Vision aided Inertial Localization for Remotely Operated Vehicle (ROV)

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Chandra Prakash Mangipudi

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

Dr. Perry Y. Li, Advisor

June, 2016

Acknowledgements

I express my deepest regards to the committee chair Prof. Perry Li, who has been the sole support and guide in my Masters research. Without his guidance, this thesis would not have been possible.

I also acknowledge the guidance and support of the committee members Prof. Rajesh Rajamani and Prof. Stergios Roumeliotis, who have encouraged me to think out of the box and provided me concrete feedback to improve various aspects of my research.

In addition, I would like to thank my undergraduate mentor, Prof. Chandramouli Padmanabhan of IIT-Madras, who introduced me into Acoustics, Vibrations and Control systems.

Finally, I would like to thank Pieter Gagnon (M.S.) who designed and prototyped the structure of the robot and Qianrong Huang (M.S.) who built the controller for the underwater robot. My voyage towards outstanding academic research has been successful largely due to the motivation and support from my peers Mohsen Saadat and Sangyoon Lee.

Dedication

My perseverance in completing this work is solely due to the constant endurance and perennial support provided by my family and friends. I dedicate this entire thesis to my

MOTHER

&

FATHER

for being there with me in sharing moments of joy and sorrow. Each and every single achievement till date and henceforth, are your priceless gift to me. The advancements in science and technology would not have been possible without the guidance and support of my

TEACHERS

Abstract

This thesis details the implementation of a vision aided inertial localization system for a Remotely Operated Vehicle (ROV) in a controlled environment. The vision sub-system features a Raspberry-Pi equipped with the pi-camera and the inertial sub-system consists of an automotive/consumer grade MEMS-IMU operated by an Arduino board (Arduino Due). A novel PnP estimation algorithm, Linear Least Squares - Gradient SO(3) algorithm ($\boldsymbol{SO(3) - PnP}$), is introduced for pose estimation using the vision sub-system. The position is estimated using a linear least squares approach while the orientation is computed iteratively using a gradient descent algorithm. The inertial sensors are used as a dead-reckoning system in between the vision measurements. The cross-talk between the vision and inertial sub-systems is established using ethernet(UDP). Sensor fusion is achieved using Kalman filters. Laboratory experiments validate the accuracy of the Vision-IMU module and enhance the possibility that they can be deployed in an un-controlled ocean environment to compute the pose of the ROV (within a target workspace) w.r.t. a given landmark.

Contents

List of Tables

List of Figures

Listings

List of Symbols

δt	Time interval between successive sensor readings
${}^{G}_{C}R$	Rotation matrix representing transformation in orientation from Camera Frame to Global Frame
${}^{G}P_{C}$	Translation vector representing the origin of $\{C\}$ in $\{G\}$
${}^{C}p_{i}$	Object Point i in Camera Frame of Reference
${}^{G}P_{i}$	Object Point i in Global Frame of Reference
$\vec{\boldsymbol{b}}$	Gyroscope bias vector in the IMU Frame $\{I\}$
$\vec{\boldsymbol{g}}$	Gravity Vector
K_{acc}	Scale Factor of Accelerometer
$\vec{\boldsymbol{b}}_{acc}$	Bias of Acceleormeter
$\vec{\boldsymbol{w}}_{acc}$	Signal noise of Accelerometer
g	Magnitude of gravity vector
J	Cost Function to be optimized
A_{picam}	Intrinsic Camera Matrix
f_x, f_y	Focal length of Camera in x-direction and y-direction
c_x, c_y	Center Point of Image in Pixel co-ordinates
W_{img}, H_{img}	Dimensions of image in pixels

k_1, k_2	Radial Distortion co-efficients of Camera
p_1, p_2	Tangential Distortion co-efficients of Camera
U, S, V	Left Eigen-Vectors, Diagonal Matrix and Right Eigen Vectors obtained using SVD analysis
μ_{ji}	Moments of Image of order i about x-axis and order j about y-axis
x	Variable
$\hat{x}$	Estimate of variable x
$\hat{x}^*$	Formulated variable x in terms of other variables and constants
∂x	Partial derivative of variable x
$\bar{x}$	Mean of variable x
N^+	Symmetric matrix formed from matrix N
N^X	Skew-Symmetric matrix formed from matrix N
$\vec{\boldsymbol{\omega}}$	Angular Velocity vector
$[\vec{\boldsymbol{\omega}}]_\times$	Skew symmetric matrix of vector $\vec{\boldsymbol{\omega}}$
Ω	Quaternion Rotation Matrix due to angular velocity $\vec{\boldsymbol{\omega}}$
ϕ	Roll Angle
θ	Pitch Angle
ψ	Yaw Angle
$H(z)$	Discrete Transfer Function
q	Qauternion representing orientation of $\{C\}$ w.r.t. $\{G\}$
$\otimes$	Quaternion Multiplication
$R(q)$	Rotation matrix corresponding to Qauternion q

t_{cap}	Time at which image was captured
t_{proc}	Time at which image was processed
$\mathcal{Z}$	Transformation from time domain to discrete frequency domain
$\mathcal{Z}^{-1}$	Inverse transformation from discrete frequency domain to time domain
K_{rot}, K_{trans}	Kalman Gain
P_{rot}, P_{trans}	State Co-variance Matrix
Q_{rot}, Q_{trans}	Process Error Co-variance Matrix
R_{rot}, R_{trans}	Measurement Error Co-variance Matrix
$\vec{\boldsymbol{u}}$	Input Vector
$\vec{\boldsymbol{x}}$	State Vector
$\vec{\boldsymbol{y}}$	Measurement Vector
$I = I_{n \times n}$	Identity Matrix of order n
$\mathbf{0} = \mathbf{0}_{m \times n}$	Zero Matrix of appropriate dimensions
f_a	Focal Length of camera in air
f_w	Focal Length of camera in water
n_a	Index of refraction of air
n_w	Index of refraction of water
n_l	Index of refraction of camera lens
R_1	Radius of curvature 1 for spherical lens
R_2	Radius of curvature 2 for spherical lens

Chapter 1

Introduction

Underwater robotics is a relatively young field of research and involves a number of challenges. The underwater robots are primarily classified as Autonomous Underwater Vehicle (AUV) or Remotely Operated Vehicle (ROV). The former is an independent robot (typically used for research and small scale models) while the latter is operated with the help of a mother ship for industrial applications.

The primarily disadvantage in aquatic environments is the lack of Global Positioning System (GPS) whick makes navigation more challenging than terrestrial or aerial navigation. Underwater robots are used to perform a variety of tasks ranging from simple tasks like inspection/monitoring to complicated tasks like underwater mining/repairs. The pose estimation i.e. the estimation of the position and orientation of the robot is crucial in almost every application to ensure that the robot is stabilized during manipulation/control. The current project introduces a novel yet simple algorithm utilizing camera and inertial sensors to identify the pose of the ROV in a given workspace region.

The current chapter provides a detailed insight about the problem statement. This chapter is organized as follows. Section 1.1 presents an overview of the rest of the chapters. The sensors and techniques used to solve underwater navigation are presented in Section 1.2. A detailed description of a few state of the art pose estimation algorithms is summarized in Section 1.3. Section 1.4 states the requirements of the novel sensor for the ROV. The selection of sensors to solve this problem statement is detailed in Section 1.5. A theoretical overview of the problem solution using the selected sensors is elaborated in Section 1.6. Section 1.7 concludes the introduction chapter.

1.1 Overview

The thesis is divided into 9 Chapters and a brief introduction to each of the chapters is listed below.

- *Chapter 1* Background researh on existing state of the art technology and the analytical goals pursued in this thesis
- *Chapter 2* Controllers and sensors used to build the sensor prototype
- *Chapter 3* Sensor models and calibration routines for the sensor components
- *Chapter 4* Landmark feature extraction using image processing on GPU and object detection on CPU
- *Chapter 5* Pose (Position and Orientation) estimation using landmark features.
- *Chapter 6* LTI System model and pose estimation using IMU
- *Chapter 7* Combining the estimate from IMU and Pi-camera using a Kalman Filter
- *Chapter 8* Performance of the sensor prototype in Underwater Environment
- *Chapter 9* Summarizes the key results and provides insights into further research.

1.2 Background

The underwater enviornment poses several challenges for autonomous navigation and pose estimation. A number of innovative solutions utilizing a variety of sensors have been presented to determine the pose of the ROV and the solutions have had slow and modular changes over the last decade. The goal of this project is to estimate the pose precisely so as to enable the ROV to perform tasks in underwater environment like grabbing and placing objects. A review of the prior research is discussed in detail in this section.

1.2.1 Passive Arm

A standard robot is assumed to have two manipulators to mimic a human being. The idea of a passive arm [2] is to use one of the manipulators to estimate the pose of the ROV while the other can be used to complete the designed tasks in underwater environment. The passive arm has encoders attached to monitor each degree of freedom (DOF) of the passive arm,so that by attaching the end-effector of the passive to a stationary object within the workspace, the pose of the ROV can be estimated. A schematic of the passive arm technique is shown in Fig. 1.1. The manipulator needs to be pretty long and designed to be able to maneuver all poses in the workspace. The manufacturing of manipulators is tedious as the manipulators are required to be strong and waterproof.

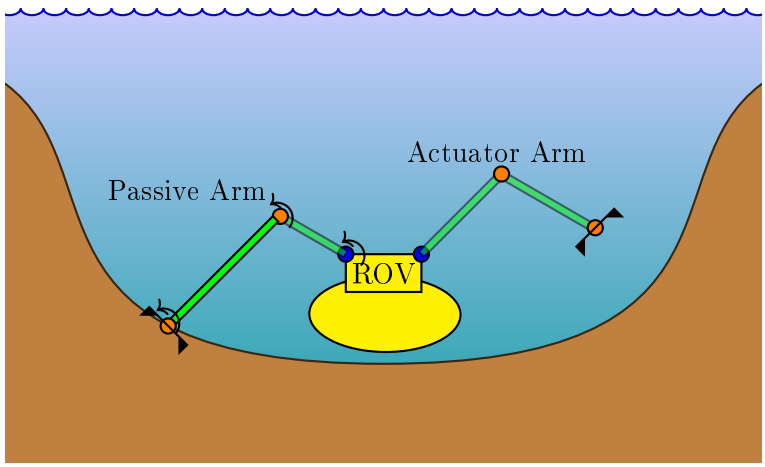


Figure 1.1: Schematic of Passive Arm (Adapted from [2])

This method allows only one of the manipulators to be operational which is not efficient and user-friendly. All the above mentioned disadvantages defeat the passive arm concept for the current ROV. The primary objective of the project is to replace this method of using a passive robotic arm to estimate the pose of the ROV with a non-contact sensor that is equally robust and accurate. Hence, we propose a vision based pose estimation.

1.2.2 Underwater Sensor Networks

Underwater Sensor Networks (UWSN) [3] have had a huge success to estimate the position of the ROV. They are similar in architecture to the Global Positioning System (GPS) but differ only in the fact that the pressure Nodes send acoustic signals rather than radio signals as in the GPS. The trivial method utilizes the principle of Range-differencing [4], which in the event of 3 acoustic source nodes reduces to tri-angluation or tri-lateration.

A schematic representing the sensor networks is shown in Fig. 1.2.

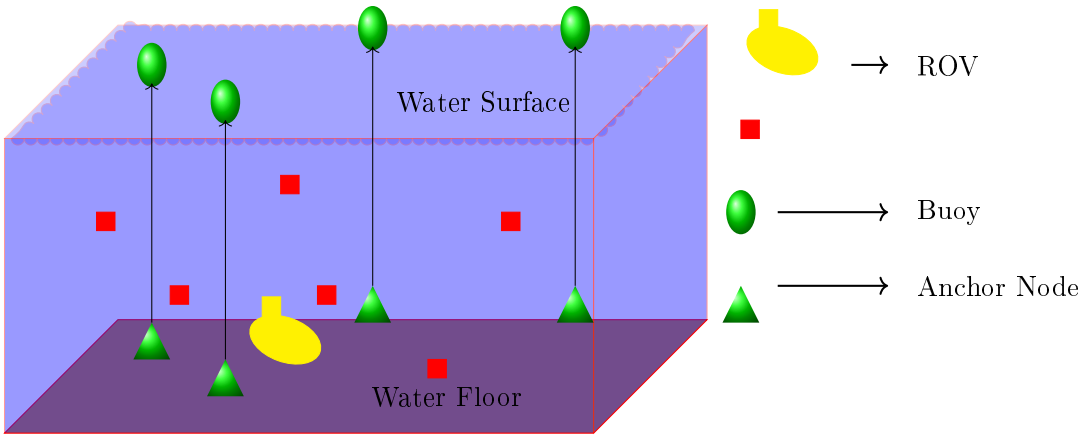


Figure 1.2: Schematic of Underwater Sensor Network [3]

Large-scale UWSN [5] employ a grid of sensors mounted in a fixed pattern (typically rectangular). Anchor nodes have access to GPS information with the help of surface buoys and normal nodes communicate with the anchor nodes to estimate their position using a variety of algorithms[6].

The anchor nodes and the normal nodes are acoustic sources which are actuated in a known sequence and a sensor is mounted on the ROV which can receive the acoustic waves. With the knowledge of the position of the nodes and the time of receiving the signals, the position of the ROV can be estimated.

Innovative research in UWSNs have been accomplished to account for improvements in sensor positioning [7],[8], [9] acoustic path selection [10], mobile nodes[11], silent positioning [12] , novel estimation [13] and energy-efficient[14] strategies.

1.2.3 Underwater Computer Vision

The underwater vision is challenging due to improper natural lighting and opacity. In the presence of artificial lighting and at short distances with clear water, computer vision would be very helpful in achieving robust pose estimation.

One of the golden techniques used in achieving underwater robot navigation, where the environment is unknown, is Concurrent Localization and Mapping (CLM) [15] better known as Simultaneous Localization and Mapping (SLAM) [16]. SLAM is based on the principle that using multiple observations the pose of the ROV and the features of the landmark can be simultaneously estimated. However, it requires a clear view of distinct features of the environment and is computationally expensive to implement on small ROVs.

Another popular tool which is used for feature extraction, Scale Invariant Feature Transform (SIFT), utilizes image gradients to extract salient features from an image. It is computationally tedious and is featured in large ROVs [17]. Stereo vision [18] would yield precise pose estimates but would require 2 cameras and the images from the cameras must be taken at the same time stamp .

Laser based image processing and dead reckoning [19] [20] is a new possibility that has been experimented in underwater environments. The feature points of interest are generated by a line laser and the ROV pose is estimated by observing these points using a camera.

The landmark [21] [22] based image processing is our topic of interest. In our project, we currently use four balls with distinct colors positioned in a Global Frame {G} in the form of a tetrahedron to estimate the pose of the ROV. In general, the landmark can have n balls with n distinct colors. In the future, the spherical balls can be replaced by laser dots or light bulbs. The key feature of the current work is based on pose estimation using the gradient descent algorithm to achieve a quick and precise pose estimate.

1.3 Pose Estimation

Perspective n Point (PnP) problem [23] is an exciting research area involving the computation of camera extrinsic matrix (rotation and translation or pose) using the image points of n object points, given the camera intrinsic matrix (focal length and image properties). It has extensive applications in Computer Vision, Augmented Reality and Robotics. A schematic of the PnP problem is shown in Fig. 1.3.

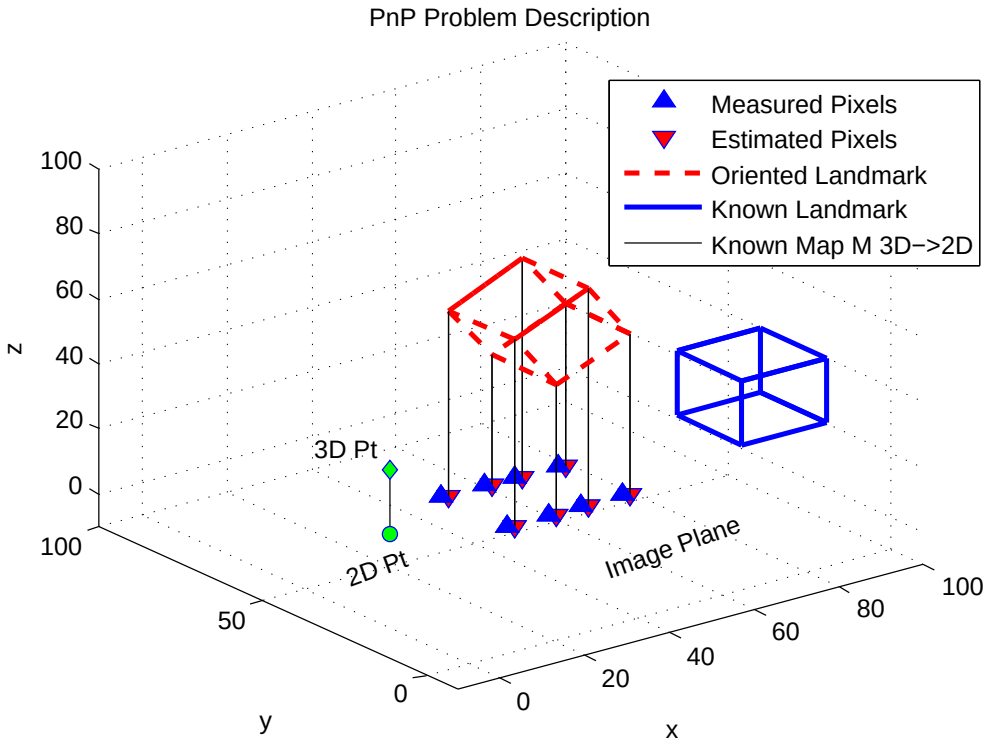


Figure 1.3: PnP Problem Description

A brief description of the classification of a few pose estimation algorithms is presented in table 1.1

Table 1.1: State of the Art PnP Estimation Algorithms (adapted from [1])

method	author	iterative	closed form	n=3	n general	linear	nonlinear
P-PnP	Garro[1]		✓		✓	✓	
parameterized 3D	Lowe[24]	✓			✓		✓
DLS	Hesch[25]		✓		✓		✓
Linear DLT	Fiore[26]		✓		✓	✓	
E-PnP	Lepetit[27]		✓		✓	✓	
direct parameter 3pt	Kneip[28]		✓	✓			✓
R-PnP	Shiqi Li[29]		✓		✓		✓
pose points/lines	Ansar[30]		✓		✓	✓	
LHM	Lu[31]	✓			✓	✓	
perspective 3pt	Gao[32]		✓	✓			✓
POSIT	DeMenthon[33]	✓			✓	✓	

1.3.1 Literature Review

A schematic of the pin hole camera model is shown in Fig. 1.4. The corresponding projection equation is shown in (1.1). The camera parameters like focal length (f_x, f_y), center points (c_x, c_y) as well as the global positions $({}^{G}P_i)$ are given. The pixel coordinates (u_i, v_i) are obtained from camera measurements. The scale factor z_i is the z-component of the unkonwn translation vector $({}^{C}P_G)$. The PnP pose estimation problem reduces to solving the extrinsic camera matrix $({}^{C}_{G}R, {}^{C}P_G)$, which is generally solved in the camera reference frame {C}. A complete derivation of the camera model is presented in Section 5.2.

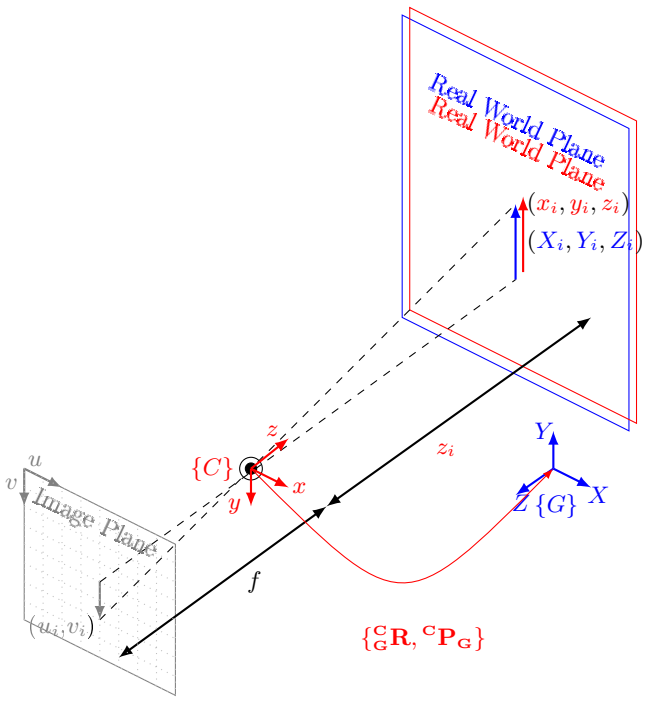


Figure 1.4: Pin Hole Camera Model

$$z_i \underbrace{\begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}}_{Pixels} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \approx \frac{W_{img}}{2} \\ 0 & f_y & c_y \approx \frac{H_{img}}{2} \\ 0 & 0 & 1 \end{bmatrix}}_{IntrinsicMatrix} \underbrace{\begin{bmatrix} {}^{C}_{G}R & | & {}^{C}P_{G} \end{bmatrix}}_{ExtrinsicMatrix} \underbrace{\begin{bmatrix} {}^{G}P_i \\ 1 \end{bmatrix}}_{GlobalPosition} \tag{1.1}$$

The orientation between two co-ordinate systems given 3D-3D correspondences (point clouds) was solved by Horn et. al. [34]. Later, the concept of pose estimation from points and point clouds using linear least squares was discussed by Haralick et. al [35]. We discuss a few of the most popular and accurate PnP pose estimation algorithms.

1.3.2 Iterative Methods

In this sub-section, we will discuss 3 of the most popular iterative algorithms - Pose from Orthogonality and Scale - Iterations (POSIT), Lu-Hager-Mjolsness(LHM) and the Procrustes-PnP (P-PnP).

The POSIT method developed by DeMenthon [33] is an early iterative algorithm developed in 1992. The idea behind the POSIT algorithm is to iterate for the scale z_i (1.1) and the orhtogonal rotation matrix. A mathematical approach is described briefly by the following algorithm.

Algorithm 1: Short Procedure - POSIT algorithm [33]

Steps:

1. Construct the matrix containing the object points $A_{(n-1)\times 3}$ such that $A(i,:) = ({}^{C}P_G)_i - ({}^{C}P_G)_0 \ \forall i \in (0, n-1)$. Aso construct the object matrix B as the pseudo-inverse of matrix A i.e. $B = (A^T A)^{-1} A^T$
2. Initialize the correction co-efficients $\epsilon_i = 0$ and corrected image projections $(u_i^* = u_i, v_i^* = v_i), \forall i \in (0, n-1)$
3. Compute the vectors $\vec{i}, \vec{j}$ and the translation scale Z_0.
   - a Construct the matrix with image points $C_{(n-1)\times 2}$, s.t. $C(i,:) = [u_i^*, v_i^*](1+\epsilon_i) - [u_0^*, v_0^*]$
   - b Construct the matrix $D_{3\times 2}$, s.t. $D = BC$. Call the rows $D(0,:) = I, D(1,:) = J$
   - c Calculate the scale s as $s = \frac{||I||+||J||}{2}$
   - d Compute the unit vectors $\vec{i} = \frac{I}{||I||}, \vec{j} = \frac{J}{||J||}$
4. Compute new ϵ_i
   - a Compute $\vec{k} = \vec{i} \times \vec{j}$
   - b Compute $Z_0 = \frac{f}{s}$
   - c Compute $\epsilon_i = \frac{A(i,:)\cdot k}{Z_0}$
5. If $\epsilon_{k+1} - \epsilon_k > threshold$, repeat steps **2-4**
6. After convergence, obtain the Rotation matrix ${}^{C}_{G}R = \begin{bmatrix} \vec{i}^T & (\frac{\vec{k}}{||\vec{k}||} \times \vec{i})^T & (\frac{\vec{k}}{||\vec{k}||})^T \end{bmatrix}^T$ and the translation vector ${}^{C}P_G = \begin{bmatrix} \frac{u_0^*}{s} & \frac{v_0^*}{s} & \frac{Z_0}{s} \end{bmatrix}^T$

Lu-Hager-Mjolsness (LHM) rearranged the cost function in terms of object space error in 3D Cartesian co-ordintes [31]. For convenience, we represent $R = {}^{C}_{G}R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix}$

The perspective projection measurement equation is described by (1.2).

$$\vec{\boldsymbol{v}}_i = \begin{bmatrix} u_i \\ v_i \end{bmatrix} \frac{R\,{}^{G}P_i + t}{r_3^T\,{}^{G}P_i + t_z} \tag{1.2}$$

The projection matrix V_i is defined by (1.3).

$$V_i = \frac{\vec{\boldsymbol{v}}_i \vec{\boldsymbol{v}}_i^T}{\vec{\boldsymbol{v}}_i^T \vec{\boldsymbol{v}}_i} \tag{1.3}$$

The minimization problem can be written in terms of the standard least squares error equation (1.4)

$$J = ||R\,{}^{G}P_i + t - q_i|| \tag{1.4}$$

The iterations are carried out in SO(3) space using *Orthogonal Iteration*(OI). The optimization problem is expressed mathematically as follows:

$$\textbf{Objective:} \quad \min_{R,{}^{C}P_G} \quad J(R) = \sum_{i=1}^{n} ||(I - \hat{V}_i)R\ {}^{G}P_i + {}^{C}P_G(R)||$$
$$\text{given} \quad {}^{G}P_i\,, \hat{V}_i = \textit{Projection Matrix}, \vec{q}_i = \hat{V}_i(R\,{}^{G}P_i + {}^{C}P_G)$$

A short procedure of the solution is shown in algorithm 2.

Algorithm 2: Summary of LHM Algorithm [31]

Steps:

I Formulate ${}^{C}P_G = \frac{1}{n}(I - \frac{1}{n}\sum_j^n \hat{V}_j)^{-1} \sum_j^n (\hat{V}_j - I)R\ {}^{G}P_i$

II Iterate to find optimal position and rotation

1 Use a guess value of rotation matrix ${}^{C}_{G}\hat{R}^{-}$

2 Compute the covariance matrix $M(R) = \sum_i^n \vec{q}_i^T(R){}^{G}P_i$

3 Update, $\hat{R}^{-}_{k+1} = \ arg\ \max_{R}\ tr(\hat{R}_k^{-T} M \hat{R}_k^{-}), t_{k+1} = t(\hat{R}^{-}_{k+1})$

4 Iterate steps 2–3 till convergence

Procrustes-PnP (P-PnP) [1] formulates the problem as an anisotropic orthogonal procrustes problem [36]. The solution proceeds as follows.

$$z_i L_i = A_{cam} \begin{bmatrix} R \mid t \end{bmatrix} {}^{G}P_i \tag{1.5}$$

Re-writing the above model

$$\underbrace{\begin{bmatrix} z_1 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & z_n \end{bmatrix}}_{Z} \underbrace{\begin{bmatrix} \vec{\boldsymbol{p}}_1^T \\ \vdots \\ \vec{\boldsymbol{p}}_n^T \end{bmatrix}}_{P} R + \underbrace{\begin{bmatrix} \vec{\boldsymbol{c}}^T \\ \vdots \\ \vec{\boldsymbol{c}}^T \end{bmatrix}}_{1c^T} = \underbrace{\begin{bmatrix} \vec{\boldsymbol{M}}_1^T \\ \vdots \\ \vec{\boldsymbol{M}}_n^T \end{bmatrix}}_{S} \tag{1.6}$$

The cost-function to be optimized is given by

$$\textbf{Objective:} \quad \min_{R,1c^T} \quad J = ||\Delta||, \Delta = S - Z\,P\,R - \mathbf{1}c^T$$

$$\text{given} \quad P, S$$

Algorithm 3: Summary of P-PnP Algorithm

Steps:

1. Start with $Z = 0$ or any initial guess of the depth matrix
2. Compute $R = U\ diag(1, 1, det(UV^T))\,V^T$, with $UDV^T = P^T Z(I - \mathbf{1}\mathbf{1}^{\mathbf{T}}/n)S$
3. Compute $c = (S - Z\,P\,R)^T\mathbf{1}/n$
4. Compute $Z = diag(P\,R(S^T - c\,\mathbf{1}^T))diag(P\,P^T)^{-1}$
5. Iterate steps 2-4 till convergence

This method is similar to the POSIT as it involves computing the scale co-efficients, the translation vector and the Rotation matrix. However, the repeated SVD computation is quite expensive and computationally inefficient.

1.3.3 Closed Form Solutions

In this subsection, we discuss 3 of the recent and most efficient closed form solutions - Direct Least Squares (DLS), Efficient-PnP (E-PnP) and Robust-PnP (R-PnP) .

DLS [25] computes the cost function using the normalized unit vectors ${}^{C}\bar{p}_i$ of the object points in Camera co-ordinate system ${}^{C}P_i$ as shown below in (1.9). The camera model y_i with measurement noise ϵ is shown in (1.7). A measurement obtained from the image is represented by $\bar{y_i}$.

$$y_i = {}^{C}\bar{p}_i + \epsilon \tag{1.7}$$

$$z_i {}^{C}\bar{p}_i = ||R\,{}^{G}P_i + {}^{C}P_G|| \tag{1.8}$$

$$J = \sum_{i=1}^{n} ||\bar{y}_i - \frac{1}{z_i}(R\,{}^{G}P_i + {}^{C}P_G)||^2 \tag{1.9}$$

The measurement (1.8) is re-arranged as showin in (1.10)

$$\underbrace{\begin{bmatrix} {}^{C}\bar{p}_1 & & -I \\ & \ddots & \vdots \\ & {}^{C}\bar{p}_n & -I \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_n \\ {}^{C}P_G \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} R & & \\ & \ddots & \\ & & R \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} {}^{G}P_1 \\ \vdots \\ {}^{G}P_n \end{bmatrix}}_{b} \tag{1.10}$$

The least squares solution is used to estimate the scale factors α_i and the position vector ${}^{C}P_G$, which leads to a sparse matrix $W(R)$ to be solved. The Rotation matrix is represented using the Cayley-Gibbs-Rodriguez (CGR) parameters.

The resulting cost function is a quartic polynomial in terms of the rotation matrix parameters s_1, s_2, s_3. This leads to a polynomial function comprising of 3 quartic equations. Each of the stationary points($3 \times 3 \times 3 = 27$) are examined and the pose is estimated from the global minimum. It is currently one of the best pose estimation algorthms.

E-PnP [27] reduces the problem by choosing 4 optimal object points referred to as control points and then estimates the pose from these 4 points using a closed form linear constrained solution. The equation to solve are presented below

$$\forall i = 1:4,\ z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_c \\ 0 & f_y & v_c \\ 0 & 0 & 1 \end{bmatrix} \left(\underbrace{\left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & {}^{C}X_G \\ r_{21} & r_{22} & r_{23} & {}^{C}Y_G \\ r_{31} & r_{32} & r_{33} & {}^{C}Z_G \end{array}\right]}_{\sum_{j=1}^{4} \alpha_{ij}} \begin{bmatrix} X_i^c \\ Y_i^c \\ Z_i^c \\ 1 \end{bmatrix} \right) \tag{1.11}$$

Re-arranging the above equations, we get

$$\sum_{j=1}^{4} \alpha_{ij} f_x X_j^c + \alpha_{ij}(u_c - u_i) Z_j^C = 0 \tag{1.12}$$

$$\sum_{j=1}^{4} \alpha_{ij} f_y Y_j^c + \alpha_{ij}(v_c - v_i) Z_j^C = 0 \tag{1.13}$$

The above system of linear equations can be re-written in terms of the variable $\vec{\boldsymbol{x}} = \begin{bmatrix} c_1^T & c_2^T & c_3^T & c_4^T \end{bmatrix}^T$ as shown in (1.14)

$$M\vec{\boldsymbol{x}} = 0 \tag{1.14}$$

The null space of the matrix M provides the solution to the linear system of equations and is solved using Singular Value Decomposition (SVD).

The E-PnP is widely used and is very accurate if the outliers are minimal (as it computes control points based on all the feature points). The E-PnP performance is affected to a larger degree by noise.

R-PnP [29] uses the property that the distance between any two points in global co-ordinates remains the same at any viewing angle of the camera. For example, given any 3 points,

$$r_i^2 + r_j^2 - 2r_i r_j cos(\theta_{ij}) - d_{ij}^2 = 0 \tag{1.15}$$

$$r_i^2 + r_k^2 - 2r_i r_k cos(\theta_{ik}) - d_{ik}^2 = 0 \tag{1.16}$$

$$r_j^2 + r_k^2 - 2r_j r_k cos(\theta_{jk}) - d_j^2 = 0 \tag{1.17}$$

The system of equations can be cast into a 4^{th} order polynomial as shown below:

$$f(r) = ar^4 + br^3 + cr^2 + dr + e = 0 \tag{1.18}$$

For n-points, the largest residual of the polynomial is used to select the axis z_a, which gives the arbitrary rotation matrix R'.

$$z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & \frac{W_{img}}{2} \\ 0 & f_y & \frac{H_{img}}{2} \\ 0 & 0 & 1 \end{bmatrix} \left(\underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{R'} \begin{bmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + \begin{bmatrix} {}^C X_G \\ {}^C Y_G \\ {}^C Z_G \end{bmatrix} \right) \tag{1.19}$$

The above equation can be re-arranged into linear equations in terms of the state variable $\vec{\boldsymbol{x}} = \begin{bmatrix} c & s & {}^C X_G & {}^C Y_G & {}^C Z_G & 1 \end{bmatrix}^T$,

$$\begin{bmatrix} A_{2n\times 1} & B_{2n\times 1} & C_{2n\times 4} \end{bmatrix} \vec{\boldsymbol{x}} = 0 \tag{1.20}$$

The variables for this over-determined system of linear equations is solved using SVD.

1.3.4 SO(3) – PnP

The proposed algorithm in this thesis, Linear Least Squares - Gradient SO(3) (*SO(3) – PnP*) splits the perspective projection equations into a linear part for the position estimation (least squares formulation) and a non-linear part for the rotation matrix. The rotation matrix is iteratively estimated by optimally rotating in the *SO(3)* space using the gradient descent approach. The algorithm is fine-tuned to converge in 10-15 iterations. The complete algorithm can be studied in detail in Chapter 5

1.4 Problem Statement

The current problem statement is to create a compact navigation system which will estimate the pose of the underwater robot in a given workspace. Some of the key and essential features of the navigation system are listed below.

1. The navigation system [37] should have a range accuracy better than 0.5cm. in a 2m. sphere (1% accuracy) and an orientation estimate better than 0.5°(Euler Angles).

2. The sensor module has to be compact (not exceeding $10cm^3$), light-weight and robust.

3. It should have a high update frequency (200Hz) and should be reliable.

1.5 Research Design

In comparison to the broad range of available sensors, we chose to utilize a camera-IMU system (using a fixed known Landmark [38]). A comparison of sensors for this project based on optimizing all the required constraints suggested that a camera-IMU system would deliver the required performance. The camera system is the PiCam to be operated with a Raspberry Pi, a low-cost single board computer and the Inertial Measurement Unit (IMU) system comprises of a consumer grade ST Micro-Electro Mechanical Systems (MEMS) IMU to be operated with an Arduino Due. A picture of the underwater prototype is shown in Fig. 1.5 .

(a) Final Prototype - Front view **(b)** Final Prototype - Rear view

Figure 1.5: Sensor Prototype

1.6 Theoretical Framework

The vision algorithm used in this project is based on the Special Orthogonal Group (SO(3)) of rotation matrices to iteratively estimate the orientation of the robot using the camera. A least squares solution is employed to estimate the position of the robot using computer vision. The landmark should comprise of a minimum of 4 spheres (differentiated by color) in 3D (non-planar) to ensure robustness. The camera measurements are very accurate, however they are limited to a frequency of $10Hz$ due to processing speed. The IMU sensor data with an update frequency of $200Hz$ is used to smooth the pose estimate between the camera measurements.

