## E-150 ASSIGNMENT 2 (100 POINTS)

## MODELING AND OPTIMIZATION OF UAV SWARMS

Due: 09/30/21

In this project, you will use machine learning / a genetic algorithm to find optimal control parameters for a swarm of autonomous vehicles tasked with mapping a set of targets quickly while avoiding collisions with obstacles and other vehicles. This problem could represent a real-world scenario like inspecting a disaster zone or construction site. All necessary constants are defined in a glossary at the end of this document.

## DYNAMICS AND INTEGRATION

This project will model autonomous vehicles hereafter simply called "agents" with the following simplifying assumptions:

- The effects of buoyancy, lift, and gravity are of secondary importance and may be neglected.
- The agents may propel themselves in any direction in 3D space.
- The agents may be idealized as point masses.
- The agents know the locations of all targets, obstacles, and other agents.

This problem should be modeled in a fixed Cartesian basis. The generic equations for the position, velocity and acceleration in this basis are given in vector form by:

$$\boldsymbol{r} = r_1\boldsymbol{e}_1 + r_2\boldsymbol{e}_2 + r_3\boldsymbol{e}_3 \tag{1}$$

$$\boldsymbol{v} = \dot{\boldsymbol{r}} = \dot{r}_1\boldsymbol{e}_1 + \dot{r}_2\boldsymbol{e}_2 + \dot{r}_3\boldsymbol{e}_3 \tag{2}$$

$$\boldsymbol{a} = \ddot{\boldsymbol{r}} = \ddot{r}_1\boldsymbol{e}_1 + \ddot{r}_2\boldsymbol{e}_2 + \ddot{r}_3\boldsymbol{e}_3 \tag{3}$$

Each agent with a given number $i$ will follow Newton's second law of motion:

$$m_i\boldsymbol{a}_i = \boldsymbol{\Psi}_i^{tot} \tag{4}$$

Where $\boldsymbol{\Psi}_i^{tot}$ is the total force vector acting on the agent, defined by:

$$\underbrace{\boldsymbol{\Psi}_i^{tot}}_{\text{net force}} = \underbrace{\boldsymbol{F}_{p,i}}_{\text{prop. force}} + \underbrace{\boldsymbol{F}_{d,i}}_{\text{drag force}} \tag{5}$$

The components of the force for this general system are defined as:

$$\boldsymbol{F}_{p,i} = F_{p,i}\boldsymbol{n}_i^* \tag{6}$$

$$\boldsymbol{F}_{d,i} = \tfrac{1}{2}\rho_a C_{d,i} A_i \|\boldsymbol{v}_a - \boldsymbol{v}_i\|(\boldsymbol{v}_a - \boldsymbol{v}_i) \tag{7}$$

The unit vector $\boldsymbol{n}_i^*$ will be determined by the control parameters described next.
Use the Forward Euler method to integrate equation 4 in time to a final time of $t_f$:

$$\boldsymbol{v}_i(t+\Delta t) \doteq \boldsymbol{v}_i(t) + \boldsymbol{a}_i(t)\Delta t = \boldsymbol{v}_i(t) + \boldsymbol{\Psi}_i^{tot}(t)\tfrac{\Delta t}{m_i} \tag{8}$$

$$\boldsymbol{r}_i(t+\Delta t) \doteq \boldsymbol{r}_i(t) + \boldsymbol{v}_i(t)\Delta t \tag{9}$$

## OBJECTS, INTERACTIONS AND CONTROL PARAMETERS

Agent behavior is controlled by $\boldsymbol{n}_i^*$, the direction of the propulsive force given in equation 6. For this assignment, there is **no option to control the magnitude of the thrust**. A given agent interacts with all targets, all other agents, and all obstacles at every time step to determine $\boldsymbol{n}_i^*$.

