VISUAL SLAM FOR MICRO AERIAL VEHICLES

Siddharth Shankar Jha

VISUAL SLAM FOR MICRO AERIAL VEHICLES

Report submitted to
Indian Institute of Technology Kharagpur
for the award of the degree
of

Fourth Year Dual Degree

in

Electrical Engineering

by

Siddharth Shankar Jha

Under the guidance of

Prof. Alok Kanti Deb

(Department of Electrical Engineering)

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
APRIL 2018

DECLARATION

I certify that

- The work contained in this Report is original and has been done by me under the guidance of my supervisor.
- The work has not been submitted to any other Institute for any degree or diploma.
- I have followed the guidelines provided by the Institute in preparing the Report.
- I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
- Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the Report and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Signature of the Student

CERTIFICATE

This is to certify that the Report entitled **Visual SLAM for micro aerial vehicles**, submitted by **Siddharth Shankar Jha** to Indian Institute of Technology Kharagpur, is a record of bonafide research work under my supervision and is worthy of consideration for the award of the degree of Bachelor of Technology of the Institute.

Date:
Place:

Prof. Alok Kanti Deb
Department of Electrical Engineering

ACKNOWLEDGEMENT

I would like to dedicate my project thesis to Indian Institute of Technology, Kharagpur for making me what I am now in terms of knowledge, personality and social status and also for providing best academic facilities which helped me to complete this project.

I would like to extend my gratitude and sincere thanks to my supervisor **Prof. Alok Kanti Deb** (IIT Kharagpur) Electrical Engineering Department, for supervising the Project report on **Visual SLAM on micro aerial vehicles** as without his constant motivation and support during my work, this would not have been possible. I truly appreciate and value the esteemed guidance and encouragement from the beginning to the end of this work. I am thankful to my faculty advisor, Prof. Debdoot Sheet for his help and support in suggesting the right direction in academics.

Last but not least, I would like to thank my parents for their constant support and love throughout my life.

Siddharth Shankar Jha

ABSTRACT

This work describes the development of a Visual Simultaneous Localization And Mapping (VSLAM) framework that is light on resources and is fast enough to run on embedded computers and thus can be run from atop a quadcopter. The entire development of the VSLAM frontend has been done from scratch. Chapter 1 introduces the SLAM problem and describes why it is such a coveted research topic. Chapter 2 lays the basic mathematical foundation of the SLAM problem. Chapter 3 deals with the actual implementation of the visual reconstruction and loop closing software. Chapter 4 describes the different modes of processing of the generated point cloud data. Finally in Chapter 5, a conclusion of the total work done until now is presented and the future implementation is discussed.

CONTENTS

Title Page		i
Certificate by the Supervisor		iv
Acknowledgements		v
Abstract		vi
Contents		vii
Chapter 1	Introduction	1
Chapter 2	Probabilistic SLAM Problem Definition	3
Chapter 3	Visual SLAM	4
Chapter 4	Point Cloud Processing	15
Chapter 5	Conclusions and Future Work	18
References		20
Appendix A	The Codebase	21

Chapter 1: Introduction

1.1 Overview

The SLAM (Simultaneous Localization and Mapping) problem is a very challenging and rigorously researched problem in the world of robotic mapping and navigation. SLAM comprises the simultaneous estimation of the state of a robot equipped with on-board sensors, and the construction of a model (the map) of the environment that the sensors are perceiving. It initially appears to be a chicken-and-egg problem but there have been several approaches suggested that solve the problem.

This project aims to solve the problem with a unique approach that is robot-independent and requires just a calibrated monocular camera system in addition to one real world scale fixing accessory input, such as an Inertial Measurement Unit (IMU) or an IR sensor (present in a depth camera).

1.2 History and Literature Review

Earlier SLAM approaches, like in [1] & [2] used extended Kalman filters and particle filters for solving the problem. Filtering-based approaches may achieve faster processing due to their continuous marginalization of past states, but early fix of linearization points may results in sub-optimal results.

The current de-facto standard formulation of the SLAM estimation backend is Maximum a posteriori (MAP) estimation using Nonlinear Graph Optimization. It was proposed first in the paper [3]. In [4], it was stated that a monocular visual-inertial setup is the minimum sensor suite that allows both autonomous flight and sufficient environment awareness. Figure 1.1 shows a visual-inertial SLAM method with MAP estimation technique.

There has been a lot of work done on the control of quadcopters, including altitude control using downward facing distance sensors. Altitude control was demonstrated using vision-based approaches in [5], but it didn't generate an explicit map of the environment.

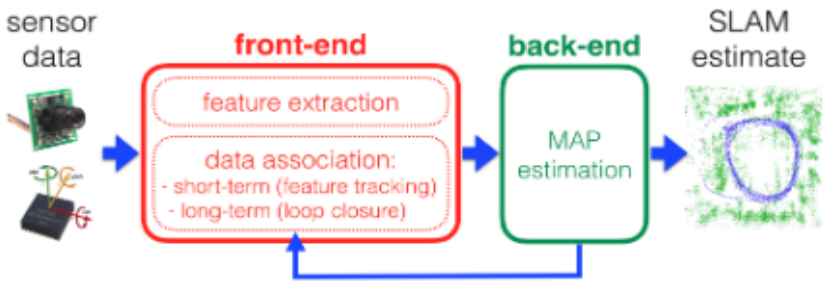


Figure 1.1: Basic block diagram of the SLAM problem

The Visual SLAM problem has many related sister problems. For example, visual odometry is the problem in which the global map optimization and loop closing is not taken care of. Structure from motion (SfM) is a technique to estimate camera motion and 3D structure of the environment in a batch manner, but it usually is performed (several images of a particular structure are recorded and the correspondences between the images are used to develop a detailed 3D reconstruction of the environment.

The problems of visual odometry and SLAM can be solved in a similar manner, but as a rule of thumb the visual odometry estimates start deteriorating over time due to accumulation of error.

The Motivation behind this work is to develop a lightweight visual SLAM system that can run on an embedded processor, such as the Odroid C4 and consumes low amount of memory for on-line processing of scene data. The final objective should be to make a robust visual-inertial SLAM system that can be used to control the altitude of a real quadcopter in real-time by generating fast occupancy maps of environment and segmenting them.

