Sparse 3D Reconstruction From A Set Of Images

Vardan Papyan
Computer Science Faculty
Technion

Batchen Golden
Computer Science Faculty
Technion

October 30, 2013

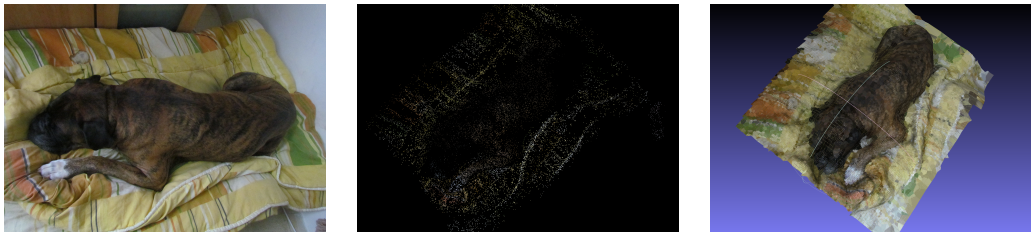
Directed by: Alex Kreimer & Professor Ehud Rivlin

1 Introduction

The goal of this project is to create a sparse 3D reconstruction of a scene from a series of images of it. To accomplish that, some assumptions must be made:

- The camera holds the pinhole model.
- Only the camera is moving, while the scene is static.
- All the images are taken with the same camera. More specifically, if the pictures are taken with different cameras, they must have the same focal length.
- All images have the same resolution

Other then this demands there are no additional requierments of the images or the scene. Specifficaly, we do not demand the set of images to have any particular order.
In our work we have relied on a code written by Roy Shil [12]. The first section will present the basic algorithm as implemented in the original code. Then, in section 3, we will talk about a theory presented in [1] and explain how we've tested it and how it helped improve the basic algorithm. The next sections, 4-6, describe the process we went through while doing the project. Each of this sections present a different challenge we have encountered and what changes it made us do. In sec. 7 we will explain how our algorithm can be exploited also for egomotion extraction, even though its not its original purpose. Finally, in sec. 8, we will present the graphic user interface we have created in order to make it easy for anyone to experiment with the algorithm.



Original image **Sparse reconstruction** **Dense reconstruction**

Figure 1: Example of final results- The image in the middle is a sparse reconstruction of Benny the dog, which is the outcome of our algorithm. The image on the right is a dense reconstruction of the dog done using the external software [2], which received as input the result of our algorithm.

2 Algorithm Description

This section describes the outline of the basic algorithm, as implemented in [12], before any improvements were made.

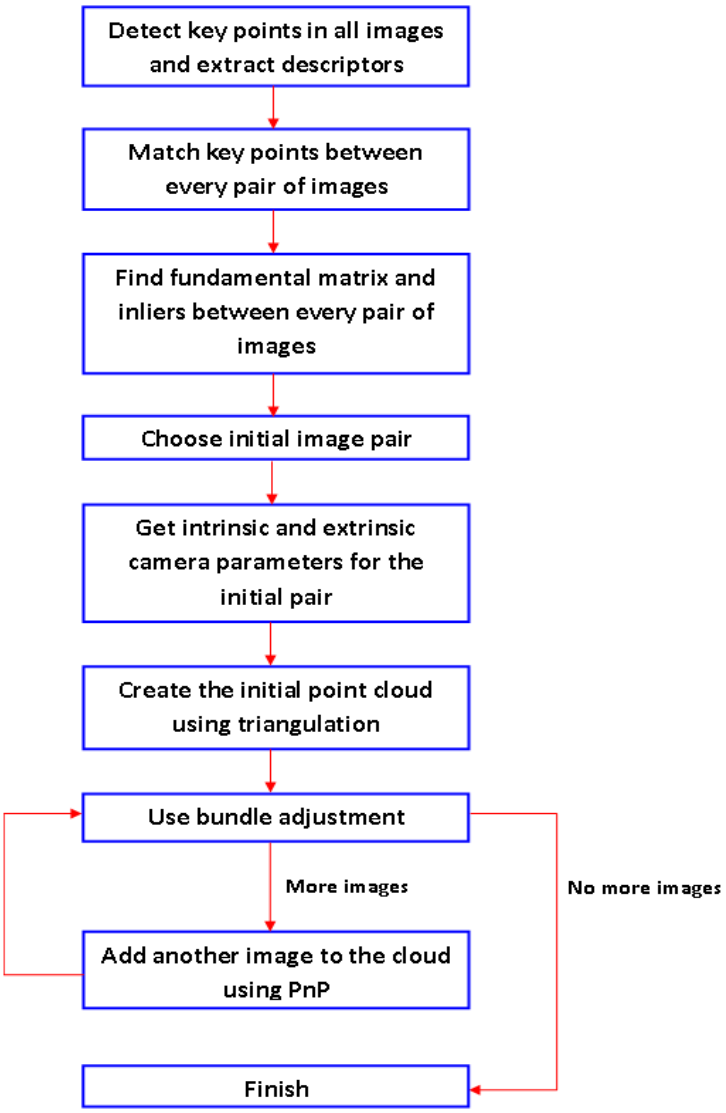


Figure 2: Algorithm outline

2.1 Detect key points and extract descriptors

The first step of the reconstruction process is to detect key points, also known as features, in each image. Key points are pixels in the image which are "interesting", meaning they have unique properties and are likely to be identified as key points also in different pictures of the same object. The process of feature detection is at the bottom of many computer vision algorithms, and as such, many methods have been developed for it. Each of these methods has its own advantages and disadvantages, and it is up to the programmer to find the one most suitable for the specific purpose of the algorithm. For the initial configuration of the algorithm we chose Opencv's PyramidFAST. Later on, this choice was changed several times in order to achieve different things (see sections 4-6 for more details).
After the key points have been detected, descriptors must be extracted. A key point descriptor is a vector containing useful information about that point to be used in the next stage, when trying to decide whether two key points are the same. For our algorithm we chose to use ORB descriptor extractor.

Original image **Sift features** **PyramidFAST features**

Figure 3: Comparison of the two methods used for detecting features. As you can see, the sift method (image in the middle) gives us a significantly low amount of features. The PyramidFAST method (image on the right) on the other hand, gives us plenty of features, but some of them aren't unique at all and probably won't be useful (for example, all the points on the brown wall).
Notice how neither of the methods detect points on smooth areas, such as the yellow parts of the image, and therefore this parts won't appear in any reconstruction based on this matches.

2.2 Match key points

In this stage we look at each pair of images and we try to find pairs of corresponding points. Two key points should be considered a match if they both are projections of the same 3D world point (for example, the two pixels showing the corner of a table in both images should be a match). The decision whether two key points are a match or not is based on the information found in their descriptors from the previous stage.
Like feature detection, feature matching also has many implementation methods and we need to choose the method most suitable for us. Choosing the wrong method may result in too few matches (which will lead to a very sparse and unrecognizable reconstruction) or too many false matches (which may even result in a complete failure of the reconstruction!). We found Opencv's Brute-force descriptor matcher to be good for our purposes.

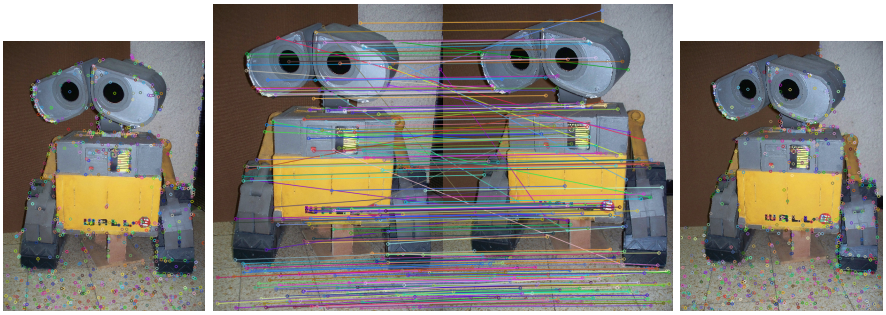


Figure 4: The lines in the middle picture connect features from the left image to their corresponding features in the right image. The pictures on both sides show all found features in both images. It's clear that not all features have a match, some because they simply don't appear in the other image (for example the grid on the right of the left image). Also, some matches are clearly false (mostly the lines which aren't horizontal).

