Visual-Inertial Simultaneous Localization and Mapping for Micro-Aerial Vehicle

A Thesis

submitted by

AJAY PRATAP SINGH

in partial fulfilment of the requirements
for the award of the degree of

Dual Degree (B.Tech. and M.Tech.)

Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

9 June 2017

THESIS CERTIFICATE

This is to certify that the thesis titled **Visual-Inertial Simultaneous Localization and Mapping for Micro-Aerial Vehicle**, submitted by **Ajay Pratap Singh**, to the Indian Institute of Technology, Madras, for the award of the degree of **Dual Degree(B.Tech. and M.Tech.)**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Anurag Mittal
Project Guide
Associate Professor
Department of Computer Science and Engineering
IIT-Madras, 600 036
http://www.cse.iitm.ac.in/ amittal/

Place: Chennai

Date: 9th June 2017

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my Guide for providing me the opportunity to work on Autonomous Drone Project. His continuous motivation and guidance has been very helpful. I would also like to thank my team members and all other members of CV Lab. Also, various researchers in this field for providing useful resources such as source codes, documentations, datasets, etc.

Last but not the least, I would like to thank my Parents, Brother and Friends for supporting me always.

ABSTRACT

KEYWORDS: SLAM, Autonomous Navigation, LSD-SLAM imu, Visual Interial, Multi-View Reconstruction, 3D Reconstruction, Mapping, Localization, VO, VIO, VIN

Autonomous Navigation for a MAV requires it to have its localization and map of current environment. We reproduced the results of the four state-of-the-art papers: (1) LSD-SLAM, (2) ORB-SLAM, (3) SVO, (4) DSO. The first two methods are SLAM, in which the output is localization of agent in real time with map of environment. The last two methods are VO, in which localization of agent is given as output.

We implemented the fusion of LSD-SLAM with IMU sensors to enhance its preformance in shaky/rapid movements or when vision system fails (blurry vision, dark environment, featureless environment, etc.).

We also implemented a fusion of SVO with IMU sensors. Due to the light-weight and fast nature of both SVO and IMU, we can use this fusion in previously seen environments or semi-autonomous navigation, where only localization is needed.

Experimental evaluatation of our approach with respect to existing state-of-the-art systems on benchmark datasets and IITM-CVL datasets shows that our system performs better.

Both the approaches do not require GPU, but can be modified to use GPU and increase performance.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

IITM	Indian Institute of Technology, Madras
base	Base station or Main Drone Operator on Field
SLAM	Simultaneous Localization and Mapping

NOTATION

r	Radius, m
α	Angle of thesis in degrees
β	Flight path in degrees

CHAPTER 1

Introduction

Miniature Aerial Vehicles (MAVs) are have become extremely popular in recent times. Military was the primary user of such devices, but since the last decade, civilians and researchers across the globe have shown increased interest. Also, there has been growth in the startups related to MAVs. The applications of a flying camera are limitless.

Primary uses of such MAVs are mainly surveillance, since they are small and may not always be able to carry payload. MAVs can be integrated with a wide variety of sensors. These sensors help in navigating or gathering data of environment. Common sensors used with MAVs are:

- IMU : This is short range sensor. It provides MAVs acceleration and orientation. Data from this sensor can be used to measure approximate location, for short durations.
- GPS : This is very long range sensor. It provides Latitude, Longitude and Altitude coordinates from Satellites. This sensor data is not very accurate and will not work indoors.
- Sonar : This is short range sensor. It provides a rough distance of closest objects in the target cone.
- LASER : This is medium range sensor. This provides rich representation of environment. But the cost of equipments, power required to scan and computational cost prohibits the use of it in MAVs.

1.1 Autonomous Navigation and its challanges

Navigation has four major divisions:

- Localization
- Mapping
- Path Planning
- Motion Controls

Autonomous Navigation is the task of changing the position of agent from current location to destination. For this, as agent has to generate local map and update with global map, localize itself in the global map, and control itself to follow the path as planned by planning algorithm.

1.2 Problem Statement

The objective of this thesis is to explore exisiting SLAM algorithms and improve them to work for MAV to navigate in any previously unknown environment.
A major challange for all pure vision related solutions is tracking loss, due to fast/shaky movements, illumuination changes, point-of-view changes, deformity shape of target, etc. To overcome this, we propose a fusion of IMU with LSD-SLAM algorithm.

CHAPTER 2

Simultaneous Localization and Mapping

2.1 Introduction

The most important problem in robotics community, since its very beginning, has been to create a truly autonomous agent to traverse any previously unknown environment, accomplish the goals and return back safely. Now, leaving the mechanics and electronics aside, the problem boils down to autonomous navigation and path planning.

Path planning requires the agent to have map of the current environment, its location in that environment and constraints, if any. In short the output of a generic SLAM system is input to path planning. But in this thesis, we shall only discuss methods to solve autonomous navigation.

For a system to navigate autonomously, it is required to compile the following important information from its sensors:

1. Map of the environment
2. Location of agent in environment

Position and orientation is also called *pose*, and process of locating an agent (finding pose) in the computed map is also known as *pose estimation*. And, the complete process of pose estimation (of agent) along with simultaneously creating a map of world (around agent) is called **Simultaneous Localization and Mapping** (SLAM) in Robotics Community.

In some cases. we may revisit a previously known environment. Hence, we may have already computed and saved the map of the environment. In that case we can skip the map generation part of the environment (maybe refine it a little), and focus on only localization. Thus, Visual Odometry can be employed to trace the path of agent in real-time, instead of full SLAM algorithm, making the computation fast and inexpensive.

Whereas, in case of a completely new environment, the MAV is required to have information of both map and localization, as mentioned above. We call this class of problem as Kidnapped Robot Problem, which will be discussed in the next section.

2.2 Kidnapped Robot Problem

Kidnapped Robot Problem is the ultimate challange for an agent in Robotics. An agent needs to construct the environment it currently in, and localize itself in it. Then match the previously visited parts of map with the new data to find out loop closures. This is similar to a situation where one person wakes up in a maze (akin to Kidnapping in an isolated place), and tries to find the way out. While finding out the way, the person tries to make a mental map of the environment and tries to remember where it is using reprojection of camera in 3D world using Multiple view Geometry. Now, when the place is revisited, it is called **Loop Closure**. Similar to what a person does when they revisit a previously seen place, they correct their mental trajectory and map. Thus loop closures are a very efficient way to minimize error (deviation from actual trajectory) in SLAM. Path planning algrithms can be designed to find many loop closures using partial maps and hence minimize the errors.

2.3 Chicken-Egg Problem

Having discussed the KRP, let us now come to **Chicken-Egg Problem**. We know that SLAM requires us to have both current map of the environment and location of agent in that map simultaneously. But this creates a deadlock.

- To generate a consistent map, one needs to have accurate location of agent where it took the images from.
- To localize an agent in map, one need to have the map to estimate the agent's proximity from landmarks.

Researchers have come up with various methods to solve this problem. This has led to creation of surprisingly newer fields, which were not known to us, and it is discussed in the next section.

2.4 Paradigms and Classifications of SLAM

In the past, researchers have developed algorithms which can be classified in three major paradigms of SLAM.

1. Extended Kalman Filter methods
2. Particle Filtering methods
3. Graph Optimization methods

Apart from this, the amount of previous data considered while computing SLAM, the classification can be as:

1. Online SLAM : previous data is not fully stored, only recent n frames are considered to compute pose and generate map.
    - **Pros:**

- Computationally fast and inexpensive
- Low storage space needed
- Ideal for MAV or smaller devices with low computational power and space

- **Cons:**
  - Not very accurate pose and map
  - Prone relatively more as compared to Full SLAM with respect to drift from ground truth with respect to time
  - Not very ideal for loop closures

2. Full SLAM : full pose and map is stored
   - **Pros:**
     - Reliably accurate pose and map
     - Better loop closures
   - **Cons:**
     - Computationally heavy as data grows dynamically
     - Storage space required is high as growth is non-stopping
     - Storing full map in (limited) main memory will lead to system failure for large environments or longer durations

Also, there can be various sensors that can be deployed to extract the information of environment and estimate location of agent in it:

- Laser
- Sonar
- Vision

SLAM can also be classified in terms of how it takes input information and pre-processing of information from sensors.

