

CS 7638 - Robotics: AI Techniques - Meteorites Project

Fall 2021 - Due Monday, September 20th, Midnight AOE

Introduction

In this project, Earth is threatened by a shower of meteorites falling in your location. It is your task to take sensor readings of the locations of these meteorites, estimate where each of the meteorites will be one tenth of a second later using Kalman Filters, and finally, destroy each meteorite before it hits the ground by firing your laser turret at it.

This project consists of two parts:

- Estimation—estimate the locations of the meteorites—80% of grade
- Defense—aim and fire your laser turret at incoming meteorites before they hit the ground—20% of grade

Submitting Your Assignment

Your submission will consist of ONLY the `turret.py` file, which you will upload to Gradescope. Do not archive (zip, tar, etc.) it. Your code must be valid Python version 3 code; it will be graded with Python 3.9.

Your Python file must execute NO code when imported. We encourage you to keep any testing code in a separate file that you do not submit. Your code should also NOT display a GUI or visualization when we import or call your function under test. If we have to manually edit your code to comment out your own testing harness or visualization, you will receive a -20 point penalty.

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). For an example of this, note how the author of this project's code cited the source for the `clamp` function in `runner.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.)

Detailed Project Description

The motion model of the meteorites is of the form

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

for the meteorite's x-position, and

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

for its y-position. $S_{acc} = \frac{1}{3}$ is a constant.

Time is delimited in discrete steps ($t = 0.0, 0.1, 0.2, \dots$). Each timestep is 0.1 seconds in duration. Each meteorite's motion can be modeled using x, y, dx, dy, a , similarly to how we modeled example systems in the lectures. a is acceleration; note that the x- and y-components of the meteorites' motion are correlated! See the "Note on Deriving the F Matrix for Meteorites" PDF on Canvas (located in Canvas > Files > Misc. Tutorials) for details on how to derive the state transition matrix from the above equations of motion. The Kalman Filter tutorial located in the same directory is also a helpful resource for this project.

Your turret's observations of the meteorites' positions are noisy, so you will leverage the uncertainty-handling properties of Kalman Filters to help you estimate their positions more precisely.

Environment:

In this project, your world is a 2-unit-by-2-unit square, with the X-range $[-1, 1]$ and Y-range $[-1, 1]$; $(-1, -1)$ is the lower left corner, and your turret is located at $(0, -1)$. All meteorite locations and estimates will be with respect to this coordinate system. Meteorites fall from the upper portion of the box to the ground, with the ground being $y = -1$. The laser turret's aim angle is 0.0 rad when the laser is pointed along the ground to the right, and π rad when the laser points along the ground to the left.

Next, we detail the functions you will be modifying in `turret.py` for each part of the project.

Estimation: Function `observe_and_estimate`

For the estimation part of the project, you will be estimating the location of each meteorite visible on the screen one timestep in the future given the noisy measurements of meteorite locations you have for the current timestep (`noisy_meteorite_observations`).

Inputs:

The `observe_and_estimate` function takes in a tuple of tuples of meteorite ID numbers, x-coordinates, and y-coordinates; that is, the `noisy_meteorite_observations` argument has the form

```
((0, -0.83, 0.46),  
 (1, 0.44, 0.8),  
 (3, -0.72, -0.3),  
 ...  
 (1003, 0.34, 0.1))
```

Note that the meteorites in `noisy_meteorite_observations` are not guaranteed to be sorted in any sort of order, so do not expect the ID numbers to be sequential.

Outputs:

The output of the `observe_and_estimate` function should be a tuple of tuples of estimated meteorite locations one timestep into the future.