2.3 Find fundamental matrix and inliers

After we found a set of matching key points for each pair of images, we now need to use it to find the fundamental matrix between each pair. A fundamental matrix is a $3*3$ matrix which is very important in the world of computer vision. It holds the following constraint:

$$x'^T F x = 0$$

where x and x' are the image coordinates (represented homogeneously) of corresponding points in an image pair. Our goal would be to find F solving the equation system obtained from this constraint applied to all matches. The problem is, not all the matches we found are good, and if we take them into

consideration in our calculations the resulting F will be wrong.
The solution to this problem is the RANSAC algorithm, which is a method for iteratively estimating parameters of a mathematical model given a set of observed data which may contain errors. In our case, the RANSAC algorithm is given the list of matches found in previous stages and finds a fundamental matrix which holds the equation for maximum number of matches. The result of the algorithm is the estimated F matrix and a list showing for each match whether it holds the equation for the final estimated F (in which case its called an inlier) or not (in which case its called an outlier).
Once RANSAC finished running we remove all matches classified as outliers and proceed only in case we have sufficient inliers left (inliers percentage is above some threshold).

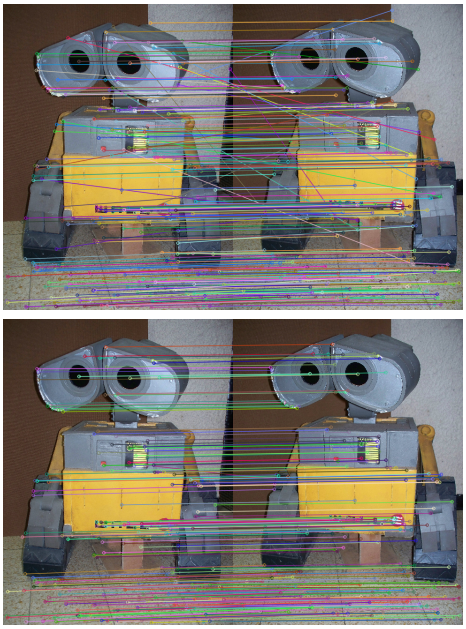


Figure 5: A comparison between feature matches before (on top) and after (on the bottom) inliers filter. As you can see, using the fundamental matrix we were able to remove most of the false matches, and now only horizontal lines remain. This may come with a price- some of the good matches may be falsely tagged as bad, and we'll lose them in the process.

2.4 Choose initial image pair

Once we have found the fundamental matrix for each pair of images, we can now start creating our 3D point cloud. Since the reconstruction is done a pair

at a time, we first need to choose a pair of images from which to build our initial cloud. This may seem like an easy task, but we found these seemingly arbitrary choice to have crucial effect on the final result. Eventually, we choose to use a method which chooses two images taken from positions not too close to each other, but which still have enough key point matches between them.
For more details about the method used, the reasons for using it and the consequences of choosing a different initial pair see section 3.

2.5 Get intrinsic and extrinsic camera parameters

At this point, the only thing missing to begin the 3D reconstruction are the camera's intrinsic parameters (focal length and central point) and extrinsic parameters (rotation and translation in 3D space) for both images in our initial pair.
First we note that one of our algorithm requirements is that all images have the same intrinsic parameters. But, our algorithm does not require to know these intrinsic parameters. Instead, we use an initial guess and then refine it using bundle adjustment (see sec. 2.7). Our initial guess for the focal length is max(width_of_image,height_of_image) and for the central point is (0.5*width_of_image,0.5*height_of_image).
For the extrinsic parameters, we first notice that what we actually need is the relative position of the second camera in respect to the first camera, and the absolute rotation and translation of the first camera can be set arbitrarily to $(0°, 0°, 0°)$ and $(0, 0, 0)$. To find the relative position of the second camera we recover the essential matrix using the previously found F matrix, the initial guess for the intrinsic parameters, and the connection:

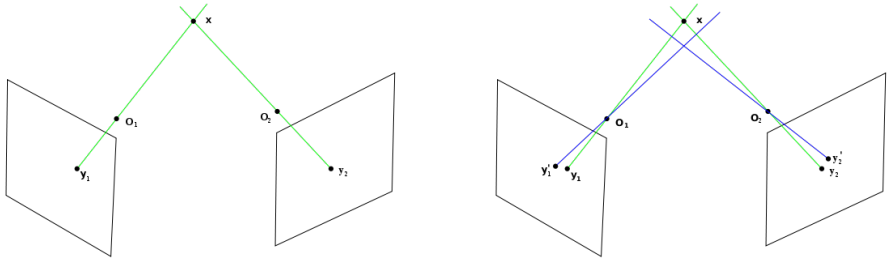
$$E = K'^T F K$$

where E is the essential matrix, K is the intrinsic matrix and F is the fundamental matrix. After recovering E a SVD (singular value decomposition) of it can be used to determine the rotation and translation between the two images.

2.6 Create the initial point cloud using triangulation

Finally, we can start reconstructing the original 3D world points from their 2D image projections in our initial image pair. This process is called triangulation and is done using the fact that the 3D point is the intersection point of the two 3D lines connecting the focal point to the projected point in both images. Tough in theory finding the intersection point between these two lines is basic math, in practice, our calculations are not precise, and therefore the lines may not intersect at the exact location of the real 3D point, or worst, they may not intersect at all (see figure 6). Therefore, the problem becomes finding a 3D point which is the closest (using some norm) to both lines. In our case, we solve this as a LS problem.

This triangulation process is repeated for all the matches in the initial pair, and thus creates the initial point cloud.



Ideal triangulation **Real life triangulation**

Figure 6: **Left:** x is a 3D point, y_1 and y_2 are it's projections and O_1, O_2 are the camera's focal points. In the ideal case the two green lines intersect exactly at x. **Right:** The location of the projected points is not exact and instead of y_1 and y_2 we get y_1' and y_2'. As a result, instead of the green lines we get the blue lines, which don't intersect. Note: both images are taken from [3]

2.7 Use bundle adjustment

Bundle adjustment is a general name for a tool which takes a list of cloud points, positions of cameras (translations and rotations) and intrinsic parameters of cameras and refines them by trying to minimize the reprojection error between the image locations of observed and predicted image points. There are several methods to solve this minimization problem. We chose to use SSBA (Simple Sparse Bundle Adjustment [4]), which employs a sparse variant of the LevenbergMarquardt algorithm.
This bundle adjustment process is done every time the point cloud is updated to refine it and the intrinsic and extrinsic parameters of all cameras involved.

2.8 Iteratively add images to the point cloud

At this point we have an existing point cloud, which means we have a reconstruction of a certain quality. But, the more we add images to the existing cloud, the more dense it will get, and, hopefully, the more it's quality will improve.
The process of adding points from another image to the cloud is different than the process of building the initial cloud. In this case, we have existing 3D points and if we can find their projections in the new image we can use them to determine the camera's location in relation to the previously added cameras. Lets assume the first image of the initial pair has the point p1 and the second image

of the initial pair has p2, when p1 and p2 are matching points. p1 and p2 are triangulated to a 3d point P. If we add a new image which has p3, which matches either p1 or p2, we say that there is a 3d-2d correspondence between P and p3. Using these kind of 3d-2d correspondences between the current cloud and the considered image we can find the new image's position using the PnP algorithm.
The next step is to choose the next image to join the cloud. We want the calculations to be precise and therefore we choose the one with the most 2D-3D correspondences to the existing cloud. After we chose an image, we use Opencv's solvePnP function to find it's position in relation to the cameras already presented.
Finally, using the parameters of the new camera to be added, we triangulate the points in the new image with the points in each of the previously added images, thus creating a thicker cloud. After the cloud is updated bundle adjustment is used again. This process repeats itself iteratively until there are no more images to add.

