Tampere University

Teo Niemirepo

CAMERA LOCALIZATION AND 3D SURFACE RECONSTRUCTION ON LOW-POWER EMBEDDED DEVICES

Bachelor of Science
Faculty of Information Technology and Communication Sciences
Examiner: University Lecturer Erja Sipilä
December 2021

ABSTRACT

Teo Niemirepo: Camera localization and 3D surface reconstruction on low-power embedded devices
Bachelor of Science
Tampere University
Electrical Engineering
December 2021

This Thesis explores the opportunities for real-time camera localization and 3D surface reconstruction on an embedded device and demonstrates one practical implementation of such. Previous implementations are analyzed, and their usability on embedded platforms is discussed. The importance of accurate and fast localization in modern and future applications is considered and taken into account in the practical implementation of the system.

3D localization and surface reconstruction can be utilized in a vast number of use cases. Some of the more prevalent use cases are its use in advanced robotics, security and military applications, geo scanning, aviation industry, and the entertainment sector. The recent advancements in extender reality and mobile devices have accelerated the adoption of high-performance localization even further.

In its core, the problem of 3D localization involves inferring the position and rotation of the device both in the local case in reference to the last few frames and in the global case in reference to all of the previous frames and reconstructed 3D landmarks. Augmenting the localization problem with the reconstruction of robust 3D point clouds and a surface adds additional constraints to the requirements. Mainly, the importance of both local and global camera pose consistency is accentuated due to the triangulation of the camera-space 2D image features into world-space 3D points necessitating the fulfillment of the cheirality constraint. Additionally, deviations in the camera poses induces unwanted noise into the point surface and causes cumulative distortions in the form of the 3D surface.

The implemented 3D localization and reconstruction system utilizes various simultaneous localization and mapping techniques for localizing the camera and a diverse set of structure-from-motion algorithms for reconstructing the real-world in virtual space. Concepts from edge computing and mobile robotics are used in speeding up the reconstruction and visualization workflow. On a high level, the system consists of eight (8) stages: 2D feature detection and matching, camera localization, landmark triangulation, wireless point cloud streaming, point cloud structuration, Poisson 3D surface reconstruction, and 3D visualization.

The algorithms involved are examined in detail and considered from the viewpoint of embedded and power constrained devices. Appropriate measures for optimization are taken when pertinent, and the performance of the system in various scenarios is quantified by the use of performance metrics.

The system is shown to be usable in real-world applications, and the obtained reconstruction results are compared against state-of-the-art open-source and academic solutions. The system is open-source under the MIT license and available on GitHub.

Keywords: Simultaneous Localization and Mapping, Structure-from-Motion, Embedded System

The originality of this thesis has been checked using the Turnitin Originality Check service.

TIIVISTELMÄ

Teo Niemirepo: Kameran paikantaminen ja 3D-pinnan rekonstruointi pienitehoisilla sulautetuilla järjestelmillä
Kandidaatintyö
Tampereen yliopisto
Sähkötekniikan tutkinto-ohjelma
Joulukuu 2021

Tämä kandidaatintyö tutki mahdollisuuksia reaaliaikaisessa kameran lokalisoinnissa ja 3D-pinnan muodostamisessa sulautetuilla laitteilla ja demonstroi yhden käytännön toteutuksen sellaisesta. Aikaisempia toteutuksia analysoitiin ja niiden käyttökelpoisuutta sulautetuilla järjestelmillä tutkittiin. Tarkan ja nopean lokalisaation tärkeyttä nykyisissä ja tulevissa sovelluksissa tarkasteltiin ja otettiin huomioon implementaatiovaiheessa.

3D-lokalisaatiota ja pinnan rekonstruointia voidaan hyödyntää useissa käyttötapauksissa. Joitain yleisimpiä käyttötarkoituksia ovat robotiikka, turvallisuus ja sotilaalliset sovellukset, geoskannaus, ilmailuteollisuus sekä viihdeala. Viimeaikaiset edistykset laajennetun todellisuuden *(XR, Extended Reality)* sovelluksissa ja mobiililaitteiden suoritustehossa ovat kiihdyttäneet korkeasuorituskykyisten lokalisaatiototeutusten käyttöönottoa.

3D-lokalisaatio pitää sisällään laitteen sijainnin ja asennon päättelemisen sekä lokaalissa kehyksessä verrattuna muutamaan aikaisempaan kameran kuvaan sekä globaalissa kehyksessä verrattuna kaikkiin aikaisempiin kameran kuviin ja rekonstruoituihin 3D-maamerkkeihin. Lokalisaatio-ongelman suurentaminen lisäämällä tarpeen vakaille 3D-pistepilville asettaa lisärajoitteita systeemille: sekä lokaalin että globaalin kameran asennon yhtäpitävyys ja tarkkuus kasvattaa merkitystään kamera-avaruuden 2D-kuvapisteiden trianguloiminen 3D-avaruuteen asettaman keiraliteettirajoituksen vuoksi. Tämän lisäksi poikkeama kameroiden asennossa saa aikaan ei-toivottua kohinaa pistepintaan ja aiheuttaa kumulatiivisia vääristymiä 3D-pinnan muodossa.

Toteutettu 3D-lokalisaatio- ja rekonstruointijärjestelmä käyttää erilaisia SLAM *(Simultaneous Localization and Mapping)* -tekniikoita kameran lokalisointiin ja asennon havaitsemiseen sekä monipuolisia SfM *(Structure-from-Motion)* -algoritmeja todellisen maailman jäljittelemiseen virtuaalimaailmassa. Konsepteja reunalaskennasta ja mobiilirobotiikasta käytetään laskennan nopeuttamiseen ja tulosten visualisointiin. Korkealla tasolla systeemi koostuu kahdeksasta (8) vaiheesta: 2D-kuvapisteiden havaitseminen ja yhteensovitus, kameran lokalisaatio, 3D-maamerkkien triangulointi, langaton pistepilven lähetys, pistepilvien rakentaminen, Poisson 3D-pinnan rekonstruointi ja 3D-visualisointi.

Käytettyjä algoritmeja tutkittiin yksityiskohtaisesti ja niitä käsiteltiin sulautettujen järjestelmien ja muiden tehorajoitettujen laitteiden näkökulmasta. Tarkoituksenmukaisia optimointeja käytetään asiaankuuluvasti, ja systeemin suorituskykyä erinäisissä tilanteissa kvantifioidaan erilaisia suorituskykymittareita käyttäen.

Systeemin näytetään olevaan käyttökelpoinen oikean maailman sovelluksissa ja saatuja rekonstruointituloksia verrataan uusimpiin tekniikoihin. Systeemi on jaettu GitHubissa avoimena lähdekoodina MIT-lisenssillä.

Avainsanat Simultaneous Localization and Mapping, Structure-from-Motion, Sulautetut Järjestelmät

Tämän kandidaatintyön alkuperäisyys on tarkistettu Turnitin Originality Check palvelulla.

FOREWORDS

This Bachelor’s Thesis was made for the Tampere University’s Electrical Engineering study program during the autumn of 2021. The subject of the Thesis is camera localization and 3D surface reconstruction on low-power embedded devices. The subject is remarkably important, especially considering the recent advancements in the fields of robotics, computer vision, and other mobile applications.

I would like to thank the personnel in my current workplace, Ultra Video Group, for largely sparking my interest in real-time computer vision and its applications. I would also like to thank Erja Sipilä for her time overseeing the writing of my thesis, and my family for their continuous support over the years.

Tampere, 1.12.2021

Teo Niemirepo

TABLE OF CONTENTS

LIST OF IMAGES

LIST OF CODES

ABBREVIATIONS

2D	2-Dimensional
3D	3-Dimensional
AKAZE	Accelerated KAZE (tr. KAZE (風), Japanese: “wind”)
AR	Augmented Reality
BRIEF	Binary Robust Independent Elementary Features
DLT	Direct Linear Transformation
E-PnP	Efficient Perspective-n-Point
FAST	Features from Accelerated Segment Test
FLANN	Fast Library for Approximate Nearest Neighbors
fps	Frames-Per-Second
IMU	Inertial Measurement Unit
k-d tree	*k* -dimensional tree
LDSO	Direct Sparse Odometry with Loop Closure
LiDAR	Light Detection and Ranging
LSH	Locality Sensitive Hashing
MPL	Multi Probe Level
ORB	Oriented FAST and Rotated BRIEF
PnP	Perspective-n-Point
RANSAC	Random Sample Consensus
RGB-D	Red Green Blue – Depth
SfM	Structure from Motion
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization and Mapping
SoC	System on a Chip
SURF	Speeded Up Robust Features
SVD	Singular Value Decomposition
UVC	USB Video Class
XR	Extended Reality

1. INTRODUCTION

The problem of accurate, robust, affordable, and scalable localization in the realm of robotics and computer vision has been intensely researched since the late 1980s [1]. Various sensor fusion schemes [2] utilizing IMUs *(Inertial Measurement Units)*, LiDARs *(Light Detection and Ranging)*, multispectral cameras, and other sensors have been tried, but to this day, the one style of approach showing the most promise in future applications is solutions using pure RGB cameras and thus mimicking human understanding of the surrounding environment. The rise of neural networks and high-performance image processing techniques has accelerated the adoption of camera - based implementations even further [3] [4] [5].

The use cases of high-performance and accurate localization, both in the local and global scenario, are important in robotics applications. Other relevant use cases can be found, for example, in the realm of security and military applications, geo scanning, drones and other aviation technologies, and the entertainment sector.

3D localization also proves its usefulness in the mobile device software industry as well. Augmented Reality *(AR)* and Extended Reality *(XR)* have been on the forefront of technological progress and computer vision advancements in recent years [6], and most of the solutions involved in the implementations require the user's device to be localized very accurately.

