Implemention of Visual Simultaneous Localisation and Mapping (VSLAM) on Arm-based Architecture

Submitted in partial fulfillment of the requirements

of the degree of

Bachelor of Technology and Master of Technology

by

Kapil Yadav

(*Roll no.* 10D070062)

Supervisor:

Prof. Velmurugan Rajbabu

Department of Electrical Engineering

Indian Institute of Technology Bombay

2015

Dissertation Approval

This dissertation entitled **"Implemention of Visual Simultaneous Localisation and Mapping (VSLAM) on Arm-based Architecture"**, submitted by Kapil Yadav(Roll No. 10D070062), is approved for the award of degree of Bachelor of Technology and Master of Technologyin Electrical Engineering.

Examiners

____

____

Supervisor

____

Chairman

____

Date: 6th July 2015

Place: ____

Declaration of Authorship

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signature: ....

Kapil Yadav

10D070062

Date: 6th July 2015

Kapil Yadav/ Prof. Velmurugan Rajbabu (Supervisor): **"Implemention of Visual Simultaneous Localisation and Mapping (VSLAM) on Arm-based Architecture"**, *Dual Degree Dissertation*, Department of Electrical Engineering, Indian Institute of Technology Bombay, July 2015.

Abstract

Visual Simultaneous Localization and Mapping (VSLAM) is an useful algorithm for high-end robots that perform complex tasks like navigating autonomously in a map, picking and moving objects autonomously, etc. It requires significant computational power to process incoming video in real time and take robot control decisions with sufficient speed and accuracy. Such systems tend to be large such as a laptop with significant computing capability. Howwever, with the improvement in microprocessor technology, more processing power can be packed in small Single Board Computers (SBC). This thesis is the report of an attempt to implement Visual SLAM algorithm on recently released Raspberry Pi 2 that has 900Mhz quad core processor. A simple robotic vehicle that has a Kinect camera and Raspberry Pi 2 mounted on it has been built. The objective is to move the vehicle autonomously to different positions in a small region using only VSLAM algorithm for localisation and mapping. The corresponding interface software i.e. a Robot operating System (ROS) package has also been developed and can be used by other similar systems.

Contents

List of Figures

List of Tables

-

Chapter 1

Introduction

Visual SLAM (Simultaneous Localization and Mapping) is a problem of building a 3D model of an unknown scene by a moving camera that travels through the scene and simultaneously recovering the camera's trajectory. It has received much attention in last few years because of its use in autonomous systems and recent improvements in computational power. As a result, it has been implemented on mobile devices [15], used in Augmented Reality [8] and other applications. Camera based algorithms are particularly attractive because of their low cost and light weight properties. However, most of the algorithms are computationally intensive.

We will first define the problem; review its existing solutions and finally, review merits and demerits of VSLAM. This thesis is the report of an attempt to implement VSLAM using Arm-based Single Board Chip (SBC).

1.1 Need for a navigation system

All autonomous robots need to be aware of their own position w.r.t. its environment to do their task autonomously. And in some cases, need 3-D maps of their surroundings. For example when a home robot is commanded to give a cup of coffee, it should be able to track its own position and trajectory while moving towards the coffee machine as well needs to be aware of its surroundings. Similarly, when a quadcopter is commanded to take images, it needs to know its position and its trajectory to successfully complete the task and be aware of its surroundings to autonomously navigate. A driverless car needs to know its position on the road w.r.t. its surroundings. Visual SLAM can also be used as a blind man's aid for indoor and outdoor navigation. In case of a robotic arm, more precision is required when picking up an object because it needs to know its own position and trajectory to align itself with the object. For autonomous robots, one often needs 3-D map of the surroundings to navigate and determine their path through the surroundings.

1.2 Alternatives of VSLAM

Various alternatives have been proposed for localization (Visual Odometry). For example, one can use wheel odometry, GPS, Laser odometry, wifi signals, or IMU. However, all of these have their respective disadvanages and fail to operate in one or the other conditions. Wheel odometry fails when wheel slips or in uneven terrain, etc. and is not applicable in robots without wheels. GPS is not present inside buildings, below tree canopy, GPS – denied environments, etc. It is dependent on external satellite infrastructure to function and has less accuracy. One can also use laser odometry for localization but LIDARs are expensive and bulky and need high computational power. On the prospect of using WiFi signals, they are not present everywhere and external infrastructure is required. For IMU, error accumulation takes place over time and cannot generate 3-D maps. To generate 3-D maps (SLAM) (without using any external markers), the alternatives are LIDAR but it is bulky, computationally intensive, requires high power consumption since it is an active device and acquires no color information [1]. On the other hand, camera is light-weight and has low power consumption.

1.3 VSLAM as a solution - merits and demerits

Visual SLAM has various advantages as well as disadvantages -

Advantages

1. Same infrastructure can be used to generate partial 3-D maps of the environment since we have camera orientation information and surrounding video (Visual SLAM).
2. Can have more accuracy than existing solutions such as GPS.[2]
3. Particularly useful in robots that need a camera for other tasks too.
4. Very general approach that can be used indoors as well as outdoors

Disadvantages

1. Need a camera
2. Can be computationally intensive
3. Difficult to implement

In short, it solves both the problems of localization and mapping with one sensor – camera. For successful implementation of Visual SLAM in real – world scenario, we need Visual SLAM algorithm to work in indoor as well as outdoor environment, ablility to generate large maps, accurate enough so that autonomous navigation can be performed, fast enough and computationally light for real time performance and camera system to be light and small so that it can be used in embedded systems. Several attempts have been made to solve the Visual SLAM problem but they have often lagged behind in one or more of the above mentioned properties. And hence, there is still a lot of scope for improvement and research.

1.4 The Approach

We describe the approach for Visual SLAM next .

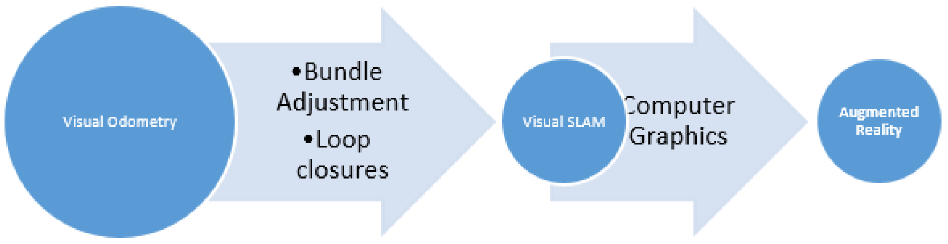

Figure 1.1: The approach with increasing difficulty

Visual odometry is the tracking the camera movements by analyzing a series of images taken by the camera. It is the most important and crucial step. Once visual odometry is accurately implemented, then by using bundle adjustment and detecting loops closures[3] in the visual odometry data, 3-D maps can be generated. This process gives us the algorithm for Visual SLAM. This thesis deals with the details of implementation of the first step i.e. visual odometry for indoor environments. Once this is achieved then, generating 3-D maps of the indoor environment will be the next step. Devising new algorithmic approaches that can generate more accurate maps by using lesser computational power that existing algorithms can be the next step. Another direction of approach after maps are generated can be to apply the algorithm in various indoor autonomous robots like home cleaning bots, task-performing robots, indoor quadcopters, in mobile phones (monocular Visual SLAM). Cognition can be added to the algorithm and robots can be built for one of the above mentioned applications. The specific application can be chosen later depending on the quality of maps generated, computational power required by the algorithm, its real-time performance, etc. Another approach after generating 3-D maps can be to venture in the field of augmented reality.