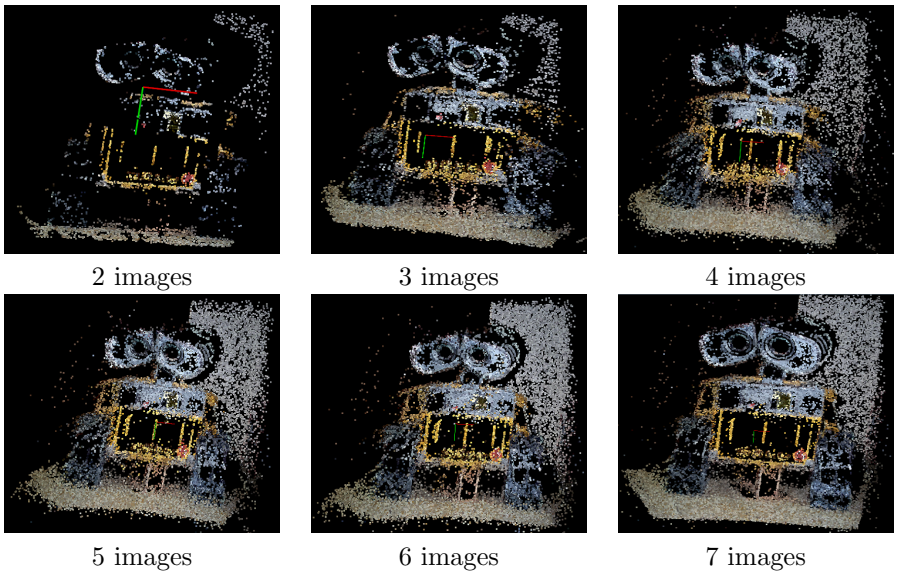


Figure 7: This sequence of pictures shows the resulting point cloud when using an increasing amount of images (the red and green lines in all pictures are there to display the cordinate system and aren't part of the reconstruction). In this example, the more we add images we indeed get a better quality for the reconstruction. Notice the contribution of bundle adjustment which rearranges the points after each iteration (it's noticeable especially if you look at the floor in the last image, which looks much more precise than the floor in the image before that).

3 Choosing The Initial Image Pair

3.1 Background

In this section we will discuss broadly about how to choose the pair of images from which to begin the reconstruction. Choosing this pair is a crucial stage in the algorithm, since the initial point cloud is built based on this pair alone.
Our initial suggestion was to choose the pair which has the most key point matches. The idea was that the thicker the initial cloud is, the more exact the following triangulations will be and the better the resulting reconstruction will be. But, results showed this is not always the case (see figure 8).

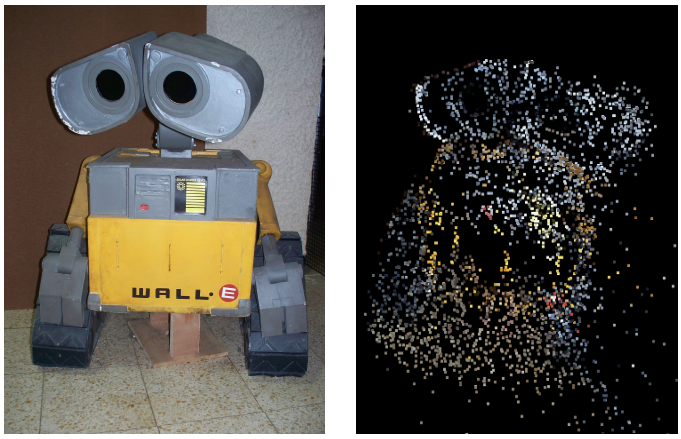
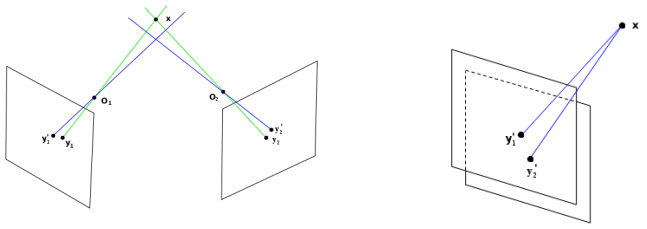


Figure 8: Example of a bad reconstruction result received when choosing an initial pair with a relatively high number of matches

To get better results we decided to try the method suggested by Snavely and Seitz in [1]. In their paper, they point out that even though we want the initial pair to have a lot in common, we should be careful not to use images too close to each other, or else ill conditioned calculations will occur during triangulation in the next stages. The reason is, the triangulation process tries to find an intersection point between the two 3D lines connecting the focal point and the projected point in both cameras, and when the images are too close to each other this lines are almost parallel (see figure 9).



general case triangulation triangulation for close images

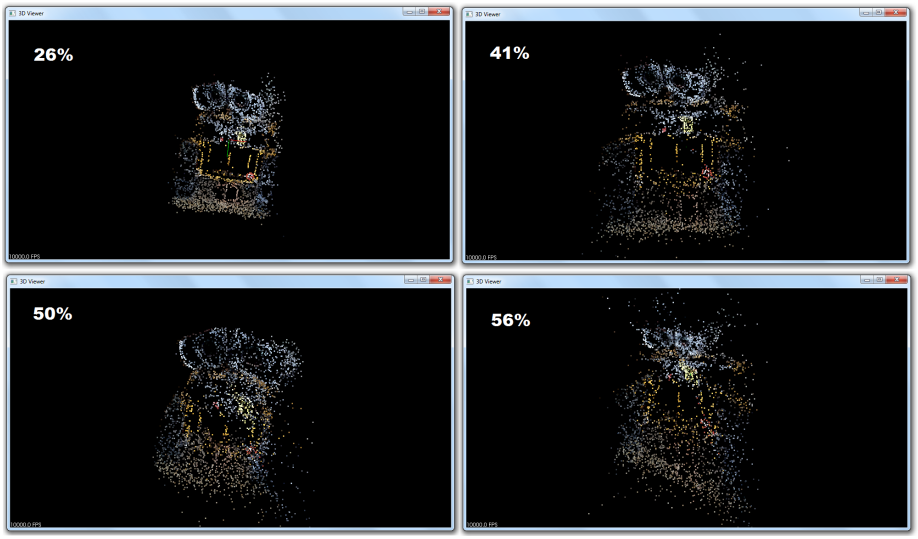
Figure 9: In the general case, the original lines (in green) are far apart, and therefore, even if the reprojected points aren't exact, finding an intersection point between the new lines (in blue) should still work pretty well. In case the images are too close, on the other hand, the lines are originally very close to each other, and so assuming some reprojection error, recovering the intersection point becomes a problem.

