Johnathan Larymore

4/17/19

PHY 420

Research Report

This paper is to delve into the engineering challenges of developing a hover drone with the aid of a computer simulation model. This 2D Drone simulation was programmed using the python language and displayed via VPython. This 2D simulation, while impractical in reality, is important as it allows insight on the behavior of a real 3D drone.

The first place to start when making a computer model of a physical system is to first interpret the physical system and associate the appropriate physical equations of the model. The drone's center of mass will be treated as a point mass which obeys Newtons Laws. This center of mass will also be treated as the center of a rigid body object which behaves according to the rotational representation of Newtons Laws.

Although incomplete, Newtons Laws can be used to approximate accurate results for the position, velocity and acceleration of an object, with a known mass undergoing a net force. Knowing these net forces and applying Newtons 2^{nd} Law, one can integrate with respect to time determine the change in velocity and again to determining the displacement. When given initial values for both velocity and position in addition of knowing what forces are present, one can determine an object's coordinates at all times. This can be achieved by a variety of numerical time step integration techniques from an imported package or algorithms such as the Euler and Runge-Kutta, which are better suited as they require no additional imports and can be reproduced in any other code platform.

In constructing this model, Python was used as it is an interpretive language and easier understood. VPython works in conjunction with the base Python language and was used because of its ability to create and handle objects along with its visual capabilities. To iterate the motion of the drone, the Runge-Kutta algorithm was chosen. This time step method, while slower than the Euler method, provides a more precise solution than the Euler method, with minimal error due to the size of the time step. This is accomplished by performing a weighted average of four mini steps for each time step. Within the Runge-Kutta method, Newtons laws will be applied to the system, and it will calculate the new the position, velocity, and acceleration along with the rotational values of the drone from their previous values.

Setting up the code first consisted of using a Python cell to declare constants such as gravity, mass, and the moment of inertia, which was approximated to be that of a solid cylinder rotated about its center. In this same cell, a function was created to calculate to Runge-Kutta time update. Within this function it is important to calculate all values simultaneously as the direction of the force would depend on the force and the rotational components of the rigid body. Additional functions were created to determine the acceleration in both the X and Y directions and another to determine the angular torque. This first cell encompasses all the physical laws wished to be applied on the simulation.

Cell 2 is a VPython cell which is used to give a visual display of the drone simulation. A box object is created for the drone with desired length and width to represent the drone. The time step values and initial conditions are also declared within this cell. We will choose to have our systems centered at the origin with all the components of motion at rest. Also, a small time step value is needed to improve accuracy. A slider widget was also inserted to adjust the force for

each thruster. At the end of the cell, a while loop is constructed to iterate the time update of the simulation.

Now with a model code constructed, the first test can be performed to verify its accuracy to that of expected theoretical values of known systems. The first test was to verify that the drone acts the same as a free-falling object. This was done by applying only the gravitational force in the vertical direction. Within the wile loop the simulation was set to run and then stop after ten seconds. At this time, the drone's position and velocity were recorded and compared to the corresponding expectational values which can be obtained using the Kinematic equations for contact acceleration. Test one successfully verified that the drone acts as an object in free fall when no other forces are applied.

The second test was to ensure that the drone behaved as a rigid body. This was done by first temporarily zeroing out the accelerations in both the vertical and horizontal directions to ensure only rotational motion would occur. A force of 1N from a thruster was applied to one side of the drone resulting in a net torque of 5Nm which caused the drone to rotate. Similar to test one, the motion of a rigid body with constant angular acceleration was compared to the expected values of the rotational kinematic equations. Test two successfully confirmed that the drone behaved as a rigid body of equal net torque and moment of inertia.

Now that the virtual drone appears to obey Newtons laws of motion for a point mass and a rigid body, it was then time to attempt to fly the drone. Within the Python cell the acceleration for both the x and y components, the force of the thrusters was inserted. The slider for the thruster force were altered within the VPython cell. Each thruster was given a range between zero and twice the magnitude of the drone's weight and set initially to half the drone's weight. These values are expected to make the net force and net torque zero. Which would mean the

drone would hover in a stationary position, successfully passing test 3. Test 4, which was flying the drone, came immediately after test 3. While movement was created, this test displayed that alterations were needed for increase stability as the drone was uncontrollable.

In the first alteration the code was critiqued to minimize the torque this was done by making the thruster force range smaller. The maximum value was set to .55 x the weight of the drone and the minimum value was set to .45 x the weight of the drone. Test 5 displayed that the alteration proved successful in making the drone less volatile but in doing so the vertical accelerating capabilities decreased significantly which caused the drone to slowly fall out of control. The thrusters were then set to a maximum value of .75 x the weight of the drone and the minimum value was set to .4 x the weight of the drone. This yielded a better control than both the previous trials.

In the second alteration, the torque was minimized by increasing the moment of inertia. This can be understood as simply shifting weight away from the center of mass. This was altered by editing the moment of inertia in the Python cell. Test 6 displayed that this alteration yielded the best stability without losing vertical acceleration power. Even with these alternations, it became apparent more complex control systems are needed for ideal stability.

The findings of this research display the value of computer modeling. While this 2D drone simulation may be far from being realistic, it has given insight on how aerial systems behave without having to do hands-on work. With this research, a base model for a 2D drone was created and explored. Although the model can be enhanced to 3D or including forces such as air resistance, it was still possible to determine that aerial stability of a drone depends on the distribution of its mass as much as the force from the thrusters.