VAN - Final Project

Matan Aivas Ilai Granot

July 2022

1 Project Introduction and Overview

The goal of the course (and this project that summs it), was to implement the SLAM (Simultaneous localization and mapping) algorithm. In this algorithm the target is, given the output sequence of any sensor or group of sensors of an object in motion, to simultaneously perform a mapping of the environment of that object, and localization, i.e. the location of the object in the generated mapping. While performing any one of the two tasks is relatively easy performing both mapping and localization together - is a difficult problem.
In this course we implemented the SLAM algorithm for visual sensors (cameras) on a car, and specifically stereo cameras. That is, two cameras that have fixed locations with respect to each other and the car. We used the images taken by those cameras throughout the drive and used them as an input to our algorithm. For each point in "time" (the images were taken at 10 Hz) we have as input to the algorithm one pair of images (hereafter - a Stereo-image).
In an high-level overview, the algorithm consists of several steps:

- Stage 1: PnP
  Creating a primary estimation of the transformations between frames, as well as point-cloud mapping of the surroundings.
- Stage 2: Bundle adjustment
  Using the previous estimation of the frames transformations and point clouds as input for an algorithm which constructs probabilistic graphical models for short sequences of the estimated transformations and tracks. It then optimizes the estimation of every short sequence separately. In these graphs, every node is a transformation (pose) or a 3D point. The factors are projections of 3d points to the frames.
- Stage 3: Pose graph construction
  Using the previous optimised bundles (factor graphs) to construct a pose graph, in which every node is the first (and last) pose in each bundle. Thus reducing significantly the dimensionality of the problem we want to optimize.

- Stage 4 : Loop closures
  Iterating through the pose graph and finding poses that are likely to be in close proximity to one another. When such a pair is found, perform features extraction and matching between the corresponding frames to make sure it's a match, and if so - add it as a new factor (constraint) to the pose graph, and re-optimize it.

Such a tool is useful in many applications and usages in modern life - autonomous parking of cars, robots navigating in a warehouse etc, and thus plays a crucial role in modern life.
In the following section, we'll dive into each of the different stages.

2 Deep dive

2.1 Stage 1

The PnP stage itself is composed of several parts.
The first is Stereo Frame processing. This entire stage is encapsulated in our StereoFrame class which is in 'code/FramesDir'. The object has a main method called "process_stereo" which is called upon initialization. For each stereo frame we get, we are looking for keypoints in each of the two frames its composed of. A Keypoint is a distinctive point in the image which is invariant to rotation, scale and distortion, thus enabling us to track and match it across different images (in reality those characteristics don't fully hold, but hold enough for the process to work). We use cv2 library to process each of the images to receive a a list of keypoints and a list of corresponding descriptors for that image (this is done in "__detect_kpts").
Given the descriptors of the two images we match the points between the left and right images, and remove outliers based on stereo pattern and cross-checking("__match"). Finally using the cameras matrices, we perform triangulation - to get a 3D point per each matched keypoint (the 3D point is in the coordinates system of the left camera.)("__triangulate").
The next step in the pnp - is multiframe processing. This is encapsulated in our MultiFrame class which is in 'code/FramesDir'. This object receives two consecutive StereoFrame objects, and matches between the keypoints of the left image of the first StereoFrame, to those of the second StereoFrame (done in "match" method of the class). Following is an approximation of the movement between the two frames - which is done in the method "ransac" which iteratively calls "solve_pnp" - which samples 4 points that were matched, and computes the relative transformation between the two frames - induced by them. The "ransac" method also updates the supporters of each transformation computed (method "find_supporters"), in order to choose the best transformation - and later refine it, using all of the inliers (method "refine_tranformation"). Repeating this process over the stereo frames along the drive, we get the sequence of transformations that represent the trajectory of the vehicle travel, which will

be the input of the next and second stage of the SLAM algorithm, the bundle adjustment.

2.2 Stage 2

For this stage we create tracks database which holds data on 3D points that were matched across different frames. This is done using "TrackDb" class which is in 'code/TracksDir'. The object has a method "update" which receives a multi-frame object, and updates the tracks accordingly (this class also uses the object 'Track' which is defined in the same directory). This is used in 'Drive' which is a class in 'code/FramesDir', and encapsulates the pnp-phase processing of the entire drive, under its method 'run'.

Due to the discrete nature of the pixels, and other factors - our measurements are noisy: the points angles relative to the camera, the distance of the points in the camera coordinate system, the accuracy of the feature extraction algorithms and more. Furthermore, we are bound to have some "conflicts" in our data - a 3D point that was "created" in a certain frame, will have a somewhat different location in the next frame. We are interested in modelling this uncertainty and weighting it in our calculations in order to adjust the relative poses of the cameras in a way that will best match our measurements - i.e in such a way that a measurement about which there is less uncertainty will have a greater effect on the end result than measurements about which the uncertainty is greater. We model this uncertainty using a graphical model called Factor Graph, in which the nodes are the camera positions and points in the world, and the edges are the projection of 3d point to the camera.

The noise assumed in the measurements is gaussian noise. After constructing the factor graph, we use the Levenberg-Marquardt algorithm to optimise the model's parameters. In our code, the construction of the factor graph by adding: the camera nodes, the projection constraints, the noise and the optimisation, is implemented in the 'BundleGraph' class, which is located in the 'GraphOptimization/BundleDir' directory. The main logic is done under the 'optimize' method.
Since the process involves calculating marginal distributions of a joint distribution, and the number of factors in the common distribution is the sum of the number of cameras and the number of measurements (projections), the calculation is too heavy to perform on the entire sequence at once. We therefore divide the trajectory into short sections, for each of which we construct a factor graph as described above, and by the end of the process we have bundles of factor graphs each of which is optimised individually by the Levenberg-Marquardt algorithm. In our code, the process of splitting the frames sequences by key frames and iterating over the short sections to construct factor graphs is implemented in the 'BundlesManager' class method 'process_drive', which is located in the 'GraphOptimization/BundleDir' directory.