The method they suggested to use in order to detect whether two images are too close or not is using a homography. Assuming a pinhole camera model, any two images of the same planar surface in space are related by a homography. Their idea was that a pair of images will make a good initial pair if that pair's matches can't be modeled by a single homography. Therefore, we can easily measure how good is a pair of images by trying to find a homography between the two images and checking the inliers percentage (the higher it is - the worst that pair is).
The combination of these requirement and the fact that we still want the initial cloud to be thick enough, yields the following algorithm for choosing the initial pair:

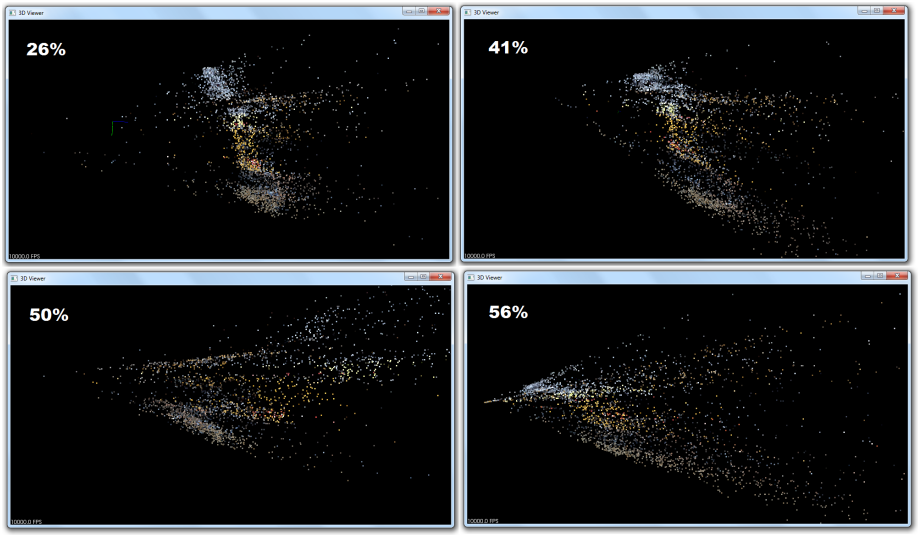
1. Find homography between every pair of images and save inlier percentage.
2. Sort the list of image pairs in ascending order of inlier percentage.
3. Choose the initial pair to be the first pair in the list having at least 100 key point matches. If no such pair can be found- return failure (no reconstruction can be done in this case).

3.2 Experimental Results

In the pictures bellow you can see the results of reconstructions obtained from initial pairs with different percentages of homography inliers. According to the theory, we expect the best reconstruction to be obtained from the pair with the lowest homography inlier percentage and the worst to be from the pair with the highest homograhpy inlier percentage.



frontal view



profile view

Figure 10: Comparison of reconstructions from different initial pairs. The numbers in white are the percentages of homography inliers for the initial pair. As you can see, this results backups the theory.

In order to quantify the quality of our reconstruction we use the following method:

1. Set total error to 0.

2. For every view:

    2.1. Project all cloud 3D points onto the view.

    2.2. For every 3D point:

        2.2.1. If the 3D point has a corresponding 2D feature in the view add the the distance squared between the projected point and the corresponding 2D point to the total error.

3. Return the square root of total error.

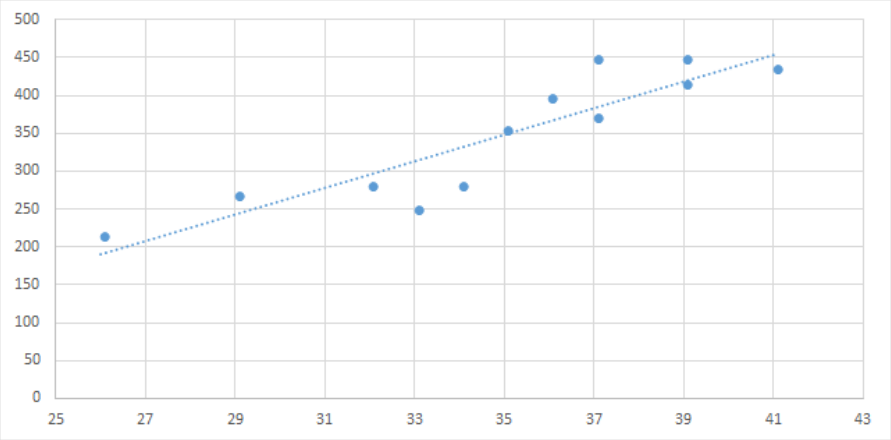


Figure 11: This graph measures the calculated error as a function of the homography inliers percentage of the initial pair. It shows yet again that choosing the pair with the lowest homography inlier percentage will give the lowest reconstruction error.

4 360 Degrees Reconstruction

4.1 The struggle

As described in the algorithm description (sec. 2), the initial configuration of the project used FAST in pyramids for feature detection, ORB for descriptor extraction and nearest neighbors for descriptor matching. This initial configuration proved to be successful in several reconstruction attempts, for most of which the camera didn't surround the object but instead captured it's front from different angels.
The problem occurred when trying to reconstruct an object from pictures surrounding it in order to obtain a full 3D model. When taking the pictures too far away from each other, finding matches became harder, and in many cases

the last images to join the cloud couldn't contribute to it because they didn't have enough 2D-3D correspondences. As a result, only one side of the object appeared in the final reconstruction. When taking the pictures close to each other, the number of images required to surround the object increased significantly (we were now dealing with reconstruction from 20 or more images instead of 3-7) and the clouds we were getting were very inaccurate.
A good dataset example to demonstrate the problem is the dataset of the Chinese temple.

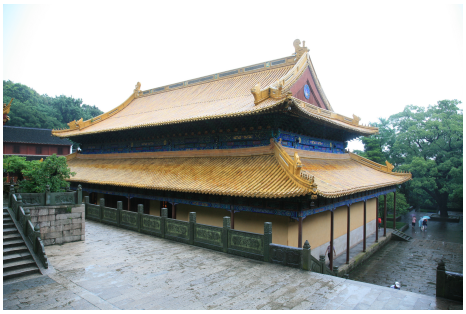


Figure 12: An original image from the dataset of the Chinese temple

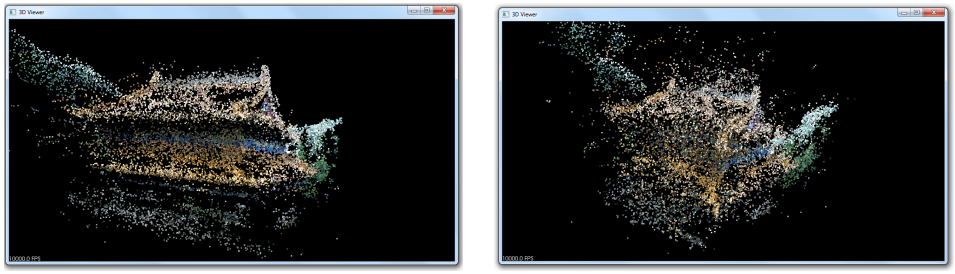


Figure 13: The reconstruction results when using the original algorithm (FAST+ORB). As you can see, when looking from a certain angle the reconstruction looks good, but when turning the image it becomes clear that reconstruction only succeeded for one side and failed for the others.

4.2 The solution- using SIFT and Lowe's ratio test

A careful examination of the problematic datasets discovered that the reason for the failure in must cases of using a large number of images lies in the inaccuracy of the feature matcher. The FAST feature detector finds a relatively large number of features and it causes the matcher to find many false matches.

If this false matches are mistakenly believed to be inliers, they participate in the calculations and ruin the whole process.
Therefore, we decided to find a better method for feature detection and descriptor extraction, which will find only features with good qualities that are likely to be matched correctly. We settled on SIFT due to its scale invariance.
In addition, we noticed that in many cases the reason for a false match was that the image contained a repeated pattern, such as the tiles of a roof, causing different points on it to look similar enough to confuse the matcher. In order to avoid this problem, we decided to use Lawe's ratio test [5] which is another way to filter out bad matches before calculating the fundamental matrix. In this test, for each point p in the first image we find the two best matches p1 and p2 when the distance between p to p1 is smaller or equal to the distance between p and p2 in the second image. Then we check the ratio between the distance of p to p1 and p to p2. If the ratio is below some threshold, then p and p1 are considered a good match. When the image contains a repeated pattern, we expect the distance of p to p1 and p to p2 to be very similar, and therefore this test is expected to filter out false matches due to a repeated pattern.

