January 2016

A Large Scale Inertial Aided Visual Simultaneous Localization And Mapping (SLAM) System For Small Mobile Platforms

Ashraf Qadir

A LARGE SCALE INERTIAL AIDED VISUAL SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM) SYSTEM FOR SMALL MOBILE PLATFORMS

By

Ashraf Qadir

Bachelor of Science, Bangladesh University of Engineering and Technology, 2005

Master of Science, University of North Dakota, 2010

A Dissertation

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Grand Forks, North Dakota

August

2016

This dissertation, submitted by Ashraf Qadir in partial fulfillment of the requirements for the Degree of Doctor of Philosophy from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

Dr. Jeremiah Neubert

Dr. William Semke

Dr. Matthew Cavalli

Dr. Naima Kaabouch

Dr. Ronald Marsh

This dissertation is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved.

Wayne Swisher
Dean of the School of Graduate Studies

Date

PERMISSION

Title	A Large Scale Inertial Aided Visual Simultaneous Localization and Mapping (Slam) System for Small Mobile Platforms
Department	Mechanical Engineering
Degree	Doctor of Philosophy

In presenting this dissertation in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by Dr. Jeremiah Neubert who supervised my dissertation work or, in his absence, by the Chairperson of the department or the dean of the School of Graduate Studies. It is understood that any copying or publication or other use of this dissertation or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my dissertation.

Ashraf Qadir
Date: 08/02/2016

TABLE OF CONTENTS

LIST OF FIGURES

Figure Page

LIST OF TABLES

Table Page

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Dr. Jeremiah Neubert, for introducing me to the world of robotics and perception systems, and allow me to work on the exciting research area. I am grateful for all the discussions, constructive suggestions and his guidance that not only made me a better engineer but also a better human being. I am also grateful to Dr. William Semke for his guidance, and support throughout the research project. I am thankful to the members of the advisory committee for their feedback and suggestions on the thesis.

I have had a wonderful six years at UND in the Robotics and Intelligent Systems Lab (RISL). I am thankful to all the previous and current students in the lab who helped me in the experiments and also made the lab a fun place to work.

I would like to thank my wife Noor for all her patience and support. Finally I am grateful to my parents whose support is the reason where I am today.

Ashraf Qadir

To My Mom and Dad

ABSTRACT

In this dissertation we present a robust simultaneous mapping and localization scheme that can be deployed on a computationally limited, small unmanned aerial system. This is achieved by developing a key frame based algorithm that leverages the multiprocessing capacity of modern low power mobile processors. The novelty of the algorithm lies in the design to make it robust against rapid exploration while keeping the computational time to a minimum. A novel algorithm is developed where the time critical components of the localization and mapping system are computed in parallel utilizing the multiple cores of the processor. The algorithm uses a scale and rotation invariant state of the art binary descriptor for landmark description making it suitable for compact large scale map representation and robust tracking. This descriptor is also used in loop closure detection making the algorithm efficient by eliminating any need for separate descriptors in a Bag of Words scheme. Effectiveness of the algorithm is demonstrated by performance evaluation in indoor and large scale outdoor dataset. We demonstrate the efficiency and robustness of the algorithm by successful six degree of freedom (6 DOF) pose estimation in challenging indoor and outdoor environment. Performance of the algorithm is validated on a quadcopter with onboard computation.

CHAPTER I

INTRODUCTION

The ability of a robot to perform tasks those are otherwise too dangerous, boring, onerous has led to a significant advancement in robotics research and development as well as their applications in industries such as auto, medical, manufacturing and space industries. As an example, the Mars Rover Curiosity or the underwater robot Caribou help us learn about places that are too dangerous to go. Robots have also begun to assist us in our everyday works. Starting with iRobot's robotic vacuum cleaner "Roomba' almost 12 years ago, cleaning robots are becoming ubiquitous.

@ http://mars.nasa.gov/msl/multimedia/images/

a. Mars rover Curiosity

@http://www2.ece.gatech.edu/research/labs/bwn/UWASN/figures/caribou.jpg

b. Under water robot Caribou

Figure 1: Examples of robots in operation. Figure a. illustrates the Mars rover Curiosity. Among other sensors it has one wide angle stereo camera (navcams) for ground navigation along with a pair of narrow angle multispectral cameras for imaging (MastCam). Figure b. illustrates the Caribou underwater robot. The robot comes with a side scan sonar and a sub-bottom profiler.

Recently Unmanned Aerial Vehicles are being used for a wide range of applications such as target tracking [1], precision agriculture [2], monitoring construction works etc.

Performing the tasks assigned to a mobile robot requires interaction with the environment and in turns it requires the robot to be able to sense its surroundings. Sometimes purely reactive strategies are sufficient. For example, some robotic vacuum cleaners achieve their tasks without any prior planning. They change their direction arbitrarily using a random walk once hit an obstacle. However to perform more complex tasks and in unknown environment, robots require knowledge of certain quantities such as its own location, position of its goals, locations of other objects in its immediate environment. In other words, the robot needs to simultaneously map unknown environment and estimate its current relative location which is essentially the fundamental simultaneous localization and mapping (SLAM) problem. However, these variables are seldom directly observable in an unknown environment and the robots employ sensors to acquire knowledge of its surroundings. The sensor information is then used to create an internal model of the state of the world along with the robots current location. The model is continuously updated with new senor information and used to make decisions on how to accomplish the assigned tasks. Choice of the sensors used by the robots depends on a variety of conditions such as applicability of a certain sensor for a certain task, the cost, form factor, power consumption, etc. This is particularly true for a mobile robot which often has limited computation and power budget.

Using camera as the main source of information for sensing the environment is a very active research topic since a camera is lightweight, consumes less power and also provides a rich amount of information. Nowadays, digital cameras are inexpensive and have a small form factor along with low power consumption. They can also operate reliably under harsh conditions since there are no moving mechanical parts. However, the

challenge is to process the large amount of information in real time to generate the model of the environment as well as the location of the camera at each instance. Each image contains hundreds of thousands of pixels and inferring the relevant information under real time constraints is challenging. In addition, monocular SLAM using a single camera that consists of only one lens and one image sensor is difficult compared to other types of sensors that provide range/bearing information such as laser range finders. As the depth information is not available, it needs to be inferred from the inter-frame motion. Given a world point observable in two or more image frames, the depth can be estimated using triangulation but only up to a scale as the triangulation needs to be performed over time and the depth depends on the relative displacement between the two camera positions.

Despite the difficulties, vision is an appealing sensor as it is the most frequent sensor in nature. A large amount of species including humans rely mainly on vision for localization and navigation tasks demonstrate the applicability of visual SLAM. Our visual cortex enables us to perceive and interpret visual scene and images taken with cameras. We extract geometric information form images and also analyze their semantic contexts. This inspires a large number of research and the computer vision community made a great progress in developing systems that can detect humans, objects, locations, events, etc.

In this thesis we emphasis on monocular SLAM; we focus on developing a reliable and computationally efficient SLAM algorithm for in small mobile platforms such as a miniature aerial vehicles. In the remaining sections of this introduction we look at the formulation of the visual SLAM systems where the environment is represented as a 3D point cloud and then outline the main contributions of the thesis.

Visual SLAM

We assume that we are given M input images of a scene acquired by a single moving camera at different times. The 3D structure of the scene is modeled as N 3D points that are partially observed in the M images. The projection of a scene point $x_i \in R^3$, observed using a camera P_j with six degree of freedom will result in an image point $u_{ij} \in \Omega \subset R^2$. If we have a measurement of the image point $\overline{u_{ij}} \in \Omega$, the error between the predicted and observed image point can be written as:

$$\Delta u_{ij} = u_{ij} - \overline{u_{ij}} \tag{1}$$

The probability density function over the error is often assumed to be a multivariate Gaussian distribution with diagonal covariance matrix $\sigma_{ij} \in R^{2\times 2}$:

$$p(u_{ij} \mid P_j, x_i) \infty \exp(\frac{1}{2} \Delta u_{ij}^T \sigma_{ij}^{-1} \Delta u_{ij}) \tag{2}$$

Assuming that the observations of multiple scene points across multiple cameras is an independent process, then for structure and camera motion parameters $X = \{x_1, x_2, ...x_N\}$, $P = \{P_1, P_2, ...P_M\}$ along with observations $\overline{U} = \{\overline{u_{ij}} \mid c_{ij} = 1\}$, the probability density function over all observation can be written as:

$$p(\overline{U} \mid X, P) \propto \prod\prod p(\overline{u_{ij}} \mid x_i, P_j) \tag{3}$$

Using Bayes rule we can write the likelihood function of the structure and motion as:

$$p(X, P \mid \overline{U}) \propto p(\overline{U} \mid X, P) P(X, P) \tag{4}$$

where $p(X, P)$ is the prior over the structure and camera motion parameters. The most likely structure and camera parameters can be estimated by maximizing the posterior distribution given in above equation. Equivalently we can minimize the energy function resulting from the negative log likelihood of $p(X, P \mid \overline{U})$. Optimization over the parameters is performed using a non-linear iterative minimization scheme that requires an initial estimate of the point positions and the camera poses. A graphical representation of the visual SLAM problem as a Bayesian network is shown below:

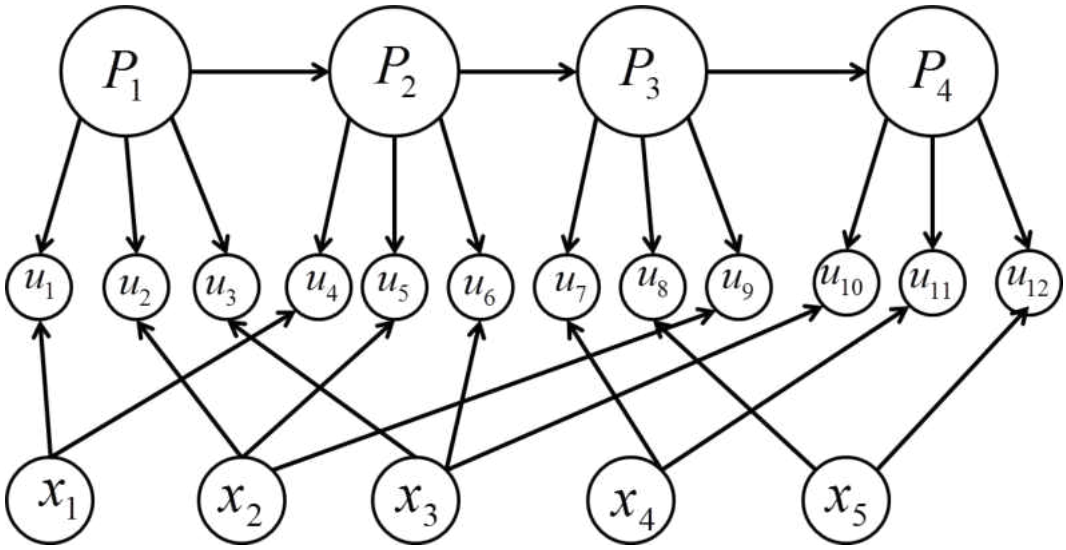


Figure 2: The visual SLAM problem shown as a Bayesian Network that represents the causal relationships between a camera with pose P_j viewing a 3D point x_i imaged as u_{ij}.

A number of assumptions are made in the above formulation of the visual SLAM systems. It was assumed that the correspondences of the observed points across multiple images are known as well as initial estimates of the parameters are available. In practice these are the challenges that need to be solved. In addition, we are interested in real time structure and camera pose estimation.

Main Contributions

A Real Time Large Scale Monocular SLAM System

A state of the art large scale monocular SLAM algorithm is presented which is robust and is able to operate in indoor and outdoor environments. The major features of the algorithm are

1. Decoupling of the frontend tracking and exploration and backend optimization so that it can leverage the multiprocessing capacity of the processor while ensuring the robustness of the algorithm. The algorithm uses state of the art binary feature descriptors for both 3D point cloud map representation as well as appearance based loop closure detection. The algorithm is optimized by employing parallel processing of the computationally critical parts of the algorithm.
2. A novel re-localization algorithm for faster recovery in the event of tracking failure that increases the robustness of the algorithm.
3. A robust loop closure detection and correction algorithm based on geometric and temporal verification to ensure correct topological structure of the model of the environment. In addition, the loop detection and correction algorithm is computationally efficient.

Inertial Aided Monocular Visual SLAM For Small Mobile Platforms

The real time monocular SLAM of the previous section is extended with the IMU data available from the autopilot. Major contributions in this section are

1. Development of an inertial aided large scale visual SLAM algorithm for small mobile platforms with limited payload capacity. The algorithm runs on a small form factor single board computer onboard a small quadrotor. To our knowledge, this is the first

full scale SLAM algorithms capable of running real time on a miniature aerial vehicle. Fusion of inertial and visual information in the algorithm ensures reliable pose estimation as well as metric scale estimation.

2. Development of a low cost miniature aerial vehicle as a test bed for the proposed algorithm. The test bed development include both hardware and software development for control and communications.
3. A C++ framework of the algorithm that can be used as an off the shelf product. The algorithm is being used with other projects in the RISL (Robotics and Intelligent System Lab) at UND.

Thesis Organization

The thesis is organized as follows: In section II we provide the mathematical backgrounds that are necessary to make the document self-content and easy to follow. Only the definitions and derivations are provided that are used in the algorithms and a list of references are provided for the interested reader to acquire more in depth knowledge on the mathematical topics.

Section III describes the real time monocular SLAM algorithm in details. First a detailed process flow of the algorithm is provided and then each section is explained in details. The result section includes performance evaluation of the algorithm on the indoor dataset as well as publicly available challenging outdoor dataset. The indoor dataset is created from images of the indoor lab environment for initial testing and validation of the developed algorithm. In order to validate the effectiveness of the algorithm in challenging outdoor environment, a publicly available data set is used that include static and partially dynamic city streets as well as environment that contain only vegetation.

Section IV describes the extension of the SLAM algorithm that leverages IMU information from the mobile robotics platform to complement the visual SLAM algorithm described in the previous section. We provide the detail description of how the IMU information is integrated in the SLAM algorithm to generate the motion prediction as well as the fusion of the IMU and visual information for metric scale estimation. The performance evaluation

CHAPTER II

PRELIMINARIES

Camera Projection Model

The camera projection model used in this thesis is the central camera projection model. The projection model describes how a point in 3D world is drawn on the image plane. Let us denote a 3D world point $X = [X, Y, Z]^T$ and its corresponding image plane point $z = [u, v]^T$. We follow the standard image coordinate convention that the top left corner of the image is the origin o, with u-axis pointing to the right and the v-axis pointing down. The pose of the camera in the world coordinate system is denoted as rigid body transformation $T_C^W \in SE(3)$ with the origin as the center of projection. The coordinate convention of the camera frame is chosen as x-axis pointing to the right, y-axis pointing down and z-axis pointing forward. This convention simplifies the projection of a point from the camera coordinate to the image coordinate.

In order to project the point from world to image plane, the first step is to transform the point from world to camera frame as

$$X^C = T_W^C X = R_W^C X + t_W^C \tag{5}$$

Then we employ the pinhole projection model to project the point $X^C = [x, y, z]^T$ to the normalized image plane $z = 1$. Denoting the projection function as ***proj*(.)**, we can write

$$proj(X^C) = \frac{1}{z}\begin{pmatrix} x \\ y \end{pmatrix} \tag{6}$$

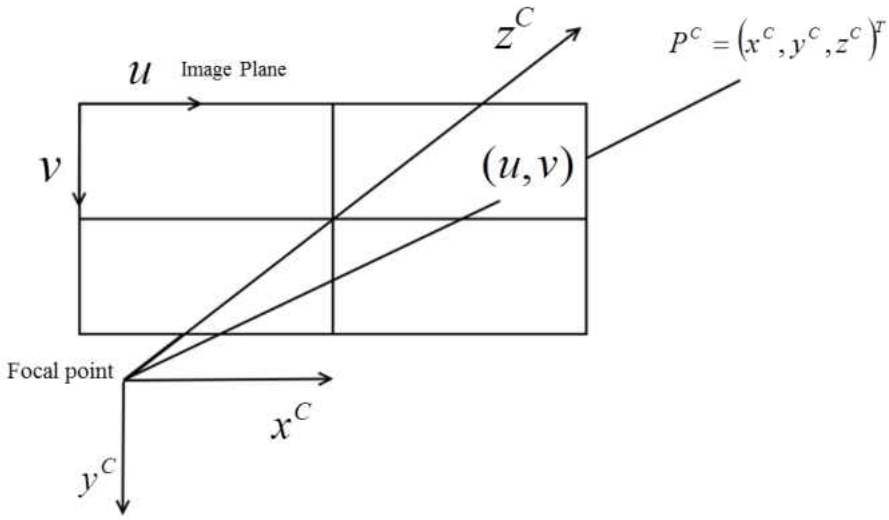


Figure 3: Central camera projection model

If f is the focal length of the camera and the principal point $p = (c_x, c_y) \in I$ in the image plane, the projection can be written as

$$proj(K, X^C) = f \cdot \frac{1}{z}\begin{pmatrix} x \\ y \end{pmatrix} + p \tag{7}$$

with $K = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix}$ being the intrinsic camera matrix.

Putting these together, our forward projection model from a 3D world point to the image point becomes

$$z = proj(K, T_W^C X) \tag{8}$$

We also require an inverse of the projection from image plane to the normalized image plane $z = 1$. This is written as $x = K^{-1} z$ with

$$K^{-1} = \begin{pmatrix} 1/f & 0 & -c_x/f \\ 0 & 1/f & -c_y/f \\ 0 & 0 & 1 \end{pmatrix} \tag{9}$$

This is a linear model that does not take into account the lens distortions. In order to use this model, a preprocessing step is used to un-distort the images. This is done very easily using OpenCV [3] computer vision library.

Epipolar Geometry

Epipolar geometry plays an important role in describing the relation between corresponding image points of a 3D world point when viewing from different camera position. The forward projection model described above associates a point in 3D world coordinate to an image point. However, due to the projective nature of a monocular camera, the image point can be associated with an infinite ray in the world. That means, given a relative displacement $T_a^b = [R, t]$ between two camera poses, a point in frame *a* corresponds to a line in frame *b*. This concept is demonstrated in the Figure 2. This is true since two image points x and x', the 3D world point X and the two camera centers are coplanar. This is written as

$$l_b = Fx \tag{10}$$

where, F is called the fundamental matrix. Given two camera matrices $P = K[I \mid 0]$ and $P' = K[R \mid t]$, the fundamental matrix can be calculated as follows [22s]:

$$F = K'^{-T}[t]_\times RK^{-1} = [K't]_\times K'RK^{-1} = K'^{-T}RK^T[KR^T t]_\times \tag{11}$$

Since the corresponding image point x' in the b frame lies on the line l_b we get the relation between the corresponding image points in terms of fundamental matrix F as

$$(x')^T l_b = 0 \text{ or } (x')^T Fx = 0 \tag{12}$$

However, the fundamental matrix has a singularity which corresponds to pure rotation. In case of pure rotation, it can be easily verified from the equation that $(x')^T Fx = 0$ for all corresponding pairs x and x'.

If the camera intrinsic matrices K and K' are known, we can set the first of the two cameras as the origin and then the fundamental matrix between two camera poses $P = [I \mid 0]$ and $P' = [R \mid t]$ is called the essential matrix and is of the form

$$E = [t]_\times R = R[R^T t]\times \tag{13}$$

The essential matrix then satisfies the relation between two normalized image coordinates $\hat{x}$ and $\hat{x}'$ as $\hat{x}'^T E\hat{x} = 0$. The relation between fundamental matrix and essential matrix can be written as

$$E = K'^T FK \tag{14}$$

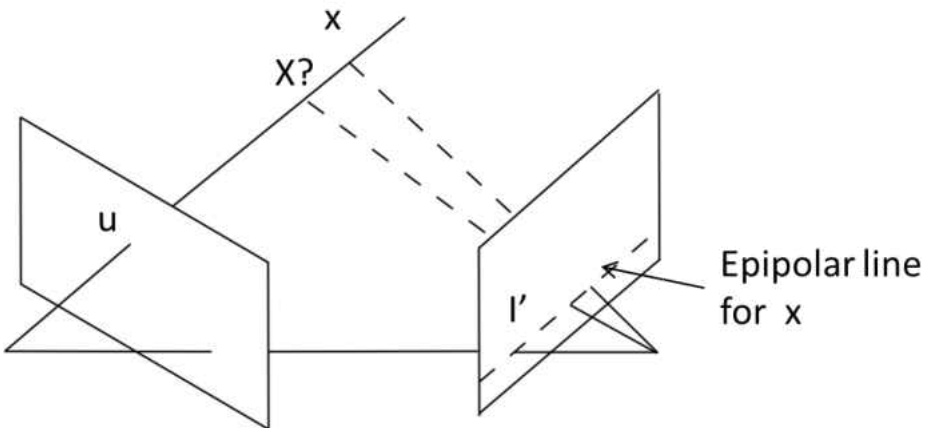


Figure 4: Epipolar Line Geometry in two views describes the incident relationship between a image point in one image and the epipolar line in the other image

Given a set of point correspondences $x_i \leftrightarrow x_j$ in two images, a linear solution for the fundamental matrix up to a scale can be found from at least 8 points correspondences [22].

Camera Poses from Essential Matrix

Once the essential matrix is known, it can be used to compute relative camera poses up to a scale ambiguity. Assuming that the first camera pose is a canonical pose $P = [I \mid 0]$, the second camera rotation and translation can be computed by first factorizing the essential matrix into a product of a skew symmetric matrix S and a rotation matrix R. Denoting an orthogonal matrix W and a skew symmetric matrix Z as

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; Z = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

A block decomposition of the skew symmetric matrix can be written as $S = kUZU^T$ where U is orthogonal. Writing $Z = diag(1,1,0)W$ up to sign, S can be written up to scale as $S = Udiag(1,1,0)WU^T$ and then $E = SR = Udiag(1,10)(WU^T R)$. This is singular value decomposition of E with two equal singular values. Because of the two equal singular values, the SVD is not unique and there is two possible factorization of the matrix $E = SR$.

These factorizations are written as $S = UZU^T$ with $R = UWV^T$ or UW^TV^T. The factorization for S determines t part since $S = [t]_\times$ with $\|t\| = 1$. Now it can be easily shown that $t = U(0,0,1)^T = u_3$, since $St = [t]_\times t = t \times t = 0$. However, the sign of t is

ambiguous. As a result there are 4 possible choices for the camera rotation R and translation t based on two choices of R and two signs of t. The 4 potential camera matrices can be written as

$$P_b = \left[UWV^T \mid +u_3\right] \text{ or } P_b = \left[UWV^T \mid -u_3\right] \text{ or } P_b = \left[UW^TV^T \mid +u_3\right] \text{ or } P_b = \left[UW^TV^T \mid -u_3\right] \tag{15}$$

From the 4 potential solutions, the valid solution is found by performing 3D reconstruction of the corresponding points and then counting the number of valid 3D reconstruction. A valid 3D reconstruction is the one where the 3D position of the point is in front of both cameras. Theoretically, only one point is sufficient to decide between the four solutions, however, it is always possible that the point correspondence is not correct between two camera views and checking all the points and count the number of inliers is desirable.

Matrix Lie Groups and Optimization on Matrix Manifolds

Throughout the thesis, the rigid motions in 2D and 3D spaces are represents as Matrix Lie groups. This section is a collection of definitions and useful mathematical formulas that are used in subsequent chapters. We avoid rigorous introduction to Manifolds, Lie Groups, their algebras and the mathematical details of Lie Groups in general. However, we provide the definitions along with the mathematical derivations required to make the document self- contained. Specifically, we list the properties for matrix Lie groups of 2D and 3D rigid body transformations and derive the jacobians used in this thesis.

Motivation

Standard optimization algorithms work properly on Euclidean vector spaces (i.e. spaces isomorphic to $\Re^n$). However, sometimes the variables do not constitute a Euclidean vector space. A classic example is the representations of 3D rotations. A 3D rotation can be minimally represented using the Euler angles representation where the overall rotation is represented as a sequence of three individual rotations. In robotics, the usual convention is the roll-pitch-yaw (RPY) convention. However, this parameterization consists of two degenerate cases, specifically when pitch approaches ± 90. In this case, the gimbal lock occurs where a change in roll becomes a change in yaw. There is not a unique correspondence between any possible rotation in 3D and a triplet of roll-pitch-yaw angles. These situations need to be detected and handled.

One popular alternative is to over parameterizing the variables. For example, using quaternions (with 4 values) to represent the 3D rotations and normalize the parameterization in some ways. However, over parameterization has its own challenge in optimization. Optimization algorithms are not aware of any inner constraints between the parameters and they will optimize some DOF which do not actually exist. In addition, sometimes the parameters have non-euclidean behavior where a small change of the same magnitude to different parameters results in a change of the variables of quite different magnitude. As a result, solving optimization problems on manifolds becomes increasingly popular where, the variables are globally over parameterized, but local changes are represented with a minimal representation.

We are interested in rigid transformations in 2D and 3D spaces where the transformations need to be composed, inverted, differentiated and interpolated. Matrix Lie groups provide an elegant way to represent 3D rigid transformations and perform above mentioned operations on the rigid body transformations.