Targets and obstacles are stationary and positioned randomly throughout the region given by:

$$(|x| \leq 100\ m),\ \ (|y| \leq 100\ m),\ \ (0 \leq z \leq 10\ m) \tag{10}$$

Important note: you should create a single (random) arrangement of obstacles and targets at the beginning of your code and reset to that arrangement for every trial. It is important that all trials run on the same set of target and obstacle locations to make cost function comparisons fair. The agents start *at rest* in any arrangement you want[1] within the region defined by:

Purposefully arrange the agents within this region. As with the obstacles and targets, reset to one initial arrangement for each trial. You should be able to reduce their number for debugging.

**Objects interact in the following ways:**

- If an agent passes within `agent_sight` of a target, that target is "mapped" and removed from the array of target positions.[2]
- If two agents pass within `crash_range` of each other, both agents "crash" and are removed from the array of agent positions.[3]
- If an agent is within `crash_range` of an obstacle, the agent "crashes." The obstacle remains.
- If an agent leaves the region $(|x| \leq 150\ m),\ \ (|y| \leq 150\ m),\ \ (|z| \leq 60\ m)$, it "crashes."
- In the unlikely event that an agent is close enough to interact with multiple objects in the same time step, the order of priority is: agent-target, agent-agent, agent-obstacle.

A trial should stop automatically once all targets are mapped or all agents are crashed.

Clearly, the agents should propel themselves away from obstacles and other agents (at least at close range) and toward targets. However, deriving an ideal set of rules to determine their behavior is not practical. Accordingly, we will use a general control law with parameters that can be optimized via a genetic algorithm. The distance between objects should be a key function input, since faraway objects should be treated differently than nearby ones. We will use the Euclidean distance or 2-norm between objects, defined between agent $i$ with position $\boldsymbol{r}_i$ and target $j$ at $\boldsymbol{T}_j$ as:

$$d_{ij}^{mt} \overset{\text{def}}{=} \|\boldsymbol{r}_i - \boldsymbol{T}_j\| = \sqrt{(r_{i1} - T_{j1})^2 + (r_{i2} - T_{j2})^2 + (r_{i3} - T_{j3})^2} \tag{11}$$

The unit vector between $\boldsymbol{r}_i$ and $\boldsymbol{T}_j$ is given by:

$$\boldsymbol{n}_{i\to j}^{mt} = \frac{\boldsymbol{T}_j - \boldsymbol{r}_i}{\|\boldsymbol{T}_j - \boldsymbol{r}_i\|} \tag{12}$$

The interaction vector $\hat{\boldsymbol{n}}_{i\to j}^{mt}$ between an agent and a target is defined by the direction of $\boldsymbol{n}_{i\to j}^{mt}$ and a weight given by an exponentially decreasing function[4] of $d_{ij}$:

$$\hat{\boldsymbol{n}}_{i\to j}^{mt} = (\underbrace{w_{t1}e^{-a_1 d_{ij}^{mt}}}_{attraction} - \underbrace{w_{t2}e^{-a_2 d_{ij}^{mt}}}_{repulsion})\boldsymbol{n}_{i\to j}^{mt} \tag{13}$$

[1] Make sure all agents start far enough apart not to crash into each other immediately!

[2] Be thoughtful about removing or flagging objects. If you naively change the size of an array during a `for` loop over that array's indices, for instance, you will have some problems. Additionally, resizing arrays is slow. You should not waste time iterating over irrelevant objects and they must not influence the remaining agents. The exact way you resolve this issue is up to your creativity. Some strategies will be presented in the help sessions.

[3] You may neglect the edge case where three or more agents collide simultaneously.

[4] Exponentially decreasing functions have several good characteristics for this problem: they are very smooth, decrease rapidly, and do not explode for any value of $d_{ij}$. In principle, other function types could be used.

