Robot Motion Tracking Using Time-of-flight And Structured Light Sensors For Indoor Navigation

by

KARAN VISHNU RAO

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2018

To my father and mother for giving me this opportunity and believing in me.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Kamesh Subbarao, for believing in me and giving me this wonderful opportunity to learn. I am grateful to him for guiding me and teaching me all those interesting and useful tools which equipped me to tackle the problem in a practical sense.

I would like to thank my committee members Dr. Alan Bowling and Dr. Ratan Kumar for taking time to serve on the committee.

I would like to thank my friends at the Aerospace Systems Laboratory, Murali, Pengkai, Ameya, Denish, Paul, Roopak, Rajnish, Diganta, Katy, Mitchel, Ziad and Abel for making my master's program fruitful. To Murali, Rajnish, Roopak and Ameya for helping me when there was an emergency.
Special thanks to my Father and Mother for believing in me and giving me this opportunity and for all the sacrifices that they made. This degree is as much as my parents' as it is mine. Thanks to my brother for all the fun and support.

April 19, 2018

ABSTRACT

Robot Motion Tracking Using Time-of-flight And Structured Light Sensors For Indoor Navigation

KARAN VISHNU RAO, M.S

The University of Texas at Arlington, 2018

Supervising Professor: Kamesh Subbarao

Simultaneous localization and mapping for mobile robots has been an active research field for several years with focus on the problems of accuracy and dependability of the data from the robot's peripherals. The position estimates for mobile and industrial robots is usually achieved by using the information from the global positioning system in an outdoor environment and for indoor environment, technologies such as LiDAR sensors and Infrared (IR) camera based motion capture systems such as VICON^{TM} are used. The main issue with these approach for indoor navigation is the monetary cost to associated with these technologies.

The purpose of this thesis is to provide a different approach for position and motion estimation of a robot for indoor localization and navigation. The presented position estimation technique is developed as a cost effective and viable replacement for the above mentioned indoor navigation systems and other GPS denied areas. The problem of tracking the robot is handled by using a triangulation technique which uses depth measurement sensors. The experimental setup is based on multiple sensors running on individual computers, connected via a wireless network. The sensors used in this setup are characterized with regards to their localization capability. The position and motion estimation technique is experimentally verified by using the sensor work-space environment setup under different working conditions.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Robots have been used in the manufacturing sector for a long time. With the shift in warehouse technology to make way for autonomous aerial and ground vehicles for handling and delivering packages to consumers, path planning has come to the forefront as a major research problem. The concept of autonomous path planning has been handled with several different solutions. For an accurate path planning it is key for the vehicle to locate itself in the working environment. This is referred to as the localization problem in the unmanned vehicle systems community.

The localization problem is usually dealt with the use of global positioning system (GPS). This is a viable option when the autonomous vehicle is in an outdoor environment, since the GPS provides accurate feedback. In an indoor environment, for example a warehouse, the GPS data is significantly erroneous.

This has sparked interest in the minds of the researcher to find a viable solution to this indoor localization problem, especially in a GPS denied environment. Researcher have turned their attention to computer vision algorithms and range measurement sensors to solve the simultaneous localization and mapping problem.

There has been innovative work done in this field using Kinect sensor for the computer vision ethos for the main reason being a cost effective sensor. This was the driving factor to tackle the problem presented in this thesis with the use of the Kinect sensors.

In this work, two different technologies of range measurement sensors are incorporated. The primary goal here is to show how the fusion of these sensors can affect the position estimates and also to demonstrate the robustness of the position estimation algorithm with different range measurement sensors.

In the past, the RGB camera output along with computer vision algorithm was the cornerstone for detecting and tracking the objects in the field of view. In this thesis, focus is placed on the IR depth sensor along with camera integrated in the Kinect, to provide the range measurement to unique tags generated by a Robot Operating System (ROS) package. By using this data, an Extended Kalman Filter(EKF) is implemented to estimate the position of the unique tag in real-time.

1.2 Related Work

Researchers have come up with numerous solutions to tackle the position estimation problem. Immense work is going on in finding the solutions for motion tracking in autonomous robots in GPS denied environments. In [1], the authors use vision data to navigate an unknown indoor GPS-denied environment. They showed that their algorithms for the guidance and control of an aerial vehicle is a viable option in the real environment. In [2], a quadrotor helicopter was equipped with a laser rangefinder and successfully implemented an EKF for data fusion. The authors of the paper [3], use a passive radio frequency identification RFID-assisted mobile robot system for mapping and surveillance of indoor environments.

The feasibility of using a structured-light range sensor for mobile outdoor and indoor robots is discussed in [4]. An in-depth analysis and comparison of the structured light sensor and time-of-flight sensor is very well documented in [5].

When cost effective range measurement devices such as Kinect v1 was introduced to the world in 2010, there was a sudden exploration expedition to utilize this tool which was primarily developed for the gaming community. The scientific community realized that this tool enabled to solve critical problems related to the unmanned vehicle systems. Up until the introduction of this device, the solutions were expensive range measurement sensors or complex computer vision algorithms used in stereo vision camera systems, which were computationally expensive.

There has been significant work done in characterizing the Kinect v1 and Kinect v2 sensors [6], [7]. The calibration of a structured light sensor is explained and demostrated in [8]. A detailed analysis of the depth measurement attribute of the Kinect is discussed in [9]. The applications and roadblocks for the Kinect and its RGBD image output is detailed in [10]. The authors show how to install the drivers for operating systems like Ubuntu and also provide an insight to various open source libraries that is available for developers to work on the Kinect sensor. The authors in [11], show how to integrate the Kinect sensor with Simulink for real-time object tracking. Here they make use of a custom "VU-Kinect" block in order to access the sensor data.

People have done incredible work in the past and found new ways to use this tool in research problems across the world. Although the primary application for the Kinect sensor was to capture the motion of human skeleton for gaming purposes, researcher have recognized this in-expensive, powerful tool to achieve successful results for numerous real world problems. With Kinect sensors, people have developed depth map to control the altitude of a quadrotor helicopter [12]. Kinect sensors have been incorporated to provide a natural user interface for controlling 3-D virtual globes as described in [13]. Kinect is shown to be a promising virtual reality (VR) neurological rehabilitation tool for use in the clinic and home environment in [14]. The authors of this paper demonstrate the comparison of the motion tracking between the low-cost Microsoft Kinect and a high-cost multi-camera lab-based system OptiTrack. Kinect has been used for computer vision applications in autonomous vehicles [15] as a low cost range sensor in place of stereovision, LiDAR and RADAR which have their limitations and higher price. The authors of [16], demonstrate their algorithm which detects objects using color and depth segmentation with Kinect sensor. They also discuss how Kinect is a less expensive option compared to other technologies like LADAR sensors and more robust than stereo vision system. In [17], the author compares the Kinect sensor and VICON^{TM}. A more fundamental issue of computer vision and object recognition is addressed by the authors in [18], who propose a depth kernel descriptors based on the Kinect sensor output. Motion tracking using multiple depth sensors for real-time application can be seen in [19].

The paper [20] shows how kinects can work as a perception sensor network to track a human. Most of the work seen here have focused on fusing data from the depth sensor and the RGB camera output of the Kinect, this involves complex computer vision algorithms. The rich information provided by the Kinect sensor is incorporated by a particle filtering framework as shown in [21], for a visual tracking problem. Also the authors of the paper [22] use the Kinect sensor output into a particle filter frame-work to estimate the scene as a collection of object poses.

There has been lot of work done previously in the field of obstacle detection and avoidance using kinects, in [23], the author has shown how Kinect sensor along with obstacle avoidance algorithm, developed by them can detect and perform evasive maneuver within one meter distance. [24] is an other example of the use of Kinect for obstacle avoidance instead of expensive LiDARs. LiDAR can normally only detect obstacles on a plane level so the authors of [25] made use of Kinect to detect negative obstacles in an indoor environment. In this paper [26], it has been demontrated how Kinect v2 performs 2D SLAM in an indoor environment with respect to a 2D laser scanner. In [27], it can be seen how cognitive service robots mounted with Kinect sensors have been used in perceiving,learning and recognizing 3D objects. This paper successfully demonstrates, open-ended learning capabilities of the robot using Kinect sensor.

It is key to have a framework for real-time communication between sensor data for the solution proposed in this thesis. Robot Operating System (ROS) provides the answer. There has been significant work done in characterizing ROS as a useful framework for communication. In [28], the authors have demonstrated this capability of ROS for the application of obstacle avoidance for a mobile robot.

Work has been done previously where Kinect v2 was used to perform a 3D reconstruction of indoor scenes in [29]. Kinect sensor was directly mounted on an UGV for its navigation and mapping in [30]. This paper demonstrates the use of ROS and Kinect along with the issues with the interaction between them.

1.3 Problem Description

This thesis addresses the problem of position estimation and motion tracking in a GPS denied environment by using a triangulation technique as seen in Fig. 1.1 to obtain a closed-form analytic solution and an Extended Kalman Filter is used when one of the sensor data is not obtained. This thesis also validates position estimation and motion tracking for multiple robots in the work space.

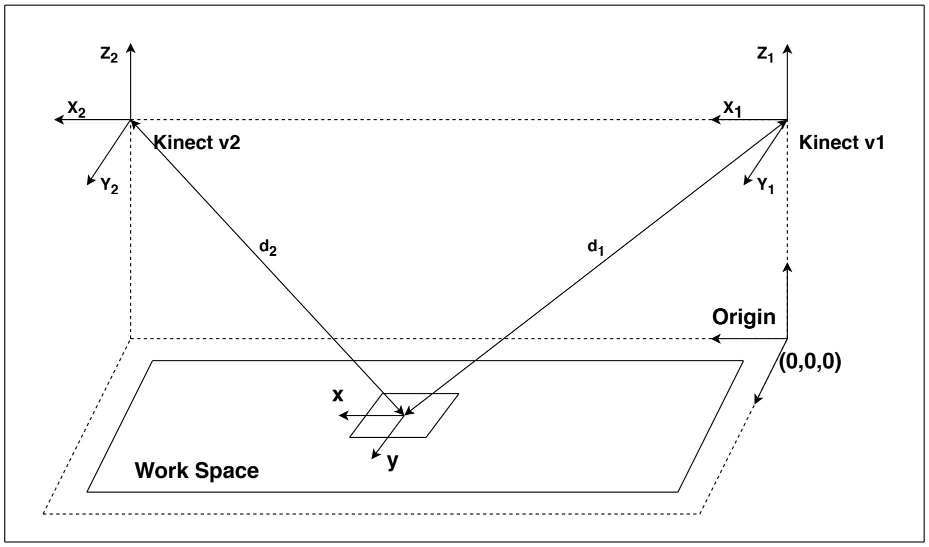


Figure 1.1: Frames for triangulation and work space

Solution Methodology

There are two sensors used in this work, Kinect v1 and Kinect v2, which are introduced in chapter 2. These sensors are characterized and the results are presented in chapter 6. The data from the two sensors are obtained via the ROS network set up, which is described in chapter 3. The depth measurements given as,

$$d_1 = \sqrt{(X_1 - x)^2 + (Y_1 - y)^2 + (Z_1 - z)^2} \tag{1.1}$$

$$d_2 = \sqrt{(X_2 - x)^2 + (Y_2 - y)^2 + (Z_2 - z)^2} \tag{1.2}$$

The depth measurements as shown in Eq. 1.1 and Eq. 1.2, are used to compute the position of the robot. The analytic solution is obtained and the motion tracking of the robot is tested. Next, the position estimation and the motion tracking is performed using the EKF for a single robot, which is then extended to multiple robots.

1.4 Thesis Outline

This thesis is organized as follows: In Chapter 2, Kinect sensors are described in detain. Chapter 3 introduces the augmented reality ROS package and AR marker tags. Chapter 4, introduces the triangulation technique used to find the position of the markers. In Chapter 5, the experimental setup for testing the analytic solution and the EKF algorithm is presented. Chapter 6 presents the sensor and algorithm analysis and the experimental results. Finally, in Chapter 7, concluding remarks are stated.

CHAPTER 2

SENSORS

2.1 Sensor Technologies

Robots require a method to perceive the world in order for the autonomous behavior to take effect. Sensors make this possible. A sensor is a device, module, or subsystem whose purpose is to detect events or changes in its environment and send the information to other electronics, frequently a computer processor. A sensor is always used with other electronics, whether as simple as a light or as complex as a computer [31].

There are numerous categories of sensors that are used for capturing various real-world phenomena. The main category of sensors that this thesis focuses on are proximity sensors. A proximity sensor often emits an electromagnetic field or a beam of electromagnetic radiation (infrared, for instance), and looks for changes in the field or return signal. The object being sensed is often referred to as the proximity sensor's target. Proximity sensors can have a high reliability and long functional life because of the absence of mechanical parts and lack of physical contact between sensor and the sensed object. Some of the different types of proximity sensors are Capacitive, Doppler effect, Eddy-current, Inductive, Magnetic, Optical, Photoelectric, Photocell, Laser rangefinder, Passive thermal infrared, Radar, Reflection of ionizing radiation, Sonar (typically active or passive), Ultrasonic sensor, Fiber optics sensor and Hall effect sensor.

In this work, two categories of proximity sensors are used, namely structured-light sensor and time-of-flight sensors. These are studied in-depth in the following sections of this chapter.

2.2 Structured Light Sensors

Structured light is the process of projecting a known pattern on to a work space. The way these patterns deform when striking surfaces allow vision systems to calculate the depth and surface information of the objects in the work space.

Principle

Projecting a unique pattern of light on a 3D object will distort the pattern and this can be used for geometric reconstruction of the 3D object. There are many other variants of structured light projection that are possible like patterns of parallel stripes, which is widely used. The structured light sensor uses a plane of laser light. The laser plane is generated by a cylindrical lens and illuminates a profile of range data in each camera image. The sensor has no moving optical components.

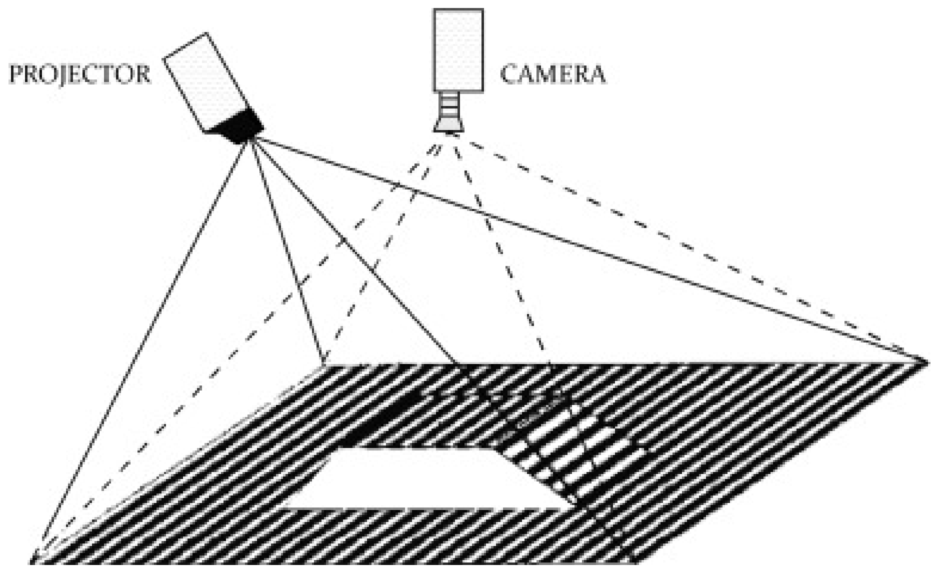


Figure 2.1: Principle behind Structured-Light technology
Source : https://www.sciencedirect.com/science/article/pii/S0143816616000166

Applications

There have been numerous applications of structured-light. One of the main areas is in perception sensors. Many companies have adopted this technology for their depth perception like Microsoft Kinect uses a pattern of projected infrared points to generate a 3D image of the work space [32]. Google project Tango SLAM utilizes structured light as one of the depth technologies [33]. Recently, Apple came out with Face ID system which works by projecting more than 30,000 infrared dots onto a face and produces a 3D facial map [34]. This is being used capture road pavement structure and roughness [35].

Kinect v1

Kinect is a motion sensing input device designed for Xbox 360 by Microsoft for console gaming purposes. The name Kinect is inspired by the words “kinetic”, meaning to be in motion, and “connect” meaning it “connects you to the entertainment and friends you love” [36]. It was first introduced in November 2010. Microsoft released a software development kit for windows in the year 2011, which opened up opportunities for people to use this device as a tool to solve problems beyond the gaming environment. The sensor contains a range chipset technology which was developed by Israeli developer PrimeSense. PrimeSense, used an infrared projector, camera and a special microchip to develop a system that generates a grid from which the range to a certain 3D object can be found. This is a form of structured-light sensor and they call it light coding, which employs a type of image-based 3D reconstruction. The light coding, process codes the scene with near-IR light i.e. scatters the IR light particles onto the scene, light that returns distorted depending upon where things are. The solution then uses a Complementary Metal-Oxide Semiconductor (CMOS) image sensor to read the coded light back from the scene using triangulation algorithms and extracts the 3D data.

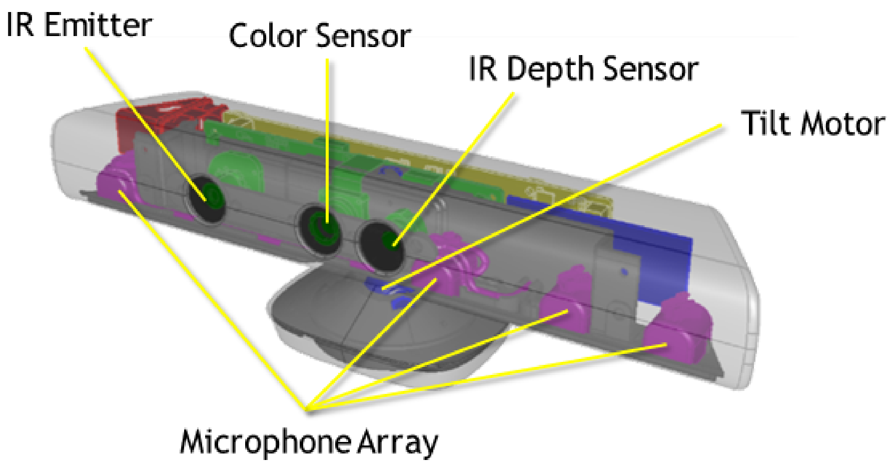


Figure 2.2: Kinect v1 components
Source: https://msdn.microsoft.com/en-us/library/jj131033.aspx

From Fig. 2.2, the parts of the Kinect v1 can been see in detail, which are responsible for the motion capturing.

The main part of the Kinect v1 are a pair of cameras, both of which incorporate CMOS image sensors from Aptina Imaging. The infrared camera uses the MT9M001 sensor with 5.2-micron pixels; larger pixels do well in low light and, with correct filtering, lend themselves nicely to infrared applications. MT9M112 sensor from the color camera provides the RGB input. Both sensors have 1.3-megapixel resolution. PrimeSense PS1080 is the system-on-chip which controls the infrared projector, processes inputs from the cameras and collects audio input. And the communication is via USB 2.0 with the application processor, a Marvell product whose Aspen die mark indecates that the PXA168a is low-power and low-cost component. Kinect v1 is also equiped with a pair of Wolfson Microelectronics WM8737L stereo A/D converters, with built-in microphone preamplifiers to accommodate the array of microphones. The Kinect v1 also houses a MEMS accelerometer, the Kionix KXSD9, this is a part of the tilt-control loop, which also includes the A3906 stepper and dc motor driver from Allegro Microsystems. Other noteworthy components include the uPD720114 USB hub controller from NEC and a pair from Texas Instruments: the TAS1020B USB audio streaming controller and ADS7830 8-bit, eight-channel A/D converter.[37]

Kinect v1	**Technical Specifications**
Viewing angle	43 vertical by 57 horizontal field of view
Vertical tilt range	27
Frame rate (depth and color stream)	30 frames per second (FPS)
Audio format	16-kHz, 24-bit mono pulse code modulation (PCM)
Depth Sensor	IR structured light depth sensor
Accelerometer characteristics	A 2G/4G/8G accelerometer configured for the 2G range, with a 1 accuracy upper limit.

