Prasanna Kumar Routray

Robô de Entretenimento para Apanhar Bolas em Voo

Entertainment Robot for Catching a Flying Ball

Prasanna Kumar Routray

Robô de Entretenimento para Apanhar Bolas em Voo

Entertainment Robot for Catching a Flying Ball

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Automação Industrial, realizada sob a orientação científica do Doutor Filipe Miguel Teixeira Pereira da Silva, Professor Auxiliar do Departamento de Electrónica,Telecomunicações e Informática da Universidade de Aveiro, e do Doutor Paulo Miguel de Jesus Dias, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri / the jury

presidente / president

Prof. Doutor Pedro Nicolau Faria da Fonseca
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Miguel Armando Riem de Oliveira
Professor Auxiliar Convidado do Departamento de Engenharia Mecânica da Universidade de Aveiro (arguente)

Prof. Doutor Filipe Miguel Teixeira Pereira da Silva
Professor Auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro (orientador)

agradecimentos / acknowledgements

First of all, I would like to thank my thesis advisor, Professor Doutor Filipe Miguel Teixeira Pereira da Silva, and my co-advisor, Professor Doutor Paulo Miguel de Jesus Dias, for their guidance and advice throughout the entire duration of Thesis work and related studies which finally resulted as this thesis. I would like to thank Professor Vítor Santos, coordinator of the LAR, for his guidance throughout this work. I would also like to thank Professor António Amaro and Professor Mário Rodrigues of the School of Health Sciences of the University of Aveiro for their support during human data acquisitions at the Human Motion Analysis Lab. I feel very proud, honored and privileged for having the opportunity to work with these professors. I thank Lentin Joseph for his guidance and helpful advice at critical junctures of this work. I would like to thank the University of Aveiro, Department of Electronics, Telecommunications and Informatics, and Department of Mechanical Engineering, and all the associated administrative staff, for the excellent facilities offered during the course of this research. I thank all my colleagues and teachers at Department of Mechanical Engineering, for the helpful scientific discussions, and for providing a friendly atmosphere. I also use this opportunity to express my gratitude to everyone who supported me throughout this course. Last but not the least, I thank my parents who are always there to support my endeavours.

Palavras-chave Interação humano-robô, apanhar bola em voo, sensor Kinect, estimação de trajectória, manipulador robótico Cyton

Resumo

A Interação Humano-Robô (IHR) surge hoje como uma parte importante e desafiadora da robótica, requerendo tecnologia sofisticada e lidando com importantes aspectos de segurança. Em consonância com isso, a IHR pode ser usado para testar e avaliar tecnologias robóticas avançadas. A tarefa de apanhar uma bola em voo por um sistema robótico pode ser utilizada para avaliação sistemática de sistemas de visão e braços robóticos, quer individualmente quer de forma integrada. A realização deste jogo entre um humano e um robô é um exemplo de uma forma de interação segura que não envolva contato físico. O objetivo principal desta tese está centrado no estudo de um cenário para a realização da tarefa de apanhar um bola em voo usando tecnologia comercial. Na prossecução deste objectivo, são abordados três problemas principais: (1) o desenvolvimento de um sistema de visão para a detecção e seguimento da bola usando um sensor Kinect, (2) a aplicação de algoritmos capazes de fornecerem uma estimativa precisa da trajetória da bola em voo, e (3) o controlo do braço robótico que permita a intercepção da bola. O desenvolvimento da arquitectura software é suportado pelo Robot Operating System (ROS) baseado numa plataforma open-source de arquitetura distribuída. Foram realizados vários testes experimentais para validar as soluções propostas e avaliar o desempenho do sistema em diferentes situações. O teste de viabilidade do trabalho proposto foi realizado com base na simulação do sistema completo tilizando dados pré-gravados do sensor Kinect.

Keywords Human-robot interaction, ball catching, Kinect sensor, trajectory estimation, Cyton manipulator arm

Abstract Human-Robot Interaction (HRI) is now an important and challenging part of robotics as it requires high accuracy and sophisticated technology, along with safety as the first and fore-most aspect. In line with this, HRI can be used for testing and evaluating advanced robotic technologies. Ball catching by a robotic system is one such task that can be used for systematic evaluation of vision and robotic systems, either individually or in an integrated manner. Playing ball catching between a human and a robot is an example of such a form of safe interaction not involving physical contact. The main goal of this thesis is focused towards the study of a possible scenario for ball catching task by a robotic manipulator using off-the-shelf technologies. In the pursuit of that objective, three main problems are addressed: (1) to develop a vision system for ball detection and tracking using a Kinect sensor, (2) to provide trajectory estimation of the flying ball, and (3) to control the robotic arm for interception of the flying ball. The complete software development is supported by Robot Operating System (ROS) with open-source platform and distributed architecture. Several experimental tests are conducted to validate the proposed solutions and to evaluate the system's performance in different situations. Simulation of integrated system for ball catching task is also implemented using pre-recorded data-sets from Kinect sensor for feasibility test of proposed work.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Increasingly present in our daily lives and used to perform various tasks, robotic systems are engaged in new challenges involving the interaction with humans. In the field of robotics, Human-Robot Interaction (HRI) has received considerable attention by the academic community. Human—Robot Interaction (HRI) is a field of study dedicated to understanding, designing, and evaluating robotic systems for use by or with humans [1]. Considering HRI as an advanced testing platform for robotic systems, ball-catching application can be considered as a real challenging task for evaluation of new or existing robotic systems. In this context, the challenging scenario provided by the problem of catching flying objects has been extensively considered in the literature [2]. A body of work has been devoted to playing ball catching between a robot and a human partner as a form of safe interaction (*i.e.*, not involving physical contact). Robotic systems should be able to behave according to situation and ball catching task can be used to test advanced capabilities in terms of vision, perception and arm dynamics.

This dissertation was proposed by the Institute of Electronics and Informatics Engineering of Aveiro (IEETA) in the scope of current activities aiming to design and evaluate robotic systems for Human-Robot Interaction. The robotic system considered for this work consists of a Robai Cyton Gamma 1500 arm along with a vision system based on a depth sensor (*i.e.*, Kinect sensor) installed on the robot's base. This work involves technical and scientific knowledge in the areas of perception, vision, manipulator arm dynamics, estimation and control.

1.1 Motivation

Human-Robot Interaction provides numerous opportunities and robotic systems are becoming more and more advanced in terms of manipulator arm capability and vision systems. Highly dynamic manipulators and advanced vision systems combined together can be used to accomplish difficult tasks. The problem of ball-catching by a robotic system requires a combination of perception and action across time for successful completion: fast reactions in response to perceived movements. In addition to imposing stringent time constraints, a ball catching task requires trading-off the time allocated to perception and action, namely when only one is possible at a given instant. In general, the longer we perceive, the smaller the uncertainty in perceptual estimates. However, a longer perception phase leaves less time for action, which results in less precise movements. In line with this, controlling robot end-effector for catching a flying target is a challenging task requiring the consideration of two closely related problems: First, predicting accurately the trajectory of the moving object, a complicated problem given that sensory inputs are subjected to noise and, more generally, uncertainty. Second, controlling robot's end-effector for catching the moving object.

This work proposes the design of a robotic system able to perform a safe and entertaining task that directly engages a robot with a human partner for catching a flying ball using off-the-shelf technology. Addressing the main challenges of the task demands the implementation of prediction and action based abilities in robots. These required information's are obtained through a motion capture system based on a depth sensor.

1.2 Objectives

Human-Robot Interaction in terms of ball catching application requires study of computer vision system along with manipulator arm dynamics. Depth sensor based vision system is to be used to track the ball and predict its trajectory, position and time required to reach the landing point along with the orientation of the robot to intercept ball trajectory. Once robot has the desired parameters it will try to catch the ball. In order to compensate sensor inaccuracy in robotics (*i.e.*, sensory uncertainties) that influences the estimation of the dynamics of the flying object, proper characterization of vision system is required. As both robot and object are moving, frequent re-estimation and re-planning of manipulator arm motion is required to reach target location in a limited available time. A major cause of failure is the robot's limitation in terms of acceleration, which severely restricts its performance in accomplishing the task. In order to solve those problems, a set of well-defined objectives are framed for this work. The main objectives of this work are listed below.

1. Setup of complete system for the purpose of experimental analysis in terms of both hardware and software. Study of proposed method for ball catching

application.

2. Development of a vision system application that must be suitable for ball catching application with off-the-shelf technology. This goal requires robust method for ball detection and tracking in 3-dimensional space. Study of estimation methods, which are suitable for estimation of trajectory of flying ball, so as to get the position and time required to reach the landing point.

3. Kinematics analysis of the 7 DOF manipulator arm, considering joints limits, joints velocity limits, end-effector velocity in Cartesian space and work-space constraints. This analysis is to be done in accordance with the ball catching application

4. Experimental analysis in simulation mode of developed vision system and manipulator arm motion control system for feasibility test. This means that the developed system is suitable for ball catching application with available resources or not. Simulation based analysis of developed vision system and manipulator arm motion control system on a stand-alone basis to validate the developed methods. The final part is about integration of developed system.

1.3 Dissertation Structure

This dissertation is divided in seven chapters supported by appendices. Chapter 1 presents an introduction to this work and a brief explanation of the topics studied. Chapter 2 presents a review of ball-catching systems, some of the techniques and approaches and also the usual motion control systems available and suitable for this dissertation work. Chapter 3 provides an overview of the experimental set-up, presenting some specifications of the Cyton Gamma 1500 robotic arm and the depth sensor (*i.e.*, Kinect sensor), that are to be used for ball-catching purpose. Chapter 4 is dedicated towards study of vision system for ball detection, tracking and trajectory estimation of a flying ball. Chapter 5 presents the kinematic analysis of the Cyton Gamma 1500 robotic arm and an explanation of the processes to execute robot motion control. In Chapter 6 are discussed the experimental results along with an evaluation of the complete integrated system. Finally, in Chapter 7 are presented some conclusions obtained during this project work and also some possible ideas to improve it in future work.

CHAPTER 2

State-of-the-Art

This chapter reviews the literature relevant for this dissertation work aimed at gaining a better understanding of the problem of robot ball catching, in order to gain a deeper insight into the different approaches taken and assessing the most recent progresses. From the viewpoint of robotics, several research works embarked on a two-pronged approach: one focusing on making improvements to robot design and the vision system, the other on machine learning and computational approaches to provide the necessary algorithms for perception and robot control. However, there is a dichotomy in the way the ever-more sophisticated software is coupled with the used technology. While some works only use off-the-shelf components, without any specialized hardware, others make use of cutting-edge components to provide solutions shaped by technology. At a later stage these works are reviewed by highlighting how estimation of trajectories of moving objects are carried out and how robot's motion is controlled.

2.1 Ball Catching Manipulators

For the case of ball catching application, there are different types of robotic manipulator arms available. These manipulator arms can be of industrial robotic manipulator or assistive robotic manipulator type, but now a days industrial robotic manipulators are more confined to industrial sector. In this regard assistive robotic manipulators are gaining more and more ground either in terms of medical applications or in educational sector. At research and academic level, there are some important case study among which the Rollin' Justin, the A100 Audio-Animatronics, KUKA LWR arm, and Cyton Gamma 1500 arm can be considered. In the next sub-sections are outlined the most relevant ball-catching manipulators.

2.1.1 A100 Audio-Animatronics

A100 Audio-Animatronics [3] is a Walt Disney Imagineering Robot. This robot has 39 degrees-of-freedom, 38 of which are driven by hydraulic actuators. This type of robot platform is currently commonly employed in the theme parks. The robot stands on a 60 cm base containing its hydraulic valve manifold, pressure transducers, and computer connections. Its feet are fixed to the base so stability and balance are not a concern. For throwing and catching it uses its left arm, which has seven degrees of freedom plus five fingers with one degree of freedom each. For additional motions such as orienting the robot towards the participant, simulating ball following, and acknowledging catching failure, additionally the pelvis, torso, shoulder shrugs, neck and eyes are used. Figure 2.1 presents the Walt Disney Imagineering Robot, A100 Audio-Animatronics.

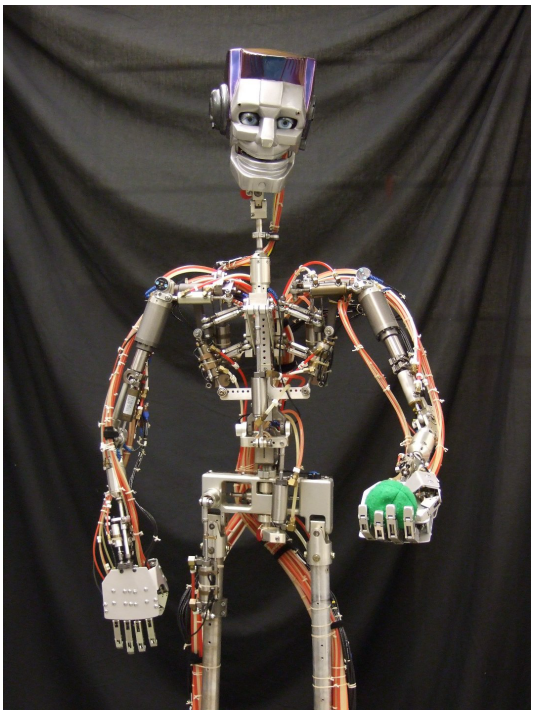


Figure 2.1: A100 Audio-Animatronics, Walt Disney Imagineering robot [3].

The control system for A100 Audio-Animatronics allows for the update of desired joint position set-points at 30 Hz, a rate considered quite slow for reactive control. However, as this robot was designed only for prerecorded trajectory playback, it is one the limitation of this robot. Lower level hydraulic valve controllers realize the desired positions of each actuator at a control loop of 1 kHz. The maximal hand velocity is approximately 1.5 m/s. The left hand of the robot is augmented with a plate to cover and protect the finger actuators and a foam rim to provide a more cup-like shape suitable for catching and to maintain as much of a human-like appearance as possible [4].

2.1.2 Rollin' Justin

Rollin' Justin [5] is an autonomous, programmable humanoid robot. This humanoid robot consists two Kuka assistive robotic manipulators as it's hands and communication with this robot is done by wireless communication. The two modular 7 DOFs assistive manipulator arms consists two four-fingered hands are attached to the torso, that is attached to a mobile base. This structure of Rollin' Justin makes it one the most sophisticated and accurate robotic system. Figure 2.2 presents the the Rollin' Justin, the upper torso combined with the mobile platform.

Figure 2.2: (DLR) Rollin' Justin [6].

Rollin' Justin was designed to perform two-handed manipulation and should be able to reach objects up to 2-meter high as well as objects on the floor. Initially, this robot only had the upper body, however later was implemented with a mobile platform that allowed the robot to interact with humans in different kinds of tasks. Rollin' Justin main features are the visual tracking that gives it the capability to track and grasp freely moving objects in 6 DOFs. The robot also allows commands via speech recognition and dual arm path planning [6] [7].

2.1.3 KUKA LBR Arm

Kuka LBR iiwa [8] arm is very responsive due to its integrated sensors. Since they are not situated in the end-effector, but in the robot itself, simple tools can be used

for any task. This particular manipulator arm can be used in industry environment as well as as an assistive robotic system. The robot can also be programmed by hand moving it (*i.e.*, Robot learning by demonstration [9]. This along with its 7-axes, means that it can easily reach any point within its work zone. It also retains its position during programming, so, Kuka affirms that programming their LBR iiwa series arms are very intuitive and efficient. Its rounded shape contains no sharp edges that could be harmful to a human working beside it. A generic Kuka LBR iiwa arm is as shown in Figure 2.3.

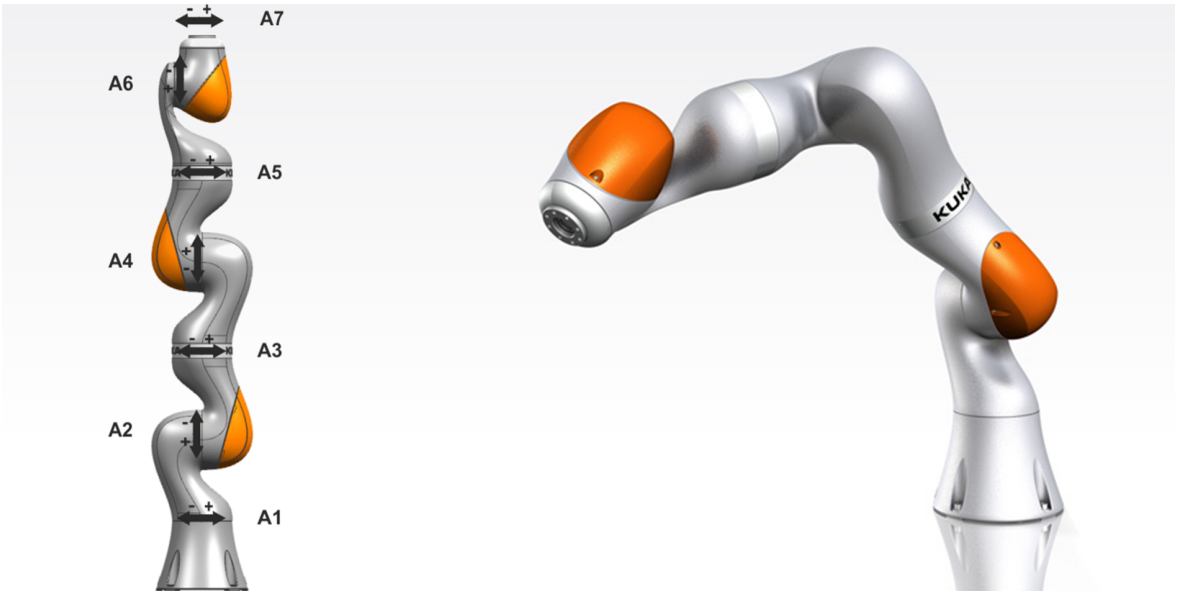


Figure 2.3: 7-axis Kuka LBR iiwa manipulator [8].

It's sensors are capable enough to detect small external forces made by an obstacle or a human. These sensors are also independent from each other. These characteristics make the LBR iiwa arm a safe working partner following basic rules of a robot, either for feeding human workers pieces or holding them while your employees work on it. At research and academic level it (*i.e.*, KUKA LWR 4+, older version of KUKA LBR iiwa robots) is being used for several types applications such as, to catch a hammer, a tennis racket, an empty bottle, a partially filled bottle, and a cardboard box at Learning Algorithms and Systems Laboratory, School of Engineering, EPFL, USA [10].

2.2 Vision Systems

Many of the today's robots are inspired by nature. Robots are replacing humans in the assistance of performing some repetitive and dangerous tasks which humans prefer not to do, or are unable to do due to some limitations. With vision system, a robotic system can be more efficient and it has numerous advantage over basic manipulator arm. Robotic system with vision can be used for manipulation of various

tasks starting perception to execution. Ball catching task is one application where, robot's capability can be tested. Ball catching task requires proper hand-eye coordination because catching a thrown ball with a hand is not easy, neither for humans nor for robots. It requires a better visual sensing mode to reach the necessary precision in space and time. There are two basic methods of vision system (*i.e.*, stereo vision and depth sensor based vision system), that are used for ball catching purpose. These methods are presented in detail in next subsections.

2.2.1 Stereo Vision System

"Two Eyes = Three Dimensions (3D)!" Each eye captures its own view and the two separate images are sent on to the brain for processing. When the two images arrive simultaneously in the back of the brain, they are processed together into one picture. The mind combines the two images by matching up the similarities and adding in the small differences. The small differences between the two images add up to a big difference in the final picture! The combined image is more than the sum of its parts. It is a three-dimensional stereo picture. The word "stereo" comes from the Greek word "stereos" which means firm or solid [11]. main concept of stereo-vision can be understood from Figure 2.4.