In the following chapters, different existing algorithms have been implemented and their performance tested. Then we made a small robotic vehicle whose aim is to autonomously navigate in small region using only visual odometry.

-

Chapter 2

Visual Odometry - algorithm

Visual odometry is the process of estimating the egomotion of an agent (e.g., vehicle, human, and robot) using only the input of a single or multiple cameras attached to it. In order to generate 3D maps, one needs to correctly estimate camera motion which will in turn affect the stitching together of the point-clouds of the scenes. Therefore, visual odometry is the most important part of the VSLAM algorithm. an accurate visual odometry leads to an accurate VSLAM algorithm.

2.1 Pre-Liminary concepts

2.1.1 Camera Projection

We want a mathematical model to describe how 3-D world points get projected into 2D pixel coordinates. Objects in world coordinate system are usually represented by (U,V,W) and in camera coordinates by (X,Y,Z). We want to establish the following forward relation –

Most of the vision problem is related to finding backward projection. However, we will discuss the forward projection first. Conversion from camera coordinates to film coordinates is called perspective projection. It is represented as follows –

$$x = f\frac{X}{Z}, y = f\frac{Y}{Z}$$

where (x,y) are the film coordinates, (X,Y,Z) are camera co-ordinates and f is the focal length. These equations are obtained by applying similarity of triangle in the geometry containing

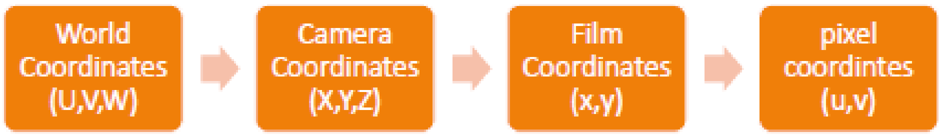

Figure 1- forward projection

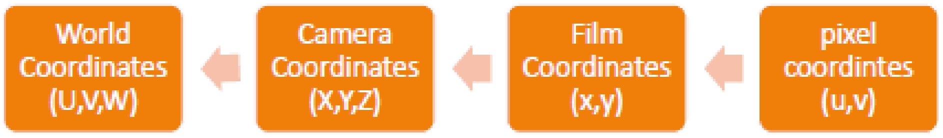

Figure 2 - backward projection

Figure 2.1: Camera Projection : forward and backward projection between world and camera co-ordinates

projection of a point from camera coordinates to film coordinates. Upon furthur solving for the transformations, the final transformation equation comes out to be -

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{s_x} & 0 & o_x \\ 0 & \frac{1}{s_x} & o_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_1 1 & r_1 2 & r_1 3 & 0 \\ r_2 1 & r_2 2 & r_2 3 & 0 \\ r_2 1 & r_2 2 & r_3 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} U \\ V \\ W \\ 1 \end{bmatrix}$$

where $r_{i,j}$ and ci are the elements of rotation matrix and translation matrix C from world co-ordinates to camera co-ordinates, (U,V,W,1) is augmented vector in world coordinates system. The 1^{st} matrix represents affine transformation, 2^{nd} matrix represents perspective transformation, 3^{rd} and 4^{th} matrices represents rotation and translation from world co-ordinates to camera co-ordinates.

2.1.2 Essential Matrix

Essential matrix(E) is a 3x3 matrix that can relate corresponding points in stereo images for the same 3-D point in the real world (assuming the camera satisfies the pin-hole camera model). It establishes constrains between matching image points. Using essential matrix, we can determine both relative position and orientation between cameras and the 3D coordinates in the real world.

If y and y' are homogenous normalized image coordinates that correspond to the same 3-D point in left and right image in stereo, then essential matrix is defined such that –

$$(y')^T E y = 0$$

Finding Essential matrix for stereo vision

Let the 3-D coordinates of a real world point P be (x_1, x_2, x_3) and (x'_1, x'_2, x'_3) for left and right images in their respective camera's coordinate system. The coordinates can be expressed as –

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{x_3} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \begin{bmatrix} y'_1 \\ y'_2 \end{bmatrix} = \frac{1}{x'_3} \begin{bmatrix} x'_1 \\ x'_2 \end{bmatrix} \text{(since the images are normalized)}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ 1 \end{bmatrix} = \frac{1}{x_3} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y'_1 \\ y'_2 \\ 1 \end{bmatrix} = \frac{1}{x'_3} \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix} \text{(in homogenous form)}$$

$$\text{OR } y = \frac{1}{x_3}\widetilde{x}, y' = \frac{1}{x'_3}\widetilde{x'}$$

where y' and y are homogenous coordinates. Also, since $\widetilde{x}$ and $\widetilde{x'}$ are coordinates of same 3-D point but in different coordinate system, they can be related as –

$$\widetilde{x'} = R(\widetilde{x} - t)$$

Where R and t are rotation and translation matrices respectively. R is 3x3 matrix and t is 3-D vector. Let's define essential matrix and then check its validity. Let the essential matrix be –

$$E = R[t]_x$$

where $[t]_x]$ is cross product's matrix representation i.e.

$$[t]_x = \begin{bmatrix} 0 & -t_z & t_z \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{bmatrix}$$

2.1.3 Random sample consensus (RANSAC)

RANSAC algorithm is based on formulating a hypothesis model from randomly sampled datasets and then verifying this hypothesis by using remaining datasets. This process is repeated for different initial datasets and the hypothesis that shows highest consensus from other data points is selected. In case of Visual SLAM or Visual Odometry, the hypothesis is the transformation (R,T) and data points are the feature correspondences. RANSAC algorithm can be summarized as –

1. Randomly select a set of s feature points from all available feature points in the image.
2. Fit a model to these selected points.
3. Calculate distance of all other points from this model
4. If distance is smaller from some value d, put it in the inlier set.
5. Repeat from step 1 again until maximum number of iterations is reached. And store the model having largest inlier dataset.
6. Finally, estimate the model using the largest inlier dataset.

Number of iterations necessary is given by the following formula –

$$iterations = \frac{log(1-p)}{log(1-(1-w)^s)}$$

Where w is the probability that a data-point is an outlier, s is the number of data-points from which the model is generated and p is the required success probability. RANSAC is probabilistic model and its output defers every time it is run. However, its result is generally stable if iterations are considerably high.

2.2 Formulating the visual odometry problem

Let us assume that a camera is moving in a 3-D environment. The camera can be either monocular or stereo and consequently the series of images captured can be –

$$I_{0:n} = I_1, I_2, I_3, ..., I_n \text{ and } I_{1,0:n} = I_1, I_2, I_3, ..., I_n, I_{r,0:n} = I_1, I_2, I_3, ..., I_n$$

for monocular and stereo camera respectively. Here n indicates discrete instances of time. We want to find out rigid body transformation between these images. Let $T_{k,k-1}$ be the rigid body transformation between consecutive images I_k and I_{k-1}.Then$T_{k,k-1}$ can be expressed as

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}$$

where $R_{k,k-1}$ is a 3 times 3 rotation matrix and $t_{k,k-1}$ is a translation vector. Let C_k be the camera pose at time instant k. Thus, all camera poses till current time instant can be represented as $C_{0:n}$. We want to find C_n - the current camera pose. It can be found out by successively multiplying all consecutive transformations $T_{1,0:n,n-1}$ to original camera pose C_0.

