CS 7638 - Robotics: AI Techniques - Asteroids Project

Fall 2020 - Due Monday, September 14th, Midnight AOE

Introduction

This project asks you to track a collection of moving asteroids and:

- estimate their future location
- pilot a craft around them to a goal location

The asteroid field

The asteroids travel in a square with corners (-1,-1) and (1,1). Asteroids outside of the bounding box are considered "out of play" for the moment.

Time is delimited in discrete steps (t=0,1,2,...), and the asteroid's XY location is determined by quadratic equations in time:

$$x(t) = a_x t^2 + b_x t + c_x$$

$$y(t) = a_y t^2 + b_y t + c_y$$

Each asteroid's motion can be modeled using x, y, dx, dy, ddx, ddy, similar to the models explored in the lectures.

After drifting outside of the bounding box, most asteroids never return, but a few (<5%) reemerge onto the field later.

Student submission

Your work on this project is to implement the class `Pilot` in the `pilot.py` file. The `Pilot` class must implement three methods:

- `observe_asteroids`: called once per time step, informing the pilot of the latest asteroid measurements. Measurements will include Gaussian noise. Only asteroids currently in the field are measured.
- `estimate_asteroid_locs`: predict the locations of asteroids in the time step after the latest observation.
- `next_move`: given the craft's current state (and previously obtained asteroid observations), choose the next move for the craft to execute.

Your `Pilot` instance is initialized with two variables:

- `min_dist`: in the estimation task, this value specifies how *close* your asteroid location estimates must be to count as a match. In the navigation task, this value specifies how *far* your craft must stay from each asteroid to avoid hitting it.
- `in_bounds`: a rectangle object with properties `x_bounds` and `y_bounds`, each of which contain a pair of min and max values. In the estimation task, you need only estimate asteroids currently in bounds. In the navigation task, your craft may not leave the bounds; doing so will count as a failure.

Task 1: estimation

Estimation counts for the majority of the credit in this project.

On each time step, the pilot will be asked to estimate the locations of all asteroids on the next time step. For example, if the `observe_asteroids` method is invoked at time step `t`, which means that it carries the observations at time step t, the `estimate_asteroid_locs` will be asked to estimate the locations of the asteroids at time step `t + 1`.

The pilot's estimates from the prior step will be compared with the asteroids' current locations. Estimates within `min_dist` will be considered matches.

The estimation is successful if 90% of the asteroids currently active match the prior step's estimates before iteration and temporal limits are reached. See the "Testing your code" section below for more information about the limits.

Task 2: navigation

The navigation task is intended to be tackled after completing the estimation task. Your navigation code will likely make use of your estimation code.

The task initializes a craft below the asteroid field and asks you to pilot it through the field and into a goal area above it. ($y > 1.0$)

The craft (implemented as `CraftState` in `craft.py`) has the following properties:

- current position, heading, and velocity (`x`, `y`, `h`, `v`)
- performance characteristics (`max_speed`, `speed_increment`, `angle_increment`)

Each move by the craft is specified by:

- angle change: the craft may turn left, right, or go straight with respect to its current heading. Turns adjust the craft's heading by `angle_increment`.
- speed change: the craft may accelerate, decelerate, or continue at its current velocity. Speed changes adjust the craft's velocity by `speed_increment`, maxing out at `max_speed`.

Note that in the navigation task, piloting your craft out of bounds is considered a crash, and the bounds are specified by `in_bounds` in the pilot initialization.

The navigation is successful if your craft reaches the goal area before the iteration and temporal limits are reached. See the "Testing your code" section below for more information about the limits.

Testing your code

Two local test scripts are provided with this project:

`test_all.py` runs your code against all available test cases. This script closely mirrors the auto-grader we will use to grade your work. To run:

```
$ python test_all.py
```

`test_one.py` runs your code against a single test case, with additional display options. This script is intended to assist debugging.

```
$ python test_one.py --help
usage: test_one.py [-h] [--case {1,2,3,4,5,6,7}]
                   [--display {turtle,text,none}]
                   {estimate,navigate}

positional arguments:
  {estimate,navigate}   Which method to test

optional arguments:
  -h, --help            show this help message and exit
  --case {1,2,3,4,5,6,7}
                        test case number
  --display {turtle,text,none}
```

For example, to test the navigation task on case 3 with the visualization, run this command:

```
$ python test_one.py --case 3 --display turtle navigate
```

Please note that the gray circles represent the actual locations of the asteroids. Green dots are your estimates for those asteroids that match the asteroid locations, i.e., close enough to the asteroids. Red dots are your estimates which are too far from the actual asteroid positions.