3 Stage 3

At this point, we have a bundle of optimised factor graphs, each for some short section of the trajectory. Now, we construct a graph for the whole trajectory. We call that graph a Pose Graph, and its nodes are camera positions (poses) of the keyframes only, and the edges are relative transformations between the poses.

From each of the optimised factor graphs we extract the relative transformation from the first camera position of the graph to the last. The poses in the pose graph represent the first and last cameras of every graph in the bundle. Note that since now we have a graph with significantly less factors, (much fewer nodes, and even less edges since we discarded the projections and keep only the transformations), we can calculate the marginal distributions for all of the trajectory and perform optimization. At this point, we still did not improve the previous estimation (of the bundle) since we did not add any new constraint or new measurement to the model, the pose graph. In our code, the generation of the pose graph using the bundles from the previous stage is implemented in the 'PoseGraph' class, which is located in the 'GraphOptimization/PoseDir' directory, and this entire stage is handled in the init of the class.

4 Stage 4

Now, we have a pose graph modelling the whole trajectory, but since in its construction we introduced noise (through the noisy measurements it was constructed from), and since this noise is accumulated throughout the trajectory, at the end of the trajectory we already have significant uncertainty about the poses configuration, and large drifts.

We want to add new constraints and have fresh measurements to fix the drift caused by the accumulated noise. We do so by finding loops in the trajectory. Suppose we found two poses that are from "far" sections of the trajectory and share a decent amount of key points ("far" in the sense that they were sampled at different times). Then, we could perform feature extraction and matching for those two frames, triangulations and PnP, and calculate their relative transformation, adding a new edge (which is a fresh measurement and constraint) to the pose graph. The problem is, though, that there are a lot of possibilities for loop closure, and performing those operations is very expensive in terms of computation. So, we need a computationally light method of filtering candidates for loop closure. For that purpose, we iterate over the poses (the pose graph nodes), and for every pose in position i, we iterate over the poses in positions $j < i$ and find the shortest path between pose j and pose i (in the straightforward sense - minimal number of edges). Then, for that shortest path, we calculate the relative covariance by summing up the relative covariances along the path (as we introduced gaussian noise to the model - this is a good approximation of the uncertainty) and use the mahalanobis distance as measure of distance between the poses. Only to the poses that cross some threshold of that

distance measure, we then perform the expensive consensus match, and only if the consensus match succeeds, we calculate the relative transformation between the poses, add the edge (fresh constraint) to the pose graph, and optimise using Levenberg-Marquardt algorithm.
Finally, after looping through all of the poses, looking for loop-closures and optimizing accordingly, the pose graph at this point is the output of the SLAM algorithm. In our code, the iteration over the pose to find loop candidates, calculating relative covariances, and computing the mahalanobis distance is implemented in the method 'close_loops' of the 'PoseGraph' class, which is located in the 'GraphOptimization/PoseDir' directory.

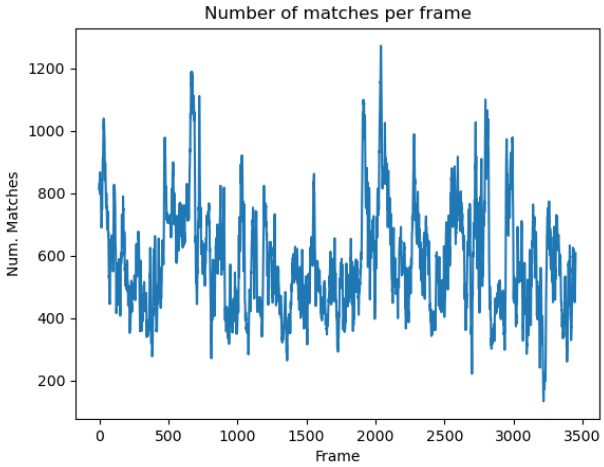
5 Performance Analysis

5.1 General statistics

- Total number of tracks - 319245
- Number of frames - 3450
- Mean track length - 4.598
- Max track length - 141
- Mean number of frame links - 425.5

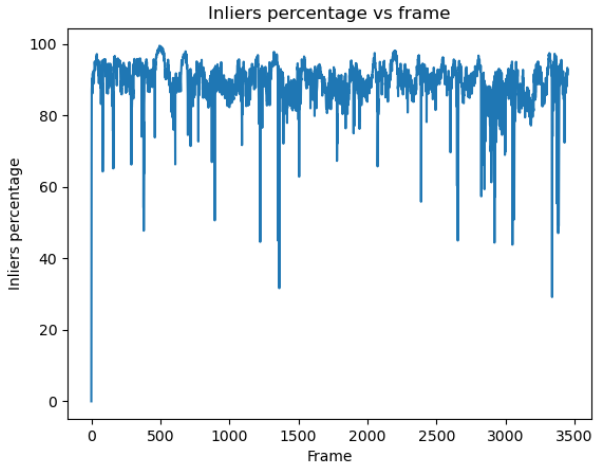
5.2 Number of matches per frame

A graph of the number of matches per frame is important for the tracking analysis of the algorithm. The number of matches constitutes an upper bound for the connectivity of the frames in the algorithm, and if there are few matches for the frames as a whole or even in certain frames, we will know that we need to improve our feature extraction and matching.
When we reached exercise 5, we had frames in which we did not have enough constraints, causing failures of computations (inverse of matrices by gtsam), this led us to go back to our detection choices, where we replaced to "Akaze". This increased significantly the number of matches we had in each frame. In the following graph we can see that the average number of matches per frame is around 600. There are some frames where we have a relatively low number of matches (we'll address this in our discussion and improvements part).