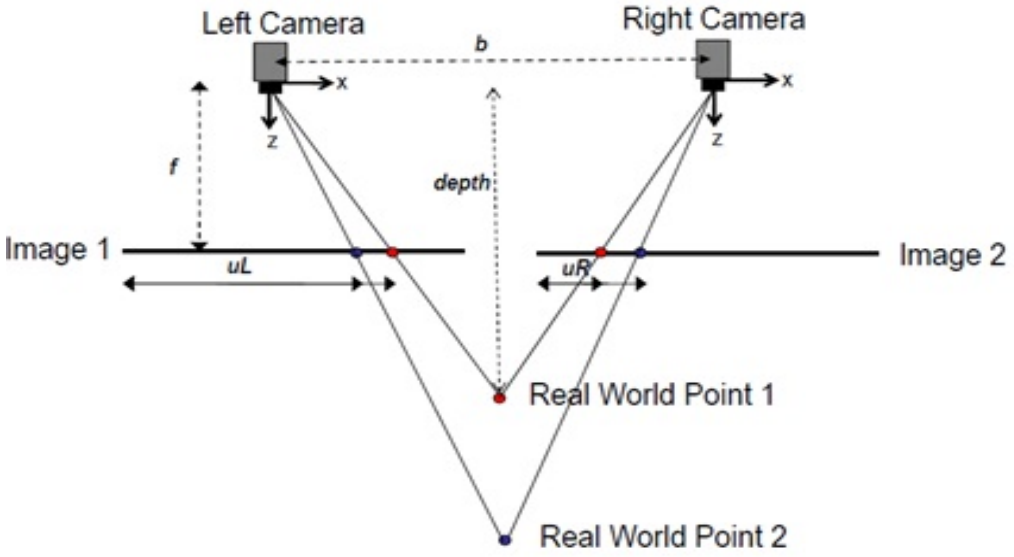


Figure 2.4: Stereo vision concept [11].

With stereo vision one can see an object as solid in three spatial dimensions (*i.e.*, width, height and depth or x, y and z). It is the added perception of the depth dimension that makes stereo vision so rich and special. With stereo vision, one can see where objects are in relation to it's own body with much greater precision, especially when those objects are moving toward or away from it in the depth dimension. A person can see a little bit around solid objects without moving his head and he can

even perceive and measure ”empty” space with his eyes and brain [11]. In many of the cases this kind of vision system is used for ball-tracking task. This specific work of ”Estimation and prediction of multiple flying balls using probability hypothesis density filtering [6]” is done using stereo vision system.

2.2.2 Depth-Sensor Based Vision System

This kind of sensors basically work with the principle of structured light. Microsoft® Kinect sensor is today's best known structured light-based 3D sensor. The structured light approach, is one example of an active non-contact scanner; non-contact because scanning does not involve the sensor physically touching an object's surface, and active because it generates its own electromagnetic radiation and analyses the reflection of this radiation from the object. Typically, active non-contact scanners use lasers, LEDs, or lamps in the visible or infra-red radiation range. Since these systems illuminate the object, they do not require separate controlled illumination of the object for accurate measurements. An optical sensor captures the reflected radiation [11].

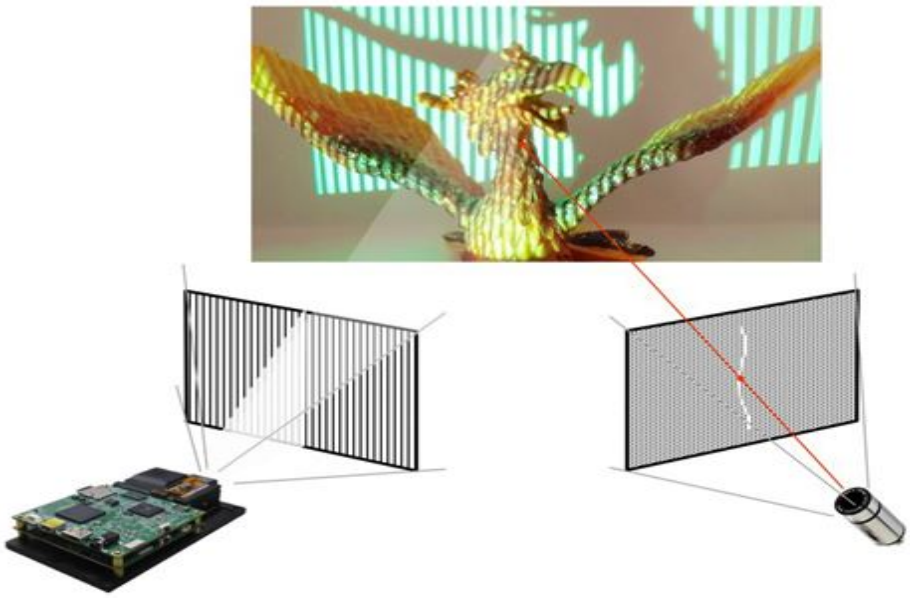


Figure 2.5: Depth sensor based vision system concept [11].

Structured light is an optical 3D scanning method that projects a set of patterns onto an object, capturing the resulting image with an image sensor. Structured light replaces the previously discussed stereoscopic vision sensor's second imaging sensor with a projection component. Similar to stereoscopic vision techniques, this approach takes advantage of the known camera-to-projector separation to locate a specific point between them and compute the depth with triangulation algorithms. Thus, image processing and triangulation algorithms convert the distortion of the projected patterns, caused by surface roughness, into 3D information (Figure 2.5).

In the case of Disney's robot A100 Audio-Animatronics depth sensor based vision system is used [3].

2.3 Technology-Oriented Works

Robots have been playing catch with humans for years: it is a dynamic, interaction that does not require physical contact with the human while also a technologically complex task. The majority of experimental set-ups reported in literature on link-flexible robot arms operate only in the horizontal plane. For these systems gravitational effects are neglected. Recent works are driven by challenges experienced in terrestrial applications, where the non-linear gravitational influence has to be considered. Over the years the robotic systems used as well as the perception and planning methods got more and more complex.

2.3.1 Related Works

Jorn Malzahn [2] proposes a control architecture, which enables a multi-link flexible robot arm under gravitational influence to catch multiple balls sequentially thrown by a human. A net at the end-effector is utilized to intercept the balls when they pass the vertically oriented robot plane of motion. The ball detection, tracking as well as the prediction of the ball intercept location is based on a wall-mounted Kinect RGB-D sensor. Previously caught balls represent a varying payload, which induces also dynamic disturbances due to the pendulum motion. These disturbances as well as the coupled flexible-link vibrations are damped with a model free independent joint controller. The inverse kinematics approach is based on neural networks and augmented by an on-line payload estimation. The ball trajectories generated by the human thrower are not reproducible. There are no selected operating points, but the trajectory may pass through the robot work-space at any point.

Georg Batz [12] proposes a special approach to accomplish the task: the non-prehensile catching using single Kinect RGB-D sensor. Jwu-Sheng Hu [13] proposes a robotic ball catcher with embedded visual servo processor. Servo processor is having the powerful parallel computing capability, which is used to track and navigate a flying ball. Author uses the recursive least square algorithm for trajectory prediction. Planning human-like robot trajectories for catching rapidly moving targets is a challenging task. Seungsu Kim [14] considers a novel approach to control the timing of motions when these are encoded with autonomous dynamical systems(DS). Accurate timing of motion is crucial if a robot must synchronize its movement with that of a fast moving object. Disney has designed a humanoid robot that can play catch and juggle with humans. The A100 Audio-Animatronics [3] can catch a ball and throw it back with a human participant using a Microsoft® Kinect sensor. According to a video the company released, Disney plans on using similar technology to make its rides more interactive and realistic. The color and depth cameras on the Kinect sensor are used to track the colored balls and a Kalman filter or equation helps predict where they'll land.

2.3.2 Our Approach

The ultimate goal is to perform a ball catching task directly engaging the physical robot system with a human partner. However, the main objectives of the proposed work are divided in to five sub goals: First, to study about suitable vision system with off-the-shelf technology (*i.e.*, Kinect sensor). Second, to evaluate the available robotic system's performance in terms of joints velocity limits and end-effector velocity in Cartesian space. Third, to study the basic estimation methods that can be used for ball catching task. Fourth, to explore simulation scenario for testing of developed vision system in ROS platform. Fifth, to study the possible scenario for integration of developed vision system and manipulator arm control system for ball catching task. Considering robotic system as an integral part of this work, the primary objective is to evaluate the performance of the robotic system by performing several experiments before final integration, for example, the maximum velocity in joint space and end-effector Cartesian space, that the manipulator arm can attain. Below are presented the sub goals more precisely.

1. **Development of vision system:** A suitable vision system should be established in order to detect and track the ball. This task is to be accomplished using a depth sensor, that is available in the laboratory. This vision should be reliable and robust in the sense that, it should be able to detect the ball under different circumstances. If the vision system is not robust enough then, it will be difficult to detect and track a flying ball that will be ultimately used for ball catching purpose. In order to address the associated problems, several trials under different environmental conditions are to be carried out for setting a result oriented application.

2. **Development of manipulator arm motion control system:** It is worth noting that for the application ball catching task, it is necessary to evaluate the manipulator arm under different operational conditions. These conditions include, testing of maximum joint space velocities the arm can attain along with an overall analysis of end-effector velocity in Cartesian coordinate space. These experiments are to be carried out for many trials in order to get a concrete information about the manipulator arm's capability. This can be termed as the performance evaluation of the available manipulator arm (*i.e.*, Cyton Gamma 1500) in laboratory.

3. **Study of estimation methods:** For the purpose of ball catching task, if robust ball detection and tracking can be done, then it is necessary to estimate ball's trajectory along with landing point over a plane of fixed height or inside a certain volume. Several methods for the purpose of estimation of trajectory exists, such as polynomial approximation, energy minimization, Kalman filter. Some of these methods are of concern for the purpose of ball trajectory estimation in the scope of this dissertation work, considering the task of ball catching by a robotic arm.

4. **Simulation scenario:** Before implementation of any kind of real-time application, it is always required to asses the developed system in a simulation environment. Simulation environment provides safe and hassle-free implementation of developed ideas for testing purpose. ROS has some of it's own simulation environments such as, rviz and Gazebo. Gazebo simulator provides real-time environment for simulation purpose and rviz is the basic visualization platform that can be used for simulation and planning along with MoveIt! library. These platforms are to be explored for implementation of developed applications before real-time implementation. In addition to that MATLAB® can also be used for simulation purpose.

5. **Integration of developed systems:** Ultimate goal of this dissertation work is the integration of developed individual systems (*i.e.*, vision system and manipulator arm motion control system) for real-time ball catching task. For this to happen, several trials are to be done in order to get conclusion about feasibility of the proposed method for this dissertation work.

Estimation of ball trajectory requires good sample at the very beginning of ball catching task. Depth sensor has it's internal error like all other sensors, and there can be other influential factors for error accumulation. Ground truth is a required for comparison of estimation methods along with the data from depth sensor. For the purpose of ground truth, a key component of the work is the acquisition of motion capture data. Motion capture will be performed at the Human Gait Laboratory (ESSUA-UA) equipped with a gold-standard Vicon Opto-electronic system with 8 infra-red cameras [15]. A standard marker set will be attached to the flying ball for motion capturing purpose. The three-dimensional coordinates of the markers will be collected and stored for later analyses. Thus, an important activity will be the definition of the most adequate experimental protocols with the choice of marker placement over the ball.

CHAPTER 3

Experimental Set-up

This chapter is about the components that are required to carry out experiments concerning this dissertation work. For the application of ball catching task, it is required to go for a complete setup and assessment of available features before experimentation phase. Components can be hardware, software and in this case both are required to accomplish ball catching task. It requires a robotic manipulator, a vision system that should be able to give enough information of a flying ball, software development tools for the purpose of controlling through program. Software development tools are meant to control the hardware through programming.

The main hardware components consist of a dexterous robotic arm that is ©Cyton Gamma 1500 by Robai corporation [16], a Microsoft® Kinect sensor [17], and a central processing unit (PC-based). From the very beginning of the project a few assumptions underlying the work were defined: First, the vision system consists of a depth sensor which holds a compromise among accuracy, sensing range and price.

In the case of software, according to the requirement of this work, the software architecture should be based on distributed computation and control type. The key element for this software architecture is the Robot Operating System5 [18] (ROS), that provides an extensive list of libraries and tools to create robotic applications. Other libraries include the OpenNI driver for the depth sensor and the dynamixel controller for low level control of the manipulator arms, OpenCV [19] inside ROS for vision system in order to achieve image processing capability.

3.1 Overall System Architecture

In accordance with the overall system architecture, the software architecture is based on the Indigo version of the ROS framework under Ubuntu 14.04. C/C++ program-

ming language to be used for coordination among all the components. Figure 3.1 illustrates the overall system's architecture, both in terms of hardware and software level.

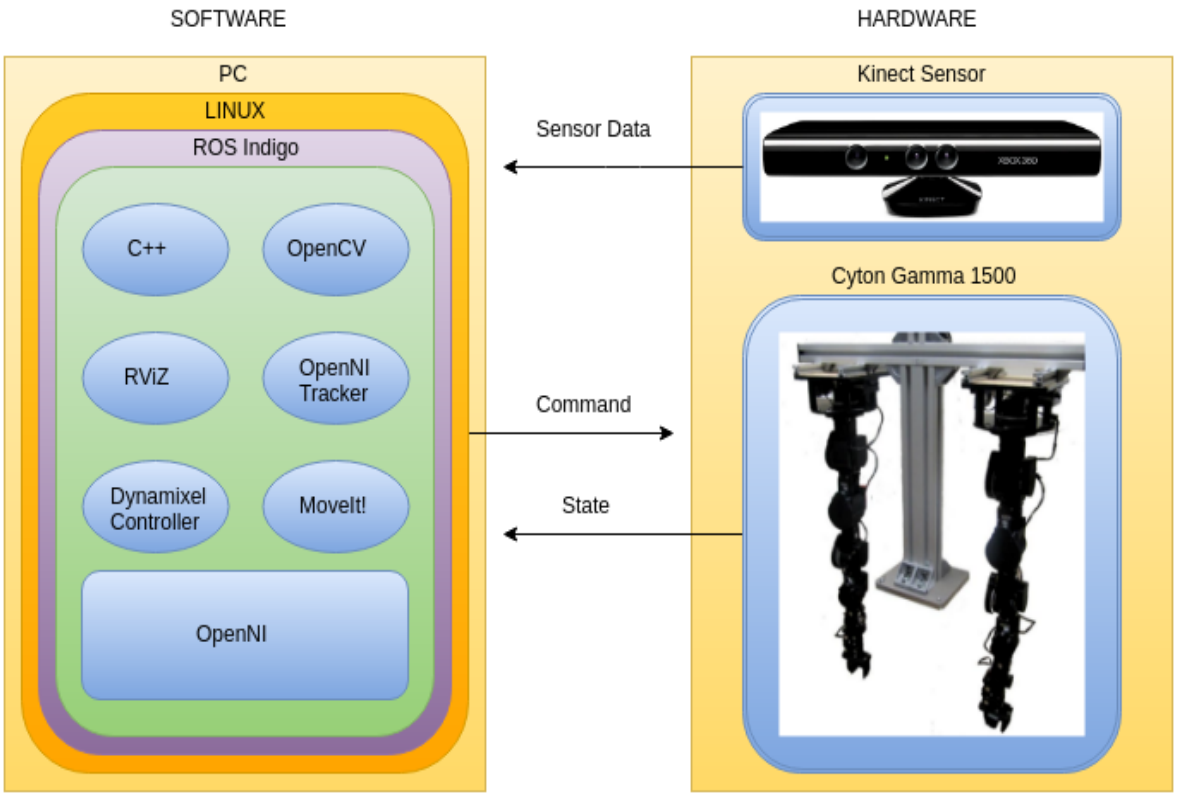


Figure 3.1: Integrated system architecture.

The Hardware used in this thesis includes Cyton Gamma 1500 dexterous robotic arm and a low cost depth sensor. Both of these hardware can be connected to PC with USB interface. Cyton Gamma 1500 supports TTL as well as serial mode of communication, where as depth sensor supports USB based communication. The communications protocol between the PC and the Cyton Gamma 1500 robotic system is established through USB controllers in TTL mode, with a speed limit of approximately 8000 bps ~ 4.5 Mbps [20] [21], using the functions provided by the dynamixel controller to send and receive the commands. The dynamixel is a smart actuator system developed to be the exclusive connecting joints on a robot or mechanical structure. Dynamixel motors are designed to be modular and daisy chained on any robot or mechanical design for powerful and flexible robotic movements.

Among software components ROS works as a primary agent inside Ubuntu platform. OpenCV, dynamixel controller, MoveIt!, rviz, OpenNI and OpenNI tracker, all constitute together to form a complete software system for ball catching application by Cyton Gamma 1500 dexterous robotic arm using vision system based on depth sensor.

3.2 Cyton Gamma 1500 Manipulator Arm

Cyton Gamma 1500 was introduced by Robai Corporation and this model offers increased joint torques compared to other versions. This arm has 7-DOFs and also a motor to control the gripper. These arms have kinematic redundancy that enables the placement of the end-effector at a position and orientation in many different ways. It can be interesting to avoid obstacles or to reach the goal in different configurations but it turns the kinematics of the arm more difficult. Each joint actuator can provide position, speed, load, voltage and temperature feedback information and the user is able to configure these parameters using some predefined python script for dynamixel controllers. Figure 3.2 presents an overall structure and dimension of 7 DOF Cyton Gamma 1500 manipulator arm.

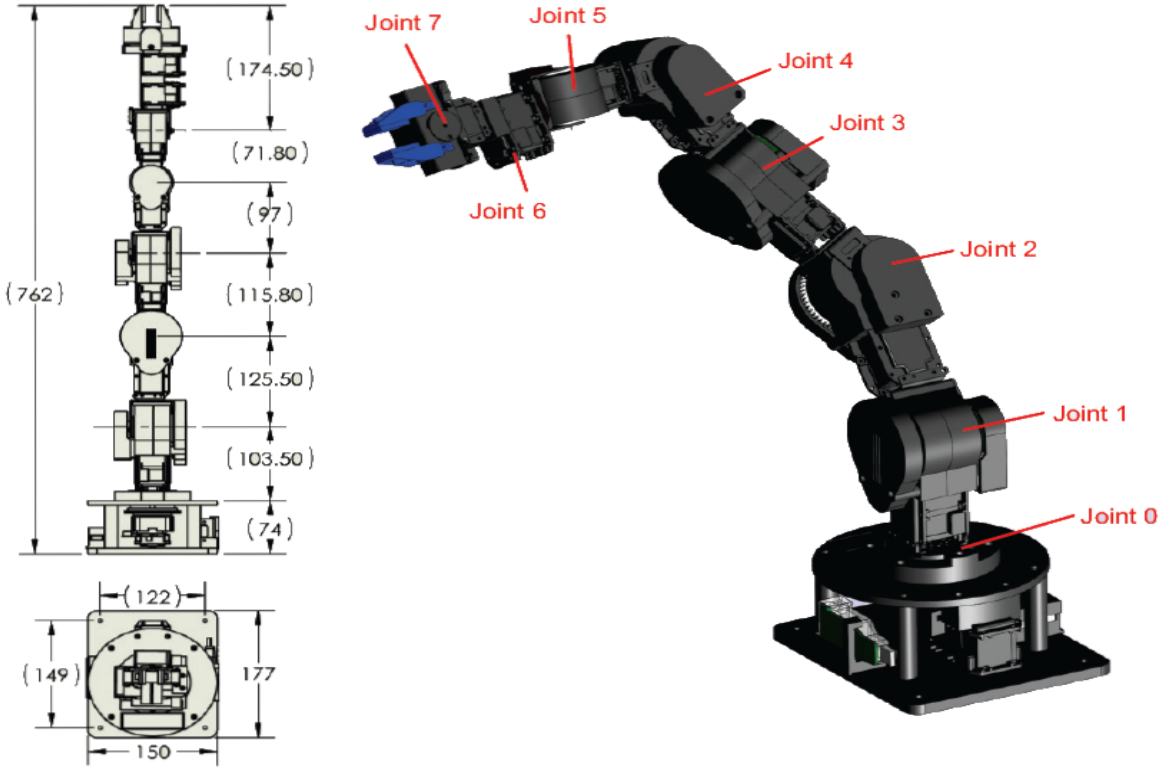


Figure 3.2: 7-DOF Cyton Gamma 1500 manipulator arm dimensions [22].

