CS 8803: AI for Robotics

Final Project

Overview:

For this project, we will be moving out of the realm of simulated robot data and into a real-world situation. You will be provided partial video clips of a HEXBUG Nano micro robot moving in a wooden box and tasked with predicting the movement of the robot after the video subsequence. There is an obstacle (a candle) inside the box. In this screenshot you can see the hexbug just above and to the left of the candle.

There will be ten videos, each 62 seconds long and recorded at 30 frames per second. For each video, you will receive the first 60 seconds (1800 frames) of training data. You then need to predict the positions of the robot in each of the following 2 seconds (remaining 60 hidden frames) without any new measurement data. (Yes, this is unlike the move-measure paradigm we covered in lectures.) Your project grade is determined from the accuracy of your predictions over all of the videos as well as a **Project Report** documenting your work and analysis.

The pedagogical purpose of this exercise is for you to apply the knowledge and techniques you have learned in this class to real-world data. Since real-world data is usually messier and require different variables to work with than simulated data, this will give you experience with the practical, as opposed to the theoretical, side of robotics.

Description of the robot: *The HEXBUG Nano is a tiny, collectible, micro robotic creature that uses the physics of vibration to propel forward and explore its environment. Powered by a tiny motor, and 12 fixed, angular legs, the industrious critter traverses the ground beneath it and quickly navigates ... When coming into contact with an object in its path, the energetic insect will switch directions and scurry away on a new path due to its persistent random behavior. (In our videos, the robot is roughly 1.5mm per pixel.)*

Although the project involves predictions based on video data, it will **not** be necessary to use Computer Vision techniques. In addition to each of the 10 test videos, we will provide you with a text file that contains extracted centroid coordinates (a pair of integers) for the video that you should use as input to your algorithm -- note that noise exists in many forms throughout this project, and this is intentional. If you choose to do additional video processing, you will be able to submit your own version of processed input data for grading.

Please finally note that you need to **adapt** any techniques covered in video lectures past our simple implementations and variables. In order to make a more accurate prediction, additional outside research into more sophisticated techniques will be necessary. For the questions: *Can I use X algorithm or Y technique or precompute Z assumptions?* The answer is yes. We encourage thoughtful experimentation, discovery, and cleverness that lead to better accuracy. Except for violations of the Student Honor Code, everything is fair game.

Specifications:

Write a Python program that reads a list of data points representing the coordinates of the centroid (pairs of integers) at roughly center of mass of the robot in each frame of the video file. Your program should return a prediction of the coordinates of the centroid corresponding to the robot's position for the next 2 seconds, i.e. 60 frames, after the end of input.

Your prediction results should be in the form of 60 pairs of integers, each pair separated by a newline, with the two elements of the pair separated by a comma. The prediction should be output in a text file called `prediction.txt`.

Sample input (1800 rows):

```
121,45
129,44
...
587,175
```

Sample output (60 rows):

```
621,245
615,237
...
235,39
```

Each pair of integers represents one frame of the video. There will be 60 remaining frames for testing in each video that are not shown; therefore you will need to give 60 pairs of integers as output. The first element of each pair is the x-coordinate of the centroid, given by the number of pixels from the left side of the video window. The second element of each pair is the y-coordinate of the centroid, given by the number of pixels from the top of the video window.

The main executable of your program should be named `finalproject.py` and invoked as follows:

```
$ python finalproject.py input_file
```

Where `input_file` is the name of one of the 10 provided test files containing a list of centroid positions. Again, the result will be a list of 60 coordinate pairs in the same format output to a file called `prediction.txt`.

We will grade your project by running another Python program which runs the process `python finalproject.py test01.txt` and then compares the resulting output file `prediction.txt` to the actual (permanently witheld from you) data. This process repeats for all of the inputs to calculate your score. (See the bottom of the **Grading** section for more details.) You will be given a copy of this program -- it is called `grading.py`.

It should take **no longer than 1 minute to output all ten predictions**, i.e. your `finalproject.py` should take no more than 5-6 seconds each time it is run. Do not modify the grading program and please use the same filenames as provided.

