# MULTI-VIEW STRUCTURE FROM MOTION SOFTWARE PLATFORM FOR AUGMENTED REALITY APPLICATIONS

A Thesis

by

Cengiz Hüroğlu

Submitted to the
Graduate School of Sciences and Engineering
In Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in the
Department of Electrical and Electronics Engineering

Özyeğin University
May 2014

# MULTI-VIEW STRUCTURE FROM MOTION SOFTWARE PLATFORM FOR AUGMENTED REALITY APPLICATIONS

Approved by:

---

Professor A. Tanju ERDEM, Advisor
Department of Electrical and Electronics Engineering
*Özyeğin University*

---

Associate Professor M. Oğuz Sunay
Department of Electrical and Electronics Engineering
*Özyeğin University*

---

Assistant Professor A. Özer Ercan
Department of Electrical and Electronics Engineering
*Özyeğin University*

Date Approved: 27 May 2014

*To my family*

# ABSTRACT

Rapid development of sensor, computer and display technologies led to increased usage of structure from motion techniques in many applications including navigation, human-computer interaction, training, entertainment, and augmented reality. Structure from motion techniques usually involve 2D feature detection, feature matching, and 3D feature extraction. We present a software platform which brings together state-of-the-art open-source third-party tools for structure from motion. These tools are generally used via a command prompt and have a large number of parameters. The output files of these tools also have their own specific format and users usually need to write special parsers to extract the information they need from these files. Our software platform provides several easy-to-use visual interfaces to configure and set the important parameters of these tools as well as parse/save their out files as desired. Our platform also provides visual interfaces to display the detected 2D features and extracted 3D points on their respective image frames, as well as allowing for making 3D distance measurements between feature points. We are planning to make the source-code of our system openly available to researchers.

**Keywords:** Feature Detection, SIFT, Feature Matching, Camera Calibration, Radial Un-distortion, Structure from Motion

# ÖZETÇE

Sensör, bilgisayar ve görselleme teknolojilerindeki hızlı gelişme, hareketten yapı çıkarımı tekniklerinin navigasyon, insan-bilgisayar etkileşimi, eğitim, eğlence ve eklenmiş gerçeklik gibi uygulamalarda kullanımını arttırmıştır. Hareketten yapı çıkarımı, genel olarak 2B öznitelik tespiti, öznitelik eşleştirme ve 3B öznitelik çıkarımından oluşur. Bizler gelişmiş bazı açık kaynak kodlu 3. parti araçları biraraya getiren bir yazılım platformu sunuyoruz. Bu araçlar genellikle komut satırından kullanılmaktadırlar ve birçok parametreye sahiptirler. Bu araçlar çıktı olarak kendi özel formatlarında dosyalar üretmektedirler ve kullanıcının bilgileri alabilmesi için bu dosyaları çözümlemesi gerekmektedir. Yazılım platformumuz, bu araçların konfigürasyonu ve önemli parametrelerin ayarlanması ve bunların yanı sıra da çıktıların istenildiği gibi çözümlenmesi ve saklanması için görsel birtakım arayüzler sunmaktadır. Platform ayrıca 2B özniteliklerin ve çıkarılan 3B noktaların karşılık gelen imgeler üzerinde gösteriminin yanında öznitelik noktaları arasında 3B mesafe ölçümüne olanak tanıyan görsel arayüzler de sağlamaktadır. Araştırmacıların kolayca erişimi için platformun kaynak kodunu açmayı planlıyoruz.

**Anahtar Kelimeler:** Öznitelik Tespiti, SIFT, Öznitelik Eşleştirme, Kamera Kalibrasyonu, Radyal Bozulma Düzeltme, Hareketten Yapı Çıkarma

# ACKNOWLEDGEMENTS

I did my MSc studies at the Computer Vision laboratory in Özyeğin. I would like to start by thanking my supervisor Prof. Tanju Erdem for all his supports. I really appreciate our many discussions and I hope to continue our fruitful collaboration in the future.

I would like to thank to my thesis committee members for their time. I would also like to thank to my group members, Ahmet Kermen and Instructor Tarkan Aydın for sharing their knowledge and their assistance.

Istanbul, May 2014 Cengiz Hüroğlu

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER I

# INTRODUCTION

The motivation of the thesis and the targeted contributions of the thesis are given in this chapter. Also a literature overview of SfM and augmented reality tools is provided here.

## *1.1 Motivation*

The importance of multi-view Structure from Motion and Augmented Reality researches based on camera and object tracking in 3D environment and supporting with virtual items gradually increases. In order to achieve a realistic feeling of immersion, the rendering of the virtual content has to be in alignment with real objects in the video and this requires a high-accuracy 3D tracking. Reliability of tracking systems should be increased by using true and reliable methods and tools. The first and the most important stage of these kind of applications is estimating the 3D scene model and motion information.

Augmented Reality (AR) is one of the fields of computer vision research that combines real world and digital data which enables users to see real and virtual objects together in the same place. This area is closely related with almost all computer vision subjects such as 2D-3D geometries, feature detection and match, 3D reconstruction, camera and object tracking, multi-view stereo, 3D animations etc. Instead of replacing the real world totally, AR systems require to complete and enrich the real world [1].

Figure 1 shows a flowchart for a simple augmented reality system [2]. The capturing module captures the image from the camera. The tracking module calculates the correct location and orientation for virtual overlay. The rendering module combines

the original image and the virtual components using the calculated pose and then renders the augmented image on the display.

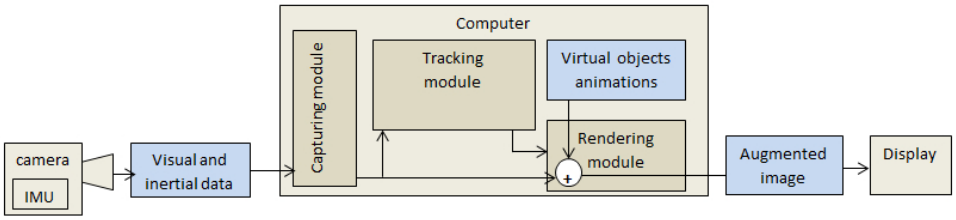



Figure 1: Flowchart for a simple AR system

The main issues need to be solved for a successful AR experience can be listed as:

1. A reliable 3D scene map (3D feature points on scene) must be extracted.
2. An accurate way of 2D feature detection method is needed.
3. The correspondence between of 2D features need to be found with a convenient method.
4. Feature detection and match algorithms should run real-time for a realistic feeling.
5. An effective sensor-fusion method is needed to combine data for a more accurate estimation.
6. And finally the virtual reality objects should be added on the scene and they should be stable according to the movement of the camera.

The scope of the provided platform in this thesis is to develop a high performance visual user interface enabling feature detection and 3D map extraction for use of real-time applications such as 3D camera motion tracking and Augmented Reality systems.

## *1.2 Objective of the Thesis*

The objective of the software platform we developed in this thesis is to develop a user interface that provides researchers an easy-to-use toolkit to do some common computer vision processes and get the data they need for their work. So the researchers working on sensor fusion, augmented reality, occlusion tracking etc. could benefit from this platform. This platform will:

1. Isolate the users from the inner details of the methods and tools out of their research scopes.
2. Include an effective feature detection and matching component.
3. Include a reliable Structure from Motion tool.
4. Output of this software can be used as input data or ground truth of for many computer vision tasks and applications such as scene reconstruction, object/-camera tracking, augmented reality and interaction.
5. The output information can also be used in sensor fusion tracking experiments to correct the drift on tracking.
6. Users can configure the application and set the parameters of the tools.

## *1.3 Literature Review*

### 1.3.1 AR Applications

AR is applicable to many areas such as medical education, remote control, entertainment, and cultural heritage [3] [4]. An application sample of revitalization of historic heritage can be seen in Figure 2.

Figure 2: An AR based system for personalized tours in cultural heritage sites

In Figure 3 you can see some usages of tablet PCs and mobile phones as AR medium. WIKITUDE Augmented Reality browser (left) on tablet PC and a free iPhone application called Home Scan (middle) which allows users to stand on a sidewalk and open their iPhone and visually see which homes are for sale. Another application is an exhibition at the Newcastle Museum entitled Reconstructing Victorian Newcastle.

Figure 3: Sample AR applications. A mobile application (left) and two tablet applications

AR is also used for training purposes (Figure 4). On the left side you can see a system for teaching maintenance and repair, and for training purposes in several other fields as well. And on the right side a Dismounted Soldier Training System (DSTS) comprises of sensors attached to various body parts, a virtual reality headset

and life-sized/weight weapons. Up to nine soldiers can be simultaneously immersed in a 3D combat environment where they can train in realistic but safe simulations. The sensors detect physical moment and moving through the scenes is achieved using a joystick toggle.

Figure 4: Two applications for training. Augmented reality for maintenance and repair (left), and a tactical training systems for use of security forces.

### 1.3.2 SfM and AR Toolkits and Libraries

Researchers and developers have created a great number of augmented reality tools (software libraries, toolkits, SDKs, etc.) that are used for AR application development. They usually contain the methods for core augmented reality functionalities: 3D reconstruction, tracking, graphic adaptation and interaction [2].

In the context of augmented reality, authoring means defining the content for an AR application and creating the rules for augmentation (e.g. animation paths), and an authoring tool is the implement for doing so. Some AR tools have components of both core AR functionalities and authoring, such as Artisan [5], which is a front end and management system for FLARToolkit and Papervision3D.

AR tools often use third party libraries for lower level tasks (external tools) and wrap them into the level needed for AR. They use OpenCV for computer vision and image processing, for example, and Eigen or LAPACK for linear algebra. In addition, they may provide an interface for existing tools for image acquisition (e.g.

Highgui) and camera calibration (e.g. OpenCV), or provide their own utilities for these tasks. An AR application developer may naturally use any other software for image acquisition and calibration as well. Respectively, AR applications normally use existing graphics libraries and 3D engines for graphics and rendering (e.g. OpenGL, Open Scene Graph, OGRE, Papervision3D, etc.).

#### *1.3.2.1 ARToolKit*

The first library for creating augmented reality applications was ARToolKit [6]. Together with its descendants, it is probably the best-known and most commonly used tool for creating augmented reality applications. ARToolKit product family consists of libraries for creating stand-alone applications, web applications and mobile applications for several platforms, e.g. ARToolKitPro (C/C++ markerbased tracking library), FLARToolKit (the Flash version of ARToolKit), ARToolKit for iOS (the iPhone port of ARToolKit Pro) [6] [7].

#### *1.3.2.2 VisualSFM*

Some open-source toolkits provide Structure from Motion (SfM) functionalities which are critical for AR systems. VisualSFM [8] is a GUI application for 3D reconstruction using SfM. VisualSFM is able to run very fast by exploiting multicore parallelism in feature detection, feature matching, and bundle adjustment. This toolkit is can work with Yasutaka Furukawa's multi-view stereo system PMVS/CMVS [9] tool and to prepare data for Michal Jancosek's CMP-MVS [10].

#### *1.3.2.3 SFMToolkit*

SFMToolkit [11] is yet another SfM-MVS toolkit using Bundler, SiftGPU, and PMVS/CMVS inside. You can see the workflow and a sample application result of SFMToolkit in the Figures 5 and 7. This workflow is almost the same for all MVS systems. You can see a brief list of SfM-MVS softwares in Table 1 [12].

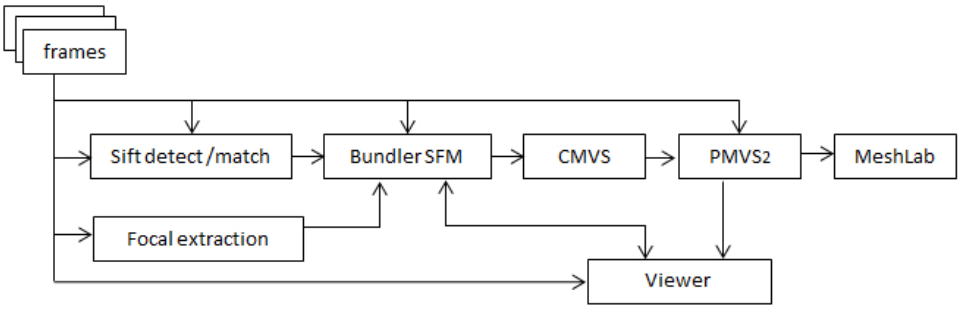



Figure 5: MVS worklow of SFMToolkit

User takes or collects a lot of pictures from different angles of the same place (Figure 6). And then compute structure from motion and get a sparse point cloud using Bundler). At the end you can see the 3D reconstruction of the scene angles. Finally we have a dense point cloud divided in cluster by CMVS and computed by PMVS2 (Figure 7).

#### *1.3.2.4 Bundler*

Bundler [13] is a Structure from Motion system for unordered image collections (for instance, images from the Internet). Bundler takes a set of images, image features,

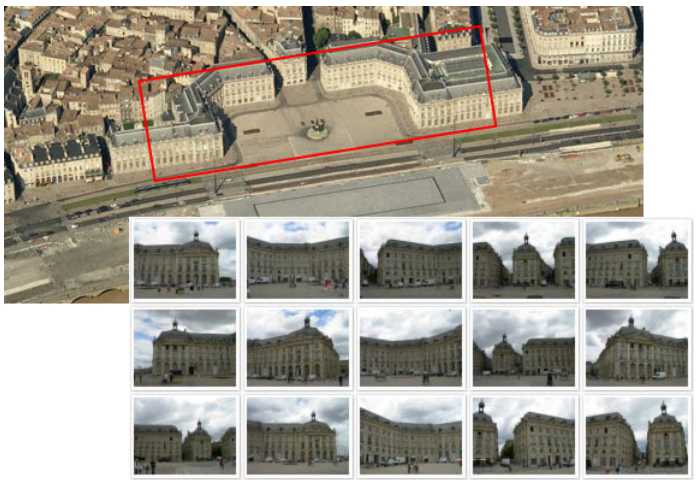


Figure 6: Picture of Place de la Bourse, Bordeaux, FRANCE taken from Bing(top) and a collection of images of it(bottom)

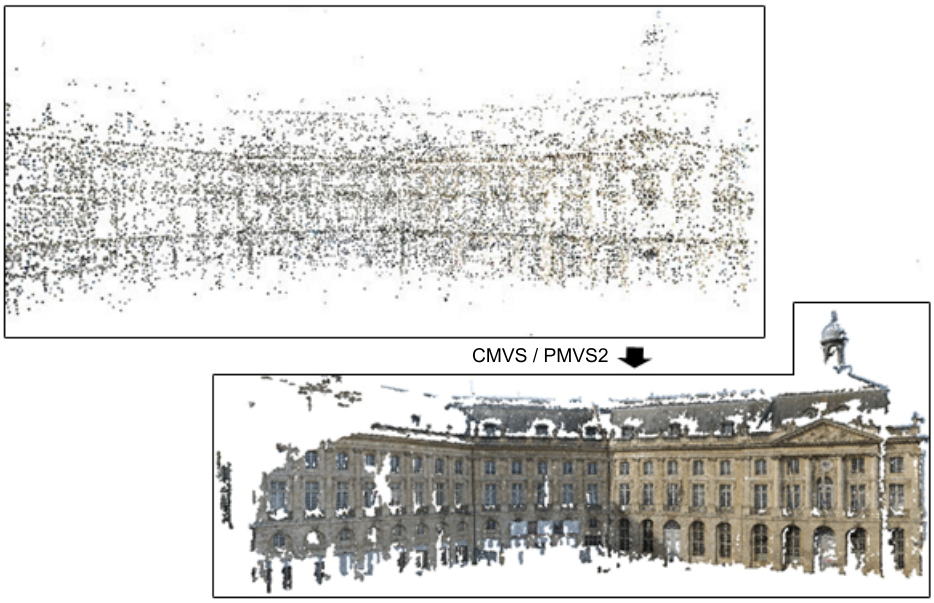



Figure 7: 3D points as a point cloud extracted by Bundler(top) and Reconstruction of the place with CMVS/PMVS(bottom)

and image matches as input, and produces a 3D reconstruction of the camera and (sparse) scene geometry as output. This SfM system was used in a lot of projects like Photo Tourism [14], SFMToolkit, VisualSFM etc.

#### *1.3.2.5 ALVAR, SMMT and DART*

For example, VTTs ALVAR [15] is a software library for creating virtual and augmented reality applications with support for several platforms, PC and mobile environments alike. It has both a marker-based and a feature-based tracking functionality. Furthermore, it has some support for diminished reality and rendering. The SMMT library (SLAM Multimarker Tracker for Symbian) [16] is an example of a very specialised AR tool. As its name suggests, it is suitable for multi-marker AR application development on Symbian and it uses the SLAM approach for tracking. On the other hand, some tools are more core AR tools such as the abovementioned ALVAR and SMMT libraries, and others are more authoring tools such as DART (The Designer's Augmented RealityToolkit) [17].

Augmented reality tools are difficult to compare, as some of them are specialised to one purpose (e.g. marker-based tracking or mobile environments), some support only certain platforms (e.g. Windows or iOS) and others support several platforms and are used for several purposes. We may classify AR tools based on the environments they use (mobile, PC, VR, etc.), the platforms they support (Windows, Linux, Symbian, iOS, Android, etc.), the language they use (C++, Java, etc.), the approach they use for tracking (marker, multi-marker, features), the algorithms they use for tracking (SLAM, PTAM etc.), or the functionalities they have (diminishing, interaction, etc.) Alternatively, we could have a more commercial viewpoint and compare the licensing and pricing issues as well.

In practice, people are often more interested in the performance of the applications created with the tools rather than the approach they use. However, the performance comparison is difficult due to the large diversity of abovementioned platforms, levels and functionalities, and because there is no standard for AR, not to mention a standard for benchmarking AR tools.

We can summarise that there is a large variety of tools available for AR application development and the best tool depends mostly on the application, which defines the environment, platform, functionalities needed, etc. Yet, developers have other aspects as well, e.g. how familiar they are with the tools, how easy they are to use and what third party libraries they require, etc.

To be able to choose right tools and methods, we need to consider some design constraints.

**Speed:** Feature tracking system we will employ in our platform must be able to run at interactive rates with high-performance to have a real-time sense and a successfully augmented vision. For the purposes of real-time augmentation, the speed of our tracking and pose estimation process is critical. This requirement will be hold by using a GPU based feature detection and match module (SiftGPU) implemented

in C++.

**Robust to Scale and Orientation Changes:** The frames used in this system can be in different scale and orientations. The feature detection method should be robust to these changes. SIFT based feature detection methods provide scale and orientation invariant process ability.