Technical Features

In Table 3.1 are described some technical specifications of the Cyton Gamma 1500 manipulator. They are related to physical characteristics such as its weight and length, to its performance such as its maximum linear speed and repeatability and also to its electrical and control interface. Joint specifications related to Cyton Gamma 1500 is presented in Table 3.2. Joint angle and joint velocity limits as well as the corresponding servos are also presented in Table 3.2. Table 3.3 ,presents two types of

Dynamixel servo MX-28 and MX-64 motors that are used in Cyton Gamma 1500.

Table 3.1: Cyton Gamma 1500 technical specifications [22].

Parameter	**Specified value**
Total weight	3 Kg
Payload at full reach	1200 g
Payload at mid reach	1500 g
Arm length from base to tip	76 cm
Reach	68 cm
Maximum linear speed	45 cm/sec
Maximum speed (free move)	70 cm/sec
Repeatability	± 0.5 mm
Input Voltage	100-240V AC or 12 DC 2A battery
Current	2.5 A max in normal use
Control interface	USB or RS485
Total independent joints	7

Table 3.2: Cyton Gamma 1500 joint specifications [22].

Joint name (type)	**Angle limits (degrees)**	**Joint velocity limits (Degrees/sec)**	**Servo model**
Shoulder roll - joint 0 (spin)	-150 to 150	75	MX-64
Shoulder pitch - joint 1 (articulate)	-105 to 105	75	MX-64
Shoulder yaw - joint 2 (articulate)	-105 to 105	75	MX-64
Elbow pitch - joint 3 (articulate)	-105 to 105	65	MX-28
Wrist yaw - joint 4 (articulate)	-105 to 105	110	MX-28
Wrist pitch - joint 5 (articulate)	-105 to 105	330	MX-28
Wrist roll - joint 6 (spin)	-150 to 150	330	MX-28

Table 3.3: Cyton Gamma 1500 dynamixel servo motor specifications [20] [21].

Servo model	Servo resolution (degrees)	Servo gear ratio	Position sensor	Stall torque (N.m)
MX-64	0.088	200:1	Contact-less absolute encoder (12 bit, 360 degree)	6.0 (@ 12V, 4.1A)
MX-28	0.088	193:1	Magnetic potentiometer (12 bit, 360 degree)	2.3 (@ 12V, 1.5A)

3.3 Kinect Sensor

For long ago, robots and computers were able to process images that are provided by cameras, and with hard effort, it was possible to extract information about objects. By processing two dimensional images, three dimensional data are extracted. The computational cost was high and the precision and quality of the measures were not good considering invested equipment cost. The 3D sensors, such as the Kinect sensor shown in Figure 3.3 [23], changed all these with the implementation of depth sensor along with RGB camera in one small box.

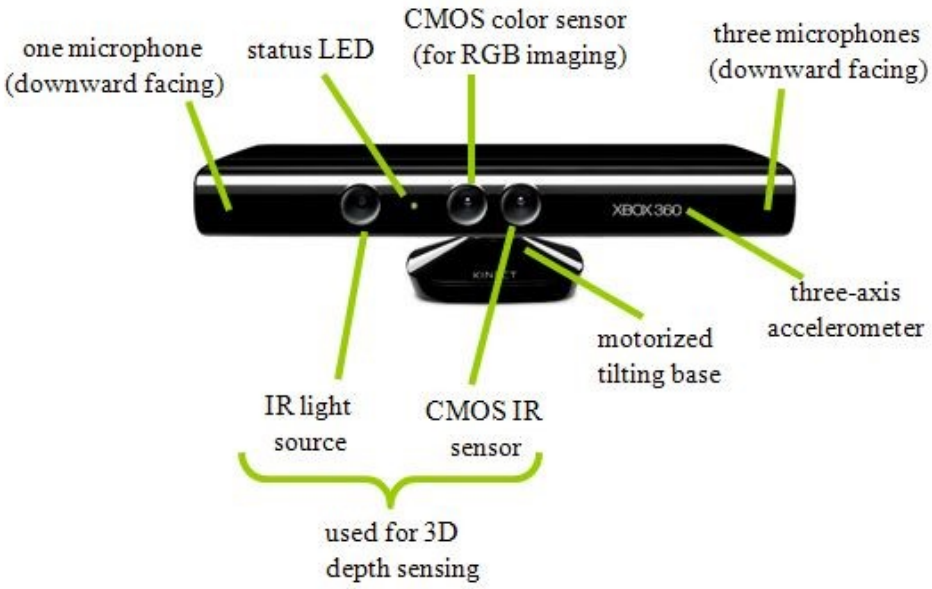


Figure 3.3: An assembled Microsoft XBOX Kinect sensor [23].

It can be considered a special case of 3D sensors, that is based on light coding. Actually, the technology used in the depth sensor is active triangulation with struc-

tured light. It has a CMOS RGB sensor. This video camera aids in facial recognition and other detection features by detecting three color components (*i.e.*, red, green, and blue). Microsoft calls this an "RGB camera" referring to the color components it detects.

Technical Features

Kinect sensor is not made up of just one sensor. It has got several components. The motorized base enables to change the vertical range of acquisition up or down of ± 28° as mentioned in Table 3.4. The Figure shows the three main sensor of the Kinect or better the Kinect sensor's eyes: two cameras and an Infra-red (IR) projector. The first component on the left is the IR projector as illustrated in the Figure 3.3, the central components is a Color Complementary Metal Oxide Semiconductor (CMOS), a simple Red, Green and Blue (RGB) camera with a resolution of 640×480 32-bit color at 30 frames/sec and lastly on the right we have the IR CMOS or the IR Receiver with a resolution of 320×240 16-bit depth at 30 frames/sec. The device is also equipped with an array of microphones that permits the Kinect to receive vocal commands. The multi-array microphone enables acoustic source localization and ambient noise suppression. The four microphones are disposed in a line, three of them in the left side and another in the right, all of them placed below of the device.

Table 3.4: Technical specifications of Microsoft XBOX Kinect V1.0 [24].

Sensor item	**Specification**
Viewing angle	43° vertical by 57° horizontal field of view
Mechanized tilt range (vertical)	± 28°
Frame rate (depth and color stream)	30 frames per second (FPS)
Resolution, depth stream	QVGA (320×240)
Resolution, color stream	VGA (640×480)
Audio format	16-kHz, 16-bit mono pulse code modulation (PCM)
Audio input characteristics	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing such as acoustic echo cancellation and noise suppression

From Figure 3.4, it is evident that the Kinect sensor is not a single piece device but it's composed by a lot of different components and technologies to offer the user a new entertainment experience that involves different senses.

Figure 3.4: A disassembled Microsoft XBOX Kinect sensor [24].

3.4 Software Development Tools

This section is all about the software tools that are required for this dissertation work. Considering Cyton Gamma 1500, it requires low level interface software tool for controllers and depth sensor requires a middle-ware for data acquisition. Low level interface with Cyton Gamma 1500 is accomplished by Dynamixel driver. High level control can be done either in C++ or python. Dynamixel driver is compatible with ROS's latest versions and overall control interface for Cyton Gamma 1500 can be implemented in ROS using Dynamixel motor package. For the purpose of joint level control only Dynamixel motor package is enough, but for the mode of inverse kinematics control, additional tools are needed for the calculation of joint space parameters from the end-effector Cartesian coordinate. Method of inverse kinematics calculation can be either analytic or based on numerical methods.

In next subsections are presented some features of the ROS, the OpenNI package used to acquire data. OpenCV for image processing required for vision system, rviz for visualization in ROS and OpenNI tracker for human skeleton. ROS was chosen for this work because of it's advantage over other platforms such as: it is more versatile, flexible and easier to use. It allows us to create individual processes, which in turn can communicate between each other through specific kind of messages. Each process can be programmed using different programming languages. Thus, if it is necessary to add new functionality or features to the system it is just a matter of creating a new process and incorporating it to the existing system. OpenNI package works as a middle-ware between ROS and depth sensor. This allows us to retrieve images in different format, point-cloud data and information about connected camera. OpenCV enables us to perform image processing task for vision system. rviz is the visualization tool integrated in ROS, that can be used for vision system or

manipulator arm based visualization either in real-time or in simulation mode.

3.4.1 Robot Operating System Framework

The Robot Operating System (ROS) framework is the unifying element of this project. It defines the interaction between the Cyton Gamma 1500 manipulator and the depth sensor, using the functionality already present on the ROS platform and other software personally developed. ROS is a framework that is widely used in robotics. The idea is to create functionality that can be shared and used in other robots without much effort. ROS was originally developed in 2007 by the Stanford Artificial Intelligence Laboratory (SAIL). The platform is now maintained by the open-source community and Willow Garage.

ROS Programming Environment

Lot of research institutions and companies have started to develop projects and adapt their products to be used in ROS. With the large increase of the community in research and development of robotic systems it was noticeable the several difficulties developing software applications for robotic systems. This is mostly due to the fact that each robot have a particular communication protocol, each camera has a specific image format and the need that can be acquired data from sensors running in different computers are some of the many difficulties experienced when there is the need to develop robot software. In order to solve or minimize this difficulties Robot Operating System was developed. Table 3.5 presents a comparison between conventional methods and ROS for robot control.

Table 3.5: Conventional Vs ROS method of robot control.

General Purpose	**ROS**
Explicitely for general purpose	Explicitely for Robots
Native language programming	language Independent
Sequential architecture	Asynchronous distributed
Programming IDE	Software framework
Proprietary/open source	Open-source BSD license
Heavily coded	ROS frameworks are very light
Programs	nodes
Communication	Messages
Splintered usage	Industry-wide and academic usage

ROS provides interesting and useful features and services such as hardware abstraction, simple mechanisms of communication between processes, package management, software reuse, easy and rapid testing and language independence. Those

features made ROS a powerful tool for robotics software development. ROS programs and libraries that perform a certain function are grouped into packages. Inside each package are stored source code, libraries, binaries, manifest.xml file, where the declaration of a packages dependencies over other packages are stored. The CMakeList.txt file is the one, which contains instructions for the CMake compilation of any package.

Figure 3.5 presents a simple example of message exchanging between three nodes. Nodes are represented by ellipses and topics are represented by rectangles. This example shows three nodes, A, B, and C and three topics, A, B and C. Node A does not subscribe any message but it is publishing messages on topic A and B. Node B and C are subscribing to messages of topic B. Node B is publishing messages on topic C. Node C is subscribing only to topic C for any different task. Each node can be programmed to publish to a certain topic at a specific frequency and it can be different in all nodes of the program. This explains that the exchange of complex messages can be easily managed by ROS framework and also easier to implement.

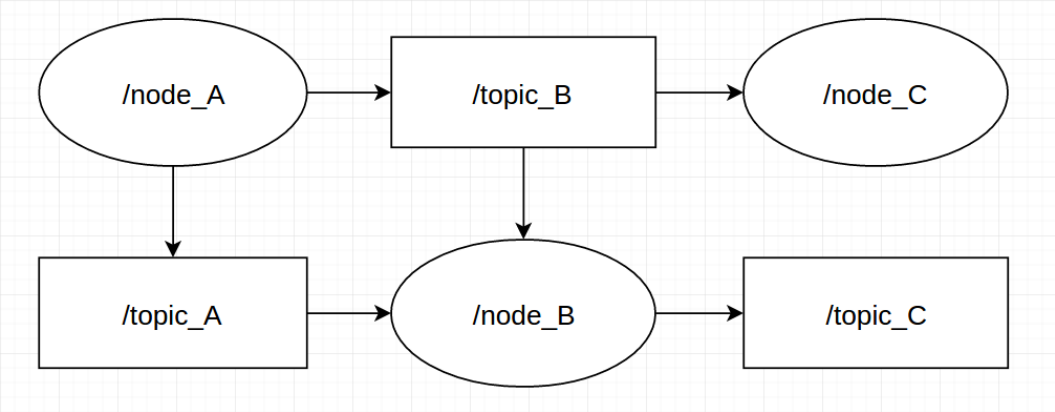


Figure 3.5: An example of nodes and topics exchanging information.

Communication with the help of nodes and topics is the primary method of communication between but it has some limitation. First it is unidirectional and the messages are published to any node that wants to subscribe it and there is no response. Alternatively, services are bi-directional and implements one-to-one communication. Thus, one node sends information to another and waits for response, this method is based on server/client topology as shown in the diagram of the Figure 3.6.

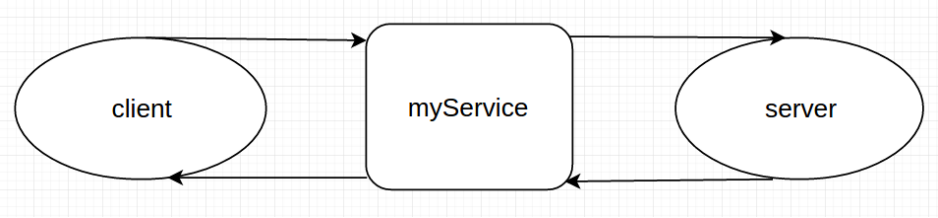


Figure 3.6: Basic services communication mechanism in ROS.

ROS Visualization Tool (rviz)

ROS visualization tool (rviz) is a 3D visualizer for displaying sensor data and state information from ROS. Using rviz, you can visualize Baxter's current configuration on a virtual model of the robot. You can also display live representations of sensor values coming over ROS Topics including camera data, infra-red distance measurements, sonar data, and more [25].

3.4.2 OpenCV

OpenCV (Open Source Computer Vision) is a library of programming functions for real time computer vision. It is developed by Willow-Garage, which is also the organization behind the famous Robot Operating System (ROS).

OpenCV Programming Environment

OpenCV is released under a BSD license and hence it's free for both academic and commercial use. It has C++, C, Python and Java interfaces and supports Windows, Linux, Mac OS iOS and Android. OpenCV was designed for computational efficiency and with a strong focus on real-time applications. Written in optimized C/C++, the library can take advantage of multi-core processing. Enabled with OpenCL, it can take advantage of the hardware acceleration of the underlying heterogeneous compute platform. Usage of OpenCV ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics. Some of the acceptable features of OpenCV over other available tool are given below.

- **Speed:** OpenCV is basically a library of functions written in C/C++. We are closer to directly provide machine language code to the computer to get executed. So ultimately we get more image processing done for our computers processing cycles, and not more interpreting. As a result of this, programs written in OpenCV run much faster than similar programs written in Matlab. So, OpenCV is very fast when it comes to speed of execution. For example, we might write a small program to detect people's smiles in a sequence of video frames. In Matlab, we would typically get 3-4 frames analyzed per second. In OpenCV, we would get at least 30 frames per second, resulting in real-time detection.

- **Memory Management:** OpenCV is based on C. As such, every time we allocate a chunk of memory we will have to release it again. If we have a loop in our code where we allocate a chunk of memory in that loop and forget release it afterwards, we will get what is called a 'leak'. This is where the program will use a growing amount of memory until it crashes from no remaining memory. Due to the high-level nature of Matlab, it is 'smart' enough to automatically allocate and release memory in the background.

- **Development Environment:** Matlab comes with its own development environment. For OpenCV, there is no particular IDE that we have to use. Instead, we have a choice of any C programming IDE depending on whether we are using Windows, Linux, or OS X. For Windows, Microsoft Visual Studio or NetBeans is the typical IDE used for OpenCV. In Linux, its Eclipse or NetBeans, and in OSX, we can use Apple's Xcode.

3.4.3 OpenNI

Open Natural Interaction (OpenNI) is a non-for-profit organization created by Primesense, Willow-Garage, Side-Kick, Asus and Appside in 2010. It has the aim to certify the compatibility of NI devices such as the Kinect sensor. OpenNI API is an abstract layer that provides the interface to the physical devices and middle-ware components [24]. Figure 3.7 presents generic OpenNI architecture.

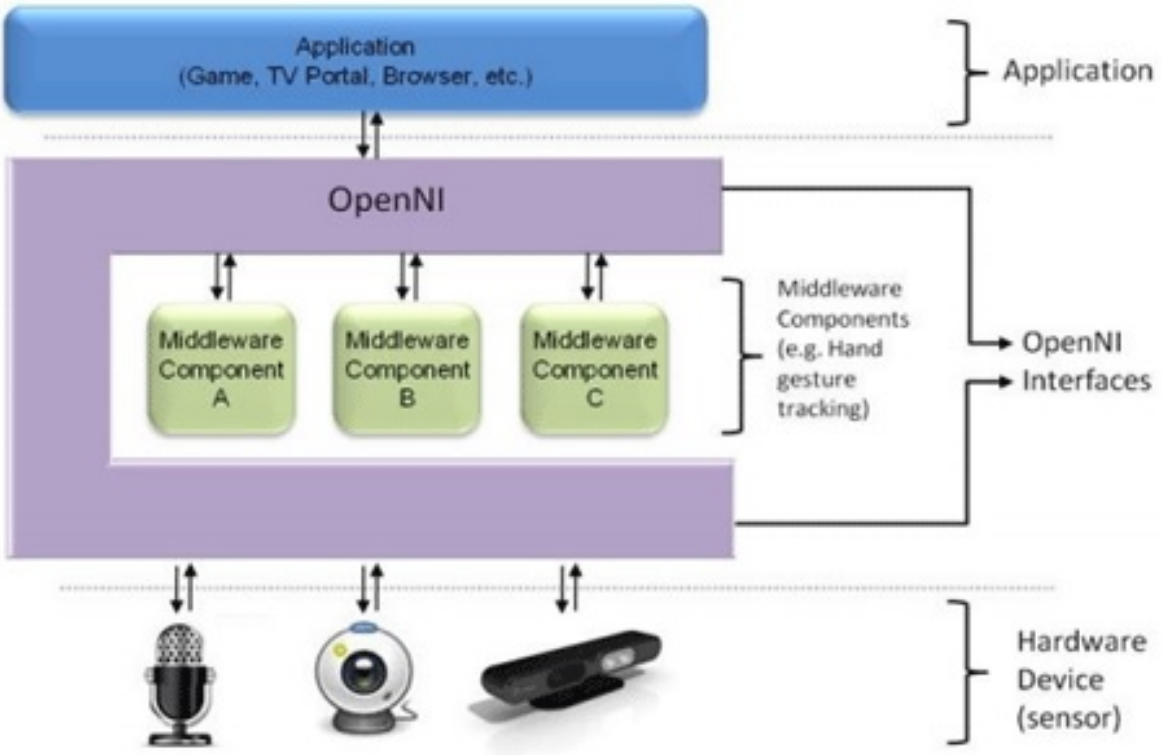


Figure 3.7: OpenNI architecture [24].

OpenNI provides compatibility for NI (natural interaction) devices such as the Kinect sensor and is now an integral part of ROS. The OpenNI community provides developers with a full range of software tools along with a vivid ecosystem platform for effective collaboration and promotion that address the complete development life-cycle of discovery, development and distribution. OpenNI can be installed and used very easily in a limited time in accordance with ROS.

OpenNI Tracker for Human Skeleton

OpenNI tracker allows one to track a person's skeleton using a depth sensor. It also gives the positions, relative to the camera frame, of the person's head, torso, hands, knees, elbows, etc. According to the documentation, the positions of the person's head, torso, arms, legs, etc. are supposed to be published as TF transforms obtained from That is, each body part will be considered its own coordinate frame, and accordingly openni_tracker publishes the transformation necessary to convert a body part coordinate frame to the camera's coordinate frame. Figure 3.8, presents a skeleton of human body obtained from depth sensor using openni_tracker.