Table 2.1: Kinect v1 specifications
Source : https://msdn.microsoft.com/en-us/library/jj131033.aspx

2.3 Time-of-Flight Sensors

Time-of-Flight refers to the time that an object, particle or acoustic, electromagnetic or other wave needs to travel a distance through a medium. The time measured as mentioned can be used for a time standard as a way to measure velocity or path length through a given medium. A time-of-flight camera is a range imaging camera system that resolves distance based on the known speed of light, measuring the time-of-flight of a light signal between the camera and the subject for each point of the image. The time-of-flight camera is a type of LiDAR, in which the work space is captured with each laser or light pulse.

Principle

The time-of-flight generally works by actively illuminating the work area using near infrared intensity-modulated, periodic light. Due to the distance between the camera and the object and known speed of light, a time shift is caused in the optical signal which is equivalent to a phase shift in the periodic signal. This shift is detected in each sensor pixel. This time shift is transformed into the sensor-object distance as the light has to travel the distance twice.

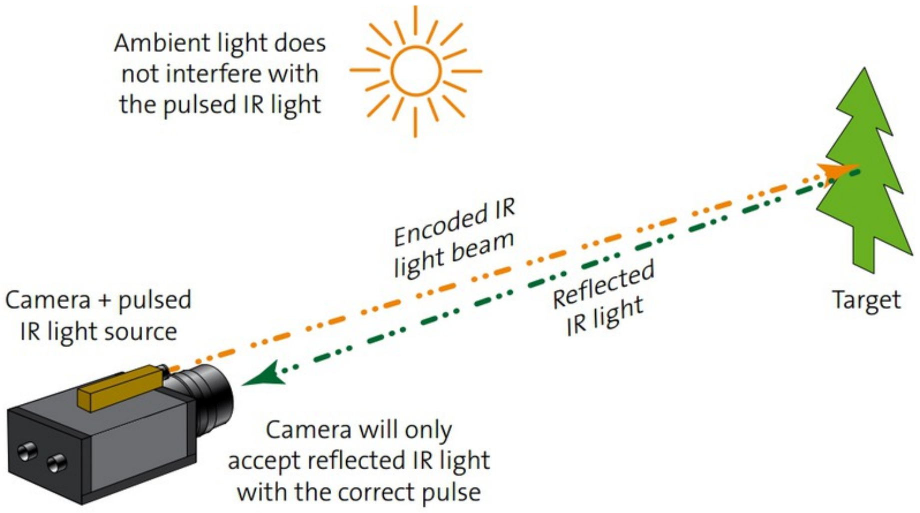

Ambient light does not interfere with the pulsed IR light
Encoded IR light beam
Reflected IR light
Target
Camera + pulsed IR light source
Camera will only accept reflected IR light with the correct pulse

Figure 2.3: Principle behind Time-of-Flight technology
Source: https://www.stemmer-imaging.co.uk/en/knowledge-base/cameras-3d-time-of-flight-cameras/

Applications

Time-of-Flight sensors are very popular class sensors which have numerous applications in the current time-line. These sensors find an important place in the field of robotics. They are used in the mobile robots for obstacle avoidance problems. They have been used to study the Earth's surface topography. Gaming is another major industry that uses this technology for motion capture and augmented reality gaming. These time-of-flight cameras are installed in vehicles to avoid collisions with pedestrians. These sensors play a key role in the advancement of machine vision.

Kinect v2

After the success of Kinect v1, Microsoft came out with an improved motion capture system for its Xbox One line of consoles for gaming. Kinect 2.0 or also referred to as Kinect v2 was released in 2013. Microsoft changed the core technology from structured light sensor by PrimeSense to a time-of-flight sensor. The new Kinect is found to be more accurate and has better depth measurements. It has a wider field of vision. This system can track without visible light instead uses an active IR sensor.

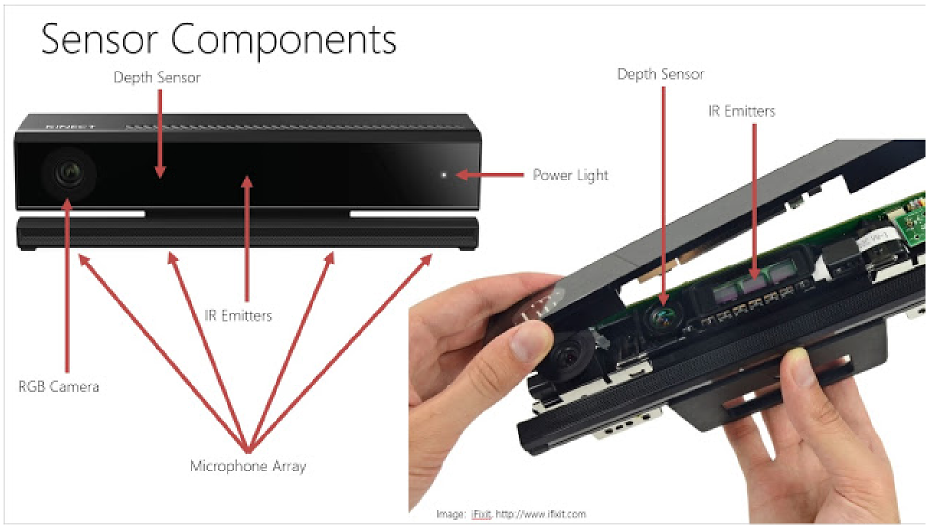


Figure 2.4: Kinect v2 components
Source: https://www.physio-pedia.com/The_emerging_role_of_Microsoft_Kinect_in_physiotherapy_rehabilitation_for_stroke_patients

From Fig. 2.4, It can be observed that there is a distinct difference in the design and hardware construction of Kinect v2 from Kinect v1. The new Kinect v2 does not have a motor driven base instead, the angle of tilt must be adjusted manually. This is because the Kinect v2 has a wider field of view hence does not require to shift itself to focus the objects in the work space. Kinect v2 has an full HD, RGB camera along with IR emitters and a depth sensor. It is equipped with an array of microphone for audio inputs.

Kinect v2	**Technical Specifications**
Viewing angle	70 horizontal & 60 vertical field of view wide-angle lens
Vertical tilt range	27
Frame rate (depth and color stream)	19201080 (AKA 1080p) 30 fps 16:9 camera
Audio format	4 microphone array operating at 48 kHz
Depth Sensor	IR (infrared) TOF (Time-Of-Flight) depth sensor for 3D tracking
Accelerometer characteristics	Non-motorised manually hand-adjustable-only tilt

Table 2.2: Kinect v2 specifications
Source : http://123Kinect.com/everything-Kinect-2-one-place/43136/

By comparing the technical specifications of Kinect v1 and Kinect v2, it is clear that Kinect v2 is far more accurate in range measurements and motion capture. People have proved by various experiments the advantages of Kinect v2 as seen in [38]. This thesis uses both the generation of sensors since they represent two different and unique category of sensor technologies, namely structured light and time-of-flight.

CHAPTER 3

Data Acquisition

3.1 ROS

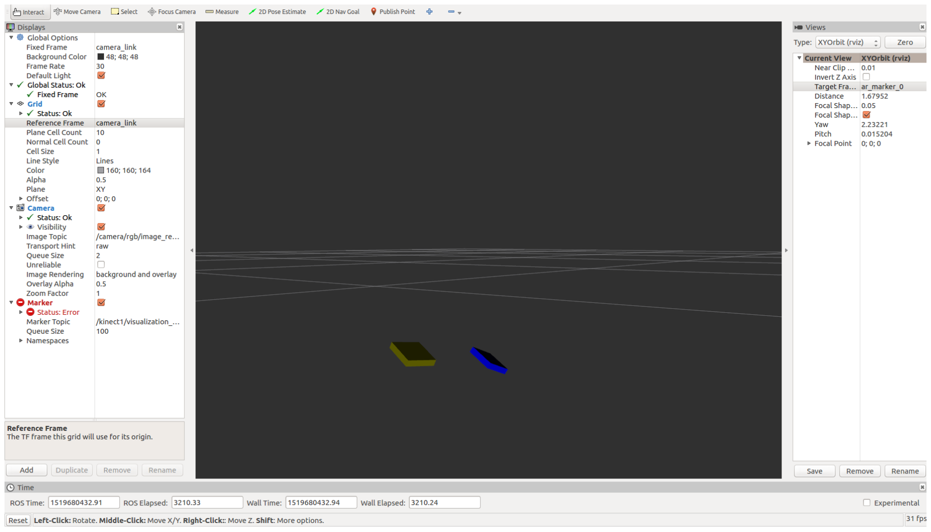
Robot Operating System (ROS) is not an operating system but rather a set of robotics software frameworks which provides services designed for heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. ROS is a low latency framework but real-time code can be integrated.

In [39] it can be seen how ROS is incorporated for visual SLAM in an indoor environment. ROS plays an important role in this thesis work. The range measurements from the sensors via low latency framework is obtained using ROS. ROS heterogeneous computer cluster capability is key for this research work. Each of the sensors is considered as a separate node which is connected to a single unique master node. All the nodes and the master node in ROS are connected via a single common Wifi connection. The Kinect sensors being used can interact with windows by using the Kinect SDK provided by Microsoft for developers but since ROS is not fully supported in Windows, Windows operating system was not used for this work. ROS is fully supported in Linux, hence the Ubuntu 16.04 LTS which is a distribution of Linux is used. But there is no SDK by Microsoft for Ubuntu. There are alternative, open-source software packages which are available on Ubuntu for interacting with the Kinect sensors. Openni_Kinect, wraps the OpenNI "natural interaction" drivers, as well as higher level libraries like skeleton and gesture tracking. This driver is officially supported by PrimeSense, has great performance, and provides the full capabilities of the sensor, including in-sensor registration for RGB and depth (no calibration required), support for different depth and RGB resolutions, and full audio support [40].

To access Kinect v1, openni_launch file must be launched in ROS and for this to work, two packages must be installed. First is OpenNI 1.5.4 and then secondly a sensor module to access the Kinect sensor, SensorKinect. The launch file creates nodelet grph to transform raw data from the device driver into suitable products for processing and visualization [41].

In order for Kinect v2 to interface with ROS, a ROS package must be installed and setup called iai_kinect2. It contains a collection of tools and libraries required to receive data from the Kinect v2 sensor for robotics application. It comes with a calibration tool for calibrating the IR sensor of the Kinect One to the RGB sensor and the depth measurements, along with a library for depth registration and a software bridge between libfreenect2 [42] and ROS. libfreenect2 is the driver for Kinect for windows v2 devices. The driver supports RGB, IR and depth image transfer and registration of RGB and depth images. Since image processing is tedious for the computer's processor, some of this load can be allocated to the GPU via GPU acceleration by installing OpenCL and CUDA for a Nvidia GPU.

For visualization purposes, a ROS package, rviz is used in this work. It provides camera output, point cloud output and depth points in a intuitive graphic user interface. This is important since it displays to the users what the Kinect cameras can see and what is their field of view.



Global Options	
Fixed Frame	camera_link
Background Color	■ 48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
✓ Fixed Frame	OK
Grid	✓
✓ Status: Ok	
Reference Frame	camera_link
Plane Cell Count	10
Normal Cell Count	0
Cell Size	1
Line Style	Lines
Color	▨ 160; 160; 164
Alpha	0.5
Plane	XY
Offset	0; 0; 0
Camera	✓
✓ Status: Ok	
Visibility	✓
Image Topic	/camera/rgb/image_re…
Transport Hint	raw
Queue Size	2
Unreliable	☐
Image Rendering	background and overlay
Overlay Alpha	0.5
Zoom Factor	1
Marker	✓
Status: Error	
Marker Topic	/kinect1/visualization_…
Queue Size	100
Namespaces	

Reference Frame
The TF frame this grid will use for its origin.

Type: XYOrbit (rviz)

Current View	XYOrbit (rviz)
Near Clip …	0.01
Invert Z Axis	☐
Target Fra…	ar_marker_0
Distance	1.67952
Focal Shap…	0.05
Focal Shap…	✓
Yaw	2.23221
Pitch	0.015204
Focal Point	0; 0; 0

ROS Time: 1519680432.91 ROS Elapsed: 3210.33 Wall Time: 1519680432.94 Wall Elapsed: 3210.24 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click::** Move Z. **Shift**: More options. 31 fps

Figure 3.1: Rviz used for visualization

3.2 Augmented Reality Package

There are numerous methods to achieve object detection and tracking. Most of these methods are based on complex algorithm for stereo vision camera setup. It makes sense to go this route when the focus is on characterizing the object since it is important in an obstacle avoidance scenario where determining what the object is, helps the robot maneuvering. In this thesis work, focus is on the position estimation and not on obstacle avoidance algorithm. For position estimation, it is important to continuously have a good detection rate of the object being tracked. This is achieved by using unique augmented reality marker tags.

Augmented Reality (AR) is a technique of superimposing virtual objects in the user's view of the real world. ALVAR is a set of SDKs and products that help create augmented reality applications. The ALVAR SDK is capable of creating AR applications with the most accurate, efficient and robust implementation for markers, multi marker based tracking. ALVAR core contains a low-level AR toolkit and fast marker detector which is key for this research work.

ar_track_alvar is a ROS package which is essentially a ROS wrapper for ALVAR. The main functions of this package are:

- It generates the AR tags of varying size, resolution, and data/ID encoding.
- Identifying and tracking the pose of individual AR tags, optionally integrating Kinect depth data (when a Kinect is available) for better pose estimates.
- Identifying and tracking the pose of "bundles" consisting of multiple tags. This allows for more stable pose estimates, robustness to occlusions, and tracking of multi-sided objects.

ALVAR is adept at handling a variety of lighting conditions, optical flow based tracking for more stable pose estimation. This tag identification method does not effect the identification time also when the number of tags increases.

This package is capable of handle both individual tags as well as multiple tags bundled together. There are 18 predefined unique markers which are 4.5 cm squares. The package provides options to generate other unique markers with different ID numbers, border widths and size. It is possible to identify and track the poses of multiple AR tags that are each considered individually. A node called individualMarkers is created. This individualMarkers node assumes that a Kinect is being used as the input device, so that depth data can be integrated for better pose estimates.

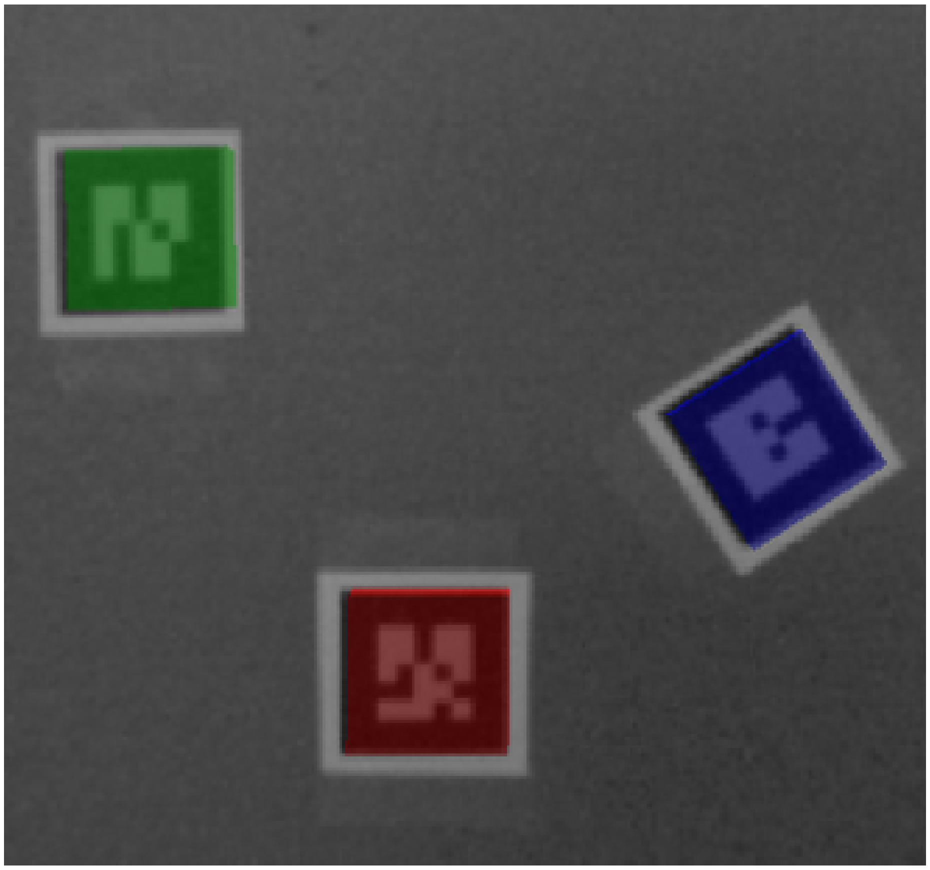


Figure 3.2: Individual markers visualized in rviz
Source : http://wiki.ros.org/ar_track_alvar

Sometimes it is advantageous to treat “bundles” of multiple tags as a single unit. For example, this can allow for the estimation of the pose of a many-sided object, even when some of the tags cannot be seen. It can also lead to more stable pose estimates and provide robustness to occlusion. In order to bundle the tags, first a master tag must be chosen. A tag bundle is defined by an XML file that lists a set of tag IDs and their positions relative to a “master” tag. The coordinate system for the rest of the tags is defined by the master tag.

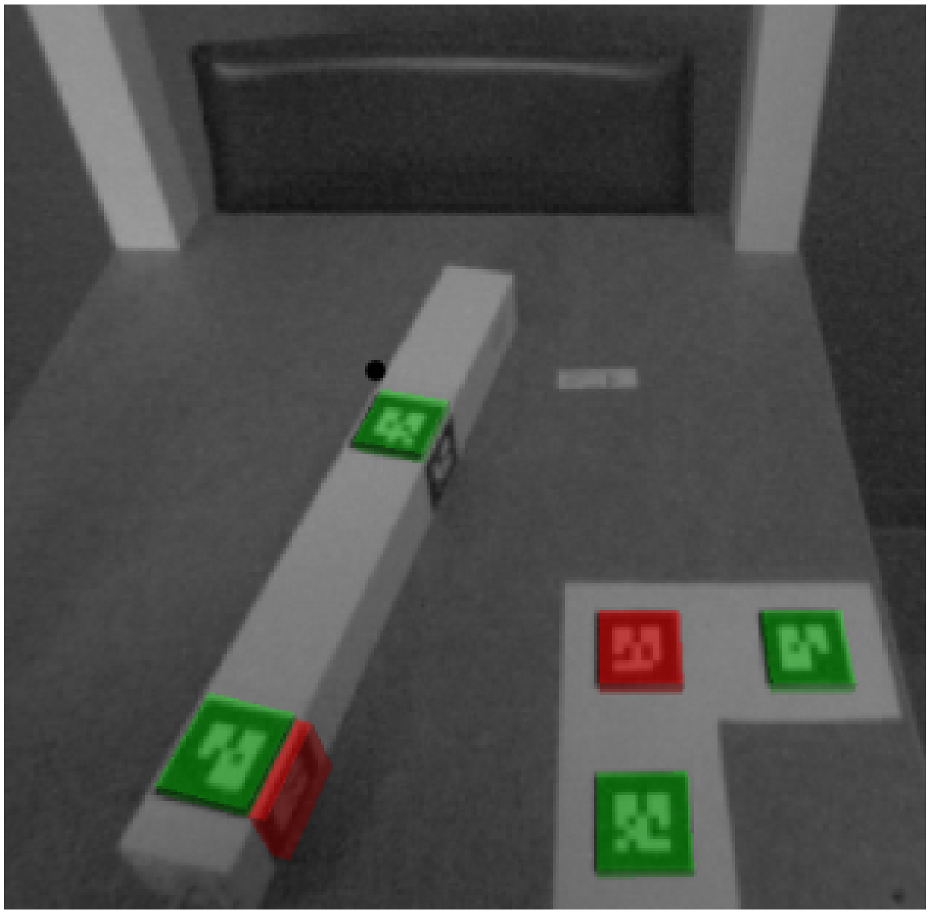


Figure 3.3: Multiple markers bundle visualized in rviz
Source : http://wiki.ros.org/ar_track_alvar

In Fig. 3.3 the bundled tags are visualized as green where as the other individual markers are visualized as red. This helps in demarcating the positive tags for pose estimation.