Chapter 2: Probabilistic SLAM Problem Definition

Given a series of sensor observations o_t over discrete time steps , the SLAM problem is to compute an estimate of the agent's location x_t and a map of the environment m_t . All quantities are usually probabilistic, so the objective is to compute:

$$P(m_t, x_t | o_{1:t}) \tag{2.1}$$

Applying Bayes' rule gives a framework for sequentially updating the location posteriors, given a map and a transition function $P(x_t | x_{t-1})$

$$P(x_t | o_{1:t}, m_t) = \sum_{m_{t-1}} P(o_t | x_t, m_t) \sum_{x_{t-1}} P(x_t | x_{t-1}) P(x_{t-1} | o_{1:t-1}, m_{t-1}) \tag{2.2}$$

Similarly the map can be updated sequentially by

$$P(m_t | x_t, o_{1:t}) = \sum_{x_t} \sum_{m_t} P(m_t | x_t, m_{t-1}, o_t) P(m_{t-1}, x_t | o_{1:t-1}, m_{t-1}) \tag{2.3}$$

In this project we deal with Visual SLAM. So the observations o_t become location of features in monocular (lacking depth) images from a calibrated camera. In section 3, all aspects of visual SLAM are described.

Chapter 3: Visual SLAM

The basic framework followed by all major visual SLAM algorithms is mainly composed of three modules as follows:

1. Initialization: To define the coordinate system, identify parts of image suitable for pose reconstruction. Construct an initial environment in the global coordinate system.

2. Tracking: The reconstructed map is tracked in the image to estimate the camera pose of the image with respect to the map. 2D–3D correspondences between the image and the map are first obtained from feature matching or feature tracking in the image. Then, the camera pose is computed from the correspondences by solving the Perspective-n-Point (PnP) problem.

3. Mapping: the map is expanded by computing the 3D structure of an environment when the camera observes unknown regions where the mapping is not performed before.

In addition to these, another important component of visual SLAM is global map optimization. In this process, the map is refined by considering the consistency of whole map information. When a map is revisited such that a starting region is captured again after some camera movement, reference information that represents the accumulative error from the beginning to the present can be computed. Then, a loop constraint from the reference information is used as a constraint to suppress the error in the global optimization. This is called a loop closure and is integral for maintaining the consistency of the solution.

This section describes all the aforementioned steps and includes implementation details. The steps (in order) are:

- Feature Detection
- Tracking and Feature Trail Generation
- Essential Matrix computation & decomposition
- Triangulation of initial trails
- Global Graph Optimization
- Loop Closures

3.1 Feature Detection

For estimating the motion between two images, high quality feature points are tracked between consecutive images. Harris corners are used in the current implementation, being fast to compute track. The input image is searched for patches that produce a large $E(u,v)$ value defined as:

$$E(u,v)=\sum_{x,y} w(x,y)[I(x+u,y+v)-I(x,y)]^2 \tag{3.1}$$

where E is the difference between the original and the moved window, u is the window's displacement in the x direction, v is the window's displacement in the y direction, $w(x,y)$ is the window at position (x,y). This acts like a mask, ensuring that only the desired window is used. I is the intensity of the image at a position (x,y) and hence $I(x+u,y+v)$ is the intensity of the moved window.

Corners generally satisfy this E maximization criteria well. For getting Harris corners, the terms inside the square bracket are expanded using Taylor series and get an approximation on $E(u,v)$

$$E(u,v)\approx\begin{bmatrix}u & v\end{bmatrix} M \begin{bmatrix}u\\ v\end{bmatrix} \tag{3.2}$$

where M is defined as:

$$M=\sum w(x,y)\begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

While this may not produce perfect results, the approximation is very close to the real value and hence satisfies real-world needs. It was figured out that eigenvalues of the matrix can help determine the suitability of a window. A score, R is calculated for each window:

$$R=det(M)-k(trace(M))^2 \tag{3.3}$$

All windows that have a score R greater than a certain value are chosen as corners.

3.2 Tracking and Feature Trails

Optical flow is the pattern of apparent motion of image objects between two consecutive frames caused by the movement of object or camera. It is a 2D vector field where each vector is a displacement vector showing the movement of points from first frame to second. Optical flow thus can be used to track points from one image to its successor. It works on a few assumptions

1. The pixel intensities of an object do not change between consecutive frames.
2. Neighbouring pixels have similar motion.

Consider a pixel $I(x, y, t)$ in first frame. It moves by distance (dx, dy) in next frame taken after dt time. So since those pixels are the same and intensity does not change,

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{3.4}$$

Then taking Taylor series approximation of right-hand side, removing common terms and dividing by dt :

$$f_x u + f_y v + f_t = 0 \tag{3.5}$$

where:

$$f_x = \partial f / \partial x ; f_y = \partial f / \partial y$$

$$u = dx / dt ; v = dy / dt$$

which is the optical flow equation.

The Lucas-Kanade method takes a 3x3 patch around the corner, and enforces a constraint of equal motion on them to obtain a solution of (u, v) .

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \tag{3.6}$$

Once the tracked points are located in the second image, a technique called forward-backward optical flow [6] is used to track-back the tracked points to the first image and observe the disparity between the original and the tracked-then-tracked-back point. If the disparity is below a certain threshold, the point is chosen as a suitable corner for tracking applications.

For the tracking , the concept of feature trails is introduced.

1. All the features that were successfully tracked are considered as potential trails. They are stored in a vector.
2. Meanwhile, new features are also detected in the second image. To ensure that no two features correspond to the same corner in the image, a grid occupancy mask is used while detecting features. All grid cells with active feature trails or potential trails, are not considered for detection.
3. As soon as a potential trail reaches a displacement greater than threshold of pixels from it's starting point, it is removed from the potential trail vector and added to a feature trail vector.

4. For subsequent images, all feature trails and potential trails are tracked via forward – backward optical flow and those failing the tracking are deleted.

The mentioned tracking algorithm was run on the EuRoC dataset, a widely used SLAM benchmark. Figure 3.1 shows the result of the tracking algorithm on the dataset.

