CS 7638 - Robotics: AI Techniques - Hop Scotch Project

Summer 2025 - Due June 2nd, 11:59PM AOE

Table of Contents

Introduction

While travelling through the milky way galaxy searching for earth, your spaceship has an engine malfunction rendering the spaceship engine mostly inoperable, only able to navigate via jumps. Luckily, a shower of asteroids dot the vast cosmic landscape, and can serve as galactic ferries for our spaceship. It is your task to receive sensor readings of the locations of these asteroids, predict where each of the asteroids will be one timestep later, and finally, jump/"hop scotch" from asteroid to asteroid to reach homebase, earth.

This project consists of two parts:

1. Estimation: Estimate where asteroids will be one timestep into the future
2. Jumps: Select asteroids to 'ride' on with the ultimate goal of reaching homebase.

Submitting Your Assignment

Your submission will consist of ONLY the `spaceship.py` file, which you will upload to Gradescope.

Project Description

The motion model of the asteroids takes the form

$$x(t) = c_{pos_x} + c_{vel_x}t + \frac{1}{2}c_{acc}t^2$$

for the asteroid's x-position, and

$$y(t) = c_{pos_y} + c_{vel_y}t + \frac{1}{2}c_{acc}t^2$$

for its y-position.

Each timestep is $dt = 1$ seconds in duration. Each asteroid's motion can be modeled using x, y, dx, dy, ax, ay. where 'a' is acceleration. C_pos, C_vel, and C_acc are respectively, the position, velocity and the acceleration term for the asteroid.

World:

In this project, your world is a rectangle of variable width x height dimension (e.g. 2x2, 4x2 etc.). (0,0) is the lower left corner. This coordinate system is used throughout this project to define all entity locations. Homebase/"Earth" is considered the upper value of the y coordinate (i.e. upper field boundary). It has been colored green. You will want to guide your spaceship here.

Part A: Estimation

Goal:

To provide "estimates" of an asteroid's coordinates (x & y) which fall within an expected margin of the asteroids true position.
In this part of the project, you are predicting the location of asteroids one timestep into the future given the asteroids' current true positions. The (x,y) position measurements provided in test cases 1 and 2 and 16 will have no noise (i.e. the measurement information (asteroid_observations) provided to the predict_from_observations function is NOT noisy.), the remaining test cases will be noisy measurement positions. Once an asteroid leaves the field, it is removed from the tracked asteroids and is no longer passed in as input.

Scoring: Your estimates will be considered a 'match' if they fall within an expected margin of the asteroids true position. Those matches are used in calculating 3 scoring criteria. Each asteroid ID estimate returned from predict_from_observations function is scored on 3 primary criteria:

1 - The number of timesteps it took for an asteroid's estimate to 1st match (i.e. How quickly were you able to get a match).
2 - The Max number of consecutive matches of an asteroid over its time within the field boundary (i.e. How consistently were you able to match).
3 - The percentage of matches over the asteroid's time within the field boundary (i.e. How accurately were you able to match) Scores for a test case are the summed total of all scoring criteria from all asteroids.

Visualization: When you run a case with the turtle visualization option, you should see something like what is shown in the image above. The gray circles represent the actual locations of asteroids. A red dot indicates a prediction that is too far from the asteroid's actual location to count as correct, and a green dot indicates an estimate close enough to be counted as correct.

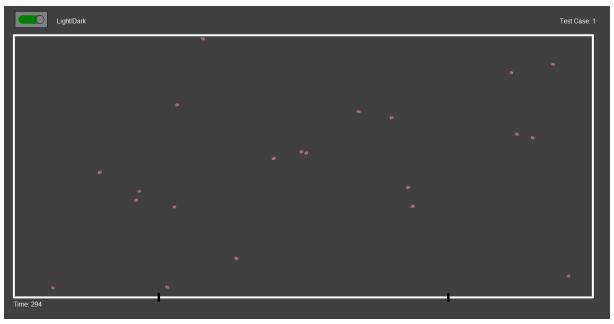


Figure 1: Estimation visualization

Notes on testing your Estimation Code Test cases 1 and 2 are meant to help you verify that your implementation works correctly in simple scenarios before applying it to more complex scenarios. Passing this part of the project does not guarantee that your implementation is correct, but cases 1 and 2 are small enough test cases that you can work through a timestep or two for one or more asteroids by hand while stepping through your code with a debugger to verify that your implementation is consistent with your hand calculations.

