Sparse Feature-Based Visual Odometry and SLAM
COMPGX04 Coursework 2

Maud Buffier

April 25, 2017

1 Sliding Window Visual Odometry

1.1 Complement Implementation of the Visual Odometry System

For this task, the code can be run from 'mainTask1.m'. The 3 questions are described in this script and can be performed separately by commenting the rest of the code. I extended the script of the provided file 'visualOdometry.m' and its prototype is now :

[rte_all, ate_all, rte_window, ate_window, stand_dev, meanMagSD, timeArray] =

visual_odometry(NUM_INITIALISE, WINDOW_SIZE, GPS, covariance, RTE_ATE, time)

The parameters are :

- NUM_INITIALISE indicates the number of frames to perform the initialization
- WINDOW_SIZE the size of the sliding window in the algorithm
- GPS (1 or 0) indicates if we want to add prior poses at camera positions for frames 1, 250 and 500.
- covariance (1 or 0) indicates if we want to get back the covariance measurement over time
- RTE_ATE (1 or 0) computes the Relative Trajectory Error (RTE) and Absolute Trajectory Error (ATE) metrics over time if the value is 1.
- Time (1 or 0) indicates if we want to store the value of the computational time for each frame.

The function returns :

- rte_all and ate_all, the value of the metrics computed from the start of the run to the current time
- rte_window and ate_window, the value of the metrics computed over just the length of the sliding window
- stand_dev and meanMagSD which gives indication about the behavior of the covariance for the camera positions.
- timeArray an array the computational time for each frame

Using this function, I can execute the algorithm with different values and plot the result using mainTask1.m. All the indication regarding the code are commented directly in the file with **TODO** indications.

Regarding the loop closure I made a choice I wanted to explain here. When I implemented it following the skeleton, I could have done 2 things : updating the indices in the features_used vector (containing the indices of the frame which has seen the feature for the last time) during a loop closure or not updated it. I tried both and :

- if I do update the indices, it means that a new loop closure will be detected after 'windowSize' frames (as the features will be thought to be in the new window)
- if I don't, during all the frames where there is a « loop closure » a big bundle adjustment is performed. Hence, it's slower but we obtain better result

According to me, the "big bundle adjustment" should be performed only once, when the loop closure is detected. I discussed this issue with a TA and it explains that it was due to a short cut in the implementation. Indeed, no map-point/feature (such as sift) association is implemented here and value are created artificially. Hence, in a real implementation, when we realized we are using points already seen, we close the loop once and then, the condition isn't satisfied any longer. This technique is explained in [1]
Hence, I choose to update the features_used vector after the loop closure. Hence, I indicated the last frame used as the one where the loop closure has been detected. We will see the consequences in the next questions.

1.2 Characterizing the Performance of the VO Algorithm

This question has been realized using 10 as a window size and as a number of frame to initialize. I have separated this question in 2 parts :

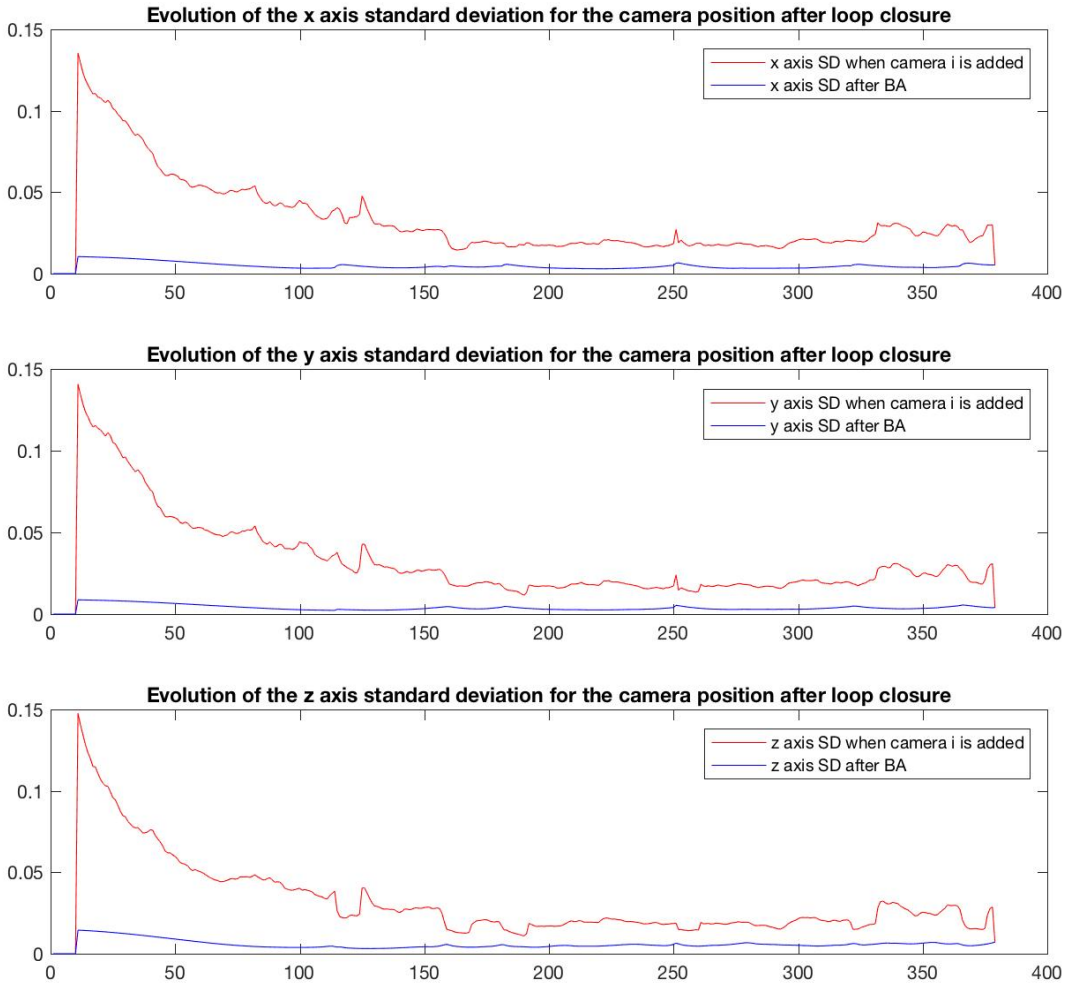
1. Observing the behaviors of the covariance after the loop closure . They are extracted using the marginalize operation of GTSAM
2. Observing the results of the metrics ATE and RTE during the process, both from the start to the current time and only on the current window.

