AE483: Lab #4
Collision Avoidance

T. Bretl

November 6, 2017

1 Goal

Your primary goal in the next three weeks will be to design, implement, and test (in simulation and in experiment) a planner that—along with your controller from the previous lab—makes a quadrotor move from one position to another while maintaining a zero yaw angle and avoiding obstacles.

2 First Week

Your objective this week is to design, implement, and test a planner in simulation.

2.1 Reimplement controller

Download the MATLAB scripts `lab4_simulate.m` and `lab4_visualize.m`, as well as the data files `quadmodel.mat` and `mocapmodel.mat`. This template code is similar to what you saw in Lab #3, with a few key extensions to allow the implementation of a planner. Before proceeding, you need to repeat your control design and implementation from Lab #3, using this new template code. In particular, please make the following changes to `lab4_simulate.m`:

- Modify the parameters to match your quadrotor. These parameters are now packed into a `struct` called `params`. Mass is `params.m` instead of `m`, gravity is `params.g` instead of `g`, etc.
- Modify the function "`h()`" that is used by `ode45` to integrate the equations of motion. The code from Lab #3 should work perfectly here.
- Modify the function "`controller()`" to implement your controller. Your implementation from Lab #3 should still work, with a few changes to variable names—all we are doing is putting this implementation into its own function, to keep things organized. You may want to design your controller before starting the simulation (e.g., where it says "`DESIGN`" in the code) and to store x_e, u_e, and K in `params.xe`, `params.ue`, and `params.K`, respectively. These parameters will then be available to you in `controller()`. Make sure you satisfy bounds on spin rates.

Then, make the following change to `lab4_visualize.m`:

- Modify the function "`UpdateGeometry()`" exactly as you did in Lab #3.

You should now be able to run the simulation and to visualize the results (control at hover):

```
>> [data, params] = lab4_simulate;
>> lab4_visualize(data, params, 'movie.mp4');
```

2.2 Implement planner

As we have seen in class, one strategy for collision avoidance is

$$q(t+\Delta t) = \begin{cases} q(t) - k_{\text{des}} \nabla_q f(q(t)) & \text{if } \|k_{\text{des}} \nabla_q f(q(t))\| \leq b_{\text{des}} \\ q(t) - b_{\text{des}} \left(\frac{\nabla_q f(q(t))}{\|\nabla_q f(q(t))\|} \right) & \text{otherwise,} \end{cases}$$

where

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

and

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

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

In these expressions, we denote o_{desired} and o_{goal} by q and q_{goal}, as usual. The number of (spherical) obstacles is n_{obs} and the distance to the i'th obstacle is $d_i(q)$. Implement this planner by modifying the function `planner()` in `lab4_simulate.m`. You may want to use this coding pattern:

```
function o_desired = planner(t, x, o_desired, o_goal, obst, params)

    % Rename o_desired, o_goal, and the radius of the bounding volume for convenience
    q = o_desired;
    q_goal = o_goal;
    r = params.r;

    % Get attractive part of gradient
    %  if ( ... )
    %      gradf = ...
    %  else
    %      gradf = ...

    % Get repulsive part of gradient
    for i=1:length(obst)
        % Get center and radius of i'th spherical obstacle
        p = obst{i}.p;
        s = obst{i}.s;
        % Compute distance and gradient of distance to this obstacle
        %  d = ...
        %  dgrad = ...
        % Compute repulsive gradient for this obstacle and add it to total gradient
        %  if ( ... )
        %      gradf = gradf + ...
    end

    % Take a step
    %  if ( ... )
```

```
    %       q = ...
    %   else
    %       q = ...

    % Recover o_desired
    o_desired = q;

end
```

You may also want to define k_{att} as `params.k_att`, b_{att} as `params.b_att`, etc., where it says "`DESIGN`" in `lab4_simulate.m`. These parameters will then be available to you in `planner()`.

2.3 Test

Tune your parameters (k_{att}, b_{att}, etc.) so that the quadrotor (represented by the blue sphere) reaches the goal (represented by the green sphere) successfully most of the time in simulations with ten spherical obstacles placed at random. Why did you choose the parameters you did? (What bad things happen with different values?) Does the quadrotor ever get stuck? Why? Does the quadrotor ever leave its bounding volume? Why? You are encouraged to play around with your planner. Move the obstacles and add more of them. Change the goal location. Try to get other groups' planners stuck. Etc. Have fun.

2.4 In-Lab Deliverables

Confirm that you have done the following thing in lab on the first day:

- Show the TA a movie of your working planner, with ten spherical obstacles placed at random.