Part B: Jumps

Goal:

The True location of the spaceship/agent is currently represented as the blue Rocket, and the blue dot at its center represents the asteroid it is currently riding. The large blue circle surrounding the spaceship represents the jump range/distance the agent can hop. Locations outside this range (i.e. >) are too far to jump and will result in invalid jumps! Get the spaceship blue dot to homebase. For part B of the project, you will be devising a simple algorithm to select which asteroid to navigate to. Part B of the project makes use of the predictions of the asteroid locations computed by predict_from_observations in the Estimation Part A portion of the project. When part B code is run you will see an image similar to this.

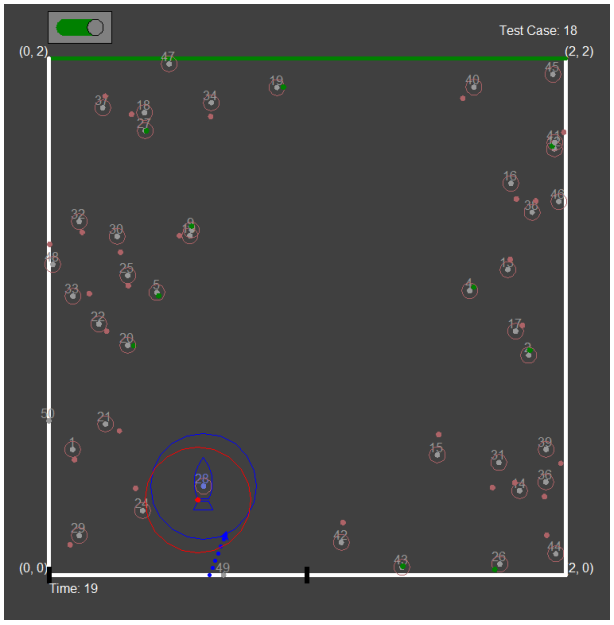


Figure 2: Estimation visualization

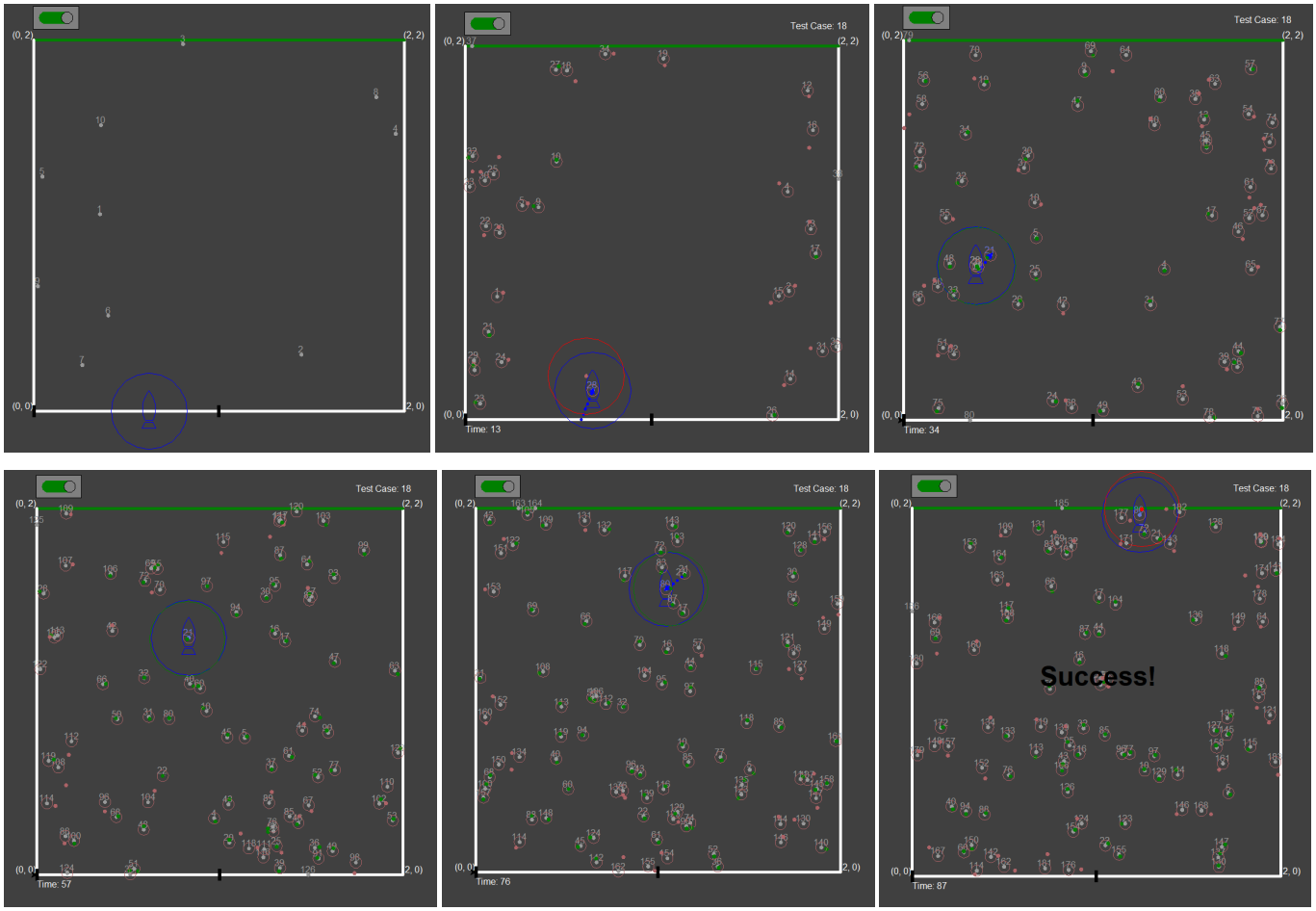
Once the spaceship jumps on an asteroid, its state parameters (position etc.) are updated to the true state parameters of the ridden asteroid (Note:True NOT noisy values). Note that while the agent is located at an asteroid's true location, you do NOT have access to "TRUE" values of that location and only have noisy measurements to rely on. You are thus advised to work on part A before moving on to part B.

