AE483 Lab Manual: Lab #4
Collision Avoidance

T. Bretl

November 10, 2019

1 Goal

Your goal in the next three weeks will be to design, implement, and test a planner that—along with your controller from the previous lab—makes the drone move from one position to another while maintaining zero yaw angle and avoiding obstacles. You will need to work steadily and to be organized in order to achieve this goal. We recommend the following schedule:

- Complete all of Section 2 in lab during Week 1.
- Complete all of Section 3 in lab during Week 2.
- Complete all of Section 4 in lab during Week 3.

It is possible to work ahead—we strongly encourage you to try. One reason to work ahead is that, very soon, you will all start thinking about and working on your final projects (as you are finishing up this fourth lab). You will write a report that describes your approach to collision avoidance and your results that will be submitted as a group no later than 11:59PM on Friday, November 22, 2019 (the day before fall break). Details will be posted to slack.

2 Simulation

2.1 Implement

One strategy for collision avoidance is to define a potential function $f(o_{\text{desired}})$ that is small at the goal and large near obstacles and to choose the desired position by gradient descent:

$$o_{\text{desired}}(t+\Delta t) = \begin{cases} o_{\text{desired}}(t) - k_{\text{des}} \nabla f(o_{\text{desired}}(t)) & \text{if } \|k_{\text{des}} \nabla f(o_{\text{desired}}(t))\| \leq b_{\text{des}} \\ o_{\text{desired}}(t) - b_{\text{des}} \left(\dfrac{\nabla f(o_{\text{desired}}(t))}{\|\nabla f(o_{\text{desired}}(t))\|} \right) & \text{otherwise,} \end{cases}$$

where

$$f(o_{\text{desired}}) = f_{\text{att}}(o_{\text{desired}}) + \sum_{i=0}^{n_{\text{obs}}} f_{\text{rep},i}(o_{\text{desired}})$$

and

$$f_{\text{att}}(o_{\text{desired}}) = \begin{cases} \frac{1}{2} k_{\text{att}} \left\| o_{\text{desired}} - o_{\text{goal}} \right\|^2 & \text{if } \left\| o_{\text{desired}} - o_{\text{goal}} \right\| \leq b_{\text{att}} \\ k_{\text{att}} b_{\text{att}} \left(\left\| o_{\text{desired}} - o_{\text{goal}} \right\| - \frac{1}{2} b_{\text{att}} \right) & \text{otherwise} \end{cases}$$

$$f_{\text{rep,i}}(o_{\text{desired}}) = \begin{cases} \frac{1}{2} k_{\text{rep}} \left(\frac{1}{d_i(o_{\text{desired}})} - \frac{1}{b_{\text{rep}}} \right)^2 & \text{if } d_i(o_{\text{desired}}) \leq b_{\text{rep}} \\ 0 & \text{otherwise.} \end{cases}$$

In these expressions, the gradient ∇f is taken with respect to o_{desired}. In particular, the gradient of $f_{\text{att}}(o_{\text{desired}})$ with respect to o_{desired} is

$$\nabla f_{\text{att}}(o_{\text{desired}}) = \begin{cases} k_{\text{att}} \left(o_{\text{desired}} - o_{\text{goal}} \right) & \text{if } \left\| o_{\text{desired}} - o_{\text{goal}} \right\| \leq b_{\text{att}} \\ k_{\text{att}} b_{\text{att}} \left(\frac{o_{\text{desired}} - o_{\text{goal}}}{\left\| o_{\text{desired}} - o_{\text{goal}} \right\|} \right) & \text{otherwise.} \end{cases}$$

Similarly, the gradient of $f_{\text{rep,i}}(o_{\text{desired}}))$ with respect to $o_{\text{desired}})$ is

$$\nabla f_{\text{rep,i}}(o_{\text{desired}}) = \begin{cases} -k_{\text{rep}} \left(\frac{1}{d_i(o_{\text{desired}})} - \frac{1}{b_{\text{rep}}} \right) \left(\frac{1}{d_i(o_{\text{desired}})} \right)^2 \nabla d_i(o_{\text{desired}}) & \text{if } d_i(o_{\text{desired}}) \leq b_{\text{rep}} \\ 0 & \text{otherwise.} \end{cases}$$