In order to have a standard deployment platform you will be provided with a Virtual Machine image which runs Linux. This image will include most of the libraries you are likely to need as well as all the files you need to get started with the project. Your code will be tested on the provided Virtual Machine, so you must make sure it executes correctly in that environment. Note that the VM does not have a GUI installed, only a command-line interface. This means that if you would like to run Turtle or similar to aid you in visualization during development of your project, then that would need to be run on your local machine.

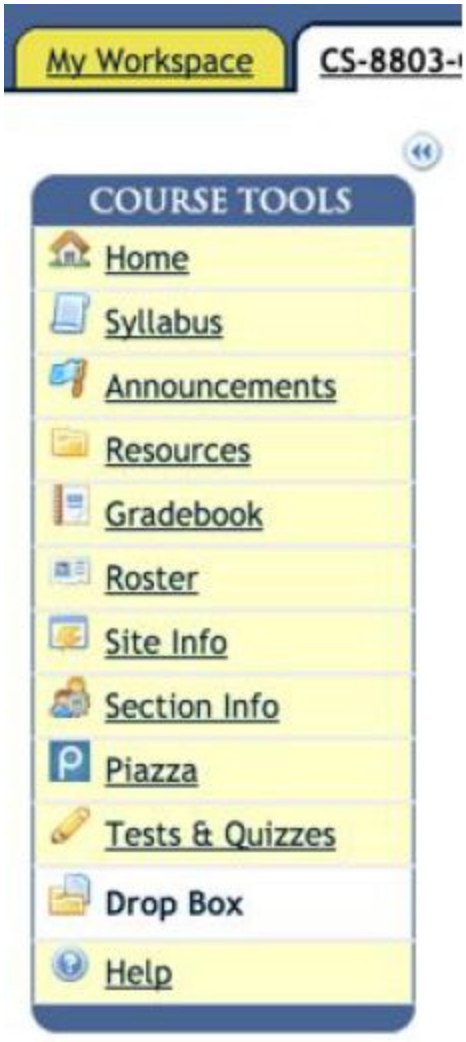
You may install and use additional Python libraries. If you do so, you must provide a list of commands used to install them in your `readme.txt` file, so that the graders can be sure to run the same commands on the grading VM before running `grading.py` on your project. If you use a library but forget to tell us how to install it and we have to email you during the grading period or figure it out on our own, this will result in a deduction.

In addition to writing the code, you will need to include a Project Report that describes your approach. See **Grading** section below for details on what information to include in your Project Report.

Submission:

One person per team should submit their team's project. That team member should put a zip archive containing the requisite files into their DropBox on T-Square.

You can find your DropBox here:



Include the following in the .zip archive that you submit:

- File called members.txt which contains the list of all your group members and their @gatech.edu email addresses. One name per line.
- Source code, including `finalproject.py` as well as any additional source files, all of which should be in Python. Make sure `finalproject.py` is at the top level, but feel free to place additional files in subdirectories if you prefer.
- If your code requires additional libraries not available on the provided Virtual Machine image put instructions on how to install them into a file called `readme.txt`

- Include your Project Report in PDF format in a file called `report.pdf`
- Folder called `inputs/` which contains the 10 provided inputs, or your own version of the inputs if you choose to process the video file yourself and generate different centroid data from those provided.
- All the above files should be placed into a folder called `finalproject`, which is in turn zipped up into a single archive called `finalproject.zip`

An example submission would be structured as follows:

```
finalproject.zip
| ------ finalproject/
      | ------ inputs/
            | ------ test01.txt
            | ------ test02.txt
            | ------ …
      | ------ finalproject.py
      | ------ members.txt
      | ------ readme.txt
      | ------ report.pdf
```

You can submit any number of times before the deadline. Only your final submission will count. Again, to limit confusion, please coordinate with your team members as to who will present the final submission. It is your responsibility to coordinate and work with your teammates for this.