1. Feature-based SLAM : Prior to computing the pose, features are extracted to facilitate the process
   - **Pros:**

- Computationally Faster
- Extracts and matches features, and hence will have minimum re-projection error
- No initialization needed

- **Cons:**
  - Only corners or edges can be used and reconstructed
  - False KF detection leads to drastic errors

2. Direct-SLAM : No pre-processing is done before tracking, full image intensity is used
   - **Pros:**
     - Since it uses whole image intensity, therefore it will have minimum photometric error
     - Is capable of reconstructing (almost) whole surface
   - **Cons:**
     - Not very robust to outliers and lightining/camera changes
     - Needs good initialization

SLAM can also be classified in terms of how it represents its map

1. Sparse Map : Only edges and corner (pixels at) points are present in map sparsely
   - **Pros:**
     - This will save space for storing the map, hence we can store map of larger environments or longer durations
   - **Cons:**
     - Since only edges and corners are present, model of the world is not very useful for planning algorithms

2. Semi-Dense Map : Pixels are stored selectively only for regions with high image intensity
   - **Pros:**
     - Fair representation of environment. This is a trade-off between speed of sparse map and richness of dense map

- **Cons:**
  - It is slower to compute than sparse map

3. Dense Map : Almost all pixels are stored
   - **Pros:**
     - Excellent representation of environment
   - **Cons:**
     - It requires a lot of space to store and also huge computation power to process such information

Visual SLAM can be mainly classified in terms of number of cameras used:

1. **Monocular Camera**: Any instrument having single camera and field-of-view within range of 0-360 degree
   - **Pros**:
     - Cheap
   - **Cons**:
     - Scale of the scene is never captured

2. **Stereo Camera**: Any instrument having two camera, seperated by a constant baseline, and field-of-view within range of 0-360 degree
   - **Pros**:
     - Scale of the scene is captured
   - **Cons**:
     - Relatively costlier than monocular camera

3. **Omnidirectional Camera**: Camera has a field-of-view of 360 degree
   - **Pros**:
     - Robust to shaking or fast movements, since there is more overlap between frames
   - **Cons**:
     - Not very conventional to use

In the last 5 years, researchers have started to use Vision sensors (assisted along with other complementary sensors, such as IMU) to implement Graph Optimization algorithms on Direct or Feature based SLAM. We shall be discussing only those methods in this thesis.

2.5 Graph based SLAM

In this section we are going to discuss only about Graph based SLAM methods, since others are almost obsolete now.
Graph based SLAM method are generally Full SLAM and for MAVs, the best sensor in terms of cost and affordability is vision (camera). Here, we represent the model of the world as point clouds of depth map, with node being the KF with depth map associated and edges being the constraints of KFs with respect to each other. There are quite a few frameworks readily available to work with, such as "A General framework for Graph Optimization" or g2o `https://github.com/RainerKuemmerle/g2o` or GTSAM `https://collab.cc.gatech.edu/borg/gtsam?destination=node%2F299`

2.6 Generic SLAM algorithm

Any generic SLAM algorithm has two major divisions:

1. Front-End of SLAM
2. Back-End of SLAM

Front-End of SLAM

In this part of the SLAM algorithm, the information from sensors is compiled into format to be used by back-end of SLAM algorithm.
For a feature based SLAM, images may be cropped and/or grayscaled, and then compared with the corresponding frames near in time-frame. The can be various

methods deployed to estiamte the camera movemement (or camera tracking) by tracking the features in consecutively timed frames and projecting back in camera coordinates.

Back-End of SLAM

The job of back end is mainly to optimize the map continuosly. Reaserchers mainly use g2o or GTSAM for graph optimization of map in SLAM. One
Also, loop closure detection is another primary task of back end, primarily for which appearance based algorithms are employed. OpenFAB map is one such algorithm.
`https://github.com/arrenglover/openfabmap/releases`

2.7 History of Modern SLAM

Table 2.1: Modern Visual SLAM

Year	Paper	Type
2007	PTAM	filtering
2011	DTAM	direct
2014	LSD-SLAM	direct
2014	SVO	hybrid
2015	ORB-SLAM	filtering
2016	DSO	direct

Some of the novel papers in SLAM have been listed above. PTAM Klein and Murray (2007). Then DTAM Newcombe *et al.* (2011). Major break-through came with LSD-SLAM Engel *et al.* (2014) in 2014. Same year SVO was published Forster *et al.* (2014). Next came ORB-SLAM Mur-Artal *et al.* (2015). Then DSO Engel *et al.* (2017)

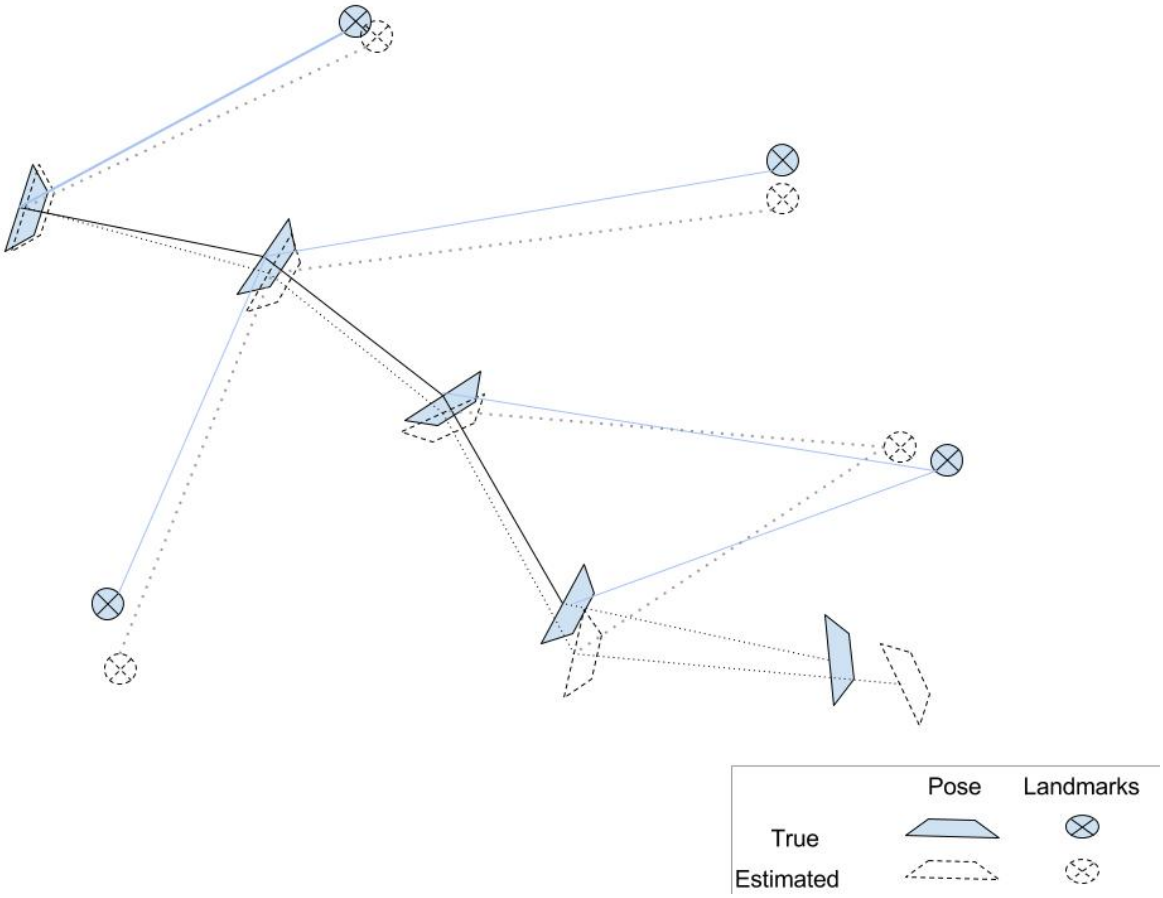

Figure 2.1: Localization and Mapping using SLAM

2.8 Applications of SLAM

The SLAM algorithm can be applied, not just for navigation in MAVs, but also in underwater vehicles, or space exporation robots.