1.2.1 Covariance estimation

To extract the covariance I used the marginalize operation of GTSAM. Hence, I extracted for each camera position, the 6*6 matrix of covariance. I extracted the 3 standard deviation for the position on the x,y and z axis. Then, I plotted 2 informations :

1. The covariance of the camera i directly after being added and the covariance of the same camera after a loop closure has been detected
2. The mean of the magnitude of the standard deviation vector from the start of the run to the current time

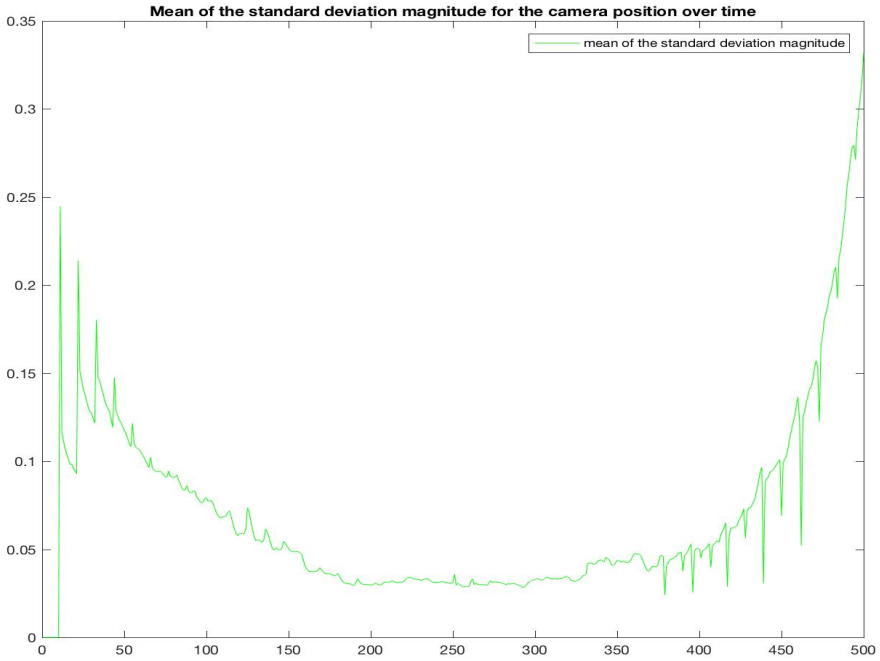
Here is the result for the standard deviation for each axis and for each camera, after being added (red) and after the loop closure detection (blue) :



Several interesting informations can be inferred : we can see that between frames 10 to 150, the standard deviation decreases for each axis (red curve). Indeed, by adding landmarks and viewing the same landmark several times, the system becomes more and more confidant about the position of the camera.
Then, we can see the loop closure detection around frame 380. We can then observe the standard deviation for each camera after the loop closure and see that the confidence of the placement has really increased (the standard deviation for the axis, which is also the size of the ellipse along this axis, is small). Indeed, the system optimizes the position using all the information it has seen before. Hence, it becomes more confidant about the positions of all the cameras.
An other thing we can notice is the small "bump" around frame 130 or 250 for the red curve. Those are due to bend on the road. Indeed, the system is seeing points it has never seen so the confidence decreases a bit.

Here is the mean of the magnitude of the standard deviation vector from the start of the run

to the current time :



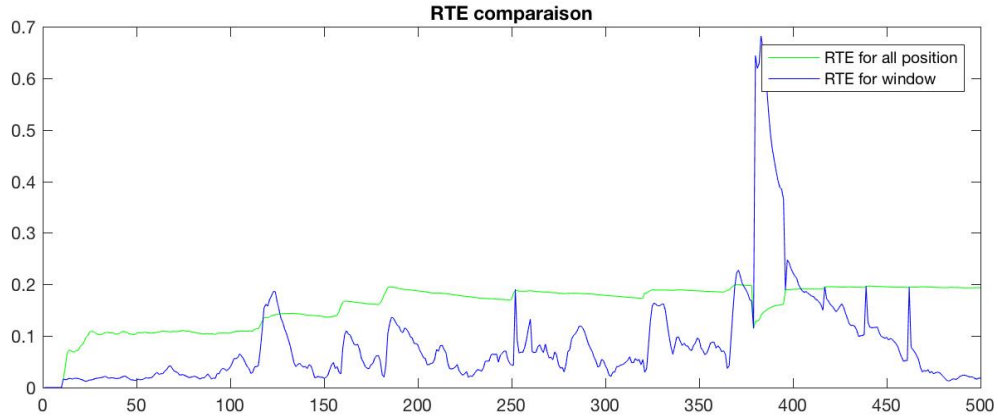
We can observe the same trend at the beginning : the ellipse size decreases. Then, it starts increasing again at the end and we can notice the big bundle adjustments performed which cause the ellipse size to decrease at time i. I think this "U" form is explained by the behavior of the car. At the beginning the speed increases, then stays stable and decreases until stopping at the end. However, we assume a constant velocity model in the code to find out the new camera position. Hence, the confidence drops as the velocity of the car doesn't follow the model at the beginning and at the end which causes the measurements not to be what expected.

