

Localization in urban environments: monocular vision compared to a differential GPS sensor

Eric Royer, Maxime Lhuillier, Michel Dhome, Thierry Chateau

▶ **To cite this version:**

Eric Royer, Maxime Lhuillier, Michel Dhome, Thierry Chateau. Localization in urban environments: monocular vision compared to a differential GPS sensor. Jun 2005, 8 p. hal-00118555

HAL Id: hal-00118555
https://hal.science/hal-00118555

Submitted on 5 Dec 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Localization in urban environments : monocular vision compared to a differential GPS sensor

Eric Royer Maxime Lhuillier Michel Dhome Thierry Chateau

LASMEA, UMR6602 CNRS and Blaise Pascal University
24 Avenue des Landais
63177 Aubière CEDEX
Eric.ROYER@lasmea.univ-bpclermont.fr

Abstract

In this paper we present a method for computing the localization of a mobile robot with reference to a learning video sequence. The robot is first guided on a path by a human, while the camera records a monocular learning sequence. Then a 3D reconstruction of the path and the environment is computed off line from the learning sequence. The 3D reconstruction is then used for computing the pose of the robot in real time (30 Hz) in autonomous navigation. Results from our localization method are compared to the ground truth measured with a differential GPS.

1. Introduction

We address the problem of real time localization in urban environment. Our goal is to develop a mobile robot able to navigate autonomously on a long distance path (from several hundred meters, to a few kilometers). We guide the robot manually for some time and record a reference video sequence. Then the robot should be able to follow the same trajectory by itself. For this application, we have developed a system that builds a three dimensional model of the robot's environment using only visual data recorded during the learning phase. After that, when the robot is near the learning trajectory, it is possible to use the current frame taken by the camera to localize the robot in real time. The only sensor used is a calibrated camera. An overview of the process is illustrated in figure 1. Map building is the more complex part of the algorithm. Fortunately, this part of the computation can be done off line. So the real time constraint applies only to the localization of the robot. That means that we can use computation intensive algorithms to build a good map. We think that a global and costly optimization step is necessary for practical cases, both for accuracy and robustness under our hypothesis. Ego-motion estimation or visual odometry which do not use a global optimization have been proposed for example by D. Nistér et al. [13], but the accuracy is not as good as error build up over time.

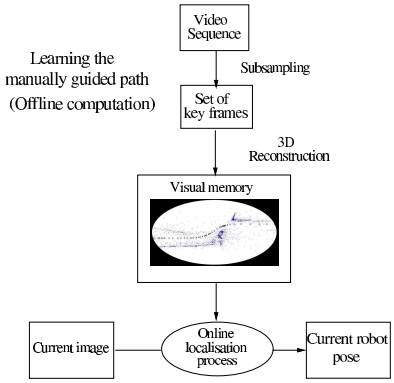


Figure 1. An overview of our vision system

In many applications, localization is provided by a GPS (Global Positioning System) sensor. A differential GPS (DGPS) has an accuracy of a few centimeters if enough satellites are visible, so it is good enough for mobile robot navigation. Unfortunately the GPS doesn't work well in urban environments where tall buildings can occlude the satellite signals. Moreover the signals can reflect on the buildings and corrupt the localization results. The worst case is when the robot is in a narrow street (6 m wide), also called an urban canyon. In this case, all the visible satellites must lie in the same vertical plane, which is a bad situation for computing a localization. On the other hand, this kind of environment has a lot of information to offer to a vision system, because there are a lots of features near the robot. So

our goal is to make a system suitable to urban environments that can provide localization information with the same accuracy as a DGPS. At the end of this paper, our localization results are compared to those obtained by a DGPS sensor.