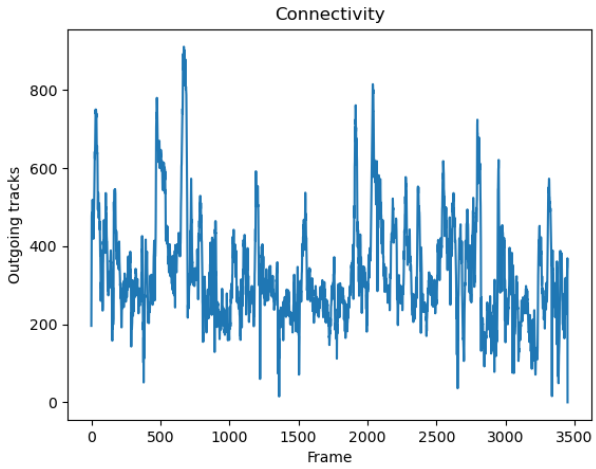
5.3 Percentage of inliers per frame

A graph of the percentage of inliers per frame - i.e. for each frame, what share of the 3D points triangulated in the previous frame, correspond well to the calculated transformation (meaning that their projection is close to the detection). As in the previous figure, the overall image looks good - around 90% inliers in normal case, but with exceptions for certain frames. This figure has also improved dramatically after our move to the "Akaze" detector.



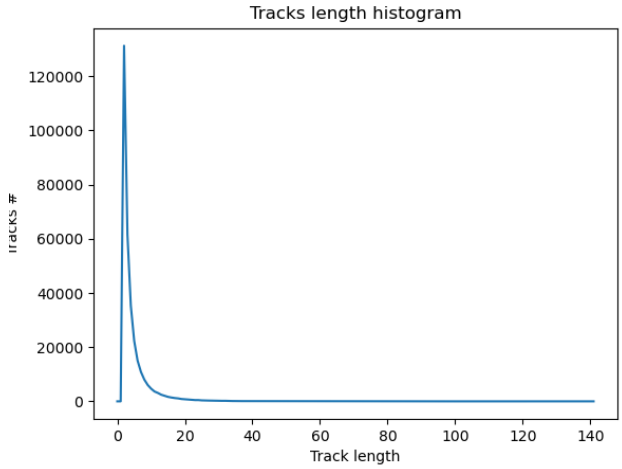
5.4 Connectivity

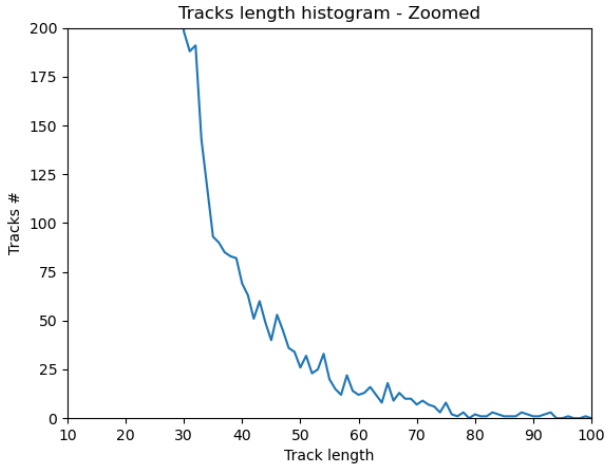
This graph displays the connectivity over the drive - i.e. for each frame, the number of tracks outgoing to the next frame (the number of tracks on the frame with links also in the next frame). Connectivity graph helps to identify frames with weak connectivity, in which we do not have enough key points that we have identified in adjacent frames. In these frames the evaluation of our transformations relies on fewer points and may be less accurate. We can see our overall connectivity is pretty good, with an average of over 300 links per frame. The change of detector mentioned above increased this number from about 100 links per frame (we believe it's mainly due to higher number of candidates to be matched). But there are some specific frames with rather low connectivity (we'll address this in our discussion and improvements part).

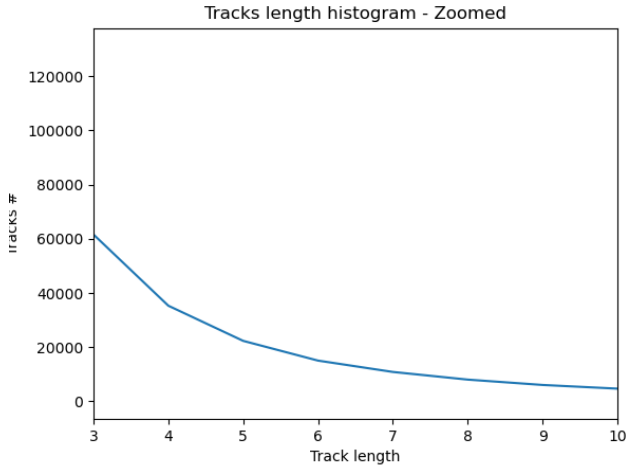


5.5 Track length histogram

The length of the tracks may provide information on the quality of our tracking. Short tracks (e.g. length 2 or 3) are considered less reliable than long tracks (e.g. length 10) because they are less resistant to outliers.