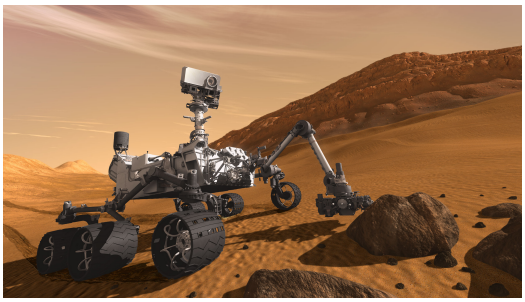

Figure 2.2: Mars Rover

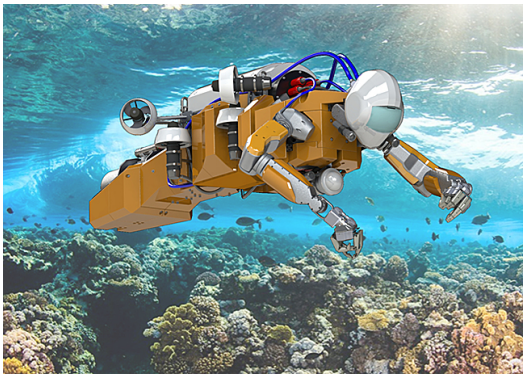

Figure 2.3: Underwater Robot

Figure 2.4: Indoor Robots

Figure 2.5: Underground Robot

CHAPTER 3

Large-Scale Direct Simultaneous Localization and Mapping

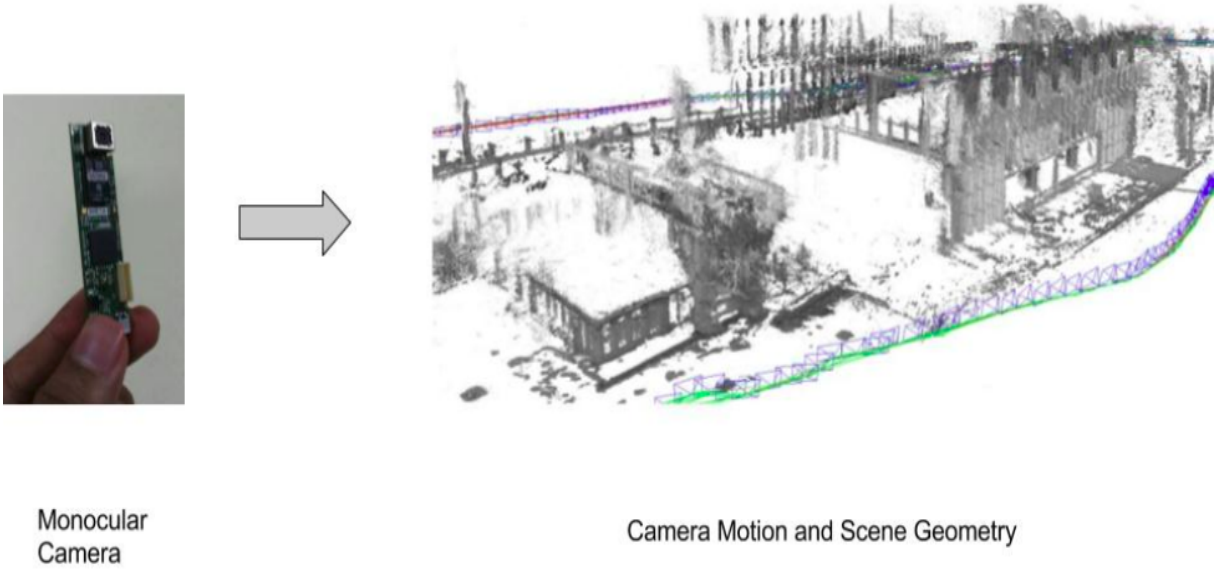

Figure 3.1: LSD-SLAM

3.1 LSD-SLAM Algorithm

The LSD-SLAM core algorithm is divided in three major parts: **tracking**, **depth map estimation** and **map optimization**:

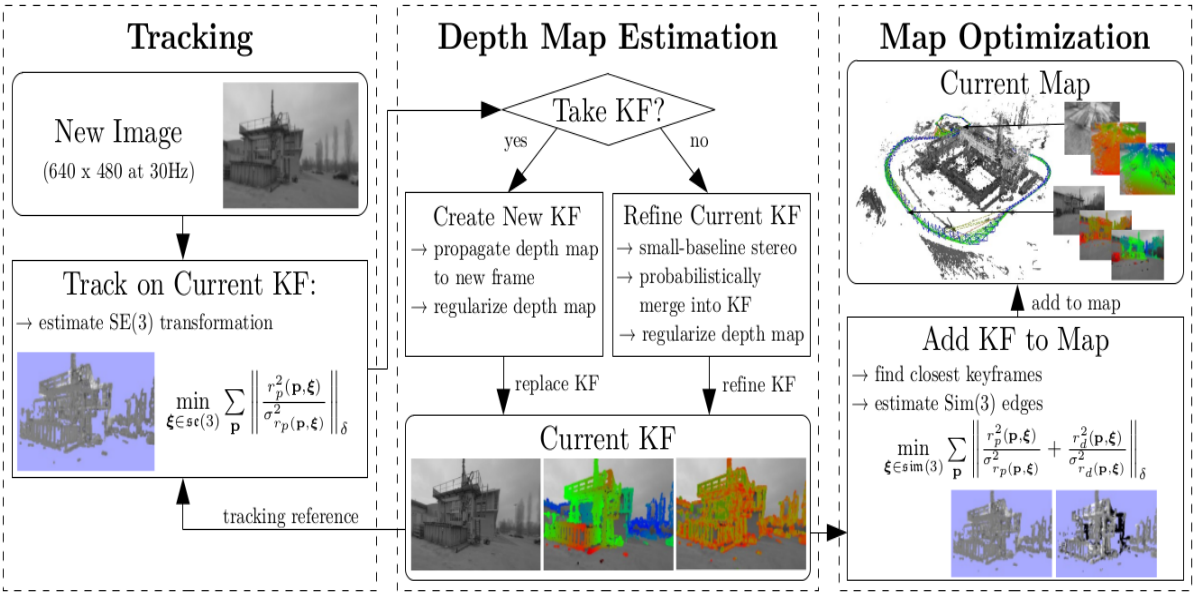

Figure 3.2: Overview of LSD-SLAM Algorithm

3.1.1 Tracking

This part of the algorithm is responsible for keeping a track of the camera. In other words, the current pose (position-orientation) of camera is estimated here, relative to previous pose, by minimizing variance-normalized photometric error [Eq. 7.1]

$$E_p(\xi_{ji}) = \sum_{\mathbf{p}\epsilon\Omega_{D_i}} \left\| \frac{r_p^2(\mathbf{p}, \xi_{ji})}{\sigma^2_{r_p(\mathbf{p},\xi_{ji})}} \right\|_\delta \tag{3.1}$$

$$r_p^2(\mathbf{p}, \xi_{\mathbf{ji}}) := \mathbf{I}_i(\mathbf{p}_i) - \mathbf{I}_\mathbf{j}(\omega(\mathbf{p}_i, D_i(\mathbf{p}_i, \xi_{ji}))) \tag{3.2}$$

$$\sigma^2_{r_p(\mathbf{p},\xi_{ji})} := 2\sigma_I^2 + V_i(\mathbf{p}) \tag{3.3}$$

Following are the key assumptions for tracking:

1. KeyFrame with Depth Map exists. This creates a paradox for tracking first

frame. Solution to this will be discussed in section **Breaking Circularity**

2. Brightness consistency holds across camera frames. This can leads to *loss of tracking*, since there are illumination variations as a function of time for the same region, and algorithm will not be able to recognize current CameraFrame with respect to previously visited KeyFrame.