A solution for visual navigation with reference to a prerecorded image sequence was presented by Yoshio Matsumoto et al. [11] but the method did not provide the pose of the robot at each frame. Seung-Hun Jeon et al. [9] proposed a method that allows to compute the pose of the robot in the case of indoor environments with horizontal and vertical planes. In our case, we want to be able to find a localization in an outdoor environment without assumptions on the geometry of the scene. An approach closer to ours has been proposed by Kiyosumi Kidono et al. [10]. After a human guided phase, the robot builds a map of the environment and uses it to localize itself in autonomous navigation. This system relies both on vision and odometry. Additionally it assumes a calibrated stereo rig for vision data acquisition and movements are done in 2D on a ground plane. In our case we use only one camera and no odometry, the ground doesn't have to be planar. Recently, the idea of using a visual memory has been proposed by Remazeilles et al. [15]. In this approach, the path is modeled as a set of key frames and the robot goes from one frame to the next by visual servoing. This approach avoids building a 3D reconstruction but is not able to provide a 3D localization of the robot. The human guided approach is different from the Simultaneous Localization And Mapping (SLAM) because in our case, the landmarks are visible in a short period of time. They can't be observed again and again. Recently, monocular SLAM has been developed. For example the work of Davison [3] provides a way of computing the camera pose in real time with only one camera. But this approach assumes that the landmark database is kept small (under about 100 landmarks). This is well suited for computing localization in a room but not in a street where landmarks can be observed for a few meters and are replaced by new ones.

In section 2 we present the method used to build the map from the reference image sequence. Then in section 3 the method used to compute the localization of the robot in real time is detailed. Finally we present some results in section 4 and we compare them with the ground truth obtained with a differential GPS sensor.

2. Reconstruction of the reference sequence

2.1. Overview

The goal of the reconstruction is to obtain the position of a subset of the cameras in the reference sequence as well as a set of landmarks and their 3D location, all of these given in a global coordinate system. For the reconstruction we use a calibrated monocular image sequence. For our experiments, the camera was calibrated using a planar calibration pattern. Camera calibration is important because the wide angle lens we used has a strong radial distortion. Knowing the internal parameters and distortion coefficients makes the structure from motion more robust and increases the accuracy of the reconstruction. In addition, it works even if the scene is planar in some images, which is not the case for uncalibrated approaches (for example [2], [14]).

Every step in the reconstruction as well as the localization relies on image matching. Matching a pair of images is done by detecting interest points in each image. Harris corner detector [6] is used for this step. For each interest point in image 1, we select some candidate corresponding points in a region of interest defined in image 2. Then a Zero Normalized Cross Correlation score is computed between interest point neighborhoods. And the pairs with the best scores are kept to provide a list of corresponding point pairs between the two images. Matching images this way may sound too slow for a real time application, but it is possible to implement a very efficient corner detector using SIMD extensions of modern processors.

In the first step of the reconstruction, we extract a set of key frames from the reference sequence. Then we compute the epipolar geometry and camera motion between key frames. Additionally, the interest points used to compute the epipolar geometry are reconstructed in 3D. These points will be the landmarks used for the localization during the on-line phase. They are stored with their neighborhood in the images so that it is possible to match them with interest points detected in new images.

2.2. Key frame selection

If there is not enough camera motion between two frames, the computation of the epipolar geometry is an ill conditioned problem. So we select images so that there is as much camera motion between key frames while still being able to match the images. The first image of the sequence is always selected as a key frame, it is noted I_1. The second key frame I_2 is chosen so that there is at least M common interest points between I_1 and I_2. Then when key frames I_1 through I_n are chosen, we select key frame I_{n+1} so that there is at least M interest points in common between I_{n+1} and I_n and at least N common points between I_{n+1} and I_{n-1}. This ensures that there are enough point matches between key frame to compute camera motion. In our experiments we detect 1500 interest points in each frame and we choose $M = 400$ and $N = 300$.

2.3. Camera motion computation

At this point we have a subset of the frames from the video sequence recorded during the learning step. The

structure and motion algorithm used to build a 3D reconstruction of the scene and camera motion can be separated in three parts. With the first three key frames we compute the camera motion over the three frames by computing an essential matrix. Then for each successive set of three images, the motion is computed using a pose estimation algorithm. These computations produce an initial estimate of the camera motion, and a hierarchical bundle adjustment is used to refine this initial estimation.