The accelerometer readings are high-pass filtered to remove gravity and double integrated to get the position estimate from the IMU. A Kalman filter is designed for sensor fusion. A flowchart of the entire navigation module (real time implementation) is shown in Fig. 1.6

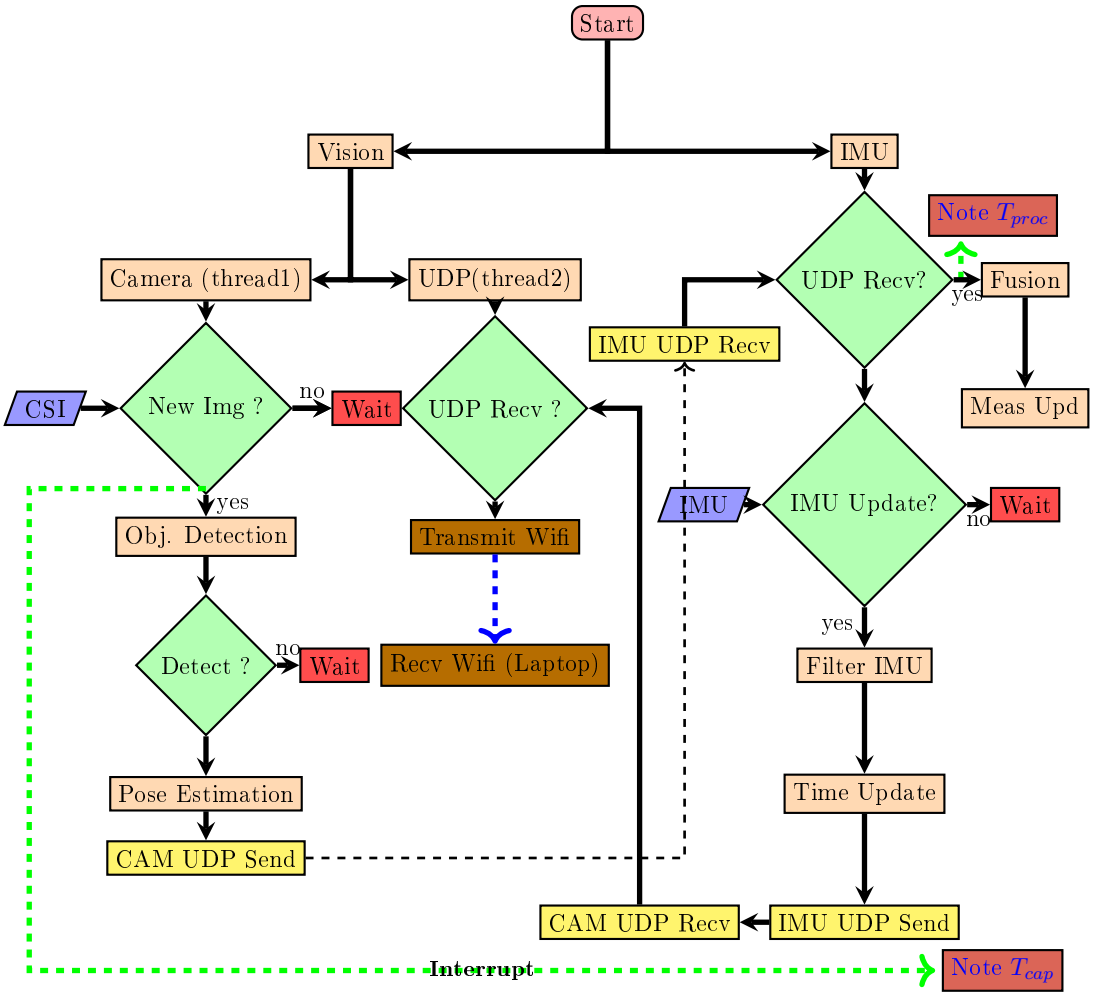


Figure 1.6: Flowchart - VINS

1.7 Summary

This chapter describes the bacground of the problem statement and the strategy used to achieve the pose estimation in a controlled underwater environment. A brief introduction is given about a few state of the art algorithms. It also provides a foundation to the entire thesis, summarizes the prior research, formulates the problem statement and gives an overview of the solution methodology.

Chapter 2

Hardware

This chapter introduces the hardware and the related software used in this project. The sensor (IMU and Pi-camera) and their communication interfaces are described in detail. The different co-ordinate frames of reference used in this thesis and their co-relations are elaborated.

2.1 Overview

The sensors used in this project are a MEMS based pololu IMU (LSM303-DLHC - accelerometer and magnetometer unitaccelerometer and magnetometer unit, L3G - gyroscope) and a HD Picam camera. Section 2.2 describes in detail the IMU characteristics while Section 2.3 details the Pi-Camera. The embedded systems used for processing the sensor data are an Arduino Due for the IMU and a Raspberry-pi for the Picam as seen in Section 2.4. The different co-ordinate frames of reference and their inter-relation are elaborated in Section 2.5.

2.2 IMU

The IMU data from the 3-axis accelerometer, magnetometer and gyroscope is obtained using the I2C protocol. The IMU data is processed in the Arduino Due. The sampling time for the IMU is $\boldsymbol{\delta t = 5ms}$, which is a very short time interval. A schematic of the IMU is shown in Fig. 2.1a. The sensor charactersitics are listed in table 2.1,2.2

LSM303DLHC Acc/Mag Characteristics	**Value**
Linear Acc. Sensitivity change (T)	$\pm 0.01\%/°C$
Linear acceleration typical Zero-g level offset accuracy	$\pm 60mg$
Linear acceleration Zero-g level change vs. temperature	$\pm 0.5mg/°C$
Acceleration noise density	$220\mu g/\sqrt{Hz}$
Magnetic resolution	$2mgauss$
Magnetic cross-axis sensitivity	$\pm 1\%FS/gauss$
Maximum exposed magnetic field	$10000gauss$
Disturbing magnetic field	$2000gauss$
Operating temperature range	$-40\ to\ 85°C$
Linear acceleration measurement range*	$\pm 2g$
Linear acceleration sensitivity*	$1mg/LSB$
SCL clock frequency*	$400kHz$
Linear Accelerometer Data Update Rate*	$400kHz$
Magnetometer Measurement Range*	$\pm 1.3gauss$
Magnetic gain setting*	$\pm 1100LSB/gauss$

* user choice

Table 2.1: Accelerometer Specifications - LSM303DLHC

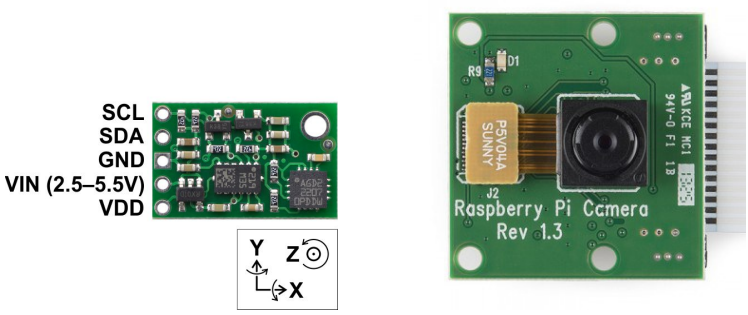
L3GD20 Gyro Characteristics	**Value**
Gyro Sensitivity change (T)	$\pm 2\%/°C$
Rate noise density	$0.03dps/\sqrt{Hz}$
Zero-rate level change vs. temperature	$\pm .03dps/°C$
Gyro measurement range*	$\pm 250dps$
Gyro sensitivity*	$8.75mdps/digit$
Digital zero-rate level*	$\pm 10dps$
Digital output data update rate*	$380kHz$

* user choice

Table 2.2: Gyroscope Specifications - L3G

2.3 Picam

The Raspberry Pi 2 features a full HD camera supporting **1080*p*** full HD resolution at **30*fps***. The camera is attached to the Pi 2 with the help of a Camera Serial Interface (CSI). The Pi 2 features the Broadcom BCM2836 System on a Chip (SoC), with a quad-core ARM Cortex-A7 Central Processing Unit (CPU) and a VideoCore IV dual-core Graphical Processing Unit (GPU). The processing power of 1 GB of RAM is distributed between the CPU(512 MB) and GPU(512 MB) while the storage memory is based on a MicroSD card. A schematic of the picam is shown in Fig. 2.1b. The specifications of the Pi Cam are listed in table 2.3.



(a) Schematic of Pololu IMU **(b)** Schematic of Picam

Figure 2.1: Schematic - Sensors

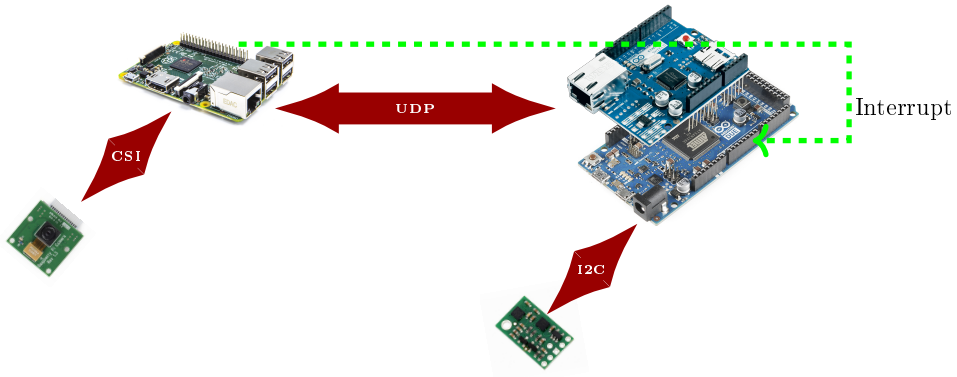


Figure 2.2: Schematic - Sensor Interfaces

In the current project, the sensor resolution is limited to that of $\mathbf{1280{\times}720}\boldsymbol{pixels@10}\boldsymbol{fps}$ to account for the image processing and object detection.

PiCam Characteristics	**Value**
Size around	$\mathbf{25 \times 20 \times 9}\boldsymbol{mm}$
Weight	$\mathbf{3}\boldsymbol{g}$
Still resolution	$\mathbf{5}\boldsymbol{Mpix}$
Video modes	$\mathbf{2592 \times 1944}\boldsymbol{p}\mathbf{15, 1080}\boldsymbol{p}\mathbf{30, 720}\boldsymbol{p}\mathbf{60}$ & $\mathbf{640 \times 480}\boldsymbol{p}\mathbf{60/90}$
Linux integration	V4L2 driver available
C programming API	OpenMAX IL & others available
Sensor	OmniVision OV5647
Sensor resolution	$\mathbf{2592 \times 1944}\boldsymbol{pixels}$
Sensor image area	$\mathbf{3.76 \times 2.74}\boldsymbol{mm}$
Pixel size	$\mathbf{1.4}\boldsymbol{\mu m} \times \mathbf{1.4}\boldsymbol{\mu m}$
Optical size	1/4”
Full-frame SLR lens equivalent	$\mathbf{35}\boldsymbol{mm}$
S/N ratio	$\mathbf{36}\boldsymbol{dB}$
Dynamic range	$\mathbf{67}\boldsymbol{dB}$@8X$\boldsymbol{gain}$
Densitivity	$\mathbf{680}\boldsymbol{mV/lux/s}$
Dark current	$\mathbf{16}\boldsymbol{mV/s}$@$\mathbf{60}\boldsymbol{C}$
Well capacity	$\mathbf{4.3}\boldsymbol{Ke-}$
Fixed Focus	$\mathbf{1}\boldsymbol{m}$ to $\boldsymbol{\infty}$
Focal length	$\mathbf{3.60}\boldsymbol{mm} \pm \mathbf{0.01}$
Horizontal field of view	$\mathbf{53.50 \pm 0.13}\boldsymbol{degrees}$
Vertical field of view	$\mathbf{41.41 \pm 0.11}\boldsymbol{degress}$
Focal ratio (F-UDP)	**2.9**

Table 2.3: Camera Specifications - Raspberry Pi camera (Picam)

2.4 Overview of Controllers & Sensor Interfaces

The Raspberry Pi is used as a Digital Acquisition System (DAQ) board for the Pi-cam while the sensor fusion is performed on the Arduino Due. A schematic of the sensors and the various interfaces used is shown in Fig. 2.2.

2.4.1 Raspberry Pi

A standard Mobile Industry Processor Interface (MIPI) interface (CSI-2) conncets the Pi-cam with the Pi-2. A detailed description of the image processing and object detection on the Pi-2 can be found in Chapter 4. The following libraries are used in the RPi.

- The open source library "*Wiring Pi*" is used to generate an interrupt when a new image is captured.
- The standard library "sys/socket" is used for User Datagram Protocol (UDP) communication with Arduino(ethernet) as well as with Laptop/ground station(Wifi).
- The image processing library "OpenCV 2.4.9" [39] is used for object detection.
- The matrix library "Eigen-3" [40] is used for pose estimation on the Rpi.
- The Khrnons Application Programming Interface (API) OpenGL ES 2.0 graphics library [41] is used to generate textures and process the image on the GPU.

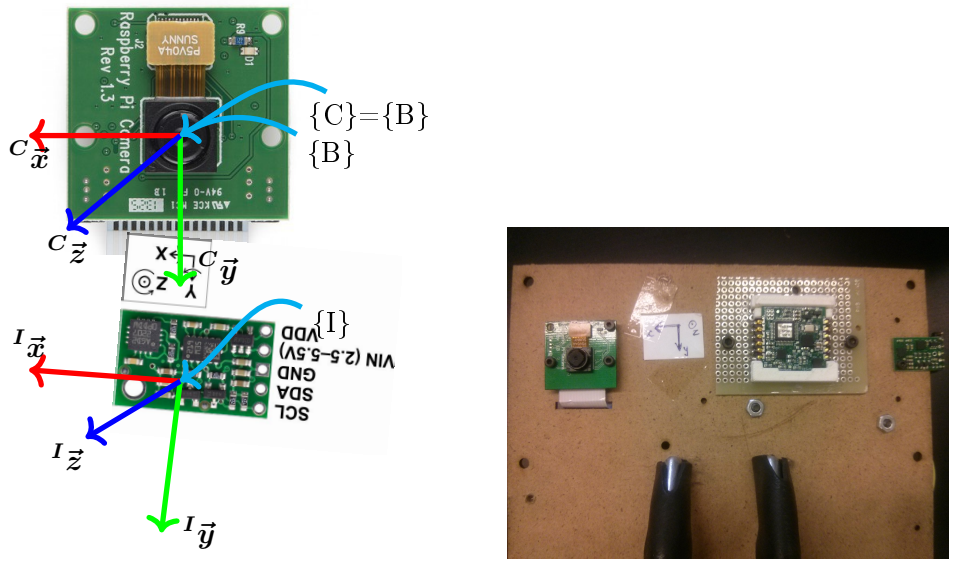
2.4.2 Arduino

The standard libraries of the Arduino utilized for the project are listed below

- The Inter-Ic Bus (I2C) library is used to communicate with the Pololu IMU. The contributed libraries "lsm-303" and "l3g" were used to obtain raw sensor readings from the IMU.
- The Serial Peripheral Interface (SPI), "ethernet" and "ethernetUDP" standard libraries were used to communicate the Arduino with the Rpi using UDP protocol.
- The General Purpose Input Output (GPIO) and "serial" libraries were used to communicate with the user.
- The Interrupts standard library was used to read interrupts when a new image is captured/processed and also to obtain the encoder readings.
- The matrix librarry "Eigen-3"[40] was used to perform matrix and vector operations relating to the filtering of the raw sensor data and implementation of the Kalman filter for sensor fusion.

2.5 Co-Ordinate Frames

The primary co-ordinate frames used in this project are the global co-ordinate system $\{G\}$ attached to the landmark, the camera co-ordinate system $\{C\}$ which conincides with the body co-ordinate system $\{C\} = \{B\}$ and the IMU co-ordinate system $\{I\}$. As the IMU and camera are rigidly attached to one another and to the ROV, the raw sensor readings in the IMU frame $\{I\}$ are converted to the camera co-ordinates $\{C\}$ for state estimation. A detailed description of the same is provided in Chapter 3.



(a) Schematic - Sensor Frames

(b) Proof of Concept Sensor Apparatus

Figure 2.3: Schematic of Co-ordinate Frames of Reference

2.6 Equivalence of Co-ordinate Frame Transformation

It is often convenient to solve a set of equations in a particular frame of reference and then transform the solution to the required co-ordinate system. In this project, as the camera measurements are performed in the Camera Co-ordinate system $\{C\}$, it is convenient to solve the for the unknown pose in $\{C\}$ and later transform the pose to the Global frame of reference $\{G\}$. This section details the pose equivalence between the two co-ordinate frames.

The co-ordinate transformation from one reference frame to the other can be performed easily as shown in Fig. 2.4. The transformation matrix of one reference frame represented in another reference frame is given by (2.1)

$$\begin{aligned} {}^{G}_{C}\boldsymbol{R} &= {}^{C}_{G}\boldsymbol{R}^{\boldsymbol{T}} \\ {}^{G}\boldsymbol{P}_{C} &= -{}^{C}_{G}\boldsymbol{R}^{\boldsymbol{T}}\,{}^{C}\boldsymbol{P}_{G} \end{aligned} \tag{2.1}$$

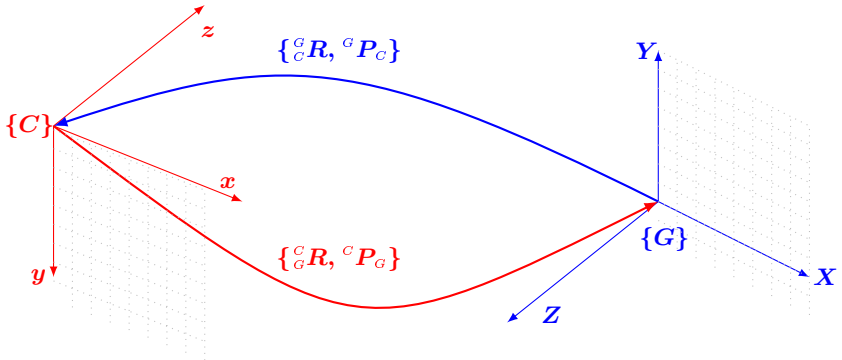


Figure 2.4: Co-ordinate Frame Transformation Equivalence

Any point in one reference frame can be related in another reference frame using (2.2)

$${}^{C}\boldsymbol{p_i} = {}^{C}\boldsymbol{P}_{G} + {}^{C}_{G}\boldsymbol{R}\,{}^{G}\boldsymbol{P_i} \tag{2.2}$$

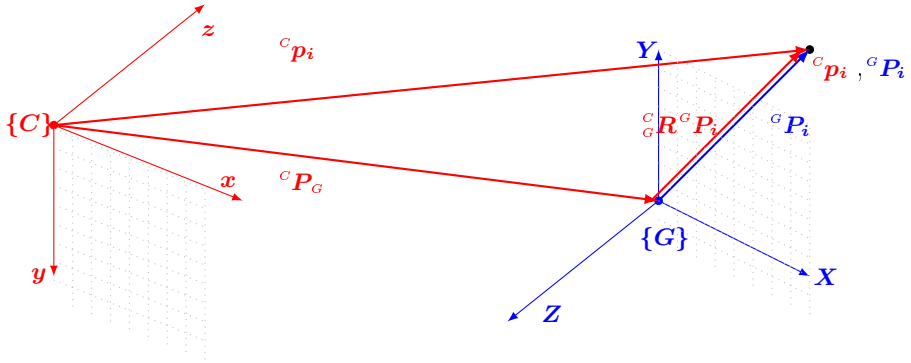


Figure 2.5: Co-ordinate Frame Vector Equivalence

2.7 Conclusion

In this Chapter, we have provided an insight of the target hardware and software being used for the ROV. A brief introduction is made about the co-ordinate frames of reference that will be used in the forthcoming chapters. The sensor-calibration is discussed in detail in Chapter 3.

Chapter 3

Calibration

This chapter describes in detail the sensor models, the procedure used to calibrate the IMU, the Pi-camera and cross-calibration between the IMU and the Pi-camera.

3.1 Overview

This chapter is organized as follows. Section 3.2 discusses calibration procedure for 3-axis gyroscopes and Section 3.3 describes the calibration of 3-axis accelerometers, which constitute the IMU. Section 3.4 gives a detailed description of the pin hole Pi-camera and the auto-calibration procedure using a checker board pattern. Section 3.5 describes the cross-calibration between the IMU and the pi-camera, which completes the calibration routines. The results and the conclusions are discussed in Section 3.6 which concludes the chapter.

3.2 Gyroscopes

The sensor data from the gyroscope (${}^{I}\vec{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^{T}$) is the measure of the angular velocity. However, the raw sensor data is usually corrupted by a constant mean signal commonly known as the gyro bias. In the sensor frame, the gyro bias is assumed to be a constant as shown in (3.1).

$$\vec{\boldsymbol{b}} = \vec{\boldsymbol{b}}_0 + \vec{w} \tag{3.1}$$

The bias $\vec{\boldsymbol{b_0}}$ is found from a simple average of the steady state gyroscope readings for a period of about 10s.

A simple transformation matrix can be used to transform the angular velocity from the IMU frame $\{I\}$ to the camera frame $\{C\}$ as the two frames are located on the same inertial body. The translation distance between $\{I\}$ and $\{C\}$ is assumed to be very small and hence the change in angular acceleration (Coriolis acceleration and centripetal acceleration) due to this distance has been ignored(refer 3.5.4). After bias removal, the angular rate from the gyroscopes in the camera frame $\{C\}$ is given by (3.2).

$$^{C}\vec{\boldsymbol{\omega}} = {}^{C}_{I}R\,(^{I}\vec{\boldsymbol{\omega}} - \vec{\boldsymbol{b}}) \tag{3.2}$$

3.3 Accelerometer

The accelerometer is used for position estimation by double integration of acceleration. Hence, a small bias in the accelerometer will yield a large drift in position. In slow moving vehicles, the accelerometer(which measures the direction of gravity) and magnetometer(which measures the direction of earth's magnetic field) can be used to estimate the orientation of the ROV using dead reckoning [42]. However, in this project the primary function of the acceleormeters is to aid the position estimation from the camera. A standard model for the MEMS accelerometer is assumed to have a scale factor matrix (with cross-corelation terms) $K_{acc} \in R_{3\times3}$ and a bias $b_{acc} \in R^3$ [43], which are estimated using auto-calibration. The sensor model used to compute the actual acceleration ($\vec{a}$) from the measured acceleration ($\vec{a}_m$) with prior knowledge of the magnitude of acceleration (gravity g in static case) is used to calibrate the accelerometer is given by (3.3).

$$\vec{\boldsymbol{a}} = K_{acc}\vec{\boldsymbol{a}}_m + \vec{\boldsymbol{b}}_{acc} \tag{3.3}$$

At steady state, ideally the magnitude of the accelerometer should be earth's acceleration due to gravity $|g|$. Hence, a number of measurements can be taken in different static orientations. As the magnitude should be the same, ideally a plot of the acceleration in 3D co-ordinates should be a sphere. However, due to the scale factor and accelerometer static bias, the sphere is distorted. From a number of the observation points the scale factor and the bias can be estimated.

Due to the presence of the scale factor and bias terms, the error is typically in the range of 2-3% of $|g|$. By estimating the scale factor and bias accurately, this error is reduced to about 0.2-0.3% of $|g|$.

The estimation of the scale factor K_{acc} and the bias b_{acc} is performed using an ellipsoid fit. A number of points in different orientations at steady state are obtained from the accelerometer. The cost function J to be minimized is given using (3.4).

$$\begin{aligned} \textbf{Objective:} \quad & \min_{K_{acc},\vec{b}_{acc}} \quad J = \sum_{i=0}^{n} \mid \; ||\vec{\boldsymbol{a}}_i||^2 - ||g||^2 \; \mid \\ & \text{given} \quad \vec{\boldsymbol{a}}_i = \text{static acceleration measurement} \end{aligned} \tag{3.4}$$

Simplifying, (3.3), we obtain

$$J = \sum_{i=0}^{n} |a_i^T * a_i - |g|^2| = \sum_{i=0}^{n} |(K_{acc}\vec{\boldsymbol{a}}_{mi} + \vec{\boldsymbol{b}}_{acc})^T (K_{acc}\vec{\boldsymbol{a}}_{mi} + \vec{\boldsymbol{b}}_{acc}) - ||g||^2| \tag{3.5}$$

$$J = \sum_{i=0}^{n} |(\vec{\boldsymbol{a}}_{mi}^T K_{acc}^T + \vec{\boldsymbol{b}}_{acc}^T)(K_{acc}\vec{\boldsymbol{a}}_{mi} + \vec{\boldsymbol{b}}_{acc}) - ||g||^2| \tag{3.6}$$

$$J = \sum_{i=0}^{n} |\vec{\boldsymbol{a}}_{mi}^T (K_{acc}^T K_{acc})\vec{\boldsymbol{a}}_{mi} + 2\vec{\boldsymbol{b}}_{acc}^T K_{acc}\vec{\boldsymbol{a}}_{mi} + \vec{\boldsymbol{b}}_{acc}^T \vec{\boldsymbol{b}}_{acc} - ||g||^2| \tag{3.7}$$

The cost function J is non-linear with respect to the scale factor and the bias as it involves product terms. Hence, we try to relate the cost function to a standard convex function of an ellipsoid and solve it using linear least squares.

The standard equation of an arbitrary ellipsoid in 3 Dimensional (3D) space with center $\vec{\boldsymbol{v}}$ and symmetric, positive semi-definite matrix A is given by (3.8)-3.9

$$(\vec{\boldsymbol{x}} - \vec{\boldsymbol{v}})^T A (\vec{\boldsymbol{x}} - \vec{\boldsymbol{v}}) = 1 \tag{3.8}$$

$$\vec{\boldsymbol{x}}^T \underbrace{A}_{F} \vec{\boldsymbol{x}} \underbrace{-2\vec{\boldsymbol{v}}^T A}_{G} \vec{\boldsymbol{x}} + \underbrace{\vec{\boldsymbol{v}}^T \vec{\boldsymbol{v}}}_{H} = 1 \tag{3.9}$$

Given the co-ordinates of the ellipsoid ($\vec{\boldsymbol{x}}$), we estimate the matrix A and the center point $\vec{v}$. Relating (3.9) to (3.7), we solve for the best possible matrix K_{acc} and vector $\vec{\boldsymbol{b}}_{acc}$ which satisfy (3.10).

$$K_{acc}^T K_{acc} = A \tag{3.10}$$

$$2\vec{\boldsymbol{b}}_{acc}^{T}K_{acc} = -2\vec{\boldsymbol{v}}^{T}A$$

From the datasheet of the accelerometer, we know that the cross-corelation terms are less than 2.5% and the deviation of scale factor is less than 3% of ideal. Hence, the scale factor K_{acc} is close to identity I, so that $K_{acc} \approx I + k_{acc}\ s.t. k_{acc}(i,j) < 0.025, k_{acc}(i,i) < 0.03$. Now the problem reduces to an ellipsoid fitting problem to estimate the parameters of the ellipsoid.

$$J = \sum_{i=0}^{n} |\vec{\boldsymbol{a}}_{mi}^{T} F \vec{\boldsymbol{a}}_{mi} + G\vec{\boldsymbol{a}}_{mi} + H| \tag{3.11}$$

Using the ellipsoid fit, we can compute the unknowns F, G, H using the equations (3.10) in (3.11). The estimated quantities F, G, H are used to approximate the bias and scale factor using (3.9), (3.10) as shown in (3.12)

$$F = K_{acc}^{T}K_{acc} = I + k_{acc} + k_{acc}^{T} + k_{acc}^{T}k_{acc} \approx I + k_{acc} + k_{acc}^{T} \tag{3.12}$$

$$\frac{k_{acc} + k_{acc}^{T}}{2} = \frac{F - I}{2} = \begin{bmatrix} k_{11} & \frac{k_{12}+k_{21}}{2} & \frac{k_{31}+k_{13}}{2} \\ \frac{k_{12}+k_{21}}{2} & k_{22} & \frac{k_{23}+k_{32}}{2} \\ \frac{k_{31}+k_{13}}{2} & \frac{k_{23}+k_{32}}{2} & k_{33} \end{bmatrix} \tag{3.13}$$

Clearly the diagonal terms ($k_{acc}(i,i)$) can be computed using (3.13). The cross terms are only related by the sum. Hence, the matrix k_{acc}) can be assumed to be the sum of a known symmetric matrix (k_{acc}^{*}) and an unknown skew-symmetric matrix (Ω_{acc}).

$$k_{acc} = \begin{bmatrix} k_{11} & \frac{k_{12}+k_{21}}{2} & \frac{k_{31}+k_{13}}{2} \\ \frac{k_{12}+k_{21}}{2} & k_{22} & \frac{k_{23}+k_{32}}{2} \\ \frac{k_{31}+k_{13}}{2} & \frac{k_{23}+k_{32}}{2} & k_{33} \end{bmatrix} + \begin{bmatrix} 0 & \gamma_{xy} & \gamma_{xz} \\ -\gamma_{xy} & 0 & \gamma_{yz} \\ -\gamma_{xz} & -\gamma_{yz} & 0 \end{bmatrix} = k_{acc}^{*} + \Omega_{acc} \tag{3.14}$$

The estimate for the bias and scale factor are shown below

$$K_{acc} = I + k_{acc}^{*} + \Omega_{acc}\ ;\ b_{acc} = \frac{(K_{acc}^{-1}G)^{T}}{2} \tag{3.15}$$

The optimal value of the cross-corelation terms Ω_{acc} is computed by iterating over the possible range of values (-0.025 to 0.025) which minimizes the cost function given by (3.4). This leads to an optimum value of the scale factor and bias.

An alternate method to compute K_{acc} and b_{acc} is to directly optimize (3.7) with respect to K_{acc} and b_{acc} using Matlab's the function $lsqcurvefit$ from the curve fitting

toolbox of Matlab. The "trust-region-reflective" algorithm is used to find the optimum curve parameters of a non-linear function which minimizes the error norm between the curve and the observed data.

3.3.1 Simulation of Accelerometer Calibration

The accelerometer measurements are simulated using the known gravity vector in random static orientations with the addition of a random Gaussian noise to the measurement. Let R be a randomly generated rotation matrix and $\vec{w}$ represent a random Gaussian noise vector ($\sigma = 0.05, \mu = 0$). The measurement points are simulated using (3.3) as shown in (3.16).

$$\vec{\boldsymbol{a}}_m = K_{acc}^{-1}(R\vec{\boldsymbol{g}} + \vec{w} - \vec{\boldsymbol{b}}_{acc/sim}) \tag{3.16}$$

The assumed scale factor and bias used to compute the measurement from the given in (3.17)

$$k_{acc/sim_{actual}} = \begin{bmatrix} 0.0050 & -0.0250 & -0.0080 \\ 0.0100 & 0.0150 & -0.0060 \\ -0.0090 & 0.0100 & -0.0100 \end{bmatrix} \; ; \; \vec{\boldsymbol{b}}_{acc/sim_{actual}} = \begin{bmatrix} 0.1 \\ -0.2 \\ 0.3 \end{bmatrix} \tag{3.17}$$

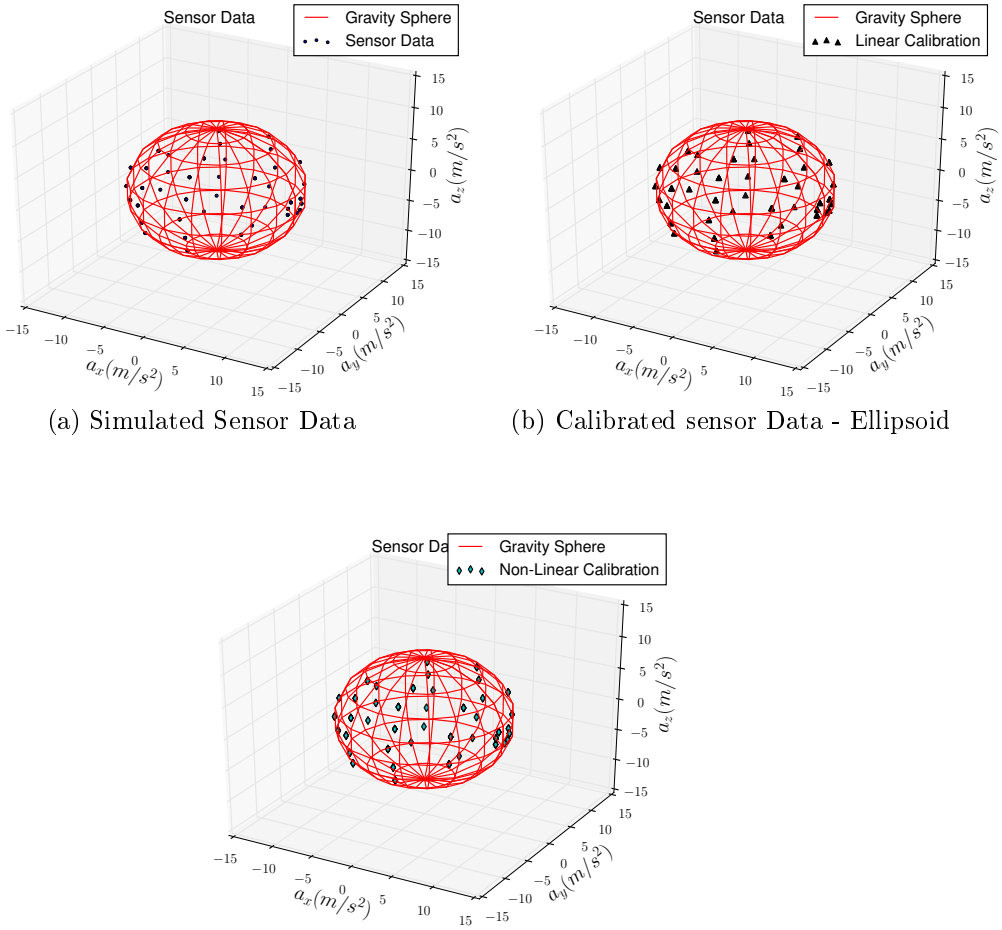
The results of the simulation are shown in Fig. 3.1. The estimated values of the scale factor and bias using the Matlab curve fitting function $lsqcurvefit$ (LQ parameter optimization) is given in (3.18)

$$k_{acc/sim_{MATLAB}} = \begin{bmatrix} 0.0050 & -0.0076 & -0.0085 \\ -0.0072 & 0.0153 & -0.0036 \\ -0.0086 & 0.0077 & -0.0099 \end{bmatrix} \; ; \; \vec{\boldsymbol{b}}_{acc/sim_{MATLAB}} = \begin{bmatrix} 0.0965 \\ -0.2011 \\ 0.3008 \end{bmatrix} \tag{3.18}$$

The estimated values of the scale factor and bias obtained using the ellipsoid fitting are given in (3.19)

$$k_{acc/sim_{ellipsoid}} = \begin{bmatrix} 0.0051 & -0.0095 & -0.0105 \\ -0.0055 & 0.0155 & 0.0020 \\ -0.0065 & 0.0020 & -0.0098 \end{bmatrix} ; \vec{\boldsymbol{b}}_{acc/sim_{ellipsoid}} = \begin{bmatrix} 0.0963 \\ -0.1992 \\ 0.3021 \end{bmatrix} \tag{3.19}$$

As seen, the estimated diagonal terms match the diagonal terms very closely but the skew-symmetric terms Ω_{acc} are quite different from the actual values.



(a) Simulated Sensor Data

(b) Calibrated sensor Data - Ellipsoid

(c) Caibrated sensor data - MATLAB

Figure 3.1: Simulation Result - Accelerometer Data from IMU

A comparison of the rms error before and after calibration (Fig. 3.2) shows the significant reduction of error after accelerometer calibration.

