Imperial College London

FINAL YEAR PROJECT REPORT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

ElasticFusion on the Google Tango Tablet

Author:
Jiahao Lin (CID: 00837321)
Email: jiahao.lin13@imperial.ac.uk
Supervised by Dr Stefan Leutenegger
Second marker: Professor Andrew Davison

Date: June 13, 2017

1 Abstract

Abstract
The abstract is a very brief summary of the report's contents. It should be about half a page long. Somebody unfamiliar with your project should have a good idea of what it's about having read the abstract alone and will know whether it will be of interest to them. Note that the abstract is a summary of the entire project including its conclusions. A common mistake is to provide only introductory elements in the abstract without saying what has been achieved.

=============

Google Tango[1] is an augmented reality computing platform by Google that give mobile devices the ability to sense 3D space information. Equipped with this platform Google's Tango tablet is an Android device that carry sensors including 4MP(Mega Pixel) color camera, Inertial Measurement Unit (IMU), and Infra-Red depth camera. it also carrys NVIDIA Tegra K1 processor which provides more powerful computational power than other mobile devices. Given this cheap commodity mobile hardware and software, this project aims to build a fully self-contained 3D dense surface reconstruction solution using state-of-the-art RGB-D(csolor and depth) SLAM (Simultaneous Localization And Mapping) algorithms.

In the recently published dense RGB-D SLAM algorithms, ElasticFusion[2] published by Dyson Robotics Lab[1] at Imperial College London is one that achieve state-of-the-art performance by applying local and global loop closures using a deformation graph, so that drifts raised by accumulated errors can be recovered and global consistency of the map is maintained. The fact Elastic Fusion scored high marks on benchmarks for trajectory estimation and surface estimation makes it perfect to be applied on Google Tango Tablet. Also, the current code exist for ElasticFusion running on desktop computer made heavily use of CUDA(Compute Unified Device Architecture) for GPU general data processing, which is supported by the NVIDIA Tegra K1 processer on Google Tango tablet.

This project have attempted to port the code for ElasticFusion onto Google Tango Tablet and integrated with 3D points data from Tango platform, although the result of running ElasticFusion on the tablet has shown to be not effective enough for self-contained real time application due to processing power and sensor problem on the tablet and time and effort limitation in this project, this attempt has still proved that it is possible and more convenient to record the 3D space data on a hand held portable device first, and then let the data to be processed on a more power desktop computer for high quality 3D surface reconstruction.

Also, since the Tango platform on the tablet provides pose estimation feature, this pose is simply feed into the ElasticFusion program run on the tablet, so that the ElasticFusion part doesn't need to waste time and processing power to execute the

[1] `http://www.imperial.ac.uk/dyson-robotics-lab/`

camera tracking part. However, the pose provided by Tango is not accurate enough and during the reconstruction process ElasticFusion will alter the pose if global or local loop closure happens, in this project a simply way to correct pose input to ElasticFusion by Tango is developed to over come this problem.

2 Acknowledgements

Firstly I would like to express my deepest appreciation to all those who provided me the possibility to complete this report. A special gratitude I give to my project supervisor, Dr. Stefan Leutenegger, for his patient overseeing and guidance throughout this project, also his time and effort invested to give useful advices.

Secondly I would also like to thank my friends for accepting nothing less than excellence from me. Last but not the least, I would like to thank my parents for supporting me spiritually throughout the progress of this project and my life in general.

Contents

1 Abstract 2

2 Acknowledgements 4

3 Introduction 7

4 Background 10
- 4.1 SLAM papers . . . 10
  - 4.1.1 Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust Perception Age . . . 10
  - 4.1.2 Kinectfusion . . . 14
  - 4.1.3 Keyframe-Based Visual-Inertial SLAM Using Nonlinear Optimization . . . 15
  - 4.1.4 Elasticfusion . . . 17
- 4.2 Related Work . . . 21
  - 4.2.1 CHISEL . . . 21
  - 4.2.2 InfiniTAM . . . 22
  - 4.2.3 ORB-SLAM on mobile devices . . . 23
  - 4.2.4 LSD SLAM . . . 26
  - 4.2.5 VINS-Mobile . . . 26

5 Project Method Overview 29

6 Project Implementation 32
- 6.1 Tango Platform . . . 32
- 6.2 Logging data from Tango . . . 39
- 6.3 Compiling ElasticFusion on Tango . . . 39
  - 6.3.1 Dependencies . . . 39
  - 6.3.2 OpenCV . . . 39
  - 6.3.3 CUDA . . . 39
  - 6.3.4 Pangolin . . . 39
  - 6.3.5 SuiteSparse . . . 40
  - 6.3.6 OpenGL . . . 40
  - 6.3.7 Eigen . . . 40
- 6.4 Running ElasticFusion on Tango . . . 41
  - 6.4.1 Running ElasticFusion with pose on Tango . . . 41
  - 6.4.2 Running ElasticFusion dataset on Tango . . . 41
  - 6.4.3 Running Tango dataset on Tango . . . 41
- 6.5 Combine ElasticFusion with pose from Tango in Loop closure mode . . . 42

7 Evaluation 44
- 7.1 Reconstruction Accuracy . . . 44
- 7.2 depth sensor noise . . . 44
- 7.3 Computation power problem . . . 44
- 7.4 Optimization problem . . . 44

7.4.1 Software limitation 45

8 Conclusions and Future Work 46

9 Bibliography 47

10 Appendix 48

11 User Guide 49

12 Program Listings 50

13 Word Count 51

3 Introduction

This is one of the most important components of the report. It should begin with a clear statement of what the project is about so that the nature and scope of the project can be understood by a lay reader. It should summarise everything you set out to achieve, provide a clear summary of the project's background, relevance and main contributions. It should explain the motivation for the project (i.e., why the problem is important) and identify the issues to be addressed (i.e., why the problem is difficult). The introduction should set the scene for the project and should provide the reader with a summary of the key things to look out for in the remainder of the report. When detailing the contributions it is helpful to provide pointers to the section(s) of the report that provide the relevant technical details. The introduction itself should be largely non-technical. It is sometimes useful to state the main objectives of the project as part of the introduction. However, avoid the temptation to list low-level objectives one after another in the introduction and then later, in the evaluation section (see below), say something like "All the objectives of the project have been met blah blah...". A project that meets all its objectives is, by definition, weak and unambitious. Concentrate instead on the big issues, e.g. the main questions (scientific or otherwise) that the project sets out to answer.

=============

==========

The main objective of this project is to develop a fully self-contained mobile dense 3D surface reconstruction solution running on Google Tango Tablet using the ElasticFusion algorithm. Ideally the 3D surface reconstructed should be rendered on screen in real time from the tablet's viewpoint when user is holding the tablet to scan the room, and after the scanning stopped the 3D map created should be able saved on device to allow for later use or shown on other devices. The motivation of behind this project is the lacking of high quality 3D surface reconstruction application on mobile device. Applications for 3D reconstruction currently exist on Tango Platform mostly use volumetric representation, and can only produce coarse model of surrounding space, a screen shot of using 3D mesh created using Google Tango's demo mesh builder application is shown in Fig 1. Also the existing applications fail to produce surface map without misalignments during the scanning process, which is often caused by loopy motion of the scanning device in the process. When doing 3D reconstruction, simple fusion of 3D point clouds would produce drifts caused by errors accumulating over time, therefore by applying ElasticFusion algorithm to the solution, even extremely loopy trajectories of the tablet and long duration of scanning process would not result in decrease of accuracy of the map. On the contrary, revisit areas that has previously been mapped will only help to maintain the global consistency of the map. The part of algorithm that made this possible in ElasticFusion is the local and global loop closure checking after camera tracking process, which will be discussed in detail later.

Previously, ElasticFusion can only be run on a desktop computer with powerful

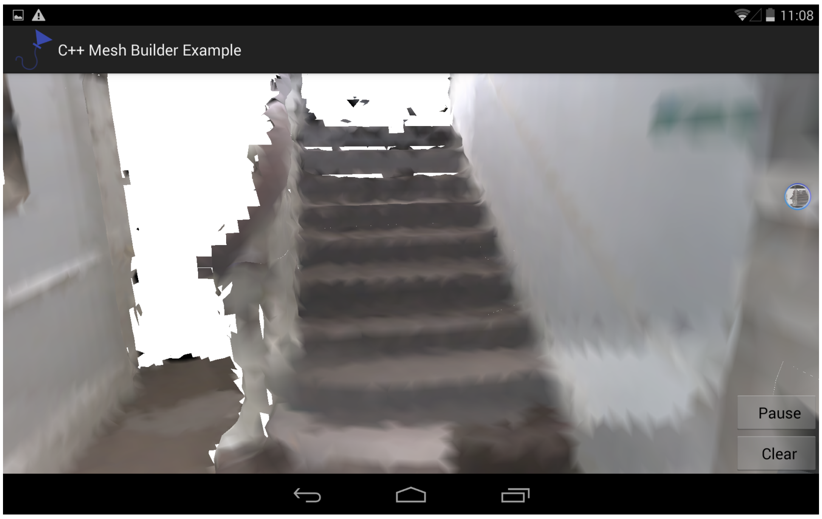


Figure 1: ScreenshotTango5

graphic card, using a raw dataset that is given, or connect the computer to a RGB-D camera such as Microsoft Kinect or ASUS Xtion Pro Live or Intel RealSense 3D Camera. That makes it non convenient for a user to scan the space he or she is currently in for 3D reconstruction, therefore the Google Tango tablet with both RGB and depth camera would be the perfect mobile device to implement a portable version of ElasticFusion on. The Google Tango tablet carries Android 4.2.2 OS, has 4GB of RAM, a quadcore Nvidia Tegra K1 graphics card supporting CUDA, six-axis gyroscope and accelerometer, a wide-angle 120 field of view tracking camera which refreshes at 60Hz, a projective depth sensor which refreshes at 3Hz, and a 4 megapixel color sensor which refreshes at 30Hz. The processing power on this tablet succeeds most mobile devices on the market right now, which makes running ElasticFusion on the device alone become possible. Since ElasticFusion requires both high end processor and graphic card on a desktop computer, from the read me of ElasticFusion on Github: "A very fast nVidia GPU (3.5 TFLOPS+), and a fast CPU (something like an i7)." is recommended for a desktop computer, therefore one of important problem in this project is that Tango tablet's processing power is still not strong enough to run ElasticFusion for real time application. However, the attempt make in this project have still shown the possibility of running a state of the art SLAM algorithm on mobile device.

In the original design for this project, if the development of this solution is successful, an augmented reality application could be created as demo to illustrate the interaction of virtual object with scanned 3D surface map. But after various experiments in this project, a conclusion that ElasticFusion could not be run on Tango tablet as a real time application drawn, due to processing power and depth sensor accuracy problem, although due to time and effort limitations, the ElasticFusion on Tango application produces in this project still produce incorrect reconstruction model and the program has not been optimized. Despite all the above, the contribution of this project still includes the attempt made to implement this solution, and the possibility of using Tango tablet as a pure scanning device for recording data, and let the data to be processed on other computers for doing 3D surface reconstruction.

In the solution, an Android application is developed using Android Native Development Kit (NDK), which allows the use of Tango C API to acquire depth camera data and 3D pose data, also makes it possible to reuse the existing code for ElasticFusion that is written in C++. Therefore most of the application is written in C++, with a few Java code for the Android application. The application uses the code for Advanced Estimation in Robotics (433H) course practical exercise by Dr Stefan Leutenegger in Department of Computing at Imperial College London as the basic frame work, firstly the code is combined with the logger for ElasticFusion[2] to write the RGB, depth and pose data in format required by ElasticFusion acquired from Tango into two files on the Tango tablet, which allows ElasticFusion to be run on desktop using these two files as raw dataset. Then the core function part of code for ElasticFusion is ported and adapted to Android with its dependencies changed or removed, so that ElasticFusion can be run on Android while its source remains in C++. One thing worth notice in here is that this project uses an internal version of ElasticFusion source code at Dyson Robotics Lab in Imperial College London, instead of the open source version on Github[3]. The final produced application is able to log the data from sensors on Tango into files, or run ElasticFusion from sensor data in real time, or run ElasticFusion from raw dataset files on the device with or without pose data. This provides various way to do experiments or measure efficiency during the project. However, due to time and effort limitations, no interface for showing the progress of ElasticFusion running is built, this is different from original planning but the reasons will be explain in detail later. Right now on the screen of the application, interface only shows the image that color camera is taking and a few buttons, screen shot and description will be provided in later section.

Finally, simple change is made on both desktop version and Android version of ElasticFusion to allow for pose feed into the program to be corrected overtime when global or local loop closure happens. This help maintain consistency between the pose feed into ElasticFusion and the internal pose stored in the program. Since the original ElasticFusion can only estimate pose from RGB-D data, or use an input pose as ground truth pose, or use the input pose as a bootstrap pose for optimization, the adaptation made in this project enables the pose feed into ElasticFusion to be consistent with the true pose while also not waste time to do the camera tracking process when the processing power is not so strong like on the Tango tablet.

[2] `https://github.com/mp3guy/Logger1`
[3] `https://github.com/mp3guy/ElasticFusion`

4 Background

The background section of the report should set the project into context by relating it to existing published work which you read at the start of the project when your approach and methods were being considered. There are usually many ways of solving a given problem, and you shouldn't just pick one at random. Describe and evaluate as many alternative approaches as possible. The published work may be in the form of research papers, articles, text books, technical manuals, or even existing software or hardware of which you have had hands-on experience. Your must acknowledge the sources of your inspiration. You are expected to have seen and thought about other people's ideas; your contribution will be putting them into practice in some other context. However, avoid plagiarism: if you take another person's work as your own and do not cite your sources of information/inspiration you are being dishonest; in other words you are cheating. When referring to other pieces of work, cite the sources where they are referred to or used, rather than just listing them at the end. Make sure you read and digest the Department's plagiarism document .
In writing the Background chapter you must demonstrate your capability of analysis, synthesis and critical judgement. Analysis is shown by explaining how the proposed solution operates in your own words as well as its benefits and consequences. Synthesis is shown through the organisation of your Related Work section and through identifying and generalising common aspects across different solutions. Critical judgement is shown by discussing the limitations of the solutions proposed both in terms of their disadvantages and limits of applicability.
Typically you can look for Background work using different search engines including:
* Google Scholar * IEEExplore * ACM Digital Library * Citeseer * Science Direct
Note 1: Often the terms Background, Related Work or State of the Art are used interchangeably.
Note 2: Keyword search is wonderful, but you need the right Keywords.
Note 2: IEEExplore, ACM Digital Library and Science Direct require you to be on the College network to download the PDF of papers. If at home, use VPN.

=============

This section includes two parts, the first part includes some paper related to SLAM and SLAM algorithm including ElasticFusion, the second part includes some SLAM applications developed to be run on mobile devices.