Therefore, the visual odometry problem can be formulated as finding $T_{k,k-1}$ for all values of k (original camera pose C_0 will be given) and concatenating them to the previous camera pose.

2.3 Solving visual odometry problem

2.3.1 Overview

As mentioned in section 4, in order to solve odometry problem we want to find rigid body transformations $T_{k,k-1}$ between consecutive images I_k and I_{k-1}. To solve these $T_{k,k-1}$, these are two approaches –

1. *Appearance based methods (global methods)* – These methods use pixel intensity and color information to establish relationship between consecutive images. These methods fail in homogenous environments and are computationally intensive but are easy to implement.

2. *Feature based methods* – These methods use salient feature like SIFT, harris corner detectors, etc to establish relationship between consecutive images. These are fast but need robust implementation for high accuracy.

The Visual odometry process can be graphically represented as follows -

2.3.2 Camera Modeling - Camera Parameters

One can use perspective camera, omnidirectional camera or spherical one. I have used perspective camera, so I will be explain perspective camera model only.

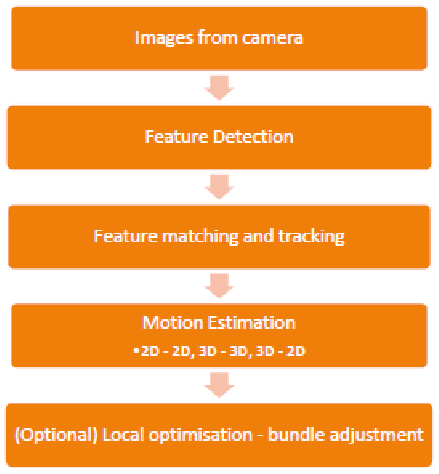
Images from camera
Feature Detection
Feature matching and tracking
Motion Estimation
•2D - 2D, 3D - 3D, 3D - 2D
(Optional) Local optimisation - bundle adjustment

Figure 2.2: Visual Odometry pipeline

Perspective camera and its calibration

As discussed in section 3, perspective camera model can be represented by pin-hole projection system. The governing equation for 3-D real world to 2-D image co-ordinate conversion is given by –

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a_u & 0 & u_0 \\ 0 & a_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Where $\begin{bmatrix} x & y & z \end{bmatrix}_T$ is a point in camera reference frame and $\begin{bmatrix} x & y \end{bmatrix}_T$ is the point in image plane, a_u ,a_v are focal lengths and u_0 ,v_0 are image coordinates of the projection center Camera calibration is performed using planar checkerboard like pattern. The dimensions of the checkerboard patterns are known and several pictures of the image are taken. Least-square minimization method is used to find the intrinsic and extrinsic parameters of the camera. I have used RGB-D camera Microsoft Kinect for XBOX 360 on Dual core computer running Ubuntu.

2.3.3 Feature detector, descriptor, matching and tracking

Feature Detection

Two types of features can be used for detection namely point features and line features. The point features extract salient keypoints in the image whereas line features extract lines as suggested by its name. However line features pose problems during matching because they are more susceptible to occlusion and perspective view changes. So, we will be using point features for visual odometry. Feature detection and matching can be done in two ways –

Table 2.1: Corner detector vs blob detector

Corner detector	Blob detector
Detects corners in an image	Detects regions that differ from surroundings in terms of color, texture and intensity
Computationally fast	Computationally slow
Less distinctive, hence, difficult to localize	More distinctive, hence, easily localizable
Better localized in image position	Less localized in image position
Less localized for different image scale	More localized for different image scale

1. Detecting features in one image and tracking them in subsequent images. This can be done using local search techniques such as co-relation. This method is more suitable when there are less abrupt changes in the camera movement.

2. Independently detect features in all images and them match them for performing visual odometry. This method is robust to abrupt camera movements.

For Visual Odometry, point features are the most commonly used features. They can be categorized as –

1. *Corner detectors* – like Forstner, Moravec, FAST, Harris, etc

2. *Blob detectors* – like SURF, SIFT, CENSURE, etc.

A comparision between corner and blob detectors is given below –

SIFT feature detection is most commonly used in visual odometry and VSLAM applications. SIFT features are both scale invariant and perspective invariant. Another detector SURF is also used. This detector is built on same concepts of SIFT but is computationally faster

Feature distribution in an image

The distribution of features in an image significantly affects the accuracy of Visual Odometry. Higher the number of features in every image, higher will be the accuracy. Moreover, homogeneity of features will also affect the accuracy. For better accuracy, features should be evenly distributed throughout the image. Since, SIFT features are more concentrated in textured regions, this is not the case for most of the times. To enforce homogeneity, every image is divided into a grid and features are extracted in every box of the grid. This significantly increases accuracy and is widely used in VO applications.

Feature descriptor

After feature detection, it has to be represented in such a way that the feature is compact and can be easily matched with other descriptors. Some feature descriptors are discussed below –

1. *Intensity based descriptors* - They store intensity of pixels of patches around the the feature point. For inter-image matching, they use error matrices such as normalized cross-correlation (NCC), sum of squared differences (SSD), census transform, etc. Census transform stores the feature by building binary vector with its values consisting of 1 or 0 depending on whether the neighboring pixels are more or less than the center pixel intensity value. For census transform, inter-image matching is done by calculating hamming distance. However, these descriptors are susceptible to small changes in orientation, intensity and scale, hence, are not preferred.

2. *SIFT feature descriptor* - It is a histogram of local gradient orientations. In this method, each patch is divided into 4x4 quadrants. For every quadrant, a histogram of 8 gradient orientation is made. So, a total of 128 elements are present in each descriptor. This descriptor is found to be highly stable with least variance to scale, intensity and orientation changes.

3. *BRIEF, FAST descriptors* - these descriptors have emerged recently and are believed to be faster as well as invariant to scale, orientation and intensity.

In most of the VO implementations, proven SIFT descriptors are used.

Feature Matching

Corresponding features are matched across images for tracking camera motion. For intensity based descriptors, SSD or NCC is used. For SIFT descriptors, Euclidean distance is used. For SIFT features, distance-ratio test is also used to prevent outliers. The distance-ratio test measures the ratio of closest match distance and second closest match distance. If this ratio is less than some fixed quantity, then the closest match is accepted. This helps in removing outliers and also removing incorrect feature matches when repetition of patterns present in images. The threshold can be set by user or can be dynamically adjusted according to the conditions. In the simplest case, all features of first image are compared with all features with the second image and those features are paired that have minimum distance between them. Comparing all features of one image with all features other image is computationally intensive. If we search for corresponding features in a specific region in the second image, then the process can become faster. Motion model (constraints on camera motion)can be built to define these regions. Other sensors like IMU, laser, GPS can also be used to detect motion. When motion model is known (from IMU), then epipolar lines concept can be used to significantly limit matching. Epipolar lines can be constructed directly from the 2 camera positions and the 2-D feature in first image. The corresponding feature will be present on these epipolar lines. Concept of epipolar lines is also used in stereo vision case to compute correspondences between left and right images. Further, left and right images are rectified such that the epipolar lines for all features in the left image are horizontal in the right image. This process is called image rectification and helps in reducing computational complexity since image rectification can be done on a separate dedicated GPU.