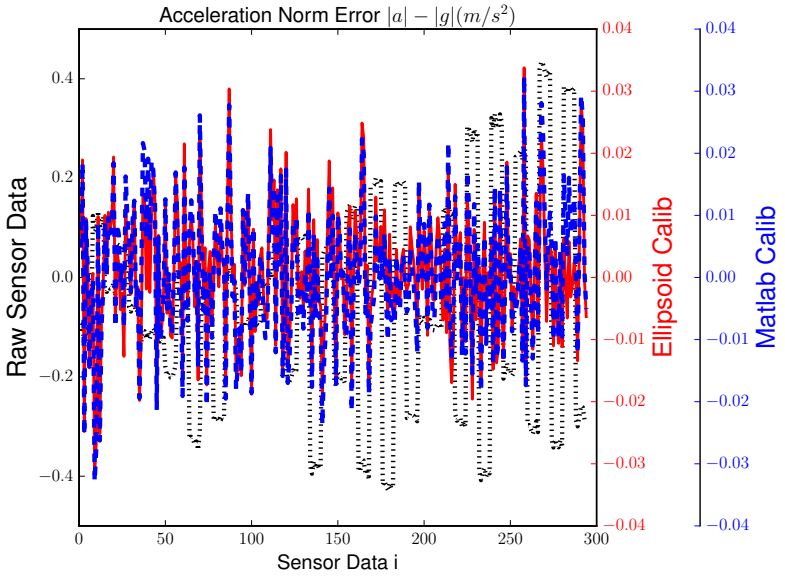


Figure 3.2: Simulation Result - Acceleration Norm error from IMU

$\epsilon_{rms/sim_{raw}} = 0.2054, \epsilon_{rms/sim_{Matlab}} = 0.0080, \epsilon_{rms/sim_{ellipsoid}} = 0.0101$

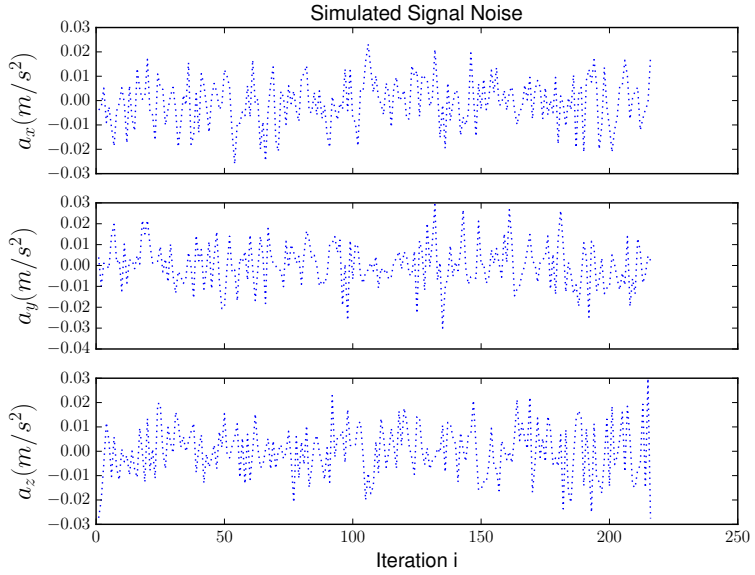


Figure 3.3: Simulation Result - Zero mean white noise in Accelerometer

3.3.2 Experimental Calibration of Acclerometer

In static steady state, the accelerometer measurements are obtained in various orientations as measurement points.

The accelerometer data obtained from the LSM303DLHC accelerometer (Pololu IMU) is shown in Fig. 3.5. Using the theory described in Section 3.3, the scale factor matrix K_{acc} and the bias vector $\vec{b}_{acc}$ are computed using the ellipsoid fitting equation (3.15) and the direct optimization using Matlab curve fitting toolbox.

The estimated scale factor and bias for the non-linear calibration using $lsqcurvefit$ is given in (3.20)

$$k_{acc/est_{nlin}} = \begin{bmatrix} 0.0188 & 0.0139 & 0.0009 \\ -0.0172 & -0.0209 & -0.0004 \\ 0.0051 & -0.0013 & -0.0226 \end{bmatrix} \; ; \; \vec{\boldsymbol{b}}_{acc/est_{nlin}} = \begin{bmatrix} -0.0455 \\ 0.1266 \\ -0.4156 \end{bmatrix} \tag{3.20}$$

The estimated scale factor and bias using the least squares algorithm is given in (3.21)

$$k_{acc/est} = \begin{bmatrix} 0.0191 & -0.0023 & 0.0060 \\ -0.0003 & -0.0205 & -0.0008 \\ -0.0000 & -0.0008 & -0.0224 \end{bmatrix} \; ; \; \vec{\boldsymbol{b}}_{acc/est} = \begin{bmatrix} -0.0497 \\ 0.1260 \\ -0.4154 \end{bmatrix} \tag{3.21}$$

The error in norm of acceleration before and after calibration is shown in Fig. 3.4

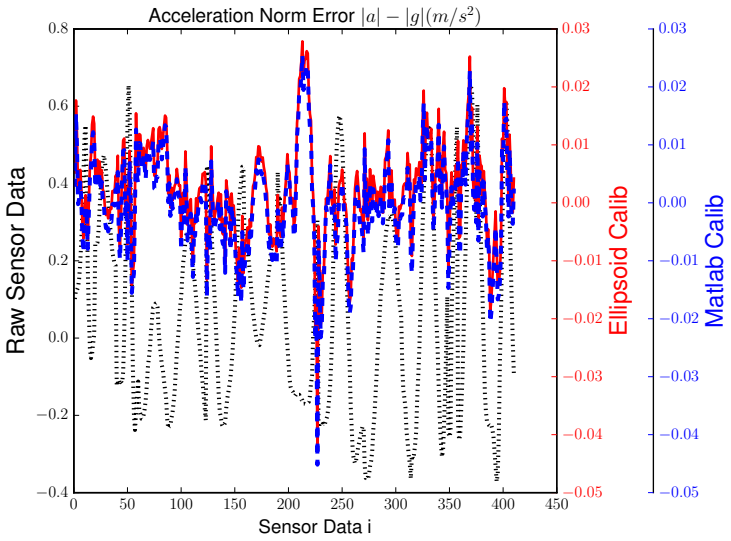
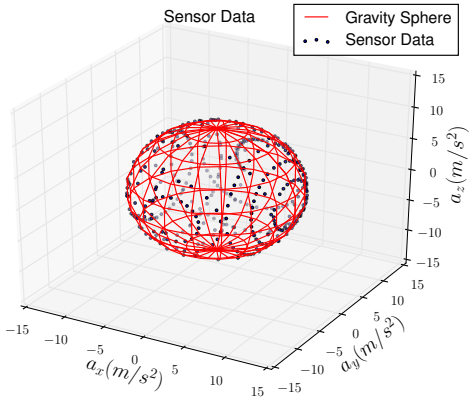


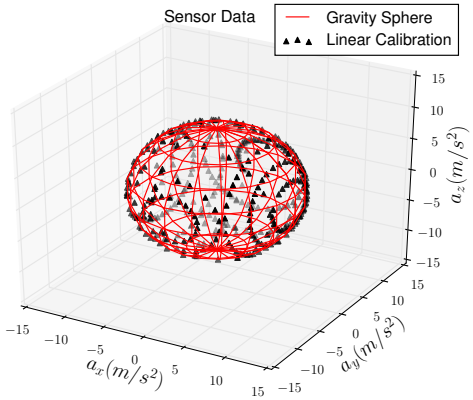
Figure 3.4: Experimental Result - Acceleration Norm error from IMU

$$\epsilon_{rms/raw} = 0.2658, \epsilon_{rms/est_{ellipsoid}} = 0.0087, \epsilon_{rms/est_{Matlab}} = 0.0062$$

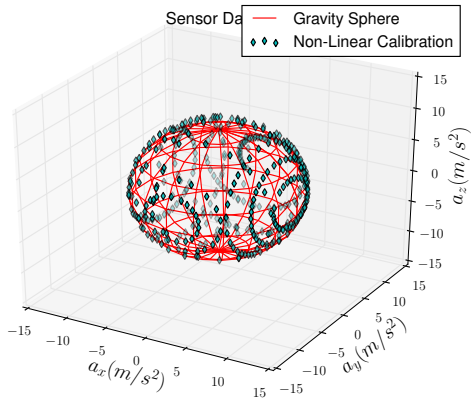
The difference in the RMS error between the Matlab based curve fitting and the simplified ellipsoid fitting is negligible. The former involves tedious computations while the latter is light weight and can be implemented on the Arduino Due board.



(a) Simulated Sensor Data



(b) Calibrated Sensor Data - Ellipsoid



(c) Calibrated Sensor Data - MATLAB

Figure 3.5: Experimental Result - Accelerometer Data (IMU)

3.4 Picam Camera

The camera is assumed to be a pinhole camera (focal length f_x, f_y and image center c_x, c_y) with the following realtion between object co-ordinates in camera reference frame and the measured pixels.

$$ {}^{C}p_z \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{A_{picam}} {}^{C}\vec{\boldsymbol{p}} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{C}p_x \\ {}^{C}p_y \\ {}^{C}p_z \end{bmatrix} \tag{3.22} $$

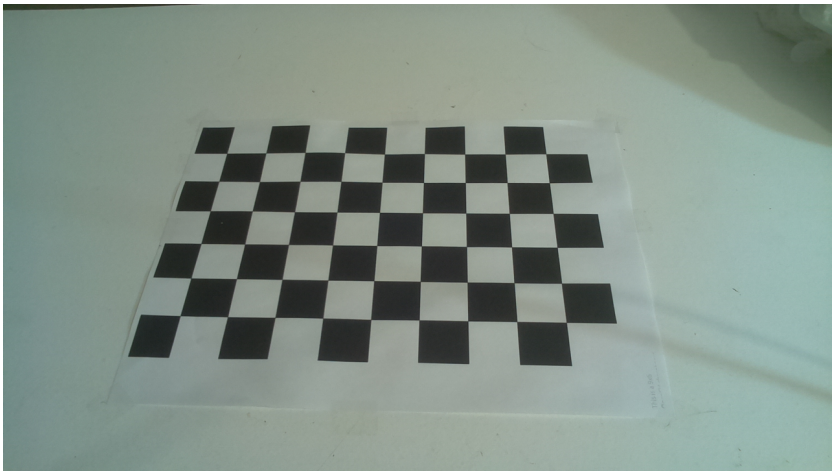
The estimation of the focal length(f_x, f_y) and the center of the camera(c_x, c_y) are obtained using a classical black-white cheker-board pattern in OpenCV [44]. The corrected image pixels $\vec{\boldsymbol{u}}_c$ are obtained from the captured image pixels $\vec{\boldsymbol{u}}$ using the co-efficients due to tangential (p_1, p_2) and radial (k_1, k_2) distortion, with the radial distance given by $(r = \frac{\sqrt{{}^{C}p_x^2 + {}^{C}p_y^2)}}{{}^{C}p_z})$ are modeled as shown in (3.23).

$$ \vec{\boldsymbol{u_c}} = \vec{\boldsymbol{u}} + \vec{\boldsymbol{u}}(1 + k_1 r^2 + k_2 r^4) + \begin{bmatrix} 2p_1 u_x u_y + p_2(r^2 + 2u_x^2) \\ p_1(r^2 + 2y^2) + 2p_2 u_x u_y \end{bmatrix} \tag{3.23} $$

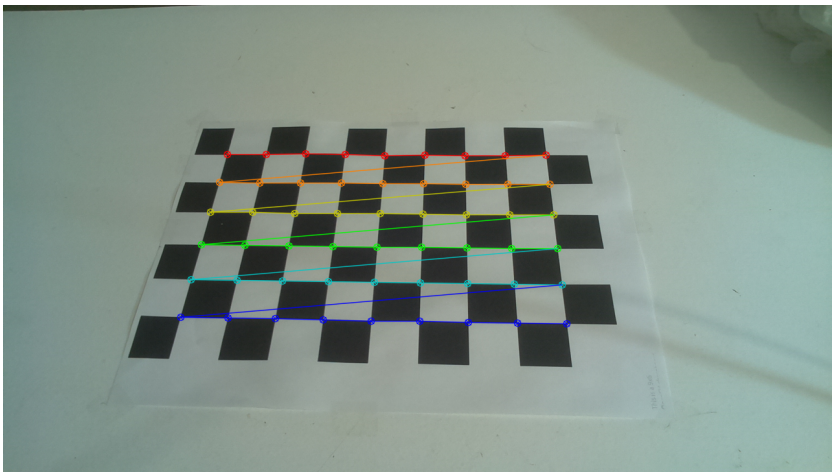
3.4.1 Calibration Procedure

A brief description of the calibration procedure is discussed below:

- Obtain static images of the chessboard pattern in several different poses (≈ 10) at a distance about 2m. away from the camera center.
- Compute the image pixel co-ordinates using OpenCV builtin function "findChessboardCorners()" as shown in Fig. 3.6
- Use the default square size ($= 2.5cm$) and the image co-ordinates to compute the pose, intrinsic matrix and the distortion co-efficients.



(a) 9X6 chessboard with square size 2.5mm



(b) Detected Cornes of chessboard using OpenCV

Figure 3.6: Calibration Pattern - Standard Checker board

The intrinsic camera matrix obtained by the non-linear model taking into effect the radial and tangential distortion using the OpenCV function "calibrateCamera()" is given in (3.24)3.25

$$A_{picam} = \begin{bmatrix} 1169.81 & 0 & 623.60 \\ 0 & 1166.96 & 335.90 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.24}$$

$$D_{picam} = \begin{bmatrix} k_1 \\ k_2 \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} -0.04527 \\ 0.70996 \\ -0.00301 \\ 0.002199 \end{bmatrix} \tag{3.25}$$

The intrinsic camera matrix obtained using a pin-hole camera without any radial or tangential distortion using the OpenCv "$calibrateCamera()$" is given in (3.26)

$$A_{picam} = \begin{bmatrix} 1165.96 & 0 & 639.5 \\ 0 & 1165.96 & 359.5 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.26}$$

The effect due to non-linearity is very minimal and is ignored in the pose estimation problem. The RMS error after re-projection of object points using the computed intrinsic matrix after calibration is found to be $\epsilon_{rms} = 0.15\, pixels$.

3.5 Picam IMU Calibration

After calibration of Picam and IMU, the estimation of the rotation matrix between the Picam and IMU is achieved using the Markley solution to Wahba problem [45].

3.5.1 Apparatus

We make use of a Gravity level bench and a Mercury indicator to make sure the bench is perpendicular to gravity. A checker board is mounted on top of the level bench and video from the camera is recorded along with the accelerometer data at several possible static poses for about 60s in each pose, making sure that the chessboard is in sight and the sensor is in steady state. A schematic of the apparatus as seen from the pi-camera is shown in Fig. 3.7

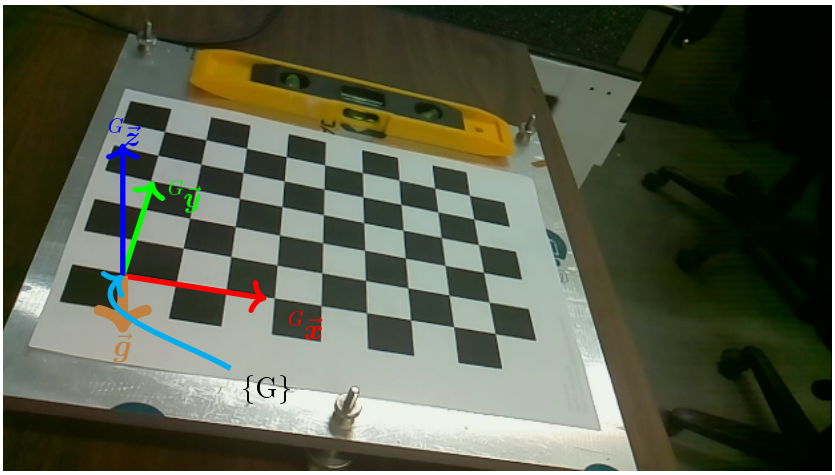


Figure 3.7: Camera-Imu Calibration Apparatus

The orientation of the camera w.r.t. the checker board (calibration marker frame of reference $\{G\}$) can be estimated using the image (from video) of the checker board at the stationary location and the calibration matrices of the pi-camera. Each image from the video is processed to obtain an estimate of the orientation of the camera w.r.t. the marker R_j. The results for each particular orientation from the camera are stored in separate files $vid\#i$ while the IMU data for each orientation are saved in $imu\#i$.

3.5.2 Orientation Calibration

The gravity direction in $\{C\}$ is given by (3.27) and can be visualized in Fig.3.7.

$$ {}^{C}\vec{\boldsymbol{a}} = {}^{C}_{G}R\ {}^{G}\vec{\boldsymbol{a}} = {}^{C}_{G}R \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{3.27} $$

The unit gravity direction i.e. unit normal to the board is computed from each image ${}^{C}\vec{\boldsymbol{a}}^{"}_{j}$ of the video (composed of M_1 static images at the same pose i) and is averaged to get the mean gravity ${}^{C}\vec{\boldsymbol{a}}_i$ (3.28).

$$ {}^{C}\vec{\boldsymbol{a}}^{'}_{i} = \frac{\sum_{j=1}^{M_1} {}^{C}\vec{\boldsymbol{a}}^{"}_{j}}{M_1},\ {}^{C}\vec{\boldsymbol{a}}_i = \frac{{}^{C}\vec{\boldsymbol{a}}^{'}_{i}}{||{}^{C}\vec{\boldsymbol{a}}^{'}_{i}||} \tag{3.28} $$

The average of the IMU acceleration (normalized) at the same location i (composed of M_2 data readings) is computed from each of the measurements $({}^{I}\vec{\boldsymbol{a}}_j)_{imu}$ (3.29).

$$ {}^{I}\vec{\boldsymbol{a}}_i = \frac{\sum_{j=1}^{M_2} \frac{{}^{I}\vec{\boldsymbol{a}}_j}{||{}^{I}\vec{\boldsymbol{a}}_j||}}{M_2},\ {}^{I}\vec{\boldsymbol{a}}_i = \frac{{}^{I}\vec{\boldsymbol{a}}_i}{||{}^{I}\vec{\boldsymbol{a}}_i||} \tag{3.29} $$

The optimal orientation matrix between the camera and the IMU can be obtained by taking sufficient readings $N, s.t. i \in (1, N)$ so that we have the same gravity vector in both $\{C\}$ and $\{I\}$. The optimal orientation given N vectors in $\{C\}$ and their corresponding vectors in $\{I\}$ can be used to compute ${}^{C}_{I}R$ by minimizing the cost function in (3.30) w.r.t ${}^{C}_{I}R$ [45].

$$ J = \sum_{i=1}^{N} ||{}^{C}\vec{\boldsymbol{a}}_i - {}^{C}_{I}R\ {}^{I}\vec{\boldsymbol{a}}_i||^2 \tag{3.30} $$

We find a matrix B as shown in (3.31)

$$ B = \sum_{i=1}^{N} {}^{C}\vec{\boldsymbol{a}}_i \times {}^{I}\vec{\boldsymbol{a}}_i \tag{3.31} $$

We now find the SVD of the matrix B as shown in (3.32)

$$ B = USV^T \tag{3.32} $$

The optimal orientation ${}^{C}_{I}R$ is given by (3.33)

$$ {}^{C}_{I}R = UV^T \tag{3.33} $$

3.5.3 Results

The final optimal orientation between the pi-camera and the IMU is given in (3.34)

$$ {}^{C}_{I}R = \begin{bmatrix} 0.9996 & -0.0128 & 0.0231 \\ 0.0100 & 0.9931 & 0.1171 \\ -0.0244 & -0.1168 & 0.9929 \end{bmatrix} \tag{3.34} $$

The results of the transformation are shown in Fig. 3.8.

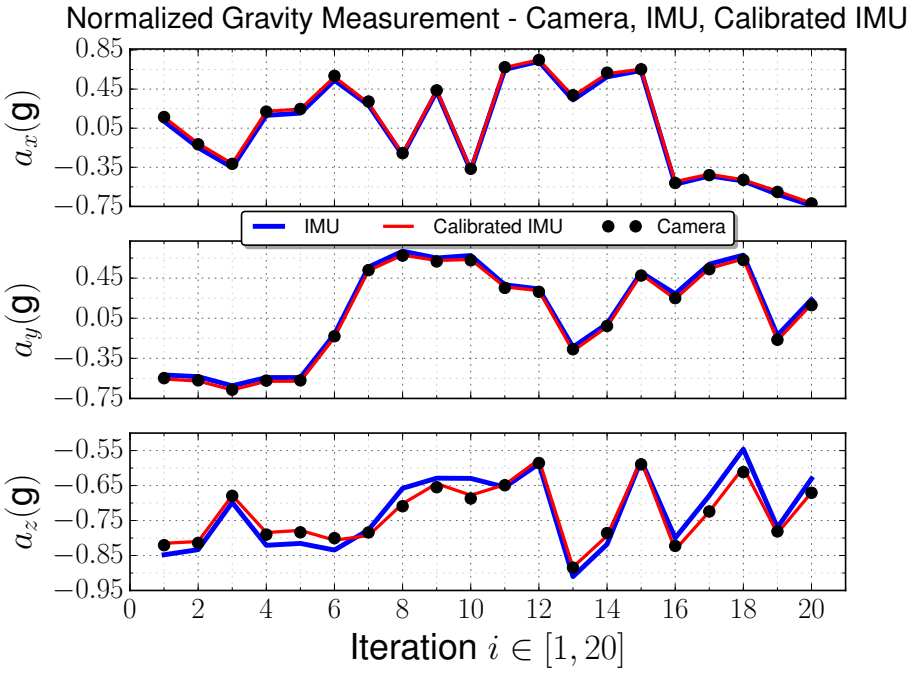


Figure 3.8: Camera-Imu Calibration Results

3.5.4 Effect of Translation distance between IMU and Camera

The translational difference in origin between the IMU frame {I} and the camera frame {C}, $|{}^{C}\vec{P}_I|$ induces a difference of the acceleration of the two origins $\Delta^{C}\vec{\boldsymbol{a}}$ composed of a centripetal acceleration (${}^{C}\vec{\boldsymbol{a}}_{cen}$) and a tangential acceleration (${}^{C}\vec{\boldsymbol{a}}_{tan}$) as shown in (3.35).

$$\Delta^{C}\vec{\boldsymbol{a}} = {}^{C}\vec{\boldsymbol{a}}_{cen} + {}^{C}\vec{\boldsymbol{a}}_{tan} = \underbrace{{}^{I}\omega \times ({}^{I}\omega \times {}^{C}\vec{P}_I)}_{{}^{C}\vec{\boldsymbol{a}}_{cen}} + \underbrace{{}^{I}\alpha \times {}^{C}\vec{P}_I}_{{}^{C}\vec{\boldsymbol{a}}_{tan}} \tag{3.35}$$

Since the angular velocity of the ROV (${}^{I}\omega < 0.1$ rad/s) and the center distance ($|{}^{C}\vec{P}_I|$<2.5 cm) are small, which would make the acceleration due to radial and tangential components negligible ($\approx 0.025 cm/s^2$) when compared to gravity and body acceleration.

If necesary, the simultaneous calibration of camera, IMU along with the translation distance can be computed using EKF[46].

3.6 Conclusion

This chapter details the sensor calibration process to minimize the sensor error. The individual sensors of the IMU , the gyroscopes and the accelerometers are calibrated using the known standard sensor models. The Pi-camera is calibrated using a standard chess board pattern. Finally, we compute the cross calibration between the IMU and the camera. This chapter prepares the sensors for experiments.

Chapter 4

Image Processing and Object Detection

4.1 Overview

The raw image obtained from the camera contains the pixel co-ordinates of all the points in the field of view of the camera. A number of transformations need to be performed to extract the useful features of the landmark.

This chapter details the image processing and object detection algorithms used to identify and obtain the landmark features from an image of the camera video.

Section 4.2 describes the color space transformations performed on the GPU using Open Graphics Library (OpenGL) Embedded Systems (ES) 2.0 textures. The color space transformations isolate the landmark from the background of the image.

Section 4.3 elaborates the process to extract the features of the landmark. The feature extraction is performed in the CPU using Open Computer Vision (OpenCV) standard libraries.

4.2 Image Processing

4.2.1 Landmark Features

The landmark comprises of 4 distinctly colored spherical balls as shown in Fig. 4.2a and their unique features are their respective center points(object points). The location of the object points w.r.t. each other are given and the image points are determined using the captured image. The projection of the sphere onto a image from any orientation is a circle. Hence, the circumference(contour) of the circle in the image can be used to locate the center of the sphere.

4.2.2 Color Space Transformation

The raw sensor data from the Camera is in a compressed YUV (Y-Illumination, UV - chrominance) format. However, the compressed format is not suitable for obtaining the landmark features. The RGB (Red-Green-Blue) pixel co-ordinates (which resemble the human eye perception) can be obtained by linear iterpolation from the compressed YUV. However, the RGB space has both illumination (intensity of lighting) and the chrominance (color) interlaced. So, the RGB image is converted into LAB (L- Lighting, AB- color) color space. The chromaticity (AB) components are used to filter for the color of the landmark (independent of the lighting). This image in LAB space allows us to extract the 4 distinctly colored spherical balls. With the knowledge of the landmark colors, the color filter separates the spherical balls and color codes them.

4.2.3 GPU based processing

A color space conversion involves obtaining the data of each pixel point and converting from one color space to another. However, it a computationally expensive to run it on the CPU as the process would be serially computed one pixel after the other. The GPU enables parallel processing of all the pixel points using millions of threads and hence is approximately 10 times faster than the CPU for each color space conversion, which enables real time color space conversion at 10 Hz.

The operations on the GPU are executed with the help of OpenGL Shading Language (GLSL). The vertex shader specifies the co-ordinates of the pixels and the fragment

shader imparts the color to the pixels. The color space transformation and sequential filters are carried out in a layered fashion i.e. each operation is performed sequentially on the previous layer. A sample example of a simple GPU shader is shown in Appendix D. A flowchart of the image processing sequence is shown in Fig. 4.1 and a sample image undergoing the transformation through the sequence is shown in figure 4.2.

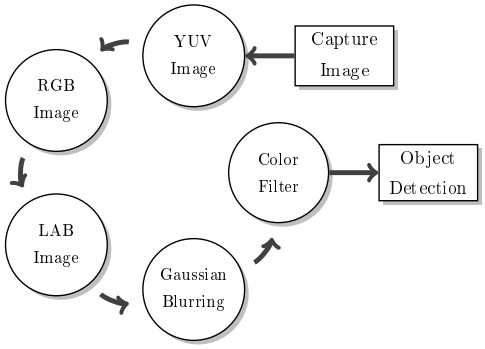
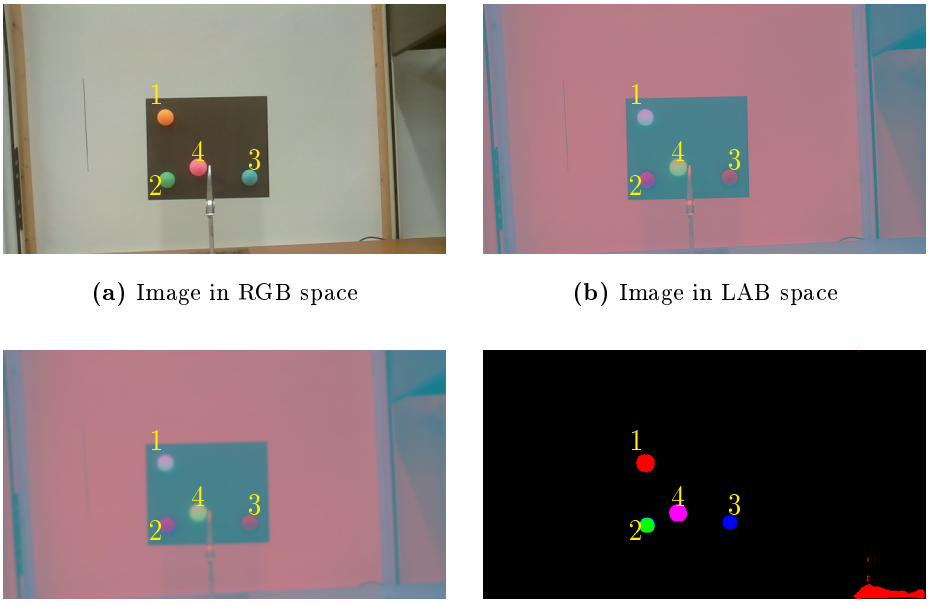


Figure 4.1: Flowchart - Image Processing on GPU



(a) Image in RGB space

(b) Image in LAB space

(c) Gaussian Smoothing

(d) Color Fltered Image

Figure 4.2: Image Processing Sequence

4.2.4 Color Space Conversion

The raw sensor data is obtained from the camera in the compressed camera format of Y'UV420 (chromaticity bandwidth reduced by 50%). The Y-Luma U,V- chrominance (YUV) image space is sequentailly converted to Red Green Blue (RGB) space and eventually to the L-Lightness, A,B-chromaticity (LAB) space [47], to perform feature detection using color filtering. The equations representing the color space conversion from YUV to LAB are shown in (4.1) -(4.7). A sample RGB image (4.2a) transformed into LAB space (4.2b) is shown in Fig. 4.2.

$$\begin{aligned} R &= Y + 1.370705(V - 0.5) \\ G &= Y - 0.698001(V - 0.5) - 0.337633(U - 0.5) \\ B &= Y + 1.732446(U - 0.5) \end{aligned} \tag{4.1}$$

$$f(k) = \begin{cases} (\frac{k+0.055}{1.055})^{2.4}, & \text{if } k \geq 0.0405 \\ \frac{k}{12.92}, & \text{otherwise} \end{cases} \tag{4.2}$$

$$\begin{aligned} X &= 0.4124f(R) + 0.3576f(G) + 0.1805f(B) \\ Y &= 0.2126f(R) + 0.7152f(G) + 0.0722f(B) \\ Z &= 0.0193f(R) + 0.1192f(G) + 0.9505f(B) \end{aligned} \tag{4.3}$$

$$X_0 = 0.95047 \quad Y_0 = 1.00000 \quad Z_0 = 1.08883 \tag{4.4}$$

$$x = \frac{X}{X_0} \quad y = \frac{Y}{Y_0} \quad z = \frac{Z}{Z_0} \tag{4.5}$$

$$g(k) = \begin{cases} k^{\frac{1}{3}}, & \text{if } k \geq 0.008856 \\ 7.787k + 0.137931, & \text{otherwise} \end{cases} \tag{4.6}$$

$$\begin{aligned} L &= \begin{cases} 1.1485\, g(y) - 0.1584, & \text{if } g(y) \geq 0.008856 \\ 8.94356\, g(y), & \text{otherwise} \end{cases} \\ a &= 2.2644(g(x) - g(y)) + 0.5 \\ b &= 0.904977(g(y) - g(z)) + 0.5 \end{aligned} \tag{4.7}$$

4.2.5 Gaussian Filtering

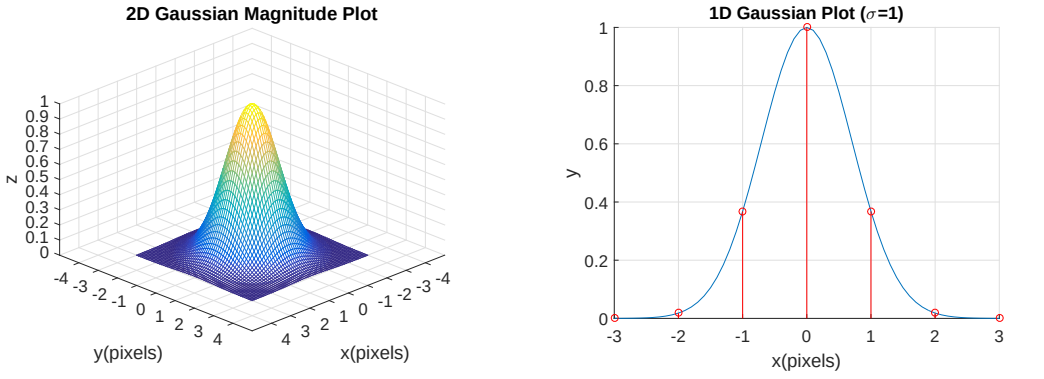
A spatial Gaussian filter is utilized to smooth the colored regions while leaving the edges unaltered. The Gaussian filter applied to the image in the LAB space is shown in figure 4.2c.

The Gaussian filter shown in (4.8) is a circularly symmetric function, hence the 2-D spatial filter can be generated using 2 separate 1-D filters in x and y dimensions. The Gaussian filter is implemented in the x-direction first followed by the y-direction rather than as a 2D filter to reduce computation time.

$$H(i,j) = \frac{1}{2\pi\sigma_x\sigma_y} e^{-\frac{(i-H/2)^2}{2\sigma_x^2} - \frac{(j-W/2)^2}{2\sigma_y^2}} = \underbrace{\frac{1}{\sqrt{2\pi}\sigma_x} e^{-\frac{(i-H/2)^2}{2\sigma_x^2}}}_{H_x(i,j)} \underbrace{\frac{1}{\sqrt{2\pi}\sigma_y} e^{-\frac{(j-W/2)^2}{2\sigma_y^2}}}_{H_y(i,j)} \tag{4.8}$$

The frequency response of a 1-D Gaussian filter is shown in figure 4.3. The Gaussian filter acts as a low pass filter and removes the high frequency components in the image. The magnitude response of the Discrete Fourier Transform (DFT) of the LAB image (refer Fig. 4.2b) and the LAB Gaussian image(refer Fig. 4.2c) are shown in figure 4.4.

The Gaussian distribution is prominent till 3σ after which it decays to 0. Thus, we use the filter values till 3σ in x and y dimension of the image.

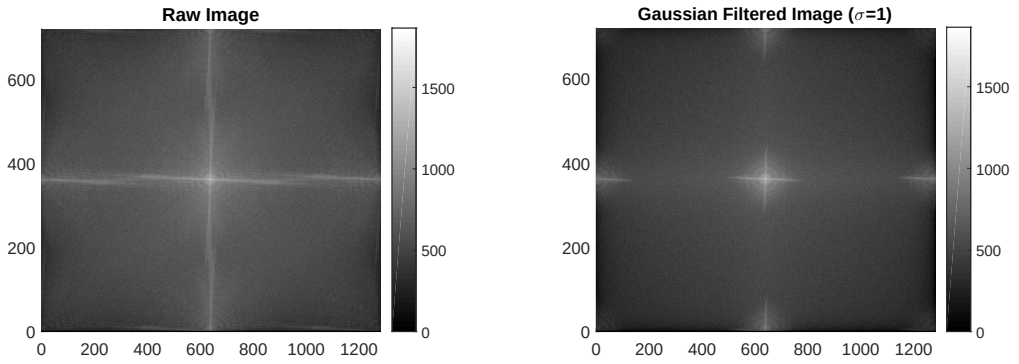


(a) 2-D Gaussian $\sigma = 1.0$ **(b)** Spatial 1-D Gaussian Filter with $\sigma = 1.0$

Figure 4.3: Gaussian Filter

Pixel Location	-3	-2	-1	0	1	2	3
Filter Value	0.0044	0.0540	0.2420	0.3991	0.2420	0.0540	0.0044

Table 4.1: Co-efficients of Discrete 1-D Gaussian Filter($\sigma = 1.0\, pixel$)



(a) DFT of image 4.2b in LAB space

(b) DFT of image 4.2c in LAB space with gaussian filter

Figure 4.4: Schematic - DFT of Images for Gaussian Filter Selection

4.2.6 Color Filtering

A simple least squares based color filter is implemented on the processed image (LAB color space). The color filter checks for the l_2-norm of the difference in AB co-ordinates (error) between the current pixel color and the color of each of the standard landmark spheres. Let $P'(i,j) \in R^3$ represent the LAB data of the current pixel (i,j) and let C_k represent the LAB data of the landmark k , the error of the current pixel is computed w.r.t. each of the landmark colors $k \in (1,4)$.

The landmark colors are now color coded with standard colors on the GPU (like Red (1) Green(2) Blue(3) Pink(4)) for convenience. This step could has been added so that the processing on the CPU is independent of the landmark color used. A specific color $P(i,j)$ is assigned to the current pixel based on the error as shonwn in (4.9)-(4.10). The image obtained from the color filter is shown in Fig. 4.2d.

The error w.r.t. each landmark color is different based on the chromaticity variation. The brighter colors like pink should have less variation in error because of its closeness to

white light.The maximum color difference for each landmark color are $E_{max,i}, i \in [1,4]$

$$e_k(i,j) = (P'(i,j).a - C_k.a)^2 + (P'(i,j).b - C_k.b)^2 \tag{4.9}$$

$$P(i,j) = \begin{cases} (1,0,0), & Red, & \text{if } e_1(i,j) \leq E_{max1} = 0.1 \\ (0,1,0), & Green, & \text{else if } e_2(i,j) \leq E_{max2} = 0.1 \\ (0,0,1), & Blue, & \text{else if } e_3(i,j) \leq E_{max3} = 0.1 \\ (1,0,1), & Pink, & \text{else if } e_4(i,j) \leq E_{max4} = 0.05 \\ (0,0,0), & Black, & \text{otherwise} \end{cases} \tag{4.10}$$

4.3 Object Detection

The image processing and color space conversion isolate the landmark from the background. This section details the process of the detection of the center points of the spheres (image points) which comprise the landmark. This is accomplished with the help of contours using OpenCV library. A flowchart of the object detection operation is shown in Fig. 4.5 and the object detection algorithm applied on a sample landmark is shown in Fig. 4.6.

4.3.1 Grayscale Image Conversion

The Target Image is converted into grayscale for for further analysis. The conversion equation used for conversion from the color coded RGB image (4.6a) to the Grayscale image (4.6b) is shown in (4.11). The opencv function "cvtcolor" is used for this conversion.

$$gray = 0.299 * R + 0.587 * G + 0.114 * B \tag{4.11}$$

By default, the Region of Interest (ROI) is the entire image (1280×960). However, if the landmark is detected in the previous image, the ROI is reduced to a rectangular boundary with a maximum of $ROI_{len} = 50$ pixel points from the nearest landmark points. The ROI reduces the computation time as we are only interested in our landmark features and not the entire image.