For the first image triplet, the computation of the camera motion is done with the method proposed by Nistér [12] for three views. It involves computing the essential matrix between the first and last images of the triplet using a sample of 5 point correspondences. Computing the essential matrix E between two images is done by writing the epipolar constraint each of the 5 points projections must verify : ${q'^i}^T E q^i = 0, \forall i \in \{1..5\}$ and a 6^{th} relation that is verified by any essential matrix :$EE^TE - \frac{1}{2}trace(EE^T)E = 0$. These relations lead to a 10^{th} order polynomial equation, so there are at most 10 solutions for E. Each matrix E gives 4 solutions for (R, T). The solutions for which at least one of the 5 points is not reconstructed in front of both cameras are discarded. Then the pose of the remaining camera is computed with 3 out of the 5 points in the sample. This process is done with a RANSAC [4] approach : each 5 point sample produces a number of hypotheses for the three cameras. The best one is chosen by computing the reprojection error over the three views for all the matched interest points and keeping the one with the higher number of inlier matches.

We need an algorithm to compute the pose of the second camera. A review of calibrated pose estimation algorithms is given by Haralick et al. [5]. If the internal parameters of the camera are known, with three 3D points P^i whose projections in the image are known, it is possible to compute the pose of the camera. We chose Grunert's method as it is described in [5]. This method relies on trigonometrical computations in the tetrahedron formed by the three 3D points and the optical center of the camera. Relations are computed between the distance of each of the 3D points to the optical center and this leads to a 4^{th} order polynomial equation. There are at most 4 solutions for each sample of 3 points. The solution is chosen in the RANSAC process.

For the next image triplets, we use a different method for computing camera motion. Assume we know the location of cameras C_1 through C_N, we can compute camera C_{N+1} by using the location of cameras C_{N-1} and C_N and point correspondences over the image triplet $(N-1, N, N+1)$. We match a set of points P^i whose projections are known in each image of the triplet. From the projections in images $N-1$ and N, we can compute the 3D coordinates of point P^i. Then from the set of P^i and their projections in image $N+1$, we use a calibrated pose estimation algorithm to compute the location of camera C_{N+1}. In addition the 3D location of the reconstructed interest points are stored as they will be the landmarks used for the localization process. The pose estimation algorithm is based on Grunert's method and a RANSAC selection process. Random samples of three points are used to compute the pose of camera C_{N+1}. The advantage of a such an iterative pose estimation process is that it can deal with virtually planar scenes. After the pose computation, a second matching step is done with the epipolar constraint based on the pose that had just been computed. This second matching step allows to increase the number of correctly reconstructed 3D points. This is particularly important because we need as many points as possible for the computation of the next camera. When doing this there are 20 %more points correctly reconstructed.

2.4. Hierarchical bundle adjustment

The computation of camera motion previously presented doesn't give a very good solution. Moreover, the computation of camera C_N depends on the results of the previous cameras and errors can build up over the sequence. In order to correct this problem, we use a bundle adjustment which provides a better solution. The bundle adjustment is a Levenberg-Marquardt minimization of the cost function $f(C_E^1, \cdots, C_E^N, P^1, \cdots, P^M)$ where C_E^i are the external parameters of camera i, and P^j are the world coordinates of point j. The cost function is the sum of the reprojection errors of all the inlier reprojections in all the images :

$$f(C_E^1, \cdots, C_E^N, P^1, \cdots, P^M) = \sum_{i=1}^{N} \sum_{j=1, j \in J_i}^{M} d^2(p_i^j, K_i P^j)$$

where $d^2(p_i^j, K_i P^j)$ is the squared euclidian distance between $K_i P^j$ the projection of point P^j by camera i, and p_i^j is the corresponding detected point. K_i is the 3×4 projection matrix built from the parameters values in C_E^i and the known internal parameters of the camera. And J_i is the set of points whose reprojection error in image i is less than 2 pixels at the beginning of the minimization. After a few iteration steps, J_i is computed again and more minimization iterations are done. This inlier selection process is repeated as long as the number of inliers increase.