Feature Tracking

Feature tracking is done in applications where small-scale environment is used. In this method, features are detected in first image and same features are tracked in subsequent images. Thus, if there is a large changes in feature sets for subsequent images, then this method fails.

2.3.4 Outlier Removal

Some of the matched features are incorrectly associated and they reduce the accuracy of estimated camera motion. Outliers occur because of various factors such as occlusions, blur, image noise, illuminations, etc. Outlier removal is an important task in VO and can affect the accuracy significantly. Most commonly used outlier removal technique is RANSAC. Using RANSAC is discussed in section 3.5

2.3.5 Motion Estimation

Motion estimation is the most important step in Visual odometry. At this stage, transformations T_k are calculated from feature sets f_k and f_{k-1} corresponding to images I_k and I_{k-1}. By concatenating all these transformations T_k , camera's motion can be tracked. Motion estimation can be performed using following 2 methods depending on camera used –

1. *Motion from 3-D feature sets* - Used by stereo or RGBD camera systems. In this method, features fk-1 and fk are both defined in 3-D i.e. depth of every feature point is known. Motion estimation using this method is absolute.

2. *2Motion from 2-D feature sets* - Used by monocular camera systems mostly. In this method, features fk-1 and fk are both defined in 2-D. Motion estimation using this method is more computationally intensive and trajectory obtained is relative.

Error Propogation For visual odometry, transformations Tk,k-1 are concatenated to previous camera poses to form current camera pose Ck . Therefore, if there is error in transformation Tk,k-1, it will affect the accuracy of current camera pose Ck. And since current camera pose will be used to calculate next camera pose Ck+1, the error will propagate to all future camera poses. To limit this effect, a number of solutions have been proposed. Most simple solution is to calculate Ti,j for any values of i and j i.e. images used for calculating transformations need not be adjacent. This will reduce error propagation because lesser number of transformations will be used for the overall camera motion estimation. However, this method is not feasible at a number of times because images that are not adjacent might have less common features and transformation calculations might be erroneous.

Chapter 3

Evaluation of VSLAM algorithms

Sdeveral organisations and universities have implemented VSLAM according to their specific requirements. Some have made their implementation as open-source code. as part of this thesis, some of these open-source algorithms were implemented and their performance was reviewed. This section describes the history of VSLAM and a review of the implemented algorithms.

3.1 VSLAM algorithms

Most of the early research in visual odometry before 2005 was motivated by the NASA Mars exploration program in an attempt to provide all-terrain rovers that can localize themselves as they move in the presence of wheel slippage due to uneven and rough terrains. In 2005, one of the first implementation of VSLAM algorithms (for cars) caught the worldwide attention[**?**]. It was the self-driving STANLEY car developed by Stanford university that won the DARPA Grand Challenge. Since then, there have been active research going on in this area. Several modifications have been done by researchers on the basic feature-based VSLAM to make them adequate for their requirements.

3.2 Review of existing algorithms

Dense VSLAM [4] that minimizes both the photometric and the depth error over all pixels to compute poses instead of only depth data for RGBD cameras. This increases their performance in scenes with low texture as well as low structure where there are not enough features to use feature-based algorithm efficiently. It also significantly decreases the pose-estimation error as photometric data is used in addition to depth data.

Parallel Tracking and Mapping (PTAM) [5] is a popular technique developed in 2007 by Georg Klein and David Murray. This algorithm proposed to split tracking and mapping into two separate tasks. They were processed in parallel threads on a dual-core computer for a hand-held camera. The use of parallel threads for localisation and mapping allowed the use of computationally expensive batch optimisation techniques not usually associated with real-time operation. Thus, it allowed to use cheap cameras for SLAM with accuracy and robustness rivalling the expensive stereo and laser based systems. However, it was suitable for small AR workspaces only.

Large Scale Direct Monocular SLAM [6] (LSD-SLAM) uses direct (i.e. featureless) monocular SLAM algorithm. This allows to build large-scale, consistent maps of the environment with minimum scale-drift. Basic VSLAM algorithms uses features in the images to estimate poses.

Semi-direct monocular visual odometry (SVO) [7] algorithm is another example of using pixel intensities to estimate poses. This removes the requirement of feature calculations and makes the algorithm fast enough to be used in embedded environments. This is claimed to run at 55 frames per second on an onboard embedded computer. However, direct and semi-direct method are at a disadvantage in homogenous environmets.

Real-Time Appearance-Based Mapping (RTABMAP) [2] is a new algorithm that is capable of multi-session mapping along with loop-closure detection. A new graph-based SLAM system is proposed that uses a memory management approach perform to global-loop closure in real-time.

3.3 Performance review of implemented algorithms

Some of the algorithms described in the earlier section were implemented and their performance is discussed in this section.

3.3.1 LIBVISO2 - Library for Visual Odometry 2

Libviso2 is a C++ library that can build 3D-maps from high-resolution stereo sequences in real-time. It uses sparce stereo matching and visual odometry to generate 3D strucutre. It can run at 25 frames per second. It does not support RGBD camera. We implemented this algorithm and ran it on karlsruhe_sequences [5] to test its speed and accuracy. We sumarize our results here -

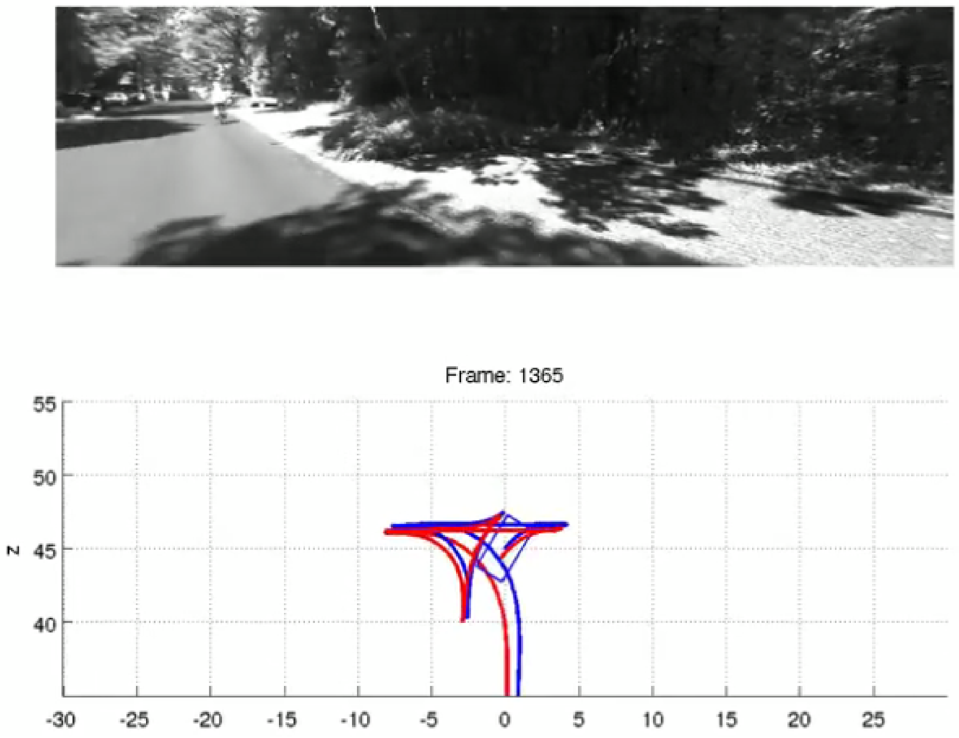

Figure 3.1: Running Libviso2 on karlsruhe dataset. Red line is the ground truth and blue line is the libviso2 trajectory line of a car moving in the streets

It assumes fixed height from the ground i.e. it is a 4-dof algorithm. It is best for outdoor applications during daylight as the indoor becnhmarks had low accuracy and were not useful for indoor navigation. This algorithm is more susceptible to changing light conditions. The speed was about 25 frames per second. Fig 2.1 shows the trajectory of a car moving in the streets. The red line is the trajectory of the car calculated using the visual odometry and blue line is the ground truth.