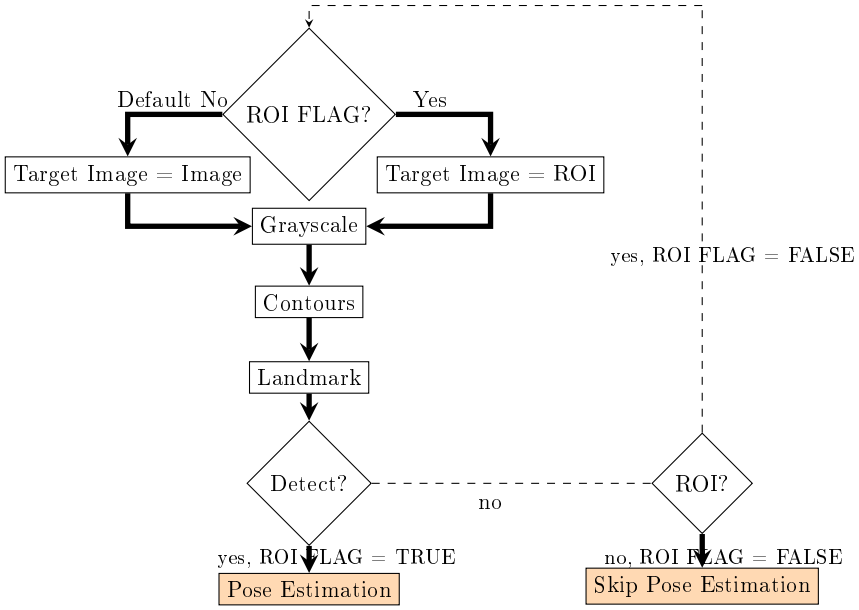
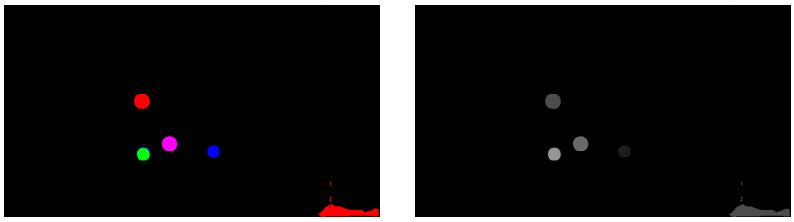
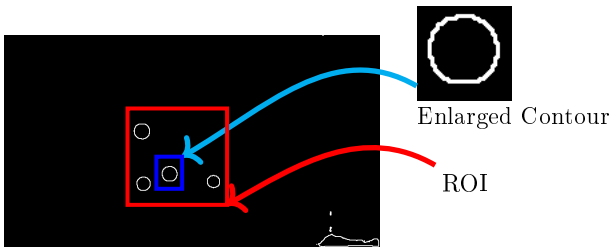


Figure 4.5: Flowchart - Object Detection in CPU



(a) RGB color coded Image from GPU **(b)** Grayscale Image



(c) Contours

Figure 4.6: Landmark detection process in a sample image

4.3.2 Contours

The grayscale image is converted to a binary image to locate the center points. The edges are detected using a Laplacian of Gaussian (LOG) function, which converts the grayscale image to a binary image. The edges represent the circumference of the circle (projection of a sphere). The center of the circle can be determined from the circumference. A sample of a contour is shown in figure 4.7.

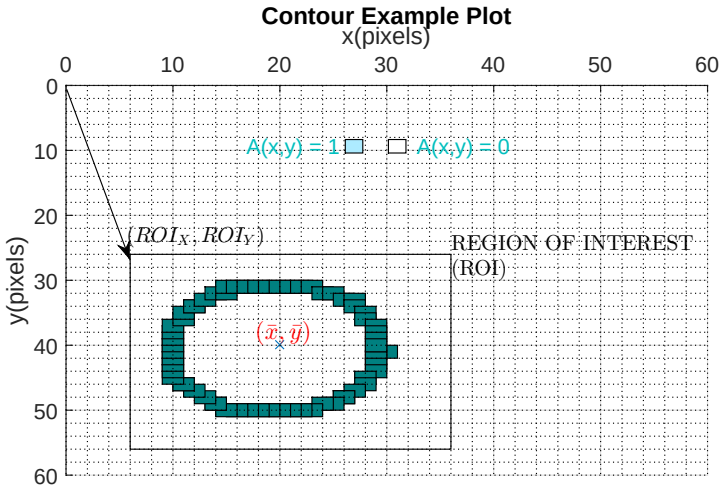


Figure 4.7: Example of a circular contour

4.3.3 Detection of Center Point of a Contour

A contour is a closed chain of pixel points in the binary image. The circumference of edge points comprise a contour which will be used to estimate the center of the circle. Hence, we define the binary image of edges as shown in (4.12).

$$A(x,y) = \begin{cases} 1, & \text{if pixel} = \text{edge point} \\ 0, & \text{if pixel} \neq \text{edge point} \end{cases} \tag{4.12}$$

In this particular landmark, the center point of a sphere with known particular colors are the feature points. Hence, all possible sphere centers in the binary image are potential landmark feature points.

The center point of a spherical contour is obtained by finding the center of mass of the detected contour points. The spatial moments μ_{ji} with order (j, i) in the (x, y) pixels system of the pixel points $A(x, y)$ is given as shown in (4.13)

$$\mu_{ji} = \sum_{x,y} A(x, y) x^j y^i \tag{4.13}$$

The center of mass of the sphere gives the estimate of the center of the circular contour. In the event that a smaller Region of Interest (ROI) of the image is chosen, the center points are shifted to the origin of the base image from the location of the ROI (ROI_x, ROI_y). The center of the circular contour is thus given by (4.14)

$$(\bar{x}, \bar{y}) = \begin{cases} (\frac{\mu_{10}}{\mu_{00}}, \frac{\mu_{01}}{\mu_{00}}), & \text{if ROI = FALSE} \\ (\frac{\mu_{10}}{\mu_{00}} + ROI_x, \frac{\mu_{01}}{\mu_{00}} + ROI_y), & \text{if ROI = TRUE} \end{cases} \tag{4.14}$$

The RGB value of the center points $(\bar{x}, \bar{y})$ are now obtained from the RGB image (4.6a) in Section 4.3.1. The color of each of the potential feature point from the binary image is compared with the color code of the available landmark colors (Red=1, Green=2, Blue=3, Pink=4) to check if it is indeed a landmark point and to relate the detected feature point to the appropriate landmark point.

False Positives

The false positives in the binary image are filtered using the following additional checks:

1. The color is used to remove most of the background noise.
2. A check for a minimum contour area is made to ensure that small noise points are not identified.
3. The current estimate of the pose is compared with the previous estimate to make sure it is within a sufficient range and to avoid false positive features.

4.4 Conclusion

The camera image in the raw YUV format has been processed to extract the features (image points) of the landmark, which are the center points of the spherical balls. The image points obtained in Chapter 4 can be used to compute the pose of the camera **{C}** w.r.t. the landmark **{G}** using the $SO_3 - PnP$ algorithm.

Chapter 5

SO(3) – PnP Pose Estimation Algorithm

5.1 Overview

This chapter details the Pose Estimation algorithm **SO(3) – PnP** used to determine the pose of the camera w.r.t. the landmark. The extracted feature points in pixel co-ordinates from the camera image (chapter 4) are used to determine the pose of the landmark. Section 5.2 describes the model of a pin hole camera, which lays the foundation for the camera measurement equation. The pose estimation is solved as an error minimization problem as shown in Section 5.3. The simulations which validate the proposed algorithm are performed in Section 5.4 while real-time implementation on target hardware is elaborated in Section 5.5. A summary of the performance comparison with state of the art methods is discussed in Section 5.6.

5.2 Camera Model

A schematic of the projection of an object point on the image plane is shown in Fig. 5.1. In Camera Frame {C}, the co-ordinates of each landmark point i is represented by ${}^{C}p_i = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix}^T$. The same landmark point in the Global Frame {G} is given by ${}^{G}P_i = \begin{bmatrix} X_i & Y_i & Z_i \end{bmatrix}^T$ The pixels co-ordinates are obtained from the camera intrinsic

matrix as shown in (5.1).

Using similar triangles and the fact that light travels in straight lines, the object points (3D co-ordinates in real world) and the image points $\begin{bmatrix} u_i & v_i \end{bmatrix}^T$ (2D pixel co-ordinates of image) are related by the camera matrix and the image centers (c_x, c_y)[1] as shown in (5.1).

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \frac{1}{z_i} \begin{bmatrix} f_x x_i \\ f_y y_i \end{bmatrix} + \begin{bmatrix} c_x \approx \frac{W_{img}}{2} \\ c_y \approx \frac{H_{img}}{2} \end{bmatrix} \tag{5.1}$$

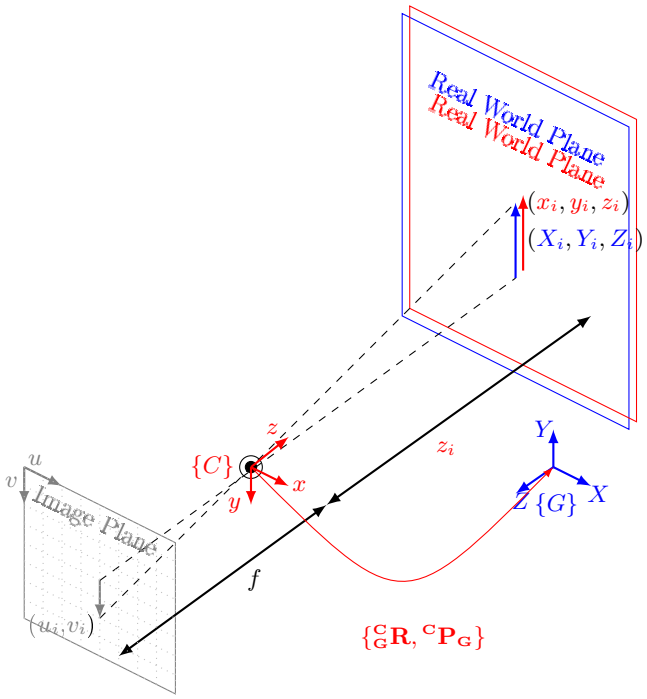


Figure 5.1: Pin Hole Camera Model

Rearranging (5.1) to make it look linear in the unknowns, we obtain (5.2)

$$z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \approx \frac{W_{img}}{2} \\ 0 & f_y & c_y \approx \frac{H_{img}}{2} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \tag{5.2}$$

[1] The dimensions of the image in pixels is given by (W_{img}, H_{img})

The co-ordinates of the landmark points in the global frame {G} are related to the camera frame as shown in (5.3).

$$\underbrace{\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}}_{{}^{C}p_i} = \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}}_{{}^{C}_{G}\mathbf{R}\,[1]} \underbrace{\begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix}}_{{}^{G}P_i} + \underbrace{\begin{bmatrix} {}^{C}x_G \\ {}^{C}y_G \\ {}^{C}z_G \end{bmatrix}}_{\mathbf{{}^{C}P_{G}}\,[2]} \tag{5.3}$$

Subsituting (5.3) in (5.2), we obtain the standard form equation 1.6

$$z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left(\begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X_i \\ Y_i \\ Z_i \end{bmatrix} + \begin{bmatrix} {}^{C}x_G \\ {}^{C}y_G \\ {}^{C}z_G \end{bmatrix} \right)$$

$$z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \left(\left[\begin{array}{ccc|c} r_{11} & r_{12} & r_{13} & {}^{C}x_G \\ r_{21} & r_{22} & r_{23} & {}^{C}y_G \\ r_{31} & r_{32} & r_{33} & {}^{C}z_G \end{array}\right] \begin{bmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{bmatrix} \right)$$

$$\underbrace{z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix}}_{Pixels} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{IntrinsicMatrix} \underbrace{\left[\begin{array}{c|c} {}^{C}R_G & {}^{C}P_G \end{array}\right]}_{ExtrinsicMatrix} \underbrace{\begin{bmatrix} {}^{G}P_i \\ 1 \end{bmatrix}}_{GlobalPosition} \tag{5.4}$$

The standard form (5.4) is non-linear w.r.t. to the rotation matrix and the translation vector. It is complicated to solve for the PnP problem. Hence, we re-arrange (5.1) to get (5.5), which is still non-linear in rotation but is linear w.r.t. to the position.

$$z_i \begin{bmatrix} u_i - c_x \\ v_i - c_y \end{bmatrix} = \begin{bmatrix} f_x & 0 \\ 0 & f_y \end{bmatrix} \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}_i = \underbrace{\begin{bmatrix} f_x & 0 & -(u_i - c_x) \\ 0 & f_y & -(v_i - c_y) \end{bmatrix}}_{\alpha_i} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \alpha_i \, {}^{C}p_i = \alpha_i \left({}^{C}P_G + {}^{C}_{G}R \, {}^{G}P_i \right) \tag{5.5}$$

[1] Rotation of {G} w.rt. {C}
[2] Translation of {G} w.rt. {C}

5.3 Pose Estimation

The PnP problem can be formulated in terms of the object points ${}^{G}P_i$, the matrix of the knowns α_i and the unkonwn pose $({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}, {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}})$of a rigid body/landmark.

The measurement equation with a given pose $({}^{C}_{G}R, {}^{C}P_G)$ (5.5) is shown in (5.6).

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix}_i = \alpha_i \left({}^{C}P_G + {}^{C}_{G}R \, {}^{G}P_i \right) \tag{5.6}$$

The measurement error due to the unknown pose$({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}, {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}})$ as a variable is shown in (5.7)

$$e_i = \begin{bmatrix} e_x \\ e_y \end{bmatrix}_i = \alpha_i \left({}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}} + {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \, {}^{G}P_i \right) \tag{5.7}$$

The error for each landmark point i can be clubbed to form the net error vector as shown in (5.8).

$$e = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} \alpha_1 \, {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}} + \alpha_1 \, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \, {}^{G}P_1 \\ \alpha_2 \, {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}} + \alpha_2 \, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \, {}^{G}P_2 \\ \vdots \\ \alpha_n \, {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}} + \alpha_n \, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \, {}^{G}P_n \end{bmatrix} \tag{5.8}$$

The objective of the problem is to estimate the unknown transformation $\{{}^{\mathbf{G}}_{\mathbf{C}}\mathbf{R}, {}^{\mathbf{G}}\mathbf{P}_{\mathbf{C}}\}$ between the Camera Centered Co-ordinate System (CCCS) and the Real World Co-ordinate System (RWCS). Refering to (5.8), we can formulate the problem as estimating the unknown extrinsic matrix given the matrix of the known and a least squares cost function as presented below.

$$\textbf{Objective:} \quad \min_{{}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \in SO(3), {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}} \in \mathbb{R}^3} \quad J = \frac{1}{2} e^T({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}, {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}}) \, e({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}, {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}})$$

$$\text{given} \qquad {}^{G}P_i \, , \alpha_i$$

The optimization is achieved using the steps outlined below:

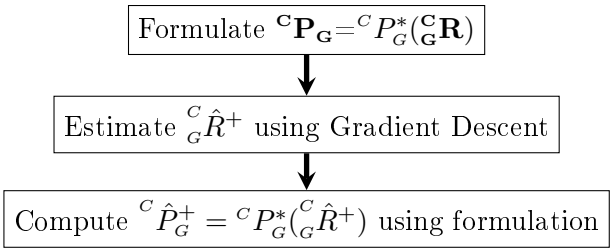


Figure 5.2: Flowchart - SO(3) - PnP Strategy

5.3.1 Formulation of Position Estimate using Least Squares

-

$$\begin{aligned} \textbf{Objective:} \quad & \min_{{}^{\mathbf{C}}\mathbf{P_G} \in \mathbb{R}^3} \quad J = \frac{1}{2}\, e^T\, e \\ & \text{given} \qquad {}^{G}P_i \,,\, \alpha_i \\ & \text{where} \qquad e = e({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}, {}^{\mathbf{C}}\mathbf{P_G}) \end{aligned}$$

The error e is linear w.r.t. the center position ${}^{\mathbf{C}}\mathbf{P_G}$ but is non-linear w.r.t. to the rotation matrix ${}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}$. Simplifying 5.8 results in (5.9)

$$e = \begin{bmatrix} e_1 \\ e_2 \\ . \\ . \\ . \\ e_n \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha_1\, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}\, {}^{G}P_1 \\ \alpha_2\, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}\, {}^{G}P_2 \\ . \\ . \\ . \\ \alpha_n\, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}\, {}^{G}P_n \end{bmatrix}}_{\gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R})} + \underbrace{\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ . \\ . \\ . \\ \alpha_n \end{bmatrix}}_{\alpha^{2}} {}^{\mathbf{C}}\mathbf{P_G} = \gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) - (-\alpha)\, {}^{\mathbf{C}}\mathbf{P_G} \tag{5.9}$$

$$e = \gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) - (-\alpha)\, {}^{\mathbf{C}}\mathbf{P_G} \tag{5.10}$$

Assuming the rotation matrix to be a known variable, we can formulate an optimal estimate ${}^{C}P^{*}_{G}$ in terms of the rotation matrix ${}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}$ based on the linear least squares solution as the error is linear w.r.t. the position. This simplification leads to (5.11).

$${}^{C}P^{*}_{G} = -(\alpha^T\alpha)^{-1}\alpha^T(\gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R})) \tag{5.11}$$

[2] Note: α is not a square matrix and has dimensions $2n \times 3$

Utilizing this optimal estimate for the position in our actual error problem 5.9, we get the modified error $e^* = e^*({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R})$ in terms of a single variable - the rotation matrix.

$$e^* = \gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) - \alpha(\alpha^T\alpha)^{-1}\alpha^T\,\gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) = \underbrace{(I_{2n} - \alpha(\alpha^T\alpha)^{-1}\alpha^T)}_{\beta(\alpha)}\gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) = \beta(\alpha)\,\gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) \quad (5.12)$$

Here $\beta(\alpha)$ is a function of the measured pixels and the camera intrinsic matrix. The optimal rotation matrix ${}^{C}_{G}\hat{R}$ which minimizes this error function $J = \frac{1}{2}e^{*T}e^*$ is computed in an iterative fashion using the Gradient Descent Algorithm.

5.3.2 Estimation of Rotation Matrix using Gradient Descent

$$\begin{aligned} \textbf{Objective:} \quad & \min_{{}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \in SO(3)} && J^* = \frac{1}{2}\,e^{*T}\,e^* \\ & \text{given} && {}^{G}P_i\,,\ \alpha_i\,,\ {}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}} = {}^{C}P^*_G({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) \\ & \text{where} && e^* = e^*({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) \end{aligned}$$

We use the gradient descent algorithm to find an optimal minimum for the updated error function defined in (5.13).

$$J^* = \frac{1}{2}\,e^{*T}\,e^* = \frac{1}{2}\sum_{i=1}^{n} e_i^{*T} e_i^* \quad (5.13)$$

The derivative of the cost function due to a differential change in rotation matrix is given by (5.14).

$$\partial J^* = e^{*T}\,\partial e^* = \sum_{i=1}^{n} e_i^{*T}\,\partial e_i^* \quad (5.14)$$

Substituting (5.7) in 5.14, we obtain 5.15

$$\partial J^* = \sum_{i=1}^{n} e_i^{*T}\,\partial(\alpha\,{}^{C}P^*_G + \alpha_i\,{}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}\,{}^{G}P_i) \quad (5.15)$$

The optimal estimate of the position for each choice of the rotation matrix is given by (5.11). Hence, (5.15) simplifies to (5.16)

$$\partial J^* = \sum_{i=1}^{n} e_i^{*T}\,\partial(\alpha_i\,{}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}\,{}^{G}P_i) = \sum_{i=1}^{n} e_i^{*T}\,\alpha_i\,\partial^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}\,{}^{G}P_i \quad (5.16)$$

Since, the rotation matrix ${}^{C}_{G}\mathbf{R} \in SO3$, the differential rotation of the rotation matrix is given by $\partial {}^{C}_{G}\mathbf{R} = \Omega {}^{C}_{G}\mathbf{R}$ s.t. $\Omega = -\Omega^T$ for some skew-symmetric Ω. Thus using vector identity A.1 and matix identity A.2, (5.16) simplifies to (5.17)

$$\partial J^* = \sum_{i=1}^{n} e_i^{*T} \alpha_i \mathbf{\Omega} {}^{C}_{G}\mathbf{R}\, {}^{G}P_i = \sum_{i=1}^{n} tr(\alpha_i \mathbf{\Omega} {}^{C}_{G}\mathbf{R}\, {}^{G}P_i e_i^{*T}) = tr(\underbrace{\sum_{i=1}^{n} {}^{C}_{G}\mathbf{R}\, {}^{G}P_i\, e_i^{*T}\, \alpha_i}_{N({}^{C}_{G}\mathbf{R})}\, \mathbf{\Omega}) \quad (5.17)$$

Simplifying further (5.17), we get (5.18)

$$\partial J^* = tr(N({}^{C}_{G}\mathbf{R})\mathbf{\Omega}) \quad (5.18)$$

Decomposing the matrices further would simplify the solution, i.e. let us assume that the matrices N and $\mathbf{\Omega}$ are given as shown below:

$$N({}^{C}_{G}\mathbf{R}) = \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} \; ; \; \mathbf{\Omega} = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

Using the matrix identity A.3, the (5.18) reduces to (5.19)

$$\partial J^* = tr(N({}^{C}_{G}\mathbf{R})\mathbf{\Omega}) = (n_{23} - n_{32})p + (n_{31} - n_{13})q + (n_{12} - n_{21})r \quad (5.19)$$

We choose $\vec{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^T$ for any given $||\vec{\omega}||$, the direction of steepest descent for ∂J as shown in (5.20).

$$\vec{\boldsymbol{\omega}} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = - \begin{bmatrix} (n_{23} - n_{32}) \\ (n_{31} - n_{13}) \\ (n_{12} - n_{21}) \end{bmatrix} \quad (5.20)$$

$$\vec{\boldsymbol{k}} = \frac{\vec{\boldsymbol{\omega}}}{||\vec{\boldsymbol{\omega}}||} \quad (5.21)$$

$$\mathbf{\Omega^*} = [\vec{\boldsymbol{k}}]_{\times}{}^{3} \quad (5.22)$$

A schematic of the estimaton is shown in Fig. 5.3. After estimating the rotation axis $\vec{k}$, the rotation angle θ is chosen in known steps with a given initial stepsize, say $\theta_0 = \frac{\pi}{72}$.

[3] Skew symmetric matrix from $\vec{\boldsymbol{k}}$

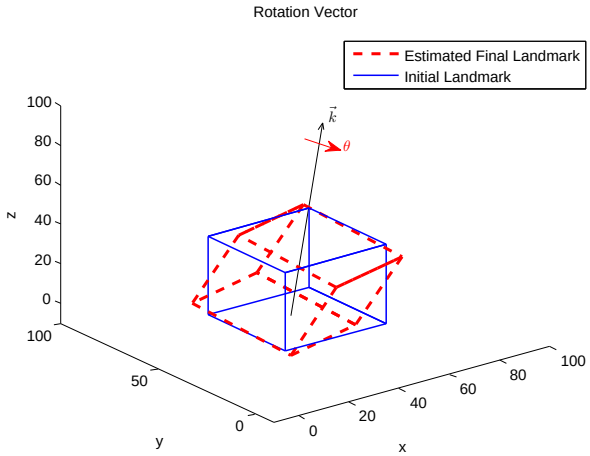


Figure 5.3: Estimation of Rotation Axis $\vec{k}$ and rotation angle θ

The new estimate of the rotation matrix is obtained from the previous estimate using the above angular rate which minimizes the cost function as shown in (5.23).

$$ {}^{C}_{G}\hat{R}^{+} = e^{\mathbf{\Omega^{*}}\theta}\hat{R}^{-} \tag{5.23} $$

After this differential rotation, we compute the maximum angle of rotation along the same rotation axis which would still minimize the cost function using the golden section search.

When the cost function converges w.r.t. the angle θ, the new rotation axis at this state is computed using (5.19). The process is repeated till the time the cost function decreases.

The gradient descent algorithm thus converges and leads to an optimal estimate of the rotation matrix.

5.3.3 Position Estimation

The formulation of the position estimate in terms of the rotation matrix is used to estimate the position using (5.24).

$$ {}^{C}\hat{P}^{+}_{G} = -(\alpha^{T}\alpha)^{-1}\alpha^{T}(\gamma({}^{C}_{G}\hat{R}^{+})) \tag{5.24} $$

5.3.4 Transformation to Global frame

The pose of the landmark w.r.t. the camera is obtained in the above steps. However, the pose of the camera w.r.t. the landmark is of interest to track the ROV and hence

we shift the co-ordinate system as described in Section 2.6.

$$\begin{aligned} {}^{G}_{C}\hat{R}^{+} &= {}^{C}_{G}\hat{R}^{+T} \\ {}^{G}\hat{P}^{+}_{C} &= -{}^{C}_{G}\hat{R}^{+T}\,{}^{C}\hat{P}^{+}_{G} \end{aligned} \tag{5.25}$$

5.3.5 Summary

A summary of the algorithm is presented in 4.

Algorithm 4: Summary of $\mathbf{SO(3) - PnP}$ Algorithm

Steps:

I Formulate the position ${}^{\mathbf{C}}\mathbf{P}_{\mathbf{G}}$ as a function of the rotation matrix ${}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}$

II Estimate the rotation matrix using the measured pixels iteratively

1. Use a guess value of rotation matrix ${}^{C}_{G}\hat{R}^{-}$
2. Compute the euler rotation axis $\vec{k}^{i}$, rotating along which the error decreases the maximum
3. Use binary search to estimate the maximum rotation angle θ, which will cause the error to still decrease
4. Iterate steps 2-3 till convergence which is gradient descent in SO(3) space

III Use formulation in I to estimate the position ${}^{C}\hat{P}^{+}_{G}$ from the estimated rotation matrix ${}^{C}_{G}\hat{R}^{+}$

IV Compute the pose in the the global frame {G} i.e. compute ${}^{G}_{C}\hat{R}^{+} = {}^{C}_{G}\hat{R}^{+T}$, ${}^{G}\hat{P}^{+}_{C} = -{}^{C}_{G}\hat{R}^{+C}\hat{P}^{+}_{G}$

V Obtain the quaternion from the rotation matrix $q_{cam} = q({}^{G}_{C}\hat{R}^{+})$

5.4 Simulation in Matlab

The $\mathbf{SO(3) - PnP}$ algorithm was simulated in Matlab with the addition of random gaussian noise to check the functionality and accuracy of the algorithm.

A fixed landmark was assumed with the following co-ordinates (cm) in the global frame/ landmark frame.

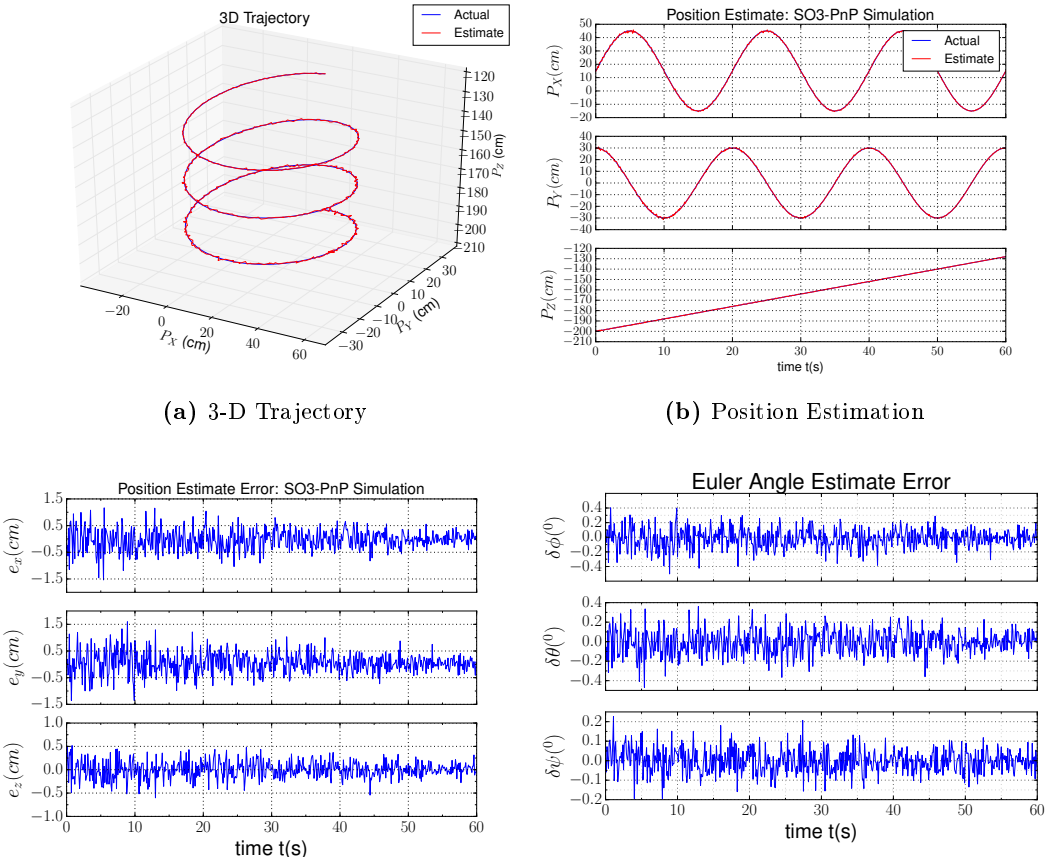
Marker	x	y	z
1	0	-15	0
2	0	0	0
3	20	0	0
4	5	5	-10

Table 5.1: Marker Co-ordinates in Global Frame used for Simulation

The landmark is assumed to be stationary and the camera is tranformed by a known pose, which changes in the form of a spiral. The camera focal length $f_x = f_y = 1000$ and an image dimension $W_{img} = 1280, H_{img} = 720$ were assumed.A random Gaussian noise is added to the pixel data to simulate real world noise($\sigma = 0.30 pixels, \mu = 0$) from the camera. The pose estimate from the camera is compared with the actual pose.

The translation error is of the magnitude of $\pm 0.5cm$ at a distance of $|d| = 2m$ and the rotation error in Euler angles is of the magnitude of $\pm 0.5^0$ which validates the requirements of the algorithm.

As the magnitude of the random Gaussian noise(σ) is increased, the error increases rapidly. Hence, an accurate measurement of the marker points (minimum noise) is required to ensure good pose estimate from computer vision.



(a) 3-D Trajectory

(b) Position Estimation

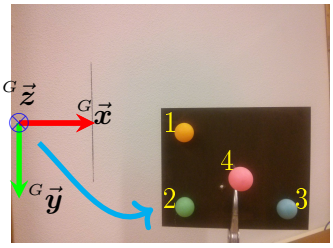
(c) Position Error

(d) Euler Angle Estimate

Figure 5.4: Simulation- Gradient Descent Algorithm (Actual Pose vs Estimated Pose)

5.5 Experiments

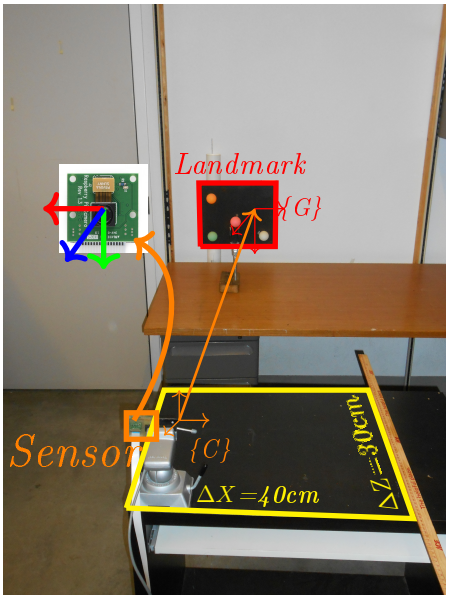
An experiment was performed to validate the gradient descent algorithm. A rectangular trajectory(40×30 cm) is traced by the vision system which consists as shown in Fig. 5.5 . A plot of the of the estimated motion trajectory of the camera system is shown in Fig. 5.7. The landmark is located approximately at a distance of 1 m. from the camera system.



(a) Schematic of Landmark

Point	x	y	z
1	0	-15	0
2	0	0	0
3	20	0	0
4	10	-8	-10

(b) Landmark Co-ordinates (cm)



(c) Trajectory Schematic

Figure 5.5: Landmark - Camera Marker

A schematic of the trajectory used for error analysis is shown in Fig. 5.6. The range of motion along each side of the rectangles is obtained using experiments. The mean

distance along each side of the rectangle is compared with the actual trajectory length ($\Delta X = 40cm, \Delta Z = 30cm$) to obtain an estimate of the error.

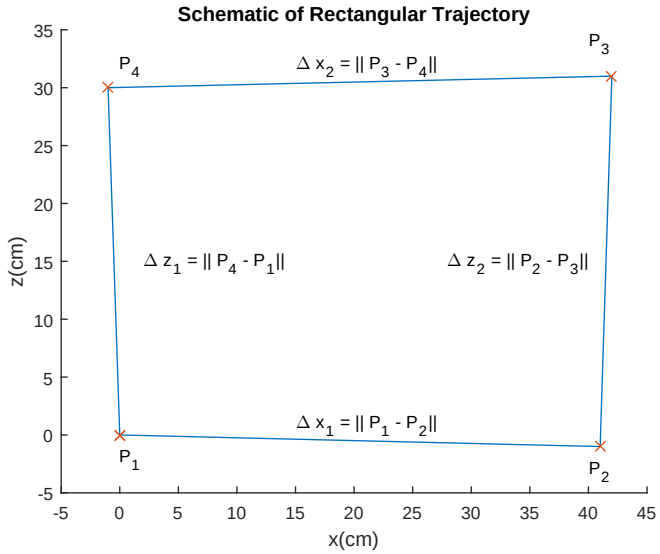
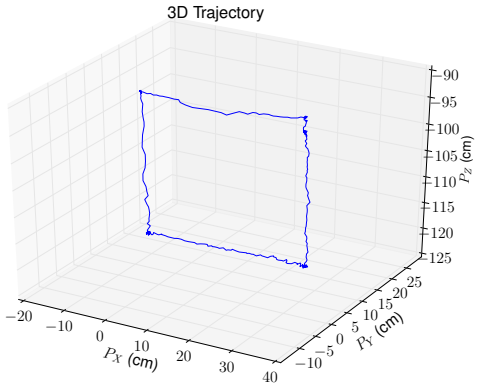


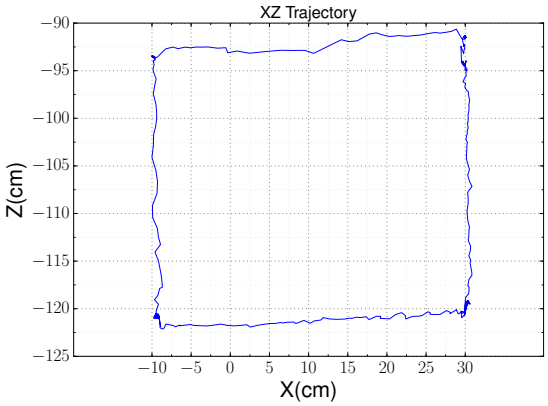
Figure 5.6: Schematic of Trajectory used for error analysis

The rectangular trajectory lengths (Fig. 5.6) as obtained from the experiments (5.7b-5.7c) are $\Delta x = \frac{\Delta x_1 + \Delta x_2}{2} = \frac{40.1+39.715}{2} = 39.91cm; \Delta z = \frac{\Delta z_1 + \Delta z_2}{2} = \frac{40.1+39.715}{2} = \frac{29.89+29.59}{2} = 29.74cm$. The x-axis deviation is slightly lower than the z-axis deviation($\Delta e_x = |39.91 - 40| = 0.09cm; \Delta e_z = |29.74 - 30| = 0.26cm$).

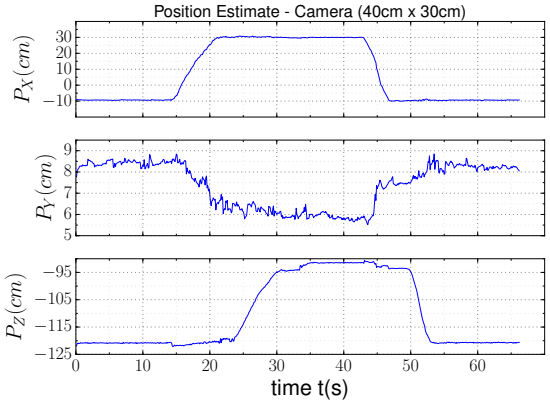
The absolute Euler angles representing the orientation of the cam-imu system in the global co-ordinates are shown in Fig. 5.7d. An approximate deviation of +/-2.5 degrees in Euler angles is seen. This may be due to the non-uniform movement of the sensor board.



