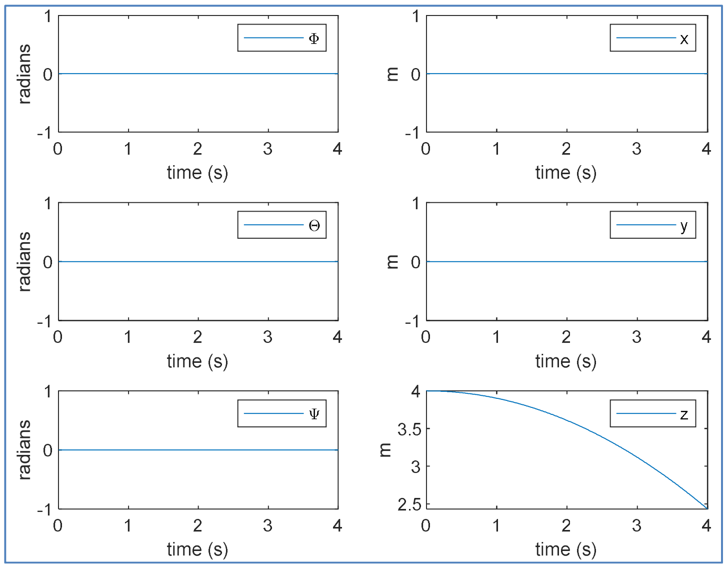


```
TT=ones(4,1);%use constant thrust for part 1A
%TT(2)=1.0005;TT(4)=1.0005;
TT=TT.*((m*g)/sum(TT))*0.98;%adjust so total thrust is equal to the weight
FF=@(t,x) F(t,x,TT);
```

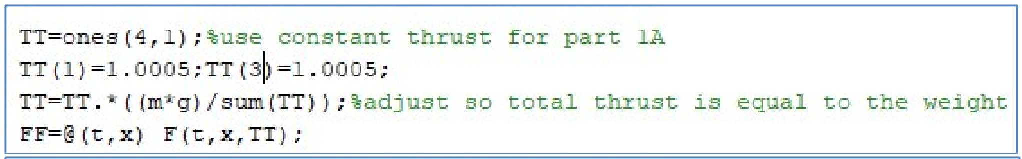
This time I increased the total thrust by 2% and then decreased the thrust by 2%. The results:



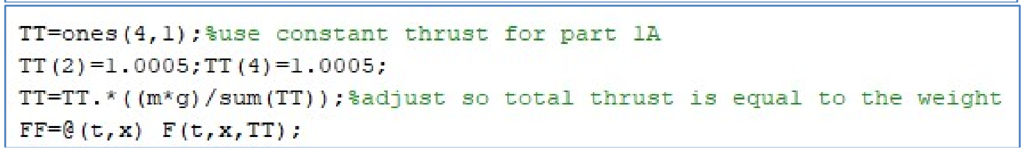
As expected, the drone increases height exponentially when there's an increase in thrust. Note that my assumptions did not consider the change in thrust vs altitude or the effects of drag. This will allow the drone to increase in height at a more rapid pace.



Similarly, the decrease in thrust causes the drone to decrease in altitude exponentially.

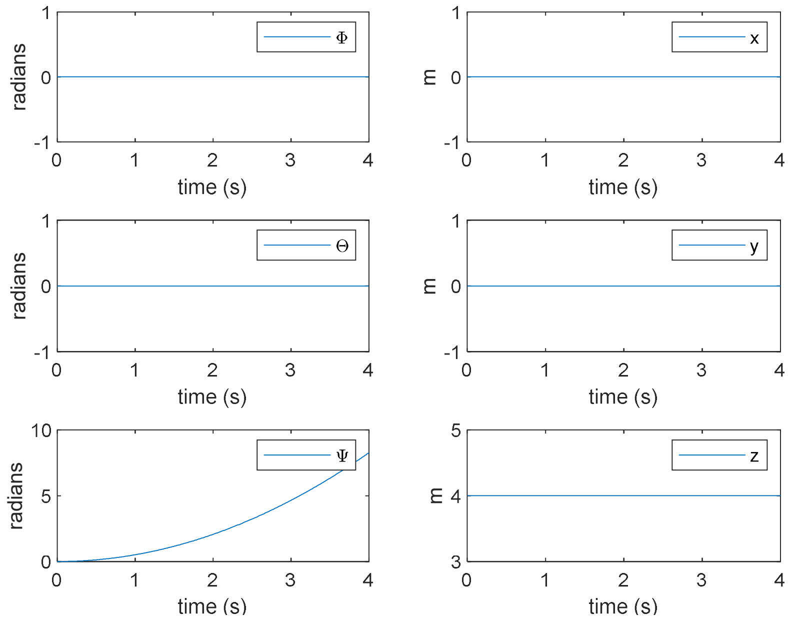


```
TT=ones(4,1);%use constant thrust for part 1A
TT(1)=1.0005;TT(3)=1.0005;
TT=TT.*((m*g)/sum(TT));%adjust so total thrust is equal to the weight
FF=@(t,x) F(t,x,TT);
```

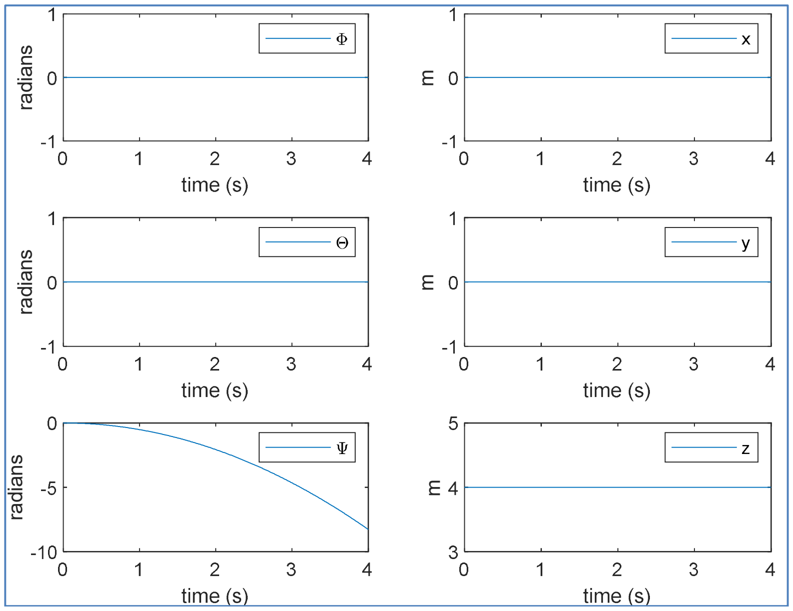


```
TT=ones(4,1);%use constant thrust for part 1A
TT(2)=1.0005;TT(4)=1.0005;
TT=TT.*((m*g)/sum(TT));%adjust so total thrust is equal to the weight
FF=@(t,x) F(t,x,TT);
```

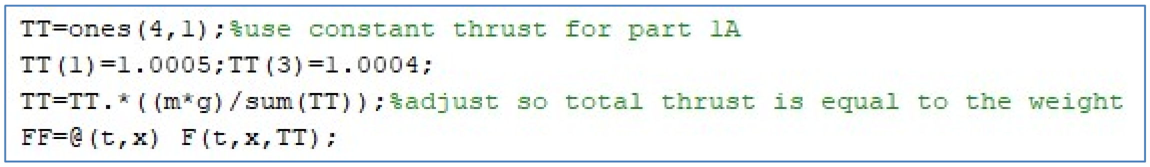
In these tests, I've slightly increased the thrust of the propeller's opposite of each other. First thrusters 1 and 3 and then another test on thrusters 2 and 4. This should cause the drone to spin. Results:



With thrusters 1 and 3 slightly higher, the drone spins in positive (CCW) direction about the z-axis.