Lie Groups, Lie Algebras and Associated Properties

A Lie Group G is both a group and a manifold with smooth group operation. The Lie group g has an associated Lie algebra, which can be identified as the tangent space around the identity element in the group. The associated Lie algebra is a vector space which is generated by differentiating the group transformations along chosen directions in space, at the identity element of the group. The tangent space has the same structure for all group elements; however, a coordinate transformation is required to when a tangent vector is moved from one tangent space to another. The basis elements of the tangent space are called generators and the tangent vectors are represented as linear combinations of the generators.

As a vector space the Lie algebra g is isomorphic to R^n, and we can define the "hat operator" $\wedge : x \in \Re^n \mapsto \hat{x} \in g$, which maps n-vectors $x \in \Re^n$ to elements of g. In the case of matrix Lie groups, the elements $\hat{x}$ of g are also $n \times n$ matrices, and the map is given by

$$\hat{x} = \sum_{i=1}^{n} x_i G^i \tag{16}$$

Properties. Lie group properties:

1. Differential quantities related to a group such as velocities, Jacobians, and covariance of transformations are well represented in the tangent space around a transformation.

This is an optimal space because the tangent space is a vector space with the same dimension as the degrees of freedom of the group.

2. The exponential map converts any element of the tangent space exactly into a transformation in the group.
3. The adjoint linearly and exactly transforms tangent vectors from one tangent space to another.
4. Matrix Lie groups has a group action. For example, 2D rotations act on 2D points and 3D transformations act on 3D points.
5. Matrix Lie groups are not commutative in general. For example, two invertible square matrices $A, B \in \Re^{n \times n}$, their product $AB \neq BA$. However, the group elements commute with the group identity element: $AI = IA$. Thus if we go infinity close to the identity, we enter a space which is commutative. This space is the tangent space at the origin: lie algebra for the group.

Below we introduce the Lie groups and their associated Lie algebras used in this thesis. We also provide the useful Jacobians that are required in the optimizations.

$SO(3)$, The Group of 3D Rotations

The Group Representation. The elements of the group $SO(3)$ are represented by 3D rotation matrices. Since the rotation matrices are orthogonal, their inversion is equivalent to transposition

$$\begin{aligned} R &\in SO(3) \\ R^{-1} &= R^T \end{aligned} \tag{17}$$

Lie Algebra of $SO(3)$. The group s an associated Lie algebra $so(3)$ which is the set of 3x3 skew symmetric matrices. The base of $so(3)$ are 3 skew symmetric matrices (the generators) correspond to infinitesimal rotations along each axis. The generators are defined as

$$\begin{aligned} G_1^{so(3)} &= \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}_\times = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} \\ G_2^{so(3)} &= \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}_\times = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{pmatrix} \\ G_3^{so(3)} &= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}_\times = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{aligned} \tag{18}$$

An element of $so(3)$ can be represented as the linear combination of the generators. For example

$$\begin{aligned} &\omega \in \Re^3 \\ &\omega_1 G_1 + \omega_2 G_2 + \omega_3 G_3 \in so(3) \end{aligned} \tag{19}$$

Exponential Map. The exponential map takes the member of $so(3)$ to rotation matrices is defined as

$$\begin{aligned} &\exp : so(3) \to SO(3) \\ &\omega \to R_{3\times 3} \end{aligned} \tag{20}$$

The map is simply the matrix exponential and has the closed form solution as

$$e^{\omega} = e^{[\omega]_\times} = I_3 + \frac{\sin\theta}{\theta}[\omega]_\times + \frac{1-\cos\theta}{\theta^2}[\omega]_\times^2 \tag{21}$$

where, the angle $\theta = |\omega|$. The exponential map can be inverted to get the logarithm map from $SO(3)$ to $so(3)$ as

$$\ln(R) = \frac{\theta}{2\sin\theta} \cdot (R - R^T) \tag{22}$$

$SE(3)$, The Group of 3D Rigid Body Transformations

The Group Representation. The group of rigid transformations in 3 space is denoted as $SE(3)$ and its members are the set of 4x4 matrices with the structure $T = \begin{pmatrix} R & t \\ 0_{1\times3} & 1 \end{pmatrix}$, where $R \in SO(3)$, and $t \in \Re^3$.

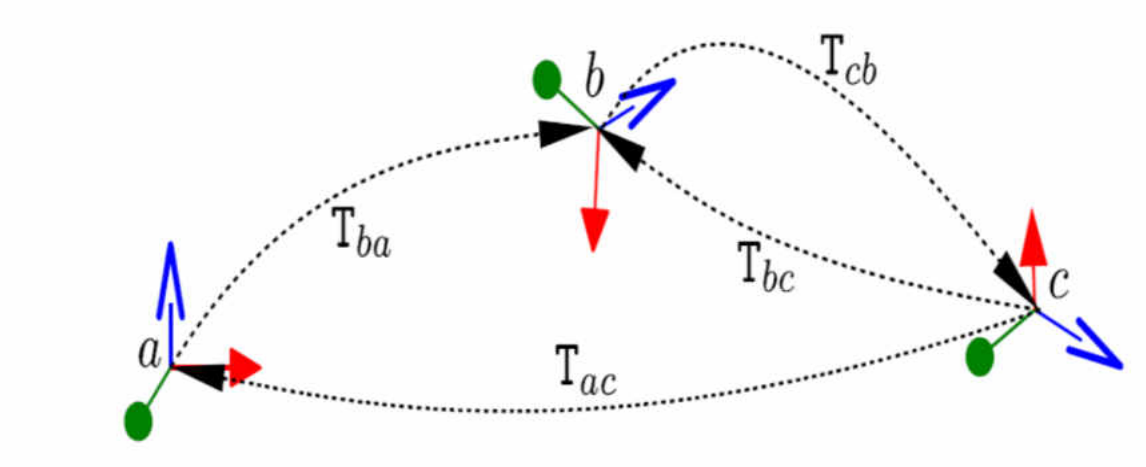


Figure 5: Poses represented as rigid body transformations

Properties. Properties of 3D rigid body transformations are described as

1. $SE(3)$ is a 6 dimensional manifold; three corresponding to 3D translation vector and the other three corresponding to the 3D rotation vector.
2. $SE(3)$ is a semidirect product of the groups $SO(3)$ and $\Re^3$
3. If $g_1, g_2 \in SE(3)$, then their composition $g_1 g_2 \in SE(3)$

4. $I \in T_{4\times4}$ is the identity element of $SE(3)$

5. If $g \in SE(3)$, then $g^{-1} = \begin{bmatrix} R^T & -R^T p \\ 0 & 1 \end{bmatrix} \in SE(3)$

Lie Algebra of $SE(3)$. The associated Lie algebra for the group is denoted as $se(3)$ whose bases are six 4x4 matrices, each correspond to either infinitesimal rotation or translation along the axes.

The generators are defined as

$$G_1^{se(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \quad G_2^{se(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \quad G_1^{se(3)} = \left(\begin{array}{ccc|c} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right)$$

$$G_1^{se(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \quad G_1^{se(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \quad G_1^{se(3)} = \left(\begin{array}{ccc|c} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 0 \end{array}\right) \tag{23}$$

An arbitrary element in $se(3)$ has six coordinates where each coordinate multiplies a generator matrix.

Exponential Map. The exponential map from $se(3)$ to $SE(3)$ can be defined as follows:

For a vector $v = \begin{pmatrix} t \\ \omega \end{pmatrix}$ that represents the 6 vector of coordinates in the Lie algebra $se(3)$, we define the algebra,

$$a\lg\begin{pmatrix} t \\ \omega \end{pmatrix} = \left(\begin{array}{c|c} [\omega]_\times & t \\ \hline 0 & 0 \end{array}\right) \tag{24}$$

where $t, \omega \in \Re^3, \theta \equiv \sqrt{\omega^T \omega}$. The exponential map $\exp : se(3) \mapsto SE(3)$ is well defined, surjective (onto) and has the closed form solution

$$e^v = \exp\left(a\lg\begin{pmatrix} t \\ \omega \end{pmatrix}\right) = \left(\begin{array}{c|c} e^{[\omega]_\times} & V.t \\ \hline 0_{1\times 3} & 0 \end{array}\right) \tag{25}$$

where, $V = I + \left(\frac{1-\cos\theta}{\theta^2}\right)[\omega]_\times + \left(\frac{\theta-\sin\theta}{\theta^3}\right)[\omega]_\times^2$ and $e^{[\omega]_\times}$ is defined as before.

$SIM(3)$, Group of Similarity Transforms

The Group Representation. $SIM(3)$ is the group of affine transformations in 3D space which are the composition of a rotation, a translation and a scale. The group has 7 degrees of freedom (DOF): 3 for translation, 3 for rotation and 1 for scaling. Members of this group are the set of 4x4 matrices with the following structure

$$T = \left(\begin{array}{c|c} sR & t \\ \hline 0 & 1 \end{array}\right) \tag{26}$$

with

$$\begin{aligned} &R \in SO(3) \\ &t \in \Re^3 \\ &s \in \Re^+ \end{aligned} \tag{27}$$

A matrix $A \in R^+ \times SO(3)$ has the following properties: $AA^T = A^T A = s^2 I$ and $\det(A) = s^3$

Lie Algebra of $SIM(3)$. The Lie algebra *sim*(3) for the group consists of all (3+1)x(3+1) matrices of the form

$$\begin{pmatrix} \lambda I_3 + \Omega & u \\ 0 & 0 \end{pmatrix} \tag{28}$$

with $\Omega \in so(3), u \in R^3, \lambda \in R$

The generators of Sim(3) include the ones of SE(3) plus $G_7^{sim(3)} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 \end{array}\right)$

Exponential Map. The exponential map can be defined as follows:

Given a 4x4 matrix B of the form

$$\begin{pmatrix} \lambda I + \Omega & u \\ 0 & 0 \end{pmatrix} \tag{29}$$

where Ω is any real 3x3 matrix, $\lambda \in R$ and $u \in R^3$, we have

$$e^B = \begin{pmatrix} e^\lambda e\Omega & Vu \\ 0 & 1 \end{pmatrix} \tag{30}$$

with

$$V = \frac{(e^\lambda - 1)}{\lambda} I_3 + \frac{(\theta(1 - e^\lambda \cos\theta) + e^\lambda \lambda \sin\theta)}{\theta(\lambda^2 + \theta^2)} \Omega + \left(\frac{(e^\lambda - 1)}{\lambda\theta^2} - \frac{e^\lambda \sin\theta}{\theta(\lambda^2 + \theta^2)} - \frac{\lambda(e^\lambda \cos\theta - 1)}{\theta^2(\lambda^2 + \theta^2)} \right) \Omega^2$$

(32)

Denoting $\Omega = [\omega]_\times = \begin{pmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ \omega_y & \omega_x & 0 \end{pmatrix}$, and $\theta = \sqrt{\omega_x^2 + \omega_y^2 + \omega_z^2}$

Nonlinear Least Square Methods

In nonlinear least square methods, the parameter values of a model are estimated by minimizing a nonlinear cost function. Given a set of measurements $z_i \in R^m$ predicted by a model $\hat{z}_i(x)$, where x is the vector of model parameters; nonlinear least square methods estimate the model parameter values by minimizing the weighted sum of squared errors cost function

$$f(x) = \frac{1}{2}\sum_i (z_i - \hat{z}_i(x))\Sigma_i (z_i - \hat{z}_i(x))^T \tag{31}$$

where $(z_i - \hat{z}_i(x))$ is the feature prediction error and Σ_i is an arbitrary symmetric positive semi-definite weight matrix. When the observations are independent of each other and are perturbed by Gaussian noise of mean zero, and constant variance, minimizing the sum of squared errors is equivalent to minimizing the negative log likelihood. As such, the weights are chosen to approximate the inverse measurement covariance of z_i. In order to simplify the formulations of the least squares, the measurements can be assembled into a compound measurement vector $Z \equiv (z^T{}_1, z_2^T, ..., z^T{}_k)^T$ and the weight matrices into a compound block diagonal weight matrix $\Sigma \equiv diag(\Sigma_1, \Sigma_2, ..., \Sigma_k)$. Under such compounding, the weighted squared error $f(x) = \frac{1}{2}\Delta Z(x)\Sigma\Delta Z(x)^T$ is same as the sum of squared errors in equation (33).

Gauss Newton method is one of the most commonly used techniques for optimizing nonlinear least square problems. It is an approximation of the second order Newton method that uses a search direction vector and step size to iteratively update an initial estimate of the parameters X. Given the nonlinear weighted SSE cost function

$f(x) = \frac{1}{2}\Delta Z(x)\Sigma\Delta Z(x)^T$ with the prediction error $\Delta z(x) = z - \hat{z}(x)$, the gradient and Hessian matrix can be written as

$$g = \frac{df}{dx} = \Delta z^T \Sigma J \tag{32}$$

and

$$\mathrm{H} = \frac{d^2 f}{dx^2} = J^T \Sigma J + \sum_i (\Delta z^T \Sigma)_i \frac{d^2 z_i}{dx^2} \tag{33}$$

with $J = \frac{dz}{dx}$ is the jacobian. The second term in the Hessian matrix can be ignored if the prediction error $\Delta z(x)$ is small or the model is nearly linear which means $\frac{d^2 z_i}{dx^2} \approx 0$.

Dropping the second term gives the Gauss-Newton approximation $H \approx J^T \Sigma J$. With this approximation, the normal equation to solve for the steps become

$$(J^T \Sigma J)\delta x = -J^T \Sigma \Delta Z \tag{34}$$

The parameters update equation is then $x \to x + \delta x$

A variant of the Gauss Newton method is the Levenberg-Marquardt (LM) that alters the normal equation as follows

$$(J^T \Sigma J + \mu I)\delta x = -J^T \Sigma \Delta Z \tag{35}$$

where a small value of the algorithm parameter μ results in a Gauss Newton approximation and a large value achieves a gradient descent update. As a result the parameter μ is initialized with a large value so that the first updates are steps in the steepest descent direction. Then value of μ decreases after an update, when the residual error is minimized. The algorithm continues until a convergence criterion is met.

Optimization on Manifolds

When optimizing on Manifold, the state vector $x \in M$ is a point on an n-dimensional manifold. The prediction model $\hat{z}(x) = h(x) : M \to R^m$ now a function on the manifold that predicts the measurements z from x. However, Lie groups are not as easy to treat as vector space $\Re^n$ and computing the jacobians require special attention. For example, if we consider the group $SO(3)$, computing a derivative in the form $\frac{\partial R}{\partial R_{(i,j)}}$ makes no sense as any infinitesimal change to a single entry of an orthogonal matrix would make the matrix non-orthogonal and we would leave the space of $SO(3)$. Elements of $SO(3)$ have only 3 DOF and there are exactly three Cartesian directions about which we can modify R which are the basis vectors of the tangent space.

Following the idea presented above, and denoting a small increment $\varepsilon \in R^n$ in the linearization of the manifold around a point $x \in M$ (using M s Lie algebra as a vector base), the jacobian matrix is computed by first left-multiplying the point x by the exponential map of the increment and then differentiating the resulting expression around origin. This can be written as

$$J \leftarrow \left. \frac{\partial f(x \oplus \varepsilon)}{\partial \varepsilon} \right|_{\varepsilon=0} \tag{36}$$

where $x \oplus \varepsilon = \exp(\hat{\varepsilon})x$. In the following section we provide an example of computing the jacobian for the 3D point projection function.

Point Projection Jacobian

For a 3D point in world coordinate frame, the projection function $h_C(X)$ maps the point from world to normalized image plane $z=1$. The transformation of the world point to the camera frame can be written as $\begin{pmatrix} x \\ y \\ z \end{pmatrix} = TX = a = RX + t$, where $a = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ is the 3D point in current camera frame T. Projection of this point on the normalized image plane $z=1$ is our measurement prediction and can be written as

$$h_C(X) = proj(a) = \begin{pmatrix} \frac{x}{z} & \frac{y}{z} \end{pmatrix}^T \tag{37}$$

The derivative of the projection function with respect to an arbitrary variable α can be computed using the chain rule as

$$\frac{\partial h_c(X)}{\partial \alpha} = \frac{\partial proj(a)}{\partial a} \frac{\partial a}{\partial \alpha} \tag{38}$$

where $\frac{\partial proj(a)}{\partial a}$ is the common 1st factor in all jacobians of the point projection model.

The term $\frac{\partial proj(a)}{\partial a}$ is expanded as

$$\frac{\partial proj(a)}{\partial a} = \frac{\partial \begin{bmatrix} x/z & y/z \end{bmatrix}}{\partial \begin{bmatrix} x & y & z \end{bmatrix}^T} \tag{39}$$

$$= \frac{1}{z} \begin{bmatrix} 1 & 0 & -x/z \\ 0 & 1 & -y/z \end{bmatrix} \tag{40}$$

2nd term can be written as $\frac{\partial a}{\partial \alpha} = \frac{\partial (RX + t)}{\partial \alpha}$. With respect to a landmark, it can be expressed as

$$\frac{\partial(RX+t)}{\partial X} = R \tag{41}$$

and with respect to the camera pose, the derivative is expressed as

$$\frac{\partial TX}{\partial T} = \left.\frac{\partial e^{\hat{\varepsilon}}}{\partial \varepsilon}\right|_{\varepsilon=0} TX \tag{42}$$

Now using 1st order Taylor series approximation we can write,

$$\left.\frac{\partial e^{\hat{\varepsilon}}}{\partial \varepsilon}\right|_{\varepsilon=0} \approx \left.\frac{\partial(I+\hat{\varepsilon})}{\partial \varepsilon}\right|_{\varepsilon=0} \tag{43}$$

$$= \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0_{3\times3} & -[e_1]_\times \\ 0_{3\times3} & -[e_2]_\times \\ 0_{3\times3} & -[e_3]_\times \\ I_3 & 0_{3\times3} \end{pmatrix} \tag{44}$$

Finally the jacobian can be written as

$$\frac{\partial TX}{\partial T} = \begin{bmatrix} 1 & 0 & 0 & 0 & z & -y \\ 0 & 1 & 0 & -z & 0 & x \\ 0 & 0 & 1 & y & -x & 0 \end{bmatrix} \tag{45}$$

Bundle Adjustment

Bundle adjustment is essentially an iterative optimization technique which aims to generate jointly optimal three dimensional structure and viewing parameters by minimizing a cost function. Usually the cost function is the distance between the

reprojection of a three dimensional model and the associated features in the image. In this section we give a brief introduction to the method. An excellent tutorial on bundle adjustment as well as mathematical details can be found in [17]. There are several frameworks such as g2o[9] , ceres solver[61], etc. available to perform bundle adjustment if the three dimensional structure is represented as a 3D point cloud.

Let us assume that a camera is moving in a 3D space and recording a sequence of images $I_1, I_2, I_3 ... I_n$. Assuming the scene geometry is represented by a set of discrete 3D points $x_1, x_2, ... x_m$ for which we have an initial estimation from a reconstruction method along with the initial estimation of the camera poses $T_1, T_2, T_3 .... T_n$. We also have a set of observations Z for the 3D points where $z_{i,j} \in Z$ is a measurement of point x_i in image frame I_j. Bundle adjustment can be considered as the refinement part starting from the initial estimation by iteratively minimizing the distance $d_{i,j}$ between the observations and the reprojection of the 3D points in the image frames $d_{i,j}(T_j, x_i) := z_{i,j} - \hat{z}(T_j . x_i)$. The parameter space is essentially a high dimensional manifold that consists of the set of 3D points along with the set of camera poses $T_1, T_2, T_3 .... T_n$.

Multivariate Gaussian

Let $\xi \in R^n$ be a random vector with mean $\mu \in R^n$, and covariance $\sum \in S^n_{++}$ (n dimensional positive definite cone). We write $\xi_t \sim N(\mu_t, \sum_t)$ to state that ξ is guassian with mean μ and covariance $\sum$. We can write the probability distribution as

$$p(\xi_t) = N(\xi_t, \mu_t, \Sigma_t)$$
$$= \frac{1}{(2\pi)^{n/2}\sqrt{\det \Sigma_t}} \exp\left(-\frac{1}{2}(\xi_t - \mu_t)^T \Sigma_t^{-1}(\xi_t - \mu_t)\right) \quad (46)$$

The above parameterization is referred to as moment form or the standard form of the Gaussian density function.

Now we derive the canonical form of the normal density function

$$p(\xi_t) = N(\xi_t, \mu_t, \Sigma_t) \quad (47)$$

$$= \frac{1}{(2\pi)^{n/2}\sqrt{\det \Sigma_t}} \exp\left(-\frac{1}{2}(\xi_t - \mu_t)^T \Sigma_t^{-1}(\xi_t - \mu_t)\right) \quad (48)$$

$$= \frac{1}{(2\pi)^{n/2}\sqrt{\det \Sigma_t}} \exp\left(-\frac{1}{2}\xi_t^T \Sigma_t^{-1} \xi_t + \xi_t^T \Sigma_t^{-1} \mu_t - -\frac{1}{2}\mu_t^T \Sigma_t^{-1} \mu_t\right) \quad (49)$$

$$= \frac{\exp\left(-\frac{1}{2}\mu^T \Sigma_t^{-1} \mu\right)}{(2\pi)^{n/2}\sqrt{\det \Sigma_t}} \exp\left(-\frac{1}{2}\xi_t^T \Sigma_t^{-1} \xi_t + \xi_t^T \Sigma_t^{-1} \mu_t\right) \quad (50)$$

This can be compactly written as

$$N^{-1}(\xi_t; \nu, \Lambda) \equiv \frac{\exp\left(-\frac{1}{2}\nu_t^T \Lambda_t^{-1} \nu_t\right)}{(2\pi)^{n/2}\sqrt{\det \Lambda_t^{-1}}} \exp\left(-\frac{1}{2}\xi_t^T \Lambda_t \xi_t + \xi_t^T \nu_t\right) \quad (51)$$

Equation (3) is the canonical (or normal) form of the density function with

$$\nu_t = \Sigma_t^{-1} \mu_t \quad (52)$$
$$\Lambda_t = \Sigma_t^{-1} \quad (53)$$

Properties

1. Conditioning is easy in canonical form.
2. Merginalization is easy in the standard form.

CHAPTER III

A REAL TIME MONOCULAR SLAM SYSTEM

Introduction

A single camera is a viable sensor choice in applications where the weight and power budget of the platform is very limited such as autonomous navigation of a small unmanned aerial vehicle (UAV) [4] in a GPS denied environment or augmented reality applications on a handheld device. Monocular SLAM also has the advantage of being effective in differently scaled environments such as an indoor office environment or a large scale outdoor environment. Range sensors such as depth camera can only provide reliable measurements in a small depth range and not suitable for large outdoor environment. However, the pure projective nature of a single camera makes monocular SLAM system more challenging as the depth of the image features cannot be measured from a single frame. Depth needs to be inferred by means of camera motion and feature observations from different viewpoints with sufficient parallax, which is the angle between the captured rays from a three dimensional feature to the optic center of the camera. Another challenge with monocular SLAM system is the scale drift. Scale of locally constructed map and the corresponding motion estimation tend to drift over time because of gauge freedom [17]. Moreover, the metric scale cannot be measured with a single camera and additional information source is required to recover the scale [13, 36].

Literature Review

Most early approaches on monocular SLAM systems [13, 14, 15] used sequential filtering methods where the state of the system consists of the current camera pose and the map features. This state is continuously updated from the measurements by updating the joint probability distribution of the camera pose and the feature parameters. In filtering approaches, previous camera poses are marginalized out and the features that are required for pose estimations are retained. The pose marginalization creates new links between the features that are connected to the pose whose joint probabilities are then updated. Propagation of joint probability distribution is computationally expensive for a large number of features and the number of features that can be stored in the map is limited. More recent works [6, 7] adopted the key frame plus optimization methods, essentially the well-known bundle adjustment methods that incorporate a set of previous poses along with their associated landmarks in the optimization. As shown in [11], that the ability of optimization methods to efficiently incorporate a large number of measurements provides better accuracy compared to filter based methods for the same computational work. Among the monocular visual SLAM systems proposed in the literature, PTAM [6] remains most popular because of its real time performance in accurate tracking and map building. PTAM decouples the map creation from tracking to achieve constant time frame rate tracking. The map creation and optimization are performed in a separate thread that can run at a slower rate. The system uses a FAST [5] corner detector and 8x8 pixel patches at different image scales which are matched using zero mean SSD. PTAM was designed for augmented reality applications in a small work place and proven to be very successful in reliable tracking and map generation even with

erratic, loopy motion of the camera. However, the SLAM performance degrades as the map progressively becomes expensive and full map optimization is no longer practical given the time constraint. In order to tackle the scale issue of the PTAM, Strasdat et al. [10] proposed a graph based optimization scheme that uses a two region approach to achieve constant time optimization. The inner region uses the pose-point correspondences in the pure BA sense whereas the outer region establishes pose-pose constraints by marginalization of the landmark points.

Decoupling of the tracking and mapping into two different threads allows the mapping to run at a lower frame rate than the tracking. The camera pose can be tracked at a high frame rate from the map points where the map expansion and optimization run in the backend. However, the limitation of this formulation is exposed during rapid exploration of unknown environment.