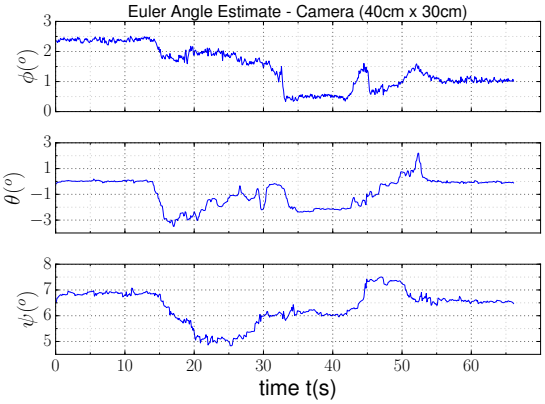
(a) 3-D Trajectory in Global co-ordinates



(b) 2-D Trajectory in Global co-ordinates



(c) Position vs Time plot



(d) Absolute Euler Angle plot

Figure 5.7: Experimental Result - Trajectory Tracking using Camera

Hence, at a distance of about 1m, the error is approximately about +/-0.1 cm in the x-y axes and about +/- 0.25cm in z-axes, which validates the $\mathbf{SO(3) - PnP}$ algorithm.

5.6 Algorithm Comparison

In this section, we compare our $\mathbf{SO(3) - PnP}$ algorithm with the state of the art algorithms. The simulations were performed with an image size of $(640 \times 480 pixels)$, a focal length of $(f = 800 pixels/cm)$. The landmark points were chosen as uniform random 3D points in the camera co-ordinate system within the range $\pm x = \pm y = \pm z = 100cm$. A random gaussian noise given by $\sigma = 3\,pix, \mu = 0$ is then added to the projections in the simulated camera image. The simulations for performance comparison were performed in Matlab. The mean errors in translation(distance percentage) and rotation(angle between estimated and actual) for the algorithms is shown in Fig. 5.8.

About 100 iterations of the experiment were performed for a given known pose and the error with the actual pose is compared for each of the algorithms. The key statistics of interest are the mean error in translation and rotation as well as the computation time required to obtain the pose.

The cost function of the DLS[25] (refer Section 1.3.1) and SO(3)-PnP is the same linear least squares and hence, they converge to the same result. However for small number of points n, the SO(3)-PnP outperforms DLS-PnP in speed.

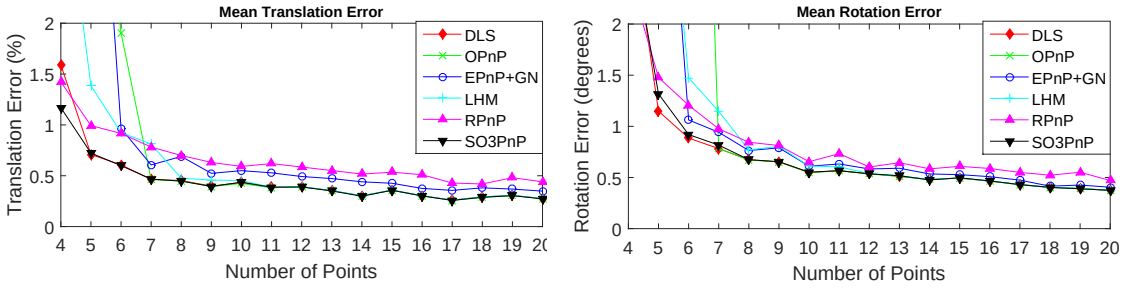
LHM[31](refer Section 1.3.1 is an iterative algorithm which utilizes expensive repeated SVD computations and hence has larger computation time than SO(3) - PnP.

E-PnP[27](refer Section 1.3.1 uses 4 optimal points referred to as critical points based on the generic n points and computes pose based on these critical points. As E-PnP uses less control points, it requires less computation time but a smaller noise could lead to larger deviations in pose.

O-PnP[36] solves the pose estimation problem using orthogonal iterations. It is quite similar to the SO(3)-PnP but uses a non-linear cost function.

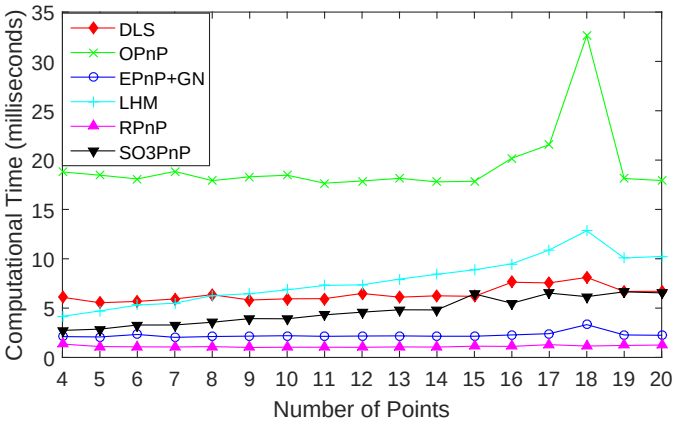
R-PnP [29](refer Section 1.3.1 uses the property that the distance between any two points in global co-ordinates remains the same at any viewing angle of the camera. The method uses linear approximations and is quite different from the methods discussed so far.

The results are shown in Fig.5.8. The $\mathbf{SO(3) - PnP}$ algorithm performs very well for small n $(4 \leq n \leq 20)$, in terms of both accuracy and computation time.



(a) Mean Translation Error

(b) Mean Rotation Error



(c) Mean Time

Figure 5.8: Comparison of $\mathbf{SO(3)-PnP}$ with state of the art PnP algorithms

5.7 Conclusion

Given the camera matrix, the object points (3D) and their corresponding image points (2D) the SO(3)-PnP algorithm has been used to compute the pose of the moving camera w.r.t. a stationary landmark. The theory behind the estimation algorithm has been discussed in detail. Similarities and comparisons with a few state of the art algorithms has been outlined. The proposed algorithm for pose estimation $\mathbf{SO(3) - PnP}$ using computer vision compares well w.r.t. several state of the art algorithms. As the camera measurements are slow, the IMU sensor can be used to smooth the camera measurements and achieve optimal pose estimation in real time.

Chapter 6

Position and Orientation Estimation using IMU

6.1 Overview

The MEMS IMU sensor comprises of 3-axis accelerometers and 3-axis gyroscopes. The sensors have a very high update frequency (typically 200 Hz). They provide a very good estimate of the orientation and gives a fairly good estimate of the velocity but the position estimate drifts and is not very accurate due to double integration of noisy accelerometer sensor data. However, in a short interval of time (typically 1s), the IMU can be used to estimate pose. The procedure for estimating pose using IMU is elaborated in this chapter. The MEMS IMU sensor comprises of 3-axis accelerometers and 3-axis gyroscopes. The sensors have a very high update frequency (typically 200 Hz). They provide a very good estimate of the orientation and gives a fairly good estimate of the velocity but the position estimate drifts and is not very accurate due to double integration of noisy accelerometer sensor data. However, in a short interval of time (typically 1s), the IMU can be used to estimate pose. The procedure for estimating pose using IMU is elaborated in this chapter.

This chapter is dedicated entirely to the IMU and pose estimation using IMU. The rest of this Chapter follows the outline described below. Section 6.2 details the orientation estimate from the gyros. The filter design for the accelerometers and the position estimation from the accelerometers is described in Section 6.3. Section 6.4 compares the

performance of the IMU with encoders. The results and conclusions of the experiments are summarized in 6.5.

6.2 Orientation Estimation - Theory

The gyroscopes provide the angular velocity in the IMU frame $\{\mathbf{I}\}$, which is converted to the camera frame $\{\mathbf{C}\}$ using the pre-calibrated orientation matrix ${}^{C}_{I}R$. In continuous time, the rate of change of orientation using the measured gyroscope data in Camera Frame {C} ${}^{C}\vec{\boldsymbol{\omega}} = \begin{bmatrix} p & q & r \end{bmatrix}^{T}$ is given by (6.1)

$$ {}^{G}\dot{q} = \frac{1}{2}\begin{bmatrix} 0 & -p & -q & -r \\ p & 0 & r & -q \\ q & -r & 0 & p \\ r & q & -p & 0 \end{bmatrix} {}^{G}q = \frac{1}{2}\Omega \, {}^{G}q \tag{6.1} $$

Discretizing (6.1), we get (6.2).The quaternion thus obtained must be normalized to ensure it is stll a quaternion as shown in (6.3). The time interval is typically of the range of $\delta t = 5ms$.

$$ {}^{G}q(t+\delta t) = {}^{G}q(t) + \frac{1}{2}\Omega \, {}^{G}q(t)\delta t \tag{6.2} $$

$$ {}^{G}q(t+\delta t) = \frac{{}^{G}q(t+\delta t)}{||{}^{G}q(t+\delta t)||} \tag{6.3} $$

The initial orientation of the IMU in the global co-ordinates is obtained from camera i.e. ${}^{G}q(t_0) = \hat{q}_{cam}$

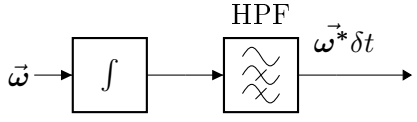


Figure 6.1: Block Diagram - Gyroscope Filter

The block diagram of the high pass filter used to remove the static gyroscope bias is shown in Fig. 6.1. A very low cutoff frequency ($f = 1Hz$) was used.

6.3 Position Estimation - Theory

The accelerometer provides the current acceleration experienced by the MEMS sensor. Twice integrating the acceleration in the Global frame $\{G\}$, we get the absolute position. The accelerometer reads a combination of the Earth's static gravity ($g = 9.81 m/s^2$) and the acceleration of the sensor/ROV. However, due to twice integration, a small sensor noise and bias or error due to gravity compensation in the acceleration can cause a huge deviation in the position. The key to achieving a good position estimate would be to remove the static component of gravity and to remove any sensor noise.

This can be achieved with the help of a band pass filter to remove the static component. In underwater scenarios, the probability of a constant acceleration by the ROV is minimal. Hence, we try to remove the static component of acceleration (primarily gravity) using a high pass filter $f = 0.5Hz$ and the high frequency component due to vibrations using a low pass filter $f = 10Hz$. The filtered acceleration is twice integrated to get the position estimate.

In the Arduino development environment, we use a circular FIFO queue (500 elements long) to store the previous sensor readings ${}^{G}\vec{\boldsymbol{s}} = \left[\vec{\boldsymbol{a}}, \vec{\boldsymbol{\omega}}\right]$ and the state readings ${}^{G}\hat{\vec{\boldsymbol{x}}}_{store} = \left[\hat{\mathbf{q}}, \hat{\vec{\boldsymbol{x}}} = (\hat{\vec{\boldsymbol{v}}}, \hat{\vec{\boldsymbol{p}}})\right]$ to account for the time delay in image processing as well as to implement the discrete filter.

Buffer Index	1	2	3	...	$N_b - 1$	N_b
Acc. Data	$\vec{\boldsymbol{s}}(t_1)$	$\vec{\boldsymbol{s}}(t_2)$	$\vec{\boldsymbol{s}}(t_3)$	...	$\vec{\boldsymbol{s}}(t_{N_b-1})$	$\vec{\boldsymbol{s}}(t_{N_b})$

Figure 6.2: FIFO buffer to store sensor readings

The Band Pass Filter (BPF) is a second order Butterworth filter with the transfer function in the discrete frequency domain given by (6.4). The static component of gravity and the high frequency component of acceleration due to vibrations is removed

by the BPF. The filter is implemented in the manner of a Direct Form-II transposed structure using the Eigen Library and the buffers storing sensor data. The filter closely represents the Matlab's 'filter' function. A block diagram of the filter is shown in Fig. 6.3.

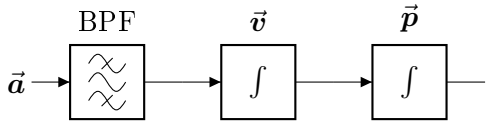


Figure 6.3: Circuit Diagram - Accelerometer filter

$$H(z) = \frac{10^{-3}(0.2276z^{-4} - 0.4551z^{-2} + 0.2276)}{1.0000z^{-4} - 3.9396z^{-3} + 5.8198z^{-2} - 3.8207z^{-1} + 0.9406} \tag{6.4}$$

6.4 Experiments

In order to validate the application of the IMU for position estimation in short intervals of time, the IMU was moved in a straight line trajectory along a smooth rail and the position estimate from the IMU is compared with a magnetic encoder to check for accuracy of motion. A schematic of the test bed is shown in Fig. 6.4. The experiments were performed on a single axis (x-axis $\{\mathbf{G}\}$ for translation) and (z-axis for rotation) using the rotary and linear encoders of the 3-DOF measuring apparatus.

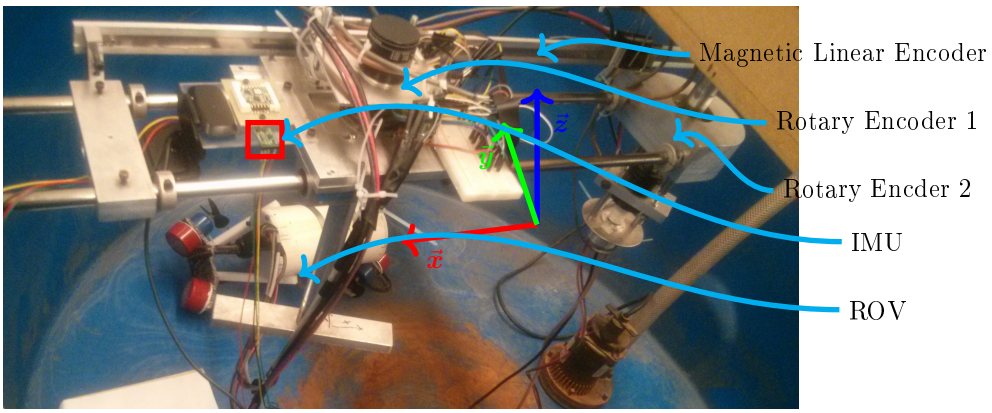


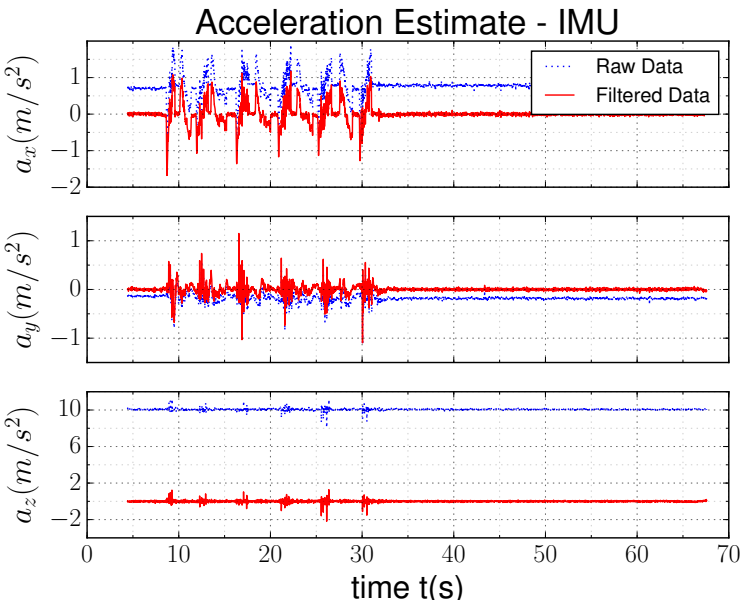
Figure 6.4: Test Apparatus (3-DOF)

6.4.1 Position Estimation

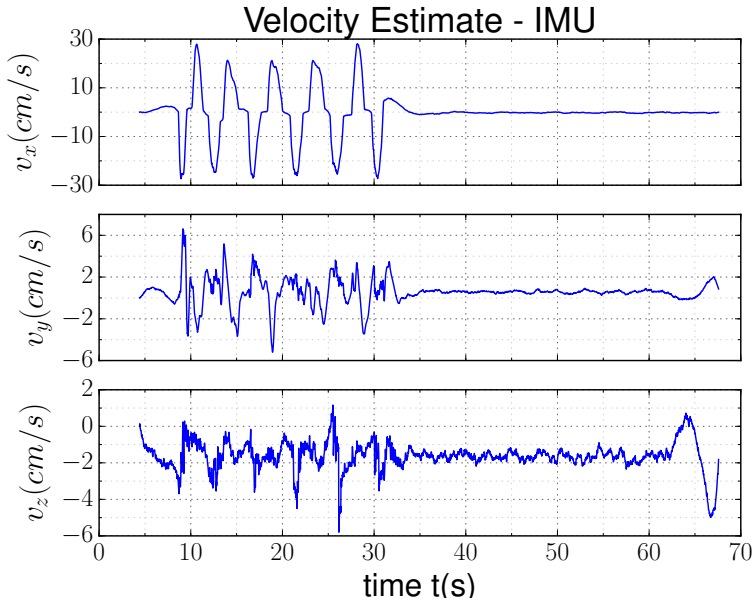
The translation computed by double integration from the IMU is compared with the position recorded by the linear magnetic encoder. The results of the position estimate are shown in Fig. 6.3

The IMU drifts a lot ($\approx t^2$) for small velocities and hence cannot be used alone. In a time period of about $1min.$, the drift can be as large as $|d| > 1m.$. In short time intervals, the drift is not so large ($\pm 5cm.$) as seen in Fig. 6.3f.

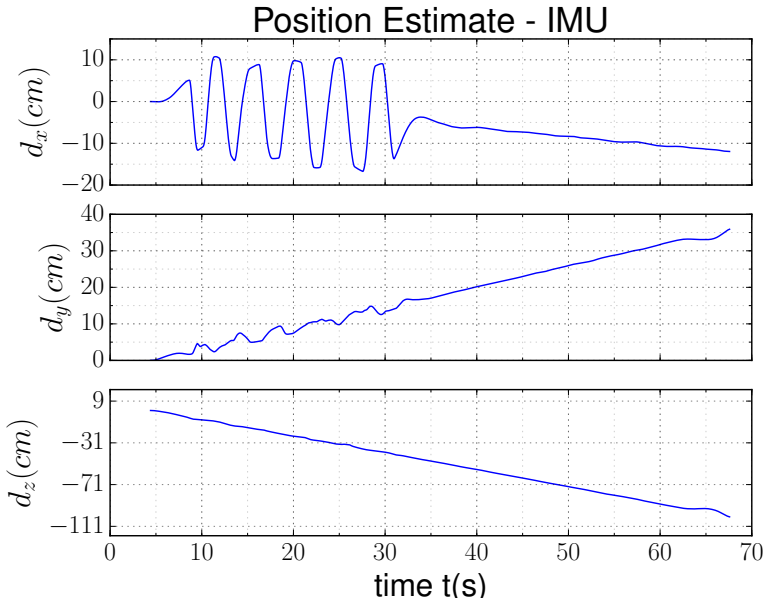
It requires a stable position sensor like a camera to compensate for the drift and to achieve good position estimate.



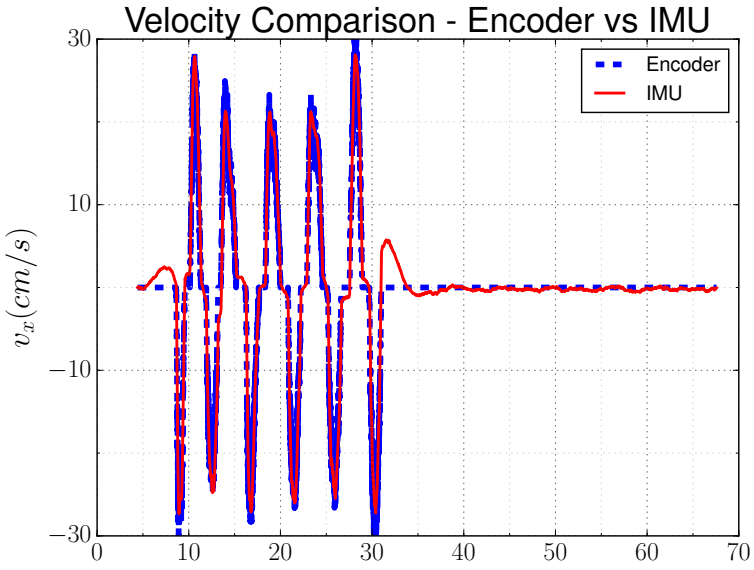
(a) Acceleration Plot



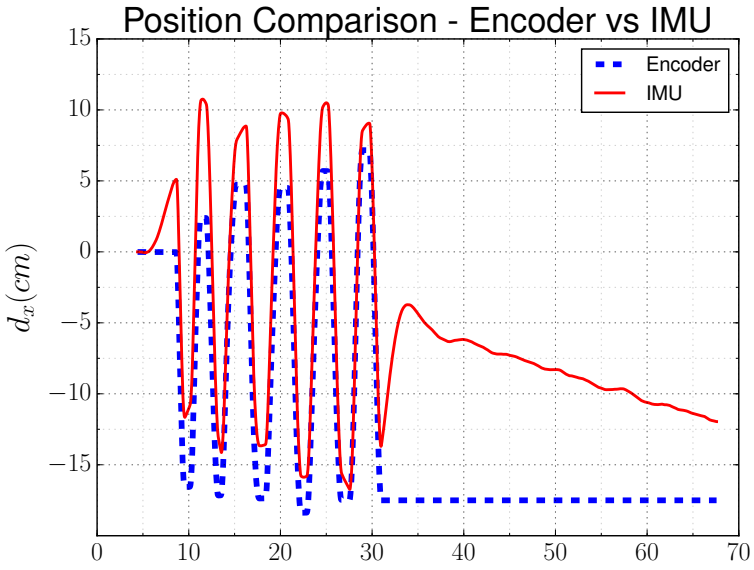
(b) Velocity Plot



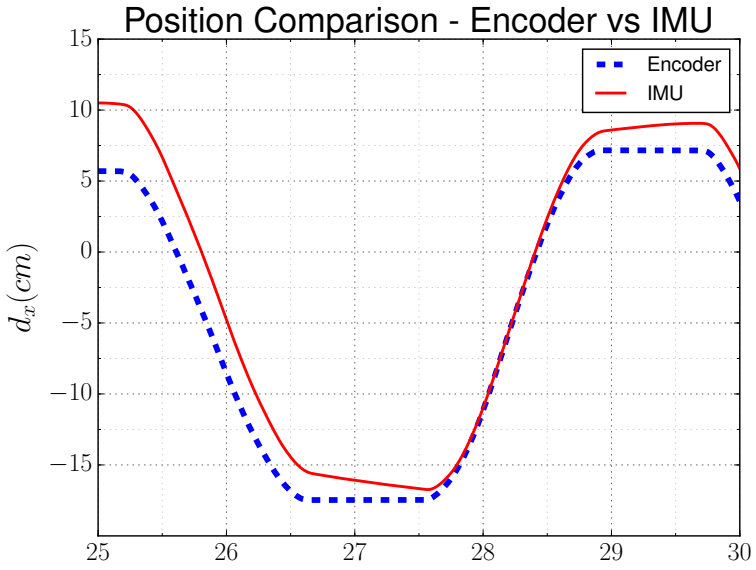
(c) Position Plot



(d) Velocity Plot (x-axis)



(e) Position Plot (x-axis)

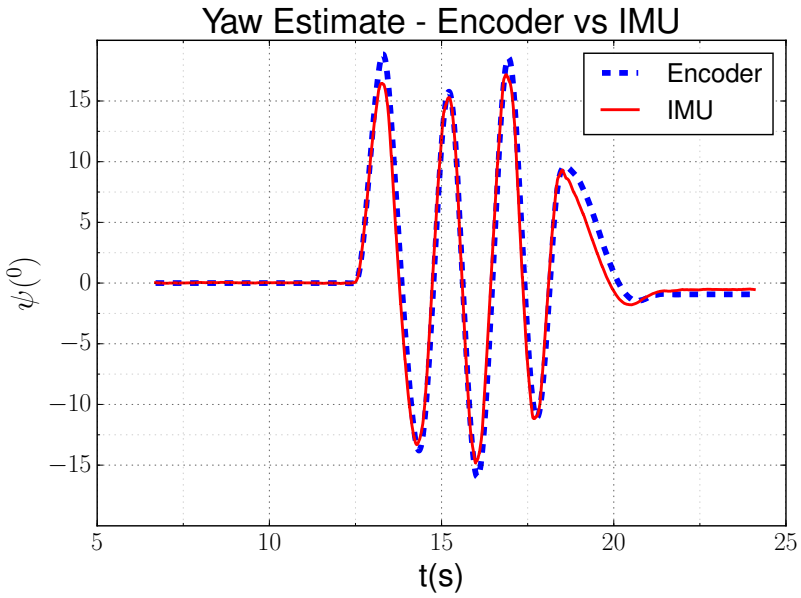


(f) Position Plot (x-axis)

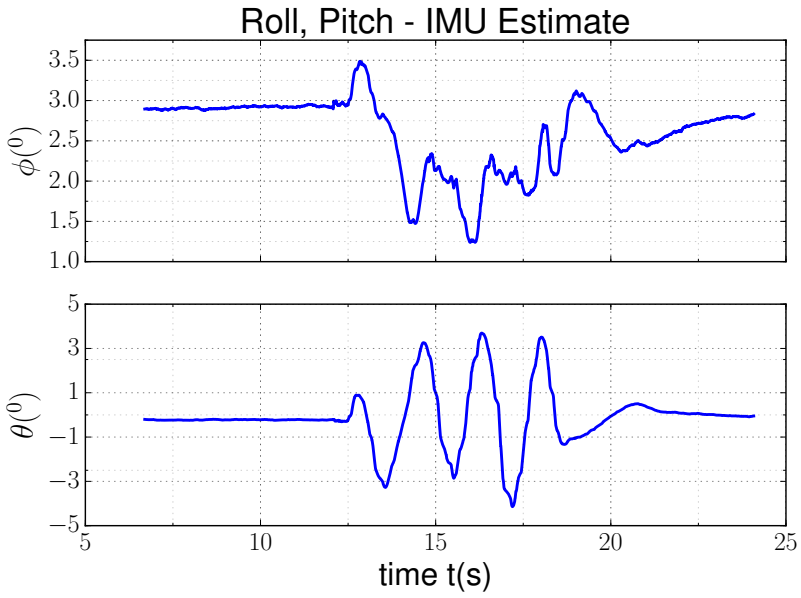
Figure 6.3: Experimental Result - IMU position Estimation on Linear Track

6.4.2 Orientation Estimation

A test was performed using the rotary encoder to check the yaw by rotating the arm. The IMU yaw was compared with the central rotary encoder and a very close match was obtained as shown in Fig. 6.4b. The slight deviation may be caused due to a slack in the arm which causes a difference in the axis of rotation of the test bed and the IMU.The roll-pitch may have slight disturbance if the motion is not uniform as shown in Fig.6.4b. The maximum deviation in peak to peak measurements is of the order of $\pm 1^0$ for yaw. The error in roll and pitch is of the order of $\pm 5^0$.



(a) Yaw Estimate



(b) Roll-Pitch Estimate

Figure 6.4: Experimental Result - Orientation Estimation (Yaw Rotation)

6.5 Conclusion

In this chapter, we present pose estimation using the IMU as a stand-alone sensor. Even though the accuracy of the IMU is not very high, we see the potential to use it for intermediate camera measurements. The position and orientation from the IMU have been compared experimentally with the encoders and an attempt has been made to utilize the IMU to estimate the position and orientation in 2-DOF. However, due to large time varying drift in position readings from IMU, the IMU cannot be used alone. The gyro bias also drifts over a long period of time and requires a compensating measurement sensor to achieve robust pose estimate.

Thus, we see that the IMU alone cannot be used to give an accurate state estimate but fused with a camera, it can be used to increase the accuracy of the state estimate in between the camera readings.

Chapter 7

Optimal Estimation and Sensor Fusion

7.1 Overview

This chapter describes the optimal estimation and sensor fusion using the IMU and Pi-camera. The sensor fusion is carried out using Kalman filters for both the orientation and the position. The theory behind orientaton fusion is discussed in Section 7.3 while the theory behind position fusion is discussed in Section 7.4. The results of sensor fusion are illustrated in Section 7.6. Section 7.7 concludes the chapter with the merits of sensor fusion.

The camera measurement (Chapter 5) is processed to obtain the oritentaion quaternion q_{CAM} and the position vector P_{CAM} in Global frame {G}.

The IMU sensor data with a frequency $f_{imu} = 200Hz$ (Chapter 6) in Global Frame {G} is used in the time update step and integrated with the Kalman Filter. The pose estimate from the camera measurements are too slow ($f_{cam} = 10Hz$) for real time pose estimation and control of underwater robot. Hence, the sensor data from the IMU is added to smooth the pose of the robot for intermediate readings from the camera.

The sensor fusion using Kalman Filtering estimates the optimal estimate for the orientation and position vector based on the two sensor data. The fused position is compared with the trajectory data and the fused orientation is compared with the YEI-IMU (known accuracy of $\pm 0.5^0$).

7.2 Kalman Filter - Overview

This section describes the generic implementation of a discrete time Kalman Filter, which will be used for both position and orientation.

The position fusion resembles a generic linear time invariant (LTI) system and hence a LTI system is used to present the Kalman filter. A system state($\vec{x}$) with known dynamics (F, G) is represented in (7.1).

$$\vec{x}_{k+1} = F\,\vec{x}_k + G\,\vec{u}_k + \vec{w}_k \tag{7.1}$$

However, the state dynamics alone is not enough to accurately determine the system. High process noise and unknown starting point of the system can lead to large errors. Hence, a measurement $\vec{y}$ of the system is made to ascertain the system state at any known regular time intervals and appropriate corrections are made to ensure optimal estimation of the system state.

$$\vec{y}_k = H\,\vec{x}_k + \vec{v} \tag{7.2}$$

A knowledge of the variance ($E(\vec{w}\vec{w}^T) = Q$) of the process noise($vecw$) and the variance ($E(\vec{v}\vec{v}^T)$) of the measurement noise($vecv$) enables the Kalman filter to optimally estimate the system state w.r.t. the process noise and measurement noise. The expected value of variable x is given by $E(x)$ and the initial state is assumed to be x_0. The state co-variance matrix is represented by P_k

$$E(\vec{w}\vec{w}^T) = Q, \quad E(\vec{v}\vec{v}^T) = R \tag{7.3}$$

$$\hat{\vec{x}}_0^+ = E(x_0), \quad P_0^+ = E[(x_0 - \hat{\vec{x}}_0^+)(x_0 - \hat{\vec{x}}_0^+)^T] \tag{7.4}$$

The steps in a discrete Kalman filter [48] can be summarized as shown in Algo. 5

Algorithm 5: Discrete Time Kalman Filter

Given: $E(\vec{w}\vec{w}^T) = Q,\ \ E(\vec{v}\vec{v}^T) = R,\ \ \hat{\vec{x}}_0^+ = E(x_0),\ \ P_0^+ = E[(x_0 - \hat{\vec{x}}_0^+)(x_0 - \hat{\vec{x}}_0^+)^T]$

1. $P_k^- = F\, P_{k-1}^+ \, F^T + Q$ // State Covariance
2. $K_k = P_k^- H^T (H\, P_k^-\, H^T + R)^{-1}$ // Kalman Gain
3. $\hat{\vec{x}}_k^- = F\, \hat{\vec{x}}_{k-1}^+ + G\, \vec{u}_{k-1}$ // A Priori Step
4. $\hat{\vec{x}}_k^+ = \hat{\vec{x}}_k^- + K_k(\vec{y}_k - H\hat{\vec{x}}_k^-)$ // A Posteriori Step
5. $P_k^+ = (I - K_k\, H)P_k^-$ // Covariance Update

The generic discrete Kalman Filter needs compensation [48] to accommodate for time delayed measurements as shown in Algo. 6

Algorithm 6: Delayed Measurement at time k_0 (visible at time k)

1. $S_k = H\, P_k^-\, H^T + R\, \hat{\vec{x}}^+(t_{proc})$ // Retrodict covariance estimate from k to k_0
2. $r_k = y_k - H\, \hat{x}_k^-$ // Measurement residual at time step k
3. $\hat{x}^-(k_0, k) = F(k_0, k)[\hat{\vec{x}}_k^- - Q\, H^T S_k^{-1} r_k]$ // $\hat{x}(k_0, k)$ Retrodict state estimate from k to k_0
4. $P_w = Q - Q\, H_k\, S_k^{-1}\, H\, Q$ // Process Covariance
5. $P_{xw} = Q - P_k^-\, H_k^T\, S_k^{-1}\, H\, Q$ // Process State Covariance
6. $P(k_0, k) = F(k_0, k)(P_k - P_{xw} - P_{xw}^T + P_w)F^T(k_0, k)$ // State Covariance
7. $S_{k_0} = H\, P(k_0, k)H^T + R$ // Measurement Covariance at time k_0
8. $P_{xy} = [P_k^- - P_{xw}]F^T(k_0, k)\, H^T$ // State Measurement Covariance
9. $\hat{\vec{x}}_k^+ = \hat{\vec{x}}_k^- + P_{xy}\, S_{k_0}^{-1}(\vec{y}_{k_0} - H\hat{\vec{x}}^-(k_0, k))$ // A Posteriori state Update
10. $P_k^+ = P_k^- - P_{xy}\, S_{k_0}^{-1}\, P_{xy}^T$ // A Posteriori Covariance Update

7.3 Orientation Fusion

The state dynamics of the orientation using IMU is described in detail in 6.2.

State Dynamics

The orientation state is represented by the quaternion $\mathbf{q} = {}^G\mathbf{q}$ and the input from the gyroscopes is represented by $\Omega = skew(\vec{\omega} = \begin{bmatrix} p & q & r \end{bmatrix}^T)$. The discrete time model of

the system is given by (7.5).

$$\mathbf{q}_{k+1} = \mathbf{q}_k + \frac{1}{2}\,\Omega\,\mathbf{q}_k\,\delta t \tag{7.5}$$

$$\mathbf{q}_{k+1} = \frac{\mathbf{q}_{k+1}}{||\mathbf{q}_{k+1}||}$$

The time update is the propogation of the system state using the gyroscope data. The time interval between the updates is consistent at $5ms$.

Measurement Equation

The measurement is the orientation of the camera q_{cam}, which is described in detail in 5.3.5. It is given by (7.6). The ideal time interval is assumed to be $100ms$.

$$y = q_{cam} \tag{7.6}$$

The orientation fusion is achieved using a Kalman filter. However, there is a time delay in receiving the measurement update i.e. the pose captured from the camera is delayed due to processing time. Hence, the measurement data from the camera at time t_{cap} is only obtained at time t_{proc} which is typically delayed by the range of 70-100ms. The buffer is long enough to hold 5 times the measurement delay (typically 5*20(time updates in 1 measurement)=100).

7.3.1 Kalman Filter

The fused orientation $\hat{\mathbf{q}}^{+}(t_{cap})$ is obtained from a time delayed Kalman filter as described in Algo. 6. A simplified discrete Kalman filter is shown in table 7.1.

