TTK4255 Robotic Vision, Spring 2020

Final Project

Simen Haugo (simen.haugo@ntnu.no)

Important dates and guidelines

- Deadline for registering project and group: Monday, March 9th
- Final delivery deadline: Sunday, April 26th
- Final presentations: Monday and Tuesday, **April 27th and 28th**
- Your project score counts toward 20% of your final grade.
- You can work in groups of up to 3 people.

Link to project registration form

docs.google.com/spreadsheets/d/1HUtEKrlPkqSl2t77wefUvUyiEWPf7IEe2k82Wx72DMU/

Final presentation

Except for the hyperspectral imaging project, there is no written report. Instead, you will receive a score based on your presentation and your slides.

- All group members receive the same score.
- Students working alone have 10 minutes to present.
- Groups of two have 13 minutes and groups of three or more have 15 minutes.
- On the deadline each group must hand in their slides as a PDF to Blackboard.
- The first slide should show your names, group number and project number.

Larger groups are expected to have more impressive execution on the evaluation criteria. Students who are in their third year and lose two weeks due to excursion are not expected to have done as much and will be evaluated fairly. Other circumstances that reduce the time you have to work on the project should be reported to me (Simen) via email.

Exception: Hyperspectral imaging project

If you choose to do the hyperspectral imaging project, you will not give an oral presentation, but instead hand in a written report. More details can be found in that section. The same deadline (Sunday, April 26th) applies for the report.

Project suggestions

$\boxed{1}$ Propose your own project

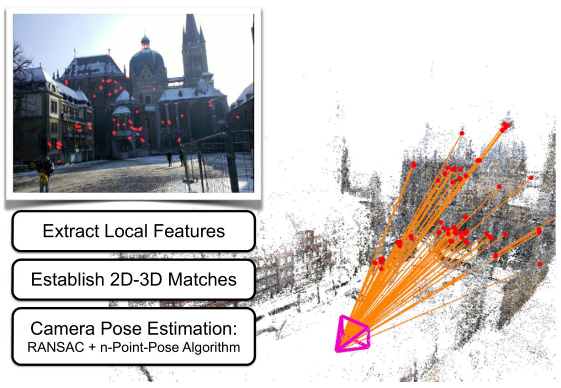
If you want to propose your own project, you need to send me (Simen) an email with a brief project description, including what you plan to implement yourself and what you plan to reuse from existing software.

- You must get your project proposal approved within Monday, March 9th
- You are still allowed to work in groups of up to three people. Only one of you need to send the e-mail for approval.

Your final presentation should include the following:

- *Goal*: What did you want to achieve or learn with the project?
- *Related work*: Shortly summarize how the problem has been solved by others.
- *Methods*: Describe the methods you used.
- *Results*: Show what you achieved.
- *Discussion*: What worked well and what did not work well? Were there unexpected results or unforeseen challenges?
- *Future work*: Things you would like to try but didn't have time for.

2 Structure-based visual localization



Left (from [Sattler, 2015]): Typical steps involved in a structure-based visual localization system. Right: A place on Gløshaugen campus.

Goal

Visual localization is the problem of determining the exact position and orientation from which a query photo was taken. The goal of this project is to implement a 3D structure-based localization system for (a small part of) Gløshaugen campus.

Expected core functionality

More details are in the resources section, but generally a 3D structure-based localization system works as follows:

1. *(Done once beforehand):* Create a 3D point cloud model of the places of interest, where each 3D point is associated with an image feature descriptor (e.g. SURF).
2. Given a new query image, establish 2D-3D matches between features detected in the query image and 3D points in the model via descriptor matching.
3. Estimate the camera pose using a PnP solver inside a RANSAC loop.

You are expected to do each of these steps. At the end, you should have a runnable program that estimates the pose of a given query image relative to a coordinate frame that you define. One of the possible extra credit tasks is to create a single 3D model that contains all the places in a shared coordinate frame. This can be difficult to do, so for the core functionality you are allowed to create separate, small 3D models for a few places, e.g. sections of Hovedbygget and Ohma Electra. This can be done using a two-view 3D reconstruction approach, as in exercise 6. In that case, the pose for a query image can be defined relative to either of the views that the model was triangulated from.

Evaluation

Your project score is decided based on the criteria below.

