UNIVERSITY OF MARYLAND

Homework 5

ENME 808T: Network Control Systems
Department of Mechanical Engineering | Fall 2024

Due: Monday, December 9th, 2024 by 11:59 pm

Project Description

For this assignment, we will do a real-robot implementation of some of the distributed controllers learned in class. The overall mission is a simulated *multi-robot disinfection* operation.

In this hypothetical scenario, the world has been ravaged by a global pandemic. To promote social distancing and curb the spread of the disease, there has been a boom in the deployment of robotic systems have to assist in performing tasks such as symptom monitoring, area disinfection, deliveries, etc. As a prominent roboticist, you have been tasked with the deployment of a team of robots to assist in safe disinfection efforts during the public health crisis. In particular, you are to design decentralized controllers to enable a team of robots with disinfecting certain high-traffic areas in a clinical setting.

The agents will be deployed from a disinfectant refueling station, which they must revisit after each use. For cyber-security reasons, agents are not connected to the existing communication infrastructure and are not able to communicate with one another. They are equipped with short-range radio localization equipment to detect each others position and identities, and their interactions can be modeled using a Δ-disk graph. For further security, only two agents are aware of the navigation waypoint locations, assigned at random after each refueling trip to reduce the risk of a robot being tampered with to hijack the network. Thus, agents must remain connected to ensure they will not be stranded roaming the halls.

To safely navigate the environment, agents are able to detect obstacles within their sensing range. Each agent is equipped with *eight range sensors*, uniformly spaced around the body of the robots on an inertially-fixed rotating base (i.e., in the fixed directions $\{0°, 45°, \dots, 315°\}$ from the center of the robot). Each sensor returns the distance to any object detected in the range $[\delta, \Delta]$. If no objects are detected, the sensors return a value of `Inf`. All robots share a common sense of direction. Within the target disinfection area, all agents begin to spray the disinfecting solution in a radius of δ around each robot. Beacons in the room (the position of the four corners) allow for all agents to be aware of the disinfection domain.

There are four sub-missions to clear. For each sub-mission and between each waypoint, you may design a formation graph topology under each sub-mission case in the `getFormationGraph` script. This script takes in the current scenario and navigation waypoint, and should return a weighted adjacency matrix, with nonzero entries denoting the distance of the edge that is to be maintained. You will design decentralized controllers in the `controller` script. Each agent will have access to:

- Current submission.
- Own state and identity.
- Δ-disk graph neighbor states and identities.
- Minimum safety distance δ.

- Maximum interaction distance Δ.
- Sensor data.
- Current simulation time (timer).
- Persistent memory (double) entry for each robot.
- Formation specification: desired distance to current neighbors (if any).
- Mission specific info.

You may test your designs for each sub-mission by setting the `missionChoice` variable at the top of the `main` script. At the end of each sub-mission, you will be prompted if you would like to save your final robot pose to be used as initial conditions for the following sub-mission.