As monocular SLAM systems suffer from scale drifts for large loops, loop closure detection is an important part of the accuracy of the SLAM system. In most systems the loop closure detection is performed using appearance only [27, 25, 65] SLAM framework. The appearance based methods use scale invariant strong features such as SIFT [31] or SURF [32] descriptors in a bag of words scheme to find matching between the current frame and a previously generated frame. Recently loop closure detection methods based on binary descriptors such as ORB [46] or BRIEF [47] have been proposed in literature. We also employ an appearance based loop closure detection method where we use the same BRISK [17] descriptor that allow us to integrate the loop closure method in the algorithm with no additional cost of descriptor generation and storage for loop closure.

Loop Closure Detection

The ability for a robot to recognize an already visited place is important for longer duration map generation. When a robot visits a previously mapped area, detection of the already mapped region a.k.a. loop closure detection provides accurate data association. As a result, loop closure detection thereby reduces drifts and uncertainties in robot navigation and mapping by generating consistent map. Most recent loop closure detection algorithms are appearance based algorithms [26- 30] that require little to no a priori knowledge of the robots position. In the appearance based scheme, the entire image is represented as an observation and the loops are detected on the basis of image similarity. The basic idea is to create a database from the images during exploration of the robot so that the most similar image can be retrieved when the robot visits already mapped region. Similar to content based image retrieval, most appearance based algorithms use a visual Bag of Words (BOW) approach for image similarity measurement. In BOW approach, the images are represented as a set of visual features taken from a dictionary. This visual dictionary is built by clustering the sets of visual features into a collection of generalized visual features or visual words. Then the images are represented by a histogram of occurrences of each visual word in the images. Visual resemblance between the current image and a previous image in the database is quantified by measuring the similarity of their corresponding histograms of visual words.

Bag of Words methods were initially developed for object recognition and content-based image retrieval. In their seminal work, Sivic and Zisserman [28] used Bag of Words for detecting similar scenes in video sequences. SIFT descriptors were extracted from a set of training images and then clustered using k means algorithm to generate the visual

vocabulary. When a new image arrives, its descriptors are quantized using these visual words and then it's similarity with previous images was computed using a Term Frequency Inverse Document Frequency (TF-IDF) weighting. Nister and Stewenius [29] improve the computational efficiency of the vocabulary building process by implementing a vocabulary tree based on a hierarchical k means approach thereby allowing large training dataset.

One of the most popular loop detection method is the FAB-Map system [19] that performs loop detection in trajectories 70 km and 1000 km in length with 48.4% and 3.1% recall respectively, and with no false positives. In this method, the images are represented as a Bag of Words, and the words' co-visibility probabilities are learnt offline using a Chow Liu Tree. However, robustness of the loop detection decreases when the images depict very similar structures for a long time, which can be the case when using frontal cameras [36]. Galvez-Lopez and Tordos [23] proposed a loop detection method based on binary feature descriptors such as BRIEF and ORB for fast image matching. Their algorithm uses the hierarchical BOW model proposed by Nister [29] along with a direct indexing for fast descriptor matching. A temporal consistency check and a geometrical check are performed to increase the reliability of the loop detection.

Some authors have also proposed [27] loop detection methods based on online vocabulary generation. In the work of Angeli [27], two visual vocabularies (for appearance and color) are created online in an incremental fashion. The two BOW representations are used together as input of a Bayesian filter that estimates the detection probability.

Organization

In this chapter we present our real time monocular SLAM algorithm. Our algorithm differs from the previous methods in terms of both efficiency and reliability. The algorithm process flow as well as parallel implementation of the time critical components makes the algorithm computationally very efficient without compromising the reliability of the system. We demonstrated the effectiveness of the algorithm on challenging long duration visual odometry data set where the front facing camera mounted on a vehicle. The description of the algorithm is organized as follows: we start with a brief review of the binary descriptor namely BRISK descriptor used in our work. In the following sections we present the algorithm and provide detailed descriptions of the basic building blocks.

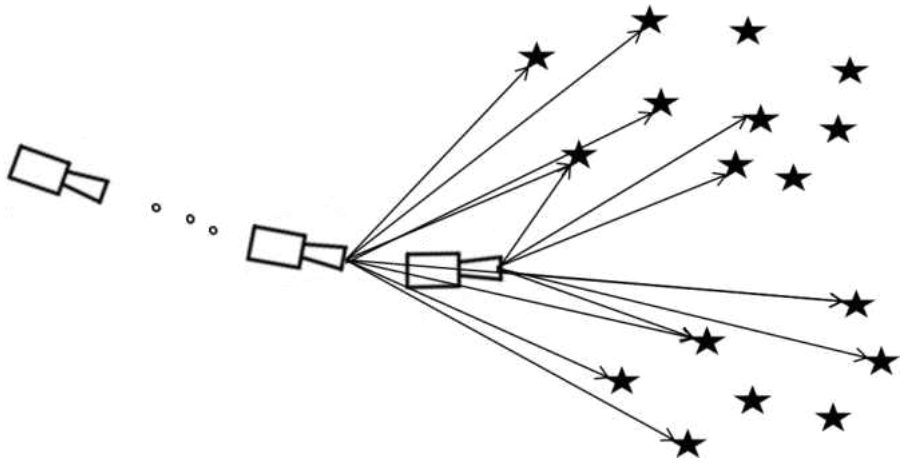


Figure 6: Monocular frames and 3D points. The algorithm estimates the pose of the frames and the positions of the 3D points at the same time, using image projections.

Brisk Feature Descriptor

In this section, we present a brief review of the BRISK[17] approach for feature description and matching. The original implementation of the BRISK uses the AGAST

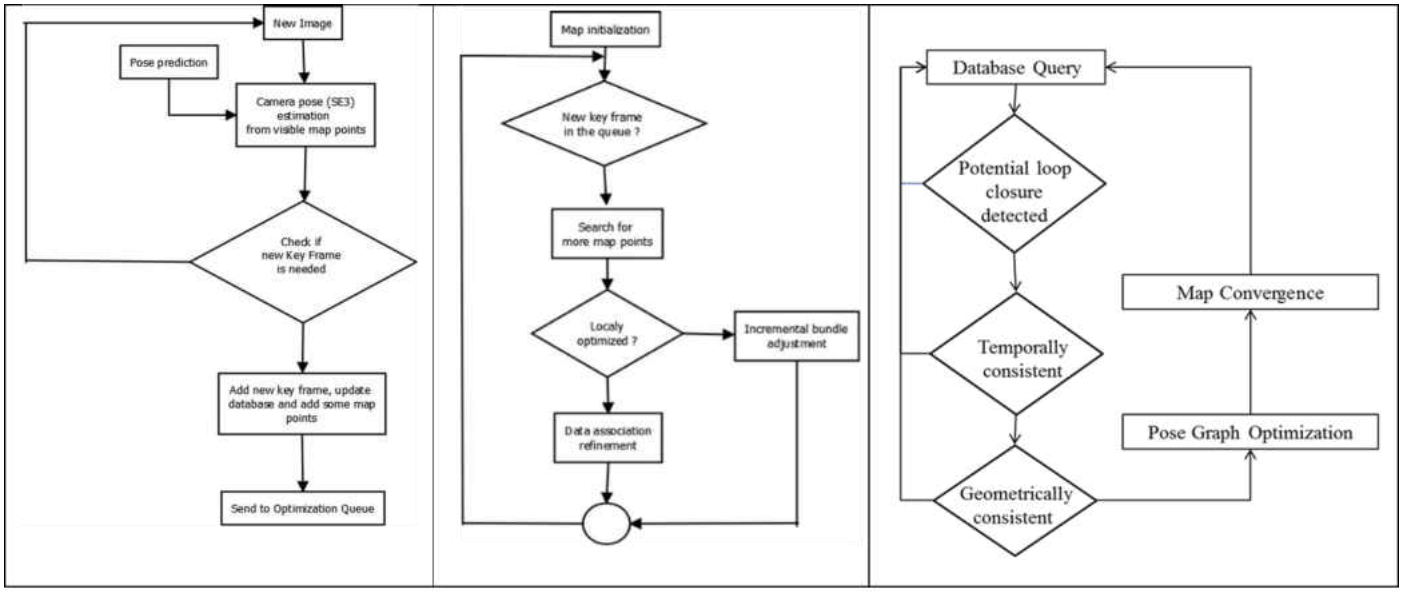
[39] corner detector which is an extension of the FAST [5] corner detection. However, after experimenting with both AGAST and FAST corner point detection, it was found that detection time decreased significantly with the FAST corner detection compared to AGAST detector. As a result our implementation of feature detection uses the FAST corner point detection. The scale invariant is achieved by detecting salient points at different levels in a scale space pyramid, performing non-maxima suppression and then interpolation across scales in the pyramid. The descriptor is built from a set of pair wise brightness comparisons and the results of these comparisons are stored as a binary string. Each bit in the binary string is the result of exactly one comparison. The feature description is performed using a symmetric pattern where sample points are positioned in concentric circles surrounding the detected salient point. Intensity at each sample point in the pattern is obtained after applying Gaussian smoothing in its neighborhood pixels. The kernel size of the Gaussian smoothing is proportional to the distance from the corner point. The descriptor uses two types of sample point pairs for comparison in order to estimate the dominant orientation of the corner point and the descriptor building. A set of long distance sample point comparisons is used for the orientation estimation. For each long distance comparison, the gradient is estimated by computing the vector displacement between the two sample points in the pair and weighted by the relative difference in intensity. The set of the long distance gradients are then averaged to deterring the dominant orientation of the corner point.

After determining the dominant direction, the sampling pattern is then scaled and rotated before the descriptor is created from a set of short distance sample point pair

comparisons. The resultant bit string consists of 512 bits representing local gradients and shape in the patch.

The SLAM Algorithm

The basic algorithm can be divided into 3 major parts: The tracker and part of mapper at the front end, the backend optimization and the loop closure. In order to make the algorithm computationally efficient, we make full use of the multicore processor by parallelizing the tasks when possible. The algorithm is shown in Figure 7. Bellow we explain the major components of the SLAM system.



Frontend Tracking and Mapping Backend Map Refinement and Optimization Loop Closure Detection and Correction

Figure 7: SLAM algorithm process flow. The algorithm is divided into three parallel flows. The camera pose estimation from map points tracking and the map exploration is performed in the main process. The pose graph optimization is performed in the backend in a separate process. The loop closure detection and correction process runs in a separate thread at a lower frequency.

3D Map Representation

The 3D map is represented by a 3D point cloud along with a set of camera poses called key frames. The 3D points cloud map consists of the collection of feature points M located in the world coordinate system W. Each map point stores it 3D world coordinate

m^W, its own index, a reference to the source key frame where it was first detected along with the indices of the detector and descriptor in the source frame. Each map point also stores a list of key frames where it was successfully matched. The map also contains the list of key frames. Each key frame stores the vectors of the key points and their descriptors computed from all levels in the image pyramid. Camera pose associated with each key frame is represented as a coordinate transformation between the World and Camera coordinate systems as T_C^W. Each key frame also stores the list of indices of the map points that are visible in that key frame along with their corresponding image positions. In addition, each key frame stores a list of neighboring key frames based on scene overlap defined by the map points co-visibility.

Map Initialization

Map initialization starts with running the key points detection and BRISK descriptor generation in the first image frame. This image frame constitutes the first key frame in the map. The key frame stores the feature points and descriptor vectors along with the initial camera pose relative to the world coordinate system. A subset of detected key points and their descriptors are selected for tracking. A quad tree method similar to Mei et al. [66] is used to ensure that the selected feature points are uniformly distributed. This set of feature points is then tracked in the image sequences until there is sufficient base line between the current image frame and the first frame. This is done by computing the 8-point fundamental matrix [24] with RANSAC [42] scheme between the two frames along with a heuristic.

When a new image arrives, its key point detection and descriptor generation is performed first. Then for each candidate point, a search region is created around the last detected

position in the image and only a subset of feature points in the current frame are selected for feature matching that lie within the search region. This step greatly improves the tracking efficiency by reducing the number of false matching as well as computation costs. After feature matching is performed, the feature points' current positions are updated to the new image location and a new search region is created for the next frame. As a result we have a 2D-2D correspondence set between the first image frame and the current frame. This set of correspondences is then used to compute a fundamental matrix with RANSAC scheme. This fundamental matrix is used for both outlier rejection and also to determine if there is sufficient base line between the first frame and current frame. We reject the matching if it fails the epipolar constraint test. Then an essential matrix is computed from the fundamental matrix using the camera intrinsic parameters. This essential matrix is then used to compute the transformation matrix up to a scale between the first and current frame. Then 3D positions of the feature points in the inliers list are estimated using triangulation. The 3D position is considered to be valid if it is in front of both the first frame and the current frame and the re-projection error is below a pre-defined threshold. We also disregard any point which is located very far by computing the difference between the pixel positions in two images. Experiments show that a pixel distance of 8 is a sufficient distance for generating valid 3D position. A point in the inliers list is considered valid when both conditions are met. We conclude that a sufficient baseline is achieved when at least 60% of the inlier points are valid. At this point the current frame is considered as the second key frame. These two key frames along with the valid points are used to initialize the map.

Tracker for Camera Localization

The tracker is responsible for estimating the camera pose from the successfully tracked map points. Every 3D landmark in the environment is associated with a BRISK descriptor that is used to search for the landmark in the environment by performing descriptor matching in the current image.

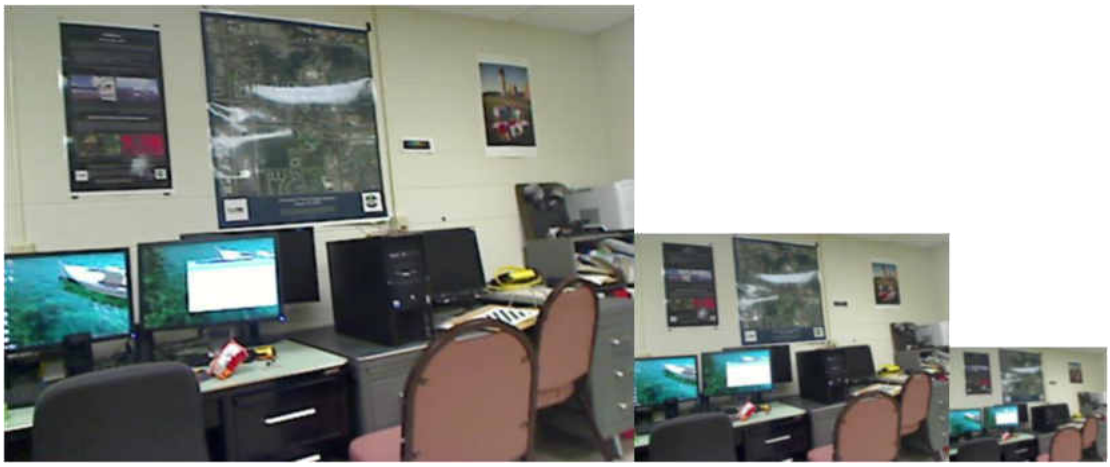


Figure 8: Image pyramid construction by successive half sampling of the image. Half sampling is done by taking an average of 4 neighboring pixels.

When a new image arrives, first a three layers image pyramid is constructed by successive half sampling the image. Fig 3 demonstrates the image pyramid generation. Then the corner points are detected and their descriptors are generated at each level in the image pyramid. At this point we take full advantage of the parallel processing capability of multicore processor to speed up the detector and descriptor generation process. Since our algorithm is designed for small platforms with single board computers, which generally do not have a powerful GPU, the parallel processing implementation is done on CPU. The process is shown in Fig 9. A parallel multithreaded process is developed where a separate thread is used to perform detector and descriptor generation for each pyramid

level. This is done using the open source framework of Intel's threading building block (TBB) [67]. At the end of the operations, we combine the detectors and descriptors from separate pyramid level.

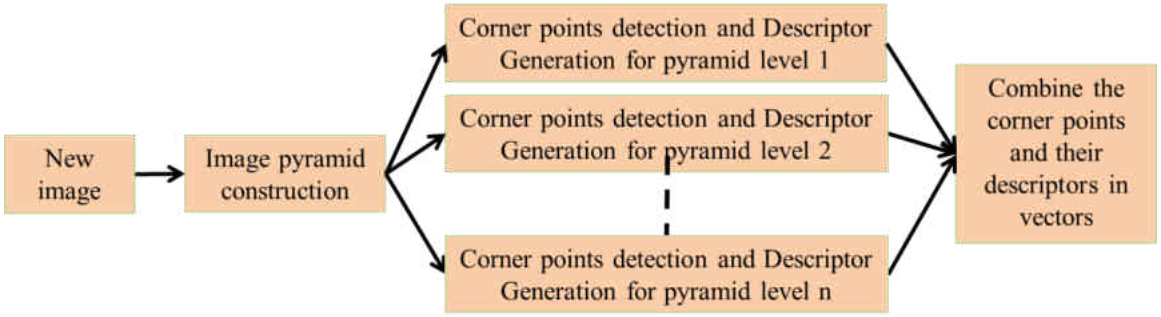


Figure 9: Parallel detector and descriptor generation. A separate thread is used to perform the feature points detection and descriptor generation for each scale

This set of corner points is then matched to the stored map points to generate 3D-2D correspondences between the 3D landmarks and their associated 2D positions in the current image. The camera pose is then updated from the 3D-2D matches. However, instead of searching for all map points in the current image, we limit the search for the map points that are most probable to be visible in the current image.

Given the predicted camera pose from the prediction model, a subset of the potentially visible map points is selected. This subset is selected from a number of key frames that are closest to the current predicted camera pose. We have used 30 key frames in our implementation for selecting the potential visible set of map points. Now for each map point in the potential visible set, a neighborhood region is created around the predicted image position in the current frame that is obtained from the projection model, and a subset of corner points are selected that lie inside the region. These corner points are then compared with the map point by performing the descriptor matching. Similar to the

detector and descriptor generation, we again parallelize of the map points matching. The number of potentially visible map points is divided into a predefined number of sets and then for each set a separate thread is used for map points matching. Again we use the Intel's threading building block framework for the parallel map points matching. In order to avoid any memory sharing, each thread gets a copy of the detector and descriptor vectors for the current image. This causes increased memory consumption by the threads; however, the compact nature of the binary descriptors limits the cost in memory usage. In addition, the computational efficiency far outweighs the additional memory consumption. This step allows for performing real time tracking even in a computationally constraint system. For each successful match, a 3D-2D correspondence is stored for camera pose update. Example of feature points matching between two images is shown in Figure 10.

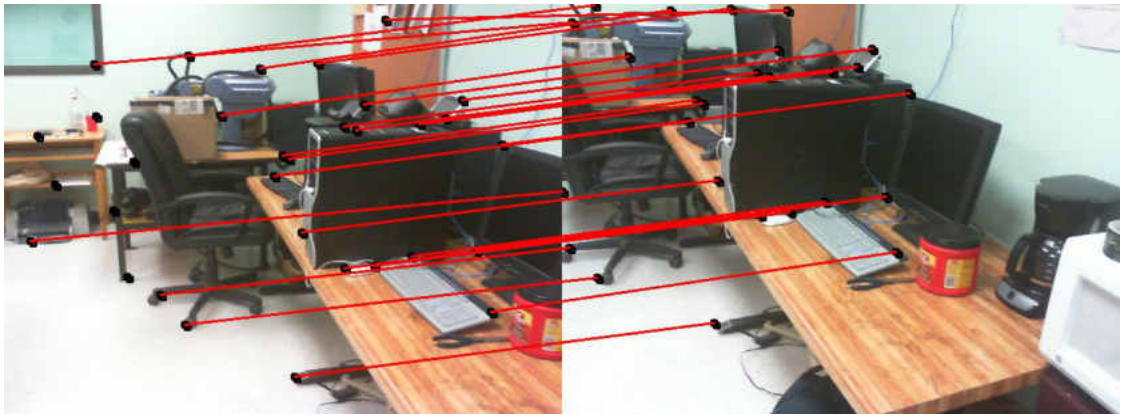


Figure 10: Feature points matching between 2 image frames.

At the end of map point tracking, we obtained a set of 3D-2D correspondences between the 3D world positions x_k and corresponding image position u_k. This set of correspondences is used to update the camera pose using the numerical optimization algorithm explained in section II. Essentially, the camera pose is iteratively optimized by minimizing a cost function

$$f = \sum_k (e_k^T \Lambda e_k) \tag{56}$$

with respect to the pose T_i. Here e_k is the feature prediction error

$$e_k = u_k - proj(T_i, x_k). \tag{57}$$

Accuracy of the camera pose estimation using the numerical optimization methods is affected greatly in the presence of outliers. For example, an incorrect feature matching will cause a false 3D-2D correspondence. The presence of outliers can cause the minimization to incorrectly converge to local minima.

One way to handle the outliers is the use of robustifier [19] where the quadratic cost function is replaced by a robust kernel $\rho(.)$ with larger error terms have less influence on the overall cost. In this work a Huber kernel [24] is used as it is convex in nature. However, it still does not guarantee global optimization since the effectiveness of the robustification depends on the error model which is often unknown.

In order to increase the accuracy of the pose estimation and remove the map points with false correspondences, we employ an efficient P3P [37] with RANSAC scheme to generate candidate 3D-2D point correspondences before final pose estimation with nonlinear optimization. In the RANSAC scheme, 4 points correspondences are randomly selected to get an initial estimate of the camera pose relative to the world. In this method, 3 points are used to generate four possible solutions and the 4th 3D-2D correspondence is then used for disambiguation by computing re-projection error and selecting the solution with the minimum value. This estimate is then used to find the number of inliers by

computing their re-projection errors. The camera pose with highest number of inliers is our candidate pose for optimization with the point correspondences in the inliers list.

Key frame Inclusion and Map Expansion

The initial map consists of the first two key frames and a small set of 3D map points. As the camera starts to explore new places, the map is expanded by adding new key frames and map features.

A new key frame is added to the map when the distance of the current camera from the closest key frame exceeds a minimum distance threshold. Minimum distance threshold is computed as a weighted combination of linear and angular distances. The linear distance depends on the mean depth of the observed features where we use an absolute angular distance. The key frame is initialized with the current camera pose along with the key points and descriptor vectors of the current image frame. The key frame also stores the indices of the map points that are successfully tracked along with their current image position. Now, in order to generate new map points, current frame feature points are matched with feature points in other key frames in the map. This is done by performing epipolar search for feature points in the other frames and then using triangulation to generate 3D position.

First a set of neighboring key frames are selected based on their distance to the current position and the key frames from this subset are chosen for new map points generation that satisfy the linear and angular distance threshold. Essentially we select a set of neighboring key frames that satisfy the minimum distance threshold and also share co-visibility with the current frame. A set of new feature points that are not the observations

for the existing map features are potential candidates for new map features. The key points are then searched in the other key frames. For each feature point a set of feature points that are within a certain distance along the epipolar line in the other key frame are selected for descriptor matching. If more than one match is found, then the point is discarded as not discriminative enough. Successfully matched points are used in triangulation to find their 3D position and added to the map.

Previous methods such as PTAM use a separate thread for map expansion where they allowed new key frame addition along with map point generation to run at a slower rate than the tracker. However, during rapid exploration of the robot, the map expansion needs to keep pace with the robot motion to avoid tracking failure. We use a heterogeneous method where new key frame addition to the map along with partial new map points generation are performed in the same process as the tracker when the condition for new key frame addition becomes true. However, the downside is that it adds significant computational burden on the front end tracker. In order to tackle the issue of computational cost, we divide the new map points generation to both frontend and backend. We select the key frame that shares the highest co-visibility and also satisfy the minimum distance threshold for map point generation in the frontend. We take advantage of the multicore processor to parallelize the task at hand. We use a combination of parallel task and parallel data framework for generating new map points. Figure 11 shows the process of new key frame addition and map points generation.

The total number of key points that are our potential candidates for map point generation is divided into a predefined number of blocks where each block consists of a fixed number of points and we keep a list of their original indexing. Each block uses a separate

thread for feature matching and triangulation. Once the computation in each thread is finished they are combined together before the successfully generated points are inserted into the map.

