3D Reconstruction Report

Group 2 VT 2013

Version 1.0

Status

Reviewed	Group 2	2013–05–16
Approved		

2013–05–16

Project ChubbyDino

Computer Vision, VT 2013, Group 2
Department of Electrical Engineering (ISY), Linköping University

Participants

Name	Responsibilities	Phone	E-mail
Gustav Häger	Visualization	070–649 03 97	gusha124@student.liu.se
Alexander Sjöholm	Point extraction	076–225 11 74	alesj050@student.liu.se
Martin Svensson	Nonlinear optimization	070–289 01 49	marsv106@student.liu.se
Mattias Tiger	Pose estimation	073–695 71 53	matti166@student.liu.se

Project guide: Fahad Khan, Linköping University, fahad.khan@liu.se
Course leader: Per-Erik Forssén, 013–28 56 54, per-erik.forssen@liu.se

Contents

List of Figures

Document history

Version	Date	Changes	Sign	Reviewed
0.1	2013–04–29	Initial draft	MS	
1.0	2013–05–16	Final Report	Group 2	

1 Introduction

This document constitutes the final documentation of a project performed as a part of the course Computer Vision (TSBB15) at Linköping University. The purpose of the project is to implement a computer program capable of, given a set of images and known intrinsic camera parameters, estimate the location of image points in 3D as well as the camera pose. 3D reconstruction is, in computer vision, defined as the process of capturing shapes, locations and/or appearances of real objects. The project goal was to create a 3D-reconstruction application capable of performing the aforementioned tasks as well as estimating the camera trajectory. The application was implemented in C++ mainly using the OpenCV API.

The purpose of this document is to provide a thorough description of how the application was created and the problems encountered during the implementation work.

2 System description

The system implementation is divided into four main modules, Feature extraction, Pose estimation, Non-linear optimization and 3D visualization.

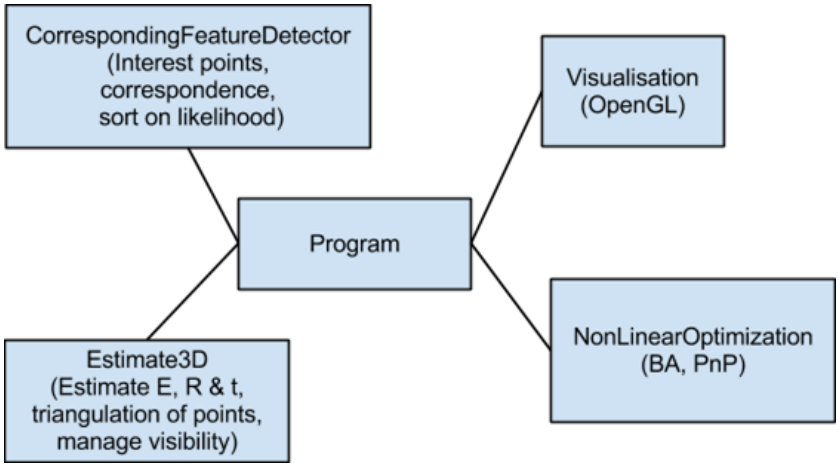

Figure 2.1: *Modules of the system*

The first step is feature extraction using the Harris operator, the pairwise correspondences is then computed using SIFT.

Then an iterative process is performed cycling though all camera-pairs, each consisting of two parts. The first part contains an initial camera pose estimation, of the new camera, using the essential matrix estimated from F using the Gold standard algorithm. The camera pose is then improved using PnP on all previously known 3D-points. An initial 3D-point triangulation is then calculated. The second part is a bundle adjustment over all current 3D points and cameras by the means of a non-linear optimization.

When all camera poses and estimated 3D-points have been iteratively added and optimized the result is rendered using OpenGL.

2.1 Main Program

The main program is composed of three main parts. The 2D correspondence extractor, the 3D reconstruction and pose estimation pipeline and the visualization module, for more information about each module, see their respective sections.