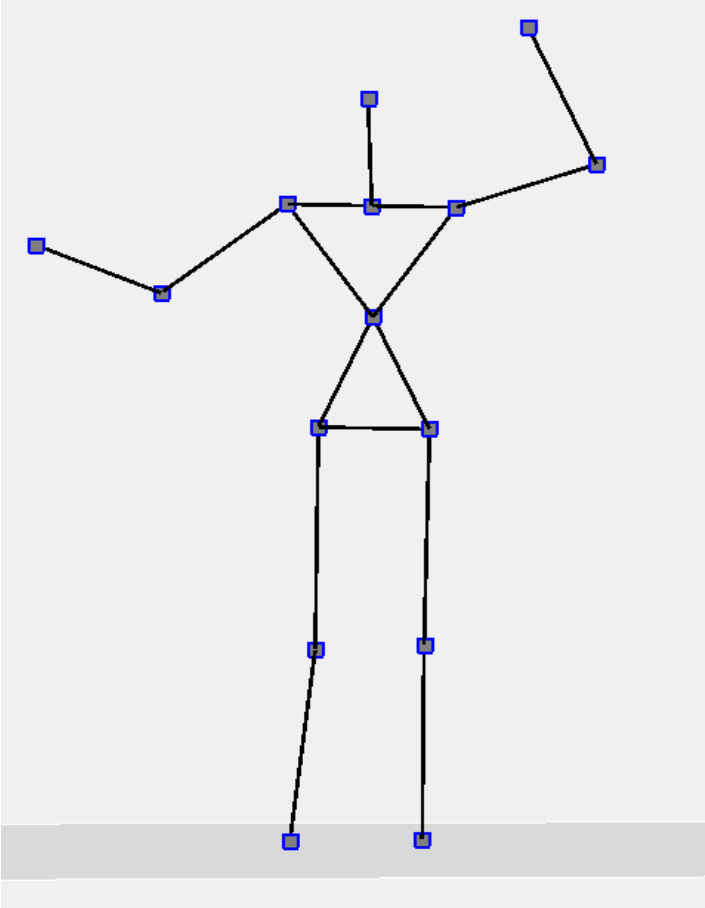


Figure 3.8: Human skeleton from OpenNI tracker.

When the user is detected, one can ask NITE to start looking for it's skeleton data (i.e. position of all joints of the person's body). The "calibration" (as is this process called) is fully automatic (no need to make any special poses), however the user needs to be standing (not sitting/lying) and the sensor should see the majority of his body. Usually, it is also better if the user is moving (*i.e.*, not standing still). Once the skeleton is detected, one can get 3D coordinates of all joints of user's body, with the exception of joints that are not visible (*i.e*, if a user's hand is behind his back).

CHAPTER 4

Vision System for Trajectory Estimation of a Flying Ball

This chapter describes the vision system algorithms developed for estimating the position of a flying ball in 3D space using a depth sensor (*i.e.*, RGB-D camera). One key component of the perceptual system is the capability to detect and recognize the target object - a ball - in a static and controlled environment. Two basic formulations are evaluated for comparison purposes: detecting the ball through a color-based algorithm and using the 3D point-cloud available from the Kinect. Visual tracking is another key problem in the development of the vision-based system. Once detected, the ball is tracked and localized in 3D, considering the case where the model of the tracked object is fully known (model-based tracking).

When the ball is in moving state, it becomes more difficult to track. Ball detection is much more easier than tracking, because detection occurs in static-state and tracking in dynamic-state. It is usual that the sensors give noisy measurements along with errors due to several internal and external factors. These factors affect ball tracking severely in-accordance with sensor imperfections. To address the problem of noisy and erroneous sensor data, Kalman Filter is to be used to estimate and correct measurements with each new sample of data. Since tracking the flying ball is a spatiotemporal (belonging to both space and time or to space-time.) process, Kalman filtering is used for improving trajectory estimation results and robustness. Trajectory estimation is a necessary and important part of this work. If and only if the ball detection and tracking is robust enough and works under various circumstances, trajectory estimation can be done subsequently. Next sections are dedicated towards study of trajectory of a flying ball, ball detection and tracking methods and finally trajectory estimation methods.

4.1 The Trajectory of a Flying Ball

The trajectory of a flying ball is modeled as a parabola (ballistic model), such that the only force acting upon it is the gravitational force that points downwards ignoring air resistance. It pulls the ball straight downwards the entire time when in the air, while there is no additional force to maintain the horizontal motion because of the object's inertia. The result is an accelerated motion in the vertical direction and a constant motion in the horizontal plane. Thus, one can look on the overall motion of the ball as an overlapping of the horizontal and the vertical motions, where neither influences the other. In order to analyze the motion of a ball and implement its mathematical description in the 2D space, one can use a coordinate system with the x-axis in the horizontal direction and the y-axis in the vertical direction. At any time t, the ball's horizontal and vertical positions are described by the expression 4.1, and 4.2 [26].

$$distance(m) = v_0 \cdot t \cdot \cos(\theta) \tag{4.1}$$

$$height(m) = v_0 \cdot t \cdot \sin(\theta) + \frac{1}{2} g \cdot t^2 \tag{4.2}$$

Figure 4.1, presents the basic 2 dimensional projectile motion, with same initial velocity (*i.e.*, v_0), for different values of inclination angle (*i.e.*, θ).

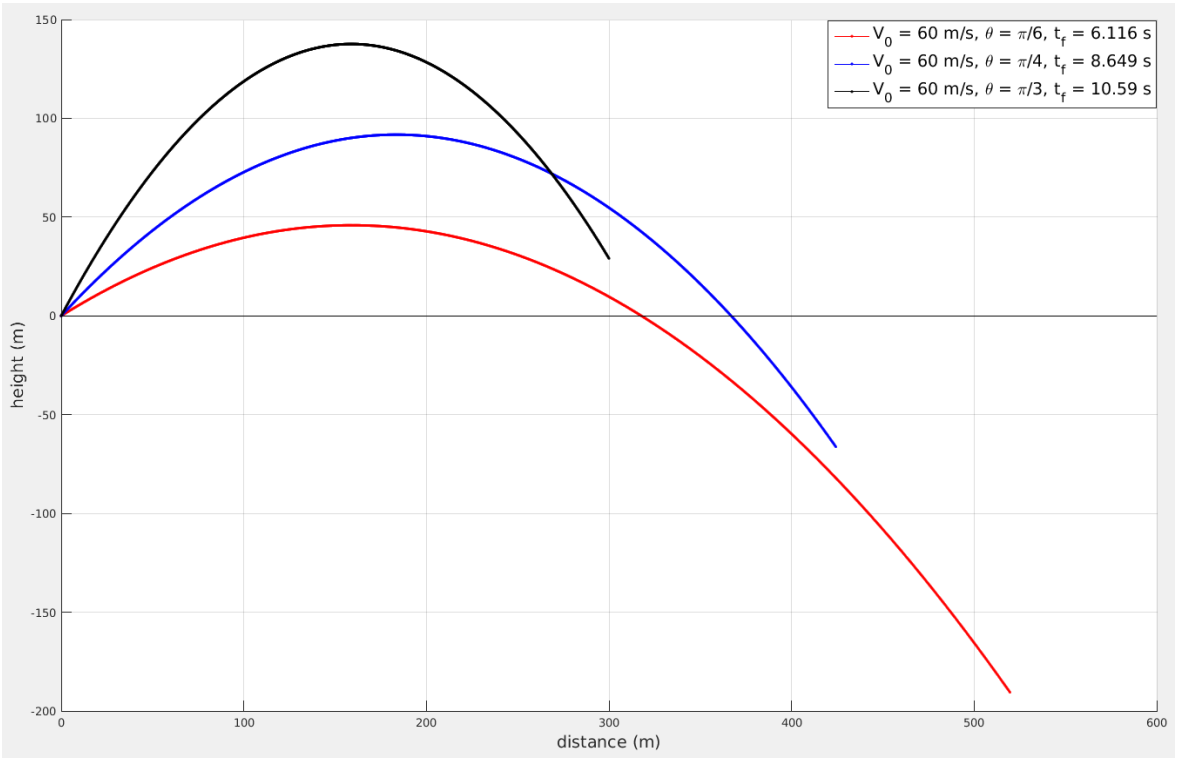


Figure 4.1: Basic 2 dimensional projectile motion, different values of inclination angle considered with same initial velocity.

Figure 4.2, presents the basic 2 dimensional projectile motion, with same inclination angle (*i.e.*, θ), for different values of initial velocity (*i.e.*, v_0).

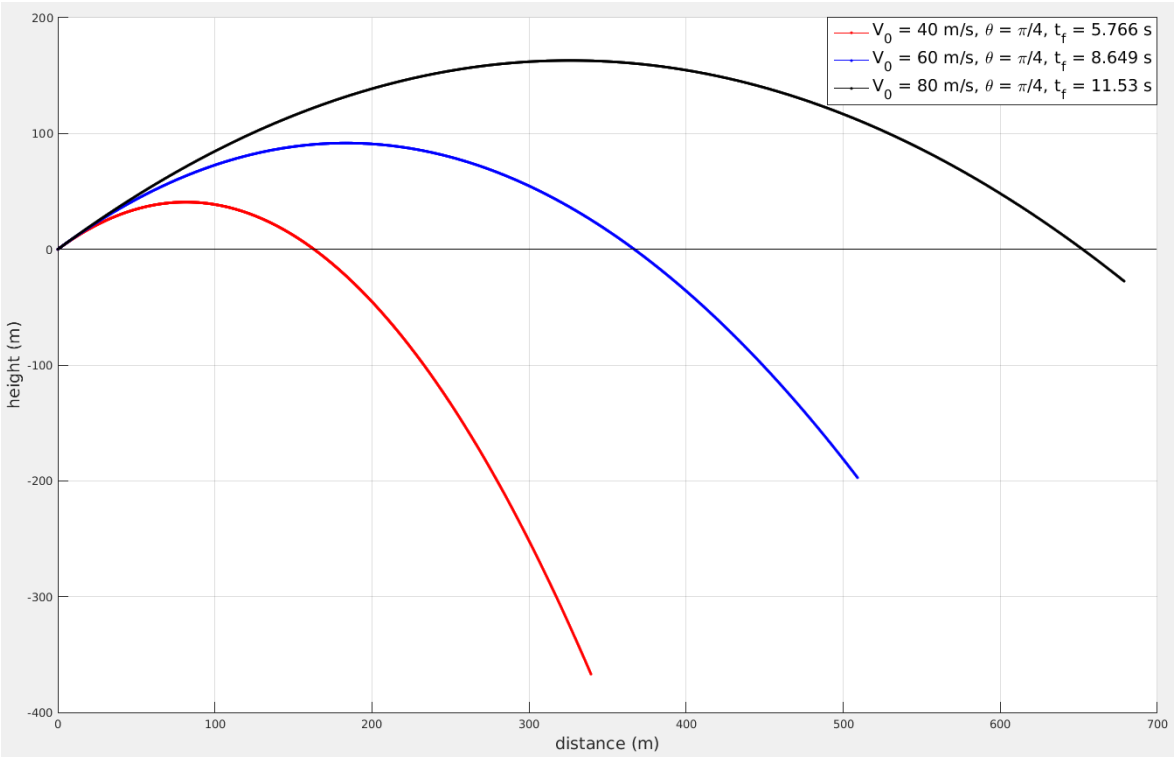


Figure 4.2: Basic 2 dimensional projectile motion, different values of initial velocity considered for same inclination angle.

From Figure 4.1 and 4.2, it is evident that the distance covered by projectile completely depends upon initial velocity v_0) and inclination angle (*i.e.*, θ). Where as height attained by projectile depends upon an additional factor g, the acceleration of gravity. Maximum height, maximum distance and time of flight are some of the important parameters, and are described below.

Time of Flight

The total time t, for which the projectile remains in the air is called the time of flight. Using equation 4.2, by setting height as zero, time of flight can be computed as given below.

$$t_f = \frac{2 \cdot v_0 \cdot \sin(\theta)}{g} \tag{4.3}$$

Maximum Height of the Projectile

Maximum height of the projectile is the greatest height that the object will reach. The increase in height will last until $v_y = 0$, that is, described by the following equations:

Time to reach the maximum height is half of the time of flight and is given by expression 4.4.

$$t_h = \frac{v_0 \cdot \sin(\theta)}{g} \tag{4.4}$$

Using equation 4.4 and 4.2, maximum height can be computed as given below.

$$h_{max} = \frac{v_0^2 \cdot \sin^2(\theta)}{2 \cdot g} \tag{4.5}$$

Maximum Distance of the Projectile

The horizontal range d of the projectile is the horizontal distance it has traveled when it returns to its initial height $y = 0$. Using equation 4.3 and 4.2, where height becomes zero maximum displacement can be expressed as 4.6.

$$d_{max} = v_0 \cdot t_f \cdot \cos(\theta) = \frac{v_0^2 \cdot \sin(2 \cdot \theta)}{g} \tag{4.6}$$

4.2 Ball Detection and Tracking

Object detection, segmentation, tracking, and estimation are some of the most important and challenging fundamental task in the field of computer vision. These are widely used in machine vision sector for inspection, registration and manipulation of different kind of tasks. In many cases, robotic arms need some kind of mechanism to recognize objects to act on them, in an autonomous way. But the algorithms for object recognition or detection and tracking have many limitations because of the changes in illumination, occlusion, scales, background and dynamics.

Ball detection and tracking in dynamic state is even more challenging considering sensor imperfection, environmental factors and computational capability. Two different methods of ball detection and tracking are used for this dissertation work. First method uses OpenCV to handle color processing in the images to detect the ball. Second method of detection and tracking is without color, *i.e.* using point-cloud data of work-space acquired from depth sensor. Both of the above stated methods were implemented and a comparative analysis was performed to evaluate their limitations.

4.2.1 Color Based Ball Detection and Tracking

One of the easiest way to detect and segment an object in 3D space from an image is the color based method. Object detection is a critical part in many applications such as image search, scene understanding, etc.. However, it is still an open problem due to the variety and complexity of object classes and backgrounds. The object and the background should have a significant color difference in order to successfully

segment objects using color based method. Hue, Saturation, Value or HSV is a color model that describes colors (*i.e.*, hue or tint) in terms of their shade (*i.e.*, saturation or amount of gray) and their brightness (*i.e.*, value or luminance). RGB format is the way computers treats color, and The HSV color space is quite similar to the way in which humans perceive color.

In a first step, color image is converted to HSV color-space using OpenCV functionality. This HSV image can be thresholded to extract the desired object. Second, is to get the center of detected object in a color image and third, is to get 3D coordinate of the detected object in 3D scene.

Object Detection Concept

To detect a desired object, a color image frame of video is taken and converted to HSV color-space. This can be done using track-bars initially and then with static limits. In OpenCV, detecting objects is like finding white object from black background. So, it must be taken into consideration that the object to be found is white and background should be black. After thresholding (*i.e.*, partitioning an image into a foreground and background) the image, some small white isolated objects can be found in the image. It may be because of the noises in the image or the actual small objects which have the same color as the main object. These unnecessary small white patches can be eliminated by applying morphological opening. Morphological opening can be achieved by a erosion, followed by the dilation with the same structuring element. In addition to that, thresholded image may also have small white holes in the desired object, may be because of the noises in the image. These unnecessary small holes in the main object can be eliminated by applying morphological closing. Morphological closing can be achieved by a dilation, followed by the erosion with the same structuring element.

In this application, Moments' method is used to calculate the position of the center of the detected object. In addition to that, noise of the binary image should be at minimum level to get accurate result.

Computation of the Center of a Detected Object

Moment is a quantitative measure (*i.e.*, descriptor), popularly used in mechanics and statistics, to describe the spatial distribution of set of points. In most simplistic terms, moments are set of scalars that provide an aggregated measure of a set of vectors. The definition of moments is the same across domains of mechanics, statistics, and computer vision. The concept of moment in statistics and mechanics has been borrowed in computer vision to coarsely characterize the shape of an object in an image. These moments capture basic statistical properties of the shape, including the area of the object, the centroid (*i.e.*, the center (x, and y) coordinates of the object), orientation, along with other desirable properties. For this particular work centroid (*i.e.*, center of the detected object), is the only desired parameter and this information is extracted using OpenCV functions. The spatial moments of of detected

object m_{ji} are computed using expression 4.7.

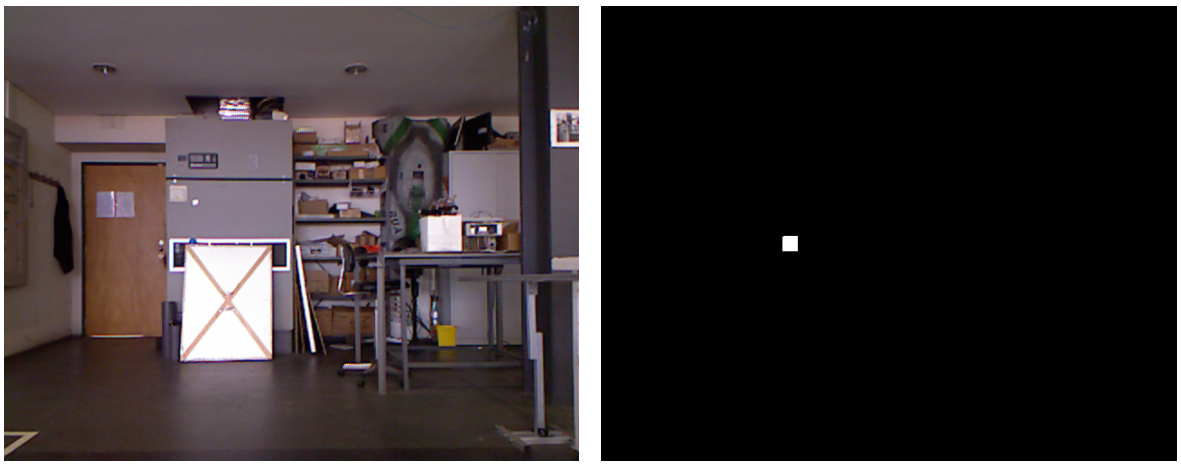
$$m_{ji} = \sum_{x,y} (array(x,y) \cdot x^j \cdot y^i) \tag{4.7}$$

Where, i,j are the order of moments.

Where, $(\bar{x}, \bar{y})$ is the center of the object and are calculated using the expression 4.8 given below.

$$\bar{x} = \frac{m_{10}}{m_{00}}$$
$$and$$
$$\bar{y} = \frac{m_{01}}{m_{00}} \tag{4.8}$$

Where, m_{10} is the 1st order spatial moment around x-axis, m_{00} is the 0th order central moment, and m_{11} is the 1st order spatial moment around y-axis. Figure 4.3a and 4.3b, presents a color image of 3D scene and detected object in color image. In this case a blue ball is considered as the desired object in the 3D scene. Figure 4.3a is the color image of 3D scene and Figure 4.3b is the thresholded image with morphological operations.



(a) RGB image with a blue ball.

(b) Thresholded image with detected ball.

Figure 4.3: Color based ball detection.

Implemented Algorithm for Ball Detection and Tracking

The ball detection and tracking algorithm is implemented in ROS using depth sensor, OpenNI, and OpenCV. Image format in ROS is different from OpenCV. It is necessary to convert the acquired image (*i.e.*, image in ROS format retrieved with the help of OpenNI), to OpenCV format. ROS passes around images in its own sensor_msgs/Image message format, but this format is not compatible in OpenCV and needs to be converted, to use images in conjunction with OpenCV.

CvBridge is a ROS library that provides an interface between ROS and OpenCV as shown in Figure 4.4. CvBridge defines a CvImage type containing an OpenCV image, its encoding and a ROS header. CvImage contains exactly the same information that sensor_msgs/Image does, so conversion is possible from one to the other for the purpose of representation and processing [27]. This image in OpenCV format, can be processed to detect object and subsequently center of the detected object.

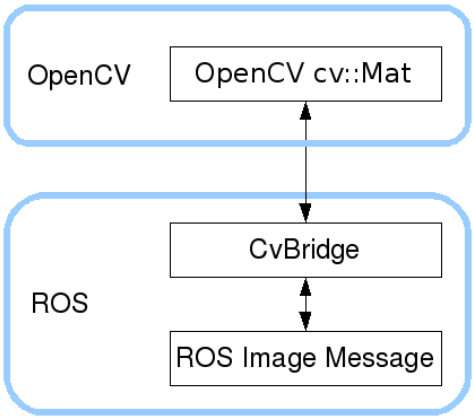


Figure 4.4: Converting ROS image messages to OpenCV images [27].

It should be noted that the center of the detected object is in pixels and not in metric system. To get ball position in 3D space it is necessary to have more information about the work-space. More information about the work-space can be found in point-cloud data (*i.e*, produced by depth sensor). After the conclusion of initial part of center extraction, 3D position of ball can be found using point-cloud data. A point-cloud is a set of data points in some coordinate system. In a three-dimensional coordinate system, these points are usually defined by X, Y, and Z coordinates, and often are intended to represent the external surface of an object. For this to happen two different topics(*i.e.*, RGB image topic, and point-cloud topic), should be subscribed with synchronization in time as shown in Figure 4.5.