**Robust to Lighting Changes:** Major changes in lighting could affect an augmentation systems ability to detect certain features in a pattern layout. Thus a method that can handle lighting changes is a major consideration when it comes to robustness in augmented reality. SIFT also provide a solution that is robust to lighting changes.

**Reliable 3D Space Map:** The 3D space map should be constructed before the Augmented Reality tracking and its reliability must be high. For 3D space model extraction we use pre-recorded video of the application area and process a high performance Structure from Motion tool named Bundler.

**Easy-to-use User Interface:** Third party tools such as Bundler or SiftGPU are generally used via command prompt. User needs to know the usage, provided SDKs and the parameters. By developing the application as a Windows desktop application with windows forms we aimed to provide an easy to use interface.

**Configurability:** Application should be configurable and user could be able set the parameters of modules. Platform architecture is designed as modular so any new component can be integrated in it with a minimum effort. User interface has settings screens and parameter inputs to set and configure the system.

To conclude, we determined two main open-source computer vision tools to integrate in our design. These are Bundler [13] for Structure from Motion and SiftGPU [18] for feature detection and match. The details of these tools are given in Chapter 2.

## 1.4 Contributions of the Thesis

We present reliable 3D map reconstruction and real-time robust feature detection and matching utilities that every researcher working on AR and feature tracking subjects may need. So they can focus on their own subject and gain time. Output of this software can be used as input data or ground truth of some computer vision researches and applications such as scene reconstruction, object/camera tracking, augmented reality and interaction. The output information can also be used in sensor fusion tracking experiments as corrective information to correct the drift on tracking.

Third party tools such as Bundler or SiftGPU are generally used via command prompt and have their own mechanisms and they have a lot of parameters to run. Users need to know the usage and the parameters. By developing this application as a Windows desktop application, we aimed to provide an easy to use interface. Our system contains tools for feature extraction and 3D reconstruction integrated inside. It also provides an interface to reach the functions of the modules inside via the wrapper utility functions. Bundler's or SiftGPU's outputs are in the form of text files including a lot of information. User needs to know the format details of the files in order to parse it, convert it to format he/she needs and then use it. ArOZU provides user interfaces to configure and set the processes and parse/save the files created.

There are also some interfaces we provide to view the detected 2D features on frames or extracted 3D points on respective frames.

We are going to distribute the source-code and binary of the system. Any researcher needs the functionalities provided with our system or wants to add new functionalities on it can use it.

## 1.5 Outline of the Thesis

This thesis is organized in the following way:

First chapter contains the motivation of this thesis, literature overview, contributions achieved. It mentions on the problem and its importance, real time AR applications and the purpose and the objectives of this work.

In the second chapter, background information used in this thesis is given. It gives overview of EKF-based tracking, 3D reconstruction and feature extraction concepts and tools used in the thesis. Details of two main tools we integrated in our platform is given here.

Chapter 3 gives details of the platform designed and developed for use in augmented reality and sensor-fusion experiments. The user interface and components developed and the usage of the platform is detailed in this chapter.

In Chapter 4 the experimental setup and the results and the evaluation of results are given. Chapter 5 gives a summary and discussion together with possible future research directions.

## *1.6 Chapter Summary*

In this chapter we talked about Augmented Reality subject and main issues of realistic applications to present our motivation. We also did a brief survey on AR applications, prominent tools and softwares used for 3D map reconstruction and motion tracking. Next chapter provides background information about the methods and tools we chose in our research.

Table 1: Examples of software implementing SfM-MVS

| Software | Notes |
|---|---|
| **Freely available** | |
| Bundler Photogrammetry Package[b, c] | Photogrammetry tool using Bundler and PMVS2 |
| SFMToolkit[b, c] | Similar software to above |
| OSM-bundler[b, c] | Open source equivalent of Microsoft Photosynth, uses Python scripts and has a Linux distribution. |
| VisualSFM [c] | A graphical user interface and versions for Windows, Linux and Mac. OSX, but camera model is more restricted than that used in Bundler. |
| **Commercial** | |
| Photosynth | Evolved from Bundler. SfM only, no dense reconstruction. Can incorporate a very wide variety of images, but does so at the cost of reconstruction accuracy. |
| Arc3D | A webservice allowing users to upload digital images to our servers where we perform a 3D reconstruction of the scene and report the output back to the user. |
| CMP SfM Web service b | SfM webservice similar to above |
| Autodesk 123D Catch | SfM webservice similar to above |
| My3DScanner | SfM webservice similar to above |
| **Web sites** | |
| PhotoScan | Full SfM-MVS-based commercial package |
| Acute3D | Full SfM-MVS-based commercial package |
| PhotoModeler | Software based on close-range photogrammetry, now implements some SfM technology. |

b uses Bundler to compute structure from motion
c uses PMVS2 as a dense multi-view matcher

# CHAPTER II

# BACKGROUND

This chapter provides an overview of the concepts, techniques and software tools we used in our platform in order do 3D reconstruction, feature detection and feature matching.

## *2.1 SiftGPU - 2D Feature Detection and Matching*

For real time systems such as tracking and augmented reality applications all the steps should run at interactive rates with high-performance to have a real-time sense and a successfully augmented vision. First step of 3D structure reconstruction is feature detection. Finding point correspondences is vital to successful triangulation and calculating camera pose. In order to find corresponding features over video frames it must be possible to detect features as described in the previous section, but it is also important to identify features and match them between frames. We call this feature description and it involves extracting feature information. As with feature detection, a wide variety of feature description algorithms have been presented over the years. A good feature descriptor must exhibit these three characteristics [19]:

1. Repeatability: The feature descriptor should be reliable, finding the same physical interest points under different viewing conditions. It must have high accuracy and a low false-positive rate. It should be invariant to changes in rotation, translation and scale.

2. Robustness: The feature descriptor must be able to identify the same point between frames even if there are changes in illumination, changes in noise and small changes of the viewpoint.

3. Speed: It must be able to extract feature information and match it against a large database as quickly as possible, preferably in real-time.

To meet the performance requirements we decreased the time of feature extraction and matching steps by using a GPU accelerated component. In these section we will give a brief information about Scale Invariant Feature Transform (SIFT) method and then the SiftGPU [18], GPU version of it, integrated in our platform.

D. Lowe [20] introduces a method for image feature extraction called the Scale Invariant Feature Transform (SIFT). It is invariant to image scaling, translation and rotation as well as being at least partially invariant to changes in illumination and 3-D projective transforms. This approach transforms an image into a large collection of local feature vectors called "SIFT keys" that are used for identification.

The major stages of SIFT used to generate the set of image features are: scale-space extrema detection, keypoint localization, orientation assignment and keypoint descriptor.

### 2.1.1 SiftGPU

SiftGPU [18] is an implementation of David Lowe's Scale Invariant Feature Transform (SIFT) [20] for GPU. SiftGPU processes pixels parallelly to build Gaussian pyramids and detect DoG Keypoints. Based on GPU list generation, SiftGPU then uses a GPU/CPU mixed method to efficiently build compact keypoint lists. Finally keypoints are processed in parallel to get their orientations and descriptors. SiftGPU also includes a GPU exhaustive/guided sift matcher SiftMatchGPU. It basically multiplies the descriptor matrix on GPU and finds the closest feature matches on GPU. Both GLSL and CUDA implementations are provided. It runs on GLSL by default , which works for both ATI and nVidia. You can optionally use CUDA for nVidia graphic cards. The following steps can use GPU to process pixels/features in a parallel way:

1. Convert color to intensity , and up - sample or down - sample input images
2. Build Gaussian image pyramids (Intensity, Gradient, DOG)
3. Keypoint detection ( sub - pixel and sub - scale localization )
4. Generate compact feature lists with GPU histogram reduction
5. Compute feature orientations and descriptors

By taking advantages of the large number of graphic processing units in modern graphic cards, this GPU implementation of SIFT can achieve a large speedup over CPU. Not all computation is faster on GPU, so this library also tries to find the best option for each step. Running SiftGPU requires a high - end graphic card that has a large graphic memory to keep the allocated intermediate textures for efficient processing of new images. The GPU must also support dynamic branching.

#### *2.1.1.1 Running SiftGPU*

SiftGPU is open source C++ project and can be downloaded from official web site. SiftGPU uses DevIl Image library, GLEW and GLUT inside. User will need to make sure target system has all the depending libraries. SiftGPU should be able to run on any operation system that supports these libraries. For Windows system Visual Studio solution files are provided. Linux/Mac makefile is also provided. When source code compiled on Windows a dll named SiftGPU.dll is built. After adding this library to any C++ project, classes and their functionalities can be used.

Table 2: Parameters of SiftGPU

| parameter | description |
|---|---|
| -fo (int) | First Octave to start detection (default: 0) |
| -tc2 (int) | Set a soft limit(max.) to number of detected features (ML) |
| -e (float) | SIFT detection edge threshold (ET) (default : 10.0) |

SiftGPU has a large parameter set for different purposes. We just list some of important parameters that we used in our experiments in Table 2. Two important parameters of SIFT detection represented in Configuration Window are "SIFT Edge Threshold (default 5.0)" (ET) and "Maximum Feature Number Per Frame (default 1000)" (ML). First parameter defines the edge threshold that we also used it to keep the number of features in appropriate range. Second one is the upper limit of feature numbers to be detected. It is not a strict rule, SiftGPU tries to keep the numbers around the limit set. SIFT match has only one parameter "Match Threshold (default 16)" can be modified on user interface. You can find details in official documentation [21].

#### *2.1.1.2 Output Format*

SiftGPU creates an individual file for each frame with ".key" suffix. The format of this file is shown in Listings 2.1.

Listing 2.1: Format of SIFT key files

```
<num_features> <size_descriptor> [two integers:# of features and
descriptor size (def. 128 )]
<feature1>
<feature2>
   ...
```

Each feature entry "featureI" contains the 2D point information (in y, x, scale, orientation order) and descriptor of the feature. Each feature entry has the form as in Listings 2.4.

Listing 2.2: SiftGPU 2D Feature Format

```
<point> [a 4-vector describing the 2D point (in y, x, scale, orientation)]
<descriptor> [a 128-vector describing the feature]
```

Note that SiftGPU takes the top-left corner of the frame as coordinate center as illustrated in Figure 8.

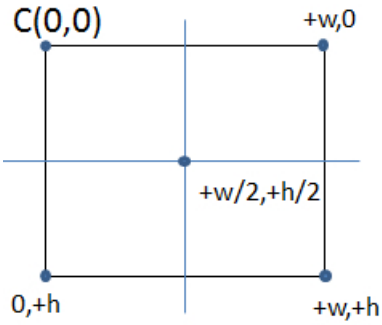



Figure 8: The coordinate system of SiftGPU takes top-left of frame as coordinate center.

## *2.2 3D Map Extraction with Bundler*

### 2.2.1 Obtaining 3D information from 2D Images, SfM

Structure from Motion (SfM) [20] refers to the process of estimating three-dimensional structures from two-dimensional image sequences which may be coupled with local motion signals. It is studied in the fields of computer vision and visual perception. In biological vision, SfM refers to the phenomenon by which humans (and other animals) can recover 3-D structure from the projected 2D (retinal) motion field of a moving object or scene. Humans perceive a lot of information about the three-dimensional structure in their environment by moving through it. When the observer moves and the objects around him move, information is obtained from images sensed over time [22].

Finding structure from motion presents a similar problem as finding structure from stereo vision [23]. In both instances, the correspondence between images and the reconstruction of 3D object needs to be found.

To find correspondence between images, features such as corner points (edges with gradients in multiple directions) need to be tracked from one image to the next. The feature trajectories over time are then used to reconstruct their 3D positions and the camera's motion [22].

Given: $n$ matching image points $x_j^i$ over $m$ views

Find: the cameras $P^i$ and the 3D points $X_j$ such that $x_j^i = P^i X_j$

$$\overbrace{P^i X_j}^{min} \sum_{j \in points} \sum_{i \in views} d(x_j^i, P^i X_j)^2$$

Algorithm for Structure from Motion:

1. Compute interest points(features) in each image
2. Compute point correspondences between consecutive image pairs
3. Extend and verify correspondences and cameras over image triplets
4. Extend correspondences and cameras over all images
5. Optimize over $P^i X_j$

SfM recovers camera poses and 3D points. However, the reconstructed 3D points are usually sparse, containing only distinctive image features that match well across photographs. The next stage in 3D reconstruction is to take the registered images and recover dense and accurate models using a multiview stereo (MVS) algorithm. The output of SfM (cloud of 3D points) is enough for our case so we do not need any MVS stage.

### 2.2.2 Bundler

Bundler is one of the state-of-the-art SfM systems for unordered image collections (See Figure 9). Extracting high quality 3D models from such a collection is challenging for several reasons and by developing such a SfM tool writers aim to meet these needs [24].

1. The photos are unstructured they are taken in no particular order and we have no control over the distribution of camera viewpoints.

2. They are uncalibrated the photos are taken by thousands of different photographers and we know very little about the camera settings.
3. The collections are enormous so that there may be thousands of photos to process.
4. The algorithms must be fast to reconstruct an entire city in a single day, making it possible to repeat the process many times to reconstruct all of the worlds significant cultural centers.

3D reconstruction is an off-line process that does not need to run real-time. You do it once per scene and use it as reference in every tracking experience in the same scene. 3D Map extraction with Bundler is an incremental reconstruction process that may take days with respect to the numbers and sizes of images and the size of feature match list. Bundler is not implemented on GPU and its iterative and incremental algorithm is not that convenient for parallelization. 3D reconstruction of does not need to work real-time. It is a preparation done before real-time tracking.

#### *2.2.2.1 Bundler Inputs and Outputs*

Bundler takes a set of images, image features, and image matches as input, and produces a 3D reconstruction of the camera and scene geometry as output as seen in Figure 10. The system reconstructs the scene incrementally, a few images at a time.

Figure 9: Bundler takes unordered image collections as inputs and outputs 3D map of the scene

At the end we come up with a 3D map file named “bundle.out”. Content of this map file is used to produce the point cloud of the scene and the camera poses.

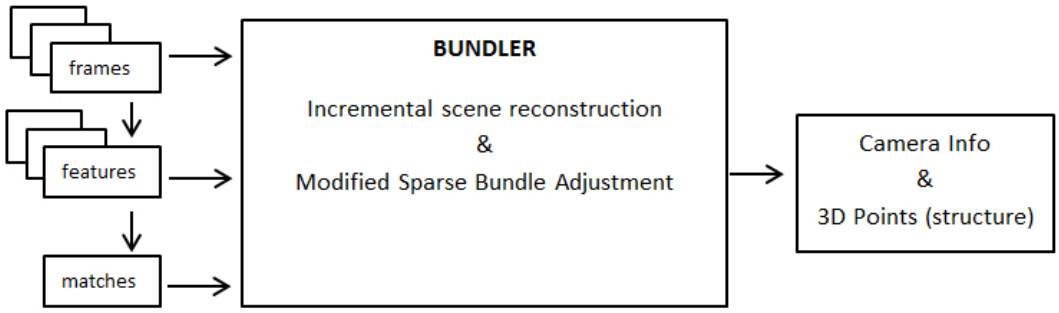



Figure 10: Flowchart of Bundler SfM Tool, inputs, process and output

To be able to evaluate the results of experiments we used same camera during all video recordings (same intrinsic parameters and same resolution) so all the frames has the same parameters. We also did some changes in Bundler software to integrate it in our platform. You can see the details of these modifications in section 4.2.

Bundler has a number of internal parameters, so there are a large number of command-line options [25] [13]. There are a number of other options in addition to the default ones listed below in Table 3.

The parameters set we used in our experiments is told in section 4.2.

#### *2.2.2.2 Running Bundler*

Bundler works with images in JPEG format [25]. The easiest way to start using Bundler is to use the included bash shell script, “RunBundler.sh”. The process steps in the “RunBundler.sh” file is listed below:

Table 3: Parameters of Bundler

| parameter | description |
|---|---|
| functional parameters | |
| –variable_focal_length | directs bundler to optimize for an independent focal length for each image |
| –use_focal_estimate | directs bundler to use the estimated focal lengths obtained from the Exif tags for each image |
| –constrain_focal | constrain the focal length of each camera to be close to the initial focal length estimate (from Exif tags). This option adds penalty terms to the bundle adjustment objective function |
| –constrain_focal_weight 0.0001 | weight on the penalty terms for the focal length constraints (a small weight is typically sufficient) |
| –estimate_distortion | directs bundler to estimate radial distortion parameters for each image |
| –run_bundle | directs bundler to estimate radial distortion parameters for each image |
| file and folder settings | |
| –match_table matches.init.txt | specifies the file where the match files are stored |
| –output bundle.out | specifies the name of the final output reconstruction |
| –output_all bundle_ | specifies that all intermediate reconstructions should be output to files with prefix "bundle_" |
| –output_dir bundle | the directory all output files should be written to, typically called "bundle" |

1. Set base path (BASE_PATH) and other file directories

2. Create image list as txt file (list.txt)

3. Extract focal from a xml list (f)

4. Update image list by including f

5. Run SIFT detection and create key files (imageName.key )

6. Run key match and create match file(matches.init.txt)

7. Set options of Bundler

8. Run Bundler with the files created and get output file (bundle.out)

Bundler itself is typically invoked as:“>bundler list.txt –options_file options.txt” The first argument is the list of images to be reconstructed (created with the “extract_focal.pl” utility). Next, an options file containing settings to be used for the current run is given. “RunBundler.sh” creates an options file that will work in many situations.

#### *2.2.2.3 Output Format and Scene Representation*

After all possible images have been registered and 3D reconstruction completed, Bundler outputs a final file named “bundle.out”.

Bundler also produces a “ply” file containing the reconstructed cameras and points is written after each round. These “ply” files can be viewed with several viewer tools. The bundle files contain the estimated scene and camera geometry as in Listings 2.3 [25]:

Listing 2.3: Bundler File Format

```
# Bundle file v0.3
<num_cameras> <num_points>    [two integers]
<camera1>
<camera2>
    ...
<cameraN>
<point1>
<point2>
    ...
<pointM>
```

Each camera entry “cameraI” contains the estimated camera intrinsics and extrinsics. It includes focal length, two radial distortion coefficients, 3x3 rotation matrix and translation vector as a 3-vector.

Each point entry has the form as in Listings 2.4.