- Core functionality (**50%**):
  - You have acquired 3D point cloud model(s) using your own camera
  - You can match features between a query image and the model
  - You can estimate camera poses using PnP and RANSAC
  - You can successfully localize new query images
  - You can explain how you solved each step
  - You can explain potential issues related to your implementation of each step
  - You can propose alternative approaches to parts of the system
  - You have used own visualizations in the presentation
- Extras (**50%**):
  - You have implemented one or more of the suggested extras
  - You can explain the motivation for the extras you implemented
  - You have done qualitative and/or quantitative comparisons against the baseline

A non-exhaustive list of extras

50% of your project score is determined by implementing extra functionality. Below is a list of recommended ideas:

- Use SfM software to create a unified 3D model (see resource section below).
- Refine the estimated pose (and possibly intrinsics with distortion) by minimizing reprojection error between all inlier 2D-3D matches, using non-linear least squares.
- Localize query images taken by a camera with unknown intrinsics.
- Localize images of Gløshaugen taken from Google Images.
- Localize airphotos or old historical images of Gløshaugen (see e.g. Wikipedia).
- Compare different feature descriptors.
- Try out a deep learning-based feature descriptor, e.g. LIFT.
- Compare different PnP solvers in terms of the number of RANSAC iterations, overall speed and accuracy relative to reprojection error optimum.
- Use more than one feature descriptor per 3D point to improve robustness, e.g. localize during lighting conditions or times of the day.
- Don't use standard RANSAC.
- Assume the camera has a known height, pitch and roll and utilize this.

Recommended final presentation template

Your final presentation should demonstrate the evaluation criteria. A possible template or prioritization of topics is given below, but feel free to present stuff in a different order if it is more natural.

1. (ca. 2 minutes) Show that your system's core functionality works
   - Show the localized pose for a few query images along with the 3D model.
   - Show 2D-3D inlier matches for a query image.
2. (ca. 8-13 minutes) Describe how your system works including extras
   - Which places did you decide to support?
   - How did you acquire the 3D model and ensure its accuracy?
   - How did you match features?
   - How did you estimate the camera pose?
   - If you implemented any extras, describe how and why, and show a qualitative and/or quantitative comparison against the basic implementation.

Resources

General background

To learn about visual localization in general, see CVPR 2015 Tutorial on Large-Scale Visual Place Recognition and Image-Based Localization. Slide 1-30 covers the type of system you will implement in this project and provides references to useful software. The rest of the tutorial goes beyond what we expect you implement for the project, but gives you an idea of potential challenges.

3D model acquisition

The principles of structure and motion estimation is covered in lecture 8 and 10. A simple way to acquire a small 3D model is a two-view reconstruction approach as in exercise 6. First, establish some 2D-to-2D matches between two views, e.g. by tracking keypoints or by matching features. You can then estimate the relative camera motion using the algorithm from exercise 6. Given the motion, you can triangulate the 3D position of new point matches. Use this to triangulate features that you detect in one view and match against the other view, and store the 3D points along with the extracted image descriptor. You will have to repeat this process for each place of interest, creating a separate two-view reconstruction for each. The Matlab page for two-view reconstruction has some good tips, even if you're not using Matlab.

As the above approach is limited in accuracy and model size, one of the suggested extras is to acquire the 3D model using existing Structure-from-Motion (SfM) software. We recommend COLMAP, which is cross-platform and very easy to use. Make sure to read the tutorial. The relevant sections are those that involve sparse reconstruction (not dense reconstruction). NB! You don't need a GPU to use COLMAP for sparse reconstruction. Simply make sure to uncheck the **use_gpu** option in both feature extraction and feature matching. If you forget, and you don't have a GPU, COLMAP will simply crash.

Note that COLMAP uses its own feature descriptor, which is unlikely to be compatible with the features you will detect in the query images. However, you can import custom detected features. You should be able to convert most feature descriptors into the required format, although you may need to convert floating-point to integer. Alternatively, you can create a sparse reconstruction using the software's features, export the camera poses as a txt file (see also the format description), and triangulate your own features.

Feature matching

Feature matching was covered in lecture 3. The brute force approach is to compare each feature descriptor in the query image against each descriptor in the model, and pick the closest one. A faster alternative is approximate matching using a kd-tree. Additionally, you will want to reduce the number of false matches using Lowe's ratio test: accept a 2D-to-3D correspondence if the two nearest neighbors pass the ratio test with a sufficiently high threshold (e.g. 0.7).

