ECE 8 - Lab 3: Introduction to Quadcopters (Part 1)

November 3, 2022

1 Introduction

Quadcopters, also known as quadrotors, are a special type of aerial vehicle (or drone) that consist of four driven propellers placed in a square formation with equal distance from the center of mass of the vehicle. The propellers in this particular formation allows the drone to move about in three dimensional space through select control of the rotation speed (or angular velocities) of the individual propellers. Quadcopters are commonly used as small unmanned aerial vehicles (UAV) due to their low cost and simple structure in applications including, but certainly not limited to, surveillance, search and rescue, and small material delivery.

In this lab, we will introduce you to the simple quadcopter including its components, basic mechanics, and the tools to build simple flight controllers for the one-dimensional and two-dimensional quadcopter systems.

1.1 Lab Objectives

By the end of this lab you should be able to:

1. Identify the key components of a quadcopter as well as their 1D and 2D models. Identify the state components of the 1D and 2D models of a quadcopter.
2. Understand ordinary differential equations (ODEs) and how to solve them.
3. Understand the use of functions in MATLAB.

1.2 Necessary Files, Tools, and Equipment

- MATLAB

1.3 Deliverables

1. A `.pdf` version of your main `.m` file using the `publish` function, plus any additional `.m` and `.mat` files, due Friday, 11/11/2022, at 6 PM via Canvas.

1.4 Grading Rubric

This assignment consists of 10 points.

- Each exercise is worth 3 points.
- Properly commented code is worth 1 point total.

2 Quadcopter Basics

Before we begin on the basics of quadcopter modeling and flight control we must first understand the components that define a quadcopter. As mentioned in the introduction, quadcopters are a special type of rotorcraft comprised of four propellers arranged in a square formation. Each *propeller* is driven by an individual electric *motor* that is controlled by a small computer called a *Flight Control Unit* (FCU). The FCU is a special type of computer that is responsible for translating user input into aerial flight using information from the vehicle's on-board sensors and connected motors using pre-programmed flight controllers and temporal logic. The last but certainly not least component of a drone is the *battery* as a drone simply cannot fly without a means to power its motors and FCU. Figure 1 provides more details below.

Figure 1: Main components of a quadcopter.

As mentioned, the FCU is responsible for translating user input to aerial flight using pre-programmed flight controllers. However, in order to design the flight controllers we must first understand the physics that govern the aerial flight and more importantly we need a coordinate system to assist in describing such aerial flight.

In Figure 2 below we find two coordinate frames, a *body* reference frame for the quadrotor and an *inertial* reference frame that describes the space in which the quadrotor moves. The quadrotor has six *degrees of freedom* or six ways in which it can move. We have three degrees of *translational* motion, up/down (z_b), right/left (y_b), and forward/back (x_b) and three degrees of *rotational* motion about each aforementioned axis of translational motion as marked by ψ (yaw), θ (pitch), and ϕ (roll).

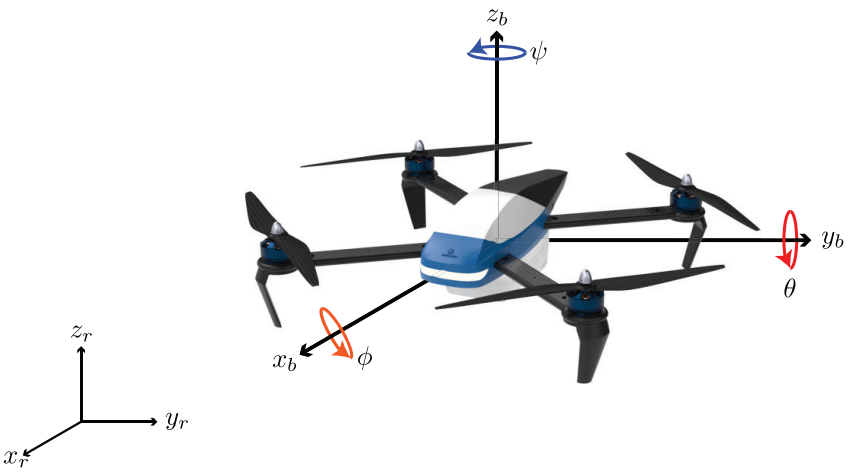

Figure 2: Quadcopter coordinate frame and state variables.