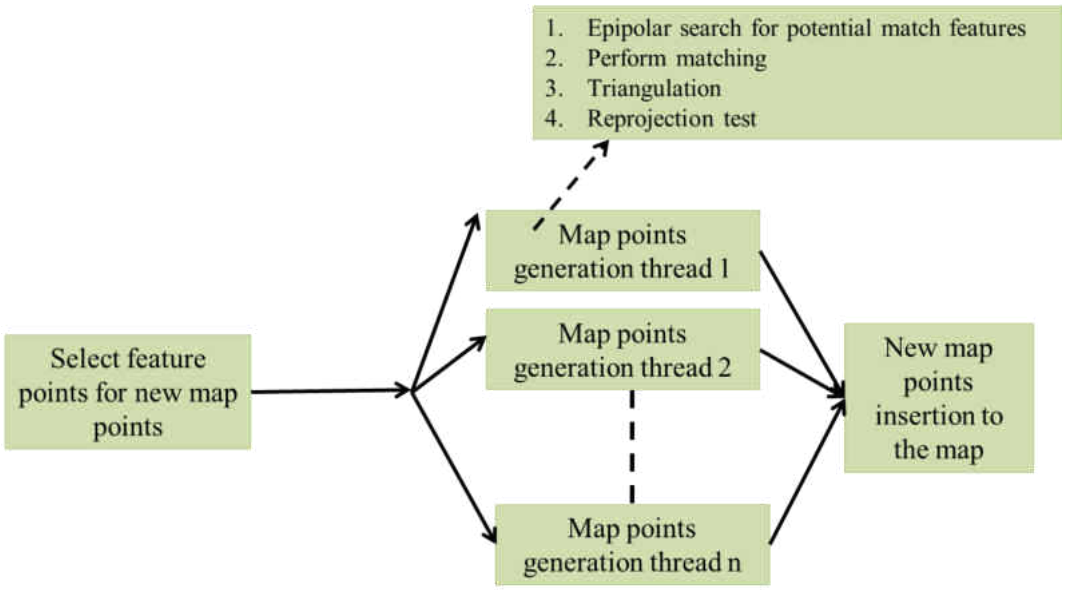


Figure 11: Parallel map points generation. The set of feature points are divided into a fixed number of packets. For each packet a separate thread is used for map points generation.

Data Association Refinement

In this stage we add new map points and also search for data association in the neighboring key frames. A set of key frames are selected that satisfy the minimum baseline and co-visibility threshold and also were not being selected in the frontend. New map points are generated using the method described previously. Since new map points are generated using triangulation between two key frames, it is possible that they are also visible in other key frames in the neighborhood of the two key frames. A large neighborhood of 12 key frames is selected to search for measurements of these new points. If successful matches are found, the new points are added to the list of map points visible from the key frames along with their corresponding image locations.

Constant Time Back End Bundle Adjustment

As the robot explores new places over time, the map size gets larger and the number of parameters namely the key frames and the land marks keep constantly increasing. Thus if were to perform full bundle adjustment every time the map expands, the computational cost would become unbounded as it increases linear to cubic in complexity with the number of parameters. As we are interested in real time performance, it is important to ensure that the cost of performing a single iteration does not exceed a certain threshold.

Essentially we need to limit the number of parameters in the optimization. In a pure visual odometry sense, the natural way to achieve this is to consider last n key frames in the optimization in a sliding window [68] mode along with the map points visible in these key frames. However, complexity arises when selecting map points for optimization. In order to understand the problem, let us consider a situation where a map point x has its source key frame that is not included in the sliding window for optimization. In addition, this map point may be viewed in only one or two key frames in the current sliding window whereas it is visible in more key (e.g. 8) frames which are outside the sliding window. When we include all the observations, the likelihood of accurate triangulation in the optimization is very high. On the other hand, if we consider only one or two key frames that are in the active window, triangulation accuracy will degrade since it is only weakly constrained by the one or two active key frames in the sliding window.

In order to address the issue, we implement a graph based incremental optimization method that is motivated by the method used in [68]. The sliding window consists of an ordered sequence of key frames. Every time a local bundle adjustment is performed, the index of the sequence is updated once and a new key frame is added while removing the

key frame with the lowest index. Now we define a set K_1 as the active frames that include the key frames in the sliding window. K_1 also includes additional key frames that share significant scene overlap with the key frames in the sliding window. This scene overlap is determined by co-visibility of map points. The construction of the graph is explained as follows: Each map point consists of a list of key frames where it is visible along with the source key frame where it was first generated. For each map points that are visible in the sliding window key frames, we get the list of key frames that are not in the sliding window. Then we determine if a key frame in this list shares overlapping scene region with any key frame in the sliding window by computing the co-visibility. Co-visibility is computed by counting the number of map points that are shared by both key frames. If the number is above a predefined threshold, we conclude that the key frames share sufficient overlapping region. In that case, we include the key frame in the active frames list. Else the key frame is included in a second list denoted as K_2. The key frames in the set K_2 are set as fixed which ensures that the current local map is being anchored to the previous map.

The resultant cost function consists of the weighted re-projection errors e_r

$$\chi^2(\mathrm{X}) = \sum_{k \in K} \sum_{m \in M} e_r^{m,k\,T} W_r^{m,k} e_r^{m,k} \tag{58}$$

Where k denotes the key frame index, and m denotes the landmark index. $W_r^{m,k}$ denotes the information matrix of the measurements of landmark m in the key frame k.

This local bundle adjustment is performed in a separate thread in the backend along with the data association update and can be run at a slower rate than the frontend tracking and mapping.

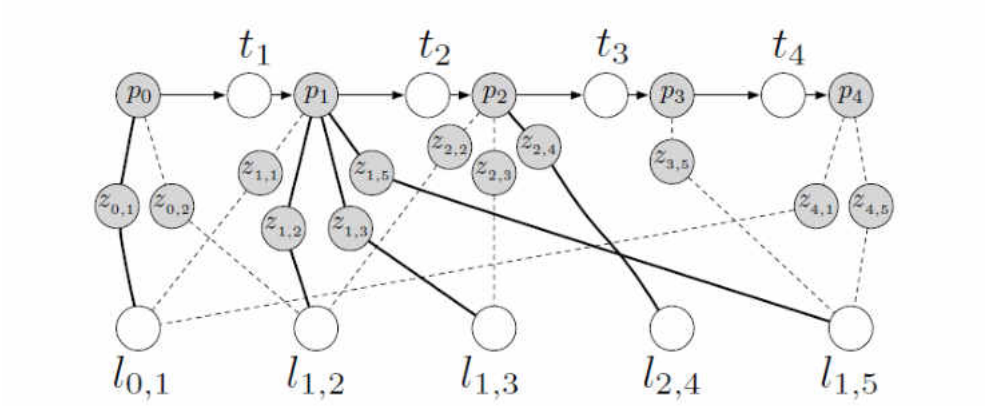


Figure 12: An example of camera poses and 3D landmarks configuration for incremental Bundle Adjustment

Loop Closure Detection and Pose Optimization

We use a similar approach to [23] for our vocabulary building, database generation and image query. The open source version of their implementation is modified to use BRISK descriptors for vocabulary generation from training images and then online database generation using the key frames generated from the SLAM algorithm. The basic steps are briefly described in the following sections:

Vocabulary Building

The vocabulary is generated offline from a set of training images. This training set is created independently from a scene which is similar to the environment where the robot navigation and SLAM generation occurs. First a set of BRISK descriptor vectors are extracted from the training images and then the descriptor space is discretized into a set of visual words W. Following the method described in the re-localization step, the vocabulary is structured as a tree. As a result we get a tree with W leaves, which are the words of the vocabulary. Each word is assigned a weight depending on its relevance, decreasing the weight of the words that are very frequent and thus less discriminative.

This is done by using the “term frequency- inverse document frequency (TF-IDF)” method described in [29].

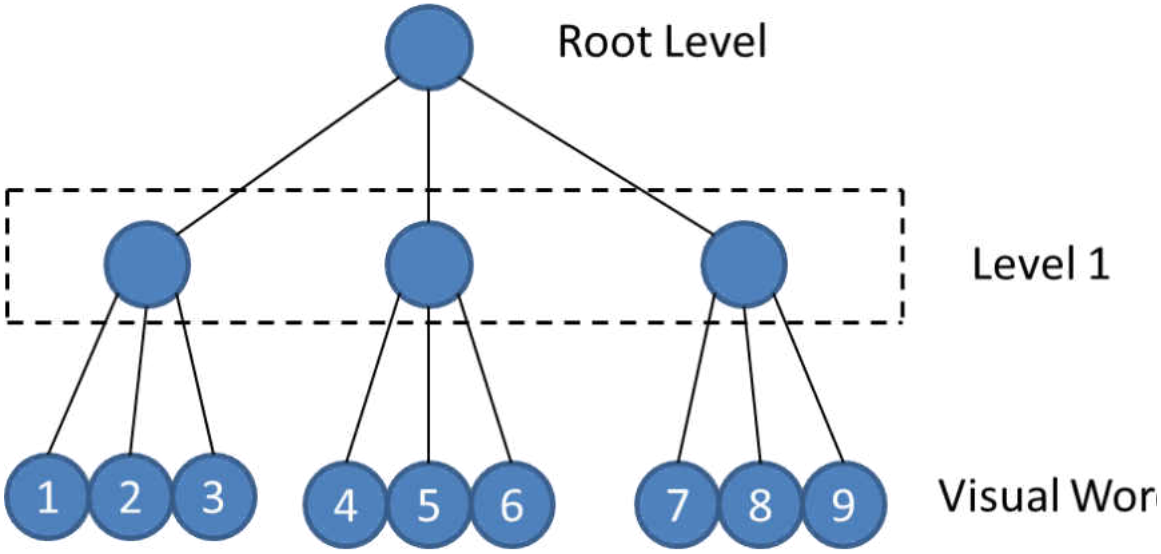


Figure 13: Vocabulary tree generation. Each word contains its descriptor along with a list of frames where it is visible. It also stores the frequency score of the word based on how frequent it is visible.

Database Generation

In order to perform the loop closure, we use an image database of hierarchical BoW of the key frame descriptors. Every time a new key frame is added to the map, the descriptor vector for the key frame is converted into a BoW vector $v_t \in \boldsymbol{R}^W$ by traversing the vocabulary tree from root to the leaves and by selecting at each level, the intermediate nodes that minimize the Hamming distance. In addition to the BoW vector, an inverted index is also maintained. For every word w_i in the vocabulary, a list of key frame indices where the word w_i is visible is stored in the inverted index structure. In addition, a list of neighboring key frame indices is also stored. This neighborhood is generated based on map point co-visibility. As mentioned in section 3, each key frame stores the list of map points indices that are visible in that key frame. Based on this list, we find a set of key

frames that share common map points. When the number of shared map points between the current key frame and another key frame is over a predefined threshold, we include the other key frame index in the neighborhood list. The neighborhood list of the other key frame is also updated. Similarity between two BoW vectors v_1 and v_2 is measured by calculating the L_1 score $s(v_1, v_2) \in [0,...,1]$ [22] as

$$s(v_1, v_2) = 1 - \frac{1}{2}\left|\frac{v_1}{|v_1|} - \frac{v_2}{|v_2|}\right| \tag{59}$$

Loop Closure Detection

A multi stage detection method has been implemented that protects against false loop closing since a erroneous loop closure will result in a topologically incorrect map which in turn will cause failure in the localization and map building.

When a new key frame is added to the map, its descriptor vector is converted to the BoW vector and added to the database. We compute the neighborhood list as explained above. Then we perform the similarity measure between the BoW vectors of the current frame and the frames in the neighborhood and store the minimum similarity score. Then the rest of the BoW vectors in the database that are not in the neighborhood list are searched for the matching candidate. However, the loop candidate searching does not start until the robot travels certain duration of time.

Once the database is searched, we have an ordered list of matching candidates based on the similarity score that are not in the neighborhood of the current frame. Now the key frame indices with the similarity score greater than the minimum neighborhood score for the current key frame are considered as potential loop candidates and subject to further

computation. If no indices are found with similarity score greater than the minimum neighborhood score, we consider that no loop candidate is detected.

Temporal Consistency Check. If a potential loop candidate is found, then then temporal and geometric consistency check is performed in order to ensure that no false loop candidates are added to the map for optimization. Temporal consistency is performed by creating clusters of topologically related loop closing hypotheses where loop closure sequences that relate similar positions of trajectory are grouped together. We perform a simple incremental way of clustering based on the time of loop closure candidate detection. With the first loop closure candidate that arrives, we initialize the first cluster, and wait for next few loop candidate search. If there is more than one potential loop closure candidate is found, their temporal consistency is checked. If a candidate frame is not in the neighborhood of the other loop closure candidates, then that frame is stored in its own cluster. When the next loop closure candidate arrives, we check its temporal consistency by performing the neighborhood search. If the new loop candidate frame is in the neighborhood of the frames in a cluster, it belongs to that cluster. Otherwise, it is stored in its own cluster. If we don't get any loop candidate for a cluster in the next 2 consecutive searches, the cluster is removed and the loop candidate is considered as false loop detection. A new cluster needs to be initialized for the next loop closure candidate. A cluster is to be considered as a valid loop candidate cluster if it contains at least 3 potential loop closure candidates.

Geometric Consistency Check. Once we have a cluster with the minimum number of loop candidate frames required for temporal consistency, we perform the geometric consistency check between the search frames and their corresponding loop candidate

frames. The geometric consistency check consists of computing a Fundamental matrix with RANSAC between the search frame I_t and the loop frame L_t with number of inliers at least 15. For all the map points in the search frame I_t, their corresponding image positions in loop frame are searched by performing feature matching between the map point's descriptor with the descriptors in the loop frame. We perform an exhaustive search to find all the map points in the loop frame. Now we have a set of image point correspondences between the search frame and loop frame with outliers. Then the fundamental matrix is computed with RANSAC from the point correspondences and the number of inliers is counted. If the number of inliers is over the threshold, the frames are considered as geometrically consistent.

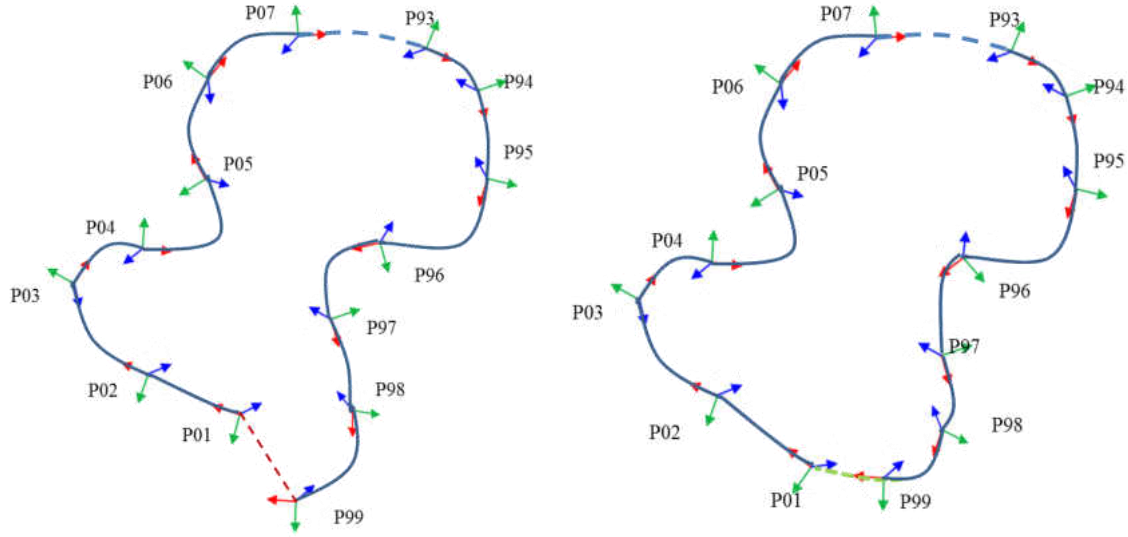
Once a geometric consistency is found between two frames, the fundamental matrix is used to perform correspondence search between two sets of map points in the frames I_t and L_t. For all the map points in the search frame I_t we search for their corresponding map points in the loop frame using the epipolar constraints. Essentially we find a set of 3D-3D correspondences between the two frames. The 3D-3D correspondences are then used to find a similarity transform between the frames using the method described in [41].

Graph Based Loop Closure

Once we have a loop closure candidate after successful temporal and geometric consistency check, the loop correction problem can be formulated as a large bundle adjustment problem since we have to update over all map points and camera poses that are in the loop. However, optimization over a large number of frames and map points is

computationally expensive. Another major issue is that bundle adjustment is not a convex problem. The further the robot travels, the positional uncertainty increases and it is possible that bundle adjustment will get stuck in local minima.

In order to solve the problem, we adopt the reduced form of the BA problem namely pose graph based optimization [40]. Instead of solving for all the parameters in a full SLAM problem, graph based SLAM methods solve for a sparse set of relative pose constraints by marginalizing out the landmark parameters onto pose parameters. This leads to a pose graph as shown in Figure 19.



a. Pose Graph without loop Correction

a. Pose Graph after loop Correction

Figure 14: Example trajectory starting at the first frame P01 and ending at P99. The loop closure between frame 1 and frame 99 adds an extra edge to the graph. Loop correction generates the set of pose configurations that minimizes the cost function in the nonlinear optimization

Let $T \in \{T_i,...T_n\}$ be a set of parameters of poses and consider two poses from the set as T_i and T_j. The relative constraint between the initial estimates of T_i and T_j is calculated as $T_{ji} = T_j T_i^{-1}$. The constraint generated from loop closure candidate is computed as described in the previous section. In the pose graph SLAM formulation, these relative constrained are regarded as virtual measurements. Now the idea is to optimize the poses

T_i,.....T_n in such a way that the pose concatenations $T_{ji}T_iT_j^{-1}$ are as close to identity as possible. Initially the sequential pose concatenations $T_{ji}T_iT_j^{-1}$ are set as identity except for the one containing the loop closure constraint. The purpose is to estimate an optimal configuration set of the poses that minimize the error over all constraints and hence the loop closes. Considering the residual error between two poses as d_{ij}, the cost function to minimize can be written as

$$\chi^2(T_1,....T_n) := \sum_{T_{ji}} d_{ij}^T \Lambda_{T_{ji}} d_{ij} \tag{60}$$

The information matrix can be set as identity or can be computed accurately by point marginalization similar to [25].

As the monocular SLAM systems suffer from scale drift over time, it is important to take scale into consideration when optimizing. So the optimization is performed based on 7 DOF similarity constraints $S_k \in SIM(3)$. In order to optimize over 7 DOF similarity space, each 6 DOF pose T_k and relative pose constraints T_{ij} are transformed into a similarity S_k and S_{ji}. This is done by adding a scale $s = 1$ for all the relative constraints except the loop closure constraint where the scale is computed as described before. The pose graph SLAM is then solved using the g2o [9] sparse Cholesky solver.

Once the poses are corrected from the optimization, the next step is to update the landmark positions. This is done by mapping each point relative to its corrected source frame as $x_j^{corr} = (S_k^{corr})^{-1}(T_k x_j)$. Afterwards, each similarity transform S_k^{corr} is transformed back to a rigid transform T_k^{corr} by setting the translation to st and leaving the rotation unchanged.

Relocalization

The tracking system relies on a motion model to predict the camera pose and then the predicted camera pose is used to limit the region to search for visual feature correspondences. As a result a rapid camera tracking is achieved. However, motion blur, sudden motion change or occlusion can cause the tracking system to fail and even corrupt the map. The typical solution is to implement a data driven detection of pose when the map already exists. Since linear matching becomes computationally very expensive when it comes to perform comparison against a large data set, usually the linear search algorithm is replaced with an approximate nearest neighbor search algorithm that offers significant speed up.

Most nearest neighbor algorithms for vector features use hierarchical decomposition of the search space and not readily suitable for binary features because of the assumption that the features lie in a vector space where the dimensions of the feature vectors can be continuously averaged. As a result most approximate nearest neighbor search algorithms for binary features matching proposed in literature are based on hashing methods such as locality sensitive hashing[45], semantic hashing[68], min hashing[69] etc. A hierarchical decomposition based binary features matching algorithm has been proposed in [44] where multiple hierarchical trees are constructed and searched in parallel during nearest neighbor search. Each tree is constructed by performing a hierarchical decomposition of the search space by successively clustering the input dataset where the cluster centers are randomly selected from the input points. Each non-leaf node contains a cluster center and the leaf nodes contain the input points that are to be matched. The algorithm was tested with BRIEF and ORB features and found to demonstrate significant improvements on the

search performances when multiple trees are used in parallel. However, building multiple trees requires considerable amount of computation power and serves as a bottleneck for low power systems.

In this work, we propose a novel algorithm for faster matching binary features and we implement our algorithm for the scale invariant BRISK feature descriptors. The algorithm is used for relocalization in the event of tracking failure in the SLAM algorithm.

Building the Tree

The tree building process starts with selecting N points from the input dataset. The number N represents a parameter of the algorithm which is the number of initial clusters created from all the points in the dataset with each point as a cluster center. Similar to [34], we also call N as branching factor. These N points are chosen using the k-means++ seeding [33]. This process is followed by assigning points to each center that closer to that center than any of the other centers. Since for binary descriptors, the average for the dimensions of the feature vectors cannot be computed, we used a voting scheme to refine the centroid of clusters. For each cluster c, we use an accumulator vector v of the length of the BRISK descriptor that holds the result of bit accumulation for all the descriptors that belong to the cluster. An element of the accumulator array is increased by one when the corresponding bit of the descriptor is one. At the end, the majority rule is used to set the bit of the cluster centroid. If the majority of the descriptors voted for 1 as the value for bit element v_i , then the corresponding cluster center bit takes 1, otherwise it takes 0. The algorithm is repeated recursively for each resulting clusters until the number of descriptors in each cluster goes below a certain threshold which is the maximum number of leaf nodes.

Table 1: Algorithm for Building the Tree

1. **Input**: descriptor dataset D
2. **Output**: hierarchical clustering tree
3. Parameters: branching factor N, maximum number of leafs $L_{\max}$
4. **if** size of $D < L_{\max}$, **then**
5. Create leaf nodes with the points in D
6. **else**
7. $P \leftarrow$ select N points with the kmeans++ seeding from D
8. $C \leftarrow$ cluster the points in D around the nearest centers P
9. $P' \leftarrow$ redefine the cluster centers with majority voting and re-associate descriptors to clusters C'
10. **for** each cluster $C'_i \in C'$ **do**
11. create not leaf node with ceter P'_i
12. recursively apply the algorithm to the centers P'_i
13. **end for**
14. **end if**

Searching for Nearest Neighbors

The searching method starts with a single traverse of the tree during which the node closest to the query descriptor is picked and recursively explored while the unexplored nodes are stored in a priority queue. The search ends when the number of points examined in the tree exceeds a threshold which is a parameter of the algorithm. Increasing the threshold increases the number of exact neighbors found with the search being more expensive.

Pose Estimation from Map Points

When sufficient matches between current image and the map points are found, a set of 3D-2D point correspondences are created and then the camera pose estimation is

performed using the efficient P3P algorithm with RANSAC scheme described in the previous section. Once estimated, the camera pose is then updated using the pose optimization method using the inliers only and the tracker starts again.

Experiments & Results

In this section we present the experimental results. With extensive testing, we demonstrate that the developed system is more efficient and capable of running on a low power computer carried by a small quadcopter. The accuracy of the generated 3D point cloud map and the camera trajectories are reasonable accurate and comparable to previously developed SLAM system with considerably faster performance. Tests on data collected during the flight of the quadcopter along with the real time flight tests demonstrate the algorithm's effectiveness.

Performance Evaluation on Indoor Dataset

In order to validate the effectiveness of the proposed algorithm, it is evaluated in both indoor and outdoor environment. Initially, we captured a few video sequences from different indoor environments with a camera mounted on the quadrotor. The images frames and the inertial sensor information are time stamped before storing on the disk drive. The camera used for the experiments is a standard Logitech webcam with image size 640x480. The images are captured at 25 fps. To evaluate the performance of the algorithm, the videos are generated by varying the camera movement from erratic, locally loopy motion to rapid sideways displacement for exploration and also backward and forward movement.

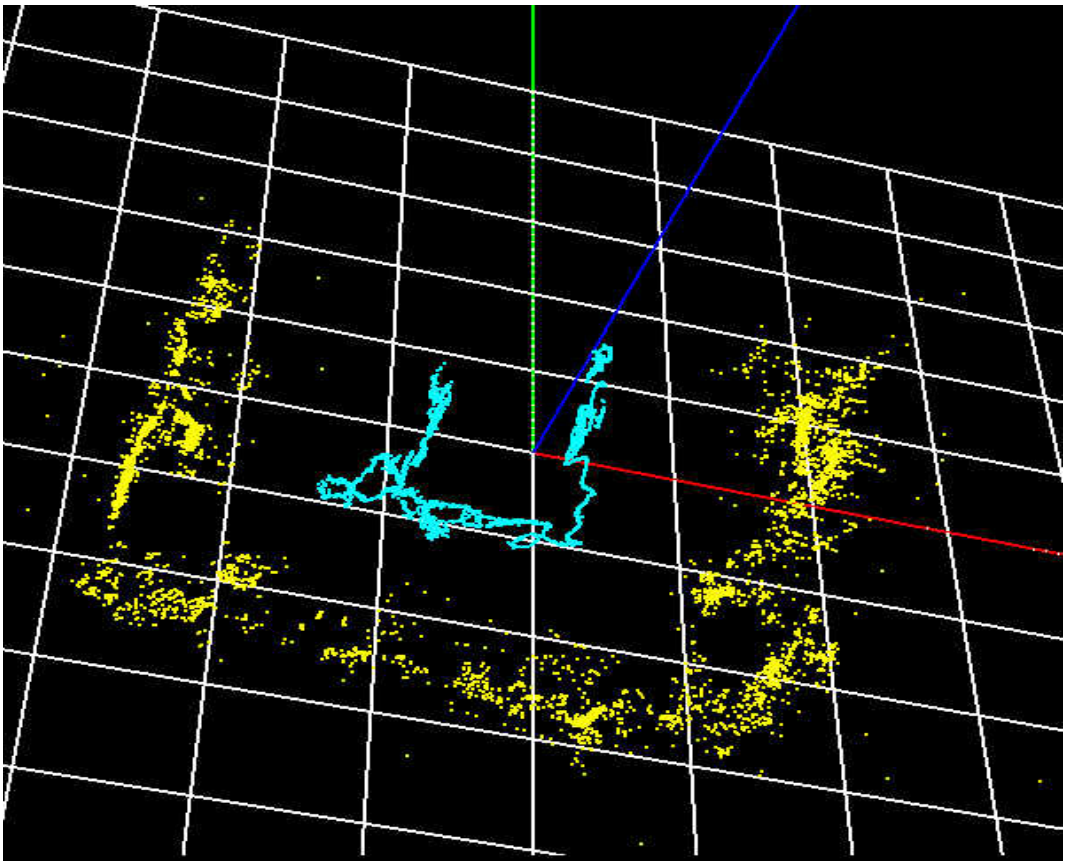


Figure 15: Point Cloud Map with Camera Trajectory. The camera trajectory is illustrated with the cyan lines while the yellow points denote the map points.

