COMP-417 Robotics
Assignment 4
Due Dec 7, 2017

November 26, 2017

Version 2.9 of this assignment specification. The latest version is at http://www.cim.mcgill.ca/~dudek/417/417-asst4-2017.pdf with extra details (sample output) occurring Friday Nov 24, so check back on Friday end-of-day.

1 Change log for this document

2.3: Since Version 2.0 of the assignment, an extra section called "Explanation" has been added at the bottom and some typos corrected.
2.3.1: Typos
2.4: Details on the avoidance of urban tiles added, and tip on when to implement this (see Troubleshooting).
2.4.1: Provided picture illustrative of working system
2.5: Mentioned the fact that your total path should be 200 to 400 tiles long. Using too long a path could require a lot of disk space for the tile images. Also added explicit indications of how bonuses might be computed. Added extra troubleshooting note regarding **sklearn.model_selection** on Trottier-building computers.
2.5.1: Wording changes, Notably, re-worded suggestion in Sec 1.5.1 for clarity, and Sec 1.3 (item 1) about running PCA with **multiple** different values for the number of components . You need to examine how the number of components impacts the result (hint: not that much unless you make the problem itself harder by adding an extra class).
2.6: Added **anchor=tk.NW** to the method call **canvas.create_image(0, 0, anchor=tk.NW, image=photo)** in the sample code in mydrone.py so that the image is anchored at the top-left, which seems to be more intuitive.
2.7: Tip 2.4.2: regarding TileServer **imagePixelsToLL(self, actual_pX, actual_pY, zoomLevel, imx, imy)** that maps image images to lat, long. 2.8: Final revision of the mydrone.py code providing more hints and a more intuitive demo. Careful not to clobber any code of your own if you download it.
2.9: Original zip file failed to include training image in classes directory.

2 GPS-guided Drone simulation

2.1 Overview

This assignment involves building a high-level vision-based drone navigation system. You have some latitude in the precise task description (see below), but the idea is to develop a simulated drone that can selectively fly over some kinds of terrain, such as forests, while avoiding urban environments. Your simulated drone will do this using real satellite data, a vision-based terrain classifier, and a probabilistic path generation algorithm that you invent.

This is supposed to sound impressive, interesting and "big", but actually be quite realizable since you will provided with some tools to simplify the task. Make sure you read the subsection below on how this will be evaluated.

2.2 Specifics

You will be writing code to fly a simulated "drone" over a landscape. Your simulation should move around one tile at a time and look down at the image it sees below it. Look in the TileServer code to determine the size of the GPS change that is required to move by one tile (256 pixels at zoom level 18). Your drone should be able to move forward or turn left or right: the specific rate and parameters for these turns are up to you.

As you move, you should feed the tiles you "see" to the classifier called **classifyOne()** in the geoclass module. It will classify tiles as either water, urban, or arable (parkland) and return a corresponding class number (from 0 to 2). You should avoid flying over urban tiles: if you enter one, back up to where you came from.

Your mark is based on two things you need to do:

1. Alter the number of components for the PCA/SVM learning module (line 345 of **geoclass.py**, which reads `n_components = ...`) and re-run the learning, Plot the results versus the number of components and explain them in writing. You can re-run the PCA learner once by running

   ```
   python geoclass.py -test
   ```

   This will print a small table showing the accuracy of the classifier given the current settings. Test how this changes as you alter the number of basis functions between 3 and about 18. If you run the program without the **-test** flag the learned model will be saved for use with the drone simulator.

2. Fly the plane to cover a region of terrain using two alternative coverage algorithms implemented int he method **draw_objects()** of **mydrone.,py**. While doing this, avoid tiles that are urban: if you enter one, go back (make sure you start on a non-urban tiles). One of these coverage algorithms should be random (Brownian) motion which is your baseline (take a small random motion on each step). The other is a coverage algorithm you can choose or invent, but the idea is to fully cover a region as if you were search for a lost person or object on the ground. For each algorithm, compute the number of tiles covered in a fixed amount of distance travelled, and the distribution of terrain types visited. Plot the results on the map. Record you sequence of tiles you visit and generate a "video" of the flight. Submit these items with a description of what you did. Your total path covered by the drone should about 200 to 400 distinct tiles (don't double count for this). You should also report the ratio of unique tiles visited to total tiles visited (i.e if you go back and forth between 2 tiles 50 times, this ratio is about 2:50, or 0.04).