We have now reached the point where we can begin to describe how the quadrotor is able to move using its four motor driven propellers. Figure 3 provides a *free-body diagram* of the quadrotor showing the applied forces, moments, and resulting reactions on the quadrotor body in a simple hover condition. As can be seen, we have four forces F_i provided by the propellers to counteract the force of gravity given by mg where m represents the mass of the vehicle and g represents the acceleration due to gravity. Each force exerted by the individual propellers is given by a function of their rotational speed or angular velocity ω_i where

$$F_i = k_F \omega_i^2 \tag{1}$$

Here, k_F is a constant parameter that accounts for a number physical parameters relating to motor torques and aerodynamics that you don't need to concern yourself with at the moment.

In addition to the forces exerted by the propellers to counteract gravity, each motor produces a moment or torque τ_{M_i} that gives the quadrotor a tendency to spin about the z_b axis or yaw (ψ). This torque is also a function of the angular velocity of the respective propeller and is given by

$$\tau_{M_i} = k_M \omega_i^2 \tag{2}$$

where k_M is a physical constant parameter that can be ignored for now. To counteract this tendency, the motors in a quadrotor are configured such that the motors on opposing arms spin in the same direction while the adjacent motors spin in the opposite direction. If the motors are spinning at the same speed in their respective directions, the torques of the two motors spinning in the same direction should counteract the torques of the motors spinning in the opposite directions. This is also highlighted in Figure 3 by noting the direction of the arrows for both the angular velocities (ω_i) and torque moments (τ_{M_i}).

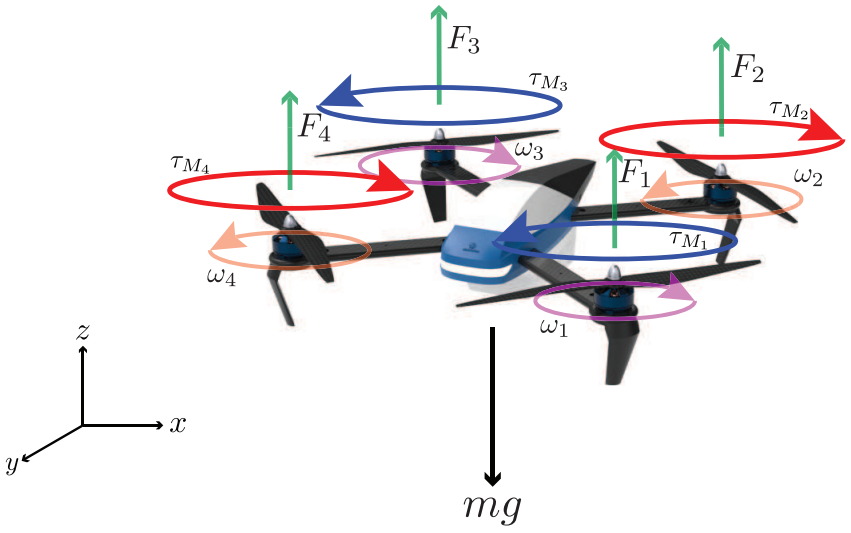

Figure 3: Free-body diagram of a quadcopter.

Let's now move on to understand how these equations and components translate to motion of a quadrotor. To start, we will begin with the one-dimensional model.

2.1 1D Quadcopter Dynamics

In the one dimensional quadcopter model, we only consider translational motion that allows the quadrotor to move up and down, in this case the z axis as shown in Figure 4. Thus, for now we can ignore any moments cause by the torque of the motors and variations in the angular velocities that would

cause the vehicle to pitch or roll. For the one dimensional case, we only need to consider the forces that act along the z axis, namely, the force due to gravity mg and the force or thrust generated by the rotating propellers.

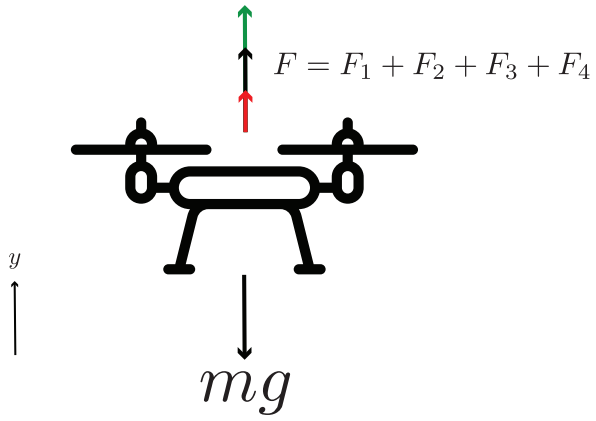

Figure 4: 1D Model of a Quadcopter.