The number of (spherical) obstacles is n_{obs} and the distance to the i'th obstacle is $d_i(o_{\text{desired}})$. Implement this planner by modifying the function `planner()` in `ae483_03_simulate.m`, starting with the template provided in class:

https://uofi.box.com/s/j0zy19k38wkeh60qa96jgg9yq0irae83

You should, of course, also modify this template so that it includes your own controller and your own parameter values. Note that `ae483_00_test.m` includes the same "standard flight test" that you saw in Lab 3, but instead of using the waypoints to choose the desired position directly, we are now using the waypoints to choose the *goal* position—the planner is responsible for making the desired position track the goal position while avoiding obstacles (the controller is still responsible for making the drone track the desired position).

2.2 Test

Tune the parameters that govern the planner (k_{att}, b_{att}, etc.) so that the drone follows each trajectory that is defined by the three flight tests in `ae483_00_test.m` ("takeoff and land," "back and forth," "standard flight test") without colliding with obstacles. You should try placing obstacles both by hand—in diabolical locations, perhaps—and at random, by modifying the code in `ae483_03_simulate.m`. (Why did you choose the parameters you did? What bad things happen with different values? Does the drone ever get stuck? Why? Does the drone ever leave its bounding volume? Why?) One parameter you should pay particular attention to is the radius of the bounding volume—the sphere—that is centered at the desired position and that the drone is always supposed to remain inside. What should the radius of this sphere be? The "right answer" to this question depends very much on the performance of your hover controller—for this reason, I would expect every single group to be choosing a different radius. Show your TA at least one movie from each of the three flight tests, with your working planner.

3 Hardware

3.1 Implement

Follow the instructions in Appendix G to add both an obstacle and a planner to your ground-station. Implement in C, in the function `planner()` in `planner.c` in your ground-station code, the same method of collision avoidance that you implemented previously in MATLAB (Section 2).

3.2 Test (simulation)

Your TA will show you how to run a simulator that calls the function `planner()` that you just wrote in C. Use this simulator both to debug your implementation and, if necessary, to further tune your parameters (k_{att}, b_{att}, etc.) so that the quadrotor reaches the goal successfully most of the time. (Did you have to use different parameter values then in your MATLAB simulation?)

3.3 Test (hardware)

You should have a working controller from Lab #3 and a working planner from Section 3.1 that you debugged and tuned in Section 3.2. Put the obstacle somewhere in the drone's workspace. Fly the drone, verifying (both by eye and by looking at the logged data) that it tracks the goal while avoiding the obstacle. Please be sure also to record video of your flight. Note that both the goal and the obstacle can move during this process—your planner should continue working just fine. If your planner (implemented in C) worked in simulation but not hardware, you might consider running your code with the drone sitting unarmed on the ground. The planner, as we have discussed it, does not care where the drone *actually* is—it will happily move the desired position around, whether or not your controller is on. Logging data with the ground station running and the drone unarmed is a useful way to continue debugging your code, if necessary.

4 Analysis

At minimum, your analysis must involve the following two sets of hardware experiments:

- Do a hardware experiment in which your planner works well. This experiment must involve a situation in which a collision would have happened without the planner. Your results must include a picture-in-picture video that compares the real flight to a simulated flight.
- Do a sufficient number of hardware experiments to identify all possible ways in which your planner might fail (i.e., all *failure modes* of your planner). Be creative—and careful! If you can argue convincingly that a failure mode would be dangerous in hardware experiment, you can show this failure mode in simulation. You are encouraged to improve your planner in order to fix the failure modes that you uncover in these experiments, as best you can.

For each experiment, you should consider the performance of your whole system—both the planner (did the desired position track the goal position, did the bounding volume around the desired position always avoid the bounding volume around the obstacle position) and the controller (did the drone always remain inside the bounding volume around the desired position).