It's not a good idea to compute all the camera locations and use the bundle adjustment only once on the whole sequence. In that case, increasing errors could produce an initial solution too far from the optimal one for the bundle adjustment to converge. Thus it is necessary to use the bundle adjustment throughout the reconstruction of the sequence. Using an adjustment after each new frame is reconstructed is possible but very time consuming, and impractical for large sequences. A faster solution as described in [7] is to use the adjustment hierarchically. A large sequence is divided into two parts with an overlap of two frames in

order to be able to merge the sequence. Each subsequence is recursively divided in the same way until each final subsequence contains only three images. Each image triplet is processed as described above. For the first triplet we obtain the geometry by computing an essential matrix. Each remaining triplet has its first two frames in common with the previous one. So the first two cameras of the triplet are deduced from the previous triplet. The third camera is computed using the pose estimation algorithm. After each triplet has been computed we run a bundle adjustment over its three frames.

In order to merge two sequences S^1 and S^2, we use the last 2 cameras S^1_{N-1} and S^1_N of S^1 and the first 2 cameras S^2_1 and S^2_2 of S^2. As the images are the same, the cameras associated after merging must be the same. So we apply a rotation and a translation to S^2 so that S^1_N and S^2_2 have the same position and orientation. Then the scale factor is computed so that $d(S^1_{N-1}, S^1_N) = d(S^2_1, S^2_2)$, where $d(S^i_n, S^j_m)$ is the euclidian distance between the optical centers of the cameras associated with S^i_n and S^j_m. This doesn't ensure that S^1_{N-1} and S^2_1 are the same, so a bundle adjustment is used on the result of the merging operation. Merging is done until the whole sequence has been reconstructed. The reconstruction ends with a global bundle adjustment. The number of points used in the bundle adjustment is on the order of several thousands.

3. Real time localization

The output of the learning process is a 3D reconstruction of the scene : we have the pose of the camera for each key frame and a set of 3D points associated with their 2D positions in the key frames. At the start of the localization process, we have no assumption on the vehicle localization. So we need to compare the current image to every key frame to find the best match. This is done by matching interest points between the two images and computing a camera pose with RANSAC. The pose obtained with the higher number of inliers is a good estimation of the camera pose for the first image. This step requires a few seconds but is needed only at the start. After this step, we always have an approximate pose for the camera, so we only need to update the pose and this can be done much faster.

Here we describe the update process in order to find the current camera pose. The current image is noted I_{cur}. First we assume that the camera movement between two successive frames is small. So an approximate camera pose (we note the associated camera matrix K_0) for image I_{cur} is the same as the pose computed for the preceeding image. Based on K_0 we select the closest key frame I_{key} in the sense of shortest euclidian distance between the camera centers. I_{key} gives us a set of interest points IP_{key} reconstructed in 3D. We detect interest points in I_{cur} and we match them with IP_{key}. To do that, for each interest point in IP_{key}, we compute a correlation score with all the interest points detected in I_{cur} which are in the search region. For each interest point in IP_{key} we know a 3D position, so with K_0 we can compute an expected position of this point in I_{cur}. In the matching process the search region is centered around the expected position and its size is small (20 pixels width and 12 pixels heigth). After this matching is done, we have a set of 2D points in image I_{cur} matched with 2D points in image I_{key} which are themselves linked to a 3D point obtained during the reconstruction process. Figure 2 shows the interest points matched between a frame from the video and the corresponding key frame. With these 3D/2D matches a better pose is computed using Grunert's method through RANSAC for rejecting outliers. This gives us the camera matrix K_1 for I_{cur}. Then the pose is refined using the iterative method proposed by Araújo et al. [1] with a few modifications in order to deal with outliers. This algorithm is a minimization of the reprojection error for all the points using Newton's method. At each iteration we solve the linear system $J\delta = e$ in order to compute a vector of corrections δ to be subtracted from the pose parameters. e is the error vector formed with the reprojection error of each point in x and y. J is the Jacobian matrix of the error, Araújo gives a way to compute J explicitly. In our implementation, the points used in the minimization process are computed at each iteration. We keep only the points whose reprojection error is less than 2 pixels. As the pose converges towards the optimal pose, some inliers can become outliers and conversely. Usually, less than five iterations are enough.