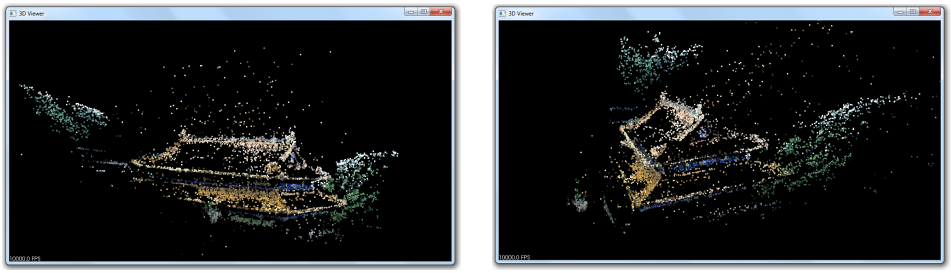


Figure 14: The reconstruction results when using the new algorithm (SIFT+RatioTest). Now the reconstruction looks well from all angles, and it also looks more precise than before. One problem which comes to eye at this point is that the reconstruction looks too sparse. We will deal with this problem in the following section.

5 Exploring Optical Flow

5.1 The struggle

After using SIFT and Lowe's ratio test we were able to achieve 360 degrees reconstruction for the first time. But, most 360 degrees reconstructions were too sparse and still weren't good enough. It appeared as though in our attempts to get better matches for the computations, we have demanded too much of our matches, and now in many cases there simply aren't enough of them left. Changing the thresholds to enable more matches didn't seem to solve the problem, since in order to make a difference in the number of points, wee had to

lower the threshold too much and the quality was ruined again. A good dataset example to demonstrate the problem is the dataset of the Dinosaur [6] (With thanks to Wolfgang Niem, University of Hannover). This dataset contains a total of 37 images.

Figure 15: An original image from the dataset of the Dinosaur

Figure 16: The reconstruction results when using SIFT+Lawe's ratio test. Though 360 degrees reconstruction is achieved, it is too sparse.

5.2 The solution- using optical flow

We decided to look for a different method for feature detection and feature matching. We looked for a method that would find more features and enable more matches, but still would be accurate and robust enough to enable 360 degrees reconstruction. The method we found to answer this two criteria was using techniques for the estimation of optical flow. Optical flow is the relative motion between successive frames in a sequence of ordered images. In order to estimate it, features are detected in one image, and then, instead of detecting features again in the following images, a matcher is used to try and find the same features in the other images.
In order to use optical flow methods we had to take away the freedom given to

the user to take pictures in any order, and demand that the supplied images be ordered. We changed the first two stages of the algorithm (feature detection and matching) to do the following:

1. find features in each image using the method good features to track (GFTT).
2. for each image- while there are matches left:
   (a) find all matches in the following image using the Lucas-Kanade method [7]
   (b) cut the number of features by half according to the tracking error and move on to the next image

We expected this method to be accurate, because the feature detection technique, GFTT, is oriented to finding trackable features, and therefore it is likely that this features will eventually bring good matches. In addition, the fact we are specifically looking for the same features over more than two images increases the chances of finding the same 3D point in several views, which increases the effectiveness of the bundle adjustment. The bundle adjustment takes as input 3D points along with a list of views they appear in, and the longer the list the better the optimization it makes. If we take into consideration the fact that now we demand the images to be ordered, we can also say that finding matches between every pair of images is wasteful and even dangerous. That is because images far apart are likely not to have a lot in common, which increases the chances of making false matches.

Figure 17: The reconstruction results when using SIFT+RatioTest (on the left) compared to the reconstruction results using optical flow (in the middle). Clearly optical flow makes the cloud denser without damaging the precision. The picture on the right presents the Results of dense reconstruction done from the optic flow point cloud using an external software [2].

6 Two Iterations

6.1 The strugle

So far, we have seen three methods for feature detection and matching:

- Using PyramidFAST and ORB for feature detection and descriptor extraction, and nearest neighbors for feature matching- The number of features using this methods is usually large, but in many cases it's not accurate enough.
- Using SIFT for feature detection and descriptor extraction and nearest neighbors followed by Lowe's ratio test for feature matching- The number of features using this methods is usually really small, but it's accuracy is high.
- Using Optical flow methods (GFTT and LK)- This method differs from the other two because it finds features across more than two images and it doesn't match features between every pair of images. The results usually give a precise cloud, denser than SIFT's cloud but significantly sparser than FAST+ORB's cloud.

We wanted to find a way to benefit from both the advantages of the first method (large number of features) and the other methods (high accuracy) to create a cloud which is both accurate and dense enough.

6.2 The solution- using two iterations

Our idea was to combine the first method with one of the other two using two iterations. The first iteration will use SIFT or OF in order to create an initial cloud, and the second iteration will rely on the results of the first iteration and use FAST+ORB to make the cloud denser. The new algorithm's scheme is as follows:

First iteration

Run the basic algorithm using SIFT or OF and save the resulting cloud and the intrinsic and extrinsic parameters of all cameras.

Second iteration

1. Run the basic algorithm using PyramidFAST+ORB up until the point before creating the initial cloud (stages 1-3 in the algorithm description).
2. Using the intrinsic and extrinsic parameters recovered in the first iteration, triangulate all the new matches.

This method is expected to be better, because the calculations during the crucial stages of the algorithm (such as locating the initial pair, adding points using PnP and using bundle adjustment) are done using exact matches received in reliable techniques. Additionally, the final point cloud should be dense, since we use all features FAST has to offer.
Another thing is that for the second iteration, if there are any errors in feature matching, each of this errors will only produce a single bad 3D point in the cloud, and it won't affect the next calculations (as opposed to the PnP method, which relies on the existing 3D points for its calculations). This bad 3D points can sometimes be removed from the cloud using graphical and statistical methods alone. We used some of this methods found in the PCL library [8].
Bellow are some example results we got for both new and familiar datasets using this new method.

Figure 18: Original images taken from the dataset of the Spartan Warrior from [9]. The full dataset contains 48 images.

Figure 19: Pictures from the reconstruction made using SIFT+RatioTest

Figure 20: Pictures from the reconstruction made using the two iterations method. The first iteration uses SIFT+RatioTest and the second one uses FAST+ORB.

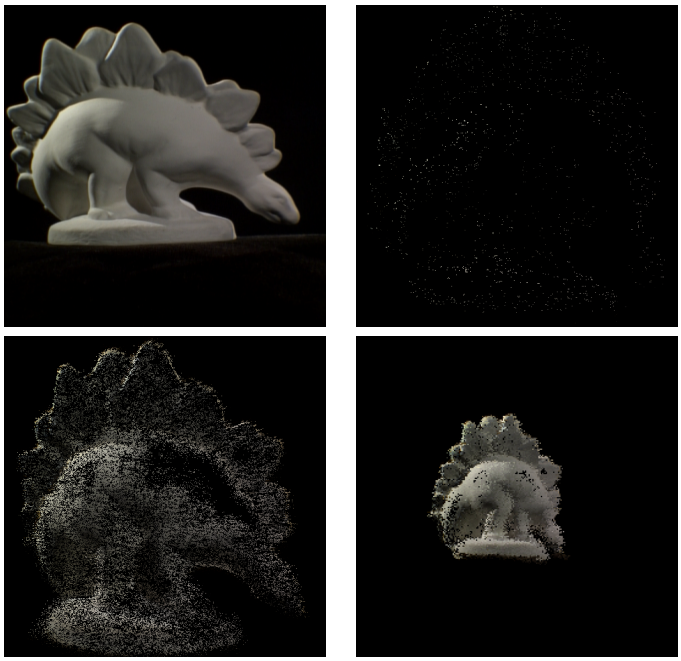


Figure 21: An example using another dataset of a Dinosaur from [10]. **Top left:** one of the original images. **Top right:** the reconstruction result when using SIFT alone. **Bottom left:** the reconstruction result when using SIFT for the first iteration and FAST+ORB for the second. **Bottom right:** a zoom out look of the two iterations reconstruction.