3.3.2 FOVIS - Fast Odometry from VISion

Fovis [8] is one of the first libraries that uses RGBD camera for visual odometry in 6-dof. A RGB-D camera (Kinect XBOX 360) on 1.9GHz Dual core linux-based desktop computer was used to implement FOVIS. Kinect camera was moved in various patterns and the VO algorithm's results and performance as was recorded. For plotting camera trajectory in 3-D, PointCloud library by WillowGarage [8] was used.

The algorithm was able to successfully track the camera movements with accuracy in centimeters. But the algorithm was too slow for real time applications. It was running at only 5-6 fps, which restricted camera movement speed considerably for accurate tracking. Several

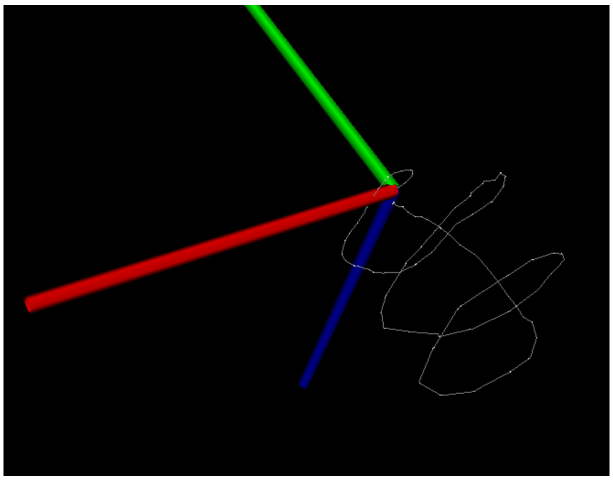

Figure 3.2: Running fovis using handheld kinect : Trajectory of Kinect camera traced by the VO algorithm as the camera is moved in a spiral manner and placed back to its original position

reasons for the low fps were suspected – kinect camera was changed to rule out faulty camera, several parts of the algorithm were turned off to identify which part is consuming the most time. It was inferred that acquiring depth data from kinect was consuming most of the time and was bringing the overall fps down. One reason for this behavior can be using non-standard openni library that is required to run Kinect on Ubuntu (since there is no official library for Ubuntu from Kinect developers). Fig 2.2 shows the trajectory of Kinect camera traced by the VO algorithm as a handheld kinect camera is moved in a spiral manner and placed back to its original position

3.3.3 SVO - Semi Direct Visual Odometry

SVO [7] can estimate poses and even make 3D point-clouds using only a monocular camera (e.g. a simple webcam). A Logitech digital webcam was used on a dual core computer to implement this algorithm. Here are some images of the implementation -

Since, this algorithm uses direct method i.e. pixel intensity values for localisation, it can perform satisfactorily only in rich environments. In, homogenous scenes where there are very few keyframes, this algorithm loses its localisation trajectory. Also, it is more computationally intensive than other algorithms because it uses 3 or more frames to traingulate a point to compensate for availability of only monocular camera. However, since it loses its localisation trajectory often, it is not suitable for reliable localisation in embedded environment. Fig 2.3 shows the trajectory of kinect camera as it is moved around.

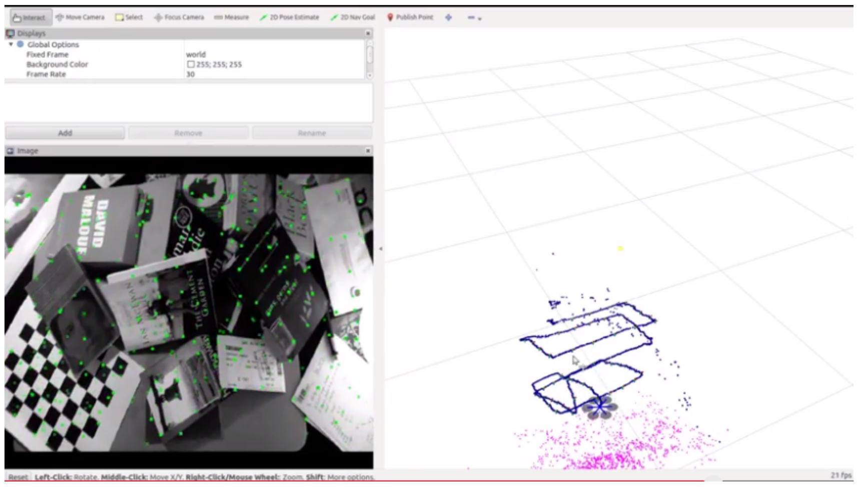
Figure 3.3: Running SVO using handheld webcam

3.3.4 RTABMAP - Real Time Appearance Based Mapping

RTABMAP [2] is a new RGBD-camera based algorithm released in 2014 that is capable of multi-session mapping along with loop-closure detection. It has both visual odometry and SLAM packages. It itself uses several other libraries like point cloud library, eigen3(Mathematical functions library), opencv(image processing library), sqlite (for making gui for visualization), etc. This algorithm is good even at camera rotations and has accuracy in centimeters. Its frame rate is software limited to 1Hz. This needs to increased for better accuracy but it also increases computational load on the system. It does not lose its localisation easily and when it does due to insufficient depth points or homeogenous surface, it can resume its localisation once any one of the present camera frames matches the previous keypoint frames in the system.