Time Update	**Measurement Update**
$\hat{\mathbf{q}} = \begin{cases} \hat{\mathbf{q}}_k^+ & \text{if Measurment update in step k} \\ \hat{\mathbf{q}}_k^- & \textit{otherwise} \end{cases}$	
1.*State Update* $\hat{\mathbf{q}}_{k+1}^- = \hat{\mathbf{q}} + \frac{1}{2}\,\Omega\,\hat{\mathbf{q}}\,\delta t$ 2. *Quaternion Normalization* $\hat{\mathbf{q}}_{k+1}^- = \frac{\hat{\mathbf{q}}_{k+1}^-}{\|\hat{\mathbf{q}}_{k+1}^-\|}$ 3. *Covariance Update* $P_{rot}^- = P_{rot}^- + Q_{rot}$	1. *Kalman Gain Computation* $K_{rot} = P_{rot}^-(P_{rot}^- + R_{rot})^{-1}$ 2. *Measurement Update* $\hat{\mathbf{q}}_k^+ = \hat{\mathbf{q}}_{k-1}^- + K_{rot}(\mathbf{q}_{cam} - \hat{\mathbf{q}}_{k-1}^-)$ 3. *Quaternion Normalization* $\hat{\mathbf{q}}^+(k) = \frac{\hat{\mathbf{q}}^+(k)}{\|\hat{\mathbf{q}}^+(k)\|}$ 4. *Covariance Update* $P_{rot}^+ = (I - K_{rot})P_{rot}^-$

Table 7.1: Orientation - Sensor Fusion

7.3.2 Forward Time Propagation of Fused Orientation

Let us consider the pose estimate from the time the image was captured (t_{cap}) and the time when the image was processed(t_{proc}) at time intervals of IMU update ($t_1, t_2 ... t_n$). The data looks similar to the one shown in table 7.2

Time Series	t_{cap}	t_1	t_2	t_3	...	t_n	t_{proc}
Orientation Series	$\hat{\mathbf{q}}^-(t_{cap})$	$\hat{\mathbf{q}}^-(t_1)$	$\hat{\mathbf{q}}^-(t_2)$	$\hat{\mathbf{q}}^-(t_3)$	...	$\hat{\mathbf{q}}^-(t_n)$	$\hat{\mathbf{q}}^-(t_{proc})$
Measurement Data							$q_{cam}(t_{cap})$

Table 7.2: Sensor buffer after Time Update

Time Series	t_{cap}	t_1	t_2	t_3	...	t_n	t_{proc}
Orientation Series	$\hat{\mathbf{q}}^+(t_{cap})$	$\hat{\mathbf{q}}^-(t_1)$	$\hat{\mathbf{q}}^-(t_2)$	$\hat{\mathbf{q}}^-(t_3)$	...	$\hat{\mathbf{q}}^-(t_n)$	$\hat{\mathbf{q}}^-(t_{proc})$

Table 7.3: Sensor buffer after Measurement Update

At time t_{proc}, the orientation from the camera(q_{cam}) at time t_{cap} becomes available. Hence, we perform the sensor fusion at time t_{cap} and after sensor fusion at time t_{cap}, the orientation data is shown in table 7.3.

The quaternion multiplication is represented by $\otimes$. A slight variation of the compensation due to time delay in measurement is shown in Algo. 7.

Algorithm 7: Algorithm to Forward Propogate fused orientation

Given: $\hat{\mathbf{q}}^{+}(t_{cap})$, $\hat{\mathbf{q}}^{-}(t_{cap})$, $\hat{\mathbf{q}}^{-}(t_{proc})$

Compute: $\hat{\mathbf{q}}^{+}(t_{proc})$

Steps:

1 $\delta q(t_{cap}) = (\hat{\mathbf{q}}^{-}(t_{cap}))^{-1} \otimes \hat{\mathbf{q}}^{+}(t_{cap})$ // Error quaternion at t_{cap}

2 $\hat{\mathbf{q}}^{+}(t_{proc}) = \hat{\mathbf{q}}^{-}(t_{proc}) \otimes \delta q(t_{cap})$ // Fused quaternion at t_{proc}

Proof of Algorithm 7:

By definition, the orientation $\hat{\mathbf{q}}^{-}(t_{proc})$ is given in (7.7) 7.8

$$\hat{\mathbf{q}}^{-}(t_{proc}) = \underbrace{\delta\hat{\mathbf{q}}^{-}(t_{n\to n-1}) \otimes ... \otimes \delta\hat{\mathbf{q}}^{-}(t_{2\to 1}) \otimes \delta\hat{\mathbf{q}}^{-}(t_{1\to cap})}_{\hat{q_r}^{-}} \otimes \hat{\mathbf{q}}^{-}(t_{cap}) \tag{7.7}$$

$$\hat{\mathbf{q}}^{-}(t_{proc}) = \hat{q_r}^{-} \otimes \hat{\mathbf{q}}^{-}(t_{cap}) \tag{7.8}$$

But the rotation between the time t_{cap} and the time t_{proc}, i.e. $\hat{q_r}^{-}$ stays constant even after fusion, since it is computed from the IMU and only the initial orientation at time t_{cap} is changed. Hence, it follows that

$$\hat{\mathbf{q}}^{+}(t_{proc}) = \hat{q_r}^{-} \otimes \hat{\mathbf{q}}^{+}(t_{cap}) \tag{7.9}$$

Let us find an error quaternion Δq as shown in (7.10)-7.11

$$\hat{\mathbf{q}}^{+}(t_{cap}) = \hat{\mathbf{q}}^{-}(t_{cap}) \otimes \Delta q(t_{cap}) \tag{7.10}$$

$$\Delta q(t_{cap}) = (\hat{\mathbf{q}}^{-}(t_{cap}))^{-1} \otimes \hat{\mathbf{q}}^{+}(t_{cap}) \tag{7.11}$$

Substituting (7.10) into (7.9), we get

$$\hat{\mathbf{q}}^{+}(t_{proc}) = \hat{q_r}^{-} \otimes (\hat{\mathbf{q}}^{-*}(t_{cap}) \otimes \Delta q(t_{cap})) \tag{7.12}$$

$$\hat{\mathbf{q}}^{+}(t_{proc}) = (\hat{q_r}^{-} \otimes \hat{\mathbf{q}}^{-}(t_{cap})) \otimes \Delta q(t_{cap}) \tag{7.13}$$

$$\hat{\mathbf{q}}^{+}(t_{proc}) = \hat{\mathbf{q}}^{-}(t_{proc}) \otimes \Delta q(t_{cap}) \tag{7.14}$$

The co-variance propogation follows the algorithm 6

7.4 Position Fusion

7.4.1 Linear Time Invariant (LTI) System Model

The sensor reading is the accelerometer data in the body co-ordinates $\vec{\boldsymbol{u}}^B = \begin{bmatrix} a_x^B & a_y^B & a_z^B \end{bmatrix}^T$. Since, we are interested in the global co-ordinates $\vec{\boldsymbol{u}}^G$, we transform the acceleration to $\{G\}$ using the estimate Rotation matrix $R(\hat{\mathbf{q}})$ and remove the gravity to estimate the acceleration in $\{G\}$. This is further follwed by a Band Pass Filter (BPF) to remove the static component of gravity and high frequency noise to obtain the final input $\vec{\boldsymbol{u}}$ in the global co-ordinates $\{G\}$.

$$^G\vec{\boldsymbol{u}} = R(\hat{\mathbf{q}})^B\vec{\boldsymbol{u}} - {}^G\vec{\boldsymbol{g}} \tag{7.15}$$

$$\vec{\boldsymbol{u}} = BPF(^G\vec{\boldsymbol{u}}) \tag{7.16}$$

The position fusion for the ROV is performed in a similar manner to the orientation fusion using a Kalman filter. The state $\vec{\boldsymbol{x}} = \begin{bmatrix} p_x & p_y & p_z & v_x & v_y & v_z \end{bmatrix}^T$ and the control input or acceleration $\vec{\boldsymbol{u}} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T$ constitue the state dynamics as shown in (7.17). The process noise (accelerometer) is represented by $\vec{\boldsymbol{w}}$.

$$\dot{\vec{\boldsymbol{x}}} = \begin{bmatrix} 0_{3\times3} & I_{3\times3} \\ 0_{3\times3} & 0_{3\times3} \end{bmatrix} \vec{\boldsymbol{x}} + \begin{bmatrix} 0_{3\times3} \\ I_{3\times3} \end{bmatrix} \vec{\boldsymbol{u}} + \vec{\boldsymbol{w}} = A_c\vec{\boldsymbol{x}} + B_c\vec{\boldsymbol{u}} + \vec{\boldsymbol{w}} \tag{7.17}$$

Discretizing the continuous time equation by first order approximation (as the sampling time interval ($\delta t = 0.005$)) is quite small, we obtain the following equations:

$$\vec{\boldsymbol{x}}_k = e^{A_c\delta t}\vec{\boldsymbol{x}}_{k-1} + \int_{t_{k-1}}^{t_k} e^{A_c(t_k-\tau)}d\tau B_c\vec{\boldsymbol{u}}(k-1) + \int_{t_{k-1}}^{t_k} e^{A_c(t_k-\tau)}\vec{\boldsymbol{w}}(\tau)d\tau \tag{7.18}$$

$$A = (I + A_c * \delta t) \tag{7.19}$$

$$\vec{\boldsymbol{x}}_k = A\vec{\boldsymbol{x}}_{k-1} + B\vec{\boldsymbol{u}}_{k-1} + \vec{\boldsymbol{w}}_{k-1} \tag{7.20}$$

The solution to the discrete LTI system 7.20 for n time steps is given by (7.21)

$$\vec{\boldsymbol{x}}_{k+n} = A^n\vec{\boldsymbol{x}}_k + \sum_{i=k}^{k+n-1} A^{k+n-1-i}B\vec{\boldsymbol{u}}_i + \sum_{i=k}^{k+n-1} A^{k+n-1-i}\vec{\boldsymbol{w}}_i \tag{7.21}$$

7.4.2 Kalman Filter

The position fusion is accomplished using a Kalman filter. However, as seen in the orientation, there is a time delay in the position measurement obtained from the camera. In order to compensate for the time delay, we propogate our system forward in time. The equations used for Kalman Filter are shown in table 7.4.

Time Update	**Measurement Update**
$\hat{\vec{\boldsymbol{x}}}_k = \begin{cases} \hat{\vec{\boldsymbol{x}}}_k^+ & \text{if Measurment update in step k} \\ \hat{\vec{\boldsymbol{x}}}_k^- & \textit{otherwise} \end{cases}$	
1. *State Update* $\hat{\vec{\boldsymbol{x}}}_{k+1}^- = A\hat{\vec{\boldsymbol{x}}}_k + B\vec{\boldsymbol{u}}_k$ 2. *Covariance Update* $P_{trans}^- = AP_{trans}^- A^T + Q_{trans}$	1. *Kalman Gain Computation* $K_{trans} = P_{trans}^-(P_{trans}^- + R_{trans})^{-1}$ 2. *Measurement update* $\hat{\vec{\boldsymbol{x}}}_k^+ = \hat{\vec{\boldsymbol{x}}}_{k-1}^- + K_{trans}(\vec{\boldsymbol{y}}_{cam} - \hat{\vec{\boldsymbol{x}}}_{k-1}^-)$ 3. *Covariance Update* $P_{trans}^+ = (I - K_{trans})P_{trans}^-$

Table 7.4: Position - Sensor Fusion

7.4.3 Forward Time Propogation of Fused Position

The state of the system at time t_{proc} (when the pose from the camera at time t_{cap} is computed), is shown in table 7.5.

Time Series	t_{cap}	t_1	t_2	t_3	...	t_n	t_{proc}
Orientation State	$\hat{\mathbf{q}}^-(t_{cap})$	$\hat{\mathbf{q}}^-(t_1)$	$\hat{\mathbf{q}}^-(t_2)$	$\hat{\mathbf{q}}^-(t_3)$	...	$\hat{\mathbf{q}}^-(t_n)$	$\hat{\mathbf{q}}^-(t_{proc})$
Position State	$\hat{\vec{\boldsymbol{x}}}^-(t_{cap})$	$\hat{\vec{\boldsymbol{x}}}^-(t_1)$	$\hat{\vec{\boldsymbol{x}}}^-(t_2)$	$\hat{\vec{\boldsymbol{x}}}^-(t_3)$	...	$\hat{\vec{\boldsymbol{x}}}^-(t_n)$	$\hat{\vec{\boldsymbol{x}}}^-(t_{proc})$
Measurement Data					...		$\vec{\boldsymbol{y}}_{cam}(t_{cap}), q_{cam}(t_{cap})$

Table 7.5: Sensor Buffer after Time Update

The position fusion is performed at time (t_{cap}) using the prior state $\hat{\vec{\boldsymbol{x}}}^-(t_{cap})$ and the camera position $\vec{\boldsymbol{y}}_{cam}(t_{cap})$ to obtain the fused position at the captured time $\hat{\vec{\boldsymbol{x}}}^+(t_{cap})$. After sensor fusion, the state of the system is as shown in table 7.6.

Time Series	t_{cap}	t_1	t_2	t_3	...	t_n	t_{proc}
Orientation State	$\hat{\mathbf{q}}^+(t_{cap})$	$\hat{\mathbf{q}}^-(t_1)$	$\hat{\mathbf{q}}^-(t_2)$	$\hat{\mathbf{q}}^-(t_3)$	...	$\hat{\mathbf{q}}^-(t_n)$	$\hat{\mathbf{q}}^+(t_{proc})$
Position State	$\hat{\vec{\boldsymbol{x}}}^+(t_{cap})$	$\hat{\vec{\boldsymbol{x}}}^-(t_1)$	$\hat{\vec{\boldsymbol{x}}}^-(t_2)$	$\hat{\vec{\boldsymbol{x}}}^-(t_3)$	...	$\hat{\vec{\boldsymbol{x}}}^-(t_n)$	$\hat{\vec{\boldsymbol{x}}}^-(t_{proc})$

Table 7.6: Sensor Buffer after Measurement Update

The algorithm to obtain the fused position estimate $\hat{\vec{\boldsymbol{x}}}^+(t_{proc})$ is shown in 8.

Algorithm 8: Algorithm to Forward Propogate fused position

Given: $\hat{\vec{\boldsymbol{x}}}^+(t_{cap})$, $\hat{\vec{\boldsymbol{x}}}^-(t_{cap})$, $\hat{\vec{\boldsymbol{x}}}^-(t_{proc})$, $\delta q(t_{cap}) = \delta q$

Compute: $\hat{\vec{\boldsymbol{x}}}^+(t_{proc})$

Steps:

1 $\delta\vec{\boldsymbol{x}}(t_{cap}) = \hat{\vec{\boldsymbol{x}}}^+(t_{cap}) - \hat{\vec{\boldsymbol{x}}}^-(t_{cap})$ // Error compensation at t_{cap} due to fusion

2 $\hat{\vec{\boldsymbol{x}}}^+(t_{proc}) = \hat{\vec{\boldsymbol{x}}}^-(t_{proc}) + R(\delta q)\, A^n\, \delta\vec{\boldsymbol{x}}(t_{cap})$ // Fused position at t_{proc}

Proof of Algorithm 8:

$$\hat{\vec{\boldsymbol{x}}}^-(t_{proc}) = A^n\hat{\vec{\boldsymbol{x}}}^-(t_{cap}) + \sum_{k=0}^{n-1} A^{n-1-k}B\vec{\boldsymbol{u}}_k + \sum_{k=0}^{n-1} A^{n-1-k}\vec{\boldsymbol{w}}_k \tag{7.22}$$

The fused position $\hat{\vec{\boldsymbol{x}}}^+(t_{cap})$ is obtained using Kalman filter as shown in table 7.4. Due to linear nature of the state equations, after fusion, the state propogation is shown in (7.23).

$$\hat{\vec{\boldsymbol{x}}}^+(t_{proc}) = A^n\hat{\vec{\boldsymbol{x}}}^+(t_{cap}) + \sum_{k=0}^{n-1} A^{n-1-k}B\vec{\boldsymbol{u}}_k + \sum_{k=0}^{n-1} A^{n-1-k}\vec{\boldsymbol{w}}_k \tag{7.23}$$

Let us define a position compensation error due to sensor fusion $\delta\vec{\boldsymbol{x}}(t_{cap})$ as shown in (7.24).

$$\delta\vec{\boldsymbol{x}}(t_{cap}) = \hat{\vec{\boldsymbol{x}}}^+(t_{cap}) - \hat{\vec{\boldsymbol{x}}}^-(t_{cap}) \tag{7.24}$$

Substituting (7.24) in (7.23),

$$\hat{\vec{\boldsymbol{x}}}^+(t_{proc}) = \underbrace{A^n\hat{\vec{\boldsymbol{x}}}^-(t_{cap}) + \sum_{i=0}^{n-1} A^{n-1-i}B\vec{\boldsymbol{u}}(i) + \sum_{i=0}^{n-1} A^{n-1-i}\vec{\boldsymbol{w}}(i)}_{\hat{\vec{\boldsymbol{x}}}^-(t_{proc})} + A^n\delta\vec{\boldsymbol{x}}(t_{cap}) \tag{7.25}$$

Thus, the position estimate at current time in the absence of orientation compensation can be computed as follows:

$$\hat{\vec{\boldsymbol{x}}}^{+}(t_{proc}) = \hat{\vec{\boldsymbol{x}}}^{-}(t_{proc}) + A^{n}\delta\vec{\boldsymbol{x}}(t_{cap}) \tag{7.26}$$

The effect of orientation compensation at time (t_{cap}) is non-linear as the acceleration input $\vec{\boldsymbol{u}}^{*}(i)$ depends on the rotation ((7.15)) followed by high pass filtering((7.16)) and the product term $A^{n-1-i}B$. However, we assume that the change in orientation is negligible and hence we only focus on the translation part.

$$\hat{\vec{\boldsymbol{x}}}^{+}(t_{proc}) = \hat{\vec{\boldsymbol{x}}}^{-}(t_{proc}) + A^{n}\delta\vec{\boldsymbol{x}}(t_{cap}) \tag{7.27}$$

Due to the special property of the matrix A (7.19) and A_c (sparse), (7.17), the matrix power can be computed as shown in (7.28)

$$A^{n} = I_{6\times 6} + n\delta t A_c \tag{7.28}$$

The covariance propogation follows the algorithm 6.

The block diagram of the optimal state estimator is summarized in Fig. 7.1.

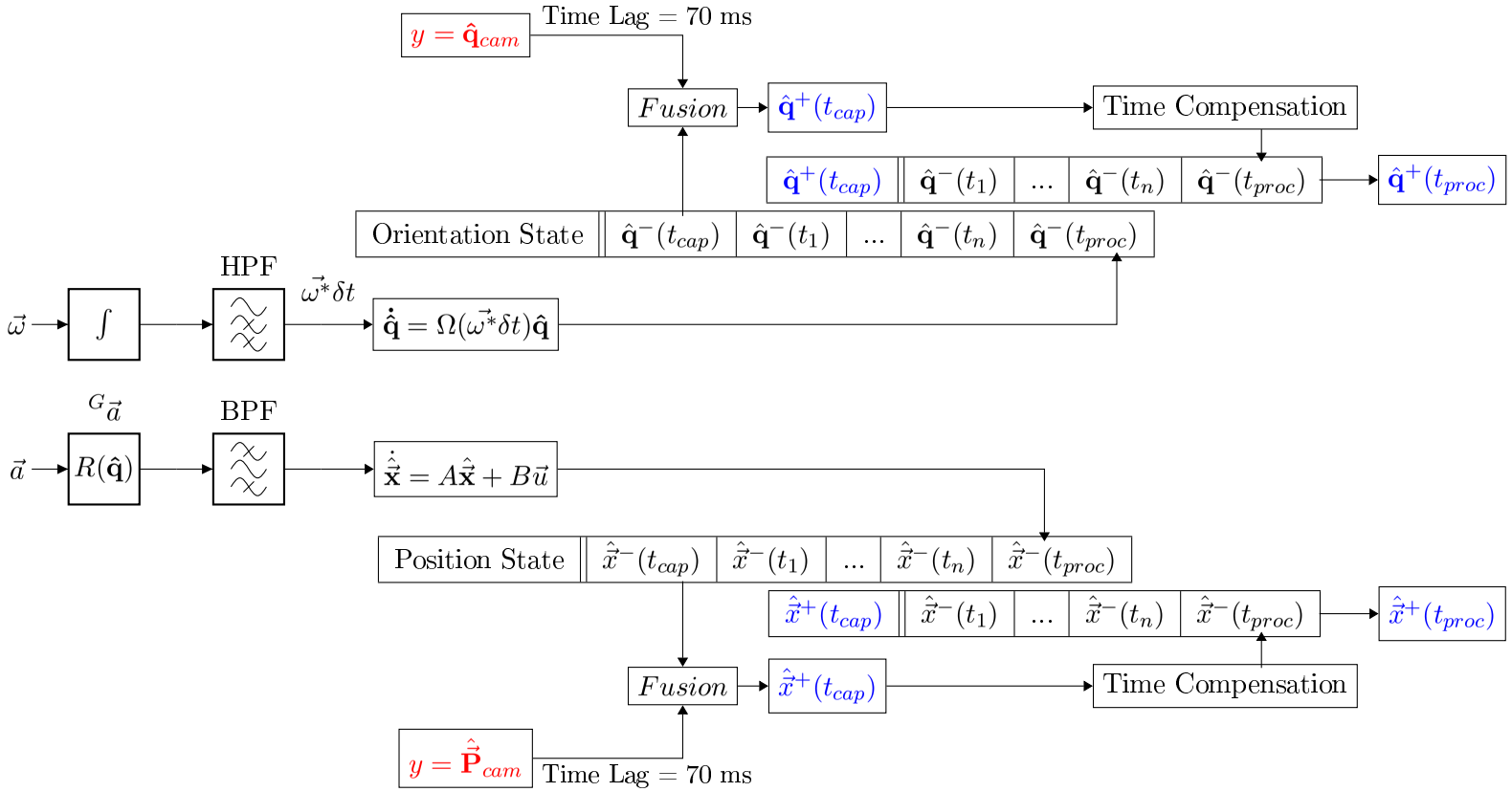
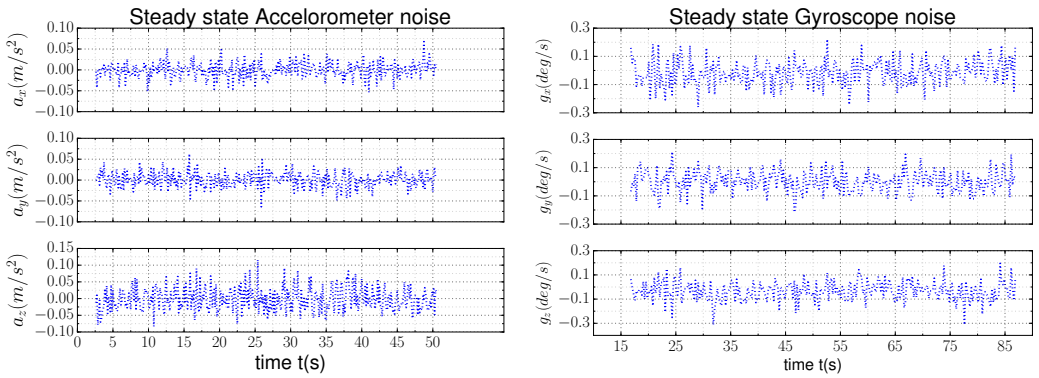


Figure 7.1: Block Diagram - Sensor Fusion using Kalman Filter

7.5 Co-Variance Estimation

Process Noise

The steady state value of the sensor readings were used to obtain an estimate of the co-variances used in the Kalman filter. A steady value of the filtered acceleration (after removal of gravity) was collected at different orientations. The co-variance of the accelerometer data (process noise) gives a measure of the process co-variance matrix in cm^2 $Q_{trans} = 2 * E(\vec{w}^T \vec{w}) = diag([0.05\ 0.05\ 0.05\ 10\ 10\ 10])$. Similarly, the steady value of the gyro values gives a measure of the process co-variance matrix in rad^2/s^4 $Q_{rot} = 2 * E(\vec{w}^T \vec{w}) = diag([0.05\ 0.05\ 0.05])$. A sample of the noise from the sensors in one particular pose is shown in Fig. 7.2.



(a) Acc Noise Estimate **(b)** Gyro Noise Estimate

Figure 7.2: Process Noise

Measurement Noise

The noise in the vision system is modeled as a white gaussian noise with twice the error obtained from the camera calibration i.e. noise in the pixels is given by $\sigma_{pix} = 2 * e_{rms}$. We apply this gaussian image noise in our simulations to estimate the error in the position and orientation. From the error in the simulations, we compute the measurement co-variance matrix for translation (cm^2) $R_{trans} = E(\vec{v}^T \vec{v}) = diag([0.25\ 0.25\ 0.25])$ and rotation (rad^2) $R_{rot} = E(\vec{v}^T \vec{v}) = diag([0.05\ 0.05\ 0.05])$.

The co-variance of measurement noise depends on the distance between the marker

and the ROV. Hence, an adaptive co-variance was proposed which depends on the factor ($\frac{r_{marker}}{z}$). However, as the distance was fixed at approximately 1m., a constant covariance matrix was utilized in our experiments. The exact value of co-variance matrix was estimated from simulations in Matlab (Chapter 5).

7.6 Experiments

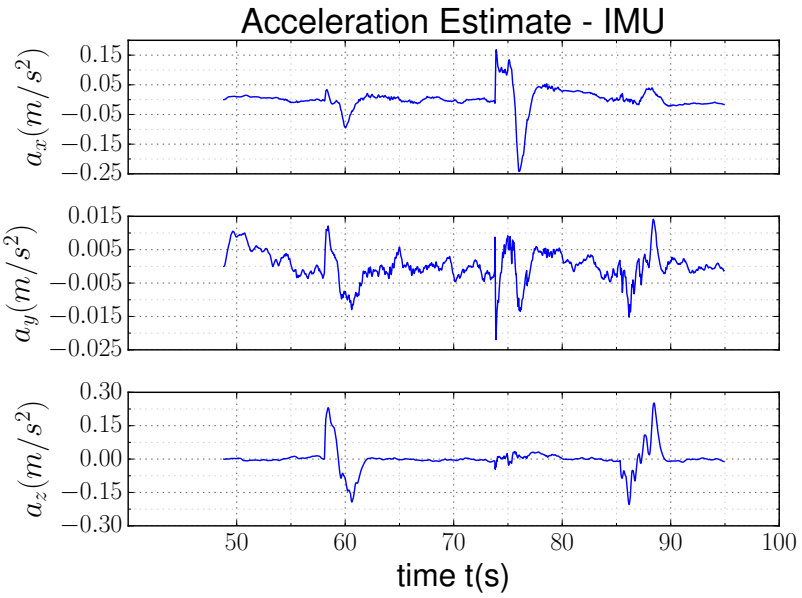
The sensor fusion was studied by moving the sensor(cam/imu) on a trajectory similar to the one shown in Fig. 5.5 but the rectangular dimensions were 34.5×36.5 cm. The models compared here are the cam/imu fusion and the camera based predictive model. The results are shown in Fig. 7.1.

Fig. 7.3a shows the sensor readings of the accelerometer as the sensor is moved. The camera based predictive model uses the previous camera velocity data to extrapolate the readings to obtain the position till the new reading arrives. The entire sensor fusion is performed by taking the camera readings at 0.5s (7.2a-7.1c), even though the camera readings originally arrive at 100ms. As we lack complex 6-DOF measuring apparatus, this way we can check for the accuracy of the sensor fusion in 1s time intervals.

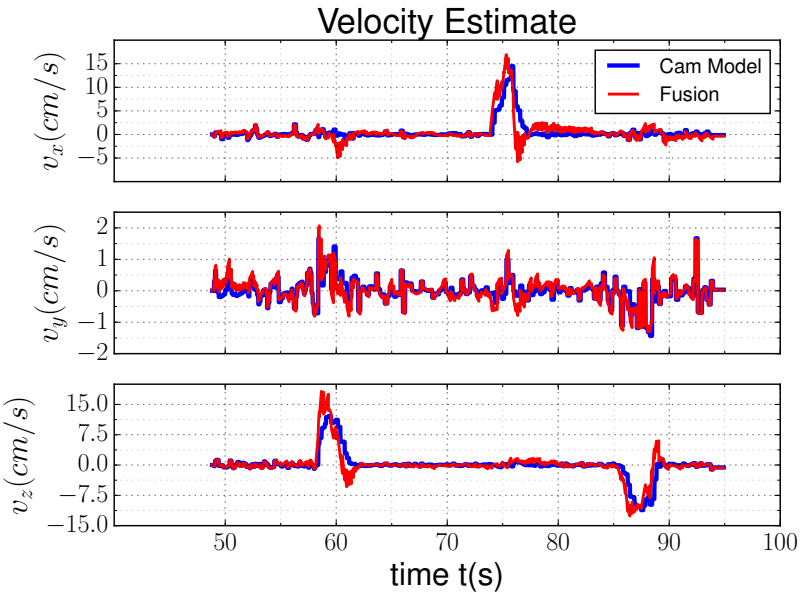
The sensor fusion with camera decreases the drift in the position and smoothes the position reading for the time steps in between the camera readings(Fig. 7.2d) . The sensor fusion also helps the state estimate by removing the velocity bias and maintaining zero-mean velocity at steady-state (i.e. no motion, (Fig. 7.3b)).

As expected the camera/imu system does comparatively better than the camera alone readings when the system is moving but due to small residual velocities, during steady state (i.e. no motion) the camera readings alone give a more accurate estimate.

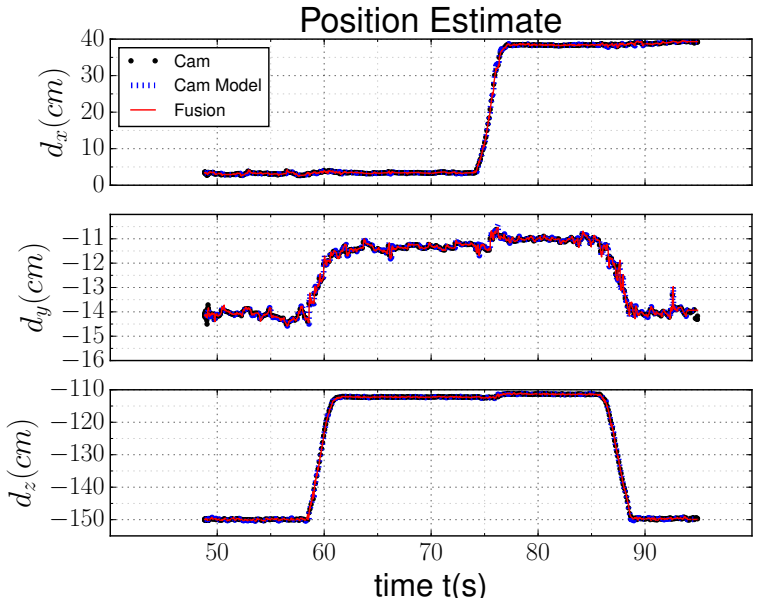
However, when we use all the camera readings (i.e. 100ms), the sensor fusion and the camera predictive model converge (Fig. 7.3b-7.2d). This may be due to the small velocities used while testing. In the presence of large velocities of the system, the cam/imu system would yield better performance than the camera-predictive model.



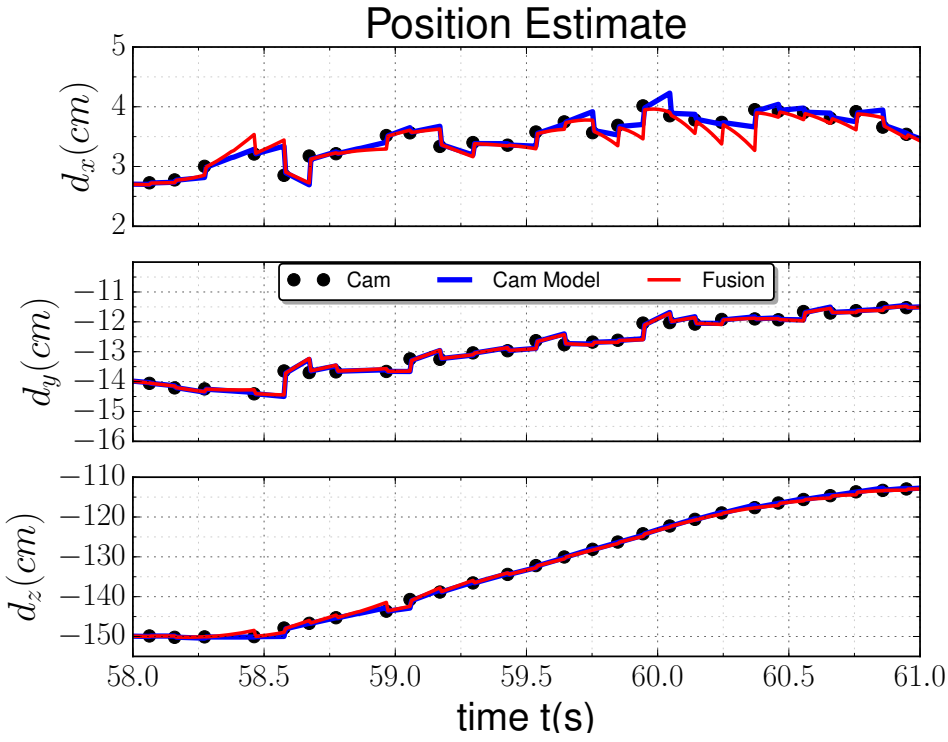
(a) Acceleration Plot



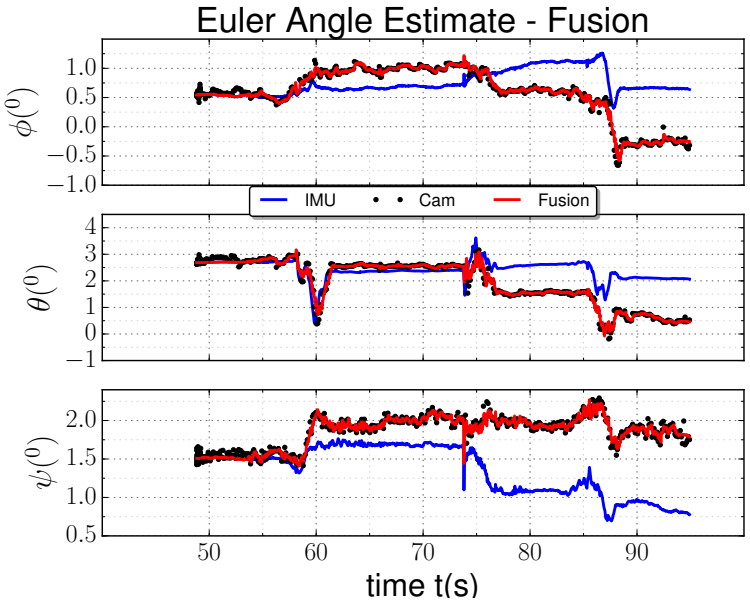
(b) Velocity Plot



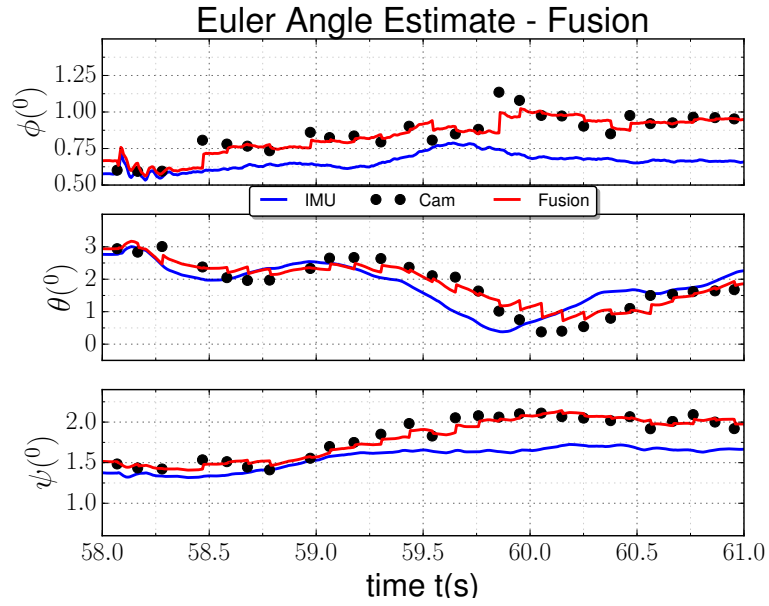
(c) Position Plot



(d) Position Plot Magnified

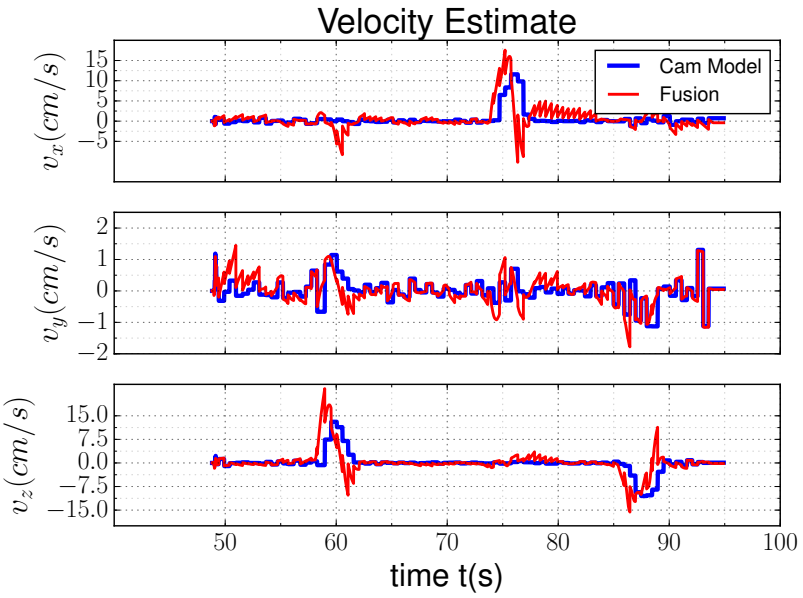


(e) Orientation Plot

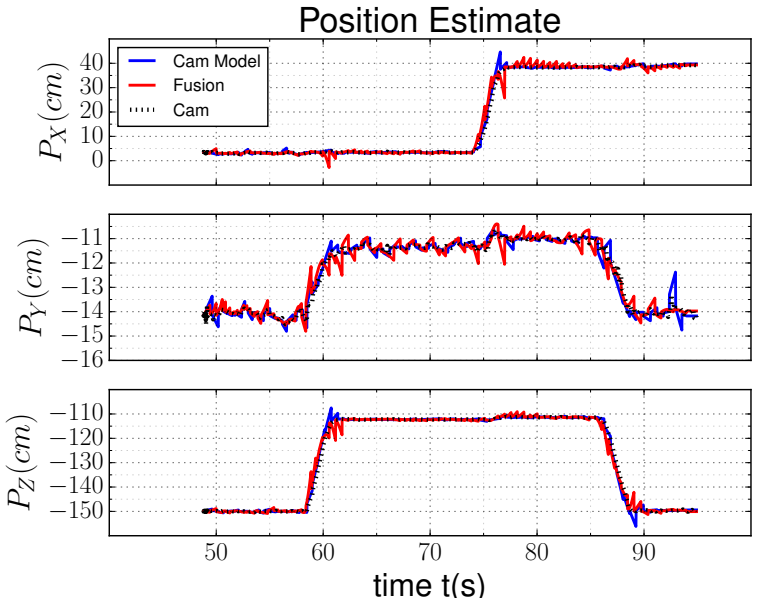


(f) Orientation Plot - Magnified

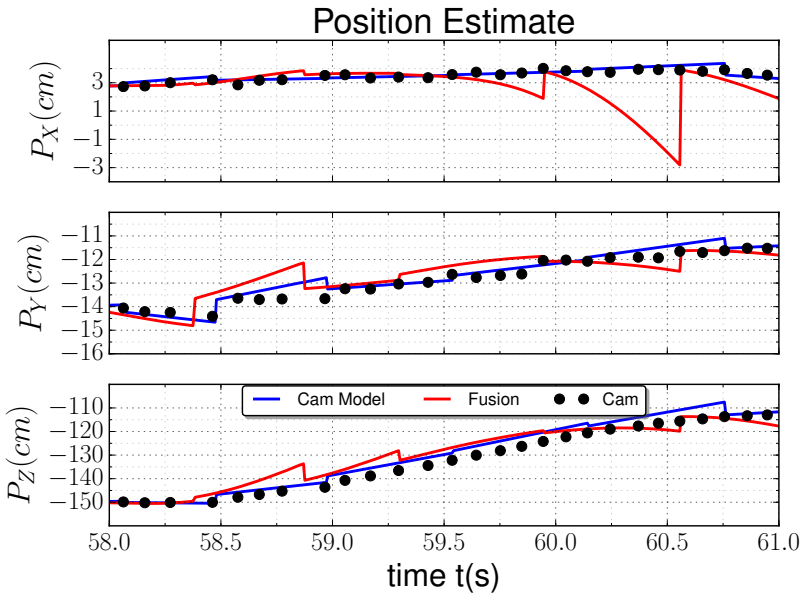
Figure 7.1: I Sensor Fusion - Position Estimation (Camera measurements @ 0.1s)