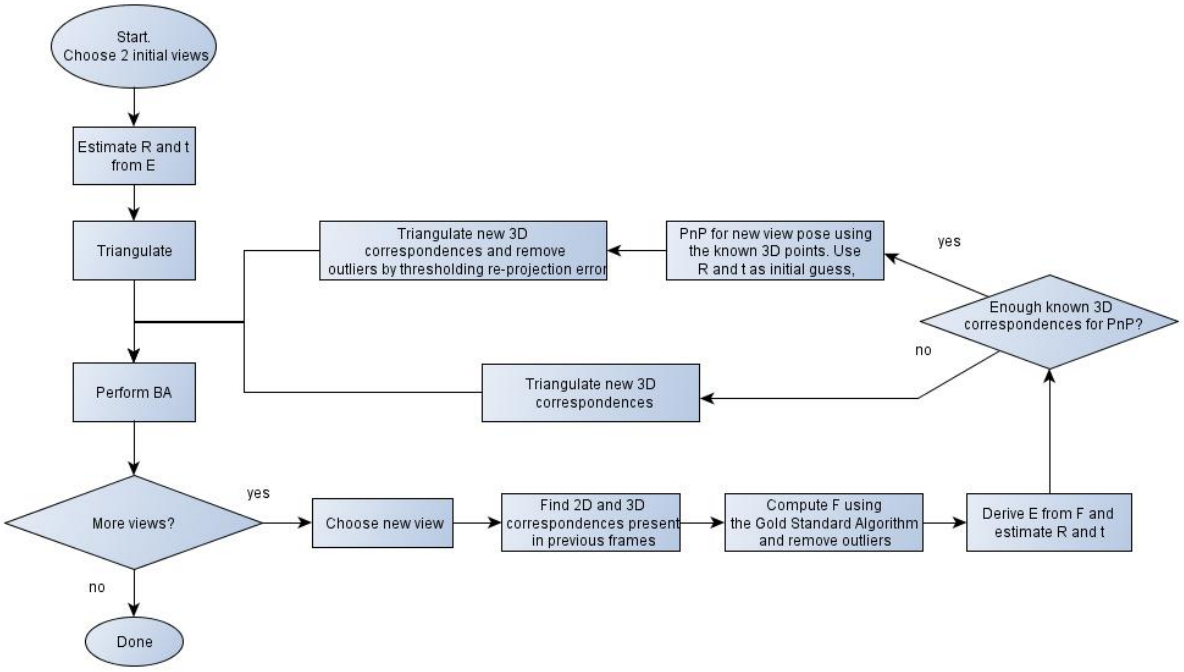

Figure 2.2: *Data flow between modules.*

2.1.1 3D reconstruction pipeline

The 3D reconstruction pipeline is where the image point pairs are used to triangulate 3D point estimates and estimate the camera poses (rotation and translation) relative to the first camera. A rough flowchart of the main program is included below.

Data from the reconstruction is saved in the clear text .alx format after adding a new camera, and after bundle adjustment for this camera. iterationX.1.alx for the former and iterationX.2.alx for the later.

2.2 Data structures

In order to keep a good structure on the program and a logical separation of functionalities, several different data structures have been created. The main thought when designing the data structures was to enable easy access to all needed data for each stage in the pipeline each with different requirements and needs. The interfaces from Estimate3D provide such access to different arrangement of the same data, such as the 3D points, which have to be possible to both access and modify in several different steps of the pipeline.

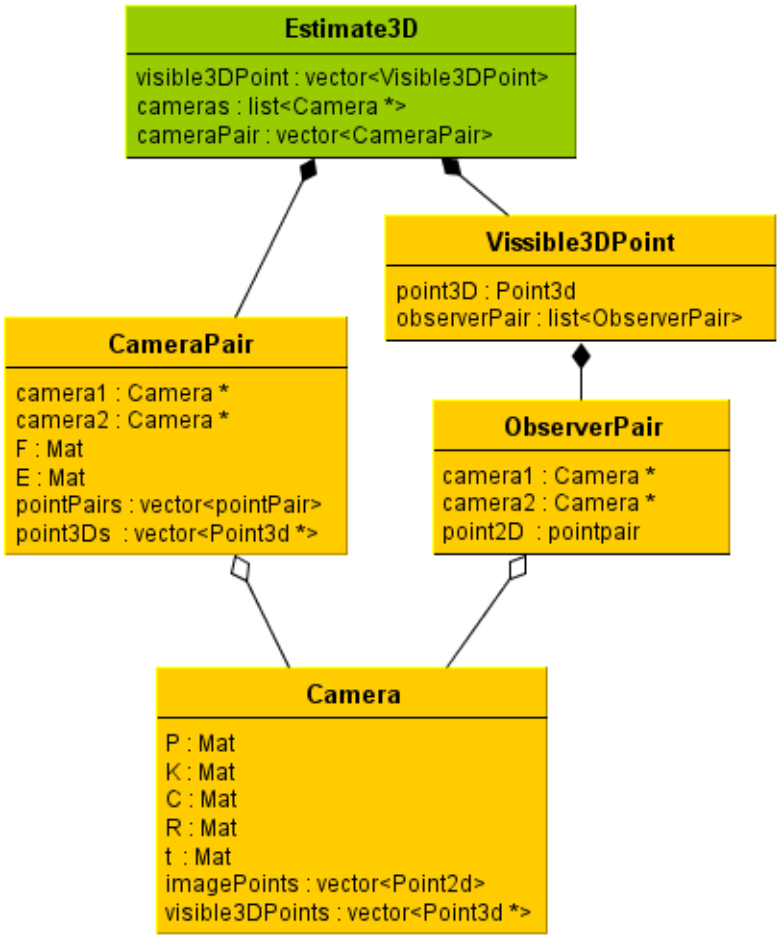

Figure 2.3: *UML diagram of the main data structures. All estimated 3D-points of the model are shared between the necessary data structures, allowing for efficient updating of the 3D-points as well as efficient extraction of the most interesting data constellations used by Gold Standard, BA and Visualisation. Estimate3D act as the common interface to all data concerning the 3D model. It is a 'visibility function' of sorts for each camera, for each camera pair as well as for which cameras and their image coordinates observing a given 3D-point.*

