Multi-Robot Systems: Control, Communication, and Security
Professor: Stephanie Gil
CS 286 - Spring 2022
Assignment Leads (TFs): Henry Cerbone, Ninad Jadhav
Due: Friday, March 11, 2022 at 11:59pm EDT

Programming Assignment 3

Instructions

Collaboration: Collaboration is allowed and encouraged, but each group must submit unique code and write-up. One submission per group. You must list all your collaborators. MAX group size is 3 people.

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. "Guaranteeing Spoof-Resilient Multi-Robot Networks" by Stephanie Gil, Swarun Kumar, Mark Mazumder, Dina Katabi and Daniela Rus. [1]
2. "Unifying geometric, probabilistic, and potential field approaches to multi-robot deployment," by Schwager, Rus, and Slotine. [2]
3. "Resilient Multi-Agent Consensus Using Wi-Fi Signals" by Stephanie Gil, Cenk Baykal, and Daniela Rus. [3]

Problems

1. (Coverage with Spoofed Agents) Your goal is to extend the coverage algorithm from Schwager et al. [2] to implement the coverage-with-spoofers algorithm given in Gil and Kumar et al. [1], Section 3 Problem 2 (page 4). You will test it specifically on coordinating multi-agent coverage of servers over clients. Base code provided in `coverage-spoof.py` gives you a Schwager et al. coverage implementation, which you completed in Programming Assignment 2, and you will be making modifications to this code. It also includes a Client class, and each Client will initialized as either a spoofer or legitimate. Since you don't have realistic Wifi signals to work with, you will simulate the process of creating confidence metrics via Wifi signals by querying an oracle function (`sample_alphas` in Part c). As before, the environment is set up as a 10×10 grid with resolution of 0.1 by default, but robots move in the environment in a continuous fashion.

(a) 5 pts: Complete the `define_rho` function in Environment to define an importance function $\rho(q)$ over the environment. As in Gil and Kumar et al. [1], the importance function should be decomposable into terms $\rho_i(q)$, $i \in [c]$, where c is the number of clients, depending on each client's position, i.e., $\rho(q) = \rho_i(q) + \cdots + \rho_c(q)$. The Schwager et al. coverage implementation given to you as a part of the `coverage-spoof.py` base code assumes an importance function value of 1 over all points in the environment, so you should appropriately replace this throughout the code with the $\rho(q)$ you create via `define_rho`. This updated importance function will be used for the remainder of Question 1. Note that `define_rho` is called once after the Environment is initialized.

Output: Code written in `coverage-spoof.py`. No simulation visualizations or discussion necessary for this question.

(b) 15 pts: Run the server-client simulation now on a test case without an α-modified importance function. For this question's visualization and all further Question 1 visualizations, spoofer clients should be visualized at opacity of 0.1, and legitimate clients should be visualized at opacity 0.9. Your output should demonstrate how the system evolves over 200 time steps.

Output: Include an image of your program output in the writeup PDF, along with 2-4 sentences describing your result and why it evolved as such.

(c) 15 pts: Complete the `sample_alphas` function in Environment to update the confidence metrics of the clients, as defined in Gil and Kumar et al. [1]. This function is called once for every iteration of gradient descent you perform. As a substitute for receiving real wifi signals, this function will allow you to stochastically observe α_i for client i by sampling from a normal Gaussian distribution. Note that it is only locally within this function that the identity of the spoofer or legitimate clients affects the system (see comment in `sample_alphas`). In this respect, `sample_alphas` is an oracle function since it has information not available to the rest of the system and is able to discern spoofers from legitimate clients to sample α_i from the appropriate distribution, a value typically constructed via Wifi signals as in Gil and Kumar et al. [1]. By default, legitimate clients sample from a distribution with $E(\alpha_i) = 0.8$ and spoofer clients sample from a distribution with $E(\alpha_i) = 0.2$. Why might these be good values?

Output: Code written in `coverage-spoof.py`, and 2-4 sentences in the writeup to answer the question. No simulation visualizations necessary for this question.

(d) 15 pts: Alter the `update_gradient` function in Environment to use an α-modified importance function. Rerun this version of the simulation on the same test case you used in Part B, and compare and contrast the results with that of Part B. Your output should demonstrate how the system evolves over 200 time steps.

Output: Code written in `coverage-spoof.py`. Include your visualization and corresponding discussion (2-4 sentences) in your writeup.

(e) 15 pts:

i. 10 pts: Run α-modified simulations on two other combinations of the variables `spoofer-mean` and `legit-mean`, the means of the normal Gaussian distributions which which spoofer clients and legitimate clients sample their α_i values, respectively. Include corresponding visualizations of the systems' evolution.

ii. 5 pts: Also consider if we set `spoofer-mean` = `legit-mean` = 0.5.