CHAPTER 4

Motion Tracking

4.1 Triangulation

In trigonometry and geometry, triangulation is the process of determining the location of a point by forming triangles to it from known points. Triangulation is a very popular technique used in optical 3D measuring systems. Triangulation today is used for many purposes, including surveying, navigation, metrology, astrometry, binocular vision, model rocketry and gun direction of weapons. This technique has been in use from a very long time. History has recorded the Greek philosopher Thales using similar triangles to estimate the height of the pyramids of ancient Egypt in the 6th century BC.

To determine the position of the mobile robot in its environment, two different coordinates systems (or) frames need to be defined.

- Inertial Coordinate System: This coordinate frame is fixed in the environment or the plane in which the mobile robot traverses. It is denoted by $\{X_I, Y_I, Z_I\}$
- Body Coordinate System: This coordinate frame is fixed to the robot and it moves along with the robot. This frame denoted by $\{X_b, Y_b, Z_b\}$.

For the derivation of the triangulation solution, the Inertial and body frame are assigned as shown in Fig. 4.1,

- Inertial frame is attached with X_I pointing towards local East, Y_I pointing towards local North, Z_I going up.
- Body frame is attached with X_b pointing robots right side, Y_b pointing towards robots forward direction, Z_b going up.

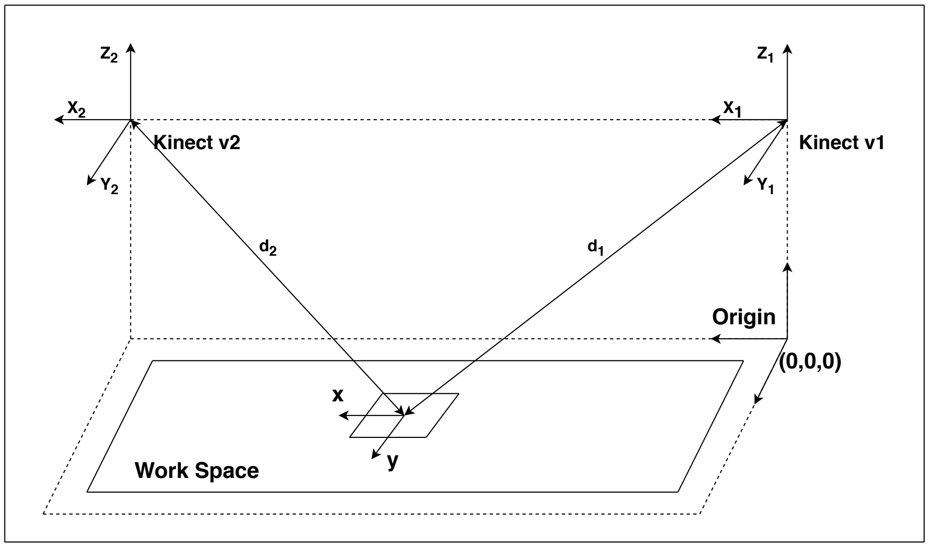


Figure 4.1: Frames for triangulation

The concept of triangulation to estimate the position of the marker can be visualized from Fig. 4.1. Here d_1 and d_2 are the two range measurements from the two distinct sensors. Since the distance between the sensors is constant and is known before hand, the triangle relationship is setup. The measurement d_1 is from Kinect v1 and measurement d_2 is from Kinect v2. These outputs are in the inertial coordinate frame described above since the frame transformations are taken care of by the ar_track_alvar package so the angles are not computed explicitly. d_1 and d_2 are range measurements, which mathematically implies,

$$d_1 = \sqrt{(X_1 - x)^2 + (Y_1 - y)^2 + (Z_1 - z)^2} \tag{4.1}$$

$$d_2 = \sqrt{(X_2 - x)^2 + (Y_2 - y)^2 + (Z_2 - z)^2} \tag{4.2}$$

The terms x, y and z represented in the Eq. 4.1 & Eq. 4.2 is the estimated position of the robot with respect to the origin in the inertial coordinate frame. The

$\{X_1, Y_1, Z_1\}$ and $\{X_2, Y_2, Z_2\}$ are the position of the sensors in the inertial coordinate frame.

4.2 Analytic Method

In the previous section, it is established that triangulation solution will yield the position estimate of the robot when in motion which is required. The problem is setup as 3 unknowns namely x, y, z but there are only two equations, Eq. 4.1 & Eq. 4.2. This is not solvable analytically using only range measurement from two sensors, hence an assumption is made that $z = 0$. This indicates that the solution obtained from the analytic method always considers the markers to be on the same plane as the inertial coordinate system defined. By considering the following assumption, the Eq. 4.1 & Eq. 4.2 simplify to become,

$$d_1 = \sqrt{(X_1 - x)^2 + (Y_1 - y)^2 + Z_1^2} \tag{4.3}$$

$$d_2 = \sqrt{(X_2 - x)^2 + (Y_2 - y)^2 + Z_2^2} \tag{4.4}$$

By squaring the equations on both the RHS and LHS,

$$d_1^2 = (X_1 - x)^2 + (Y_1 - y)^2 + Z_1^2 \tag{4.5}$$

$$d_2^2 = (X_2 - x)^2 + (Y_2 - y)^2 + Z_2^2 \tag{4.6}$$

The Eq. 4.5 & Eq. 4.6 are solved for x and y using symbolic toolbox in MATLAB, which yields the following result that is present the appendix A, the result is used in a simulink file to estimate the x and y position of the marker in the inertial coordinate frame with respect to the origin.

The analytic approach is very accurate but always requires two measurement data to compute the solution. This creates a problem when the marker is not detected by one of the sensors and when one of the sensor is not accessible for a brief period of time. A sequential state estimator can solve these issues.

4.3 Extended Kalman Filter

Kalman filtering, a type of sequential state estimator, is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables that tend to be more accurate than those based on a single measurement alone, by estimating a joint probability distribution over the variables for each time-frame. The Kalman filter has numerous applications in technology. A common application is for guidance, navigation, and control of vehicles, particularly aircraft and spacecraft. The algorithm is a two-step process. In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the next measurement is obtained, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty and a optimal gain called kalman gain is computed and added to the estimate to reduce the error. The algorithm is recursive. It can run in real time, using only the present input measurements and the previously calculated state and its uncertainty matrix.

Model	$\dot{\boldsymbol{x}}(t) = \boldsymbol{F}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) + \boldsymbol{G}(t)\boldsymbol{w}(t), \boldsymbol{w}(t) \sim N(\boldsymbol{0}, Q(t))$ $\tilde{\boldsymbol{y}}(t) = \boldsymbol{H}(t)\boldsymbol{x}(t) + \boldsymbol{v}(t), \boldsymbol{v}(t) \sim N(\boldsymbol{0}, \boldsymbol{R}(t))$
Initialize	$\hat{\boldsymbol{x}}(t_0) = \hat{\boldsymbol{x}}_0$ $\boldsymbol{P}_0 = E\{\tilde{\boldsymbol{x}}(t_0)\tilde{\boldsymbol{x}}^T(t_0)\}$
Gain	$\boldsymbol{K}(t) = \boldsymbol{P}(t)\boldsymbol{H}^T(t)\boldsymbol{R}^{-1}(t)$
Covariance	$\dot{\boldsymbol{P}}(t) = \boldsymbol{F}(t)\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{F}^T(t)$ $-\boldsymbol{P}(t)\boldsymbol{H}^T(t)\boldsymbol{R}^{-1}(t)\boldsymbol{H}(t)\boldsymbol{P}(t) + \boldsymbol{G}(t)\boldsymbol{Q}(t)\boldsymbol{G}^T(t)$
Estimate	$\dot{\hat{\boldsymbol{x}}}(t) = \boldsymbol{F}(t)\hat{\boldsymbol{x}}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t)$ $+\boldsymbol{K}(t)[\tilde{\boldsymbol{y}}(t) - \boldsymbol{H}(t)\hat{\boldsymbol{x}}(t)]$

Table 4.1: Summary of Continuous-Time Linear Kalman Filter

The continuous time model linear kalman filter is summarized in the Table 4.1. First, initial conditions for the state and error covariances are given. Then, the gain $\boldsymbol{K}(t)$ is computed with the initial covariance value. Next, the covariance and state estimate are numerically integrated forward in time using the continuous-time measurement $\tilde{\boldsymbol{y}}(t)$ and known input $\boldsymbol{u}(t)$. The integration of the state estimate and covariance continues until the final measurement time is reached. The detailed derivation for each of the equations is present in [43].

Most physical dynamic systems involve continuous-time models and discrete-time measurements obtained from sensors. Therefore, the system model and measurement model are given by

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{F}(t)\boldsymbol{x}(t) + \boldsymbol{B}(t)\boldsymbol{u}(t) + \boldsymbol{G}(t)\boldsymbol{w}(t) \tag{4.7}$$

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{H}_k\boldsymbol{x}_k + \boldsymbol{v}_k \tag{4.8}$$

In this work, the measurement is non-linear in nature hence a non-linear dynamic model must be incorporated. For these non-linear model problems, a filter called Extended Kalman Filter is the most popular solution. The EKF, though not precisely "optimum", has been successfully applied to many nonlinear systems over the past many years. The fundamental concept of this filter involves the notion that the true state is sufficiently close to the estimated state. Therefore, the error dynamics can be represented fairly accurately by a linearized first-order Taylor series expansion. Consider the first-order expansion of $\boldsymbol{h}(\boldsymbol{x}(t), t)$ about some nominal state $\bar{\boldsymbol{x}}(t)$:

$$\boldsymbol{h}(\boldsymbol{x}(t), t) \approx \boldsymbol{h}(\bar{\boldsymbol{x}}(t), t) + \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right|_{\bar{\boldsymbol{x}}(t)=0} [\boldsymbol{x}(t) - \bar{\boldsymbol{x}}(t)] \tag{4.9}$$

In the EKF, the current estimate is used for the nominal state estimate, so that $\bar{\boldsymbol{x}}(t) = \hat{\boldsymbol{x}}(t)$.

Model	$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t), t) + \boldsymbol{G}(t)\boldsymbol{w}(t), \boldsymbol{w}(t) \sim N(\boldsymbol{0}, Q(t))$ $\tilde{\boldsymbol{y}}_k = \boldsymbol{h}(\boldsymbol{x}_k) + \boldsymbol{v}_k, \boldsymbol{v}_k \sim N(\boldsymbol{0}, \boldsymbol{R}(t))$
Initialize	$\hat{\boldsymbol{x}}(t_0) = \hat{\boldsymbol{x}}_0$ $\boldsymbol{P}_0 = E\{\tilde{\boldsymbol{x}}(t_0)\tilde{\boldsymbol{x}}^T(t_0)\}$
Gain	$\boldsymbol{K}_k = \boldsymbol{P}_k^- \boldsymbol{H}_k^T(\hat{\boldsymbol{x}}_k^-)\left[\boldsymbol{H}_k(\boldsymbol{x}_k^-)\boldsymbol{P}_k^- \boldsymbol{H}_k^T(\hat{\boldsymbol{x}}_k^-) + \boldsymbol{R}_k\right]^{-1}$ $\boldsymbol{H}_k(\hat{\boldsymbol{x}}_k^-) \equiv \left.\frac{\partial \boldsymbol{h}}{\partial \boldsymbol{x}}\right\|_{\hat{\boldsymbol{x}}_k^-}$
Update	$\hat{\boldsymbol{x}}_k^+ = \hat{\boldsymbol{x}}_k^- + \boldsymbol{K}_k\left[\tilde{\boldsymbol{y}}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_k^-)\right]$ $\boldsymbol{P}_k^+ = \left[\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k(\hat{\boldsymbol{x}}_k^-)\right])\boldsymbol{P}_k^-$
Propagation	$\dot{\hat{\boldsymbol{x}}}(t) = f(\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t), t)$ $\dot{\boldsymbol{P}}(t) = \boldsymbol{F}(t)\boldsymbol{P}(t) + \boldsymbol{P}(t)\boldsymbol{F}^T(t) + \boldsymbol{G}(t)Q(t)\boldsymbol{G}^T(t)$ $\boldsymbol{F}(t) \equiv \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right\|_{\hat{\boldsymbol{x}}(t), \boldsymbol{u}(t)}$

Table 4.2: Summary of Continuous-Discrete Extended Kalman Filter

The detailed derivation for each of the equations in the Table 4.2 is present in [43]. There are two system dynamics models which this work focuses on, the constant velocity model and the constant acceleration model.

4.3.1 Constant Velocity

The constant velocity model for the motion of the marker is described as

$$\dot{\boldsymbol{x}} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{G}\boldsymbol{w} \tag{4.10}$$

where,

$$\boldsymbol{F} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{G} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and the states of the model is proposed as

$$\boldsymbol{x} = \begin{bmatrix} x & y & \dot{x} & \dot{y} \end{bmatrix}^T \tag{4.11}$$

The measurement model is a non-linear range measurement model proposed as

$$\boldsymbol{y} = \begin{bmatrix} \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + z_1^2} + v_1 \\ \sqrt{(x_2 - x)^2 + (y_2 - y)^2 + z_2^2} + v_2 \end{bmatrix} \tag{4.12}$$

The Jacobian necessary for the EKF is computed by taking the partial derivative of the measurement equation Eq. 4.12 with respect to each of the states mentioned in Eq. 4.11. This yields the following H matrix

$$\boldsymbol{H} = \begin{bmatrix} \frac{-(x_1-x)}{\sqrt{(x_1-x)^2+(y_1-y)^2+z_1^2)}} & \frac{-(y_1-y)}{\sqrt{(x_1-x)^2+(y_1-y)^2+z_1^2}} & 0 & 0 \\ \frac{-(x_2-x)}{\sqrt{(x_2-x)^2+(y_2-y)^2+z_2^2}} & \frac{-(y_2-y)}{\sqrt{(x_2-x)^2+(y_2-y)^2+z_2^2}} & 0 & 0 \end{bmatrix} \tag{4.13}$$

The states of the dynamic model given in Eq. 4.11 is propagated in time along with the Jacobian in Eq. 4.13 is coded in a MATLAB function block in simulink as explained in the next chapter.

4.3.2 Constant Acceleration

The constant acceleration model for the motion of the marker is described as

$$\dot{\boldsymbol{x}} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{G}\boldsymbol{w} \tag{4.14}$$

where,

$$\boldsymbol{F} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{G} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

and the states of the model is proposed as

$$\boldsymbol{x} = \begin{bmatrix} x & y & \dot{x} & \dot{y} & \ddot{x} & \ddot{y} \end{bmatrix}^T \tag{4.15}$$

The measurement model is a non-linear range measurement model proposed as

$$\boldsymbol{y} = \begin{bmatrix} \sqrt{(x_1 - x)^2 + (y_1 - y)^2 + z_1^2} + v_1 \\ \sqrt{(x_2 - x)^2 + (y_2 - y)^2 + z_2^2} + v_2 \end{bmatrix} \tag{4.16}$$

The Jacobian necessary for the EKF is computed by taking the partial derivative of the measurement equation Eq. 4.16 with respect to each of the states mentioned in Eq. 4.15. This yields the following H matrix

$$\boldsymbol{H} = \begin{bmatrix} \frac{-(x_1-x)}{\sqrt{(x_1-x)^2+(y_1-y)^2+z_1^2)}} & \frac{-(y_1-y)}{\sqrt{(x_1-x)^2+(y_1-y)^2+z_1^2}} & 0 & 0 & 0 & 0 \\ \frac{-(x_2-x)}{\sqrt{(x_2-x)^2+(y_2-y)^2+z_2^2}} & \frac{-(y_2-y)}{\sqrt{(x_2-x)^2+(y_2-y)^2+z_2^2}} & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.17}$$

The states of the dynamic model given in Eq. 4.15 is propagated in time along with the Jacobian in Eq. 4.17 is coded in a MATLAB function block in simulink as explained in the next chapter.

CHAPTER 5

Experimental Setup

5.1 Components

For this work there is a set of hardware in combination with a specific configuration of software to support and run the thesis objectives. The overall hardware setup can be visualized from Fig. 5.1.

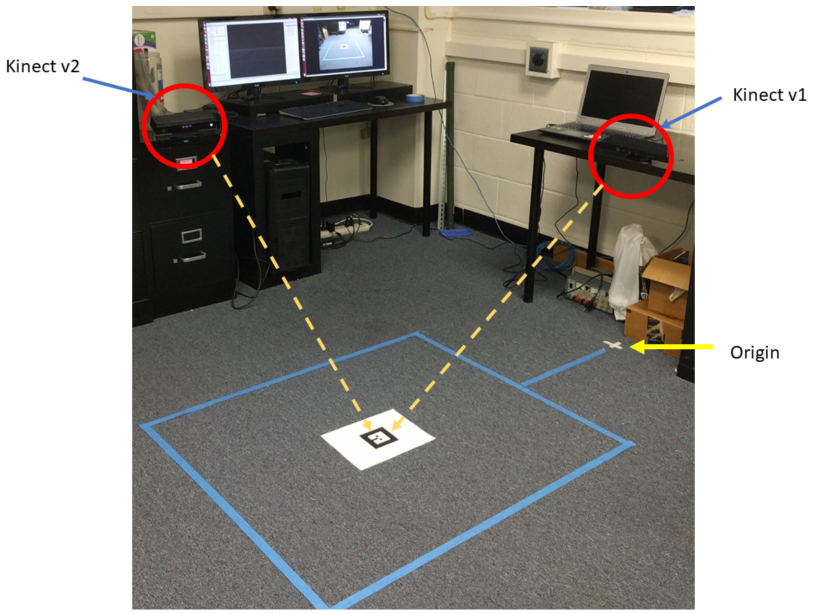


Figure 5.1: Hardware setup

The setup seen in Fig. 5.1 is one of the possible configurations, where the two sensors are at 90° to each other. The white cross on the right side is the origin in the inertial coordinate frame. The blue box represents the active work-space of the two sensors in this current configuration. This work-space will change with a different configuration of the sensors.

5.1.1 Workstation

Workstation is the primary computer which is used in this research work to compile and execute the core algorithms pertaining to the position estimation of the robot. This runs an Intel i5 processor coupled with a 4 Gb Nvidia graphics card along with 16 Gb of RAM and a 1 Tb hard-disk. The operating system used here is a Linux distribution, Ubuntu 16.04 LTS.

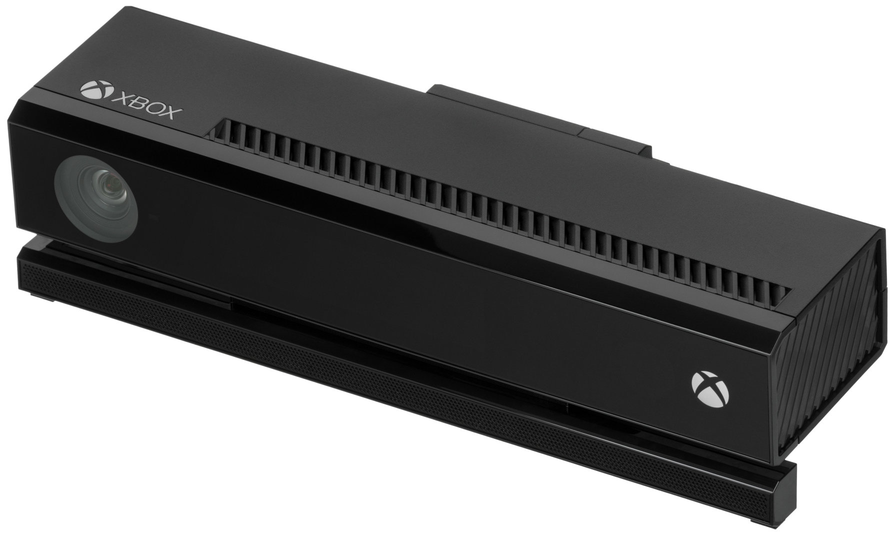


Figure 5.2: Microsoft Kinect One or Kinect v2
Source : https://en.wikipedia.org/wiki/Kinect

The time-of-flight principle based sensor used for this work is the Microsoft Kinect One, seen in Fig. 5.2. This is connected to the workstation via a USB 3.0 port. Kinect v2 is always required to be connected to a USB 3.0 controller since

USB 2 is not supported by the libfreenect2 driver. The USB connector required to connect the Kinect v2 and computer is a special device manufactured by Microsoft, seen in Fig. 5.3. The sensor is powered by a standard AC power outlet. The USB is dedicated only for the transfer of data.

