Vision Aided Navigation 2022 - Exercise 5

Prefix:

In exercises **3** and **4** we created feature tracking for all the image pairs, removed the outliers and produced an initial estimate for the trajectory of the vehicle. The estimate used the matches between consecutive images using a deterministic paradigm.

In this exercise we run bundle adjustment to leverage the information across multiple images as well as taking a probabilistic approach to the problem. The results of the previous exercises will be used as initial starting point for the bundle optimization.

To keep the computation cost down we will run the optimization on small segments of consecutive images. We use the GTSAM optimization library to handle the mathematical details of the process.

The result of the computation is an accurate estimation of the local trajectory of the vehicle and can be used as **Visual Odometry**.

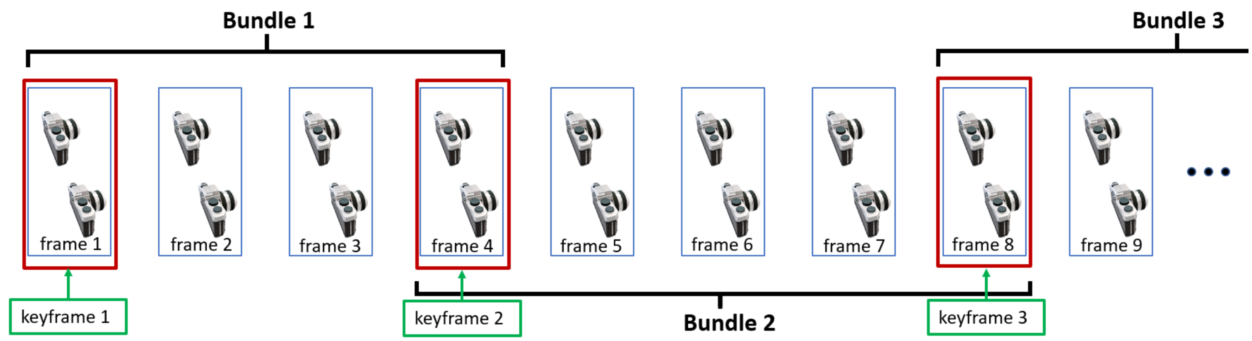
5.1 Using the tracking database pick a random track of length ≥ 10.
For all the frames participating in this track, define a *gtsam.StereoCamera* using the global camera matrices calculated in exercise **3** (PnP).

- For extrinsic camera matrix $[R|t]$, what would be the transformation from the camera's coordinate system to the global coordinate system? (Note the opposite direction here)

Using methods in *StereoCamera*, triangulate a 3d point in global coordinates from the last frame of the track and project this point to all the frames of the track (both left and right cameras).

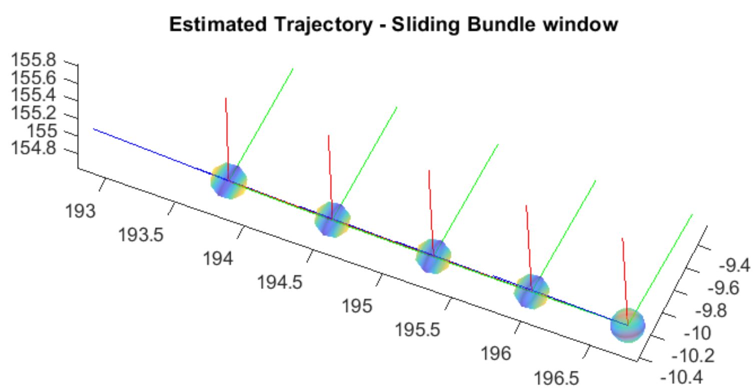
- Present a graph of the reprojection error size (L_2 norm) over the track's images.
- Create a factor for each frame projection and present a graph of the factor error over the track's frames.
  What is the factor error as a function of the reprojection error?

5.2
Bundle Adjustment window
We perform local Bundle Adjustment on a small window consisting of consecutive frames.
Each bundle 'window' starts and ends in special frames we call **keyframes.**
Keyframes should be chosen such that the number of frames in the window is small (5-20 frames) with some meaningful movement between them. Decide on a criterion to mark specific frames as keyframes - the decision can use the estimated distance travelled, elapsed time, features tracked or any other relevant (and available) criterion.



We use the tracking from the previous exercises as constraints for the optimization - the tracking is used to construct factors (reprojection measurement constraints) between the frames and the tracked landmarks. As an initialization for the optimization, we use the relative locations calculated by the PnP and triangulated 3d landmarks locations.
In defining the Bundle optimization take care to avoid creating an ill-formed problem.
The first Bundle window consists of the first two keyframes with all the frames between them, with all the relevant tracking data.