2.3 Implementation details

2.3.1 Files

In the associated zip file http://www.cim.mcgill.ca/~dudek/417/asst4.zip, you will find three pieces of core code, some data and some utility code:

1. A file called **mydrone.py** which serves as the stub of a program to control your simulation. Put your code here,

2. A file called **TileServer.py** which can be included by mydrone, and which provides satellite images based on GPS coordinates. It can return a huge mosaic of tiles, or just one tile for a specific position. See also Tip 2.4.2

3. A file called **geoclass.py** that does classification (using PCA) on image patches. You can examine the last 10 lines of this file to see how to call the key routines.

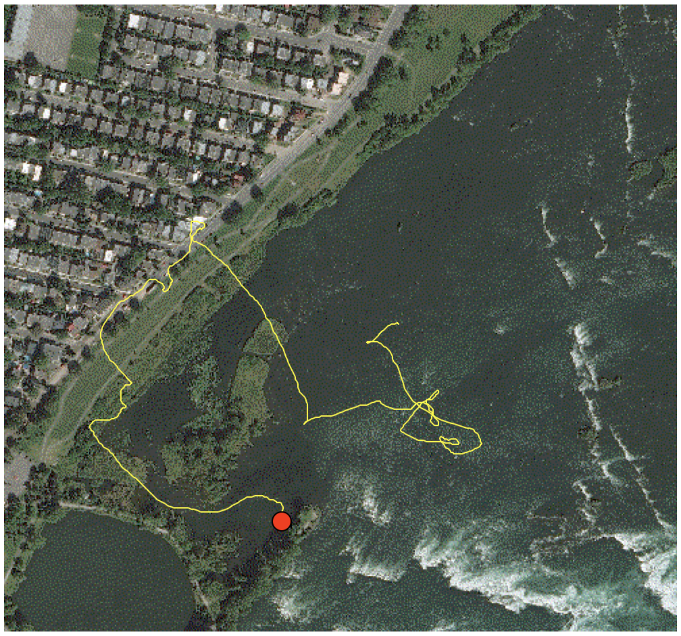

Figure 1: Simulated flight recording

4. A folder called "classes" that contains directories holding example images used to learn each terrain class (*water, urban, arable*)
5. A file called **classifier.state** that stores pre-learned classifier data. If this is absent a new one will be generated from the examples in the classes folder. This is created and used by geoclass and not meant to be inspected manually.
6. A file called **loader.py** which is a utility module for geoclass used to load the images associated with the different classes of terrain. No need to touch this.
7. A file called **drawSample.py** that you used in the RRT assignment is provided. It is not really used, but provides sample code on drawing on the Tk window.

You should also download the file `http://www.cim.mcgill.ca/~dudek/417/mydrone.py` (if available) for the latest version in case any extra hints are provided on Nov 24th.

2.3.2 Next steps

Getting the image for your current GPS location can be achieved using the **TileServer** function:

```
tiles_as_image_from_corr(lat, lon, zoomLevel,  tilesX, tilesY, tilesOffsetX, tilesOffsetY)
```

where **lat,lon** is your drone's current position (latitude and longitude) and the zoomLevel should be fixed at 18. Using **tilesX=1, tilesY=1, tilesOffsetX=0, tilesOffsetY=0** allows you to get just the current tile under you.

Note that the **geoclass** method **generate_and_train_classifier** has a variable called **verbose**; setting this to **1** provides more output which you may or may not want to see.

You can use any programming language you like, but you will want to use sample code provided to you in Python, so that is the natural and appropriate choice. You will need to draw some figures and assuming you are using Python you will want to use PIL (Pillow) and/or Tk (the drawing library which is not pretty, but easy to use). The mydrone code provided already uses Tk as does the drawSample module. OpenCV is also an option and a good tool to know, but it is not required, recommended or expected.

Sample output from **geoclass** if run from the command line:

```
01. python geoclass.py
02. Loading saved state from classifier.state
03. urban
04. (1, 'urban')
05. urban
06. (1, 'urban')
```

On lines 3-4, and again on lines 4-5, above, we see the output from the classification method, which classified an image file as an "urban" environment (class #1). The method is invoked at the end of the file. You would then use this method in your drone simulator by importing the geoclass module.