Listing 2.4: Bundler 3D Point Format

```
<position>      [a 3-vector describing the 3D position of the point]
<color>         [a 3-vector describing the RGB color of the point]
<view list>     [a list of views the point is visible in]
```

You can see the details in official documentations [25]. The coordinate system of Bundler is illustrated in Figure 11.

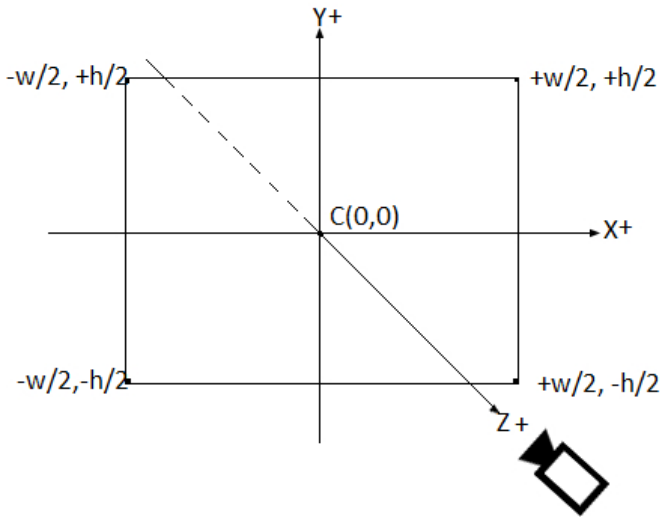



Figure 11: The coordinate system of bundler takes center of frame as coordinate center.

The pixel positions are floating point numbers in a coordinate system where the origin is the center of the image, the x-axis increases to the right, and the y-axis increases towards the top of the image. Thus, $(-w/2, -h/2)$ is the lower-left corner of the image, and $(w/2, h/2)$ is the top-right corner (where $w$ and $h$ are the width and height of the image). In addition, in the camera coordinate system, the positive z-axis points backwards, so the camera is looking down the negative z-axis. Bundler uses a pinhole camera model; focal length ($f$), two radial distortion parameters ($k1$ and $k2$), a rotation ($R$), and translation ($t$). The formula for projecting a 3D point

$X$ into a camera $(R, t, f)$ is:

$P = R * X + t$ (conversion from world to camera coordinates)

$p = -P/P.z$ (perspective division)

$p' = f * r(p) * p$ (conversion to pixel coordinates)

where $P.z$ is the third $(z)$ coordinate of $P$. In the last equation, $r(p)$ is a function that computes a scaling factor to undo the radial distortion:

$r(p) = 1.0 + k1 * ||p||^2 + k2 * ||p||^4$

Finally, the equations above imply that the camera viewing direction is:

$R' * [00 - 1]'$ (i.e., the third row of $-R$ or third column of $-R'$, where $'$ indicates the transpose of a matrix or vector) and finally the 3D position of a camera is:

$-R' * t$ .

## *2.3 Sensor Fusion and Tracking*

Sensor fusion is the combining of sensory data or data derived from sensory data from disparate sources such that the resulting information is in some sense better than would be possible when these sources were used individually. The term better in this case can mean more accurate, more complete, or more dependable, or refer to the result of an emerging view, such as stereoscopic vision (calculation of depth information by combining two-dimensional images from two cameras at slightly different viewpoints) [26].

Basic block diagram of the whole process of fusion of measurements from Inertial Measurement Units with visual data from cameras is shown in Figure 12.

Because the results of both IMU and camera measurements contain errors, a special method has to be used to combine the results from both sources is fused using an Extended Kalman Filter (EKF). This method heavily relies on accurate modeling (in the form of process and observation models) of the system [27] [28]. EKF consists of two main steps: predict and update. The predict phase uses the state estimate from

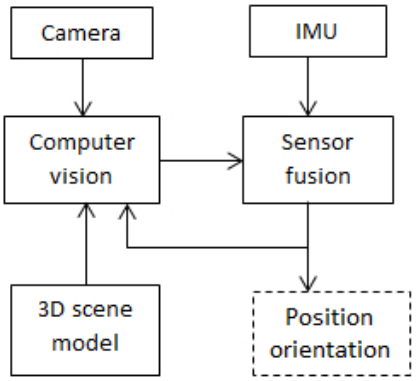



Figure 12: Block diagram of a camera and sensor based tracking system

the previous time step to produce an estimate of the state at the current time step. This predicted state estimate is also known as the a priori state estimate because, although it is an estimate of the state at the current time step, it does not include observation information from the current time step. In the update phase, the current a priori prediction is combined with current observation information to refine the state estimate. This improved estimate is called as posteriori state estimate. Typically, the two phases alternate, with the prediction advancing the state until the next scheduled observation, and the update incorporating the observation [29].

### 2.3.1 IMU Sensor and Camera Calibration

To have a successful camera and IMU sensor fusion and use both sensor data together for a better tracking we need to know the calibration parameters between two sensors [45]. If calibration could not successfully done the error of motion estimation will cumulatively increase by time and cause inaccuracy on pose estimation. Figure 13 shows the coordinate system used for calibration process. We assumed that the z-axis of world coordinate system is parallel to gravity vector.

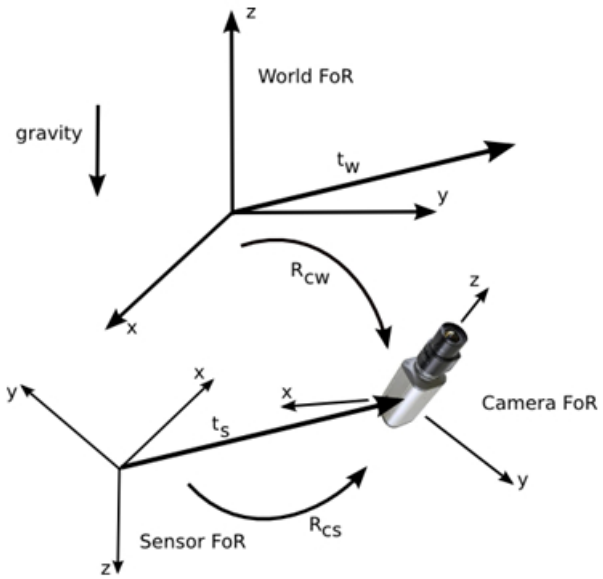



Figure 13: Camera, world and IMU coordinate systems

For calculations we used a set of calibration frames and IMU measurements captured from different angles (Figure 14). Camera position is found on these frames with RANSAC method.

Figure 14: Frame samples used for calibration

### 2.3.2 IMU Sensor and Camera Synchronization

It is crucial to synchronize two sensors before fusing them. Reliability of sensor fusion and pose estimation results depends on the time rates and relation between

the sensor measurements. There are two alternatives for synchronization: software solution and hardware solution. Software solution is easier to apply but it is error-prone and success depends on the external factors such as software method used and speed of the hardware configuration running it. In the hardware solution the devices communicate directly over hardware so the delay amount will be less but more predictable than the software way.

In our setup we use Firefly MV FFMV-03M2C camera unit supports ) IIDC v1.31 Trigger Modes 0 and 3 trigger modes. There is a 7-pin GPIO trigger connection used to trigger camera to capture frames. The IMU device is STM 66 STEVAL_MKI062V2 that can give logical outputs via its GPIO pins. These pins connected with cameras pins to trigger camera and have synchronized sensor data. One of the difficulties while trying to run sensor devices together in a sync mode was the difference in frame rate capacities. IMU sensor supports high rates (up to 120 Hz) but camera supports (up to 60Hz). We had to do some researches on hardware specifications and did some experiments by changing device firmware and configuration softwares in order to come up with an accurate sync trigger.

## *2.4 3D Rendering and Visualization*

A detailed information about AR applications is provided in Chapter 1. For graphics rendering and visualization some open source tools like are commonly used. Ogre(Object-oriented Graphics Rendering Engine) [30] is one of these tools. It supports a number of common file formats, but also makes use of Ogre specific file formats. Cam Pose and sparse point cloud (3D points) created by any SfM system and the virtual 3D animation characters are given to Ogre3D. Ogre combines 3D Map information, frames and 3D animation character together. User can navigate in all directions inside the scene via mouse and keyboard.

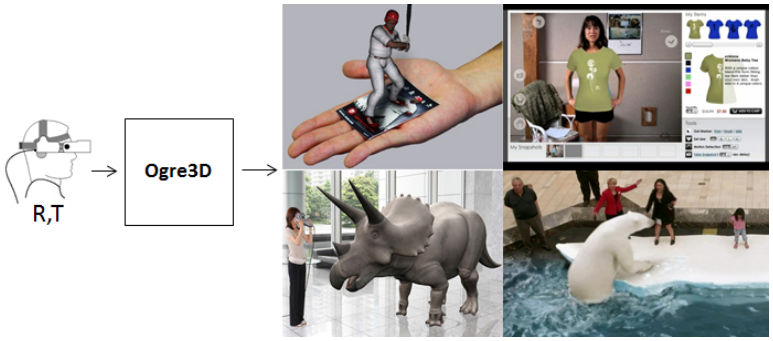



Figure 15: A sample illustration of AR applications on OGRE using camera information

## *2.5 Chapter Summary*

We provided a brief overview of Structure from Motion tool Bundler, SIFT and it's graphical processing unit accelerated version SiftGPU and sensor fusion for tracking. The details of software platform presented by us are given in Chapter III.

# CHAPTER III

# PLATFORM DEVELOPMENT

This chapter provides an overview of the components of the developed platform, the hardware setup and the graphical user interface application.

We touched on some requirements of the targeted application in previous chapter. Also presented some solutions and decisions to overcome these issues. Here we describe the details of the system designed and developed. This chapter briefly touches on the other modules presented by the main project to give an idea on overall 3D map extraction, AR and camera tracking life-cycle.

## *3.1 Development Environment*

The need for real-time tracking and the data rate to be processed of the system directed us to develop a system with C++ language to get high performance processing capability. Some of the open-source tools we used such as Bundler and SiftGPU are also developed in C++.

In order to have a better and easy operation we developed graphical uses interfaces using QT C++. Other tools and technologies used are: MATLAB, MS VS 2008, C++, OpenCV, OpenGL, CUDA and some open source CV tools such as SiftGPU and Bundler. SiftGPU and Bundler are also using some other third party tools inside.

## *3.2 Overall System Design*

Design of the platform is illustrated in Figure 16.

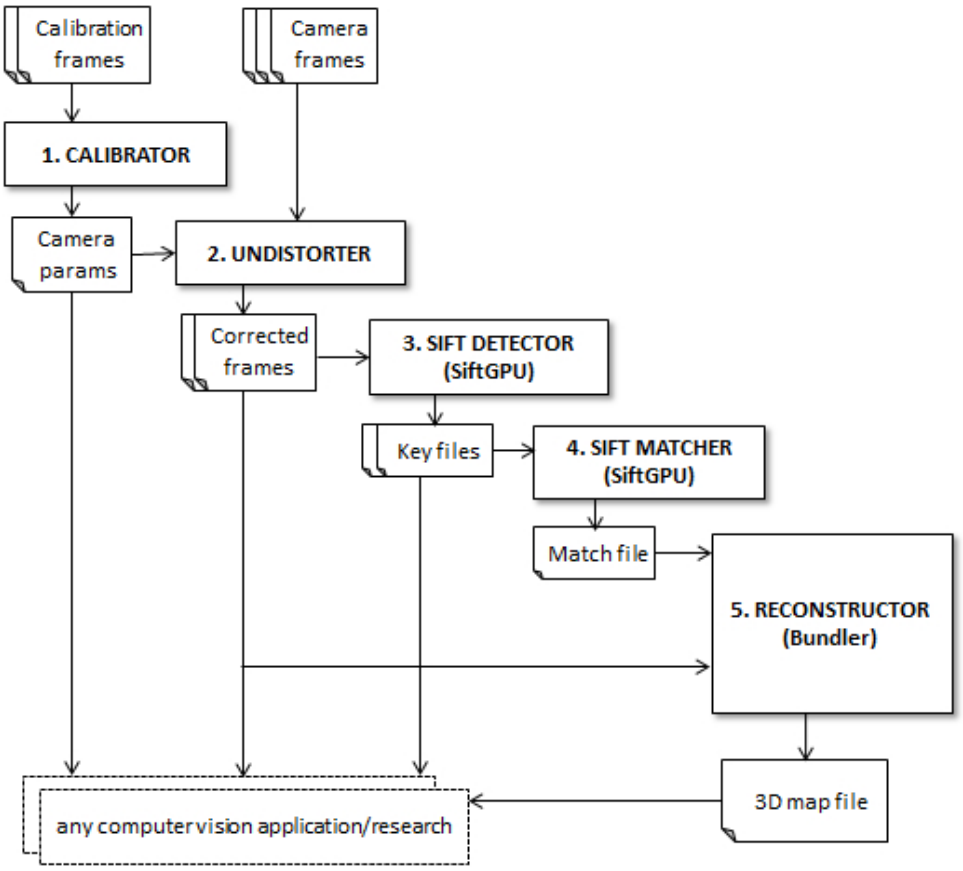



Figure 16: Flowchart of overall system

Flow of the system can be summarized as below:

1. Camera calibration is done to estimate camera intrinsics by Calibrator
2. Camera frames are radial-undistorted and resized to fit undistorted content by Undistorter
3. Features on each frame are detected using SIFT Detector
4. Features are matched by SIFT Matcher
5. The 3D Space(map) and camera pose information is extracted by Reconstructor
6. Any other external application consumes the output of the system

System is designed modular, so any module can be easily changed, new methods can be easily added. We only show the main modules here. These modules have some sub-software modules inside, but we do not touch these object oriented software design concepts to keep it more comprehensible. All the components except the graphical user interface application are built as dynamic link libraries (dll) so that they can be used in any other application. We call the executable application as "ArOZU" which is the short version of "Augmented Reality Tool of Özyeğin University". Detailed descriptions of the system will be given in the upcoming chapters.

## *3.3 Description of the Components and Data Flow*

Table 4 lists the components of the developed platform, their inputs, functions and outputs.

Table 4: Components of the System

| component | input | output |
|---|---|---|
| Calibrator | calibration frames | camera matrix including focal length |
| Undistorter | camera frames | undistorted frames |
| SIFT Detector | corrected(undistorted) frames | Sift key files |
| SIFT Matcher | SIFT key files | SIFT match file |
| Reconstructor | corrected frames + SIFT match file | 3D camera poses and 3D feature points |

### 3.3.1 Data Acquisition

To record camera frames we used a HMD setup with the inertial sensors (IMU) and cameras in a coupled approach designed for a 3D EKF tracking research. In our experiments we just used the saved frames coming from one of the cameras of this setup. Camera frames are stored as ".pgm" files.

Figure 17: Camera and IMU combined on same media and mounted on an industrial safety helmet

### 3.3.2 Calibrator

Calibrator is the module responsible for camera calibration. Calibration stage is important to extract camera intrinsic parameters. A special set of frames (calibration frames) recorded by camera are given to this module and the camera parameters are estimated as output.

### 3.3.3 Undistorter

Next stage after camera calibration process is Undistortion. By the help of this module the camera frames will be undistorted and resized to remove the radial distortion caused by camera lenses.

### 3.3.4 SIFT Detector

Sift Detector is a wrapper module over GPU based SIFT (Scale-Invariant Feature Transform) detection implementation tool called SiftGPU [18]. This utility module gets real-time camera frames as input to detect features (GPU based SIFT implementation) and gives detected features and 128-byte descriptor vectors as output.

### 3.3.5 SIFT Matcher

SIFT Matcher has feature matching skills. Does pairwise across each pair of frames using approximate nearest neighbor matching. Matched features are saved in a text file.

### 3.3.6 Reconstructor

Reconstructor is the 3D Map constructer module. The work flow of mapper is:

1. Gets video file as input
2. Uses SiftGPU and Bundler
3. Gives reconstructed 3D Map of space
4. 3D Map is saved once and used during all tracking process

This module is a wrapper module that calls SiftGPU and Bundler methods inside. SiftGPU is used to detect and match features on the frames and provides input to SfM tool Bundler. Bundler is used to create 3D Map of the environment. Simply, takes detected features and matches of multiple frames as input and gives 3D structure information as output.

The basic usage of Bundler on Windows platforms is done over the tool called Cygwin simulating the linux environment [25]. The actual purpose of using Cygwin is to run linux scripts on Windows platform. But user can write Windows script files for the same purpose. In Cygwin based scripts third party tools are used to do SIFT feature detection and matching processes. Bundlers executable gets the outputs of detection and match processes as input with some user defined parameters. Detailed description of Bundler tool and usage is told in Section 2.2. In our case we did some basic modifications on original Bundler code to build it as a dynamic linked library (dll). So we could integrate it with our own user interface application written in QT C++. This method came up with the opportunity to debug in Bundler code. SIFT

related modules are employed to do SIFT detection and match stuff as fast as it can by using GPU card skills. Bundler takes created file list and match file as input parameters. Bundler, also do focal length extraction and radial distortion jobs if its configuration says to do. In our case we use radial undistortion and focal extraction pre-applied images. Brief information about this pre-process is given in Section 4.2. All the images are taken from same camera setup, so all the frames has the same focal length value. We pass this fixed focal length value and undistorted frames to decrease the complexity and increase the reliability of 3D map extraction process.

### 3.3.7 3D Map

Mapper employs Bundler and creates 3D information of the scene. This information is stored in a file named bundle.out. 3D Map basicly contains :

- the camera information of each frame containing the focal length f, radial distortion parameters, rotation matrix and translation vector
- the 3D points found containing the 3D position of each point and the list of frames and feature indexes related with the 3D point

The details of bundle files are told in Section 2.2.

### 3.3.8 External Applications

The products or sub-products of our system can be used in any application or research. For example the 3D map created by our platform is used as input in an EKF based sensor fusion research.

## *3.4 User Interface Development*

Third party tools such as Bundler or SiftGPU are generally used by some shell scripts via command prompt. Running reconstruction process rely on bash and perl being installed. User need to know the usage of these tools and other external tools such

as Cygwin to run the scripts written to run the executables and share data between them. By developing the application as a Windows desktop application aimed to provide an easy to use interface and isolate the user from the confusing details.

### 3.4.1 Development Platform

All the tools we used are C++ based, so the user interface is developed with QT which is a C++ based graphical user interface development toolkit. It has a very well documented SDK and wide variety of demos and samples [31].

### 3.4.2 Windows and Use-cases

Figure 18 shows the main screen of the application.