Figure 15 illustrates the generated map of one of the sequences. The video consists of 2997 frames of an indoor environment of size 10x8 meters. At the end of the sequence, a total of 188 key frames were generated with 5654 feature points. The camera trajectory is illustrated by the cyan line and the yellow points denote the feature points. The map shows that there are few obvious outliers. However, the accurate structure of the map and the camera trajectory illustrates the effectiveness of the algorithm. Figure 16 shows a few snap shots of the environment where the map in Figure 13 is generated. The images show the map points that are successfully matched.

Performance evaluation of the earlier version of the algorithm is shown in Figure 17. The algorithm runs real time on a laptop computer with Intel i7 processor and 3 GB RAM. The top figure shows the number of feature points successfully matched per frame while the bottom figure shows the time required in seconds to perform the matching and the iterative pose estimation for the same image sequences. The time for each frame shown in the graph is the time required to process one frame that includes getting the frame from disk, computing the detectors and descriptors, project map points on the image, perform matching to obtain 3D-2D correspondences, iterative pose updates, making decisions if new key frame is needed and add new key frames and key points if the condition is true. The number of matches is high in places where the camera revisits compared to the places of rapid exploration. For the longest video sequence, there were few instances where the number of matched features was low. The first instance was due to large rotational motion of the camera while the linear motion was small. Condition for new key frame addition was triggered only after the distance between the current frame and the closest frame reach the threshold. The second instance was due to the rapid motion of the camera. The images in this sequence were collected using a standard webcam mounted on a quadrotor. During rapid camera motion, the generated images became blurred. The detector generates a smaller set of feature points in those frames resulting in a small number of map points in those locations. The blurriness of the images was reduced to some extent by increasing gain and decreasing exposure values. However, it was not possible to completely eliminate it. In addition, some area of the environment consists of texture less surfaces resulting in no interest points.

Figure 16: Few snap shots of the environment where the SLAM algorithm was run. The pink dots represent the map points that are successfully tracked for camera pose estimation.

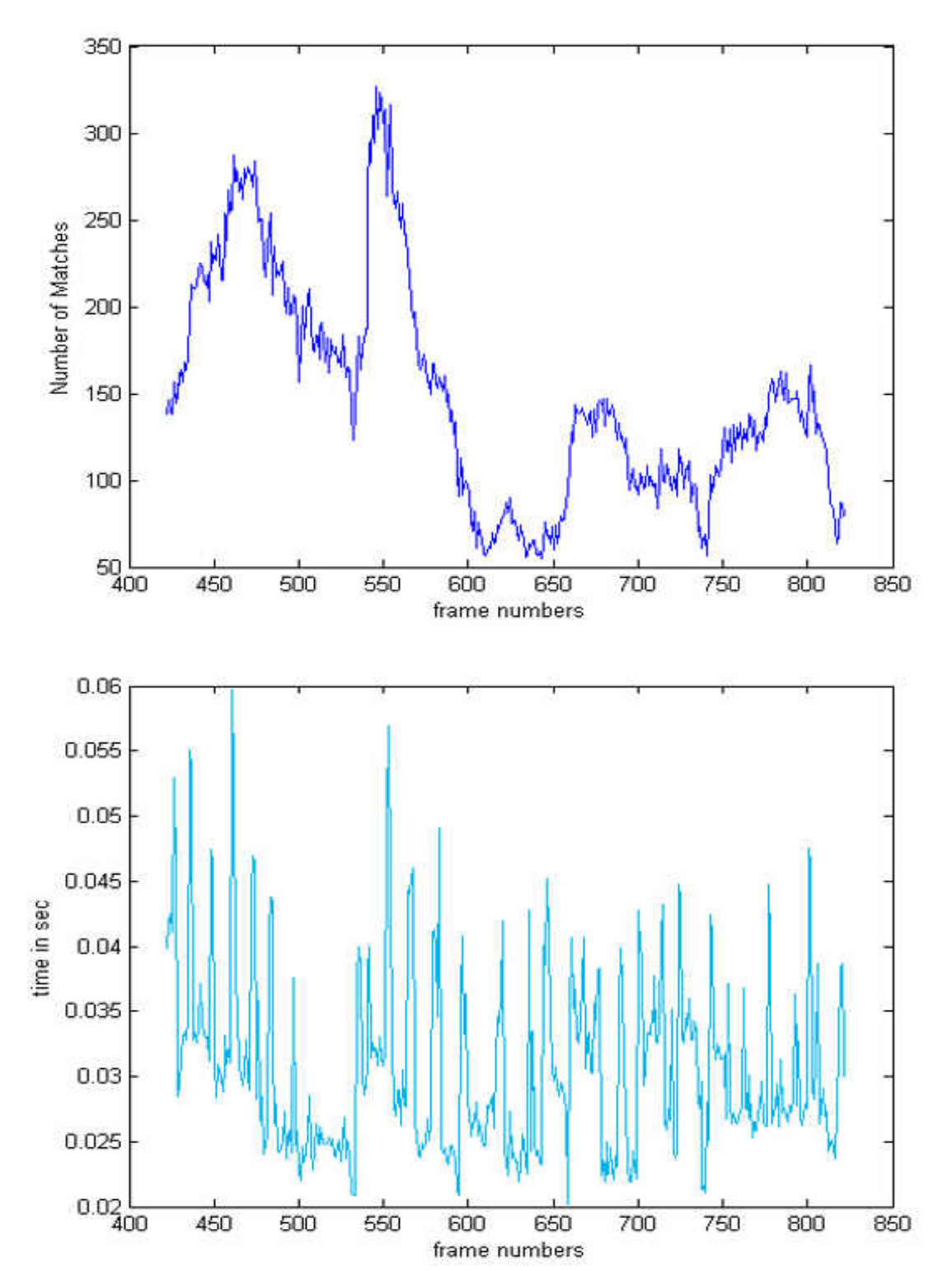


Figure 17: Performance evaluation of the algorithm. The top figure shows the number of matches per frame while the bottom figure shows the time required for tracking each frame

Performance Evaluation with Outdoor Dataset

Outdoor evaluation of the proposed algorithm is performed using the publicly available KITTI visual odometry dataset [44]. The dataset consists of 22 stereo sequences captured with a stereo camera mounted on a vehicle. The sequences are of various lengths ranging from 100 to 800 meters. All the sequences are captured at 10 fps. This dataset is quite challenging for monocular SLAM algorithm because of the forward motion of the vehicle with varying speed ranging from near stop to 60 kilometers per hour. In addition the sequences consist of environments with different structures and vegetation. In our experiments, we use images from the left camera of the stereo pairs. The algorithm was successfully able to run on the dataset and generate accurate camera trajectories. However, the trajectories show significant scale drifts when there was no loop in the image sequences.

Error Analysis

In order to evaluate the accuracy of the SLAM algorithm the generated trajectory is compared against the ground truth camera pose data from the dataset. Since the scale in unknown in the monocular SLAM system, initially the scale is fixed by computing the distance between the first two key frames and then comparing with the corresponding ground truth poses from the dataset. Error in orientation is evaluated by first computing the relative orientation between two successive key frames. This relative orientation is then compared with the relative orientation of the ground truth poses. Figure 18 shows the comparison between the ground truth and generated trajectory form the developed SLAM system. This trajectory consists of several loop closures. The algorithm was successfully able to detect the loop and performed loop closure.

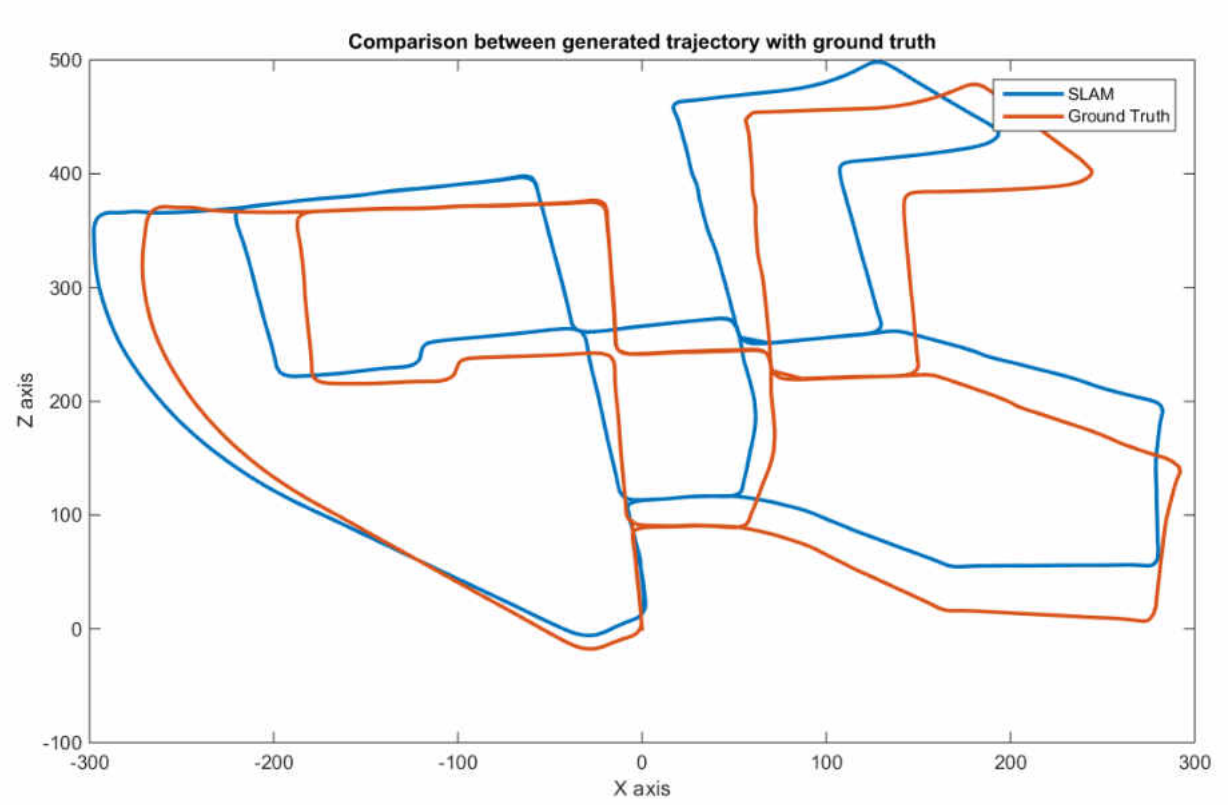


Figure 18: Comparison of trajectory generation between the ground truth and the SLAM algorithm on KITTI dataset. The scale of the trajectory is set using the ratio of distances between the first two key frame poses and their corresponding ground truth distances. The trajectory consists of several loop closures thereby preventing the scale drift in the pose estimation.

Error in relative orientations in the generated trajectory is shown in Figure 19. Result shows that orientation error stays within a small range around 0 during the full length of the trajectory. Figure 20 shows a few generated trajectory from the dataset and compared with the ground truth. The scale is fixed in the beginning with first two key frames. The left images show the generated trajectories while on the right the relative orientation errors are plotted. The orientation errors are computed by comparing the relative orientations between the key frames generated by the SLAM algorithm and the relative orientations. Results show that the relative orientation errors consistently stay within a small threshold.

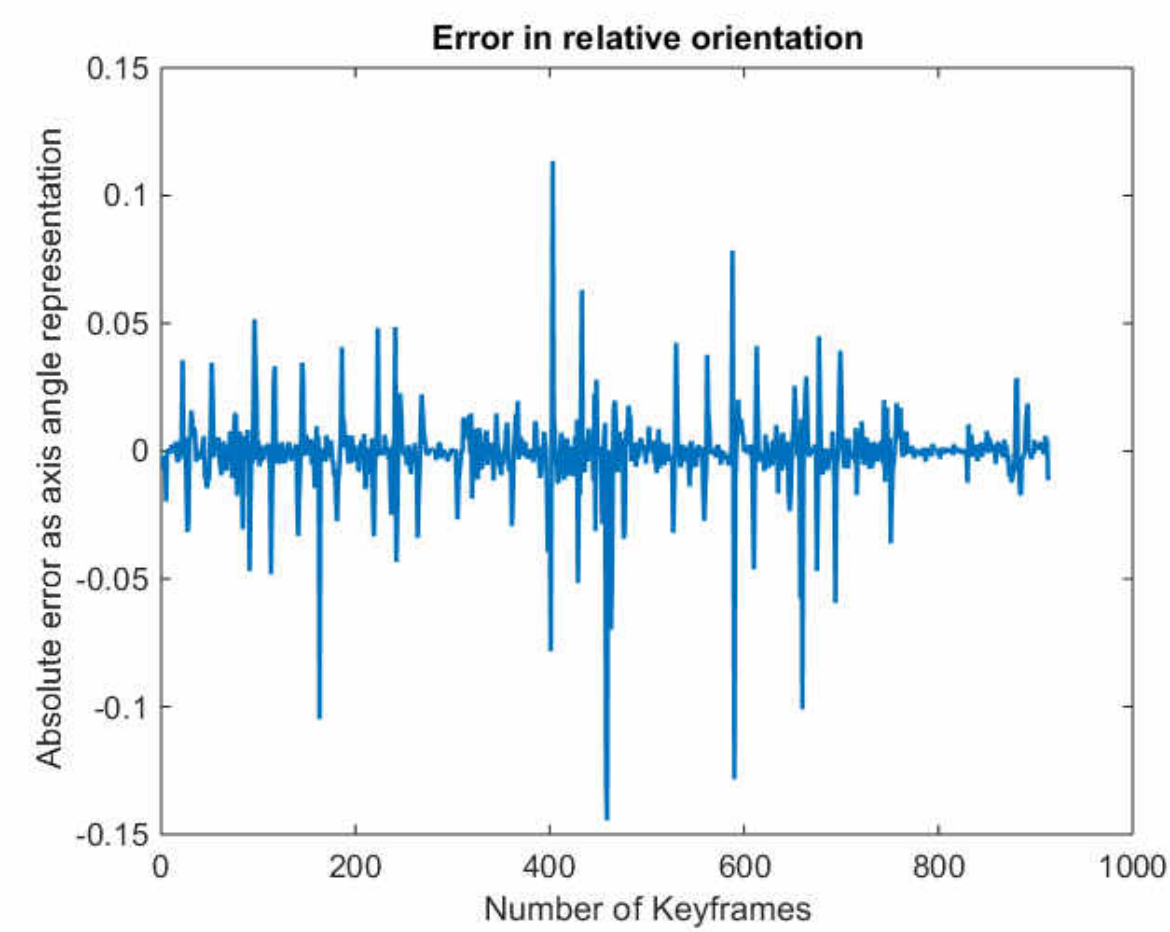


Figure 19: shows the error in relative orientation between two key frames. The computed relative orientation is compared against ground truth to generate the error.

Loop Closure Detection and Correction

We tested our loop detection and correction algorithm in the two different image sequences in the KITTI dataset. The first sequence is 4541 images long and the second sequence is 2400 images long. Our results show that in each case the algorithm is successfully able to perform the loop detection and correction. Figure shows some images from the loop detection algorithm.

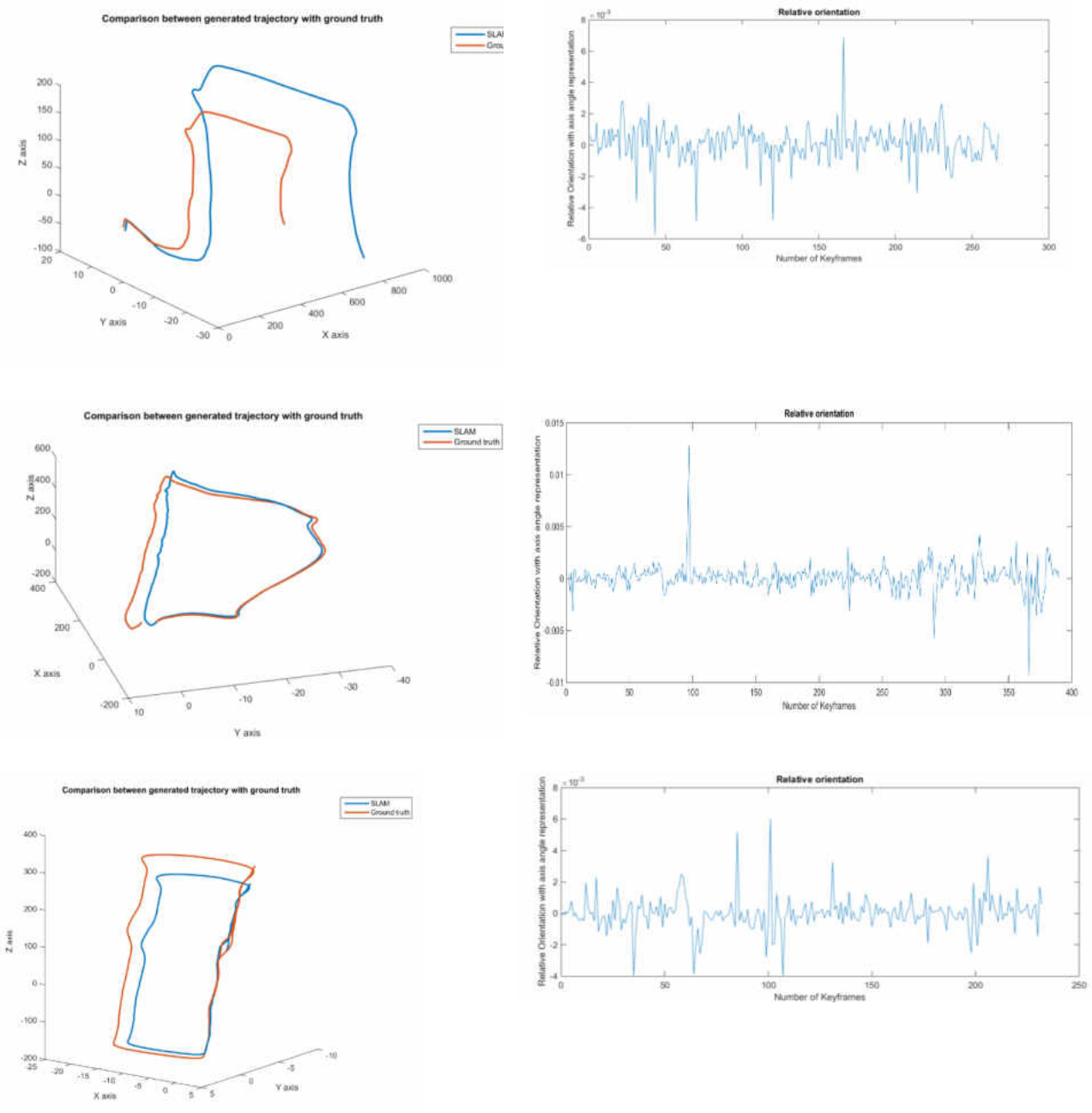


Figure 20: Comparison of generated trajectory. The generated trajectory demonstrated the scale drift in the monocular slam algorithm. The scale of the generated trajectory is fixed by computing the distance ratio between the first two key frame and their corresponding ground truth data

The vocabulary is created using a set of 138 images that are independent of the data set used for evaluation. The vocabulary tree is created using a depth level 5 and branching factor 12. Some example loop closure detections in two long image sequences in the KITTI dataset is shown in Figure 21.

Figure 21: Some example loop closure detections in the KITTI dataset.

Figure 22 demonstrates the importance of computing the fundamental matrix in order to find the map point correspondences between the loop frame and the current frame. Initially a brute force method is applied to find the corresponding image positions in the loop frame for the map points in the current frame. The top figure shows the matching with a significant number of outliers. The bottom figure shows the outlier rejection by computing the fundamental matrix with a RANSAC scheme and then by performing epipolar constraint test. This is important in order to compute the similarity transform between the current frame and the loop candidate frame.

Figure 22: Using fundamental matrix for outlier rejection in loop matching. Top figure: A brute force searching for the map points correspondences results in a significant number of outliers. Bottom figure: rejecting outliers to find correct points correspondences between the loop candidate frame and the current frame.

Comparison between the trajectories generated by the algorithm with and without loop closure is shown in Figure 23. The results clearly demonstrate the importance of loop closure in monocular SLAM algorithm as the algorithm suffers from scale drift in absence of loop closure.

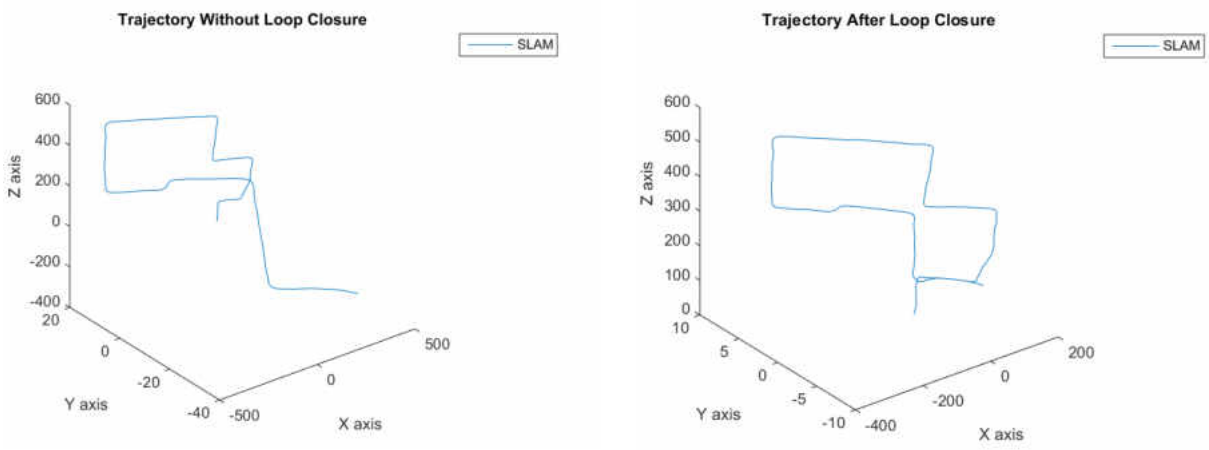


Figure 23: Trajectory generation with and without loop closure. The left image shows the generated trajectory without loop closure. The right image shows the corrected and optimized trajectory after loop closure is detected

Time required by the major components of the loop closure detection is shown in Table 2. The database query to find a loop candidate frame takes a very small amount of time. This demonstrates the effectiveness of the algorithm to build a long duration map where the loop closure detection can be performed in a very short amount of time.

Table 2: Time break down of the major components of the loop detection

Items	Time in Sec
Data Base Query	0.045
Map Points Matching	0.08
Fundamental Matrix With RANSAC	0.047

Computational Study

We evaluated the efficiency of the algorithm by measuring the time required for the computationally expensive components of the algorithm. The algorithm was run multiple times and then the times required are averaged to obtain an estimate of the time required for the components of the algorithm. The major time critical components are feature detection and descriptor generation, new map points creation and the map points matching. Especially on the ODROID computer, feature detector and descriptor generation is particularly slow with serial processing. However, the time improves significantly when done in parallel.

Comparison Between Serial and Parallel Processing. In order to evaluate the performance enhancement with parallel processing, the algorithm is run on the dataset using both serial and parallel processing. The effectiveness of the parallel feature

detection and descriptor generation is demonstrated in Figure 24 as majority of the time for tracking each frame is spent on the detection and descriptor generation. On a standard laptop, time required for the parallel feature detection and descriptor generation is less than half of what is required for the serial feature detection. For each frame 2000 feature points are generated on 8 pyramid level. The image size used in the comparison is 1241x376.