2.4 Troubleshooting and tips

2.4.1 Order of work, and avoidance of urban tiles

I suggest that you implement urban tile avoidance last. First do the PCA evaluation. I suggest you plot the results by hand, avoiding the CS-person trap of automating stuff that can be done more easily manually. Then implement basic coverage. If you do these two things, you should be pretty quick and you will have learned much of the code and "locked in" a lot of the marks. Then add the classification and avoidance part.

2.4.2 Finding the latitude or longitude from a position on the returned TileServer image

You might want to use the TileServer **imagePixelsToLL(actual_pX, actual_pY, zoomLevel, imx, imy)** that maps image images to lat, long. Depending of how you proceed this may not be needed or desirable, but it might be useful for some people. It takes the returned actual_pX, actual_pY from the TileServer which is a position on the earth's surface corresponding to the top-left corner of a mosaic image, and (imx,imy) which is a location within a mosaic image, and returns the coordinates.

2.4.3 sklearn.model_selection

Some machines in the Trottier labs only have an older version (0.17) of sklearn installed. This will lead to an **ImportError**. We have asked the CS staff to install a newer version (0.18 or later), but in the meantime you need to install a local version of sklearn in your local directory or else use a different computer as described below.

If you see this error on your own personal computer, you can just do: `pip install -U scikit-learn`
(The **-U** flag is used to update.)

On CS machine where you lack root (superuser) status, I suggest you install scikit-learn in your local directory as follows:

```
pip2.7 install --user --install-option="--prefix=" -U scikit-learn
```

Note that the command **pip2.7** is a special CS version of pip that uses Python 2.7: you may want to adjust this to suit your local situation and preferred Python version.

2.4.4 Pillow version issues

If you get an error like:

```
Tostring() has been removed. Please call tobytes() instead
```

your version on Pillow (the Python image processing module) might be too new for the Python version you are using. You can try installing an *older* version of Pillow via via the horrendous:

```
    pip uninstall pillow
    pip install Pillow==2.1.0
```

2.4.5 Printing in geoclass.py

Note that printing in the geoclass module uses the Python3 style print statement syntax that works like a function. Otherwise all this code is based on Python 2,7.

2.4.6 Zoom levels

Zoom level 18 is recommended, but the higher zoom level 20 also works in the Montreal region. Zooming levels more than 20 will probably not return useful image tiles, and even zoom level 20 will not work in some places including rural parts of Quebec.

2.5 Evaluation

Plot the PCA/SVM results and explain them in words. Provide a rational explanation for what you observe, but not this is not a stats or machine learning class and you are not expected to exhibit great subtle knowledge of PCA and or SVM classification. This is worth 25% of the assignment grade.

Illustrate the paths for you follow graphically, drawn on top of satellite maps. Explain the results and comment of the pros and cons of the two algorithms. Show the ratio of classes covered, avoid bad stuff (urban tiles), show coverage ratio to total distance travelled. As noted above, provide a description of what your code does, record you sequence of tiles you visit and (ideally) generate a "video" of the flight.

You can earn up to 40% bonus marks by doing other interesting things of your choosing, such as doing computing tile transition probabilities (very small bonus), processing at a finer resolution then 256-pixel tiles (which involves changing the TileServer to make it return tiles even smaller than those returned by Bing, using fractional tiles [bigger bonus]), or implementing another search method, or adding another terrain class (which is probably a bit painful).

Don't worry about bonus marks if you are out of time and have exams to study for: get this in on time and move on with other priorities.

2.6 Explanation

As discussed in class, PCA (principal components analysis) is a dimensionality reduction technique that takes a high-dimensional object (like an image) and allows us to approximate it in allow-dimensional subspace. An SVM (support vector machine) can then be used to define a threshold between different classes. For example, if we were classifying people response to "is it warm or cold" and SVM might discover that the threshold of 20°C was a good separator between temperatures described as warm and cool. In higher dimensional spaces, this threshold is called a liner separator or a hyperplane. Non-linear separators also exist and geocode.py supports using them, but they are not recommended for this assignment.

GPS coordinates refer to points on a spherical projection. Since we usually think of moving around on a locally-planar region, we need to do some conversion in general and specifically for this assignment. You should recognize that these tiles and the planar approximation you are using relate to the notions of a **chart** and **atlas** that we have seen in class long ago.