Currently, one of the main areas-of-interest for localized XR applications can be found in the mobile world [7]. In these applications, the importance is rarely only on the localization aspect of the system, but the 3D reconstruction element as well. This introduces a plethora of constraints on the overall system: it must not only be accurate, but highly performant and moreover computationally efficient. High-performance implementations, such as the one introduced in this Thesis, will be at the leading edge of the future of XR. While these solutions are certainly important in the mobile world, the mobile world is hardly the only trade interested in it. Various XR technologies are used extensively, for example, in the automotive world and industrial applications as well [8].

Monocular RGB cameras offer some notable advancements over other sensoring devices, most evident being their relatively cheap price compared to the sensing density and the large number of opportunities in pattern detection and signal processing [9].

Camera systems are also highly scalable and might not even require any new hardware, considering how common cameras are in modern mobile electronics.

Merely localizing the camera in 3D space is usually not enough to be usable in more advanced robotics applications. For example, a dynamically updating and high-accuracy 3D map of the environment is usually required in biped or quadrupedal robotics, where updating the target position of the limb accurately to match the environment is vital to prevent the robot from falling over or causing an unwanted accident [10].

In recent years, the impact of cloud computing has shifted the emphasis of data processing to remote platforms, but with modern microprocessors the computations can be brough to the edge [11] or done on the embedded platform itself. On most cost-effective and mobile platforms doing all of the processing on the same device is not feasible due to power limitations. Offloading parts of the non-critical processing to a remote machine can often be advantageous, such as reconstructing the 3D map of the environment. This offloading approach also shares its usefulness in swarm robotics, where multiple mobile robots can share the same environmental map and contribute to its growth and accuracy [12].

This technical Thesis aims to explore one such hybrid implementation, in its core utilizing an RGB camera and a low-power embedded platform. The processing of the 3D surface and the visualization aspect of the system are offloaded to a remote device in order to increase the viability and runtime performance of the localization system. A practical implementation was chosen as the central topic due to its inherent importance in the ever-evolving and advancing technological world, particularly in the realm of mobile robotics and modern computer vision applications. The implementation introduced in this Thesis explores the difficulties and real-world optimizations used in the state-of-the-art academic and commercial applications.

The remainder of this Thesis is structured as follows. Section 2 investigates the previous most prominent academic and commercial solutions involving camera localization and 3D surface reconstruction. Section 3 provides an overview of the various algorithms used in 3D localization in general and in Section 4, where a practical implementation of embedded localization is detailed. Section 5 outlines the performance and reconstruction quality evaluations, and Section 6 gives the conclusions and explores grounds for further research.

2. RELATED WORK

Simultaneous localization and mapping *(SLAM)* techniques, alongside with structure-from-motion *(SfM)* methods are well known concepts and have been previously studied in depth. Arguably, between surface reconstruction and 3D localization, the more important and demanding obstacle is the accurate localization of the camera. With current techniques, it is easier to infer the 3D surface when the transformations of the cameras are known due to the inherent nature of 3D projection and triangulation. Additionally, as the subject of this Thesis is more closely related to 3D SLAM, various SLAM frameworks are of particular interest.

This chapter aims to examine earlier implementations and discuss their viability on low-power platforms and in real-time situations. Both open-source and closed-source solutions are explored in this chapter.

2.1 SLAM Solutions

The main objective of SLAM algorithms is to accurately localize the device-of-interest, both locally in the immediate vicinity of the device, and globally in a larger scale, beyond the field-of-view of the device [13]. In practice, this means the device should be able to localize itself in relation to a couple of previous frames and in relation to all of the previous frames. These two concepts are not necessarily mutually inclusive, as the localization error in relation to the previous frame can usually be considered negligible, if not zero, but without some form of global optimization, the position error often accumulates to a noticeable degree. For example, in the case of large-scale 3D localization, this can often be observed by the position error between the first and last camera frames to have the magnitude of multiple meters.

Some of the most prominent open-source SLAM frameworks are LSD-SLAM [14], Kimera [15], and various versions of ORB-SLAM, such as the most recent one at the time of writing, ORB-SLAM3 [16]. These are solutions designed mostly for high-power applications, where the available processing power is virtually not limited in comparison to embedded microprocessors. For example, ORB-SLAM3 recommends on their GitHub page [17] at least an intel core i7 -series processor for real-time applications. This is several orders of magnitude more demanding for what is feasible on an affordable embedded device.

2.2 Structure-from-Motion Solutions

The state-of-the-art SfM solutions often have the aim of producing extremely high-fidelity and textured 3D models. This apparent scope is often more restricting than the definition of structure-from-motion would require: *“The question addressed is how the 3-D structure and motion of objects can be inferred from the 2-D transformations of their projected images when no 3-D information is conveyed by the individual projections”* [18], and indeed SLAM frameworks could be classified under the term SfM. Nevertheless, SfM is nowadays often associated with primarily recovering the 3D data of a scene, even if the more correct interpretation would be to classify SfM as a toolbox of various algorithms and implementations.

Most of the high-fidelity SfM solutions fall under the umbrella term photogrammetry, the most prevalent and full-featured open-source solution being AliceVision MeshRoom [19]. One of the oldest and best-known closed-source solutions is Agisoft MetaShape [20].

Photogrammetry software do not run in real-time, often requiring hours or days to reconstruct even smaller scenes. Real-time SfM software is often associated with either augmented reality or advanced robotics applications [21]. Dynamic pathfinding in robotics and real-time 3D scanning of environments in AR applications are common examples of such. In current applications, the world-under-capture is often be assumed to be static or rigid, and the camera is the only object in motion.

2.3 RGB-D Based Approaches

The quality of the created 3D mesh and the reconstruction speed of the pipeline can be significantly improved by introducing dedicated depth data as additional input to the system using RGB-D (*Red Green Blue - Depth)* images. With the use of a stereo camera pair, or a purpose-made depth camera, depth data could be acquired easily and efficiently. While the current implementation introduced in this Thesis will not make use of RGB-D images, it may prove to be an interesting ground for future research. It is also worth noting, most of RGB-D solutions also make use of the same approaches as traditional SLAM and SfM applications, thus making them relevant in this scope [22].

Most of the solutions utilizing various depth sensing techniques are closed source or commercial solutions. These commonly use custom and expensive 3D scanners alongside with tailored software. For example, [23] and [24] are such solutions. The open-source space of 3D reconstruction is decidedly lacking, there being only a handful of worthy implementations. The primary open-source implementations utilizing RGB-D images are: VoxelHashing [25], BundleFusion [26], and Open3DGen [27].

3. ALGORITHM DESCRIPTIONS

There exist many different approaches to camera localization, depth estimation and 3D surface reconstruction. The use of an embedded platform constraints the usable algorithms considerably. As is often the case, elevated robustness and accuracy generally require more processing power. This means, some compromises must be made. Mainly, algorithms with low memory profile and good single-threaded execution speed should be emphasized. Embedded devices rarely are fortunate enough to have multiple cores. This proves to be problematic, as most of the algorithms introduced in this Thesis, such as feature detection, feature matching, and feature triangulation, are able to make heavy use of multi-core processors.

This chapter will explore in detail the various algorithms used in the practical implementation of this Thesis. Particular care is given to consideration of execution performance, whenever applicable.

3.1 Camera Calibration

Camera calibration comprises of obtaining the camera-specific intrinsic parameters, also known as the camera matrix, and the distortion coefficients. To simplify the math involved, the pinhole camera model [28] is used in the calculations. This is not realistic with real-world modern cameras, which use lenses and cause barrel or pincushion distortion, thus creating the need for undistorting the captured image. The result of undistortion is an image or set of 2D points, which satisfies the perspective camera model

$$\boldsymbol{u} = \boldsymbol{P}\boldsymbol{x}\,, \tag{1}$$

where the image 2D point is $\boldsymbol{u}$, the camera projection matrix is $\boldsymbol{P}$ and the 3D world point is $\boldsymbol{x}$. The projection matrix $\boldsymbol{P}$ is defined as

$$\boldsymbol{P} = \boldsymbol{k}[\boldsymbol{R}|\boldsymbol{t}]\,, \tag{2}$$

where $\boldsymbol{R}$ is the camera rotation matrix and $\boldsymbol{t}$ is the camera's translation vector [29].

The most common undistortion technique involves the use of either 5 or 7 undistortion coefficients, using only $5\ k_n$ radial distortion parameters or utilizing the additional $2\ p_n$

tangential distortion parameters as well. In this case, only the necessary radial distortion coefficients will be considered. The distortion coefficient vector is in the form of

$$\boldsymbol{d} = [k_0, k_1, k_2, k_3, k_4]. \tag{3}$$

The camera matrix is defined as $\boldsymbol{k}$, where f_x and f_y are the camera's horizontal and vertical focal lengths and c_x and c_y are the image's principal point, which usually coincides with the image's center point. The camera intrinsic matrix [29] is defined as

$$\boldsymbol{k} = \begin{matrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{matrix}. \tag{4}$$

One of the most common calibration algorithms is the one introduced by Zhegyou Zhang [30], which is also used in the implementation introduced in this Thesis, indirectly by the use of OpenCV [31]. This method requires a checkerboard image, usually printed on an A4 piece of paper, as a calibration reference. The checkerboard corners are detected, and the known relationship between the corners in 3D space is used to parametrize the 2D-3D point correspondences. The end products are the camera calibration parameters: the distortion coefficients and the intrinsic matrix [30].

To save processing time, the entire RGB images are usually not undistorted. Instead, the undistortion can only be done on the detected 2D feature points. The benefit of this is evident, when the number of points-of-interest are considered in each case: a 1920×1080 RGB image has 2073600 points to undistort, whereas the length of feature point vectors is usually measured in the thousands. The latter is multiple orders of magnitude computationally lighter.