4.1 SLAM papers

4.1.1 Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust Perception Age

This survey paper: past present and future of SLAM [3] gave a detail description about the current development state of modern SLAM, including standard formulation and models, different methods used in SLAM. Below we give a brief summary of basic knowledge about SLAM from this paper.

Simultaneous Localization and Mapping (SLAM) means constructing the model of environment (map) that the robot is in, and estimating the robot state within the environment at the same time. Usually the robot is equipped with some kind of sensor: RGB camera, depth sensor, IMU, or GPS, So that the robot is able to perceive the environment in some way from these sensor. In standard models, the robot state includes its position and orientation, velocity, sensor biases and calibration parameter. Using the robot status and data read from sensors, the environment(map) constructed could be representation in different forms. With the existence of the map, errors raised in estimating robot state could be eliminated and also doing "loop closure" when robot revisits a place in the map allows drift to be corrected and hence improve accuracy of the estimated status of robot.

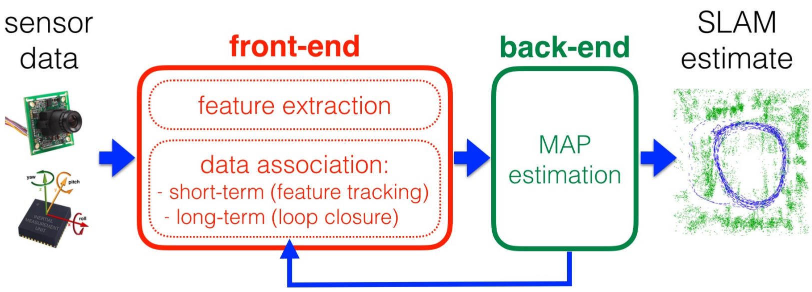


Figure 2: Front-end and back-end in a typical SLAM system[3]

The structure of SLAM system is mainly divided into two parts: front-end and back-end The front-end extract features from sensor data and associate them to the model used to make predictions, then the back-end does estimation of robot state and the map using this model produced by front-end. As you can see in the figure 2, this process forms a continuous process of tracking and update.

Classical formulations of SlAM is to take probabilistic approaches such as models based on Maximum Likelihood Estimation, Maximum-a-posteriori (MAP) estimation, EKF(Extended Kalman Filters), and particle filters.
Many of the popular SLAM algorithm models it as a maximum-a-posteriori estimation problem by taking probability approach. And usually using a factor graph to show relationships and constraints between variables. In these models, often an observation or measurement model is used to represent the probability of observe measurements given the state of robot, given some random measurement noise that usually model to be under a zero-mean Gaussian distribution. By using Bayes theorem, the MAP estimate, in which the posterior is the probability of robot state given a measurement, can be calculated given some prior belief of the robot state. In the case that there is no prior information known about robot state, the MAP estimate would simply become a maximum-likelihood-estimate. When we are given a set of independent robot states and measurements, these variable would become fac-

tors(nodes) in the factor graph, and their probabilistic relationships would become the constraints(edges) in the factor graph.

Factor graph allows us to visualize our problem in a simple way and provides an insight into how the variables are related together by their constraints. 3 from the paper have shown an example of it for a simple SLAM problem.

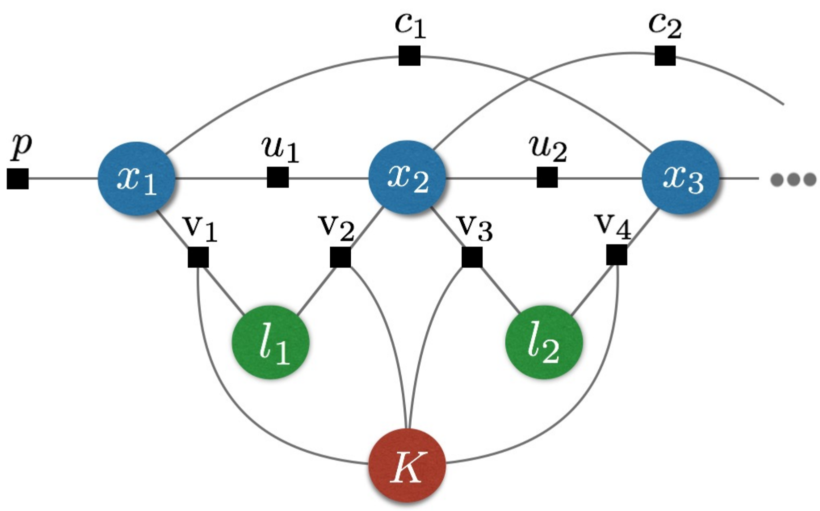


Figure 3: SLAM as a factor graph[3]

As explained in [3]: "Blue circles denote robot poses at consecutive time steps ($x1$, $x2$ ...), green circles denote landmark positions ($l1$, $l2$ ...), red circle denotes the variable associated with the intrinsic calibration parameters (K). Factors are shows are black dots: the label "u"marks factors corresponding to odometry constraints, "v "marks factors corresponding to camera observations, "c"denotes loop closures, and "p"denotes prior factors."

When calculating the MAP estimate of robot state, we can transform the problem into minimizing the negative log of posterior, assuming the noise is zero-mean Gaussian distributed, this becomes a non-linear least squares problem, since we will be minimising the measurement error's $l2$ norm. This method is similar to the BA (Bundle Adjustment) method in computer vision area, which minimise the reprojection error. However factor graph can not only contain visual and geometry factor, but also a variety of data from different sensors. Also the MAP estimate is calculated every time a measurement is arrived, instead of performing calculation after all the data is given.

Different from factor graph optimization, a similar approach is called pose graph optimization, which estimate trajectory of robot using measurements between relative poses as constraints, and optimized these poses to achieve better accuracy trajectory.

Other than optimization-based estimation, some filtering approach using EKF model have also achieve great results, such as MSCKF(Multi-State Constraint Kalman Filter of Mourikis and Roumeliotis)[4], EKF is used for performing filtering on the linearised point of estimate, and have shown to be accurate in visual-inertial navigation.

In terms of the representation of the map, one way is to have a group of features or landmarks in 3D, these features are usually extracted from sensor images to be allow easy distinguishable from environment, such as edges and corners. The method used to extracting features from images are well-established within computer vision area, using descriptors like ROB, SIFT, and SURF. The discriminative and repetitive properties of landmarks allow the correspondence of each landmark and sensor measurement(image), and then via triangulating, the robot is able to compute the relative pose between measurements and geometric information about landmarks, therefore achieve localization and mapping.

Other than feature based approach, sometimes raw dense representation of the environment is used. different than extracting landmarks, this approach stores a dense 3D map using high resolution points, the 3D geometry of environment is described by these vast amount of points, so called point clouds. With stereo or depth cameras, lase r depth sensors, 3D information of the points can be easily obtained, therefore this method is popular in RGB-D SLAM. Sometimes, these points doesn't only stores simple 3D information, in Elasticfusion[2], each 3D point is represented by a surfel, which is a disk with certain radius centred by the 3D point and stored information such as normal vector of that point on the surface and color. These suefel disk combine together can encode the geometry of environment better than simple points.
At more a higher level than raw, unstructured points, dense related element that partitioned by space is also used to represent surface and boundaries, such as surface mesh models(connected polygons formed by points) and implicit surface representations like octree and occupancy grid. KinectFusion[5] storing surfaces using volumetric representation by uniformly subdivide a 3D physical space into a 3D grid of fixed size voxels, voxels defining surface has a zeros value of the truncated signed-distance function (TSDF) function that stores distance information relative to the surface. We can see that represent the environment in these types of dense data requires huge amount of storage, however they give very low level geometry information which is suitable for 3D reconstruction and rendering.

Comparing feature-based and dense SLAM methods, for the purpose of this project, dense SLAM algorithm Elasticfusion[2] is used as 3D reconstruction is the main objective instead of state estimation, since Elasticfusion maintains a dense 3D surfel map that make use of all the information about the environment obtained from sensors it is more robust in localization and mapping that feature-based approaches.

4.1.2 Kinectfusion

KinectFusion[5] is a dense RGB-D SLAM method that enables high quality 3D reconstruction and interaction in real time. This method make use of low cost mobile RGB-D camera, scanning the environment in real time and acquire 3D point information about the space. During the reconstruction process, holes caused by absence of depth measurement in original 3D point cloud is filled in, and the model is refined by repetitive scanning over time.

Firstly the 3D points reading are then stored in a 3D vertex map and their normal vectors are calculated by reprojection of neighbouring vertexes. After that the camera pose is tracked by using Iterative Closest Point (ICP)[6][7] algorithm to align current 3D points with the previous frame. In ICP algorithm, first the corresponding point of each current 3D point is found by doing projective data association, which reprojects each previous point into image coordinates and use this to look up current vertexes that may be correspondence points. In the second part these correspondences are tested by calculated distance and angle between them and reject any outliers that are beyond certain threshold. Please be noted that the running of ICP in real time is because the use of GPU to parallel processing a current vertex per thread, according to the original paper.

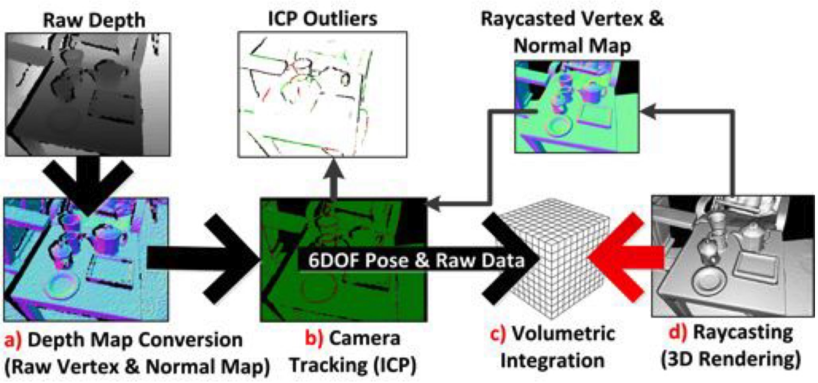


Figure 4: Overview of tracking and reconstruction pipeline from raw depth map to rendered view of 3D scenem[5]

The structure of KinecFusion is given in Fig 4 from original paper. The result of ICP is a transformation matrix from previous camera pose to the current pose, which is later used for converting the coordinates of current vertexes into the global coordinate frame. These coordinate data is then fused into the current model that uses a volumetric representation, as mentioned before. The 3D space is uniform divided into fixed size voxel grid. The geometry of surface is implicitly stores in these voxels by having zero values of Truncated Signed Distance Functions (TSDFs) at the surface, positive value in the front of surface, and negative at the back of it. Also, to improve efficiency and minimise storage usage only a truncated area around the surface is actually stored, thus the name ”truncated”.

When integrating the current vertexes into voxel grid, each voxel is perspective projected back onto the image to calculate the difference between it and the measurement depth of this coordinate. The distance difference is then normalized to maximum TSDF distance and updated using weighted average with previous value. Similar to the implementation of tracking stage, the integration and update process also make use of GPU parallel processing, each thread is assigned a slice of of grid along the $Z-axis$, and sweep through each voxel by its (x, y) position for processing, as shown in Fig 5.

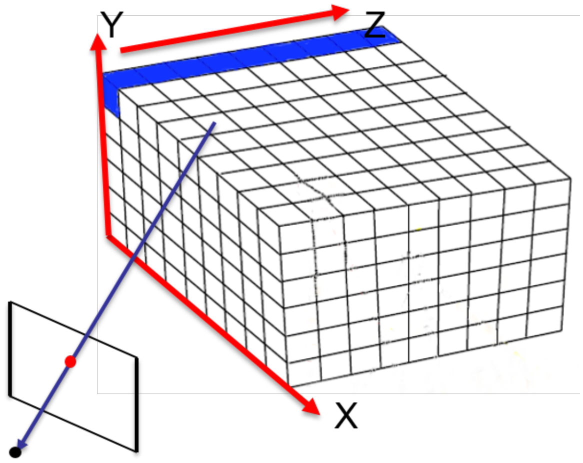


Figure 5: Volumetric Integration using GPU thread [8]

Finally, in order to render the 3D reconstructed surface raycasting is performed. This is done by having each GPU thread to follow a ray from the camera center along one pixel in the image into the voxel grid in camera coordinate space, the ray is extended until a zero-crossing voxel is met, this voxel represents the surface point observed by this pixel and the position is extracted by doing a trilinear interpolation on this voxel point. And the normal at this position is calculated by find the surface gradent of TSDF values around the position. With these information for each pixel, a live 3D reconstruction view from camera view point can be rendered using camera's pose and intrinsic information.

4.1.3 Keyframe-Based Visual-Inertial SLAM Using Nonlinear Optimization

Other than using depth sensor with RGB camera, another kind of sensor equipped on the Google Tango tablet is Inertial Measurement Unit (IMU). Okvis[9] is a sparse feature-based SLAM approach that tightly coupled IMU measurement with stereo visual measurement using nonlinear optimization.

Usually in optimization-based visual SLAM, the structure of geometry is to relate camera poses using landmarks, and optimization is performed to minimize the error of landmark reprojection in different observing frames. However, in visual inertial SLAM, the IMU measurement is used to add additional constraint between camera poses and speed and estimation biases from gyroscopes and accelerometers. The structure graph from original paper is shown in Fig 4 with detail labelling of different variables and constraints.

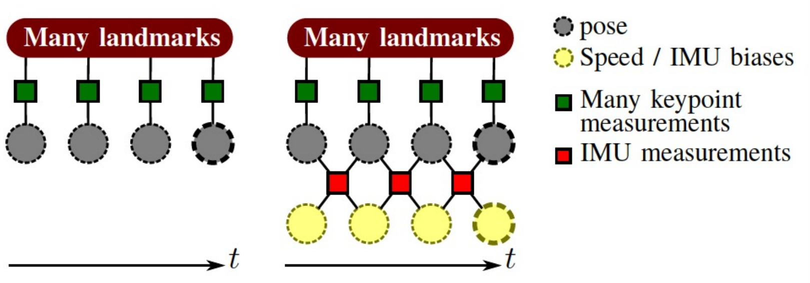


Figure 6: graph of variables and their relationships in visual SLAM on the left and visual intertial SLAM on the right [9]

In order to tightly couple the visual and inertial measurements, a joint nonlinear cost function that contains both visual reprojection error term and IMU error term is created. These both errors are eliminated by optimizing this one error function.

To reduce complexity and ensuring real time operation, a fixed size temporal window of robots poses and relative states is maintained, and old robot poses will be marginalized out as new frames are inserted.
In the visual frontend, a sparse map of frames and landmarks is stored. For each new frame, keypoints are extracted using SSE(Streaming SIMD Extensions)-optimized Harris corner detector and BRISK(Binary Robust Invariant Scalable Keypoints) descriptor, their 3D positions are then triangulated by stereo and put into the sparse map. During the process landmarks and keypoints are brute-forcely matched and outliers are rejected by using predicted pose from IMU data. In the optimization window, two kind of frames is maintained, first is a temporal window of the S most recent frames, second is N keyframes in the past. A frame is determined as keyframe if the number of matched points with previous frame is less than 50 to 60% of the number of detected keypoints.