Tracks length histogram
Tracks #
Track length
0
20000
40000
60000
80000
100000
120000
0
20
40
60
80
100
120
140

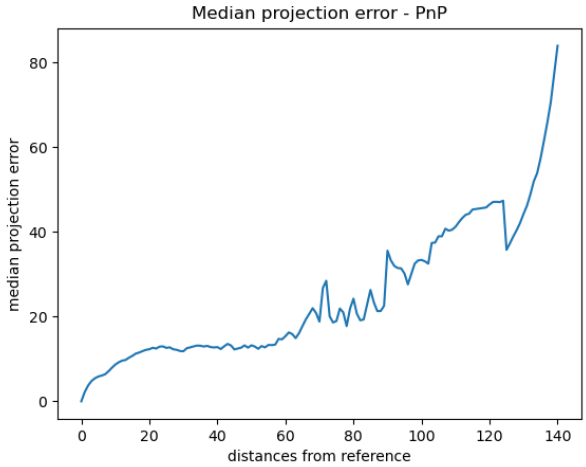

Tracks length histogram - Zoomed
Tracks #
Track length
0
25
50
75
100
125
150
175
200
10
20
30
40
50
60
70
80
90
100



From the zoomed plots we see the that although the majority of the tracks is of length 2, as might be expected, there are over over 20,000 with length over 5, and over 10,000 with length larger than 10. This means that the tracking works well.

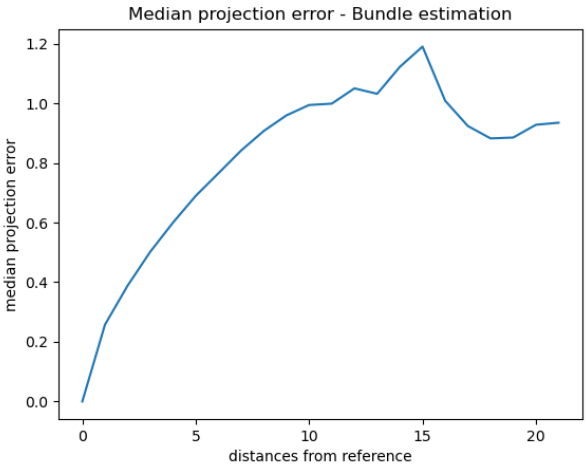
5.6 Projection error of tracks - PnP

The following graph displays the Median projection error of the different track links as a function of distance from the reference frame - using the PnP estimation. Since this is a costly computation we have randomly sampled 100 frames from the drive, and then calculated the median for every track that appeared in those frames.
The plot acts as a sanity both for the way we build tracks, and to our estimation of the transformation using the pnp. As expected the projection distance grows as we move further away from the frame the 3D point was triangulated from. This is because a triangulation from a further perspective would be less accurate, and projection to a smaller perspective is more sensitive to localization error.
We further see that the tracks are being built correctly But, it's also clear that the pnp is fine in the small scope of a few frames, but diverges quickly after that.



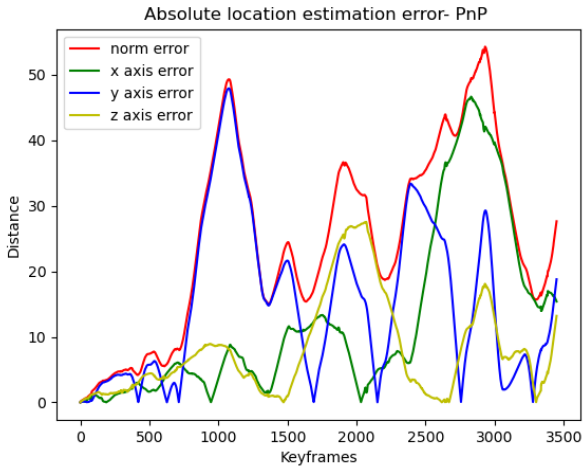
5.7 Projection error of tracks - Bundles

The following is the same as the previous graphs, but using the bundles (as opposed to just the PnP).
Compared to the previous graph the improvement and "charms" of the bundle graph is explicit - the median projection error decreases dramatically - to about a pixel error for ten frames gap, compared with about 10 pixels error for the PnP with the same delta of frames.



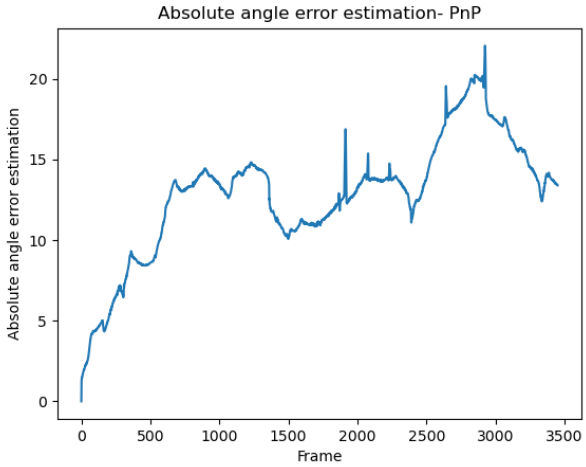
5.8 Absolute PnP - Location error

The graph that displays the absolute location error value - allows identifying trends of the error in the various axes (for example, lower error on the y axis, which can clarify things about the algorithm itself), but also tells us in which frames the errors are greater and allows us to cross check these frames with other graphs that provide information about the algorithm operation by frames, e.g. if we see high absolute error in a certain area, and in this area in the connectivity graph we see a particularly weak connectivity, we know that the error can be reduced by improving the feature extraction, matching and tracking.
This graph also enables us to compare different PnP vesrions - and to asses their overall performance, according to our preferences.
In this graph we can clearly see our drifts (as we saw in the top-view in exercise 4) - which amounts to 50 meters at some points. We can also see that the drifts are mainly in two axes - the longitutinal and lateral axes - as might be expected.