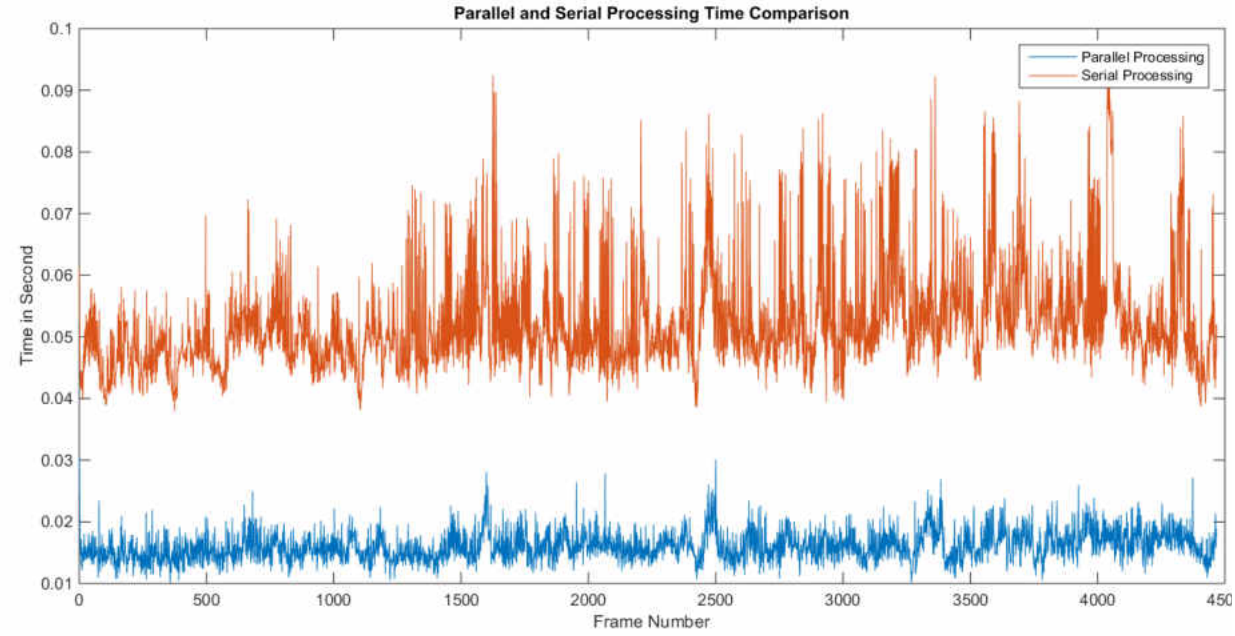


Figure 24: Comparison between parallel and serial feature detection and descriptor generation. With parallel detection and descriptor generation, the time requirement is reduced to less than half of the time required for serial processing.

Figure 25 shows the comparison between times required to track one frame using serial and parallel descriptor generation and map points tracking. Tracking one frame include detection and descriptor generation, map points matching and pose optimization from the matches found in the image.

Performance Evaluation on The Onboard Computer

In order to evaluate the performance of the SLAM algorithm on ODROID XU3 computer, the SLAM algorithm was run on ODROID using the image sequence 00 of KITTI dataset with the same parameters settings as the laptop computer. The algorithm runs significantly slower that the laptop. However, the computational time clearly suggests that it is possible to use the algorithm for onboard large scale localization and mapping with a low power small form factor computer.

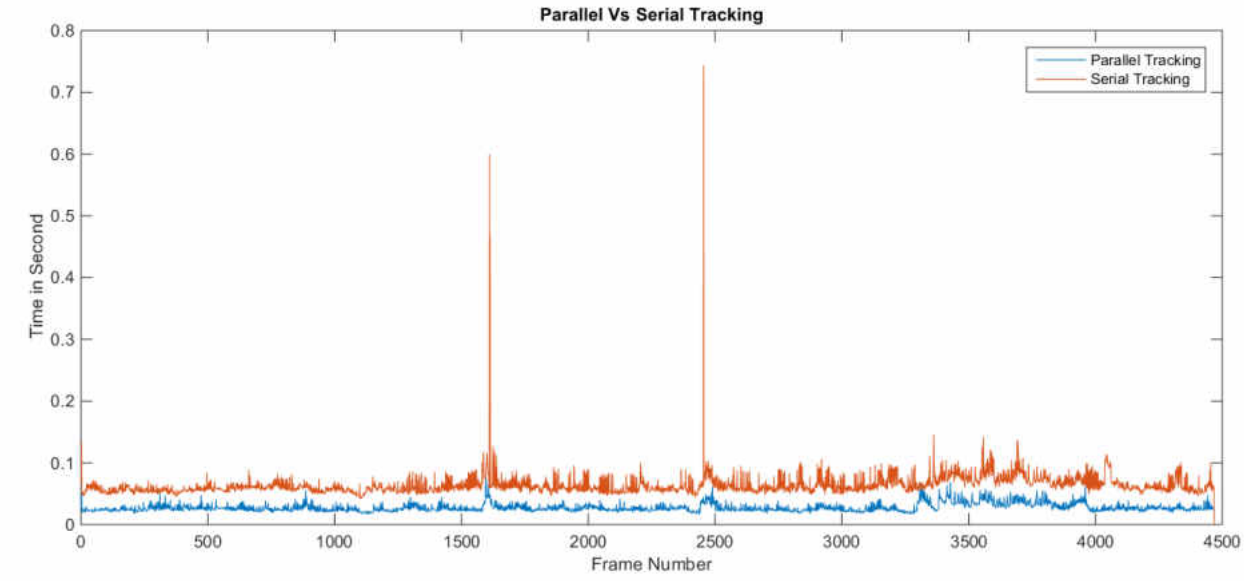


Figure 25: Comparison between parallel and serial tracking on the KITTI dataset.

With the parallel threading for detector and descriptor generation, map points tracking and new map points generation, average tracking time for 1600 images is 0.1238 seconds per frame. The major time critical components of the tracking system is the detector and descriptor generation which requires on an average 0.085 sec or 85 millisecond for 2000 features on 8 pyramid level on an image size 1241x376. Figure 26 shows the processing time per frame on the ODROID.

The algorithm was also compared with the open source ORB SLAM algorithm [20]. To compare the performances, both algorithms were run on the ODROID using the same parameters settings for detector and descriptors generation. On the KITTI dataset, the average tracking time for the ORB SLAM was found 0.23 seconds per frame which required almost twice the amount of time to track each frame. In worst case, it took over 0.3 seconds to track one frame.

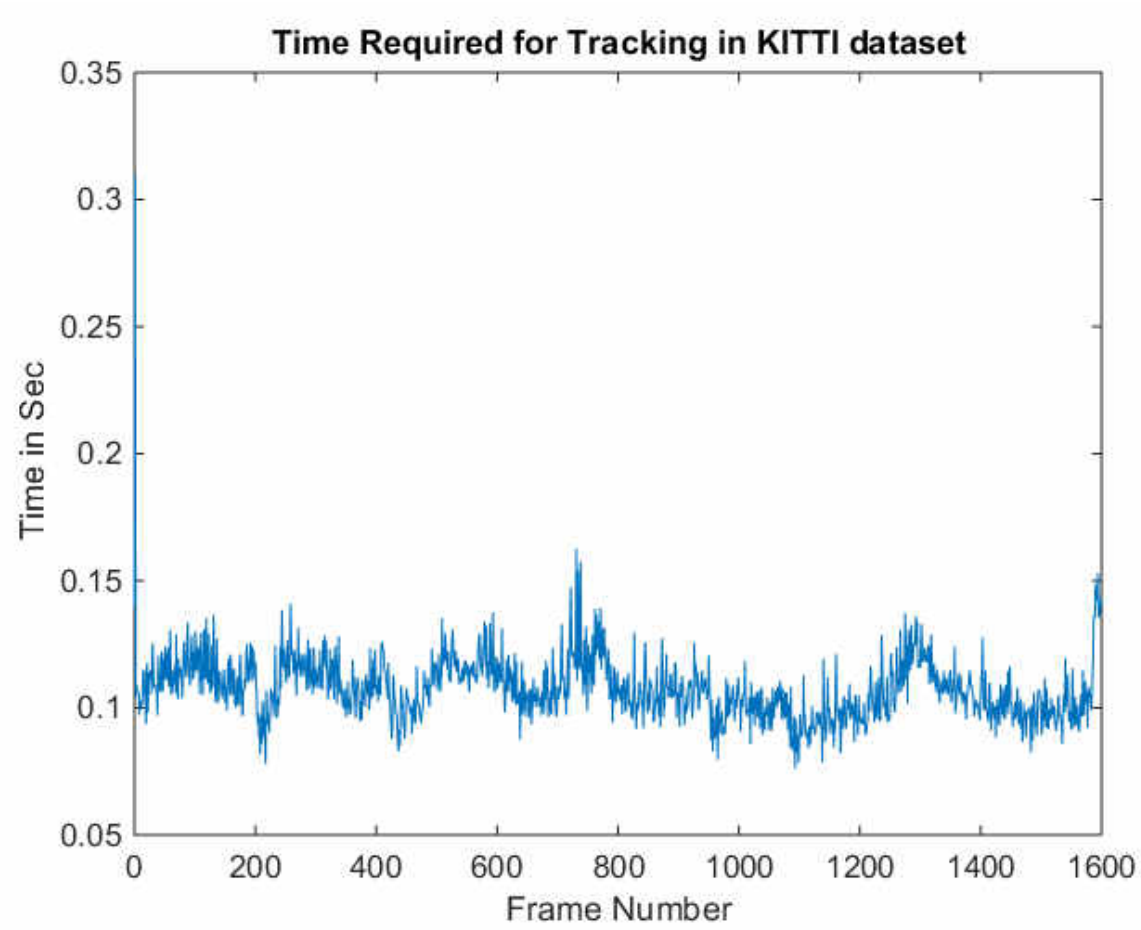


Figure 26: Time requirement for tracking per frame on the ODROID. The algorithm was run on the KITTI dataset to evaluate the performance of the algorithm

Comparison On Flight Data. The developed SLAM algorithm is also compared with the publicly available ORB SLAM algorithm on a dataset generated during the manual flight of the quadcopter. The image sequence include quad takeoff and flight in

an indoor environment. The images were captured with a chameleon 3 camera with image size 1228x376. In order to compare the performances, 2000 corner points are detected on 8 image pyramid levels. The FAST threshold was set at 20 in both cases.

The result demonstrate the robustness and efficiency of the developed algorithm as the ORB SLAM failed right after initialization. This is due to the amount of time required for the ORB SLAM to track each frame. Initially the quadcopter was sitting motion less on the ground before takeoff. During this time, the ORB SLAM took about 0.45 second to process one frame. In comparison, the developed algorithm took about 0.24 second to process each frame. Since a large number of features were tracked (~900) during the initialization, the computation time remained high until the SLAM algorithm was initialized. Once initialized, the algorithm was able to process each frame at a rate of 0.13 second. However, the time required for processing one frame suggests that the image size is too large for real time performance.

Figure 27 shows the time required for tracking one frame. The camera position from the SLAM algorithm is shown in Figure 28.

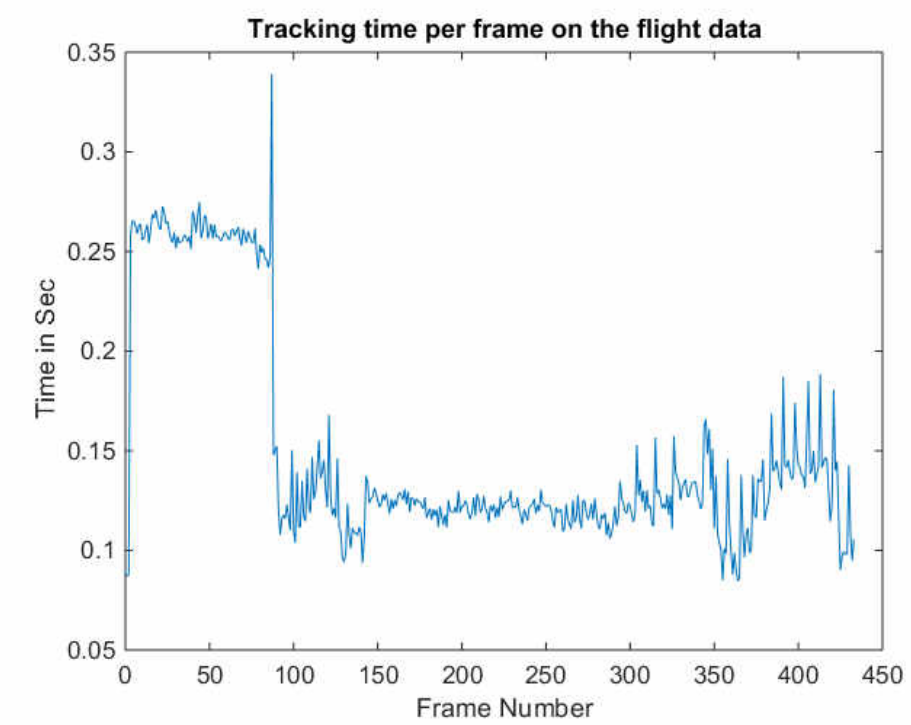


Figure 27. Time required for processing one frame on the flight data

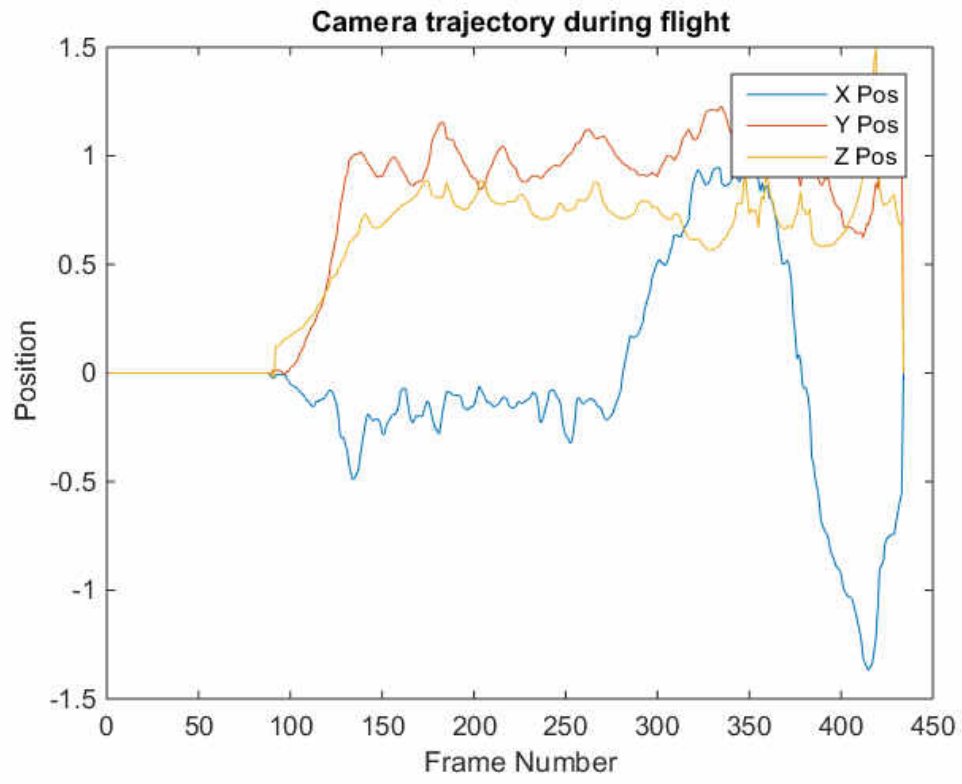


Figure 28. Camera Poses generated by the SLAM algorithm on the flight data

Summery

In this chapter we presented our real time scalable monocular SLAM system. The algorithm process flow is shown first and the major components of the algorithm are described in details. In order to achieve robustness in rapid exploration, camera pose tracking and map expansion is performed in the frontend while the incremental local bundle adjustment for map optimization is performed in the backend. The map expansion occurs only when the robot visits previously unexplored area. Parallelization of the time critical components help make the algorithm computationally efficient and capable of running real time. A robust and efficient appearance based loop closure detection algorithm is developed that run in a separate thread. Temporal and geometric consistency check is performed to prevent from false loop closure detection.

Performance evaluation of the algorithm was done using indoor and outdoor dataset. Initially the indoor dataset was created by mounting the camera on the quadrotor and carrying the quadrotor in an indoor environment. Additional dataset was created by flying the quadrotor with the camera and the onboard computer. In order to evaluate the algorithm in outdoor environment, we have used the publicly available KITTI visual odometry dataset. Results demonstrate the algorithm's effectiveness in the challenging outdoor environment such as partially dynamic environment as well as environment with only vegetation.

CHAPTER IV

INERTIAL AIDED VISUAL SLAM FOR SMALL MOBILE PLATFORM

Introduction

Over the last few years, a substantial amount of research has been done on integrating visual and inertial sensors for localization and navigation in the mobile robotics community [50-53]. The complementary nature of these sensors provides rich information to build a system that is capable of navigation in an unknown environment without any external infrastructure. This is a key advantage for mobile robots operating in an unstructured and unknown environment. However, mobile robots have limited computational capacity and require a reliable and efficient method to estimate the physical quantities related to navigation. This is particularly true for small aerial robots such as micro aerial vehicles (MAV).

Aerial robots offer great potential for applications ranging from precision agriculture, construction site monitoring to search and rescue. They have the capabilities to reach places where it is impossible or hazardous for human beings. However, the reach the full potential of the aerial robots, certain degrees of autonomy is imperative. While the use of GPS in outdoor environment is the most common way to achieve the autonomy, sole reliance on GPS signal is problematic as reception cannot be guaranteed (signal can be lost or compromised). Ideally we would like to have an aerial robot that is equipped with

proprioceptive sensors capable of localization itself and perform navigation in the absence of any external information or communication.

Localization and mapping based on laser or RGBD cameras have been successfully implemented for autonomous micro-aerial vehicles [50, 51]. However they are only usable in certain environments. For example, laser based localization requires at least a partially structured environment for incremental motion calculation [51, 52]. On the other hand RGBD camera has a limited depth range thus making these sensors not feasible for unstructured outdoor environment.

An onboard vision based state estimation method is described in [53] where vision and IMU information are used in a visual inertial navigation system for MAV localization and trajectory control. The system uses a combination of two cameras where a primary camera is used for fast frame to frame tracking in a visual odometry framework. The second camera runs at a much slower rate to generate map points using triangulation. The use of a second camera in a stereo scheme allows for metric scale recovery. The system uses a KLT tracker [54] with Shi-Tomasi corners [55] for frame to frame motion estimation. The pose estimation is performed at 20 fps on a 1.6 GHz Intel Atom based onboard computer. However, the system does not address the full SLAM problem and from our experience KLT tracker is not suitable for repeated environment as well as large change in scale and illumination.

A real time onboard vision based navigation system was proposed in [56] where they used a visual inertial system within an EKF framework for localization. The information coming from visual and inertial system is fused for state estimation in a loosely coupled

manner for pose and scale estimation. For visual pose estimation, the system used a modified version of PTAM where the original algorithm was simplified to achieve onboard processing capability for a small hexacopter. The simplification came at a cost where they discard all bust a few previous frames as the system was not intended for large scale map building. In addition, the implementation used a downward looking camera essentially limit its applicability.

Using stereo systems for MAVs have also been proposed in literature [57,62]. A stereo vision based approach for autonomous mapping and exploration using a forward looking stereo system is presented in [57] where they use a combination of onboard and offboard computation for map generation and navigation. The onboard computer is used for pose estimation in a visual odometry framework from the stereo images and the images are sent to a ground computer for map generation and loop closing. Indoor and outdoor experiments are performed while the MAV was flying at a low altitude. Since the range estimation from a stereo system is quite limited, scale estimation in larger environment is a limitation in the system.

In light of these developments, we propose a full scale SLAM system that is capable of running onboard the small UAV. The SLAM algorithm is capable of running in both indoor and outdoor environments in a GPS denied environment. We also provide a filter based multi-sensor fusion framework where additional sensors such as GPS (when available) can be incorporated in the state estimation easily. In this chapter, we provide a detailed description of the SLAM algorithm that is developed to run onboard a small computer carried by the quadroter developed in our lab.

Coordinate Frames

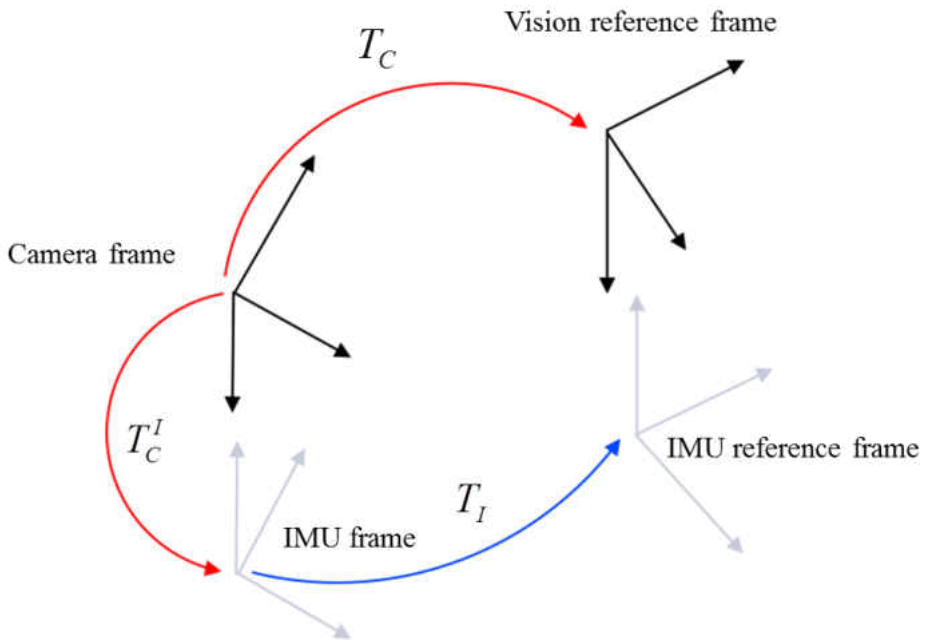


Figure 29: Vision and IMU reference frames and their relative transformation

The coordinate systems for our framework consists of an inertial frame, a body fixed frame for the quadrotor and the vision reference frame. For our test purposes, the inertial frame considered is the earth fixed NED (North East Down) coordinate frame with x axis pointing north, y axis pointing east and the z axis pointing to the center of the earth. The origin of the inertial frame is selected as the position when the SLAM algorithm starts. The body fixed frame is attached to the quadrotor with x axis pointing forward, y axis pointing to the right and the z axis pointing downwards. The center of the body frame is considered as the center of the IMU. Assuming that the camera is rigidly mounted on the platform, a constant transformation matrix between the IMU and the camera center is

considered in this work. This transformation matrix is estimated by measuring the distance between the coordinate center of the flight controller and the camera.

Prediction From IMU

The autopilot is equipped with three accelerometers and three gyros measuring the accelerations of the three orthogonal coordinates and angular velocities about the coordinates respectively. The gyros generate angular rate signals when the platform that the image arrivals and can be integrated between the poses to estimate the relative orientation and positions between two successive image arrivals.

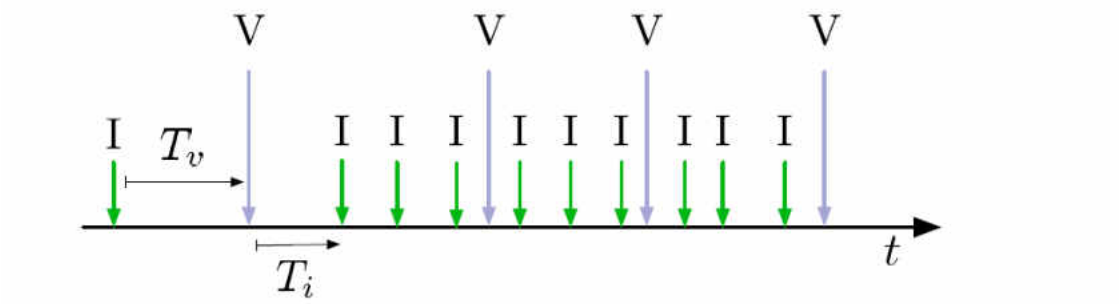


Figure 30: Sensor measurement over time. The measurements from IMU arrive at a faster rate than the camera images.

Denoting the attitude of the IMU at time t_1 and t_2 as I_1 and I_2, the relative rotation can be computed by propagating the rotation matrix using the gyro measurements ω_t to estimate the attitude change during the time between two image arrival. With high sampling rate (Δt is small) the propagation of the rotation matrix can be performed as follows [6]:

$$R_{t+1}^{I_1} = R_t^{I_1} \Delta R_t \tag{61}$$

$$\Delta R_t = I + \frac{\sin \varphi_t}{\varphi_t} \lfloor \overline{\varphi}_t \rfloor_\times + \frac{1 - \cos \varphi_t}{\varphi_t^2} \lfloor \overline{\varphi}_t \rfloor_\times^2 \tag{62}$$

$$\varphi_t = \sqrt{\varphi_{x,t}^2 + \varphi_{y,t}^2 + \varphi_{z,t}^2} \tag{63}$$

$$\overline{\varphi}_t = (\varpi_t - \overline{b}_{g,t})\Delta t \tag{64}$$

Where, $R_{t+1}^{I_1}$ is the propagated rotation matrix computed from the rotation matrix $R_t^{I_1}$ of the last time step t and the relative rotation ΔR_t in the interval t and $t+1$. $\overline{\varphi}_t$ is the rotation vector and $\overline{b}_{g,t}$ is the bias vector in gyro measurement.

