Dartmouth COSC 89/189 Assignment 4
Quadrotors: rigid-body simulation and control

Due on November 13

1 Introduction

In this assignment, you will learn how to design the controller of a popular robot: quadrotors. In the first part of the assignment, you will develop a rigid body simulator and implement a hierarchical PID controller in simulation. In the second part of this assignment, after you verify the performance of your controller design in simulation, you will deploy it on a popular palm-sized quadrotor: the Parrot Rolling Spider.

2 Simulation

In this section, you will implement the rigid body dynamics and the position controller for a quadrotor in simulation. We have provided you some skeleton code, including the quadrotor model, the sensors, and the control loop. With the initial code, the virtual quadrotor simply stays at the origin forever. After you finish the assignment, you will see it take off from the ground, fly to four target positions one by one in a diamond shape (defined by the four red dots) , and fly back to the origin with the existence of the environment noise.

All of your implementation will be in `MultiCopter.h`. **For 1.1 and 1.2, please use `flag = 0` in `main.cpp`** as indicated in the comments.

2.1 The linear motion

The first task for you is to implement the linear motion of the copter inside the `Advance_Rigid_Body` function:

```
// -- LV1 TASK: 3.1 --
rigid_body.position = old_p;
rigid_body.velocity = old_v;
VectorD net_force=VectorD::Zero();

// -- Your implementation starts --
// -- Your implementation ends --
```

Your task includes computing `net_force` and using it to update the position and velocity of the center of mass. Read the comments in the code carefully if you are unsure how to do it. After you implement this part, you will see that the quadrotor first takes off then oscillates along the z axis with the orientation unchanged, i.e., it translates like a single mass point (Do not forget to set `flag=0` in `main.cpp`).

2.2 The angular motion

Next, we will add the angular motion to the quadrotor simulator:

```
// -- LV1 TASK 3.2 --
rigid_body.orientation = old_orientation;
rigid_body.omega = old_omega;
VectorD body_net_torque = VectorD::Zero();

// -- Your implementation starts --
// -- Your implementation ends --
```

You need to first compute the net torque then use it to update the orientation and angular velocity. You can find more details and tips in the comments.

After you finish this part, you can verify your implementation by commenting out your code for 1.1 and observing the quadrotor spin along the x axis. Do not forget to uncomment your linear motion for the remaining tasks!

2.3 The altitude controller

Now we will walk you through the process of implementing the hierarchical controller to stabilize its motion. **For this and the following control task, please use `flag=1` in `main.cpp`.**

Let's start with the altitude controller:

```
////LV2: PD Controller
real AltitudeController(const real total_weight, const real z_ref, const real z,
    const real z_rate)
{
  const real P_z = 0.2;
  const real D_z = 0.3;
  // -- TASK 3.3: Implement your altitude controller here --
  real total_thrust = 0.0;

  // -- Your implementation starts --
  // -- Your implementation ends --

  return total_thrust;
}
```

Here `total_weight` is a positive number equal to mg, `z_ref` is a negative number representing the reference altitude, `z` is the altitude from the sensor, and `z_rate` is the time derivative of `z`, which corresponds to $\dot{z}$ in the equation. The function should return a positive `total_thrust` that matches T_z in the equation.

Put your implementation inside this function. Once it is done, you should see your quadrotor take off and follow the diamond trajectory while maintaining the altitude well. You will also notice that the copter slowly spins along the z axis due to the environmental noise, which we will fix shortly. As a sanity check, you can set `D_z=0` and will notice the height of the quadrotor oscillates, a typical phenomenon of a P controller.

2.4 The attitude controller

Finally, let's fix the self-spinning by implementing a yaw controller in the following function:

```
real YawController(const real yaw_ref, const real yaw, const real yaw_rate)
{
  const real P_yaw = 0.004;
  const real D_yaw = 0.3 * 0.004;
  // -- TASK 3.4 --
  // Implement your yaw controller here.
  real total_torque = 0.0;

  // -- Your implementation starts --
  // -- Your implementation ends --
  return total_torque;
}
```

You need to calculate and return the correct `total_torque` based on the rigid-body dynamics equation in the course notes. The variables `yaw_ref`, `yaw`, `yaw_rate` correspond to ψ_{ref}, ψ, and $\dot{\psi}$ in the equation respectively. We have tweaked the P and D gain (`P_yaw` and `D_yaw`) for you. Your task is to calculate τ_ψ and assign it to `total_torque`.

After you finish this function, the quadrotor should take off, maintain its altitude and heading, and counteract any horizontal drift caused by the noises (For simplicity, we have already implemented the horizontal controller for you). This concludes all your simulation tasks.