4. Results

4.1. Checking our results

In order to evaluate the accuracy of both our reconstruction and our localization algorithm, we used a differential GPS sensor to record the position of the vehicle. The GPS data is used as the ground truth. Recording GPS signals with a great accuracy is not possible everywhere. So we had to find a place with not too many buildings so that enough satellites were visible. This is an unfavourable case for our algorithm because it was designed to work best in a dense urban environment where a lot of visual features are available. So we made two kind of experiments. Some video sequences where recorded along with the GPS data in a place where the signals where available to have a comparison with the ground truth. And a second set of sequences were made in three different places (from a wide road to a narrow street) to evaluate how much the accuracy of the localization changes from one place to the other. We could also record longer sequences in these places. In city centers we have been able to compute reconstructions for sequences

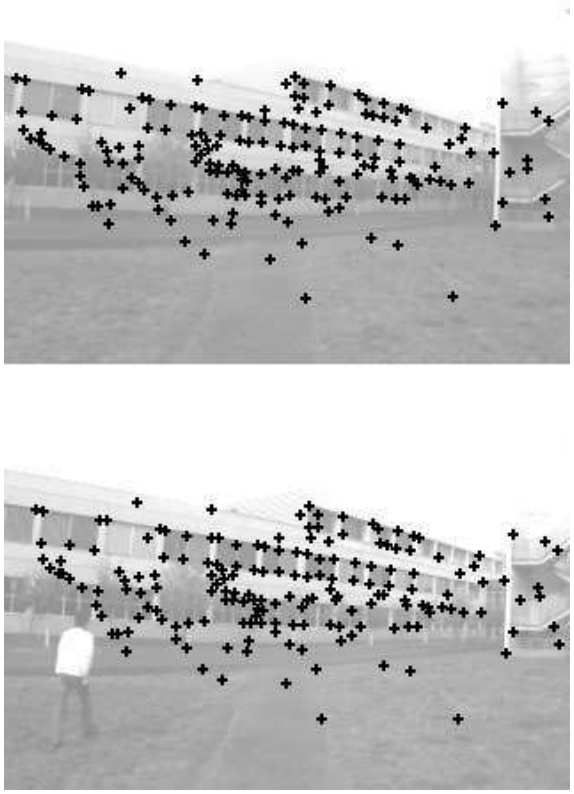


Figure 2. Matching interest points between a video frame and the corresponding key frame (only inliers are shown)

up to 500 meters long and 250 key frames. Such a reconstruction appears on figure 9 with images extracted from the video on figure 10.

4.2. Comparison with the ground truth

Comparing positions obtained by the GPS or with our vision algorithm is not completely straightforward. Two operations are needed so that both data sets can be compared. First the GPS sensor is not mounted on the vehicle at the same place as the camera. The GPS is located at the midpoint between the rear wheels of the car, while the camera is between the front wheels. So the two sensors don't have the same trajectory. From the GPS positions, we computed a "virtual" GPS which indicates what a GPS would record if it was at the same place as the camera. In addition, the 3D reconstruction is done in an arbitrary euclidian coordinate system, whereas the GPS positions are given in another coordinate system. So the whole 3D reconstruction has to be



Figure 3. A few images from one of our video sequences

transformed using a rotation, translation and scale change. The approach described by Faugeras et al. [8] is used to compute this transformation. After these transformations have been made, for each camera we are able to compute the error on the position in meters. For this, we assume that the GPS data is exact. The GPS sensor we used is a Real Time Kinematics Differential GPS (Thalès Sagitta model). It is accurate to 1 cm in an horizontal plane. The accuracy on a vertical axis is stated to be 2 cm but on our hardware platform we could not have more than 20 cm accuracy. So we discarded the vertical readings and all the localization errors reported in this article are measured in an horizontal plane only.

We recorded three video sequences on approximately the same trajectory and we computed reconstructions on each video sequence. Then to evaluate our reconstruction algorithm, we used one of these sequences as the reference sequence in order to run the localization algorithm on the two other ones. The length of the path for these sequences was about 80m. You can see a few images on figure 3. Figure 4 shows the reconstruction obtained from video1 as seen from above. The black squares are the position of the key frames, while the 3D points appear as dots. Some pedestrians were passing by, but this did not perturb the algorithms.