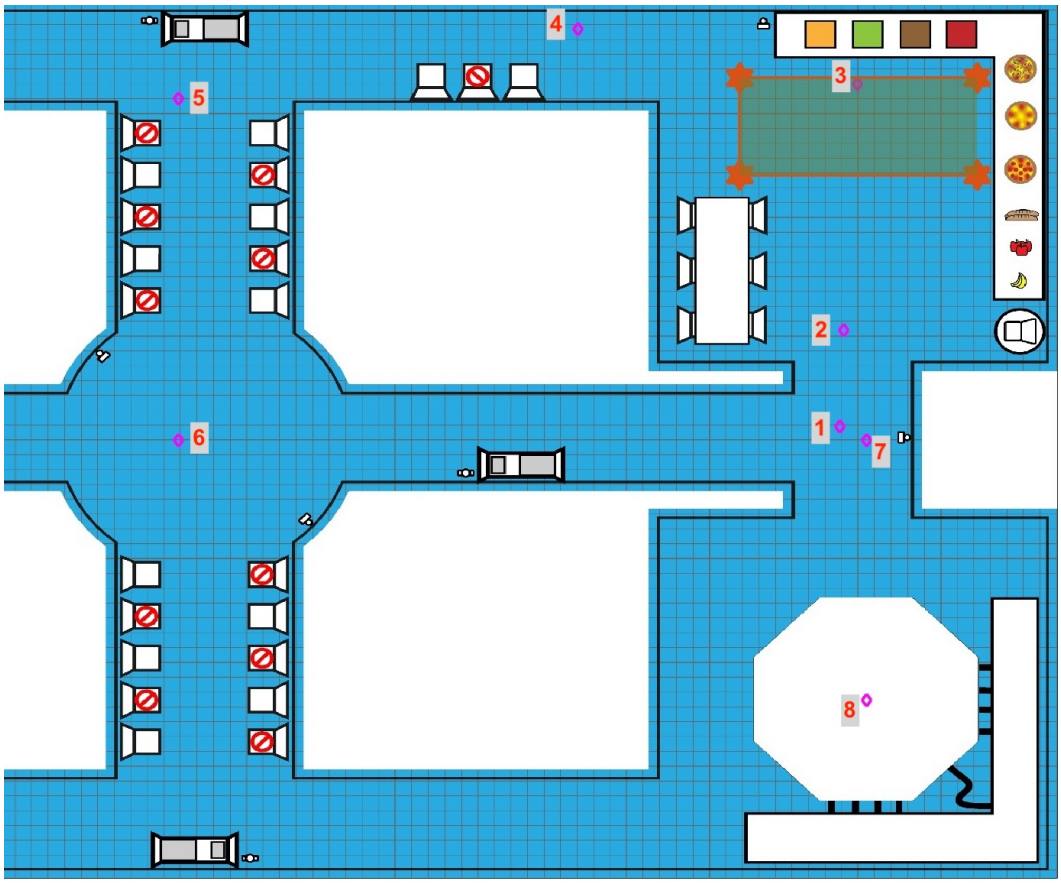

Figure 1: Hospital Map. A team of six robots start out in refueling station on bottom-right quadrant. They must navigate the to the room in the top-right quadrant before commencing a disinfection routine and returning to refuel while inspecting other areas to plan future trips. The magenta navigation waypoints which must be visited in order are only accessible to the lead robot.

The sub-missions are:

1. *Initial Deployment*

   The agents are initially deployed from the refueling station. The agent with the largest ID (leader) is the only one that is aware of sequence of navigation waypoints that must be visited, leading into the target disinfection area. Assuming that the agents are initially connected, their first objective is to navigate through these waypoints while ensuring the network remains connected and avoiding collisions between themselves and obstacles in the environment.

   For this sub-mission, you may design a formation graph topology under the 'init' case and for each waypoint in the `getFormationGraph` script. This script takes in the current scenario and waypoint and should return a weighted adjacency matrix, with nonzero entries denoting the distance of the edge that is to be maintained. You will design decentralized controllers in the `controller` script. Data specific to this sub-mission includes:

   - Current submission: 'init'
   - Mission specific info:
     - Leader agents: current waypoint target `xt`.
     - Follower agents: none (empty array).

   The sub-mission is cleared when a leader agent visits navigation waypoints 1 through 3 *and* all agents are inside the disinfection region.

2. *Room Disinfection*

   The agents must disinfect the majority of the domain enclosed by the beacons. As agents approach points of a uniformly spaced grid for the first time, a disinfection count increases.

   For this sub-mission, you may design a formation graph topology under the 'room' case in the `getFormationGraph` script. This script takes in the current scenario and should return a weighted adjacency matrix, with nonzero entries denoting the distance of the edge that is to be maintained. You will design decentralized controllers in the `controller` script. Each agent will have access to:

   - Current submission: 'room'
   - Mission specific info for each agent:
     - `domain`: 2-by-4 array of rectangular domain corners.
     - `AgentVCell`: 2-by-M array of vertices pertaining the agents Voronoi cell.
     - `NeighborVCells`: cell array of matrices containing the vertices for the Voronoi cells of Delaunay neighbors within the Δ-disk range.
     - `DelaunayNeighbors`: binary array with ones if neighboring agents are also Delaunay neighbors.

   The sub-mission is cleared when the agents disinfect the majority ($> 85\%$) of the room.

3. *Return to Refueling Station*

   The agents must way their way back to the refueling station, while also surveying the disinfection status of other areas in the hospital, to plan for future disinfection activities.

   For this sub-mission, you may design a formation graph topology under the 'return' case in the `getFormationGraph` script. This script takes in the current scenario and should return a weighted adjacency matrix, with nonzero entries denoting the distance of the edge that is to be maintained. You will design decentralized controllers in the `controller` script. Data specific to this sub-mission includes:

   - Current submission: 'return'
   - Mission specific info:
     - Leader agents: current waypoint target `xt`.
     - Follower agents: none (empty array).

   The sub-mission is cleared when a leader agent visits navigation waypoints 4 through 8 *and* all agents are within the refuel platform.