3. Image Intensity noise, σ_I^2 is Gaussian

Note: Since depth information is not yet available for current CameraFrame, we cannot evaluate scale of it. Hence, minimization is perfomed on **se(3)**

3.1.2 Depth Map Estimation

We divide this section in three parts:

KeyFrame Selection

When a new CameraFrame arrives, a decision is made weather to save the CameraFrame as KeyFrame or not. A weighted combination of relative distance and angle is thresholded to the current KeyFrame.

$$dist(\xi_{ji} := \xi_{ji}^{\mathbf{T}} \mathbf{W} \xi_{ji}) \tag{3.4}$$

where **W** is a diagonal matrix having weights.
KeyFrame will be created if it crosses this range. Otherwise, CameraFrame will be used to just refine the existing Depth Map of current KeyFrame.
Note: Each KeyFrame is scaled such that its mean inverse depth is one

Depth Map Creation

When a CameraFrame is chosen to become KeyFrame, its Depth Map is initialized by projecting points from previous KeyFrame into it. This is followed by one iteration of spatial regularization and outlier removal as proposed in [9]. After this, the Depth Map is scaled to have a mean inverse depth of one. This scaling factor is incorporated into the **sim(3)** camera pose. Finally, the CameraFrame replaces the previous KeyFrame and becomes the current KeyFrame, which will be used for tracking new CameraFrames.

Depth Map Refinement

When a CameraFrame is chosen to not become KeyFrame, it is used to refine the current KeyFrame.
Many small-baseline stereo comparisons are performed for specific image regions as proposed in [9]. The result is incorporated into existing Depth Map, thereby refining it and potentially adding new pixels.

3.1.3 Map Optimization

The Map consists of set of set of KeyFrames and tracked **sim(3)**-constraints. This is the back-end of LSD-SLAM algorithm and is continuosly optimized in background using pose graph optimization. The error function that is minimized is:

$$E\left(\xi_{W1}...\xi_{Wn}\right) := \sum_{(\xi_{ji},\Sigma_{ji})\epsilon\varepsilon} \left(\xi_{ji} \circ \xi_{Wi}^{-1} \circ \xi_{Wj}\right)^T \Sigma_{ji}^{-1} \left(\xi_{ji} \circ \xi_{Wi}^{-1} \circ \xi_{Wj}\right). \quad (3.5)$$

W is the world frame

3.2 Miscellaneous

3.2.1 Detecting Loop Closure

When a new KeyFrame K_i is added to the Map, a collection of few possible loop closure KeyFrames $K_{j_1}, ..., K_{j_n}$ is created. Closest ten KeyFrames and one KeyFrame proposed by an appearance-based mapping algorithm [11] are used to create this collection, in order to detect large-scale loop closure.

Reciprocal Tracking Check is performed to avoid insertion of falsly tracked loop closure: For each candidate K_{j_k}, we track both $\xi_{j_k i}$ and $\xi_{i j_k}$, and estimate if they are statistically similar by:

$$e\left(\xi_{j_k i}, \xi_{i j_k}\right) := \left(\xi_{j_k i} \circ \xi_{i j_k}\right)^T \left(\Sigma_{j_k i} + Adj_{j_k i} \Sigma_{i j_k} Adj_{j_k i}^T\right)^{-1} \left(\xi_{j_k i} \circ \xi_{i j_k}\right) \tag{3.6}$$

If the above quantity is sufficiently small, only then these KeyFrames with loop closures are added to global map.

3.2.2 Breaking Circularity

As mentioned previously, an existing Depth Map is used to track new KeyFrames and this KeyFrame is used to either make a new KeyFrame or refine current KeyFrame using stereo, for Depth Map. Then this new KeyFrame is used further to track new CameraFrames.

We can break this classical circular dependency, which we need to break. We initialize using a Random Depth Map, and perform alernate optimizations between pose and Depth Map

CHAPTER 4

Visual-Inertial SLAM with Monocular Camera

4.1 Introduction

LSD-SLAM algorithm works well in most environments, but fails in environments with a lot of illuminance changes, or darker environment, or where there are no features to track, and also where there are sudden movements, thereby resulting in visual tracking loss.

To overcome these, we propose an integration of Vision and IMU Sensors to provide an algorithm which is robust to sudden changes in motion and illuminance. Both sensors provide an estimate of pose. IMU sensors being prone to accumulated errors, is reinitialized with *Previous Tracked Frame* and *Current KeyFrame* and we take an average of both *Pose Estimates*. We then obtain *Relative Pose* of *Current Tracked Frame* with respect to *Current KeyFrame*, and then **set-Scale** appropriately*. Now we minimize both *Pose Estimates* from Vision and IMU Sensors, to provide us *Final Pose* of *Current Tracked Frame*.

Vision works (almost perfectly) during slow and steady movement of camera. Whereas, IMU works (ideally) during sharp and abrupt movements of camera. So, they exhibit complementary nature with respect to each other. We leverage this complementary behaviour of the two sensors and propose a novel integration of IMU with LSD-SLAM to work in both slow/steady as well as abrupt/shaky situtations.

4.2 Inertial Measurement Unit

IMU is a electronic sensor for estimating orientation, angular acceleration and linear acceleration of agent. It utilizes gyroscopes and accelerometers to estimate the relative measurements with respect to initial state.

Typically, 9 DOF IMU is used and it consists of 3 axis gyro, 3 axis accelerometer, 3 axis magnetometer, which has an interface with I2C (serial bus) from Arduino. The major drawback is that is does not detect the constant linear velocity, and hence cannot be used to estimate the correct velocity or position.

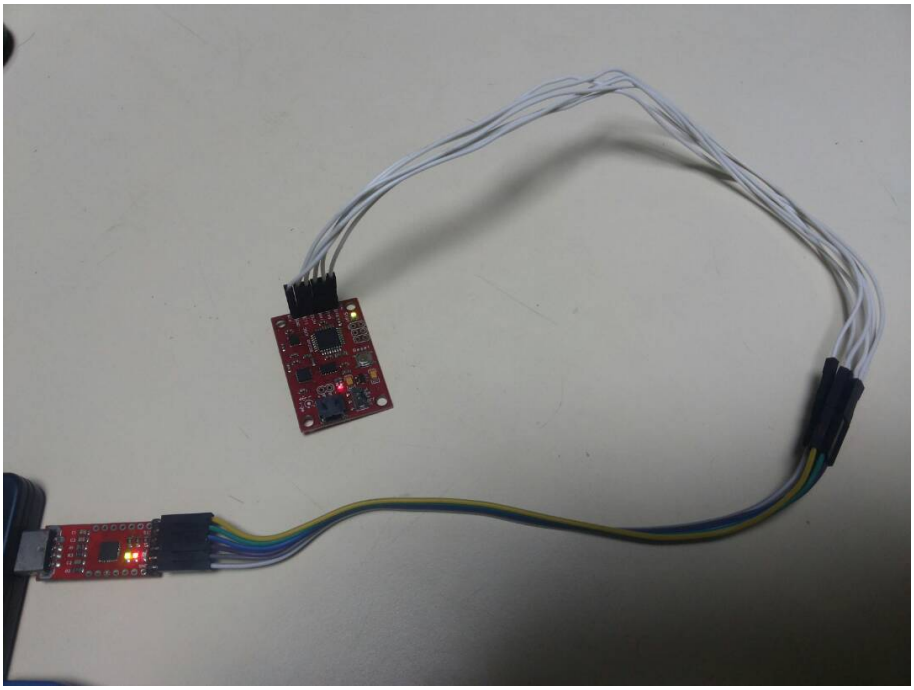

Figure 4.1: IMU (used with ZED Camera) to emulate Visual-Inertial system

4.3 Visual Inertial SLAM Fusion

Since IMU provides reliable (orientation) pose estimates for agent, it can be fused with vision pose to minimize the error in camera frame pose. Also, translation (of agent) can be obtained from IMU for frames with very less time difference (because the rate at which IMU produces readings in much higher than the fps of a typical camera). Thus, this translation obtained (altough having noise) can still be fused with translation obtained from lsd-slam algorithm. This fusion, if done over very short period of time, with correctly calibrated instruments, can significantly reduce the error in complete pose estimation of agent. Also, it can be used to estimate the scale of images in monocular camera.