What does each of these 3 combinations imply about the system's ability to distinguish between spoofers and legitimate clients? Your outputs should demonstrate how the system evolves over 200 time steps.

Output: Test cases included in `coverage-spoof.py` Include images of your program output for each of the 3 cases and corresponding discussion (2-4 sentences each) in your writeup.

Your report for this question should contain:

- Code for part (a)
- 1 plot of the coverage result and written explanation for part (b)
- Code and written explanation for part (c)
- Code as well as 1 plot of coverage result and written explanation for part (d)
- Code for test cases as well as 3 plots of the coverage results and written explanations for part (e)

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. (Using Spoofed Information) This problem deals with the `spoofing.py` file. Your mission (should you choose to accept it) is to build a system of agents to work with other loyal legitimate agents to converge close to their common true mean while ignoring malicious spoofing agents that are sending false signals. The legitimate agents are able to approximately identify between legitimate and spoofed signals (with $\alpha = -0.5$ being definitely spoofed, and $\alpha = 0.5$ being definitely legitimate), but don't know what to do about it. (This follows section 2 and the first part of section 3 in Resilient Multi-Agent Consensus Using Wi-Fi Signals by Gil et al. [3]). *Note that in this paper the α values are shifted by -0.5 from paper 1 (Gil et al. 2017 [1]).*

   We've defined a function to generate values of α for you at every time step. Legitimate and spoofed agents will get values with $\mathbb{E}[\alpha = 0.4]$ and $\mathbb{E}[\alpha = -0.4]$ and standard deviation 0.1 respectively.

   (a) 15 pts: Implement the updates to the W matrix according to equation (3), where the time-varying $\beta(t)$ function means we must update W in the `transition_W` function. Note that we have implemented the dynamical system presented in eqn. (2), and provided you with a function that returns values for beta presented in the algorithm (in the class `AlphaWeights`). Test this on the example case provided, and attach an image of your program output.

   Output: Attach image of your program output.

   (b) 10 pts: Currently we have set up 20 legitimate agents and 4 spoofers, with pretty similar initial state values for all legitimate agents. What happens if the initial agent states are more scattered? Modify the initial values of the main function by using numpy.random.randn. Generate 20 random values of x_L for each agent with mean 1.5 for each run, but with standard deviations of 0.1, 1, 3, and 5, while keeping the 4 spoofed values at 4. Plot the final converged state value for each case (make a graph w/ 4 points, w/ x-axis as stan devs and final converged value in y-axis) Compare the convergence rate and final converged value between these cases - what do you notice, and can you explain why?

Output: Attach images of your 4 program outputs as well as a plot of the 4 setups. Write up a few sentences.

(c) 10 pts: Set your legitimate agents to have initial state values of mean 1.5 and standard deviation 1, but now set the 4 spoofers to have initial state values of $x_S = 0$, 2, 4, and 6 respectively. Estimate the final converged value for the legitimate agents, and plot them for each case (make a graph w/ 4 points, x-axis for spoofer value, y-axis final converged value). Does this have any effect on convergence or final converged value?

Output: Attach images of your 4 program outputs as well as a plot of the 4 setups. Write up a few sentences.

(d) 5 pts: Keep your legitimate agents to have initial state values mean 1.5 and standard deviation 1. Set the spoofers back to an initial state value of $x_S = 4$; what happens when you increase the number of spoofers? Try at least 4 different values for the number of spoofers n_s, and do the same plot as you did for part (c).

Output: Attach images of your 4 program outputs as well as a plot of the 4 setups. Write up a few sentences.

(e) Use a system of 20 legitimate (initial state values mean 1.5, standard deviation 1) and 4 spoofers (initial state value of $x_S = 4$). What happens if the spoofers change their state values over time (eg. x_S changes over time)? Change your spoofers to follow the following two patterns in state values:

i. 5 pts: A sine-wave pattern with state values centered on 4, with amplitude 1 and period 50 timesteps (Hint: look at numpy.arange() and numpy.sin(), and the code written for coverage last week)?

ii. 5 pts: Values increasing exponentially according to $x[t+1] = 1.01 * x[t]$?

Output: Plot the 2 setup outputs, and do some explaining

Your mission report should contain:

- 1 plot of the consensus plot from part (a)
- 4 plots of the consensus plots and 1 plot of final converged value for the 4 cases from part (b)
- 4 plots of the consensus plots and 1 plot of final converged value for the 4 cases from part (c)
- 4 plots of the consensus plots and 1 plot of final converged value for the 4 cases from part (d)
- 2 plots of the consensus plots from part (e)

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.