Goal:

To get full credit for the estimation part of the project, your `observe_and_estimate` function will need to provide “close enough” estimates of all meteorites within the 2-by-2 box within 500 timesteps (50 seconds) (and on Gradescope or when using `test_all.py`, 10 real-world “wall-time” seconds) for at least five (5) consecutive timesteps. A meteorite location estimate is close enough when the Euclidean distance between the estimate and the actual location is less than 0.02 units. If 90% of your meteorite estimates for a case are close enough for five consecutive timesteps within 500 timesteps or (when using `test_all.py`) ten wall-time seconds—whichever is shorter—you’ll get full credit for that case’s estimation portion. Passing back estimations for non-existent meteorites (e.g. meteorites that have hit the ground and have an ID of -1) will not affect your score.

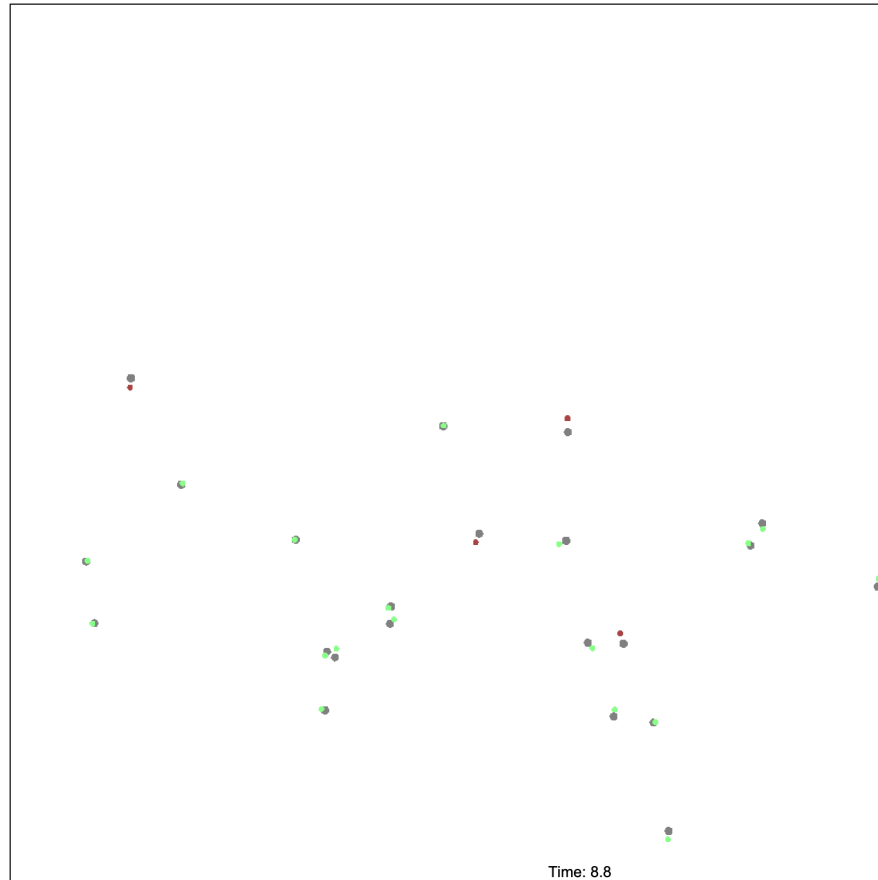
How To Test Your Estimation Code

To test your code on an estimation case and see a visualization of the simulation, run the following in your Python environment (the `case` argument may be 1-8; the command to run case 1 is shown here):

```
python test_one.py --case 1 --display turtle estimate
```

When you run the estimation case with the `turtle` visualization option, you should see something like what is shown in the image below. The gray circles represent the actual locations of meteorites. A red dot indicates an estimation that is too far from the meteorite’s actual location to count as correct,

and a green dot indicates an estimate close enough to be counted as correct.



HINT: On line 24 of `turtle_display.py`, change the `DEBUG_DISPLAY` variable to `True` to show meteorite IDs in the GUI.

A similar command lets you run the test with only text output (no visualization). This text-only mode is what `test_all.py` uses. (see “Testing Everything” below)

```
python test_one.py --case 1 --display text estimate
```

As before, the `case` argument may be 1-8.