- Print the total factor graph error before and after the optimization.
- Plot the resulting positions of the first bundle both as a 3D graph (for example using *gtsam.utils.plot_trajectory*) and as a view-from-above (2d) of the scene, with all cameras and points.



5.3

Choose all the keyframes along the trajectory and solve all resulting Bundle windows.
Extract the relative pose between each keyframe and its predecessor (location + angles).
Calculate the absolute pose of the keyframes in global (camera 0) coordinate system.

- Present a view from above (2d) of the scene, with all keyframes (left camera only, no need to present the right camera of the frame) and 3D points.
- Overlay the estimated keyframes with the Ground Truth poses of the keyframes.
- Present the keyframe localization error in meters (location difference only - Euclidean distance) over time.

GTSAM

We use GTSAM - Georgia Tech Smoothing and Mapping Library for factor graph optimization. The library is implemented n C++, use '`pip install gtsam`' to use the python wrapper.

- The camera pose (transformation) can be represented as `gtsam.Pose3` object.
  This class represents 3D poses and transformations, allows access to the rotation and translation parts of the transformation and more.
  `gtsam.Pose3` object can be constructed from a pair of `gtsam.Rot3` (rotation) and a `gtsam.Point3` (translation)
  The transformation represented by `Pose3(R,t)` is $Rx + t$ **from pose coordinates to world coordinates** (opposite from what we are used to)

- The initial poses are stored in a *gtsam.Values* object.
  Each variable (node) in the optimization is identified with a unique key. Key-value pairs are stored in *gtsam.Values*, where a key is associated with a specific value:
  ```
  c1 = symbol('c',1)
  q1 = symbol('q',1)
  pose_c1 = gtsam.Pose3()
  loc_q1 = gtsam.Point3(0, 0, 0)
  initialEstimate = gtsam.Values()
  initialEstimate.insert(c1, pose_c1)
  initialEstimate.insert(l1, loc_q1)
  ```

  A particular value can be retrieved using `initialEstimate.atPose3(key) / initialEstimate.atPoint3(key)`.

- `gtsam.utils.plot_3d_points` / `plot_trajectory` can be used to display the list of poses. The poses are passed as *gtsam.Values*.
  `gtsam.utils.set_axes_equal(1)` turns on equal axis scales.

- `graph = gtsam.NonlinearFactorGraph()` creates a general factor graph.
  Factors can be added using `graph.add(factor)`.

- for a KITTI stereo frame with intrinsic matrix $K = \begin{bmatrix} f_x & skew & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$ and the right camera extrinsic matrix (relative to the left camera) $[I|t]$ with $t = (baseline \quad 0 \quad 0)^T$
  a stereo camera frame can be set using:
  `K = gtsam.Cal3_S2Stereo(`f_x`, `f_y`, skew, `c_x`, `c_y`, -baseline)`

- There are many possible kinds of factors already implemented. For example:
  `gtsam.PriorFactorPose3(key, pose, uncertainty)`
  represents a measurement of the location of a camera.

```
gtsam.GenericStereoFactor3D(StereoPoint2(xLeft, xRight, y),
                            uncertainty, poseKey, landmarkKey, K)
```
represents a projection measurement to a stereo camera pair (frame).
`factor.error(values)` returns the factor error for the specific values in '`values`'.

- Uncertainty is modeled using covariance matrices. There are various ways to define one, for example:

  `gtsam.noiseModel.Diagonal.Sigmas(np.array([a, b, c]))` $\rightarrow \begin{bmatrix} a^2 & & \\ & b^2 & \\ & & c^2 \end{bmatrix}$

  `gtsam.noiseModel.Isotropic.Sigma(2, 1.0)` $\rightarrow \begin{bmatrix} 1 & \\ & 1 \end{bmatrix}$

  The whole covariance matrix can be set using
  `gtsam.noiseModel.Gaussian.Covariance(S)`

- Many optimizers are implemented, for example Levenberg-Marquardt optimizer:
  `optimizer = gtsam.LevenbergMarquardtOptimizer(graph, initialEstimate)`
  where initialEstimate is a *gtsam.Values* list.

- An optimizer can be run using `result = optimizer.optimize()`
  where '*result'* is a *gtsam.Values* list.
  `optimizer.error()` returns the current error of the factor graph.

- `frame = gtsam.StereoCamera(Pose3 leftCamPose, Cal3_S2Stereo K)` represents a stereo frame.
  `stereoPoint2 = frame.project(Point3)` projects a 3d point to both cameras and
  `point3 = frame.backproject(StereoPoint2)` triangulates a 3d point from a match between the left and right cameras.