After building the 3D reconstruction from the key frames with the structure from motion algorithm, we used the localization algorithm to get the pose of each camera in the sequence (not only key frames). Then we computed the mean error for every frame. The results are given in table 1 as well as the number of key frames and the number of points in each sequence. Figure 5 shows the positions computed for each key frame (little circles), with reference to the trajectory recorded by the DGPS (solid line).

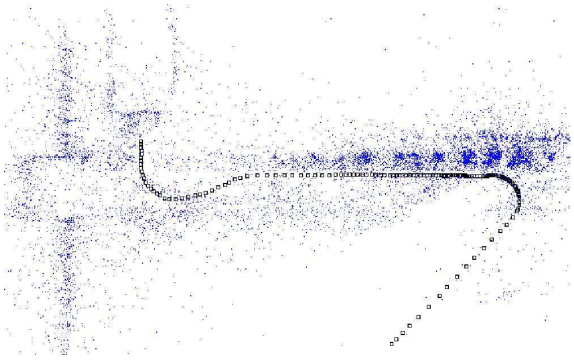


Figure 4. Top view of a 3D reconstruction

Sequence	mean error	number of key frames	number of 3D points
video1	15 cm	143	15149
video2	18 cm	174	17025
video3	13 cm	191	17772

Table 1. Mean reconstruction error and number of frames and points for a 80m long path

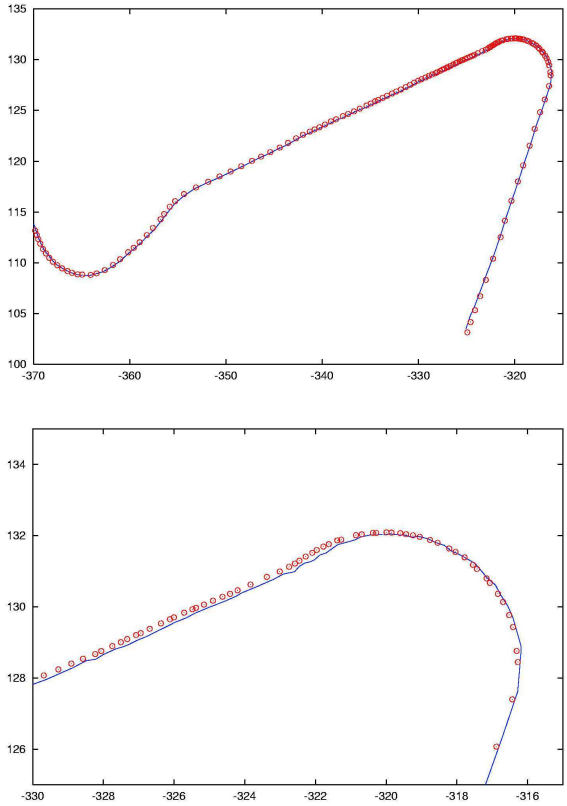


Figure 5. Position of the key frames with reference to the trajectory recorded by the DGPS (units in meters). Whole trajectory (top) and close up view (bottom)

We computed a localization for each frame of video$_i$ using video$_j$ as the reference sequence for each i and j such that $i \neq j$. Then we measured the distance between the position obtained by this algorithm and the position measured by the GPS for each camera. The mean error over all the frames of each sequence was then computed. So we made 6 localization experiments. The mean error was between 12 cm and 17 cm depending on the video sequence. Computing an average value for all the 6 experiments gave 15 cm. An example of the localization error for each frame in the sequence is displayed on figure 7. The corresponding trajectories are displayed on the same graph on figure 6.

4.3. Localization accuracy in different environments

In places where recording GPS signals is not possible, we used another way to check our results. We recorded two video sequences simultaneously with two cameras. The cameras where rigidly fixed on a car, one on the right side and the other on the left side. The left sequence is used as the reference trajectory. The other one is the sequence to be localized. With this setup, the localization algorithm should indicate a constant distance between the current localization of the robot and the reference trajectory.

We selected three places for these experiments : a wide road (sequence named "Wide"), a road with a wall on one side (called "Intermediate") and a narrow street with walls on both sides (called "Narrow"). One image extracted from each of the three video sequences are shown on figure 8.