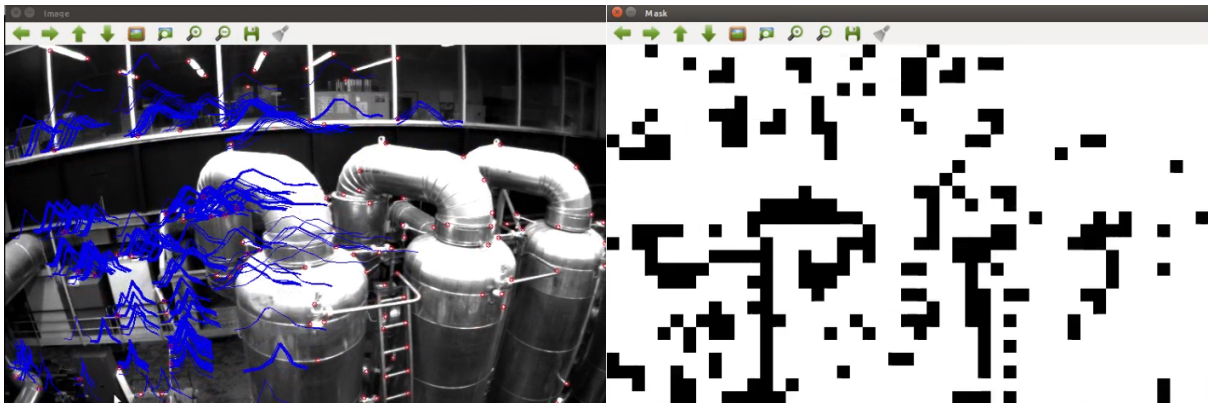


Figure 3.1: Result of Tracking on EuRoC dataset. The blue lines are the feature trails, the grid occupancy mask is on the right

3.3 Essential Matrix computation & decomposition

Epipolar geometry is the intrinsic projective geometry between two views of cameras. It is independent of scene structure, and only depends on the cameras' internal parameters and relative pose. The fundamental matrix F encapsulates this intrinsic geometry. It is a 3x3 matrix of rank 2. If a point in 3-space M is imaged as x in the 1st view, and x' in the second, then the image points satisfy the relation $x'^{T}Fx=0$. The essential matrix is the specialization of the fundamental matrix to the case of normalized image coordinates.

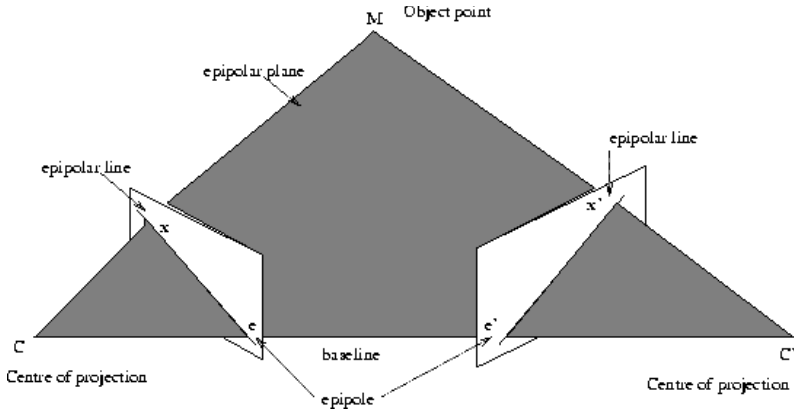


Figure 3.2: Epipolar Geometry & Constraints

Figure 3.2 shows the epipolar constraints that are imposed on the motion of the images. As seen in the figure, the centre of projection, epipole, epipolar line and the object point are all coplanar. This constraint is realized in the essential matrix. For evaluating the essential matrix, Nister's five-point algorithm [7] is used, that asks for 5 non-coplanar correspondences between two views from the same

camera. More points if available are used to verify and perfect the model by the use of a RANSAC(RANdom SAmple Consensus) [8] based outlier rejection approach.

Decomposition of the essential matrix gives the rotation and translation between the two images, where the translation is defined up to an unknown scale factor. This can be evaluated using Singular Value Decomposition (SVD).

An SVD of E gives

$$E = U \Sigma V^T$$

where U and V are orthogonal 3 × 3 matrices and Σ is a 3 × 3 diagonal matrix with

$$\Sigma = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The diagonal entries of Σ are the singular values of E which, according to the internal constraints of the essential matrix, must consist of two identical and one zero value. Define W as,

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

Then a solution for R and t may be calculated as:

$$[t]_x = UW\Sigma U^T \tag{3.7}$$

$$R = UW^{-1}V^T \tag{3.8}$$

3.4 Triangulation of initial trails

Once the points have been tracked and the feature trails have been initialized, a heuristic to choose trails with sufficient motion and in sufficient density is chosen to triangulate suitable trails for the initialization of the 3D reconstruction and odometry recovery. Triangulation is performed using the estimate of the distance between the two frames (the two viewpoints). The resultant points are arbitrarily scaled, and the scale needs to be fixed in order for the odometry to work properly. The scale is fixed by taking the average of the ratio of distances between original points and the newly triangulated points. This is demonstrated in Figure 3.3.