In the marginalization process, initially the first $N+1$ frames forms the marginalization window, as shown in Fig 7. When a new frame is inserted into the marginalization window, there are two ways to marginalize old state. If the oldest frame in temporal window is not a keyframe, then the state of that frame with its measurements, speed and biases will be dropped, as shown in Fig 8. On the other hand if it is a keyframe, landmarks that are visible in first temporal frame but in most recent keyframe is also dropped, as shown in Fig 9.

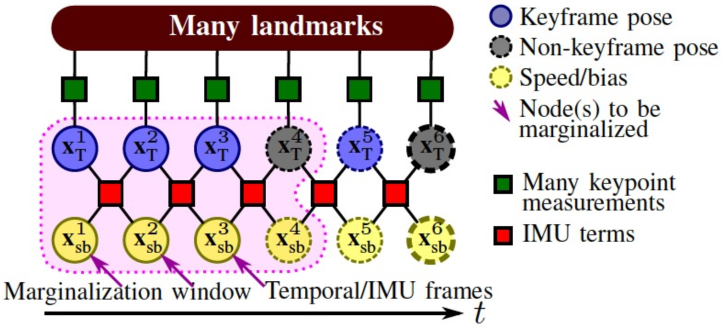


Figure 7: Graph of initial marginalization window [9]

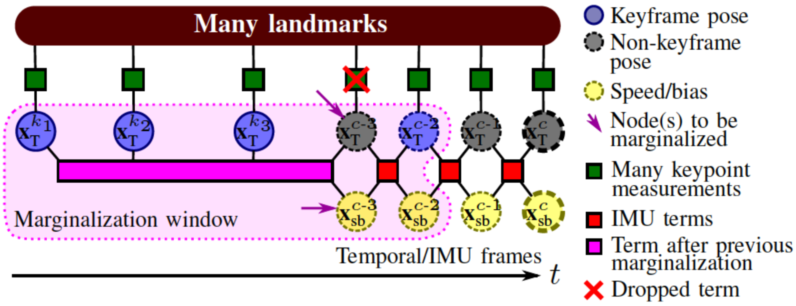


Figure 8: Graph for marginalization in the first case [9]

4.1.4 Elasticfusion

Elasticfusion[2] is a RGB-D dense SLAM method for 3D reconstruction in room scale without using pose graph it also use surfels to represent the map instead of voxel grid such as in KinectFusion[5]. In the scanning process, it iteratively check for local and global loop closures and refine the map by using a deformation graph to apply non rigid deformation to the surface. However, while this approach has achieved great results in room scale, it may be difficult to be used on bigger or outdoor scenes due to complex light condition and size of memory needed to store the surfels.

In the surfel map, each surfel stores the information about a local surface area around the center of the surfel, the surfels are lay out in a wat that holes on the surface are minimised. The information about a surfel include its position, normal vector, color, weight and radius. The radius is larger as distance between image center and surfel is greater. The surfels are extracted from point cloud using the similar method as in [10], illustrated in Fig 10, but differently, in here the surfels are divided into active and inactive parts. Only active surfels are used for camera pose estimation and depth map fusion, while the inactive parts are the surfels that have not been observed for a given time period and can be reactivated in a local loop closure. In ElasticFusion, OpenGL shading language is used to manage the surfel map and update, fuse and render the view.

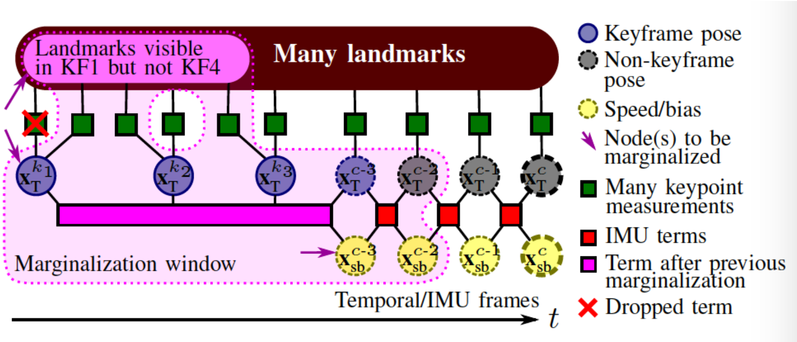


Figure 9: Graph for marginalization in the second case [9]

The detail structure of ElasticFusion is shown Fig 11 in the original paper. From

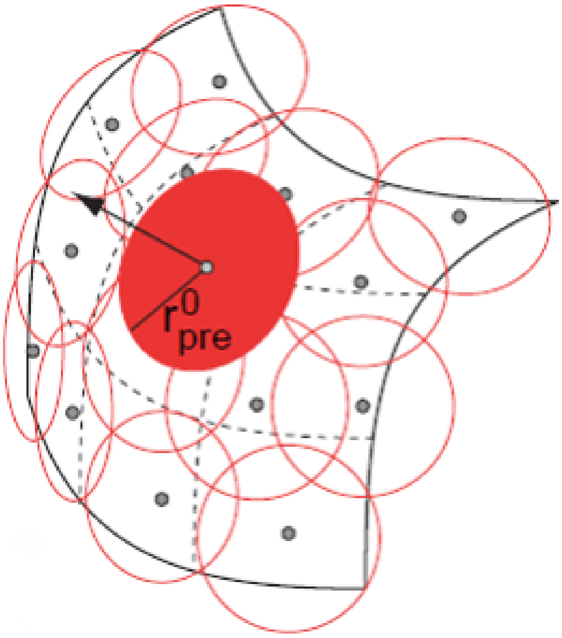


Figure 10: surfel2

the graph can be seen that firstly in the camera pose estimation step, dense frame-to-model camera tracking is used, geometric and photometric pose estimations are combined into a joint cost function to for minimization. Geometric error is calculated using frame to model projective data association and ICP algorithm as in kinecFusion[5], and photometric error is gain by calculating the intensity difference between current RGB image and backprojected image from the model. Note that here in the tracking process only active surfels are used.

After camera pose tracking for each frame, an active predicted model is produced, this model is then rendered into an image for global loop cloure checking, to try recover from drifts produced by accumulating error, and realign the camera pose. To enable global loop closure checking, key frame information from past is stored in a randomized fern encoding database using same technique as in [11], it allows stor-

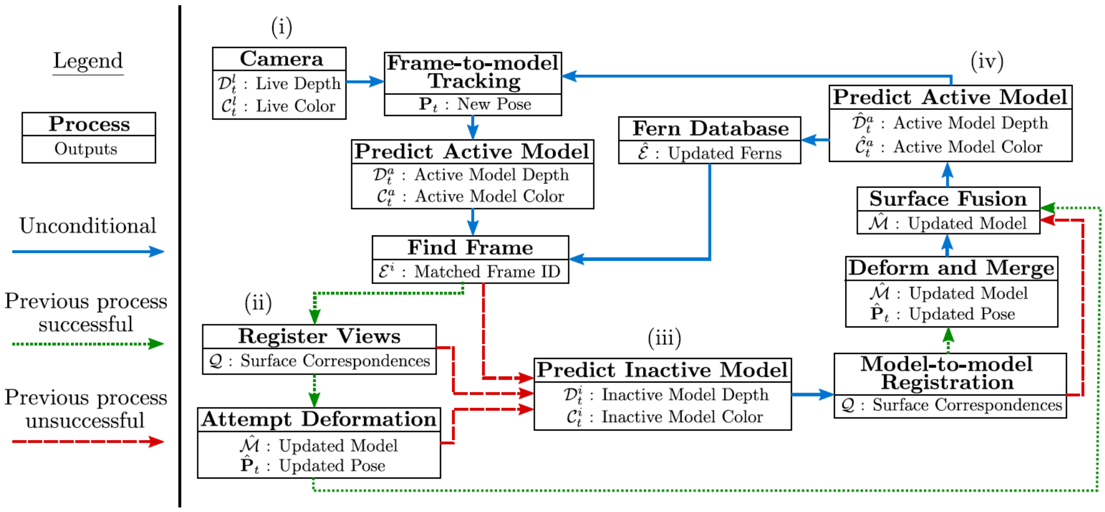


Figure 11: elas1

ing and matching frames without putting redundant information into the database. If a high quality match between the current predicted view of active model and a frame in fern database is found, points are sampled from matched frame and active view to construct surface constraints that are used to register the surface together and optimize the deformation graph. Each surface constraint contains a pair of points and timestamps to represent the position of point and the destination position that it should reach the destination after deformation.

If no global loop closure is found then local model to model loop closure is checked by try to register the currently predicted inactive model with active model under the current frame, similar method is used when high quality alignment is found, constraints are extracted and fed into the deformation graph, then the inactive surfels in this alignment is reactivated to allow for fusion into the map.

The realignment of map is done by applying non rigid deformation using a deformation graph. Deformation graph is constructed each frame for efficiency, with nodes sampled uniformly from the surfels using similar method as in DynamicFusion[12], this is illustrated in Fig 12 to shown the relation between surfel and deformation graph noded. Each node have a rotation matrix and position vector that represents optimization parameter and neighbourhood of each node forms the edge of the graph. Each surfel has a set of influencing nodes that affects the deformation of that surfel.

The optimization of deformation of the model is done by minimising five cost function in total using the surface constraints constructed in the steps described above. The parameters in surface constraints after optimization reflects the surface registration happened in loop closure. These five cost functions include the following functions: maximises the rigidity of deformation, guarantees the smoothness of deformation in the graph between neighbour nodes, minimises the error in position

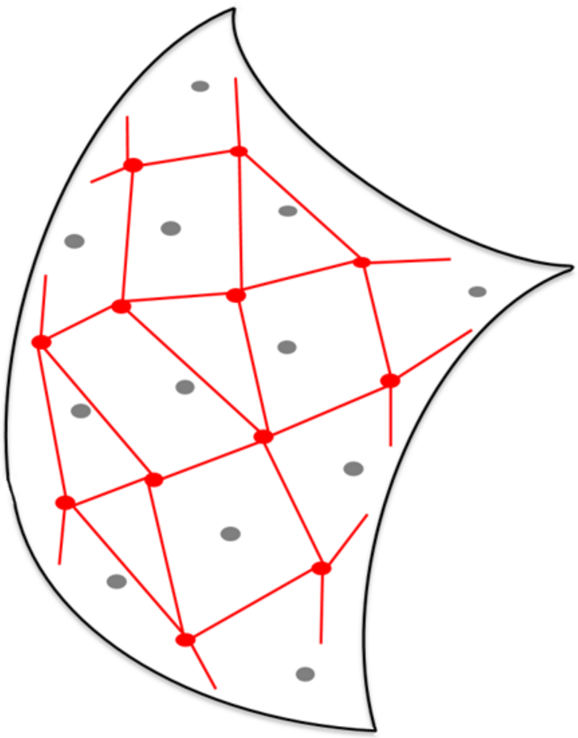


Figure 12: surfel1

constraints, that is reduce the difference between source and destination position, also ”pins” the inactive surface of the model without moving to make sure is the active surface that deforms into the inactive surface. In the global loop closure optimization process an extra cost function is added, it introduce the set of relative constraints to prevent the previous surface registrations from being pulled apart by subsequent global loop closures. The optimization problem is solved using iterative Gauss-Newton algorithm on CPU and the result gives a relative transformation matrix that bring the surfaces registered into alignment. The graph Fig 13 from the original paper have shown the process of a scanning with loop closure happens when the scanning camera revisits area that has been scanned before, with active model coloured by surface normal overlaid on the inactive model in greyscale.

After the optimization, deformation graph is applied to the surface by using a weighted distance method to calculate the transformation of each surfel by its influencing nodes. Model is then deformed using this transformation to reflect the registration of surfaces in loop closure. Finally, the current camera data is fused into the latest updated model to update the surfel map under current viewpoint.

Before new frame of data come in, the current active model is rendered to an image using the latest predicted pose to prepare for tracking, and also being decided whether this frame will added to the fern database.

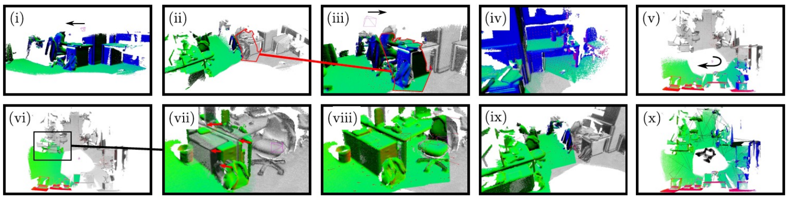


Figure 13: loopclosure

4.2 Related Work

This section gives some SLAM systems that have been implemented on mobile devices, such as Android phone or tablet, and Apple iPhone or iPad. However, most of these systems mainly focus on the state estimation part of SLAM rather then the mapping part, which means they are not able to be used for 3D surface reconstruction purpose.

4.2.1 CHISEL

CHISEL[13] is a large scale dense 3D reconstruction solution implemented on Google Tango Devices. The graph from the original paper that shows CHISEL application running on the Googlt Tango phone can be seen at Fig 14. It uses dynamic spatially-hashed truncated signed distance field[14] to represent the map and useds the visual-inertial odometry provided by Tango as frontend for localization. To save memory and computation power, space that does not contain surfaces is culled out. By using space carving, even though Tango's depth sensor provides data with high noise, reconstruction with an accuracy of 2-3 cm is still achieved in large scenes, the Fig **??** show a picture of global map reconstructed by CHISEL form the original paper, from this picture can be seen that CHISEL is able to scan and reconstruct the area of a flat with relatively high quality detail. To generalise the solution onto other mobile devices that does not have a powerful GPU,in CHISEL no parallel processing is used to utilise the powerful general purpose GPU provided on Google Tango tablet.

In the pose estimation stage, CHISEL uses the pose estimated provided by Tango platform as a black box, by try to register each depth scan with the model, poses are optimised in the same way as using ICP algorithm, this allows small drifts between frames to be recovered, however during long process of scanning large scene, drifts will still build up overtime due to the lack of loop closure process.

Since the map is stored using spatially-hashed data structure, insert and look up is very fast. The map is divided into chunks, and each chunk is a fixed grid of voxels. These chunks are then spatially-hashed into a spatial 3D hash map. When looking for chunks that need to be updated or drawn, a camera frustum is created for culling chunks that do not intersect with the frustum or not have a depth data, the rest of chunks left is then processed for update or drawn.

For fusing depth scan data into the model, projection mapping is used to compare the depth value on image with the projected visual hull of a voxel, the result is then used to update the TSDF value and weight of that voxel.