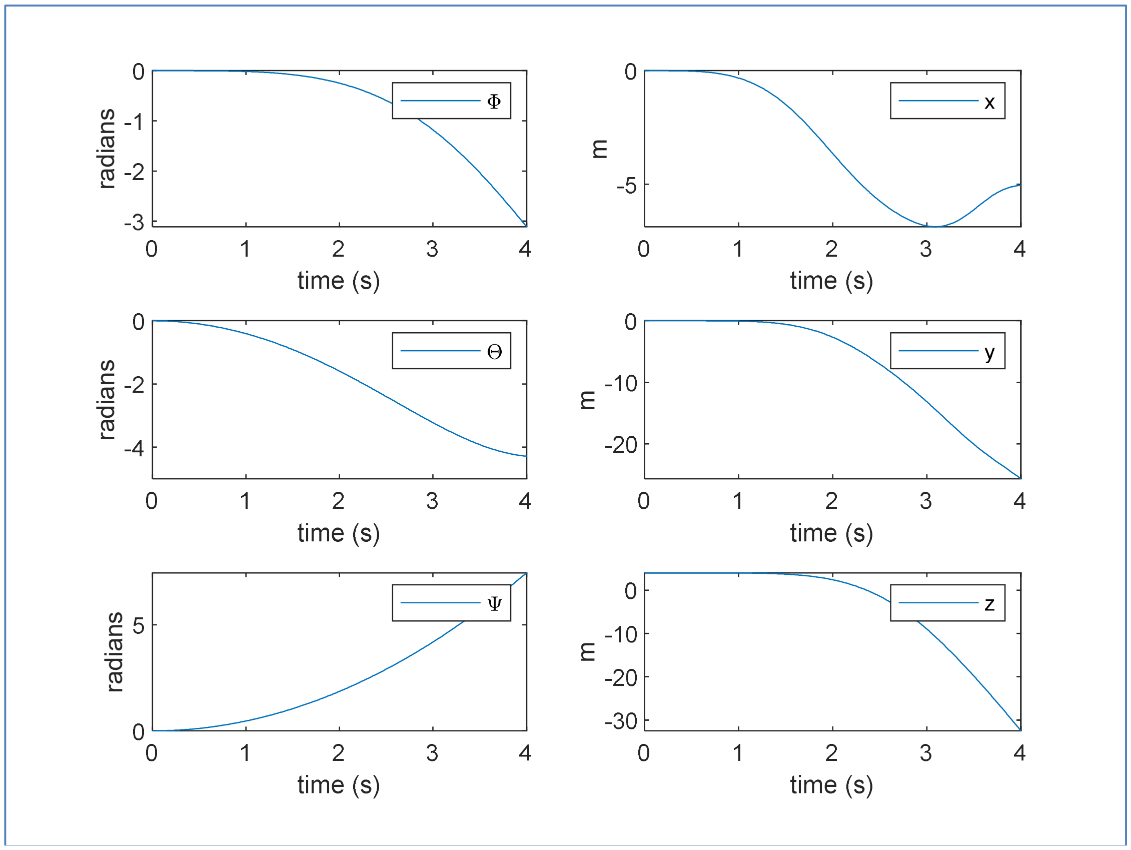


Similarly, thrusters 2 and 4 being higher caused the drone to spin in negative (CW) direction about z-axis.



```
TT=ones(4,1);%use constant thrust for part 1A
TT(1)=1.0005;TT(3)=1.0004;
TT=TT.*((m*g)/sum(TT));%adjust so total thrust is equal to the weight
FF=@(t,x) F(t,x,TT);
```

In this test, I made thrusters 1 and 3 slightly higher but thruster 1 is slightly higher than thruster 3. This will cause the drone to tilt at an uneven orientation. Results:



The results show that the drone drops in altitude after about 1 second. The unbalance in the thrusters cause the drone to rotate about both the x and y axes.
The x-position goes negative and then slowly comes back up which indicates that the drone is spiraling out of control. (This can also be observed in the animation).

Part C: Numerical Optimal Control

Reiterating:

$H_n = T_n * b,\ where\ b = \frac{b_h}{b_t}\ ; T = \sum T_n$

$x_1 = \Phi; x_3 = \Theta\ ; x_5 = \Psi\ ; x_7 = r_x; x_9 = r_y; x_{11} = r_z$

The state equations:

$\dot{x}_1 = \dot{\Phi} = x_2$

$\dot{x}_2 = \ddot{\Phi} = \frac{l(T_2 - T_4) - (I_{zz} - I_{yy})x_4 x_6}{I_{xx}}$

$\dot{x}_3 = \dot{\Theta} = x_4$

$\dot{x}_4 = \ddot{\Theta} = \frac{l(T_3 - T_1) - (I_{xx} - I_{zz})x_2 x_6}{I_{yy}}$

$\dot{x}_5 = \dot{\Psi} = x_6$

$\dot{x}_6 = \ddot{\Psi} = \frac{b(T_1 + T_3 - T_2 - T_4) - (I_{yy} - I_{xx})x_2 x_4}{I_{zz}}$

$\dot{x}_7 = \dot{r}_x = x_8$

$\dot{x}_8 = \ddot{r}_x = \frac{(\cos(x_1)\sin(x_3)\cos(x_5) + \sin(x_1)\sin(x_5))T}{m}$

$\dot{x}_9 = \dot{r}_y = x_{10}$

$\dot{x}_{10} = \ddot{r}_y = \frac{(\cos(x_1)\sin(x_3)\sin(x_5) - \sin(x_1)\cos(x_5))T}{m}$

$\dot{x}_{11} = \dot{r}_z = x_{12}$

$\dot{x}_{12} = \ddot{r}_z = \frac{(\cos(x_1)\cos(x_3))T - mg}{m}$

Given, some initial condition, an optimal control will be used to bring the drone to a fixed final position while minimizing the control and minimizing time.

Constraints:

$0 \leq T_n \leq T_{max}$

$x_{11} \geq 0$ ←the drone cannot go below the ground at 0 altitude

t_f is unspecified. However, $t_0 = 0, t_f > 0$.