Test cases are provided in the `cases` subdirectory. We will use similar but different cases to grade your code.

These testing suites are NOT complete, and you may need to develop other, more complicated, test cases to fully validate your code. We encourage you to share your test cases (only) with other students on Piazza.

There are two limits that will end execution of a test case before the goal is met. One is iteration based and the other is temporal.

In both the included test files, `test_one.py` and `test_all.py`, as well as on Gradescope, each test case will need to be finished within a set limit of 1000 iterations. A message of "too many steps" will be displayed to the console if the goal has not been met before this limit is reached.

Additionally, in `test_all.py` and on Gradescope (NOT enforced in `test_one.py`), each test case will be limited to a timeout of 10 seconds. It is guaranteed that the grader on Gradescope will have a timeout greater than or equal to the 10 seconds in the `test_all.py` file provided to you. A message of "execution_time_exceeded" will be displayed to the console if the goal has not been met before this time limit. Hitting this limit means that the timeout was exceeded before either the goal was reached or 1000 steps were run.

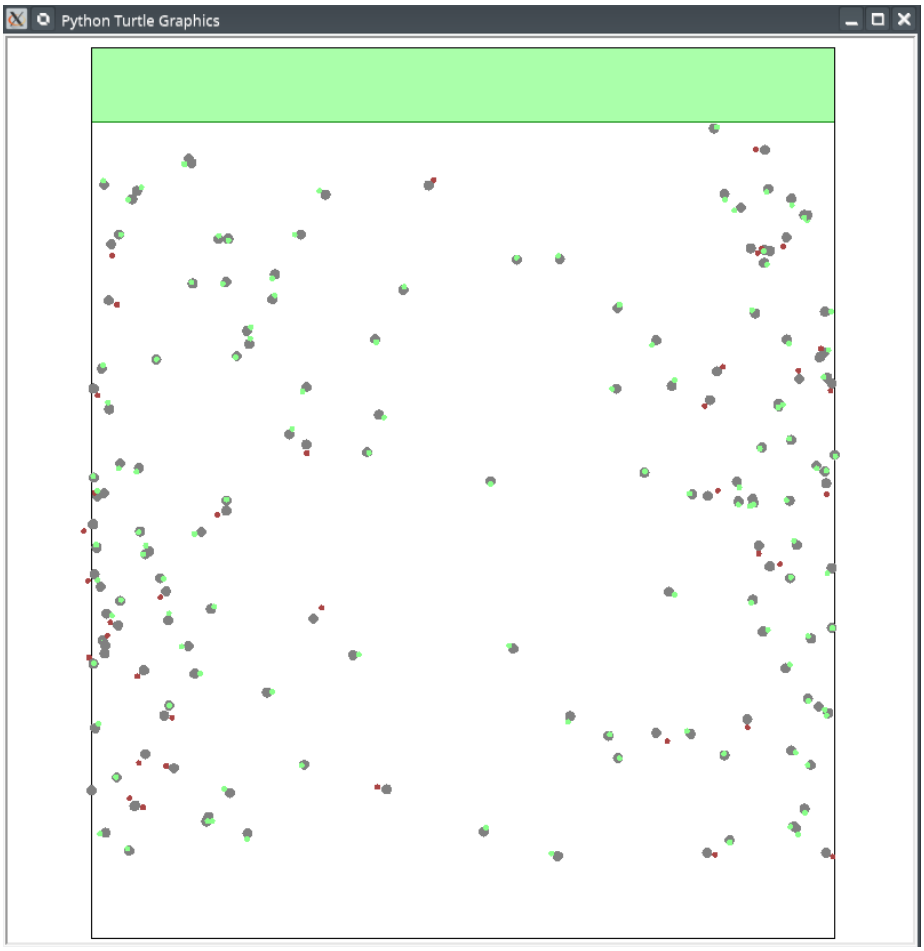

Figure 1: test_one.py visualization

Generating new test cases

The included file `generate_test_case.py` can be used to generate new test cases. It has a number of inputs, as follows:

```
$ python generate_test_case.py --help
usage: Generate a test case parameters and write to file.
[-h] [--t_past T_PAST] [--t_future T_FUTURE]
[--t_step T_STEP] [--noise_sigma NOISE_SIGMA]
[--asteroid_a_max ASTEROID_A_MAX]
[--asteroid_b_max ASTEROID_B_MAX]
[--craft_max_speed CRAFT_MAX_SPEED]
[--craft_angle_increment CRAFT_ANGLE_INCREMENT]
[--min_dist MIN_DIST] [--seed SEED]
outfile

positional arguments:
  outfile               name of file to write

optional arguments:
  -h, --help
      show this help message and exit
  --t_past T_PAST
      time in past (negative integer) from which to start generating asteroids
  --t_future T_FUTURE
      time into future (postigive integer) at which to stop generating asteroids
  --t_step T_STEP
      add an asteroid every N-th time step
  --noise_sigma NOISE_SIGMA
      sigma of Gaussian noise applied to asteroid measurements
  --asteroid_a_max ASTEROID_A_MAX
      maximum magnitude for quadratic asteroid coefficient
  --asteroid_b_max ASTEROID_B_MAX
      maximum magnitude for linear asteroid coefficient
  --craft_max_speed CRAFT_MAX_SPEED
      max speed for the student-piloted craft
  --craft_angle_increment CRAFT_ANGLE_INCREMENT
      heading change increment for student-piloted craft
  --min_dist MIN_DIST
      minimum distance craft must be from asteroid to avoid collision
  --seed SEED