3.2 2D Image Feature Detection

All SLAM and SfM applications fundamentally rely on the ability to detect robust and consistent features in images. These image features must be reproducible, temporally and spatially consistent, and efficient to compute. The feature detection algorithm must also be able to individually describe every feature to a distinctive-enough degree, where the same real-world feature can be detected and distinguished when viewed through another viewpoint, even if the camera view is rotated and thus the orientation of the feature is not consistent throughout the frames [32].

There are two major types of feature descriptors: binary descriptors and floating-point descriptors. In general, floating-point descriptors, such as SIFT [33] and SURF [34] are slower and not as memory efficient but considerably more robust and temporally more stable. For years, SIFT and SURF were the de-facto standard in many high-quality computer vision and SfM applications [35].

In modern solutions, binary feature descriptors, such as AKAZE [36] *(Accelerated KAZE)* and ORB [37] *(Oriented FAST (Features from Accelerated Segment Test) and Rotated BRIEF (Binary Robust Independent Elementary Features)),* have seen an increase in their use due to the advent of more intelligent matching and filtering algorithms.

Binary features are uniquely suited for use in low-power platforms, where processing power and the amount of available system memory is limited. The ORB feature detection algorithm was chosen to be used in this implementation due to its superior speed and memory requirements.

The ORB feature detection algorithm is one of the fastest and more robust detection algorithms used [37]. It is scale and rotation invariant, making it optimal for camera localization. By the nature of ORB being a binary descriptor, it is also memory efficient [38]. This is more important on an embedded device, where the amount of system memory is limited to begin with.

3.3 2D Feature Matching and Filtering

Feature matching algorithms find correspondences between two sets of feature descriptors. These algorithms compare the descriptors, and using various metrics try to find pairs of features that are the most similar [39].

The brute force matcher [40] is a greedy algorithm, which compares all pairs of individual features. While this may result in more matches, it is also computationally heavier and more prone to false positives. A common way of improving the quality of the matches is to cross-check the feature correspondences and verify, that both features match each other in their opposite feature sets.

The FLANN *(Fast Library for Approximate Nearest Neighbors)* feature matching algorithm [41] finds the approximate nearest neighbor feature matches using a *k*-d tree [42]. Compared to the brute force matcher, the results are not as accurate, but considerably faster with large datasets. The effect of jitter and inconsistencies in the feature matches is mitigated to a degree with the use of RANSAC *(Random Sample Consensus)* [43] in later stages of the localization pipeline. The performance and accuracy of the FLANN matcher can be changed easily with the multi-probe locality-sensitive hashing *(LSH)* index [44]. A higher index value will yield better and more robust results with the expense of drastically reduced performance. The performance implications of FLANN and brute force matching is discussed in more detail in chapter 5.

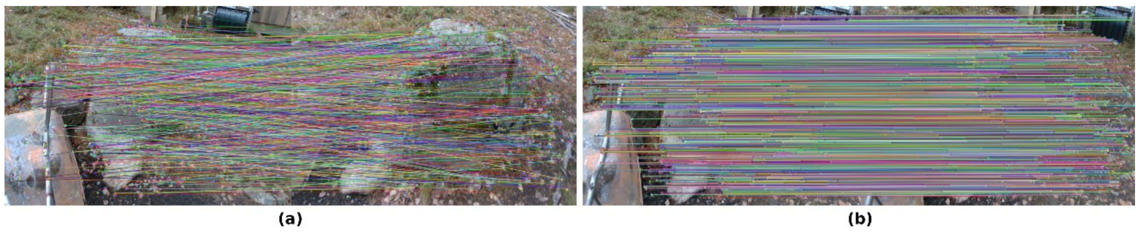

Figure 1 *– Feature matches between two images visualized. (a) Invalid matches. (b) Good quality matches.*

The feature matching algorithms are not perfect, and thus require the feature matches to be filtered for outliers. These outliers most commonly exhibit themselves as drastically invalid feature matches, as shown in Figure 1. Experimentally, FLANN matchers are usually slightly more robust in automatically only keeping the inlier feature matches, but often still require additional filtering due to the approximate nature of the algorithm.

In nearly all feature matching cases, a distance ratio check is done on the matched features. This filters out the majority of bad matches, but still leaves some for further filtering. For floating point descriptors, the distance metric often used is the Euclidean distance, and for binary descriptors Hamming distance is used. Only the feature matches with the distance less than a pre-specified threshold are kept.

In real-time SLAM applications, it can often be assumed the movement between subsequent frames is very small, and thus the coordinates of the matched features should lie very close to each other in image coordinates. This special case allows for the use of geometric filtering. For example, the distance between the matched feature coordinates can be calculated using Pythagoras theorem and matches which have a distance beyond a specified threshold can be removed.

Another form of geometric filtering that can be used in this special case is homographic filtering. A homography matrix between the two frames can be computed from the feature points of the frames. The homography algorithm often makes use of RANSAC, thus mitigating the effect of outliers. After acquiring the homography matrix, the feature points of the first frame can be transformed into the space of the second frame. In the ideal case, the two sets of points are now identical. In the real world, these two sets of points are now very close to each other. By again specifying a threshold, these pointsets can be compared, and only points within the specified Euclidean distance are kept. This form of filtering is by far the more robust between the alternatives, but comes with the disadvantage of filtering out points, which do not fulfill the homography constraint. In practice, this means points that do not lie on a plane are removed. In urban environments, where flat surfaces are common, homography filtering often yields good enough results. The implementation for filtering feature matches with the homography matrix used in this Thesis, which extends on top of OpenCV [31], is shown in Code 1. [45].

```
std::vector<std::pair<uint32_t, uint32_t>> feature_matches;
std::vector<cv::Point2f> feature_points_1, feature_points_2;
/** ... */

// compute the homography matrix
const cv::Mat homography = findHomography(feature_points_1,
    feature_points_2,  cv::RANSAC, HOMOGRAPHY_RANSAC_THRESHOLD);

// create a vector to hold the good matches
std::vector<std::pair<uint32_t, uint32_t>> good_matches;

// loop through the feature matches and check for homography constraint
for (size_t ii = 0; ii < feature_matches.size(); ii++)
{
    // transform the 1st point to the space of the 2nd frame
    cv::Mat col = cv::Mat::ones(3, 1, CV_64F);
    col.at<double>(0) = feature_points_1[ii].x;
    col.at<double>(1) = feature_points_1[ii].y;
    col = homography * col;
    // homogeneous coordinate transform
    col /= col.at<double>(2);

    // calculate the Euclidean distance between the points
    const double dist =
        sqrt(pow(col.at<double>(0) - feature_points_2[ii].x, 2)
        + pow(col.at<double>(1) - feature_points_2[ii].y, 2));

    // check for distance threshold
    if (dist < HOMOGRAPHY_FILTER_MAX_DIST)
        good_matches.push_back(matches[ii]);
}
```

***Code 1** – Feature match homography filtering, as taken from the code presented in this Thesis.*

3.4 Feature Triangulation

The main method for triangulating 3D points from 2D correspondences and the respective camera projection matrices uses DLT *(direct linear transformation)* [46] and SVD *(singular value decomposition)* [47]. This triangulation approach is resilient against noise in the 2D position of the feature points and extends efficiently into multiview cases, where more than two feature correspondences are used. Augmenting the triangulation with additional camera viewpoints reduces the effect of jitter or inaccuracies caused by noise in the 2D feature positions while only being marginally slower, compared to only using two viewpoints. The DLT algorithm finds the least squares -optimal 3D point and minimizes the reprojection error [48].

According to the perspective camera model, the relationship of a 3D point $\boldsymbol{x}$ in world-space coordinates and its 2D projection in camera-space coordinates $\boldsymbol{u}$ is

$$\boldsymbol{u} = \boldsymbol{P}\boldsymbol{x}. \tag{5}$$

Let the same 3D point $\boldsymbol{x}$ be

$$\boldsymbol{u}' = \boldsymbol{P}'\boldsymbol{x} \tag{6}$$

in the coordinates frame of another camera. These equations can be represented as

$$\boldsymbol{u} \times \boldsymbol{P}\boldsymbol{x} = 0, \tag{7}$$

which can be expanded to

$$\begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix} \times \begin{bmatrix} \boldsymbol{p}^{1T} \\ \boldsymbol{p}^{2T} \\ \boldsymbol{p}^{3T} \end{bmatrix} \boldsymbol{x} = 0. \tag{8}$$

To recover the 3D point $\boldsymbol{x}$, the equations representing the two different views can be combined into the general form of

$$\boldsymbol{A}\boldsymbol{x} = 0, \tag{9}$$

where, in the case of two views, $\boldsymbol{A}$ is represented as

$$\boldsymbol{A} = \begin{bmatrix} u_x\boldsymbol{p}^{3T} - \boldsymbol{p}^{1T} \\ u_y\boldsymbol{p}^{3T} - \boldsymbol{p}^{2T} \\ u'_x\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{1T} \\ u'_y\boldsymbol{p}'^{3T} - \boldsymbol{p}'^{2T} \end{bmatrix}. \tag{10}$$

The result is a homogeneous system of linear equations, which can be solved with singular value decomposition [48]. The code which implements this functionality can be seen in Code 2. This code is also capable of triangulating multiview features from different camera viewpoints, although it is worth noting this degrades the SVD performance drastically. In most cases, the result of two-view triangulation is robust enough to be considered adequate.

```
Eigen::Vector3d triangulate_multiview(
    const std::vector<Eigen::Vector2d>& feature_points,
    const std::vector<Mat34>& projection_matrices)
{
    // create the A matrix used in solving the SVD
    Eigen::Matrix4d A = Eigen::Matrix4d::Zero();

    // loop through the feature points, and add them to the A -matrix
    for (size_t ii = 0; ii < feature_points.size(); ii++)
    {
        const Vector3d point = feature_points[ii].homogeneous().normalized();
        const Mat34 term =
            projection_matrices[ii] - point * point.transpose()
            * projection_matrices[ii];

        A += term.transpose() * term;
    }

    Eigen::SelfAdjointEigenSolver<Eigen::Matrix4d> eigen_solver(A);

    // acquire the eigen vectors and take the first one, which most closely
    // resembles the true-to-life 3D point, return the triangulated 3D point
    return eigen_solver.eigenvectors().col(0).hnormalized();
}
```

***Code 2** – The code implementation of the triangulation algorithm used in this Thesis.*

3.5 Essential Matrix Decomposition

By decomposing the essential matrix [49], the rotation and relative translation between two sets of feature correspondences can be recovered by using the properties of epipolar geometry [50]. The translation between the frames is of unit length and merely indicates the direction of the movement.