For each of the three experiments, we built a 3D reconstruction using the left sequence. We used this reconstruction as the reference in order to localize the right camera. We also computed the pose of each camera (not only key frames) in the right sequence by using our localization algorithm on the right sequence also. Then for each frame, we computed the euclidian distance between the right and the left camera centers. This distance should be constant, so the error we made in the localization process is given by the standard deviation of this distance. For these three experiments, the standard deviation was 50 cm for the "Wide" sequence, 25 cm for the "Intermediate" sequence and 5 cm for the "Narrow" sequence. These results are not directly comparable to those presented in section 4.2 because they don't integrate the drift that can happen in the reconstruction. Nevertheless these experiments suggest that the accuracy is greatly increased if more buildings are near the

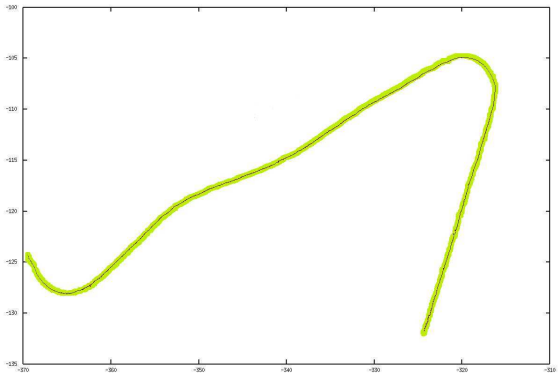


Figure 6. Trajectory recorded by the DGPS (broad line) and the trajectory computed with the localization algorithm (thin line)



Figure 8. The three video sequences : Wide, Intermediate, Narrow

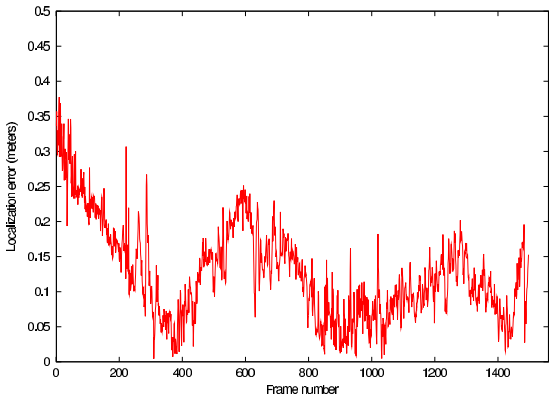


Figure 7. Localization error for each frame

camera. That's exactly the kind of environment where the accuracy of the GPS decreases considerably, so the two localization approaches are complementary and should work well in data fusionning.

4.4. Computation times and memory usage

All the timings were made on a 2.4GHz Pentium 4 processor with 320x240 images and 1500 interest points detected in each frame. Map building with about 150 key frames takes about one hour. The localization process runs at the video frame rate (30 Hz). Detecting interest points takes 13 ms, matching requires 10 ms, Computing the pose with RANSAC and the iterative algorithm takes 10 ms. The Memory space needed to store the database for the large sequence shown on figure 9 is approximately 26 Mb. So we can expect to store 20 m per megabyte of memory without data compression. Of course it depends on the sequence as more key frames are needed where the road turns. Fortunately, only a small part of the database needs to be loaded into memory at a given moment.

5. Conclusion

We have shown a method for self localization along a path. After a human guided experience, we compute a map of the environment in an off line learning step. With this map, the robot is able to compute its pose. The accuracy of our algorithm was measured using a DGPS as the ground truth. The mean localization error is about 15 cm in the unfavourable case of an open area where the GPS is available. Other experiments suggest that the accuracy should be increased when working in a narrow urban canyon even if we can't measure it with the DGPS. This algorithm will be used for autonomous navigation soon. The accuracy is not as good as a DGPS sensor, but our vision system has several advantages over the GPS. First it is able to provide the orientation even if the vehicle doesn't move. The price is greatly reduced as only a camera and a laptop computer are required. And more important, our algorithm is better suited to narrow streets and city centers where GPS signals are not available. The localization algorithm runs at the video frame rate (30 Hz). We plan to integrate a better motion model in order to have a better initial estimation of the camera pose. This should reduce the time used for matching and RANSAC.