If you prefer not to use the T-Square Drop Box interface, you may instead map your T-Square Dropbox to a local folder on your computer. Instructions for this may be found by going to https://t-square.gatech.edu/portal/help/main and then clicking “Drop Box” and then “Uploading files to Drop Box using WebDAV”.

Grading:

The project will be scored out of 40 points. Including extra credit, the best possible score for a team is 45/40. The breakdown for grading is as follows:

- 0-5 points: **Meeting specifications**
    - Full credit for this section will be given if your submission meets all requirements in this document.
    - Points in this category will be deducted for:
        - Missing information about installing prerequisites in the required readme.txt file
        - Not outputting a list of 60 pairs of integers (outputting floating-point numbers will result in a deduction)
        - Program that takes longer than 1 minute to execute all 10 test cases

    - In general you will lose one point for each manual fix that graders have to do to your submission, or its output, in order to run and evaluate it. Note that required fixes here may result in points lost elsewhere too.
- 0-5 points: **Functionality** - The following criteria will be used as a guideline.
    - The program should *correctly* implement at least one filtering, tracking, or path-planning technique taught in the course *or* beyond the scope of the course. Note that errors in implementation will result in deductions. Also note that you are free to use external libraries (within reason), as long as a) you give proper attribution, b) you give installation instructions for the graders, and c) the installation process is easy and friendly.
    - The program's functionality should match the description given in the Project Report.
    - The technique you implement should be correctly applied to the problem. Simply implementing an algorithm but not applying it to the problem in a sensible way will result in deductions. Make sure you justify this in your report.
    - To be awarded credit, you are responsible for helping the grader run your program on their copy of the project VM. Again, note that required fixes here may result in points lost elsewhere too.
- 0-15 points: **Project Report** - The following criteria will be used as a guideline. (Note that up to 2 extra credit points are possible here.)
    - Teams with the top three reports will each be awarded additional 2 points of extra credit.
    - Your report should demonstrate an **understanding of the problem space** and the **implementation of your algorithm(s), technique(s), and process** at a reasonable level of sophistication for a graduate level Computer Science course.
    - To earn full credit, you must discuss why your chosen algorithm gives the best results, compare against other algorithms, and support your argument with detailed, insightful analysis. A picture is worth 1000 words so feel free to use visual aids liberally.
    - For the graders' sanity, your report should be *no longer than* 15 pages long. There is no minimum length, but make sure to fully document your approach.
    - There should be concise but thorough comments describing your program's functionality in your source code. Pasting small snippets of your source code into your report to aid discussion is fine, but please do not include superfluous source code in the report PDF.
- 0-15 points: **Accuracy** - We will judge your project's accuracy using the method described below. (Note that up to 3 extra credit points are possible here.)
    - Teams which submit valid solutions will not receive fewer than 5 points.
    - The top three teams with the most accurate predictions will receive 18/17/16 points respectively for their place.
    - Teams in the next 10% will receive 15 points.
    - Teams in the following deciles, 11-20%, 21-30%, etc will receive two fewer points per decile (13 points for 11-20%, 11 points for 21-30%, etc).

        - For example if 50 teams submit solutions: the top 3 will receive top marks, the next 5 will receive 15 points, the following 5 will receive 13, the following 5 will receive 11, the next 5 will receive 9, the next 5 will receive 7, and the remaining 22 teams will get 5 points.
        - Exception: you will be provided with a sample solution which just predicts that the hexbug will remain at the last known point for a further 60 frames. The (in)accuracy of this solution will be considered as a baseline. Your solution should certainly not do worse than this! If it does, you may receive fewer than 5 points for accuracy.
    - Finally, while earning full credit for this section may seem challenging, all teams may earn strong overall marks with consistently strong performance across the other sections.

Scoring: Your program's accuracy will be judged against the centroid positions in the remaining 60 frames of each of the ten test videos. We will compute the L2 error between your prediction and the actual data. The lower your error, the better your accuracy.