The cost function is calculated as:

$J = t_f + \frac{1}{2}\int_{t_0}^{t_f}(T_1^2 + T_2^2 + T_3^2 + T_4^2)dt = t_f + \sum_{i=1}^{N}\frac{1}{2}W_i\left(\sum_{j=1}^{4}T_j^2\right)$

Since LGL points will be used, the cost function will need to be modified to account for the fact that the integral is always assumed from -1 to 1 and then adjusted later:

$J = t_f + \frac{t_f - t_0}{2}\sum_{i=1}^{N}\frac{1}{2}W_i\left(\sum_{j=1}^{4}T_j^2\right)$

The gradient of the cost is such that:

$\frac{\delta J}{\delta x_i} = 0$

$\frac{\delta J}{\delta T_i} = \frac{t_f - t_0}{2}\sum_{i=1}^{N}W_i\,T_i$

$\frac{\delta J}{\delta t_f} = 1 + \frac{1}{2}\sum_{i=1}^{N}\frac{1}{2}W_i\left(\sum_{j=1}^{4}T_j^2\right)$

The decision vector:
$\{x_0, x_1, \dots x_{17}\}$
Where the last value is t_f and the 4 values before that are the 4 controls.

The constraint equation is of the form:

$$\begin{bmatrix} c_1 \\ \dots \\ c_{12} \end{bmatrix} = D \begin{bmatrix} x_1 \\ \dots \\ x_{12} \end{bmatrix} - \frac{t_f - t_0}{2} \begin{bmatrix} f_1(x,T) \\ \dots \\ f_{12}(x,T) \end{bmatrix} = 0$$

The gradient of the constraint equation is such that:
$\frac{\delta c_i}{\delta x_j} = D^T - \frac{t_f - t_0}{2} diag\left(\frac{\delta f_i}{\delta x_j}\right)$ ←diagonals only
$\frac{\delta c_i}{\delta x_j} = -\frac{t_f - t_0}{2} diag\left(\frac{\delta f_i}{\delta x_j}\right)$ ←non-diagonals
$\frac{\delta c_i}{\delta T_j} = -\frac{t_f - t_0}{2} diag\left(\frac{\delta f_i}{\delta T_j}\right)$
$\frac{\delta c_i}{\delta t_f} = 0 - \frac{1}{2} f_i(x,T)$

After a lot of testing, it was proving difficult to get a good simulation result and the simulations were taking a long time. For this reason, the next step was to make simplifications to the dynamics equations to improve the results.

Simplifications

$$U = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} = \begin{bmatrix} T_1 + T_2 + T_3 + T_4 \\ T_2 - T_4 \\ T_3 - T_4 \\ (T_1 + T_3) - (T_2 + T_4) \end{bmatrix}$$

-these are the new controls: U1 (total thrust) ,U2 (pitch torque),U3 (roll torque),U4 (yaw torque)
-see reference [1]

$H_n = T_n * b,\ where\ b = \frac{b_h}{b_t}\ ; T = \sum T_n$
$x_1 = \Phi; x_3 = \Theta\ ; x_5 = \Psi\ ; x_7 = r_x; x_9 = r_y; x_{11} = r_z$

Substituting U into the state equations:
$\dot{x}_1 = \dot{\Phi} = x_2$
$\dot{x}_2 = \ddot{\Phi} = \frac{lU_2 - (I_{zz} - I_{yy})x_4x_6}{I_{xx}}$
$\dot{x}_3 = \dot{\Theta} = x_4$
$\dot{x}_4 = \ddot{\Theta} = \frac{lU_3 - (I_{xx} - I_{zz})x_2x_6}{I_{yy}}$
$\dot{x}_5 = \dot{\Psi} = x_6$
$\dot{x}_6 = \ddot{\Psi} = \frac{bU_4 - (I_{yy} - I_{xx})x_2x_4}{I_{zz}}$
$\dot{x}_7 = \dot{r}_x = x_8$
$\dot{x}_8 = \ddot{r}_x = \frac{(\cos(x_1)\sin(x_3)\cos(x_5) + \sin(x_1)\sin(x_5))U_1}{m}$
$\dot{x}_9 = \dot{r}_y = x_{10}$
$\dot{x}_{10} = \ddot{r}_y = \frac{(\cos(x_1)\sin(x_3)\sin(x_5) - \sin(x_1)\cos(x_5))U_1}{m}$
$\dot{x}_{11} = \dot{r}_z = x_{12}$
$\dot{x}_{12} = \ddot{r}_z = \frac{(\cos(x_1)\cos(x_3))U_1 - mg}{m}$

The equations can still be simplified even further. It can be assumed that all inertia is the same. It can also be simplified with the small angles assumption, limiting the amount of roll/pitch. It is best to keep the full rotation available for yaw, in case it is desired to make the drone spin.

Assuming small angles on roll and pitch only and Ixx=Iyy=Izz=I:

$H_n = T_n * b,\ where\ b = \frac{b_h}{b_t}$;

$x_1 = \Phi; x_3 = \Theta\ ; x_5 = \Psi\ ; x_7 = r_x; x_9 = r_y; x_{11} = r_z$

The state equations:

$\dot{x}_1 = \dot{\Phi} = x_2$

$\dot{x}_2 = \ddot{\Phi} = \frac{lU_2}{I}$

$\dot{x}_3 = \dot{\Theta} = x_4$

$\dot{x}_4 = \ddot{\Theta} = \frac{lU_3}{I}$

$\dot{x}_5 = \dot{\Psi} = x_6$

$\dot{x}_6 = \ddot{\Psi} = \frac{bU_4}{I}$

$\dot{x}_7 = \dot{r}_x = x_8$

$\dot{x}_8 = \ddot{r}_x = \frac{(x_3\cos(x_5)+x_1\sin(x_5))U_1}{m}$

$\dot{x}_9 = \dot{r}_y = x_{10}$

$\dot{x}_{10} = \ddot{r}_y = \frac{(x_3\sin(x_5)-x_1\cos(x_5))U_1}{m}$

$\dot{x}_{11} = \dot{r}_z = x_{12}$

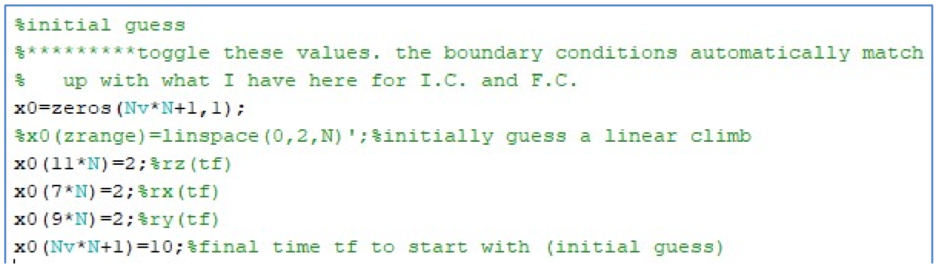
$\dot{x}_{12} = \ddot{r}_z = \frac{U_1-m}{m}$

$U_1 = x_{13}, U_2 = x_{14}, U_3 = x_{15}, U_4 = x_{16}$

Now the equations are much simpler and should hopefully help speed up calculation and improve the chances of finding a minimum within consraints.

Note that for the gradients, small angle approximation is only used after taking the partial derivatives.

Boundaries: boundaries were set per the I.C. and F.C. on the positions,orientations,velocities, and radial velocities. Boundaries were also set up to add the appropriate limits to the roll/pitch angles (+-5 degrees max) and all the controls (U).