Then using Newton's second law of motion we have

$$\begin{aligned} F_z = ma_z &= F_1 + F_2 + F_3 + F_4 - mg \\ &= \sum_{i=1}^{4} F_i - mg \end{aligned} \tag{3}$$

where F_z denotes the net forces acting on the quadrotor, a_z gives the acceleration of the quadrotor in the z direction, and m is the mass of the quadrotor. Since gravity acts in the negative direction of our z axis, the drone is subjected to a negative gravitational force as denoted by the minus sign. The remaining forces acting on the quadrotor come from the driven propellers F_i acting in the positive direction relative to the z axis. We will assume that force exerted by each propeller is the same, namely, $F_1 = F_2 = F_3 = F_4$.

One can also formulate this equation as an *ordinary differential equation* (ODE) by recognizing that the acceleration a_z is the derivative of the velocity in the z direction, i.e., $a_z = \frac{dv_z}{dt}$, thus by making the appropriate substitutions in (3) one has

$$m\frac{dv_z}{dt} = \sum_{i=1}^{4} F_i - mg \tag{4}$$

dividing both sides by the mass m gives

$$\frac{dv_z}{dt} = \frac{1}{m}\sum_{i=1}^{4} F_i - g \tag{5}$$

With these equation we can then describe the three states of operation for the quadrotor in one dimension: *hover*, *ascent*, and *descent*.

When the drone is ***hovering***, it is neither accelerating upward or downward, for this state of operation the net force F_z acting on the drone must equal zero or

$$F_z = ma_z = \sum_{i=1}^{4} F_i + mg = 0 \tag{6}$$

the ODE for this state is given by

$$\frac{dv_z}{dt} = \frac{1}{m}\sum_{i=1}^{4} F_i - g = 0 \tag{7}$$

When the drone is ***ascending***, the drone is accelerating upward relative to the z axis, for this state of operation, the net force F_z acting on the drone must be strictly **greater than** zero or

$$F_z = ma_z = \sum_{i=1}^{4} F_i + mg > 0 \tag{8}$$

the ODE for this state is given by

$$\frac{dv_z}{dt} = \frac{1}{m}\sum_{i=1}^{4} F_i - g > 0 \tag{9}$$

When the drone is ***descending***, the drone is accelerating downward relative to the z axis, for this state of operation, the net force F_z acting on the drone must be strictly **less than** zero or

$$F_z = ma_z = \sum_{i=1}^{4} F_i + mg < 0 \tag{10}$$

the ODE for this state is given by

$$\frac{dv_z}{dt} = \frac{1}{m}\sum_{i=1}^{4} F_i - g < 0 \tag{11}$$

2.2 2D Quadcopter Dynamics

Moving on to the two dimensional or planar model, we must now deal with two additional degrees of freedom, translational motion in the y direction and rotational motion about the $y-z$ plane or ϕ.

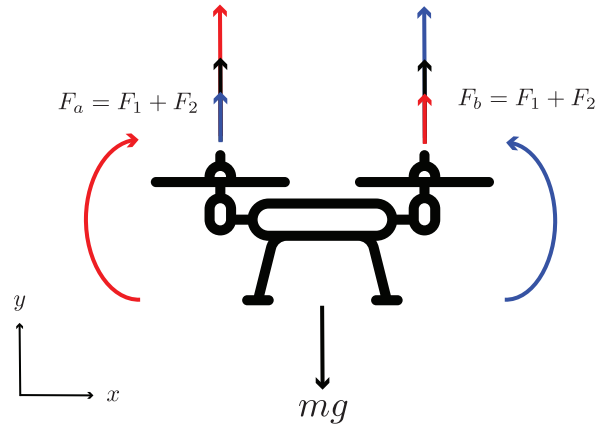

Figure 5: 2D Model of a Quadcopter.