Defense: Function `get_laser_action`

For the defense part of the project, you will be devising a simple algorithm to aim and fire your laser turret at falling meteorites. The defense part of the project makes use of the estimations of the meteorite locations computed by `observe_and_estimate` in the estimation portion of the project. (HINT: Don’t over-think your strategy here; perhaps simply aiming at the lowest meteorite

above some minimum threshold is sufficient!) A meteorite is destroyed with probability 0.75 if the laser line comes within a small distance (0.02 units) of it. The laser line itself is 1.1 units of length long, measured from the turret. The laser can only fire a limited number of shots before it runs out of power; the number of shots remaining are displayed in the GUI or command line output.

Each meteorite's ID number is unique as long as the meteorite has not been destroyed. When a meteorite is destroyed, its ID number is set to -1. This ID number change is handled by the simulation. Keep in mind that you may want your Turret to check that it does not try to aim at a meteorite with an ID of -1!

Inputs:

This function takes in a float corresponding to the laser turret's current aim angle, in radians.

Outputs:

The output of this function is either a float or a string:

- Float: The change in aim angle (in radians) you want the laser to move; if the magnitude of this value is greater than `max_angle_change` (0.0873 rad; approximately 5 degrees), it will be lowered to `max_angle_change` rad, but with the sign of the angle you outputted.
- String: Outputting the string `'fire'` will cause the laser turret to fire.

The laser cannot move and fire at the same time. Note that trying to move the laser's aim outside of the $[0, \pi]$ range will result in its aim being clamped to 0 or π , respectively. The laser's aim angle does NOT wrap around—if you output an angle change that would set the laser's current aim to, say, 3.3 rad, the laser's aim will stay π rad until you change the laser's aim back to within the $[0, \pi]$ range.

Goal:

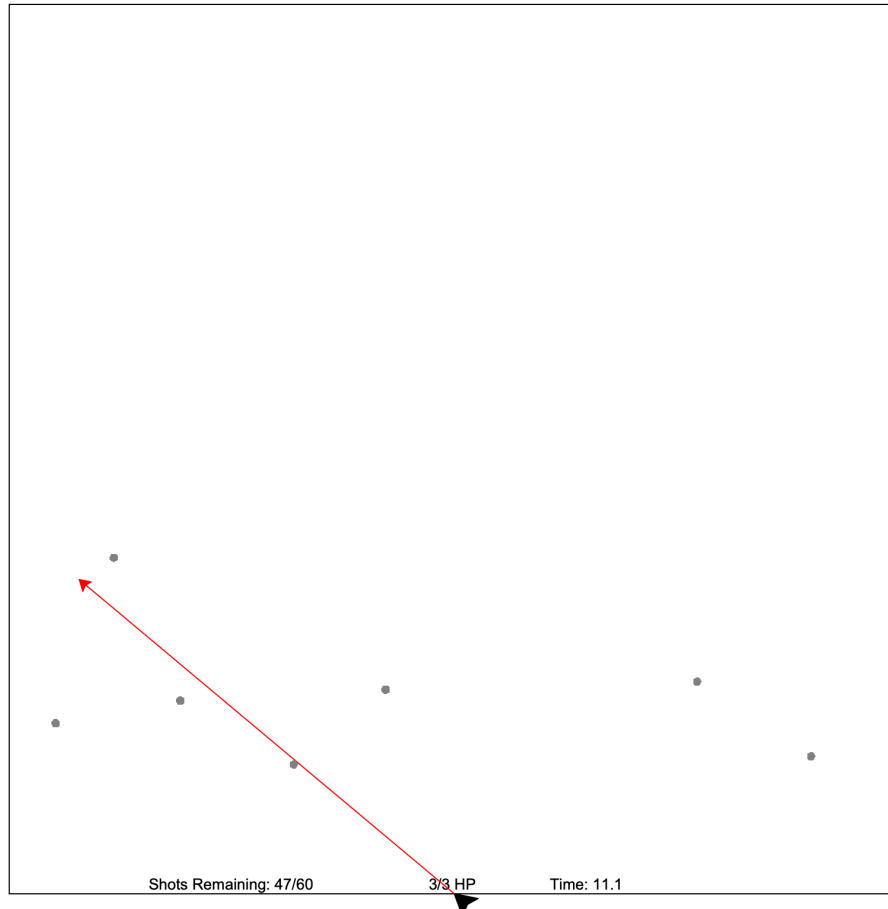
Your goal in the defense part of the project is to make sure your laser turret survives for 500 timesteps. Your laser turret starts with a specific number of health points (HP), which are shown below the turret in `turtle` simulation mode and printed to the command line in `text` mode. Each time a meteorite hits the turret or the ground ($y = -1$), the turret loses one HP. Credit is given for a case if the turret's HP is 1 or greater by the end of the 500-second bout (on Gradescope and in `test_all.py`, there is also a 45-second wall-time time limit); no credit is given if the turret's HP drops to 0 within that time limit.