Synchronization is needed to get ball position at a particular time instant from these topics. Subscribed point-cloud has 3D coordinates of all the valid points in work-space. These data can be accessed in real-time to get 3D coordinate of a particular pixel-index. The relation between 2D pixel index and 3D points are established using camera calibration parameters [28]. This information can be used to get 3D coordinate of detected objects center. Figure 4.5 presents implemented algorithm for this method of ball detection and tracking. Implemented algorithm can be divided in to four sub steps and these constitute together to form the ball detection algorithm color image and point-cloud given by depth sensor. These four steps are 1) image acquisition and conversion to OpenCV format, 2) conversion of RGB image to HSV format and then thresholding it, 3) application of morphological operations (*i.e*, dilation and erosion), and object detection using area as a parameter, 4) getting 3D coordinate of the detected object.

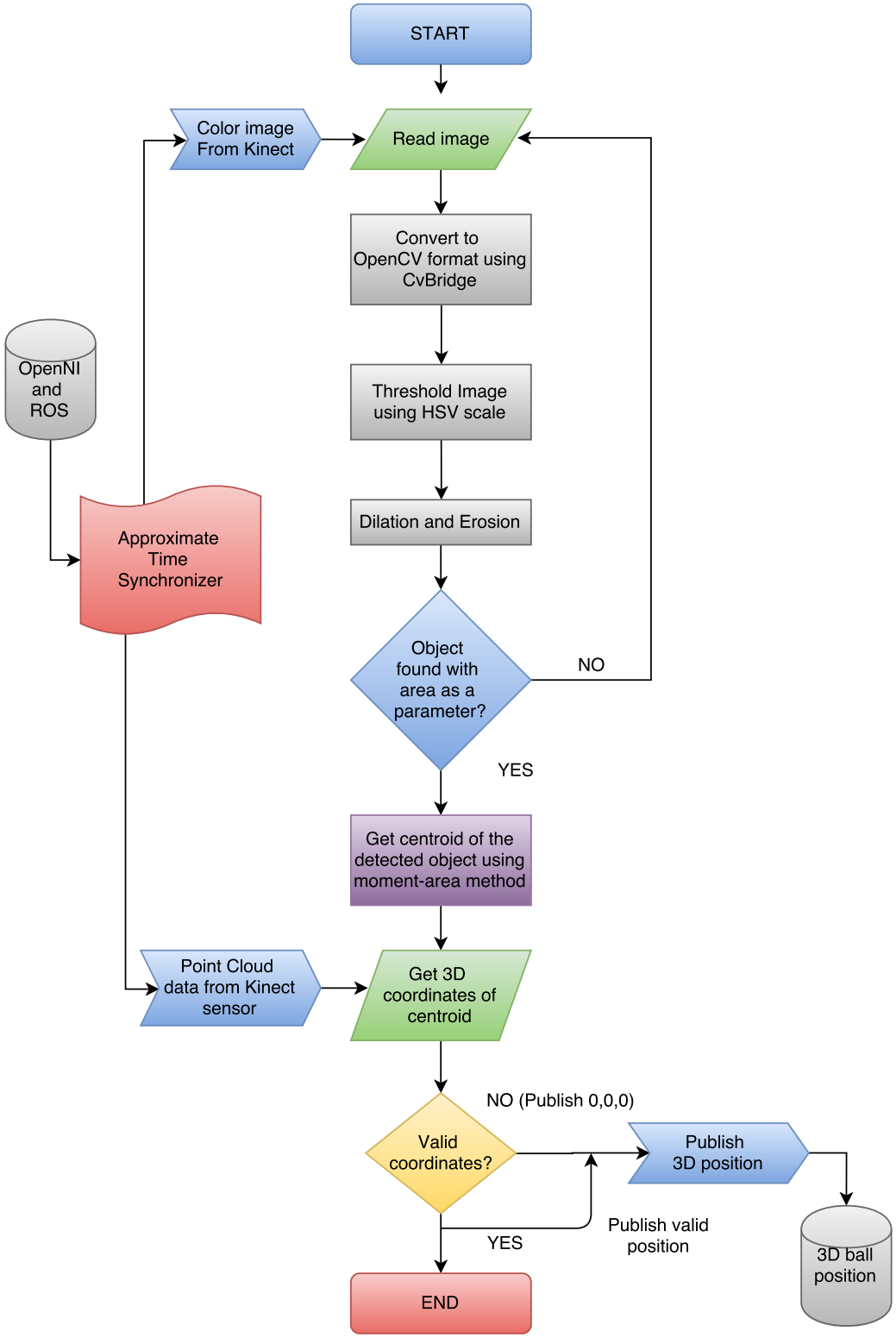


Figure 4.5: Color based ball detection flow chart.

As shown in Figure 4.5, First, the RGB image is acquired using OpenNI (*i.e*, acts as a middle-ware) and ROS functionality (*i.e*, subscribing the RGB image topic) from

depth sensor. This image is converted to OpenCV format using CvBridge [27]. Second, this OpenCV format image is converted to HSV format using OpenCV functionality and subsequently thresholded (*i.e.*, partitioning an image into a foreground and background) for object detection. Third, the thresholded image is processed with morphological operations (*i.e*, dilation and erosion) to remove noise and unwanted objects. Once noises and unwanted objects are removed, the image can be filtered using area as a parameter to get the desired object in thresholded image. Center of the detected object can be found using Moments' method. Fourth and final step is to get 3D coordinate from point-cloud data for the center of detected object. To get RGB image and point-cloud data at the same time instant, approximate time synchronization mode is used. In addition to these steps, validity of retrieved 3D coordinates are tested using maximum possible distance of ball between past and present position, and by not considering the border pixels for detection and tracking.

Color Based Ball Detection Overall Structure

Color based ball detection and tracking method is implemented in ROS platform. Figure 4.6 presents overall structure for color based ball detection and tracking method. The topics '/camera/rgb/image_color' and '/camera/depth/points' are coming from depth sensor through OpenNI. These two topics are subscribed by '/vision_node' for further processing. The topic '/camera/rgb/image_color' is responsible for RGB image in ROS format and later converted using CvBridge to get image in OpenCV format inside '/vision_node'. This converted OpenCV image is used to detect ball and subsequently centroid pixel index is calculated. The topic '/camera/depth/points' is used to get 3D coordinate of detected ball in depth camera-space. Other topics of interest are '/ballCord' and '/ball3D', these are published by '/vision_node' for further use. The topic '/ballCord' will be subscribed by '/estimation_node' to estimate ball's landing point over a plane of fixed height. The last topic '/ball3D' is meant to be used by '/visualization_node' for the purpose of visualization in rviz in real-time. It must be noted that these two topics(*i.e.*, '/camera/rgb/image_color', and '/camera/depth/points') are subscribed using approximate time-synchronization, so as to get information correctly as shown in Figure 4.5.

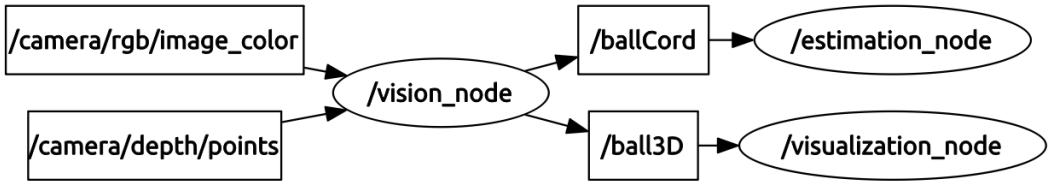


Figure 4.6: ROS graph for color based ball detection methods.

4.2.2 Point-cloud Based Ball Detection and Tracking

As stated earlier that, a point-cloud is a set of data points in some coordinate system. In a three-dimensional coordinate system, these points are usually defined by X, Y, and Z coordinates, and often are intended to represent the external surface of an object.

"The main idea behind this method is to detect any cloud of points isolated, provided that there are points in cloud around it, but at a distance more than the specified threshold limit."

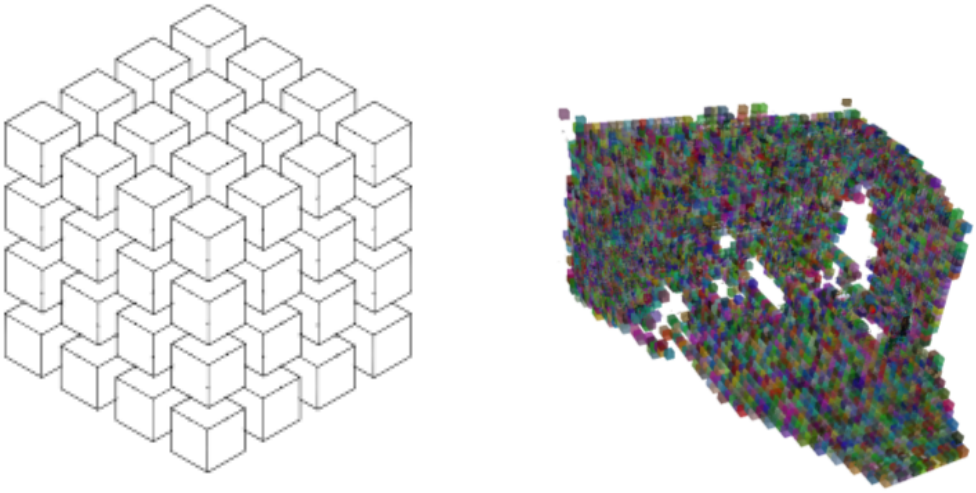
This means that the point-cloud based ball detection and tracking works only if a set of points in point-cloud found to be isolated (*i.e.*, if there are some points inside point-cloud), so that these points are completely isolated from other rest portion of point-cloud given a threshold parameter to check if they are separated or not. This isolated set of points can be considered as a flying object inside the specified volume of point-cloud. Other strategies can also be adapted to modify this method of detection such as, considering only the upper half of isolated cloud to detect object, this works only if the ball is on the ground.

This is a difficult problem to tackle given the dynamics and speed of a flying object. The problem is even more difficult when the vision system is not so good (*i.e.*, point-cloud is not stable or infrared sensor is not good enough to get complete information about working environment). But, in this case color of the object is not as issue and hence synchronization problem stands void. To achieve this, given the properties of a flying ball in the work-space, voxelization of the work-space is done in an occupancy voxel space rather than considering the whole cloud of points [29].

Point-cloud Grid Voxelization Concept

A 'voxel' is a tiny box of certain dimension, 'voxel-set' is a set of voxels and can be termed as 'mask' for this work. A 'voxel-grid' is a set of boxes in space, this can contain several 'voxel-sets'. The method of voxelization is dividing the whole work-space in to tiny boxes to get a voxelized grid. Figure 4.7a shows the basic concept of grid voxelization. In the scope of this work, to maximize the algorithm execution speed, a specified volume is considered. This volume is the place where detection and tracking experimentation will be carried out. As volume (*i.e.*, grid size) for this task is predefined, voxelization is possible with the definition of ball radius, and mask-size. For example, If a mask is considered to be of $5 \times 5 \times 5$ 3D matrix to be examined, then in all total a mask (*i.e.*, voxel-set) contains 125 voxels, and it is assumed that the ball is in the central voxel of the mask. Figure 4.7b shows a voxelized grid. The basic concept is to find voxel that contains cloud of points and is surrounded by empty voxels (*i.e.*, no cloud of point inside voxel). Number of voxels in the grid of specific volume fully depends upon ball radius, larger the ball radius less will be number of boxes (*i.e.*, voxels).

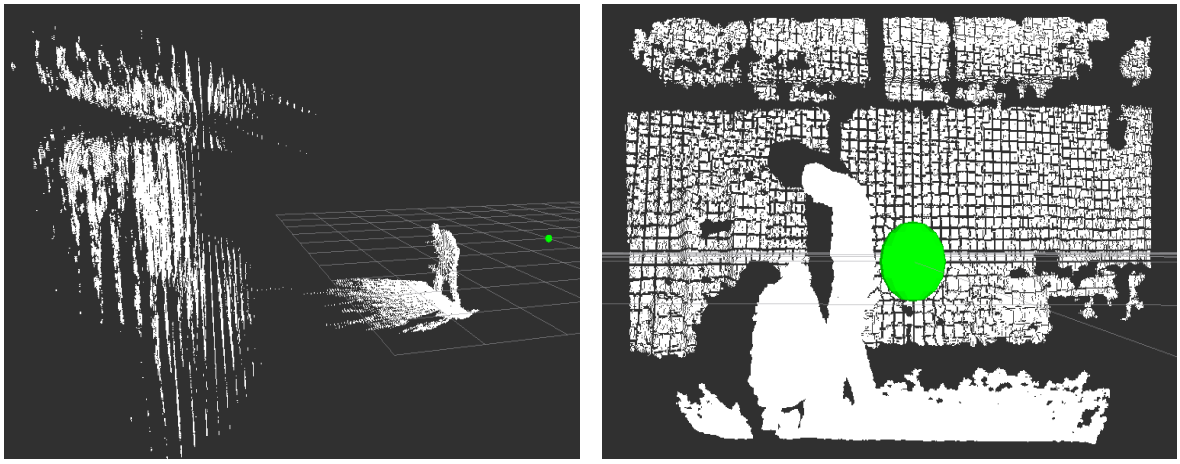
A side-way view of point-cloud from depth sensor is as presented in Figure 4.8a. It is also evident from figure 4.8a that the separation between planes increases as



(a) Voxels in a grid. (b) Voxel grid.

Figure 4.7: Voxelized grid [29].

the depth from camera increases. Figure 4.8b shows the point-cloud organization, where the black portions are points without measurement. These points without measurement can be termed as something of which the IR camera has no information or these point are too far from the depth sensor. This may be at a distance out of range of depth sensor or are shadows of any object.



(a) Perspective projection of Kinect IR camera. (b) Depth sensor point-cloud organization.

Figure 4.8: Kinect point-cloud structure.

Implemented Voxel Grid Algorithm

The values used for the grid and the mask obviously depend on the size of the ball to be detected. However, they have to be defined taking into consideration two main aspects: 1) the grid size must be large enough to allow a real flying ball, when voxelized, does not become smaller than the space between any two planes. This issue becomes more hazardous at farther distances; 2) the grid mask must be large enough to accommodate a volume larger than the ball, since some blurring is inevitable due to the high speeds achieved by a ball. With this mask approach, it is expected to rule out false positives from any other object inside the work-space. Figure 4.9 illustrates a detected ball in point-cloud.

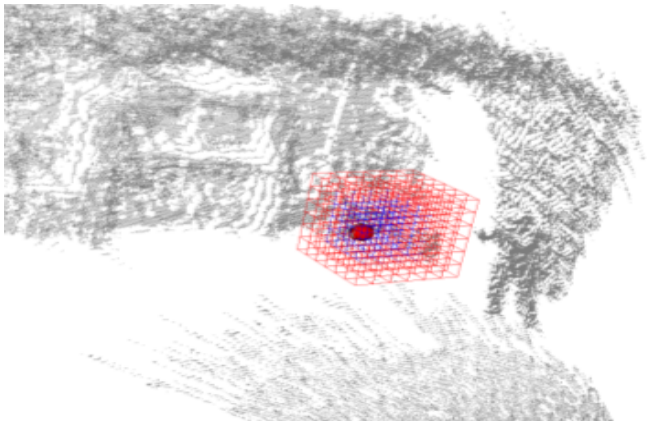
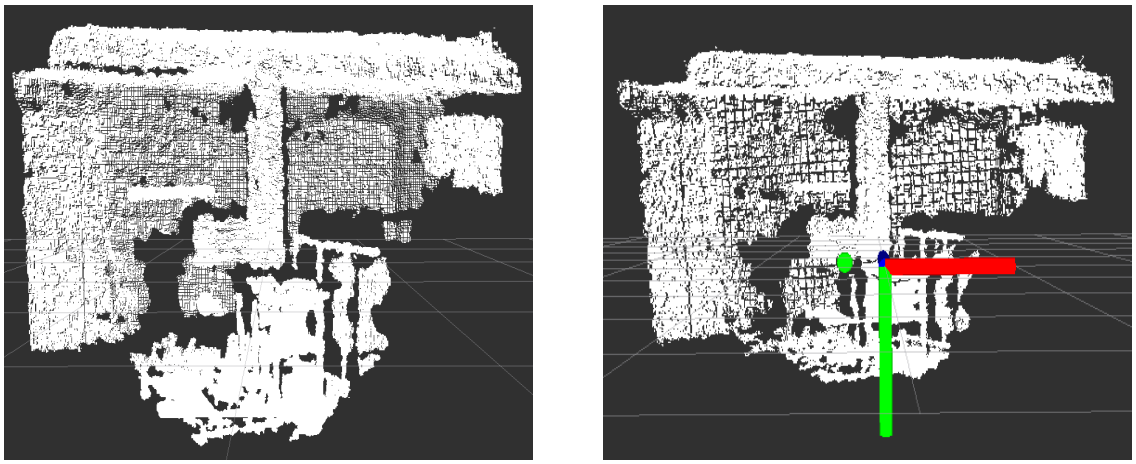


Figure 4.9: Detected ball inside a grid [29].



(a) Point-cloud from depth sensor.

(b) Ball detected in cloud of points from depth sensor.

Figure 4.10: Ball detection in point-cloud using grid voxelization method.

Figure 4.10a shows the generated point-cloud from the depth sensor and Figure 4.10b shows a detected ball exactly at the place where the ball is physically sus-

pended. This method of ball detection and tracking works in a highly non-uniform environment, because it considers a cloud of points and not single point. Even if the information provided by depth sensor has error, this method works fairly well. Figure 4.11 represents the overall flow-chart for point-cloud based ball detection and tracking.

Below is the pseudo code for implementation of this algorithm [29].

for (Each point-cloud) **do**
Voxelization of grid
Detection of flying ball in specified volume (use flying object mask)
end for

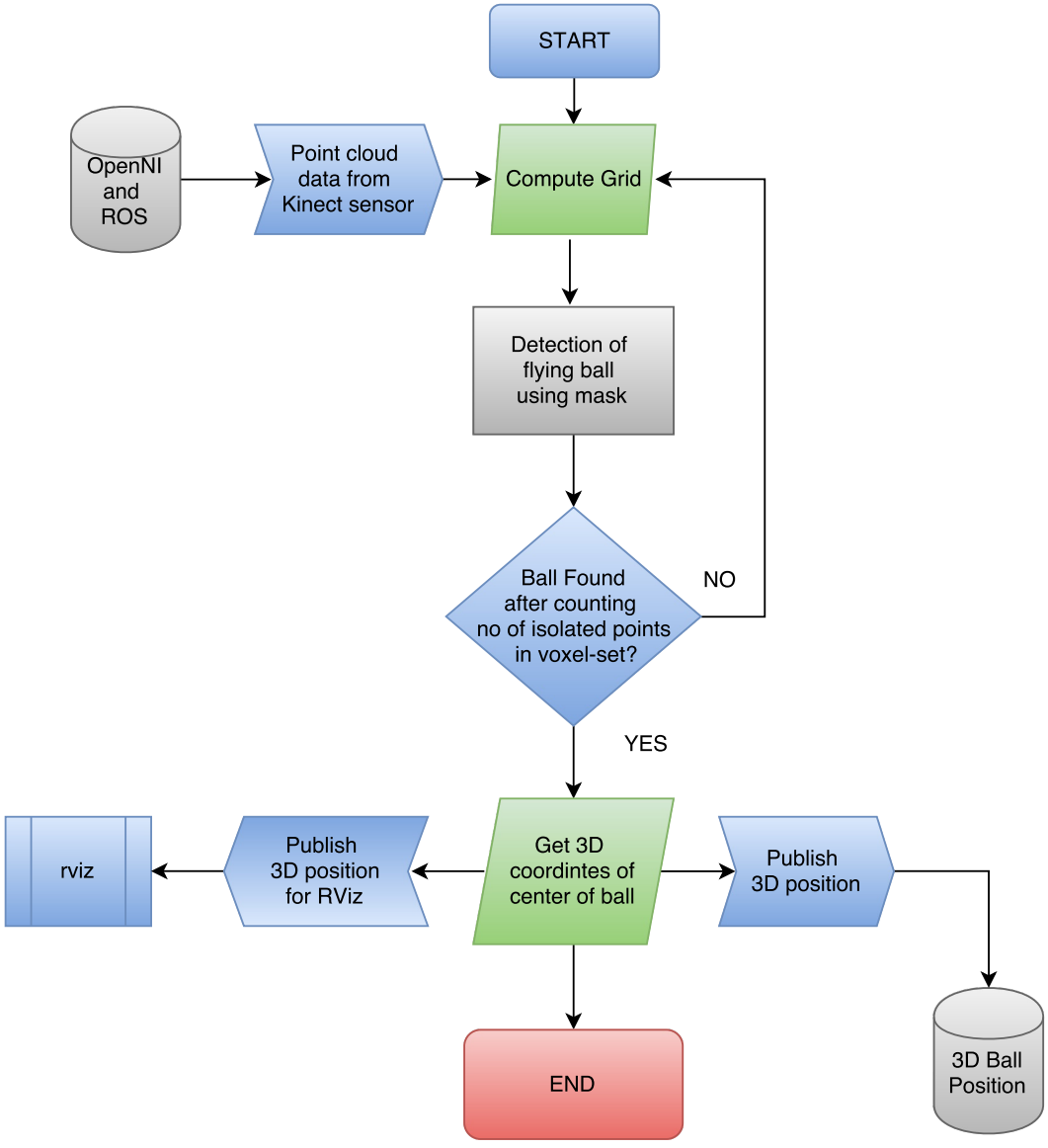


Figure 4.11: Point-cloud based ball detection flow chart.