2.2.1 PointPair, ObserverPair

The point pair structure simply consists of two point pairs (cv::Point2d) from two different images that correspond to the same 3D point and the ObserverPair class contains one PointPair as well as pointers to the specific Cameras (see below) from which the PointPair originates.

2.2.2 Visible3DPoint

This structure represents a point in 3D space. Apart from the coordinates for the point a list of all PointPairs observing it is stored as well. 2.3.

2.2.3 Camera

The camera class represent a view of the 3D scene (an image). It contains information about what 3D points are visible in the image, and which image points these correspond to. Apart from this all necessary information about the camera itself is also stored here, such as internal parameters (K-matrix), rotation matrix and translation vector of the camera relative to the first camera is also stored in this class.

2.2.4 CameraPair

The camera pair is a data structure containing the Fundamental and Essential (F and E) matrices relating two cameras to each other. Beside this information the structure also contains pointers to each respective camera as well as vectors containing related PointPairs and pointers to corresponding 3D points.

2.3 Correspondence Extractor

To get this machinery going it is necessary to find points in the images that are interesting to look at and follow through out the sequence. It is the correspondences between points in different images that will provide the information needed to reproduce the sequence in 3D.

2.3.1 Feature Extractor

The feature extraction was carried out using Harris feature detector. This method uses the local structure tensor to sort out interesting parts of the image according to (2.3.1). This will generate a function where the value in each point is the level if interest in that local region. To select points one can threshold this at an appropriate level and morphologically shrink it to points. Besides the level of threshold one may also be interested in the closest distance between feature points.

To be able to estimate the fundamental matrix of an image pair it is important to have known points all over the images. It is however not necessary to have more than seven corresponding points to calculate minimal estimate the fundamental matrix, also the calculation complexity will be reduced with fewer points. This motivates the finding few, properly spread out feature points.

$$C_{Harris} = det(A) - \kappa \cdot trace^2(A), \qquad \kappa \approx 0.04 \tag{2.3.1}$$

$$A = \begin{bmatrix} \langle \frac{\partial I}{\partial x} \rangle^2 & \langle \frac{\partial I}{\partial xy} \rangle \\ \langle \frac{\partial I}{\partial xy} \rangle & \langle \frac{\partial I}{\partial y} \rangle^2 \end{bmatrix}, \qquad I = image \tag{2.3.2}$$

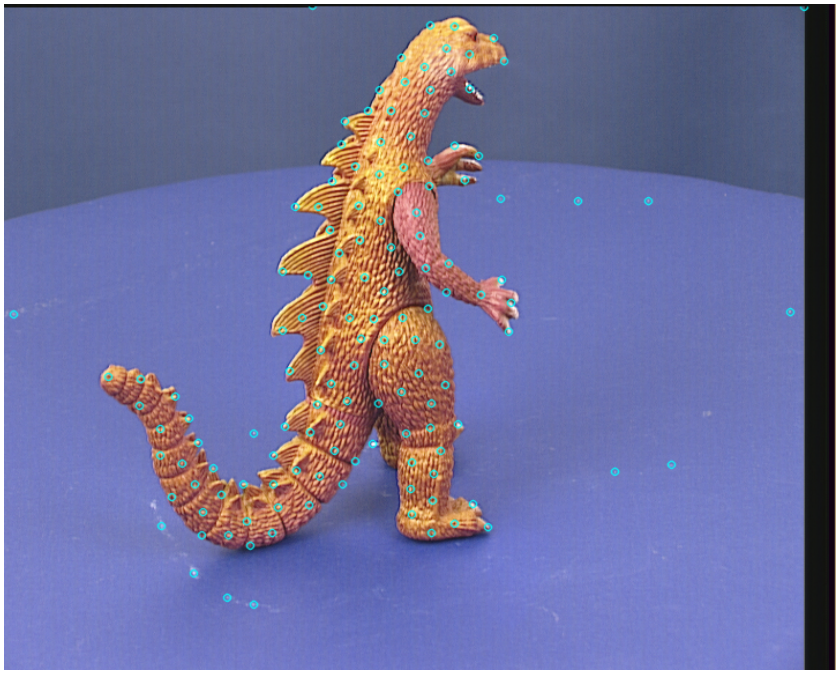

Figure 2.4: *Feature points indicated with small circles. Notice the even distribution on the dinosaur body.*

2.3.2 Descriptor Extractor

For all of the detected feature points a descriptor is calculated. This is done to be able to compare the features in different images. In this project the SIFT (Scale Invariant Feature Transform) descriptor is used. It is very effective in reducing unwanted effects of changes in illumination, scale and rotation. The way the descriptor work is that it calculates a series of histograms from gradient magnitude and orientation values from a small region around the feature point. The histograms are stored in a high dimensional vector, that once it has been normalized is the finished SIFT descriptor.