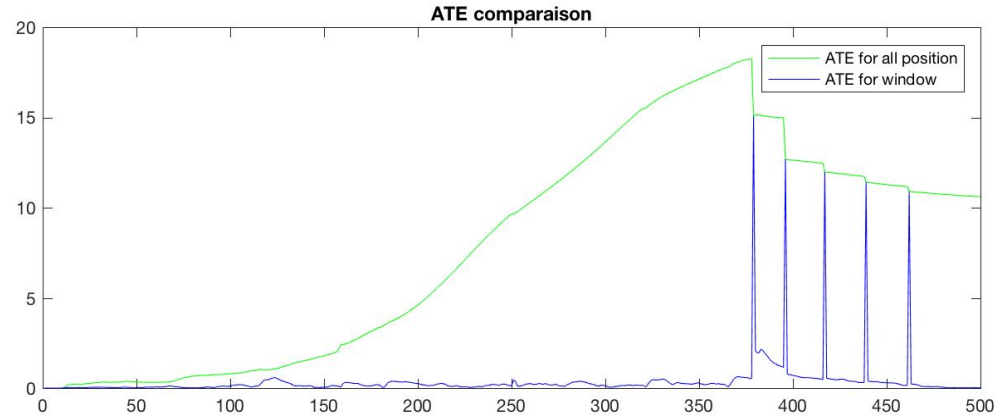
1.2.2 ATE and RTE metrics

The concepts of Absolute Trajectory Error (ATE) and Relative Trajectory Error (RTE) has been introduced in [2]. The RTE metric measures the local accuracy of the trajectory over a fixed time interval. In other words, we can say it measures the drift of the trajectory. The ATE metric gives the global consistency of the estimated trajectory. It's evaluated by comparing the absolute distances between the estimated and the ground-truth trajectory. In my case, I plotted 2 informations as required :

1. The RTE and ATE over just the length of the sliding window
2. The RTE and ATE from the start of the run to the current time

Here is the result for both :





Let's start with the RTE comparison :
We can see that for the green curve, representing the RTE metric from the start of the run to the current time, is stable. It means that the drift of the camera positions is constant over time. We can observe a drop of the drift with the bundle adjustment which optimize the position with all the information. For the blue curve, representing the RTE metric over just the length of the sliding window (of size 10) the result is different. We can clearly distinguish the bends (frame 130, 200, 280..) because the RTE metrics increases. Indeed, as the system has to find new landmarks it has never seen before, it creates a drift more important. Then during the bundle adjustment process, the metrics is very high because the system realigns so it drifted to get back to a better position. Of course, this drift isn't present in the original trajectory.

Now, let's compare the ATE metric :
As before, the green curve represents the RTE metric from the start of the run to the current time. We can see that it increases all along the trajectory. Actually, it's consistent with the RTE metric which indicates a constant drift : hence, the global error doesn't stop increasing. When bundle adjustment is performed, the error decreases significantly. It confirms the interest of performing loop closure in a visual odometry system. The blue curve represent the error between the ground-truth window and the current window. We can see the error is poor (also because the window is small hence the ATE can't be really high).

In both case, when the bundle adjustment is performed, the value for the metrics computed over just the length of the sliding window and from the start of the run to the current time are similar as all the measurements are included. It's also interesting to notice that the ATE measure-

ment at the end improves, even if the uncertainty represented by the size of the standard deviation increases as we saw before.

We can observe here the outputs of the python scripts provided :

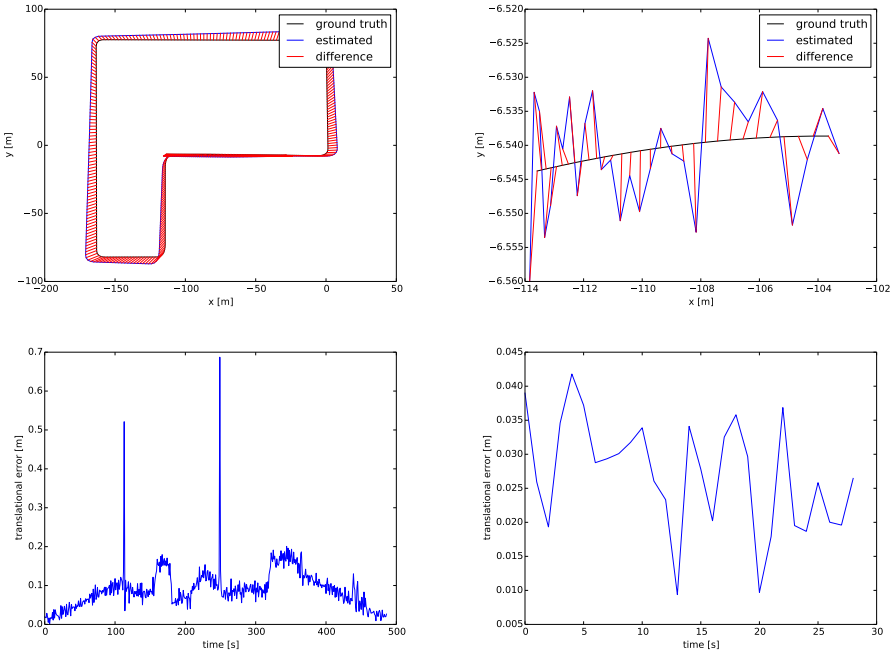


Figure 1: At the top the ATE for all the graph and only from the window, at the bottom the RTE for all the graph and for the window (the window size is 30)

As the window size is big in this case, the ATE is good for all the graph. Regarding the RTE we can observe the bends and a value stable which decreased at the end.

1.3 Optimizing the Configuration of the Algorithm

For this question, I computed the algorithm for 5 different window size 5,10, 15, 20 and 25 and I observed and compared the results. First, graphically speaking we can observe the improvement by looking at the trajectory :