The total interaction vector between agent $i$ and all targets is the sum of all interaction vectors:

$$\boldsymbol{N}_i^{mt} = \sum_{j=1}^{N_t} \hat{\boldsymbol{n}}_{i\to j}^{mt} \tag{14}$$

The interaction with obstacles and other agents follow the same form with different constants. The interaction between agent $i$ and obstacle $j$ with position $\boldsymbol{O}_j$ is given by:

$$d_{ij}^{mo} \stackrel{\text{def}}{=} \|\boldsymbol{r}_i - \boldsymbol{O}_j\| = \sqrt{(r_{i1} - O_{j1})^2 + (r_{i2} - O_{j2})^2 + (r_{i3} - O_{j3})^2} \tag{15}$$

$$\boldsymbol{n}_{i\to j}^{mo} = \frac{\boldsymbol{O}_j - \boldsymbol{r}_i}{\|\boldsymbol{O}_j - \boldsymbol{r}_i\|} \tag{16}$$

$$\hat{\boldsymbol{n}}_{i\to j}^{mo} = (w_{o1} e^{-b_1 d_{ij}^{mo}} - w_{o2} e^{-b_2 d_{ij}^{mo}}) \boldsymbol{n}_{i\to j}^{mo} \tag{17}$$

$$\boldsymbol{N}_i^{mo} = \sum_{j=1}^{N_o} \hat{\boldsymbol{n}}_{i\to j}^{mo} \tag{18}$$

The interaction between agent $i$ and agent $j$ is given by:

$$d_{ij}^{mm} \stackrel{\text{def}}{=} \|\boldsymbol{r}_i - \boldsymbol{r}_j\| = \sqrt{(r_{i1} - r_{j1})^2 + (r_{i2} - r_{j2})^2 + (r_{i3} - r_{j3})^2} \tag{19}$$

$$\boldsymbol{n}_{i\to j}^{mm} = \frac{\boldsymbol{r}_j - \boldsymbol{r}_i}{\|\boldsymbol{r}_j - \boldsymbol{r}_i\|} \tag{20}$$

$$\hat{\boldsymbol{n}}_{i\to j}^{mm} = (w_{m1} e^{-c_1 d_{ij}^{mm}} - w_{m2} e^{-c_2 d_{ij}^{mm}}) \boldsymbol{n}_{i\to j}^{mm} \tag{21}$$

$$\boldsymbol{N}_i^{mm} = \sum_{j=1,\, j\neq i}^{N_m} \hat{\boldsymbol{n}}_{i\to j}^{mm} \tag{22}$$

The total interaction between an agent and the environment is given by a weighted sum of the interactions with each type of object:

$$\boldsymbol{N}_i^{tot} = W_{mt} \boldsymbol{N}_i^{mt} + W_{mo} \boldsymbol{N}_i^{mo} + W_{mm} \boldsymbol{N}_i^{mm} \tag{23}$$

Finally, the propulsive force direction $\boldsymbol{n}_i^*$ is given by:

$$\boldsymbol{n}_i^* = \frac{\boldsymbol{N}_i^{tot}}{\|\boldsymbol{N}_i^{tot}\|} \tag{24}$$

All interaction parameters $(W_{mt}, W_{mo}, W_{mm}, w_{t1}, w_{t2}, w_{o1}, w_{o2}, w_{m1}, w_{m2}, a_1, a_2, b_1, b_2, c_1, c_2)$ will be determined by your genetic algorithm.

## GENETIC ALGORITHM

A good set of control parameters will enable agents to map targets quickly and completely without crashes. The cost function for comparing the outputs from different parameters is:

$$\Pi = w_1 M^* + w_2 T^* + w_3 L^* \tag{25}$$

$$M^* = \frac{\text{(Unmapped targets)}}{\text{(Total targets)}}, \;\; T^* = \frac{\text{(Used time)}}{\text{(Total time)}}, \;\; L^* = \frac{\text{(Crashed agents)}}{\text{(Total agents)}} \tag{26}$$

$$w_1 = 70, \;\; w_2 = 10, \;\; w_3 = 20$$

Note that all terms in the cost function are non-dimensional. The weights reflect the relative importance of each term. The "design string" for this problem contains all 15 undetermined constants:

$$\mathbf{\Lambda}^i \stackrel{\text{def}}{=} \{\Lambda_1^i, ..., \Lambda_N^i\} \stackrel{\text{def}}{=} \{W_{mt}, W_{mo}, W_{mm}, w_{t1}, w_{t2}, w_{o1}, w_{o2}, w_{m1}, w_{m2}, a_1, a_2, b_1, b_2, c_1, c_2\}^i \tag{27}$$

You should initially assume that the values of all design parameters lie in the interval:

$$0 \leq \Lambda_n \leq 2 \; \forall \; n \tag{28}$$

Scale and offset all randomly-generated design strings to span this interval.