2.3.3 Matching

Finally, to find the correspondences between feature points in different images, the SIFT descriptors are compared between all feature points in the images. The descriptors that are most alike are chosen as a pair. There is also a check done to see that the found match is the closest possible match. This is done to remove potential outliers. A final check to get rid of matches that still slip through the first test is done in a euclidean distance thresholding. Points are not allowed to match on a to great distance.

Later in the pipeline a RANSAC (Random Sampling Consensus) algorithm is used to remove outliers while estimating the fundamental matrix, F, between images. The fundamental matrix defines a necessary but not sufficient constraint for corresponding feature points called the epipolar constraint, 2.3.3. If a point pair diverge to far from zero it is discarded.

$$y_1^T \boldsymbol{F} y_2 = 0 \tag{2.3.3}$$

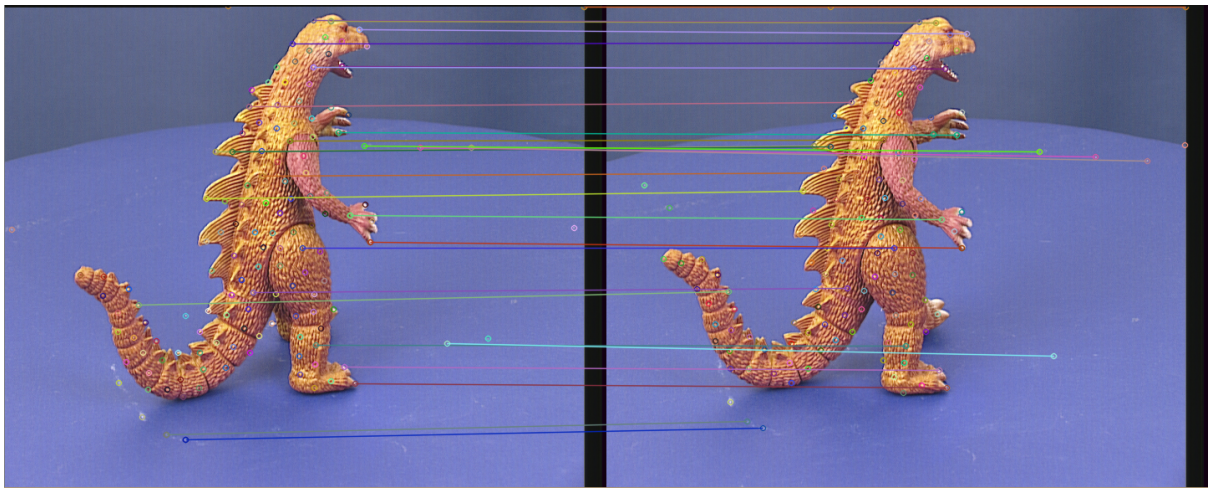

Figure 2.5: *Correspondences indicated with lines. Circles without a line attached lack correspondence.*

2.4 Pose Estimation

There are two cases of finding the pose of a camera. One is when there are only two cameras available, and the other one when there are several cameras already orientated to each other and one camera is being added to that group. In the first case an algorithm described in [5] is being used. In the second a PnP (Perspective n-Point) algorithm is used.

2.4.1 Alg. 5

Given the fundamental matrix, F, for an image pair and the internal camera parameters, K, for the camera responsible for images, it is possible to estimate the two camera poses relative one another up to an unknown scale factor. This can be done in different ways, but since the camera is calibrated and the same for all images it is convenient to use algorithm 5 in [5]. This uses a singular value decomposition of the essential matrix, (2.4.1) to find the translation, t, and rotation, $\boldsymbol{R}$, of camera two relative camera one. A minor drawback with the algorithm is that it produces four different combinations of rotations and translations that all need to be tested to find the correct one. For a more in detail description of the mathematical motivations of this algorithm see [5].

$$\boldsymbol{E} = \boldsymbol{K}^T \boldsymbol{F} \boldsymbol{K} \tag{2.4.1}$$

$$\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T = \boldsymbol{E} \tag{2.4.2}$$

$$t_1 = v3, \qquad \boldsymbol{V} = \begin{pmatrix} | & | & | \\ v1 & v2 & v3 \\ | & | & | \end{pmatrix} \tag{2.4.3}$$

$$t_2 = -t_1 \tag{2.4.4}$$

This translation has an unknown scaling yielding the possibility of both plus and minus in sign. To find the rotation an auxiliary matrix, $\boldsymbol{W}$, is defined. Also to avoid mirroring in the rotation, the suggestions are multiplied with corresponding determinant.