Camera pose estimation

The first thing you can try here is to use the 6-point DLT algorithm, with or without known camera intrinsics, which was was derived in lecture 4. For the extra credit tasks, you will want to compare different PnP algorithms. Some references to various PnP algorithms are listed in the CVPR tutorial slides.

One of the extra credit tasks is to refine the pose obtained from PnP by minimizing reprojection error using non-linear least squares. Lecture 5 covered non-linear least squares optimization in general. To optimize a pose (which has six degrees of freedom) you should use a minimal parametrization of rotation. A simple solution is to represent the rotation "step" as a small Euler angle rotation (roll, pitch, yaw) around the current rotation matrix. After solving for one step, you update the current rotation matrix by multiplying by the Euler rotation matrix, and repeat the process for another small rotation around this new matrix. You can then use your Levenberg-Marquardt implementation from exercise 4, or use an existing non-linear least squares optimization library.

Camera calibration

You will have to calibrate your camera in order to acquire the 3D model. This can be done using a checkerboard and the Matlab or OpenCV calibration app. Alternatively, you can use COLMAP's autocalibration to estimate intrinsics, even if you're not planning to use it to acquire the 3D model. This doesn't require a known calibration target and can give a more accurate calibration. COLMAP also supports a wide range of camera models.

Software

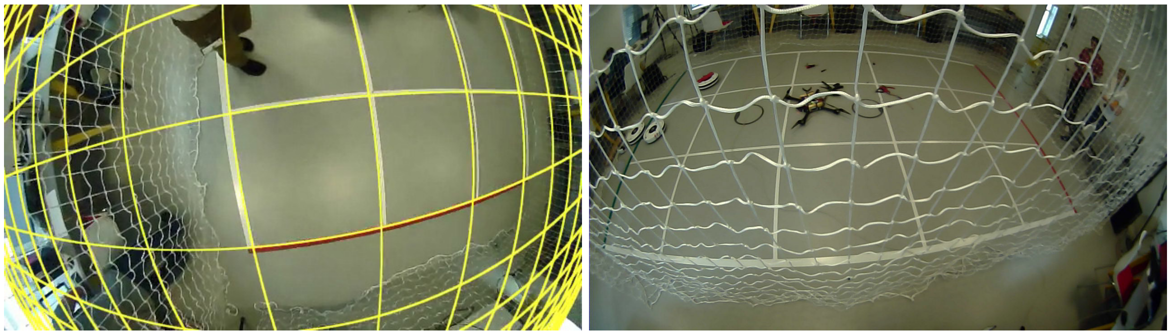
Besides basic image, vector and matrix processing, you are encouraged to use existing software for sub-problems, including

- Camera calibration
- Triangulation and motion from 2D-2D matches
- Feature detection, description and matching
- PnP solvers
- Non-linear least squares optimization
- Non-standard RANSAC

Below is a list of potentially useful software.

- Spatial Math Toolbox (Matlab) / scipy.spatial.transform (Python): Quaternion-to-matrix conversion and more
- OpenGV (C++, Python, Matlab): Triangulation and PnP.
- FLANN (C++, Python, Matlab): Fast feature descriptor matching.
- Optimization Toolbox (Matlab) / scipy.optimize (Python): Non-linear least squares.
- OpenCV (C++, Python): PnP, features, matching, and triangulation.
- Computer Vision Toolbox (Matlab): Camera calibration, features, and SfM functions

3 Grid detection


Samples from the grid detection dataset.

Goal

The goal of this project is to extend your line detection algorithm from exercise 5 to detect the grid pattern and estimate the pose of the camera relative to the grid lines.

Dataset

A dataset will be made available on Blackboard containing sequences of varying difficulty taken by a wide-angle fisheye lens camera. The dataset will also contain a description of the camera model and provide calibrated intrinsics. You can use other images for testing, but your final results should be from running your algorithms on the provided dataset.

Expected core functionality

You are expected to write a program that can take in an image from the dataset and output the pose of the camera relative to the grid lines. The rotational component should be output as yaw, pitch and roll angles, defined such that yaw is the camera's rotation about the gravity vector. The translational component should be output as x, y, z, defined such that $z > 0$ is the camera's vertical height above the floor, and x, y is its position along the grid lines. You may place the axes for x, y however you like. You should also output an image showing the estimated grid pattern on top of the original fisheye image, as in the figure, using the estimated camera pose and the known grid line spacing. Note that since the grid is not always fully visible, the pose cannot be determined absolutely; from a single 1×1 meter tile, the yaw can only be computed modulo $\pi/4$, and x, y can only be computed modulo 1 m.