The essential matrix is formulated from the translation vector t and the rotation matrix R as such:

$$\mathbf{E} = \mathbf{R} \times \mathbf{t} \tag{11}$$

In this case, the stated problem is the inverse: the translation and rotation of the camera are unknown, but it is generally possible to obtain the essential matrix from only the feature correspondences. Common algorithms for achieving this are the five-point algorithm [51], which is also used in OpenCV, or the seven-point algorithm [52]. While there exist other methods as well, these are one of the more usual ones.

After obtaining the essential matrix, it must be decomposed, and the relative pose must be extracted from it. This is done using singular value decomposition. The implementation in this Thesis uses the solution given by OpenCV. After decomposing the essential matrix, the result is four possible poses, of which only one fulfills the cheirality constraint, i.e., the pose results in positive and non-infinite triangulation. These four cases must be handled individually, and the solution which results in front-side triangulation is picked.

The unit-length translation -characteristic makes essential matrix decomposition only suitable for inferring the initial camera conditions of the first two frames in the system. Without supplementary sensor data, such as accelerometer data acquired from an IMU, or some other method for setting the scale, the first two frames must be used to set to unit scaling. While there are intelligent ways to infer realistic scaling from a set of camera views, for the sake of simplicity those will not be considered.

3.6 Perspective-n-Point

For the consecutive frames after the first two, the PnP *(Perspective-n-Point)* algorithm [53] is used to recover the pose of the camera. There are multiple different implementations of the basic PnP algorithm, the one preferred in modern SfM and multiview geometry solutions is the Efficient PnP *(E-PnP)* [54] algorithm. By design, the E-PnP algorithm has the complexity of *O(n)*, making it usable on low-power platforms and real-time applications.

Most PnP algorithms are not inherently robust when given high-noise feature points or false feature correspondences as input. Thus, RANSAC is often used in conjunction with various feature filtering mechanisms to reduce the effect of outliers and produce good quality camera poses.

3.7 Loop Closure and Bundle Adjustment

When RGB cameras are used, it is often necessary to solve for loop closure when robust global localization is required. For example, in the case of SfM solutions, non-rigid space deformation [55] is a valid method of optimizing the camera locations and the generated 3D map of the world. Traditionally with RGB SLAM, gradient decent and non-linear minimization has been used, in the form of bundle adjustment [56]. While accurate, bundle adjustment is also extremely slow and mostly unusable if the number of points is large. While there have been faster and more optimized algorithms, such as LDSO [57] *(Direct Sparse Odometry with Loop Closure)*, it is not realistically possible to use in real-time and low-power applications.

In the case of the solution introduced in this Thesis, loop closure and bundle adjustment are skipped for the sake of simplicity and to save processing power. It is also worth noting, the cumulative error can rarely develop to unmanageable levels when the environment is small, for example a small room, and the old 3D key points can be reused.

3.8 Poisson Surface Reconstruction

The Poisson surface reconstruction algorithm [58] generates smooth and watertight 3D surfaces from discrete point clouds. The Poisson surface reconstruction algorithm computes an approximation of the surface. The accuracy to which the surface is computed can be set parametrically. The Poisson surface algorithm uses an octree for capturing the 3D detail.

The Poisson surface reconstruction algorithm was chosen over the alternatives, such as the ball-pivoting algorithm [59], because it generates watertight and envelope 3D surfaces. It also produces exemplary results with sparse input point clouds. Many of the alternatives require a uniform point surface in order to produce adequate results. The implementation of the Poisson algorithm was provided by the intel Open3D library [60].

4. IMPLEMENTATION

The embedded development platform used in the implementation is the BeagleBone Pocket Beagle [61], utilizing an Octavo Systems OSD3358-SM SoC *(system on a chip)* with a 1 GHz ARM Cortex-A8 high-power core, an ARM Cortex-M3 low-power core and 512 MB of RAM. Wi-Fi connectivity is added to the system with a USB add-on. Similarly, a USB camera is used for the real-time image acquisition. The camera used is the Arducam IMX477 12MP “Raspberry Pi HQ Camera” with an additional USB conversion board attached [62]. The camera is used with the resolution of 1280×720 and theoretical framerate of $100\ fps$. The experimental setup is shown in Figure 2.

The second principal part of the system is the high-power remote device, which is used for point cloud structuration, 3D-surface reconstruction and all visualization functionality. In the case of all results demonstrated in this Thesis, the remote device utilizes an AMD Ryzen 5900X processor with 64 GB of RAM. The code for this project can be found under the MIT open-source license at github.com/teo3n/BScEmbeddedLocalization.

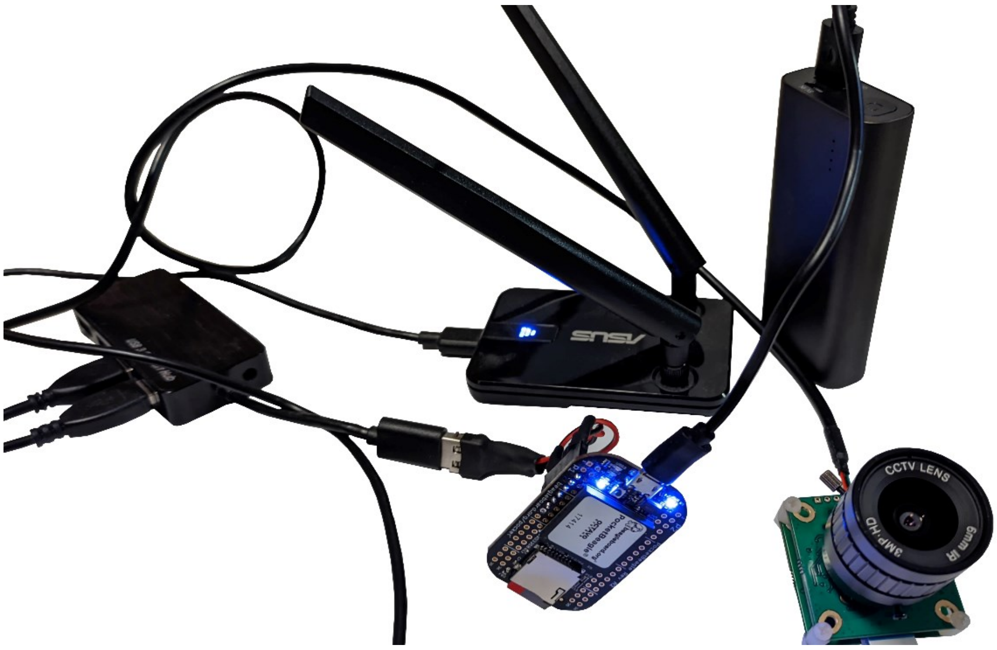

Figure 2 *–The embedded platform with a USB hub, the camera, a WiFi module, and a battery pack connected.*

4.1 Overview

On a high level, the architecture of the entire system can be divided into three main components: the localization component, the point cloud stream component, and the visualization component. From these, only the localization component is mandatory for the functioning of the system. The overview of the system architecture can be seen in Figure 3. The division of the different processing parts into multiple platforms was mandatory, in order to achieve reasonable performance on the embedded system.

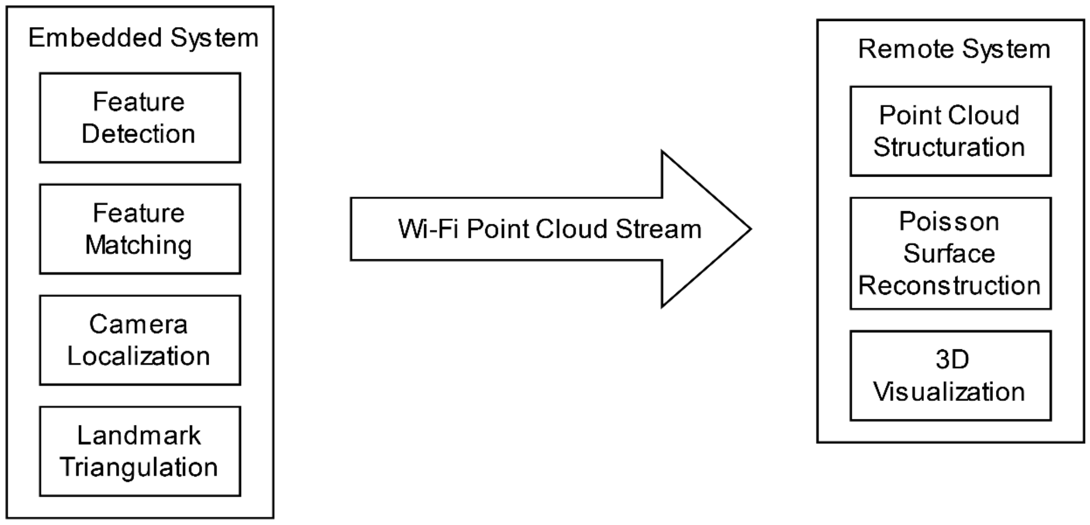

***Figure 3** – A high-level overview of the architecture of the system*

In addition to image acquisition, feature detection, matching, camera localization, and landmark triangulation is all done on the embedded system. Separating these into further parts and offloading the processing to the remote platform does not make sense from a performance perspective, as the camera localization and landmark triangulation steps use an insignificant amount of processing power. Additionally, offloading feature detection and matching to the remote device would require large amounts of data to be transmitted, if the image data is not compressed. In the case good-quality compressions would be used, the performance saved in feature detection and matching would be lost on the compression part, making the tradeoff often not feasible.