There are various ways to fuse these values, some of which shall be discussed in the later sections/chapters and experiments done have been shown in chapter 9.

Fusion of sensor data can be done using various approaches, some of proposed models are mentioned below:

- Average
  - **Pros**: Sensor data from all sensors is obtained and averaged to obtain the final pose. This ensures that data will point near the value where most sensors are pointing, in case few of the sensors give errors.
  - **Cons**: If significant amount (atleast half) of sensors fail, final value will be far away from real value.

- Weighted Average
  - **Pros**: Sensor data from all sensors is obtained and averaging is done based on weights of parameters (such as error in measurement of sensor data, relaiblilty of sensor data, availability of sensor data) to obtain the final pose. This ensures that data will point near the value where most significant sensors are pointing.
  - **Cons**: Altough better than simple averaging in most cases, this will fail catastrophically in case most significant sensor sends noisy sensor data. Also, final values obtained from averaging techinques might not always point towards true value.

- Extended Kalman filter (EKF)
  - **Pros**: Sensor data from all sensors is obtained at specific time interval and and averaged to obtain the final pose.
  - **Cons**: If significant amount (atleast half) of sensors fail, final value will be far away from real value.

- Switch based on threshold
  - **Pros**: Based on realiability, availability and error in sensor data from different sensors, we can choose to switch sensor which will result in final value.
  - **Cons**: Since it relies on only one sensor at a time, it is prone to error. Also, it switches sensor at each iteration, so it will lead to abrupt changes in sensor data, if switching is frequent.

- Heuristic
  - **Pros**: Vision data, IMU data and fused data (based on any one method mentioned above) is kept in buffer, and only one of them is used during runtime. Whenever a loop closure occurs, error is estimated for all of these trajectories, and the trajectory with minimum error will replace the current choice in optimization step (only for current visual-inertial fusion, not all type of sensor data can be fused in this way). Ideally, this should be best among all methods, with least error.
  - **Cons**: Very complicated to achieve and computationally it is expensive. Also, storage required is much high.

We have experimented with *Weighted Average* and *Switch based on threshold*, as show in Results section.

The type of coupling of inertial data along with vision data can be classified as:

- Loose Coupling
- Tight Coupling

Loose Coupling is a fusion in which we treat all packages sending us pose data as black box. We obtain pose estimates from various sensors, packages and fuse the pose data from any of the above methods. But the updated pose is not sent to

core algorithm so as to update pose with fused values.

Tight Coupling is a fusion in which we receive values from different sensors, and we choose any of the above mentioned fusion techniques to fuse data from different sensors and update the core algorithm with fused poses. This method works in most cases, since the update is reflected across the sensors and graph. In background, pose-optimization will ajust the pose, if needed. Ideally, this should give better results as compared to loose coupling.

4.4 Loose Coupling of IMU with LSD-SLAM

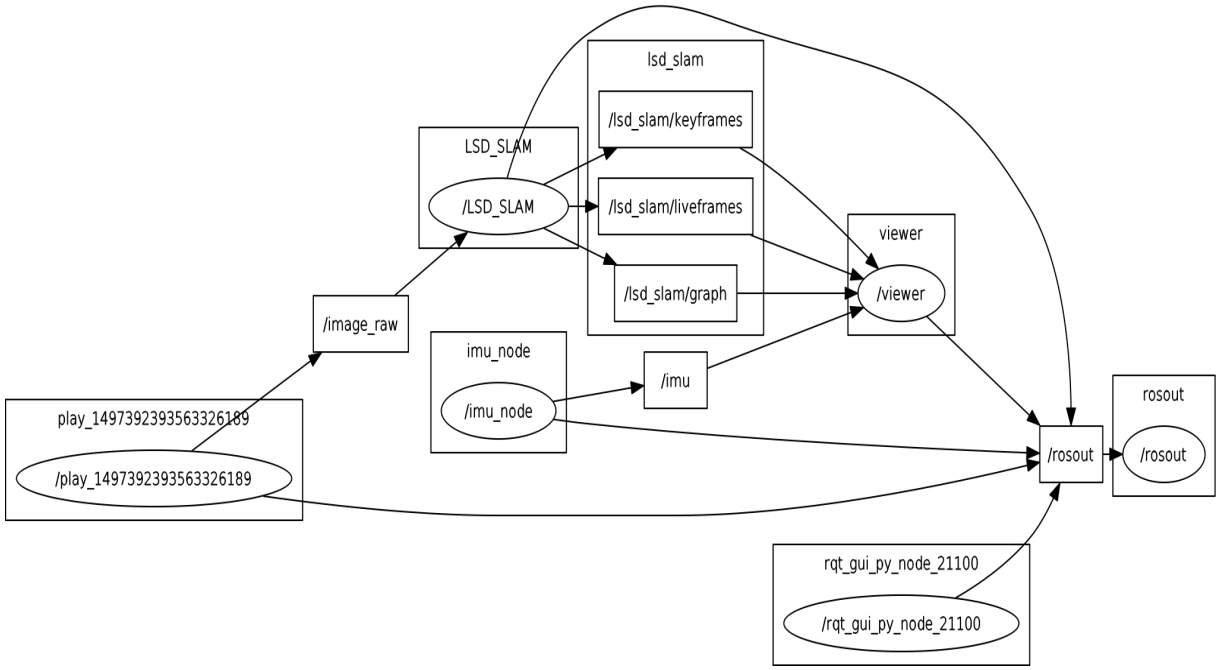

Figure 4.2: LSD-SLAM loose coupling

LSD-SLAM core algorithm publishes five topics:

1. **/lsd_slam/pose**

2. **/lsd_slam/liveframes**
3. **/lsd_slam/keyframes**
4. **/lsd_slam/graph**
5. **/lsd_slam/debug**

Viewer of LSD-SLAM subscribes to only 3 topics: /lsd_slam/pose, /lsd_slam/liveframes and /lsd_slam/keyframes. Also, the new LSD-SLAM viewer has been subscribed to IMU sensor data. This now allows viewer part to fuse the data from both sensors to couple the pose (orientation only) before displaying and publishing it.

