AE483 Homework #4:
Quad-Rotor Collision Avoidance

(due at the beginning of class on Monday, November 14)

Additional guidance about MATLAB implementation has been provided on piazza.

1. *Find the distance between two spheres.* One sphere has center $q \in \mathbb{R}^3$ and radius $r > 0$. The other sphere has center $p \in \mathbb{R}^3$ and radius $s > 0$.

   (a) Find the distance between these two spheres.

   (b) Find the gradient of this distance with respect to q.

   (c) Implement (a) and (b) as a function `[d,dgrad] = SphereSphere(q,r,p,s)` in your simulation code.

2. *Find the distance between a sphere and a halfspace.* The sphere has center $q \in \mathbb{R}^3$ and radius $r > 0$. The halfspace is defined by a plane that passes through the point $p \in \mathbb{R}^3$ and is normal to the unit vector $n \in \mathbb{R}^3$.

   (a) Find the distance between the sphere and the halfspace.

   (b) Find the gradient of this distance with respect to q.

   (c) Implement (a) and (b) as the function `[d,dgrad] = SpherePlane(q,r,p,n)` in your simulation code.

3. *Compute the gradient of an attractive potential function.* As we saw in class, the gradient of

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

   with respect to q is

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

   Implement this computation as `gradfatt = GetAttractiveGradient(q,q_goal,params)` in your simulation code.

4. *Compute the gradient of a repulsive potential function.* Let $d_i(q, A_i)$ be the distance between a sphere of center q and a convex set $A_i \subset \mathbb{R}^3$. As we saw in class, the gradient of

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

   with respect to q is

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

By linearity, the gradient of

$$f_{\text{rep}}(q) = \sum_{i=1}^{n_{\text{obs}}} f_{\text{rep,i}}(q)$$

is

$$\nabla_q f_{\text{rep}}(q) = \sum_{i=1}^{n_{\text{obs}}} \nabla_q f_{\text{rep,i}}(q).$$

Implement this computation as `gradfrep = GetRepulsiveGradient(q,obst,params)` in your simulation code.

5. *Implement gradient descent.* We have seen how to compute both attractive and repulsive potential functions. The total potential function is simply

$$f(q) = f_{\text{att}}(q) + f_{\text{rep}}(q).$$

To locally minimize $f(q)$, we can apply gradient descent:

$$\dot{q}(t) = -k_{\text{descent}} \nabla_q f(q),$$

where $k_{\text{descent}} > 0$ is a constant gain. We can approximate gradient descent in discrete time using Euler's method:

$$q(t + \Delta t) \approx q(t) - \Delta t \left(k_{\text{descent}} \nabla_q f(q) \right),$$

where $\Delta t > 0$ is a constant time step. When implementing gradient descent, we often cap the step size as follows:

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

where $b_{\text{descent}} > 0$ is a constant maximum step size.

   (a) Implement the computation of the gradient of the total potential function as `gradf = GetGradient(q,q_goal,obst,params)` in your simulation code.

   (b) Implement gradient descent as `q=GradientDescent(q,q_goal,obst,params)` in your simulation code. The argument "`q`" in this function's input represents $q(t)$. The argument "`q`" in this function's output represents $q(t + \Delta t)$.

   (c) Use your implementation of gradient descent to steer `o_desired` to `o_goal`. Since you have already implemented trajectory tracking in your simulation code, what this means is that you'll be steering the quadrotor to a goal position while avoiding collision.

6. *Tune the parameters governing a potential field approach to collision avoidance.* Four parameters govern the behavior of the potential field approach to collision avoidance that you have now implemented:

$$k_{\text{att}}, b_{\text{att}}, k_{\text{rep}}, b_{\text{rep}}.$$

   (a) Choose values for these parameters so that the quadrotor (represented by the blue sphere) reaches the goal (represented by the green sphere) successfully in your simulation code for Scenarios #1-#3 (uncomment the appropriate lines of your simulation code to try these scenarios). Try to choose values that make the result "look nice" (e.g., that minimize chattering or other strange behavior) and be relatively fast, while still

allowing the quadrotor to track the desired position (i.e., to stay completely inside the blue sphere). Explain why you chose the values that you did. (What bad things happen with different values?)

(b) Once you have chosen parameters, apply them also to Scenario #4. In this case, spherical obstacles are scattered randomly throughout the workspace. Try executing the code several times—you should find that, occasionally, the quadrotor will get stuck. Why does it get stuck? Can you make it more or less likely to get stuck by changing the parameter values?

7. *Propose one way to recover from getting stuck.* Now try Scenario #5. You'll find that this is a situation in which the quadrotor will get stuck, regardless of your choice of parameter values. Propose one way to recover from getting stuck. Implement your approach and show that it works, at least in Scenario #5. Will it work in all situations? Either argue that it will, or provide an example of a situation in which it won't.

8. *Create a new scenario.* Create your own scenario (i.e., set of obstacles), making it available for use in your simulation code. It could be one that you find compelling, strange, or beautiful. It could also be one that shows off your method of collision avoidance (augmented with the way in which you recover from getting stuck). Create a scenario that breaks your colleagues' method of collision avoidance!