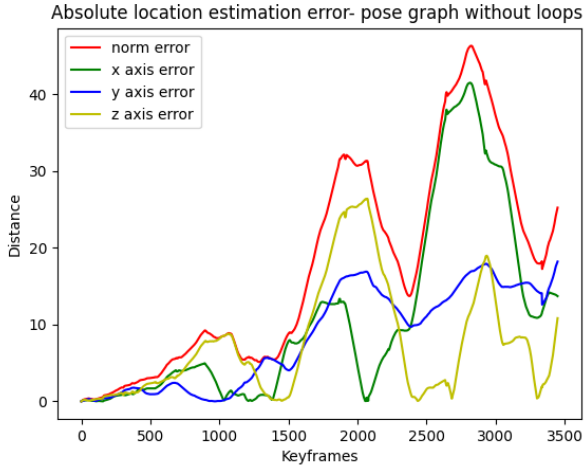
5.9 Absolute PnP - Angle error

The graph displays the absolute angle error value. Can be used in similar ways to the previous graph. Here as well we can see the drifts expressed well by the large error in angle - that at some point reaches 20 degrees.



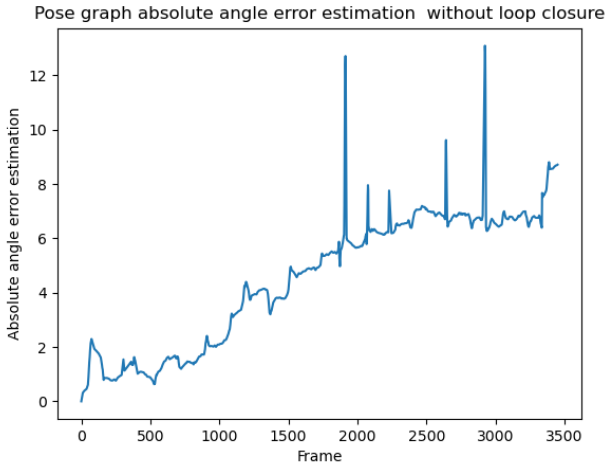
5.10 Absolute Pose Graph - Location error - No loop closures

This graph, compared to the former - shows the improvement introduced by the bundles graphs - the Location error decreased across the drive.



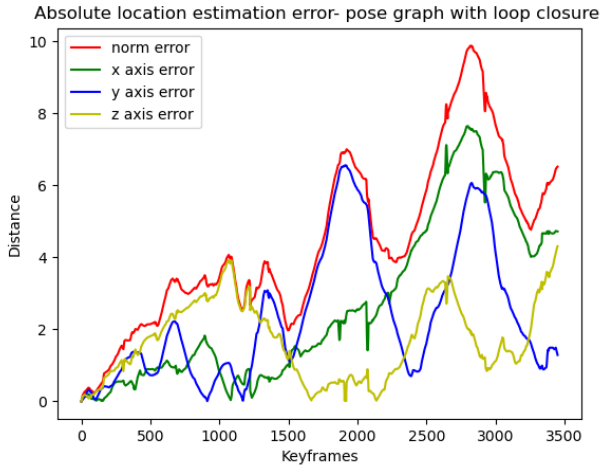
5.11 Absolute Pose Graph - Angle error - No loop closures

This graph, compared to the former - shows the improvement introduced by the bundles graphs - the angle error decreased dramatically across all of the drive.



5.12 Absolute Pose Graph - Location error - with loop closures

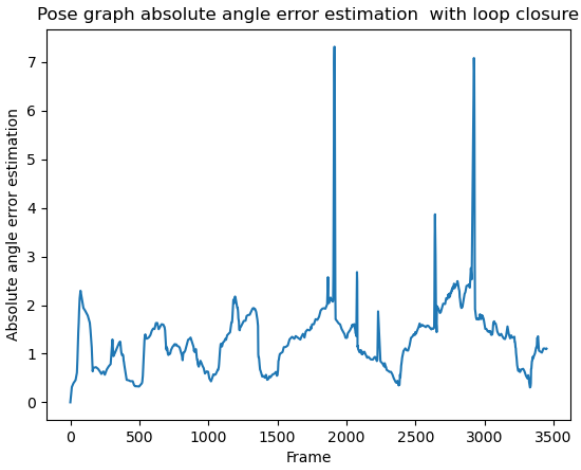
This graph also presents the benefit of the bundles graphs + the pose graph - with smaller localization error across the drive - decreased from deviation of 50 meters at worst, to 10 meters drift in the worst case.



5.13 Absolute Pose Graph - angle error - with loop closures

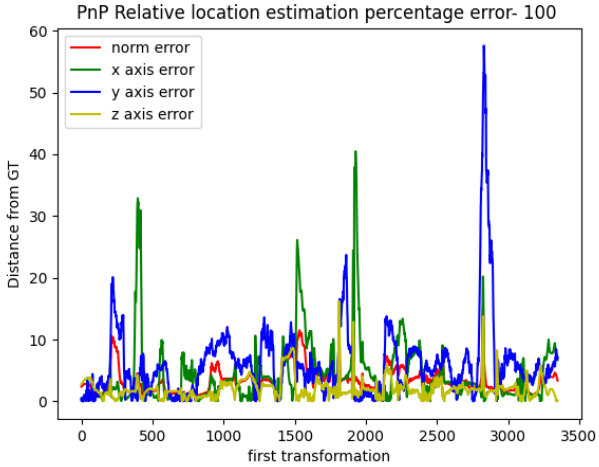
We can see that the angle error further decreased after the loop closures and the optimization. Apart from two points of a larger drift (which can point to

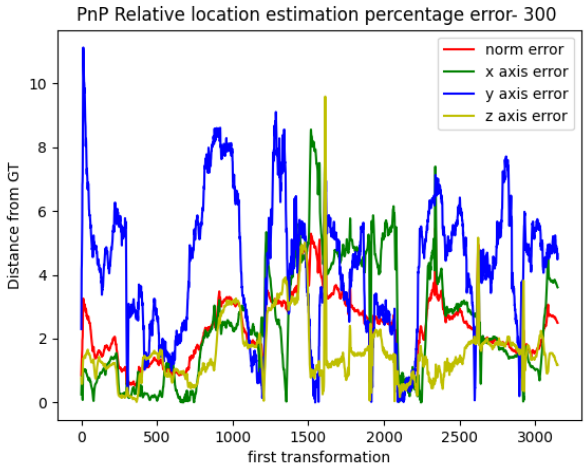
some earlier problem in those frames, such as rouge points that were added to some tracks) - the typical error of angle is less than 2 degrees.

