EE565: MOBILE ROBOTICS

LAB # 7: CALIBRATE AR DRONE'S CAMERA AND PERFORM ONLINE OPTICAL FLOW

DESCRIPTION & MOTIVATION

Vision constitutes one of the most important parts of modern day studies in robotics. In this lab students shall work with AR Drone's front camera. They will learn how camera calibration is done and how a simple optical flow application can be built.

IN-LAB TASKS

CAMERA CALIBRATION

1. Start the AR Drone, make wireless connection, use ardrone/front/image_raw topic to acquire camera feed.
2. Run the calibration node from the package "camera_calibration".
3. Remap the topics in command terminal:
   a. /image:=/ardrone/front/image_raw
   b. camera:=/ardrone/front
4. Also specify the size of chessboard i.e. 9x6 and the square size i.e. 0.025 meters in command terminal. [Take help from wiki.ros.org]
5. Now place the quadrotor in a static state, and bring your flat chessboard in front of the camera.
6. Take a lot of snaps (automatic) of the chessboard with different skews and scale of the board (I.e. take snaps at different distances from the camera).
7. Once the 'Calibrate' button lights up, click it and it will print the camera parameters in terminal. Identify the Camera matrix and Distortion matrix. Write the focal length and principal point offsets on a piece of paper and get those checked.

LUCAS-KANADE OPTICAL FLOW

You have to implement the optical flow algorithm studied in class. Here are some guidelines, but you need to implement it with you own code.

8. Take help from here
   https://code.ros.org/trac/opencv/browser/trunk/opencv/samples/cpp/fback.cpp?rev=4692
9. This is a simple implementation of optical flow algorithm.
10. Note that this code will not work for the AR Drone camera feed. So you need to write a simple subscriber (using image_transport) that will take images from AR Drone's camera.
11. For each image callback, run the optical flow calculation function i.e. `calcOpticalFlowFarneback`.
12. Optical Flow algorithm requires two successive images, so you'd always need to keep track of the last image used, and its features throughout execution.
13. You also need to take care of image conversion from rgb to grayscale to run this function.
14. Also, to find features in an image (that you'd need to do in the first iteration), use `goodFeaturesToTrack` function.
15. Show your output (i.e. arrows) in a separate window, live. For this you need the image_transport publisher.

LAB ASSIGNMENT

1. Complete the Optical Flow implementation (Lucas-Kanade) started in lab and get it checked with different parameter choices i.e. maximum number of features, different termination criteria, etc.
2. Implement Optical Flow algorithm using SIFT features. You may use previous made implementation and replace the part of feature detection that was using `goodFeaturesToTrack` function with a function that finds SIFT features.
    a. Note that SIFT features are not returned in the form of 2D points on the image, so you'd require finding a short code excerpt that converts those features to 2D image points
3. **Bonus:** You've implemented Optical Flow algorithm by remaining in image world. For bonus, you have to find a scale of translating this motion into real world. So, keep the background and camera static. Place an object in the view and move it horizontal by maintaining fixed distance from the camera about 1 meter. Now, find the difference in image pixels of the translation. This pixel difference represents 1 meter for an object placed at fixed distance. Use this scale to find the motion of camera in a static background. You'd require taking mean and variance of all image features used in optical flow