4.2 Image Acquisition

The first algorithmic stage of the reconstruction pipeline is the image acquisition stage. In Figure 4, a set of camera frames from the evaluation dataset are visualized. The testing and evaluation dataset is 248 frames long, and the frames have the resolution of

1280×720. The dataset is captured outdoors in a setting, where there are a lot of good-quality feature points to track.

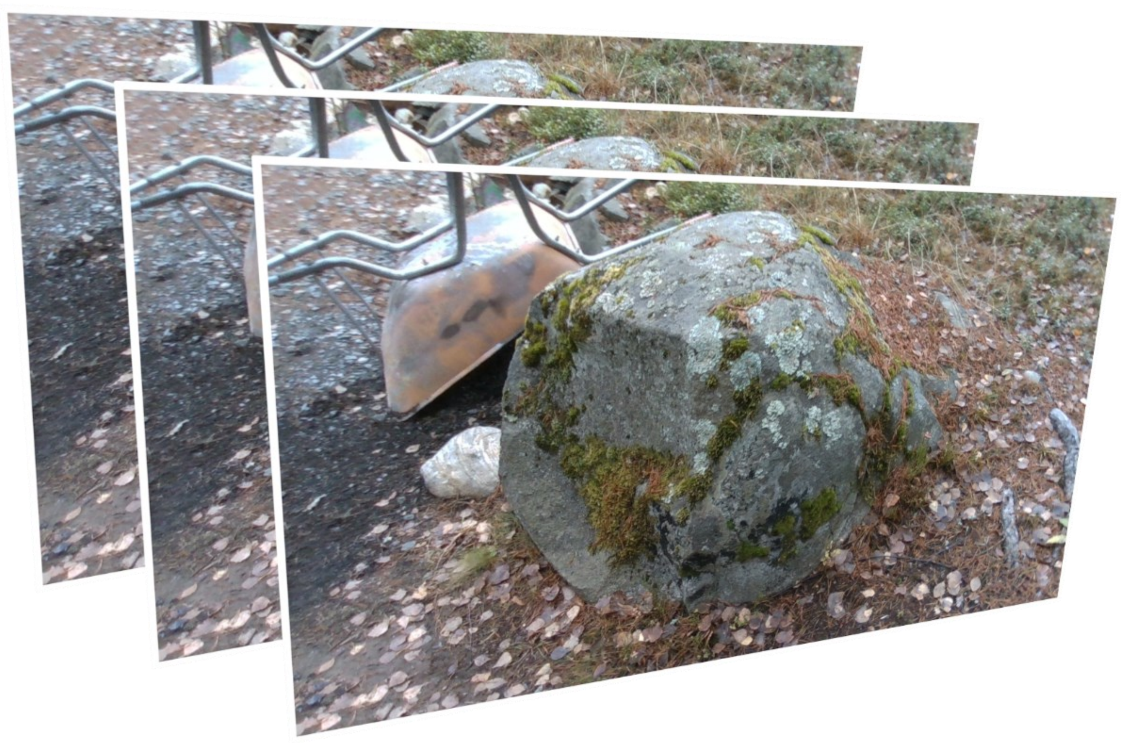

Figure 4 *– A subset from the dataset used in the evaluation of the implementation.*

When run in real-time, the system uses the connected RGB camera to acquire the frames. While the implemented system theoretically supports any UVC *(USB Video Class)* compatible camera, there are some important considerations that must be taken into account. For example, a traditional low-cost webcam with relatively low framerate ($15 - 30$ *frames-per-second*) often exhibits large amounts of motion blur, making the frames acquired often useless. Instead, a high-quality and high-performance camera was used in a high framerate mode, even though the system is not capable of fully utilizing the hardware available.

4.3 Camera Localization

The algorithm used in localizing the camera is dependent on how many frames are already localized. For the first two frames, the algorithm used decomposes the essential matrix, and for the rest of the frames PnP is used. On the grounds of reducing difficult to reproduce issues, selecting the first two frames is done manually. In practice, this means the user of the system must specify two frames, which are far enough apart from each other to achieve good triangulation, but still close enough to find a large number of robust feature matches. There are automated algorithms to evaluate the quality of the initial condition [63], but due to runtime performance constraints, these are not considered in this Thesis.

After the new frame has been localized, new landmarks can be created from the feature correspondences between the two most recent frames. It is important to note, the transformation between the two most recent frames is not guaranteed to be large enough for a robust triangulation. Instead, the detected 2D feature is added to a track of features, and the track is triangulated only after enough movement between the frames has been detected. This track has the additional benefit of resulting in more accurate 3D triangulations, due to additional data points through the use of multiview triangulation.

The camera projection matrix is defined as

$$\boldsymbol{P} = \boldsymbol{k}\left[\boldsymbol{R}^{\mathrm{T}} \middle| \left(-\boldsymbol{R}^{\mathrm{T}}\boldsymbol{t}\right)\right], \tag{12}$$

which differs slightly from the equation given in chapter 3.1. This is due to the difference in the global-local reference frame. A track of localized cameras with the length of 200 frames, along with a triangulated point cloud, is shown in Figure 5. The choosing of the first two frames for the initial localization using essential matrix decomposition can clearly be seen in the top left corner of the figure.

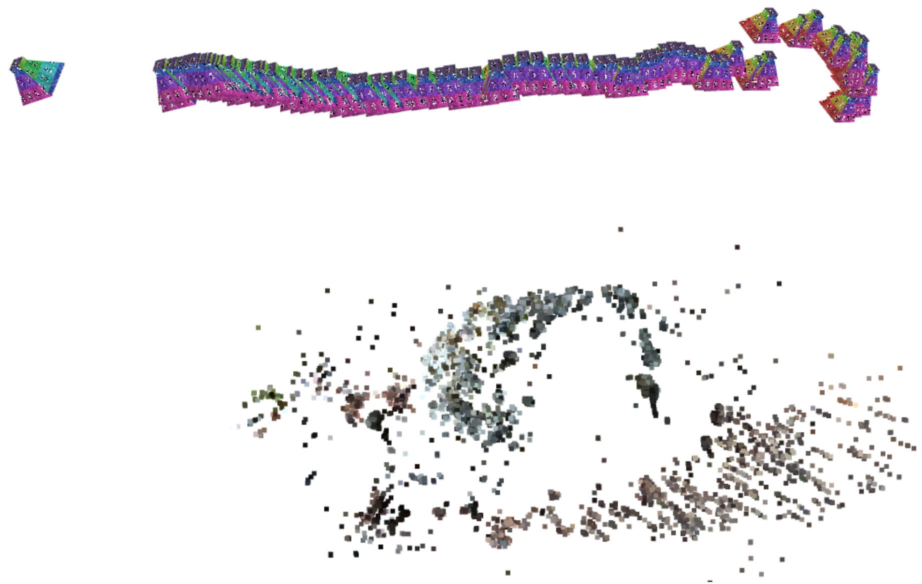
Figure 5 *– The localized camera track and the reconstructed point cloud.*

4.4 Dense 3D Point Data Generation

The point cloud generated by triangulating heavily filtered feature matches is sparse and only contains points from regions rich with distinctive feature areas. In applications, where high-quality 3D surface is not required, this may be enough, but in most other cases a denser cloud is necessary. Without the use dedicated depth data, the easiest method for obtaining less-sparse 3D data is by the use of dense depth projection [64]. Effectively, this is dense feature triangulation done on the entire image.

The dense feature triangulation is a performance intensive process and prone to noise artifacting due to how disparity maps are calculated. The non-stereo configuration of the cameras in world space reduces the feasibility of this even further. In practice, this means the camera frames are not located parallel to each other and with a known distance between them. More advanced and robust methods are not doable on low-power platforms due to their processing and memory requirements.

A more feasible approach is to still triangulate feature matches, but more densely and without extensive filtering. As these points are not used in localization, the accuracy and robustness are not critical. In essence, the workflow is the same as in normal feature matching and triangulation, but in this case the maximum number of feature matches is increased. This induces a significant loss of performance, and is thus done rarely, for example, only every $20 - 40$ frames (frame delta). It is also worth noting, dispatching this dense projection every frame would not improve the quality or density of the final point cloud significantly due to the amount of overlap in the consequent frames, and thus, the triangulated points of consequent frames. The difference in point contributions is shown in Figure 6 with the feature projection densities of $n = 0$, $n = 10000$ and $n = 100000$ respectively.