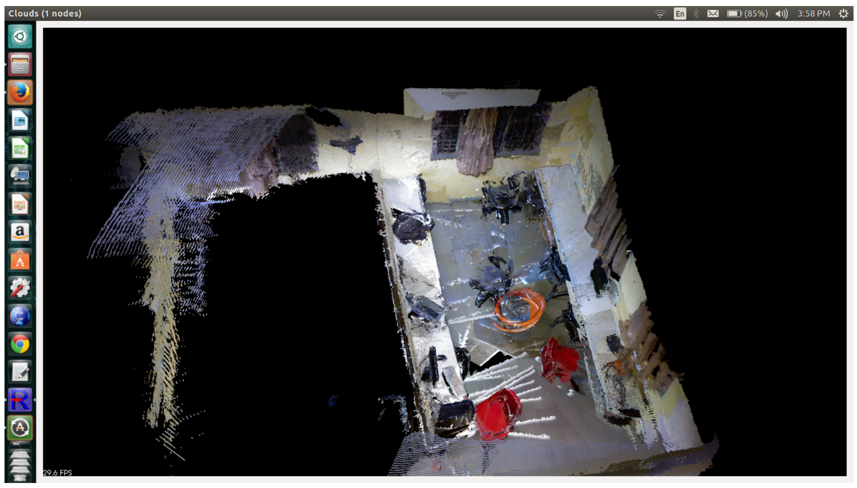
Figure 3.4: 3D map of computer room of hostel 5 created using rtabmap and handheld kinect camera

This algorithm can also be used to make 3D maps of the surroundings by a handheld kinect camera in real time. It was used to make a 3D map of a computer room at one of the hostels of IIT Bombay. Fig 2.4 shows a still from that map -

It used registered point clouds from the openni driver(kinect camera driver for ubuntu) and visual odometry to generate 3D maps with centimeter accuracy. The result were satisfactoy woth centimeter accuracy and the process could be used in any indoor conditions. However, the SLAM is computationally intensive and took more than 50 percent of the processing power of a quad-core i5 laptop.This library can also be used to record the kinect video data and perform VSLAM later on it. This is particularly useful for making maps using a robot fitted with kinect that doesnt have enough computational power.

-

Chapter 4

The Autonomous robotic vehicle

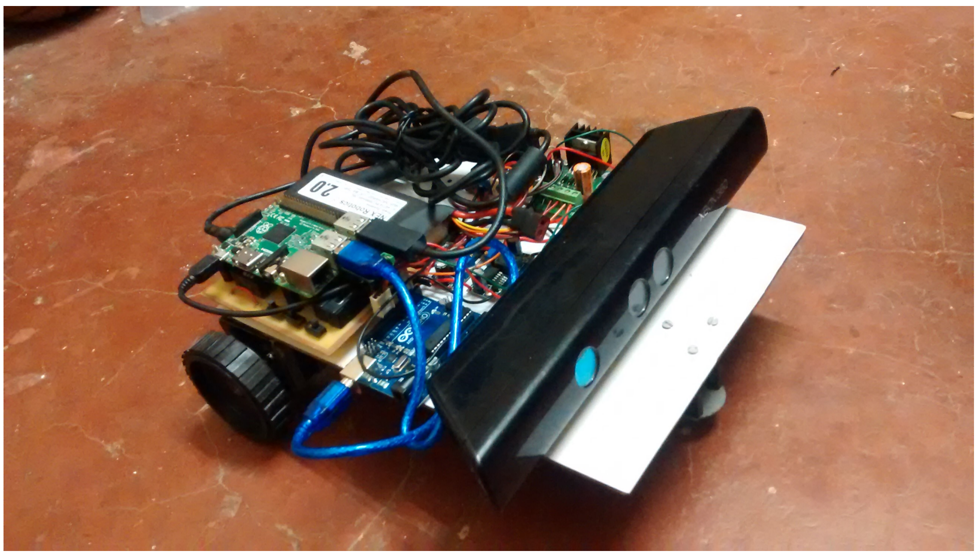

Figure 4.1: The robotic vehicle with kinect and raspberry pi 2 mounted

This section explains the implementation details of the Visual SLAM algorithm to make a robot that can generate 3D – Maps of a small environment like that of a lab. A Kinect camera is mounted in the front part of the robot. A raspberry pi 2 with 1GHz quad core processor is used to process the Kinect data and store it onboard. The bot has 3000mAh battery, 2 60rpm motors, motor driver circuit, power management circuit and other circuitry for its functioning.

4.1 Hardware description

The vehicle was assembled by hand to keep the costs limited.Several considerations has to be kept in mind while building such a vehicle so that minimum problems arise in the later stage.

4.1.1 Mechanical Assembly

The vehicle's kinect camera and motors make its weight to about 2kg.This determines the type of motors and base plate to use. The motors should have sufficient torque to be able to run the vehicle. The baseplate should be strong enough to support 2 kg. A caster wheel is used at the front to simplify design and keep the vehicl's weight less. Several mounting holes were drilled on the baseplate to mount the vaious components over it. they have to be precisely aligned with the mounted components.

4.1.2 Motors

The torque required to run the 2Kg vehicle with maximum speed of about 20cm/sec and acceleration of about 10cm/sec was calculated. It came out to be about 1.2kgcm for each of the two motors. The required rpm for the motors is about 70. Thus, motors were selected based on this information. At first, "100 RPM Side Shaft 37mm Diameter Compact DC Gear Motor" were used. The plan was to use PWM to control their speed and thus obtain a differential drive. However, DC motors availabe in the market cannot be accurately speed controled because their output torque for the same value of input varies every time. This, resulted in sometimes the vehicle turning excessively and sometimes not turning. To solve this problem, stepper motors were then chosen. I used "Moons hybrid bipolar 0.9° 2kgcm 8Ω 0.33A" motors to eliminate any problems regarding vehicle's maneuvering.

4.1.3 Motor Driver circuit

For running DC motors, LM298 IC was used. It is a dual H-bridge IC that inputs signals about motor direction and enable from the microcontroller and outputs motor current to the motors.To control DC motors' speed, a PWM is applied to the enable pin of the H-bridge. This turns the motor on and off repeatedly to get the required speed.

Driving stepper motors is a little complex than driving DC motors. In bipolar stepper motors, the current flowing through the phases has to be reversed to move the motor a step. The current direction in both the phases of the motor should follow a certain sequence to move the motor. This requires a microcontroller to decide the timings of the current direction. The timing of these current reversals decide the speed of the motor. An arduino is used in a vehicle. This arduino sends signal to the H-bridges about the motor current and direction and the H-bridge

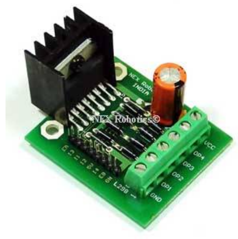

Figure 4.2: L298 motor driver circuit

Figure 4.3: A4988 motor driver circuit for stepper motor

IC reverses the current in the motors accordingly.

However, motor drivers with only H-bridges are not sufficient for a robust system and drive the motors safely. There is no overcurrent protection for motors, no active current controlling. H-bridges cannot control the current decays rates that determine the maximum speed of the stepper motors. Moreover, l298 IC is susceptible to frequency jerks in the input supply and gets damaged easily. If using l298 with an AC convertor, then a 50V capacitor with minimum 100uF capacitance is needed at the input side.

Another driver based on a new IC A4988 was used to drive the stepper motor. It is smallerand has current rating of 1.5A per phase. Also, it has active current controlling which can be used to increase the motor speed by being able to driving it at voltages higher than rated voltages.It has an inbuild controller and therefore, needonly the direction and step inputs from the main microcontroller.

4.1.4 Power circuit board

The power circuit board is custom developed form the institute's PCB lab. This board supplies power to all the systems in the vehicle. It takes in input from either the onboard Li-Polymer battery or from the AC adapter and converts it into 12V for powering Kinect camera and motors, 5V for powering raspberry pi. It also has fuses in all the outputs to prevent anything form being damaged. However, I later realized that 7805 IC used to convert 12V into 5V is not properly regulated and when it is used to power the raspberry pi, the raspberry pi restarts at frequent intervals because of input power fluctuations. Using decoupling capacitors at the output didnt help. To solve this problem, a buck convertor was used that a proper regulated 5V supply.