How To Test Your Defense Code

To test your code on a defense case and see a visualization of the simulation, run the following in your Python environment (the `case` argument may be 1-8):

```
python test_one.py --case 1 --display turtle defense
```

When you run the above command, you should see something like the image below.



HINT: On line 24 of `turtle_display.py`, change the `DEBUG_DISPLAY` variable to `True` to show meteorite IDs in the GUI.

A similar command lets you run the test with only text output (no visualization); this is the mode that `test_all.py` uses to run all test cases. (See “Testing Everything” below) As before, the `case` argument may be 1-8.

```
python test_one.py --case 1 --display text defense
```

Testing Everything

To test all eight of the local estimate and defense cases using the `text` display option, use the command

```
python test_all.py
```

This is the testing mode used by Gradescope.

Generating New Test Cases

The cases used for grading on Gradescope are not the same as those provided to you, though they are very similar. If you wish to generate additional test cases to more rigorously test your code, you can use `generate_test_case.py` to generate new test cases. For reference, here is the guidance that is printed to the console when running the `--help` argument with `generate_test_case.py`:

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

usage: Generate parameters for a test case and write them to file.

```
[ -h ] [ --turret_x TURRET_X ] [ --turret_hp TURRET_HP ]
[ --laser_shots_remaining LASER_SHOTS_REMAINING ] [ --t_past T_PAST ]
[ --t_future T_FUTURE ] [ --t_step T_STEP ]
[ --noise_sigma_x NOISE_SIGMA_X ] [ --noise_sigma_y NOISE_SIGMA_Y ]
[ --nsteps NSTEPS ] [ --meteorite_c_pos_max METEORITE_C_POS_MAX ]
[ --meteorite_c_vel_max METEORITE_C_VEL_MAX ] [ --min_dist MIN_DIST ]
[ --max_angle_change MAX_ANGLE_CHANGE ] [ --seed SEED ]
outfile
```

positional arguments:

outfile	name of file to write
---------	-----------------------

optional arguments:

-h, --help	show this help message and exit
--turret_x TURRET_X	X-location of turret (should be in the range (-1.0, 1.0))
--turret_hp TURRET_HP	Turret's initial health point count
--num_laser_shots NUM_LASER_SHOTS	Initial number of laser shots turret can fire
--t_past T_PAST	time in past (negative integer) from which to start generating meteorites
--t_future T_FUTURE	time into future (positive integer) at which to stop generating meteorites
--t_step T_STEP	add a meteorite every N-th time step
--noise_sigma_x NOISE_SIGMA_X	sigma of Gaussian noise applied to the x-component of meteorite measurements
--noise_sigma_y NOISE_SIGMA_Y	sigma of Gaussian noise applied to the

```

                                y-component of meteorite measurements
--min_y_init MIN_Y_INIT        Lowest initial meteorite y-coordinate
--max_y_init MAX_Y_INIT        Maximum initial meteorite y-coordinate
--nsteps NSTEPS                Number of timesteps to simulate
--dt DT                        Duration of a single timestep
--meteorite_c_pos_max METEORITE_C_POS_MAX
                                maximum magnitude for meteorite
                                position term coefficient
--meteorite_c_vel_max METEORITE_C_VEL_MAX
                                maximum magnitude for meteorite
                                velocity term coefficient
--min_dist MIN_DIST            minimum distance estimate must be from
                                meteorite location to be considered correct;
                                also, if the laser comes within this
                                distance of a meteorite, the meteorite
                                is destroyed with p=0.75.
--max_angle_change MAX_ANGLE_CHANGE
                                maximum increment of the laser's angle,
                                in radians
--seed SEED                    random seed to use when generating
                                meteorites