In the rendering stage, incremental Marching Cubes is used to generate triangle meshes for the chunk when it is been updated by a depth scan, triangles are generated at the zero isosurface of the TSDF volxel grid.
CHISEL has shown a nice memory-efficient approach to scanning large scene for high quality 3D reconstruction, however it fails to maintain global consistency of the global map, therefore drifts will increase over time due to lacking detection of global loop closure. Despite that, CHISEL provides a great example for utilising the Tango platform as a dense SLAM frontend for 3D reconstruction purpose.

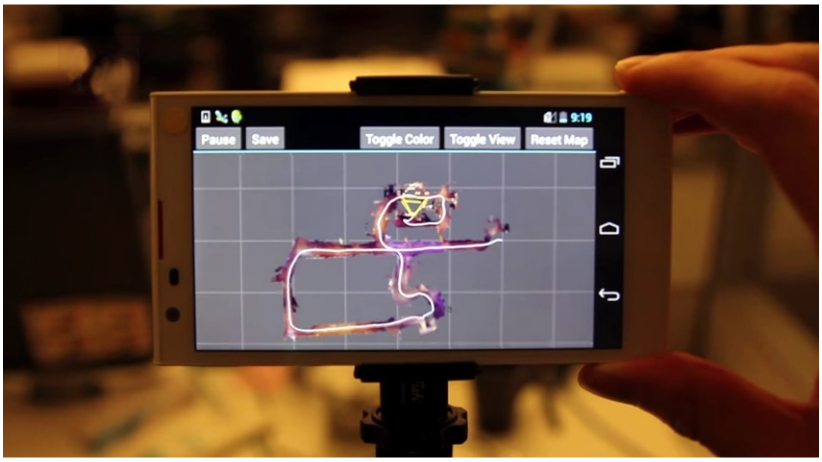


Figure 14: CHISEL1

4.2.2 InfiniTAM

InfiniTAM[15] is an open source cross-platform real time large scale RGB-D SLAM framework publish by University of Oxford[4]. It provides both dense fixed size 3D volume like the one used in KinectFusion[5] and sparse volume using voxel block hashing method[14]. The design of the framework also enables other representations, such as octrees, to be added easily.

InfiniTAM is optimised to run at high frame rate in multiple platform, with NVIDIA Titan X graphics card on desktop computer, it can run at over 1000 fps(frame per second), on iOS platform it utilise Apple Metal Graphics API to reach over 25 fps of frame rate. Also on Android devices with NVIDIA K1 processor such as NVIDIA Shield Tablet and Google Tango Tablet, by using NVIDIA CUDA to for parallel data processing on GPU the frame rate is able to reach over 40 fps.

The architecture designed for InfiniTAM makes it easy to extend functionalities and add new features. Also the C++ source code for InfiniTAM is provided on Github

[4] `https://github.com/victorprad/InfiniTAM`

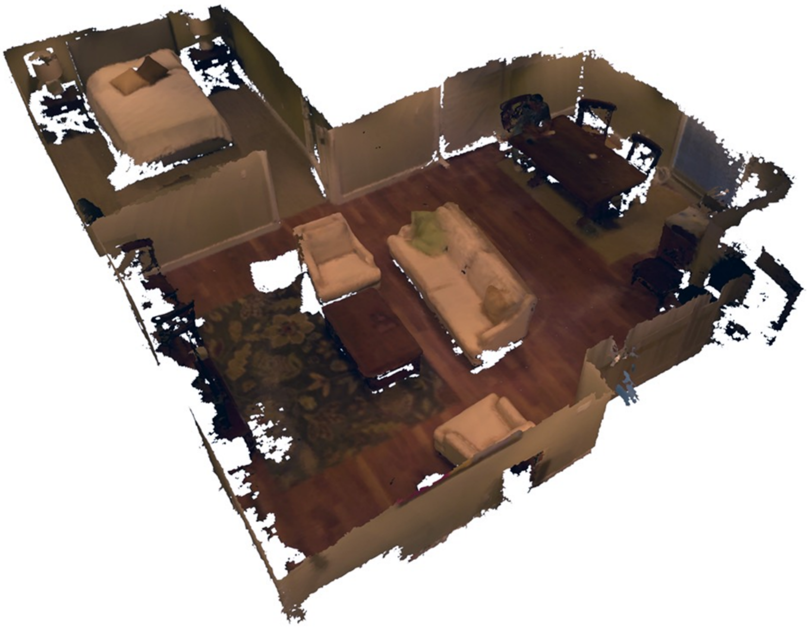


Figure 15: CHISEL2

for both pure CPU and GPU implementations. The Android version of InfiniTAM is similar to ElasticFusion since it also use both OpenGL and CUDA in the program, it has provided an example of dense 3D reconstruction solution on mobile device using volumetric space representation and CUDA for GPU acceleration. The sample screen shot of using InfiniTAM from the official website is given below at Fig 16.

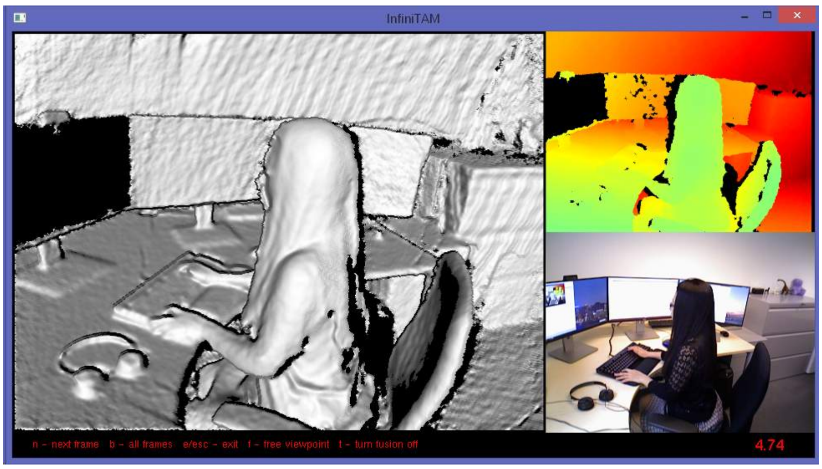


Figure 16: InfiniTAM

4.2.3 ORB-SLAM on mobile devices

ORB-SLAM is a monocular real time SLAM system based on detecting feature points of input image, with loop closure detection and camera relocalization function. It's

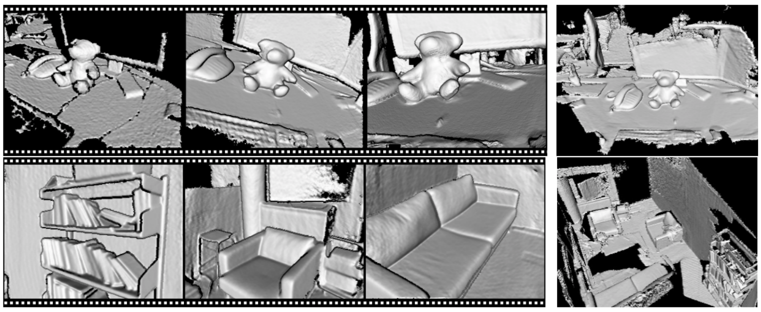


Figure 17: InfiniTAM2

capable of being used in both indoor and outdoor scenes at various scales, and its localization stays robust even with rapid motion of camera.

ORB-SLAM2[16][5] is an open source system builds on previous work of ORB-SLAM, supports not only monocular but also stereo and RGB-D cameras used for computing camera trajectory and sparse 3D reconstruction with true scale.

ORB-SLAM system is divided into three thread performing different functions, the tracking thread is responsible for extracting key points from the input image, as shown in picture Fig 18 from the original paper, this is done by using ORB descriptors, which later are used for pose estimation from the last frame and then decide whether this is a new key frame. the LocalMapping thread inserts the key frame and create map points to perform a local BA(Bundle Adjustment) for optimization, and it maintains a covisibility graph for keypoint and pose tracking. Lastly the LoopClosing thread detects loop closure using Bag-of-words method[17], and perform loop correction by optimising a essential graph if loop closure is detected. With loop closure detection, as shown in the picture Fig 19 from original paper, ORB-SLAM is able to compute camera trajectory consist with the world and reconstruct a sparse map of the world in accurate alignment even in large outdoor scenes.

Since ORB-SLAM2 is a robust SLAM system with robust tracking and loop closure function, there has been many work to implement it on mobile devices using its open source code. Here a few of these work that is open sourced on Github is listed with its dependencies used.

- `ORB_SLAM2_Android`[6] ported ORGB-SLAM2 on Android, using DBoW2[7] for loop detection, clapack[8] and eigen3[9] for math calculation, and g2o[10] for graph optimization and OpenCV for Android. Pangolin[11] was in the original open source

[5] `https://github.com/raulmur/ORB_SLAM2`
[6] `https://github.com/FangGet/ORB_SLAM2_Android`
[7] `https://github.com/dorian3d/DBoW2`
[8] `http://www.netlib.org/clapack/`
[9] `http://eigen.tuxfamily.org/index.php?title=3.0`
[10] `https://github.com/RainerKuemmerle/g2o`
[11] `https://github.com/stevenlovegrove/Pangolin`

code for visualization and user interface and is removed in this implementation.

- `ORB-SLAM-Android`[12] is also an implementation of ORB-SLAM on Android using DBoW2 and g2o.

- `ORB_SLAM_iOS`[13] is implemented on iOS platform with dependencies ported in the same way.

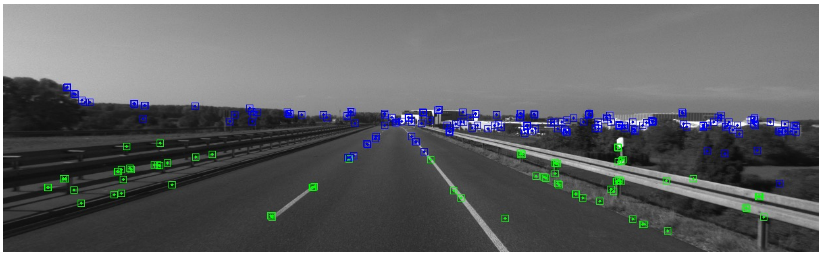


Figure 18: orb1

Figure 19: orb2

[12] `https://github.com/castoryan/ORB-SLAM-Android`
[13] `https://github.com/egoist-sx/ORB_SLAM_iOS`

4.2.4 LSD SLAM

LSD-SLAM(Large-Scale Direct Monocular SLAM)[18][14] is a real time direct monocular SLAM system. It uses pixel intensities of input image to track the pose directly, the alignment two images is done by minimising photometric error using Gauss-Newton algorithm. And depth map is estimated using extended Kalman filter (EKF), a $Sim(3)$ pose-graph of keyframes is maintained to allow for loop closure, using g2o library to optimize the pose graph, this corrects drift occur during the process. As the result, a semi-dense 3D map of point clouds is reconstructed in high accuracy. The structure of LSD-SLAM is shown in Fig 20 using the graph from original paper.

LSD-SLAM not only runs a desktop CPU in real time, it can also run on a modern smartphone for AR application without any GPU acceleration. In this paper [19], an AR application is built on top of direct semi-dense visual odometry using LSD SLAM. In this experiment the application runs on Sony Xperia Z1 smartphone with Android platform, real time performance of at least 30 fps frame rate is reached makes it capable of used as interactive application, this is due to the separation of tracking and mapping part in the application, and also using NEON optimization for computational heavy parts. The structure of application is shown in Fig 22, also Fig 21 shows the virtual object added in AR application, depth map estimated from the current image and collision mesh created for virtual object to interact with. Both pictures are from the original paper.

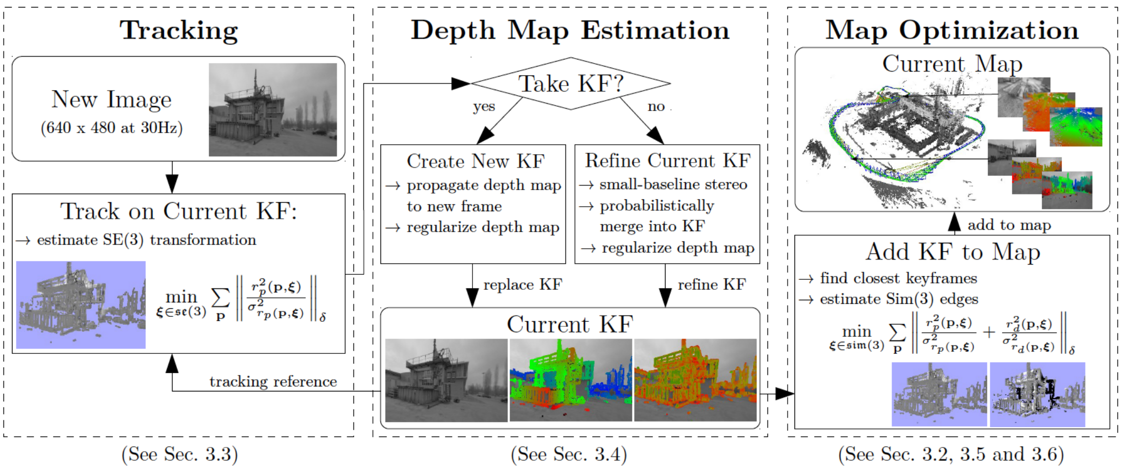


Figure 20: lsdslam1

4.2.5 VINS-Mobile

VINS-Mobile[20] is a open source[15] real time Monocular Visual-Inertial State Estimator on mobile devices developed by members in HKUST Aerial Robotics Group. It can run on iOS devices for providing localization function in AR (augmented reality)

[14] `https://github.com/tum-vision/lsd_slam`
[15] `https://github.com/HKUST-Aerial-Robotics/VINS-Mobile`

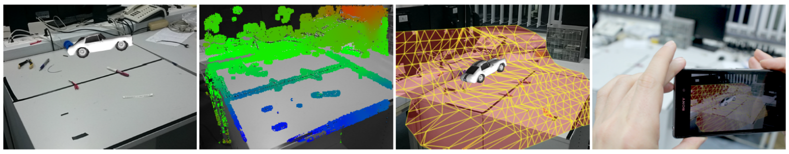


Figure 21: lsdar1

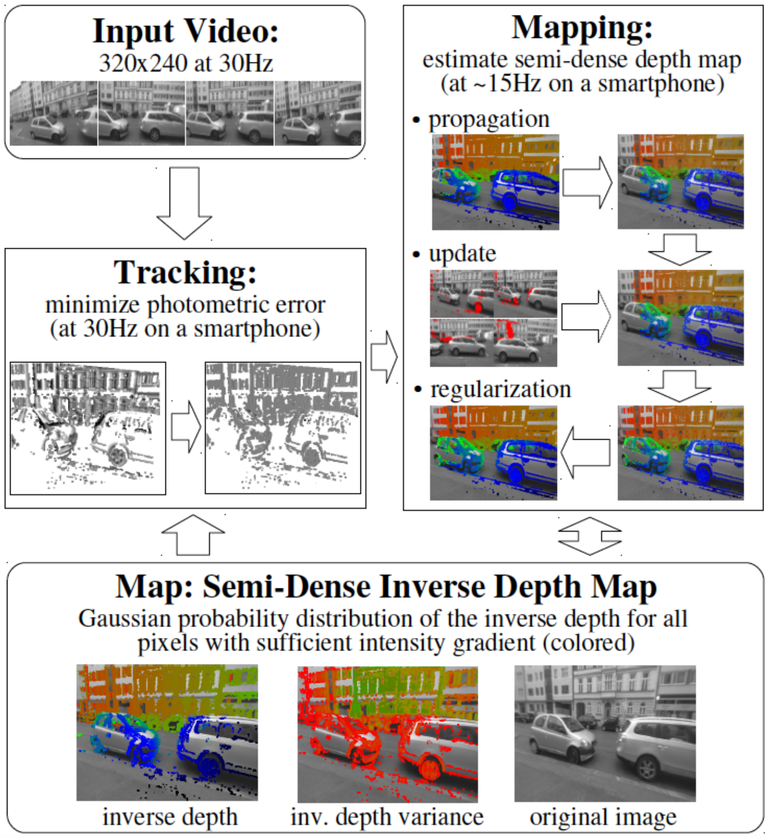


Figure 22: lsdar2

applications and also on UAV (Unmanned aerial vehicle) for state estimation.