Main screen of our AROzu has three main parts. Toolbar(1), Main Pane(2) and Progress Pane(3). Main Pane hosts the widgets developed for calibration, feature detection, feature match and SfM steps. There are four tool buttons on the Toolbar opening the widgets: "Calibration and Lens Undistortion Window", "SIFT Detection and Match Window", "Reconstruction Window" and the "Batch Experiment Window". Progress information of the processes is listed in the right pane.

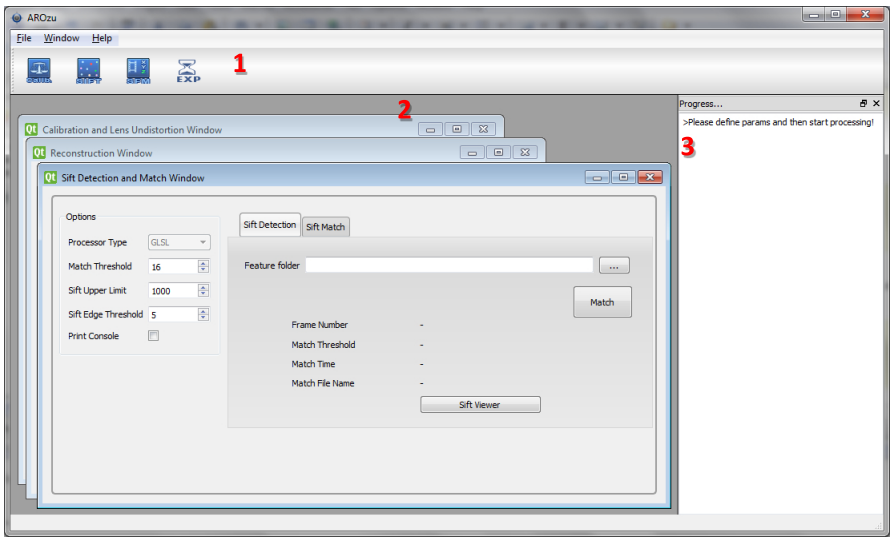



Figure 18: ArOZU main secreen

#### *3.4.2.1 Calibration and Lens Undistortion Widget*

Calibration button on toolbar opens the window used for camera calibration and lens undistortion (See Figure 19). Camera calibration runs the Calibration module with the parameters set in "calibration pattern settings" part on a set of frames containing a checkerboard pattern. After the calibration process the camera intrinsics and the coefficients vector estimated will be displayed on the screen. During camera recordings depending on the camera lenses some amount of distortion (radial distortion) happens on frame. This artifact should be removed before a successful experiment. Lens (radial) distortion is corrected in the Lens Undistortion panel. This operation also crops the content of the frame to remove the effects (black content occurred around the frame) of undistortion. Parameter settings and brief descriptions of these two stages are listed in Table 5.

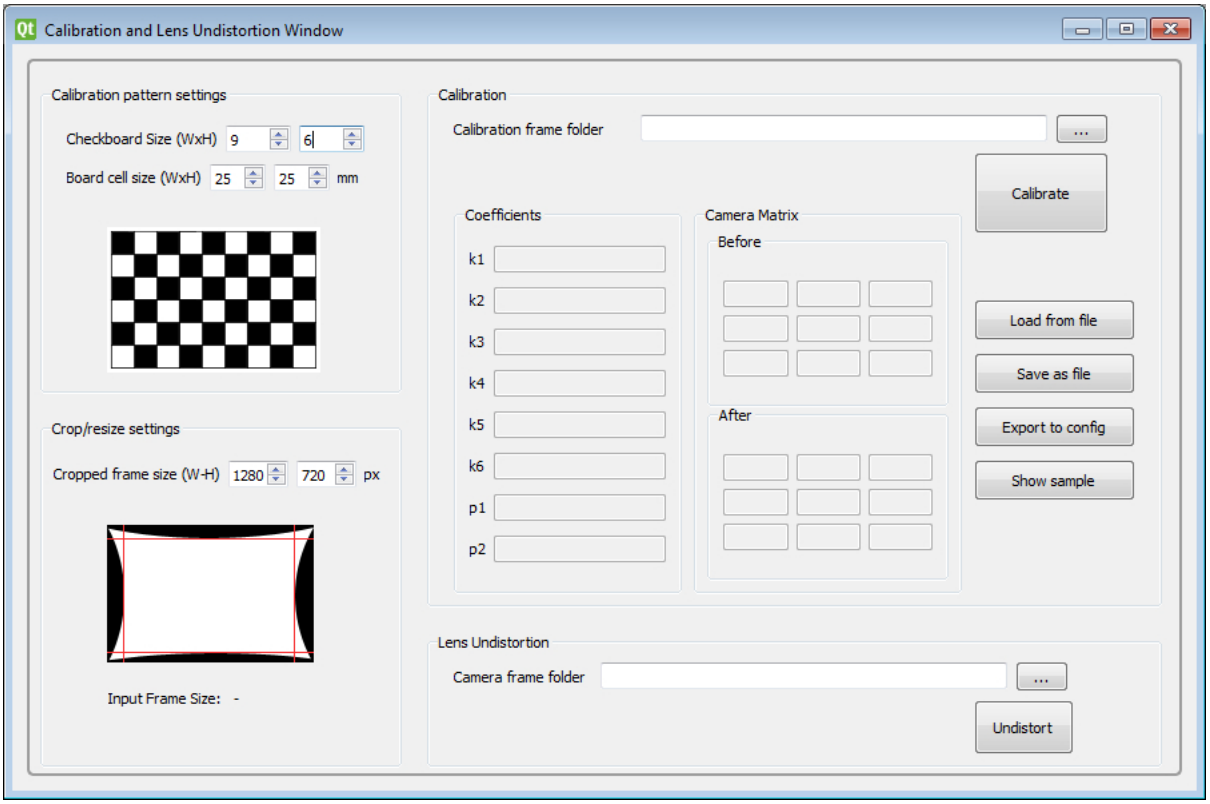



Figure 19: Calibration screen

After the undistortion process camera matrix is updated with respect to the new form of the frame.

Table 5: Parameters of calibration and undistortion

| component | description |
|---|---|
| Calibration frame folder | Folder of calibration frame set |
| Checkerboard size | Number of columns(X) and rows(Y). 9x6 is set as default. |
| Board cell size | Size of each cell in mm |
| Cropped frame size | Size of frame after undistortion |
| Camera frame folder | Folder of frames to be undistorted |

There are some other buttons on the screen for different functionalities as described in Table 6.

Table 6: Functionalities of buttons on calibration screen

| button | functionality |
|---|---|
| Load from file | Loads pre-calculated calibration parameters from a file |
| Save as file | Exports the calibration parameters as a file |
| Export to config | Sets the configuration (focal length) with new values calculated |
| Show sample | Opens a viewer screen to see sample frames (before/after) |

#### *3.4.2.2 SIFT Detection and Match Window*

SIFT button on the toolbar is for opening SIFT Detection and Match Window. This window calls the Sift Detector module's functions inside. Options pane on this screen

is for parameter settings of detection and match operations. Parameter settings and brief descriptions are listed in Table 7.

Table 7: Parameters of SIFT detection and match

| component | description |
|---|---|
| Frame folder | Folder of frames to detect features |
| Feature folder | Folder including the key files generated during detection |
| Match threshold | Threshold used for limiting the minimum match count between frames |
| SIFT upper limit (ML) | Integer value set for limiting upper limit of features detected |
| SIFT edge threshold (ET) | Another way of limiting the detection. Higher the threshold higher the feature count. |
| Print console | Print process details on console screen during run |

First tab of this window is for feature detection (Figure 20). On this tab user manually chooses the frame folder to apply SIFT detection and starts the process. After Detect button finishes its action, some statistics such as frame number, total detection time and the name of the Matlab data file (".mat") file that contains detection time per frame of the process is shown on the screen. To see the detected SIFT features marked on the screen user can press the SIFT Viewer button and open a display screen. A screenshot of SIFT Viewer Window during experiments can be seen in Section 4.2. User can view the list of detected features of each frame and features marked image and can save this image as a "jpeg" file.

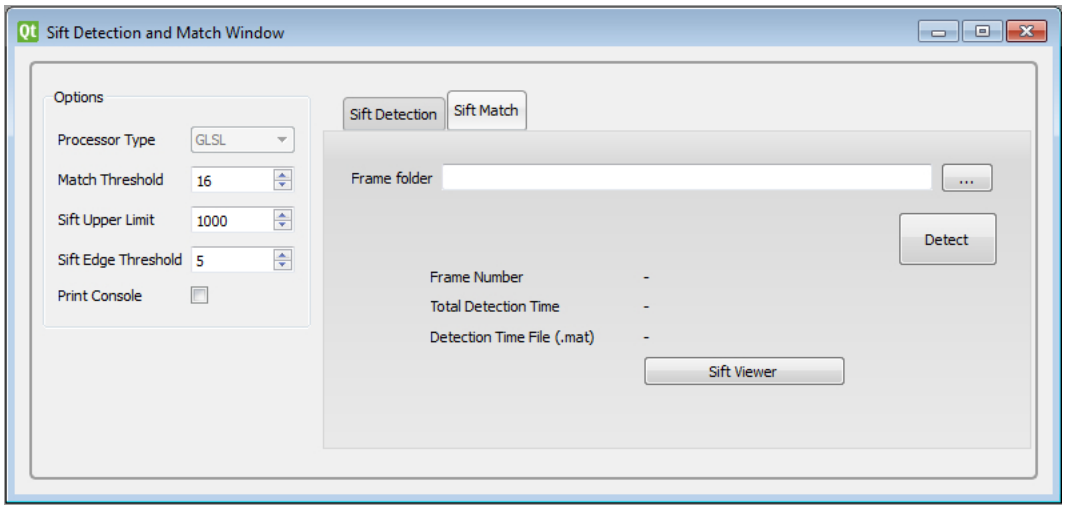



Figure 20: SIFT Widget screen (detection tab)

Next step after detecting features is to match them. Second tab on the screen (Figure 21) is for matching features and saving the result as a single file.

After choosing the feature file folder and pressing Match button, SiftGPU matches all the feature files (".key" files) and finally some statistics about the process such as frame number, total match time and the file name that contains the match result is shown on screen.

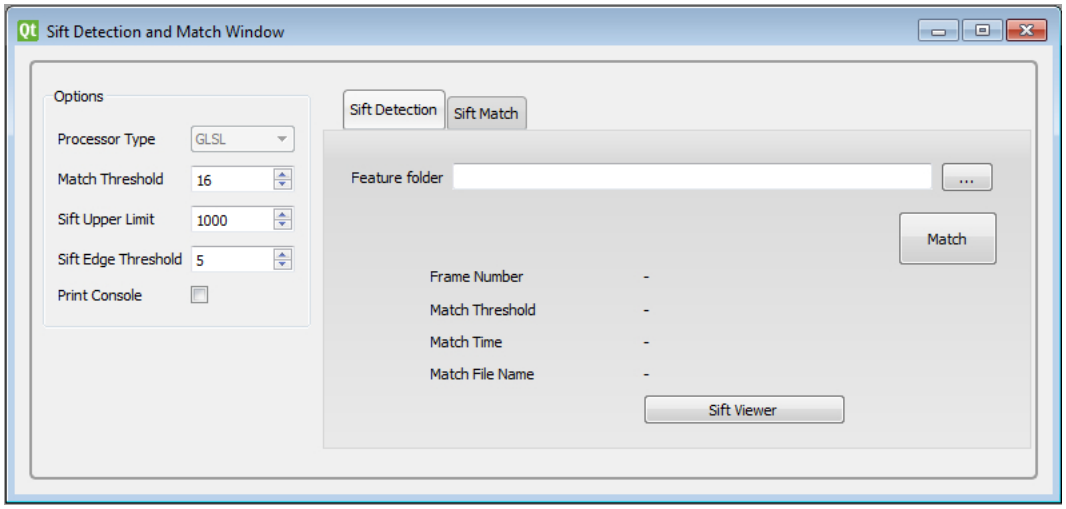



Figure 21: SIFT Widget screen (match tab)

By pressing the SIFT Viewer button after detection process completed, a window named SIFT Viewer is opened. This window consists of two main parts. The canvas that shows the 2D feature points as randomly colored dots on frame and the list of 2D points in a form of (X,Y) coordinates.

#### *3.4.2.3 Reconstructon Window*

SfM button opens the Bundler Widget window (see Figure 23) that employs MAPPER module to construct 3D Map. Bundler Widget screen provides the main functionalities needed for experiments. In first tab (Run Bundler) it is possible to start processing from feature detection (integrating SiftGPU detect and match functionalities inside) or start using ready-made feature and match files.

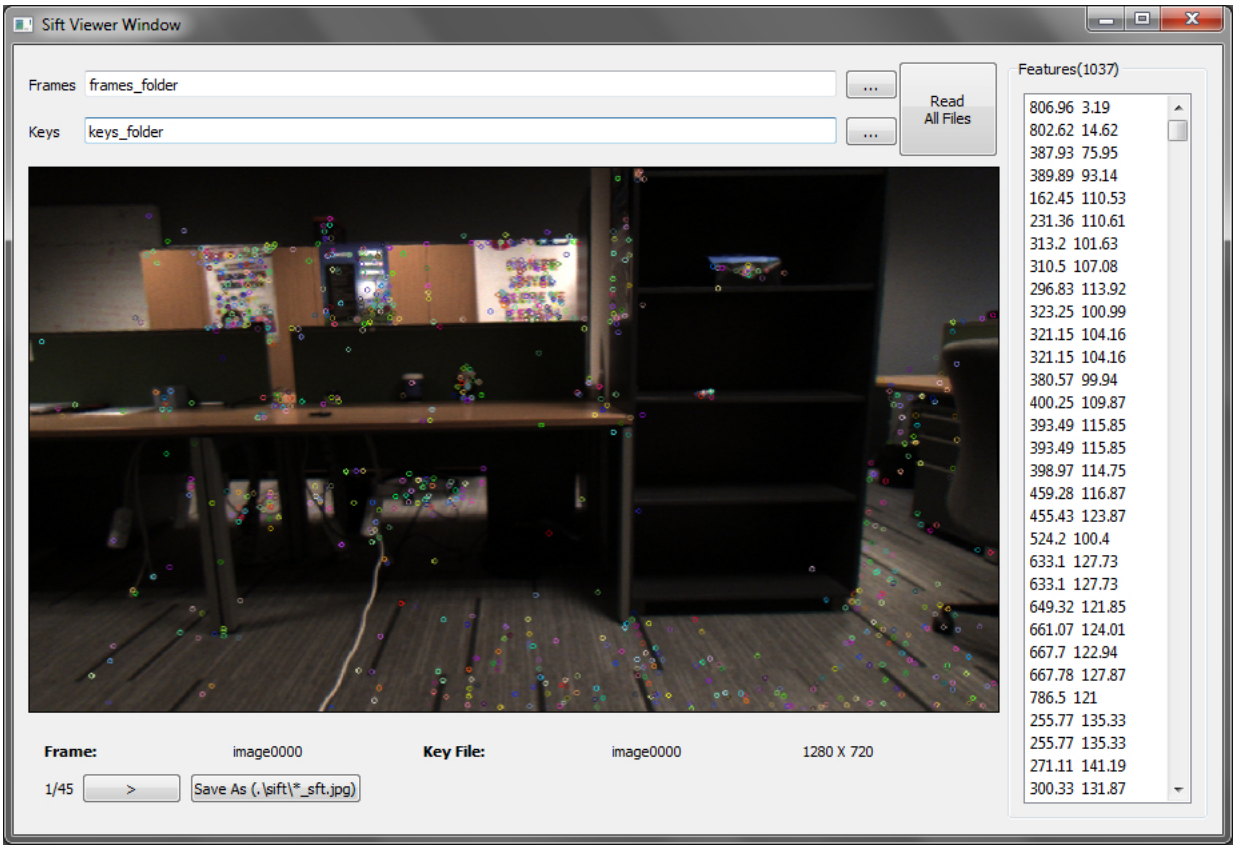



Figure 22: SIFT Viewer screen

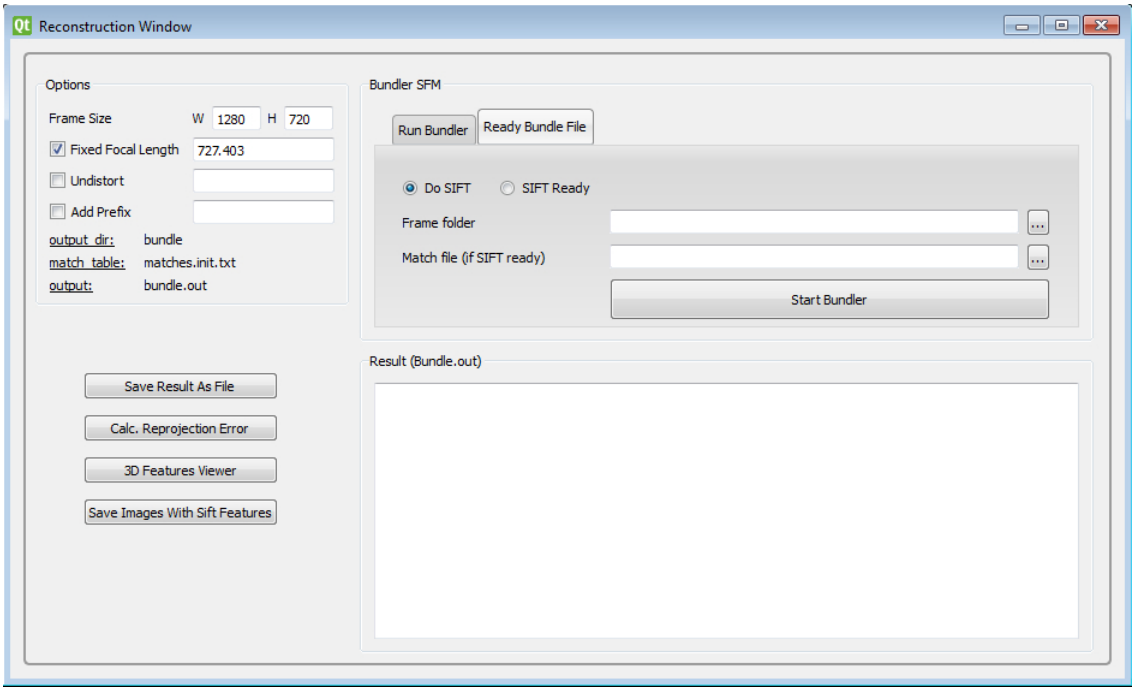



Figure 23: Reconstruction screen

As shown in Figure 24 second tab (Ready Bundle File) is for loading the results (3D map) of a previous run. As mentioned before we prefer to give calibrated frame sets (so that focal length is extracted and lens-undistortion applied on Calibration Window) instead of leaving it to Bundler. User can justify the results by applying a reprojection error threshold.