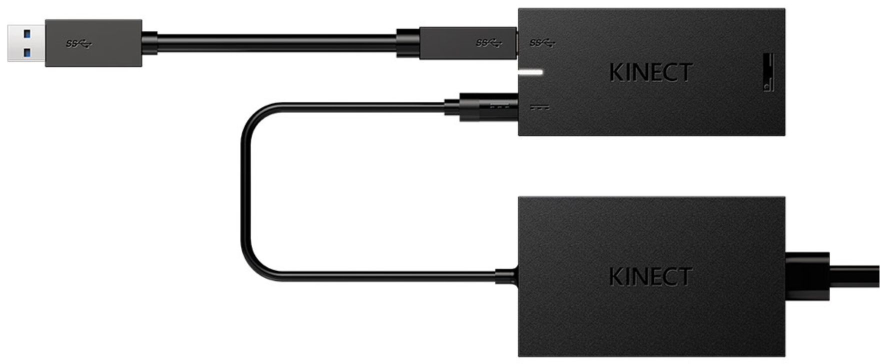


Figure 5.3: Microsoft Kinect One or Kinect v2
Source :
https://www.xbox.com/en-US/xbox-one/accessories/Kinect/Kinect-adapter

5.1.2 Laptop

Laptop is used in this research for the mobility that it provides to test the algorithm in various scenarios and to prevent the overburden on a single computing device. This modular approach is used to demonstrate that any number of sensors can be used to obtain the range measurement data in conjunction to provide more data for a more precise position estimate. Laptop acts as a node in the ROS network, which is explained in detail in the next section of this chapter. This runs an Intel i7 processor coupled with a 4 Gb Nvidia graphics card along with 12 Gb of RAM and a 1 Tb hard-disk. The operating system used here is a Linux distribution, Ubuntu

16.04 LTS.

The structured light principle based sensor used for this work is the Microsoft Kinect 360, seen in Fig. 5.4. This is connected to the laptop via a USB port. The sensor is powered by a standard AC power outlet. The USB is dedicated only for the transfer of data.

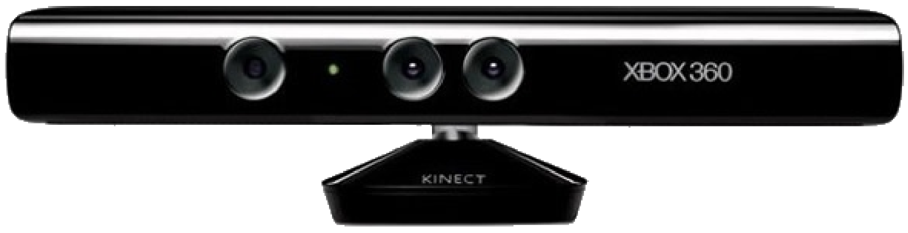


Figure 5.4: Microsoft Kinect 360 or Kinect v1
Source : https://msdn.microsoft.com/en-us/library/hh438998.aspx

5.2 Software and Network Setup

There are certain software and drivers which must be installed on each of the computing device interfacing with the sensors. All the computing devices must be running the Ubuntu 16.04 LTS operating system. Next, the robot operating system ROS must be installed. Here all systems are equipped with Kinetic Kame distribution of ROS. Using other versions of Linux and ROS might cause compatibility issues with the drivers installed for the sensors.

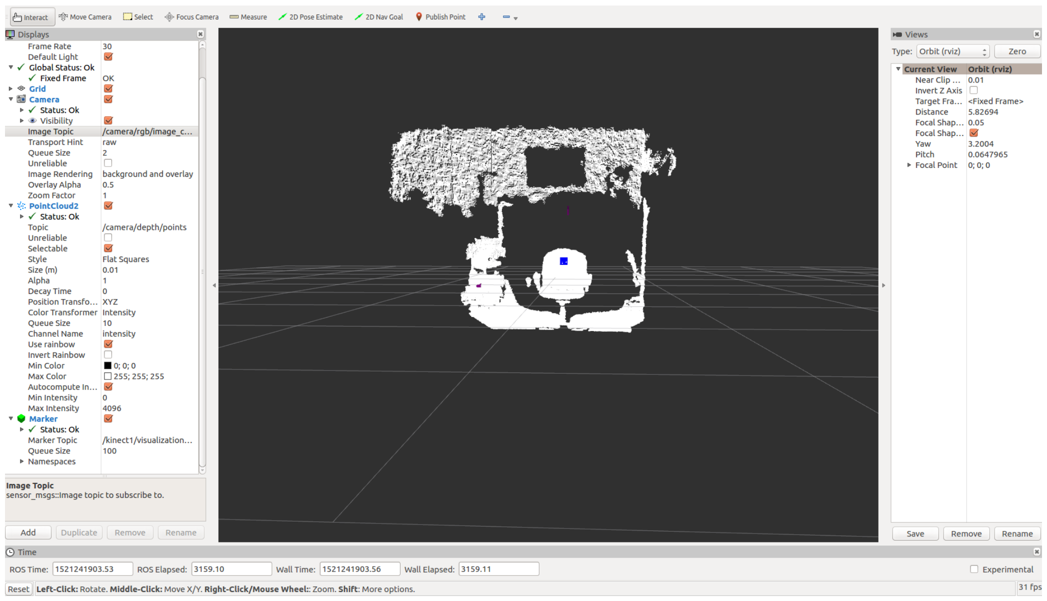
Once the ROS is installed, drivers must be installed to interface with the sensors. For the Kinect v1, openni_launch and openni_camera packages for ROS must be installed. This is the primary driver to interface with the sensor. Rviz, which is a visualization software must be installed as well. Next AR_Track_Alvar must be installed. This is the key package which is responsible for the marker detection and pose measurement frame transformations.

For Kinect v2, libfreenect2 package must be installed along with OpenCL, CUDA for Nvidia graphic cards, VAAPI and OpenNI2. These are the libraries upon which the Kinect v2 driver is built. Next, iai_kinect2 must be installed, this is the Kinect v2 driver. MATLAB and Simulink 2017a is installed. All the above mentioned drivers and supporting software must be installed correctly in each of the computing systems.

Once all the relevant drivers and sofware are installed, kinects should be tested. In different terminals, the following must be running,

- roscore
- roslaunch openni_launch openni.launch
- roslaunch kinect2_bridge kinect2_bridge.launch
- rosrun rviz rviz

Within rviz:
Add a pointcloud2 display with the topic: /camera/depth/points or
/camera/depth_registered/points
Set the fixed frame to /camera_depth_optical_frame



```
Displays
    Frame Rate              30
    Default Light           ✓
▼ ✓ Global Status: Ok
    ✓ Fixed Frame           OK
▶ Grid                      ✓
▼ Camera                    ✓
  ▶ ✓ Status: Ok
  ▶ Visibility              ✓
    Image Topic             /camera/rgb/image_c...
    Transport Hint          raw
    Queue Size              2
    Unreliable              ☐
    Image Rendering         background and overlay
    Overlay Alpha           0.5
    Zoom Factor             1
▼ PointCloud2               ✓
  ▶ ✓ Status: Ok
    Topic                   /camera/depth/points
    Unreliable              ☐
    Selectable              ✓
    Style                   Flat Squares
    Size (m)                0.01
    Alpha                   1
    Decay Time              0
    Position Transfo...     XYZ
    Color Transformer       Intensity
    Queue Size              10
    Channel Name            intensity
    Use rainbow             ✓
    Invert Rainbow          ☐
    Min Color               ■ 0; 0; 0
    Max Color               □ 255; 255; 255
    Autocompute In...       ✓
    Min Intensity           0
    Max Intensity           4096
▼ Marker                    ✓
  ▶ ✓ Status: Ok
    Marker Topic            /kinect1/visualization...
    Queue Size              100
  ▶ Namespaces

Image Topic
sensor_msgs::Image topic to subscribe to.

Views
Type: Orbit (rviz)
▼ Current View              Orbit (rviz)
    Near Clip ...           0.01
    Invert Z Axis           ☐
    Target Fra...           <Fixed Frame>
    Distance                5.82694
    Focal Shap...           0.05
    Focal Shap...           ✓
    Yaw                     3.2004
    Pitch                   0.0647965
  ▶ Focal Point             0; 0; 0

Time
ROS Time: 1521241903.53   ROS Elapsed: 3159.10   Wall Time: 1521241903.56   Wall Elapsed: 3159.11   ☐ Experimental
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.   31 fps
```

Figure 5.5: Kinect Test

From Fig. 5.5, it is confirmed that the Kinect is interfacing with ROS correctly. Next the tracker package is launched. In separate terminals the following commands are launched which creates two distinct ROS nodes.

- For Kinect v1

  roslaunch ar_track_alvar pr2_indiv.launch marker_size:=9 max_new_marker _error:=0.1 max_track_error:=0.3 cam_info_topic:=/camera/rgb/camera_info cam_image_topic:=/camera/depth_registered/points output_frame:= camera_rgb_optical_frame

- For Kinect v2

  roslaunch ar_track_alvar pr2_indiv.launch marker_size:=9 max_new_marker _error:=0.1 max_track_error:=0.3 cam_info_topic:=/kinect2/qhd/camera_info cam_image_topic:=/kinect2/qhd/points output_frame:=/kinect2_rgb _optical_frame

The inputs given in the commands above define the ar tag width, calibration of the camera and the transform frames.

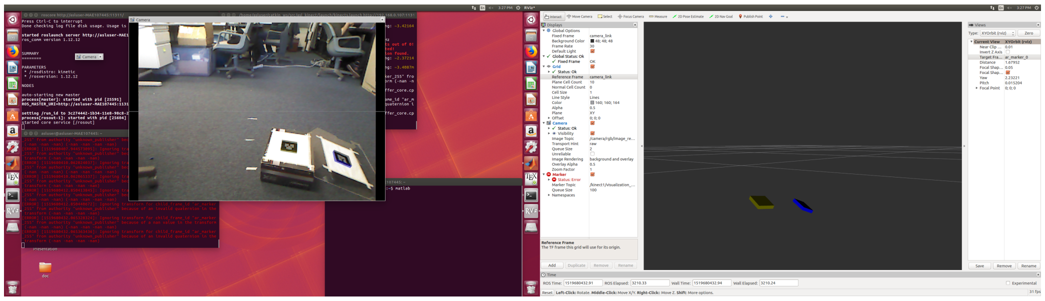


Figure 5.6: Kinect Test with tracker package activated

Fig. 5.6, the tags can be distinctly visualized. Since there are multiple files which must be launched simultaneously in-order to achieve all these functionality, two shell scripts, one for each of the kinects, were created which encapsulates all the launch files in one single executable file. They are,

- launch_k1.sh
- launch_k2.sh

The ROS network is setup at this stage. The roscore command is initiated on the workstation, making it the ROS master node. The IP addresses of the master node is exported along with the port number. On the latop, which must be on the same Wifi network, exports its IP address as well. This creates the communication between the workstation, the ROS master node and laptop, a ROS node. This grands access to all the topics being published in the ROS network to the workstation, which means that it can access the range measurement from Kinect v1 which is connected to the laptop. One of the hassles is that this exporting of IP address must be done for each new terminal opened to continue communications. To avoid this issue, a generic code[44] for exporting IP address is embedded in the two shell scripts. This automatically creates the network and sets up the communication.

This data from ROS must be accessed in MATLAB and Simulink. MATLAB and Simulink have in-built support for ROS messages via the ROS Toolbox. But the messages published from the AR_Track_Alvar is not supported in MATLAB, hence the message is converted to a custom message which is supported in MATLAB. This is achieved via a python script which runs in the background, converting all the messages to a suitable format. This republisher.py file is also included in the launch_k2.sh shell script which is executed on the workstation.

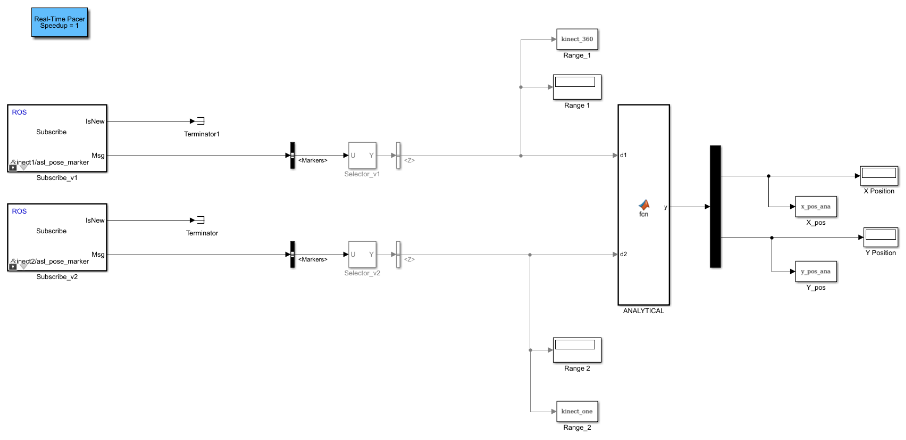


Figure 5.7: Simulink Block diagram for the analytic approach

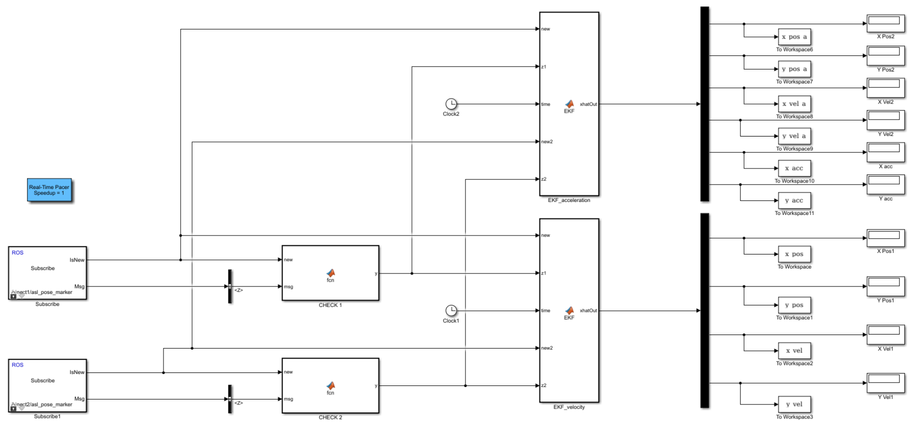


Figure 5.8: Simulink Block diagram for the EKF approach

Fig. 5.7 & Fig. 5.8, shows the simulink block diagram, which is developed to compute the position estimate and motion tracking of the marker which the focus of this research. The ROS Subscriber blocks are used to receive the data, then using the selector block, only the depth measurement is extracted. This depth measurement is checked to make sure that the data it contains is reliable i.e. the data is not NaN, which can affect the robustness of the estimation code. The reliable data is sent to the MATLAB function block. The analytical MATLAB function block contains the analytical solution equations mentioned in the previous chapter along with the sensor positions which must be defined by the user. The analytical MATLAB function block contains the analytical solution equations mentioned in chapter 4 along with the sensor positions which must be defined by the user. The EKF MATLAB function block contains the extented kalman filtering algorithm equations mentioned in chapter 4 along with the sensor positions which must be defined by the user. The output from these MATLAB function blocks are segregated using a demux and the x and y coordinated of the markers are displayed in real-time and also stored in the MATLAB work-space for analysis of the output. For multiple object tracking, one more layer of check is created as shown in Fig.5.9 and the rest of the algorithm is consistent with the EKF MATLAB function block. Here the outer layer check, separates the range measurement data with respect to the unique marker ID. Then the following streams of data are fed to the marker tracking subsystems. Each of the subsystem contains the EKF MATLAB function block, which estimates the position of that marker ID.

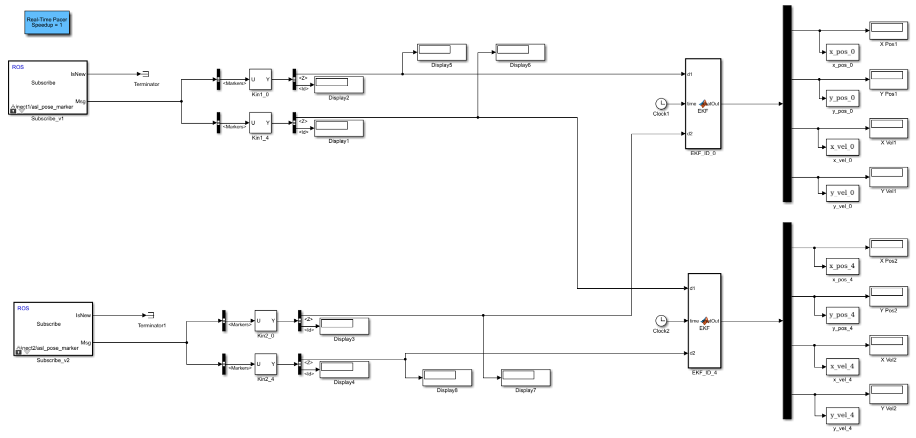


Figure 5.9: Simulink Block diagram for Multiple marker tracking

There are two models proposed for the Extended Kalman Filter and the flow of data from the sensor to the EKF algorithm to the output is clearly seen in Fig. 5.10, for the velocity model and Fig.5.11, for the acceleration model.

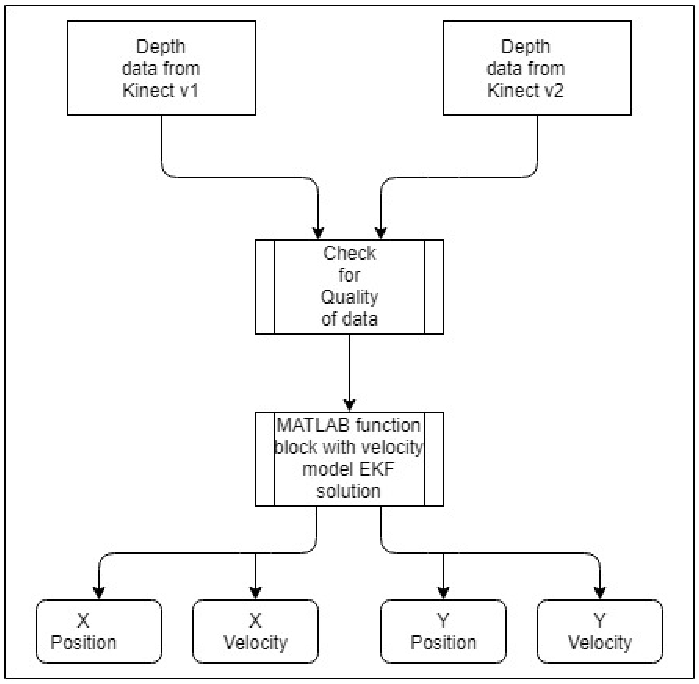

Depth data from Kinect v1
Depth data from Kinect v2
Check for Quality of data
MATLAB function block with velocity model EKF solution
X Position
X Velocity
Y Position
Y Velocity

Figure 5.10: Data flow for Velocity model in Extended Kalman Filter method

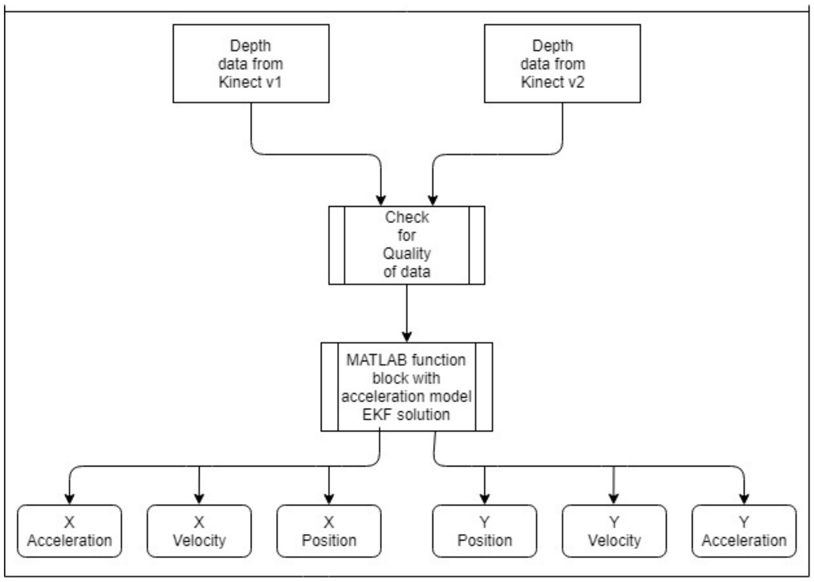

Depth data from Kinect v1
Depth data from Kinect v2
Check for Quality of data
MATLAB function block with acceleration model EKF solution
X Acceleration
X Velocity
X Position
Y Position
Y Velocity
Y Acceleration

Figure 5.11: Data flow for Acceleration model in Extended Kalman Filter method

5.3 Calibration

In most sensors, calibration is key to obtain accurate data. Since the Kinects are a cluster of sensors, they must be calibrated. The openni_camera driver provides default camera models out-of-the-box with reasonably accurate focal lengths (relating 3D points to 2D image coordinates). They do not model lens distortion, but fortunately the Kinect uses low-distortion lenses, so even the edges of the image are not displaced by more than a few pixels. The openni_launch file executed, then the following command is executed in a terminal,

```
rosrun camera_calibration cameracalibrator.py image:=/camera
/rgb/image_raw camera:=/camera/rgb –size 5x4 –square 0.0245
```

The above command calibrates the RBG camera and the IR image of Kinect v1 using the checkerboard. This data is saved in camera_info, which is retrieved when the ar_track_alvar node is operational. The similar procedure must be followed for calibration of Kinect v2 but using the kinect2_calibration in the kinect2_Bridge driver. Both use the same checkerboard as shown in Fig. 5.12.