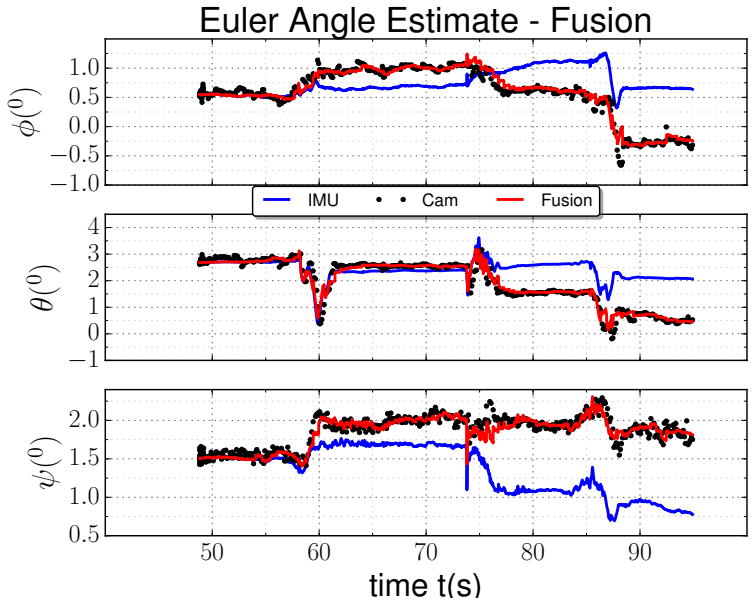
(a) Velocity Plot



(b) Position Plot



(c) Position Plot Enlarged



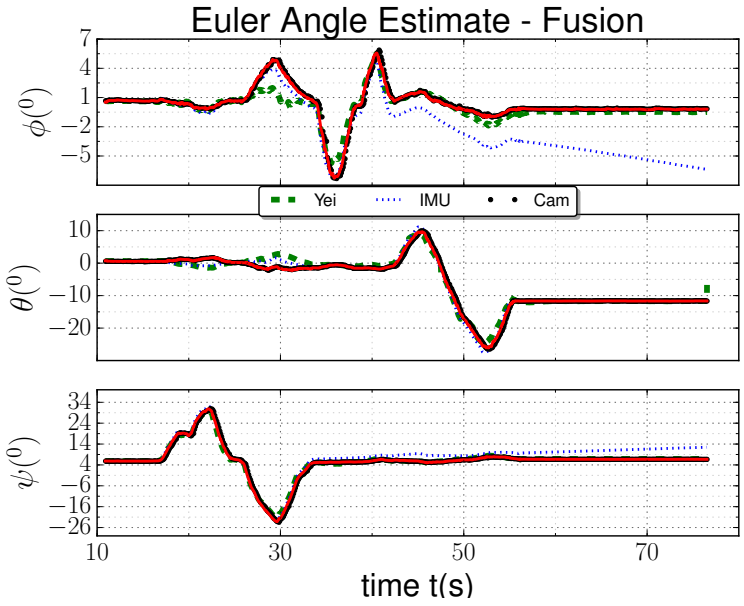
(d) Orientation Plot

II Sensor Fusion - Position Estimation (Camera measurements @ 0.5s)

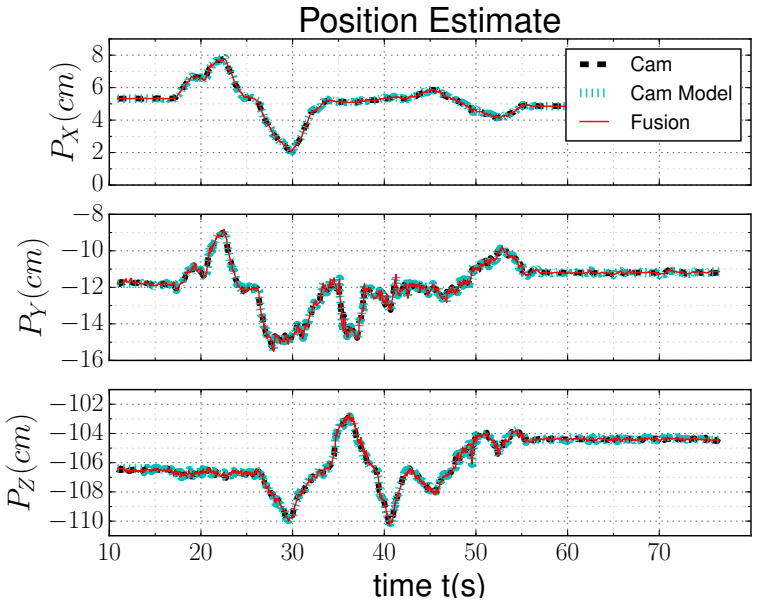
Figure 7.1: Experimental Result - Position Estimation using Sensor Fusion

The orientation experiment was carried out in a similar fashion. The reference orientation was given by a YEI-IMU, which is known to have an orientation accuracy of $\pm 0.5^0$. The results of the experiment are shown in Fig. 7.2.

The euler angles in the x and y axis are off by about $\pm 1.5^0$. This is due to the calibration error between the axes of the YEI-IMU and the camera. However, the z-axis of the YEI-IMU is mounted almost parallel to the camera euler angle and hence the error is of the order of $\pm 0.25^0$.



(a) Euler Angle Estimate



(b) Position Estimate

Figure 7.2: Experimental Result - Orientation Estimation using Sensor Fusion

7.7 Conclusion

The sensor data from the IMU is subject to noise and cannot independently determine the pose of the ROV. The camera measurements are too slow to actuate the ROV in real time. In this chapter, we present a sensor fusion approach using Kalman filter to integrate the vision system with the IMU and achieve real time pose estimation.

We infer from the experiments that the cam-imu sensor fusion yields better results than the camera or the imu used alone. Thus, the camera/imu system is highly recommended as a non-contact pose estimation sensor for the ROV in controlled environments.

Chapter 8

Underwater Experiments

8.1 Overview

The primary objective of the project is to achieve real time control of underwater robot (ROV). In this chapter, we conduct underwater experiments to validate and demonstrate the real time pose estimation using the **SO(3)-PnP** algorithm.

Chapter 7 utilizes the sensor fusion of vision and Inertial sub-systems to achieve optimal state estimation. The experiments discussed in Chapter 7 are conducted in terrestrial environments. This chapter details the experiments carried out in underwater aquatic environment.

The various sections of this chapter proceed as follows. Section 8.2 describes the calibration procedure used to calibrate the camera in aquatic environment. A comparison between the camera position and the encoder gives a fair estimate of the camera accuracy as seen in Section 8.4. Section 8.6 concludes the chapter with important results and observations.

8.2 Underwater Calibration

The focal length of the camera will change underwater due to a change in medium and hence the camera has to be re-calibrated for aquatic environments. The procedure used is similar to the calibration procedure described in Chapter 3. An image of the standard chessboard pattern used for underwater calibration is shown in Fig. 8.1.

8.2.1 Lens Makers Equation

In any medium (say air), the lens-makers equation [49] for a thin lens describes the relation between the focal length in air (f_a) of a convex lens, the index of refraction of air (n_a), radii of curvature of the lens (R_1, R_2) and index of refraction of the lens (n_l) [1] as shown in (8.1)

$$\frac{1}{f_a} = (\frac{n_l}{n_a} - 1)(\frac{1}{R_1} - \frac{1}{R_2}) \tag{8.1}$$

In water, the lens-makers equation is given by (8.2)

$$\frac{1}{f_w} = (\frac{n_l}{n_w} - 1)(\frac{1}{R_1} - \frac{1}{R_2}) \tag{8.2}$$

The index of refraction for air (n_a) and the index of refraction for water (n_w) are characteristically different and are related by the equation 8.3.

$$\frac{n_w}{n_a} = 1.33 \tag{8.3}$$

Hence, we expect the focal length to vary w.r.t. to the medium. Using (8.1) and (8.2), we can obtain a theoretical idea of the focal length variation as shown in (8.4). As a fact, $n_l > n_w = 1.33 n_a$, hence, $\frac{n_l - n_a}{n_l - n_w} > 1$. Thus, we infer that the focal length changes in water.

$$\frac{f_w}{f_a} = \frac{\frac{n_l}{n_a} - 1}{\frac{n_l}{n_w} - 1} = (\frac{n_l - n_a}{n_l - n_w})\frac{n_w}{n_a} = 1.33 \underbrace{\frac{n_l - n_a}{n_l - n_w}}_{>1} > 1 \tag{8.4}$$

8.3 Results of Underwater Calibration

The camera intrinsic matrix differs greatly underwater as compared to air. In underwater environment, the camera is calibrated in a similar procedure as in air (refer Section 3.4.1) to obtain the camera instrinsic matrix A^w_{picam} using (8.5).

$$A^w_{picam} = \begin{bmatrix} 2291.69 & 0 & 635.36 \\ 0 & 2299.71 & 369.24 \\ 0 & 0 & 1 \end{bmatrix} \tag{8.5}$$

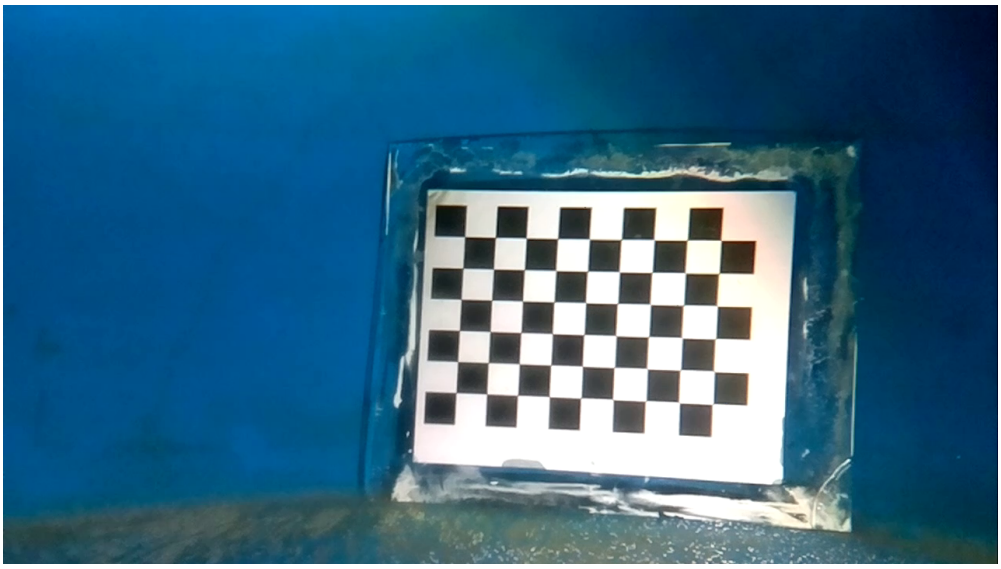
[1] Note: n_l is unkonwn but used to gain theroetical understanding

The radial and tangential distortion co-efficients D^{w}_{picam} are also computed to check for increase in non-linearity due to change in medium. The radial co-efficients are significantly larger. This is due to the refraction of light as it passes from water to the lens through the camera housing.

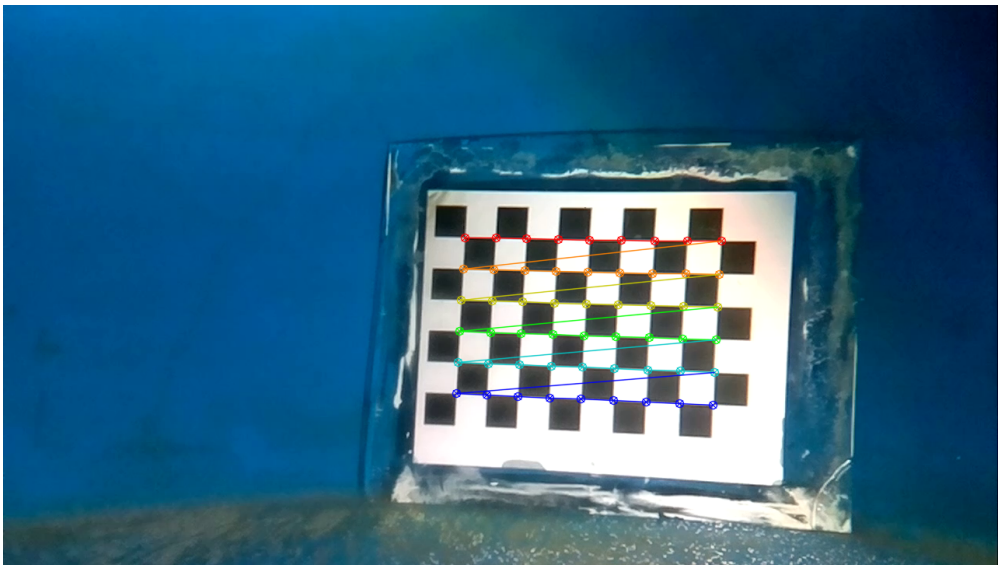
$$D^{w}_{picam} = \begin{bmatrix} k_1 & k_2 & p_1 & p_2 \end{bmatrix}^T = \begin{bmatrix} 1.1234 & -0.2056 & -0.0557 & 0.0035 \end{bmatrix}^T \tag{8.6}$$

The focal length of the same convex lens is significantly larger in water $f_w \approx 2300$ as compared to air $f_a \approx 1170$. The results verify the theoratical observation shown in (8.4)

.



(a) Underwater - 9X6 chessboard with square size 2.5mm



(b) Underwater - Detected Corners of chessboard using OpenCV

Figure 8.1: Underwater Calibration Pattern - Standard Checker board

8.4 Underwater Distance Estimation

The experimental setup is shown in Fig. 8.2. The vision-IMU system is constrained to move in a straight line along the protruding arm, while the controller stays at rest at a convenient location(on ground). The linear distance estimate (in 3D) provided by the camera-imu sensor is compared with the distance calculated by a linear magnetic encoder, mounted on the protruding arm of the test bed. The results are shown in Fig. 8.2.

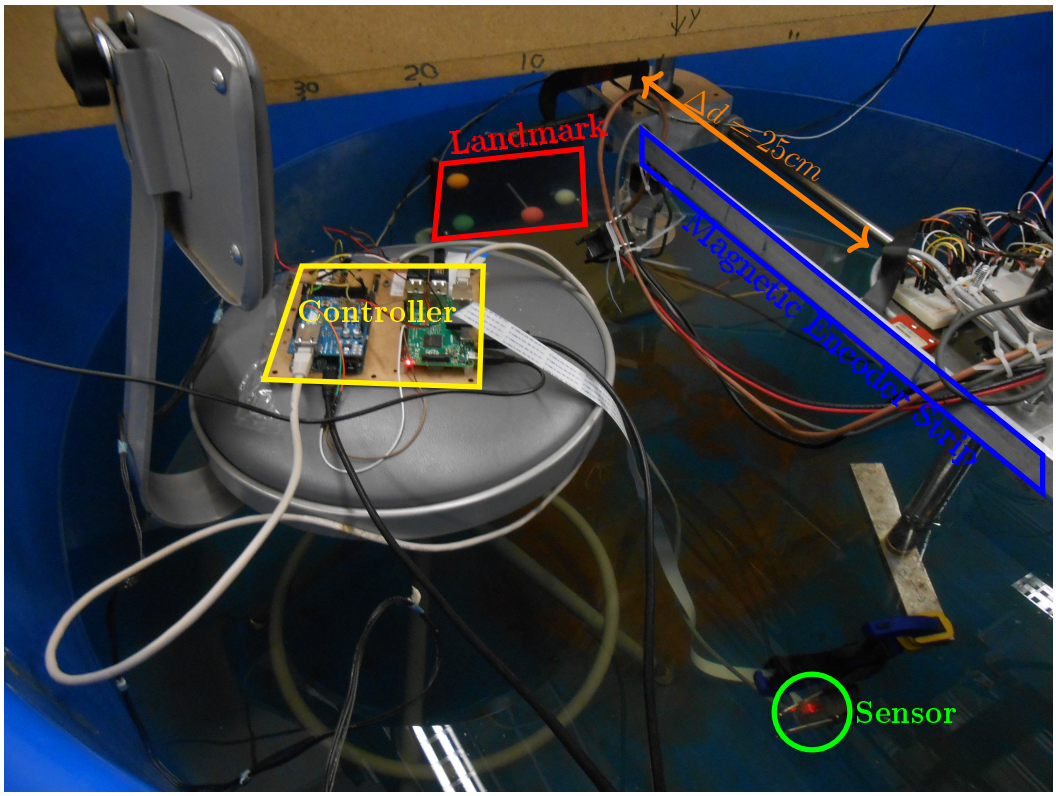
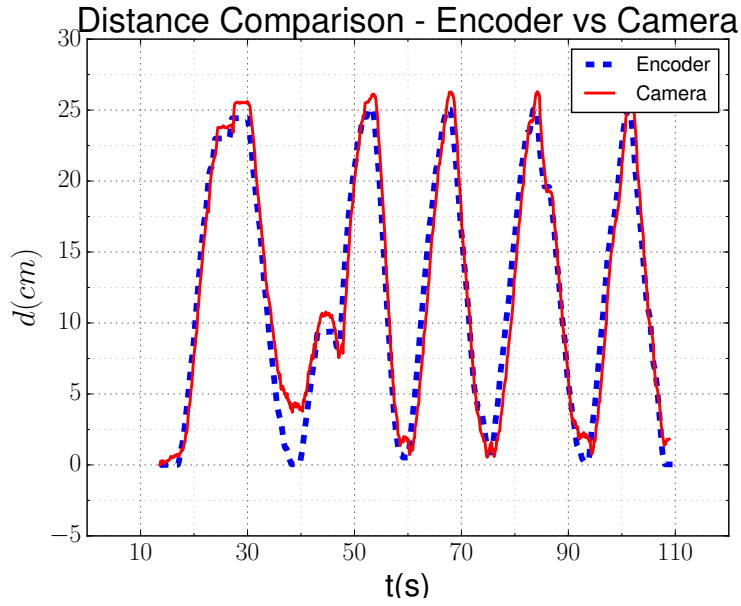
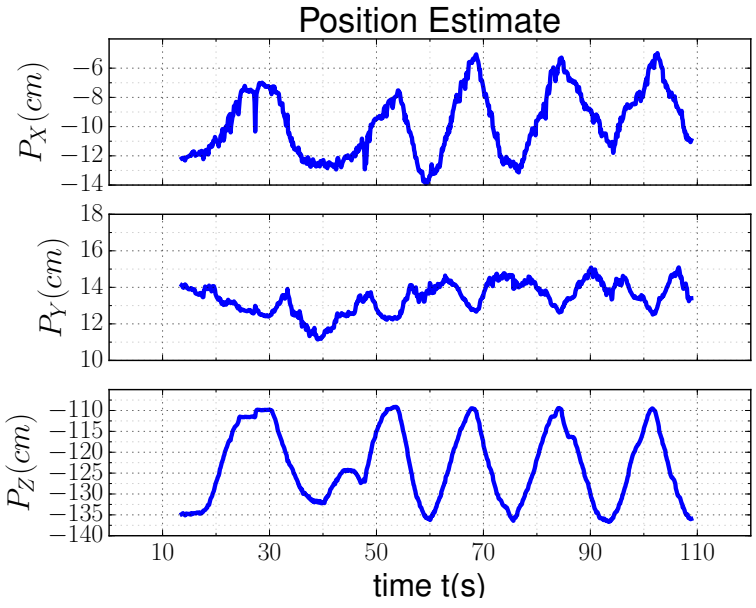


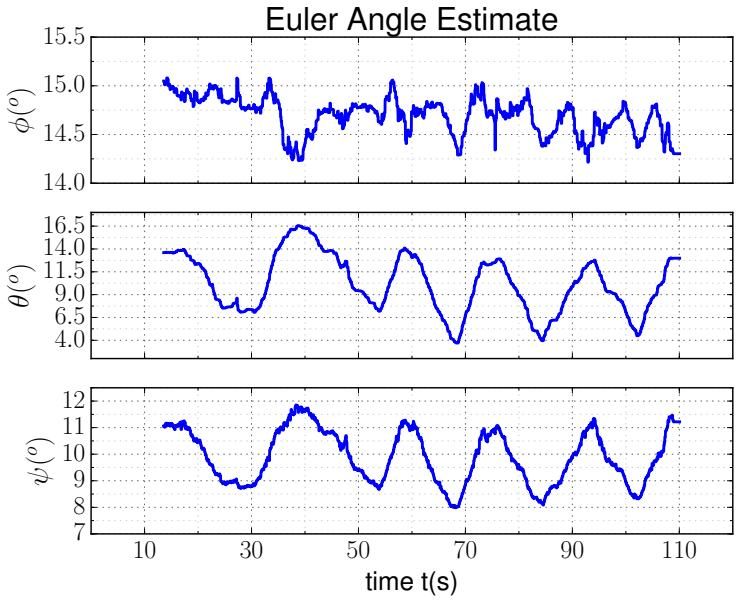
Figure 8.2: Experimental Setup - Distance Tracking in Underwater Environment



(a) Distance Plot



(b) Position Plot



(c) Euler Angle Plot

Figure 8.2: Underwater Experiments - Camea vs Encoder

8.5 Results

The distance tracking performs very well in underwater environments as seen in Fig. 8.3a with an accuracy of $\pm 0.5cm$ peak to peak distance.

The greater deviation in underwater environment is due to light refraction from the acrylic housing which causes significant radial distortion (observered from calibration co-efficients in underwater environment).

The experimental setup needs further encoders and measuring apparatus to investigate the algorithm accuracy in 6-DOF in real time.

8.6 Conclusion

In this chapter, we have prototyped and designed a Raspberry-Pi camera to operate as an underwater camera. The $SO(3) - PnP$ algorithm can be used to compute the pose in real time and track the pose of the ROV.

Chapter 9

Conclusion and Future Work

9.1 Overview

This chapter provides the essence of the research on sensors for small scale ROV's and provides further insght to develop novel underwater sensors for pose estimation. Section 9.2 summarizes the results achieved using the new pose estimation algorithm *SO(3) - PnP*. The limitations of the sensor prototype are elaborated in Section 9.3. Section 9.4 dives deep into further research to improve the sensor prototype presented in our research. The entire research is concluded in Section 9.5.

9.2 Results

In this thesis, we have presented a novel pose estimation algorithm *SO(3) - PnP* using computer vision (camera) and IMU. The target hardware is a deployable open-source platform(Raspberry Pi + Arduino). A novel underwater vision and IMU sensor has been prototyped to validate the pose estimation algorithm. The IMU sensor has high update frequency but is limited by noise while the camera measurements are very accurate but are very slow. Hence, a sensor fusion is implemented which optimizes the advantanges of both the sensors.

The algorithm has good accuracy and is computationally efficient for object points ($4 \leq n \leq 20$). In air, the position estimate is accurate to about $\pm 0.25cm$ in a sphere

of radius 2 m.and the orientation is accurate to about $\pm 0.5^0$. In water, the position estimate is accurate to about $\pm 0.5cm$.

The objective of the project was to create a novel localization system for a small-scale ROV which can be operated in a controlled environment. The sensors utilized in this project meet the requirements successfully. However, a number of challenges limit the scope of the current sensor system to be deployed in uncontrolled environments.

9.3 Limitations

A review of some of the limitations of the sensor are listed below:

1. The monocular vision is limited to about 50^0 field of vision in the plane of the camera {C}.

2. The depth estimation is not very accurate as 3D object points are projected onto the 2D camera image plane. The minimum number of landmark feature points need to be increased from $n = 4$ to $n = 8$ to achieve better accuracy.

3. The camera-IMU system should be mounted firmly to the ROV so as to ensure the pose of the sensor is the actual pose of the robot.

4. The landmark based pose estimation can only be used in a limited workspace.

5. The landmark is color differentiated from the background and hence minimal disturbance is assumed.

6. Underwater cameras need to be modeled for light refraction due to acrylic housing.

7. The prototype can currently be deployed at a depth of 20-30m. The limitation is due to the camera cable. Hence, a housing should be provided for the cables and it should be designed for high pressure underwater environments.

8. The *SO(3)* - *PnP* algorithm converges to a local minimum and hence the initial guess value of the Rotation matrix must be close to the global minimum. In the case of monocular vision, due to the limited field of view (about 50^0) the algorithm performs very well.

9.4 Future Work

In this section, we discuss some of the key features that can be implemented in the next generation of the sensors, both in hardware and software.

Hardware

1. The cables from the housing must be drawn inside a water-proof hose to ensure that the sensors work efficiently in underwater environments.
2. A thin housing layer can be used near the camera hole to minimize light refraction.
3. The limitations on field of vision and depth estimation can be compensated by using a stereo vision sensor. The future work will focus on pose estimation using stereo vision on the Raspberry-Pi Compute module (code revision) (Fig. 9.1).
4. The Raspberry Pi library for IMU can be used to remove the Arduino module completely.

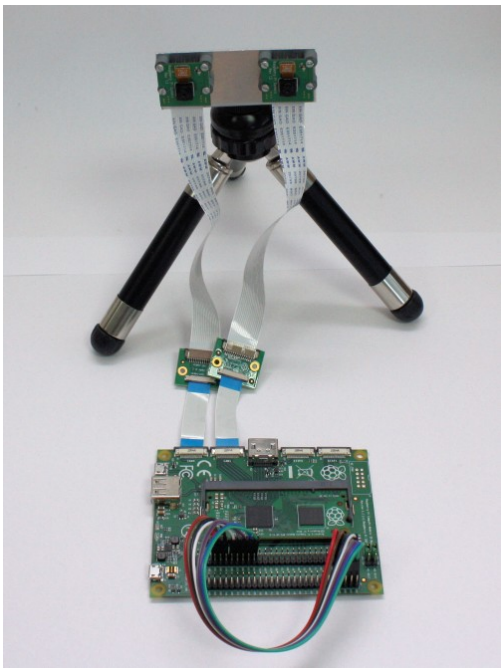


Figure 9.1: Raspberry Pi - Compute Module

Software

The *SO(3) - PnP* algorithm can be further studied and a closed form solution can be generated using Matrix Optimization. The current formulation on the error is shown in (9.1)

$$e = \beta(\alpha)\, \gamma({}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R}) \tag{9.1}$$

The error can now be simiplified as shown below in (9.2)

$$e = \beta(\alpha) \begin{bmatrix} \alpha_1 \, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \, {}^{G}P_1 \\ \alpha_2 \, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \, {}^{G}P_2 \\ \\ \alpha_n \, {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \, {}^{G}P_n \end{bmatrix} \tag{9.2}$$

$$e = \underbrace{\beta(\alpha)}_{2n\times 2n} \underbrace{\begin{bmatrix} \alpha_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \alpha_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \alpha_n \end{bmatrix}}_{(\alpha_{net})_{2n\times 3n}} \underbrace{\begin{bmatrix} {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & {}^{\mathbf{C}}_{\mathbf{G}}\mathbf{R} \end{bmatrix}}_{(\mathbf{R_{net}})_{3n\times 3n}} \underbrace{\begin{bmatrix} {}^{G}P_1 \\ {}^{G}P_2 \\ \vdots \\ {}^{G}P_n \end{bmatrix}}_{(\vec{a})_{3n\times 1}} \tag{9.3}$$

$$e = \underbrace{\beta(\alpha)\alpha_{net}}_{\kappa(\alpha)} \mathbf{R_{net}} \vec{a} \tag{9.4}$$

The cost function to optimze reduces to

$$\begin{aligned} \textbf{Objective:} \quad & \min_{\mathbf{R_{net}}} && J = \frac{1}{2}\, e^T(\mathbf{R_{net}})\, e(\mathbf{R_{net}}) \\ & \text{given} && \kappa(\alpha), \vec{a} \\ & \text{subject to} && \mathbf{R_{net}}(i,i) \underset{i,j=1:n;i\neq j}{} \in SO(3)\,,\ \mathbf{R_{net}}(i,j) = 0 \end{aligned}$$

Simplifying the cost function, we get

$$J = \frac{1}{2}\vec{a}^T {\mathbf{R_{net}}}^T \underbrace{\kappa(\alpha)^T\kappa(\alpha)}_{W_{3n\times 3n}} \mathbf{R_{net}}\vec{a} = \vec{a}^T {\mathbf{R_{net}}}^T W \mathbf{R_{net}}\vec{a} \tag{9.5}$$

The objective function now reduces to finding an optimal matrix $\mathbf{R_{net}}$, which optimizes the cost function J (in standard form), as shown below:

$$\begin{array}{lll} \textbf{Objective:} & \min\limits_{\mathbf{R_{net}}} & J = \vec{a}^T \mathbf{R_{net}^T} W \mathbf{R_{net}} \vec{a} \\ & \text{given} & W, \vec{a} \\ & \text{subject to} & \mathbf{R_{net}}(i,i) \in SO(3)\ ,\ \mathbf{R_{net}}(i,j) = \mathbf{0} \\ & & \quad i,j{=}1{:}n; i{\neq}j \end{array}$$

The above objective function can be solved using a variety of techniques like constrained optimization and SVD algorithms to achieve optimal closed form solution.

Looking into the near future, the current thesis indicates the possibility to investigate several areas in computer vision and underwater navigation.

9.5 Conclusions

In this thesis, the *PnP* algorithms have been extensively studied and a novel method to solve the pose estimation problem using *SO(3)* - *PnP* has been outlined. The merits and de-merits of the algorithm have been detailed and advanced methods for further research have been proposed. This thesis will be of particular interest to robotics enthusiasts to motivate them to pursue further research in underwater navigation using computer vision.

References

[1] V. Garro, F. Crosilla, and A. Fusiello, "Solving the pnp problem with anisotropic orthogonal procrustes analysis," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pp. 262–269, Oct 2012.

[2] L. Hsu, R. Costa, F. Lizarralde, and J. Da Cunha, "Dynamic positioning of remotely operated underwater vehicles," *Robotics Automation Magazine, IEEE*, vol. 7, pp. 21–31, Sep 2000.

[3] J. Heidemann, M. Stojanovic, and M. Zorzi, "Underwater sensor networks: Applications, advances, and challenges," *Philosophical Transactions of the Royal Society–A*, vol. 370, pp. 158–175, January 2012.

[4] X. Cheng, H. Shu, and Q. Liang, "A range-difference based self-positioning scheme for underwater acoustic sensor networks," in *Wireless Algorithms, Systems and Applications, 2007. WASA 2007. International Conference on*, pp. 38–43, Aug 2007.

[5] Z. Zhou, J.-H. Cui, and S. Zhou, "Localization for large-scale underwater sensor networks," in *NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet* (I. Akyildiz, R. Sivakumar, E. Ekici, J. Oliveira, and J. McNair, eds.), vol. 4479 of *Lecture Notes in Computer Science*, pp. 108–119, Springer Berlin Heidelberg, 2007.

[6] Z. Zhou, J.-H. Cui, and S. Zhou, "Localization for Large-Scale Underwater Sensor Networks," Tech. Rep. Technical Report: UbiNet-TR06-04 (BECAT/CSE-TR-06-15), UCONN CSE, 2006.

[7] W. Zhehao and L. Xia, "An improved underwater acoustic network localization algorithm," *Communications, China*, vol. 12, pp. 77–83, Mar 2015.

[8] G. Han, J. Jiang, L. Shu, Y. Xu, and F. Wang, "Localization algorithms of underwater wireless sensor networks: A survey," *Sensors*, vol. 12, no. 2, p. 2026, 2012.

[9] A. Teymorian, W. Cheng, L. Ma, X. Cheng, X. Lu, and Z. Lu, "3d underwater sensor network localization," *Mobile Computing, IEEE Transactions on*, vol. 8, pp. 1610–1621, Dec 2009.

[10] X. Tan and J. Li, "Cooperative positioning in underwater sensor networks," *Signal Processing, IEEE Transactions on*, vol. 58, pp. 5860–5871, Nov 2010.

[11] M. Erol-Kantarci, H. Mouftah, and S. Oktug, "A survey of architectures and localization techniques for underwater acoustic sensor networks," *Communications Surveys Tutorials, IEEE*, vol. 13, pp. 487–502, Third 2011.

[12] X. Cheng, H. Shu, Q. Liang, and D.-C. Du, "Silent positioning in underwater acoustic sensor networks," *Vehicular Technology, IEEE Transactions on*, vol. 57, pp. 1756–1766, May 2008.

[13] X. Gao, F. Zhang, M. Ito, K. Mishima, R. Onodera, N. Inagawa, and I. Yamamoto, "Prototype of positioning system for automatic motion control of underwater robot," in *OCEANS 2014 - TAIPEI*, pp. 1–6, April 2014.

[14] K. Wang, H. Gao, X. Xu, J. Jiang, and D. Yue, "An energy-efficient reliable data transmission scheme for complex environmental monitoring in underwater acoustic sensor networks," *Sensors Journal, IEEE*, vol. PP, no. 99, pp. 1–1, 2015.

[15] T. G. Kim, H.-T. Choi, and N. Y. Ko, "Concurrent estimation of robot pose and landmark locations in underwater robot," in *Control, Automation and Systems (ICCAS), 2013 13th International Conference on*, pp. 195–197, Oct 2013.