It provides highly accurate visual-inertial odometry by using sliding window optimization, and also enables automatic initialization, relocalisation and loop closure function. The drift error accumulated over time is corrected by optimise the global pose graph maintained. In this picture Fig 23 from the paper [16] can be seen that the virtual object added in the AR application remains in place even after the camera have travelled a long loopy trajectory.

The program uses ceres solver[21] from Google[17] for non-linear optimization in state estimation stage, the loop closure function is implemented using bag of words

[16]http://www.ece.ust.hk/~eeshaojie/ismar2017peiliang.pdf
[17]https://github.com/ceres-solver/ceres-solver

method DBoW2[18] similar to ORB-SLAM's loop detection function. The screenshot of AR application is shown in Fig 24 and the structure of the application is shown in Fig 25. Both pictures are from the original paper.

[18] https://github.com/dorian3d/DBoW2

5 Project Method Overview

This project develops an Android application running on the Google Tango [1] tablet that can either logs RGB, depth and pose data from Tango to files, or run ElasticFusion on tablet using data from Tango platform, or run ElasticFusion using data from dataset on the tablet.

The application starts with sample Tango demo applications from Google that uses Tango C API, the code for these applications are most written in C++ under the Java NDK(Native Development Kit) on Android platform, which allows it to use the Tango Client C API, the source code for these demo applications can be found on Github[19].
One demo application that is useful for this project is the RGB depth sync application[20], it make use of both Tango's Motion Tracking and Depth Perception feature. And it calculates a depth map by use an upsample method to match the Tango Point Cloud data with each pixel in the RGB image from color camera, and it renders the RGB image on screen with an overlay of grey scale depth map that can adjust the alpha value. The screen shot of only RGB image rendered is shown in Fig 29, and the screen shot with the depth map overlay set to about 0.5 is shown in Fig 31, also screen shot of only depth map rendered(alpha value of 1) is shown in Fig 32.
The application developed in this project make use of the code for Advanced Estimation in Robotics (433H) course practical exercise as the basic frame work, it's also an Android application running on the Google Tango tablet. On top of this application, first using the Tango Client C API the RGB image data, Tango Point Cloud data, and Tango Pose data is acquired. The RGB image data from camera is gain as the RGB color value of each pixel, and the Tango Point Cloud data is in the form of an array of 3D point coordinates, where each 3D point represent a pixel in the depth image where depth sensor has a reading. And the Pose data comes in as the latest pose estimated by Tango VIO(Visual Inertial Odometry) frontend using the IMU onboard and mono image from wide angle motion tracking camera, details of technique used in the Tango VIO can be found in [4] and [22], but in this project it is treated as a black box.
firstly the code is combined with the logger for ElasticFusion[21] to write the RGB, depth and pose data in format required by ElasticFusion acquired from Tango into two files on the Tango tablet, which allows ElasticFusion to be run on desktop using these two files as raw dataset. Then the core function part of code for ElasticFusion is ported and adapted to Android with its dependencies changed or removed, so that ElasticFusion can be run on Android while its source remains in C++. One thing worth notice in here is that this project uses an internal version of ElasticFusion source code at Dyson Robotics Lab in Imperial College London, instead of the open source version on Github[22]. The final produced application is able to log the

[19] https://github.com/googlesamples/tango-examples-c
[20] https://github.com/googlesamples/tango-examples-c/tree/master
[21] https://github.com/mp3guy/Logger1
[22] https://github.com/mp3guy/ElasticFusion

data from sensors on Tango into files, or run ElasticFusion from sensor data in real time, or run ElasticFusion from raw dataset files on the device with or without pose data. This provides various way to do experiments or measure efficiency during the project. However, due to time and effort limitations, no interface for showing the progress of ElasticFusion running is built, this is different from original planning but the reasons will be explain in detail later. Right now on the screen of the application, interface only shows the image that color camera is taking and a few buttons, screen shot and description will be provided in later section.

In this project the goal is run ElasticFusion[2] on Google Tango Tablet for real time 3D reconstruction, but the algorithm will be adapted to fit into the Tango platform, also to make use of the Tango motion tracking and depth perception API. Given that Tango's camera pose estimation already make use of RGB-D and IMU sensor data, the pose tracking could possibly be replaced with 3D pose gain directly from Tango. Also, camera frame and 3D point cloud data for a time point can be acquired with the API. However, although the Tango 3D Reconstruction Library C provides functions and structs reconstruct 3D suface using mesh or voxel grid, it is different than the surfel map that we will be using for ElasticFusion[2]. However, it will be worth to study the sample project of building mesh and rendering on screen.

First, we need to set up a simple demo Google Tango Android application for motion tacking that output real time pose data into the logs.
Then camera frames needed to be rendered on screen in real time with buttons to resume and pause the rendering.
Next step is to make use of the mesh builder in Tango 3D Reconstruction Library for fusing point clouds and use OpenGL to render it on screen from the camera view point. After that we can build out the structs and functions for surfel map and global map representation used in ElasticFusion[2], in which the surfels are divided into active and inactive types according the time it is last observed. With the surfel map classes, integration with Tango could be done by try to extract surfels from point cloud data and then render the surfels into the screen.
Then the structs and functions for deformation graph, its nodes and surface constraints will be built to allow for non rigid deformation of the surface.
In order to perform the optimization using surface constraints, global and local loop closure needs to be done. For global loop closure, ElasticFusion[2] uses randomised fern encoding database to store frame information and check for match frames. Therefore fern and frame structs and this store, lookup mechanism needs to be implemented.
With global loop closure in place, this can be tested out by manually triggering the loop closing and see whether deformation graph is applied, which will cause the surfaces to be realigned, and camera pose be updated.
If this is successful, local loop closure can also be done using model-to-model tracking by trying to register the active and inactive point clouds together under the current frame view point. This is done by using the same tracking and optimization process in global loop sure.

It will be the ideal case if the above is all achieved before the project deadline, since this is a very challenging project. And if at a certain stage it is realised that it could not be done before the deadline, the progress can be fall back to the previous milestone and start clean up and report write up.
However, extension and optimization could be done if more time is given and basic solution is completed. Until here the 3D reconstruction process will be running on Tango tablet's NVIDIA Tegra K1 processor without utilizing the CUDA feature on it to improve performance by parallel processing. A optimization could be to use CUDA in the camera tracking process. Also on the application side, extra features can be added such as saving the reconstructed surfel map on the Tango tablet, add an interactive view of reconstructed 3D surface map on the screen to allow for drag and zoom operation, showing the camera's current pose on the map. Further more, demo application and be built to utilising the augmented reality ability on Tango platform, such as placing a virtual static object on the 3D reconstructing surface.

```
struct PoseData {
    double orientation[4];
    double translation[3];
}
```

A quaternion that defines the rotation of the target frame with respect to the base frame.
A 3D vector that defines the translation of the target frame with respect to the base frame.
An actual pose struct contains other fields, such as a timestamp and a copy of the frame pair,

Frames of Reference

$SE(3)$
$\mathfrak{so}(3)$
global pose of the camera $\boldsymbol{P}_t$ (w.r.t. a global frame $\xrightarrow{F}_G$)

a transformation matrix where: $\boldsymbol{P}_t = \begin{bmatrix} & R_t & t_t & \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{SE}(3)$.
with rotation $\boldsymbol{R}_t \in SO(3)$ and translation $\boldsymbol{t}_t \in \mathbb{R}^3$.
MSCKF(Multi-State Constraint Kalman Filter of Mourikis and Roumeliotis)[4]
[22]
Google Tango [1]

```
struct RGBDdata {
  double pointCloudTimestamp;
  uint32_t pointCloudNumpoints;
  float *pointCloudPoints;
  uint8_t *image;
```

```
  double colorTimeStamp;
  int64_t m_lastTimestamp;
  TangoPoseData pose;
};
```

6 Project Implementation

The central part of the report usually consists of three of four chapters detailing the technical work undertaken during the project. The structure of these chapters is highly project dependent. They can reflect the chronological development of the project, e.g. design, implementation, experimentation, optimisation, evaluation etc. although this is not always the best approach. However you choose to structure this part of the report, you should make it clear how you arrived at your chosen approach in preference to the other alternatives documented in the background. If you have built a new piece of software you should describe and justify the design of your program at some high level, possibly using an approved graphical formalism such as UML. It should also document any interesting problems with, or features of, your implementation. Integration and testing are also important to discuss in some cases. You need to discuss the content of these sections thoroughly with your supervisor.

============

6.1 Tango Platform

Tango is a platform that uses computer vision to give devices the ability to understand their position relative to the world around them. Its similar to how you use your eyes to find your way to a room, and then to know where you are in the room and where the floor, the walls, and objects around you are. These physical relationships are an essential part of how we move through our daily lives. Tango gives mobile devices this kind of understanding by using three core technologies: Motion Tracking, Area Learning, and Depth Perception.

Google Tango[23] is an augmented reality computing platform by Google that give mobile devices the ability to sense 3D space information. Equipped with this platform Google's Tango tablet is an Android device that carry sensors including 4MP(Mega Pixel) color camera, Inertial Measurement Unit (IMU), and Infra-Red depth camera. it also carrys NVIDIA Tegra K1 processor which provides more powerful computational power than other mobile devices. Given this cheap commodity mobile hardware and software, this project aims to build a fully self-contained 3D dense surface reconstruction solution using state-of-the-art RGB-D(csolor and depth) SLAM (Simultaneous Localization And Mapping) algorithms.

[23] https://get.google.com/tango/

In the recently published dense RGB-D SLAM algorithms, ElasticFusion[2] published by Dyson Robotics Lab[24] at Imperial College London is one that achieve state-of-the-art performance by applying local and global loop closures using a deformation graph, so that drifts raised by accumulated errors can be recovered and global consistency of the map is maintained. The fact Elastic Fusion scored high marks on benchmarks for trajectory estimation and surface estimation makes it perfect to be applied on Google Tango Tablet. Also, the current code exist for ElasticFusion running on desktop computer made heavily use of CUDA(Compute Unified Device Architecture) for GPU general data processing, which is supported by the NVIDIA Tegra K1 processer on Google Tango tablet.

Tango offers APIs in C and Java, and an SDK for Unity.

This project have attempted to port the code for ElasticFusion onto Google Tango Tablet and integrated with 3D points data from Tango platform, although the result of running ElasticFusion on the tablet has shown to be not effective enough for self-contained real time application due to processing power and sensor problem on the tablet and time and effort limitation in this project, this attempt has still proved that it is possible and more convenient to record the 3D space data on a hand held portable device first, and then let the data to be processed on a more power desktop computer for high quality 3D surface reconstruction.

Also, since the Tango platform on the tablet provides pose estimation feature, this pose is simply feed into the ElasticFusion program run on the tablet, so that the ElasticFusion part doesn't need to waste time and processing power to execute the camera tracking part. However, the pose provided by Tango is not accurate enough and during the reconstruction process ElasticFusion will alter the pose if global or local loop closure happens, in this project a simply way to correct pose input to ElasticFusion by Tango is developed to over come this problem.

the Google Tango tablet with both RGB and depth camera would be the perfect mobile device to implement a portable version of ElasticFusion on. The Google Tango tablet carries Android 4.2.2 OS, has 4GB of RAM, a quadcore Nvidia Tegra K1 graphics card supporting CUDA, six-axis gyroscope and accelerometer, a wide-angle 120 field of view tracking camera which refreshes at 60Hz, a projective depth sensor which refreshes at 3Hz, and a 4 megapixel color sensor which refreshes at 30Hz. The processing power on this tablet succeeds most mobile devices on the market right now, which makes running ElasticFusion on the device alone become possible. Since ElasticFusion requires both high end processor and graphic card on a desktop computer, from the read me of ElasticFusion on Github: "A very fast nVidia GPU (3.5TFLOPS+), and a fast CPU (something like an i7)."