      random seed to use when generating asteroids
```

To create a new case, run as follows:

```
$ python generate_test_case.py my_case.py [... additional argument here ...]
```

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

argument:

```
$ python test_one.py navigate --case my_case.py --display turtle
```

Note: case files must have the `.py` extension to be imported correctly by the test code.

Implementation note

The test cases describe asteroid paths as follows:

$$x(t) := a_x(t - t_{start})^2 + b_x(t - t_{start}) + c_x$$

$$y(t) := a_y(t - t_{start})^2 + b_y(t - t_{start}) + c_y$$

These equations are slightly different than those presented earlier, in that they offset t by $-t_{start}$.

Hints and suggestions

How do I share data between `observe_asteroids`, `estimate_asteroid_locs`, and `next_move`?

In your implementation of `Pilot`, you can refer to the current pilot instance using `self` and attach additional data to it. Here is an example of the technique, implementing a simple counter:

```
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'
```

Should I build one Kalman filter to track all the asteroids?

Create and update a separate μ and Σ for each asteroid, using the Kalman filter equations. The motion model matrix (F), measurement model matrix (H), and uncertainty matrices should all be constant and the same for all asteroids.

What is a good strategy for navigation?

Compare your craft's possible paths over the next few time steps with the asteroids' predicted paths over that same time frame.

On each time step, your craft has nine possible moves: three possible direction changes times three possible speed changes. Looking ahead two time steps, your craft has `9*9 = 81` possible paths; looking ahead three, `9*9*9 = 729`. As the number of paths increases exponentially in time, your code must strike a balance between computational efficiency and navigational accuracy.

To estimate the asteroids' future positions, combine your estimates of their current location with your motion model. Keep in mind that your future estimates will lose accuracy the farther out you forecast.

Once you have forecast both the craft's and the asteroids' future locations, rank the craft's next moves according to some measure of quality and choose the move with the highest score. It is up to you to construct this measure, but you will want to incentivize (1) avoiding asteroids and (2) moving toward the goal.

Submitting your assignment

Your submission will consist of the pilot.py file (only) which will be uploaded to Gradescope. Do not archive (zip,tar,etc) it. Your code must be valid python version 3 code, and you may use the numpy library.

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).

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 repository or a repo such as Bitbucket that doesn't default to public sharing.)