ECE 8 - Lab 4: Introduction to Quadcopters (Part 2) and CoppeliaSim

November 15, 2022

1 Introduction

In this lab, we continue to introduce you to the dynamics of quadcopters – particularly in three dimensional space. Then, we will bring you to CoppeliaSim, a virtual reality simulation software that allows the simulation of many robotic systems including quadcopters. CoppeliaSim works with MATLAB to simulate your quadcopter in the 3D coordinate system.

1.1 Lab Objectives

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

1. Understand the motion of a quadcopter in the 3D coordinate system.
2. Know how to work with MATLAB and CopelliaSim.
3. Identify the variables, parameters, and functions in MATLAB to simulate a quadcopter in CoppeliaSim.
4. Collect the position of a quadcopter in the 3D coordinate system and plot it over time in MATLAB.
5. Write a program in MATLAB that sets the target position and steers the quadcopter to the target in CoppeliaSim.
6. Write a program in MATLAB that enables the quadcopter to fly in a circle in CoppeliaSim.

1.2 References and Background Reading

`https://www.coppeliarobotics.com/helpFiles/`

1.3 Necessary Files, Tools, and Equipment

- MATLAB
- CoppeliaSim
- Files in the `Lab 4` folder
  - Files in the `Api` folder (seven `.m` files, one `.dll` file, and one `.dylib` file)
  - `Lab4.ttt`

– `Part1.m`, `Part2.m`

Also, please review the material covered in Lecture 13.

1.4 Grading Rubric

- Exercise 1 is worth 3 points.
- Exercise 2 is worth 3 points.
- Exercise 3 is worth 3 points.
- Properly commented code is worth 1 point.

2 Quadcopter Dynamics

In the previous lab, we introduced the basic working principle of quadopters. In order for a quadcopter to rise into the air, a force must be created working against gravity. Quadcopters use rotors for propulsion and control. Spinning blades push air down. As Newton's third law says, all forces come in pairs, which means for every action force there is an equal (in magnitude) and opposite (in direction) reaction force. Hence, as the rotor pushes down on the air, the air pushes up on the rotor. The faster the rotors spin, the greater the lift and vice-versa. Quadcopters have also the ability to move left and right, and go forward and backward. By tilting a quadcopter to the left/right, or backward/forward, the quadcopter starts moving horizontally.

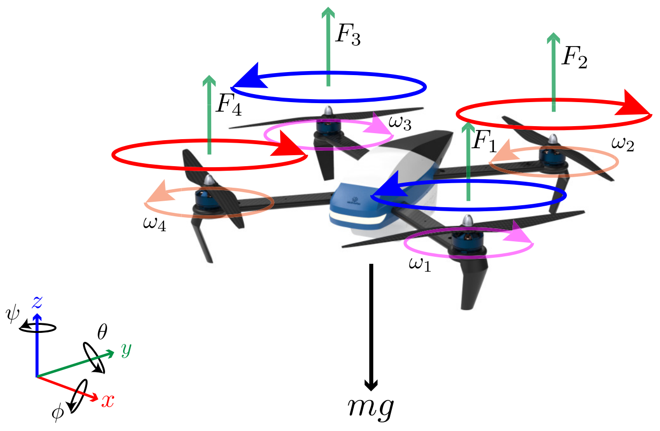


Figure 1: 3D Model of a Quadcopter.

In the three dimensional quadcopter model, quadcopters have six ways to move; namely, three *translational* motion x, y, and z and three *rotational* motion ϕ(roll), θ(pitch), and ψ(yaw). We recall two coordinate frames, a *body* frame and an *inertial* frame that describes the space in which a quadcopter moves. In Figure 1, we find an *inertial* frame as well as a *body* frame. We consider the *body* frame of a quadcopter is in alignment with the *inertial* frame at the initial position. Let's now move on to understand the motion of a quadcopter in the three dimensional space.

2.1 Vertical Motion

First, we consider the vertical motion of a quadcopter. Whenever a quadcopter is stationary, it is in alignment with the *inertial* frame. This means that the gravity acts along the quadcopter's Z-axis which is in same direction as the vertical motion of the quadcopter. Then, we can describe three vertical motion of a quadcopter: hover, ascend, or descend. To hover, the net force or thrust of the four rotors pushing a quadcopter up must be equal to the gravitational force pulling it down. If a quadcopter wants to move upwards, it just needs to set proper propeller rotating speed so that there is non-zero upward force. When a quadcopter is descending, the quadcopter decreases propeller rotating speed so the net force is downward. To sum up, when a quadcopter wants to move vertically, it needs to generate appropriate force (total thrust divided by four) on each propeller

2.2 Horizontal Motion

Now, we consider the horizontal motion of a quadcopter. When a quadcopter wants to move in either x or y direction in the *inertial* frame, it makes respecting ϕ(roll)/θ(pitch) angle along with generating required force. For example, Figure 2 shows a side view of a quadcopter moving a particular direction in the horizontal plane.