Nvidia Tegra K1 192 GPU core, Direct3D 12, OpenGL ES 3.1, CUDA 6.5, OpenGL

[24] `http://www.imperial.ac.uk/dyson-robotics-lab/`

4.4/OpenGL 4.5, and Vulkan, 365 G FLOPS, 192 CUDA cores * 2 FLOPS per core * 950MHz, sm30, floating point operations per second (FLOPS)
laptop: GTX765M PICe SSE2 , 1224 GFLOPS

=======

requirement:
A very fast nVidia GPU (3.5TFLOPS+), and a fast CPU (something like an i7). 3500 GFLOPS
Nvidia Tegra K1 approx = GT620-GT540

make it faster: Kintinuous will run at 30Hz on a modern laptop on battery power these days). You can try disabling SO(3) pre-alignment, enabling fast odometry, only using either ICP or RGB tracking and not both, running in open loop mode or disabling the tracking pyramid. All of these will cost you accuracy.

```
TangoErrorType TangoService_connectOnPoseAvailable(
    uint32_t count, const TangoCoordinateFramePair* frames,
    void (*TangoService_onPoseAvailable)(void* context,
                                         const TangoPoseData* pose)
```

```
TangoErrorType TangoSupport_updatePointCloud(
    ↪ TangoSupportPointCloudManager *manager, const TangoPointCloud *
    ↪ point_cloud)
```

```
/// Get the intrinsic calibration parameters for a given camera. The
    ↪ intrinsics
/// are as specified by the TangoCameraIntrinsics struct and are
    ↪ accessed via
/// the API.
/// @param camera_id The camera ID to retrieve the calibration
    ↪ intrinsics for.
/// @param intrinsics A TangoCameraIntrinsics struct that must be
    ↪ allocated
/// before calling, and is filled with calibration intrinsics for the
    ↪ camera
/// @p camera_id upon successful return.
/// @return @c TANGO_SUCCESS on successfully retrieving calibration
    ↪ intrinsics,
/// @c TANGO_INVALID if the @p camera_id is out of range or if
    ↪ intrinsics
/// argument was @c NULL, or @c TANGO_ERROR if an internal error
    ↪ occurs
/// while getting intrinsics.
```

```
TangoErrorType TangoService_getCameraIntrinsics(TangoCameraId
    ↪ camera_id,
    TangoCameraIntrinsics* intrinsics);
```

```
/// Connect a Texture ID to a camera; the camera is selected by
    ↪ specifying a
/// TangoCameraId. Currently only @c TANGO_CAMERA_COLOR and
/// @c TANGO_CAMERA_FISHEYE are supported.
/// The texture must be the ID of a texture that has been allocated
    ↪ and
/// initialized by the calling application. The TangoConfig flag
/// config_enable_color_camera must be set to true for
    ↪ connectTextureId
/// to succeed after TangoService_connect() is called.
///
/// @param id The ID of the camera to connect this texture to. Only
/// @c TANGO_CAMERA_COLOR and @c TANGO_CAMERA_FISHEYE are supported.
/// @param context The context returned during the onFrameAvailable
    ↪ callback.
/// @param tex The texture ID of the texture to connect the camera to.
    ↪  Must be
/// a valid texture in the applicaton.
/// @return @c TANGO_INVALID if the camera ID is not valid, or @c
    ↪ TANGO_ERROR if
/// an internal error occurred.
TangoErrorType TangoService_connectTextureId(
    TangoCameraId id, unsigned int tex, void* context,
    TangoService_OnTextureAvailable callback);
```

```
TangoErrorType TangoSupport_createPointCloudManager(
  size_t max_points,
  TangoSupportPointCloudManager **manager
)
```

```
/// Update the texture that has been connected to camera referenced by
/// @p TangoCameraId. The texture is updated with the latest image
    ↪ from the
/// camera.
/// If timestamp is not @c NULL, it will be filled with the image
    ↪ timestamp.
/// @param id The ID of the camera to connect this texture to. Only
/// @c TANGO_CAMERA_COLOR and @c TANGO_CAMERA_FISHEYE are supported.
/// @param timestamp Upon return, if not NULL upon calling, timestamp
    ↪ contains
/// the timestamp of the image that has been pushed to the connected
/// texture.
```

```
/// @return @c TANGO_INVALID if @p id is out of range or if a texture
    ↪ ID was
/// never associated with the camera; @c TANGO_SUCCESS otherwise.
TangoErrorType TangoService_updateTexture(TangoCameraId id, double*
    ↪ timestamp);
```

```
TangoErrorType TangoSupport_getLatestPointCloudAndNewDataFlag(
  TangoSupportPointCloudManager *manager,
  TangoPointCloud **latest_point_cloud,
  bool *new_data
)
```

```
/// Update a GL_TEXTURE_EXTERNAL_OES texture to the latest camera
/// image available.
///
/// If timestamp is not NULL, it will be filled with the image
/// timestamp. The texture passed in must be of type @c
/// GL_TEXTURE_EXTERNAL_OES. This is not checked.
///
/// @param id The ID of the camera to use for the update. Only @c
/// TANGO_CAMERA_COLOR and @c TANGO_CAMERA_FISHEYE are supported.
/// @param tex Texture to update. This texture must be of type @c
/// GL_TEXTURE_EXTERNAL_OES.
/// @param timestamp Upon return, if not NULL upon calling, timestamp
/// contains the timestamp of the image that has been pushed to the
/// texture.
/// @return @c TANGO_INVALID if @p id is out of range or if @c tex is
    ↪ not a
/// texture ID; @c TANGO_SUCCESS otherwise.
TangoErrorType TangoService_updateTextureExternalOes(TangoCameraId id,
                                                     unsigned int tex,
                                                     double* timestamp);
```

```
/// Get a pose at a given timestamp from the base to the target frame.
///
/// All poses returned are marked as @c TANGO_POSE_VALID (in the
/// status_code field on TangoPoseData) even if they were marked as
/// @c TANGO_POSE_INITIALIZING in the callback poses.
/// To determine when initialization is complete, register a callback
    ↪ using
/// TangoService_connectOnPoseAvailable() and wait until you receive
    ↪ valid data.
///
/// If no pose can be returned, the status_code of the returned pose
    ↪ will be
/// @c TANGO_POSE_INVALID.
```

```
///
/// @param timestamp Time specified in seconds. If not set to 0.0,
    ↪ getPoseAtTime
/// retrieves the interpolated pose closest to this timestamp. If set
    ↪ to
/// 0.0, the most recent pose estimate for the target-base pair is
    ↪ returned.
/// The time of the returned pose is contained in the pose output
    ↪ structure
/// and may differ from the queried timestamp.
/// @param frame A pair of coordinate frames specifying the
    ↪ transformation to be
/// queried for. For example, typical device motion is given by a
    ↪ target
/// frame of @c TANGO_COORDINATE_FRAME_DEVICE and a base frame of
/// @c TANGO_COORDINATE_FRAME_START_OF_SERVICE.
/// @param pose The pose of target with respect to base frame of
    ↪ reference. Must
/// be allocated by the caller, and is overwritten upon return.
/// @return @c TANGO_SUCCESS if a pose was returned successfully.
    ↪ Check the
/// @c status_code attribute on the returned @c pose to see if it is
/// valid. Returns @c TANGO_INVALID if the base and target frame are
    ↪ the
/// same, or if the base or the target frame is not valid, or if
/// timestamp is less than 0, or if the service has not yet begun
    ↪ running
/// (TangoService_connect() has not completed).
TangoErrorType TangoService_getPoseAtTime(double timestamp,
                                          TangoCoordinateFramePair frame,
                                          TangoPoseData* pose);
```

```
TangoErrorType TangoSupport_calculateRelativePose(double
    ↪ base_timestamp, TangoCoordinateFrameType base_frame, double
    ↪ target_timestamp, TangoCoordinateFrameType target_frame,
    ↪ TangoPoseData *base_frame_T_target_frame)
```

```
TangoErrorType TangoService_connectOnPointCloudAvailable(void(*)(void
    ↪ *context, const TangoPointCloud *cloud)
```

```
/// The TangoPoseData struct contains pose information returned from
    ↪ motion
/// tracking.
/// The device pose is given using Android conventions.
```

```
/// See the Android
/// <a href="http://developer.android.com/guide/topics/sensors/
    ↪ sensors_overview.html#sensors-coords">
/// Sensor Overview</a> page for more information.
typedef struct TangoPoseData {
  ///< Unused. An integer denoting the version of the structure.
  uint32_t version;

  /// The timestamp of the pose estimate, in seconds.
  double timestamp;

  /// Orientation, as a quaternion, of the pose of the target frame
      ↪ with
  /// reference to the base frame.
  /// Specified as (x,y,z,w) where RotationAngle is in radians:
  /// @code
  /// x = RotationAxis.x * sin(RotationAngle / 2)
  /// y = RotationAxis.y * sin(RotationAngle / 2)
  /// z = RotationAxis.z * sin(RotationAngle / 2)
  /// w = cos(RotationAngle / 2)
  /// @endcode
  double orientation[4];

  /// Translation, ordered x, y, z, of the pose of the target frame
      ↪ with
  /// reference to the base frame.
  double translation[3];
  /// The status of the pose, according to the pose lifecycle.
  TangoPoseStatusType status_code;
  /// The pair of coordinate frames for this pose. We retrieve a pose
      ↪ for a
  /// target coordinate frame (such as the Tango device) against a base
  /// coordinate frame (such as a learned area).
  TangoCoordinateFramePair frame;
  uint32_t confidence; ///< Unused. Integer levels are determined by
      ↪ service.
  float accuracy; ///< Unused. Reserved for metric accuracy.
} TangoPoseData;

/// The TangoTransformation struct contains an orientation and
    ↪ translation
/// that can be applied to transform a point.
```

```
/// TangoPointCloud contains information returned from the depth
    ↪ sensor.
typedef struct TangoPointCloud {
```

```
  /// An integer denoting the version of the structure.
  uint32_t version;

  /// Time of capture of the depth data for this struct (in seconds).
  double timestamp;

  /// The number of points in depth_data_buffer populated successfully.
      ↪  This
  /// is variable with each call to the function, and is returned as
      ↪ number of
  /// (x,y,z,c) points populated.
  uint32_t num_points;

  /// An array of packed {X, Y, Z, C} values. {X, Y, Z} is a coordinate
      ↪  triplet
  /// (in meters). C is a confidence value, in the range of [0, 1],
      ↪ where 1
  /// corresponds to full confidence. +Z points in the direction of the
      ↪  camera's
  /// optical axis, perpendicular to the plane of the camera. +X points
      ↪  toward
  /// the user's right, and +Y points toward the bottom of the screen.
  /// The origin is the focal center of the depth camera.
  float (*points)[4];
} TangoPointCloud;
```

tangoAPI

```
Code
```

6.2 Logging data from Tango

6.3 Compiling ElasticFusion on Tango

6.3.1 Dependencies

6.3.2 OpenCV

6.3.3 CUDA

6.3.4 Pangolin

Pangolin[25]

[25] https://github.com/stevenlovegrove/Pangolin

6.3.5 SuiteSparse

26
27

6.3.6 OpenGL

6.3.7 Eigen

[26] `http://faculty.cse.tamu.edu/davis/suitesparse.html`
[27] `https://github.com/stevenlovegrove/Pangolin`

6.4 Running ElasticFusion on Tango

6.4.1 Running ElasticFusion with pose on Tango

6.4.2 Running ElasticFusion dataset on Tango

6.4.3 Running Tango dataset on Tango

6.5 Combine ElasticFusion with pose from Tango in Loop closure mode

The project timeline is shown in Fig 41 with each task corresponding to a stage above.

7 Evaluation

7.1 Reconstruction Accuracy

7.2 depth sensor noise

depthsparse

7.3 Computation power problem

7.4 Optimization problem

Be warned that many projects fall down through poor evaluation. Simply building a system and documenting its design and functionality is not enough to gain top marks. It is extremely important that you evaluate what you have done both in absolute terms and in comparison with existing techniques, software, hardware etc. This might involve quantitative evaluation, for example based on numerical results, performance etc. or something more qualitative such as expressibility, functionality, ease-of-use etc. At some point you should also evaluate the strengths and weaknesses of what you have done. Avoid statements like ”The project has been a complete success and we have solved all the problems asssociated with blah...; - you will be shot down immediately! It is important to understand that there is no such thing as a perfect project. Even the very best pieces of work have their limitations and you are expected to provide a proper critical appraisal of what you have done.

=============

The evaluation of this project is intended to use the same mechanism as in original ElasticFusion[2] paper, as the goal of this project is to deploying this algorithm on Google Tango tablet.
In order to measure the performance of this solution by quantity and quality. Both algorithm and computational performance need to be benchmarked.
Both A. Trajectory Estimation To evaluate the trajectory estimation performance of our approach we test our system on the RGB-D benchmark of Sturm et al. [22]. This benchmark provides synchronised ground truth poses for an RGB-D sensor moved through a scene, captured with a highly precise motion capture system. We use the absolute trajectory (ATE) root-mean-square error metric (RMSE) in our comparison, which measures the root-mean-square of the Euclidean distances between all estimated camera poses and the ground truth poses associated by timestamp [22].
B. Surface Estimation We evaluate the surface reconstruction results of our approach on the This benchmark provides ground truth poses for a camera moved through a synthetic environment as well as a ground truth 3D model which can be used to evaluate surface reconstruction accuracy.
Computational Performance
The evaluation of this project is intended to use the same mechanism as in original ElasticFusion[2] paper, as the goal of this project is to deploying this algorithm on

Google Tango tablet.
In order to measure the performance of this solution by quantity and quality. Both algorithm and computational performance need to be benchmarked.
For the algorithm, trajectory accuracy can be tested using sample data sets to provide RGB-D input, then the trajectory output can be compared with the ground truth data that is obtained by using industrial standard system. The root mean square distance between every estimated pose and ground truth pose can be calculated to provide error at each timepoint. In the 3D reconstruction part, similar method can be used to compare the reconstructed surface to benchmark data for accuracy estimation. However, since the post data and image input are provided by Tango platform, a way is needed to fed the benchmark data into Tango.
For computational Performance, we can simply plot the frame rate during the process of 3D scanning to check if it is within the acceptable range for real time SLAM algorithm.
The evaluation of this project is intended to use the same mechanism as in original ElasticFusion[2] paper, as the goal of this project is to deploying this algorithm on Google Tango tablet.
In order to measure the performance of this solution by quantity and quality. Both algorithm and computational performance need to be benchmarked.
For the algorithm, trajectory accuracy can be tested using sample data sets to provide RGB-D input, then the trajectory output can be compared with the ground truth data that is obtained by using industrial standard system. The root mean square distance between every estimated pose and ground truth pose can be calculated to provide error at each timepoint. In the 3D reconstruction part, similar method can be used to compare the reconstructed surface to benchmark data for accuracy estimation. However, since the post data and image input are provided by Tango platform, a way is needed to fed the benchmark data into Tango.
For computational Performance, we can simply plot the frame rate during the process of 3D scanning to check if it is within the acceptable range for real time SLAM algorithm.

7.4.1 Software limitation

8 Conclusions and Future Work

The project's conclusions should list the things which have been learnt as a result of the work you have done. For example, "The use of overloading in C++ provides a very elegant mechanism for transparent parallelisation of sequential programs", or "The overheads of linear-time n-body algorithms makes them computationally less efficient than O(n log n) algorithms for systems with less than 100000 particles". Avoid tedious personal reflections like "I learned a lot about C++ programming...", or "Simulating colliding galaxies can be real fun...". It is common to finish the report by listing ways in which the project can be taken further. This might, for example, be a plan for doing the project better if you had a chance to do it again, turning the project deliverables into a more polished end product, or extending the project into a programme for an MPhil or PhD.

optimization
slow, GPU, memmory
resolution,

9 Bibliography

This consists of a list of all the books, articles, manuals etc. used in the project and referred to in the report. You should provide enough information to allow the reader to find the source. In particular references must contain all the information regarding the publication of the paper and must be consistently formatted. Usually this means:
* For journals: Authors, Title, Journal, volume number, issue number, page number, publisher, month, year. * For conferences: Authors, Title, Conference name, Place where held, publisher, page number, month, year. * For technical reports: Authors, Title, institution, Technical report number, month, year. * For web references: Authors, Title, Web-reference, date accessed. * URLs are optional for published work but preferred.
A weakness of many reports is inadequate citation of a source of information. It's easy to get this right so there are no excuses. Each entry in the bibliography should list the author(s) and title of the piece of work and should give full details of where it can be found. For example:
1 Bennett, A.J., Field, A.J. and Harrison, P.G., ”Modelling and Validation of Shared Memory Coherency Protocols”, Performance Evaluation, 1996, Vol. 27 28, 1996, pp. 541-562. rather than just listing the source as ”Performance Evaluation 1996”. Using a reference management programme can make your life simpler.See for example Bibdesk,JabRef,RefWorks, etc.