Since this relative rotation from the gyros measurements are in the IMU coordinate frame of the platform, a coordinate transformation is needed to estimate the relative orientation in camera frame. Assuming the relative transformation between the IMU and the camera is fixed, relative orientation in the camera frame is denoted as

$$R_{C_t}^{C_{t-1}} = R_I^C R_{I_t}^{I_{t-1}} R_C^I \tag{65}$$

It is well known that measurements from low cost gyros suffer from a slow drift term. However, the relative measurement between only two consecutive image frames limits the accumulation of the drift and a reasonable prediction for the absolute camera pose is obtained as follows

$$R_{C_t} = R_{C_{t-1}} R_{C_t}^{C_{t-1}} \tag{66}$$

One way to predict the position is to integrate the acceleration measurements between two image arrivals to get a relative displacement. However, displacement computation from double integrating acceleration measurements is very sensitive to noises as even a small error in the orientation or biases gets amplified by the integration process. In order to obtain a prediction for the motion, the method used in this work is similar to the IMU pre integration method described in [47] where the IMU measurements are integrated in

the IMU frame of last image arrival. We use the notations $\Delta P^{I_1+}, \Delta V^{I_1}$, and $R_{I_2}^{I_1}$ to represent the position, velocity and orientation components that are computed from the IMU measurements obtained between time intervals t_1 and t_2. Since the accelerometers provide specific force measurements, ΔV^{I_1} represents the change in velocity between the poses I_1 and I_2. However, the position change estimation requires an initial velocity at the beginning of the integration period. As such, the term ΔP^{I_1+} is considered as a corrective term generated from the acceleration profile during the time of integration.

Table 3 Algorithm for IMU measurement integration

1: $\Delta P^{I_1+} = 0$
2: $R^{I_1} = I$
3: for $t_1 < t < t_2$ do
4: $\Delta t = t_{t+1} - t$
5: $\Delta V_{t+1}^{I_1} = \Delta V_t^{I_1} + R_t^{I_1}(\bar{f}_t^{I} - \bar{b}_t)$
6: $\Delta P_{t+1}^{+I_1} = \Delta P_t^{+I_1} + \Delta V_t^{I_1} \Delta t$
7: $R_{t+1}^{I_1} = R_t^{I_1} \Delta R_t$
8. End for
9. Output = $\begin{bmatrix} \Delta P_t^{+I_1} \\ \Delta V_t^{I_1} \\ R_t \end{bmatrix}$

In the IMU frame of the first image arrival, the relative velocity and displacement during the time interval t_1 and t_2 can be calculated as

$$v_{del}^{I_1} = v_{t_1}^{I_1} - v_{t_2}^{I_1} = (t_2 - t_1) g^{I_1} + \Delta v_t^{I_1} \tag{67}$$

$$p_{del}^{I_1} = p_{t_1}^{I_1} - p_{t_2}^{I_1} = (t_2 - t_1)v^{I_1} + \frac{1}{2}(t_2 - t_1)^2 g^{I_1} + \Delta p_t^{+I_1} \tag{68}$$

Together the rotation and translation component of the relative motion of the platform is represented as a rigid body transformation

$$T_2^1 = \begin{bmatrix} R & p_{del} \\ 0 & 1 \end{bmatrix} \tag{69}$$

Finally, the predicted pose in the navigation frame can be computed as

$$T_{t_2}^n = T_{t_1}^n T_2^1 \tag{70}$$

After the new image arrival, the rotation matrix is set to identity and the delta components for position and velocity are set to zero before starting the pre-integration of the IMU measurements. Since the velocity estimation is subjected to large drift because of error accumulation, the initial velocity v_t^I is updated after the camera pose is optimized from the map points.

Tracking And Pose Estimation

The map feature points tracking is similar to the previous chapter where a region is created around the predicted image position for the potential visible map points and then perform descriptor matching to find map points in the current frame. However, the pose estimation method takes a different path where we take advantage of the IMU measurements. Since the relative rotation from the IMU measurements between two image arrivals can be considered reliable, we use the attitude prediction as a measurement and estimate the camera position using a 2 points RANSAC scheme [21]. Assuming the rotation is known, the only unknown parameter is the 3D translation vector and can be solved using 2 points correspondences. From the successfully tracked map points, 2

points are selected randomly to get an estimate of the camera pose and then the number of inliers is computed from the projection model. At the end, we select the camera pose with maximum number of inliers as our initial camera pose estimate and finally the camera pose is optimized using the nonlinear optimization method explained before.

Implementation Details

The original implementation of the BRISK descriptors uses the SSE instructions for Intel processors for performing both the image sampling and the hamming distance calculations. For the single board computer, the image sampling is performed using the neon instructions set for ARM processors [3]. Hamming distance calculation is performed using the standard OpenCV implementations for ARM processors.

Metric Scale Estimation

One major challenge with monocular camera is that the scale is observable with a monocular SLAM. In addition the scale tends to drift because of the gauge freedom. Most state of the art visual inertial navigation systems (VINS) [48, 49, 59] use IMU information in a filtering or nonlinear optimization scheme to recover scale of the system. A closed form solution for scale estimation using a monocular camera along with the IMU data is also shown in [60]. However, a nonlinear observability analysis of the state estimation problem shows that there exist unobservable modes with monocular VINS that can only be eliminated through motions that involve non-zero linear accelerations [56, 59]. This is challenging for a platform such as quadrotor that can have hover mode as well as complex motion. In addition the quality of the IMU data from the low cost IMU

integrated to the autopilot makes direct use the state of the art VINS systems for state and scale estimation quite impossible.

We take advantage of the ultrasound sensor that is pointing to the ground along with a vertical motion in order to set the scale when the map initialization is performed. When the SLAM is initialized, the motion of the quad is constraint in such a way that in addition to sideways motion, the quad also has some vertical motion. This condition is easy to achieve for a quad motion. As the quadrotor takeoff mostly consists of vertical motion until it reaches a hover mode, we can initialize the SLAM during the takeoff and the map will be initialized once the initialization conditions mentioned above are met. On the other hand, if the SLAM is initialized when the quadrotor is in hover mode, the hover set point can be set higher than the current height and the map can be initialized when there is sufficient parallax between the initial and current position of the quadrotor. In order to set the scale of the system, the ground height from the ultrasonic sensor is stored when the SLAM is first initialized and when the map initialization is done. The next step is to compute the difference in height between the first quadrotor pose and the pose when the map is initialized and compute the ratio of the height from direct distance measurement and the vision system. The result is our scale. Once the scale is known, the whole map along with the two quadrotor poses in the map is adjusted for the scale.

The above step ensures the proper scale of the state estimation using the monocular SLAM. However, we still need to have a method to account for the scale drift during exploration. To address this, we use an extended kalman filter (EKF) to estimate the scale when the quadrotor is in motion. We assume that the attitude computed from the vision

system is rather accurate and setup our state as a 10 elements vector consisting of 3D position, velocity, acceleration and scale.

$$X = \begin{bmatrix} p \\ v \\ a \\ \lambda \end{bmatrix} \in R^{(3\times3\times3\times1\})} \tag{71}$$

The prediction model for the EKF can be written as

$$X_{k+1} = f(X_k) + w_k \tag{72}$$

$$\begin{pmatrix} p_{k+1} \\ v_{k+1} \\ a_{k+1} \\ \lambda_{k+1} \end{pmatrix} = \begin{bmatrix} I & \frac{T}{\lambda} I & \frac{T^2}{2\lambda} I & 0 \\ 0 & I & TI & 0 \\ 0 & 0 & I & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} p_k \\ v_k \\ a_k \\ \lambda_k \end{pmatrix} \tag{73}$$

Where w_k is the Gaussian process noise and T is the time difference. Every vector is resolved in the inertial coordinate. In order to keep the algorithm fast and simple, we used the simplification [] that the uncertainty is infinity when no measurement is available and update the states when a measurement is received either from the vision system or the IMU.

The measurement update equations for the vision and IMU can be written as

$$y_{V,k} = H_{V,k} X_k = [I_3 \quad 0_3 \quad 0_3 \quad 0] X_k \tag{74}$$

$$y_{I,k} = H_{I,k} X_k = [0_3 \quad 0_3 \quad I_3 \quad 0] X_k \tag{75}$$

Innovation for the vision is written as

$$K_{V,k} = P_k^- H_{V,k}^T (H_{V,k} P_k^- H_{V,k}^T + R_V)^{-1} \quad (76)$$

$$X_k = X_k + K_{V,k}(X_V - H_{V,k} X_k) \quad (77)$$

$$P_k = (I - K_{V,k} H_{V,k}) P_K^- \quad (78)$$

And innovation for the IMU part as

$$K_{I,k} = P_k^- H_{I,k}^T (H_{I,k} P_k^- H_{I,k}^T + R_I)^{-1} \quad (79)$$

$$X_k = X_k + K_{I,k}(a_{IMU} - H_{I,k} X_k) \quad (80)$$

$$P_k = (I - K_{I,k} H_{I,k}) P_K^- \quad (81)$$

Software Architecture for onboard computation

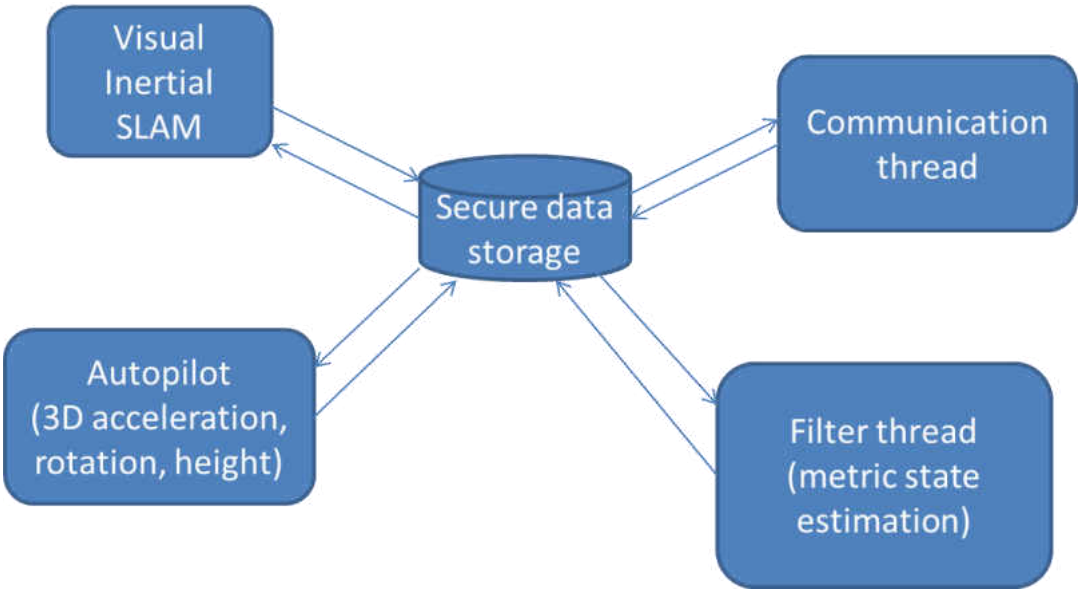


Figure 31: The modular architecture of the software that runs on the onboard computer

Loop Closure Detection

The loop detection method is already described in the previous chapter where every new key frame is checked against the database that is continuously built during the mapping

process. However, we don't implement a full pose graph optimization in order to make the implementation efficient. Here we adopt a global relaxation method by creating a graph that consists of a local region. When a loop closure is detected, the relative pose between the two key frames generates an edge in the graph. Using the two key frames as the roots, we construct a graph based on the co-visibilities list for both key frame. We essentially perform a breadth first search based on the co-visibility score and incrementally add key frames up to a fixed number of key frames based on their co-visibility score. The graph is then optimized using the procedure explained in the previous chapter. This is essentially pushing the uncertainty to the farthest key frames.

Experiments & Results

Effectiveness of the proposed algorithm to be able to run onboard has been validated during manual flights of the quadrotor in an indoor environment. The SLAM algorithm ran on the single board computer onboard the quadrotor during the flight and generated camera pose estimation. The effectiveness of the re-localization algorithm was also demonstrated during these flights. At one instance, the tracking failed due to sudden yaw movement of the quadrotor. However, once the quadrotor rotates back to previously explored place, the relocalization algorithm was able to recover from tracking failure and the tracking resumes again. Figure 32 shows the performance of the SLAM system during the manual flight of the quadrotor. The top image shows the quad pose trajectory generation from the tracker. The bottom image shows the number of successfully matched points. The tracker fails at around frame 248 due to the sudden rotation of the quadrotor. However, the system was able to relocate once the quad was moved back to previously explored region and tracking starts again.

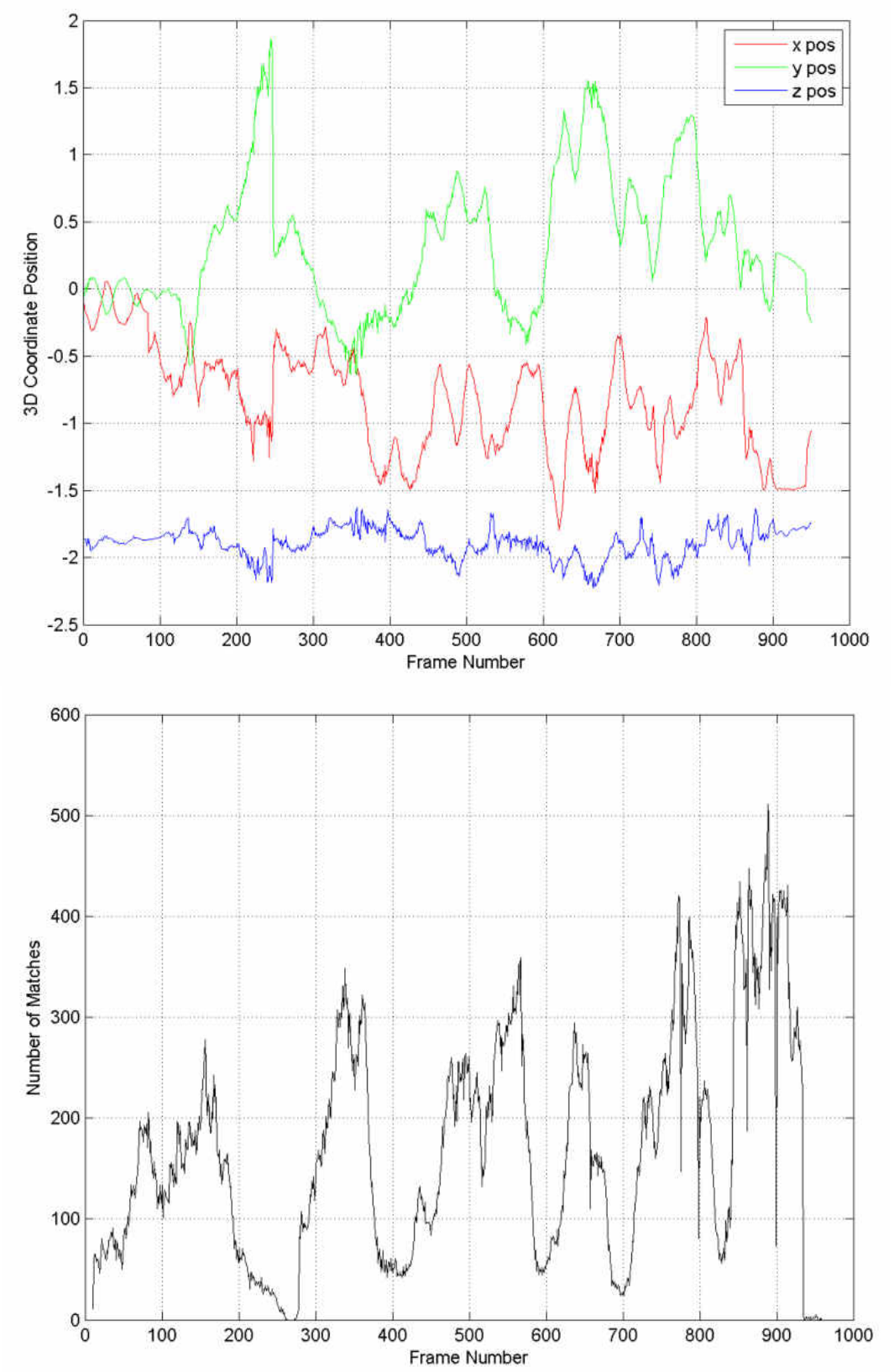


Figure 32: 3D position of the quadrotor in local NED (North-East-Down) coordinate frame generated from the SLAM during manual flight of the quad. The bottom figure shows the number of matches per frame.

Summery

In this section we presented an inertial aided scalable monocular visual SLAM algorithm that suitable for small mobile platforms equipped with a monocular camera and an inertial measurement unit. All the computations are performed onboard a small payload computer with limited power consumption and weight. Effectiveness of the algorithm is demonstrated on the dataset collected by flying a small quadcopter with the camera and the computer as well as during actual flight of the quadcopter. Results show that the algorithm is capable of running real time on the single board computer and robust to tracking failure, and it can be used for autonomous robot navigation and path planning without depending on the communication with a control station. However, the absence of any ground truth data does not allow us to perform any quantitative error analysis. Future works include evaluating the performance of the algorithm with a ground truth data as well as using the algorithm for autonomous control of the quadrotor using the pose estimation from the SLAM system.

CHAPTER V

TEST PLATFORM

In this chapter we describe the development of the test platform which includes both hardware and software development. This test platform is used for validating the developed SLAM algorithm and vision based control algorithms.

Hardware Development

A low cost quadrotor is built as part of the hardware development. The quadrotor was built using low cost off the shelf (OTS) equipment and assembled in the lab. The quadrotor is capable of carrying around 700 gms of payload with a flight time of 12 minutes. The payload consists of an onboard computer and a camera. The major components of the quadrotor are described below.

Quadrotor Frame

A DJI 450F flame wheel frame with X-configuration is used that includes the frame, 920 kV brushless DC motors, OPTO 30A electronic speed controllers (ESC) and two metal plates. The bottom plate works as a built-in power bus for connecting the battery to the speed controllers and the top plate is essentially used as the payload bay that holds the onboard computer and the camera. Two pairs of motors rotate in the opposite direction. Each motor with its attached 10” propellers contributes a maximum thrust of 420 gm.

Our design also includes powering the autopilot from the power bus thus eliminating need for external power source for the autopilot. The quadrotor is capable of flying with a 2000 gms total load including the payloads.

Figure 33: Image of the quadrotor. The pencil is used to get a perspective of the size of the quadrotor

Autopilot

An open source low cost autopilot called PIXHAWK is used in the quadrotor. The autopilot consists of a 32 bit 168 Hz ARM Cortex® M4 Processor, a 256 KB RAM, a 2 MB flash memory and runs a light weight real time UNIX operating system. The autopilot includes 5 UART ports and one I2C® interface in order to connect to external sensors and interfacing with payload computer onboard the quadrotor. It also houses the IMU, barometers and a magnetometer sensor suite.

Figure 34:Image of the PIXHAWK Autopilot

Onboard Computer

The onboard computer used in this project is the ODROID XU3 lite single board computer. The computer has a form factor of 94x70x18 mm and weighs about 72 gms. The maximum power consumption is about 20 watt making it a suitable candidate for applications where the payload capacity is very limited. The autopilot consists of a Cortex A15 1.8 GHz quad core and Cortex A7 quad core CPUs, A 2 GB low power DDR3 RAM at 933MHz with memory bandwidth of 14.9 GB/s. The peripherals include 4 USB ports which allows for serial interfacing with the autopilot as well as connecting with external sensors. The computer also runs a light weight UNIX operating system allowing for software development and compiling on the board thus eliminating the necessity of cross compilation.

Figure 35: The ODROID single board comuter used as the onboard computer.

Sensors

The sensors used in the project can be divided into two types: ones that are integrated or connected to the autopilot and the ones that are connected to the onboard computer. The first category of the sensors includes a 3 axis IMU (Inertial Navigation Unit) integrated in the autopilot, a barometric pressure sensor, an external sonar sensor and an external magnetometer. The IMU unit has a 3 axes accelerometer as well as a 3 axes gyroscope. Both the external sonar and the magnetometer sensors are connected to the autopilot using I2C interface. The camera is connected to the onboard computer using an usb interface.

Camera. The camera used in this project is a point gray Chameleon usb3 camera. The camera has a resolution of 1288x964 with a maximum frame rate of 30 fps. The camera is a global shutter camera with CCD sensor type. However, we use a 640x480 image size in this thesis with images captured at 10 Hz.

Figure 36: Image of Chameleon usb3 camera used as the front looking camera

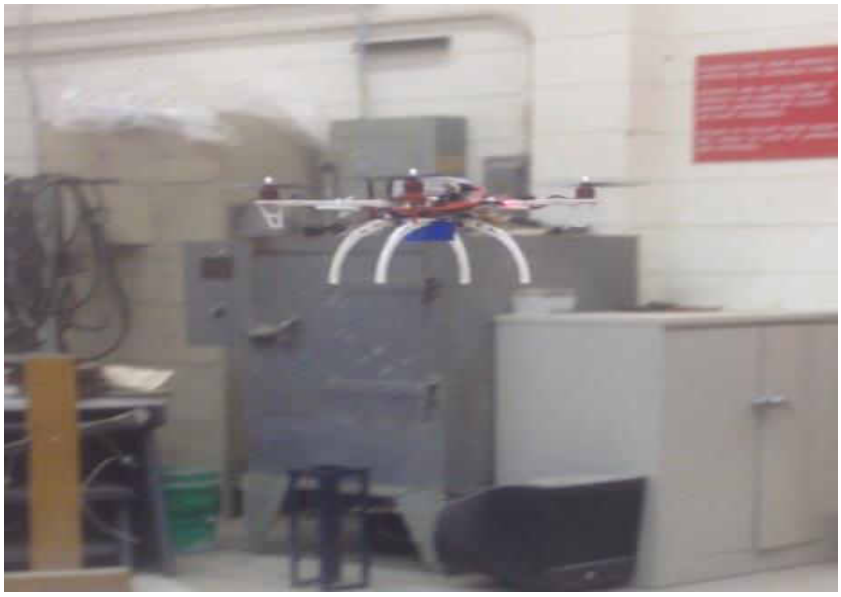


Figure 37: Image of the Developed Quadrotor during flight

Software Development

Software development for the test platform is divided into two sections: a) communication with the autopilot and ground station and b) control algorithm for the autonomous flights of the quadrotor.

Communication

The onboard computer communicates with the autopilot using a serial communication using two separate threads. In the receiving thread, the onboard computer reads the incoming sensor data and the vehicle's current state from the autopilot. The second thread is for sending vehicle position estimation from the SLAM system and control commands to the autopilot for autonomous takeoff, hover and motion. The commands are sent in the form of vehicle attitude and position. The stock firmware on the autopilot is also modified to send the sensor information at the desired rate as well as receive the control signals from the payload computer. The second form of communication developed is between the payload computer and a ground computer in order to visualize the performance of the SLAM system as well as having greater control of the events and failsafe for safe operation of the quadrotor. The communication with the ground computer is achieved using a UDP socket communication that runs on its own thread.

Control Algorithm

The control algorithm is responsible for the autonomous flight of the quadrotor using the SLAM system. The algorithm includes generating attitude and position commands for takeoff, hover and the motion of the vehicle. The control scheme in the autopilot does not allow for autonomous flying of the autopilot without receiving position estimation either from a GPS source or a vision system at a rate 2 Hz or more.

Autonomous Takeoff. Since the SLAM system does not start until the quadrotor is in hover mode, we don't have any position estimation during takeoff. As a result the algorithm continuously send attitude setpoints to the autopilot where the roll and pitch

setpoints are kept 0 in order for vertical takeoff. However the yaw setpoint is the current yaw position of the quad in the NED coordinate. The algorithm continuously receive the current yaw position updates from the autopilot and send back the same yaw position as the setpoint thus eliminating the possibility of quadrotor rotation during takeoff. In order to have a smooth takeoff, the thrust setpoints are generated using a sigmoid function where the steepness of the curve depends on the desired time to reach the maximum thrust. This time is a configurable parameter that can be adjusted to optimize the takeoff performance. The maximum desirable thrust is determined empirically by manually flying the quad at the desired height and recording the thrust required. The time required to reach the desired height from takeoff is also captured to estimate the time required to achieve the thrust in the autonomous mode.