## IMPLEMENTATION TIPS

The most challenging aspect of this assignment is reducing and working with the computational cost of running the simulation of a large number of objects so many times. Poorly-written code may take **HOURS** to run on a laptop. The solution Matlab code runs all the way through evolution with the requested parameters in less than 20 minutes, but it is possible to get faster results. To this end, it is important to implement a few techniques to reduce wasted time:

- Terminate a trial if all agents are crashed or all targets are collected.
- Do not evaluate the parent strings again. Evaluate the children and new random strings, then simply compare the previously evaluated strings' costs at the end of the generation.
- Start with a small number of objects on a smaller domain to debug your algorithm all the way through. The run time per trial is approximately proportional to $N_m(N_m + N_t + N_o)$.
- Have an easy method for plotting the location of objects within the domain when you want. It's very easy to tell if object interactions are wrong by watching an animation.
- Vectorize your code. The brute force way to create your code is to loop through every agent, every target, and every obstacle, which is effective but long. Vectorizing this process will significantly reduce run times.
- Create and print statements representing the projected run time of your entire code. Otherwise you may be waiting a LONG time to detect problems with your script.

## VARIABLE GLOSSARY

Table 1: Dynamics and Integration Parameters

| Symbol | Type | Units | Value | Description |
|---|---|---|---|---|
| $A_i$ | scalar | $m^2$ | 1 | agent characteristic area |
| $C_{d,i}$ | scalar | none | .25 | agent coefficient of drag |
| $m_i$ | scalar | kg | 10 | agent mass |
| $\boldsymbol{F}_{p,i}$ | $3 \times 1$ vector | N | Eqn. 6 | Prop. force vector |
| $\boldsymbol{n}_i^*$ | $3 \times 1$ unit vector | none | Eqn. 24 | Prop. force direction |
| $F_{p,i}$ | scalar | N | 200 | Prop. force mag. |
| $\boldsymbol{F}_{d,i}$ | $3 \times 1$ vector | m | Eqn. 6 | Drag vector |
| $\boldsymbol{r}_i$ | $3 \times 1$ vector | m | Eqn. 1 | agent $i$ position |
| $\boldsymbol{v}_i$ | $3 \times 1$ vector | m/s | Eqn. 2 | agent $i$ velocity |
| $\boldsymbol{a}_i$ | $3 \times 1$ vector | $m/s^2$ | Eqn. 3 | agent $i$ acceleration |
| $\boldsymbol{v}_a$ | $3 \times 1$ vector | m/s | $\mathbf{0}$ | Air velocity |
| $\rho_a$ | scalar | $kg/m^3$ | 1.225 | Air density |
| $\Delta t$ | scalar | s | .2 | Time step size |
| $t_f$ | scalar | s | 60 | Maximum task time |

Table 2: Objects and Interactions Parameters

| Symbol | Type | Units | Value | Description |
|---|---|---|---|---|
| `agent_sight` | scalar | m | 5 | Target mapping distance |
| `crash_range` | scalar | m | 2 | agent collision distance |
| $N_m$ | scalar | none | 15 | Number of initial agents |
| $N_o$ | scalar | none | 25 | Number of obstacles |
| $N_t$ | scalar | none | 100 | Number of initial targets |
| $\boldsymbol{O}_j$ | $3 \times 1$ vector | m | Eqn. 10 | Obstacle $j$ position |
| $\boldsymbol{T}_j$ | $3 \times 1$ vector | m | Eqn. 10 | Target $j$ position |