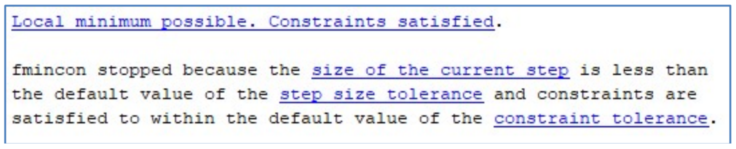
```
%initial guess
%*********toggle these values. the boundary conditions automatically match
%   up with what I have here for I.C. and F.C.
x0=zeros(Nv*N+1,1);
%x0(zrange)=linspace(0,2,N)';%initially guess a linear climb
x0(11*N)=2;%rz(tf)
x0(7*N)=2;%rx(tf)
x0(9*N)=2;%ry(tf)
x0(Nv*N+1)=10;%final time tf to start with (initial guess)
```

-for all simulations, the start and end positions were toggled to update the study. When running the code, just change the ending rx,ry,rz. There is one version of this code in the appendix which can easily be modified at this spot in the code for each of the cases shown in the next pages.

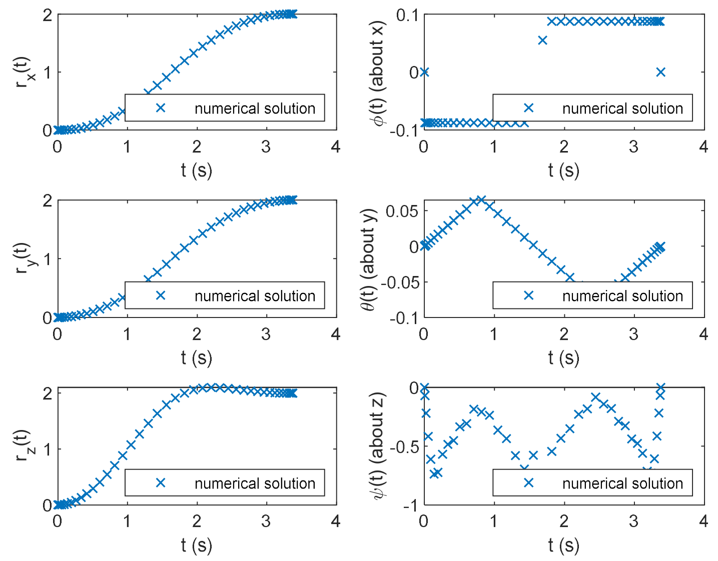
The cost function was updated to:

$J = t_f + \frac{1}{2}\int_{t_0}^{t_f} U_1^2\, dt$

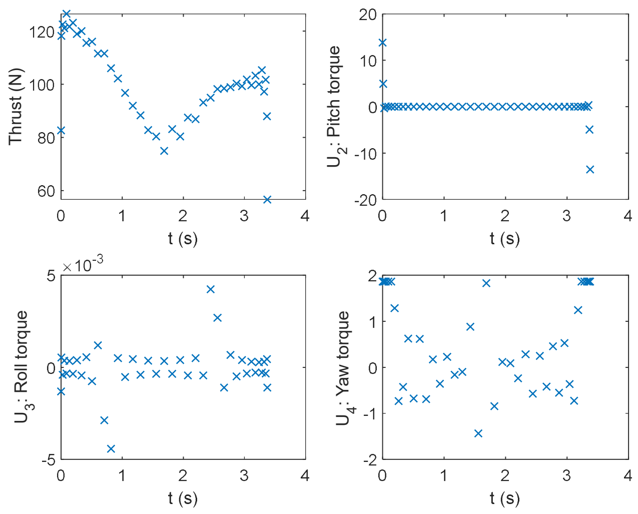
First, the drone was set to go from x,y,z=0 to x,y,z=2. N=41 LGL points were used until a local minimum was found:



```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.
```

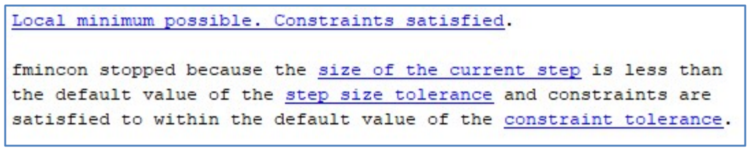


The control performed exactly as expected. There's an animation to go with this and it looks pretty realistic. The drone needed to rotate about z so that it could orient the right way such that it'll go up and tilt in the x and y direction and then it tilted back, which allowed itself to come to a stop at this position. At this point, it rotated the z-axis back.

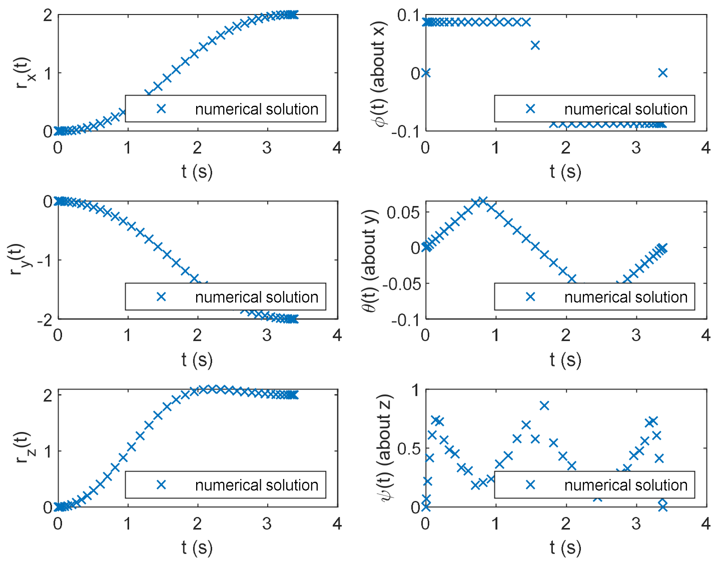


The total thrust acted appropriately, showing the max in the beginning, then as momentum picked up, it slowed down. Then as it needed to stabilize at the new position, it ramped up a little bit before throttling down. The pitch torque was much higher than the roll torque. Perhaps the rotation of the drone didn't require both a pitch and roll torque. The yaw torque was definitely throttled up and down to get the drone to behave this way.

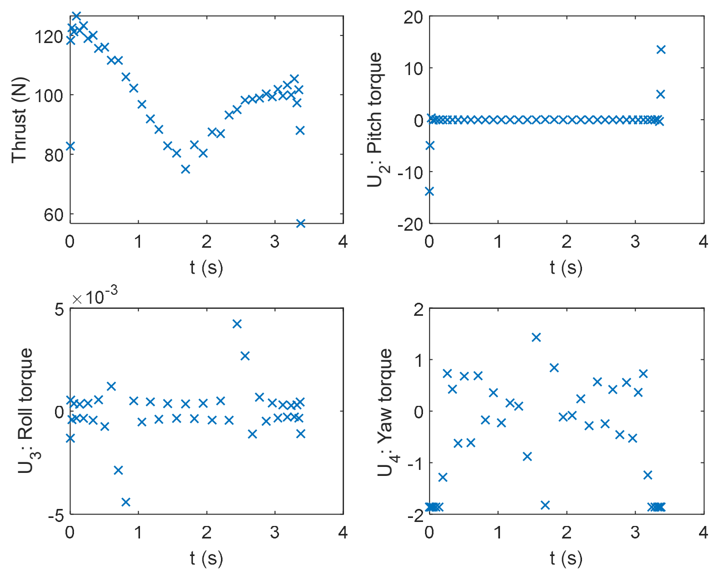
This time the final conditions were updated so that y goes from 0 to -2. It gave similar results as expected:



```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.
```



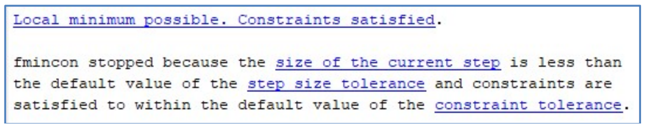
The graphs look about the same as before except that y goes from 0 to -2.