Please come back to the lab during office hours to finish this task, if necessary. You will implement and test exactly this same planner in experiment with the real quadrotor next week.

3 Second Week

Your objective this week is to implement your planner in C and to test it in simulation. (As soon as it works in simulation, you will be able to test in in experiment with the quadrotor.)

3.1 Implement

You should have a working implementation of your planner in MATLAB from last week (in the function `planner()` in `lab4_simulate.m`). Your first task is to implement this same planner in C in the function `PotentialField()` in `planner.c`. Your TA will show you which files to download and to modify. Sections 3.1.1-3.1.4 provide some guidance on the differences between MATLAB and C that are relevant to your implementation.

3.1.1 Conditional statements

A conditional statement in MATLAB:

```
if ( ... )
    % do something
else
    % do something different
end
```

A conditional statement in C:

```
if ( ... ) {
    // do something
} else {
    // do something different
}
```

3.1.2 Assignment

Assigning a value to a vector in MATLAB:

```
v = [3; -1; 4];
```

Assigning a value to a vector in C:

```
//
// v must have previously been declared as:
//    float v[3];
//
v[0] = 3;    // notice that indices start at 0 in C
v[1] = -1;
v[2] = 4;
```

3.1.3 Sums

Adding a value to a variable in MATLAB:

```
x = x + 5;
```

Assigning a value to a vector in C:

```
//
// x must have previously been declared as:
//    float x;
//
x += 5;      // option #1
x = x + 5;   // option #2
```

3.1.4 Norms

Computing the 2-norm of a vector in MATLAB:

```
vnorm = norm(v);
```

Computing the 2-norm of a vector in C:

```
//
// v and vnorm must have previously been declared as:
//    float v[3];
//    float vnorm;
//
vnorm = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
```

3.2 Test

Your TA will show you how to run a simulator that calls the function `PotentialField()` that you just wrote. Use this simulator both to debug your implementation and to 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 In-Lab Deliverables

Confirm that you have done the following thing in lab on the second day:

- Show the TA the simulator running with your working planner.

Please come back to the lab during office hours to finish this task, if necessary. Your exact same C code should work next week in experiments with the real quadrotor.

4 Third Week

Your objective this week is to test your planner in experiment with the quadrotor.

4.1 Control at hover

You should have a working controller from Lab #3. Proceed exactly as you did in that lab to set up and fly the vehicle, verifying that your controller still works (and tuning the gains or offsets, so that the performance of this controller is acceptable to you).

4.2 Collision avoidance (stationary obstacle)

You should have a working planner from last week. Replace the relevant parts of `planner.c` in the ground station code with your implementation. Put the obstacle somewhere between the start and goal position of the quadrotor. Fly the vehicle, verifying that the quadrotor moves from start to goal while avoiding the obstacle. Verify that data were collected and store these data for future use. Note that both the desired position of the quadrotor and the position of the obstacle are included in these data. Please be sure also to record video of your flight (e.g., with a smartphone).

4.3 Collision avoidance (moving obstacle)

Repeat your experiment, this time with a moving obstacle. You can move the obstacle however you like (for example, try moving it toward the quadrotor after the quadrotor has already reached the goal — what should happen in this case?), keeping in mind that the obstacle needs to be tracked by the motion camera system in order for it to be "visible" to the quadrotor. Have fun!

4.4 In-Lab Deliverables

Confirm that you have done the following thing in lab on the second day:

- Show the TA the quadrotor at hover.
- Show the TA the quadrotor avoiding a stationary obstacle.
- Show the TA the quadrotor avoiding a moving obstacle.

Please come back to the lab during office hours to finish each task, if necessary.

5 Report

Write a report with four sections:

- goal
- method of approach
- results
- discussion

This report should describe your approach to collision avoidance. It should include results from both simulation and experiment. At minimum, these results should (in some way) compare the desired position, the actual position, and the obstacle position as functions of time, both in simulation and experiment. Results from experiment should include at least one flight with a stationary obstacle and at least one flight with a moving obstacle. At minimum, the discussion section should explain any differences (if any) between simulation and experiment in terms of both implementation and results, should assess how "good" your planner is, and should provide a clear explanation of any problems or crashes that occurred.

You are encouraged to go beyond these requirements. Your report must be submitted as a PDF with single-space text no larger than 12-point font and with 1-inch margins. It must be a minimum of six pages and a maximum of eight pages. You are strongly encouraged, but not required, to use LaTeX to prepare your report. You must submit your report no later than 11:59PM on Friday, December 8. Submission details will be posted to piazza.