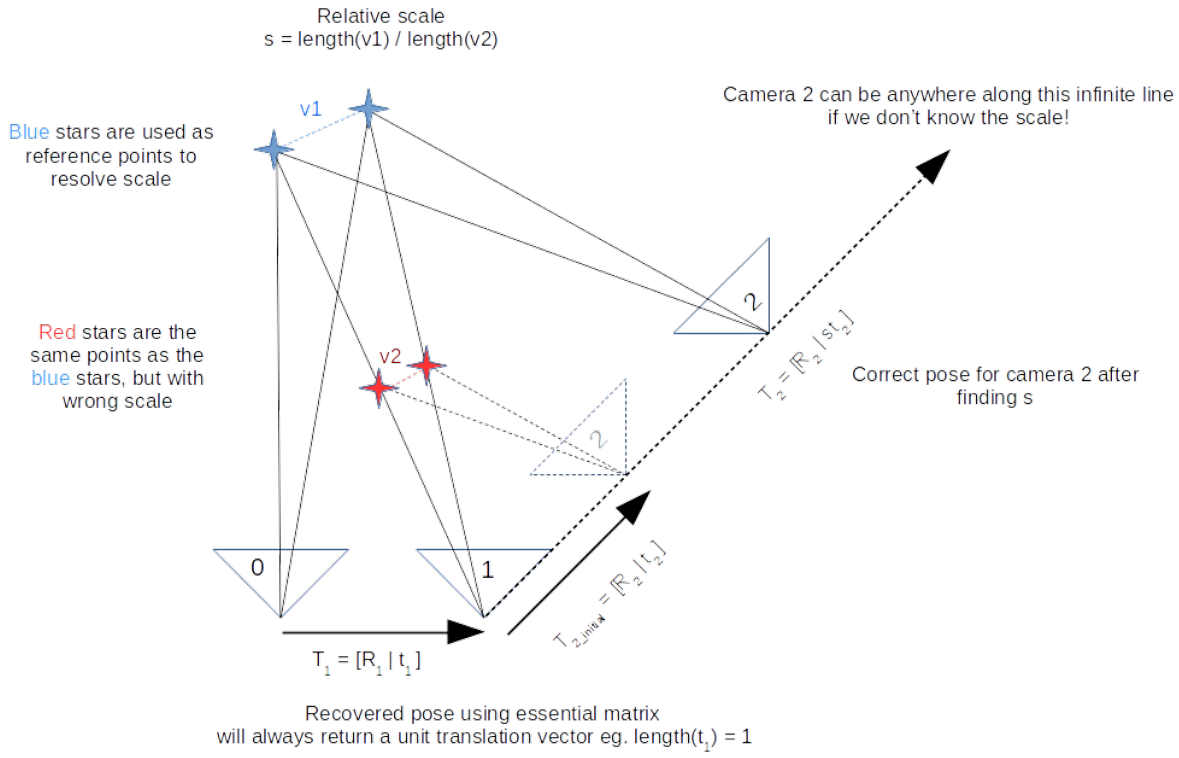


Figure 3.3: Relative Scaling Methodology

3.5 Global Graph Optimization

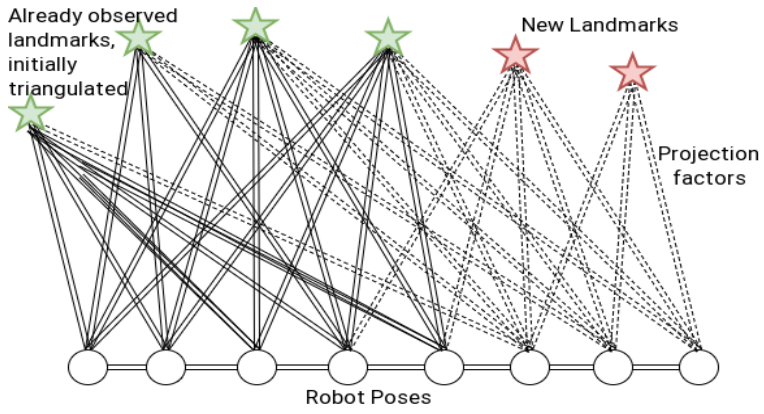


Figure 3.4: Propagating scale via projection factors

As shown in Figure 5, once the initial set of trails have been initialized (Green stars), the rest of the trails are added as projections on the set of image frames and the relative reprojection error (distance between projection of triangulated points on image plane and actual point projections on the plane) is minimized using a Nonlinear graph optimization. Red stars represent new landmarks, whose scale is not fixed until optimization. A library called GTSAM (Georgia Tech Smoothing And Mapping library) is used for this process. The final result gives us a 6DOF pose of the robot at each point in its trajectory and a 3DOF point representing the final position of a landmark with respect to the origin. This is shown in figure 3.5 for the EuRoC dataset. [9] Use of Levenberg-Marquadt Optimization and Gauss-Newton Optimization performs a very good job at reconstructing the map and generating the trajectory, but leads to very high memory consumption after a while due to there being a very dense and large graph that needs to be optimized. Experimental results demonstrated over 13 gigabytes of RAM being consumed after about 30 seconds of operation.

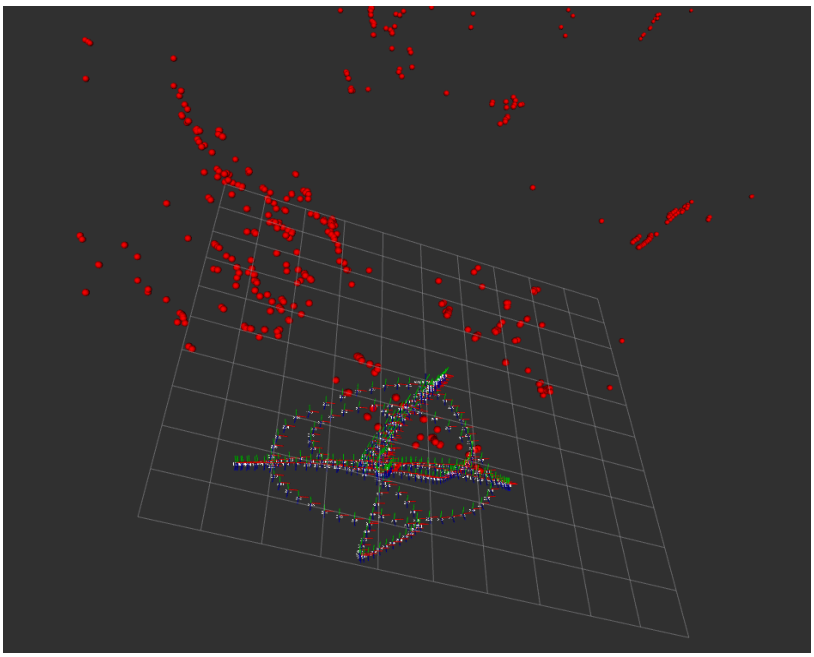


Figure 3.5: Recovered 6DOF poses (arrows) and landmarks (red dots)

The Memory consumption issue was solved by the use of Incremental Smoothing and Mapping (ISAM) optimizer. The ISAM library provides efficient algorithms for batch and incremental optimization, recovering the exact least-squares solution. It does not solve for the entire graph always, unlike Levenberg-Marquadt or Gauss-Newton optimizer, instead it onlyconsiders the most recent updates and adds them to the graph incrementally.

ISAM provides an efficient and exact solution by updating a QR factorization of the naturally sparse smoothing information matrix, therefore recalculating only the matrix entries that actually change.