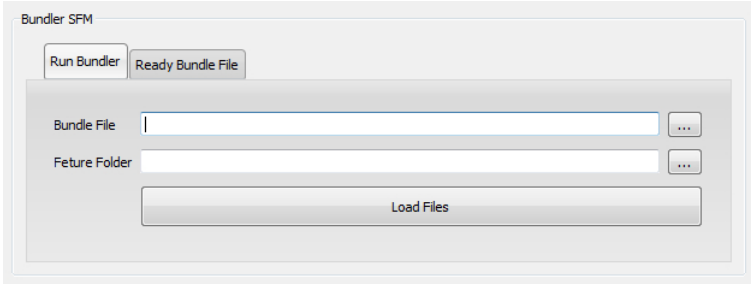



Figure 24: Reconstruction screen (ready bundle file tab)

Bundler Widget Window provides the main functionalities needed for experiments.

It is possible to start processing from feature detection or start using ready feature and match files. As mentioned before we prefer to give fixed focal length extracted in preprocessing steps instead of leaving the focal length extraction process to Bundler.

User can justify the results by applying a reprojection error threshold on the Configuration Pane. The functions of the four buttons at the bottom-left side of the window are listed in Table 9.

Table 8: Parameters of Bundler

| component | description |
|---|---|
| Frame size | Size of the input frames |
| Fixed focal length | Determines if focal length is fixed or not |
| Undistort | Determines if undistortion will be done or already done before |
| Add prefix | File name prefix for output files |
| Do SIFT / SIFT Ready | SIFT detection and match done before or not |
| Frame folder | The folder path of frames |
| Match file | The path of match file |
| Bundler file | The path of bundler file to read |
| Feature folder | The folder path contains the SIFT key files |

There are some other buttons on the screen for different functionalities as described in Table 9

Table 9: Functionalities of buttons on Bundler screen

| button | functionality |
|---|---|
| Save Result As File | Exporting the output of the SfM process |
| Calc. Reprojection Error | Calculate reprojection error of 3D map |
| 3D Features Viewer | View 3D map results and 2D features |
| Save Images With SIFT Features | Save frame with features marked on it |

Bundler Viewer Window provides a viewer screen that user can see the 3D features and 2D features with respect to the related frame. When user moves mouse on a dot (3D point) 3D point information is shown in a tooltip. User can also view the 3D distance (with Bundler's unit) between two dots by dragging mouse between them. Info panel lists the camera pose information and the 3D points extracted for the current frame. User can see these 3D points in 3D by pressing the 3D Render button at bottom-left of Bundler Viewer screen. A 3D rendering window using Ogre opens and user can navigate in 3 dimensions in the 3D point cloud.

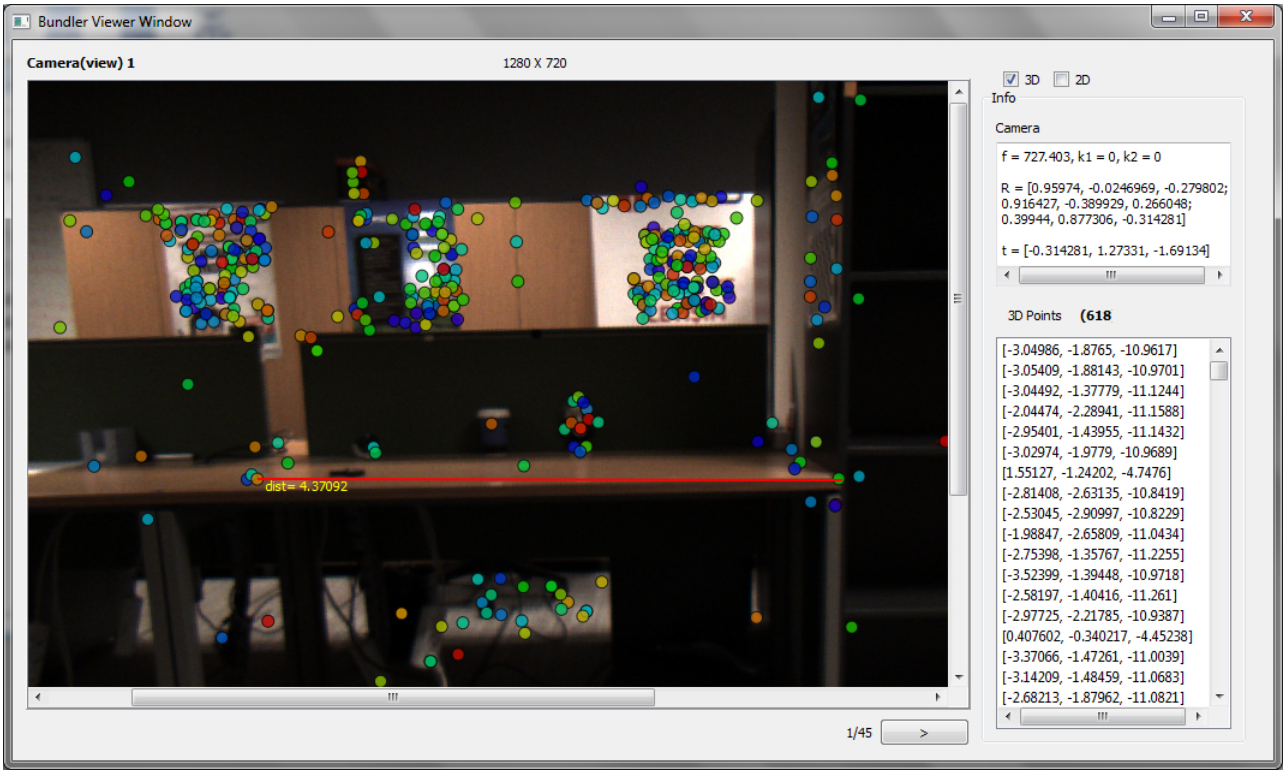



Figure 25: Bundler Viewer screen

Bundler viewer basically projects extracted 3D points as 2D points on the frame. If user wants to see the points in a 3D scene an Ogre based navigation window by pressing the 3D View button. User can move in all directions inside the scene via mouse and the direction keys on keyboard.


Figure 26: Ogre 3D Render window

#### *3.4.2.4 Batch Experiment Window*

Experiments including feature detection, match and SfM stages takes long time for big number of frames. And sometimes user needs to run same experiment a few times to calculate the average of process times. For experimental purposes such as batch processing and iterations, another window is provided. A prepared xml file including the frame and feature folder paths and parameters needed for SIFT and SfM processes can be loaded and processed as a batch for experiments.

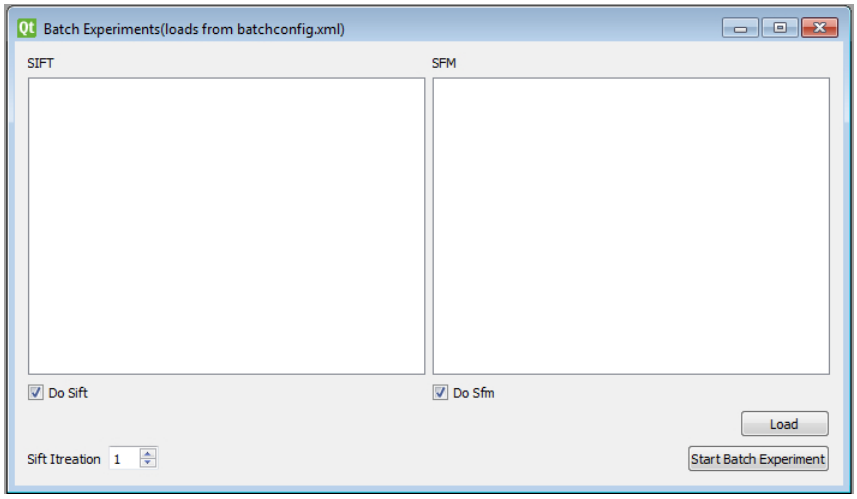



Figure 27: Batch Experiment screen

## *3.5 Chapter Summary*

We designed and developed a desktop software for use of Computer Vision researchers, especially working on camera pose tracking and augmented reality subjects. Chapter III provided details of this software platform. Our platform used and tested on Extended Kalman Filter based tracking for augmented reality experiments. Information and results of these experiments are given in Chapter IV.

# CHAPTER IV

# EXPERIMENTS AND RESULTS

Experiments done using our platform and results are provided in this chapter. There are three main stages followed in order: "Camera Calibration and Radial Undistortion", "Detecting Features and Matching Them" and finally the "Structure from Motion".

## *4.1 Preprocessing of Data*

### 4.1.1 Setup Configuration and Data Record

We designed the software as it can be used in both indoor and outdoor applications. But the experiments are done in laboratory environment. As described in Section 2.2, Bundler takes a bunch of multiple frames and feature point lists of each frame as input. These inputs stand for the motion information. Structure of the scene is constructed using these motion data. SfM process takes long time and the number and the sizes of frames, and the amount of the features effects the processing time. The quality of frames effects the performance of the feature extraction process that provides input to SfM. Quality here stand for the video motion speed, resolution of frames and the lighting conditions can cause noise or blur effect on image data.

While recording we tried to be realistic that camera motions are not to slow and not too fast. We did recording in a closed room has windows getting sunlight inside and the lights were on. Although the surfaces of furniture were not that reflective but we still had some white reflected frames. Our setup has two cameras on it. Because it is not necessary for SfM to use stereo pairs coming from two cameras, we used only frames coming from one of the cameras.

We combined all frames from different short records and came up with a frame

set of 3540 frames that corresponds to 118 sec of 30fps video. Table 10 lists some statistics about the records.

Table 10: Numbers on records

| Infromation | Value |
|---|---|
| Rate of camera | 30 Hz |
| Total duration of record | 118 sec |
| Total # of frames | 3540 frames |
| Recording resolution | 1600 x 1200 |
| Resolution after pre-processing | 1280 x 720 |

To see the effects of the number of frames on results we created 5 different frame sets sampled from whole set of 3540.

1. Frame set contains 708 frames - every 5th of the frames (0,5,10,...,3530)
2. Frame set contains 304 frames - every 10th of the frames (0,10,20,...,3530)
3. Frame set contains 177 frames - every 20th of the frames (0,20,40,...,3520)
4. Frame set contains 89 frames - every 40th of the frames (0,40,80,...,3520)
5. Frame set contains 45 frames - every 80th of the frames (0,80,160,...,3520)

### 4.1.2 Hardware Platform Configuration Running the Application

Setup connections and configurations, video recordings and experiments are all done on the same hardware platform. Table 11 shows the configuration of the platform. It is a powerful workstation with large processing and memory capacities. Graphic card was employed for GPU acceleration.

Table 11: PC hardware configuration

| Property | Detail |
|---|---|
| CPU | Intel i7-3930K 3.20 GHz |
| Memory | 16 GB |
| GPU | NVIDIA GeForce GTX 680, 4095MB, 1536 Cores, Driver 311.06 |
| HDD | 1 TB |
| OS | Windows 7 Proefessional SP1 64 Bit |

### 4.1.3 Camera Calibration and Radial Undistortion

Our cameras add some amount of lens-distortion on frames. Bundler has a parameter that activates a undistortion step, and another parameter setup to estimate focal length of frames using a camera model dictionary in the form of xml file.

In our experiments we preferred to use our own undistortion and focal-length estimation methods instead of leaving it to Bundler. In our case all the frames are from same camera. So the focal length of all frames are same and fixed.

To extract the camera parameters we used a calibration pane with checkerboard pattern. This pattern has 6x9 squares having dimensions of 5x5 cm. 30 frames captured from different angles are used for calibration. In Figure 28 three sample calibration frames can be seen.

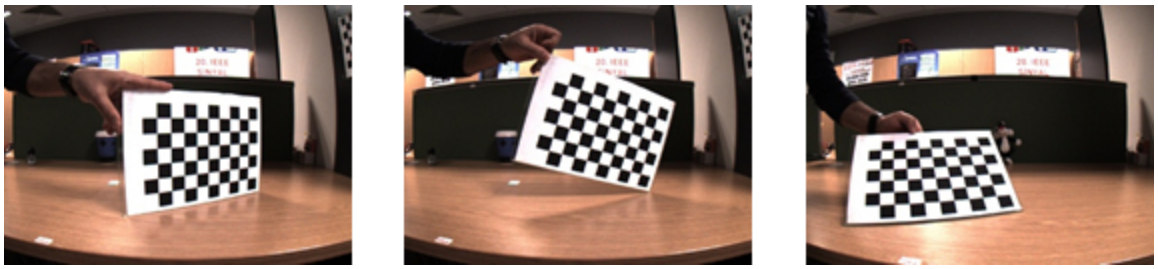


Figure 28: Sample frames used for camera calibration

The calibration window of the application is shown in Figure 29.

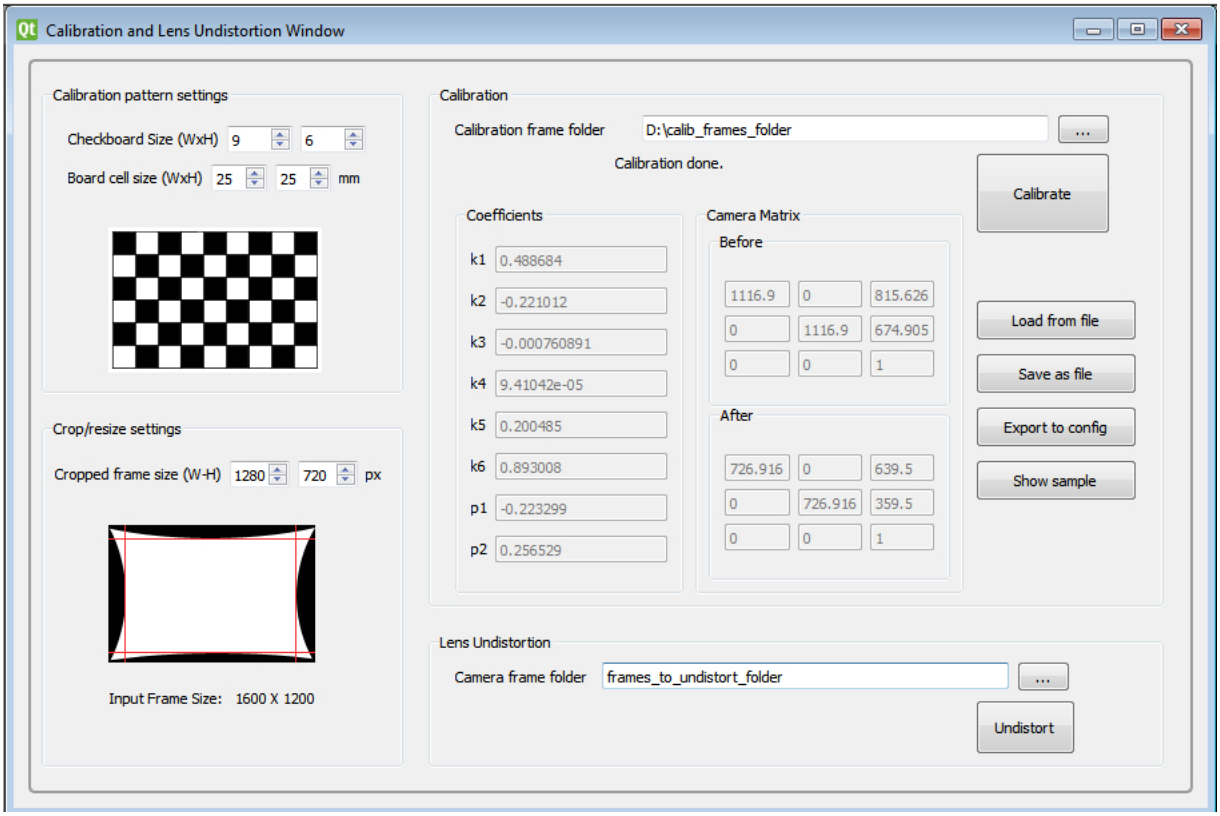



Figure 29: Screenshot of Camera Calibration window

After calibration we come up with the camera parameters including the focal length (f) and the 8 distortion parameters. Table 12 lists the parameters found after calibration. These parameters are saved as configuration parameters used to

undistort and rectify the frames. Extracted focal length will be used as input of 3D feature detection and 3D map extraction processes.

Table 12: Camera parameters extracted

| Focal length and translations | Distortion parameters |
|---|---|
| | $k1 = 0.48636$ |
| | $k2 = -0.14510$ |
| $fx = 1116.3$ | $k3 = 0.09204$ |
| $fy = 1116.3$ | $k4 = 0.08288$ |
| $cx = 816.3$ | $k5 = -0.11004$ |
| $cy = 674.7$ | $k6 = 0.10856$ |
| | $p1 = -0.00073$ |
| | $p2 = 0.00001$ |

**Lens undistortion process** is a kind of distortion applied to overcome the distortion caused by camera lens. So some portions of images become invisible and the aspect ratio and dimensions are changed after a crop operation. After resizing undistortion and rectification size of frames is reduced from 1600 X 1200 to 1280 X 720. In Figure 30 a sample frame can be seen.

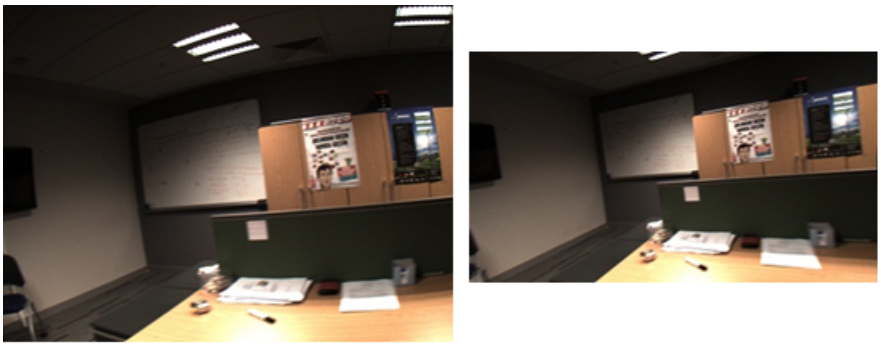


Figure 30: Before (left) and after (right) un-distortion and rectification

After undistortion process the distortion parameters listed in Table 12 becomes 0 and the new focal length is calculated as:

$$f = 727.403$$

## *4.2 3D Map Estimation*

After preprocessing steps and applying undistortion process and estimating the focal-length, the next step is extraction of 3D map. In addition to frame set alternatives listed in Section 4.1.1 we have three experiments done as listed below:

1. Experiment: Changing frame number (see Section 4.1.1) while ET and ML parameters are at default values (5.0 and 1000)
2. Experiment: Changing ML parameter (500, 750, 1000 and 1500 features) while ET is at default value (5.0)
3. Experiment: Changing ET parameter (2.5, 5, 7.5 and 10) while ML at values (1000 and 2000)

Results of these experiments and some comparison graphs are provided in next sections.