- **Sideways** - By adjusting the *roll* movement of a quadcopter, the quadcopter moves sideways (in y direction). We tilt the quadcopter to the left or the right by speeding up the rotors on one side of the quadcopter and slowing them down on the other side.
- **Forwards/Backwards** - By adjusting the *pitch* movement of a quadcopter, the quadcopter go forward or backward (in x direction). Here, we tilt the quadcopter forward or backward in the same manner as the roll movement.

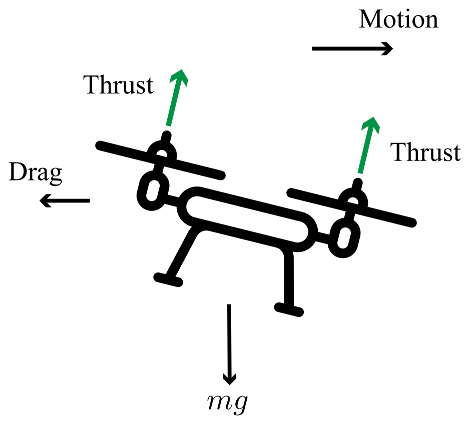


Figure 2: Horizontal motion of a Quadcopter.

2.3 Rotational Motion

Now, we consider the rotational motion of a quadcopter – particularly the *yaw* movement of a quadcopter. We know a quadcopter has two sets of rotors rotating in opposite directions (Figure 3). When a quadcopter remains facing one direction, turning neither left nor right, and not spinning on its center of gravity, the total angular momentum of the quadcopter is zero; and thus, if you want to rotate a quadcopter, the total angular momentum should be changed. The angular momentum of a quadcopter depends on how fast the rotors spin. Clockwise rotors have a positive angular momentum and counter-clockwise rotors have a negative angular momentum. For example, to rotate a quadcopter in a clockwise direction, the quadcopter increase the spin of its clockwise-spinning rotors (i.e., rotor 2 and rotor 4) and decrease the spin of its anti-clockwise rotors (i.e., rotor 1 and rotor 3).

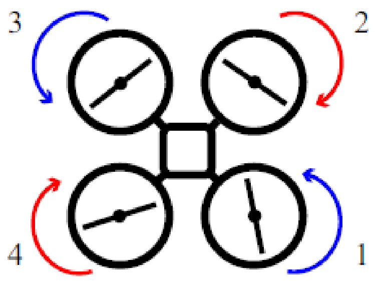


Figure 3: Rotating motion of a Quadcopter.