```

To create a new case, run as follows:

```
python generate_test_case.py my_case.py [additional arguments here]
```

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

```
python test_one.py --case my_case.py --display turtle defense
```

Note: New case files must have the `.py` extension to be imported correctly by the `test_one.py` code, and are not included in the cases executed by `test_all.py`.

FAQ

How do Kalman Filters apply in this project?

As we know the structure of the motion model that governs the motion of the meteorites, we can use Kalman filters to track their locations. Each meteorite has different coefficients in their equations of motion, but for an individual meteorite, those coefficients are constant; thus, we need to estimate as many models as there are meteorites. That is a good indicator that we can initialize one Kalman filter for each meteorite—we will have ($\#$ meteorites) KFs—and use our noisy measurements over time to improve our estimates of where each individual meteorite will be one timestep from now. You'll want to create and update separate \bar{x} s and P s for each meteorite, using the Kalman filter equations.

The state transition matrix (aka motion model matrix, F), measurement model matrix (H), and uncertainty matrices should all be constant and the same for all meteorites. Take a look at the Kalman Filter tutorial and the “Note on Deriving the F Matrix for Meteorites” PDFs on Canvas under Files > Misc. Tutorials for thorough explanations.

How do I share data between `observe_and_estimate`, `get_laser_action`, and other functions in my Turret class?

In your implementation of `Turret`, you can refer to the current turret instance using `self` and attach additional data to it. Here is an example of creating a `value` variable in a `Counter` class that can be used in other functions in the `Counter` class:

```
class Counter(object):

    def __init__(self):
        self.value = 0

    def increment(self):
        self.value += 1

    def show(self):
        print(self.value)

ctr = Counter()
ctr.increment()
ctr.increment()
ctr.show()          # should display '2'
```

Why do I get less credit on Gradescope than I do on my local machine?

Keep in mind that your local computer is likely faster than the virtual machine Gradescope is using to run your code; is there any way you can make your code a little more efficient? Also, when running `test_one.py` for an estimation case, there is no wall-time limit applied; the estimation only needs to be completed within 500 timesteps of simulation. When running `test_all.py`, which is what Gradescope uses, each estimation case is given 10 seconds and each defense case is given 45 seconds; execution time greater than those limits for a case results in an `execution_time_exceeded` result.

Are the cases we have access to the same as the cases used for grading on Gradescope?

No, the case files used on Gradescope are not the same as the cases you download from Canvas. We do our best to make sure that the cases you have access to are

similar in difficulty level to the ones on Gradescope.

How do I get help with my code?

We ask you to do some preliminary debugging before asking for a TA to review your code on Piazza. See the Code Review document on Canvas (Files > cs7638-CodeReviews.pdf) for tips and instructions for how to proceed.

May I use numpy? May I use Sebastian Thrun's matrix class? What other Python packages may I use in my code?

You may use numpy and/or Sebastian Thrun's matrix class in your submission. Please see the course FAQ in Canvas > Files > cs7638-RAIT-FAQs.pdf for details on other allowed Python packages.

May I use Python 2.X? May I use Python 3.5 or lower?

No, please use Python 3.6 or later—preferably Python 3.9. The autograder on Gradescope uses Python 3.9.