Causes for failing test case

- Attempting to jump 'onto' an asteroid that is outside the spaceship's jump radius is considered invalid.
- Attempting to jump 'onto' an asteroid that is outside the field boundaries is considered invalid.
- Attempting to jump 'from' the spaceship which is outside the field boundaries is considered invalid.

Once an asteroid moves outside the field boundary, the id number is removed from the provided argument ids. Note: To successfully complete Part B of this project, you will likely need to employ creative solutions as you select how best to traverse through the cosmic field.

Visual progression for a single case



Visual progression Images Read from Left to Right, Top to bottom.
1. The spaceship is initialized at the bottom of the field randomly between the black field markers.
2. The spaceship hops onto asteroid 28.
3. The spaceship rides along with asteroid 28.
4. The spaceship hops onto the asteroid 21.
5. The spaceship rides along with asteroid 21.
6. The spaceship hops onto the asteroid 80..
7. The spaceship reaches homebase. Success!

Visual Debugging: A "DEBUG_DISPLAY" variable/flag has been provided in settings.py. Set it to True to show the asteroid IDs and the (x, y) coordinates of the corners of the world in the GUI.

A "DISPLAY_MATCH_RANGE" variable/flag is also provided in settings.py. Set it to True to visualize the range within/outside of which an estimate will not be considered a match. These are shown in the images above as small circles surrounding each asteroid.

A "DISPLAY_ESTIMATED_SPACESHIP_MATCH_RANGE" variable/flag is also provided in settings.py. Set it to True to visualize the range within/outside of the estimated position of the agent (your estimate of where you believe the agent to be). If this flag is activated, The estimated range of the agent is shown as a large green circle (see image below).

A "DARK_MODE" toggle button is available to view the visualization in dark mode. Default mode is currently set to Dark.

Spaceship True position vs Spaceship Estimated position

As previously noted the "TRUE" position of the spaceship is visualized as the **blue** dot with the blue circle range. However the spaceship's "ESTIMATED" position is represented by red dot surrounded by the large **red circle** (Estimates are values you return in part A). To visualize the estimated positions, you will need to return appropriate values from the function (this is optional. See the function docstring for details). When the spaceship's true location matches its estimated location, the estimated range is colored green.