4.5 Tight Coupling of IMU with LSD-SLAM

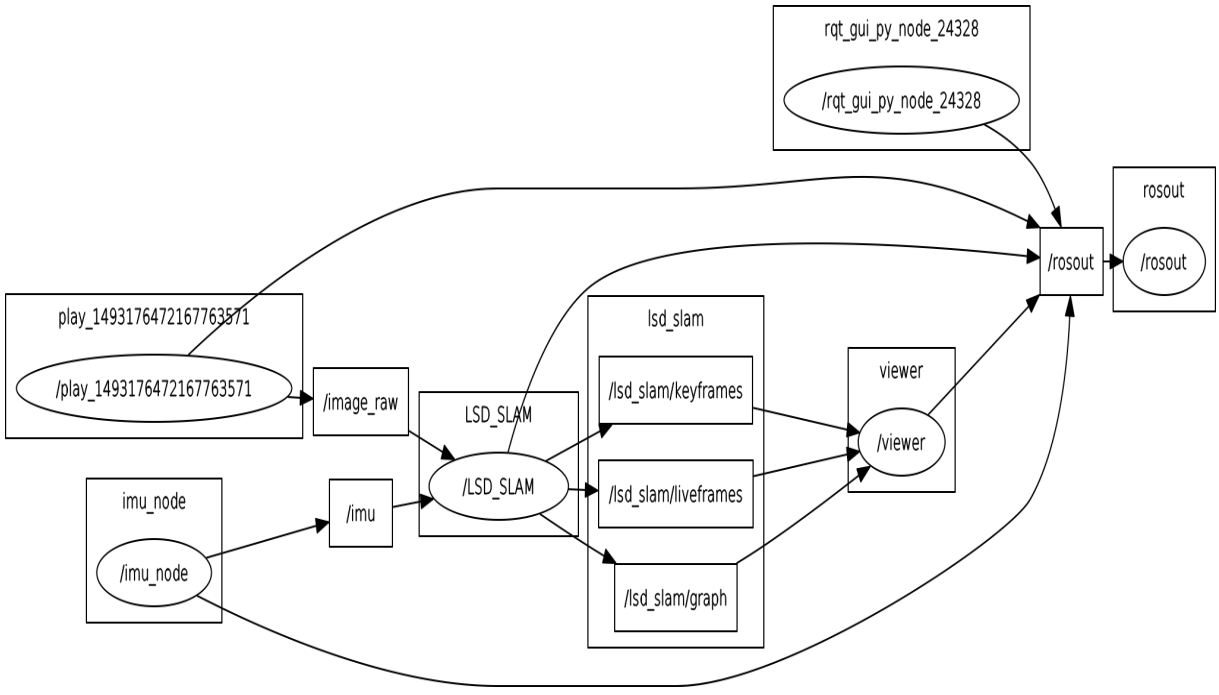

Figure 4.3: LSD-SLAM tight coupling

Instead of fusing IMU sensor data just before displaying in viewer, as done in previous section. Fusion of IMU sensor data (orientation only for now) is done with vision pose (orientation) in the step where relative pose of KeyFrame with respect to Tracked Frame.
This technique will produce better results as compared to above mentioned approach because the orientation from IMU is fused with orientation from vision (lsd-slam), and then optimization is performed over both estimates. This would be prone to less error as compared to a single estimate.

4.5.1 Description of Algorithm

As descibed previously, LSD-SLAM has 3 parts: **Tracking**, **Depth Map Estimation** and **Map Optimization**.
Pose estimation for each new *Current Camera-Frame* with respect to *Current Key-Frame* is done in *Tracking* part. Now, before initializing the relative pose of *Current Camera-Frame*, we fuse relative pose values (orientation) from IMU sensor.
NOTE: This pose (from vision) value is without scale, since Current Camera-Frame does not have any depth map associated with it.
NOTE: Since we are only fusing orientation, correlation of scale of IMU with LSD-SLAM scale does not matter.
The algorithm below describes the loose coupling of LSD-SLAM with IMU:

Algorithm 1 LSD-SLAM-IMU (loose coupling) algorithm

function LSD_SLAM_CORE($camera\_readings$)

while $True$ **do**

Extract $camera\_info$ and $image\_raw$

Track $camera\_motion$ and update map

function TRACKING($current\_frame$, $imu\_readings$)

$currentKF \leftarrow$ Find current KF

$pose\_vis, flags \leftarrow se3\_tracker(currentKF, current\_frame)$

Publish $pose\_vis$ to ROSOutputWrapper

function LSD_SLAM_VIEWER(lsd_slam_topics, imu_topic)

$pose\_imu \leftarrow calculate\_relative\_pose\_imu(imu\_readings)$

$pose\_final \leftarrow Fuse(pose\_vis, pose\_imu)$

Publish $pose\_final$ and display

function FUSE($pose\_vis$, $pose\_imu$)

$pose\_final \leftarrow weighted\_average(pose\_vis, pose\_imu)$

return $pose\_final$

function CALCULATE_RELATIVE_POSE_IMU($imu\_readings$)

$pose\_imu \leftarrow$ difference between imu reading of last and current frame

return $pose\_imu$

The algorithm below describes the tight coupling of LSD-SLAM with IMU:

Algorithm 2 LSD-SLAM-IMU (tight coupling) algorithm

function LSD_SLAM_CORE($camera\_readings$)

while $True$ **do**

Extract $camera\_info$ and $image\_raw$

Track $camera\_motion$ and update map

function TRACKING($current\_frame$, $imu\_readings$)

$currentKF \leftarrow$ Find current KF

$pose\_vis, flags \leftarrow se3\_tracker(currentKF, current\_frame)$

$pose\_imu \leftarrow calculate\_relative\_pose\_imu(imu\_readings)$

$pose\_final \leftarrow Fuse(pose\_vis, pose\_imu, flags)$

Publish $pose\_final$ to ROSOutputWrapper

function SE3_TRACKER($currentKF$, $current\_frame$)

while error < last_error **do**

$error \leftarrow photometric\_error(currentKF, back\_warped(current\_frame))$

return $pose\_vis$

function CALCULATE_RELATIVE_POSE_IMU($imu\_readings$)

$pose\_imu \leftarrow$ difference between imu reading of KF and current frame

return $pose\_imu$

function FUSE($pose\_vis$, $pose\_imu$, $flags$)

if flags == tracking lost **then**

$pose\_final \leftarrow weighted\_average(pose\_vis, pose\_imu)$

return $pose\_final$

else

return $pose\_vis$

CHAPTER 5

Implementations

5.1 Libraries and Softwares

Eigen

This library serves as a high-level C++ library for matrix, linear algebra and vector operations, geometrical transformation, etc.
This is also open-source and cross-platform. It can be downloaded at: `http://eigen.tuxfamily.org/index.php?title=Main_Page`

Sophus

This is an implementation of Lie Groups in C++ using Eigen Library, generally used in 2D or 3D geometric applicatons. This package includes special orthogonal groups SO(2) and SO(3), special Eucledian groups SE(2) and SE(3), and similarity group Sim(2) and Sim(3).
This library is most complicated to understand, and most of the tracking happen in this part only.
This library is cross-platform and open-source, and it can be downloaded from: `https://github.com/strasdat/Sophus` or `http://wiki.ros.org/sophus`.

OpenFABMAP

This is an appearance based library. This creates a bag of words for images and saves it. When a scene is possibly matched, it tries to match with another image at a higher level of pyramid. This is very crucial for loop closure. This library requires OpenCV non free module

g2o

This is a pose graph optimization library, used for Graph SLAM. This is similar to gstam which supports goegraphical variables, such as altitude, etc.

OpenCV

This is an OpenS Source Library for Comuter Vision, for performing some basic image processing tasks.

OpenGL

This library is responsible for drawing of objects and displaing in viewer.

5.2 Cameras

Two types of cameras were generally used: Monocular USB Camear and ZED Camera.

Monocular USB Camera: This camera has a resolution of 5MP with frame rate

of 15fps.

ZED Camera: This is the camera produced by stereolabs.com. It has a 4MP dual camera and can record 1080p HD videos at 100fps. Also, the field of vision is 110°. The depth sensor is able to estimate depth within range of 0.5m to 20m, having a baseline of 120mm. Shutter type of this camera is Rolling Shutter.
The best features of this camera is that is has SDK for easy development of apps and can is supported in ROS, unity, OpenCV, MATLAB across Linux and Windows.

5.3 tum-ardrone

tum-ardrone is a ROS packages for flying Parrot AR.Drone. This package contains the commands such as TAKEOFF, LAND, HOVER, GOTO(x,y,z), etc. which are useful for controlling the Drone manually or autonomously by a planning algorithm.

5.4 tum-simulator

tum-simulator is a ROS package for simulating drone in a virtual environment. User can feed control commands, and change is reflected in virtual environment. Also, the video feed from Drone is simulated along with other sensors. **Note:** We created zed wrapper and fixed imu issue

5.5 Integration

5.5.1 Overview

Firstly, the camera was attatched with IMU sensor using a cellotape. This was done to simulate Drone with Vision and IMU. Now, testing was performed with all combinations of IMU sensors and Visual Inputs, such as:

1. Vision Only
2. Vision (translation) , Vision + IMU (Orientation)
3. Vision (translation) + IMU (orientation)
4. IMU only (orientation)

After some initial testing of algorithm, same code was used to Operate on Parrot AR.Drone. It has front camera, bottom camera, and IMU sensors, which are of primary concerns for the project.
Main: Pose changes were proposed for major nodes.

1. Pose Estimation during Tracking Part (Tight Coupling)
2. LiveFrame publishing with modified poses (Tight/Loose coupling)
3. Viewer displaying modified poses (Loose coupling)

The results of experiment have been shown in Results chapter.

5.5.2 Challanges faced

Initially the lack of knowledge of Robotics terminology and their tools, caused a delay in the beginning. However, the greatest challange faced was to fuse IMU

pose with LSD-SLAM pose directly in main algorithm.