It also does periodical variable reordering to maintain sparseness of the square root information matrix in case there is a loop closure and lot of associations very far from the main diagonal are seen in the information matrix appear. It then reorders the variables intelligently to make it closer to a diagonal matrix. The Bayesian network formulation of the SLAM problem can be visualized in Figure 3.6.

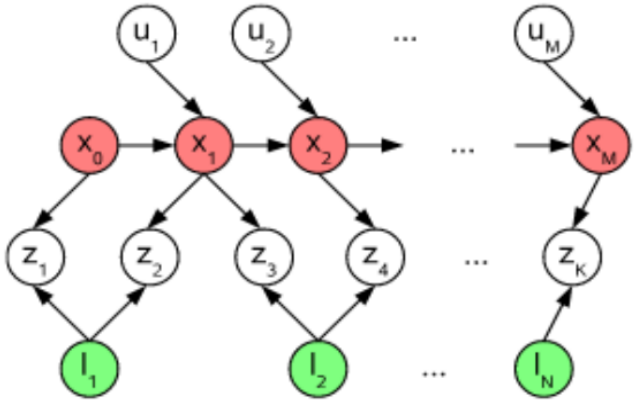


Figure 3.6: Bayesian Network Formulation

where the u_i represent the control inputs, x_i denote poses, l_i denote landmarks and z_i represent the measurements.The Joint probability of all variables is given by equation

$$P(X,L,U,Z) \propto P(x_0)\prod_{i=1}^{M} P(x_i|x_{i-1},u_i)\prod_{k=1}^{K} P(z_k|x_{i_k},l_{j_k}) \tag{3.9}$$

The optimum position of landmarks and poses is given

$$X^*,L^* = \underset{X,L}{\operatorname{argmax}}\, P(X,L,U,Z) = \underset{X,L}{\operatorname{argmin}}\left(-\log P(X,L,U,Z)\right) \tag{3.10}$$

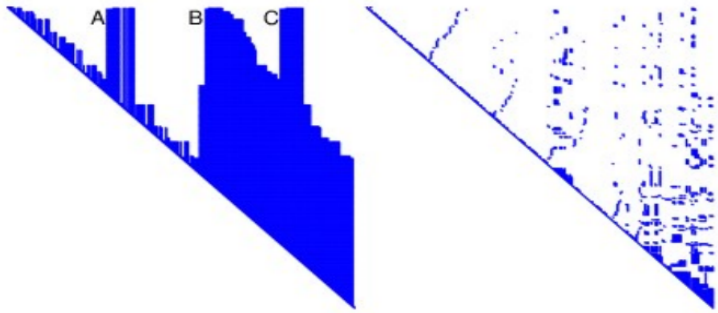
Which for a gaussian noise simplifies to a least squares problem

$$X^*,L^* = \underset{X,L}{\operatorname{argmin}}\left\{\sum_{i=1}^{M}\|f_i(x_{i-1},u_i)-x_i\|^2 + \sum_{k=1}^{K}\|h_k(x_{i_k},l_{j_k})-z_k\|^2\right\} \tag{3.11}$$

Which simplifies to

$$\theta^* = \underset{\theta}{argmin}\|A\theta - b\|^2 \tag{3.12}$$

Where θ matrix contains all the poses and landmarks. A is the Jacobian matrix. The || . || operator is the Mahalanobis norm given by e'Σe where e = error function and Σ is the covariance matrix. QR factorization of this A matrix gives the square root upper triangular information matrix, R. This matrix R is incrementally updated and the spasity & upper-triangularity is maintained. Also in case of loop closures, where very far off elements in matrix show correlations, thus periodic variable reordering is done to maintain the R matrix close to being diagonal. This is shown in Figure 3.7



(b) Factor R. (c) The same factor R after variable reordering.

Figure 3.7: Variable Reordering for Square Root Information Matrix

By the use of Incremental Smoothing and Mapping, the memory consumption was brought down to under 1 gigabyte even for extended runs.

Results produced using ISAM compared to ground truth readings are shown in Figure 3.8. A Deviation of 8cm on 3m (2.66% error) of trajectory on a generic collected dataset , and 4cm for 5m (0.8% error) of trajectory for the EuRoC dataset.

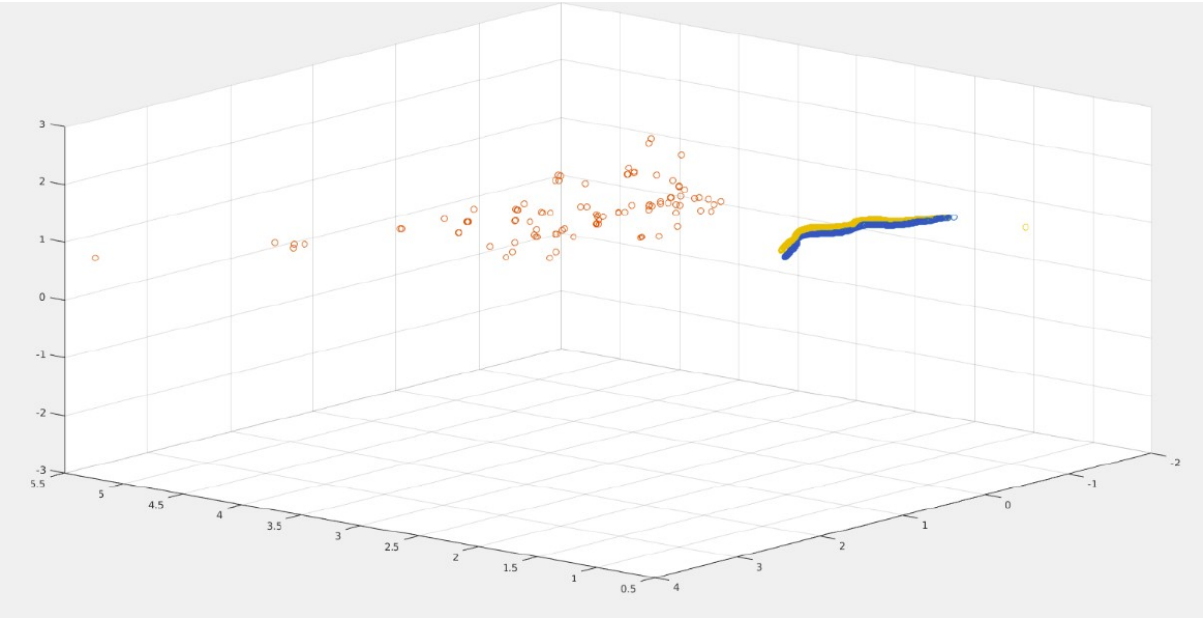


Figure 3.8: ISAM result (blue) comparison with ground truth (yellow)

3.6 Robust Loop Closures

Loop closing is the act of correctly asserting that a vehicle has returned to a previously visited location. This is necessary because very commonly in visual SLAM, previously visited location gets remapped in wrong global location. Thus the position error accumulates out-of-bound. Figure 3.9 shows how a loop closure is detected in a robot trajectory (red), and the corresponding correction factor(dotted line) that leads to an optimized trajectory (green) after optimization.