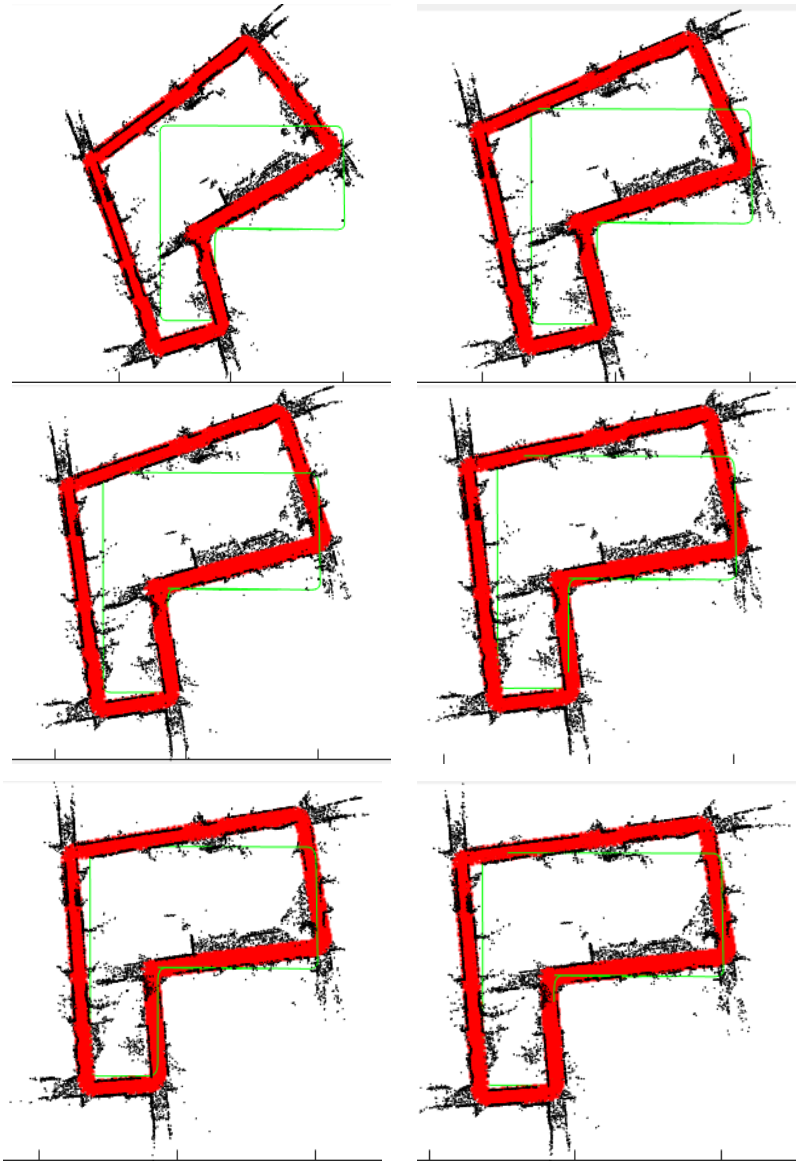
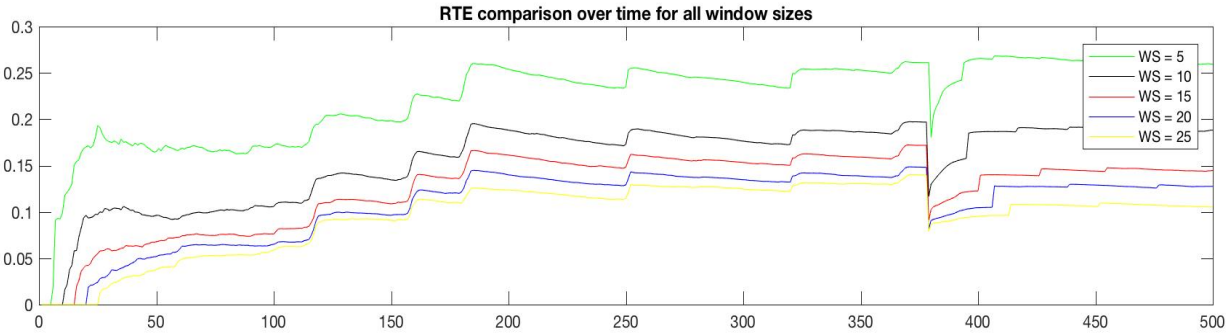
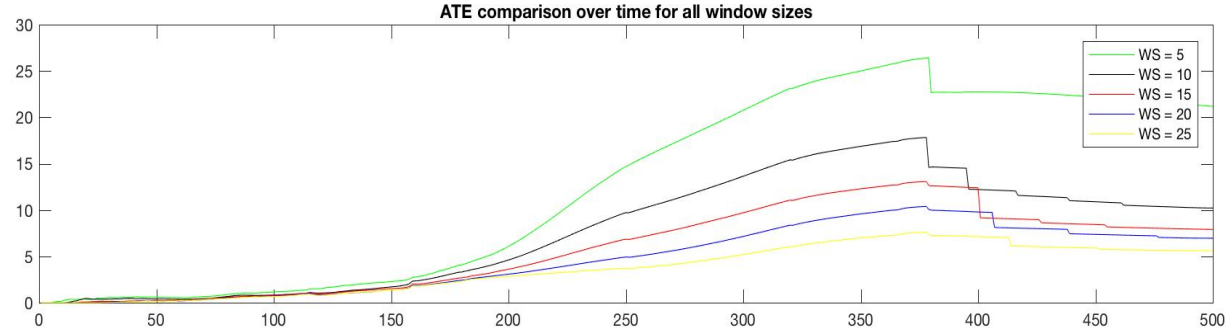
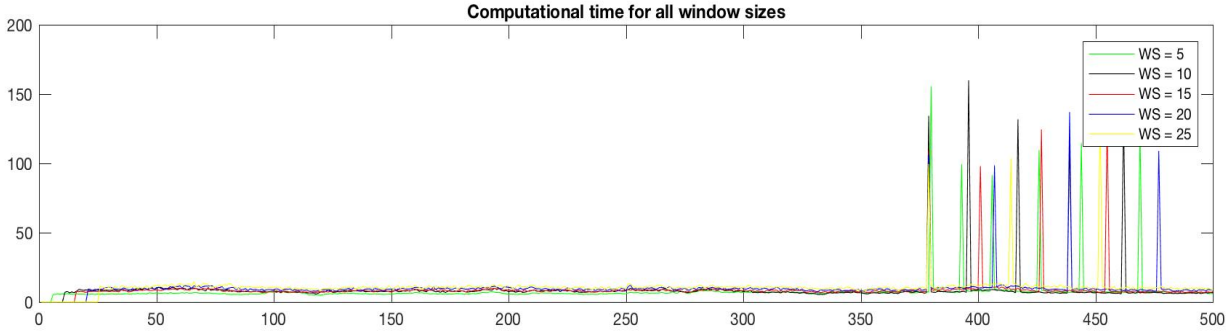


Figure 2: From the top left to the bottom right : widow size of 5,10,15,20,25 and 30

We can observe that the computed trajectory gets closer and closer to the ground truth. Hence the system is more precise. It's confirmed when we look at the ATE and RTE metrics from the start to the current frame. I also look at the computational time per frame. We can see that the bigger the window, the more accurate the trajectory is both in terms of RTE or ATE. However, the computational time in total is 25% more important for larger window (see later).