See below for an example with 4 frames of data:

```
prediction = [[0,0],[10,0],[10,15],[25,25]]
actual = [[0,0],[5,5],[10,10],[20,20]]
```

$$\sqrt{dist([0,\ 0],\ [0,\ 0])^{\ 2}\ +\ dist([10,\ 0],\ [5,\ 5])^{\ 2}\ +\ dist([10,\ 15],\ [10,\ 10])^{\ 2}\ +\ dist([25,\ 25],\ [20,\ 20])^{\ 2}}$$

$$\sqrt{((0-0)^{\ 2}\ +(0-0)^{\ 2})+((10-5)^{\ 2}\ +(0-5)^{\ 2})+((10-10)^{\ 2}\ +(15-10)^{\ 2})+((25-20)^{\ 2}\ +(25-20)^{\ 2})}$$

$\sqrt{(0+0+25+25+0+25+25+25)} = \sqrt{125} \approx 11.18$

The average of your project's L2 errors over all test cases, after dropping the worst and best scores, will be used to judge your submission's accuracy.

Extenuating Circumstances:

If you have extenuating circumstances (severe illness, a death in the family, etc.) you may contact the Office of the Dean of Students to verify the situation, and they will instruct us on whether or not to allow a late submission. Aside from that, no credit will be given for submitting assignments late. Please also inform your teammates and TAs as soon as you are able. Upon verification from the Office of the Dean of Students, it can be possible to arrange an extension beyond the end of the semester for the affected student.

If a team member disappears and does not contribute to his/her team, the remaining team members should let the TAs know and we, along with the professor, will determine the appropriate course of action. You need to report this situation to us *as early as possible* (within the first week or two) or otherwise you will be held responsible for your teammates.

Aside from the above circumstances, all members of a team will receive the same grade.

Optional Resources:

Also included are several tools and libraries. Feel free to use or modify as you like. Please note that many of these tools are being introduced for the first time this semester and thus may have limited documentation and may take some additional installation/work to run.

- **Submission Test**: a simple testing harness to make sure your script takes in arguments and outputs files correctly. If you are unfamiliar with file I/O in Python, feel free to see the placeholder scripts for basic reading/writing.
- **Visualizer**: scripts to help visualize the path taken by the robot and a comparison of two sets of points (i.e. predicted path vs actual path). See the `exampleViz.py` script for example of how to use and `examplePath.jpg` and `exampleCompare.jpg` images for samples. Requires OpenCV to run.
- **Student-made visualization tool** (with permission given to share): https://github.com/mehmetbajin/gt-omscs/tree/master/8803-1/final-project

- **Tracker**: computer vision scripts used to generate the data from the videos. Uses several techniques including a fusion contour-based and color-based foreground detector and a regional filter. The algorithm operates on image sequences rather than video, so the frames will need to be extracted first with an external program (ImageMagick, MPEG Streamclip, etc) into a folder XXXX with image frames prefixed

with XXXX for a video XXXX. Parameters for running `hexbug_tracker.py` can be found using the -h help flag. Requires OpenCV to run. Definitely feel free to ask questions on Piazza regarding use of the tracker and we will do our best to help.

More Training Data: Because most models succeed or fail on their data and amount of data, we included a video for 20 minutes of training data. For example, if you want to model how noisy the sensing algorithm is or how the hexbug collide with walls/objects, then you can generate a model by preprocessing this dataset to find noise values or predict effects of collisions. Feel free to use or not.

Version Control: We strongly recommend that teams use either Github or another version control system (SVN, Mercurial, Perforce, etc) to organize collaborative programming or to work effectively in general. All Georgia Tech students can create private Github repositories at github.gatech.edu with your GT account and password. More: http://drupal.gatech.edu/handbook/github-georgia-tech

But do not make any source code accessible to the general public. Doing so or making use of any public solutions are both violations of the Student Honor Code; and please let us know if there are any public solutions. We take this seriously. If we encounter violations, we may fail the associated student(s) or retroactively fail them.

Thanks for reading everything. Good luck!