### 4.2.1 Detecting Features and Matching Them

First step of 3D structure reconstruction and tracking is feature detection. The details of SIFT and SiftGPU can be seen in Section 2.1.

To examine the effect of the "Maximum Feature Number Per Frame" (ML) parameter on results we applied 4 cases of this parameter value as 500, 750, 1000 and 1500 features. We also done another experiment to examine the effect of the "Edge Threshold" (ET) parameter on results we applied 4 cases of this parameter value as 2.5, 5.0, 7.5 and 10 (see Table 13 for these experiments).

Table 13: SIFT parameter values examined

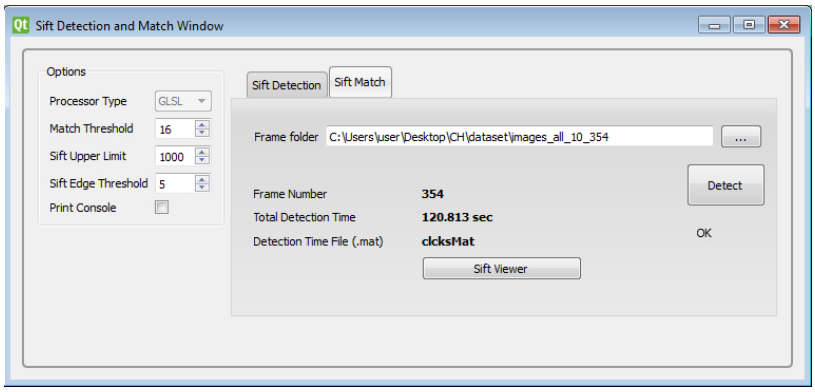
| ML | ET |
|---|---|
| 500 | 5.0 |
| 750 | 5.0 |
| 1000 | 5.0 |
| 1500 | 5.0 |

| ML | ET |
|---|---|
| 1000 | 2.5 |
| 1000 | 5.0 |
| 1000 | 7.5 |
| 1000 | 10.0 |

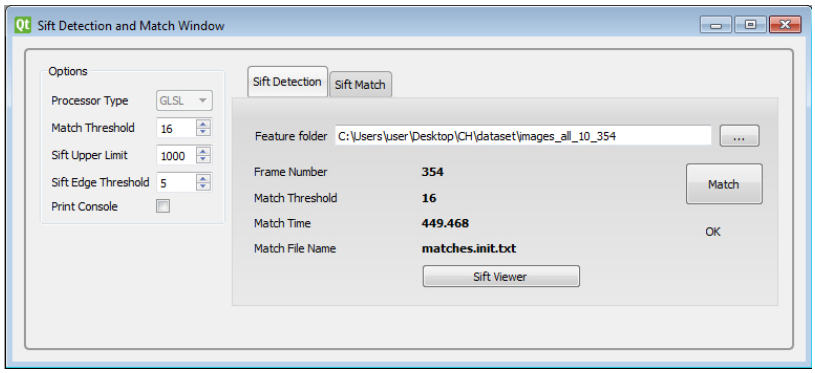
| ML | ET |
|---|---|
| 2000 | 2.5 |
| 2000 | 5.0 |
| 2000 | 7.5 |
| 2000 | 10.0 |

For each frame, SIFT detection process creates a text file located in the same directory with the frame. These text files have an extension of ".key". These key files contain SIFT keys, the 2D feature point information and descriptor of each feature point. File names have a format of "[framename].key".

For the set of 354 (every 10th of all records) frames, with respect to the content and the quality of the frame we got from 44 to 1263 feature points changing from frame to frame. The screenshot of SIFT Window is shown in Figure 34.




(a) SIFT Detection Tab




(b) SIFT Match Tab

Figure 31: SIFT Detection and Match window

SIFT Viewer screen including detected features is shown in Figure 32. Feature points are marked with randomly colored small circles drawn on frame. As shown, there are 1037 featured detected for sample frame (the first frame named as "image0000.jpeg").

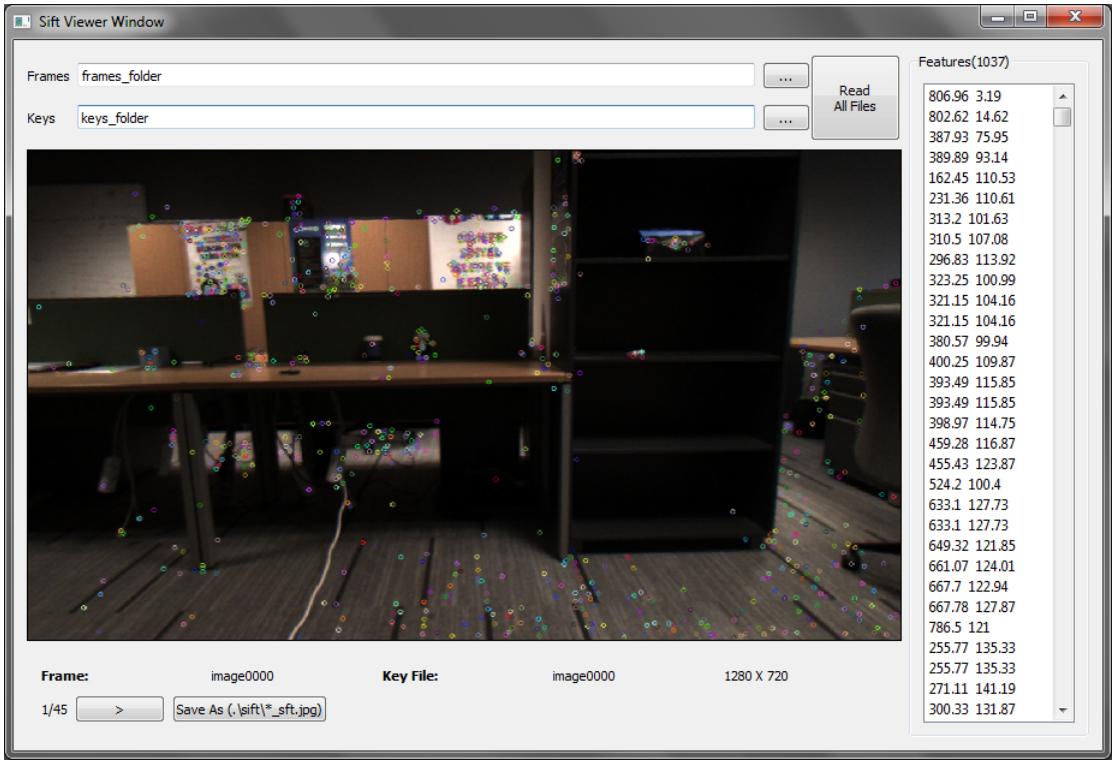



Figure 32: SIFT Viewer screen

The content of one key file of the frames is shown in Listings 4.1.

Listing 4.1: A sample portion of a sample key file

```
1037 128    // 1037 feature points detected and each feature has a descriptor array of 128
3.19 806.96 1.417 3.306  // first point's X, Y coordinates and distortion values (not used)
0.01948869 0.01035761 0.01700283 0.00020683 ... // array of descriptor
...
14.62 802.62 1.266 1.443 // second point's X, Y coordinates and distortion values (not used)
0.00001684 0.00017596 0.00193922
...
// continues until 1037th point
```

Figure 33 shows the number of features detected per frame. It changes from 44 points to 1263 points depending on the quality of the frame.

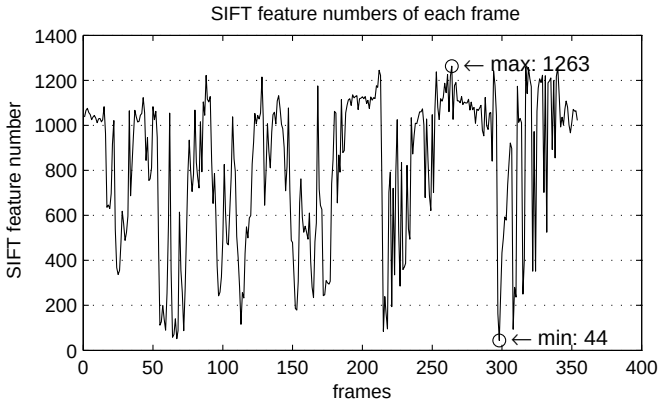



Figure 33: Feature numbers detected per frame

To reconstruct 3D map and find depth information from 2D frames we need to find correspondences between features of different frames. This matching process is the second step of our 3D structure reconstruction. It is also implemented using SiftGPU's respective functions. Matching is done frame by frame by comparing the descriptors of features. To get a more reliable match lists we applied a threshold that checks for a minimum number of matches (in our case it is 15).

After matching step, a new file named as "matches.init.txt" is created. This single file contains all the frame to frame feature point matches. For the frame set of 354 used in SfM process the text file of matches took 63.5 MB of space. In case of whole set of 3540 frames the file size is about 640 MB that any text editor application could not be able to open and view it.

During the runtime all the feature arrays of frames detected by SiftGPU and 3D map reconstructed by Bundler are also stored in custom data structures on computer memory. For the memory limits we chose to not to whole matches array in memory. Matching process is done between two frames, found matches are appended to text file and another pair of frames are matched and so on. Bundler takes this match file as parameter and parses it.

A sample portion of match file "matches.init.txt" is shown in Listings 4.2.

Listing 4.2: Match file content sample

```
0 1       // frame 0 and 1 matched
794       // there are 794 feature points matched
1 0       // feature point 1 of frame 0 is matched with feature point 0 of frame 1
2 1       // feature point 2 of frame 0 is matched with feature point 1 of frame 1
...
0 2       // frame 0 and 2 matched
770       // there are 770  feature points matched
...
352 353   // frame  352 and 353 matched
765       // there are 765  feature points matched
...
```

Some numerical results including both detection and match stages of this set of 354 frames are shown in Table 14.

Table 14: Information about output data of SIFT detection and match

| Infromation | Value |
|---|---|
| Parameter set of SiftGPU | ”-fo”, ”-1”, ”-v”, ”0”, ”-tc2”, ”1000”, ”-e”, ”5.0” |
| Min # of feature points per frame | 44 points |
| Max # of feature points per frame | 1263 points |
| Avg. # of feature points per frame | 815.44 points |
| Match file size for 354 frames | 63.5 MB |
| # of lines in match file for 354 frames | 7.230.272 lines |

#### *4.2.1.1 SIFT Detection and Match Results of Experiments*

Increasing the frame number (see Section 4.1.1) while ET and ML parameters are at default values (5.0 and 1000) is also increases the total process times (Figure 34). But it does not cause a considerable change in numbers of detected features.

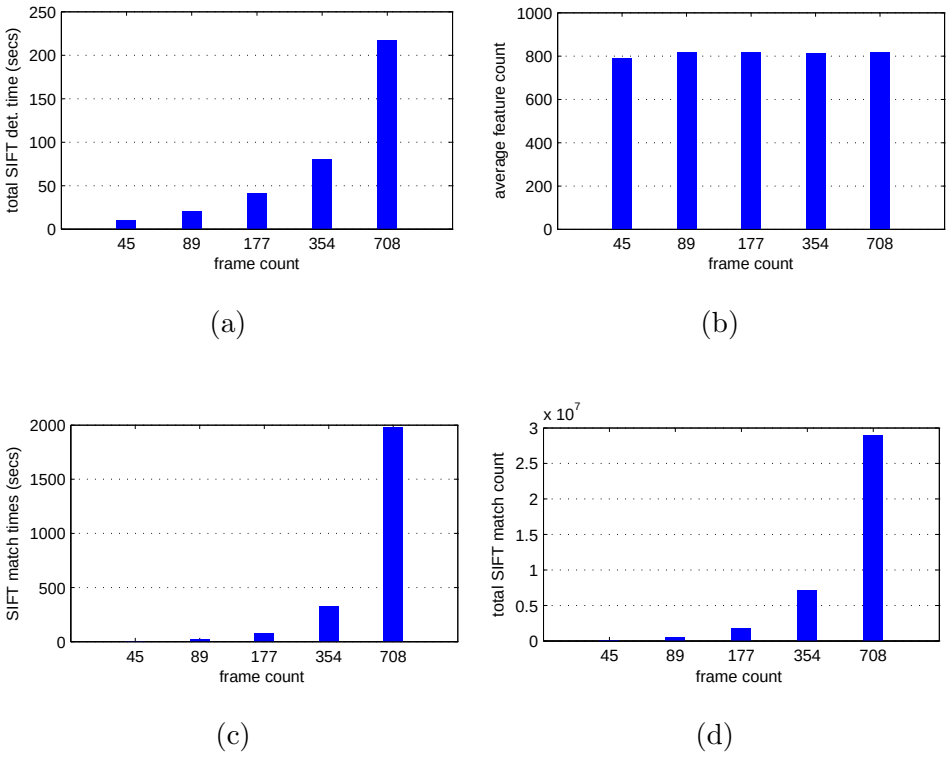



Figure 34: Experiment 1: SIFT results by frame count

ML parameter is used for limiting the maximum numbers of features to be detected (Figure 35). If we increase it detected features number increases. Because of the feature number to be detected on an image is limited, the increase is not continuous and saturates after a while.

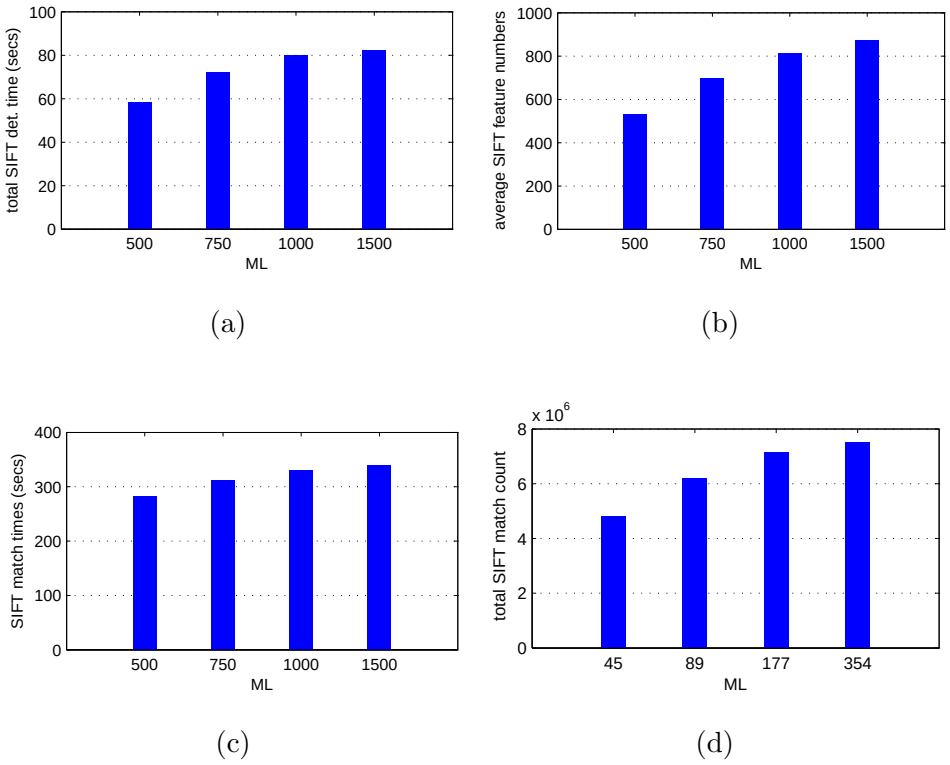



Figure 35: Experiment 2: SIFT results by ML

ET parameter is for eliminating the edge-like features in SIFT algorithm. If this value increases the detected features number increases (Figure 36). As in ML this increase saturates after a while. Both ML and ET parameters effect the the key numbers (features) and the key file sizes and so the process times of SIFT detection and match stages.

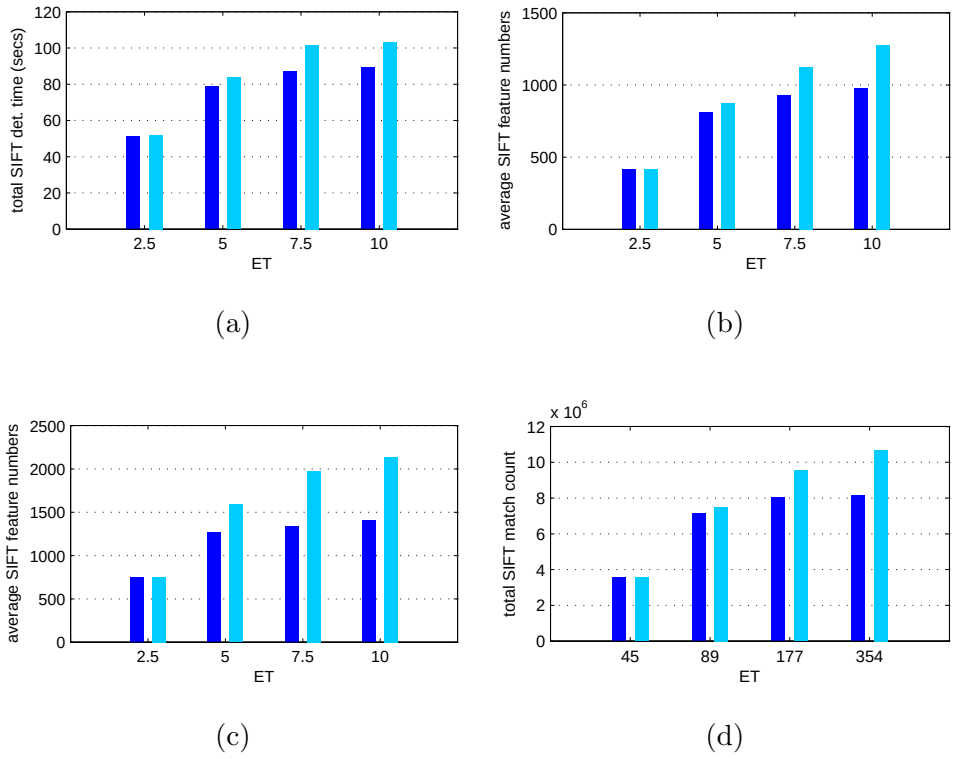



Figure 36: Experiment 3: SIFT results by ET while ML is 1000 or 2000

### 4.2.2 Structure from Motion

Bundler gets frame list and match list as input and reconstructs the scene incrementally, a few images at a time, using a modified version of the Sparse Bundle Adjustment. It took too long time (about a day) to process the list of 354 frames. At the end of the 3D reconstruction process we come up with a 3D map file named bundle.out. Reconstruction Window can be seen in Figure 37.