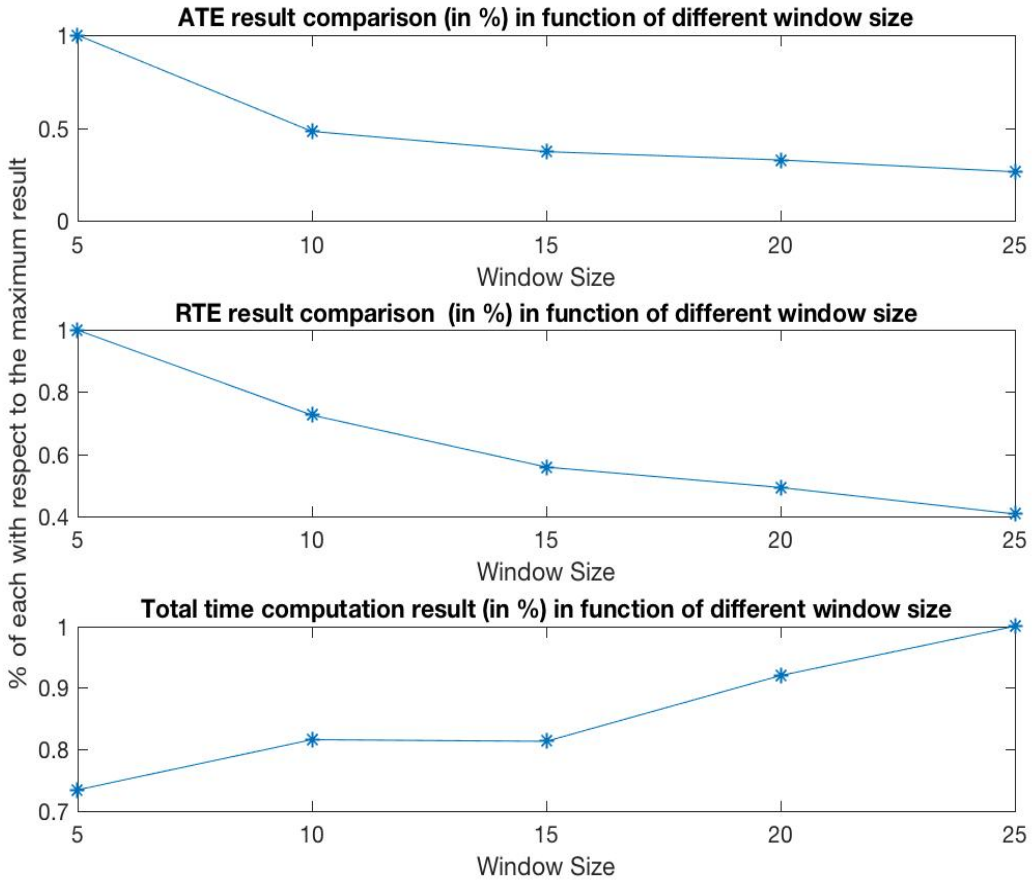






Finally, we can look at those metrics with respect to each others in order to find the best trade-off

between the window length, the computation time and the computed accuracy.



Hence, with those results, I think that the best window size would be 15. It gives a RTE and ATE results close that the best one for a window of 25 and a computational time which is only 7% higher that for a small window. Moreover, for a window of 20, the computational time jumps to 18% higher than for a window size of 5. Hence, from now I'll use a window of 15.

1.4 Effect of GPS

To modeled the effect of the GPS measurement I added a PoseTranslationPrior3D to the camera position 1,250 and 500. To determine the GPS measurement I added a noise around the grund-truth position with a standard deviation of 2m as asked. Then, I added the prior to this camera translation by creating a new 3D pose (see this part in the code).
Here is the visualization of the trajectory for the algorithm using the GPS and not using the GPS :

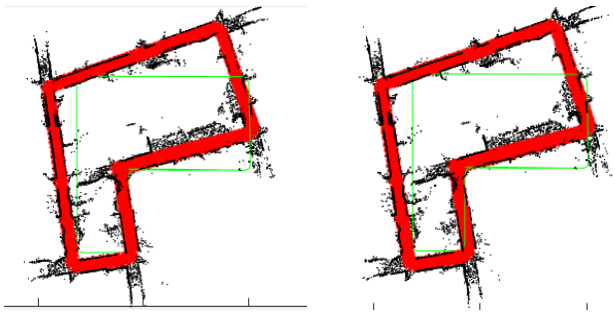
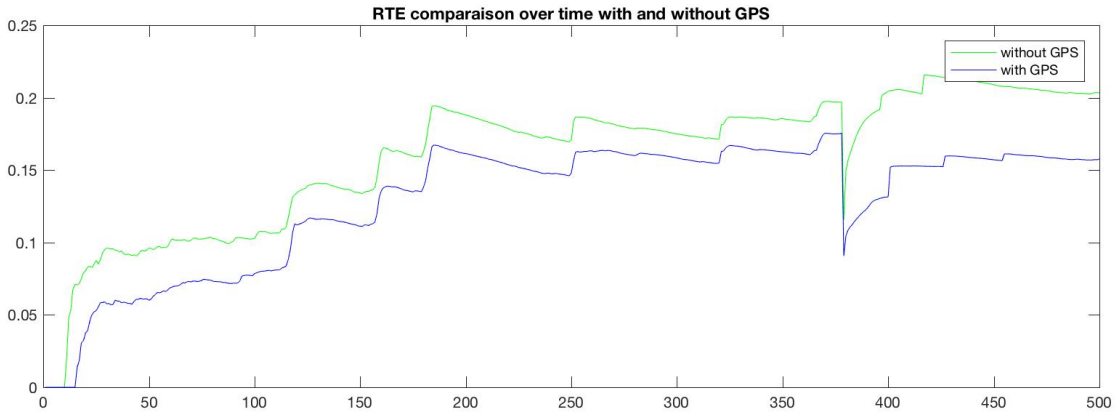
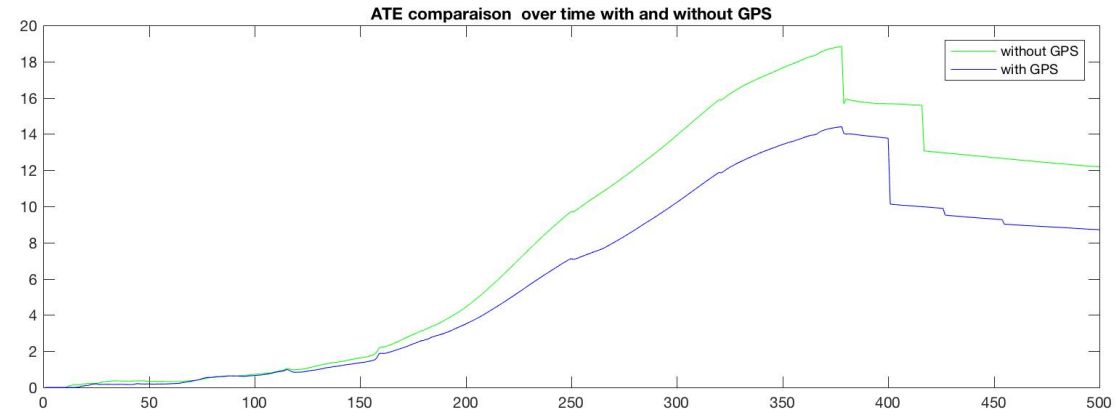