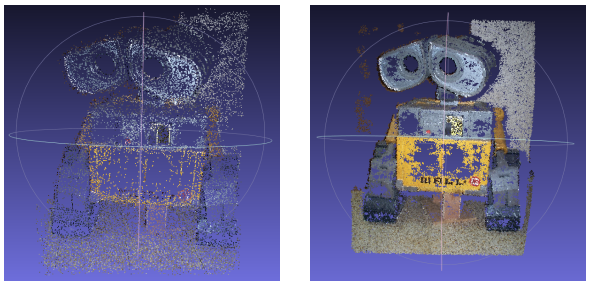
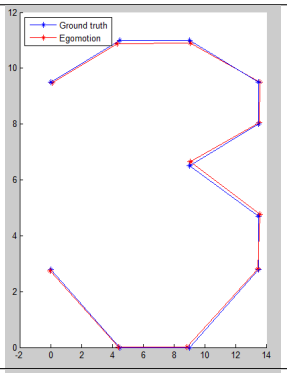
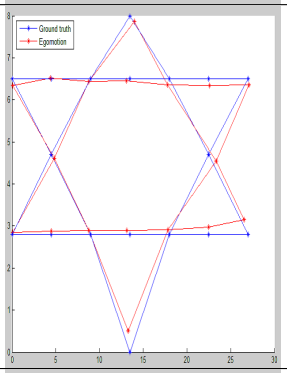
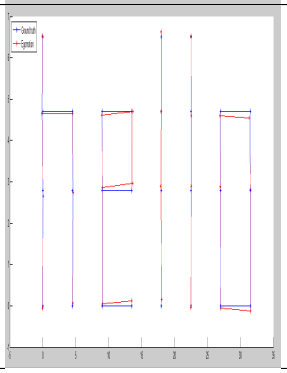
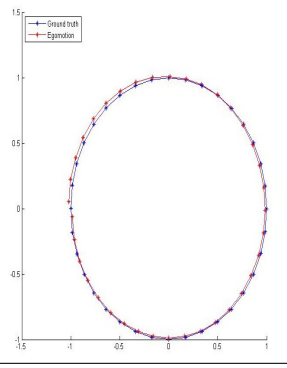


Figure 22: Comparison between the results achieved for the Walle dataset using FAST+ORB (on the left) and using the two iteration method.

7 Egomotion Extraction

Egomotion is the 3D motion of a camera within an environment. Calculating the egomotion of a moving camera in a static scene has many applications in our world today. For example, it can be used to detect the track of a moving car or to detect the path of person walking inside a building (indoor navigation).
Though the main purpose of our algorithm is to create a 3D reconstruction of the scene from the set of images, it also retrieves the egomotion of the camera taking the pictures. In fact, the $3*1$ translation vector recovered for each image in the final stages of the algorithm, is nothing more than the position of the camera in 3D space when taking the picture. Therefore, drawing the egomotion in 3D can be done by simply drawing a point for each translation vector.
In the table bellow you can see the results of a series of experiments we have conducted in order to measure the quality of egomotion extraction using our algorithm. For each experiment we have measured the "ground truth", the exact location of the cameras when taking the pictures, and compared it with the resulting egomotion. This is the time to stress out that since the position of the first camera is chosen arbitrarily and the locations of the other cameras are determined relative to it, the resulting egomotion is defined upto scale, rotation and translation. Therefore, before comparing the results to the ground truth the two must first be aligned. We use [11] for the alignment process and for the error measurement.
The following chart includes what we tried to draw (Dataset), the number of images we used (the number of points in the egomotion), the result (the blue dots represent the ground truth and the red dots represent our result) and the error measurement.

Dataset	# of images	Result	Error
The number 3	11		0.4478
Star of David	18		1.19
The word ”hello”	27		1.19
Circle	36		0.9

8 User Interface

In order to make it easier for us and for future users to use our algorithm, we have created a graphic user interface (GUI). The GUI gives the ability to exchange between the different methods and to control the different parameters, and watch how that affects the final result.
In this section we will introduce the GUI and explain how to work with it.

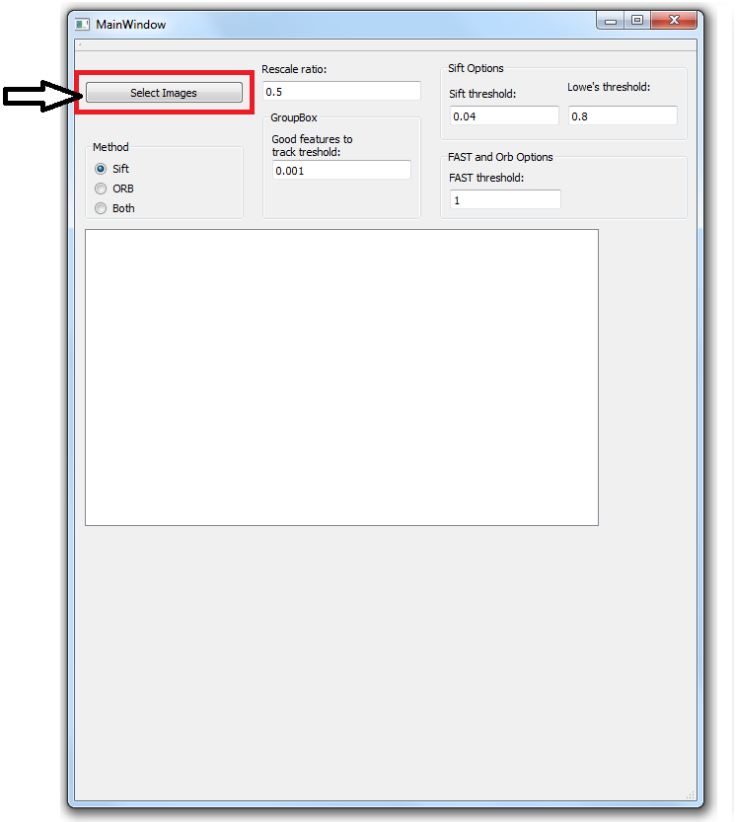


Figure 23: The opening screen- the first thing you want to do is click on the button "Select Images" and choose the images you want to use. The images can be in any format. The only restrictions on the images are the requirements presented in the introduction.

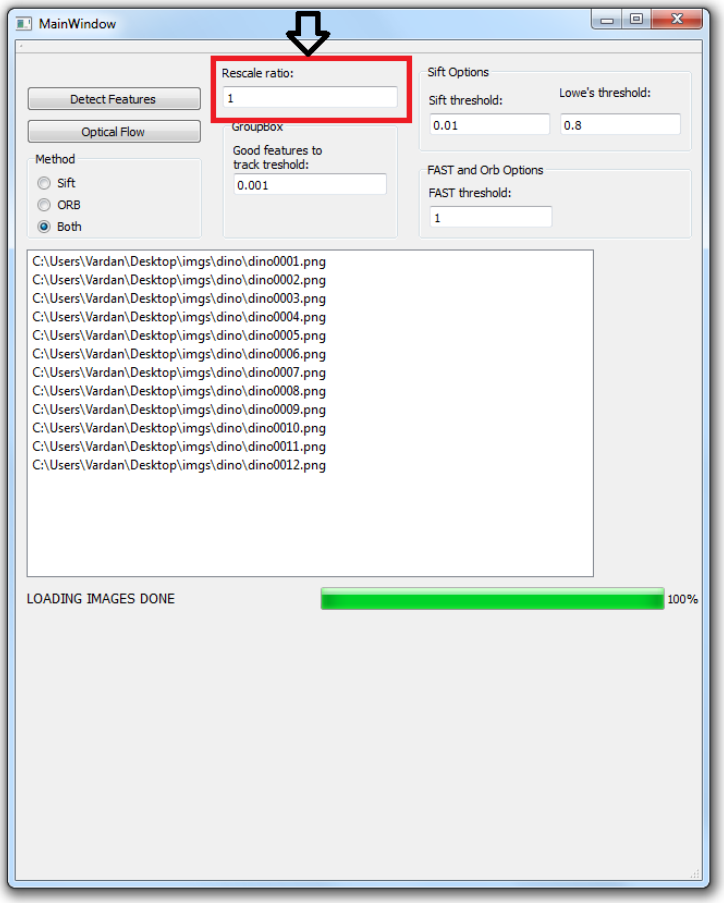


Figure 24: After choosing your images, a list of their file names appear. Clicking on an image name will open a display of the image in a separate window. If the size of the images is too big you might want to consider starting over with a rescale ratio smaller than 1.

