CSE 468/568 Assignment 4
Vision Based Perception

The objective of this assignment is to perform camera calibration and performing visual odometry. Create a new package called `lab4`. Download the files required for this assignment. Extract them in your `src/lab4` folder.
In this assignment, you will have multiple nodes, multiple launch files, and you will also submit a PDF.

1) Perform Camera Calibration (20 points)

For this section, you will calibrate your phone camera and calculate the intrinsic parameters. Print the chessboard pattern from:
https://github.com/opencv/opencv/blob/master/doc/pattern.png
on a letter sized paper. Stick it to a wall and take 20-30 images from various angles and distances, use the images to calibrate your camera. There is also one printed chessboard pattern taped to the glass panels of Davis 106, if you cannot print your own. You must submit your images along with your code, inside from `src/lab4/images/`. This will be one of your nodes in this assignment `lab4_cal.py`. Once your node spins, it needs to load the images from `src/lab4/images/` and print out the intrinsic parameters of your camera. You **ARE** allowed to use OpenCV functions for calibration (but only OpenCV).
https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html
https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
Test to see if you can launch everything by running the command:

```
$ roslaunch lab4 lab4_cal.launch
```

2) Read Ros bag File and Visualize Data (10 points)

Ros bag files are used to record/store ros messges, which can be "played" later. The message inside a bag file will be published on the same topics they were recorded from, as if they are being generated in real-time. Check out http://wiki.ros.org/Bags

For this portion, you will play the ros bag file shared with you and visualize the data. In the data that you should have downloaded and extracted, we have provided a ros bag file. The bag file should be `src/lab4/bag/sample.bag`. The bag file contains left, right and depth video feeds from a stereo camera along with a lot of other topics, you can use `rqt_bag` to explore the bag file. Go over all of them and visualize the data in rviz. Specifically, we pick left and right RGB images.
For this section, you don't need to write a ros node, but you will create a launch file to play the bag file and open rviz with your configuration. Make sure you include the rviz

configuration file in your package in the appropriate folder.
Test to see if you can launch everything by running the command:
`$ roslaunch lab4 lab4_bag.launch`

DO NOT include the dataset and bag folders in your submission.

3) Publishing kitti dataset and Visualization (25 points)

In this assignment, we want to perform visual odometry. Based on what we have learned, a node can perform visual odometry, while it can access data via ros topics. Instead of using real sensors, we can use well-known datasets; data that has been recorded and labeled, and can be used to benchmark implementations. This will allow repeatability and comparison/benchmarking. We want to use a well-known dataset named **KITTI**. Sometimes datasets are available in ros bag files. However, our dataset consists of raw images and data. We would like to use the data as if they are being published in real-time. In this section, you will create a node `view_kitti.py` that reads the raw data, and publishes them in appropriate topics. The data set is provided to you in the `src/lab4/dataset` You are required to create a node named `view_kitti.py` that will publish the images present inside `sequence/00/image_0` to the topic `kitti/image` and corresponding ground truth poses from `pose` folder to topic `kitti/pose`. This will be a `geometry_msgs/TwistStamped` message. Note that the messages should have a sequence id, so they can be in sync (i.e. a subscriber can tell which pose is for which image). For the ground truth pose, the file consists of 12 columns (0 to 11), column 3 is X coordinate and column 11 is Z coordinate, considering a world coordinate frame with XZ plain on the road. The file also includes many more lines than there are images. This is because we are only working on the first 1001 images of the full dataset. You can simply ignore the rest of the ground truth data.

You should visualize the published images and ground truth position on rviz. You will create a launch file to launch your node and open rviz with your configuration. You can show the position with a simple marker message on rviz (no orientation needed). You can also replace Z with Y in this part, to use the X-Y plane on rviz. Make sure you include the rviz configuration file in your package in the appropriate folder.

Furthermore, your node should utilize a rosparam "fps" which tells the node at which rate it should publish images and pose. Use the default value of 10 in your launch file.

Test to see if you can launch everything by running the command:
`$ roslaunch lab4 lab4_kitti.launch`

DO NOT include the dataset and bag folders in your submission.

4) Performing Monocular Visual Odometry (35 points)

In this part, you will perform Monocular Visual Odometry (VO)[1] on the images you published previously and compare the trajectory with the ground truth.

We have provided you with an implementation of monocular visual odometry estimation between two consecutive images. You will use this to compute complete trajectories and compare with the ground truth.

The zip file you downloaded has `lab4.py` and `mono_vo.py` scripts. `mono_vo.py` contains a monocular visual odometry implementation containing *featureTracking, featureDetection, getRT* which you will use for this part. You will use *getRT()* to obtain the VO. It accepts two images and returns the rotation matrix and translation vector between the two frames (frame_0 to frame_1). The rotation matrix (R) is a 3 x 3 matrix and the translation vector (t) is a 3 x 1 matrix.

Since the images are received one at a time, you will need to remember the previous image and call *getRT()*, passing the previous image and the most recent one. The rotation matrix and translation returned by *getRT()* is between two frames is a relative trajectory. In order to create a full trajectory, all poses need to be re-projected to the same coordinate frame. We consider the initial image is taken at the origin. Since each odometry estimation is relative to the last one, you may need to remember the projection matrix too. So after every pair of frames you need to do the following.