Figure 3: At the left without GPS priors and at the right : GPS prior

The difference isn't very clear but when plotting the RTE and ATE it becomes unambiguous :



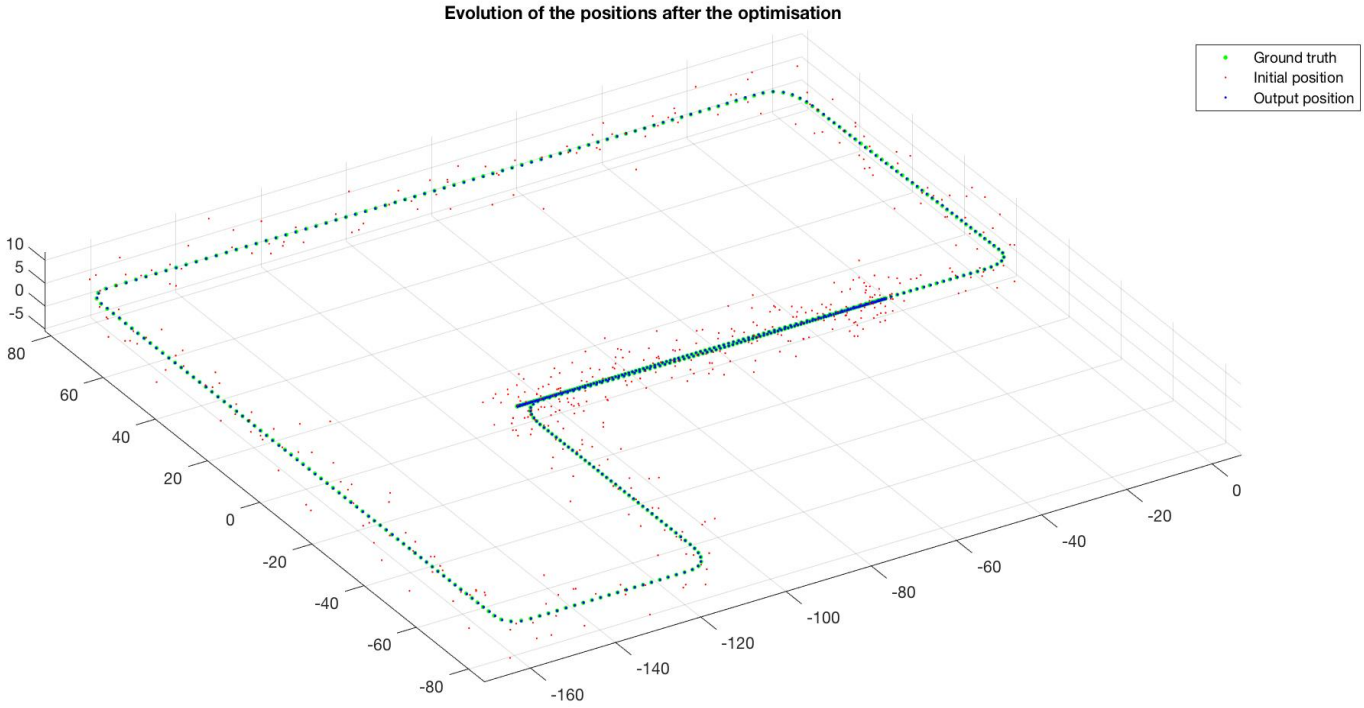


The drift (RTE measurement) is reduced, hence the global estimation with respect to the ground truth is improved. We can imagine that the result would have been even more accurate if a GPS prior would have been added at each frame.

2 Bundle Adjustment

2.1 Complete implementation of bundle adjustment system

The code for this task is in the file bundle_adjustment.m and can be run using the file 'main-Task2.m'. Here is the 3D plots of the result of estimate the 3D position of the cameras :



We can see that the ground truth position (in green), the position at the beginning of the algorithm (red) and the output position of the algorithm (blue).

2.2 Evaluate the Performance of the Bundle Adjustment Algorithm

I had lots of problem to evaluate the performance of the algorithm because I couldn't access the marginals. I got this error no matter what I tried :

Error using gtsam_wrapper
Exception from gtsam:
Indeterminant linear system detected while working near variable 8070450532247929146 (Symbol: p314).
Thrown when a linear system is ill-posed. The most common cause for this error is having under-constrained variables. Mathematically, the system is underdetermined.
Error in gtsam.Marginals (line 24) $my_ptr = gtsam_wrapper(1203, varargin1, varargin2);$

As for the first task, I tried to put priors only on the landmark positions but not on the camera ones, it didn't work out. Hence, I tried to put priors on the landmarks and the camera it didn't work either. I followed the discussion on the moodle forum and I understood it could be caused by 2 thinks :

- The system was ill because a landmark wasn't seen by the sufficient number of cameras (which is 2) and then it's position couldn't be inferred
- The baseline is too narrow meaning that the second measurements (if there is one) gives a too similar information as the first one, hence it can't be taken into account

However, I don't get why I called successfully the marginals in the first question and not in the second.
Hence, if I would have had access to the marginals, here are the curves I would have plotted :

- The mean of the standard deviation vector for all the landmarks in function of the iteration step. This way, I could have verified that the certainty of the landmark position was increasing with the number of iterations.
- For some landmarks, the value of the x,y and z variance in function of the iteration.