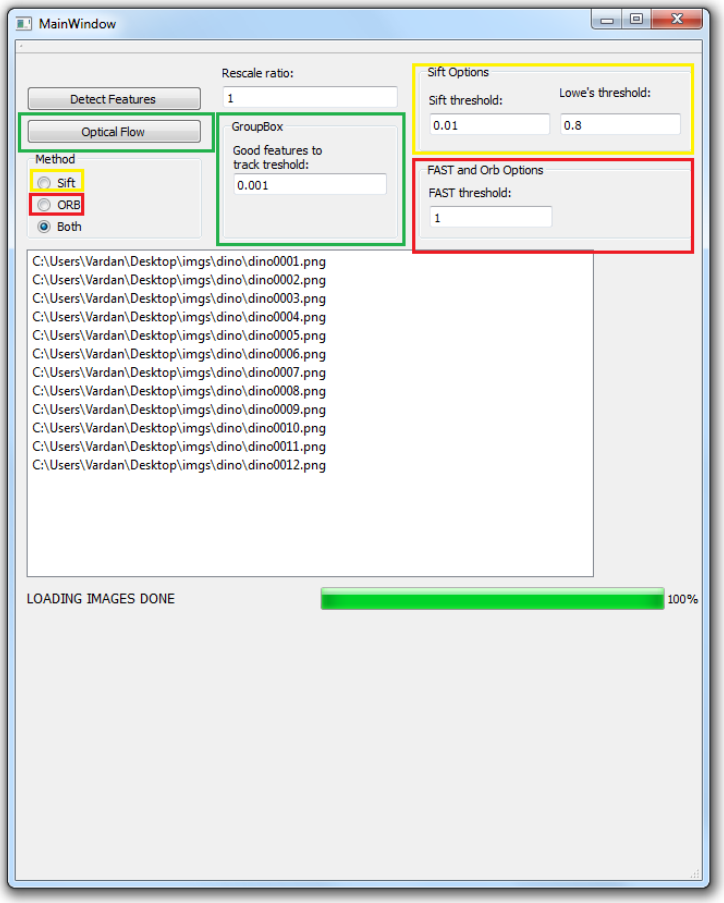


Figure 25: Next, you want to choose the method you want to use for feature detection and matching from the list of 3 options: SIFT, ORB, or BOTH (two iterations method). Another option is to use optical flow (button on the left). For each of these methods the GUI allows you to control its main parameter, the one which determines how strict the decision of feature matching will be. For SIFT, its the SIFT threshold (the lowest it gets the more features are found) and the Lawe's ratio test threshold (enter 1 to cancel its effect). For ORB its the FAST threshold and for optical flow its the good features to track threshold. After entering the relevant parameters for you, click the "Detect Features" button to continue. Or, if you would rather use optical flow, click the "Optical Flow" button. In our example we will continue by selecting BOTH and pressing "detect features".

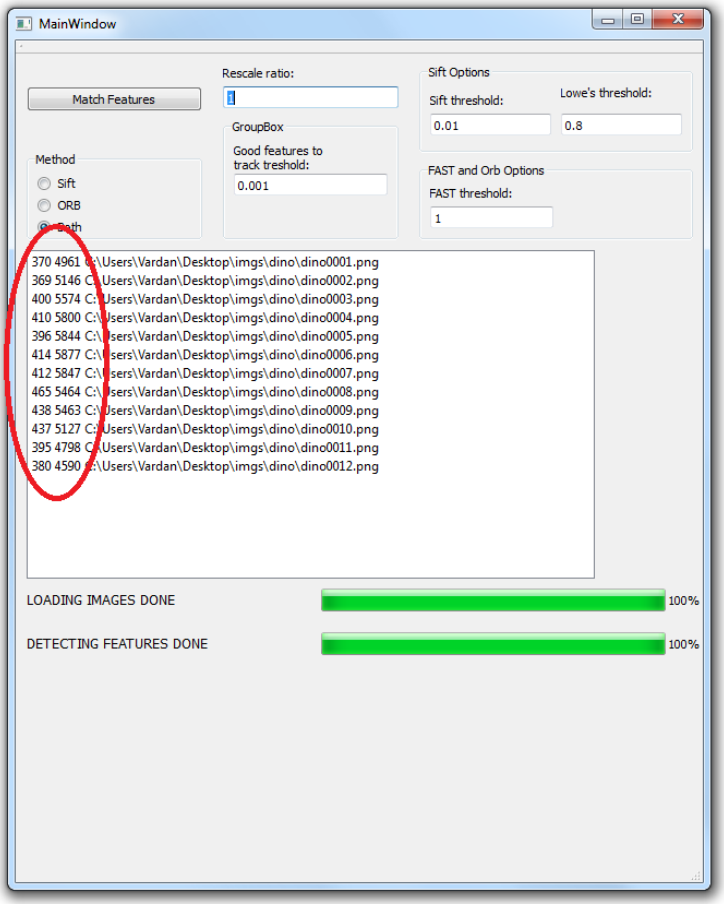


Figure 26: After feature detection is done, the number of features found for each image is added to the left of its name. In our example we choose to use the option "both", and therefore we can see two numbers- one for SIFT matches (on the left) and one for ORB matches. By clicking on the image name, a separate window will open, in which you will be able to see the image with little circles marking the features found. Whenever you're ready click the "Match Features" button to continue.

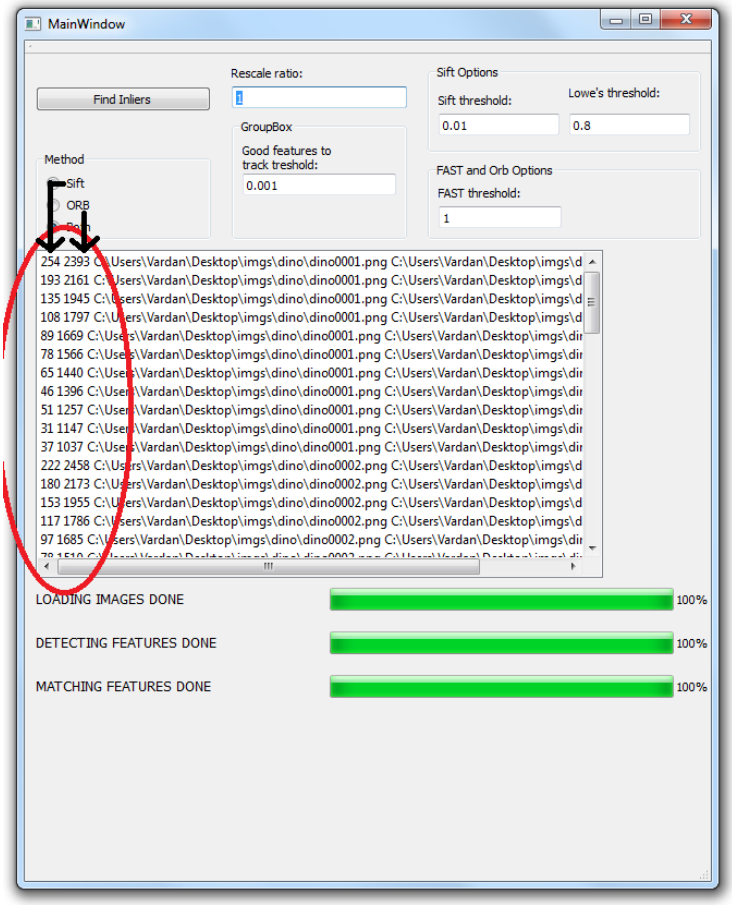


Figure 27: After feature matching is done, the list of image names is replaced by a list of pairs of image name. For each pair, the number appearing on the left of it is the number of matches found between this pair (in our case, again, we have two numbers- one for SIFT and one for ORB).By clicking on a pair, a separate window will open, in which you will be able to see both images with lines connecting features from the left image to corresponding features on the right one. To move on to the next stage click "Find Inliers".