Implemented algorithm can be divided in to three sub steps and these constitute together to form the ball detection algorithm using only point-cloud data given by depth sensor. These three steps are 1) point-cloud data acquisition and grid computation, 2) detection of flying ball using mask, 3) computation of the 3D coordinate of the detected ball and publish it for further processing.

As shown in Figure 4.11, First, the point-cloud data is acquired using OpenNI (*i.e*, acts as a middle-ware) and ROS functionality (*i.e*, subscribing the point-cloud topic) from depth sensor. This point-cloud data is used to computed grid (*i.e*, voxelization of the work space) according to ball size and works-pace. Second, this grid is searched for flying ball using mask ((*i.e*, a 3D structuring element)). Third, if the flying ball is found, 3D coordinates of it are computed and published for further processing. In this case there is no tool available for the purpose of debugging, and hence rviz is used as a visualization tool for confirmation of ball detection in real-time. It is worth of mention that, the point-cloud for the detected flying ball is on the surface and it adds an error. This error can be neglected using a compensation factor. More details about experimental analysis on ball detection methods will be discussed in Chapter 6.

Point-cloud Based Ball Detection Overall Structure

Point-cloud based ball detection and tracking method is implemented in ROS platform. Figure 4.12 presents overall structure for point-cloud based ball detection and tracking method. The topic '/camera/depth/points' is used to get cloud of points. At a later stage this point point-cloud is voxelized knowing the volume. Voxelized grid is passed through a testing phase using mask (*i.e.*, voxels set) and if any isolated set of points found then center of these points is calculated. Other topics of interest are '/ballCord' and '/ball3D', these are published by '/vision_node' for further use. The topic '/ballCord' will be subscribed by '/estimation_node' to estimate ball's landing point over a certain plane. The last but not the least topic '/ball3D' is meant to be used by '/visualization_node' for the purpose of visualization in rviz in real-time.

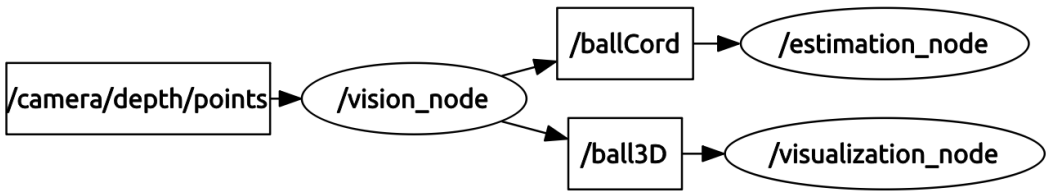
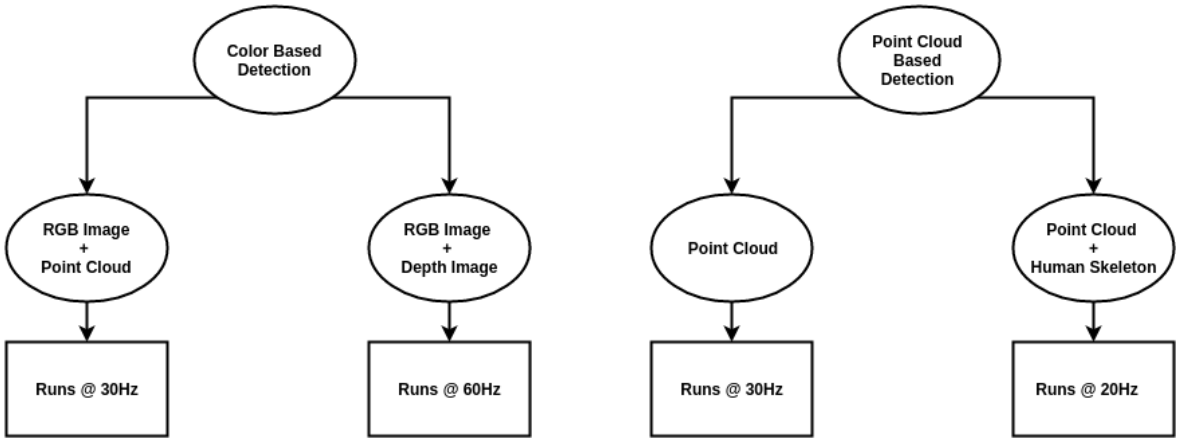


Figure 4.12: Point-cloud based ball detection overall structure.

4.2.3 Comparative Analysis of Ball Detection Methods

Two methods of ball detection and tracking were implemented in ROS using depth sensor. At this stage, it is important to compare these two methods, considering

robustness and other operational characteristics. Figure 4.13a depicts the basic diversity of color based ball detection and tracking method. Where as Figure 4.13b depicts point-cloud based ball detection and tracking methods.



(a) Color based ball detection methods. (b) Point-cloud based ball detection methods.

Figure 4.13: Comparison of implemented ball detection algorithms.

The color based method of detection is divided into two sub-methods, 1) using RGB image and point-cloud, 2) using RGB and depth image. Figure 4.13a presents the basic difference between color based detection methods. The rate of RGB image and point-cloud method is 30 Hz (*i.e*, algorithm execution speed), because of the point-cloud data. It makes the process slower, because huge amount of data is handled simultaneously to get point-cloud. The other method (*i.e*, RGB image and depth image), runs faster in terms of execution speed. The execution speed is faster here because of less data handling at a time.

Again, the point-cloud based method of detection is divided into two sub-methods, 1) using only point-cloud, 2) using point-cloud and human skeleton from openni_tracker. Figure 4.13b presents the basic difference between point-cloud based detection methods. The rate of only point-cloud based method of detection runs at 30 Hz (*i.e*, algorithm execution speed) after optimization. The other method (*i.e*, RGB image and depth image), runs slower in terms of execution speed (*i.e*, runs at approximately 20 Hz). The execution speed is slower in second case is because of skeleton data.

Color based detection: This method of detection uses both RGB image and point-cloud for detecting object. In place of point-cloud, depth can also be used to get 3D coordinate of detected ball. Below are presented some of the basic observations related to color based detection method.

- Algorithm execution speed doesn't depend upon size of the ball
- Error in 3D coordinates $\propto$ Ball distance from depth sensor

Point-cloud based detection: This method of detection uses only point-cloud for detecting object. At a later stage human skeleton information is added to make

it more robust and reliable, but this method makes the process slower. Point-cloud with human skeleton method is more robust because, even if the ball is not in flying state, it's coordinates can be computed using human hand joint position from skeleton data. With this information about ball position is available before the ball starts flying. Below are presented some of the basic observations related to point-cloud based detection method.

- Algorithm execution speed $\propto$ Size of the ball
- Number of points to detect a flying ball $\propto$ Size of the ball
- Error in 3D coordinates $\propto$ Ball distance from depth sensor
- Algorithm execution speed $\propto$ Grid size

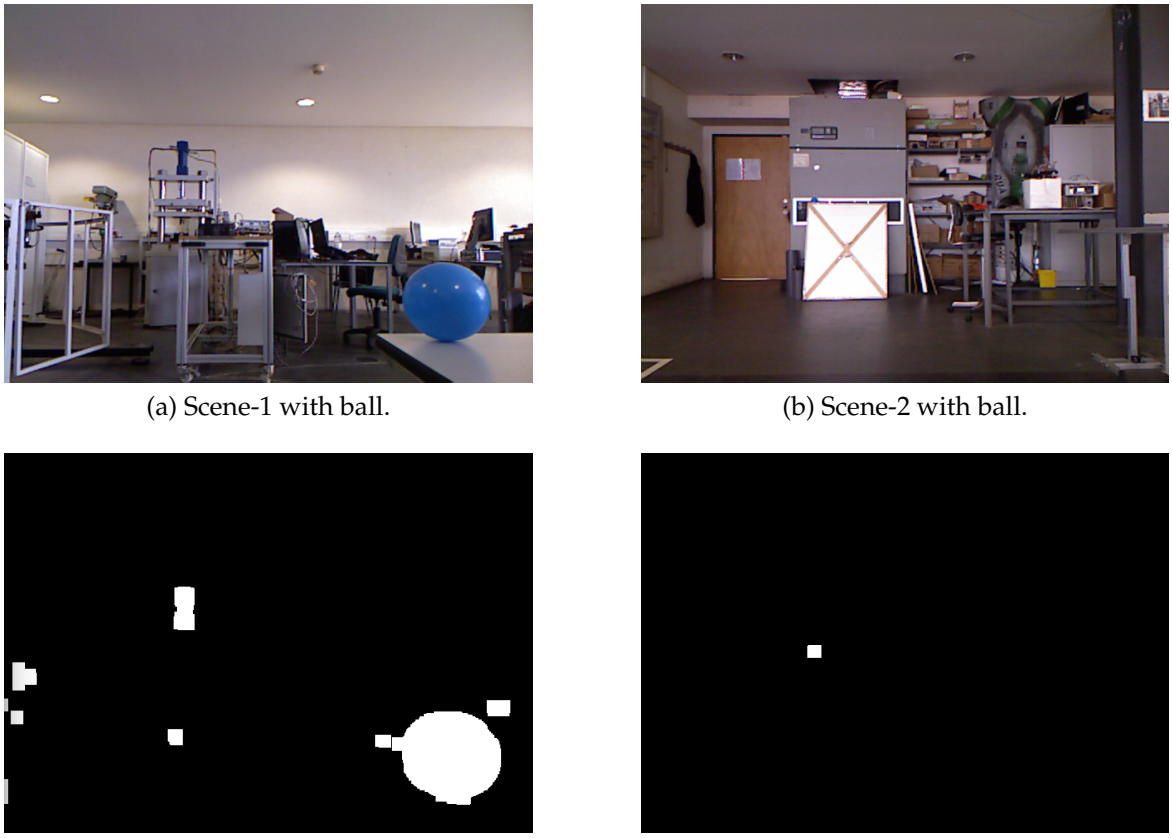
Table 4.1 presents some of the observed characteristics from the previously mentioned and implemented methods of ball detection and tracking. From the above mentioned comparative analysis it is evident that point-cloud based ball detection and tracking is robust and fast enough (*i.e*, runs at approximately 30 Hz with fairly better result in terms of ball detection and tracking) to be used for trajectory estimation and ball-catching application. Where as from Figure 4.13b it is evident that the point-cloud and human skeleton method is slower, but it gives an upper hand to the other one if speed of execution is neglected as a constraint while comparing both the algorithms. In addition to that, another advantage of this point-cloud based method is that, this method works fairly well in terms of ball detection and tracking compared to color based ball detection and tracking.

Table 4.1: Comparison of ball detection methods.

Color based ball detection and tracking	**Point-cloud based ball detection and tracking**
Runs at 30 Hz	Runs at 30 Hz
Easy to implement	not easy to implement
Performance is affected by light intensity and background	Performance is not affected by light intensity and background
can be used for balls of different size without changing the algorithm	can not be used for balls of different size without changing the algorithm
Algorithm unable to find valid coordinates of ball sometimes and gives stray positions as ball position	Algorithm finds valid coordinates of ball often
Algorithm execution speed doesn't change with size of ball	Algorithm execution speed changes drastically with size of ball

Experimental Analysis of Detection and Tracking Algorithms

From Figure 4.13a, it is evident that the RGB image and depth image method of ball detection is faster but it's not worth of concern because RGB and depth image topics are being published at a rate of 30 Hz. In addition to that, if robustness of this color based detection method is considered, it gives mixed result. Mixed result in the sense that the 3D coordinates are not valid (*i.e.*, wrong position ball) when the ball is in dynamic state. In addition that, this method is affected by external factors such as background and environment light intensity. Figure 4.14 presents a basic representation of how external factors influence color based ball detection process. In Figure 4.14a and 4.14b are the RGB images that are used to detect a blue colored ball and 4.14c and 4.14d show thresholded image, and how it worked in one case and failed in another.



(a) Scene-1 with ball.

(b) Scene-2 with ball.

(c) Ball not detected.

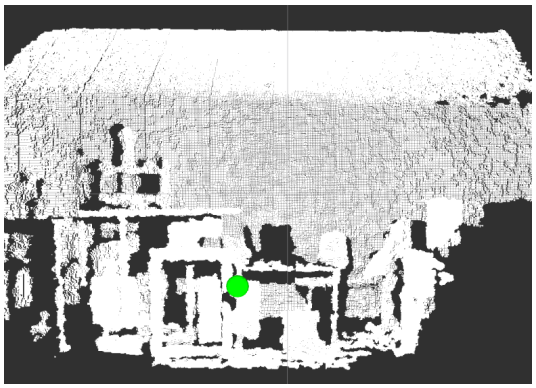
(d) Ball detected.

Figure 4.14: Color based ball detection comparison.

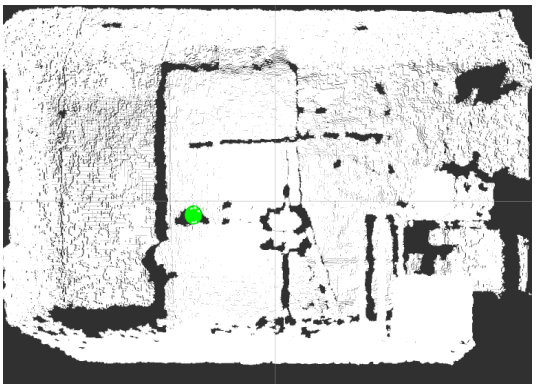
Figure 4.15 show that the 2nd algorithm (*i.e*, point-cloud based ball detection method) is less sensitive to external factors in ball detection process. In Figure 4.15a and 4.15b upper images are the RGB images are shown only to illustrate the set up (*i.e*, not used for detection) and 4.15c and 4.15d show detected ball in both the cases. These images show that the point-cloud based ball detection algorithm is less sensitive to changes in background and illumination.

(a) Scene-1 with ball.

(b) Scene-2 with ball.



(c) Detected ball in point-cloud Scene -1.



(d) Detected ball in point-cloud Scene -2.

Figure 4.15: Point-cloud based ball detection comparison.

Conclusion on Detection Algorithms

From Figure 4.14 and Figure 4.15, it is evident that the point-cloud based detection method is better than color based detection method. Where as, color based detection and tracking is easier to understand, implement and works fairly well for static objects but it is not suitable for objects in motion. Color based detection is highly affected by background and environment light intensity. On other hand point-cloud based detection and tracking has it's own limitations. This method of detection works well if the background is uniform. If the background is not uniform then this method fails sometimes because it detects something else considering that there exists some isolated points. In addition to that, execution speed of this algorithm depends upon size of ball, grid size and distance from sensor. But, point-cloud based method of ball detection is robust enough to be used for ball-catching application. It should be noted that Kinect has an error of 2.5 cm per meter and if we consider approximation and detection algorithm errors we end up with more errors. This accumulated error can extend up to 5-7%.

4.3 Trajectory Estimation Methods

Estimation of the possible pose (position and orientation) of moving targets has great significance in terms of defense industry and in the sports arena. If a shoot aiming at the target is planned, then the estimation problem is not that difficult. However, if the shoot or throw is not a planned one, then the issue of estimation becomes difficult task because of several reasons, such as air-drag, angle of throw, initial velocity etc. There exist several estimation methods that can be used as per requirement, including curve fitting methods [30] and state estimation methods (*e.g.*, Kalman filtering [31]).

Catching a flying ball implies to predict its trajectory ahead of time so that to determine the intersection point along this same trajectory. In this work, it is assumed a known model of the dynamics of the motion by modelling the trajectory of a flying ball as a parabola (ballistic model). In addition to that, this study is tuned for spherical objects by estimating only the position of the ball. Whereas, other possible estimation parameters can be velocity and acceleration along with position. On the one hand, polynomial approximation and estimation of the parameters can be done recursively through least squares optimization. In addition to that, to enable real-time tracking, the model of the ball dynamics is used in conjunction with a Kalman filter [32] [33]. for online re-estimation of the trajectory and robustness against noise sensing. Both methods were implemented and then a comparative analysis to find what they had to offer and of what they lacked of.

4.3.1 Polynomial Approximation Method

The least-squares (LS) method is one of the best-known approaches in solving a problem of finding the best polynomial approximation to the input samples [34]. It was originally developed for statistical regression, but nowadays a general concept of LS approximation is widely used for various applications beyond the statistics. For this dissertation work, this method of approximation will be used for estimation of ball position from few past samples.