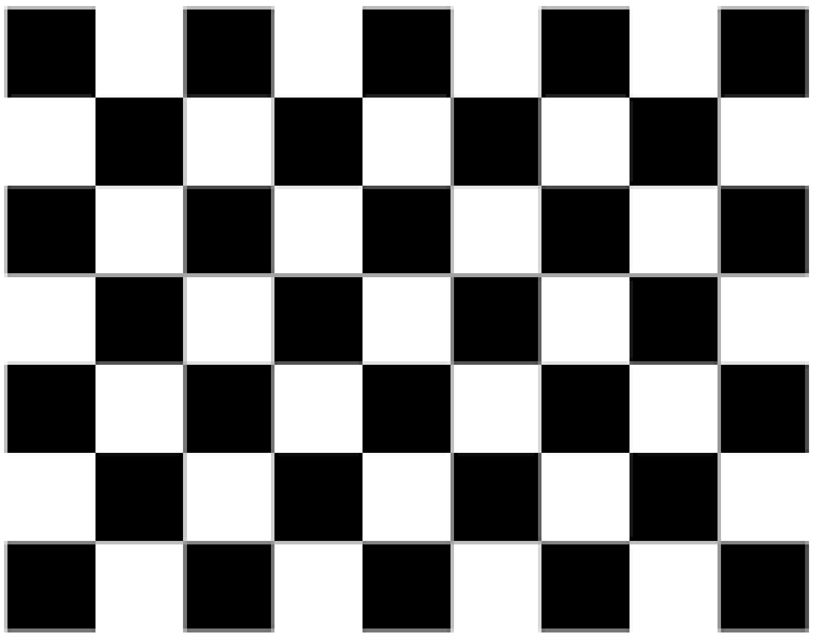


Figure 5.12: Checkerboard for sensor calibration
Source :
http://library.isr.ist.utl.pt/docs/roswiki/camera_calibration(2f)Tutorials(2f)
StereoCalibration.html

This calibration process is not entirely necessary since OpenNI is fairly accurate out-of-the-box for most applications. This calibration is performed only for very high accuracy of the sensors.

5.4 Marker

The markers are created by running the built-in artrack package using the following command

```
rosrun ar_track_alvar createMarker
```

Then by following the prompt, the new marker can be generated as per the user specifications. The user can set the marker id number and other options if needed.

The 9x9 cm tags track nicely, any smaller dimension marker only work with close proximity to the Kinect (i.e. less than 1 meter). The pr2_indiv.launch package can track multiple tags at once. The pr2_bundle.launch package is used to define a rigid body. The markers are all on the same page with a master tag defined and the others have a fixed (x,y) position from the master tag. This allows for orientation to be tracked as well.

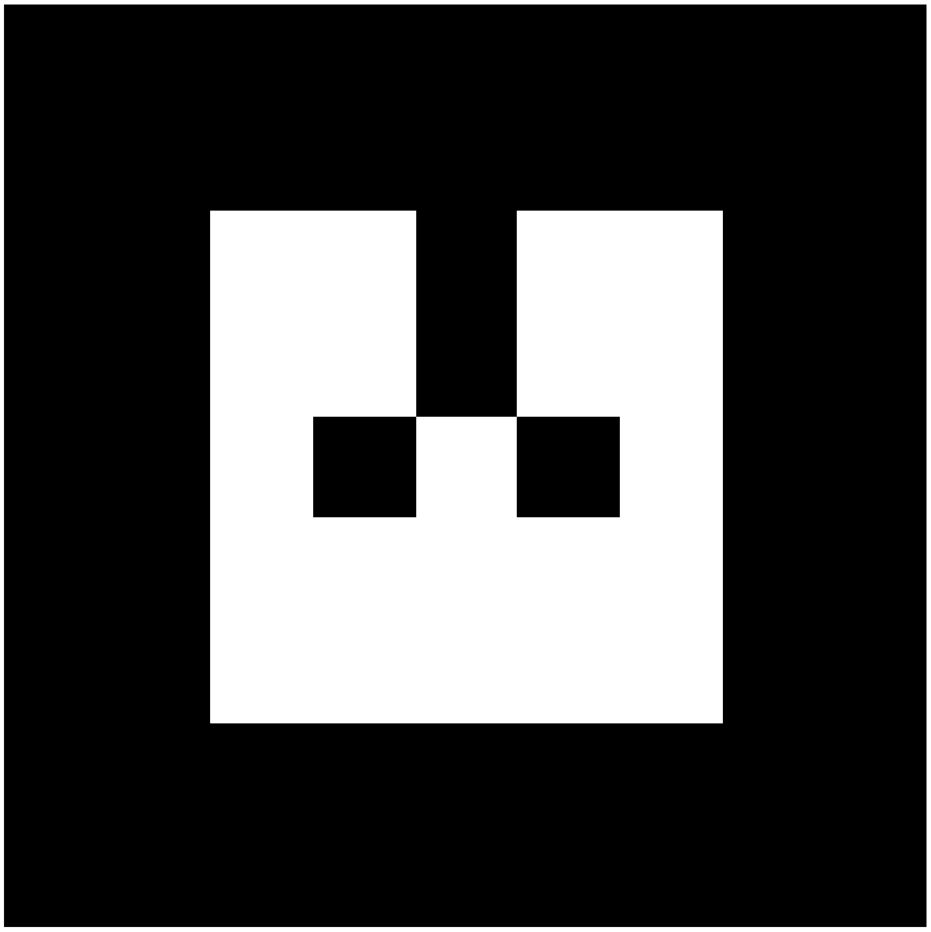


Figure 5.13: Augmented Reality package marker ID 0

CHAPTER 6

Experimental Results

6.1 Sensor Characterization

The first step is characterize the sensors being used for this research. There are two sensors used: Kinect v1 and Kinect v2. Both sensors operate on two distinct technologies hence they are characterized separately.

6.1.1 Horizontal field of view

The purpose of this experiment is to determine the horizontal field of view of each of the sensors and to obtain the maximum range of the sensor.

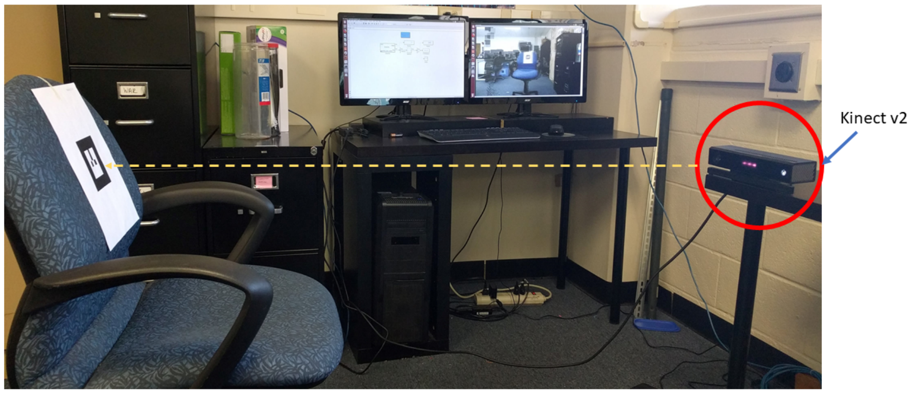
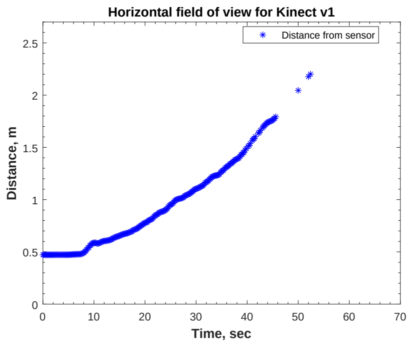
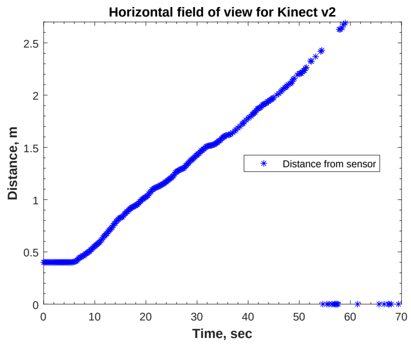


Figure 6.1: Horizontal field of view experimental setup

The two sensors are setup as shown in Fig. 6.1. The sensors are on a parallel plane to the base. This experiment was conducted by fixing the position of the sensor and mounting the marker on the back rest of the chair. The chair was moved from close to the sensor, slowly moving away from the sensor in a straight line.



(a) Kinect v1



(b) Kinect v2

Figure 6.2: Plot of Horizontal field of view

From Fig. 6.2, It can be seen that both the sensors have a range of 2.5m. But the accuracy starts reducing after about 1.7m for both the sensors. The disturbance see after 40sec is because after 1.7m the kinects can no longer effectively detect the marker. The maximum detectable range of the marker was found to be 2.5m and the minimum detectable range is

6.1.2 Work-space

The purpose of this experiment is to determine the effective work space of each of the sensors.

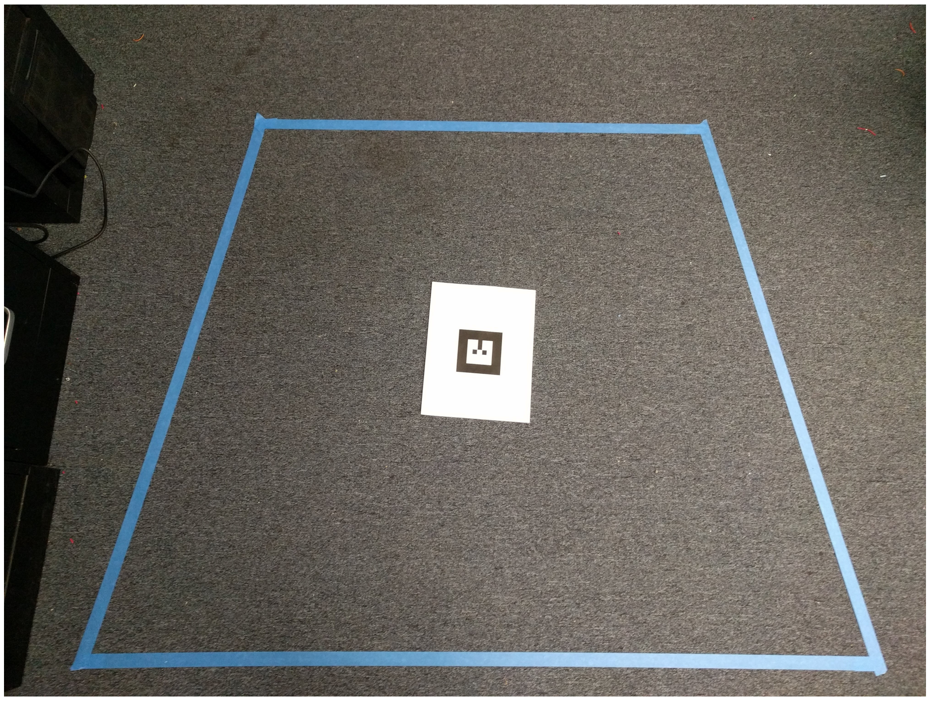


Figure 6.3: Effective work space of sensor

For this experiment, the sensor was places as height of 30" from the ground. The marker was physically moved along the floor until the kinects could no longer detect the marker ID. Fig. 6.3 was obtained when the boundary was earmarked.
The work space was found to be a trapezoid. The length of the short side was found to be 38" and the long side 70". The height was found to be 40". This result helps in determining the number of sensors and nodes required to cover a certain space for the position estimation of the marker.

6.1.3 Sensor Tilt

The purpose of this experiment is to determine the angled field of view of each of the sensors since both the sensors are equipped with a adjustable base.

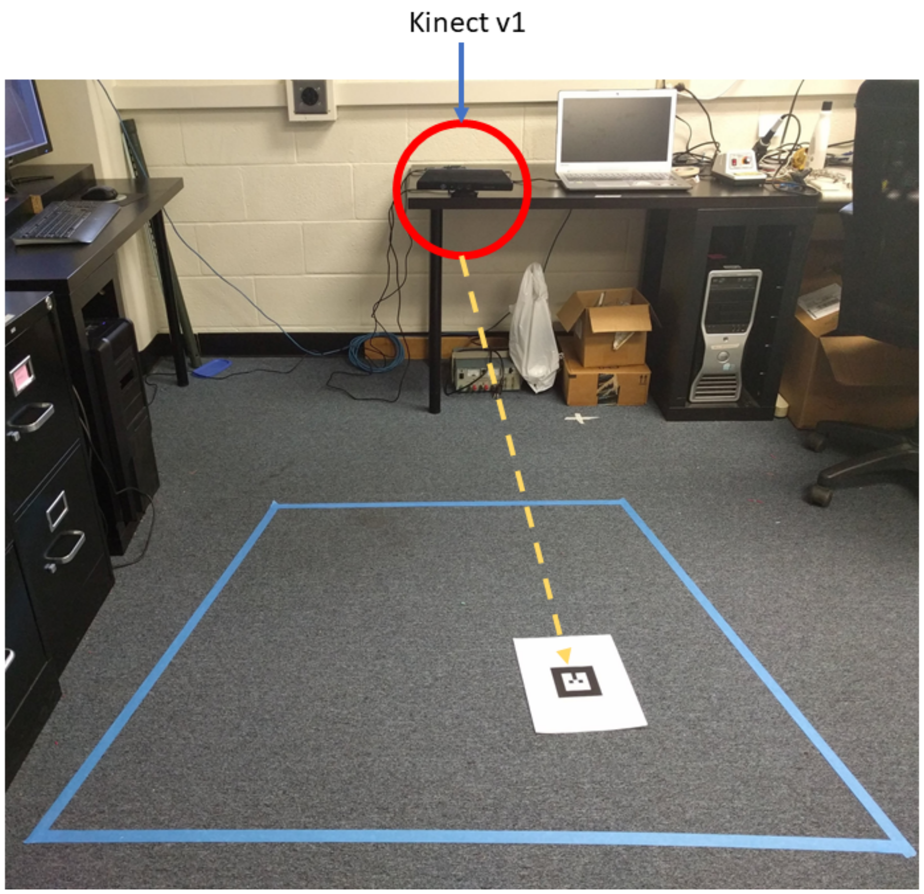


Figure 6.4: Setup for angled field of view sensor

The base of the Kinect v1 was set at specific angles and the maximum and minimum range measurement for the marker was found. The marker was place flat on the floor as seen in the setup Fig. 6.4.

These were the maximum and minimum distances from the sensor at various fixed angles,

At 18°,

- Minimum Range = $0.96m$
- Maximum Range = $1.65m$

At 36°,

- Minimum Range = 1.03m
- Maximum Range = 2.06m

At 54°,

- Minimum Range = 1.17m
- Maximum Range = 2.10m

At 72°,

- Minimum Range = 1.33m
- Maximum Range = 2.17m

At 90°,

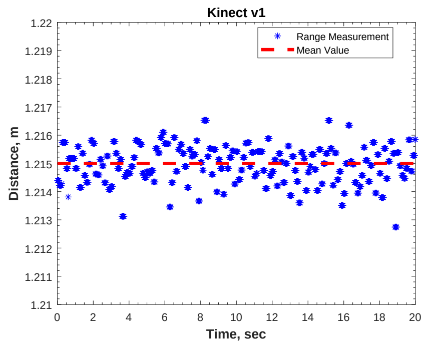
- Minimum Range = 1.6m
- Maximum Range = 2.25m

This result helps to choose the angle of the sensor based on the required range for any specific application.

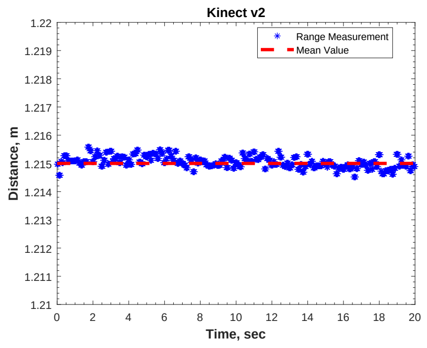
6.1.4 Accuracy

The purpose of this experiment is to determine the accuracy of the range measurement data obtained from each of the sensors.

The setup is same as the previous experiment. Here the marker was kept on the floor and the x and y coordinates of the marker with respect the origin was found using a measuring tape. The range data was collected in MATLAB and analyzed.



(a) Kinect v1



(b) Kinect v2

Figure 6.5: Accuracy of the range measurement

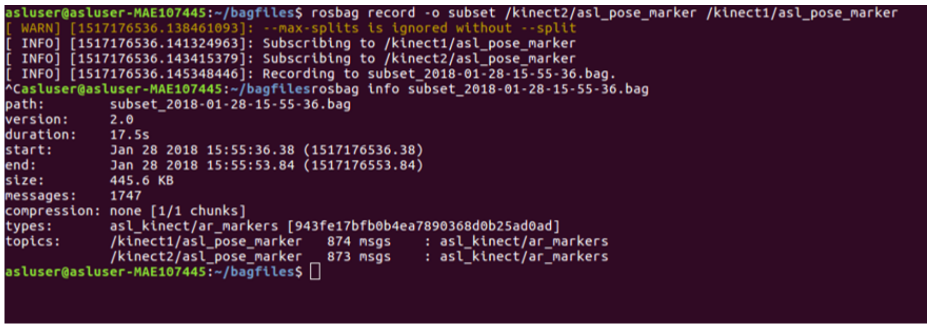
As it can be seen from the Fig. 6.5a, range data varies at from $1.212m$ to $1.217m$ and from Fig. 6.5b, range data varies from $1.214m$ to $1.216m$ which is a very small error. The correct mean value was $1.215m$ and the plot also verifies that the mean range measurement from the sensor was $1.2149m$. The data points from Kinect v2 are observed to be more repeatable. The mean, variance and standard deviation of the depth measurements can be seen in Table. 6.1.