Again using Newton's second law of motion we have the following equations to cover the motions

for the various three degrees of freedom,

$$F_y = ma_y = -\sum_{i=1}^{4} F_i \sin(\phi)$$
$$F_z = ma_z = -mg + \sum_{i=1}^{4} F_i \cos(\phi)$$
$$\tau_\phi = I_{xx} a_\phi = M_\phi$$

where F_y and F_z are the net forces in the respective y and z axes, τ_ϕ on the other hand represents the net torque about the rotational axis ϕ where I_{xx} is the inertia moment about the x axis in the 3D model. Again we have $\sum_{i=1}^{4} F_i$ representing the forces generated by the driven propellers while M_ϕ is an induced torque created by a force differential generated by the driven propellers.

The relevant ODEs for the two-dimensional model are given by

$$\frac{dv_y}{dt} = -\frac{1}{m}\sum_{i=1}^{4} F_i \sin(\phi)$$
$$\frac{dv_z}{dt} = -g + \frac{1}{m}\sum_{i=1}^{4} F_i \cos(\phi)$$
$$\frac{dv_\phi}{dt} = \frac{M_\phi}{I_{xx}}$$

3 Functions in MATLAB

Whereas MATLAB scripts are files that contain code as you would type them directly into the command window, MATLAB functions typically require an input in order to be executed. MATLAB functions can be more flexible than scripts, such that the result of running your code can vary based on your inputs, without the need for you to manually modify your files.

An example of the syntax of a MATLAB function file is as follows:

```
function output = function_name(input)
    auxiliary_variable = 1;
    output = auxiliary_variable + input;
end
```

Note that a MATLAB function only has access to variables in its own (invisible) instanced workspace. Variables declared via an input or within the function are a part of the instanced workspace, and, with the exception of the output, that instanced workspace is cleared after the function is done running. You can observe this when running `function_name.m`, such that the auxiliary variables do not appear in the workspace.

You can call and execute scripts and functions within other files, as long as they are within your current directory. You may also think of commonly used pre-defined functions, such as `plot()` or `sin()`, as always being in your directory.

4 Simulation in MATLAB via Forward Euler and `ode45`

One of MATLAB's most common uses in industry is virtual model design and simulation. MATLAB provides a rich suite of tools that allows us to model and simulate a number of different models including

quadrotors! Of the more common uses for MATLAB, is solving ordinary differential equation. In particular, one of the problems that MATLAB allows us to easily solve is of the form

$$\frac{dx}{dt} = f(t, x), \qquad x(t_0) = x_0 \tag{12}$$

where $\frac{dx}{dt} = f(t, x)$ is our ordinary differential equation that is function of time t and our system *state* x while $x(t_0) = x_0$ defines our *initial condition* that is the state of the system at time $t_0 = 0$.

To simulate an ordinary differential equation using MATLAB, we need to provide MATLAB with two files:

1) a file that defines the ordinary differential equation ($\frac{dx}{dt} = f(t, x)$) we would like to solve as a MATLAB function,

2) a script that solves our differential equation using either the Forward Euler method or `ode45` that includes details on the limits of integration or the simulation horizon $[t_0, t_f]$ (t_f refers to the final time) and the initial condition $x(t_0) = x_0$.

Now, suppose we want to simulate the position and velocity of a simple 1D point-mass system in the x coordinate governed by Newton's second law of motion $F = ma$. Let $x(t)$ define the position of our point-mass at time t and let $v_x(t)$ define the velocity of our point-mass at time t. Recall, that the derivative of our velocity function $v_x(t)$ yields the acceleration of the point-mass, i.e., $\frac{dv_x}{dt} = a_x(t)$. Thus using we have the follow system of ordinary differential equations

$$\begin{aligned} \frac{dx}{dt} &= v_x(t) \\ \frac{dv_x}{dt} &= \frac{F}{m} \end{aligned}$$

where F is a constant force acting on the point-mass, and m is the mass of the point-mass particle itself. Before we code this into a digestible form to be read by MATLAB, let us define the *vector* $\mathbf{x}$ where

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} x(t) \\ v_x(t) \end{bmatrix}$$

taking the derivative of $\mathbf{x}$ with respect to t gives

$$\frac{d\mathbf{x}}{dt} = \begin{bmatrix} \frac{dx}{dt} \\ \frac{dv_x}{dt} \end{bmatrix} = \begin{bmatrix} v_x(t) \\ \frac{F}{m} \end{bmatrix}$$

better yet, we can substitute $v_x(t) = \mathbf{x}_2$ which gives

$$\frac{d\mathbf{x}}{dt} = \begin{bmatrix} \mathbf{x}_2 \\ \frac{F}{m} \end{bmatrix} \tag{13}$$