Linear Regression for St-Line Fitting

Least Squares method of regression for straight line fitting is the simplest one. This method of fitting is simple not only because it has less number of parameters to be estimated, but also it requires less number of points for estimation compared to other fitting methods. Linear regression is a method to best fit a linear equation (straight line) of the form $y(x) = ax + b$ to a collection of points. The algorithm basically requires minimization of the sum of the squared distance from the data points to the proposed line. In addition, although the unsquared sum of distances might seem a more appropriate quantity to minimize, use of the absolute value results in discontinuous derivatives which cannot be treated analytically. The square deviations from each point are therefore summed, and the resulting residual is then

minimized to find the best fit line. This is achieved by calculating the derivative with respect to a_x, b_x and setting these to zero.

$$f(a_x, b_x) = \sum_{i=1}^{n} \left(a_x \cdot t_i + b_x - x_i\right)^2 \tag{4.9}$$

$$\frac{\partial f}{\partial a_x} = \sum \left[2 \cdot (a_x \cdot t_i + b_x - x_i) \cdot t_i\right] \tag{4.10}$$

$$\frac{\partial f}{\partial b_x} = \sum \left[2 \cdot (a_x \cdot t_i + b_x - x_i) \cdot 1\right] \tag{4.11}$$

$$\left(\sum t_i^2\right) \cdot a_x + \left(\sum t_i\right) \cdot b_x = \sum \left[t_i \cdot x_i\right] \tag{4.12}$$

$$\left(\sum t_i\right) \cdot a_x + \left(\sum 1\right) \cdot b_x = \sum \left[1 \cdot x_i\right] \tag{4.13}$$

Above equations can be represented in matrix form as follows.

$$\begin{bmatrix} s20 & s10 \\ s10 & s00 \end{bmatrix} \times \begin{bmatrix} a_x \\ b_x \end{bmatrix} = \begin{bmatrix} s11 \\ s01 \end{bmatrix} \tag{4.14}$$

Using Cramer's rule [35], a_x, b_x can be found and then a point in trajectory can be found using following equation at any instant of time t_i.

$$x_i = a_x \cdot t_i + b_x \tag{4.15}$$

Quadratic Regression for Parabolic Curve Fitting

Least Squares method of regression for parabolic curve fitting is an extension to linear regression. This method of fitting is used to extract three parameters. These three parameters are extracted so as to fit a parabolic curve. This is also called as quadratic regression to best fit a quadratic equation (parabolic curve) of the form $y(x) = ax^2 + bx + c$ to a collection of points. The algorithm basically requires minimization of the sum of the squared distance from the data points to the proposed line. This is achieved by calculating the derivative with respect to a_x, b_x, c_x and setting these to zero.

$$f(a_z, b_z, c_z) = \sum_{i=1}^{n} \left(a_z \cdot t_i^2 + b_z \cdot t_i + c_z - z_i\right)^2 \tag{4.16}$$

$$\frac{\partial f}{\partial a_z} = \sum \left[2 \cdot (a_z \cdot t_i^2 + b_z \cdot t_i + c_z - z_i) \cdot t_i^2\right] \tag{4.17}$$

$$\frac{\partial f}{\partial b_z} = \sum \left[2 \cdot (a_z \cdot t_i^2 + b_z \cdot t_i + c_z - z_i) \cdot t_i\right] \tag{4.18}$$

$$\frac{\partial f}{\partial c_z} = \sum \left[2 \cdot (a_z \cdot t_i^2 + b_z \cdot t_i + c_z - z_i) \cdot 1\right] \tag{4.19}$$

$$\left(\sum t_i^4\right) \cdot a_z + \left(\sum t_i^3\right) \cdot b_z + \left(\sum t_i^2\right) \cdot c_z = \sum \left[t_i^2 \cdot z_i\right] \tag{4.20}$$

$$\left(\sum t_i^3\right) \cdot a_z + \left(\sum t_i^2\right) \cdot b_z + \left(\sum t_i^1\right) \cdot c_z = \sum \left[t_i \cdot z_i\right] \tag{4.21}$$

$$\left(\sum t_i^2\right) \cdot a_z + \left(\sum t_i^1\right) \cdot b_z + \left(\sum 1\right) \cdot c_z = \sum \left[1 \cdot z_i\right] \tag{4.22}$$

This can be represented in matrix form as follows.

$$\begin{bmatrix} s40 & s30 & s20 \\ s30 & s20 & s10 \\ s20 & s10 & s00 \end{bmatrix} \times \begin{bmatrix} a_z \\ b_z \\ c_z \end{bmatrix} = \begin{bmatrix} s21 \\ s11 \\ s01 \end{bmatrix} \tag{4.23}$$

Using Cramer's rule a_z, b_z, c_z can be found and then a point in trajectory can be found using following equation at any instant of time t_i.

$$z_i = a_z \cdot t_i^2 + b_z \cdot t_i + c_z \tag{4.24}$$

4.3.2 Kalman Filter Based Estimation

A Kalman filter can be used to predict the state of a system where there is a lot of input noise. A state estimation algorithm determines the values of a number of parameters of a system, such as its position and velocity, from measurements of the properties of that system. The Kalman filter forms the basis of most state estimation algorithms used in navigation systems. Its uses include maintaining an optimal satellite navigation solution, integration of global navigation satellite system (*i.e.*, GNSS) user equipment with other navigation sensors, and alignment and calibration of an inertial navigation system (*i.e.*, INS). State estimation is key to obtaining the best possible navigation solution from the various measurements available. A Kalman filter uses all the measurement information input available until a given moment, not just the most recent set of measurements. Developed around 1960 mainly by Rudolf E. Kalman [36], It was originally designed for aerospace guidance applications. While it is the optimal observer for system with noise, this only true for the linear case. A non-linear Kalman Filter can not be proven to be optimal.

Basic Kalman Filter Equations

Kalman filtering, also known as linear quadratic estimation (LQE), is an optimal estimator (*i.e.*, infers parameters of interest from indirect, inaccurate and uncertain observations). It is recursive so that new measurements can be processed as they arrive. Basic representation of Kalman filter is given by expression 4.25 [32]. Figure 4.16 depicts basic data flow in a Kalman model during estimation process.

$$\begin{bmatrix} \textbf{System Description} \\ \dot{x} = \mathbf{A}x + \mathbf{B}u \\ \mathbf{z} = \mathbf{H}_x \end{bmatrix} \begin{bmatrix} \textbf{Time Update} \\ x_{k+1} = \mathbf{A}x + \mathbf{B}u \\ \mathbf{P}_{k+1} = \mathbf{A}\mathbf{P}_k\mathbf{A}^T + \mathbf{Q} \end{bmatrix} \begin{bmatrix} \textbf{Measurement Update} \\ \mathbf{K} = \mathbf{P}_k\mathbf{H}^T(\mathbf{H}\mathbf{P}_k\mathbf{H}^T + \mathbf{R})^{-1} \\ x_{k+1} = x_k + \mathbf{K}(y_k - \mathbf{H}x_k) \\ \mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\mathbf{P}_k \end{bmatrix} \tag{4.25}$$

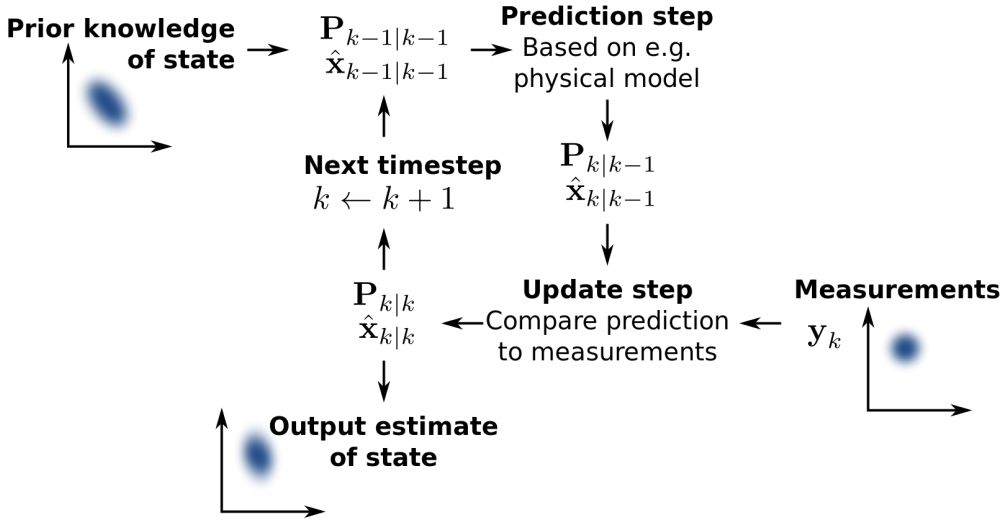


Figure 4.16: Basic representation of data flow in Kalman filter based process.

Kalman Filter Model for Ball Catching Application

The kinematics of the ballistic object in the re-entry phase is derived under the hypotheses of Newton's laws of motion. The forces acting on the target are gravity. The effects of drag, centrifugal acceleration, Coriolis acceleration, wind, lift force, and spinning motion are ignored, due to their small effect on the trajectory as the time of flight is very small. Another simplifying assumption is related to flat Earth approximation. Having assumed a flat Earth, an orthogonal coordinate reference system can be used with the concerning variables. Some of the basic projectile equations presented below, these equations 4.26, 4.37, and 4.28 represent projectile motion in 3 dimensional space.

$$x = x_0 + vx_0 \cdot t \tag{4.26}$$

$$y = y_0 + vy_0 \cdot t \tag{4.27}$$

$$z = z_0 + vz_0 \cdot t + \frac{1}{2} \cdot g \cdot t^2 \tag{4.28}$$

Kalman Filter model for ballistic motion [37] can be designed from above stated equations, according to the requirement of states. A state can a position or velocity or acceleration. State transition matrix for ball catching application without mass as a factor is as follows:

$$A = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.29}$$

State transition matrix with mass as a factor is little bit different and is as follows:

$$A = \begin{bmatrix} 1 & 0 & 0 & dt - \frac{k \cdot dt^2}{2 \cdot m} & 0 & 0 \\ 0 & 1 & 0 & 0 & dt - \frac{k \cdot dt^2}{2 \cdot m} & 0 \\ 0 & 0 & 1 & 0 & 0 & dt - \frac{k \cdot dt^2}{2 \cdot m} \\ 0 & 0 & 0 & 1 - \frac{k \cdot dt}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 - \frac{k \cdot dt}{m} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 - \frac{k \cdot dt}{m} \end{bmatrix} \tag{4.30}$$

Input control matrix for ball catching application is as follows:

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.31}$$

Observation matrix H is the ″link″ between a predicted state vector X and measurement vector z, such that $z = HX$ is satisfied. So the observation matrix is a parameter of filter and generally it doesn't change.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.32}$$

Input matrix is the one that contains controlling parameters such as components of force or drag.

$$u = \begin{bmatrix} 0 \\ 0 \\ 0.5 \cdot g \cdot dt^2 \\ 0 \\ 0 \\ g \cdot dt \end{bmatrix} \tag{4.33}$$

Kalman Filter Implementation

To implement a Kalman filter model for ball position estimation, it is required to take some parameters in to consideration. State co-variance matrix changes as the estimation starts. it contains all the information about last samples. It can be said that this method of estimation not only depends upon last sample, but also depends upon all the samples from initialization. Initialized states change with each new sample but co-variance matrices except state co-variance matrix doesn't change at all.

These matrices are initialized at the beginning of process and must be reset to this initialized value at the end of process. Initialization of state matrix is done with some reasonable value, so as to get correct estimation as soon as possible. Co-variance matrices represent the confidence value on the specific parameter, and can be initialized with guessing. This initialization methods works well if some prior knowledge about the process in terms of sensor imperfection and possible error is available.

Initialized states, This contains position and velocity in the order of x, v_x, y, v_y, z, and v_z.

$$X = \begin{bmatrix} 0.5 \\ 3.0 \\ 0.5 \\ 3.0 \\ 5.5 \\ 3.0 \end{bmatrix} \tag{4.34}$$

State co-variance matrix P is the initial guess for the co-variance of the states.

$$P = 1000 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.35}$$

Matrix Q is the process co-variance.

$$Q = 0.1 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.36}$$

Matrix R is the measurement co-variance.

$$R = 0.2 \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.37}$$

Estimation of the Landing Point

At the very beginning of estimation process, Kalman model starts giving estimated velocity and position in all the three dimensions. These estimates parameters can be termed as coefficients of ballistic model as presented in equation 4.26, 4.27, and 4.28. Expression 4.28, with known coefficients can be used to get time of flight by solving typical quadratic equation. This time of flight is a dynamic parameter, which depends upon current velocity, current position and landing plane. With time of flight, current position, and current velocity, landing point in 3D space can be calculated using expressions 4.26, 4.27, and 4.28.

4.3.3 Comparative Analysis of Trajectory Estimation Methods

Table 4.2 presents a basic comparison between discussed methods for ball trajectory estimation in terms of implementation.

Two different methods of ball trajectory estimation implemented but it is worth of a comparative analysis on experimental results of these methods to get an overall idea of what can be done to make the algorithm better. Both of these methods will be used for experimental analysis in chapter 6.

Table 4.2: Comparison of trajectory estimation methods in terms of implementation.

Kalman filter based estimation	**Polynomial approximation**
Runs at 30 Hz	Runs at 30 Hz
Performance is affected by initialized values	No problem of initialization
Estimates the very next state but, can be used to estimate up to certain plane	Estimates position at any time but, can be used to estimate up to certain plane
Estimated error is reduced in every iteration	No scope of error correction, but estimated error will reduce if subsequent measurements are added for regression
Requires process model	Requires information about the type of curve with which data to be fitted
Suitable for ball catching application	Suitable for ball catching application

CHAPTER 5

Manipulator Arm Motion Control

The main objectives of this chapter are to explore the possible solutions for the 7-DOF manipulator arm in terms of forward and inverse kinematics for ball catching application. Robot motion control applies geometry to the study of the movement of multi-degree of freedom kinematic chains that form the structure of robotic systems. The emphasis on geometry means that the links of the robot are modeled as rigid bodies and its joints are assumed to provide pure rotation or translation [38]. Kinematics is the study of the mathematics of motion with out considering the forces that affect the motion. Figure 5.1 presents the basic concept of robot kinematics in the field of robotics. Where joint values can be in degrees/radians (joint space) and exactly opposite space contains information about end-effector coordinate and orientation (Cartesian space) [39].

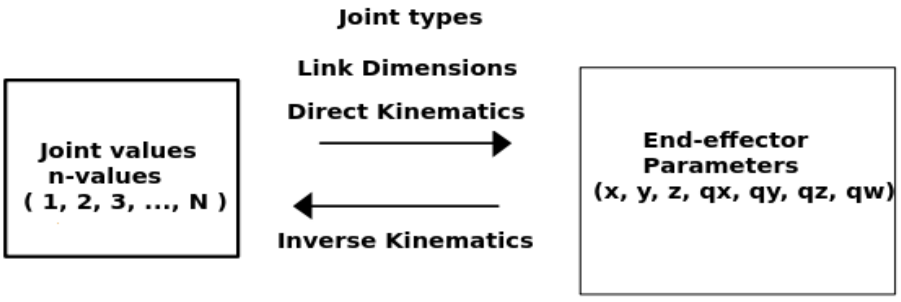


Figure 5.1: Basic kinematics formulation in the field of robotics.

The dynamixel servomotors provide control actions in a typical position mode using the actuator's built-in micro-controller. In addition to that, ROS provides dy-

namixel motor package for direct interface with dynamixel servo-motors. This dynamixel motor package will be used extensively for low-level control (*i.e.*, control in joint space). Next section is dedicated towards study of the MoveIt! software for this dissertation work in order to get possible solution for Cyton Gamma 1500 7 DOF manipulator arm kinematics, to be able to implement manipulator arm motion control for ball catching application.

5.1 MoveIt! and Robotic Arm Description in ROS

For the application of ball catching, performance evaluation of robotic arm is necessary to get an overall idea of feasibility of proposed work. This performance evaluation includes study of kinematics analysis, joint limits, joint velocity limits, end-effector velocity in Cartesian space and executions rate of motion control algorithm for manipulator arm motion control. Ball catching task requires high speed perception and actuation so as to get the work done in a specific time without having prior knowledge of initial position, and velocity of ball. This requires best performance form the manipulator to be used.

A robotic manipulator arm with 7 degrees of freedom is flexible enough to reach any point inside it's work space. It exhibits infinite redundancy if all the possible orientations are considered while calculating inverse kinematics. Inverse kinematics calculation can be done either analytically or based on numerical methods. For a manipulator of 7 degrees of freedom it is really difficult to implement analytic method if redundancy is not avoidable. For example, the manipulator arm Cyton Gamma 1500, with only one orientation it has 100 thousand points inside it's work space, where it can reach. For any kind of manipulator arm having infinite redundancy, it is difficult to get inverse kinematics with analytic method. There exist one such open source library for calculation of inverse kinematics known as MoveIt! [40].

MoveIt! is a state of the art software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotics applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains. The default kinematic solver is the KDl solver [41], which is a numerical method based solver. Other inverse kinematics solver can also be added easily. Using MoveIt! is easy, but to implement MoveIt! in ROS, robot model is required for manipulator kinematics. Before going to motion control analysis, some of the important ROS packages that are commonly used to build robot models are to be discussed. Robot models are basically used before and after MoveIt! integration for forward and inverse kinematics.

5.1.1 The robot_model package

A robot model [42] is that package, which contains all the required information about joint types, link dimensions, and transforms for whole arm, specified in the XML Robot Description Format (*i.e.*, URDF (Universal Robot Description Format)). ROS has a meta package called robot_model, which contains important packages that helps in building the 3D robot models. Figure 5.2 shows the 3D robot model created using robot_model meta package and CAD model for Cyton Gamma 1500 manipulator arm.

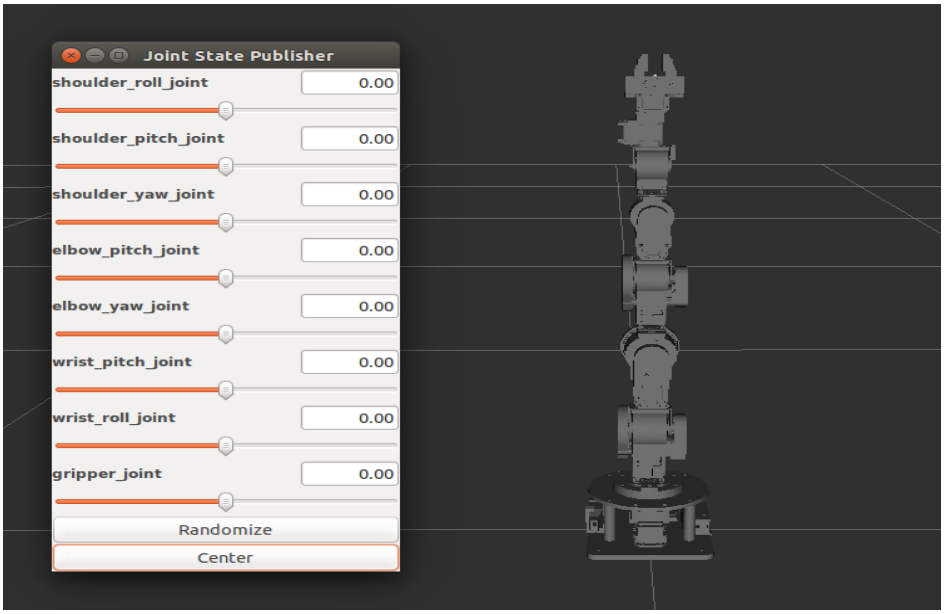


Figure 5.2: Cyton Gamma 1500 robot model in rviz.

All the important packages inside robot_model meta package are as follows [43]:

- **urdf:** The oldest in ROS for robot description and one of the important packages inside the robot_model meta package is urdf. The URDF package contains a C++ parser for the Unified Robot Description Format (URDF), which is an XML file to represent a robot model. One can define a robot model, sensors, and a working environment using URDF and can parse it using URDF parser.

- **joint state publisher:** This tool is very useful while designing robot models using URDF. This package contains a node called joint_state_publisher, which reads the robot model description, finds all joints, and publishes joint values to all non-fixed joints using GUI sliders.

- **kdl parser:** Kinematic and Dynamics Library (KDL) is an ROS package that contains parser tools to build a KDL tree from the URDF representation. The kinematic tree can be used to publish the joint states and also to forward and inverse kinematics of the robot.

- **robot state publisher:** This package reads the current robot joint states and publishes the 3D poses of each robot link using the kinematics tree build from the URDF. The 3D pose of the robot is published as ROS tf (transform). ROS tf publishes the relationship between coordinates frames of a robot.

- **xacro:** Xacro stands for (XML Macros) and one can define how xacro is equal to URDF plus add-ons. It contains some add-ons to make URDF shorter, readable, and can be used for building complex robot descriptions. It is possible to convert xacro to URDF at any time using some ROS functionality.

5.1.2 Arm Configuration from MoveIt

To be able to use MoveIt!, it is necessary to generate configuration files, that are compatible with MoveIt!. MoveIt! takes the .urdf file and generates different configuration files (*i.e.*, controllers configuration file, arm configuration file In .srdf format, joint limits file, joint names file, and kinematics file). These files are to be used for planning, manipulation, kinematics, collision checking, control, and navigation and are described below.

Controllers Configuration File

This contains list of controllers the arm contains. This will specify the controller configuration for your robot.

Arm Configuration File in .srdf Format

SRDF is compact term for Semantic Robot Description Format. This format is intended to represent information about the robot that is not in the URDF file, but it is useful for a variety of applications. The intention is to include information that has a semantic aspect to it. URDF can be used for specifying only the kinematic and dynamic properties of a single robot in isolation. URDF can not specify the pose of the robot itself within a world. It is also not a "universal" description format since it cannot specify joint loops, and it lacks friction and other properties. Additionally, it cannot specify things that are not robots, such as lights, height maps, etc.

On an implementation side, the URDF syntax breaks proper formatting with heavy attributes which in turns makes URDF more inflexible. There is also no mechanism for backward compatibility. SRDF solves all these problems. It is a complete description for everything from the world level down to the robot level. It is highly scalable, and extremely easy to add and modify elements. The SRDF format is itself described using XML, which facilitates a simple upgrade tool to migrate old versions to new versions. It is also self descriptive. The only reason to use URDF is because it has been historically used within ROS.

Joint Limits File

joint_limits.yaml allows the dynamics properties specified in the URDF to be overwritten or augmented as needed specific joint properties can be changed with the keys [max_position, min_position, max_velocity, max_acceleration]. Joint limits can be turned off with [has_velocity_limits, has_acceleration_limits]

Joint Names File

joint_names.yaml contains all the joint names that Moveit! will be using for inverse kinematics and forward kinematics calculations.