Sensor	Kinect v1	Kinect v2
Mean, m	1.2149	1.2150
Variance, m^2	$4.3545E-07$	$4.5833E-08$
Standard Deviation, m	$6.5988E-04$	$2.1409E-04$

Table 6.1: Statistical Data for the two range measuring devices

6.2 Frequency of messages

The sensors output at a certain frequency. It is important to determine this frequency to properly handle the messages which are published and subscribed by the ROS node. Synchronizing the publisher and subscriber is crucial for the robustness of the position estimation algorithm. In order to find the frequency, the rosbag feature of ROS was utilized. Rosbag stores the data from the topic specified i.e. it records all the messages for a certain amount of time. This data can be played back to analyze the data. After recording the data for a time period of $20secs$ and playing back the data, the frequency of the messages received by the publisher was found to be $16Hz$.



```
asluser@asluser-MAE107445:~/bagfiles$ rosbag record -o subset /kinect2/asl_pose_marker /kinect1/asl_pose_marker
[ WARN] [1517176536.138461093]: --max-splits is ignored without --split
[ INFO] [1517176536.141324963]: Subscribing to /kinect1/asl_pose_marker
[ INFO] [1517176536.143415379]: Subscribing to /kinect2/asl_pose_marker
[ INFO] [1517176536.145348446]: Recording to subset_2018-01-28-15-55-36.bag.
^Casluser@asluser-MAE107445:~/bagfilesrosbag info subset_2018-01-28-15-55-36.bag
path:        subset_2018-01-28-15-55-36.bag
version:     2.0
duration:    17.5s
start:       Jan 28 2018 15:55:36.38 (1517176536.38)
end:         Jan 28 2018 15:55:53.84 (1517176553.84)
size:        445.6 KB
messages:    1747
compression: none [1/1 chunks]
types:       asl_kinect/ar_markers [943fe17bfb0b4ea7890368d0b25ad0ad]
topics:      /kinect1/asl_pose_marker   874 msgs    : asl_kinect/ar_markers
             /kinect2/asl_pose_marker   873 msgs    : asl_kinect/ar_markers
asluser@asluser-MAE107445:~/bagfiles$ 
```

Figure 6.6: Use of rosbag feature of ROS

6.3 Results from Analytic Method

Once the sensors are characterized precisely, the position tracking problem is tackled. The analytic solution obtained in the previous chapter is incorporated in a MATLAB function block. The flow of data is described in Fig. 6.7. The time for simulation is set to inf and a real time pacer block is used to synchronize the simulation time and the system clock for a precise and close to real-time implementation.

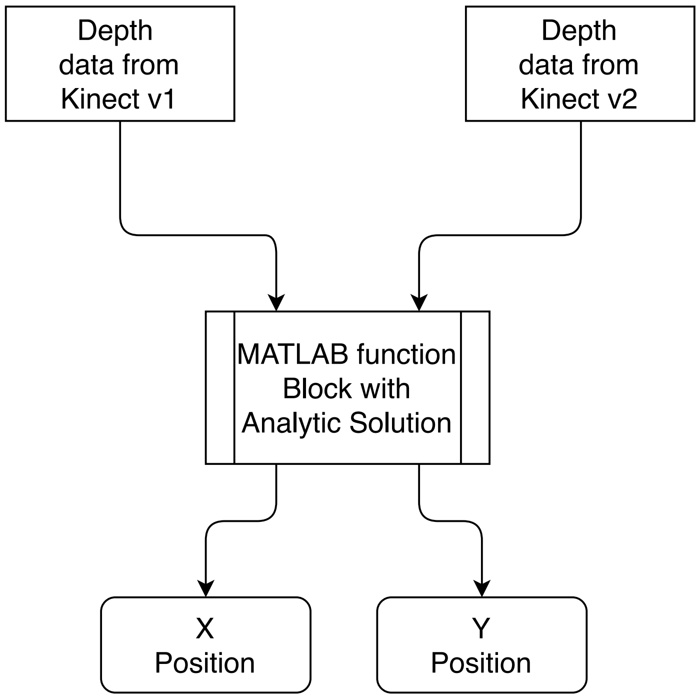

Depth data from Kinect v1
Depth data from Kinect v2
MATLAB function Block with Analytic Solution
X Position
Y Position

Figure 6.7: Data flow for analytic solution method

1. *Position estimation when the robot is stationary.*

The marker is placed inside the work-space and the tracking algorithm is executed. The analytic solution is computed in real time and outputs the position coordinates. The following plot was observed as seen in Fig. 6.8

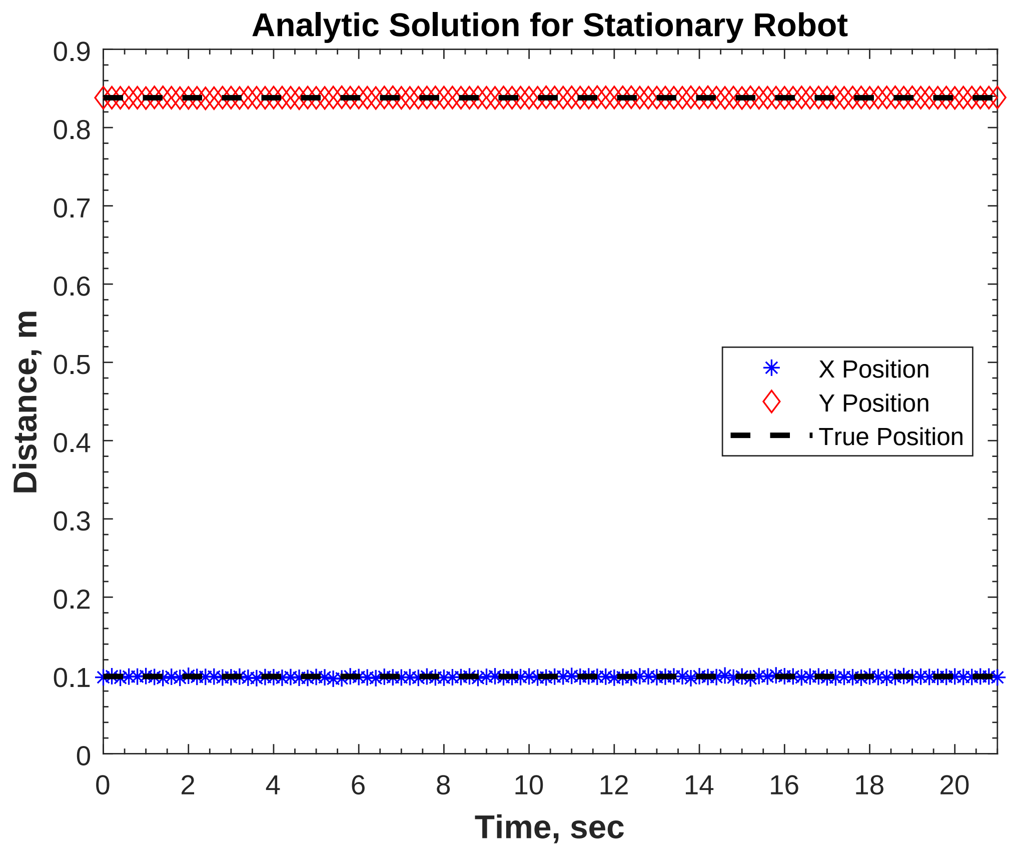


Figure 6.8: Analytical Method Solution for static case

From the Fig. 6.8, it can be seen that the x position is computed to be $0.03m$ and y position is $1.15m$. This was verified with a tape measure. Hence it can be concluded that the analytic solution provides accurate position estimate for stationary conditions.

2. *Position estimation when the robot is in motion.*

The marker is moved within the work-space and the tracking algorithm was executed. The marker was moved in a certain trajectory. The analytic solution is computed in real time and outputs the position coordinates. The following plot was observed as seen in Fig. 6.9

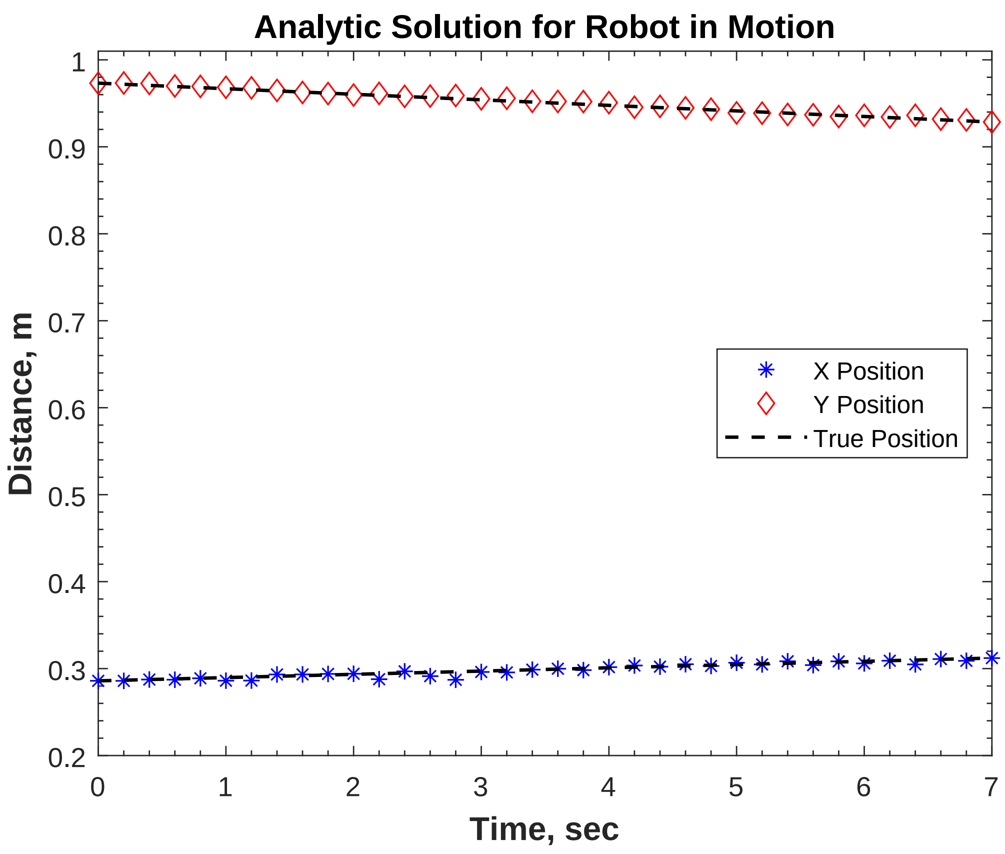


Figure 6.9: Analytic solution for motion of robot

From the Fig. 6.9, it can be seen that the positions are tracked really well. The marker was moved in the y direction from $0.9732m$ to $0.9285m$. This was verified with a tape measure. Hence it can be concluded that the analytic solution provides accurate position estimate for dynamic conditions.

6.4 Results from the Extended Kalman Filter Approach

The Extended Kalman Filter is implemented to obtain the static and dynamic position estimate for the robot. The EKF is coded in MATLAB and simulink using the depth data obtained from the two sensors. The flow of the data and computation is well understood from flowchart seen in Fig. 6.10. The check block evaluates whether the data received is valid before using it for filter computation.

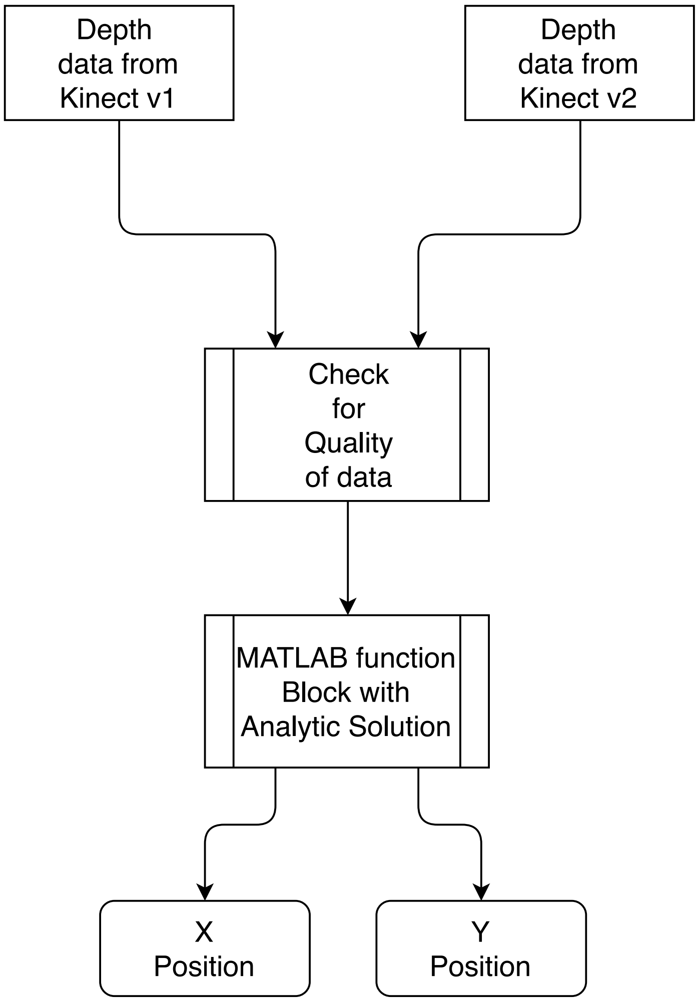

Depth data from Kinect v1
Depth data from Kinect v2
Check for Quality of data
MATLAB function Block with Analytic Solution
X Position
Y Position

Figure 6.10: Data flow for Extended Kalman Filter method

1. *Simulation*

The first step in evaluation of the filter algorithms is to verify the robustness of the filter using synthetic measurement data. Here the static robot depth measurement data is used to run the filter. The following plots were obtained as seen in Fig. 6.11

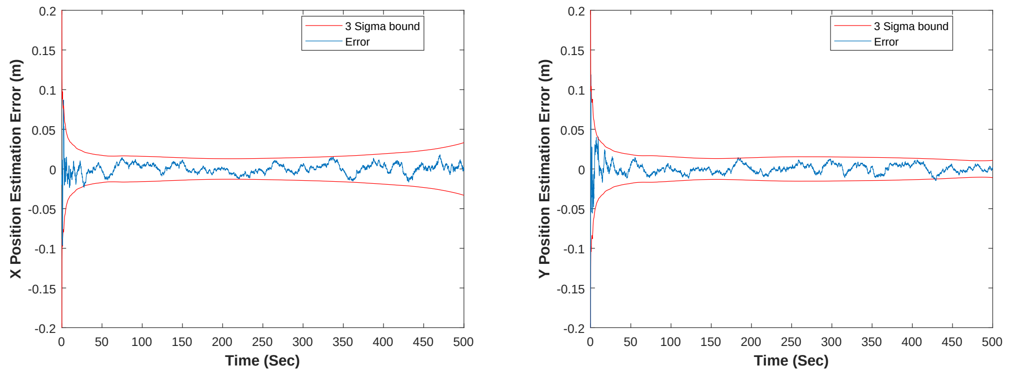


Figure 6.11: x and y position estimation using EKF

From the Fig. 6.11, it is observed that the X position and Y position errors are within their 3-sigma bounds and the error magnitude is less than $0.03m$. This is desirable for a filter.

2. *The experiment was conducted to estimate the position of the robot when stationary and to compare between Analytic Method and EKF Method.*

The marker tag was placed in the work space environment. The two sensors were placed at known locations. The EKF algorithm and the analytic solution algorithm were executed. The static position estimates were obtained and plotted hence, as seen in Fig. 6.12 & Fig. 6.13.

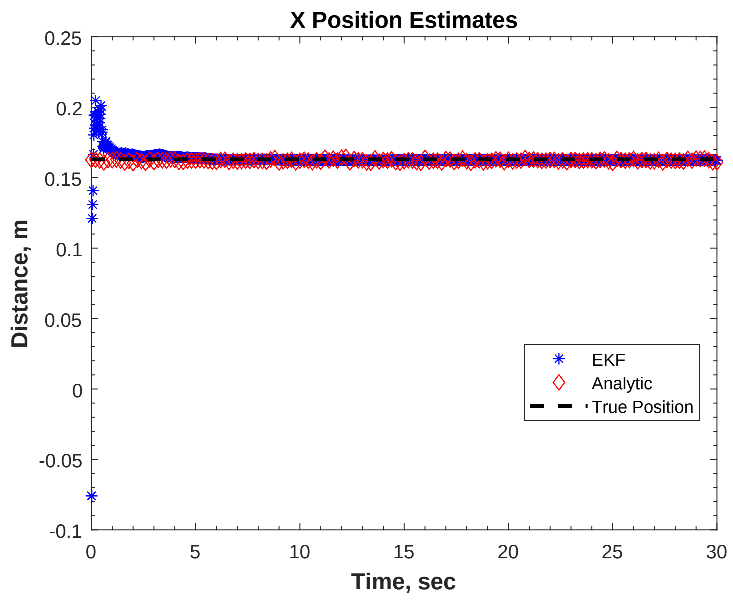


Figure 6.12: X position Comparison between Analytic Method and EKF Method

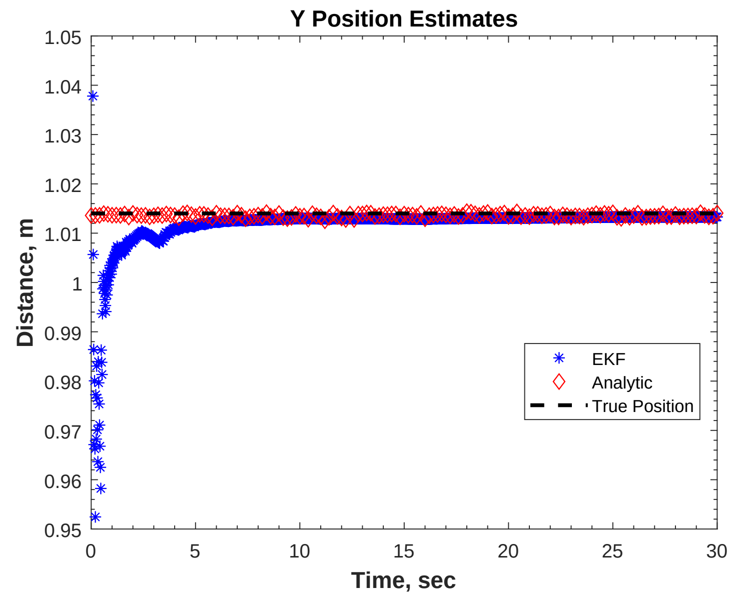
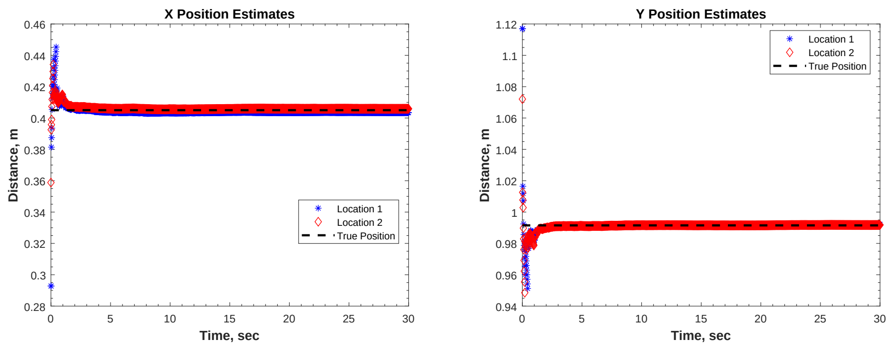


Figure 6.13: Y position Comparison between Analytic Method and EKF Method

From the plots seen in Fig. 6.12 & Fig. 6.13, it is clear that the filter does a commendable job in estimating the position accurately in comparison with the analytic solution. It is also observed that the filter corrects the initial guess error within $2secs$, which is highly desirable. This estimate from the filter and the analytic solution was further compared to the truth, which was found with the help of a tape measure. True x position was $0.1625m$ and true y position was $1.015m$. The results were found to be really good with a maximum error of $11mm$.

3. *The experiment was conducted to estimate the position of the robot when stationary and to compare the results when the sensor is place at a different location.*

The marker tag was placed in the work space environment. The two sensors were placed at known locations. The EKF algorithm was executed. The data was collected and the same experiment was repeated by changing the sensor position of Kinect v2 to a new location and the sensor location data was updated in the algorithm and executed. The following plots were observed as seen in Fig. 6.14



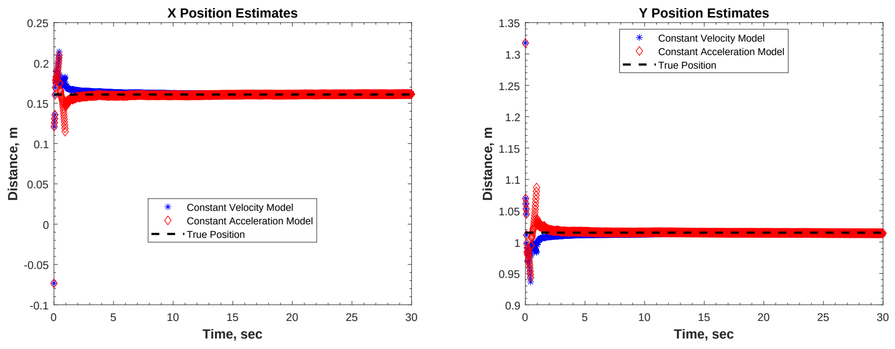
(a) X position estimate (b) Y position estimate

Figure 6.14: EKF position estimates for static robot at two sensor locations

From the Fig. 6.14, it is clearly evident that the filter converges to the correct position both in x and y directions, by the placing the sensor in two distinct locations. The second sensor for the first location was placed at $y = 1.608m$ and the second location was $y = 0.6957m$. Both the locations yield the same position estimate of a static location which is desired i.e. $x = 0.405m$ and $y = 0.9925m$.