Table 3: Genetic Algorithm Parameters

| Symbol | Type | Units | Value | Description |
|---|---|---|---|---|
| `children` | scalar | none | 6 | Strings generated by breeding |
| `parents` | scalar | none | 6 | Surviving strings for breeding |
| `S` | scalar | none | 20 | Designs per generation |
| `G` | scalar | none | 100 | Total generations |
| $L*$ | scalar | none | Eqn. 26 | Fraction of agents lost |
| $M^*$ | scalar | none | Eqn. 26 | Fraction of targets remaining |
| $T^*$ | scalar | none | Eqn. 26 | Fraction of time used |
| $w_1$ | scalar | none | 70 | Weight of mapping in net cost |
| $w_2$ | scalar | none | 10 | Weight of time usage in net cost |
| $w_3$ | scalar | none | 20 | Weight of agent losses in net cost |

## DELIVERABLES

You should present your work in a concise, typed technical report. The report should be crafted such that a classmate of yours who is not in this class could understand the purpose and methods of this project. **This report should not be longer than necessary** to convey the key information completely and clearly. There is no specific suggested length. You are *encouraged* to collaborate with your classmates. Appropriate forms of collaboration include: discussing conceptual challenges, discussing implementation strategies, and comparing results. Inappropriate forms of collaboration include: directly exchanging code, directly exchanging results, directly exchanging written report content. In short: discuss the problem freely with your classmates but write your own code and your own report.

Your figures should be clearly labeled, plotted with clear markers and legends, and have axes that include units. The report should have the following sections, addressing the following specific questions in addition to any other key information for understanding your work:

**Introduction:**

1. Outline the scenario being modeling by this project.
2. Explain the specific goals of the project.
3. Specify the methods that will be explained and the structure of the paper.

**Background and Theory:**

1. Present and explain the key equations used to model the dynamics of the agents.
2. Determine the terminal velocity of the agents.
3. Discuss the use of Forward Euler time discretization. What trade-offs does it have?
4. What are the advantages and disadvantages of using a larger value for $\Delta t$?
5. What must be true for the point-mass idealization to be reasonable? What is left out?
6. How would adding net gravity, lift, or buoyancy complicate this problem?

**Procedure and Methods:**

1. Comment on the specific cost function chosen for this problem. Why are gradient-based methods poorly suited to optimizing it (even numerically)? Why is a GA a good choice?
2. Comment on the choice of control model. Is there any rigorous, provable reason to use this framework over something else? Why might we expect it to work for some parameter values?
3. What would happen if any of the $a$, $b$, or $c$ design variables became negative? Would this be good or bad for the performance of the swarm?
4. Explain your strategy for "removing" mapped targets and crashed agents.
5. Explain how you will initially place the agents.
6. Explain any other significant choices you made in creating your code. If you have gone above and beyond in improving your genetic algorithm, explain what you implemented and why.

**Results and Discussion:**

1. Provide a convergence plot showing the total cost of the best design, the mean of all parent designs, and the mean of the overall population for each generation.
2. Provide a plot showing the individual performance components (i.e., plot $M^*$, $T^*$, and $L^*$), for the overall best performer, mean parental population, and overall population. Discuss any important observations.

3. Report your best-performing 4 designs in a table similar to the following.

| DESIGN | $\Lambda_1$ | $\Lambda_2$ | $\Lambda_3$ | *etc* | $\Pi$ |
|---|---|---|---|---|---|
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |

Table 4: The top 4 system parameter performers.

4. Discuss the results. How much variation does each parameter have between your top performers? Do any parameters have negative values? Are any values very near zero? Is anything else notable about the results and the behavior they should create?

5. Include a series of plots showing how your best design moves the agents. Include 5 approximately evenly-spaced frames from $t = 0$ until the final time for your system. Agents, targets and obstacles should all be clearly shown with distinct marker styles, axes should be labeled and the title should contain the time at which the snapshot occurs.

**Conclusion:**

1. Briefly summarize your goals, methods, and key results.

**Appendix:**

1. Include any raw data or code snippets that are critical for understanding your code and that don't fall into any other section. This section does not have to contain anything. Do not attach your entire code in the appendix.

**To submit your assignment, do BOTH of the following:**

1. Submit your PDF copy of your typed report on Gradescope before the end of day (11:59 pm PST).

2. Submit a zip file to Gradescope containing your code (as a .m file, .py file, etc) and any supporting files required to run your code.