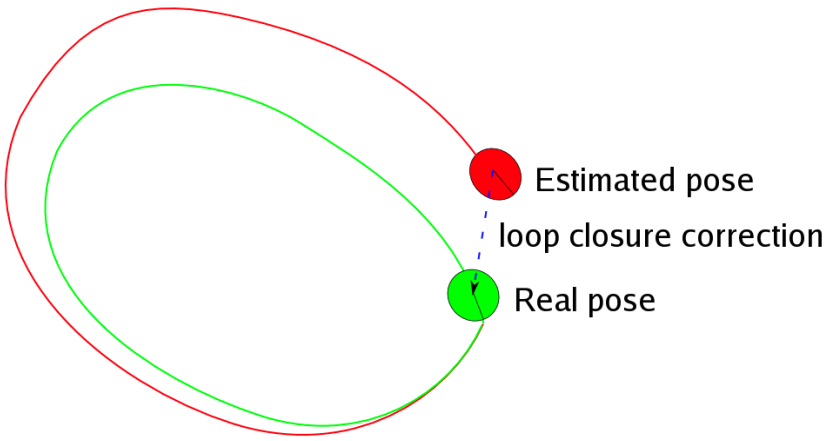


Figure 3.9: Correction of map using loop closures

DLoopDetector is an open source C++ library to detect loops in a sequence of images collected by a mobile robot. It is based on a bag-of-words database created from image local descriptors, and temporal and geometrical constraints. It works with FAST corners and BRIEF descriptors. The generated bag-of-words from the image are stored in local variables and being small in size, the vectors can be compared fairly quickly even for long times. Only a select few keyframes are checked for loop closures.

Figure 3.10: Example of Loop Closure Results

Figure 3.10 shows the loop closure algorithm's result, with the two images separated about 20 seconds apart in the video identified as similar.

Chapter 4: Point cloud processing

4.1 Point cloud stitching

The RGBD points and poses estimated using nonlinear optimization were used to stitch a dense 3D map of the environment (~100 million points). Figure 4.1 shows the resultant stitched point cloud.

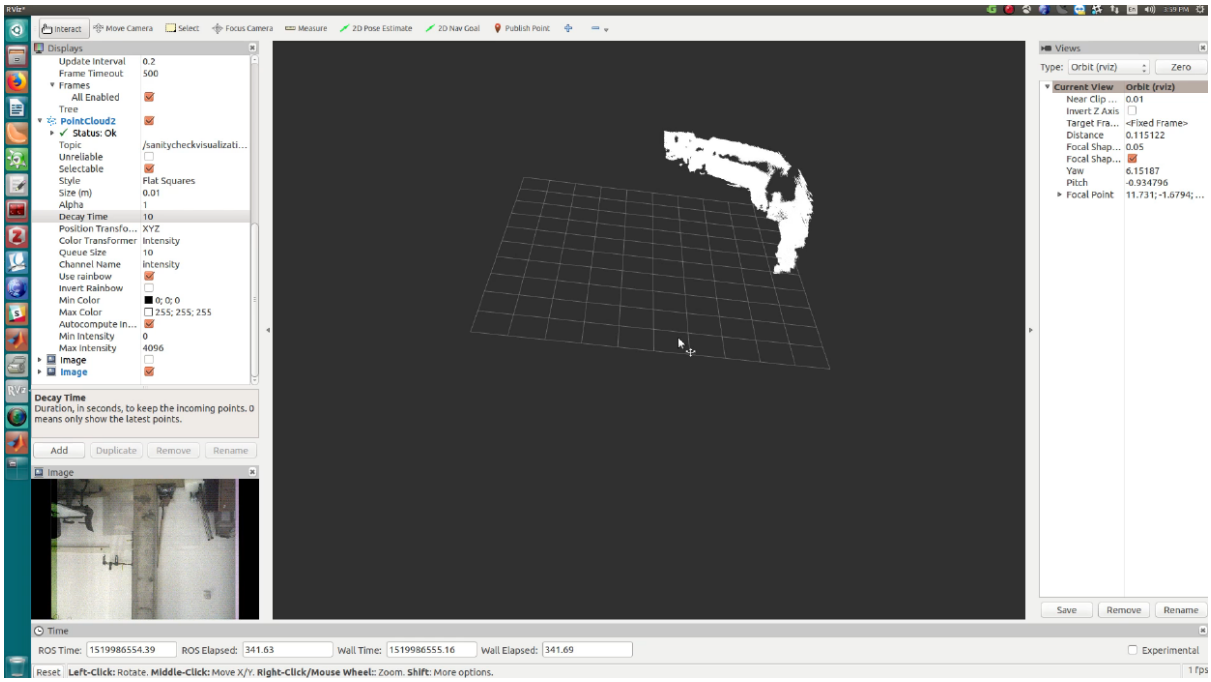


Figure 4.1: Demonstration of Point cloud stitching on Rviz

4.2 Plane Fitting on Point Cloud

Using an approach of RANSAC[8] based 3-dimensional plane fitting, large plane surfaces may be identified in the scene. Using the identified surfaces as reference an autonomous altitude control of the MAV may be achieved. A downward facing ultrasonic sensor may be used as an additional sensor for robust control. An algorithm for this approach is described:

Denote P as a point on a plane. The unit normal vector of a plane is N . A certain point P_0 is on the plane. Then, the 3D plane is expressed as follows.