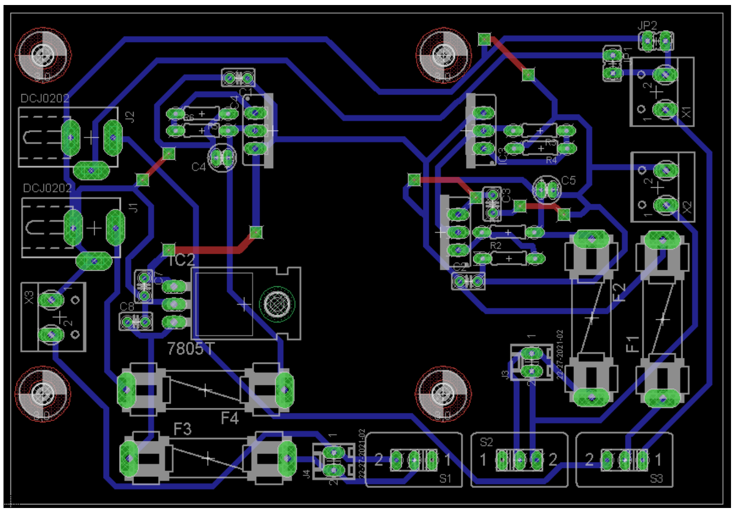

Figure 4.4: The board layout of power circuit board

4.1.5 Kinect camera

The kinect xbox 360 camera was used. The camera was opened and its base was removed to fit it properly on the base plate. Its connector was cut into two and a DC plug was connected on either side to make it possible to pwoer the kinect from 12V supply.

4.2 Algorithm description

The Raspberry Pi 2 is the main processor of the vehicle. The arduino is used to give PWM to the stepper motors and speed control. The challenge run camera driver, visual odometry code, map planner and motor velocity code all in parallel on the limited hardware of Raspberry pi 2.It was also necessary to interface kinect on raspberry pi with workable frame rate.

4.2.1 Interfacing kinect with raspberry pi

The recommended operating system for raspberry pi is raspbian. However, raspbian is a stripped down version of debian linux and does not support a number of packages found in ubuntu. Interfacing kinect needed those libraries to be installed on the system. Source install of such libraries was tried but raspberry pi was taking considerable amout of time for source installs. Sometimes even extending to a day (ROS installation from source). Thus, ubuntu was installed on it for easier accesibilty. Ubuntu support for raspberry pi 2 has come in recently becuse of the arm-hf hardware of the newer 2nd version of the SBC released in 2015. To interface kinect there are mainly two libraries namely Openni and Freenect available. Another driver librekinect is also availabe but it reds only depth stream of the kinect. Both Openni and libfreenect were tried. however, because of non-standard graphics in raspberry pi, these libraries were not able to display depth and rgb data using their native graphics GUI. The output of these can only be accessed using a third-party software like ROS,etc.

4.2.2 ROS - Robot operating System

ROS framework is used in this vehicle to easily communicate among all the parallel threads running in the system and smoothly interface with external hardware like kinect, motors, etc. It provides libraries and tools to help create software for robotic applications. This framework has the following elements –

1. *Roscore* - The master process that starts the ROS and is used as a medium for communication between various other processes that are created after crating this process.
2. *Nodes* - These are single threaded processes like the process that run when an executable file is run. Several nodes can be run in a single ROS framework. These nodes communicate each other via roscore.
3. *Messages* - - Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields such as float, int, or more complex data structure like images, transformations,etc.
4. *Topics* - Topics are like buses over which nodes exchange messages. Topics are published/-subscribed by nodes to get messages. They can be subscribed by any number of nodes. Topics is like broadcasting messages in the framework.
5. *Packages* - A package is similar to a software in windows. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.
6. *Transformations* - In ROS, transformations are maintained in a tf tree. They are used to link together various information in different co-ordinate system.

4.2.3 Parallel threads running in the system

The kinect driver, visual odometry code, path planner, arduino communication thread and transformation code are all running in parallel in ROS framework. They are communicating with each other using "topics" to create a pipeline that inputs kinect data and outputs motor velocity commands.

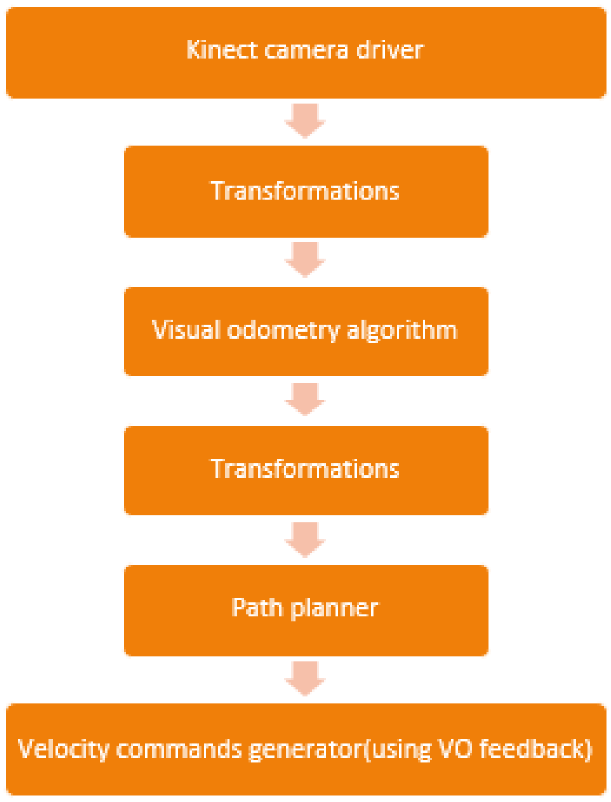
Kinect camera driver
Transformations
Visual odometry algorithm
Transformations
Path planner
Velocity commands generator(using VO feedback)

Figure 4.5: The processes pipeline overview of the code running in raspberry pi

Below is the list of significant nodes running in parallel in the system –

1. Openni Kinect camera nodes -
    - /camera/driver – this node takes camera data coming from usb as input and outputs the raw rgb image and depth image.
    - /camera/depth_points – This node takes camera depth image as input and outputs 3D point cloud of the scene.
    - /camera/depth_rectify_depth – this nodes recifies the depth image

- /camera/debayer – this node splits the incoming raw image into rgb image and mono image.

2. Visual Odometry nodes -

    - /featureTracking – subscribes to incoming rgb image from Kinect and detects features in it.
    - /processDepthMap – subscribes to depth image from the Kinect camera, processes it to generate point cloud and outputs it for visual odometry node.
    - /visualOdometry – this node takes in features tracked from featureTracking node and point cloud from depthMap node and finds the camera pose.

3. Path planner-

    - /custom_planner – this node decides the path where the robot will go i.e. gives the co-ordinates of the destination

4. Communication with arduino-

    - /listener – this node gives arduino velocity commands for left and right motor.

5. Transformations-

    - /camera_base_link – these are nodes that calculate transform between various co-ordinate frames.