Now, to simulate this system we need to first define our system of equations as a dedicated MATLAB function, let's call it `myODE.m`, we would then write

```
function dxdt = myODE(t,x)

    m = 1;
    a_x = F/m;

    dxdt = [x(2); a_x;];
end
```

where `dxdt` is our coded system of equations from (13). Then to solve this system of equations and effectively simulate our system, we create a separate file called `ode_sim_euler.m` that uses the Forward-Euler method or `ode45` to solve the system for our desired simulation horizon $[t_0, t_f]$. first, the code for the Forward-Euler method

```
% clears all variables from previous executions of scripts
clear all

% Initial time
t_0 = 0;

% Final time
t_f = 5;

% Number of time steps
N=10;

% Initial conditions
i_c = [0; 0;];

%Time step
delta = (t_f-t_0)/N; % Time step

t(1) = t_0;
x(:,1) = i_c;

% For loop, sets next (t,x) values
for k=1:N

    % Updates our time t according to the step size delta
    t(k+1) = t(k) + delta;

    % Calls the function f(t,x) = dx/dt
    x(:,k+1) = x(:,k) + delta*myODE(t(k),x(:,k));

end

% Plots of our simulation
figure(1);
plot(t,x(1,:));
xlabel('Time t');
ylabel('x');
legend('x')

figure(2);
plot(t,x(2,:));
xlabel('Time t');
ylabel('v_x');
legend('v_x')
```

where `t_0` and `t_f` define our simulation horizon $t_0 = 0$ and $t_f = 5$, respectively. Our initial conditions $\mathbf{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ are given in `i_c`. We then define our desired time step `delta` and execute the 'for-loop' that solves our system using the Forward-Euler method. Finally, the rest of the code generates plots of our simulations.

Solving the system using `ode45`, requires the following code

```
clear all

% Simulation horizon
time_window = [0, 5];

% Initial conditions
i_c = [0; 0;];

% Calling the ODE solve
[t,x] = ode45(@myODE, time_window, i_c);

% Plots of our simulation
figure(1);
plot(t,x(:,1));
xlabel('Time t');
ylabel('x');
legend('x')

figure(2);
plot(t,x(:,2));
xlabel('Time t');
ylabel('v_x');
legend('v_x')
```

where `time_window` is again our simulation horizon $t_0 = 0$ and $t_f = 5$ and `i_c` defines our initial conditions. We have provided the code to simulate the systems using both methods, try running it!

5 Exercises

For the following exercises, you will use the provided MATLAB functions that simulate the dynamic equations given above. With the given functions complete the following exercises:

1. Use MATLAB to modify the value of F in `myODE()` and simulate the 1D quadrotor model for the following quadcopter modes using both the Forward-Euler and `ode45` methods:

   (a) ***ascend*** (b) ***hover*** (c) ***descend***

   Submit 2 figures, corresponding to the position and velocity, respectively, each with subplots of the quadcopter's trajectory for each of the 3 modes. Each subplot should have curves for both the Forward-Euler and `ode45` methods. Include titles, legends, and axis labels. Also, submit your modified `myODE()` function in a separate file.

2. Use MATLAB to simulate the 2D quadrotor model for the ***hover*** state using both the Forward-Euler and `ode45` methods. You will need to create a new function `myODE2D()` that captures the dynamics of a 2D quadrotor model.

   Submit 2 figures, corresponding to the position and velocity, respectively, of the quadcopter's trajectory. Each figure should have curves for both the Forward-Euler and `ode45` methods. Include titles, legends, and axis labels. Also, submit your created `myODE2D()` function in a separate file.

3. Use MATLAB to simulate a 1D quadrotor starts at height 0, ascends for 5 seconds, hovers for 5 seconds, and then descends for 5 seconds.

   Submit 2 figures, corresponding to the position and velocity, respectively, of the quadcopter's trajectory. Each figure should have curves for both the Forward-Euler and `ode45` methods.

Include titles, legends, and axis labels. Also, submit `.mat` files of the generated data for each method.

Note: It is very IMPORTANT to comment your code. Please make sure to include a brief explanation of the code used to complete each of the exercises. You will lose points for no comments or poorly commented code.