$$\boldsymbol{W} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4.5}$$

$$\hat{\boldsymbol{R}}_1 = \boldsymbol{U}\boldsymbol{W}^{-1}\boldsymbol{V}^T \tag{2.4.6}$$

$$\hat{\boldsymbol{R}}_2 = \boldsymbol{U}\boldsymbol{W}^{-T}\boldsymbol{V}^T \tag{2.4.7}$$

$$\boldsymbol{R}_1 = det(\hat{\boldsymbol{R}}_1)\hat{\boldsymbol{R}}_1 \qquad \boldsymbol{R}_2 = det(\hat{\boldsymbol{R}}_2)\hat{\boldsymbol{R}}_2 \tag{2.4.8}$$

$$\boldsymbol{C}_{ij} = (\boldsymbol{R}_i | t_j) \tag{2.4.9}$$

To find out which combination to chose, one corresponding point is chosen from the image pair. This point is then triangulated for the four camera pairs. Three of these triangulations should end up with the triangulated 3D behind the camera leaving only one camera left which is the one we choose.

2.4.2 PnP

The PnP algorithm is used to fit an external camera to an existing set of cameras with corresponding triangulated 3D points. The algorithm tries to adjust the new camera's rotation and translation to minimize the re-projection error of the existing 3D points that by 2D correspondence are known to exist in the new camera as well. In the this procedure there is an optional outlier removal part where points producing too large re-projection errors are discarded from future 3D reconstruction.

2.5 Non-linear Optimizer

There are three points in the reconstruction pipeline where nonlinear optimization is performed. In all three cases the optimization is performed using the levmar API [2]. Levmar uses the Levenberg-Marquardt non-linear optimization algorithm to optimize a specified residual function, in this case reprojection errors of 3D points. The reprojection error is the distance, in pixels, between the projected 3D point and the corresponding point in the image plane. There are, in the implemented 3D-reconstruction pipeline, three instances where these highly non-linear functions are optimized over up to one thousand variables. With the Levmar API only accepting a residual function that take double arrays as both in and outputs, considerable preprocessing of the data is required.

2.5.1 Rotation Parametrization

In order to make it possible to find the optimal camera poses, the rotation matrices have to be parametrized. In this application they are parametrized using OpenCVs built-in Rodrigues' function, both in the PnP pose estimation and the bundle adjustment. This is a vector representation with the rotation angle described by the norm of the vector. The only ambiguity is the modulo 2π on the vector norm (ch.10.8 in [5]), and by providing good initial guesses for the pose estimation, this is not an issue.

2.5.2 Gold Standard estimation of F

The function computing the gold standard refinement step is implemented as described in alg. 11.3, step (iii) in [4]. The function takes, as an input, an initial guess of the F matrix, two projection matrices and vectors containing 2D and 3D correspondences. All input data is then preprocessed and re-arranged to fit the optimization API. The cost function described in equation (2.5.1) below is optimized over the twelve parameters of the projection matrix P_2 as well as the 3D-point coordinates, p^{3D}.

$$\varepsilon^2 = \sum_{n=1}^{N} (p_{1,n}^{2D} - \mathbf{P}_1 p_n^{3D})^2 + (p_{2,n}^{2D} - \mathbf{P}_2 p_n^{3D})^2, \qquad \mathbf{P}_1 = [\mathbf{I}|\mathbf{0}] \tag{2.5.1}$$

For more in for on how the rest of the steps in the gold standard algorithm are implemented, see section 2.4.

2.5.3 Solving the PnP

In the PnP the camera pose (rotation and translation) is estimated by minimizing the re-projection error of the visible 3D points.

In order for the data to fit the levmar interface and minimize the number of parameters the rotation matrices of each view is parametrized, as mentioned above, using the OpenCV Rodrigues' parametrization. Rotations and translations for each view are then stacked in a vector, along with all currently available 3D points. The second step is to make sure the residual function has access to all needed data, for example the visibility function and point correspondences.

The resulting cost function, to be minimized over $\mathbf{R}$ and $\mathbf{t}$ is

$$\varepsilon^2 = \sum_{n=1}^{N} (p_n^{2D} - \mathbf{K}\mathbf{C}p_n^{3D})^2, \qquad \mathbf{C} = [\mathbf{R}|\mathbf{t}] \tag{2.5.2}$$

2.5.4 Bundle Adjustment

Th largest and most time-consuming use of the non-linear optimizing module is in the bundle adjustment step of the 3D-reconstruction pipeline. The bundle adjustment function refines the estimates of both camera poses (rotation + translation) and location of 3D points for all views.