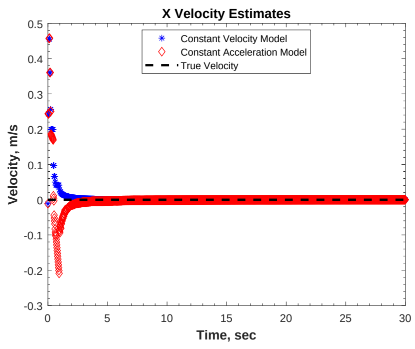
4. *The experiment was conducted to estimate the position of the robot when stationary and to compare the velocity model and the acceleration model.*

The marker tag was placed in the work space environment. The two sensors were placed at known locations. The EKF algorithm with both the velocity model and the acceleration model were executed. The static position estimates and velocity estimates were obtained and plotted hence, as seen in Fig. 6.15 and Fig. 6.16.

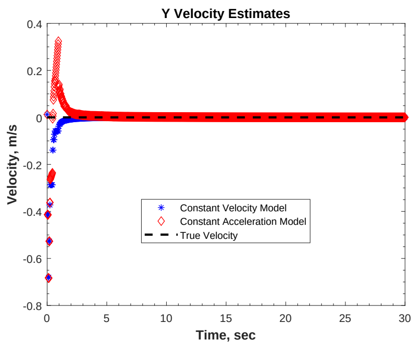


(a) X position estimate (b) Y position estimate

Figure 6.15: EKF position estimates for static robot



(a) X velocity estimate



(b) Y velocity estimate

Figure 6.16: EKF velocity estimates for static robot

From the Fig. 6.15 and Fig. 6.16, it is clear that both, velocity and acceleration model perform really well to estimate the x and y position of a static robot. Initially there is an error due to the incorrect guess of the states but this is very well rectified within $2secs$ as seen from the plots. This shows a good filter performance. The position estimates were compared to the true position, which was found by a tape measure. The estimated position and the true measurement have an error of $3mm$.

5. *The experiment was conducted to estimate the position of the robot when in motion and to compare the velocity model and the acceleration model.*

The marker tag was placed in the work space environment. The two sensors were placed at known locations. The EKF algorithm with both the velocity model and the acceleration model were executed. The marker was moved in different shapes. The dynamic position estimates were obtained and plotted hence, as seen in Fig. 6.17 & Fig. 6.18.

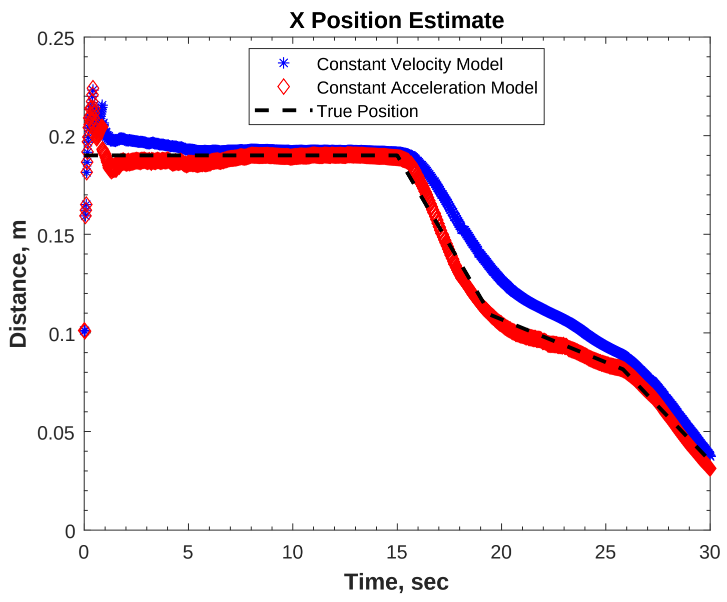


Figure 6.17: EKF X position estimates for robot in motion

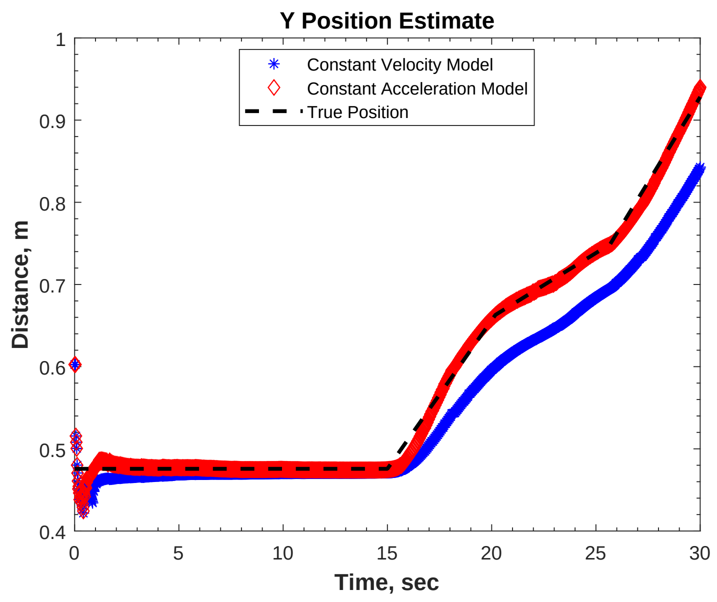
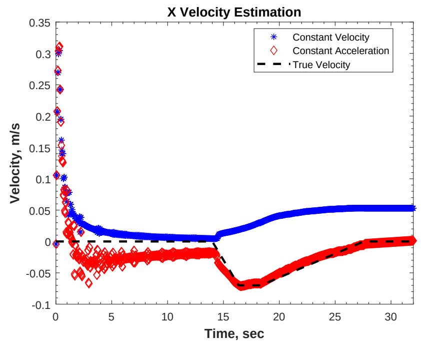


Figure 6.18: EKF Y position estimates for robot in motion

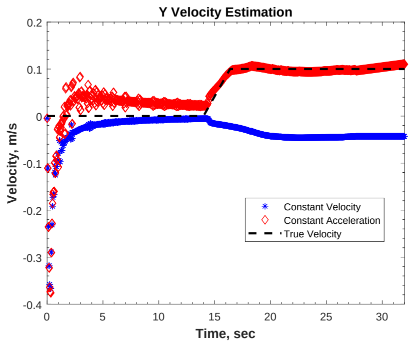
From the Fig. 6.17 & Fig. 6.18, it is clear that both, velocity and acceleration model perform really well to estimate the x and y position of a robot in motion. Initially there is an error due to the incorrect guess of the states but this is very well rectified within $2secs$ as seen from the plots. This shows a good filter performance. The position estimates were compared to the true position, which was found by a tape measure. The estimated position and the true measurement have an error of $12mm$.

6. *The experiment was conducted to estimate the velocity of the robot when in motion and to compare the velocity model and the acceleration model.*

The marker tag was placed in the work space environment. The two sensors were placed at known locations. The EKF algorithm with both the velocity model and the acceleration model were executed. The marker was moved in horizontal straight line at $0.1m/s$. The dynamic position estimates were obtained and plotted hence, as seen in Fig. 6.19.



(a) X direction



(b) Y direction

Figure 6.19: EKF velocity estimates for robot in horizontal motion

From the Fig. 6.19a and Fig. 6.19b, it is clear that the acceleration model perform really well to estimate the velocity of a robot in motion in the x and y direction. Initially there is an error due to the incorrect guess of the states but this is very well rectified within $2secs$ as seen from the plots. This shows a good filter performance. But the velocity model fails to estimate the velocity with any accuracy. The velocity estimates were compared to the true position, which was found by a video time-line. The estimated velocity and the true measurement have an error of $0.01m/s$.

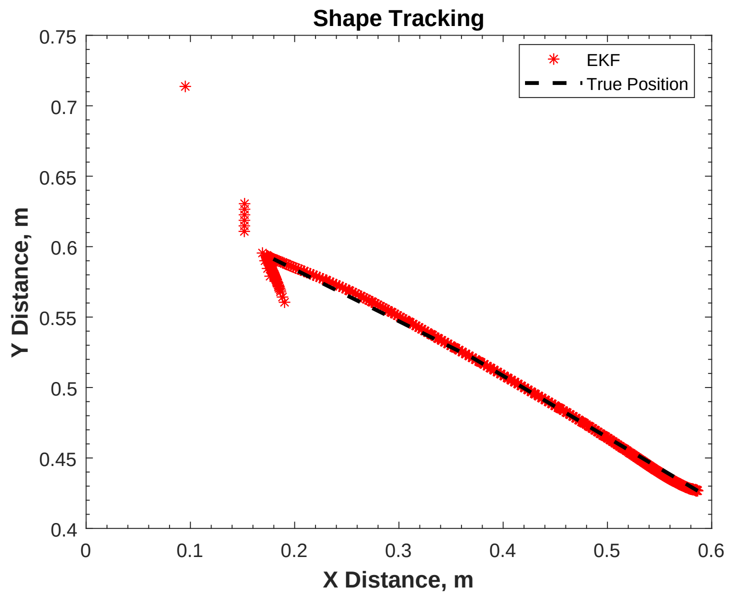


Figure 6.20: Line tracking

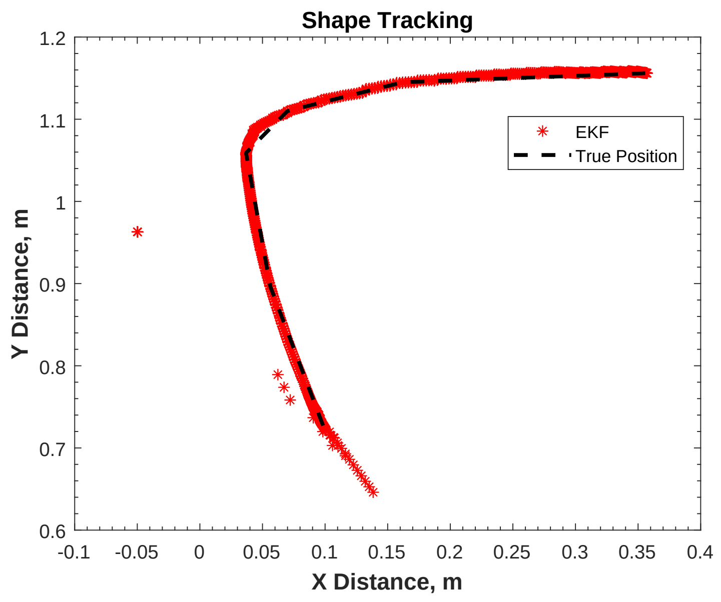


Figure 6.21: 1st shape tracking

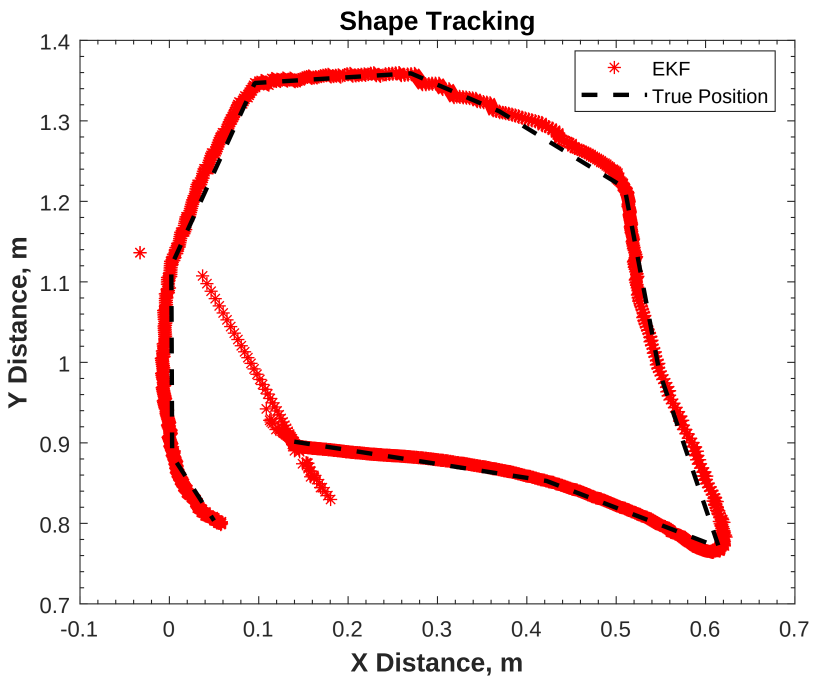


Figure 6.22: 2nd shape tracking

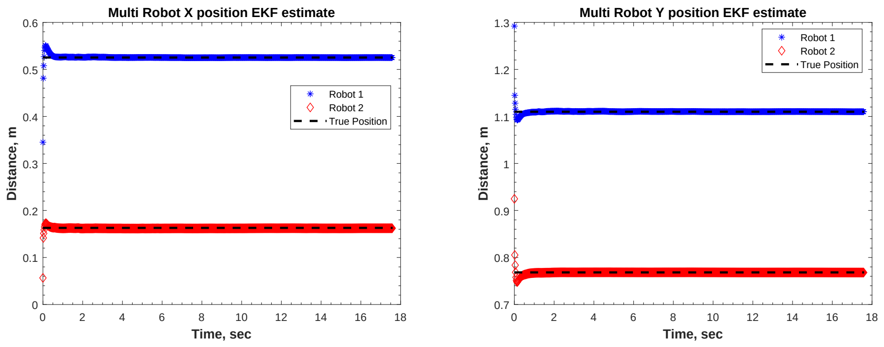
From the above plots it is certain that the filter does a commendable work in tracking the desired trajectory by estimating the robot position accurately.

6.5 Results from Multiple Object Tracking

Here two markers are simultaneously tracked and the performance of the estimation algorithm is observed. There are two cases in which the filter performance is evaluated, static state estimation and dynamic state estimation.

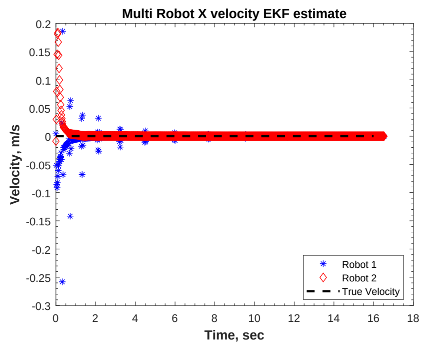
1. *The experiment was conducted to estimate the position of two robots when stationary.*

The marker tag was placed in the work space environment. The two sensors were placed at known locations. The EKF algorithm was executed. The static position estimates were obtained and plotted hence, as seen in Fig. 6.23 and Fig. 6.24.

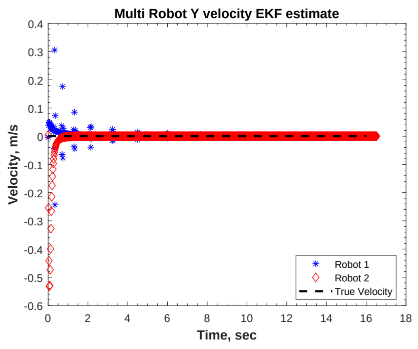


(a) x position estimate (b) y position estimate

Figure 6.23: EKF position estimates for static robots



(a) x velocity estimate



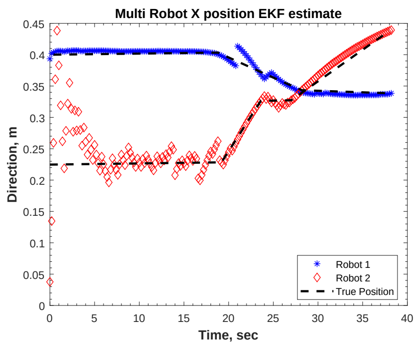
(b) y velocity estimate

Figure 6.24: EKF velocity estimates for static robots

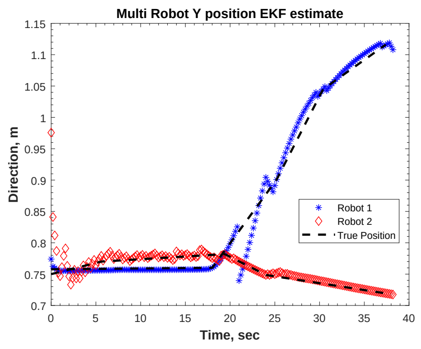
From the Fig. 6.23 and Fig. 6.24, it is clear that EKF perform really well to estimate the x and y position of two static robots in the same work-space. Initially there is an error due to the incorrect guess of the states but this is very well rectified within $1sec$ as seen from the plots. This shows a good filter performance. The position estimates were compared to the true position, which was found by a tape measure. The estimated position and the true measurement have an error of $0.002m$ for marker 1 and $0.015m$ for marker 2.

2. *The experiment was conducted to estimate the position of the two robots when both of the robots are in motion.*

The marker tag was placed in the work space environment. The two sensors were placed at known locations. The EKF algorithm was executed. The marker was moved in specified path i.e. horizontal line. The dynamic position estimates and velocity estimates were obtained and plotted hence, as seen in Fig. 6.25 and Fig. 6.26

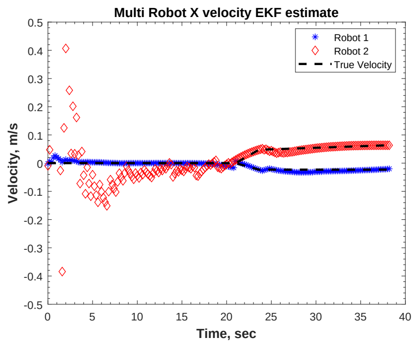


(a) X position estimate

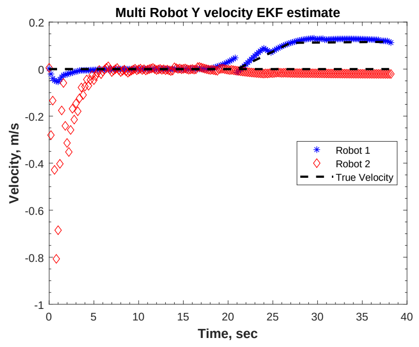


(b) Y position estimate

Figure 6.25: EKF position estimates for multi robot motion



(a) X velocity estimate



(b) Y velocity estimate

Figure 6.26: EKF velocity estimates for multi robot motion

As it is observed from the Fig. 6.25 and Fig. 6.26, the algorithm does a commendable work in estimating the position and velocity of both the robot individually in the work space. The marker 1 was moved in a straight line in the Y direction and marker 2 was moved in a straight line in the X direction. Both these motions are fairly accurately tracked as seen in the Fig. 6.25 and Fig. 6.26. There is more noise in the tracking of marker 1 but the error is small at $0.2m/s$.