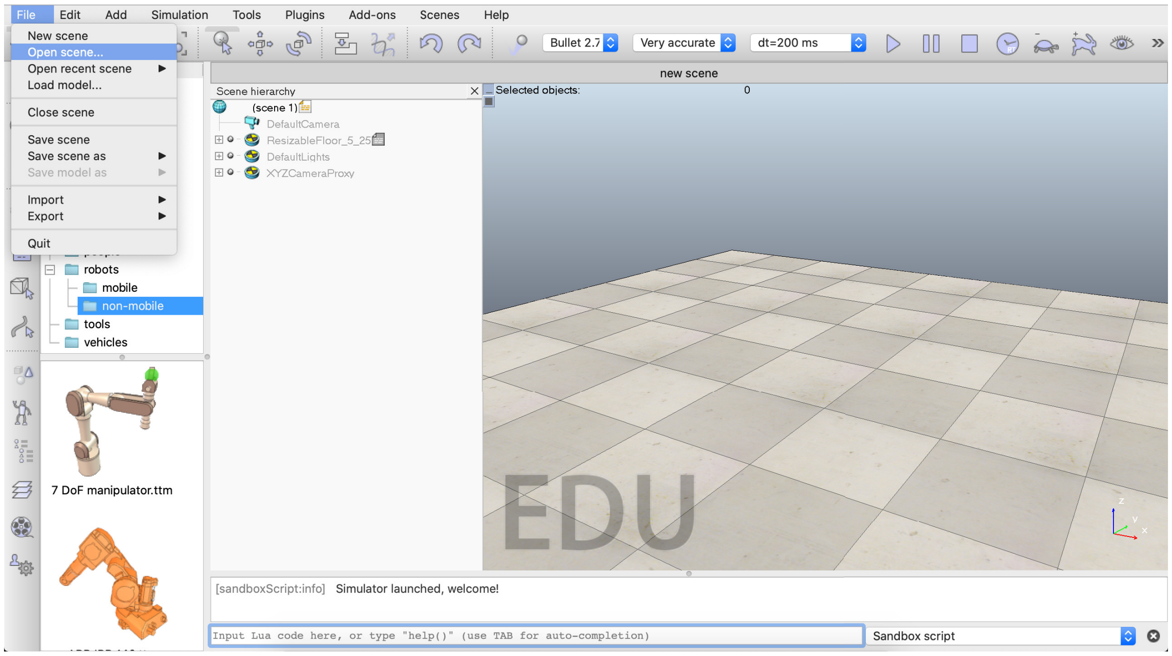
3 Getting Started with CoppeliaSim

Please have a look at the video included in the Lab4 folder for a brief introduction on CoppeliaSim.

3.1 Starting a CoppeliaSim session

Start by opening up CoppliaSim. When you launch the CoppeliaSim application, a default window appears. Then, you choose **File** → **Open scene...** and open the "`Lab4.ttt`" scene.

- **Application Window** - This is your default window as well as your main window. It is used to display, edit, simulate and interact with a scene (which contains your quadcopter and the target in Figure 7). The left and right mouse buttons, the mouse wheel as well as the keyboard have specific functions when activated in the application window.



```
[sandboxScript:info]   Simulator launched, welcome!
```

```
Input Lua code here, or type "help()" (use TAB for auto-completion)
```

- **Toolbars** - The toolbars present functions that are often used. You will change the camera view and manipulate the target position.

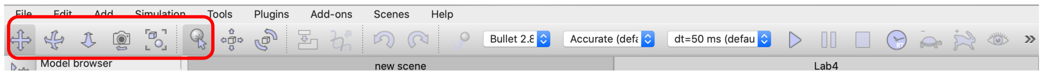


Figure 4: Toolbar buttons for camera navigation

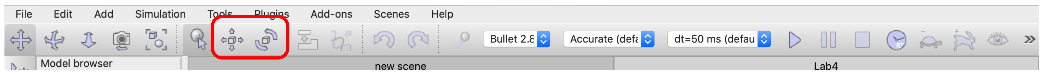


Figure 5: Toolbar buttons for object manipulation

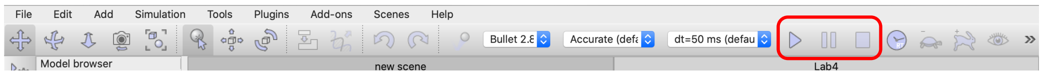
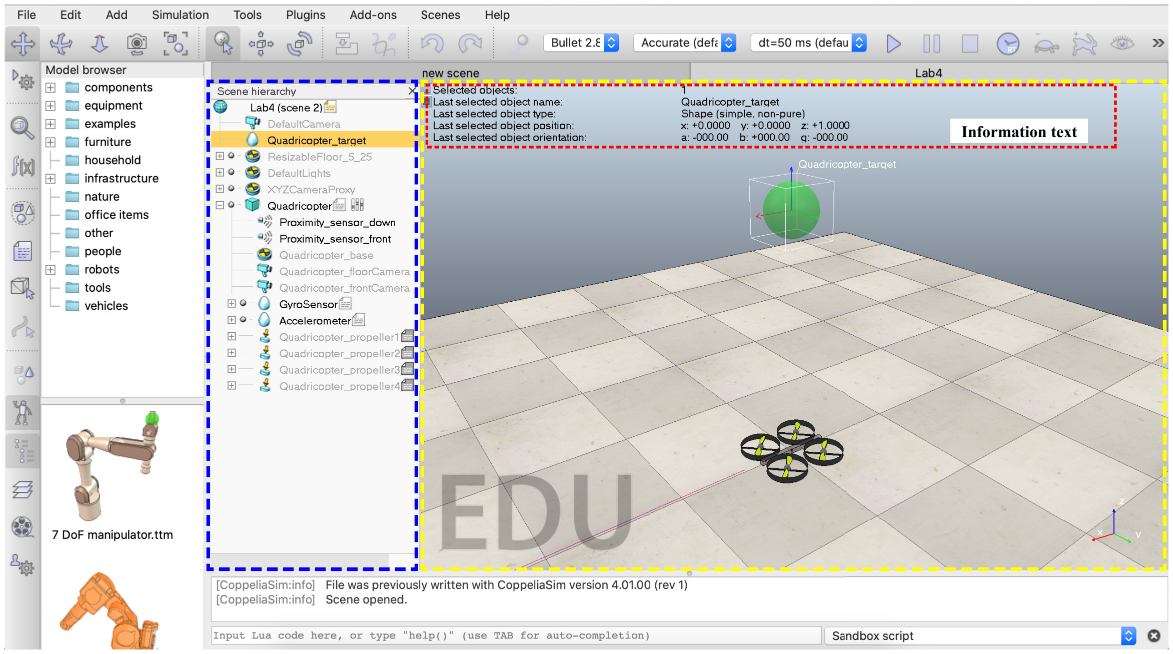