The residual function simply, for each view, calculates the re-projection error of all 3D points visible in that view, i.e the pixel distance of the reprojected 3D points to the actual image points used in the triangulation. In the equation below, V denotes the number of views, N_v the number of points visible in each view, and $p_{v,n}$ the corresponding image points. This expression is then minimized over all p^{3D} as well as all rotations and translations.

$$\varepsilon^2 = \sum_{v=1}^{V} \sum_{n=1}^{N_v} (p_{v,n}^{2D} - \mathbf{K}\mathbf{C}_v p_{v,n}^{3D})^2, \qquad \mathbf{C}_v = [\mathbf{R}_v|\mathbf{t}_v] \tag{2.5.3}$$

The number of outputs from the residual function scales rapidly with the number of views, as each view gives an additional $6 + 2N_v$ observations, where N_v is the number of 2D inliers in view v.

2.6 3D Visualization

This module visualizes the calculated points and the camera positions relative to the position of the first camera. This part of the program is written on top of the OpenGL graphics API.

It can be run as a stand alone program, or as a last step in the 3D-reconstruction pipline. In the standalone case it will read the output files generated by the reconstruction pipeline. Using the keyboard a user can then step between the added cameras, with or without the bundle adjustment step done.

It is also possible to alter the scale of the points and cameras, as there is no automatic scaling in place and the points are drawn as colored spheres.

2.6.1 Visualization

The estimated 3D-points are plotted as textured/colored spheres, using the color from the image where the corresponding interest point was first detected, or a region surrounding the point.
Cameras are plotted as cones pointing toward the point cloud.

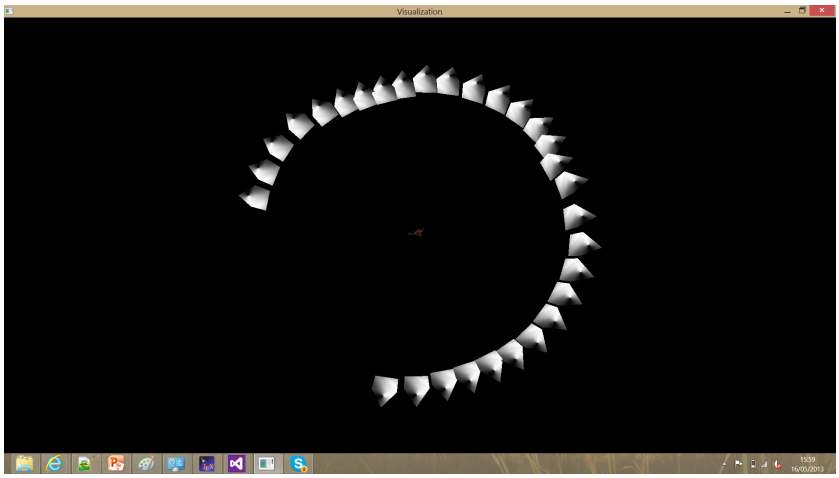

Figure 2.6: *Camera positions*

Using only a few cameras results in very poor depth accuarcy for the plotted points, as seen in 2.7

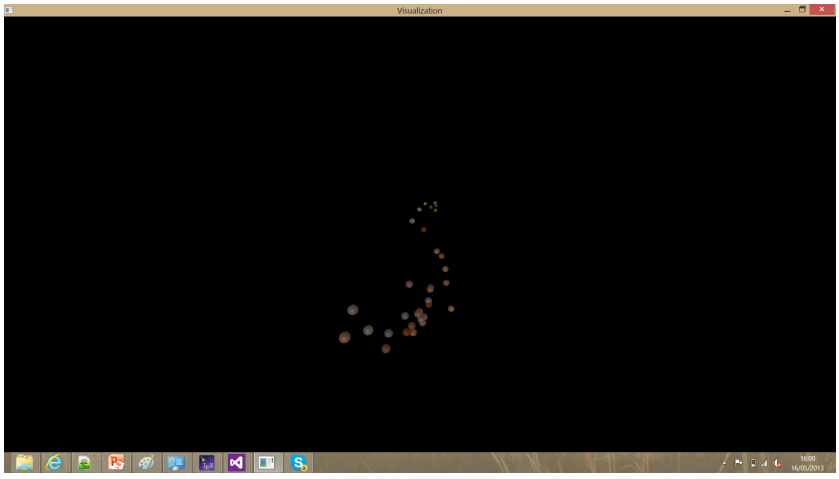

Figure 2.7: *The dinosoaur with only a few cameras, very poor depth accuracy*

Using more cameras will make the resulting point cloud become more dense, as the depth accuracy improves.

Points located on the table rather than on the dinousar will be correctly triangulated to lie in a plane, at the same level as dinosaur.