4. *Refuel*

   The agents must get into the translationally-invariant formation encoded in a given specification graph. Feasible target positions are provided to all agents in no specific order. Agents also have access to the centroid of the refuel station. Agents are allowed to go to any target location.

   For this sub-mission, you may design a formation graph topology under the 'refuel' case in the `getFormationGraph` script. This script takes in the current scenario and should return a weighted adjacency matrix, with nonzero entries denoting the distance of the edge that is to be maintained. You will design decentralized controllers in the `controller` script. Data specific to this sub-mission includes:

   - Current submission: 'refuel'
   - Mission specific info for all agents:
     - `specGraph`: Adjacency matrix whose nonzero entries denote the desired inter-agent distance for any permutation of robot IDs.
     - `feasiblePoints`: A 2-by-N matrix of a feasible configuration that satisfies the specification graph.
     - `target`: A 2-by-1 vector with the position of the refuel platform centroid.

   The sub-mission is cleared when any permutation of agent positions satisfy the inter-agent distances encoded in the specification graph weighted adjacency matrix *and* the agents' centroid is on the center of the refuel platform.

5. *Full Mission*

   You may test your designs for each sub-mission by setting `missionChoice` to the appropriate sub-mission. Once you are satisfied with your designs, you can complete the full mission by setting `missionChoice` to 'full'.

Remark: The script must return zero Roborarium warnings at the end of your implementation for the Robotarium to accept your experiment. Sources of error include actuator saturations, robot collisions, and robots crossing over the black rectangle that encodes the allocated domain on the Robotarium.

Deliverables

There are two deliverables required for full credit:

1. Project Report: Prepare a report that includes four sections, one section for each of the four sub-missions. For each of the four submissions, describe:

   (a) *A description of the formation and control design.* Include the rationale behind your choice of controller for each sub-mission, and derivations (e.g., if obtained from edge-tension functions). Also provide a description of your topology designs, and any other design consideration taken. Comment on changes and re-designs as you evaluate performance in simulation and ultimately the robotic implementation.

   (b) *Observations on the performance of your controllers.* These are your concluding remarks. For each sub-mission, did everything work as expected in simulation *and* on the Robotarium? If something was unexpected or went wrong, why do you think this was the case? How did you adjust your approach to overcome challenges? Did the simulations match up with the actual implementation?

   The report is to be submitted via Gradescope.com.

2. Video of Robotarium Implementation:

   You may find a zip file named `ENME808T_Final_Project_Robotarium_Fa24.zip` with the project files under the M-Files Module in Canvas. You will be making modifications to the `controller.m` script to incorporate your control designs and `getFormationGraph` script to encode formations.

   (a) *Simulation:* Your algorithms will need to be tested in simulation prior to running on the Robotarium by setting `simulate_true = true` in `main.m`. You can run each submission independently by setting the variable `missionChoice` in `main.m` to `'init'`, `'room'`, `'return'`, or `'refuel'` and running `main.m`. *These two lines should be the only things you modify in the* `main.m` *script.* The script will display a message indicating whether or not your designs successfully complete the submission. Once your designs are implemented and successfully accomplish each submission, you will run the full mission by setting `missionChoice` to 'full'. Upon success, a script will check for satisfaction of actuator limits and collision avoidance, inform you whether your designs are likely to be successful, and provide an estimated time of completion.

   (b) *Robotarium Experiment:* Upon successful execution of the full simulation, you are ready to submit your design as an experiment. Set `simulate_true = false` in `main.m`. Sign-in to the Robotarium website, go to the Dashboard, and select "+Submit New Experiment". Provide the requested information: Title, description, estimated duration. Enter 6 for the number of robots. There are four files that must be submitted for each experiment: `main.m`,

`controller.m`, `getFormationGraph.m`, and `missionMap.png`. You can optionally upload your initial condition 'SaveData.mat' file for a specific sub-mission experiment. Once they are uploaded, make sure to mark `main.m` as the main file. Your experiment will be marked as pending and go into a queue. You will be notified via email when the experiment has been run. When you revisit the experiment in the Dashboard, if it ran without errors, you will see a video recording of the experiment, the saved .mat data file, and a .txt log file. Check the video for a successful implementation. If the experiment did not run as expected, inspect your data and the log files to figure out what went wrong. You will need to modify your design and repeat the process until you are successful. Once you have a successful implementation, submit a video recording obtained during the Robotarium experiment for complete credit.

MATLAB files and videos of implementations are to be submitted via Canvas: elms.umd.edu.

Good Luck!