Hover. The equation for altitude Control

$$\begin{aligned} m\ddot{x} &= -u\sin\phi \\ m\ddot{y} &= u\cos\phi\sin\theta \\ m\ddot{z} &= u\cos\theta\cos\phi - mg \\ \ddot{\theta} &= \tilde{\tau}_{\theta} \\ \ddot{\phi} &= \tilde{\tau}_{\phi} \\ \ddot{\psi} &= \tilde{\tau}_{\psi} \end{aligned} \tag{82}$$

Altitude can be stabilized with a feedback linearizable input u

$$u = \frac{m(-k_d\dot{z} - k_p(z - z_{ref}) + g)}{\cos\theta\cos\phi} \tag{83}$$

Software Architecture

The software architecture includes running the algorithm in the onboard computer, communicating with the autopilot and also a ground computer. Communication between

the autopilot and the onboard computer is achieved using a serial interface. The algorithm receives raw sensor data (IMU, barometric pressure etc.) from the autopilot and sends Quad pose estimation in the NED coordinate to the autopilot. In addition, the algorithm receives quad attitude data and flight mode from the autopilot. In a separate thread, the algorithm also communicates with a ground computer using a UDP socket. The algorithm takes command from the ground and in turns send signal to the autopilot for system arming, takeoff and hover. The algorithm sends the estimated quad pose to the ground station for visualization purposes. The serial and the socket communication run on their own thread.

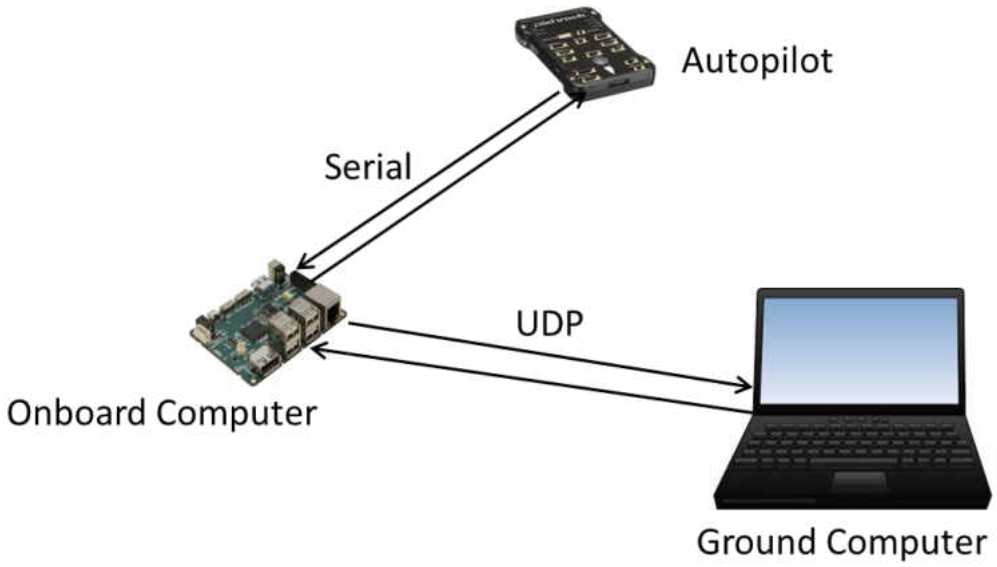


Figure 38: Communication between the onboard computer and the autopilot and also between the onboard computer and ground computer

CHAPTER VI

CONCLUSION

This thesis tackles the efficient vision based monocular SLAM by concentrating on real-time strategies for robust and locally accurate estimation of scene structure and camera motion with global consistency. Based on the previously developed key frame based SLAM algorithms a number of new techniques have been presented. The main achievements are:

- Development of a real time monocular SLAM algorithm that is proven to be robust and locally accurate. The new frontend and backend decoupling ensures that the algorithm is suitable even during fast exploration in a challenging outdoor environment with a front facing camera. Parallelization of the time critical components of the algorithm allows for real time performances without compromising the robustness of the algorithm. This is particularly important in computationally constraint systems.
- An efficient technique for monocular loop closure detection and correction using the same binary descriptor used to represent the 3D map points. This eliminates the requirement for using separate strong descriptors for loop closure detection in previous SLAM algorithms. In addition, the temporal and geometrical consistency check ensures the robustness of the correct loop closure detection.

- Development of an efficient re-localization algorithm by creating a novel tree structure for binary descriptors. The validity of the algorithm was demonstrated during the flight test of the algorithm.
- A monocular SLAM framework that can be extended and used for other purposes such as extracting semantic information from the generated 3D points cloud map.
- Augmenting the real time SLAM algorithm with IMU information when available in order to increase the robustness and efficiency of the algorithm as well as recovering the metric scale of the map. The IMU information is used for initial camera pose prediction and estimation before the pose is optimized using the nonlinear optimization method. This ensures a good initial estimation for the optimization to converge in a short amount of time. The effectiveness of the algorithm is demonstrated by performing real time SLAM generation on the onboard computer while flying the quad in an indoor environment.
- Development of a low cost quadrotor as a test platform. The quadrotor carries an ODROID onboard computer along with the camera during flight. A software framework is developed for communication and autonomous flight of the quadrotor that takes pose estimation and control signals from the onboard computer.

In order to evaluate the performance of the algorithm, initially we have collected data in an indoor environment and run the algorithm on the indoor dataset. The results clearly show the resemblance in the generated 3D structure with the actual structure of the

environment. However, the lack of ground truth prevents us to estimate the amount of error. Usually the accuracy in the algorithm is compared to the ground truth data generated in a lab environment using absolute pose estimation methods such as vicon [70] systems. A similar method is required for complete evaluation of our algorithm.

The algorithm is also validated in the publicly available KITTI dataset. The KITTI dataset provides stereo images captured with a front facing stereo camera mounted on top of a vehicle. Our algorithm was able to localize and generate 3D point cloud map in challenging outdoor environments.

In order to validate the applicability for small mobile robots, the algorithm was implemented on the ODROID single board computer carried by the small quadrotor developed as the test platform. Initially the algorithm was validated on the collected dataset and then tested during manual flight of the quadrotor. The results are shown in the previous chapter. However, the lack of ground truth data does not allow us to quantify the error in the estimated quad poses.

Discussion & Future Work

Monocular SLAM algorithm provides an efficient solution to the localization and mapping problem for small mobile platforms with strict payload constraints. This system offers an alternative solution when GPS signal is unavailable or lost thus increases the application range for small robots. However, it is also more challenging since depth information is not readily available and need to be estimated from inter frame translation over time. Any error in the relative motion estimation is accumulated in depth estimation from triangulation. One way to reduce the amount of error is to use large number of 3D

map features for camera pose estimation. However, it comes at a cost of increasing computational complexity as the computational cost increases linear to quadratic with the number of map points. In addition, using 3D point cloud as the map features also causes the SLAM algorithm to fail in environments with lack of distinctive features. This is particularly true in manmade environment where surfaces with uniform colors, repeated patterns and reflective surfaces cause distinctive feature selection and tracking impossible.

Figure 39: Example of an environment with reflective surfaces. The algorithm fails to detect and track sufficient distinctive corner points for accurate camera pose estimation.

One alternative approach is to use line features for the map representation. However line features are not as easy to handle as point features. Line features are 1-dimensional features and also it is hard to determine the end points. In addition they are only prevalent in manmade environment causing the SLAM algorithm to be applicable only in city like environment.

Most recent structure from motion algorithms are focused on dense reconstruction of the environment and extraction of semantic information of the environment. Most of these algorithms requires training and are used in an offline manner with powerful computers thus are not suitable to be applied for real time SLAM on computationally constraint systems.

In our view, the developed algorithms can be considered as the starting point for future development and there are a few paths that can be taken to improve the robustness of the vision based pose estimation, rich environment representation as well as motion planning. A hybrid of line and point features based localization and mapping can certainly increase the robustness of the algorithm. Parallel implementation of the feature detection and tracking along with a joint pose optimization would keep the computational cost under a limit.

Another hybrid method for rich environment representation using a combination of sparse and dense representation might be useful in some applications such as obstacle avoidance and semantic information retrieval where only the nearby objects are reconstructed with a dense method whereas objects that are far from the robot can be represented using sparse 3D points or lines similar to wire models.

APPENDIX

Mathematical group and its properties

A group is a mathematical structure consisting of a set G together with a binary operation $\circ : G \times G \to G$. The properties of a mathematical group are

Closure: If $g_1, g_2 \in G,$ then $g_1 \circ g_2 \in G$

Identity: The group has an identity element such that $g \circ e = e \circ g = g$ for every $g \in G$

Inverse: For each $g \in G$, there exist a unique inverse $g' \in G$, such that

$g \circ g^{-1} = g^{-1} \circ g = e$

Associativity: If $g_1, g_2, g_3 \in G$ then $(g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$

Group Actions

Group action refers to a group element acting on an element of a manifold M. For example, a left action of G on M is defined as a smooth map $\phi : G \times M \mapsto M$ such that:

1. the identity element e has no effect, i.e. $\phi(e, p) = p$ composing two actions can be combined into one action: $\phi(g, \phi(h, p)) = \phi(gh, p)$
2. for matrix Lie groups, the usual action is the matrix vector multiplication on R^n

Tangent Space

We start with some basic definitions from multivariate calculus that are used in the definition of the tangent space.

A Smooth Path. Let $X \subset \Re^m$ is an m-dimensional vector space and let $t \in [a,b]$ is a real interval. Now a smooth path $P : \Re \to X : t \to P(t)$ is a differentiable function from the real interval $[a,b]$ to the vector space. Now consider a $n \times n$ matrix as a member of a vector space $\Re^m, m = n^2$, the smooth path maps real interval to the matrix groups. For example, a smooth path for $SO(3)$ can be written as

$$R_x : [-\pi, \pi] \to SO(3), R_x(t) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(t) & -\sin(t) \\ 0 & \sin(t) & \cos(t) \end{bmatrix}$$

(84)

Where $R_x(t)$ describes the rotation around x-axis at time t.

Tangent Vectors of a Space. Let $X \subset \Re^n$ be a n-dimensional vector space and there exists a path P such that $P(0) = y; y \in X$. Now x is a tangent vector of X at y , if $x = \frac{\partial}{\partial t} P(t) |_{t=0}$. For example,

$$\frac{\partial}{\partial t} R_x(t) |_{t=0} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\sin(0) & -\cos(0) \\ 0 & \cos(0) & -\sin(0) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \qquad (85)$$

is the tangent vector of $R_x(t)$ and therefore a tangent vector of $SO(3)$. Since $R_x(0) = I$, it is a tangent vector at the identity. Now the set of all tangent vectors at y spans a vector space: the tanget space at y. Specifically, the tangent vectors at the identity of a Lie group G spans a vector space g which is the tangent space at the identity. The tangent space can be identified with the space of directional derivative operators along smooth paths through y. As an example, we show the construction of the tangent space for the group SO(3). The matrices $A \in SO(3)$ are orthogonal, so they satisfy $AA^T = I$. Assuming a smooth path $A = A(t)$ with $A(0) = I$. Now differentiating the equation $A(t)A(t)^T = I$, we get

$$A'(t)A(t)^T + A(t)A'(t)^T = 0 \tag{86}$$

for $t = 0$, $A(0) = I$ and the equation becomes

$$\begin{aligned} A'(0) + A'(0)^T &= 0 \\ A'(0) &= -A'(0)^T \end{aligned} \tag{87}$$

Now $\Omega = A'(0)$ is a skew symmetric matrix since $\Omega = -\Omega^T$. Thus the tangent space for the group $SO(3)$ is spanned by three skew symmetric matrices which corresponds to infinitesimal rotations around three rotations axes. These set of matrices are called generators for the group.

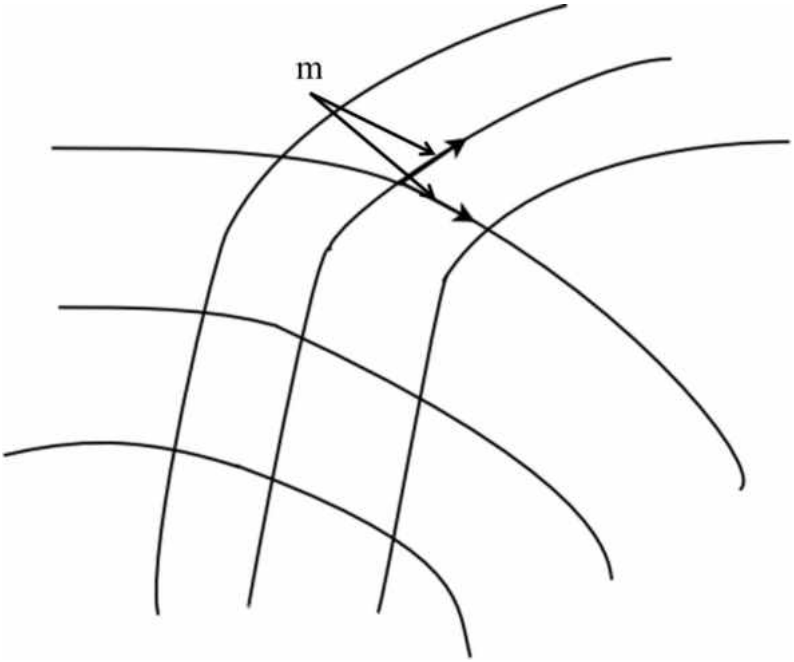


Figure 40: Example of a 2D manifold in a 3D space. The tangent vectors are represented by m

BIBLIOGRAPHY

1. Qadir, Ashraf, Jeremiah Neubert, and William Semke. "On-board visual tracking with unmanned aircraft system (uas)." arXiv preprint arXiv:1203.2386 (2012).
2. Zhang, Chunhua, Dan Walters, and John M. Kovacs. "Applications of Low Altitude Remote Sensing in Agriculture upon Farmers' Requests–A Case Study in Northeastern Ontario, Canada." PloS one 9, no. 11 (2014): e112894.
3. OpenCv computer vision library: http://opencv.org/
4. A. Markus, S. Weiss, and R. Siegwart. "Onboard imu and monocular vision based control for mavs in unknown in and outdoor environments." In *Robotics and automation (ICRA), 2011 IEEE international conference on*. IEEE, 2011.
5. Rosten, Edward, and Tom Drummond. "Machine learning for high-speed corner detection." In *Computer Vision–ECCV 2006*, pp. 430-443. Springer Berlin Heidelberg, 2006
6. Klein, Georg, and David Murray. "Parallel tracking and mapping for small AR workspaces." In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pp. 225-234. IEEE, 2007.
7. Strasdat, Hauke, J. M. M. Montiel, and Andrew J. Davison. "Scale Drift-Aware Large Scale Monocular SLAM." *Robotics: Science and Systems*. Vol. 2. No. 3. 2010.
8. Engel, Jakob, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-scale direct monocular SLAM." In *Computer Vision–ECCV 2014*, pp. 834-849. Springer International Publishing, 2014.

9. Kümmerle, Rainer, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. "g2o: A general framework for graph optimization." In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3607-3613. IEEE, 2011.
10. Strasdat, Hauke, Andrew J. Davison, J. M. M. Montiel, and Kurt Konolig. "Double window optimisation for constant time visual SLAM." In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2352-2359. IEEE, 2011.
11. Strasdat, Hauke, José MM Montiel, and Andrew J. Davison. "Visual SLAM: why filter?." *Image and Vision Computing* 30, no. 2 (2012): 65-77.
12. Eade, Ethan, and Tom Drummond. "Scalable monocular SLAM." In*Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 1, pp. 469-476. IEEE, 2006.
13. Davison, Andrew J., Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. "MonoSLAM: Real-time single camera SLAM." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 29, no. 6 (2007): 1052-1067.
14. E. Eade. Monocular simultaneous localisation and mapping. PhD Thesis, 2008
15. Clemente, Laura A., Andrew J. Davison, Ian D. Reid, José Neira, and Juan D. Tardós. "Mapping Large Loops with a Single Hand-Held Camera." In*Robotics: Science and Systems*, vol. 2, p. 2. 2007.
16. Konolige, Kurt, and Motilal Agrawal. "FrameSLAM: From bundle adjustment to real-time visual mapping." *Robotics, IEEE Transactions on* 24, no. 5 (2008): 1066-1077.

17. Leutenegger, Stefan, Margarita Chli, and Roland Y. Siegwart. "BRISK: Binary robust invariant scalable keypoints." In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2548-2555. IEEE, 2011.
18. Shen, Shaojie, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. "Vision-Based State Estimation and Trajectory Control Towards High-Speed Flight with a Quadrotor." In *Robotics: Science and Systems*, vol. 1. 2013.
19. Triggs, Bill, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. "Bundle adjustment—a modern synthesis." In *Vision algorithms: theory and practice*, pp. 298-372. Springer Berlin Heidelberg, 1999.
20. Mur-Artal, Raul, J. M. M. Montiel, and Juan D. Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." *Robotics, IEEE Transactions on* 31, no. 5 (2015): 1147-1163.
21. Kneip, Laurent, Margarita Chli, and Roland Siegwart. "Robust real-time visual odometry with a single camera and an imu." In *BMVC*, pp. 1-11. 2011.
22. Gálvez-López, Dorian, and Juan D. Tardos. "Bags of binary words for fast place recognition in image sequences." *Robotics, IEEE Transactions on* 28, no. 5 (2012): 1188-1197.
23. Galvez-Lopez, Dorian, and Juan D. Tardos. "Real-time loop detection with bags of binary words." In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 51-58. IEEE, 2011.
24. Hartley, Richard, and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

25. Cummins, Mark, and Paul Newman. "FAB-MAP: Probabilistic localization and mapping in the space of appearance." *The International Journal of Robotics Research* 27, no. 6 (2008): 647-665.
26. Konolige, Kurt, James Bowman, J. D. Chen, Patrick Mihelich, Michael Calonder, Vincent Lepetit, and Pascal Fua. "View-based maps." *The International Journal of Robotics Research* (2010).
27. Angeli, Adrien, David Filliat, Stéphane Doncieux, and Jean-Arcady Meyer. "Fast and incremental method for loop-closure detection using bags of visual words." *Robotics, IEEE Transactions on* 24, no. 5 (2008): 1027-1037.
28. Sivic, Josef, and Andrew Zisserman. "Video Google: A text retrieval approach to object matching in videos." In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 1470-1477. IEEE, 2003.
29. Nister, David, and Henrik Stewenius. "Scalable recognition with a vocabulary tree." In *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, vol. 2, pp. 2161-2168. IEEE, 2006.
30. Galvez-Lopez, Dorian, and Juan D. Tardos. "Real-time loop detection with bags of binary words." In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pp. 51-58. IEEE, 2011.
31. Lowe, David G. "Distinctive image features from scale-invariant keypoints."*International journal of computer vision* 60, no. 2 (2004): 91-110.
32. Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. "Surf: Speeded up robust features." In *Computer vision–ECCV 2006*, pp. 404-417. Springer Berlin Heidelberg, 2006.

33. Arthur, David, and Sergei Vassilvitskii. "k-means++: The advantages of careful seeding." In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 1027-1035. Society for Industrial and Applied Mathematics, 2007.
34. Muja, Marius, and David G. Lowe. "Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration." *VISAPP (1)* 2 (2009): 331-340.
35. Nistér, David, Oleg Naroditsky, and James Bergen. "Visual odometry." In*Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1, pp. I-652. IEEE, 2004.
36. Piniés, Pedro, Lina María Paz, Dorian Gálvez-López, and Juan D. Tardós. "CI-Graph simultaneous localization and mapping for three-dimensional reconstruction of large and complex environments using a multicamera system." *Journal of Field Robotics* 27, no. 5 (2010): 561-586.
37. Kneip, Laurent, Davide Scaramuzza, and Roland Siegwart. "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation." In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 2969-2976. IEEE, 2011.
38. Scaramuzza, Davide, Friedrich Fraundorfer, Marc Pollefeys, and Roland Siegwart. "Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints." In *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 1413-1419. IEEE, 2009.
39. Mair, Elmar, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. "Adaptive and generic corner detection based on the accelerated segment

test." In *Computer Vision–ECCV 2010*, pp. 183-196. Springer Berlin Heidelberg, 2010.

40. Grisetti, Giorgio, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. "A tutorial on graph-based SLAM." *Intelligent Transportation Systems Magazine, IEEE* 2, no. 4 (2010): 31-43.
41. Umeyama, Shinji. "Least-squares estimation of transformation parameters between two point patterns." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 4 (1991): 376-380.
42. Fischler, Martin A., and Robert C. Bolles. "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography." *Communications of the ACM* 24, no. 6 (1981): 381-395.
43. Geiger, Andreas, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? the kitti vision benchmark suite." In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, pp. 3354-3361. IEEE, 2012.
44. Muja, Marius, and David G. Lowe. "Fast matching of binary features." InComputer and Robot Vision (CRV), 2012 Ninth Conference on, pp. 404-410. IEEE, 2012.
45. Rublee, Ethan, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: an efficient alternative to SIFT or SURF." In Computer Vision (ICCV), 2011 IEEE International Conference on, pp. 2564-2571. IEEE, 2011.
46. Calonder, Michael, Vincent Lepetit, Christoph Strecha, and Pascal Fua. "Brief: Binary robust independent elementary features." Computer Vision–ECCV 2010 (2010): 778-792.

47. Lupton, Todd, and Salah Sukkarieh. "Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions." *Robotics, IEEE Transactions on* 28, no. 1 (2012): 61-76.

48. Roumeliotis, Stergios I., Andrew E. Johnson, and James F. Montgomery. "Augmenting inertial navigation with image-based motion estimation." In*Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 4, pp. 4326-4333. IEEE, 2002.

49. Mourikis, Anastasios I., Nikolas Trawny, Stergios I. Roumeliotis, Andrew Edie Johnson, and Larry Matthies. "Vision-Aided Inertial Navigation for Precise Planetary Landing: Analysis and Experiments." In *Robotics: Science and Systems*. 2007.

50. Dryanovski, Ivan, William Morris, and Jizhong Xiao. "An open-source pose estimation system for micro-air vehicles." In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pp. 4449-4454. IEEE, 2011.

51. Bachrach, Abraham, Samuel Prentice, Ruijie He, and Nicholas Roy. "RANGE–Robust autonomous navigation in GPS-denied environments."*Journal of Field Robotics* 28, no. 5 (2011): 644-666.

52. Shen, Shaojie, Nathan Michael, and Vijay Kumar. "Autonomous multi-floor indoor navigation with a computationally constrained MAV." In *Robotics and automation (ICRA), 2011 IEEE international conference on*, pp. 20-25. IEEE, 2011.

53. Shen, Shaojie, Yash Mulgaonkar, Nathan Michael, and Vijay Kumar. "Vision-Based State Estimation and Trajectory Control Towards High-Speed Flight with a Quadrotor." In *Robotics: Science and Systems*, vol. 1. 2013.

54. Lucas, Bruce D., and Takeo Kanade. "An iterative image registration technique with an application to stereo vision." In *IJCAI*, vol. 81, pp. 674-679. 1981.

55. Shi, Jianbo, and Carlo Tomasi. "Good features to track." In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pp. 593-600. IEEE, 1994.

56. Weiss, Stephan, Markus W. Achtelik, Simon Lynen, Margarita Chli, and Roland Siegwart. "Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments." In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 957-964. IEEE, 2012.

57. Fraundorfer, Friedrich, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. "Vision-based autonomous mapping and exploration using a quadrotor MAV." In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 4557-4564. IEEE, 2012.

58. Kelly, Jonathan, Srikanth Saripalli, and Gaurav Sukhatme. "Combined visual and inertial navigation for an unmanned aerial vehicle." In *Field and Service Robotics*, pp. 255-264. Springer Berlin/Heidelberg, 2008.

59. Jones, Eagle S., and Stefano Soatto. "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach." *The International Journal of Robotics Research* 30, no. 4 (2011): 407-430.

60. Martinelli, Agostino. "Closed-Form Solutions for Attitude, Speed, Absolute Scale and Bias Determination by Fusing Vision and Inertial Measurements." (2011).

61. Kelly, Jonathan, and Gaurav S. Sukhatme. "Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration." *The International Journal of Robotics Research* 30, no. 1 (2011): 56-79.

62. Kneip, Laurent, Margarita Chli, and Roland Siegwart. "Robust real-time visual odometry with a single camera and an imu." In *BMVC*, pp. 1-11. 2011.

63. Sameer Agarwal and Keir Mierle and Others "Ceres Solver", http://ceres-solver.org.

64. Cummins, Mark, and Paul Newman. "Highly scalable appearance-only SLAM-FAB-MAP 2.0." In *Robotics: Science and Systems*, vol. 1, pp. 12-18. 2009.

65. Mei, Christopher, Gabe Sibley, Mark Cummins, Paul M. Newman, and Ian D. Reid. "A Constant-Time Efficient Stereo SLAM System." In *BMVC*, pp. 1-11. 2009.

66. https://www.threadingbuildingblocks.org/

67. Sünderhauf, Niko, Kurt Konolige, Simon Lacroix, and Peter Protzel. "Visual odometry using sparse bundle adjustment on an autonomous outdoor vehicle." In *Autonome Mobile Systeme 2005*, pp. 157-163. Springer Berlin Heidelberg, 2006.

68. Salakhutdinov, Ruslan, and Geoffrey Hinton. "Semantic hashing."*International Journal of Approximate Reasoning* 50, no. 7 (2009): 969-978.

69. Zitnick, C. Lawrence. "Binary coherent edge descriptors." In *Computer Vision–ECCV 2010*, pp. 170-182. Springer Berlin Heidelberg, 2010.

70. http://www.vicon.com/home