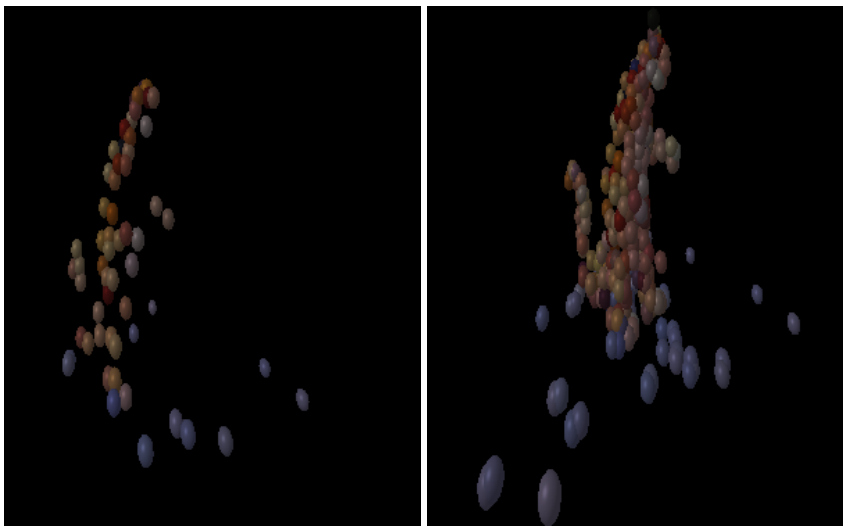

Figure 2.8: Point cloud at 2 cameras, and at 26 cameras

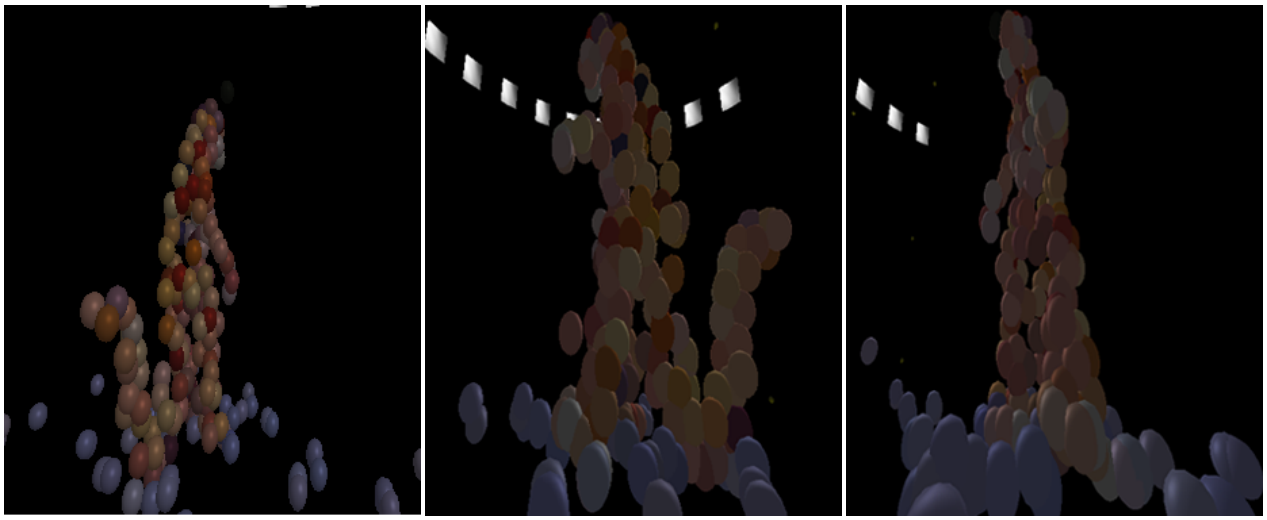

Figure 2.9: *The ground plane can be seen clearly, points where the ground had marks utilised by the point extractor. In the back some of the cameras can be seen, slightly distorted by the frustum used in the visualization.*

2.6.2 File managing

Each iteration of the reconstruction pipeline produces two files, one with the reconstruction state before the bunle adjustment and one after. The files contain all detected 2D correspondances, the calculated fundamental matrix, and the known 3D points. The data is saved as text, to make inspection easier.

3 Results

A necessity for a system like this one to work in practice is for it to be robust against outliers. The difficulty in finding and removing these outliers have been the most challenging part of this implementation.

3.1 Robustness

After applying outlier removal both in the gold standard estimation, where outliers are removed by thresholding on the epipolar line distance, and before the bundle adjustment where known 3D point correspondences from the previous frame are used as a consensus set, the implementation became robust against outliers and manages to handle the data sets tested with good enough point correspondences.

3.2 Computational load

The non-linear optimization implementation performs well in the gold standard F-matrix estimation and the pose estimation, however, after around ten frames the bundle adjustment speed slows down significantly for a number of reasons. The dinosaur set for example, (36 images) with the SIFT extractor tuned to generate around 50 points per image, iteration speed slows down significantly after around ten iterations, and the whole set takes several hours to complete. Besides the slow speed there is nothing wrong with the non-linear model and it works fine on all the data sets tried out so far.