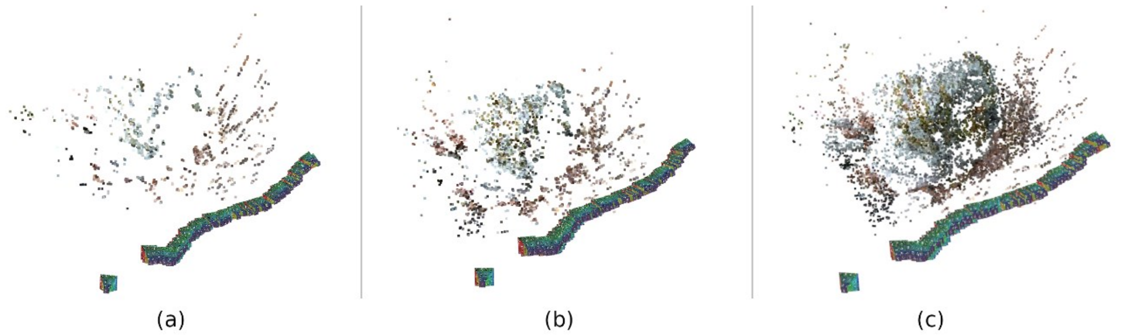

Figure 6 *– Comparisons of different sparsity point clouds. (a) Sparse filtered feature matches (n=1500). (b) Dense points with frame delta of 30 and n=10000. (c) Dense points with frame delta of 30 and n=100000.*

4.5 Wireless 3D Data Stream

The embedded platform is not powerful enough to generate high-quality 3D meshes in real-time, in addition to handling the camera localization. This introduces the need to offload some key processing to a remote platform, in this case a computer in the same local network. The data is transmitted wirelessly over Wi-Fi using WebSockets and TCP/IP utilizing the Boost Asio C++ library [65]. This approach makes it possible to use the remote system even from far away, given the embedded platform has an active internet connection.

To faithfully reproduce the 3D geometry, the camera locations and the triangulated points are transferred uncompressed. With higher density points, or if more processing power was available, the point clouds could be compressed or downsampled to reduce the required network bandwidth. In this case, it is not mandatory due to the low camera localization frequency and thus the low wireless data throughput.

4.6 3D Reconstruction

The 3D mesh reconstruction phase can be divided into two stages: point cloud structuration from the wireless data stream and 3D mesh reconstruction using the Poisson surface algorithm. The point cloud structuration step synchronizes a localized camera to the corresponding point stream.

From this restructured point cloud, a 3D mesh is generated using the Poisson algorithm. Due to the sparse nature of the point cloud, the mesh reconstruction uses a low value for the *k*-d octree. The reconstructed mesh is therefore not of high quality, but easily usable in mobile robotics applications.

Figure 7 shows the reconstructed mesh, along with a camera track. The mesh uses vertex colors for additional clarity. As can be seen from the figure, the mesh has a large "envelope" around it. This is a distinct characteristic of the Poisson surface reconstruction algorithm and produces advantageous results with the input point cloud at hand.

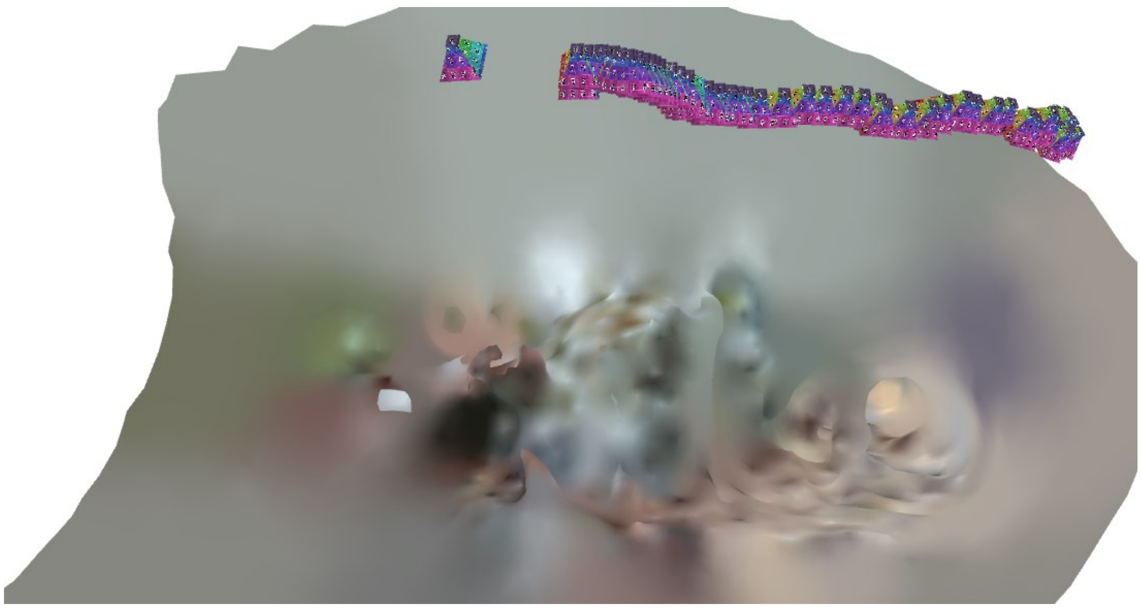

Figure 7 *– The reconstructed 3D mesh with vertex colors, generated from the combination of the sparse feature-match point cloud and the dense projection cloud.*

4.7 Remote Visualization

The final reconstructed 3D mesh along with the camera track is visualized on the remote device. A view from the remote visualizer can be seen in Figure 7. As with the rest of the system, the visualizer is written in modern C++.

In the same way as the point cloud streaming component running on the embedded device, the receiver uses the Boost Asio library for all networking functionality. The 3D rendering is done using the intel Open3D library [60]. The data is sent as a continuous floating-point buffer and the streaming -side implementation is shown in Code 3.

```
const std::vector<Eigen::Vector3d>& points, colors;
const std::shared_ptr<Frame> frame;
/* ... */

// stream buffer: frame pos + frame rot (mat3x3) + pts len + colors len
std::vector<float> data_buffer;
data_buffer.reserve(3 + 3 * 3 + points.size() * 3 + colors.size() * 3);

// loop through the 3D points and add their xyz values to the buffer
for (int ii = 0; ii < points.size(); ii++)
    for (int kk = 0; kk < 3; kk++)
        data_buffer.push_back(points[ii](kk));
// loop through the point colors and add their rgb values to the buffer
for (int ii = 0; ii < colors.size(); ii++)
    for (int kk = 0; kk < 3; kk++)
        data_buffer.push_back(colors[ii](kk));

// add the camera frame position (xyz) values to the buffer
for (int ii = 0; ii < 3; ii++)
    data_buffer.push_back(frame->position(ii));
// add the camera frame rotation matrix (3x3) to the buffer
for (int xx = 0; xx < 3; xx++)
    for (int yy = 0; yy < 3; yy++)
        data_buffer.push_back(frame->rotation(xx, yy));

auto send_buffer = boost::asio::buffer(data_buffer);
const uint32_t send_bytes = send_buffer.size();

// transmit a 4 byte stream containing the length of the data stream
boost::asio::write(*stream_handle.stream_socket,
    boost::asio::buffer({ send_bytes }), boost::asio::transfer_exactly(4));
// transmit the actual data
boost::asio::write(*stream_handle.stream_socket, send_buffer,
    boost::asio::transfer_exactly(send_buffer.size()));
```

***Code 3** – The code used in the implementation of the 3D data stream.*

5. PERFORMANCE AND QUALITY ANALYSIS

Evaluating the performance of the system is a vital part in classifying its capabilities and use cases. The main three types of performance metrics used in appraising the implementation are: runtime performance, localization accuracy, and reconstruction quality.

The aims of this chapter are to gauge the viability of the implemented system in real-world applications and to evaluate the reconstruction quality of the system. The rest of this chapter is as follows. In chapter 5.1 the runtime performance with different quality settings of the system is evaluated and compared against a high-power computer. In chapter 5.2 the camera localization accuracy of the system is evaluated approximately. In chapter 5.3 the quality of the 3D point clouds is compared against other state-of-the-art solutions.

5.1 Performance Evaluation

The runtime performance of the system was evaluated both on the BeagleBone embedded platform and on a high-performance desktop computer with an AMD Ryzen 5900X 12 core processor. The system reconstructed a scene with the length of 180 frames and the frame times were recorded. The variables changed between runs were the number of detected ORB features, either 3500 or 10000, the feature matching algorithm, either FLANN or a brute force matcher, and the FLANN multi probe level - variable, either 2, 1 or 0. The average frame times for all runs can be seen in . The best tradeoff between reconstruction quality and execution performance is to use 3500 ORB features and the FLANN feature matcher with multi probe level *(MPL)* of 1.

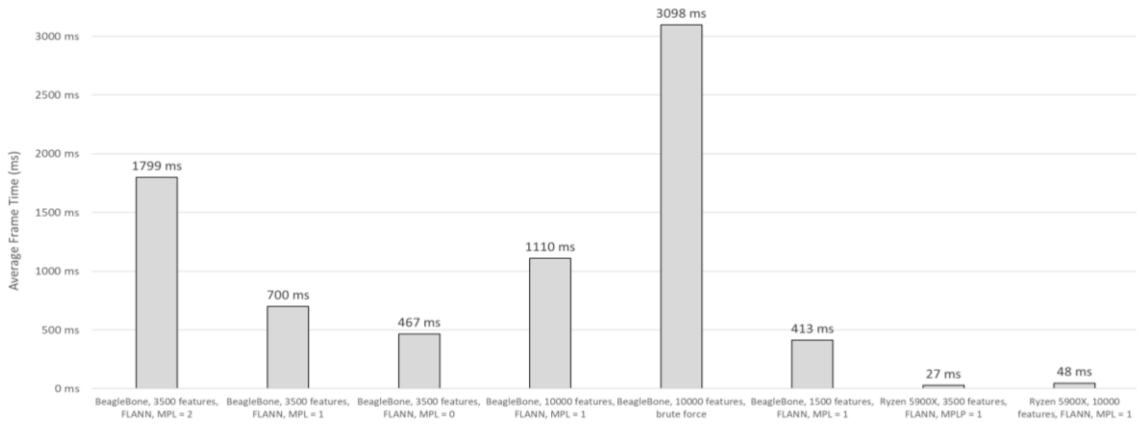

***Figure 8** – Performance comparisons between the embedded platform and a high-end desktop computer. The number of detected ORB features, the matching algorithm and the FLANN MPL were varied.*

As can be seen from Figure 8, the system cannot be considered to be “real-time” on the embedded device, as real-time speed would require a framerate of $24\,fps$ or $30\,fps,$ which would correspond to 42 ms or 33 ms frame times respectively [66]. This is not necessarily to the detriment of the system, as $1.4\,fps$ when using FLANN with MPL of 1 is still very good, considering the severe limitations of the platform. While this performance is not enough in XR applications, in robotics and mapping use cases it would nevertheless often be usable. With a more modern SoC, such as a Snapdragon 865 mobile SoC, achieving real-time performance would be possible, as can be seen from the vast number of real-time computer vision and augmented reality applications that exist in the current mobile device ecosystem. The system achieved very respectable performance when run on the high-end desktop processor.