Below is the node graph and the corresponding topics subscribed/published by it

Above is the visualization of the odometry transformation calculated by the algorithm –

4.3 Results

The vehicle is supposed to follow the path given by the path planner thread in raspbery pi. A PID algorithm is implemented in "velocity generation thread" to make the robot follow the trajectory lines by taking feedback from Visual odometry data. However, the DC motors used in the vehicle do not generate same torque for same values of the input everytime. This results in faulty maneuvering by the the vehicle and the robot is not able to follow a path that has turns. If the path is straight, the vehicle successfully follows it and stops once it reaches its

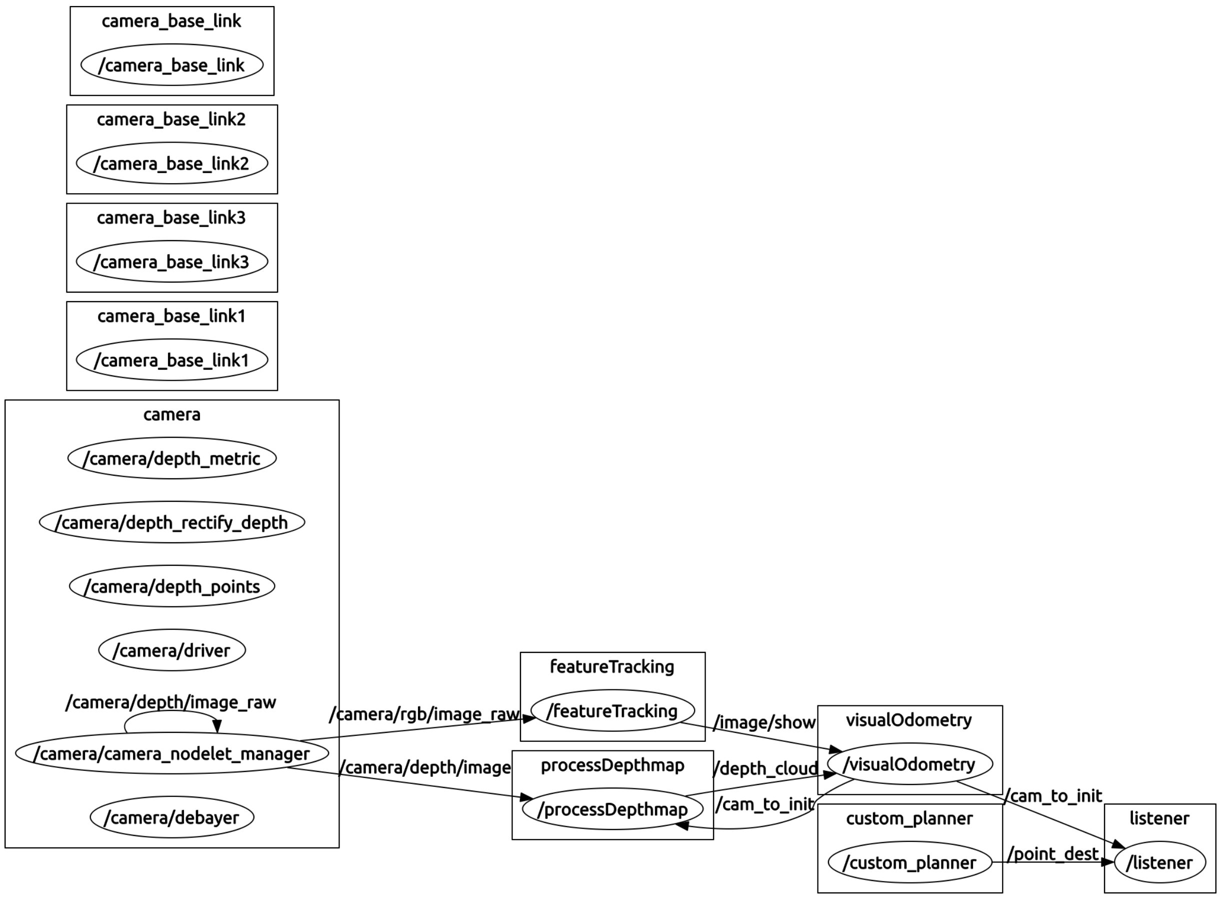

Figure 4.6: Nodes graph of the bot

destination. The visual odometry code is running properly i.e. it updates the new values of the vehicle's position accurately at 1.5Hz rate as it moves around. Therefore, I am able to track the vehicle's movements but cannot make the vehicle go to a specified location if it has to turn. To solve this problem, I removed DC motors and placed stepper motors in the vehicle. Their speed can be precisely controlled. But bipolar stepper motors require proper sequencing of current direction in its two phases to accurately run them. The driver circuit i am using currently is unable to provide enough current to the motors so that it can move the 2kg Vehicle.

To summarize, the vehicle is accurately able to localize itself as it moves around. When DC motors were used, it was able to follow straight paths and was able to automatically stop once the destination is reached. It was not able to follow path that has turns because of faulty motor outputs.

For visual odometry algorithm, update rates of average 1.5Hz were achieved during robot navigation when all the processes including Kinect driver, visual odomtery algorithm, path planner, transformations, etc. were running..Update rate of about 1.5Hz limits the robot speed

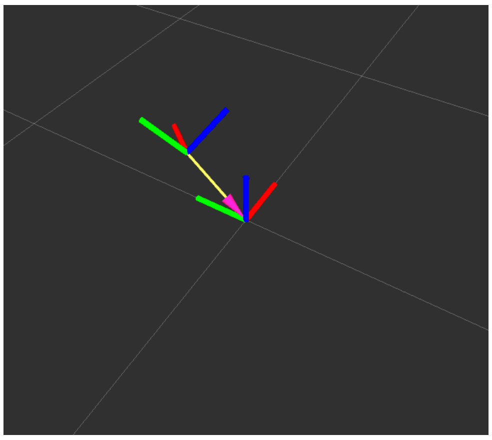

Figure 4.7: odometry

to about 10cm/s. This rate is enough for small scale indoor robot navigation. For larger environment, more powerful SBC instead of the Raspberry pi 2 can be used to eliminate this limitation.

-

Chapter 5

Conclusion and Summary

The problem of Visual Odometry was explored in depth and experimented upon. The algorithm was tested on high performance computers as well as low performance arm-based SBC(Single-board computer). Using this algorithm for robot navigation with Kinect in real time with arm-based SBC and no external help from server or human was implemented. Update rates of average 1.5Hz were achieved during robot navigation when all the processes including Kinect driver, visual odomtery algorithm, path planner, transformations, etc. were running.

However, update rate of about 1.5Hz limits the robot speed to about 10cm/s which is very less. More powerful SBC instead of the Raspberry pi 2 can be used like Odroid U3 which is 1.7Ghz qual core embedded SBC.

Bibliography

[1] Wiki Lidar. `https://en.wikipedia.org/wiki/Lidar/`. [Online; accessed 2014].

[2] Mathieu Labbe and François Michaud. Online global loop closure detection for large-scale multi-session graph-based slam. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2661–2666. IEEE, 2014.

[3] Mathieu Labbe and Francois Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *Robotics, IEEE Transactions on*, 29(3):734–745, 2013.

[4] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2013.

[5] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[6] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, September 2014.

[7] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 15–22. IEEE, 2014.

[8] Albert S Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *International Symposium on Robotics Research (ISRR)*, pages 1–16, 2011.