Figure 6: Toolbar buttons for simulation start/pause/stop

- **Scene Hierarchy** - This window displays the content of a scene (i.e., all objects composing your scene). After you load the "`Lab4.ttt`" scene, your scene hierarchy displays your target(green ball) and your quadcopter (Figure 7).

- **Information Text** - This displays information related to current object/item selection to running simulation states or parameters (Figure 7).



```
Selected objects:                   1
Last selected object name:          Quadricopter_target
Last selected object type:          Shape (simple, non-pure)
Last selected object position:      x: +0.0000   y: +0.0000   z: +1.0000
Last selected object orientation:   a: -000.00   b: +000.00   g: -000.00
```

```
[CoppeliaSim:info]   File was previously written with CoppeliaSim version 4.01.00 (rev 1)
[CoppeliaSim:info]   Scene opened.
```

```
Input Lua code here, or type "help()" (use TAB for auto-completion)
```

Figure 7: User interface of the CoppeliaSim application

3.2 Sensors

After you open the "`Lab4.ttt`" scene in CoppeliaSim, you can find sensors that are attached to your quadcopter in the scene hierarchy. For example, we introduce three different types of sensors that are typically used in quadcopter applications.

- **Accelerometer** - This sensor detects the *translational* movement of your quadcopter. It detects acceleration in a straight line along the X,Y,Z-axes. It measures acceleration along those axes but not around.
- **Gyroscope** - This sensor detects the *rotational* movement of your quadcopter. It detects angular velocities around the X,Y,Z-axes, respectively. An accelerometer measures the linear motion along an axis, whereas a gyroscope measures the rotational motion around an axis.
- **Proximity Sensors** - This sensor is used to determine distance/proximity of an object without any physical contact involved.

4 Connect MATLAB to CoppeliaSim

4.1 Communication with CoppeliaSim

The `Api` folder contains a set of files that help you communicate with CoppeliaSim in MATLAB.

Using the functions `initializeComm()` and `uninitializeComm(*, *)`, you can start to communicate with CopelliaSim and kill the connection to CopelliaSim. For example,

```
[ret_status, sim, clientID] = initializeComm();
      uninitializeComm(sim, clientID);
```

4.2 Variables and functions in the example codes

In the example code `Part1.m`, you can find the variables that store your timestamps using your system time. With `startTime` and `currentTime`, you can calculate the time that has elapsed.

The array `positions` has dimension $n \times 3$. At each timestamp, the first column stores the X position of a quadcopter, the second column stores the Y position of a quadcopter, and the third column stores the Z position of a quadcopter.

- Single values – `startTime`, `currentTime`
- An array – `positions`

In the example code `Part2.m`, you find the variable `position` and the functions `setObjectPosition(*, *, *, position)` and `pause()`. By calling the function `setObjectPosition(*, *, *, position)`, you can set the target position as the values in the vector `position`.

- A vector – `position`
- Functions – `setObjectPosition(*, *, *, position)`, `pause(*)`

5 Exercises

1. Use CoppeliaSim to move your target (the green sphere) in a circle. Modify `Part1.m` to record the position of the quadcopter in a vector and create a 3D plot of it.

   Submit your modified `Part1.m` script, alongside your created plot, in a published PDF.

2. Modify `Part2.m` so that the quadcopter in CoppeliaSim flies up to (0,0,3), and then back to (0,0,1). Create two figures: one with a 3D plot of the quadcopter's position in all dimensions, and another with a 2D plot of the quadcopter's position for the y and z coordinates. Hint: Make sure your quadcopter has enough time to move to the target, or else it will crash.

   Submit your modified `Part2.m` script, alongside your created plot, in a published PDF.

3. Create a copy of `Part2.m`, such that the quadcopter in CoppeliaSim flies in a square on the xz-plane. Create a 3D plot of the quadcopter's position in all dimensions.

   Submit your modified script, alongside your created plot, in a published PDF.

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