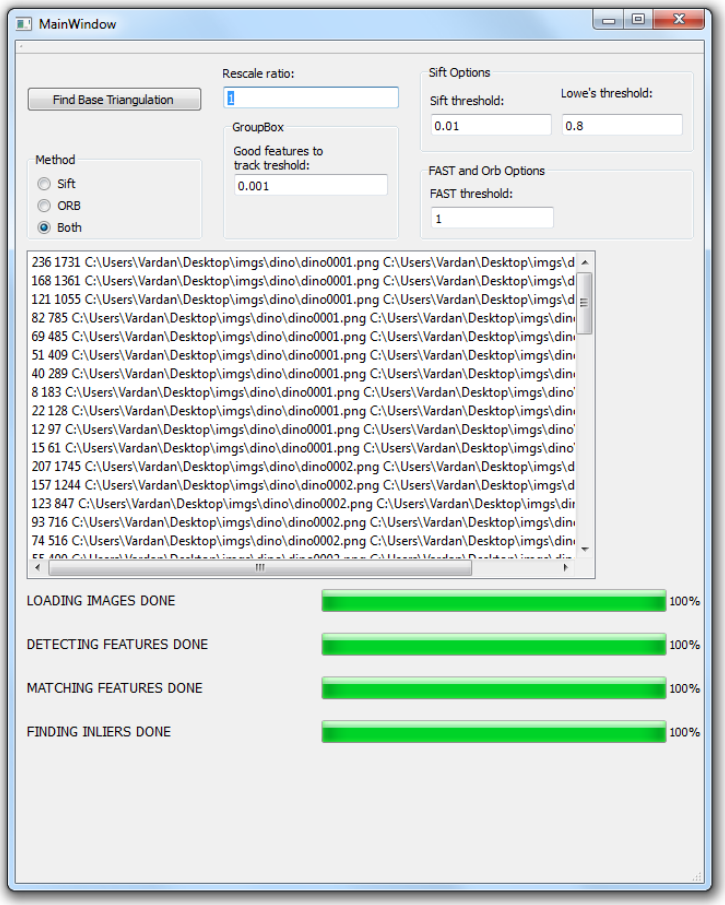


Figure 28: After the stage of finding inliers using the fundamental matrix is done, you may have noticed the only thing different is that the numbers to the left are a little smaller. As you may have guessed, this is because this numbers represent the number of matches after removing outliers. As before, by clicking on a pair in the list, a separate window will open, in which you will be able to see the matches (there should be less "bad" lines than before). Now, click the "Find Base Triangulation" button to begin the reconstruction.

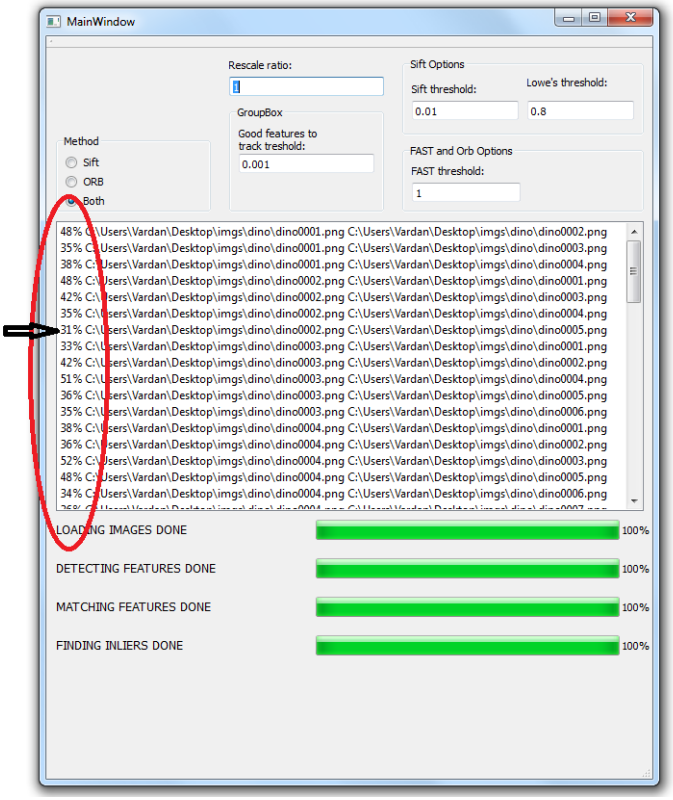


Figure 29: As you can see, the number of matches for each pair was replaced by the homography inliers percentage between that pair. As explained in sec. 3, the best reconstruction is expected to be when choosing the initial pair as the one with the lowest homography inliers percentage. The GUI gives us a simple way to test it- by clicking on a pair of images that pair will be chosen as the initial pair and the reconstruction will continue until the creation of the final cloud. When the reconstruction is done, you can click on another pair and see how your choice affected the result.

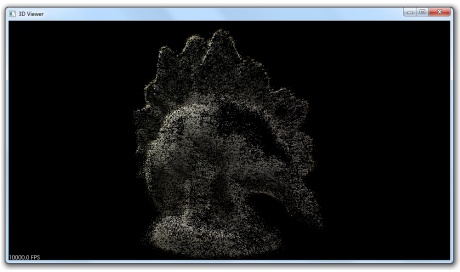


Figure 30: Once choosing an initial pair from the list, the reconstruction can be complete. This process may take a while, and when it ends a separate window will appear, in which you will be able to see a 3D display of the final cloud. You can zoom in and out using the mouse roller, or rotate it in any direction by long clicking the left button and dragging.

9 Technical Appendix- External libraries

OpenCV [13] OpenCV is the library we use for computer vision functions. Most of the functions we use are specified in the algorithm outline.

Point Cloud Library [8] Point Cloud Library (PCL) is the library we use in order to visualize the cloud obtained. We wrote a seprate program which takes as input a txt file which contains rows in format of XYZRGB and outputs a 3D graphic point cloud. This program uses Point Cloud Library (PCL).

Simple Sparse Bundle Adjustment [4] SSBA is the class we used for our bundle adjustment (sec. 2.7)

References

[1] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Modeling the world from internet photo collections, 2007.

[2] `http://meshlab.sourceforge.net`.

[3] `http://en.wikipedia.org/wiki/Triangulation_%28computer_vision%29`.

[4] `http://www.inf.ethz.ch/personal/chzach/opensource.html`.

[5] David G. Lowe. Distinctive image features from scale-invariant keypoints, 2003.

[6] `http://www.robots.ox.ac.uk/~vgg/data/data-mview.html`.

[7] http://en.wikipedia.org/wiki/Lucas%E2%80%93Kanade_method.

[8] http://pointclouds.org/.

[9] http://homes.cs.washington.edu/~furukawa/research/visual_hull/index.html.

[10] http://vision.middlebury.edu/mview/data/.

[11] http://www.mathworks.com/matlabcentral/fileexchange/26186-absolute-orientation-horns-method.

[12] http://www.morethantechnical.com/2012/02/07/structure-from-motion-and-3d-reconstruction-on-the-easy-in-opencv-2-3-w-code/#.

[13] http://opencv.org/.