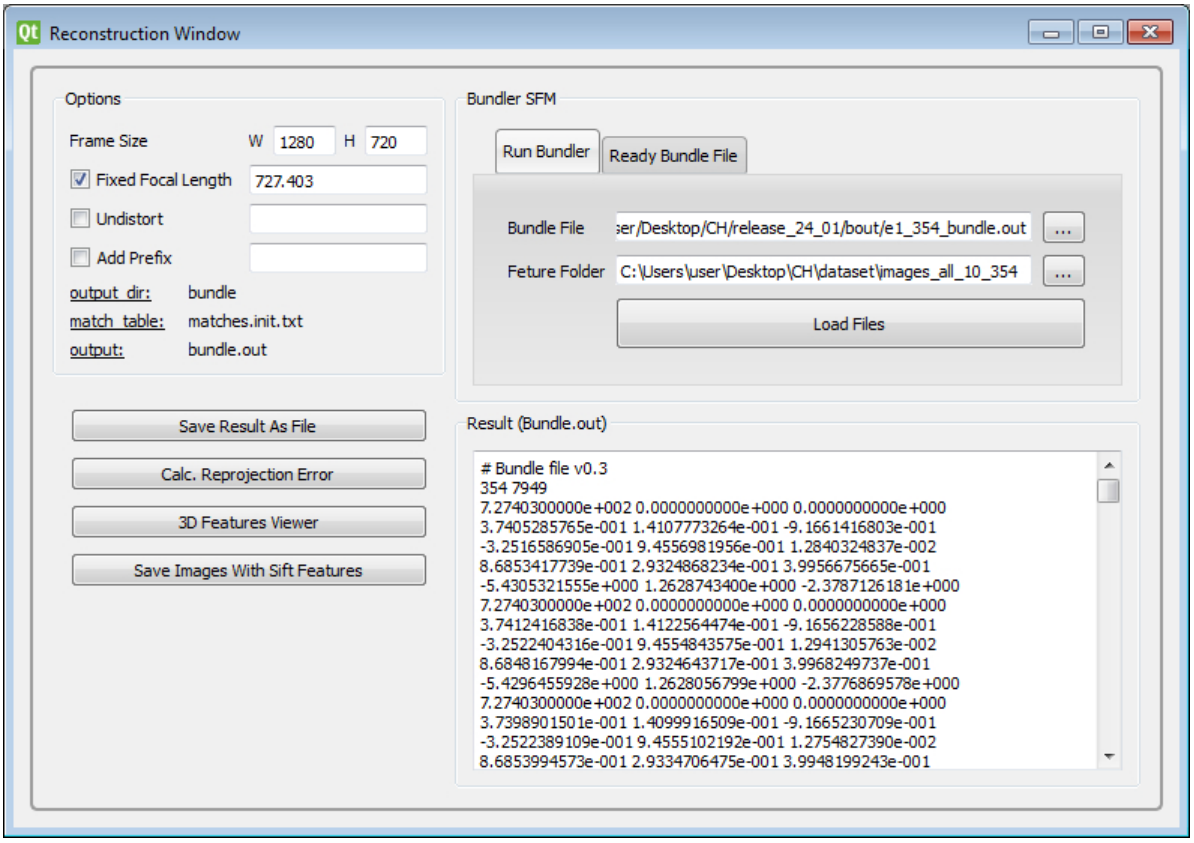



Figure 37: Reconstruction window

A sample portion of bundle.out file is shown in Listings 4.3. To see details of file format please refer to Section 2.2. It is assumed that each corresponds a different camera that has its own extrinsic and intrinsic parameters. As mentioned before we fixed the f value to 7.2740298534e+002 which was extracted by the process told in. We also found and undistorted the radial distortion effects in another pre-process. So the distortion coefficients as are automatically set to 0.0.

Listing 4.3: Content of bundle.out

```
Bundle file v0.3 // version of Bundler file format
354 7949 // there are 354 cameras and 7949 3D points
7.2740298534e+002 0.0000000000e+000 0.0000000000e+000 // first camera info
3.7405324690e-001 1.4107802249e-001 -9.1661396457e-001
-3.2516582176e-001 9.4556983331e-001 1.2840510028e-002
8.6853402745e-001 2.9324849856e-001 3.9956721744e-001
```

```
-5.4305285046e+000 1.2628757794e+000 -2.3787100785e+000
7.2740298534e+002 0.0000000000e+000 0.0000000000e+000 // second camera info
...
... // continues until 354th camera
-5.6898003631e+000 8.1801938072e-001 -8.3186371934e+000 // 1st 3D point X, Y and Z
49 43 42  // RGB color values of point
69 138 958 -517.6100 77.1800 141 796 ...//viewlist of point
... // continues until 7949th point
```

NOTE: Some camera information could not be calculated by Bundler and their matrices are returned as null (consists of 0s). This is because there are not enough feature or match information to find out rotation and translation matrixes of respective camera (frame).

Listing 4.4: Format of 3D point information

```
<position>   [a 3-vector describing the 3D position of the point]
<color>      [a 3-vector describing the RGB color of the point]
<view list>  [a list of views the point is visible in]
  <frameindex> <keyindex> <x> <y>// view 1 (x, y are Bundler's 2D projections)
  <frameindex> <keyindex> <x> <y>// view 2
  ...
```

If we apply reprojection on the 3D points listed in bundler output file, it gives a projection in pixels, where the origin of the image is the center of the image, the positive x-axis points right, and the positive y-axis points up (in addition, in the camera coordinate system, the positive z-axis points backwards, so the camera is looking down the negative z-axis, as in OpenGL). This information is important to understand and use the coordinate system of the 3D structure.

You can see the summary of 3D reconstruction process including pre-processing and feature detection/matching steps below:

1. Pre-Process the frames
2. Find the focal length from a calibration set
3. Find lens distortion coefficients and rectify all frames

4. Crop images to remove warping effects of rectification

5. Detect features and create key files of each frame

6. Do frame to frame feature matching

#### *4.2.2.1 Quality of Extracted 3D Map (RMSE)*

As described in previous section bundle.out file also contains reprojected 2D points of 3D points found. To have a successful tracking experience it is crucial to check the quality of 3D map will be used as a reference for real-time tracking. Figure 38 shows some sample 3D points reprojected frames taken from Bundler Viewer Window shown in Figure 25.

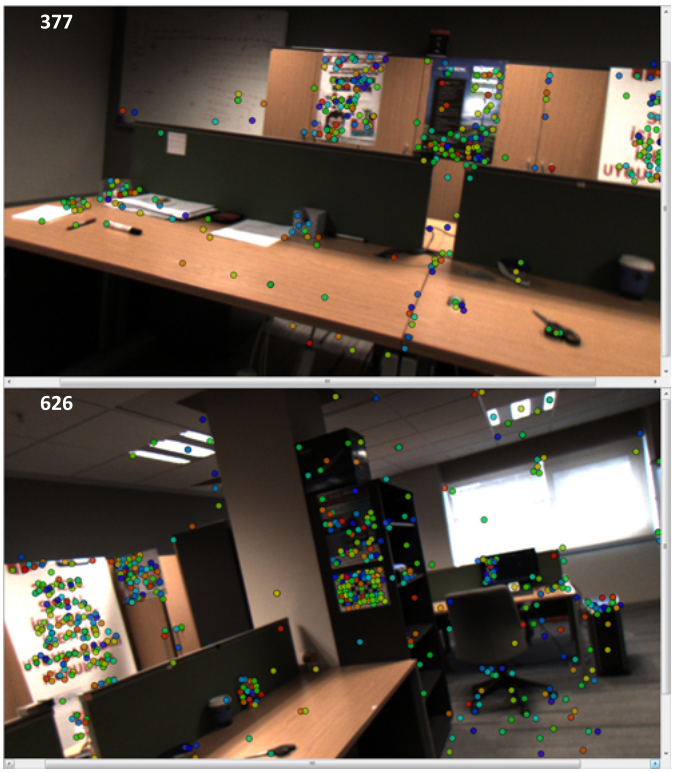



Figure 38: 3D points reprojected frame samples

To check the performance of SfM and the reliability of 3D map, we use the 2D points in key files and 3D points and camera matrices in "bundle.out" file. We applied

2D projection to all 3D points and compared them with the corresponding 2D point in key files and calculated the pixel (Euclid) distances.

Bundler's frame coordinate center is different from the SiftGPU's coordinate center. While Bundler takes the center of frame as coordinate center, SiftGPU takes it as the left-top corner. Table 15 lists the numbers used or calculated during the evaluation of the 3D map.

Table 15: Information about 3D points

| Infromation | Value |
|---|---|
| # Of Frames | 354 |
| # Of 3D Points | 7949 |
| # Of Views | 22332 (number of projections done) |
| RMSE of Distance | 1.0612 px |
| Mean Error of Distance | 0.9013 px |
| Max Distance | 7.7293 px |
| Min Distance | 0.0 px |
| Min 3D point # per frame | 0 |
| Max 3D point # per frame | 626 |
| Avg. 3D point # per frame | 63.08 |

The number of 3D points found varies for frame to frame. Figure 39 shows the 3D points per frame (see view list in Listings 4.4). At the end of the 3D reconstruction with Bundler we come up with a sparse point cloud as shown in Figure 40.

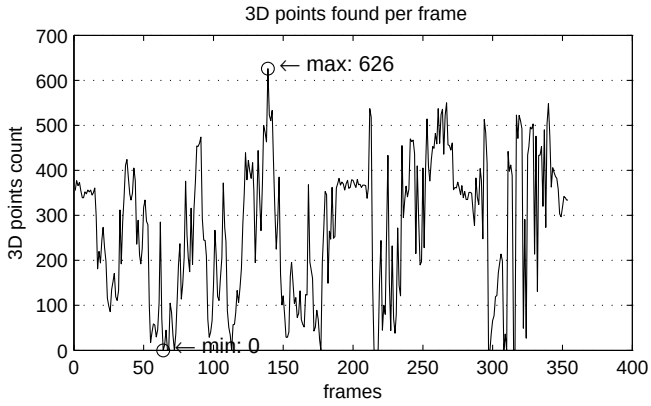



Figure 39: 3D points per frame

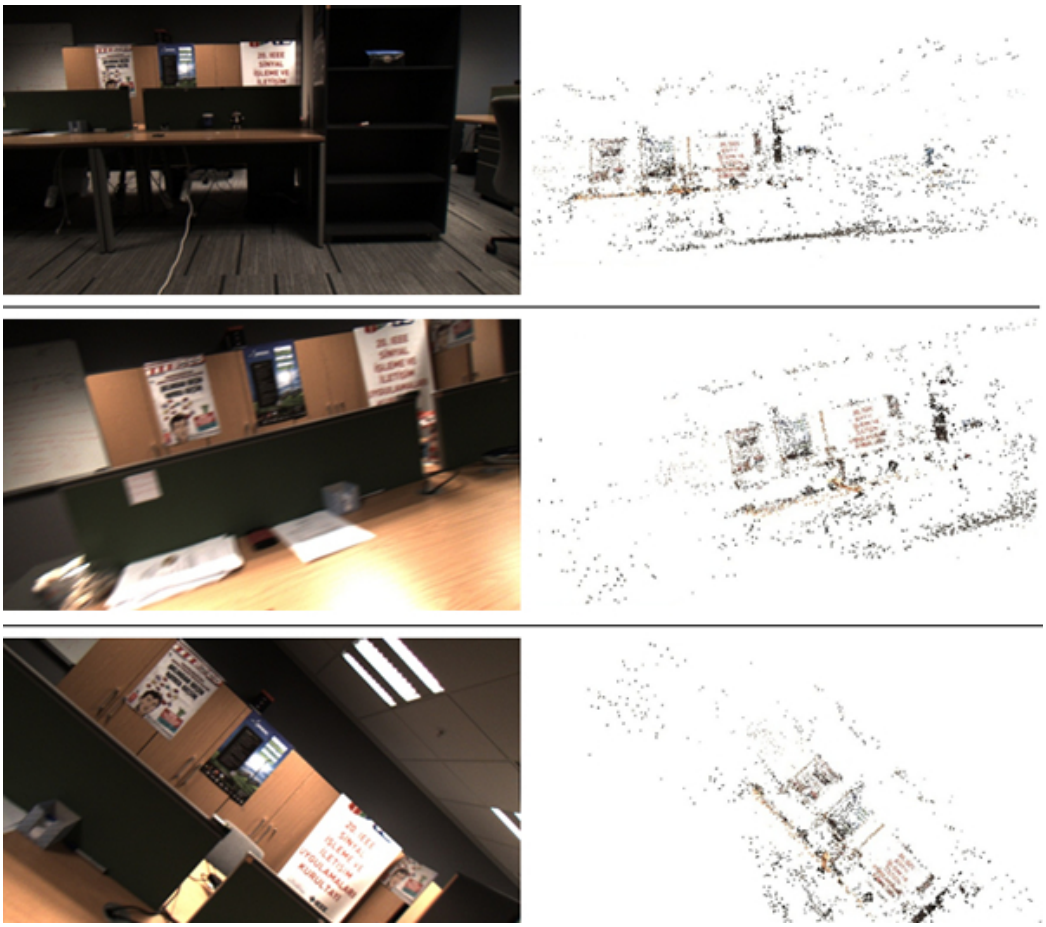


Figure 40: 3D point clouds viewed with respect to the camera pose of current frame

#### *4.2.2.2 SfM Results of Experiments*

For all of 4 experiments SfM process time and 3D points number reconstructed after SfM stage changes depending on the parameters used but the 3D reconstruction error values are below 2 pixels (Figure 41, 42, 43, 44). This results demonstrate the success and accuracy of the SfM tool (Bundler).

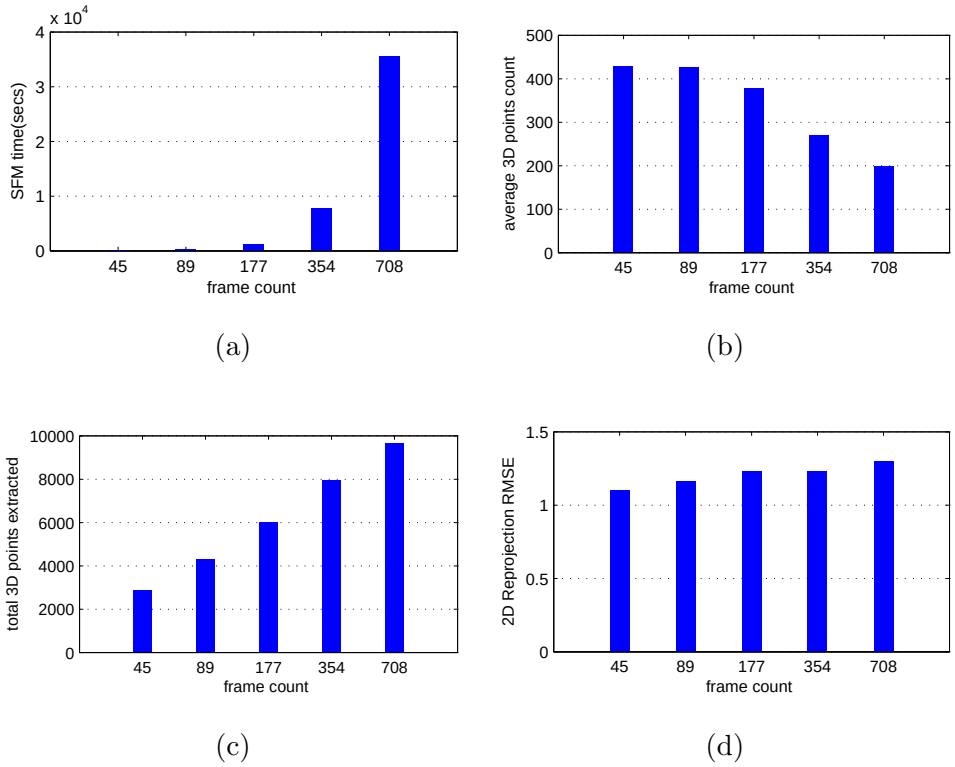
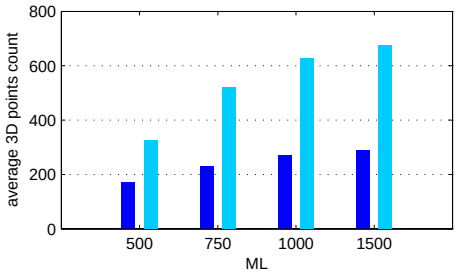


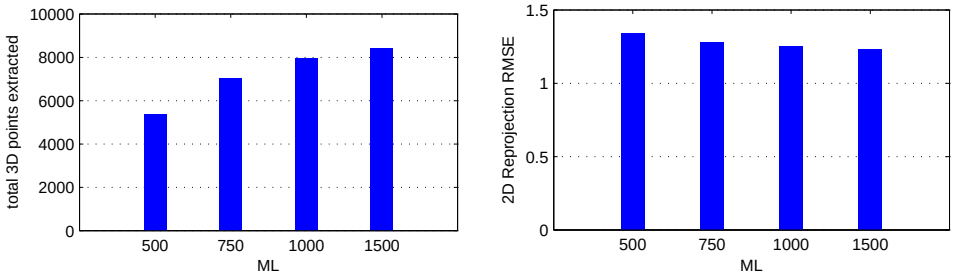

Figure 41: Experiment 1: SfM results by frame count

In Figure 41 count of total 3D points extracted directly increases while we increase the number of frames. But at the same time the average 3D points count decreases and RMSE value increases for our experiment. The quality of frames (brightness, contrast, noise level, texture level, etc.) effects the count of the detected feature numbers and the count of the extracted 3D points. If a frame is too blurred, too dark or has no enough texture on it, only a few features can be detected and as a result

no 3D points extracted. This situation decreases the average 3d points count (per frame).




