Multi-Robot Systems: Control, Communication, and Security
Professor: Stephanie Gil
CS 286 - Fall 2020
Assignment Leads (TFs): Victor Qin, Fraser Darling
Due: Friday, October 9, 2020 at 11:59pm EDT

Programming Assignment 2

Instructions

PS2 Specific Instructions : Please include a writeup with your answers for each problem and each sub-part labeled accordingly. For sub-parts requiring algorithmic implementation, please include your code and plots demonstrating the visualization of your outputs as indicated in the instructions for each sub-part below. Your writeup, code, and plots/visualization files should be placed into a zip folder labeled "YOURLASTNAME-PS2" and uploaded on to canvas by the indicated submission deadline.

Collaboration: Collaboration is allowed and encouraged, but you must write everything up by yourselves. You must list all your collaborators. MAX group size is 2 persons.

References: Please consult the assigned class readings listed below while completing this assignment. *Note that it is not permitted to use any existing code base unless provided by us or explicitly stated otherwise.* Any additional reference material should be cited; we strongly encourage you to explicitly cite readings where applicable. Standard libraries (matplotlib, numpy, scipy) are also acceptable.

Submission: Submit your solutions on Canvas in a zipped folder by the indicated due date.

Papers: Refer to the following papers for reference:

1. "Unifying geometric, probabilistic, and potential field approaches to multi-robot deployment," by Schwager, Rus, and Slotine.
2. "Coverage Control for Mobile Sensing Networks," by Cortes, Martinez, Karatas, and Bullo.

Problems

1. (Coverage, Partitioning, and Gradient Descent) Your goal is to implement the algorithm give in Schwager et al., Section 2.2 and test it specifically on coordinating multiagent coverage using different values of alpha. Code is provided in"coverage.py," along with some basic functions that should get you started with visualization and testing. The environment is set up as a 10 x 10 grid with resolution of 0.1 by default, but robots move in the environment in a continuous fashion.

(a) 15 pts: Complete the "mix_func" function in Environment to calculate $\dot{p}_i$ given in Eqn. (4) in Schwager et al. Several constants have been given to you. Assume that the value of the environment $\phi(q)$ is constant everywhere. Your output should demonstrate how the system evolves over 200 time steps. Hint: test it with 10 time steps to check for convergence before running it for 200

Output: Attach an image of your program output.

(b) 10 pts: Now, we're setting a target point to change how points in the environment are weighted. This involves changing the value function to a Gaussian-like distribution centered on the target (see Fig. 1 in Cortes et al.). A target parameter has been provided in the environment, and the cost function is already provided for you (commented out - read the code carefully!) in "update_gradient". Be sure to plot the target positions as well.

Output: Attach an image of your program output.

(c) 10 pts: Implement a moving target going in a *quarter* circle around the point (5,5) with radius 3 in 200 time steps (so with period 800 steps).

Output: Attach an image of your program output.

(d) 10 pts: Comment on how the robots react to the moving target - do they settle in a convergent position like in part (a)?

Output: 2-4 sentences

(e) BONUS - 10 pts: Returning to our original definition of value (aka no target and all points w/ equal value 1), add stochasticity to the system - what happens if robots are reporting wrong positions? Do this by altering the state that the robot reports to other agents through "_stoch_state". Comment on how convergence differs from part (a).

Output: Attach an image of your program output, and 2-4 sentences

Be sure to comment your code well, including for any additional helper methods, attributes, or other modifications you make to achieve your intended behavior for any of the above parts.

2. (Consensus) This problem deals with consensus, starting from Section 2 of Olfati-Saber. Your goal is to implement and study simulated networks of nodes communicating and coming to consensus on a single value across the network.

(a) 15 pts: Implement the "update_graph" function in Graph in "consensus.py". We've given you a set of 5 nodes setup in a line, partially connected in a circle, and fully connected, as well as functions to print node states at a time (in function "nodes_states") and plot each node's evolution over time (in function "plot_states"). Attach a plot of the output from "plot_states" for both formation setups, and analyze which one converged faster. You might find that implementing and using the "is_finished" function might help.

Output: Attach an image of your program output for each case

(b) 10 pts: Now add stochasticity to the graph, with $\sigma = 0.1$. What does this do to the convergence? Again, include plots for each of the three formations.

Output: Attach an image of your program output for each case, as well as 2-4 sentences.

(c) 10 pts: How does the effect of stochasticity differ between the linear, partial, and fully connected graphs? Can you think of how this might relate to the graph structure?

Output: 5 sentences - put some thought in it!

(d) 10 pts: Starting with the fully-connected graph, build your own test case that weights directly neighboring nodes (eg. for node 2 its neighbors are 1 and 3) twice as strong as other connections. How does this change convergence? Include plots, and a copy of your adjancency matrix in your writeup.

Output: Attach an image of your program output for each case, a copy of your adjancency matrix (image or typed), as well as 2-4 sentences.

(e) 10 pts: Write a function that calculates the Fiedler Value (second smallest eigenvalue of the graph Laplacian) for a given adjacency matrix of an undirected graph.

Output: Calculate (and write down) the Fiedler value of 3 matrices of your choosing, as well as the three test cases given (6 total values). Include both value and matrix in your writeup

(f) BONUS - 10 pts: Remove stochasticity. We've given you an adversary agent to implement. This adversary agent will have its own target, and take steps of size "epsilon" towards that target at every time step. Repeat the same process that you did in part (a) by analyzing performance and convergence - how has convergence changed?

Output: Attach an image of your program output for each case, as well as 2-4 sentences.

(g) BONUS - 5 pts: Finally, each node can't guarantee connectivity with other nodes. Implement the function "rand_adg", which should build a connection between node i and node j with probability $p \in (0, 1)$, then use this function to update the graph's adjacency matrix every time step. You can remove the adversary for this step - analyze how convergence has changed from part (a).

Output: Attach an image of your program output for each case, as well as 2-4 sentences.

Be sure to comment your code well, including for any additional helper methods, attributes, or other modifications you make to achieve your intended behavior for any of the above parts.