4 Conclusions

In general this project needed much better routines for testing and verification of indvidiual modules than the previous project, mainly due to far longer computational times involved.

4.1 Correspondence Extractor

It is important to do the correspondence extraction well, as this data is the basis for all further calculations. There are a couple of parameters that need to be tuned properly for every sequence. The features need be evenly distributed in the image the get enough information. It is for instance not good to have all features in a plane or many feature close to each other. This is solved with a minimum distance allowed between features. It is also important to look for features of the right size. This depends on resolution and image structure. To keep the number of features low and the quality high helps the program run faster.

4.1.1 Possible improvements

Automatic parameter detection based on image analysis in some way would make the program more user friendly. At the moment it requires an experienced user.

4.2 Pose Estimator

The pose estimator is very sensitive to the quality of the correspondences. It does not handle outliers or when all points are on a plane very well. This corrupts the estimation completely since it cannot provide even rough initial estimates to the bundle adjustment optimization. However, as long as the outliers are removed and the points are spread out in more than two dimensions it works very well.

Deducing which 3D-points were seen in the previous image it was possible to run PnP on the known 3D-points in order to improve the pose of the new camera significantly. Using this result it was possible to remove most problematic outliers by triangulating the new points and to threshold them according to their re-projection error. This both improved the result and increased the speed of the computations in the BA step as it was given a better initial estimate.

4.2.1 Possible improvements

If something about the camera trajectory is known one could possibly constrain the pose estimation and make it more robust.
Another possible improvement would be to find chains of corresponding points though out the image sequence which then should correspond to the same 3D-point. The goal would be to find as long chains as possible while still always having at least a minimum of image points in each image. Their points could then be cross-validated for each image and the worst points removed from the chains resulting in a split of the chain. It would possibly provide very high-quality estimates of trajectories of 3D-points captured in the image sequence. These chains could then be used in the pipeline resulting in a very sparse 3D-model with possibly very high camera pose estimation accuracy. The next step would then be to triangulate most other image points using these ”known” cameras and building up a dense 3D-model. Additional verifications, validations and re-estimations of the camera-poses might be done in order to ensure that the estimated cameras poses are indeed very accurate. The speed up from this process could be very large, as well as the quality of the model perhaps!

4.3 Non-linear Optimizer

The non-linear optimization implementation performs well and no significant problems related to neither the pose representation, nor the representation of points or the visibility function were struck. The hardest part was to figure out how to convert all the data to fit the API, but once that was figured out, the implementation was very straight-forward, and the bundle adjustment part was not much harder to implement than the Gold Standard or PnP parts.

4.3.1 Possible improvements

The improvement that would yield the largest performance increase in the bundle adjustment would be making use of the lack of independence between views and drastically lower the computational load. The original thought was to use the sparseLM API [3]. Unfortunately, due to lack of time, this was never implemented as it would require the representation of the visibility function to be remade completely.

A second significant improvement would be to make use of the LAPACK API, that was highly recommended for use in combination with levmar, and said to speed things up significantly. It was, however, rather complicated and time consuming to set up, and these are the reasons it is not used in this implementation.

4.4 3D Visualization

Implementing this part of the system in OpenGL was perhaps not the wisest decision, as the amount of work and difficulty to debug OpenGL calls made progress rather slow.

It did however make it easy to implement project-specific functions into the visualizer without having to learn yet another API.

4.4.1 Possible improvements

It should be possible to color the points according to the image data they are extracted from. If more points where obtained it should be possible to draw them only as colored dots in space.

Further one might create a mesh between the detected points, to generate a full 3D-Model. It should then also be possible to texture it using data from the images to create a complete 3D representation of the original physical object.

Another improvement would be to add runtime settings to many of the options that are currently set in the source code, as recompiling the program every time the scale of the objects need to change is somewhat clumsy.

References

[1] Sonka, M., Hlavac, V. & Boyle, R.
Image Processing, Analysis, and Machine Vision.
Toronto: Thompson Learning,
cop. 2008, 3rd ed.,
ISBN 0495244384.

[2] Lourakis, M.I.A.
levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++
`http://www.ics.forth.gr/~lourakis/levmar/`,
Jul. 2004
Accessed on May 14th 2005.

[3] Manolis I.A. Lourakis
"Sparse Non-linear Least Squares Optimization for Geometric Vision,"
European Conference on Computer Vision,
vol. 2, 2010, pages 43-56
DOI `http://dx.doi.org/10.1007/978-3-642-15552-9_4`

[4] Hartley, R & Zisserman, A
Multiple View Geometry in Computer Vision.
Cambridge University Press, West Nyack, NY, USA
March 2003, 2nd ed.
ISBN 978–05–11–18711–7

[5] Nordberg, K
Introduction to Homogeneous Represenations and Estimation in Geometry
Apr. 2013
Computer Vision Laboratory, Department of Electrical Engineering
Linköping University