**ECE 276A – Visual Inertial SLAM**

This project is intended to take as an input a dataset from a car with IMU measurements and features. The code does localization with the IMU data and Extended Kalman Filter (EKF) prediction, mapping of the landmarks with EKF updates, and then visual inertial SLAM by combining the above two steps.

This project exists to satisfy the requirement of UC San Diego class ECE 276A, Project 3. The purpose of this project and the class as a whole is for the student to learn and get familiar with performing Visual, Interial, SLAM

**Build status**

This project has been completed and tested. It can be seen in the accompanying project report that the program completed SLAM on datasets successfully. This included showing a vast improvement over using only a dead reckoning approach. Additionally, some of the code in this project may be adaptable to other projects, such as functions that transform frames, as well as performing the EKF prediction and update.

**Code style**

The goal of the student was to code this project in the Python Pep8 style, according to python.org, however, as this is project is the first time the student has coded in Python, there are some styling errors in the code that the student will look to resolve in later versions.

**Tech/framework used**

For the first part of this project, converting all the data to useable quantities, the main tech framework used was creating rotation and transformation matrices. This was done by following the information laid out in the UCSD ECE 276A lecture slides, which are referenced in the included project report. The numpy library was used extensively in all parts of this project for its ability to easily and quickly reference, perform computations on, and handle large amounts of data in arrays.

For the second part of this project, implementing an EKF, the theory used was also described in the UCSD ECE 276A lecture. The process of implementing this was done by following the mathematical approach laid out in the above mentioned slides.

**Code Example**

The easiest way to run this example code is to comment out everything the “j” loop in the main function except the part A, and view the map of the IMU location that is outputted. This can also be run on different datasets in order to see different outputs.

**Installation**

To install, please make sure the user is running Python 3.7.0 or later, with the numpy library and the math library.

Next, the user must be sure to place the included datasets in the same directory as this code is being saved. After setting the desired map as described in the code example, the user can run the program to perform SLAM. The user can also input their own data, provided it is in the same format as the included datasets (meaning essentially it must be from the same car and sensors to take advantage of the tuned parameters), and this program will be able to handle it and perform SLAM – its parameters are set-up to give good overall performance for any map, not finely tuned to one specific map.

**Important Note**

Due to memory problems, when running this code, either run it without SLAM (by commenting out Part C in the “j” loop of the main function, or run it only with SLAM (by commenting out the parts A and B). If both are tried to run at the same time, the user will not get desired output as a result of memory issues due to the large amount of data being inputted.

**High Level Description of functions and code of main loop**

load\_data:

Given as part of the utils in the project folder. Loads the dataset selected.

Visualize\_trajectory:

Plot the trajectory of the car pose and locations of the landmarks passed in. Note that this function was also given in the untils function. It was edited to allow another parameter (the landmarks) to be passed into and plotted the same way as the pose was plotted.

Vect2hat:

This function takes a 3 element vector as input and outputs the hat-map matrix.

IMU2world:

This function takes the car mean (inverse pose) as input and transforms it to the world frame, returning pose in SE(3)

createM:

Takes as input M and b and creates the M matrix used in the EKF update step.

piProjMatrix:

Takes as input a vector and removes the depth information from it.

dPidQ:

Takes as input the projection function (q) and returns the derivative of it.

GetHzHat:

Takes as input k, car mean, landmarks mean, M matrix and P matrix. It returns the jacobian matrix and z-hat vector.

updateMarxMean:

Takes current feature, current imu pose, and the given cam\_T\_imu. Returns new value of the mean that landmark.

getUHat:

Takes as input the current time step, current rotational velocity, current linear velocity. It returns the u-hat matrix and the u-curley-hat matrix.

Vect2hat46:

Takes as input a vector with 6 elements and returns the hat map matrix for that vector.

**Main Function:**

The lines up until the “j” loop load the data and initialize all the variables. They are well commented and should be easy to follow.

Loop with all time instances:

Find the change in time between the first and last time instances.

Part a: (commented out for running part C)

Get u-hat and u-curley-hat by using the above mentioned functions

The car mean and car covariances are updated with equations from slides

Predicted mean and covariance saved.

Part b: (commented out for running part C)

Loop through all features:

Check if the feaure is observed

Check if feature has been initialized

If not, update the mean

If feature has been initialized (the only way to get to this point):

Find the Jacobian and z-hat

Find Kalman gain

Update mean of current landmark

Update covaraince of current landmark

Part c: (comment out to run parts A & B)

Create W matrix with motion noise and rotation noise

Find u-hat and u-curley-hat

Predict the mean and covariance of the car

Save the car mean and landmark means

Loop through all features:

Check if the feaure is observed

Check if feature has been initialized

If not, update the mean

If feature has been initialized (the only way to get to this point):

Find the Jacobian and z-hat

Find Kalman gain

Update mean of current landmark

Update covariance of current landmark

Find the Jacobian to update car covariance

Find the kalman gain

Update the car mean

Update the car covariance

If once every 100 loops

Print the trajectory of the car and landmarks from Part c

Print the trajectory of the car and landmarks from Part c

Print the trajectory of the car and landmarks from Parts A and B

**Note from the student**

Much of the main loop could have been parsed off into functions, however the student was tuning parameters and trying different ways to resample and find correlation so the code was left in the main loop, and the program works well and is tune-able in that configuration so the student elected to leave it in that configuration and explain it in the pseudo code above.

Please also note that the code is commented in depth as well, as it is the goal of the student that the code be easy to understand for the user.

**Credits**

ECE 276A Course Website: https://natanaso.github.io/ece276a/

ECE 276A Piazza: https://piazza.com/class/k4tbsww5fuu4y0?cid=34#

Study Group: Roumen Guha, Maria Fatima, Stephen West