$$(P - P_0).N = 0 \tag{4.1}$$

First, calculate the centre of mass of the point cloud P_0 is calculated as follows using the point cloud $P_{1,}P_{2,}\ldots,P_m$

$$P_0=\frac{1}{M}\sum_{m=1}^{M}P_m \tag{4.2}$$

Here, 3D coordinates of P_0 are denoted as (x_0,y_0,z_0) . Denote the 3D coordinates of point cloud as $P_1=(x_1,y_1,z_1),P_2=(x_{2,}y_{2,}z_2),\ldots,P_M=(x_M,y_M,z_M)$. The aim is to determine $N=(Nx,Ny,Nz)$ which satisfies the following.

$$\begin{bmatrix} x_1-x_0 & y_1-y_0 & z_1-z_0 \\ x_2-x_0 & y_2-y_0 & z_2-z_0 \\ \vdots & \vdots & \vdots \\ x_M-x_0 & y_M-y_0 & z_M-z_0 \end{bmatrix}\begin{bmatrix} N_x \\ N_y \\ N_z \end{bmatrix}=\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{19}$$

This is represented as follows.

$$\boldsymbol{A}x=0$$

The rank of Mx3 matrix A is 2. Applying SVD (singular value decomposition) to A returns the solution to N , which is the right singular vector whose singular value is the smallest. In case of outliers, the sum of error terms for the dot product between all the vectors and the normal vector is minimized. RANSAC will be used to remove outliers from the data.

4.3 3D segmentation of point cloud

After the point cloud has been segmented and converted into a sets of triangles and planes, we can represent it as a 3D object. 3D objects are most commonly represented in a Wavefront OBJ file format. The OBJ file format has information about faces, vertices and optionally vertex normals and textures.

Once we have the information of the surroundings in a OBJ file format, it needs to be segmented into meaningful regions that can be used for the reference values in altitude control (more on this in section 5). The approach for 3D segmentation is done using a spectral clustering approach described in [11]. The algorithm steps are:

1. Calculate angular distance and geodesic distance between adjacent faces.
2. Create an affinity matrix using an exponential kernel.
3. Perform k-means clustering to cluster the segments.

Figures 4.2 and 4.3 shows the result of such a clustering.

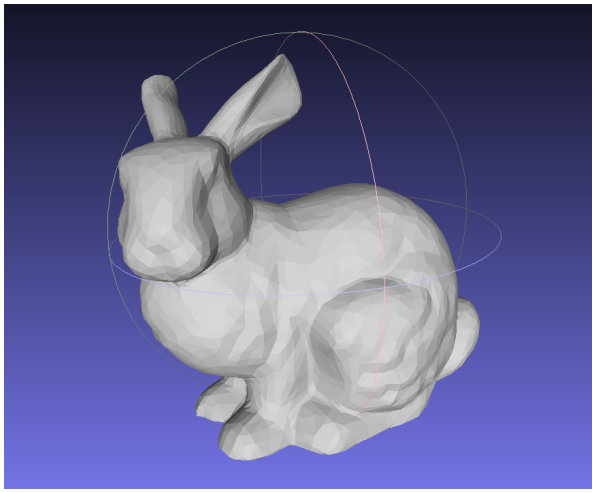


Figure 4.2: Original OBJ image

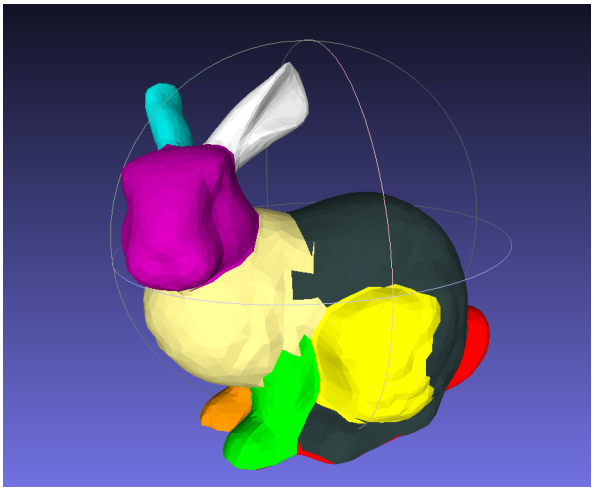


Figure 4.3: Segmented OBJ image

Chapter 5: Future Work

5.1 MAV dynamics

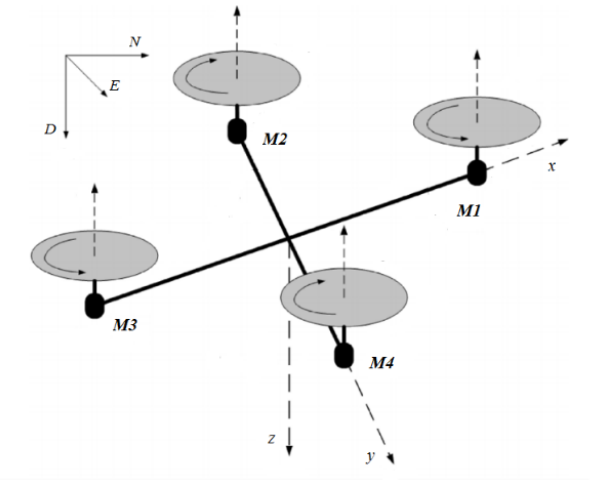


Figure 5.1: Quadcopter Coordinate Frames

The Euler angles $\Psi=[\varphi,\theta,\psi]$ represent the roll, pitch and yaw. $S_q=S_\varphi S_\theta S_\psi$ is the net rotation matrix from the standard North-East-Down (NED) reference (Figure 5.1).

$$\dot{V}_q=\frac{F^q}{m}-\Omega^q\times V^q+S_q\times g \tag{5.1}$$

$$\dot{\Omega}^q=-J^{-1}\left(\Omega^q\times\left(J\Omega^q\right)\right)+J^{-1}M^q \tag{5.2}$$

$$\dot{P}^I=S_q^T V^I \tag{5.3}$$

$$\dot{q}^I=\frac{-1}{2}\omega_q q^I \tag{5.4}$$

$V=[U,V,W]',\Omega=[P,Q,R]',P=[X,Y,Z]'$ and q represent the linear velocity, angular velocity, position and attitude of the quadrotor [10]. ω_q is the skew-symmetric matrix

$$\omega_q = \begin{bmatrix} 0 & P & Q & R \\ -P & 0 & -R & Q \\ -Q & R & 0 & -P \\ -R & -Q & P & 0 \end{bmatrix}$$

5.2 Planned Implementation

The plan is to run altitude control in real-time using only a vision + depth or a vision + IMU or a vision + depth + IMU setup quadcopter. Individual aspects of the project have been tested out and prototyped and the code is ready to be deployed on to a real quadcopter. Control strategies can then be thought of, for improving the performance with slow data rates. A suitable deadline of the completion of this entire work can be set in September of 2018.