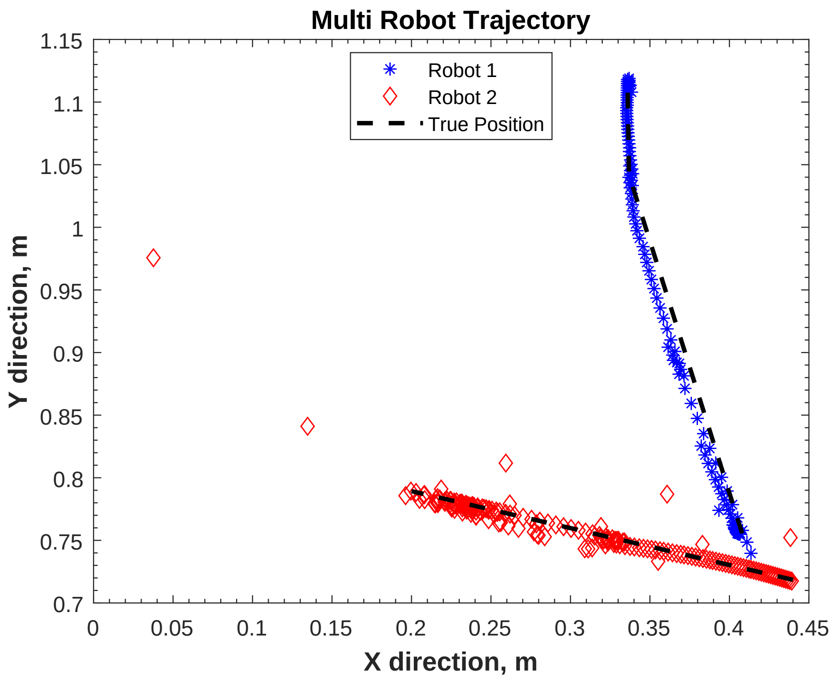


Figure 6.27: Two robot motion tracked by the filter

The Fig. 6.27, shows the path traced by the two markers in real-time. These are the estimated position obtained from the filter. This is very close to the actual path traced.

CHAPTER 7

Summary, Conclusions and Future Work

7.1 Summary and Conclusions

This thesis presents a position estimation technique for indoor navigation purpose. The idea of using two unique range measurement sensors was proposed. Each sensor was based on two fundamentally different working principles. This portraits the universal and generalized requirement for the position estimation algorithm to work with any two range measurement data streams.

The sensors were calibrated and characterized. The sensor capabilities and bounds were determined. The time-of-flight based sensor was found to be more accurate and have a further detecting range than the structured light sensor. The effective work space for each sensor individually was found to be a trapezoid. It was found that the position estimation is correct at any position of the two sensors as long as we provide the algorithm the positions of the sensor and the robot is within the common work space of both the sensors.

The analytic solution was computed for the x and y position of the robot when two range measurements are available. This was tested by implementing the analytic solution in a Simulink model which was capable of outputting the x and y positions in real time. This robot tracking was tested and the results proved the accurate tracking with a caveat that the range measurement must be always available.

In order to overcome this shortcoming, a sequential state estimator was proposed. An Extended Kalman Filter was chose since the measurement model is non linear in nature. For the system dynamic model, two models namely, constant velocity and constant acceleration model was evaluated to see which model is viable based on the attributes being tracked and the application. It was observed that both dynamic models estimate the position fairly accurately for a stationary robot and when the robot is in motion. But for velocity tracking, the constant acceleration model was found to give accurate results, which is reasonable since the velocity being provided was not constant. Hence the constant velocity model failed but the higher dimensional constant acceleration model was able to accurately capture this random change in velocity.

This work was then extended to incorporate multiple robots in the same work space and track each of them individually.It was shown that two robots can be tracked well when both are stationary and when when one of the robot is in motion.

By inspecting all the facets, it is observed that the proposed tracking method is effective in tracking single and multiple robots in the work space of the sensors. This object tracking method is demonstrated to be a viable surrogate GPS for an indoor environment. The ROS framework and the algorithm allows for additional sensors to be incorporated in order to increase the work space for object tracking. It is shown to be a cost effective replacement for other object tracking methods like Laser range finders and RADARs.

7.2 Future Work

The work presented in this thesis can be used as a framework for future research on indoor navigation problem for aerial vehicles. This work can further advance the perception technology for unmanned vehicle systems. The position estimates obtained from this work can be incorporated as feedback information for robots and assist in accurate navigation. Some future progress in this work includes:

- Addition of more sensors to encompass an entire room to be tracking capable area.
- Use different perception sensors for better range measurement data.

Appendices

Appendix A

Analytic Solution

The sensors provide the depth measurements which are of the form Eq. A.1 & Eq. A.2,

$$d_1^2 = (X_1 - x)^2 + (Y_1 - y)^2 + Z_1^2 \tag{A.1}$$

$$d_2^2 = (X_2 - x)^2 + (Y_2 - y)^2 + Z_2^2 \tag{A.2}$$

The Eq. A.1 & Eq. A.2 are solved for x and y using symbolic toolbox in MATLAB, which yields the following result,

$$\begin{gathered}
x = (X1^2 - X2^2 + Y1^2 - Y2^2 + Z1^2 - Z2^2)/(2*(X1 - X2)) - ((Y1 - Y2)*(X1*(-X1^4 + 4*X1^3*X2 - 6*X1^2*X2^2 \\
- 2*X1^2*Y1^2 + 4*X1^2*Y1*Y2 - 2*X1^2*Y2^2 - 2*X1^2*Z1^2 - 2*X1^2*Z2^2 + 4*X1^2*d1^2 + 4*X1*X2^3 \\
+ 4*X1*X2*Y1^2 - 8*X1*X2*Y1*Y2 + 4*X1*X2*Y2^2 + 4*X1*X2*Z1^2 + 4*X1*X2*Z2^2 \\
- 8*X1*X2*d1^2 - X2^4 - 2*X2^2*Y1^2 + 4*X2^2*Y1*Y2 - 2*X2^2*Y2^2 - 2*X2^2*Z1^2 - 2*X2^2*Z2^2 \\
+ 4*X2^2*d1^2 - Y1^4 + 4*Y1^3*Y2 - 6*Y1^2*Y2^2 - 2*Y1^2*Z1^2 - 2*Y1^2*Z2^2 + 4*Y1^2*d1^2 \\
+ 4*Y1*Y2^3 + 4*Y1*Y2*Z1^2 + 4*Y1*Y2*Z2^2 - 8*Y1*Y2*d1^2 - Y2^4 - 2*Y2^2*Z1^2 - 2*Y2^2*Z2^2 \\
+ 4*Y2^2*d1^2 - Z1^4 + 2*Z1^2*Z2^2 - Z2^4)^{(}1/2) - X2*(-X1^4 + 4*X1^3*X2 - 6*X1^2*X2^2 - 2*X1^2*Y1^2 \\
+ 4*X1^2*Y1*Y2 - 2*X1^2*Y2^2 - 2*X1^2*Z1^2 - 2*X1^2*Z2^2 + 4*X1^2*d1^2 + 4*X1*X2^3 \\
+ 4*X1*X2*Y1^2 - 8*X1*X2*Y1*Y2 + 4*X1*X2*Y2^2 + 4*X1*X2*Z1^2 + 4*X1*X2*Z2^2 \\
- 8*X1*X2*d1^2 - X2^4 - 2*X2^2*Y1^2 + 4*X2^2*Y1*Y2 - 2*X2^2*Y2^2 - 2*X2^2*Z1^2 - 2*X2^2*Z2^2 \\
+ 4*X2^2*d1^2 - Y1^4 + 4*Y1^3*Y2 - 6*Y1^2*Y2^2 - 2*Y1^2*Z1^2 - 2*Y1^2*Z2^2 + 4*Y1^2*d1^2 \\
+ 4*Y1*Y2^3 + 4*Y1*Y2*Z1^2 + 4*Y1*Y2*Z2^2 - 8*Y1*Y2*d1^2 - Y2^4 - 2*Y2^2*Z1^2 - 2*Y2^2*Z2^2 \\
+ 4*Y2^2*d1^2 - Z1^4 + 2*Z1^2*Z2^2 - Z2^4)^{(}1/2) + X1^2*Y1 + X1^2*Y2 + X2^2*Y1 + X2^2*Y2 - Y1*Y2^2 \\
- Y1^2*Y2 + Y1*Z1^2 - Y1*Z2^2 - Y2*Z1^2 + Y2*Z2^2 + Y1^3 + Y2^3 - 2*X1*X2*Y1 - 2*X1*X2*Y2)) \\
/(2*(X1 - X2)*(X1^2 - 2*X1*X2 + X2^2 + Y1^2 - 2*Y1*Y2 + Y2^2))
\end{gathered} \tag{A.3}$$

$$
\begin{aligned}
y = {} & (X1*(-X1^4+4*X1^3*X2-6*X1^2*X2^2-2*X1^2*Y1^2+4*X1^2*Y1*Y2-2*X1^2*Y2^2-2*X1^2*Z1^2\\
& -2*X1^2*Z2^2+4*X1^2*d1^2+4*X1*X2^3+4*X1*X2*Y1^2-8*X1*X2*Y1*Y2+4*X1*X2*Y2^2\\
& +4*X1*X2*Z1^2+4*X1*X2*Z2^2-8*X1*X2*d1^2-X2^4-2*X2^2*Y1^2+4*X2^2*Y1*Y2-2*X2^2*Y2^2\\
& -2*X2^2*Z1^2-2*X2^2*Z2^2+4*X2^2*d1^2-Y1^4+4*Y1^3*Y2-6*Y1^2*Y2^2-2*Y1^2*Z1^2-2*Y1^2*Z2^2\\
& +4*Y1^2*d1^2+4*Y1*Y2^3+4*Y1*Y2*Z1^2+4*Y1*Y2*Z2^2-8*Y1*Y2*d1^2-Y2^4-2*Y2^2*Z1^2\\
& -2*Y2^2*Z2^2+4*Y2^2*d1^2-Z1^4+2*Z1^2*Z2^2-Z2^4)^(1/2)-X2*(-X1^4+4*X1^3*X2-6*X1^2*X2^2\\
& -2*X1^2*Y1^2+4*X1^2*Y1*Y2-2*X1^2*Y2^2-2*X1^2*Z1^2-2*X1^2*Z2^2+4*X1^2*d1^2+4*X1*X2^3\\
& +4*X1*X2*Y1^2-8*X1*X2*Y1*Y2+4*X1*X2*Y2^2+4*X1*X2*Z1^2+4*X1*X2*Z2^2\\
& -8*X1*X2*d1^2-X2^4-2*X2^2*Y1^2+4*X2^2*Y1*Y2-2*X2^2*Y2^2-2*X2^2*Z1^2-2*X2^2*Z2^2\\
& +4*X2^2*d1^2-Y1^4+4*Y1^3*Y2-6*Y1^2*Y2^2-2*Y1^2*Z1^2-2*Y1^2*Z2^2+4*Y1^2*d1^2\\
& +4*Y1*Y2^3+4*Y1*Y2*Z1^2+4*Y1*Y2*Z2^2-8*Y1*Y2*d1^2-Y2^4-2*Y2^2*Z1^2-2*Y2^2*Z2^2\\
& +4*Y2^2*d1^2-Z1^4+2*Z1^2*Z2^2-Z2^4)^(1/2)+X1^2*Y1+X1^2*Y2+X2^2*Y1+X2^2*Y2-Y1*Y2^2\\
& -Y1^2*Y2+Y1*Z1^2-Y1*Z2^2-Y2*Z1^2+Y2*Z2^2+Y1^3+Y2^3-2*X1*X2*Y1-2*X1*X2*Y2)\\
& /(2*(X1^2-2*X1*X2+X2^2+Y1^2-2*Y1*Y2+Y2^2)) \qquad \text{(A.4)}
\end{aligned}
$$

In Eq. A.3 & Eq. A.4, $\{X1, Y1, Z1\}$ is the position of sensor 1 and $\{X2, Y2, Z2\}$ is the position of sensor 2 with respect to the origin.

REFERENCES

[1] Ahrens, S., Levine, D., Andrews, G., and How, J. P., “Vision-based guidance and control of a hovering vehicle in unknown, GPS-denied environments,” *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 2643–2648.

[2] Galton Bachrach, A., He, R., and Roy, N., “Autonomous Flight in Unknown Indoor Environments,” Vol. 1, 12 2009, pp. 217–228.

[3] Milella, A., Cicirelli, G., and Distante, A., “RFID-assisted mobile robot system for mapping and surveillance of indoor environments,” *Industrial Robot: An International Journal*, Vol. 35, No. 2, 2008, pp. 143–152.

[4] Le Moigne, J. and Waxman, A. M., “Structured light patterns for robot mobility,” *IEEE Journal on Robotics and Automation*, Vol. 4, No. 5, 1988, pp. 541–548.

[5] Sarbolandi, H., Lefloch, D., and Kolb, A., “Kinect range sensing: Structured-light versus Time-of-Flight Kinect,” *Computer Vision and Image Understanding*, Vol. 139, 2015, pp. 1 – 20.

[6] Zhang, Z., “Microsoft Kinect Sensor and Its Effect,” *IEEE MultiMedia*, Vol. 19, No. 2, Feb 2012, pp. 4–10.

[7] Andersen, M., Jensen, T., Lisouski, P., Mortensen, A., Hansen, M., Gregersen, T., and Ahrendt, P., “Kinect Depth Sensor Evaluation for Computer Vision Applications,” *Technical Report Electronics and Computer Engineering*, Vol. 1, No. 6, 2012.

[8] Wei, Z., Cao, L., and Zhang, G., "A novel 1D target-based calibration method with unknown orientation for structured light vision sensor," *Optics & Laser Technology*, Vol. 42, No. 4, 2010, pp. 570–574.

[9] Khoshelham, K., "Accuracy analysis of kinect depth data," *ISPRS workshop laser scanning*, Vol. 38, 2011, p. W12.

[10] Cruz, L., Lucio, D., and Velho, L., "Kinect and rgbd images: Challenges and applications," *Graphics, Patterns and Images Tutorials (SIBGRAPI-T), 2012 25th SIBGRAPI Conference on*, IEEE, 2012, pp. 36–49.

[11] Fabian, J., Young, T., Jones, J. C. P., and Clayton, G. M., "Integrating the microsoft kinect with simulink: Real-time object tracking example," *IEEE/ASME Transactions on Mechatronics*, Vol. 19, No. 1, 2014, pp. 249–257.

[12] Stowers, J., Hayes, M., and Bainbridge-Smith, A., "Altitude control of a quadrotor helicopter using depth map from Microsoft Kinect sensor," *2011 IEEE International Conference on Mechatronics*, April 2011, pp. 358–362.

[13] Boulos, M. N. K., Blanchard, B. J., Walker, C., Montero, J., Tripathy, A., and Gutierrez-Osuna, R., "Web GIS in practice X: a Microsoft Kinect natural user interface for Google Earth navigation," 2011.

[14] Chang, C.-Y., Lange, B., Zhang, M., Koenig, S., Requejo, P., Somboon, N., Sawchuk, A. A., and Rizzo, A. A., "Towards pervasive physical rehabilitation using Microsoft Kinect," *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2012 6th International Conference on*, IEEE, 2012, pp. 159–162.

[15] Eric, N. and Jang, J. W., "Kinect depth sensor for computer vision applications in autonomous vehicles," *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*, July 2017, pp. 531–535.

[16] Hernandez-Lopez, J.-J., Quintanilla-Olvera, A.-L., López-Ramírez, J.-L., Rangel-Butanda, F.-J., Ibarra-Manzano, M.-A., and Almanza-Ojeda, D.-L., "De-

tecting objects using color and depth segmentation with Kinect sensor," *Procedia Technology*, Vol. 3, 2012, pp. 196–204.

[17] Bailey, S. W. and Bodenheimer, B., "A comparison of motion capture data recorded from a Vicon system and a Microsoft Kinect sensor," *Proceedings of the ACM Symposium on Applied Perception*, ACM, 2012, pp. 121–121.

[18] Bo, L., Ren, X., and Fox, D., "Depth kernel descriptors for object recognition," *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, IEEE, 2011, pp. 821–826.

[19] Ragaglia, M., Zanchettin, A. M., and Rocco, P., "Trajectory generation algorithm for safe human-robot collaboration based on multiple depth sensor measurements," *Mechatronics*, 2018.

[20] Le, A. V. and Choi, J., "Robust Tracking Occluded Human in Group by Perception Sensors Network System," *Journal of Intelligent & Robotic Systems*, Sep 2017.

[21] Nakamura, T., "Real-time 3-D object tracking using Kinect sensor," *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, IEEE, 2011, pp. 784–788.

[22] Desingh, K., Jenkins, O. C., Reveret, L., and Sui, Z., "Physically plausible scene estimation for manipulation in clutter," *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, IEEE, 2016, pp. 1073–1080.

[23] Nair, N., "Implicit Obstacle Detection Using Kinect," .

[24] Eric, N. and Jang, J.-W., "Study on Vehicle Accident Avoidance System Using Kinect Depth Sensor Along with Notifications on Mobile Devices," *International Journal of Applied Engineering Research*, Vol. 12, No. 20, 2017, pp. 9723–9727.

[25] Ghani, M. F. A. and Sahari, K. S. M., "Detecting negative obstacle using Kinect sensor," *International Journal of Advanced Robotic Systems*, Vol. 14, No. 3, 2017, pp. 1729881417710972.

[26] Jian, L., Qiang, I., Yang, Z., Huican, L., and Heng, W., "A kinectV2-based 2D Indoor SLAM Method," *Proceedings of the 2017 International Conference on Artificial Intelligence, Automation and Control Technologies*, ACM, 2017, p. 32.

[27] Kasaei, S. H., Sock, J., Lopes, L. S., Tomé, A. M., and Kim, T.-K., "Perceiving, Learning, and Recognizing 3D Objects: An Approach to Cognitive Service Robots," 2018.

[28] Zakaria, M. F., Shing, J. C., and Tomari, M. R. M., "Implementation of Robot Operating System in Beaglebone Black based Mobile Robot for Obstacle Avoidance Application," *International Journal on Advanced Science, Engineering and Information Technology*, Vol. 7, No. 6, 2017, pp. 2213–2219.

[29] Jiao, J., Yuan, L., Tang, W., Deng, Z., and Wu, Q., "A Post-Rectification Approach of Depth Images of Kinect v2 for 3D Reconstruction of Indoor Scenes," *ISPRS International Journal of Geo-Information*, Vol. 6, No. 11, 2017, pp. 349.

[30] Calero, D., Fernandez, E., and Parés, M., "AUTONOMOUS WHEELED ROBOT PLATFORM TESTBED FOR NAVIGATION AND MAPPING USING LOW-COST SENSORS." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, Vol. 42, 2017.

[31] contributors, W., "Sensor — Wikipedia, The Free Encyclopedia," 2018.

[32] Wikipedia contributors, "Kinect — Wikipedia, The Free Encyclopedia," 2018.

[33] Wikipedia contributors, "Tango (platform) — Wikipedia, The Free Encyclopedia," 2018.

[34] Wikipedia contributors, "Face ID — Wikipedia, The Free Encyclopedia," 2018.

[35] contributors, W., “Structured-light 3D scanner — Wikipedia, The Free Encyclopedia,” 2018.

[36] Physiopedia, “The emerging role of Microsoft Kinect in physiotherapy rehabilitation for stroke patients — Physiopedia,,” 2017.

[37] Widenhofer, B., “Inside Xbox 360s Kinect controller,” 2010.

[38] Wasenmller, O. and Stricker, D., “Comparison of Kinect V1 and V2 Depth Images in Terms of Accuracy and Precision,” 11 2016.

[39] Ibragimov, I. Z. and Afanasyev, I. M., “Comparison of ROS-based Visual SLAM methods in homogeneous indoor environment,” *Positioning, Navigation and Communications (WPNC), 2017 14th Workshop on*, IEEE, 2017, pp. 1–6.

[40] Patrick Mihelich, Suat Gedikli, R. B. R., “openni camera,” 2010.

[41] Mihelich, P., “openni launch,” 2011.

[42] Xiang, L., Echtler, F., Kerl, C., Wiedemeyer, T., Lars, hanyazou, Gordon, R., Facioni, F., laborer2008, Wareham, R., Goldhoorn, M., alberth, gaborpapp, Fuchs, S., jmtatsch, Blake, J., Federico, Jungkurth, H., Mingze, Y., vinouz, Coleman, D., Burns, B., Rawat, R., Mokhov, S., Reynolds, P., Viau, P., Fraissinet-Tachet, M., Ludique, Billingham, J., and Alistair, “libfreenect2: Release 0.2,” April 2016.

[43] Crassidis, J. L. and Junkins, J. L., *Optimal estimation of dynamic systems*, CRC press, 2011.

[44] muralivnv, “asl gremlin,” 2017.

BIOGRAPHICAL STATEMENT

Karan Vishnu Rao obtained his bachelor's degree in Mechanical Engineering from Visvesvaraya Technological University, India. After graduation, he joined The University of Texas at Arlington to pursue Master of Science in Mechanical Engineering. His areas of interest include Guidance, Navigation, and Control of unmanned vehicles and robotics.