10 Appendix

The appendices contain information which is peripheral to the main body of the report. Information typically included are things like parts of the code, tables, proofs, test cases or any other material which would break up the theme of the text if it appeared in situ. You should try to bind all your material in a single volume if possible.

11 User Guide

For projects which result in a new piece of software you should provide a proper user guide providing easily understood instructions on how to use it. A particularly useful approach is to treat the user guide as a walk-through of a typical session, or set of sessions, which collectively display all the features of your system. Technical details of how the package works should be in the body of the report. Keep it concise and simple. The extensive use of diagrams illustrating the package in action prove particularly helpful. The user guide is sometimes included as a chapter in the main body of the report, but is often better in an appendix.

12 Program Listings

Complete program listings should NOT be part of the report except in specific cases at the request of your supervisor. The project report(s) must be bound in a departmental folder and must include a standard title page produced by the project co-ordinator. More of this nearer the date.
You are strongly advised to spend some time looking at the reports of previous project students to get a feel for what's good and bad. In June 1999 we introduced a ”Distinguished Project” classification, which is a formal recognition of outstanding projects for which no official prize was awarded. The complete list of prize winners and the other distinguished projects, along with links to the final reports, can be found here.

13 Word Count

There is no word count for the report as such, but ideally you should not exceed the limit of 40,000 words. Generally, our best projects tend to have the word count in the range of 20,000 - 35,000 words with a few going beyond that. This does not include the Appendices.

References

[1] Google. Google project tango, 2017. pages 2, 29, 31

[2] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 2016. pages 2, 13, 17, 30, 33, 44, 45

[3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. pages 10, 11, 12

[4] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Robotics and automation, 2007 IEEE international conference on*, pages 3565–3572. IEEE, 2007. pages 13, 29, 31

[5] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011. pages 13, 14, 17, 18, 22

[6] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992. pages 14

[7] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992. pages 14

[8] Fuxing Yin. *Volumetric Representation on KinecFusion*. csdn, 2016. pages 15

[9] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015. pages 15, 16, 17, 18

[10] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *3DTV-Conference, 2013 International Conference on*, pages 1–8. IEEE, 2013. pages 17

[11] Ben Glocker, Jamie Shotton, Antonio Criminisi, and Shahram Izadi. Real-time rgb-d camera relocalization via randomized ferns for keyframe encoding. *IEEE transactions on visualization and computer graphics*, 21(5):571–583, 2015. pages 18

[12] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015. pages 19

[13] Matthew Klingensmith, Ivan Dryanovski, Siddhartha Srinivasa, and Jizhong Xiao. Chisel: Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. In *Robotics: Science and Systems*, 2015. pages 21

[14] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 32(6):169, 2013. pages 21, 22

[15] O. Kahler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. H. S Torr, and D. W. Murray. Very High Frame Rate Volumetric Integration of Depth Images on Mobile Device. *IEEE Transactions on Visualization and Computer Graphics (Proceedings International Symposium on Mixed and Augmented Reality 2015*, 22(11), 2015. pages 22

[16] Raul Mur-Artal and Juan D Tardos. Orb-slam2: an open-source slam system for monocular, stereo and rgb-d cameras. *arXiv preprint arXiv:1610.06475*, 2016. pages 24

[17] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012. pages 24

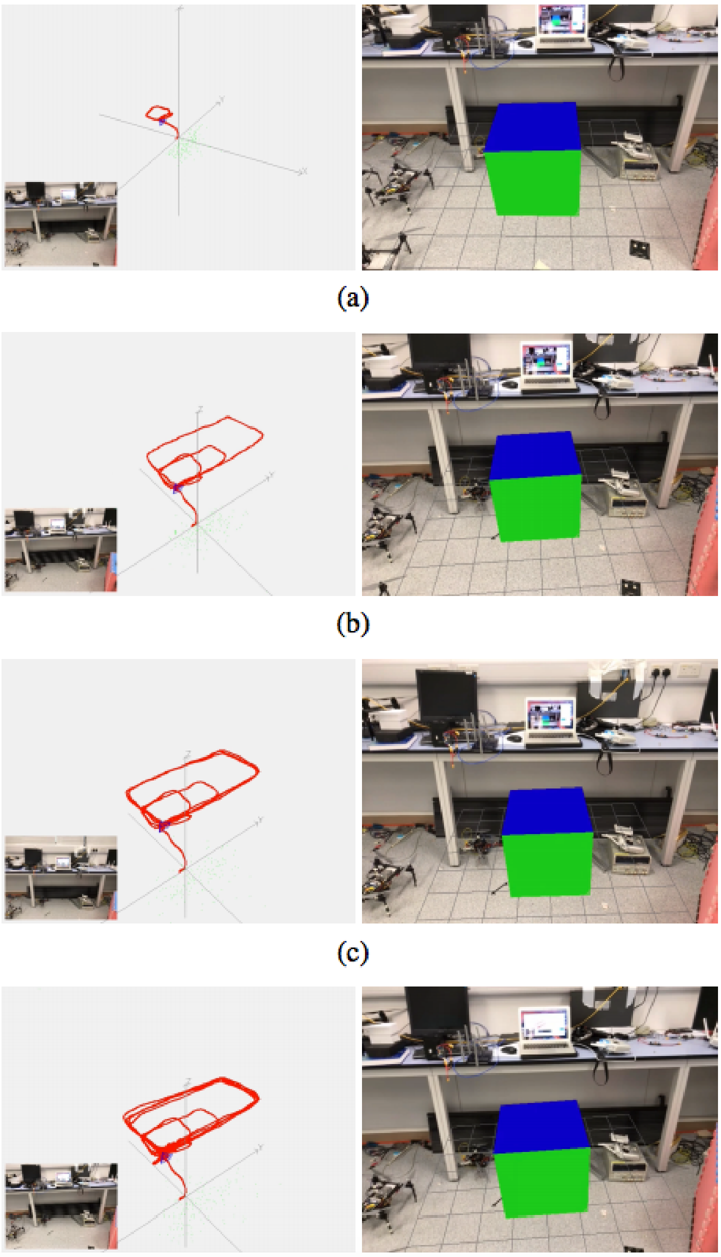
[18] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014. pages 26

[19] Thomas Schöps, Jakob Engel, and Daniel Cremers. Semi-dense visual odometry for ar on a smartphone. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 145–150. IEEE, 2014. pages 26

[20] Zhenfei Yang and Shaojie Shen. Monocular visual–inertial state estimation with online initialization and camera–imu extrinsic calibration. *IEEE Transactions on Automation Science and Engineering*, 14(1):39–51, 2017. pages 26

[21] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`. pages 27

[22] Dimitrios G Kottas, Joel A Hesch, Sean L Bowman, and Stergios I Roumeliotis. On the consistency of vision-aided inertial navigation. In *Experimental Robotics*, pages 303–317. Springer, 2013. pages 29, 31



(a)
(b)
(c)
(d)