References

1. Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B. FastSLAM: A factored solution to the simultaneous localization and mapping problem (2002)
2. Thrun, S., Burgard, W. and Fox, D., Probabilistic robotics (2005)
3. Lu, F. and Milios, E., Globally consistent range scan alignment for environment mapping (1997)
4. Shen, S., Michael, N. and Kumar, V. Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs (2015)
5. Carrillo, L.R.G., López, A.E.D., Lozano, R. and Pégard, C. Quad rotorcraft control: vision-based hovering and navigation (2010)
6. Kalal, Z., Mikolajczyk, K. and Matas, J. Forward-backward error: Automatic detection of tracking failures (2010)
7. Nistér, D. An efficient sol7ution to the five-point relative pose problem (2004)
8. Fischler, M.A. and Bolles, R.C. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography (1981)
9. Burri, M., Nikolic, J., Gohl, P., Schneider, T., Rehder, J., Omari, S., Achtelik, M.W. and Siegwart, R. The EuRoC micro aerial vehicle datasets (2016)
10. Stevens, B.L., Lewis, F.L. and Johnson, E.N. Aircraft control and simulation: dynamics, controls design, and autonomous systems (2015)
11. Liu, R. and Zhang, H.. Mesh segmentation via spectral embedding and contour analysis (2017)

Appendix A: The Codebase

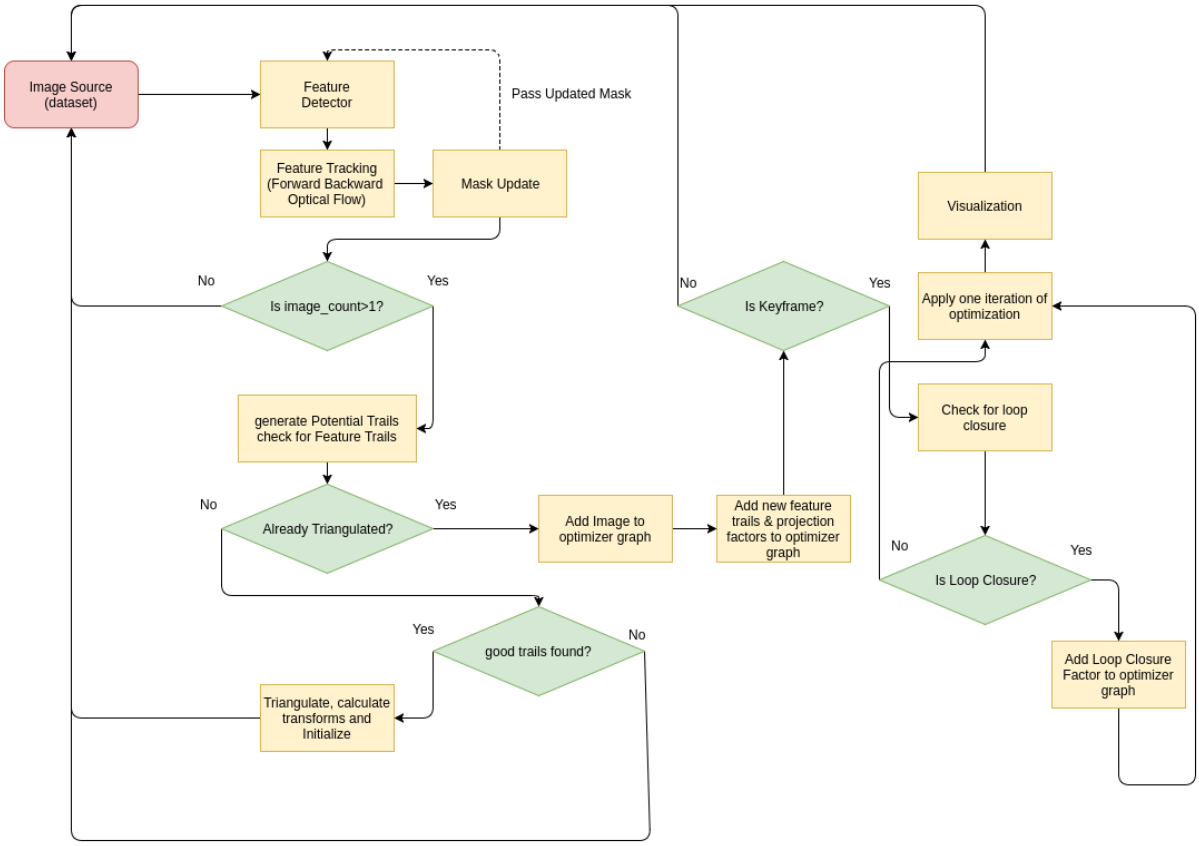


Figure A.1: Flowchart for the codebase

The code for the entire project has been written in C++. Robot Operating System (ROS is used for communication purposes, and for easy data acquisition from the EuRoC dataset. The feature detection (Harris corners), feature tracking (Lucas-Kanade tracking) and grid occupancy mask management is done using the OpenCV library in separate classes. A separate class manages all the feature trails and potential trails. The Triangulation and initialization to produce initial (green stars in Figure 3.4) is done using GTSAM library. GTSAM is then used for Nonlinear graph optimization to produce the optimized poses and landmark locations that are then visualized using Rviz (ROS visualizer). A separate class checks for disparity in images to select keyframes. A library called DloopDetector is used to check for loop closures among keyframes which are then managed by another class. A library called Eigen is used for matrix math. Overall, in excess of 5000 lines of code (sans comments) have been written for this project upto now. The codebase is designed to run on both x86 and x64 systems.