All poses are saved as elements of Sophus Library and Eigen Library. And no documentation is provided for it. Moreover the mutators are disable by default, so that one does not accidentally change the values of Sim3 poses of keyframes.

NOTE: One needs to keep in mind that LSD-SLAM does not store poses for frames are not stored as absolute, rather the are stored as relative to keyframes, whose poses are again relative to previous keyframes. This creates a major problem when one tries to update the pose. Some cases might require error correction over all keyframes stored, which is very difficult to implement correctly.

Also, the each pose is stored with a scale, and mutating the rotation and translation requires one to know the scale of keyframe. But currently, getting scale was not successful. Hence, the changes done in rotation and translation had a different scale. Thus resulting in a differently scaled DepthMap, leading to an error in merging PointClouds, and this leads to map having too much of noise.

CHAPTER 6

Results and Performance Study

We tested various existing, modified and new SLAM, Visual-Inertial SLAM, Visual Odometry, Visual-Inertial Odometry algroithms both with datasets and with live-camera. Following sections will provide more details and various evaluation matrices used to demonstrate it.

6.1 Datasets

To evaluate the 3D Map and 3D Odometry data, the following datasets are used:

1. Tum Dataset : Dataset for Mono and RGB-D camera by TUM University
   - LSD_room
   - LSD_eccv
   - LSD_foodcourt
   - LSD_machine

   Download link for dataset:

2. IITM-CVL-Dataset : Dataset created by authors of this thesis in CVL at IIT Madras during Autonomous Drone Project (June 2016 - June 2017) for testing it live
   - Parrot Ardrone Dataset (Vision + IMU)
   - Zed Dataset(with IMU)
   - Monocular USB Camera Dataset (Vision)

   Download link for dataset: http://tinyurl.com/IIT-CVL-16-17

6.2 Experiments

6.2.1 LSD-SLAM

LSD-SLAM algorithm on TUM dataset

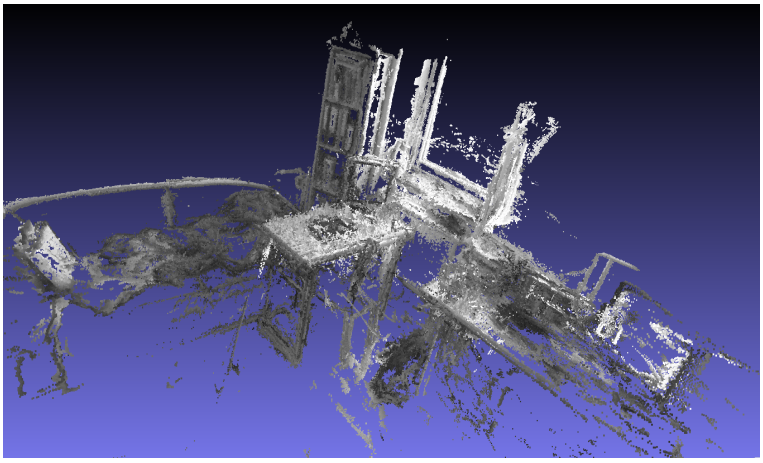

Figure 6.1: PointCloud generated by LSD-SLAM in Room [TUM Dataset]

Figure 6.2: An image of LSD-SLAM-Room [TUM Dataset]