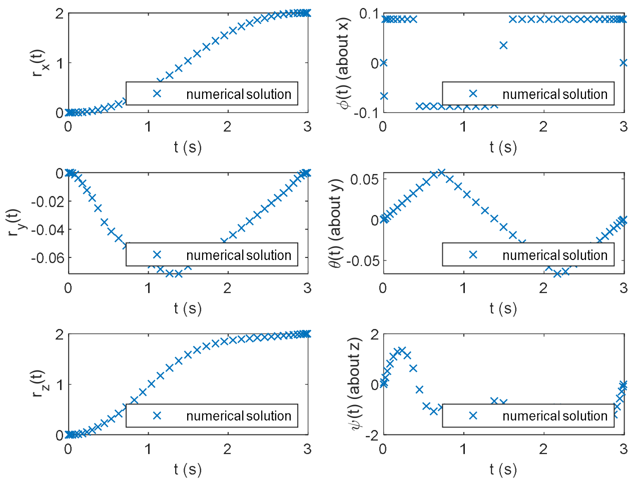
It took 3.3748 seconds to reach the goal position. Please see the appendix which includes the code for the simulation as well as the code for the animation.

This simulation used 41 LGL points and it took about a minute to find the minimum.

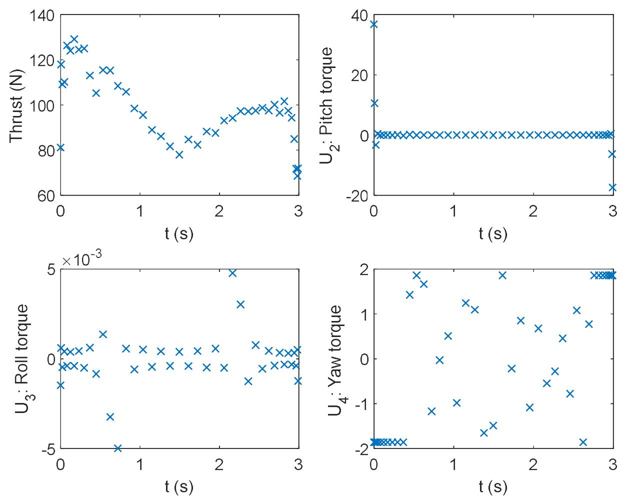
This time x,z=0 to x,z=2. The y axis was set to stay at its current position.



```
Local minimum possible. Constraints satisfied.

fmincon stopped because the size of the current step is less than
the default value of the step size tolerance and constraints are
satisfied to within the default value of the constraint tolerance.
```

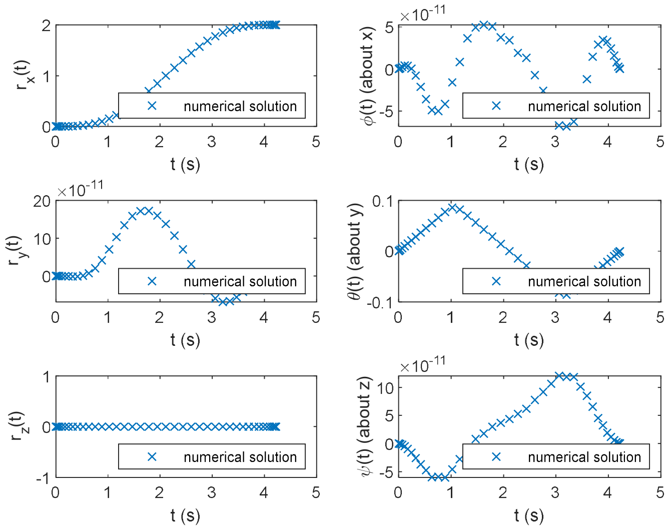


The drone dipped very slightly in the y-direction but found its way back to 0. Looking at the graphs, this was also a success.



The yaw torque seems to jump around quite a bit. When looking at the animation, it does appear like the spinning of the drone (about z-axis) is a little shaky and not perfectly smooth. The animation does still appear realistic. It is just interesting to see the result of this numerical control.

In this last study, the start and end height stayed at 0. The drone went from x=0 to x=2. There's a boundary that prevents the drone from ever going below 0. Here's the result:



The drone did not fly up and over as intuitively expected. However, this is only because of the simple constraint that prevents it from going below 0. To account for the ground, special constraints would need to be built into the equations to adjust the minimum height for different roll/pitch angles or the parameters (boundary conditions) would need to tell the drone to go up and then back down to prevent it from hitting the ground. In the end, the drone did exactly what it was told to do by the parameters. The motion in the animation for this study was very smooth.

Conclusion

Using LGL points to find a numerical solution of optimizing the control proved to be a challenging but effective approach. There were many attempts and it seems the approach that worked best was using the roll,pitch,yaw torque and the total thrust as the controls. Simplifying with the small angles assumption also helped improve how quickly a solution could be found.

Future Studies

On a future study, it wouldn't be too difficult to break this simulation up to have several way points for the drone (rather than just a start and end). To accomplish a more complicated path, the control optimization could be broken it up into multiple simulations – each with new initial/final conditions. The trajectory planning shown is evident that this should be a possibility.

References

[1] Control and Estimation of a Quadcopter Dynamic Model by Sevkuthan Kurak & Migdat Hodzic
[2] https://www.droneomega.com/drone-motor-essentials/
[3] https://www.micromo.com/technical-library/dc-motor-tutorials/motor-calculations