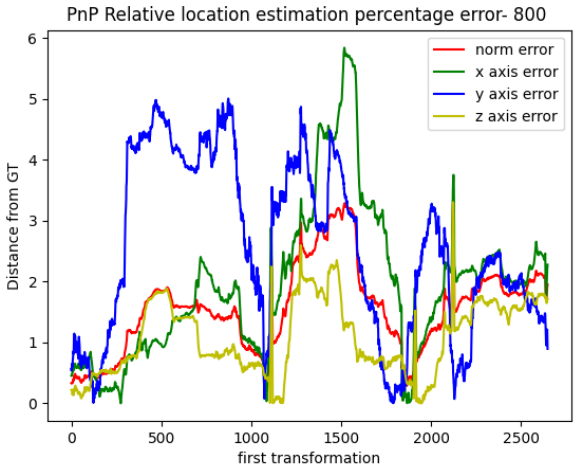


5.14 Relative PnP estimation error Location in %

Here we can observe two interesting phenomenon. the first, is that the error is usually the highest in the y axis, and this is counter intuitive for that in absolute terms the error in the y axis is lower. but here we are measuring the estimation error in percentage of the ground truth total distance, and because the y axis actual range is much smaller, the errors due to noisy measurements and discrete pixels influences more in this graph. The second is that the scale of errors decreasing as the distance of relative estimation increases, and this is also due to the error estimation in percentage, since we divide the errors in the sum of more sequential relative transformations distances.

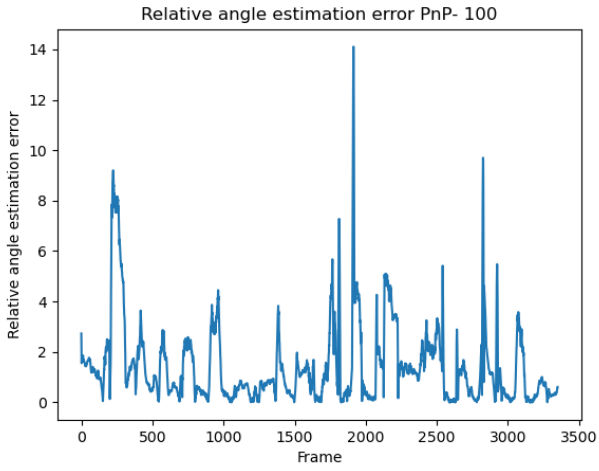

PnP Relative location estimation percentage error- 100
norm error
x axis error
y axis error
z axis error
Distance from GT
first transformation
0
10
20
30
40
50
60
0
500
1000
1500
2000
2500
3000
3500

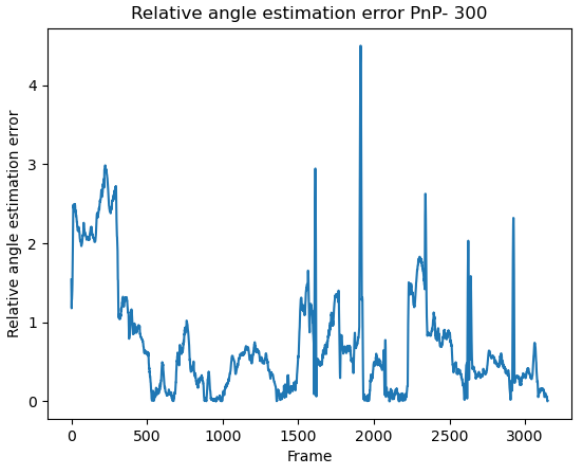

PnP Relative location estimation percentage error- 300
norm error
x axis error
y axis error
z axis error
Distance from GT
first transformation
0
2
4
6
8
10
0
500
1000
1500
2000
2500
3000

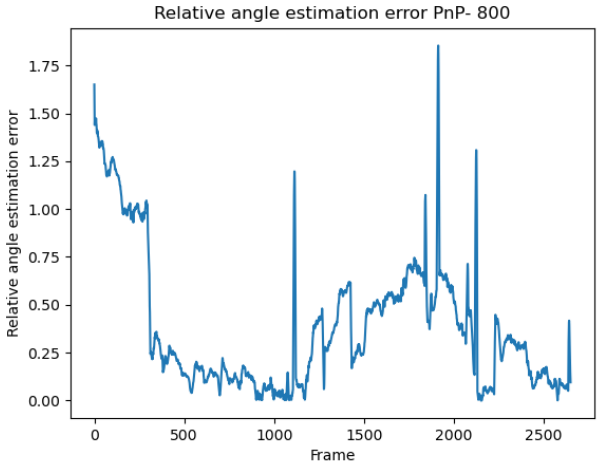


5.15 Relative PnP estimation error Angle in degrees

Here we can observe the same phenomena as above, as the scale of error decreasing as the distance of relative estimation increases, and this is also due to the error estimation in percentage, like the graphs above.