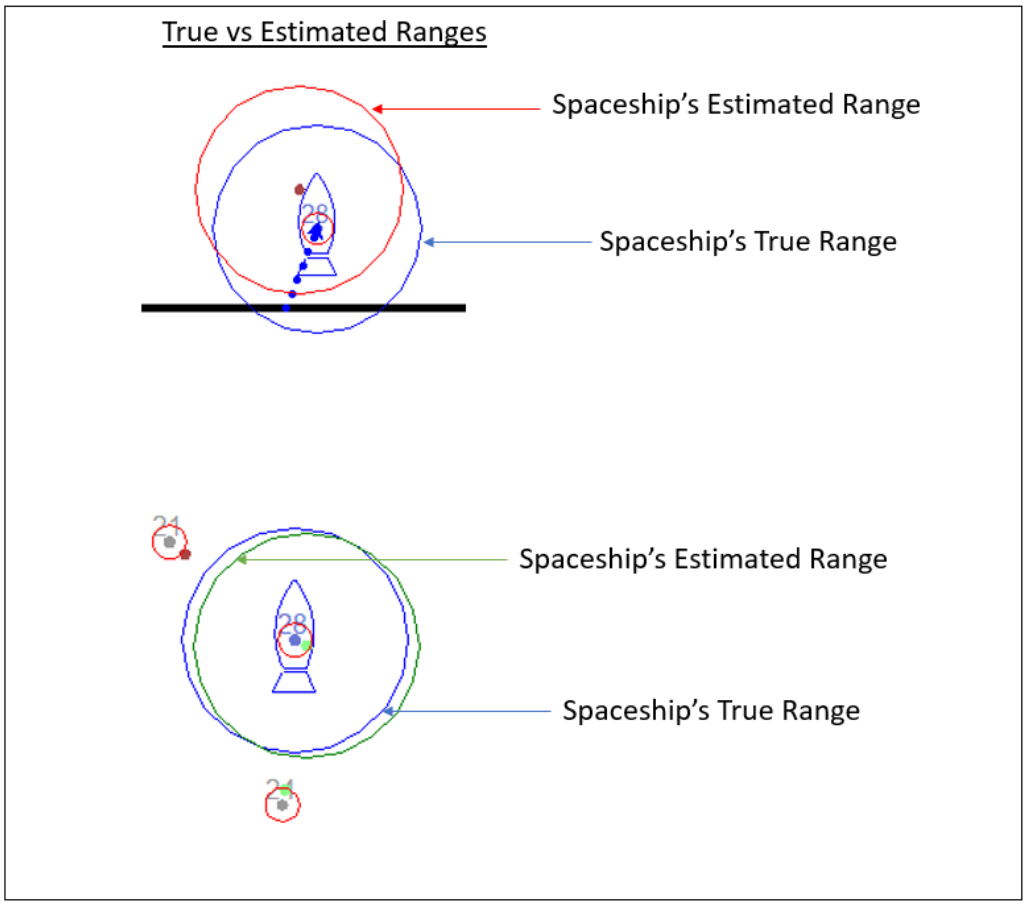


Figure 3: Hopping visualization

Running Tests

To test your code on an estimation case and see a visualization of the simulation, run the following with the necessary case. Cases arguments may be 1 - 15.

```
python test.py --case 1 --display turtle
```

To test your code on an estimation case with no visualization, run the following.

```
    python test.py --case 16 --display text
```

To test your code on a part B case and see a visualization of the simulation, run the following with the necessary case. Case argument may be 16-30):

```
    python test.py --case 16 --display turtle --method jump
```

To run your code on a part B case no visualization, run the following.

```
    python test.py --case 16 --display turtle --method jump
```

To test all local estimate and jump cases with no visualization run

```
python test.py
```

This is the testing mode used by Gradescope.

Generating New Test Cases

The cases used for grading on Gradescope are similar to those provided to you, but not the same. You can use `generate_test_case.py` to generate additional test cases to more rigorously test your code, To see all of the command line arguments for the `generate_test_case.py` script, run the following in your Python environment:

```
python generate_test_case.py --help
```

To create a new case, run as follows:

```
python generate_test_case.py --case my_case# [additional arguments here]
```

e.g.

```
python generate_test_case.py --case 36 --asteroid_match_range 0.09 --num_asteroids_per_time 10
```

To use this new test case, pass the filename to `test.py` using the `--case` argument:

```
python test.py --case 36 --display turtle estimate
```

Note: The new case files are not included in the cases executed by `test.py`. To create a case with x=4 and y=5 bounds, the params should be run as integers e.g.

```
python generate_test_case.py case50 -axb 0 4 -ayb 0 5
```

Scoring

Grades are assigned as follows:
Part A: 70%
Part B: 30% Part A comprises 15 test cases, equally weighted. Part B comprises 20 test cases also equally weighted. For part B only your highest 15 scoring test cases count towards your grade (Note: NO Extra credit!).

Academic Integrity

You must write the code for this project alone. While you may make limited usage of outside resources, keep in mind that you must cite any such resources you use in your work (for example, you should use comments to denote a snippet of code obtained from StackOverflow, lecture videos, etc). Attempts to import or reference methods/variables in the grading suite or otherwise attempt to circumvent the grading suite in your submission will be considered an academic integrity violation. For an example of this, note how the author of this project's code cited the source for the clamp function in utilities.py. You must not use anybody else's code for this project in your work. We will use code-similarity detection software to identify suspicious code, and we will refer any potential incidents to the Office of Student Integrity for investigation. Moreover, you must not post your work on a publicly accessible repository; this could also result in an Honor Code violation [if another student turns in your code]. (Consider using the GT-provided Github server for your repository, or a git server such as Bitbucket that does not default to public sharing.)

Questions

Leverage the class forums to compare strategies with your colleagues as you guide your spaceship home! Good Luck!!