Appendix – Matlab Code

Part A:

```
%System Control and Reinforcement Learning Project 1
%close all
%clear
%clc




%-----=====Project 1 part A======------

%variables for properties of the drone
Ixx=8e-4;Iyy=8e-4;Izz=6e-4;%double check these
%w=1000 rad/s. thrust = 8.5N. Torque = .215Nm
b=.0253;g=9.81;l=.2;m=10;



%*****using xk1=RK4(F,xk,tk,dt) function****
%state space equations
F=@(t,x,T) [x(2)
    (l*(T(2)-T(4))-(Izz-Iyy)*x(4)*x(6))/Ixx
    x(4)
    (l*(T(3)-T(1))-(Ixx-Izz)*x(2)*x(6))/Izz
    x(6)
    (b*(T(1)+T(3)-T(2)-T(4))-(Iyy-Ixx)*x(2)*x(4))/Izz
    x(8)
    (cos(x(1))*sin(x(3))*cos(x(5))+sin(x(1))*sin(x(5)))*sum(T)/m
    x(10)
    (cos(x(1))*sin(x(3))*sin(x(5))-sin(x(1))*cos(x(5)))*sum(T)/m
    x(12)
    (cos(x(1))*cos(x(3)))*sum(T)/m-g
    ];



TT=ones(4,1);%use constant thrust for part 1A
TT(1)=1.0004;TT(3)=1.0004;%*******update this to change drone behavior****
TT=TT.*((m*g)/sum(TT));%adjust so total thrust is equal to the weight
FF=@(t,x) F(t,x,TT);



%initialize time vectors/parameters
dt=.01;
t0=0;
tt=t0:dt:t0+4;
L=length(tt);

%initial conditions
x=zeros(12,L);
x(11,1)=4;%initial height at 4m
%x(1,1)=pi/30;
%x(3,1)=pi/12;
```

```
%solve system of equations numerically using RK4
for k=1:L-1
    %RK4 Method
    x(:,k+1)=RK4(FF,x(:,k),tt(k),dt);
end

%plot the rotations and translations
figure(4)
subplot(3,2,1)
plot(tt,x(1,:));
xlabel('time (s)');ylabel('radians')
legend('\Phi')
subplot(3,2,3)
plot(tt,x(3,:));
xlabel('time (s)');ylabel('radians')
legend('\Theta')
subplot(3,2,5)
plot(tt,x(5,:));
xlabel('time (s)');ylabel('radians')
legend('\Psi');

subplot(3,2,2)
plot(tt,x(7,:));
xlabel('time (s)');ylabel('m');
legend('x');
subplot(3,2,4)
plot(tt,x(9,:));
xlabel('time (s)');ylabel('m');
legend('y');
subplot(3,2,6);
plot(tt,x(11,:));
xlabel('time (s)');ylabel('m');
legend('z');

xx=x;





%=======show drone animation (1A and 1C)======

%set up figure 4
figure(3)
clf
limscale=4;
lim = [-1,1].*limscale;zlim=[-.25,1]*limscale;
ax=axes('XLim',lim,'YLim',lim,'ZLim',zlim);
view(3);%changes to 3D view
pbaspect(ax,[1 1 1]);%set figure as 1:1
grid on;


%create drone in 3D space
%body
```

```
[x,y,z]=cylinder([.02,.02]);%cylinder
z=z.*l;
h(1)=surface(x,z,y,'FaceColor','Red');
h(2)=surface(z,x,y,'FaceColor','Blue');
h(3)=surface(x,-z,y,'FaceColor','Green');
h(4)=surface(-z,x,y,'FaceColor','Cyan');

%propellers
[x,y,z]=cylinder([l,l]./3);
z=z.*l./10;
h(5)=surface(x,y+l,z,'FaceColor','Red');
h(6)=surface(x+l,y,z,'FaceColor','Blue');
h(7)=surface(x,y-l,z,'FaceColor','Green');
h(8)=surface(x-l,y,z,'FaceColor','Cyan');

t=hgtransform('Parent',ax);
set(h,'Parent',t)
drawnow

%animate the drone
for k=1:L
    trans=eye(4);
    trans(:,4)=[xx(7,k);xx(9,k);xx(11,k);1];

    rot = Hx(xx(1,k))*Hy(xx(3,k))*Hz(xx(5,k));

    set(t,'Matrix',trans*rot);
    pause(.01);
end







%RK4 function for computing derivatives
function xk1=RK4(F,xk,tk,dt)
%finds the next value x_k+1 based on x_k
%
%xk1=RK4(F,xk,tk,dt)
%F(t,x) is a function handle of the derivative
%xk is the current value
%tk is the current time
%dt is the time step
%xk1 is the predicted next value
%
%note: xk can be a vector (for state space equations i.e. system of ODE)

%RK4 algorithm
F1=F(tk,xk);
x2k=xk+(.5*dt).*F1;
F2=F(tk+dt/2,x2k);
```

```
x3k=xk+(.5*dt).*F2;
F3=F(tk+dt/2,x3k);
x4k=xk+dt.*F3;
F4=F(tk+dt,x4k);

%next value predicted by RK4
xk1=xk+(dt/6).*(F1+2.*F2+2.*F3+F4);
end
```