(a)

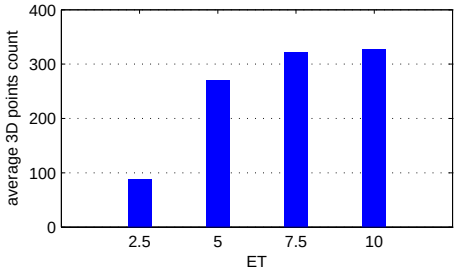



(b) (c)

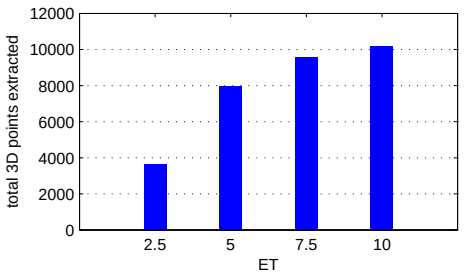
Figure 42: Experiment 2: SfM results by ML

As told in SIFT results evaluation if we increase the ML parameter, detected features number increases until a saturation level. Higher ML value results higher 3D points count and lesser RMSE value (see Figure 42).

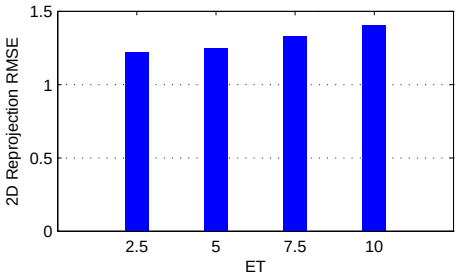
Figure 43 and Figure 44 shows the effect of ET parameter on SfM results. Any change on ML and ET parameters effect the count of features and indirectly effects the output of the SfM, 3D points count and the quality (RMSE).




(a)




(b)




(c)

Figure 43: Experiment 3/1: SfM results by ET while ML is 1000

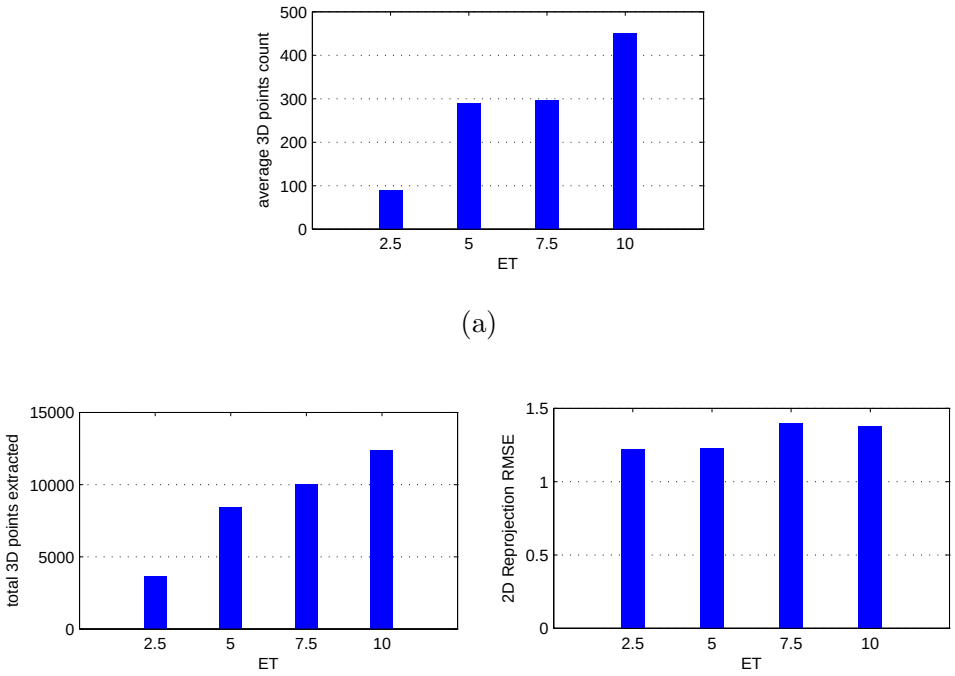
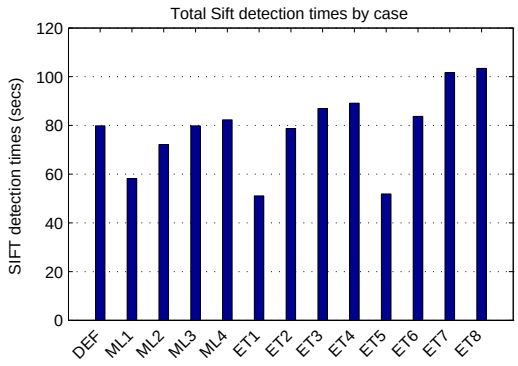


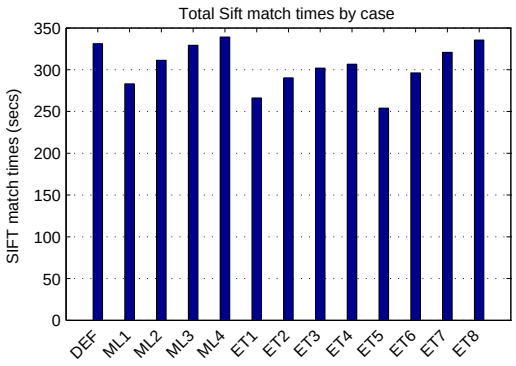

Figure 44: Experiment 3/2: SfM results by ET while ML is 2000

#### *4.2.2.3 Results of 354 Frames Set*

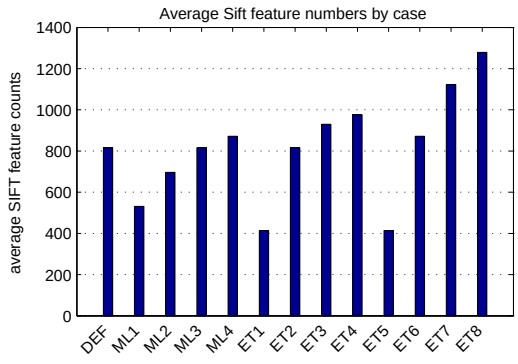
To provide a global look and comparison opportunity for all cases of parameters applied on set of 354 frames, we created graphs shown in Figure 45. Abbreviations behind the graphs stand for the ET and ML parameter cases. DEF is for default (ML=1000, ET=5.0), ML1 to ML4 stand for the cases ML= 500, 750, 1000, and 1500 cases respectively. ET1 to ET4 are cases of ET= 2.5, 5.0, 7.5, and 10.0 while ML= 1000. Rest are for ET cases while ML=2000.



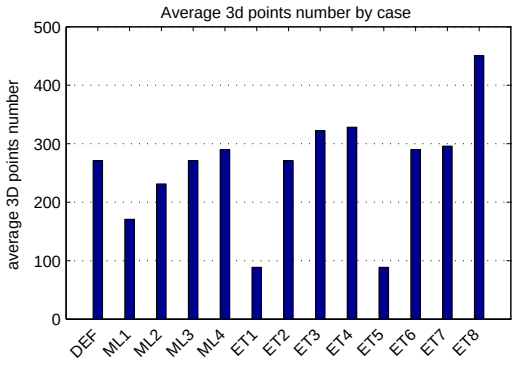

(a) Total SIFT detection times by case



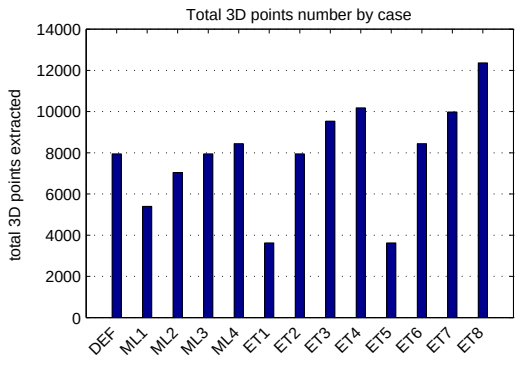

(b) Total SIFT match times by case



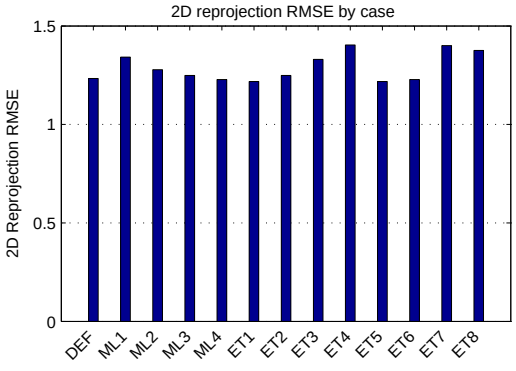

(c) Average SIFT features by case




(d) Average 3d points number by case




(e) Total 3D points number by case




(f) 2D reprojection RMSE by case

Figure 45: Comparison of all 354 frame set cases.

## 4.3 Comments on the Results

3D map reconstruction with Bundler may take long time depending on the number and resolutions of frames, but the output of the process is reliable enough for tracking applications. Real time SIFT feature detection on GPU makes it possible to have a near-real-time camera tracking.

Changes in parameters or frame number cause minor effects on the average SIFT detection time but increases the total time including detection of all frames. Average detection time (ADT) is changes between 53 ms and 57 ms. In a motion tracking or AR application the 3D scene reconstruction will be done at once at the beginning and not need to be in real-time. So the total time (3D scene reconstruction from a large frame set) is not that important for tracking and pose estimation.

## 4.4 Chapter Summary

Our platform used and tested on Extended Kalman Filter based tracking for augmented reality experiments. We tried to give a brief explanation of the test environment, hardware setup used, the processing steps and finally the results of this work. Our motivation was to present a usable and reliable software platform. Although we came up with a successful tool, we still need some improvements and developments on our system. Chapter V is touching on this subject.

# CHAPTER V

# CONCLUSION

Conclusion, contributions and future work plans are told in this chapter.

## *5.1 Conclusion*

In this thesis we designed and developed an easy-to-use all-in-one platform that provides high performance feature detection, matching and Structure from Motion functionalities. So the other researchers working on sensor fusion, augmented reality, occlusion tracking etc. could benefit from this platform as we do.

The system we present in this thesis, contains important functionalities such as robust feature detection, feature matching and extracting 3D structure from motion. Motion here stands for multiple view. System provides a user interface to do some common processes so the researchers working on sensor fusion, augmented reality, occlusion tracking etc. could benefit from this platform as we do in our own project. We benefit from some open-source computer vision tools such as SiftGPU [18] and Bundler [13] and integrate them in our user interface application, so they can easily be used.

We also did some experiments to show the usage of the system, time performance and the reliability of its functions and the effects of the main parameters of the components in results.

## *5.2 Contributions of the Thesis*

We present reliable 3D map reconstruction and real-time robust feature detection and matching utilities that every researcher working on AR and feature tracking subjects may need. So they can focus on their own subject and gain time. Output of this

software can be used as input data or ground truth of some computer vision researches and applications such as scene reconstruction, object/camera tracking, augmented reality and interaction. The output information can also be used in sensor fusion tracking experiments as corrective information to correct the drift on tracking.

Third party tools such as Bundler or SiftGPU are generally used via command prompt and have their own mechanisms and they have a lot of parameters to run. Users need to know the usage and the parameters. By developing this application as a Windows desktop application, we aimed to provide an easy to use interface. Our system contains tools for feature extraction and 3D reconstruction integrated inside. It also provides an interface to reach the functions of the modules inside via the wrapper utility functions. Bundler's or SiftGPU's outputs are in the form of text files including a lot of information. User needs to know the format details of the files in order to parse it, convert it to format he/she needs and then use it. ArOZU provides user interfaces to configure and set the processes and parse/save the files created.

There are also some interfaces we provide to view the detected 2D features on frames or extracted 3D points on respective frames.

We are going to distribute the source-code and binary of the system. Any researcher needs the functionalities provided with our system or wants to add new functionalities on it can use it.

## *5.3 Future Work*

Our platform still has some limitations:

- Current system does not support a plugin system. The only way to add a new functionality for current version is modifying the source code. Source code should be improved to support QT's built-in plugin framework for extensibility.
- In our case we did some basic modifications on original Bundler code to build it as a dynamic linked library (dll). So we could integrate it with our own user

interface application written in QT C++. If any upgrade of Bundler version is needed, similar modifications should be done in code.

- Platform is developed for 32 bit systems. So all components and the 3rd party libraries are 32bit. A 64 bit implementation will provide more memory capability.
- Current configuration will only run on Windows platform. To support Linux platforms all the components and dependencies must be rebuilt for Linux environment.

Some suggestions for future research related to the work in this thesis are the following:

- Add plugin support for extensibility
- Add 64 bit support
- Add support for other efficient SfM methods/tools and let users to add their own SfM methods
- Add support for other efficient feature detection and match methods/tools and let users to add their own feature detection modules
- Add pre-processing steps such as calibration, rectification etc. inside the platform
- Add simulation demonstrations and evaluation screens
- Integrate a 3D visualization environment inside the platform with CMVS and PMVS[9] tools
- Add more visualization and analysing tools
- Add 3D tracking and Sensor fusion methods
- GPU acceleration and system performance improvement

# Bibliography


[1] R. Azuma, "A survey of augmented reality," *Teleoperators and Virtual Environments*, vol. 6, pp. 355–385, Aug. 1997.

[2] S. Siltanen, "Theory and applications of marker-based augmented reality," tech. rep., VTT Technical Research Centre, Espoo, Finland, 2012.

[3] G. Papagiannakis and N. Magnenat-Thalmann, "Mobile augmented heritage: Enabling human life in ancient pompeii," *International Journal of Architectural Computing, Multi-Science Publishing, issue*, vol. 2, pp. 395–415, 2007.

[4] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre, "Recent advances in augmented reality," *Computer Graphics and Applications*, vol. 21, pp. 34–47, Nov. 1997.

[5] onezerothrice.com, "Augmented reality solution for flartoolkit and papervision3d," 2013. [Online; accessed 21-Nov-2012].

[6] U. of Washington HIT Lab, "Artoolkit." `http://www.hitl.washington.edu/artoolkit`, 2013. [Online; accessed 21-Nov-2012].

[7] I. ARToolworks, "Artoolkit product family." `http://www.artoolworks.com/products`, 2013. [Online; accessed 21-Nov-2012].

[8] C. Wu, "Visualsfm : A visual structure from motion system." `http://homes.cs.washington.edu/~ccwu/vsfm/`, 2013. [Online; accessed 22-Nov-2012].

[9] Y. Furukawa, "Clustering views for multi-view stereo (cmvs)." `http://www.di.ens.fr/cmvs/`, 2013. [Online; accessed 22-Nov-2012].

[10] M. Jancosek, "Cmpmvs - multi-view reconstruction software." `http://ptak.felk.cvut.cz/sfmservice/?menu=cmpmvs`, 2013. [Online; accessed 23-Nov-2012].

[11] H. Astre, "Structure from motion toolkit." `http://www.visual-experiments.com/demos/sfmtoolkit/`, 2013. [Online; accessed 27-Nov-2012].

[12] M. Vergauwen and L. Van Gool, "Web-based 3d reconstruction service," *Mach. Vision Appl.*, vol. 17, pp. 411–426, Oct. 2006.

[13] N. Snavely, "Bundler: Structure from motion (sfm) for unordered image collections." `http://www.cs.cornell.edu/~snavely/bundler/`, 2013. [Online; accessed 25-May-2013].

[14] Microsoft, "Photo tourism: Exploring photo collections in 3d." `http://phototour.cs.washington.edu/`, 2013. [Online; accessed 27-Nov-2012].

[15] P. Honkamaa, *A Library for Virtual and Augmented Reality*. VTT Technical Research Centre, Espoo, Finland, 2012.

[16] CellaGameS, "Smmt library slam multimarker tracker for symbian." `http://cellagames.com/smmt.html`, 2010. [Online; accessed 13-Feb-2013].

[17] G. Tech, "Georgia tech the designer's augmented reality toolkit." `http://www.cc.gatech.edu/dart/aboutdart.htm`. [Online; accessed 13-Feb-2013].

[18] C. Wu, "Siftgpu: A gpu implementation of scale invariant feature transform (sift)." `http://cs.unc.edu/~ccwu/siftgpu/`, 2013. [Online; accessed 07-May-2013].

[19] C. van Wyk, "Markerless augmented reality on ubiquitous mobile devices with integrated sensors," Master's thesis, University of Stellenbosch, Matieland, South Africa, Mar. 2011.

[20] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[21] C. Wu, "Siftgpu manual." `http://cs.unc.edu/~ccwu/siftgpu/manual.pdf`, 2013. [Online; accessed 07-May-2013].

[22] Wikipedia, "Structure from motion." `http://en.wikipedia.org/wiki/Structure_from_motion`, 2013. [Online; accessed 05-Apr-2013].

[23] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[24] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: exploring photo collections in 3d," in *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, (New York, NY, USA), pp. 835–846, ACM, 2006.

[25] N. Snavely, "Bundler v0.4 user's manual." `http://www.cs.cornell.edu/~snavely/bundler/bundler-v0.4-manual.html`, 2013. [Online; accessed 25-May-2013].

[26] Wikipedia, "Sensor fusion." `http://en.wikipedia.org/wiki/Sensor_fusion#cite_note-1`, 2013. [Online; accessed 02-Feb-2013].

[27] G. Bleser and D. Stricker, "Advanced tracking through efficient image processing and visual-inertial sensor fusion," in *VR*, pp. 137–144, IEEE, 2008.

[28] L. Armesto, J. Tornero, and M. Vincze, "Fast ego-motion estimation with multi-rate fusion of inertial and vision," *Int. J. Rob. Res.*, vol. 26, pp. 577–589, June 2007.

[29] L. Kleeman, "Understanding and applying kalman filtering," in *Proceedings of the Second Workshop on Perceptive Systems*, Jan. 1996.

[30] O. Team, "Ogre," 2013. [Online; accessed 12-Apr-2013].

[31] D. Oyj., "Qt website." http://www.qt.digia.com. [Online; accessed 26-Mar-2013].

# VITA

Cengiz Hüroğlu is doing his MSc thesis on Computer Vision topics at Özyeğin University, Graduate School of Science and Engineering. He has Bachelor of Science (BSc) degree from Yildiz Technical University, Electrical and Electronics Faculty, Computer Engineering Department.

Since 2004, he has nine-years full-time experience of software development and management in different companies. He has been working as a researcher/software engineer in TÜBİTAK BİLGEM BTE since 2009.