[16] J. Sun, J. Yu, A. Zhang, and F. Zhang, "Navigation positioning algorithm for underwater gliders in three-dimensional space," in *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015 IEEE International Conference on*, pp. 1269–1274, June 2015.

[17] H. Ahmad, J. Horgan, D. Toal, E. Omerdic, and S. Nolan, "A method for estimating the motion of a marine remotely operated vehicle using an underwater vision system," in *Control 2010, UKACC International Conference on*, pp. 1–5, Sept 2010.

[18] B. Zheng, H. Zheng, L. Zhao, Y. Gu, L. Sun, and Y. Sun, "Underwater 3d target positioning by inhomogeneous illumination based on binocular stereo vision," in *OCEANS, 2012 - Yeosu*, pp. 1–4, May 2012.

[19] G. Karras, S. Loizou, and K. Kyriakopoulos, "A visual-servoing scheme for semi-autonomous operation of an underwater robotic vehicle using an imu and a laser vision system," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pp. 5262–5267, May 2010.

[20] G. Karras, D. Panagou, and K. Kyriakopoulos, "Target-referenced localization of an underwater vehicle using a laser-based vision system," in *OCEANS 2006*, pp. 1–6, Sept 2006.

[21] Q. Wu, S. Li, Y. Hao, and F. Zhu, "A model-based monocular vision system for station keeping of an underwater vehicle," in *Robotics and Biomimetics (ROBIO). 2005 IEEE International Conference on*, pp. 450–454, 2005.

[22] S. Li, S. Tang, Q. Wu, and Y. Wang, "Positioning of an underwater vehicle based on model-known vision system," in *OCEANS, 2005. Proceedings of MTS/IEEE*, pp. 571–575 Vol. 1, 2005.

[23] R. M. Haralick, H. Joo, C. Lee, X. Zhuang, V. G. Vaidya, and M. B. Kim, "Pose estimation from corresponding point data," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, pp. 1426–1446, Nov 1989.

[24] D. Lowe, "Fitting parameterized three-dimensional models to images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 13, pp. 441–450, May 1991.

[25] J. A. Hesch and S. I. Roumeliotis, "A direct least-squares (dls) method for pnp," in *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, (Washington, DC, USA), pp. 383–390, IEEE Computer Society, 2011.

[26] P. Fiore, "Efficient linear solution of exterior orientation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, pp. 140–148, Feb 2001.

[27] F. Moreno-Noguer, V. Lepetit, and P. Fua, "Accurate non-iterative o(n) solution to the pnp problem," in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pp. 1–8, Oct 2007.

[28] L. Kneip, D. Scaramuzza, and R. Siegwart, "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 2969–2976, June 2011.

[29] S. Li, C. Xu, and M. Xie, "A robust o (n) solution to the perspective-n-point problem," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 7, pp. 1444–1450, 2012.

[30] A. Ansar and K. Daniilidis, "Linear pose estimation from points or lines," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, pp. 578–589, May 2003.

[31] C.-P. Lu, G. Hager, and E. Mjolsness, "Fast and globally convergent pose estimation from video images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, pp. 610–622, Jun 2000.

[32] X. shan Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, pp. 930–943, Aug 2003.

[33] D. Dementhon and L. Davis, "Model-based object pose in 25 lines of code," *International Journal of Computer Vision*, vol. 15, no. 1-2, pp. 123–141, 1995.

[34] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," *Journal of the Optical Society of America A*, vol. 4, no. 4, pp. 629–642, 1987.

[35] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1st ed., 1992.

[36] M. Bennani Dosse and J. Ten Berge, "Anisotropic orthogonal procrustes analysis," *Journal of Classification*, vol. 27, no. 1, pp. 111–128, 2010.

[37] D. Mindell and B. Bingham, "New archaeological uses of autonomous underwater vehicles," in *OCEANS, 2001. MTS/IEEE Conference and Exhibition*, vol. 1, pp. 555–558 vol.1, 2001.

[38] P. Zhang, E. Milios, and J. Gu, "Underwater robot localization using artificial visual landmarks," in *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, pp. 705–710, Aug 2004.

[39] G. Bradski, "Opencv library," *Dr. Dobb's Journal of Software Tools*, 2000.

[40] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3." http://eigen.tuxfamily.org, 2010.

[41] A. Munshi, D. Ginsburg, and D. Shreiner, *OpenGL(R) ES 2.0 Programming Guide*. Addison-Wesley Professional, 1 ed., 2008.

[42] D. Gebre-Egziabher, G. Elkaim, J. Powell, and B. Parkinson, "A gyro-free quaternion-based attitude determination system suitable for implementation using low cost sensors," in *Position Location and Navigation Symposium, IEEE 2000*, pp. 185–192, 2000.

[43] F. Mirzaei and S. Roumeliotis, "A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation," *Robotics, IEEE Transactions on*, vol. 24, pp. 1143–1156, Oct 2008.

[44] G. Bradski and A. Kaehler, *Learning OpenCV: Computer Vision in C++ with the OpenCV Library*. O'Reilly Media, Inc., 2nd ed., 2013.

[45] F. Markley, "Equivalence of two solutions of wahba problem," *The Journal of the Astronautical Sciences*, pp. 1–10, 2015.

[46] F. M. Mirzaei and S. I. Roumeliotis, "A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation," *IEEE Transactions on Robotics*, vol. 24, pp. 1143–1156, Oct 2008.

[47] A. Ford and A. Roberts, "Color space conversions," 1998.

[48] D. Simon, *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, 2006.

[49] E. Hecht, *Optics (4th Edition)*. Addison Wesley, 4 ed., Aug. 2001.

Appendix A

Matrix and Vector Properties

A few of the Matrix and Vector Properties used in this thesis are listed below

Identity I

$$\vec{x}^T\,\vec{y} = tr(\vec{y}\,\vec{x}^T) \tag{A.1}$$

Proof:

$$\vec{x}^T\,\vec{y} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = x_1y_1 + x_2y_2$$

$$tr(\vec{y}\,\vec{x}^T) = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix} = tr(\begin{bmatrix} y_1x_1 & y_1x_2 \\ y_2x_1 & y_2x_2 \end{bmatrix}) = y_1x_1 + y_2x_2$$

Identity II

$$tr(ABC) = tr(BCA) = tr(CAB) \tag{A.2}$$

Proof:

$$tr(ABC) = \sum_{ijk} A_{ij}B_{jk}C_{ki} = \underbrace{\sum_{ijk} C_{ki}A_{ij}B_{jk}}_{tr(CAB)} = \underbrace{\sum_{ijk} B_{jk}C_{ki}A_{ij}}_{tr(BCA)}$$

Identity III

$$\partial \mathbf{J}^* = tr(N\mathbf{\Omega}) = (n_{21} - n_{12})p + (n_{31} - n_{13})q + (n_{32} - n_{23})r \tag{A.3}$$

Proof:

$$N = \begin{bmatrix} n_{11} & n_{12} & n_{13} \\ n_{21} & n_{22} & n_{23} \\ n_{31} & n_{32} & n_{33} \end{bmatrix} \; ; \; \mathbf{\Omega} = \begin{bmatrix} 0 & p & q \\ -p & 0 & r \\ -q & -r & 0 \end{bmatrix}$$

$$N\mathbf{\Omega} = (N^+ + N^X)\mathbf{\Omega} = N^X\mathbf{\Omega}$$

$$N^X\mathbf{\Omega} = \begin{bmatrix} 0 & \frac{n_{12}-n_{21}}{2} & \frac{n_{13}-n_{31}}{2} \\ \frac{n_{21}-n_{12}}{2} & 0 & \frac{n_{23}-n_{32}}{2} \\ \frac{n_{31}-n_{13}}{2} & \frac{n_{32}-n_{23}}{2} & 0 \end{bmatrix} \begin{bmatrix} 0 & p & q \\ -p & 0 & r \\ -q & -r & 0 \end{bmatrix}$$

$$tr(N^X\mathbf{\Omega}) = (n_{21} - n_{12})p + (n_{31} - n_{13})q + (n_{32} - n_{23})r$$

Appendix B

Matlab Source Code - SO3-PnP

The source code of the SO3 based Gradient Descent algorithm for the PnP problem is presented below.

Listing B.1: Matlab Implementation of $SO_3 - PnP$

```
% ************************************************************************
% SO3-PnP
%
% Algorithm to estimate Position and Orientation from image data
% using SO3-PnP
% ************************************************************************
% Copyright(C)  <December 2015>  <Chandra P. Mangipudi, Perry Y. Li>
%
% This program is free software : you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program.If not, see <http://www.gnu.org/licenses/>.
% ************************************************************************
```

```
22 function [r_cam, R, error] = find_posSO3_fast(pt_img, R_guess, landmark)
23
24 MAX_ERROR_TOL=50;
25
26 W_img=1280;
27 H_img=720;
28 f = 1000;
29
30 n = length(landmark);
31 R = R_guess;
32
33 alpha=zeros(2*n,3);
34 gamma = zeros(2*n,1);
35
36 for k1=1:n,
37     alpha(2*k1-1:2*k1,:)=[f*eye(2), -pt_img(k1,:)'+[W_img/2;H_img/2]];
38 end
39
40 beta=eye(2*n)-alpha*pinv(alpha);
41 % display(alpha);
42 % display(beta);
43 Ra = R;
44 stepsize=pi/72;
45
46 [error,Veca,~] = an_error(Ra,0);
47
48 error_a=norm(error);
49 error_c=error_a; Rc=Ra; Vecc=Veca;
50
51 k1=0; k2=0; k0=0;
52
53 while (stepsize>pi/1440 && error_c>MAX_ERROR_TOL),
54     k0=k0+1;
55     while (Veca'*Vecc > 0 && error_c>MAX_ERROR_TOL)
56         Ra = Rc; Veca=Vecc;
57         Rc = Ra*quatrot(stepsize*Veca);
58         [error,Vecc,~] = an_error(Rc,0);
59         error_c = norm(error);
60 %         display(error_c);
```

```
61          k1=k1+1;
62      end;
63
64      while (Veca'*Vecc < 0 && error_c>MAX_ERROR_TOL)
65          stepsize=0.95/(1-Veca'*Vecc)*stepsize;  % Estimate angle
66          Rc = Ra*quatrot(stepsize*Veca);
67          [error,Vecc,~] = an_error(Rc,0);
68          error_c = norm(error);
69  %         display(error_c);
70          k2=k2+1;
71      end
72
73  end
74  R = Rc;
75  [error,~,r_cam] = an_error(R,1);
76
77  %% Function to compute error
78  function [error_out,vec,r_cam] = an_error(R,do_rgm)
79      for j=1:n
80          gamma(2*j-1:2*j,:)=alpha(2*j-1:2*j,:)*R*landmark(j,:)';
81      end
82
83      if do_rgm,
84          r_cam=-pinv(alpha)*(gamma);
85      else
86          r_cam=nan;
87      end;
88
89      error_out=-beta*(gamma);
90      N = zeros(3,3);
91      for j=1:n,
92          N = N+ landmark(j,:)'*error_out(2*j-1:2*j)'*alpha(2*j-1:2*j,:);
93      end
94
95      N=R*N;
96      vec=[N(2,3)-N(3,2); N(3,1)-N(1,3); N(1,2)-N(2,1)];
97      vec = vec/max(eps,norm(vec));
98  end
99  end
```

```
100 %% Function to rotate current Rotation matrix about axis(ax) and angle(ang)
101 function R = quatrot(vector)
102   ang = norm(vector);
103   ax = vector/norm(vector);
104   c=cos(ang); s=sin(ang);
105   R=[c+ax(1)^2*(1-c), ax(1)*ax(2)*(1-c)-ax(3)*s, ax(1)*ax(3)*(1-c)+ax(2)*s;
106      ax(1)*ax(2)*(1-c)+ax(3)*s, c+ax(2)^2*(1-c), ax(2)*ax(3)*(1-c)-ax(1)*s;
107      ax(1)*ax(3)*(1-c)-ax(2)*s, ax(2)*ax(3)*(1-c)+ax(1)*s, c+ax(3)^2*(1-c)];
108 end
```

Appendix C

C++ Source Code

A sample source code of the SO3 based Gradient Descent algorithm for the PnP problem in C++ using Eigen Library is presented below.

Listing C.1: C++ Implementation of $SO_3 - PnP$

```
/*************************************************************************
SO3-PnP

Algorithm to estimate Position and Orientation from image data
using SO3-PnP
*************************************************************************
Copyright(C)  <December 2015>  <Chandra P. Mangipudi, Perry Y. Li>

This program is free software : you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.If not, see <http://www.gnu.org/licenses/>.
*************************************************************************/
```

```
22 // <-------------------- Standard Libraries ---------------------->
23 #include <Eigen/Dense>
24 #include <iostream>
25 #include <stdio.h>
26 #include <math.h>
27
28 using namespace std;
29 using namespace Eigen;
30
31 // Global Constants
32 int W_img=1280, H_img=720;
33 int n = 4;
34
35 // Landmark points and Image points
36 MatrixXd cam_mat(2,2);
37 MatrixXd landmark(n, 3), pt_img(n, 2), pt_reproject(n,2);
38 VectorXd pt_temp(3);
39 float pts[4][2];
40
41 // Variables for SO3 algorithm
42 MatrixXd R(3, 3);
43 VectorXd  gamma(2 * n),  err(2 * n);
44 MatrixXd step(3, 3), N(3, 3);
45 MatrixXd alpha(2 * n, 3), Pr1(3, 2 * n), beta(2 * n, 2 * n);
46 MatrixXd R_cam(3, 3), Ra(3, 3), Rc(3, 3);
47 VectorXd r_cam(3), dummyrgm(3), Veca(3), Vecc(3);
48 float error_a, error_c, stepsize=M_PI/72;
49
50 // Iteration Variables for SO3 Iteration
51 int k3 = 0, k1 = 0, k2 = 0;
52 int k=0;
53
54 // Variables for Rotation matrix Simulation
55 float angx = 0, angy = M_PI / 8, angz = 0;
56
57 // Variables for Function quatrot
58 VectorXd ax;
59 MatrixXd R_ret(3, 3);
60 float ang,c_ang,s_ang;
```

```
61 // <-------- Function quatrot to get rotation matrix ----------->//
62 MatrixXd quatrot(VectorXd Vec)
63 {
64     ax=Vec;
65     ang = Vec.norm();
66     ax = ax / ang;
67     c_ang = cos(ang), s_ang = sin(ang);
68
69     R_ret << c_ang + ax(0)*ax(0)*(1 - c_ang),
70              ax(0)*ax(1)*(1 - c_ang) - ax(2)*s_ang,
71              ax(0)*ax(2)*(1 - c_ang)+ax(1)*s_ang,
72              ax(0)*ax(1)*(1 - c_ang) + ax(2)*s_ang,
73              c_ang + ax(1)*ax(1)*(1 - c_ang),
74              ax(1)*ax(2)*(1 - c_ang)-ax(0)*s_ang,
75              ax(0)*ax(2)*(1 - c_ang) - ax(1)*s_ang,
76              ax(1)*ax(2)*(1 - c_ang) + ax(0)*s_ang,
77              c_ang + ax(2) *ax(2) * (1 - c_ang);
78
79     return R_ret;
80
81 }
82
83 //<----------- Function to calcualte rotation error -------------->//
84 void err_calc(MatrixXd& R, int flag, VectorXd& err,
85               VectorXd& Vec, VectorXd& r_cam)
86 {
87     gamma=VectorXd::Zero(2*n);
88
89     N = MatrixXd::Zero(3,3);
90
91     for (k = 0; k < n; k++)
92     {
93         gamma.segment(2 * k, 2) = alpha.block(2 * k, 0, 2, 3)*
94                                   R*
95                                   (landmark.row(k)).transpose();
96     }
97
98     if (flag)
99         r_cam = Pr1*(- gamma);
```

```
100
101     err = beta*(- gamma);
102
103     for (k = 0; k < n; k++)
104     {
105         N = N + (landmark.row(k)).transpose()*
106             (err.block(2 * k, 0, 2, 1)).transpose()*
107             alpha.block(2 * k, 0, 2, 3);
108     }
109
110     step = R.transpose()*N.transpose() - N*R;
111
112     Vec << step(2, 1), step(0, 2), step(1, 0);
113
114     Vec /= Vec.norm();
115
116 }
117
118 // <------- Function to calcualte optimal rotation matrix -------->//
119 void findPosSO3(MatrixXd& pt_img, MatrixXd& R_guess)
120 {
121     R = R_guess;
122
123     for (int k = 0; k < n; k++)
124     {
125         alpha(2*k,2)=pt_img(k,0)-W_img/2;
126         alpha(2*k+1,2)=pt_img(k,1)-H_img/2;
127     }
128
129     Pr1 = (alpha.transpose()*alpha).inverse()*alpha.transpose();
130
131     beta = MatrixXd::Identity(2 * n, 2 * n) - alpha*Pr1;
132
133     Ra = R;
134
135     err_calc(Ra, 0, err, Veca, dummyrgm);
136
137     error_a = err.norm();
138     error_c = error_a;
```

```
139     Rc = Ra;
140     Vecc = Veca;
141 
142     stepsize=M_PI/72;
143 
144     while (stepsize > M_PI / 14400)
145     {
146         k2++;
147 
148         while (Veca.transpose()*Vecc>0)
149         {
150             error_a = error_c;
151             Ra = Rc;
152             Veca = Vecc;
153             Rc = Ra*quatrot(stepsize*Veca);
154             err_calc(Rc, 0, err, Vecc, dummyrgm);
155             error_c = err.norm();
156             k3++;
157         }
158 
159         while (Veca.transpose()*Vecc < 0)
160         {
161             stepsize *= 0.95 / (1 - Veca.transpose()*Vecc);
162             Rc = Ra*quatrot(stepsize*Veca);
163             err_calc(Rc, 1, err, Vecc, dummyrgm);
164             error_c = err.norm();
165             k1++;
166         }
167     }
168 
169     R = Rc;
170     err_calc(R, 1, err, dummyrgm, r_cam);
171 
172     R_cam = R.transpose();
173     r_cam = -R_cam*r_cam;
174 
175     R_guess = R;
176 }
177 
```

```
178 //<-------------- Function to initialize Matrices ---------------->//
179 void init_SO3()
180 {
181     // Initialize Rotation Matrices
182     R = MatrixXd::Identity(3, 3);
183     Ra = R;
184     Rc = R;
185     R_cam=R;
186 
187     // Initialize Landmark Points
188     landmark << 0, -15, 0, 0, 0, 0, 20, 0, 0, 10, -8, -10;
189 
190     // Initialize variables for iterative algorithm
191     Veca << 0, 0, 0;
192     dummyrgm << 0, 0, 0;
193 
194     r_cam<<0,0,-100;
195 
196     // Initialize Camera intrinsic parameters
197     cam_mat << 1164.0, 0, 0, 1164.0;
198 
199     // Initialize Object points on Image
200     pt_img << 482.4212354291506, 94.26442178696003,
201               536.7775078574271, 577.3537420567931,
202               1198.17578862227, 511.4722746472544,
203               779.2517415506529, 271.5419318022761;
204 
205     // Initialize portion of the gradient matrix
206     for(k=0;k<n;k++)
207         alpha.block(2*k, 0, 2, 2) = -cam_mat;
208 
209     cout << "Initialization Eigen Matrices Completed" << endl;
210 
211 }
212 
213 
214 
215 
216 
```

```
217 //<---------- Function to Reproject calculated position ---------->//
218 void reproject()
219 {
220     for(k=0;k<n;k++)
221     {
222         pt_temp=r_cam+R*(landmark.row(k)).transpose();
223         pt_reproject(k,0)=cam_mat(0,0)*pt_temp(0)/pt_temp(2)+W_img/2;
224         pt_reproject(k,1)=cam_mat(1,1)*pt_temp(1)/pt_temp(2)+H_img/2;
225     }
226 }
227
228 //<------------ Sample Main Function for Debugging --------------->//
229 int main()
230 {
231     init_SO3();
232
233     findPosSO3(pt_img, R);
234
235     reproject();
236
237     IOFormat HvyFmt(FullPrecision, 0, ", ", ";\n", "[", "]", "[", "]");
238
239     cout << "pt_reproject=" << endl;
240     cout << pt_reproject.format(HvyFmt) << endl;
241
242     cout << "errror=\n" << err << endl;
243
244     cout << "R=\n" << R_cam << endl;
245
246     cout << "r=\n" <<r_cam <<endl;
247
248     cout << endl << endl << "Enter any key to exit" << endl;
249     getchar();
250
251     return 0;
252 }
```

Appendix D

C++ Source Code

An OpenGL code comprising of a vertex shader and a fragment shader is shown below:

Listing D.1: Simple Vertex Shader

```
// Simple Vertex Shader
attribute vec4 vertex;
uniform vec2 offset;
uniform vec2 scale;
varying vec2 tcoord;
void main(void)
{
    vec4 pos = vertex;
    tcoord.xy = pos.xy;
    pos.xy = pos.xy*scale + offset;
    gl_Position = pos;
}
```

Listing D.2: Simple Fragment Shader

```
// Simple Fragment Shader
varying vec2 tcoord;
uniform sampler2D tex;
void main(void)
{
    gl_FragColor = texture2D(tex, tcoord);
}
```

Appendix E

Processing Java Source Code

A simple GUI to enable the user to track the ROV in a global co-ordinate is developed in Java - Processing. A schematic of the GUI is shown in fig. E.1.

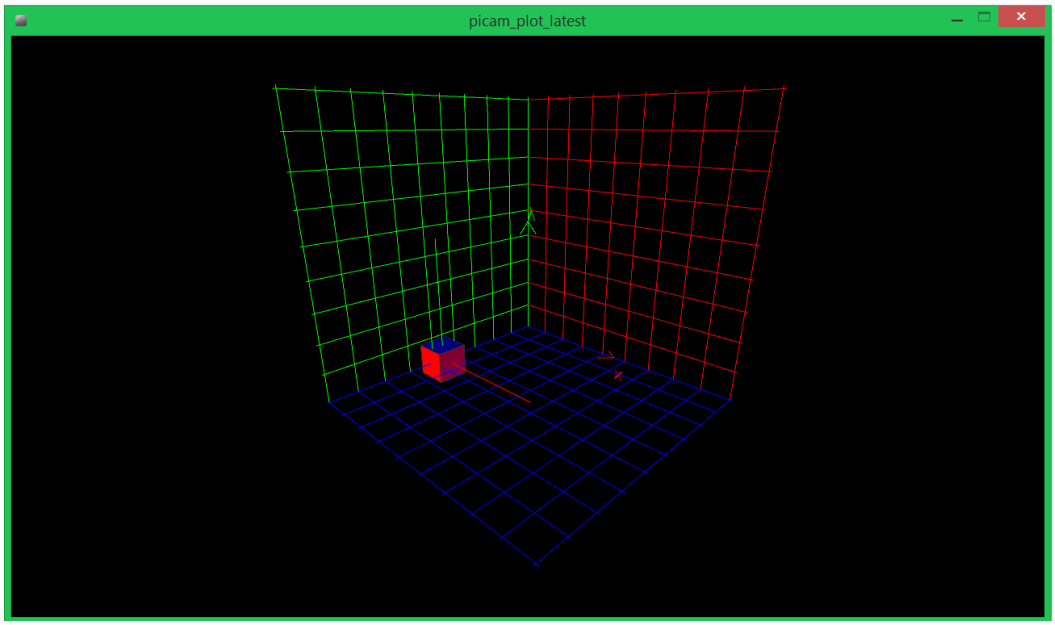


Figure E.1: GUI in Processing

The source code is shown below:

Listing E.1: GUI Processing Code

```
/*************************************************************
GUI

GUI to display the real time pose of the ROV in Global
Co-ordinates
*************************************************************
Copyright(C)  <December 2015>  <Chandra P. Mangipudi>

This program is free software : you can redistribute it
and/or modify it under the terms of the GNU General
Public License as published by the Free Software Foundation,
either version 3 of the License, or (at your option)
any later version.

This program is distributed in the hope that it
will be useful, but WITHOUT ANY WARRANTY; without
even the implied warranty of MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.See the
GNU General Public License for more details.

You should have received a copy of the
GNU General Public License along with this program.If not,
see <http://www.gnu.org/licenses/>.
*************************************************************/

import processing.net.*;
import processing.serial.*;
import hypermedia.net.*;
import processing.opengl.*;

import java.awt.*;
```

```
32 import java.io.DataOutputStream;
33 import java.io.BufferedOutputStream;
34 import java.io.FileOutputStream;
35 import java.io.FileNotFoundException;
36 import java.nio.ByteBuffer;
37 import java.nio.FloatBuffer;
38 import java.nio.ByteOrder;
39
40 import java.io.BufferedWriter;
41 import java.io.FileWriter;
42
43 // UDP receive variables
44 int num_states=65;
45
46 // Global axis definitions
47 int gl_ax_len=200;
48 int stepsize=20;
49
50 // 3-D mesh variables
51 int x_min=0, x_max=2*gl_ax_len;
52 int y_min=0, y_max=2*gl_ax_len;
53 int z_min=0, z_max=2*gl_ax_len;
54 int number_of_grid_lines=10;
55 float x_step = (x_max-x_min)/(number_of_grid_lines-1);
56 float y_step = (y_max-y_min)/(number_of_grid_lines-1);
57 float z_step = (z_max-z_min)/(number_of_grid_lines-1);
58
59 // Size of Image Window in Processing
60 int W_img=1280, H_img=720;
61 int box_len=25;              // Size of cube is 2*box_len
62
63 // Variables to rotate simulation model
```

```
64 int rotx=0, roty=0, rotz=0;
65
66 //float state[]=new float[26];
67 float state[]=new float[num_states];
68
69 //Looping Variables
70 int i=0,j=0,k=0;
71
72 // Variables for position of Camera w.r.t. World system
73 PVector pos=new PVector(0,0,200);
74 float[] eul=new float[3];
75 float[] q=new float[4];
76
77 // Define the UDP port
78 UDP udp;
79 PrintWriter fwrite;
80
81 void setup() {
82   size(W_img,H_img,P3D);
83   println("Starting Program");
84
85   fwrite = createWriter("data.txt");
86   fwrite.flush();
87
88   udp = new UDP( this, 6500 );
89   udp.log( true );  // <-- printout the connection activity
90   udp.listen( true );
91 }
92
93 void draw() {
94
95   clear();
```

```
96    camera(w/2, h/2, (h/2) / tan(PI/6), 0,0, 0,0,-1,0);
97    rotateY(radians(90));
98    scale(-1,1,1);
99
100   textSize(32);
101
102   draw_xyz_global();
103
104   pushMatrix();
105   translate(pos.x,pos.y,pos.z);
106   rotateZ(eul[2]);
107   rotateY(eul[1]);
108   rotateX(eul[0]);
109   draw_box();
110   popMatrix();
111
112 }
113
114 void draw_xyz_global()
115 {
116
117   background(0);
118   lights();
119
120   //Move Origin to Bottom Left
121   //Make 3-D co-ordinate axes
122   //x->rignt y-> top and z-> out of screen
123   //Default Co-ordinate System
124
125   // Draw the Global system Co-ordinate axes
126
127   stroke(255,0,0);
```

```
128   line(0,0,0,gl_ax_len,0,0);
129   line(gl_ax_len,0,0, gl_ax_len,gl_ax_len/10,gl_ax_len/10);
130   line(gl_ax_len,0,0, gl_ax_len-gl_ax_len/10,0,gl_ax_len/10);
131   fill(255,0,0);
132   text("x",gl_ax_len,-gl_ax_len/10,0);
133
134   stroke(0,255,0);
135   line(0,0,0,0,gl_ax_len,0);
136   line(0,gl_ax_len,0, gl_ax_len/10, gl_ax_len-gl_ax_len/10,0);
137   line(0,gl_ax_len,0, 0,gl_ax_len-gl_ax_len/10,gl_ax_len/10);
138   fill(0, 255,0);
139   text("y",0,gl_ax_len+gl_ax_len/10,0);
140
141   stroke(0,0,255);
142   line(0,0,0,0,0,gl_ax_len);
143   line(0,0,gl_ax_len, 0,gl_ax_len/10, gl_ax_len-gl_ax_len/10);
144   line(0,0,gl_ax_len, gl_ax_len/10,0,gl_ax_len-gl_ax_len/10);
145   fill(0, 0,255);
146   text("z",0,-gl_ax_len/10,gl_ax_len);
147
148   // Draw tiles for x-y-z world co-ordinate system
149
150   stroke(255,0,0);
151
152   // x-y plane grid lines parallel to y-axis
153    pushMatrix();
154      for(int j=0; j<number_of_grid_lines; j++)
155      {
156        line(x_min, 0, 0, x_max, 0, 0);
157        translate(0, y_step, 0);
158      }
159    popMatrix();
```

```
160
161     // x-y plane grid lines parallel to x-axis
162     pushMatrix();
163       for(int j=0; j<number_of_grid_lines; j++)
164       {
165         line(0, y_min, 0, 0, y_max, 0);
166         translate(x_step, 0, 0);
167       }
168      popMatrix();
169
170   stroke(0,255,0);
171
172  // y-z plane grid lines
173     pushMatrix();
174       for(int j=0; j<number_of_grid_lines; j++)
175       {
176         line(0, y_min, 0, 0, y_max, 0);
177         translate(0, 0, z_step);
178       }
179     popMatrix();
180
181     pushMatrix();
182       for(int j=0; j<number_of_grid_lines; j++)
183       {
184         line(0, 0, z_min, 0, 0, z_max);
185         translate(0, y_step, 0);
186       }
187     popMatrix();
188
189     stroke(0,0,255);
190     // x-y plane grid lines parallel to x-axis
191     pushMatrix();
```

```
192       for(int j=0; j<number_of_grid_lines; j++)
193       {
194         line(x_min, 0, 0, x_max, 0, 0);
195         translate(0, 0, z_step);
196       }
197     popMatrix();
198 
199     pushMatrix();
200       for(int j=0; j<number_of_grid_lines; j++)
201       {
202         line(0, 0, z_min, 0, 0, z_max);
203         translate(x_step, 0, 0);
204       }
205     popMatrix();
206 
207 }
208 
209 void draw_box()
210 {
211   beginShape(QUADS);
212 
213   stroke(50);
214   fill(255,0,0);
215 
216   // +Z "front" face
217   vertex(-box_len, -box_len,  box_len);
218   vertex( box_len, -box_len,  box_len);
219   vertex( box_len,  box_len,  box_len);
220   vertex(-box_len,  box_len,  box_len);
221 
222   fill(0,255,0);
223   // -Z "back" face
```

```
224   vertex( box_len, -box_len, -box_len);
225   vertex(-box_len, -box_len, -box_len);
226   vertex(-box_len,  box_len, -box_len);
227   vertex( box_len,  box_len, -box_len);
228
229   fill(0,0,255);
230   // +Y "bottom" face
231   vertex(-box_len,  box_len,  box_len);
232   vertex( box_len,  box_len,  box_len);
233   vertex( box_len,  box_len, -box_len);
234   vertex(-box_len,  box_len, -box_len);
235
236   fill(128,128,128);
237   // -Y "top" face
238   vertex(-box_len, -box_len, -box_len);
239   vertex( box_len, -box_len, -box_len);
240   vertex( box_len, -box_len,  box_len);
241   vertex(-box_len, -box_len,  box_len);
242
243   fill(255,0,100);
244   // +X "right" face
245   vertex( box_len, -box_len,  box_len);
246   vertex( box_len, -box_len, -box_len);
247   vertex( box_len,  box_len, -box_len);
248   vertex( box_len,  box_len,  box_len);
249
250   fill(70,255,200);
251   // -X "left" face
252   vertex(-box_len, -box_len, -box_len);
253   vertex(-box_len, -box_len,  box_len);
254   vertex(-box_len,  box_len,  box_len);
255   vertex(-box_len,  box_len, -box_len);
```

```
256
257   endShape();
258
259   // Axis of Camera system
260   stroke(255,0,0);
261   line(0,0,0,gl_ax_len,0,0);
262   stroke(0,255,0);
263   line(0,0,0,0,gl_ax_len,0);
264   stroke(0,0,255);
265   line(0,0,0,0,0,gl_ax_len);
266 }
267
268 // Function to receive UDP data from Arduino(debug)
269 // Pi(Wifi- Actual test)
270 void receive( byte[] data, String ip, int port )
271               throws IOException {
272   // Processing defaults to read in network byte order.
273   // ByteBuffers are good for unpacking data.
274
275   byte b;
276   int b_int;
277   //println("Receiving Messages");
278
279   ByteBuffer bb = ByteBuffer.wrap(data);
280   bb.order(ByteOrder.LITTLE_ENDIAN);   // Arduino or PC
281
282     for (int i=0; i < num_states; i=i+1)
283           state[i]=bb.getFloat(i*4);
284
285   q[0]=state[5];
286   q[1]=-state[6];
287   q[2]=-state[7];
```

```
288   q[3]=-state[8];
289
290   for(int i=0; i<num_states-1;i++)
291     fwrite.print(state[i]+",");
292   fwrite.println(state[num_states-1]);
293
294   //println("t="+state[0]);
295   q2eul();
296 }
297
298 // Function to convert from qernions to Euler Angles
299 void q2eul()
300 {
301  eul[0]=atan2(2*(q[0]*q[1]+q[2]*q[3]),
302         q[0]*q[0]+q[3]*q[3]-q[1]*q[1]-q[2]*q[2]);
303
304  eul[1]=asin(2*(q[0]*q[2]-q[1]*q[3]));
305
306  eul[2]=atan2(2*(q[0]*q[3]+q[1]*q[2]),
307         q[0]*q[0]+q[1]*q[1]-q[2]*q[2]-q[3]*q[3]);
308 }
309
310 void keyPressed()
311 {
312  if(key=='a')
313   {
314    pos.x=0;
315    pos.y=0;
316    pos.z=0;
317   }
318 }
```

Glossary

Arduino Due A popular open source ARM based microcontroller. 15

Estimation Determination of system state using measurements from sensors. 54

Gravity level bench A workbench which is normal to the earth's gravity. 39

Interrupts A hardware notification to stop the stack and to execute the code in the interrupt. 22

Kalman filter A novel way of combining data from different sensors based on their error probability. 84

Landmark An object whose geometry is known in global co-ordinates $\{G\}$. 15

Mercury indicator An object to ensure that the surface is exactly perpendicular to the earth's gravity. 39

Nodes Small devices which have acoustic transmitters and receivers. 4

Pose The position and orientation of an object w.r.t. a co-ordinate system. 54

Range-differencing Estimating position using distance to multiple known locations. 4

Raspberry Pi A pocket sized linux based personal computer which operates on ARM Cortex MCU. 20

Acronyms

$SO(3) - PnP$ Linear Least Squares - Gradient SO(3). 15

SO(3) Special Orthogonal Group. 17

3D 3 Dimensional. 29

API Application Programming Interface. 23

AUV Autonomous Underwater Vehicle. 6

CCCS Camera Centered Co-ordinate System. 52

CGR Cayley-Gibbs-Rodriguez. 14

CLM Concurrent Localization and Mapping. 10

CPU Central Processing Unit. 21

CSI Camera Serial Interface. 21

DAQ Digital Acquisition System. 22

DFT Discrete Fourier Transform. 45

DLS Direct Least Squares. 13

E-PnP Efficient-PnP. 14

ES Embedded Systems. 42

GLSL OpenGL Shading Language. 42

GPIO General Purpose Input Output. 23

GPS Global Positioning System. 6

GPU Graphical Processing Unit. 21

I2C Inter-Ic Bus. 23

IMU Inertial Measurement Unit. 16

LAB L-Lightness, A,B-chromaticity. 44

LHM Lu-Hager-Mjolsness. 12

MEMS Micro-Electro Mechanical Systems. 16

MIPI Mobile Industry Processor Interface. 23

OpenCV Open Computer Vision. 42

OpenGL Open Graphics Library. 42

P-PnP Procrustes-PnP. 12

PnP Perspective n Point. 11

R-PnP Robust-PnP. 15

RGB Red Green Blue. 44

ROI Region of Interest. 47

ROV Remotely Operated Vehicle. 6

RWCS Real World Co-ordinate System. 52

SIFT Scale Invariant Feature Transform. 10

SLAM Simultaneous Localization and Mapping. 10

SoC System on a Chip. 21

SPI Serial Peripheral Interface. 23

SVD Singular Value Decomposition. 14, 39

UDP User Datagram Protocol. 23

UWSN Underwater Sensor Networks. 9

YUV Y-Luma U,V- chrominance. 44