Part C: Numerical Simulation

```
%System Control and Reinforcement Learning
%project 1C - numerical simulation
clear
clc


global N;global xlgl;global W;global D; global t0;
global Ixx;global Iyy; global Izz; global b; global g; global l;global m;
global I;global Nv;global Ns;
t0=0;
N=41;
Nv=12+4;%number of variables (excluding tf)
Ns=12;%number of state equations
Ixx=8e-4;Iyy=8e-4;Izz=6e-4;b=.0253;g=9.81;l=.2;m=10;%parameters
I=Ixx;%assume all I are equal
MaxThrustFactor=1.5;%how many times the weight
MaxThrust=MaxThrustFactor*m*g;%max thrust as funciton of the weight
MaxAccel=MaxThrust/m;
MaxRadAccel=2*MaxThrust*l/I;%max radial acceleration
MaxAngle=5*pi/180;%don't allow the drone to have extreme roll/pitch
%MaxAcceleration=MaxThrustFactor*g;%should never accelerate more than this

xlgl=legslbndm(N);%gets x-LGL quadrature nodes with z(1)=-1 to z(N)=1
%note: these nodes go from -1 to 1. It is aka Tau
D=legslbdiff(N,xlgl);%differentiation matrix D
W=legslbwt(N,xlgl);% LGL weights (Christoffel Weights)

global xrange yrange zrange phirange thrange psirange U1range U2range U3range
U4range;
xrange=6*N+1:7*N;yrange=8*N+1:9*N;zrange=10*N+1:11*N;
phirange=1:N;thrange=2*N+1:3*N;psirange=4*N+1:5*N;
U1range=12*N+1:13*N;U2range=U1range+N;U3range=U2range+N;U4range=U3range+N;

%initial guess
%*********toggle these values. the boundary conditions automatically match
%   up with what I have here for I.C. and F.C.
x0=zeros(Nv*N+1,1);
%x0(zrange)=linspace(0,2,N)';%initially guess a linear climb
x0(11*N)=2;%rz(tf)
x0(7*N)=2;%rx(tf)
x0(9*N)=2;%ry(tf)
x0(Nv*N+1)=10;%final time tf to start with (initial guess)
MaxHeight=max(x0(zrange))*1.1;



%lower bounds and upper bounds
%there's no bounds except the start and end points
LB=-inf*ones(Nv*N+1,1);%default
LB(zrange)=0;%must not go below 0 altitude
LB(6*N+1)=x0(6*N+1);LB(8*N+1)=x0(8*N+1);LB(10*N+1)=x0(10*N+1);%I.C. for x,y,z
LB(7*N)=x0(7*N);LB(9*N)=x0(9*N);LB(11*N)=x0(11*N);%final conditions x,y,z
LB(7*N+1)=x0(7*N+1);LB(9*N+1)=x0(9*N+1);LB(11*N+1)=x0(11*N+1);%I.C.
velocities
```

```
LB(8*N)=x0(8*N);LB(10*N)=x0(10*N);LB(12*N)=x0(12*N);%final conditions
velocities
%LB([12*N+1:13*N,15*N+1:16*N,18*N+1:19*N])=-MaxAccel;%acceleration limit
x,y,z
LB(U1range)=0;%thrust must always be at least 0
LB(U2range)=-MaxThrust/4;LB(U3range)=-MaxThrust/4;LB(U4range)=-
MaxThrust*b/2;%torque limits
LB(1:N)=-MaxAngle;LB(3*N+1:4*N)=-MaxAngle;%angle limits
LB(1)=x0(1);LB(2*N+1)=x0(2*N+1);LB(4*N+1)=x0(4*N+1);%initial conditions
angles
LB(N)=x0(N);LB(3*N)=x0(3*N);LB(5*N)=x0(5*N);%final conditions angles
%LB([2*N+1:3*N,5*N+1:6*N,9*N+1:10*N])=-MaxRadAccel;%min radial acceleration
LB(Nv*N+1)=0;%tf must be at least 0
UB= inf*ones(Nv*N+1,1);%default
UB(U1range)=MaxThrust;%maximum thrust
UB(U2range)=MaxThrust/4;UB(U3range)=MaxThrust/4;UB(U4range)=MaxThrust*b/2;%to
rque limits
UB(zrange)=MaxHeight;
UB(6*N+1)=x0(6*N+1);UB(8*N+1)=x0(8*N+1);UB(10*N+1)=x0(10*N+1);%I.C. x,y,z
UB(7*N)=x0(7*N);UB(9*N)=x0(9*N);UB(11*N)=x0(11*N);%final conditions x,y,z
UB(7*N+1)=x0(7*N+1);UB(9*N+1)=x0(9*N+1);UB(11*N+1)=x0(11*N+1);%I.C.
velocities
UB(8*N)=x0(8*N);UB(10*N)=x0(10*N);UB(12*N)=x0(12*N);%final conditions
velocities
%UB([12*N+1:13*N,15*N+1:16*N,18*N+1:19*N])=MaxAccel;%acceleration limit x,y,z
UB(1:N)=MaxAngle;UB(3*N+1:4*N)=MaxAngle;%angle limits
UB(1)=x0(1);UB(2*N+1)=x0(2*N+1);UB(4*N+1)=x0(4*N+1);%initial conditions
angles
UB(N)=x0(N);UB(3*N)=x0(3*N);UB(5*N)=x0(5*N);%final conditions angles
%UB([2*N+1:3*N,5*N+1:6*N,9*N+1:10*N])=MaxRadAccel;%max radial acceleration
%ZERO=1e-5;LB=LB-ZERO;UB=UB+ZERO;%eliminate fmincon x0 shift
%*************need to add bounds for accelerations********

checkx0(UB,LB,x0)

%load x_opt and tt (data from previous simulation run)
%load('DroneSim');%used only to plot without rerunning simulation

% [y,dy]=costfun3(x0);%debug
% [f,df]=nonlinfun3(x0);


%solve using fmincon
options=optimoptions('fmincon','Algorithm','interior-point',...
    'MaxIterations',5e5,'MaxFunctionEvaluations',1e6,...
    'SpecifyObjectiveGradient',true,'SpecifyConstraintGradient',true,...
    'Display','iter');
%uncomment this to rerun the simulation
x_opt=fmincon(@costfun3,x0,[],[],[],[],LB,UB,@nonlincon3,options);
rx=x_opt(xrange);
ry=x_opt(yrange);
rz=x_opt(zrange);
phi=x_opt(phirange);
theta=x_opt(thrange);
psi=x_opt(psirange);
T=x_opt(U1range);%total thrust U1
```

```
U2=x_opt(U2range);
U3=x_opt(U3range);
U4=x_opt(U4range);

tf=x_opt(Nv*N+1)
tau=xlgl;
tt=( (tf-t0).*tau+(tf+t0) )/2;

save('DroneSim','x_opt','tt','rx','ry','rz','phi','theta','psi','T','tf','N')
;%save results


%plot the thrust
figure(2)
subplot(2,2,1)
title('U_1: thrust (overall)')
plot(tt,T,'x');
xlabel('t (s)');ylabel('Thrust (N)');
subplot(2,2,2)
plot(tt,U2,'x');
xlabel('t (s)');ylabel('U_2: Pitch torque')
subplot(2,2,3)
plot(tt,U3,'x');
xlabel('t (s)');ylabel('U_3: Roll torque');
subplot(2,2,4)
plot(tt,U4,'x');
xlabel('t (s)');ylabel('U_4: Yaw torque');




%plot positions and angles
figure(1)
title('positions and angles')
subplot(3,2,1);
plot(tt,rx,'x');
xlabel('t (s)');ylabel ('r_x(t)');
legend('numerical solution','Location','SouthEast')
subplot(3,2,3);
plot(tt,ry,'x');
xlabel('t (s)');ylabel ('r_y(t)');
legend('numerical solution','Location','SouthEast')
subplot(3,2,5);
plot(tt,rz,'x');
xlabel('t (s)');ylabel ('r_z(t)');
legend('numerical solution','Location','SouthEast')
subplot(3,2,2)
plot(tt,phi,'x');
xlabel('t (s)');ylabel ('\phi(t) (about x)');
legend('numerical solution','Location','SouthEast')
subplot(3,2,4)
plot(tt,theta,'x');
xlabel('t (s)');ylabel ('\theta(t) (about y)');
legend('numerical solution','Location','SouthEast')
subplot(3,2,6)
```

```
plot(tt,psi,'x');
xlabel('t (s)');ylabel ('\psi(t) (about z)');
legend('numerical solution','Location','SouthEast')




%non-linear constraint
function [c,ceq,gradc,gradceq]=nonlincon3(x)
%inputs:
%x = the decision vector
%
%outputs:
%c=inequality constraint
%ceq=equality constraint
%gradc=gradient of inequality constraint
%gradceq=gradient of equality constraint
global N;global D;
global t0;
global Nv;global Ns;

tf=x(Nv*N+1);
time_scale=(tf-t0)/2;

%ceq: # of state equations
%gradceq: size of x0 x # of state equations
ceq=zeros(Ns*N,1);%Dynamic Defect Constraints
gradceq=zeros(Nv*N+1,Ns*N);%gradient of Dynamic Defect Constraints

[f,df]=nonlinfun3(x);


%using the loop is convenient for bigger problems
%cycle through f1,f2
for i=1:Ns%#of state equations
    range=(i-1)*N+1:i*N;
    ceq(range)=D*x(range)-time_scale*f(range);

    %ci=zeros(1,16);%coefficient
    %ci(i)=1;

    for j=1:Nv%cycle all x accept tf
        jrange=(j-1)*N+1:j*N;
        if i==j
            gradceq(jrange,range)=D'-time_scale*diag(df(jrange,i));
        else
            gradceq(jrange,range)=-time_scale*diag(df(jrange,i));
        end
    end
    gradceq(Nv*N+1,range)=-.5.*f(range)';%adjust tf

end
```

```
c=[];
gradc=[];

end


%Nonlinear Programming Function
%(implements the state equations and their derivatives)
function [f,df]=nonlinfun3(x)
%input:
%x=decision vector
%
%outputs:
%f=State Dynamic Derivatives
%df=gradients of State Dynamic Derivatives

global N
global Ixx;global Iyy; global Izz; global b; global g; global l;global m;
global I;global Nv;global Ns;
global Trange;
f=zeros(Ns*N,1);
df=zeros(Nv*N+1,Ns);

x1=x(1:N);%
x2=x(N+1:2*N);%
x3=x(2*N+1:3*N);%
x4=x(3*N+1:4*N);%
x5=x(4*N+1:5*N);%
x6=x(5*N+1:6*N);%
x7=x(6*N+1:7*N);%
x8=x(7*N+1:8*N);%
x9=x(8*N+1:9*N);%
x10=x(9*N+1:10*N);%
x11=x(10*N+1:11*N);%
x12=x(11*N+1:12*N);%
U1=x(12*N+1:13*N);%
U2=x(13*N+1:14*N);%
U3=x(14*N+1:15*N);%
U4=x(15*N+1:16*N);%


for i=1:N

    %state equations
    f(i)=x2(i);
    f(i+N)=l*U2(i)/I;
    f(i+2*N)=x4(i);
    f(i+3*N)=l*U3(i)/I;
    f(i+4*N)=x6(i);
    f(i+5*N)=b*U4(i)/I;
    f(i+6*N)=x8(i);
    f(i+7*N)=(x3(i)*cos(x5(i))+x1(i)*sin(x5(i)))*U1(i)/m;
    f(i+8*N)=x10(i);
    f(i+9*N)=(x3(i)*sin(x5(i))-x1(i)*cos(x5(i)))*U1(i)/m;
    f(i+10*N)=x12(i);
    f(i+11*N)=U1(i)/m-g;

    % Hessian

    %derivative of fn with respect to x1,x2,..x12,T
    df(i+N,1)=1;%df1/dx2
    df(i+3*N,3)=1;%df3/dx4
    df(i+5*N,5)=1;%df5/dx6
    df(i+7*N,7)=1;%df7/dx8
    df(i+9*N,9)=1;%df9/dx10
    df(i+11*N,11)=1;%df11/d12


    %f2 with respect to U2
    df(13*N+i,2)=l/I;%df2/dU2

    df(14*N+i,4)=l/I;%df4/dU3

    df(15*N+i,6)=b/I;%df6/dU4

    %f8 with respect to x1,x3,x5,U1
    df(i,8)=(-x1(i)*x3(i)*cos(x5(i))+sin(x5(i)))*U1(i)/m;%df8/dx1
    df(2*N+i,8)=(cos(x5(i)))*U1(i)/m;%df8/dx3
    df(4*N+i,8)=(-x3(i)*sin(x5(i))+x1(i)*cos(x5(i)))*U1(i)/m;%df8/dx5
    df(12*N+i,8)=(x3(i)*cos(x5(i))+x1(i)*sin(x5(i)))/m;%df8/dU1

    %f10 wtih respect to x1,x3,x5,U1
    df(i,10)=(-x1(i)*x3(i)*sin(x5(i))-cos(x5(i)))*U1(i)/m;%df10/dx1
    df(2*N+i,10)=(sin(x5(i)))*U1(i)/m;%df10/dx3
    df(4*N+i,10)=(x3(i)*cos(x5(i))+x1(i)*sin(x5(i)))*U1(i)/m;%df10/dx5
    df(12*N+i,10)=(x3(i)*sin(x5(i))-x1(i)*cos(x5(i)))/m;%df10/dU1

    %f12 with respect to x1,x3,U1
    df(i,12)=-x1(i)*U1(i)/m;%df12/dx1
    df(2*N+i,12)=-x3(i)*U1(i)/m;%df12/dx3
    df(12*N+i,12)=1/m;%df12/dU1




end

end



%cost function
function [y,dy]=costfun3(x)
global N;global W;global t0;
global Nv;global Ns;
global U1range U2range U3range U4range;

y=0;dy=zeros(Nv*N+1,1);
```

```
U1=x(U1range);

tf=x(Nv*N+1);
wf = (tf-t0)/2;%weighting factor

for i=1:N
    y=y+W(i)*(.5*U1(i)^2);%cost function
end

%gradient of the cost function
dy(U1range)=U1.*W.*wf;%dy/dT
dy(Nv*N+1)=.5*y+1;%dy/dtf


y=y*wf+tf;


end




function checkx0(UB,LB,x0)
%check to make sure x0 is between UB and LB
L=length(UB);
issuefound=0;
for i=1:L
    if x0(i)>UB(i)||x0(i)<LB(i)
        msg=sprintf('initial condition not between LB/UB at index %1.0f. ',i);
        msg=[msg,sprintf('
LB: %1.2f\tx0: %1.2f\tUB: %1.2f\n',LB(i),x0(i),UB(i))];
        disp(msg);
        issuefound=1;
    end
end

if issuefound
    error('fix x0,LB, or UB');
end

end
```