Figure 23: vin1

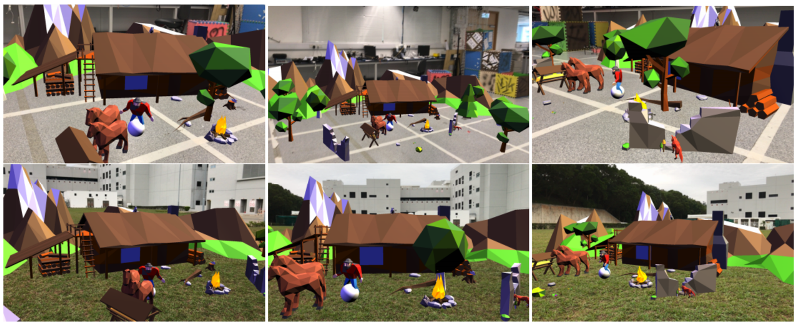


Figure 24: vin2

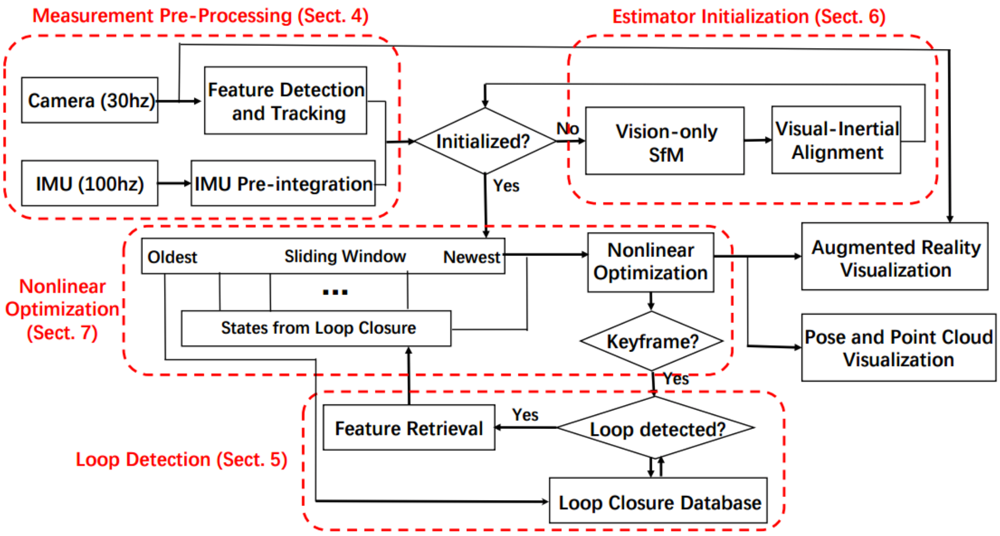


Figure 25: vin3

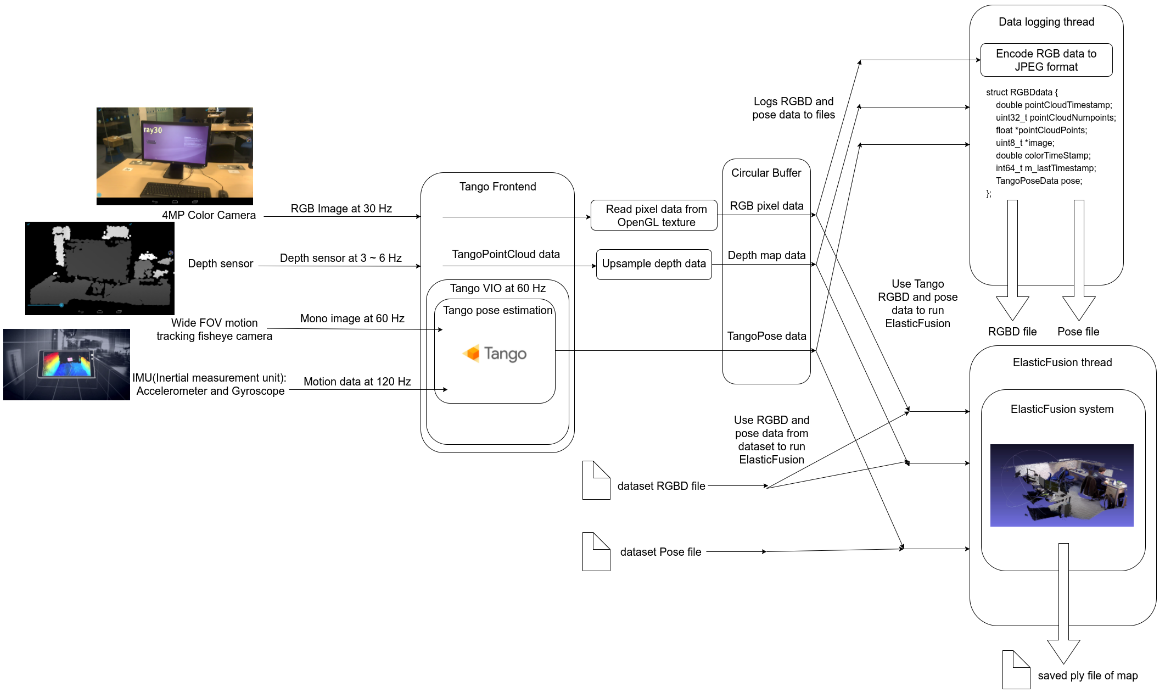


Figure 26: projectStructure

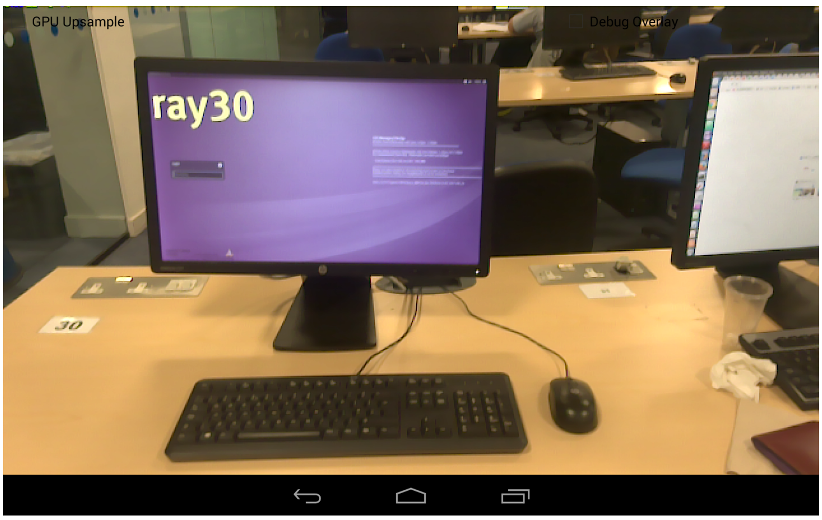


Figure 27: rgbd1

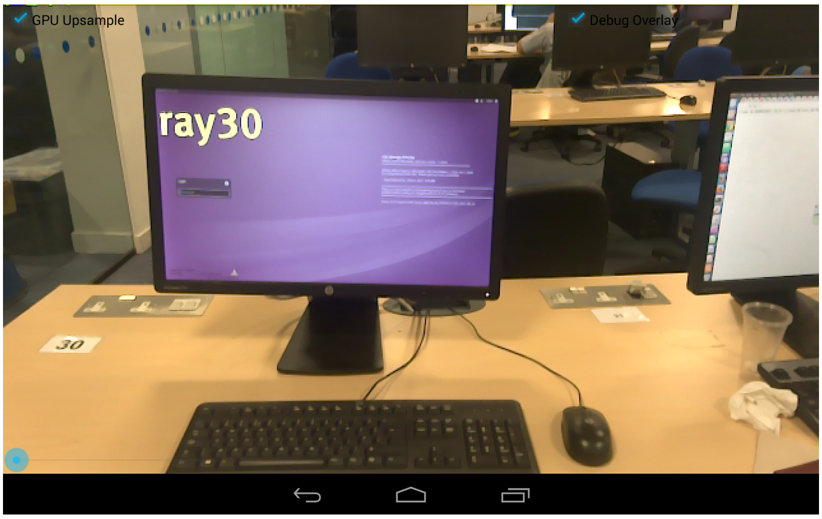


Figure 28: rgbd2

Figure 29: rgbd3

Figure 30: rgbd4

Figure 31: rgbd5

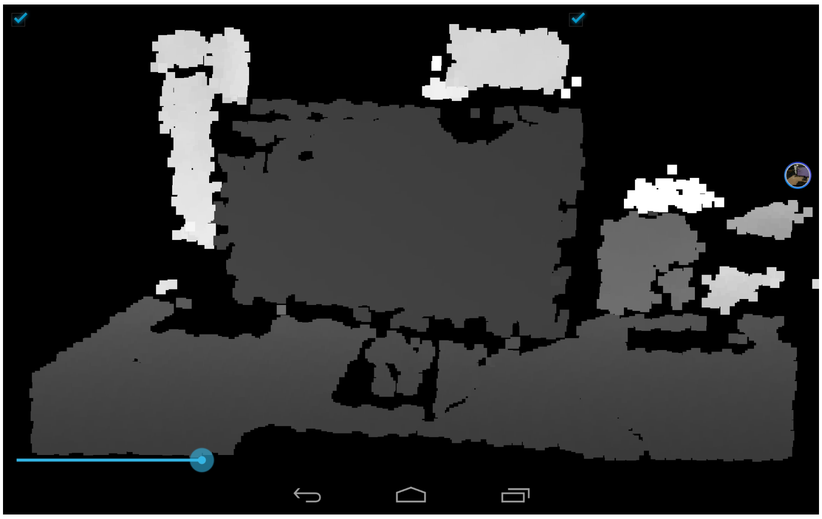


Figure 32: rgbd6

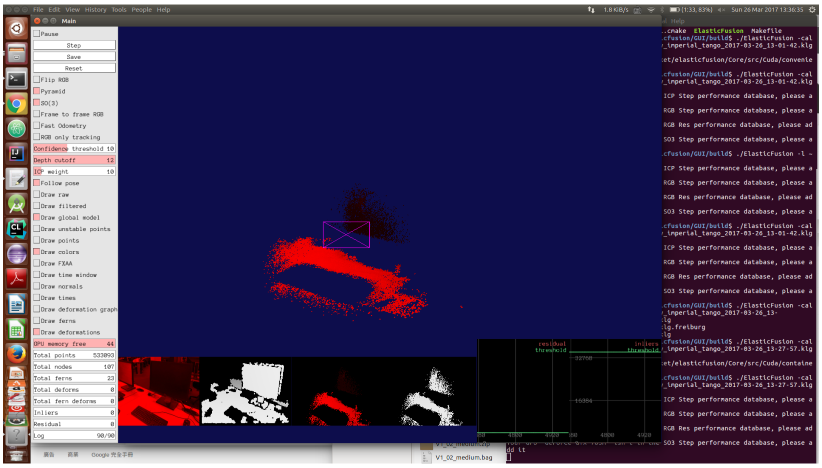


Figure 33: Screenshot1

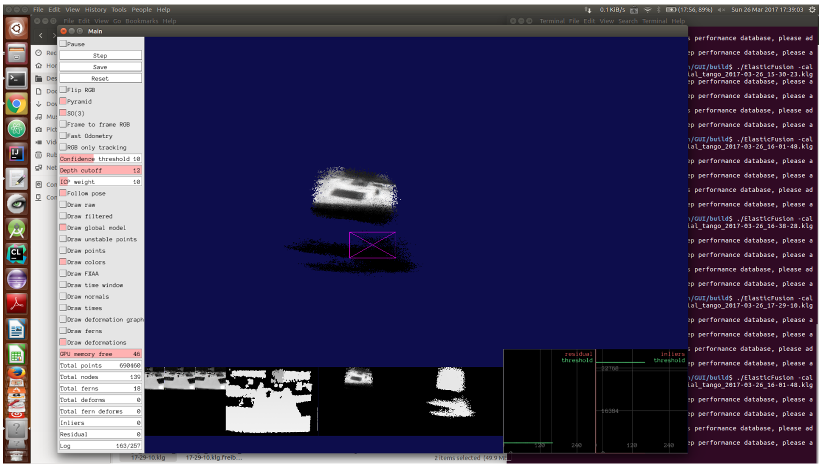


Figure 34: Screenshot2

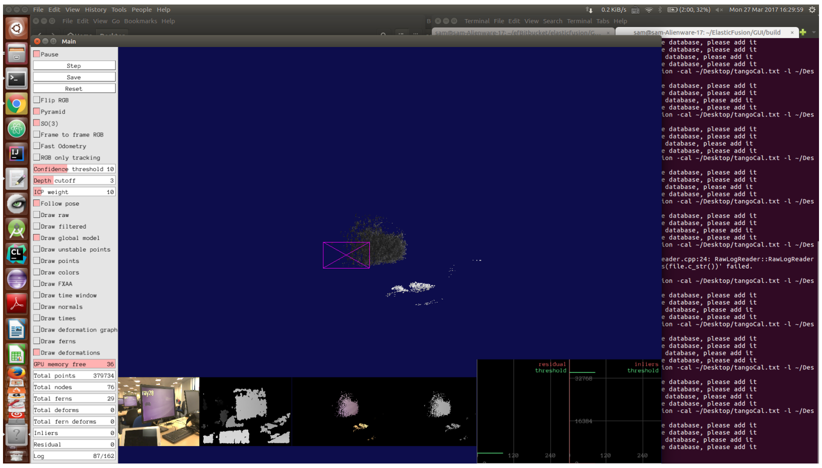


Figure 35: Screenshot3

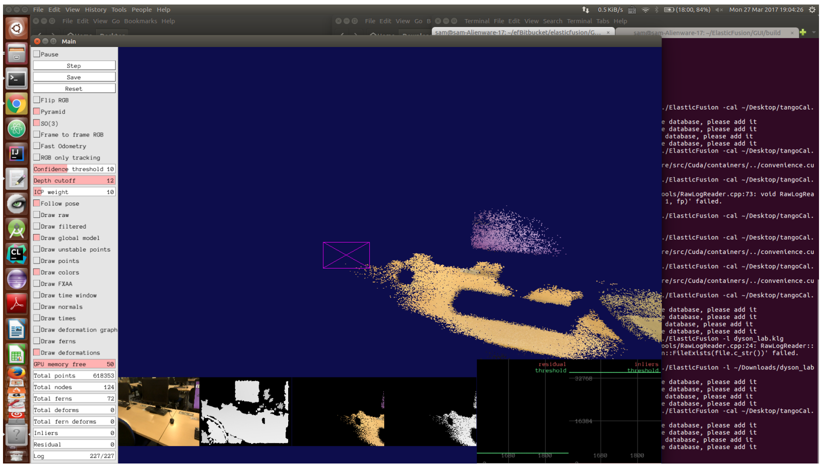


Figure 36: Screenshot9

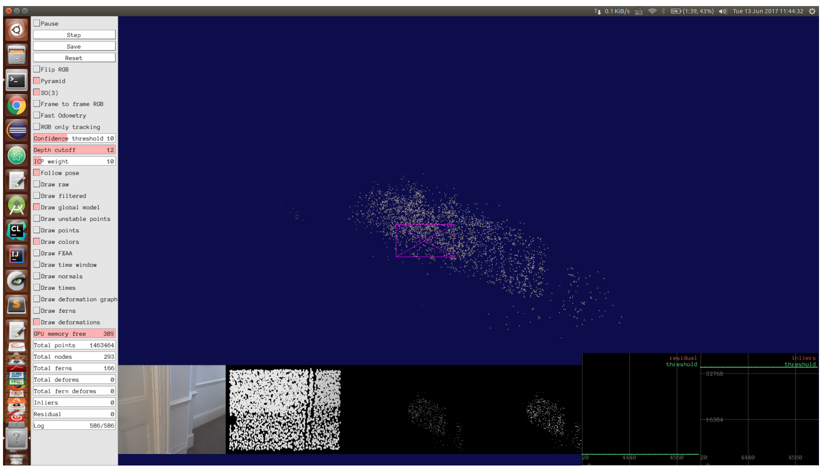


Figure 37: Screenshot10

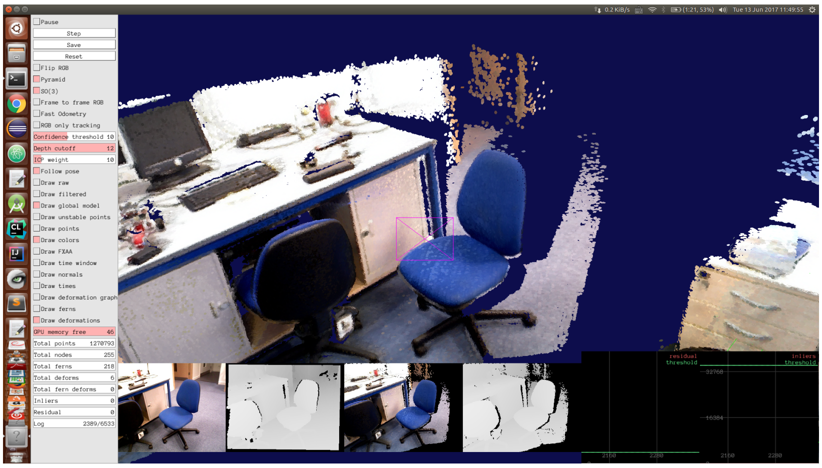


Figure 38: Screenshot11

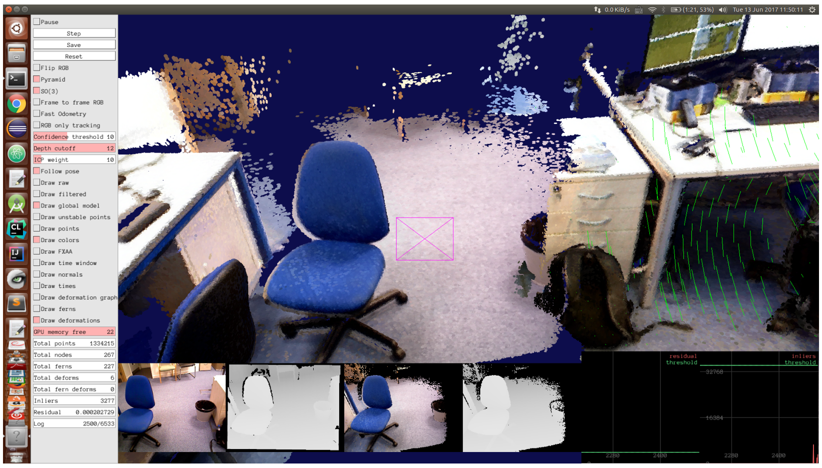


Figure 39: Screenshot14

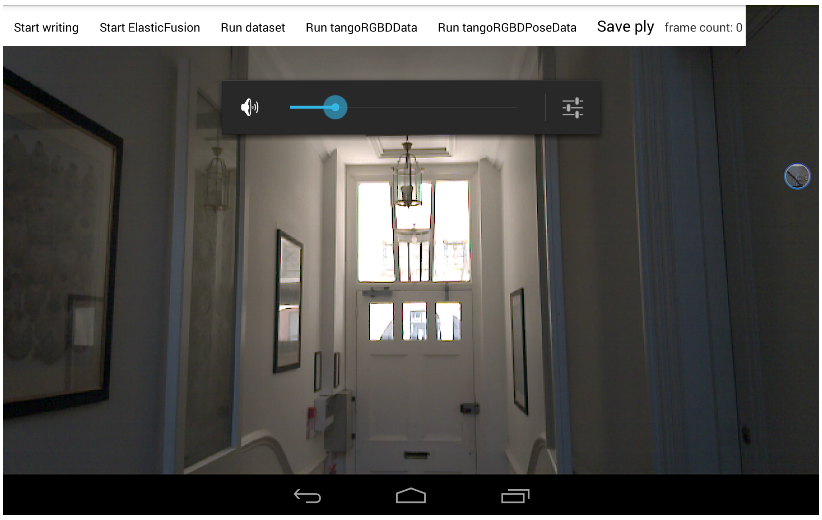


Figure 40: ScreenshotTango1

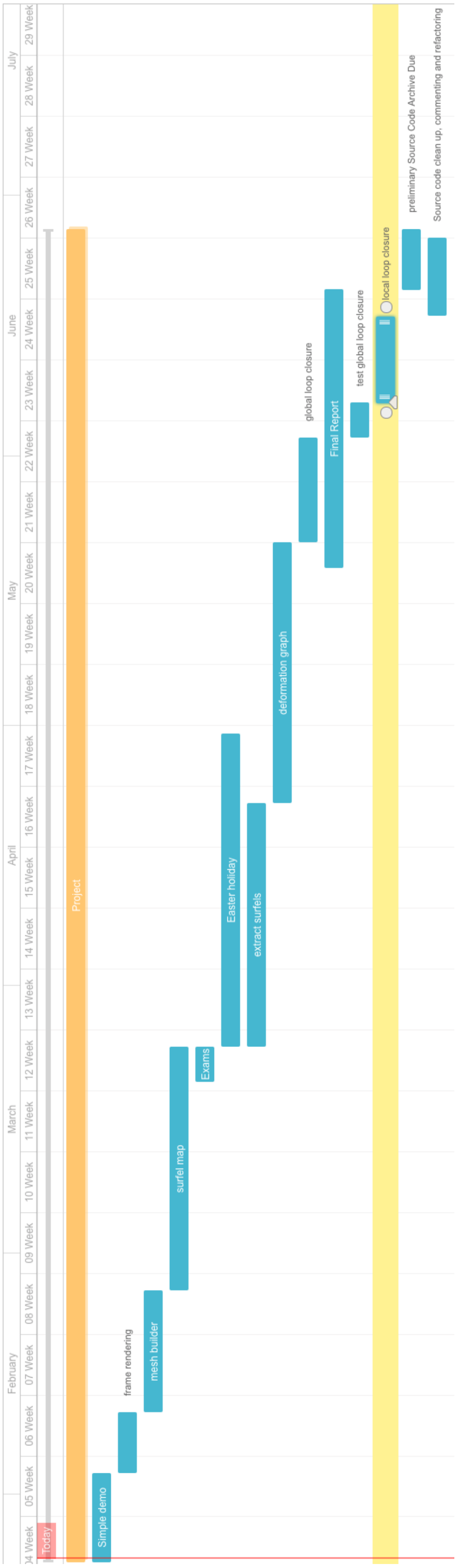


Figure 41: Graph for project timeline