In Figure 9 the embedded localization per-stage frame times are shown. The metrics were captured with the number of detected features being $n = 3500$ and by using the FLANN feature matcher with multi probe level $MPL = 1$. The creation of new landmarks and dense point projections $(n = 10000)$ were not done every frame. Instead, new landmark creation was run once every eight (8) frames and dense point projection was done twice throughout the entire sequence of 180 frames. The feature detection phase contributes by far the most to the average frame time. Exploring possible optimizations for the ORB algorithm is not within the scope of this Thesis.

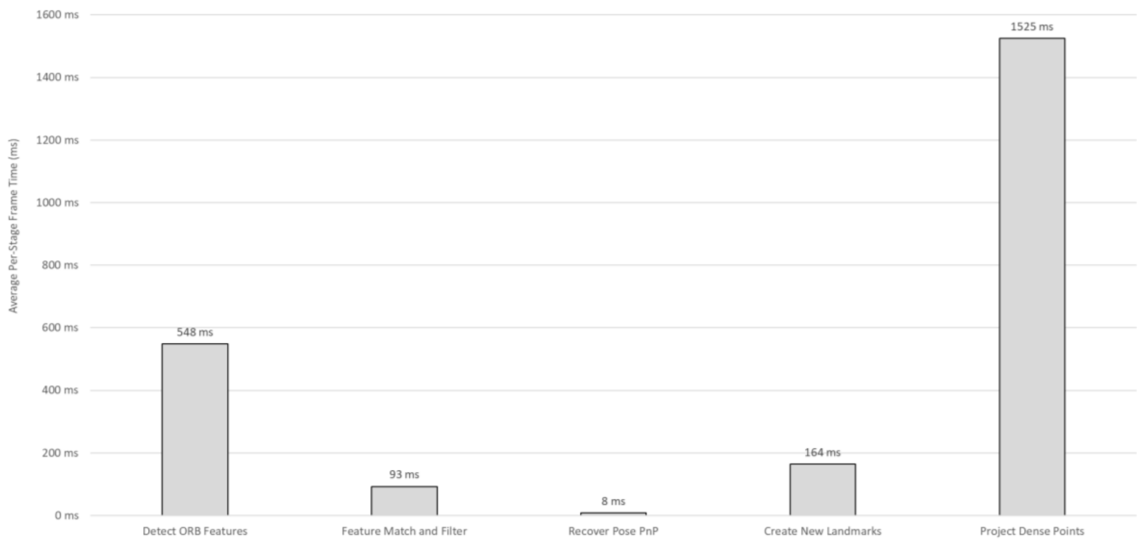

Figure 9 *– The per-stage average frame times (n=3500, FLANN, MPL = 1). Note, "Create New Landmarks" and "Project Dense Points" were not done every frame.*

5.2 Localization Accuracy Evaluation

Due to the inherent flaw of not implementing bundle adjustment, the resulting camera poses will always drift over time. As it stands, the resulting camera track is reasonably accurate in the immediate vicinity of the area-of-interest, i.e., in the local space. The cumulative positional error caused by the spatial drift makes this system more difficult to use in large-scale applications, or in applications, where absolute accuracy is required.

The implemented system is still not without its uses. It is still highly usable in small-scale positional applications, for example in the case of in aforementioned in Chapter 1 quadrupedal robotics feet positioning, or in the case of applications, where the device does not need to leave the area-of-interest.

5.3 3D Data Quality Analysis

Evaluating the 3D reconstruction quality of the system is decidedly difficult due to the sparsity of the point data. Instead, the 3D point clouds were compared against two different style of 3D reconstruction software: a high-quality photogrammetry software called AliceVision MeshRoom [19] and a state-of-the-art and real-time RGB-D 3D reconstruction software Open3DGen [27]. All solutions used the same set of RGB images, with Open3DGen receiving the additional depth data as its input. As Open3DGen uses true-to-life dense depth data, its output can be assumed to be as close to reality as possible.

As can be seen from the comparisons in Figure 10, the surface accuracy of the Thesis implementation is good. It is important to note, all the results shown only contain the point clouds, the use of extremely dense depth data in the case of Open3DGen merely makes the result look like a triangle surface. The overall sparsity of the feature points to the side of the rock, inherent to how the ORB feature detection algorithm functions, is the largest difference in quality. No other evident disadvantages can be observed.

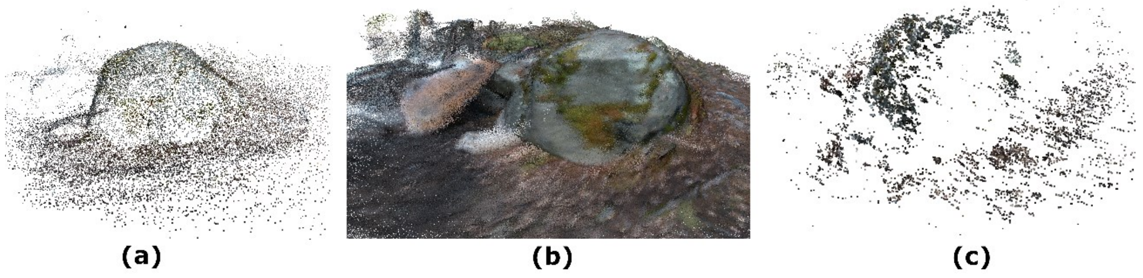

***Figure 10** – The quality of the system compared against state-of-the-art related work. (a) AliceVision MeshRoom. (b) Open3DGen. (c) Current Thesis Implementation.*

6. CONCLUSIONS

In this Thesis, the opportunities and use cases of accurate localization on embedded devices, augmented reality applications and mobile robotics were considered. Additionally, a system for localizing a camera in 3D space was implemented for a low-power embedded system. The 3D localization system incorporates an almost full structure-from-motion pipeline and a multitude of principles from simultaneous localization and mapping. In addition, a remote 3D surface reconstructing and visualization software was created for easy evaluation of the camera track and the reconstructed 3D data.

The runtime execution performance and the quality of the 3D data produced by the system were evaluated. When run on the embedded evaluation platform, the runtime performance was shown to be usable in basic robotics and mapping applications, where $1-2\ \mathrm{Hz}$ reconstruction frequency is sufficient, and the system does not need to adhere to a real-time constraint. With a more modern SoC the system could be drastically more viable in real-world applications. Further optimizations are unlikely to improve the performance of the system to a noticeable degree, unless a more efficient feature detection algorithm is developed.

Despite the sparsity of the reconstructed data, the accuracy of the 3D point cloud when reconstructed using short RGB sequences was proved to be comparable to the state-of-the-art monocular and depth-workflow 3D reconstruction software, despite the lack of a separate bundle adjustment stage. The distinct lack of processing power was the main limiting factor in considering the density of the reconstructed point cloud, and thus the estimated 3D surface.

By utilizing a dedicated RGB-D camera and by implementing bundle adjustment and loop closure detection the largest disadvantages in quality and accuracy of the system could be remedied. Implementing these may prove to be interesting grounds for future work.

7. REFERENCES

[1] R. C. Smith, M. Self, P. Cheeseman, "On the representation and estimation of spatial uncertainty", The International Journal of Robotics Research, Vol. 5, Iss. 4, 1986, pp. 56-68.

[2] X. Zhang, A. B. Rad, Y. Wong, Y. Liu, X. Ren, "Sensor fusion for SLAM based on information theory", Journal of Intelligent & Robotic Systems, Vol. 59, 2010, pp. 241-267.

[3] R. Aarthi, S.Harini, "A Survey of Deep Convolutional Neural Network Applications in Image Processing", 2018, International Journal of Pure and Applied Mathematics, Vol. 118, No. 7.

[4] C. Morikawa, M. Kobayashi, M. Satoh, Y. Kuroda, T. Inomata, H. Matsuo, T. Miura, M. Hilaga, "Image and video processing on mobile devices: a survey", Visual Computer, June, 2021.

[5] M. Yazdi, T. Bouwmans, "New Trends on Moving Object Detection in Video Images Captured by a moving Camera: A Survey", Computer Science Review, Elsevier, 2018, 28, pp.157-177.

[6] Business Finland, "Mixed reality report 2017", [Online], Available: https://www.businessfinland.fi/globalassets/finnish-customers/02-build-your-network/digitalization/mixed-reality/mixed-reality-report-2017.pdf.

[7] Qualcomm, "The mobile future of extended reality (XR)", [Online], Available: https://www.qualcomm.com/media/documents/files/the-mobile-future-of-extended-reality-xr.pdf.

[8] S. Doolani, C. Wessels, V. Kanal, C. Sevastopoulos, A. Jaiswal, H. Nambiappan, F. Makedon, "A review of extended reality (XR) technologies for manufacturing training", Technologies, 10 Dec., 2020, vol. 8, iss. 4, pp.77.

[9] R. Radke, "A Survey of Distributed Computer Vision Algorithms", Handbook of Ambient Intelligence and Smart Environments, Boston, 2010.