LSD-SLAM algorithm with ZED Camera

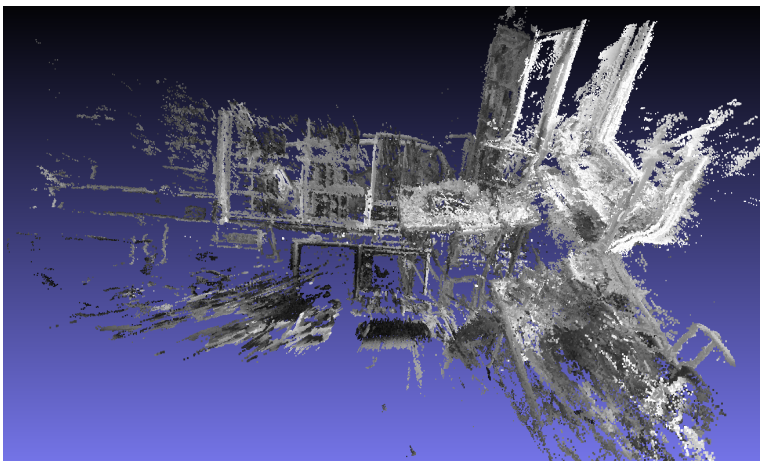

Figure 6.3: PointCloud generated by LSD-SLAM in CV Lab

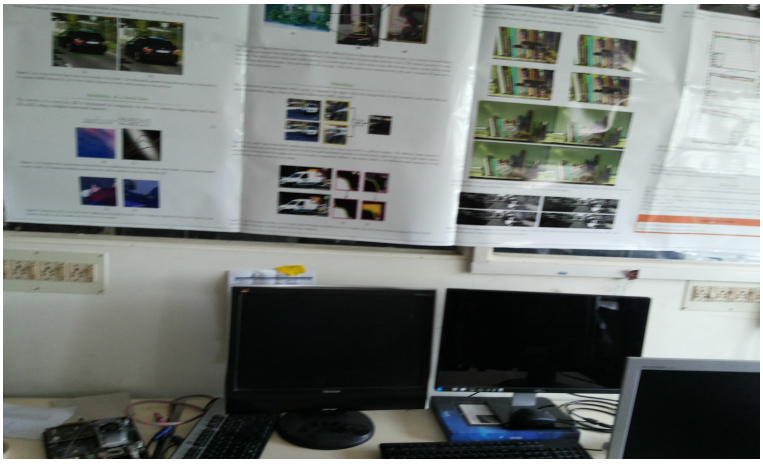

Figure 6.4: An image of same scene in CV Lab

6.2.2 Modified LSD-SLAM with IMU

Modified LSD-SLAM with Zed camera

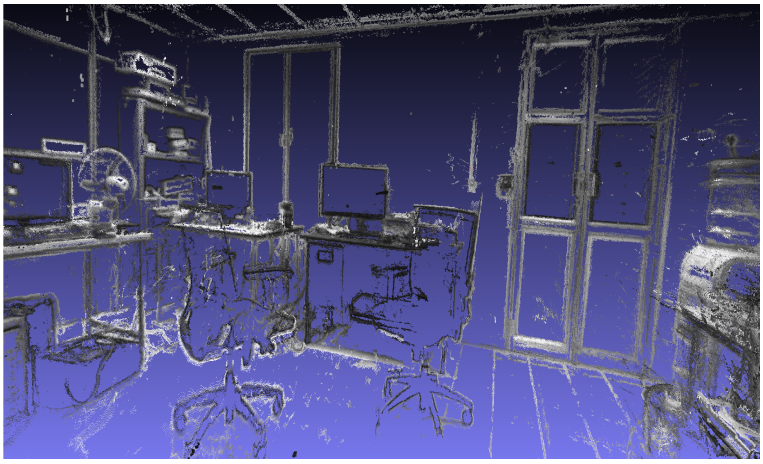

Figure 6.5: Point Cloud of CSE Department Server Room

Figure 6.6: An image of same scene in CSE Department Server Room

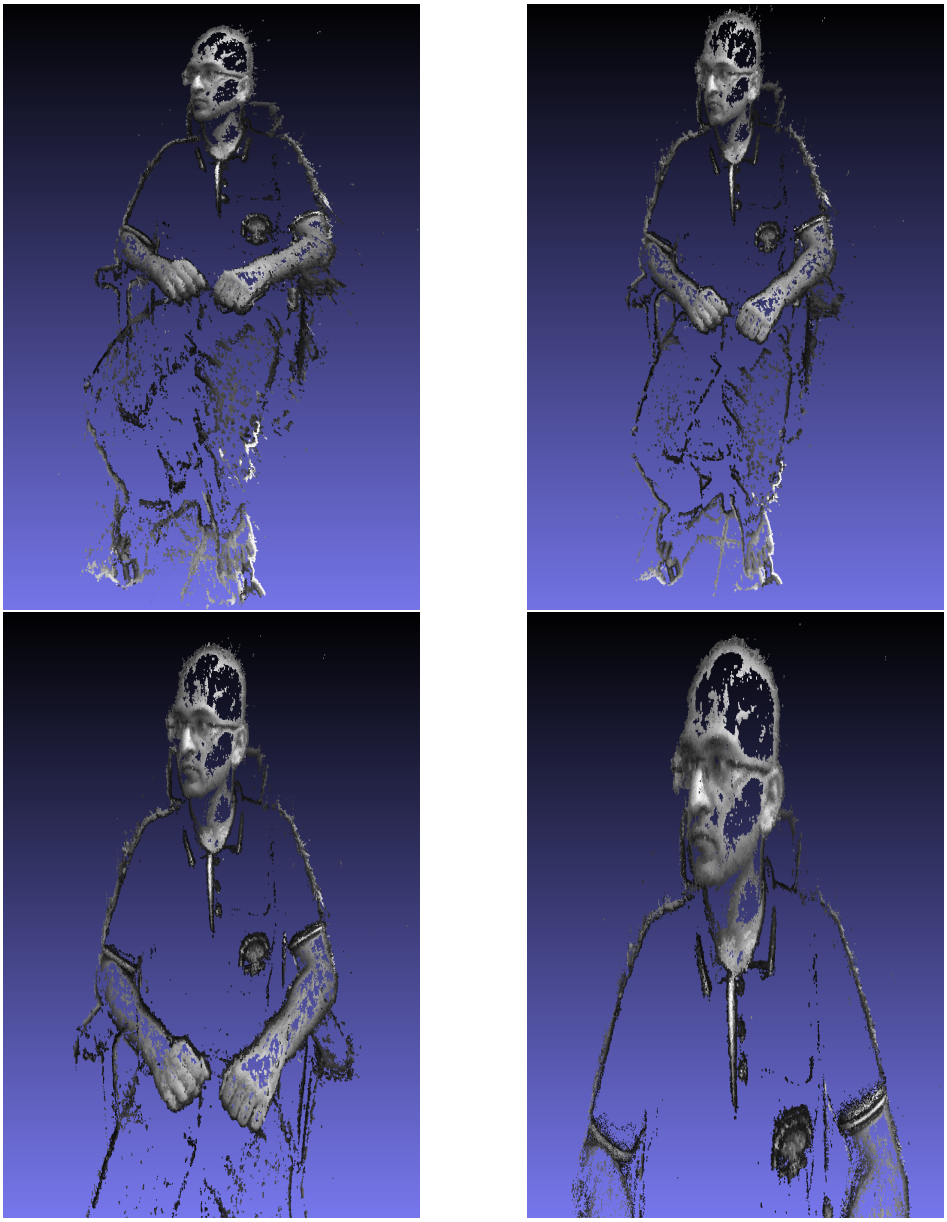

Figure 6.7: Different view-points for reconstruction of Author

6.3 Observations

Note: The code will run as per specification only for **min(length_of[vision], length_of[IMU])** [This problem arises while working with datasets, not over live streams].

6.3.1 LSD-SLAM

Experiments show that LSD-SLAM fails for regions where there is no edges or corner points. Also, for regions where there is improper ligting condition such as low illuminance or abrupt changes in illuminance. Also, for very fast motion or sudden/jerky movements, Vision fails catastrophically.

6.3.2 Loose Coupling of IMU with LSD-SLAM

Experiments indicate that this method help in retaining pose if vision looses its tracking, by using pose from IMU and have translation for very short time intervals.

6.3.3 Tight Coupling of IMU with LSD-SLAM

Experimental results show that this works better than loosly coupled, since here the tracking is rarely lost. It just gets accumulated with error, until a loop closure is found. Then we reinitialize the IMU poses with respect to updated vision pose, since after loop closure, a pose graph optimization will reduce the error accumulated.

6.3.4 Odometry based on Pure IMU

Experimental results show that using pure IMU, we are able to track orientation. But for translation is difficult to acheive, since gravitational acceleration is also factor which affects the estimation of position and velocity.

6.4 Evaluation Criteria

The evaluation criteria for the various experiments was targeted to test mainly two things:

1. Odometry Error
2. Depth Map and its Noise

6.4.1 Odometry Error

Various SLAM Algorithms generate output of agent's location in different formats. But visualization of Odometry data is best done as a path. Thus, by measuring the pose of algorithms w.r.t. groundtruth, at specific intervals of time, gives us the divergence from groundtruth. Since divergence can be measured, we can perform quantitative analysis on it across various datasets w.r.t. goundtruth.

6.4.2 Depth Map and its Noise

One of the most significant part of SLAM algorithms is the 3D Map reconstruction. One expects the map to convey the most important parts of environment, such as Walls, Edges, Windows, Doors, etc. This cannot be measured by any

Table 6.1: Evaluation - RMSE(cm) based on TUM RGB-D benchmark and simulated sequences

Dataset	**LSD-SLAM**	**[P1]**	**[P2]**	**[P3]**	**[P4]**
fr2/desk	4.52	13.50	x	1.77	9.5
fr2/xyz	1.47	3.79	24.28	1.18	2.6
sim/desk	0.04	1.53	-	0.27	-
sim/slowmo	0.35	2.21	-	0.13	-

where

- P1 : Engel *et al.* (2013)
- P2 : Klein and Murray (2007)
- P3 : Kerl *et al.* (2013)
- P4 : Endres *et al.* (2012)

appropriate means, since the 3D Map generated by various algorithms are not always in same format. Hence comparison of Maps is done by directly visualizing and estimating if the amount of noise is acceptable.

CHAPTER 7

Conclusion

Integration of IMU sensor with LSD-SLAM algorithm gives agent the ability to recover from sudden jerks, where visual tracking may be lost. Also, it can facilitate to maintain pose in environments with quick illumination changes or dark environments for short periods.

Note: This will fail to provide correct pose, if visual tracking is lost and imu is used for longer duration for pose estimation. This is due to the fact that error will be accumulated by double integrating accelerometer over time. However, *Orientation* obtained from IMU readings can be still reliable.

CHAPTER 8

Future Work

The next set of steps for the Project will be:

1. LSD-SLAM with Stereo: Implementation of cite LSD-SLAM stereo. This will provide a better estimate of scale of environment and Depth as well. Also, since baseline is known, tracking will be more accurate.

2. Autonomous Controls and Dynamic Path Planning by using Point Cloud of LSD-SLAM: User specifies the end point to Agent, and Agent decides path to destination and updates path on the fly, based on new information.

REFERENCES

1. **Endres, F.**, **J. Hess**, **N. Engelhard**, **J. Sturm**, **D. Cremers**, and **W. Burgard**, An evaluation of the rgb-d slam system. *In Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012.

2. **Engel, J.**, **V. Koltun**, and **D. Cremers** (2017). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

3. **Engel, J.**, **T. Schöps**, and **D. Cremers**, Lsd-slam: Large-scale direct monocular slam. *In European Conference on Computer Vision*. Springer, 2014.

4. **Engel, J.**, **J. Sturm**, and **D. Cremers**, Semi-dense visual odometry for a monocular camera. *In Proceedings of the IEEE international conference on computer vision*. 2013.

5. **Forster, C.**, **M. Pizzoli**, and **D. Scaramuzza**, Svo: Fast semi-direct monocular visual odometry. *In Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014.

6. **Kerl, C.**, **J. Sturm**, and **D. Cremers**, Dense visual slam for rgb-d cameras. *In Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013.

7. **Klein, G.** and **D. Murray**, Parallel tracking and mapping for small ar workspaces. *In Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE, 2007.

8. **Mur-Artal, R.**, **J. M. M. Montiel**, and **J. D. Tardos** (2015). Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, **31**(5), 1147–1163.

9. **Newcombe, R. A.**, **S. J. Lovegrove**, and **A. J. Davison**, Dtam: Dense tracking and mapping in real-time. *In Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011.