Part C - Animation

```
%System Control and Reinforcement Learning
%Project 1C Animation
%close all
%clear
%clc


%note: run the simulation first so it can save all requried variables in
%the DroneSim.mat file

%variables for properties of the drone
Ixx=8e-4;Iyy=8e-4;Izz=6e-4;%double check these
%w=1000 rad/s. thrust = 8.5N. Torque = .215Nm
b=.0253;g=9.81;l=.2;m=10;







%=======show drone animation ======

%set up figure 4
figure(3)
clf
limscale=2.5;
lim = [-1,1].*limscale;zlim=[-.25,1]*limscale;
ax=axes('XLim',lim,'YLim',lim,'ZLim',zlim);
view(3);%changes to 3D view
pbaspect(ax,[1 1 1]);%set figure as 1:1
grid on;


%create drone in 3D space
%body
[x,y,z]=cylinder([.02,.02]);%cylinder
z=z.*l;
h(1)=surface(x,z,y,'FaceColor','Red');
h(2)=surface(z,x,y,'FaceColor','Blue');
h(3)=surface(x,-z,y,'FaceColor','Green');
h(4)=surface(-z,x,y,'FaceColor','Cyan');

%propellers
[x,y,z]=cylinder([l,l]./3);
z=z.*l./10;
h(5)=surface(x,y+l,z,'FaceColor','Red');
h(6)=surface(x+l,y,z,'FaceColor','Blue');
h(7)=surface(x,y-l,z,'FaceColor','Green');
h(8)=surface(x-l,y,z,'FaceColor','Cyan');

t=hgtransform('Parent',ax);
set(h,'Parent',t)
drawnow
```

```
%animate the drone for part 1C
load('DroneSim');
```

```
% %update limits based on max and min positions
% horzlim=[min([rx-.5;ry-.5;-2]),max([rx+.5;ry+.5;2])];
% vertlim=[-1,max([rz+.5;3])];
% ax=axes('XLim',horzlim,'YLim',horzlim,'ZLim',vertlim);
dt=tt(2:end)-tt(1:end-1);
dt=[dt;dt(end)];%change in time. used to scale the timing of the animation
%animation
for k=1:N
    trans=eye(4);
    trans(:,4)=[rx(k);ry(k);rz(k);1];

    rot = Hx(phi(k))*Hy(theta(k))*Hz(psi(k));

    set(t,'Matrix',trans*rot);
    extrapause=1;
    pause(extrapause*10*tf*dt(k)/N);
end
```