[10] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, Y. Xia, "Survey of Robot 3D Path Planning Algorithms", Journal of Control Science and Engineering, July, 2016.

[11] Edge computing, [Online], https://www.microsoft.com/en-us/research/project/edge-computing/.

[12] L. Qingqing, F. Yuhong, J. Peña Queralta, T. N. Gia, H. Tenhunen, Z. Zou, T. Westerlund, "Edge Computing for Mobile Robots: Multi-Robot Feature-Based Lidar Odometry with FPGAs", November, 2019, International Conference on Mobile Computing and Ubiquitous N.

[13] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, J. J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age", IEEE Transactions on robotics, vol. 32, no. 6, December,.

[14] J. Engel, T. Schöps, and D. Cremers, “LSD-SLAM: large-scale direct monocular SLAM,” in Proc. European Conf. on Comp. Vision (ECCV), Sep. 2014, Zürich, Switzerland.

[15] A. Rosinol, M. Abate, Y. Chang, and L. Carlone, “Kimera: an open-source library for real-time metric-semantic localization and mapping,” in Proc. IEEE Int. Conf. on Robotics and Automation (ICRA), Aug. 2020, Paris, France.

[16] C. Campos, R. Elvira, J. J. Gómez, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM3: an accurate open-source library for visual, visual-inertial and multi-map SLAM,” arXiv preprint arXiv:2007.11898, July 2020.

[17] ORB-SLAM3 Source Code, [Online], Available: https://github.com/UZ-SLAMLab/ORB_SLAM3.

[18] S. Ullman, "The interpretation of structure from motion", Oct. 1976, Proceedings of the Royal Society of London, Ser. B, VOl. 203, Iss. 1153, pp. 405-426.

[19] AliceVision Meshroom. [Online]. Available: https://alicevision.org#meshroom.

[20] Agisoft Metashape. [Online]. Available: https://www.agisoft.com/.

[21] M. R. U. Saputra, A. Markham, N. Trigoni, "Visual SLAM and structure from motion in dynamic environments: a survey", Jun. 2018, ACM Computing Surveys, Vol. 51, Iss. 2, pp. 1-36.

[22] K. Litomisky, "Consumer RGB-D Cameras and their Applications", University of California, Riverside, 2012, [Online], Available: http://alumni.cs.ucr.edu/~klitomis/files/RGBD-intro.pdf.

[23] Artec Eva, [Online], Available: https://www.artec3d.com/portable-3d-scanners/artec-eva-v2.

[24] EinScan HX, [Online], Available: https://www.einscan.com/handheld-3d-scanner/einscan-hx/.

[25] M. Nießner, M. Zollhöfer, S. Izadi, M. Stamminger. "Real-time 3D reconstruction at scale using voxel hashing", ACM Transactions on Graphics 2013.

[26] Dai Angela, Niessner Matthias, Zollöfer Michael, Izadi Shahram, Theobalt Christian. "BundleFusion: real-time globally consistent 3D reconstruction using on-the-fly surface re-integration", ACM Transactions on Graphics 2017.

[27] T. T. Niemirepo, M. Viitanen, and J. Vanne, “Open3DGen: Open-Source software for reconstructing textured 3D models from RGB-D images,” ACM Multimedia Syst. Conf., Istanbul, Turkey, Sept.-Oct. 2021.

[28] R. Hartley, A. Zisserman, "Multiple View Geometry in computer vision", Cambridge University Press, 2003.

[29] R. Hartley, A. Zisserman, "Multiple View Geometry in Computer Vision Second Edition", Cambridge University Press, March, 2004.

[30] Zhengyou Zhang, "A Flexible New Technique for Camera Calibration", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22, Issue 11, Nov. 2000.

[31] OpenCV: Open Computer Vision Library, [Online], Available: https://opencv.org.

[32] Y. Li, S. Wang, Q. Tian, X. Ding, "A survey of recent advances in visual feature detection", Neurocomputing, Vol. 149, Part B, February, 2015, pp. 736-751.

[33] D. Lowe, “Distinctive image features from scale invariant keypoints”, InternationalJournal of Computer Vision, vol. 60, pp. 91–110, 2004.

[34] H. Bay, A. Ess, T. Tuytelaars, L. Gool, "Speeded-Up Robust Features (SURF)", Computer Vision and Image Understanding, Vol. 110, Issue 3, Jun. 2008, Zürich, Switzerland.

[35] T. Tuytelaars, K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey", Foundations and Trends in Computer Graphics and Vision, Vol. 3, No. 3, 2007, pp.177-280.

[36] P. F. Alcantarilla, J. Nuevo, and A. Bartoli, “Fast explicit diffusion for accelerated features in nonlinear scale spaces,” in Proc. British Machine Vision Conference, Sep. 2013, Bristol, England.

[37] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “ORB: an efficient alternative to SIFT or SURF,” in Proc. IEEE Int. Conf. on Comp. Vision (ICCV), Mar. 2011, Barcelona, Spain.

[38] E. Karami, S. Prasad, M. Shehata, "Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images", Newfoundland Electrical and Computer Engineering Conference, St. Johns, Canada, November, 2015.

[39] C. Leng, H. Zhang, B. Li, G. Cai, Z. Pei, L. He, "Local Feature Descriptor for Image Matching: A Survey", IEEE Access, Vol. 7, December, 2018.

[40] A. Jakubovic, J. Velagic, "Image feature matching and object detection using brute-force matchers", Sep. 2018, International Symposium ELMAR, Zadar, Croatia.

[41] M. Muja, D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration", International Conference on Computer Vision Theory and Applications, 2009.

[42] J. L. Bentley, "MUltidimensional binary search trees used for associative searching", Communications of the ACM, Sep. 1975, Vol. 18, Iss. 9, pp. 509-517.

[43] H. Cantzler, "Random sample consensus (RANSAC)", Jun. 1981, Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh.

[44] Q. LV, W. Josephons, Z. Wang, M. Charikar, "Multi-probe LSH: efficient indexing for high-dimensional similarity search", Proceedings of the 33rd International Conference on Very Large Data Bases, Sep. 2007, Vienna, Austria.

[45] D. Monnin, E. Bieber, G. Schmitt, A. Schneider, "An effective rigidity constraint for improving RANSAC in homography estimation", 2010, Advanced Concepts for Intelligent Vision Systems, pp. 203-214.

[46] Y. I. Abdel-Aziz, H. M. Karara, "Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry", Feb. 2015, Photogrammetric Engineering & Remote Sensing, Vol. 2, pp. 103-107.

[47] G. H. Golub, C. F. Van Loan, (1996). "Matrix Computations (3rd ed.)", Johns Hopkins University Press, 1996.

[48] D. Bardsley, B. Li, "3D Reconstruction Using the Direct Linear Transform with a Gabor Wavelet Based .

[49] D. Nister (2004), ”An efficient solution to the five-point relative pose problem”, IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 26, Issue: 6, June 2004).

[50] R. Hartley, A. Zisserman, "Multiple view geometry in computer vision", 2003, Cambridge University Press.

[51] D. Nister, "An efficient solution to the five-point relative pose problem," IEEE Trans. Pattern Anal. Mach. Intell., vol. 26, no. 6, June 2004, pp. 756–777.

[52] S. Feng, J. Kan, Y. Wu, "An improved method to estimate the fundamental matrix based on 7-point algorithm", Journal of Theoretical and Applied Information Technology, Dec. 2012, Vol. 46, No. 1, pp. 212-217.

[53] M. A. Fischler, R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography", Communications of the ACM, Vol. 24, Iss. 6, Jun. 1981, pp. 381–395.

[54] Lepetit, V.; Moreno-Noguer, M.; Fua, P. (2009). "EPnP: an accurate O(n) solution to the PnP problem". International Journal of Computer Vision. 81 (2): 155–166.

[55] T. Whelan, M. Kaess, J.J. Leonard, and J.B. McDonald, "Deformation-based Loop Closure for Large Scale Dense RGB-D SLAM", IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS, Tokyo, Japan, November 2013.

[56] G. Sibley, C. Mei, I. Reid, P. Newman, "Adaptive Relative Bundle Adjustment", Robotics: Science and Systems, 2009.

[57] X. Gao, R. Wang, N. Demmel, D. Cremers, "LDSO: Direct Sparse Odometry with Loop Closure", International Conference on Intelligent Robots and Systems, 2018.

[58] M. Kazhdan, M. Bolitho, and H. Hoppe, “Poisson surface reconstruction,” in Proc. Eurographics Symp. on Geometry Processing (SGP), June 2006, Cagliari, Sardinia, Italy.

[59] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, “The ball-pivoting algorithm for surface reconstruction,” IEEE Trans. on Visualization and Comp. Graphics, vol. 5, no. 4, Nov. 1999, pp. 349-359.

[60] Q. Zhou, J. Park, and V. Koltun, "Open3D: a modern library for 3D data processing," arxiv.org/abs/1801.09847, Jan. 2018.

[61] BeagleBone PocketBeagle, [Online], Available: https://beagleboard.org/pocket.

[62] Arducam IMX477, [Online], Available: https://www.arducam.com/product/arducam-uvc-camera-adapter-board-for-12mp-imx477-raspberry-pi-hq-camera/.

[63] J. L. Schönberger, J. Frahm, "Structure-from-motion revisited, IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[64] X. Huang, L. Fan, J. Zhang, Q. Wu, C. Yuan, "Real time complete dense depth reconstruction for a monocular camera", IEEE Conference on Computer Vision and Pattern Recognition Workshops, Jul. 2016.

[65] Boost.Asio, [Online], Available: https://www.boost.org/doc/libs/1_76_0/doc/html/boost_asio.html.

[66] S. Akramullah, "Video coding performance", In: Digital video concepts, methods, and metrics, pp. 161-208, Apress, Berkeley, Oct. 2014.