CPSC 8810
Motion Planning

Assigned: Jan 29, 2019
Due: 11:59pm, Feb 9, 2019

Programming Assignment 2 – Local Navigation with TTC Forces *(Grading: 0–10 points)*

Problem Description

In the second assignment, you will implement the predictive TTC forces approach for local navigation that we covered in lecture 2. The approach can be considered as a variant of the PowerLaw model, as both approaches rely on forces that depend on the relative displacement of agents at the moment of a collision. You will also incorporate uncertainty in the sensed velocities of the neighbors using the isotropic formulation that we will cover next week.

In the zipped file `ttc.zip`, you will find three multi-agent navigation scenarios that you can use to test your code. I have also provided `Python` code, that reads scenarios, and visualizes the agents. Your task is to modify the Agent class in the `agent.py` file.

Basic Requirements

Your program should work as follows. It should first load a simulation scenario and set the time step Δt to be 0.05 s in the `simulator.py` file. Then, for each of the n agents you need to compute a collection of forces that determine the behavior of the agent at each simulation step, and then update its position and velocity. To do so

1. Please modify the `computForces` function in order to:

   (a) Determine the goal force of the agent as explained in the class, i.e.

   $$\mathbf{f}_g = \frac{\mathbf{v}_g - \mathbf{v}}{\xi} \tag{1}$$

   Here, we assume that the goal velocity is the unit vector pointing from the current position of the agent to its goal position scaled by the agent's preferred speed (already computed for you in the `update` function). A typical value for ξ is 0.5s.

   (b) Determine *all* neighbors of the agent that are less than d_H m away from the agent. A typical value for d_H is 5-10 units.

   (c) For each neighbor estimate the corresponding time-to-collision value, τ, between the neighbor and the agent. If there is a collision, add the corresponding repulsive force to $\mathbf{f}_g$. You should refer to the lectures, but the force should be of the form:

   $$\mathbf{f}_a = \frac{\max(\tau_H - \tau, 0)}{\tau}\mathbf{n} \tag{2}$$

   where $\mathbf{n}$ denotes the unit vector that pushes the two agents apart at the moment of impact. Typical value for the time horizon τ_H is 4s.

2. Modify the `update` function to update the velocities and positions of the agents based on the computed forces by using simple Euler integration. You should cap the velocity of each agent to its maximum speed. You may also want to cap the maximum amount of force that is applied to each agent before computing the new velocity.

3. Once you have this code working, you should extend it to account for uncertainty in the sensed velocities. In particular, you will implement the *isotropic* time-to-collision formulation that we will cover next Monday. This approach works by assuming that the error between the true and sensed relative velocities is smaller than some known constant ϵ. The uncertainty-aware approach works exactly as the TTC forces model described above. The only difference is that the time-to-collision routine needs to be slightly modified since some of the coefficients of the quadratic include additional ϵ-dependent terms. In fact, if you set $\epsilon = 0$, you should be getting the vanilla TTT forces. Of course we need to have a model of the sensing error as well. In our case, we will assume that at each time step the sensed relative velocity between two agents A and B, $\hat{\mathbf{v}}_{AB}$, is the same as the true velocity, $\mathbf{v}_{AB}$.

Please test your code on the provided `3_agents`, `8_agents`, and `crossing_agents` scenarios. Once you are happy with your results, you should export each simulation into a `csv` file by setting the `doExport` flag to `True` in `simulator.py`. For each scenario, please export two simulation files, one with $\epsilon = 0$, and one with $\epsilon = 0.2$. You should also document the parameters that you used for each simulation in a `Readme` file. As opposed to project 1, a universal set of parameters should work well here. Overall, you should be able to see much better results compared to the sampling-based approach you implemented in the first assignment.

Extra credits (3pts)

Consider implementing the following extensions:

- Implement the PowerLaw model covered in lecture 6. The approach is very similar to TTC forces. The only difference is the magnitude of the avoidance force given two interacting agents. In TTC forces, the magnitude is inversely proportional to the time that it takes for the two agents to collide, while in PowerLaw the magnitude is a bit more involved as it is derived from the gradient of the PowerLaw energy function. See the lecture for more details. Once more, for each scenario, please export two simulation files, one with $\epsilon = 0$, and one with $\epsilon = 0.2$. You can set the exponential cutoff point, τ_0 to either 0s or 3s.

- Modify the sensor error model for your isotropic uncertainty implementation by assuming that at each time step the sensed relative velocity, $\hat{\mathbf{v}}_{AB}$, is randomly perturbed from the true velocity, $\mathbf{v}_{AB}$, as follows:

  $$\hat{\mathbf{v}}_{AB} = \mathbf{v}_{AB} + \eta,$$

  where η is sampled randomly from a disk centered at $\mathbf{0}$ having radius ν. Please experiment with different values of ν and ϵ, and submit results with $\epsilon = 0.2$ and $\nu = 0.1$. As long as $\nu \leq \epsilon$, you shouldn't be seen any collisions.

- Implement the adversarial uncertainty model for the TTC forces and/or the PowerLaw. The only difference to the vanilla approaches is to use $\hat{\mathbf{v}}_{AB} - \epsilon \frac{\mathbf{x}_{AB}}{\|\mathbf{x}_{AB}\|}$ instead of $\mathbf{v}_{AB}$ upon computing the repulsive forces and time to collision. Here, $\mathbf{x}_{AB}$ denotes the relative displacement between the two agents. The sensed relative velocity $\hat{\mathbf{v}}_{AB}$ can either follow the noise model above or be the same as $\mathbf{v}_{AB}$. Please submit results with $\epsilon = 0.2$, and $\nu = 0.1$ and/or $\nu = 0$. For more details see the lectures and here (Section III.B)

Submission

Submit the assignment using Canvas. You can work in pairs if you want to. If you do so, though, please let us know in advance. Along with your code, please upload the three csv files corresponding to the three scenarios that you simulated, as well as a `Readme` file indicating the parameters that you ended up using to simulate each scenario. Since we are using multiple files, please make a directory and add all files to it. You should zip your directory and upload it to the submission system.

Help

If you get stuck, please do not hesitate to contact us for help, and stop by during the office hours. We also encourage you to post questions and initiate discussions on Canvas. Your colleagues are also there to help you.