Core Team Interview Project

Introduction

The purpose of this homework project is to test your coding and **OOP** design skills.

In this project, you will create a complete but small system that generates noisy data that should fit a certain shape. then, estimates the original shape from the data, and last tests the estimation against the shape.

The project is a combination of these three parts (generator, estimator, and tester). The generator is mandatory while the other two are optional.

Whether you decide to complete all of the assignments or just parts of it, please make sure you submit a **high-quality** code and that you follow all the guidelines, because we prefer **quality** over quantity.

Please read the steps and guidelines sections before starting.

Steps:

- Before you start, think of a simple design that follows the OOP concepts
- When you get an idea of how you want to design your project, please create a UML class diagram that describes your design and attach it to your project
    - What is UML: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml
    - What is a class diagram: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/
    - A tool for creating a UML diagram: https://app.diagrams.net/
    - **IMPORTANT: please design the whole project, even if you decide to implement only parts of it**
- Create a git project in http://github.com
- Develop the project and commit to your git repo. Exactly as you would do at work
- As already said, if you feel that the project is taking too much time to implement, you can decide not to implement the estimator or the tester. You can also decide to support fewer shapes.

Guidelines:

- Use python's virtual environment
- Use pip, not conda
- Give meaningful names for variables, functions, classes, and files
- Choose a naming convention and use it consistently all over the project
- Keep a clean and well-organized code
- Remove all unnecessary comments
- Give meaningful commit messages

- Your project should contain:
    - README.md file - detailed explanation of how to install and run your project
    - requirements.txt - contains all your project's dependencies
    - design.drawio - contains your UML class diagram
    - the rest of the python code files you implemented
- When you finish, please send us only the git repo URL.
- If you have any questions, or if something is not clear, please don't hesitate to contact us at yossi.avital@briefcam.com
- Good luck!!!

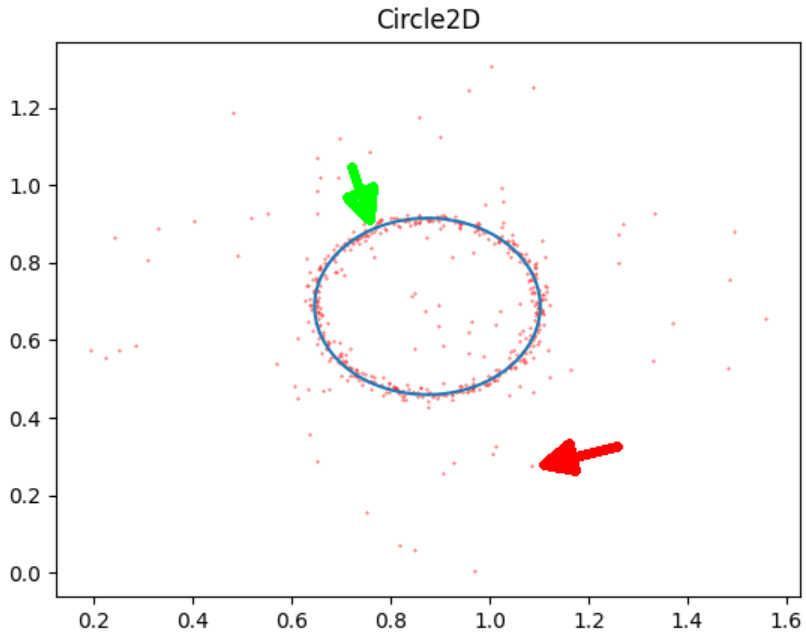
The three parts are as follows:

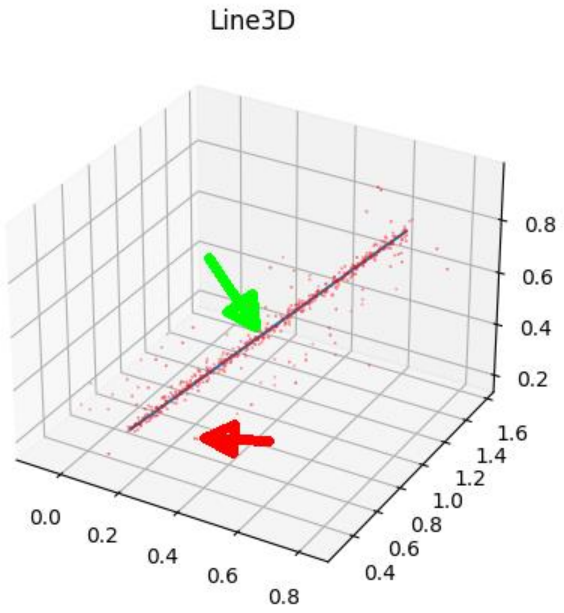
The Generator

The generator will receive several desired shapes from a config file (see below) and generate two types of points:

- Inlier points/ real-world noisy data (will constitute 80% of the generated points) – points close to the shape (we can also call the shape “ground-truth”) that originates from the shape itself plus some noise
- Outlier points (will constitute 20% of the generated points) – points that have nothing to do with the shape and are located far away from it

See the figures below (green – inlier; red – outlier):





The supported shapes are: a line in 2D space, a circle in 2D space, a line in 3D space (optional, not mandatory), and a plane in 3D space (also optional).

Create a script called generator.py that accepts the following:

- config_path: string - the path for the config.json file
- output_path: string - the path of the output file
- debug: boolean - when turned on, the generator will plot the shapes with the generated points, and any other debug information you want.
- any other params that you need

The generator will perform the following pseudocode, according to the config.json file:

- for each 'shape' in 'config.shapes':
    - for each i in range(shape.value):
        - randomly determine the params of the 'shape'
            - for example: Line3D has 6 params (x0, y0, z0, a, b, c) in parametric form or 2 3D vectors in vector form.
        - generate and store 'num_points' of points around the current 'shape'
            - choose whatever random function/s you want to generate the noisy points with, and allow the user to control the level of noise for the inlier points only using the ‘randomness’ parameter (the outlier points characteristics can be hardcoded)
        - if debug is on:
            - plot to the screen the generated geometric shape with the generated noisy points

- save all the generated points and shapes params in the file 'output_path' in any way you choose (serialization for example)
- the output should contain at least:
    - the generated points (the noisy data)
    - the shapes params (the precise model, the ground truth)

The Config File

The config file is a JSON file that defines which shapes to generate and how many instances to generate for each shape.

In addition, the config file determines how many points to generate for each instance and how noisy the data has to be.

A config file for an example:

```
{
  “shapes”: {“Line2D”: 4, “Line3D”: 8, “Circle2D”: 1, “Plane3D”: 3},
  # shape.value is the number of instances per shape you ought to generate
  “num_points”: 500,
  “randomness”: 0.7 # ranges between 0 to 1 and represents the level of noise for the inlier points
}
```

The Estimator (optional)

The estimator is responsible for estimating the shapes params from the noisy data.

To achieve that, please implement the RANSAC algorithm https://en.wikipedia.org/wiki/Random_sample_consensus.

You are not being tested on the quality of the estimator, so please implement the **simplest** RANSAC you can think of. Just make sure your code outputs reasonable results and nothing more.

Create a script called estimator.py that accepts the following:

- input_path: string - the path of the input file (the output of the generator.py)
- output_path: string - the path of the output file
- debug: boolean - when turned on, the estimator will plot the estimated shapes with the given points, and any other debug information you want.
- any other params that you need

The estimator will perform the following pseudocode, according to the input file:

- for each 'shape' in 'input_file':

        - estimate and store the params of the 'shape' using RANSAC and the shape.noisy_data
        - if debug is on:
            - plot to the screen the estimated shape with the given noisy points
- save all the estimated shapes params in the file 'output_path' in any way you choose (serialization for example)
- the output should contain at least:
    - the estimated shapes params

Note: Despite the fact, the estimator has access to all of the generator's output including the ground truth it must not use it in any way, only the generated noisy points

The Tester (optional)

The tester is responsible for determining how good the estimator is.

To do that, the tester gets the ground truth from the generator and the estimated models from the estimator and calculates the distance between them.

Create a script called tester.py that accepts the following:

- gt_input_path: string - the path of the ground truth input file (the output of the generator.py)
- est_input_path: string - the path of the estimated model input file (the output of the estimator.py)
- debug: boolean - when turned on, the tester will plot the ground truth shapes, the estimated shapes, and the noisy points, and any other debug information you want
- any other params that you need

The tester will perform the following pseudocode, according to the input files:

- for each 'shape' in 'gt_input_path' and 'est_input_path:
    - calculate, print, and store the distance between the ground truth model and the estimated model
        - you can decide on any distance function you want
    - if debug is on:
        - plot to the screen the estimated shape, the ground truth shape, and the given noisy points
- print final score for the estimator
    - you can decide on any aggregation function for the final score that you want, but print one final number