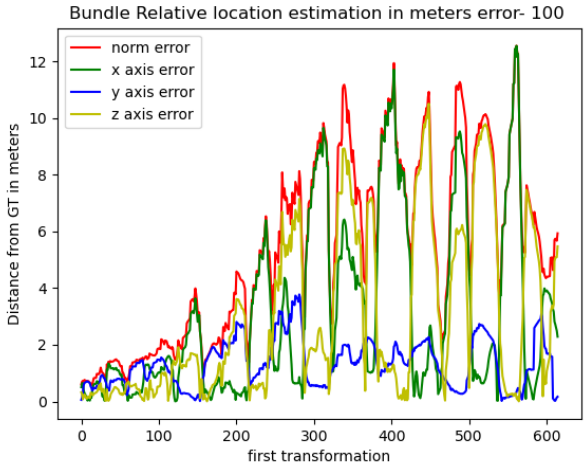




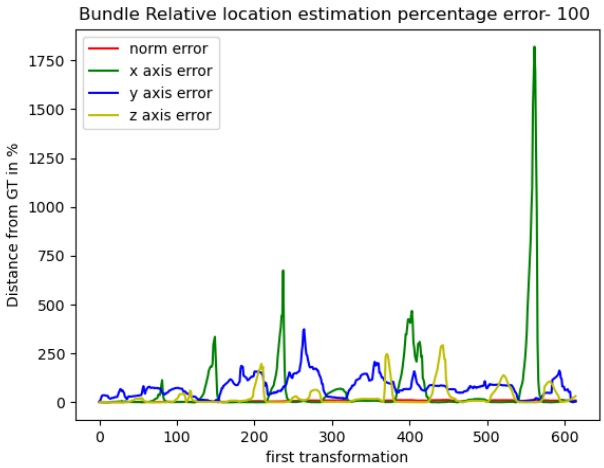


5.16 Relative Bundle estimation error

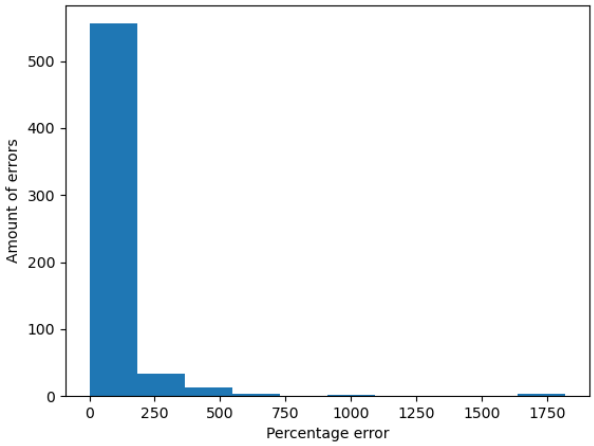
In these graphs we encountered a lot of difficulties. It turns out that during the many hours we worked on refactoring and improving the project, during the transition to Factor Graph we introduced anomalies, apparently damaged tracks, into the system, and which we did not have enough time to fix before the project submission due date.This is reflected in the relative error graphs of the bundle.
Here we can see the relative location error in meters. Though it is quite bouncy, we can see that the max error is 12 meters.



Here we can observe the graph of relative error in the bundle estimation in percentage. The percentage error is very high as the region that the bundle error is 12 meters, the ground truth total movement was less then half a meter, and so the percentage of that error is very high indeed.



To try to confirm that it is only a few damaged tracks causing this errors we generated a histogram of the errors in percentage, and it turns out that only very few such high errors exists, which strengthens the argument that it is a single number of tracks.



5.17 Number of matches and inliers per successful loop closure frame

The number of matches per successful loop closure can help us fine tune the threshold which above it we decide to close a loop. And from the relation between the number of matches in different loop closures we can try to detect false positives of loop closures.

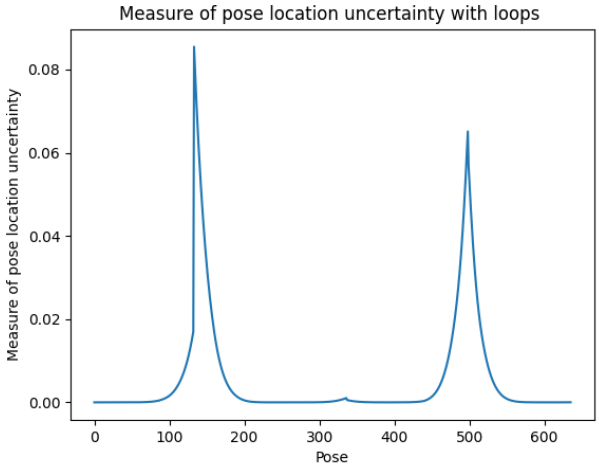
For frames - 1517-74, (matches) in first frame 543, in second frame 447.
The percentage of inliers: 73.66
For frames - 2388-337, (matches) in first frame 646, in second frame 421.
The percentage of inliers: 71.36
For frames - 3238-2305, (matches) in first frame 556, in second frame 766.
The percentage of inliers: 60.79
For frames - 3238-2291, (matches) in first frame 556, in second frame 723.
The percentage of inliers: 65.64
For frames - 3321-2382, (matches) in first frame 618, in second frame 617.
The percentage of inliers: 68.93
For frames - 3365-2413, (matches) in first frame 390, in second frame 349.
The percentage of inliers: 80.25
For frames - 3385-386, (matches) in first frame 326, in second frame 321.
The percentage of inliers: 64.11

5.18 Uncertainty Location size vs keyframe – pose graph without loop closure

Graphs that quantify the uncertainty of the algorithm regarding the position and angle of the poses allows us to better analyse the prediction of the algorithm

in different frames. For the uncertainty in the graph without loop closures it is expected that the uncertainty will increase along with the frames, as the uncertainty accumulates and no further crossovers of information about the poses in the frames are added after the algorithm has passed them. In the graph with the loop closures we expect to see much less uncertainty, as there is a cross-checking of information about the poses from different frames (loop closures), so we expect the uncertainty to grow between loop closures, but decrease as it gets closer to a some closure.

