21 November 2018

General Information:

You have to submit your solution via Moodle. We allow **groups of two students**. Please list all names in the submission comments, upload one solution per group. You have **two weeks of working time** (note the deadline date in the footer). To get the 0.3 bonus, you have to pass at least three exercises, with the fourth one being either completely correct or borderline accepted. The solutions of the exercises are presented the day after the submission deadline respectively. Feel free to use the Moodle forum or visit us during office hours (every Thursday 10:00 – 11:00) to ask questions!

For this project we use **Eigen** (http://eigen.tuxfamily.org/), **OpenCV** (https://opencv.org/) and **Ceres** (http://ceres-solver.org). The provided libraries are pre-compiled on Windows for Visual Studio 2017, but they can be compiled for other environments following instructions on the provided websites (for Linux instructions are added). The camera and depth frames have been added to the folder **assets**.

Expected submission files: World.h, keypoints.png, matches.png, results.png

Exercise 4

Tasks:

1. Feature Extraction and Description

1. Visualize frames with OpenCV via imshow (but comment out later)
2. Extract features via the OpenCV ORB detector from rgb and store into keypoints
3. Describe features via the OpenCV ORB descriptor from rgb, keypoints and store into descriptor

2. Feature Matching

1. Write brute force matching function brute_force between two frames. It calculates for every descriptor (in frame-idx0) an index to the descriptor (in frame-idx1) that has the closest distance. The distance function is evaluated with hamming_distance. Write the matches into matches_all. matches_all is a vector of pairs where first is the index to idx1 and second is the hamming distance.
2. Filter matches from matches_all and store them into matches_filtered. There are 2 filtering criteria: (1) The Hamming distance should be lower than 40. (2) The L2-distance between keypoints[idx0][i].pt and keypoints[idx1][matches[i].first].pt should be lower than 40 pixels.
3. A small present from us: You can verify matches with show_matches.

21 November 2018

3. Bundle Adjustment

1. Construct the cost function in construct_cost_function for Ceres. Feed all the matches from all frames into it via problem.AddResidualBlock. Calculate how many pose parameters you need: 1 pose is defined by 6 parameters (3 trans, 3 rot), 1 pose describes 1 camera frame. Figure out the dimension of a single residual (Check lecture about Bundle Adjustment).
2. Complete operator() in CostFunctor. Need to back-project from frame0 to world, then project from world to image1 (make sure you know what to do with intrinsics, de-homogenization and applying poses/inverse poses)

4. Submit your solution

a. Screenshot of the very first frame from 1.1: keypoints.png
b. Screenshot of matches between frame 0 and frame 2 (as in 2.3): matches.png
c. Initial cost, final cost and accumulated drift (as screenshot of console) (as in 3.2): results:png
d. Submit World.h