References

[1] H. Araújo, R.J. Carceroni, and C.M. Brown, "A fully projective formulation to improve the accuracy of

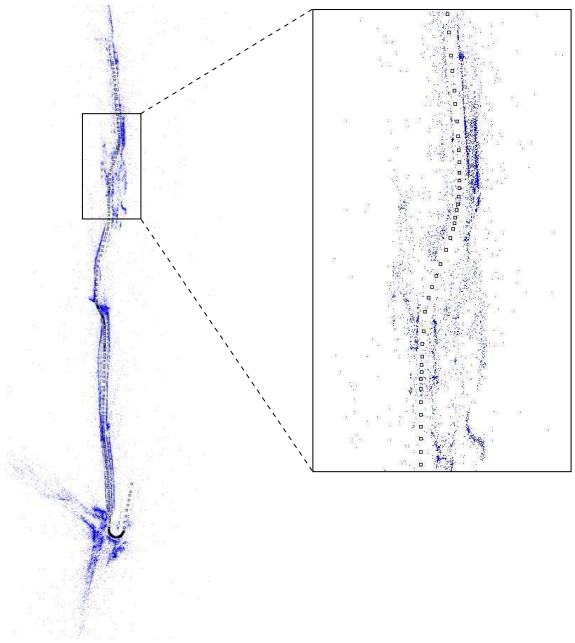


Figure 9. 3D reconstruction of a large sequence (500m long). Overview of the reconstruction on the left and a more detailed part on the right



Figure 10. A few images from a large sequence recorded in an urban area

Lowe's pose estimation algorithm", *Computer Vision and Image Understanding*, 70(2):227-238, 1998.

[2] P. Beardsley, P. Torr and A. Zisserman, "3D Model acquisition from extended image sequences", *European Conference on Computer Vision*, pp 683-695, April 1996.

[3] A. Davison, "Real-time simultaneaous localisation and mapping with a single camera", *International Conference on Computer Vision ICCV'03*, pp 1403-1410, 2003.

[4] M. Fischler and R. Bolles, "Random Sample Consensus: a Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography", *Commun. Assoc. Comp. Mach.*,24:381-395, 1981.

[5] R. Haralick, C. Lee, K. Ottenberg, M. Nolle, "Review and analysis of solutions of the three point perspective pose estimation problem", *International Journal of Computer Vision*, 1994.

[6] C. Harris, M. Stephens, "A Combined Corner and Edge Detector", *Alvey Vision Conference*, pp 147-151, 1988.

[7] R. Hartley, A. Zisserman, *Multiple view geometry in computer vision*, Cambridge University Press, 2000.

[8] O. Faugeras and M. Hebert, "The representation, recognition, and locating of 3-d objects", *International Journal of Robotic Research*, MIT Press, vol. 5, No. 3, pp 27-52, 1986.

[9] S. Jeon, B. Kim, "Monocular-based Position Determination for Indoor Navigation of Mobile Robots", *In Proc. of the 1999 IASTED international conference*, 1999.

[10] K. Kidono, J. Miura, Y. Shirai, "Autonomous Visual Navigation of a Mobile Robot Using a Human-Guided Experience", *Robotics and Autonomous Systems*, Vol. 40, Nos. 2-3, pp 124-1332, 2002.

[11] Y. Matsumoto, M. Inaba, H. Inoue, "Visual Navigation using View-Sequenced Route Representation", *In Proceedings of IEEE Conference on robotics and Automation*, pp. 83-88, 1996.

[12] D. Nistér, "An efficient solution to the five-point relative pose problem", *2003 Conference on Computer Vision and Pattern Recognition*, volume II, June 2003.

[13] D. Nistér,O. Naroditsky, J. Bergen, "Visual odometry", *2004 Conference on Computer Vision and Pattern Recognition*, Vol. 1, pp 652-659, 2004.

[14] M. Pollefeys, R. Koch and L. Van Gool, "Self-Calibration and metric reconstruction in spite of varying and unknown internal camera parameters", *International Conference on Computer Vision*, pp 90-95, 1998.

[15] A. Remazeilles, F. Chaumette, P. Gros, "Robot motion control from a visual memory", In IEEE Int. Conf. on Robotics and Automation, ICRA'04, Vol. 4, pp 4695-4700, April 2004