5.19 Uncertainty Location size vs keyframe – pose graph with loop closure



6 Discussion, Conclusions and improvements

6.1 Identifying weak points

Very high sensitivity to outliers- A point that has been triangulated and projected on successive frames in order to calculate the position of the cameras using the PnP algorithm, can greatly damage the accuracy of our model if it is an outlier resulting from an error in our feature detection and matching algorithm. Furthermore, the probabilistic models that use the PnP algorithm's output as input assume practically zero probability for the real state, for example the real positions of the cameras and projections, given such outliers, and this is because of the Gaussian noise assumption that will be detailed later.
Lower accuracy in frames with low connectivity- We observe a trend of increasing errors in successive frames where there is low connectivity, that is, a low number of tracks passing through them. This creates a conflict between our need to produce a large amount of tracks in order to improve accuracy and our

desire to avoid outliers in order to avoid severely damaging the evaluation of the model

6.2 Unrealistic assumptions

Gaussian noise assumption- The assumption is that given the projection of a point detected by the feature extraction and matching algorithms, the distribution of its true location will be some Gaussian around its detected coordinate, so that its probability of being in any location decreases exponentially in relation to the distance of that location from that coordinate. This assumption does not take into account the possibility that the feature extraction and matching algorithm may be wrong in such a way that the probability of the model placing the point in the correct place will be practically zero, and further propagating this error to the camera poses later in the process. This assumption can be interpreted as assuming that the feature extraction and matching algorithm is never wrong, since if it really was never wrong, the Gaussian noise assumption would hold.

6.3 Aspects that could be improved & suggestions

- As we mentioned in our analysis, it seems that our outlier rejection process wasn't good enough, and was probably harmed in our refactoring of the project. This did not affect the final result by much - as we saw, the trajectory after the loop-closures is a good approximation of the Ground-Truth, but fixing it will probably improve the results even more.
-
- Dynamic use of the classical algorithms for feature extraction and matching
  Different classic algorithms for feature extraction and matching may show different performances in different types of patches and backgrounds, and it may be possible to identify in real time that a certain detector is not performing optimally in a certain sequence of frames and to try other algorithms in that sequence, we carried out this experiment ourselves as part of this project, and we discuss it in detail below.

- Incorporating models from deep learning in the process of feature extraction, matching and outliers filtering
  The use of classical algorithms for feature extraction and matching does prove itself for most cases, but it uses small patches in the frame and relatively simple calculations, so it has difficulty matching points in areas of the film that have a continuous pattern, such as a fence or a sidewalk. The use of small patches of the frame does not allow the algorithm to detect such a point in relation to another object that is outside its patch, which

creates extensive and continuous parts of the film in which no reference points can be detected, and triangulating and projecting a point to an area where there are not many other projections can improve the accuracy of the PnP algorithm, creating more distinct lines to as the input for the mean square routine. Today there are deep learning models that have a good representation of the distribution of images and patches of images, and which can detect and even track different objects with high accuracy. By incorporating such models in our algorithm, in addition to or instead of the classical algorithms, we might be able to improve our tracking and accuracy and possibly even outlier filtering.

- Adjusting the Gaussian distribution assumption
  We have already expanded on the consequences of assuming Gaussian noise in situations where there may be outliers. There are methods that take into account the chance of getting outliers, such as, for example, instead of assuming Gaussian distribution on locations, assuming a multimodal distribution such as GMM that expresses the probability of outliers and allows the model to detect them and avoid using them in further calculations. The contribution of this approach to probabilistic models can be viewed as equivalent in some way to the RANSAC algorithm's contribution to the first, deterministic part of our algorithm. Indeed, there are algorithms which are robust to outliers that use such assumptions, and in those, usually if the prior probability of getting outliers is known in advance, the algorithms will converge very fast. If the prior probability of outliers is not known, there are other solutions such as the EM algorithm, but the convergence of these algorithms is much slower.

6.4 Our improvements, and Directions for future research

As mentioned above, in this project we implemented a test for whether the dynamic use of classical algorithms could improve the binding along the frames and thus prevent a bottleneck that would harm the accuracy of the algorithm. To this end, we identified the frames where there is a bottleneck in terms of connectivity in our algorithm (that uses the AKAZE detector), and analysed them using other common CV2 detectors to check whether it is possible to improve the performance on these frames by using other detectors and the results are indeed encouraging. Here are the results:
Our most extreme bottleneck in running the algorithm is at frame 3337, on which we will demonstrate the test we conducted. The results in other problematic frames are similar.
In terms of the number of matches, these are the number of matches that our detector, AKAZE, was able to produce in the following frames:
Frame : number of matches
3336 : 589

3337 : 590
3338 : 603
And in terms of the percentage of inliers:
Frame : Inliers percentage:
3337 : 29.2
3338 : 92.2
That is, the number of inliers in frame 3337 that the AKAZE algorithm was able to produce is 172.
In contrast, the BRISK detector reached the following results:
Frame : number of matches
3336 : 783
3337 : 801
3338 : 825
Frame : Inliers percentage:
3337 : 44.0
3338 : 45.8
That is, the number of inliers that the BRISK algorithm was able to produce in frame 3337 is 352.
Definitely a significant improvement.
This advantage can be explited online - i.e. while processing our MultiFrame object which dijests two StereoFrame's, if the number of inliers of points given the transformation we found is not sufficient (say α Mean(inliers)), then we can trigger another detection in both StereoFrames using other detector, and match again across time - untill the number of inliers increases enough. Since this is done only on a handful of frames it shouldn't affect much the running time of the PnP process, but should improve its results.
The time limit did not allow us to test how such a dynamic use of the classical detectors and matchers would affect the performance of the entire algorithm, since it requires another refactor of our code, plus defining a matcher between two different descriptors (this is possible given that they were used on the same frame - by comparing the pixels locations) but we believe that some noticable improvement is quite likely. In our repo there is a branch with some of the refactoring needed for this research implemented - but it's not fully completed unfortunately. We might still implement it after the course, because we believe it has the potential of being cost effective (compared to deep-learning approaches) while intruding improvements