For Translation Vector:

$$Translation\_Vector = Translation\_Vector + Rotation\_Matrix.dot(t) \quad (1)$$

For Rotation Matrix:

$$Rotation\_Matrix = Rotation\_Matrix.dot(R) \quad (2)$$

Where R and t is returned by *getRT()*.

Refer to the lecture slides on combining multiple rotations, and the comments provided in the code. Here is some background: https://www.brainvoyager.com/bv/doc/UsersGuide/CoordsAndTransforms/SpatialTransformationMatrices.html

You DO NOT need to modify `mono_vo.py`. But you'll have to read through it to complete the assignment.

`lab4.py` is the script where you will complete the functions given for plotting the obtained odometry (`plotTraj()`) and calculating Absolute Trajectory Error or ATE (`computeError()`).
Once the trajectory is calculated, you should plot the estimated trajectory together with the ground truth.

[1]For details of a mono VO pipeline, read this blog post https://avisingh599.github.io/vision/monocular-vo/

Inside `lab4.py`, you will find all the instructions commented out. Please read the comments mentioned in the file **very carefully**.

NOTE: Your node should receive image frames and ground truth poses only via subscribing to the topics published in part 3. You are not allowed to directly read them from the dataset folder.

Here is the sequence of steps you will need to complete:

- On launch, start the `view kitti.py` node written in previous part that publishes images and ground truth poses from the KITTI dataset, and the `lab4.py` node from this part.
- In `lab4.py` subscribe to the image topic as well as the ground truth pose.
- In the image callback,
  - Pass the two most recent images to *getRT()* in `mono_vo.py`. Note that you'll have to skip all these steps for the first image since there is no prior image to compute Visual Odometry with.
  - If *getRT()* returns invalid results, the most recent frame must be ignored and dropped.
  - If the result is valid, transform relative pose change to get absolute pose in the initial coordinate frame.
  - Append the new pose estimate to a global list.
- In the pose callback, append ground truth pose to a global list.
- Plot the estimated trajectory together with the ground truth trajectory. You can use `cv2.imshow()` to update the plot as you update the trajectory.
- For each entry in the estimated pose list, find the corresponding ground truth pose and compute error for each pair.
- Sum all of them to compute the Absolute Trajectory Error (ATE).

Absolute Trajectory Error

ATE is an efficient way to determine the performance of a Visual Odometry pipeline. The ATE can be evaluated by comparing the absolute distances between estimated and aligned ground truth trajectory.

The ATE is defined as the root mean square error from error matrices:

$$ATE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(P_i - P_{gi})^2} \tag{3}$$

where:
n is the total number of frames.
P_i is the point from the estimated trajectory at current frame.
P_{gi} is the point from the ground truth trajectory at current frame.

HINT: You calculate P_i by projecting the pose estimate from the `getRT()` function in `mono_vo.py`, into the inital coordinate frame. P_{gi} is published by `view_kitti.py`. The error is the Euclidean distance in 3D between the two positions.

You may only use the libraries already imported inside the scripts.

Test to see if you can launch everything by running the command:
`$ roslaunch lab4 lab4_vo.launch`

DO NOT include the dataset and bag folders in your submission.

5) Understanding Monocular VO (10 points)

Now that you have completed the previous part, it is important to dive deeper and understand how the implementation works. This part will contain questions from the implementation provided. Read it carefully and answer the questions mentioned below. Please note that for grading, you need to write your answers in a PDF and submit it as **lab4.pdf** in your package.

Question 1. What feature detection and tracking method is used in the provided pipeline? Can we use any other method? if yes, what? Please describe in only a couple of sentences.

Question 2. What steps do we take to calculate the essential matrix? Can you find the intrinsic parameters of the camera using essential matrix? If yes, please write steps on how to achieve it? Please refer to this document to learn more about essential matrix: https://en.wikipedia.org/wiki/Essential_matrix

Question 3. What was the mean ATE (Average Trajectory Error) between the estimated trajectory and provided ground truth?

Question 4. How can you optimize the ATE in this case? What possible issues can you find in the given pipeline?

Submission Instructions

You will submit `lab4.zip`, a compressed archive file containing the lab4 package (folder). The folder should contain your calibration photos from part (1) in `src/lab4/images/` folder, your source files and scripts, your rviz configuration file, and your launch files. It will also contain the `lab4.pdf` file for the last part.
Remove the dataset and bag folders before submission. We will add them under the same name for grading.
The folder should compile if we drop it into our catkin workspace and call `catkin_make`. Please take care to follow the instructions carefully so we can script our tests. Problems in running will result in loss of points.
Note: Remember to include your `rviz` configuration files.
Note: You must submit your calibration images within `src/lab4/images/`
Note: You must **NOT** include the bag file or the dataset files. Including these folders are unnecessary, and it only increases the submission size for upload/download. Make sure you remove these folders before your submission. Otherwise a penalty will be applied.

Please use UB Learns for submission.
The assignment is due Friday, November 4 before midnight.
DO NOT include the dataset and bag folders in your submission. Otherwise a penalty will be applied.