There are many ways to approach this problem. A relatively simple one is to first undistort the input image using the algorithm from exercise 2. You can then use your algorithm from exercise 5 to detect lines. To estimate the camera pose, you can use the homography estimation method from exercise 3 inside a RANSAC loop, which could look like this:

1. Initially, compute the intersection between all pairs of lines.
2. Randomly select four intersection points, and associate them with the corners of a canonical 1x1 meter tile.
3. Estimate the homography from the resulting four point correspondences.
4. Compute the number of inliers to the grid pattern estimated by the homography.
5. Repeat (2)-(4) to find the homography with the most inliers.
6. Finally, decompose the homography into rotation and translation.

We suggest you get this approach working first, and then add improvements by implementing one or more of the suggested extras. You are also allowed to approach the problem in a completely different way.

Evaluation

Your project score is decided based on the criteria and weighting below.

- Core functionality (**30%**):
  - You can detect straight lines in the image
  - You can estimate the camera pose
  - You can draw an estimated grid pattern on top of the fisheye image
  - You can explain how various algorithm parameters affects the results
- Extras (**70%**):
  - You have implemented one or more of the suggested extras
  - You have done qualitative and/or quantitative comparisons against the baseline

Suggested final presentation template

Your final presentation should demonstrate the evaluation criteria. A possible template or prioritization of topics is given below.

1. (ca. 2 minutes) Show that your system's core functionality works
   - Show the estimated pose and grid pattern for a few sample images.
   - Show some intermediate outputs, e.g. analyze the quality of the homography estimate by generating a perspective-free view of the floor.
2. (ca. 8-13 minutes) Describe how your system works including extras
   - How did you detect lines?
   - How did you recover the pose?
   - How did you draw the grid pattern?
   - If you implemented any extras, describe how and why, and show a qualitative and/or quantitative comparison against the basic implementation.

A non-exhaustive list of extras

Below are some suggestions for improvements. You don't need to do all of them to get maximum score, and other creative improvements that are not on the list will be valued.

- Make a video of your system in action.
- Track the absolute camera pose and visualize the camera trajectory.
- Improve the edge detection accuracy in both position and orientation.
- Use a peak fitting algorithm when extracting lines from the Hough accumulator.
- Replace non-maximum suppression with finding clusters directly in the undiscretized $\theta\rho$-space.
- Use an alternative line detection algorithm, e.g. split-and-merge or edge-linking.
- Refine lines using supporting points and total least squares (see Szeliski exercise 4.12).
- Detect the centerlines of the grid lines rather than the sides.
- Avoid undistorting the entire fisheye image. Instead, detect edges directly in the fisheye image and selectively undistort only these points. The simplest approach here is to undistort points and use the original Hough transform. If you want to use the oriented Hough transform, as in exercise 5, you also need to undistort the gradient directions, which will involve some chain rule differentiation.
- Use SIMD or GPU acceleration.

Resources

You are encouraged to use existing software for any part of this project. The following can be particularly useful:

- scikit-image (Python)
- Image Processing Toolbox (Matlab)
- OpenCV (Python, C++)

Besides the chapters referred to in the lecture on image processing, and in exercise 5, the following chapters can be useful:

- Corke App. J: Peak finding
- Szeliski Ch. 5.3: Mean shift and mode finding
- Szeliski App. A.2.1: Total least squares

$\boxed{4}$ Hyperspectral imaging (written report)

In the hyperspectral imaging project you will learn about hyperspectral imaging through a detailed assignment, written by Sivert Bakken and Joseph Garrett. They will give a guest lecture in week 9 (Feb. 24th), providing a suitable introduction. The assignment and data sets will be made available on Blackboard.

NB! There is no final presentation. Instead, you will hand in a written report containing your answers to the assignment questions. The deadline for submitting the report is the same (Sunday, April 26th).

Evaluation

In this project you will receive a score based on your answers to the theory questions in the assignment. Although you are allowed to collaborate with other students, you are expected to have done all tasks individually and to hand in individually written reports (no plagiarisation).

You should still register on the registration form. Write only your own name and the project number (4). Do not submit your report as a group.