Those plots could have provided me with an estimation of the trend for the spatial distribution in function of the accuracy.

However, even without marginals, I tried to infer a trend between the location of features and the computed accuracy. Indeed, for each landmark, I can look at the distance between the ground-truth and the current position at iteration i. For the camera position, I re-used the ATE metric presented in the first part in order to see the evolution of the trajectory.
Here is the result :

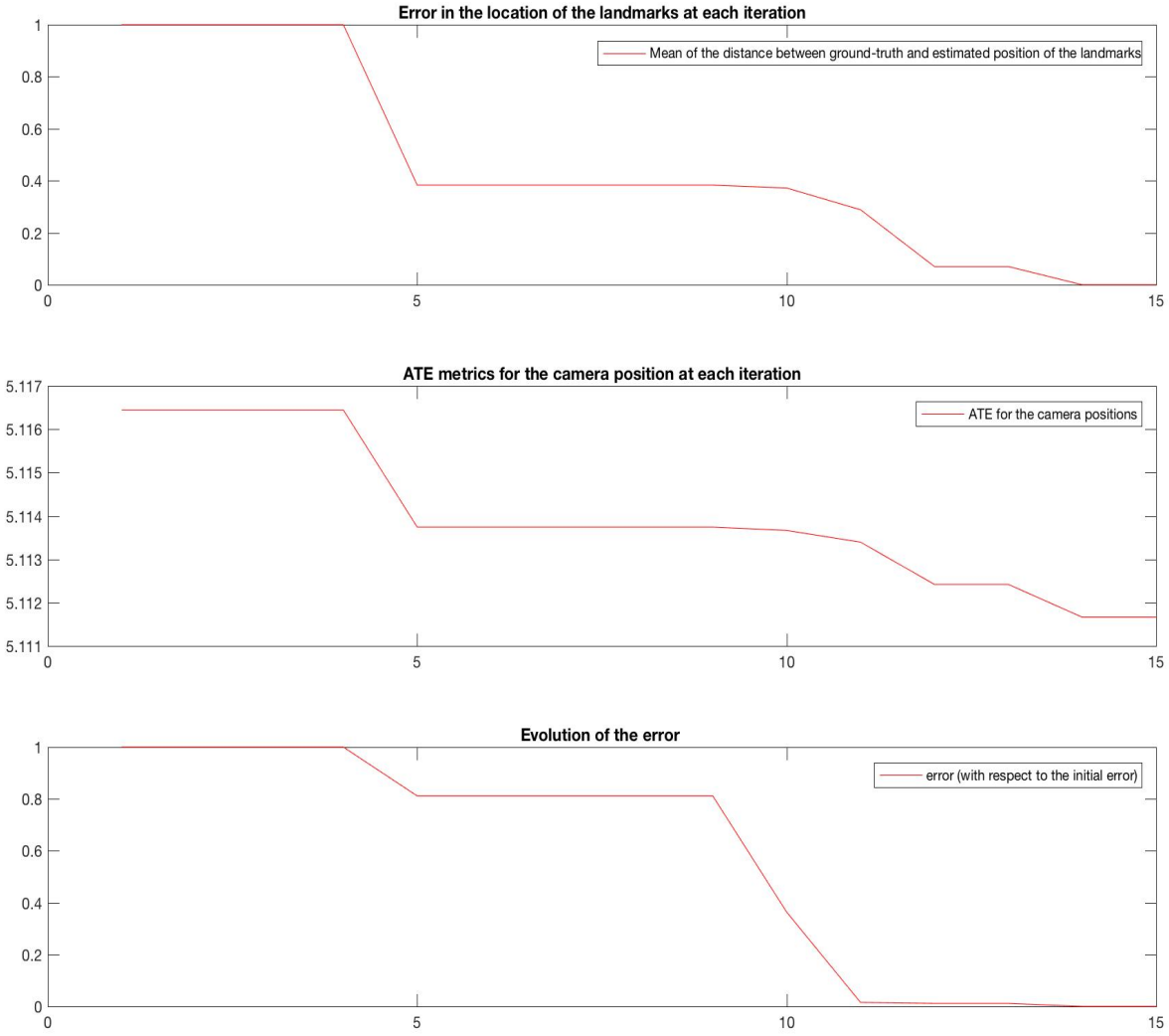


Figure 4: Put in relation the errors (for both the landmarks and the cameras) and the accuracy in function of the number of iterations

To better see the evolution, I put the error of the graph (3de graph) and the error for the landmarks (first graph) between 0 and 1.
Hence, we can see that at the 4th and 9th iterations, the accuracy of the result increases (the error decreases) whereas the mean error in the location of the landmarks and the ATE metric decrease. The correlation between those are clear : the better the features are localized, the better the accuracy is.

2.3 Explore Methods to Reduce the Complexity of the Graph

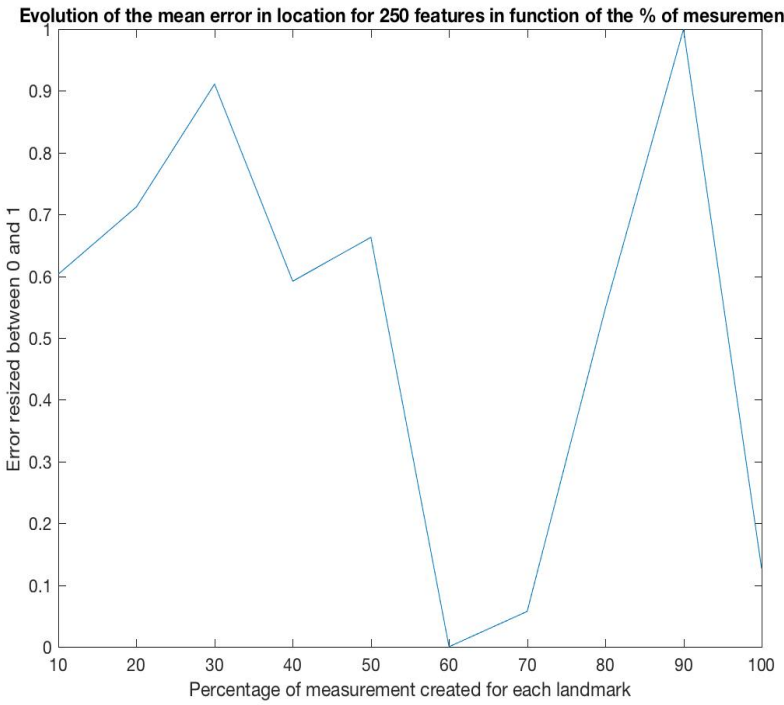
The way to reduce the complexity of the bundle adjustment process was explore in this paper [3] called Bundle Adjustment in the Large. Indeed, as any optimization problem it can be seen as the addition of several optimization sub-problems and simplified this way.
However, for this question, I did something simpler. I tried to investigate the behavior of the algorithm if only a certain percentage of the measurement where added to the graph. To do so, I create a function :

$$error\_location = bundle\_adjustment\_test(percentageWanted, ITERATIONS, nbFeatures)$$

Here is the explanation of the parameters and outputs :

- pourcentageWanted is the percentage of measurement we want to keep in the graph
- ITERATIONS is the number of iteration we are going to perform in the algorithm
- nbFeatures is the number of feature we are going to limit the number of measure and measure the error after 'ITERATIONS' iterations

The output, error_location is the mean for 'nbFeatures' features of the distance between the estimated position and the ground-truth. Hence, I tried to remove more and more measurements (or links between the camera and the features) to see if it changes the approximation of the position.



Hence, the curve has some unclear results : for example the higher error is obtained for 90% of the measurements and the lowest for 60%. However, we can still perceive a trend : the results with less than 50% of the measurements seems significantly poorer.
To plot this curve, I choose the 250 first landmarks seen by the first camera and looked how many time they were seen by the 50 next cameras. Then, for each feature, I computed a threshold

corresponding at the percentage of measurement asked. By iterating on the camera I added the measurements only if the number of measurement added for this landmark was still below the threshold.
Once again, it would have been nice to be able to observe the convergence of the covariance for the landmark. However, I couldn't access the covariance and I obtained the same error as the previous question..

To prune the graph, I then choose to only compute the bundle adjustment with only 60% of the measurements, to be able to see the difference over the complete graph. The function I implemented for this question can be found in :

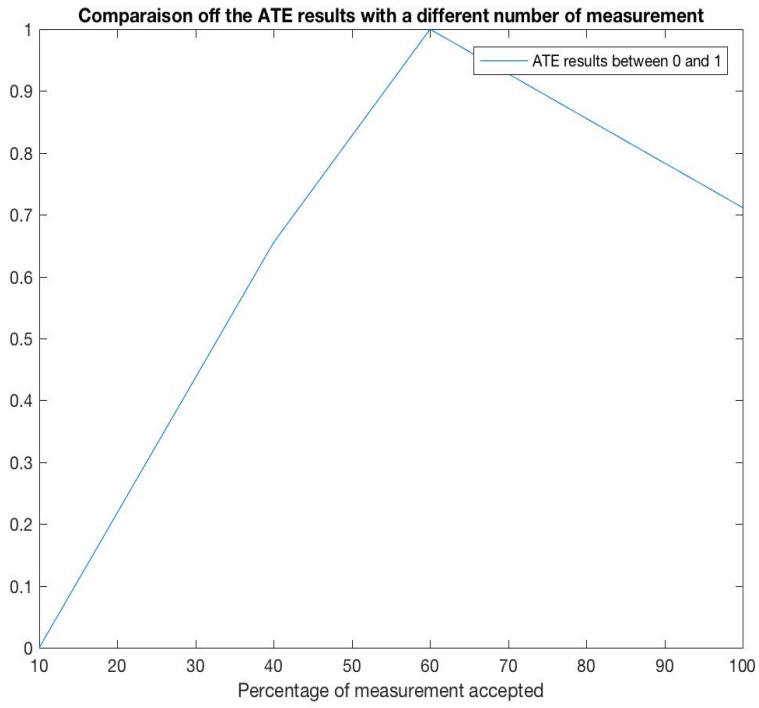
$$[error\_location, ATE\_result, accuracyArray] = bundle\_adjustment\_lessMeas$$

$$(uncertainty, ATE, accuracy, ITERATIONS, percentage, windowSize)$$

Here is the explanation for the parameters :

- uncertainty, ATE and accuracy are the same as the previous question for the bundle adjustment, used to provide an evaluation of the performances
- iteration gives the number of iteration for the optimization
- percentage indicates the percentage of the measurement we want to keep
- windowSize indicates the size of the window used in the algorithm. Indeed, we need to avoid putting all the landmarks at the beginning of the graph otherwise the last frame won't have any measurements left to infer its position. Hence, I used a window and allow only 1 measurement for a landmark in this window. Hence, the measurement are distribute on all the frames

The output is used to plot the results and observe them with a different number of measurement. Here is the result for the ATE metric :



We can see that, whereas 40% and 60% give results comparable to 100% of the measurements, having 10% gives a very bad result.
Hence, we have seen that we didn't have to add all the measurements, around 40-50% is sufficient to obtain an acceptable result.

References

[1] J. M. M. Montiel Raul Mur-Artal. ORB-SLAM: A Versatile and Accurate Monocular SLAM System. In *IEEE Transactions on Robotics*, Oct 2015.

[2] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, 2012.

[3] Steven M. Seitz Sameer Agarwal, Noah Snavely and Richard Szeliski. Bundle adjustment in the large. In *European Conference on Computer Vision*, 2010.