Kinematics File

The **kinematics.yaml** file generated by the MoveIt! Setup Assistant is the primary configuration file for kinematics for MoveIt!. The set of available parameters include:

- **kinematics solver:** This is the name of our kinematics solver plugin. Note that this must match the name that we have specified in the plugin description file.
- **kinematics solver search resolution:** This specifies the resolution that a solver might use to search over the redundant space for inverse kinematics, e.g. using one of the joints for a 7 DOF arm specified as the redundant joint.
- **kinematics solver time-out:** This is a default time-out specified (in seconds) for each internal iteration that the inverse kinematics solver may perform. A typical iteration (e.g. for a numerical solver) will consist of a random restart from a seed state followed by a solution cycle (for which this time-out is applicable). The solver may attempt multiple restarts - the default number of restarts is defined by the kinematics_solver_attempts parameter below.
- **kinematics solver attempts:** The number of random restarts that will be performed on the solver. Each solution cycle after the restart will have a time-out defined by the kinematics_solver_timeout parameter above. In general, it is better to set this time-out low and fail quickly in an individual solution cycle.

5.1.3 Dynamixel-ROS Interface

The ROS stack for interfacing the dynamixel motor is dynamixel_motor. This stack contains interface for several types of dynamixel motors. The stack consists of the following packages:

- **dynamixel_driver:** This package is the driver package of dynamixel, which can do low level Input/Output communication with dynamixel from PC. This driver has hardware interface for the previously mentioned series of servos and can do the read/write operation to Dynamixel through this package. This

package is used by high level packages such as dynamixel_controllers. There are only few cases when the user directly interacts with this package.

- **dynamixel_controllers:** This is a higher level package that works using the dynamixel_motor package. Using this package, we can create a ROS controller for each Dynamixel joint of the robot. The package contains a configurable node, services, and spawner script to start, stop, and restart one or more controller plugins at once. In each controller, we can set the speed and the torque. Each dynamixel controller can be configured using the ROS parameters or can be loaded by a YAML file. The dynamixel_controllers packages support the following kinds of controllers:
  - Joint Position controllers
  - Joint Torque controllers
  - Joint Trajectory Action controller

- **dynamixel_msgs:** These are the message definitions which are used inside the dynamixel_motor stack.

Figure 5.3 [20] presents an overall PID structure of dynamixel_controller. This controller model is similar to both MX-28 and MX-64 servo motors.

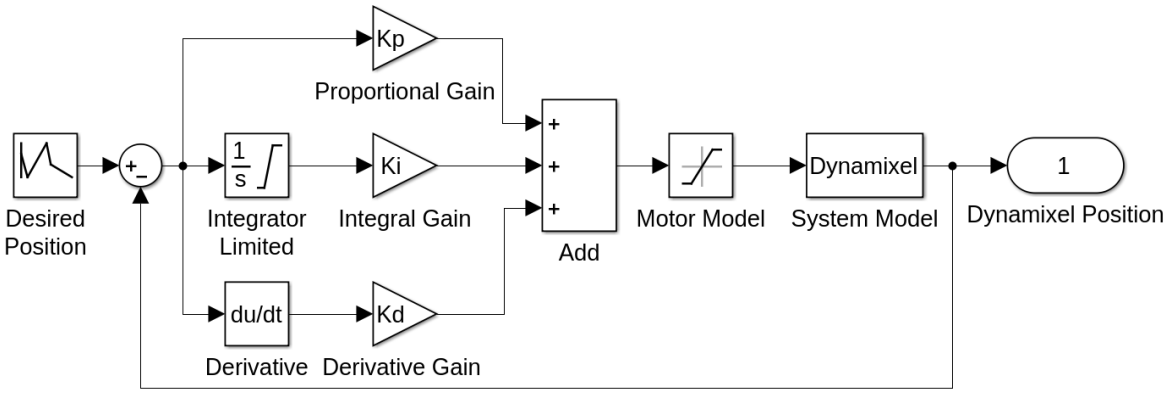


Figure 5.3: PID control structure for Cyton Gamma 1500 Servos.

This dynamixel_controller is the low level controller that directly interacts with the servo motors of Cyton Gamma 1500 arm. Servo motors of MX-series use the PID controller as a main control method. P gain refers to the value of proportional band, I gain refers to the value of integral action, and D Gain refers to the value of derivative action. Where, gain values are in between 0 to 254. Where $K_p, K_i, and K_d$

are defined as follows [20]:

$$K_p = P \cdot \frac{Gain}{8}$$
$$K_i = I \cdot \frac{Gain \cdot 1000}{2048}$$
$$K_d = D \cdot \frac{Gain \cdot 4}{1000}$$

5.1.4 Moveit and KDL for Inverse Kinematics

Cyton Gamma 1500 is nothing but a chain of dynamixel servos, and can be considered as a chain of servo motors for kinematics based operations. This kinematic chain can be presented as a KDL Chain object, and KDL solvers can be used to compute anything from forward position kinematics, to inverse dynamics. The kdl_parser includes support to construct a KDL chain from a XML of Robot Description Format (URDF) file. Figure 5.4 presents the work space for Cyton Gamma 1500 for only one orientation, inverse kinematics from MoveIt! is used for work space analysis.

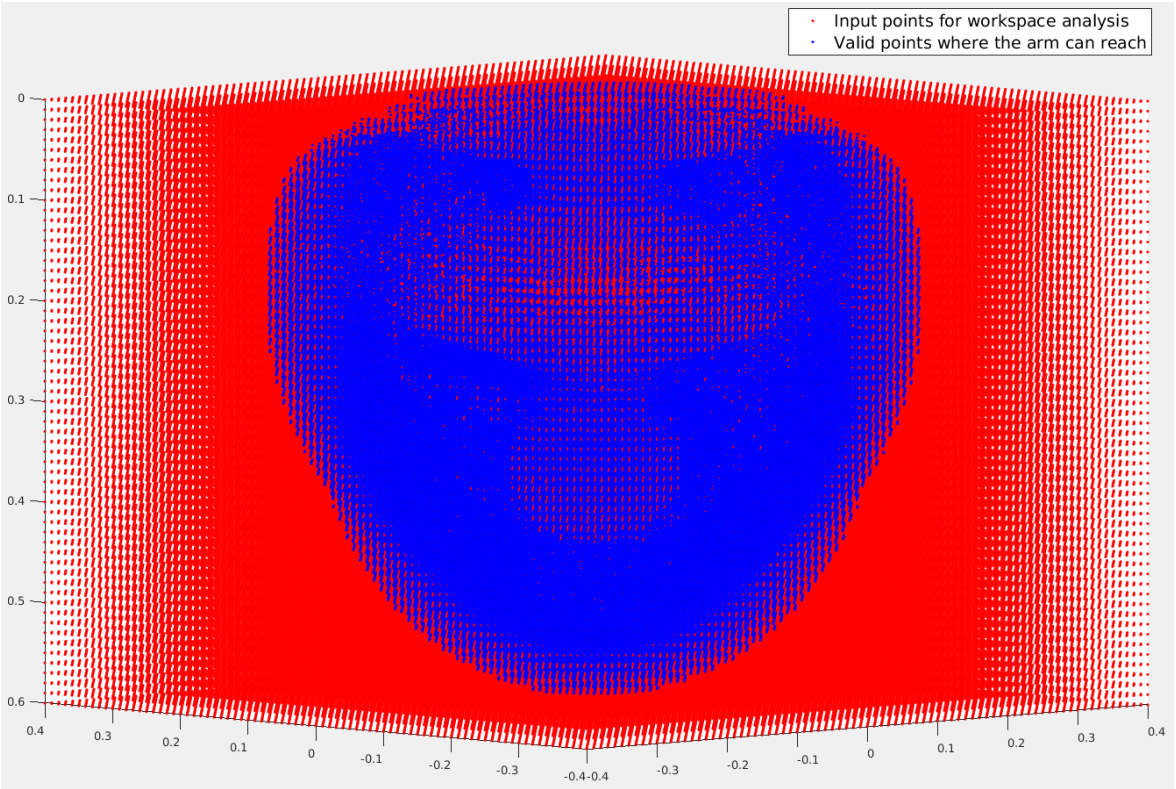


Figure 5.4: Cyton Gamma 1500 work space for a particular orientation.

MoveIt! provides a great flexibility to switch the inverse kinematics algorithms using the robot plugins. Anyone can write his own inverse kinematics solver as

a MoveIt! plugin and switch from the default solver plugin (*i.e*, KDL) whenever required. The default inverse kinematics solver in MoveIt! is a numerical Jacobian-based solver (*i.e*, KDL solver). It is possible to use it with any number of joints but it should be noted that with more than 6 DOF, the robot is redundant and there can be multiple solutions. KDL gives one of the solution, depending on which initial position (i.t., seed) it is started. Of-course there are certain positions where singularities will cause issues in finding solutions but the solver does use a damped least squares method which should help with some of those situations [43].

It can be seen from Figure 5.4, that 100 thousand points are available for only one orientation and the robotic arm can reach almost all the positions inside work space. Compared to the analytic solvers, the numerical solver can take time to solve Inverse Kinematics. To be able to interface a robotic arm in MoveIt!, one need to satisfy the components that are mentioned in previous sections. The move_group node essentially requires parameters such as URDF, SRDF, configuration files, and joint states topics along with TF from a robot to start with motion planning. MoveIt! provides a GUI based tool called "Setup Assistant" to generate all these elements. This tool is used to generate required files for Inverse kinematics implementation. At this stage, it is tried and tested with real arm to control it in inverse kinematics mode using move_group node and robot_model with c++ compatible API's.

5.1.5 Final Remarks on MoveIt! and KDL

As stated in previous sections, KDL is a numerical method based solver, that is being is used for calculating inverse kinematics solution. Below are presented some of the experimental observations.

1. Runs at approximately 5 Hz, as inverse kinematics computation takes more time.

2. Not easy to implement, because lots of additional files are required by MoveIt!.

3. Arm is more flexible with more Degrees of freedom.

4. Not suitable for ball-catching application with available resource, because it takes more time for computation.

5. This method of inverse kinematics (*i.e.*, numerical method based solver) computation is suitable for low dynamic operations.

Considering all the above stated observations, it can be concluded that this method (*i.e.*, MoveIt! for inverse kinematics using KDL solver) of arm motion control is not suitable for ball catching application with Cyton Gamma 1500 manipulator arm. From next section on-wards, it is considered that a 3 DOF configuration for Cyton Gamma 1500 manipulator arm will be used with inverse kinematics solution based on analytic method.

5.2 3 DOF Manipulator Arm Control System

To be able to control a robotic manipulator, it is essential to know its kinematics. As stated earlier that, kinematics is the study of the mathematics of motion with out considering the forces that affect the motion, this section is all about direct kinematics, and inverse kinematics of 3 DOF configuration for Cyton Gamma 1500 robotic arm. Direct kinematics is used to obtain position of the end-effector in Cartesian coordinate frame on the base of the robotic arm by knowing its joint angles. On the other hand, inverse kinematics is used to know the joint angles that the robotic arm requires to reach the desired position, without considering the path.

5.2.1 Direct Kinematics

Figure 5.5 presents coordinate frame for Cyton Gamma 1500 in 3 DOF mode. Table 5.1 presents details about DH-parameters for Cyton Gamma 1500 in 3 DOF configuration. All other joints, those will not be considered for motion control are blocked to be able to reduce the 7 DOF manipulator arm to 3 DOF structure.

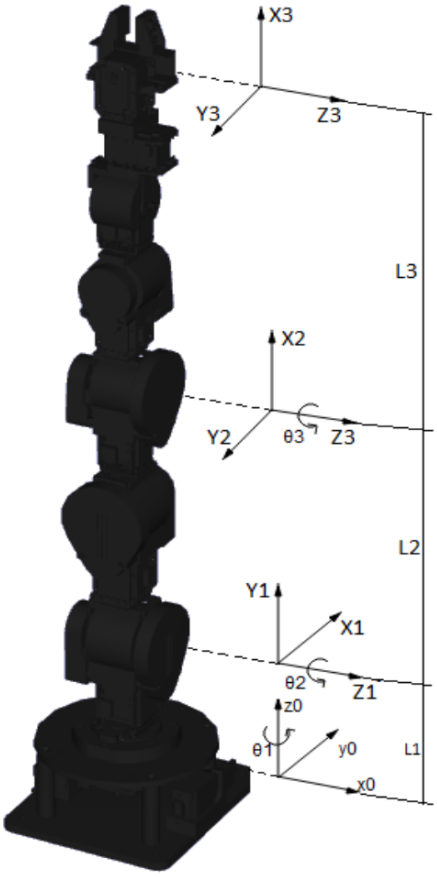


Figure 5.5: Coordinate frames of Cyton Gamma 1500 in 3 DOF mode.

Table 5.1: DH parameters of 3 DOF configuration.

link	θ_i	L_i	d_i	α_i
1	$90° + \theta_1$	0	L_1	90°
2	$90° + \theta_2$	L_2	0	0
3	θ_3	L_3	0	0

Figure 5.6 presents an overall structure of forward kinematics for Cyton Gamma 1500 robotic arm. Here cyton_FK node is the intermediate node just before dynamixel_manager. The low level control is managed by dynamixel_manager. For 7 DOF configuration the manipulator arm is controlled by setting position and velocity for all the available joints. Whereas, for 3 DOF configuration it is done by setting position and velocity for corresponding joints(3 joints) and all other joints (*i.e.*, joints that must be stiff), are set to zero position and zero velocity so that they are blocked from moving. The node robot_state_publisher can be used for real-time visualization in rviz. This can be termed as a kind of imitation of real-arm movement in visualization window in real-time.

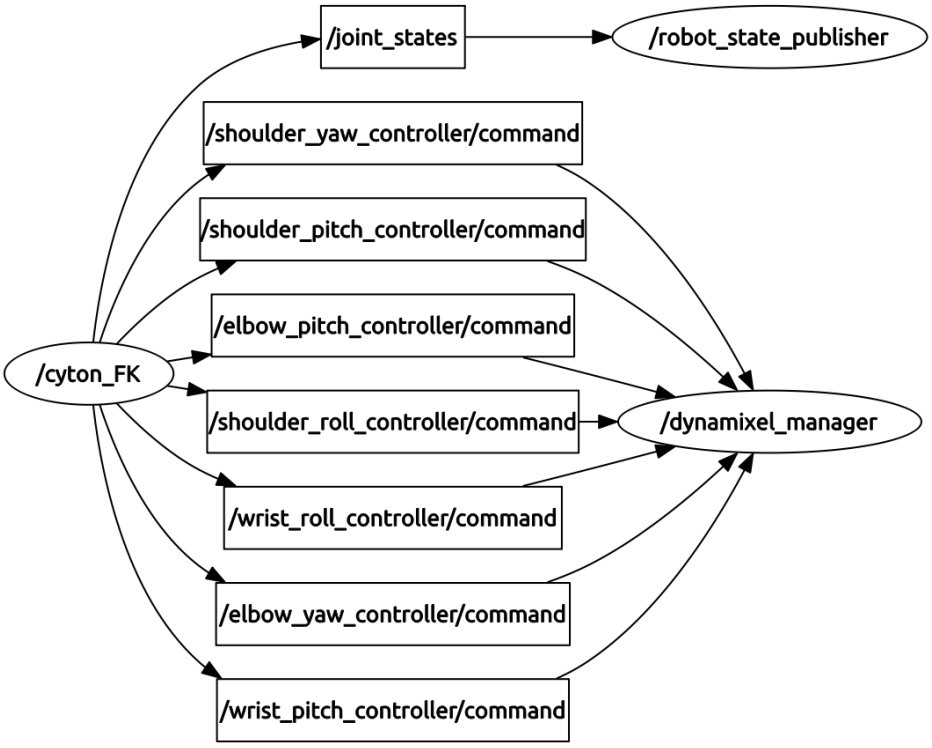


Figure 5.6: Forward kinematics control structure.

5.2.2 Inverse Kinematics

Usually, the Inverse Kinematics is determined by manipulating the direct kinematics expression. The expression of θ_1, presented in the equation (5.2), can be obtained using (5.1).

$$\frac{X}{Y} = \frac{\sin(\theta_1) \cdot (L_2 \cdot \sin(\theta_2) + L_3 \cdot \sin(\theta_2 + \theta_3))}{-\cos(\theta_1) \cdot (L_2 \cdot \sin(\theta_2) + L_3 \cdot \sin(\theta_2 + \theta_3))} \tag{5.1}$$

$$\theta_1 = arctan\left(\frac{X}{-Y}\right) \tag{5.2}$$

θ_2 can be obtained by manipulating x and z expressions.

$$\frac{X}{\sin(\theta_1)} = L_2 \cdot \sin(\theta_2) + L_3 \cdot \sin(\theta_2 + \theta_3) \tag{5.3}$$

$$Z - L_1 = L_2 \cdot \cos(\theta_2) + L_3 \cdot \cos(\theta_2 + \theta_3) \tag{5.4}$$

From expressions 5.2, 5.3, and 5.4, after manipulation the form of $k_1 \cdot \sin(\theta) + k_2 \cdot \cos\theta = k_3$ is obtained. This contains an analytic solution and (θ_2, θ_3, k_1, k_2, k_3) can be computed as follows:

$$k_1 = 2 \cdot L_2 \cdot \left(\frac{Y}{\sin(\theta_1)}\right)$$

OR

$$2 \cdot L_2 \cdot \left(\frac{Y}{-\cos(\theta_1)}\right) \tag{5.5}$$

$$k_2 = 2 \cdot L_2 \cdot (Z - L_1) \tag{5.6}$$

$$k_3 = -L_3^2 + L_2^2 + \left(\frac{Y^2}{\sin(\theta_1^2)}\right) + (Z - L_1)^2$$

OR

$$-L_3^2 + L_2^2 + \left(\frac{Y^2}{\cos(\theta_1^2)}\right) + (Z - L_1)^2 \tag{5.7}$$

The solution for θ_2 is given by the equation 5.8.

$$\theta_2 = atan\left(\frac{k_1}{k_2}\right) \pm atan\left(\frac{\sqrt{k_1^2 + k_2^2 - k_3^2}}{k_3}\right) \tag{5.8}$$

The expression of θ_3 is given by the equation (5.9).

$$\theta_3 = \pm\left(\frac{\sqrt{k_2^2 - k_3^2}}{k_3}\right) \tag{5.9}$$

5.2.3 Implementation of the Point-to-Point Motion Control

For this dissertation work, point-to-point control mode will be used for manipulator motion control. The point-to-point control mode is usually used used when the goal is to move the end-effector to a certain position regardless of the path. Figure 5.7 presents an overall structure of inverse kinematics for Cyton Gamma 1500 arm for motion control in point-to-point control mode. The topic /pubEE publishes desired position and is subscribed by /cyton_JS. The node /cyton_JS collects joint states and publishes them along with desired end-effector in /jointArmData topic. The node /cyton_IK subscribes /jointArmData and computes inverse kinematics solution. This inverse kinematics solution is used by cyton_FK for arm motion in real-time. The node cyton_JS subscribes all the joint states, so as to be able to set joint velocity for smooth motion. The node /cyton_SV calls services to set velocity for each joint according to joint values subscribed from /jointIkData.

Synchronized control of joint movement is implemented after getting joint displacement from inverse kinematics solver. Each joint has maximum speed limit and this speed is used for faster movement. Each time the joint displacement is calculated, maximum velocity for maximum displacement for synchronous mode of operation, so that all the joint start and stop at same time. The node robot_state_publisher can be used for real-time visualization in rviz. Advantage of this synchronized mode of joint control is that, it allows smooth movement and precise control. This method of control reduces vibration and the arm remains more.

5.2.4 Evaluation of the Manipulator Arm's Behavior

Robot motion planning is usually done by ignoring dynamics and other differential constraints and primary focus is on the translations and rotations required to move the end-effector to desired position with or without considering orientation. This section is dedicated for analysis of joint velocity, end-effector real position for any desired target position and relative errors associated during operation. Experimental tests are conducted with real manipulator arm to test velocity constraints and goal position error.

Evaluation of the manipulator arm's motion can be done in many different ways (*i.e.*, maximum velocity in Cartesian space, maximum velocity in joint space, and reachability of the end-effector) according to the requirements. In this case, it is important to evaluate the arm's capability in terms of maximum velocity in Cartesian space as well as reachability of the end-effector. It is assumed that the manipulator joints will be operating at maximum speed and hence maximum joint velocity analysis is not taken into account.

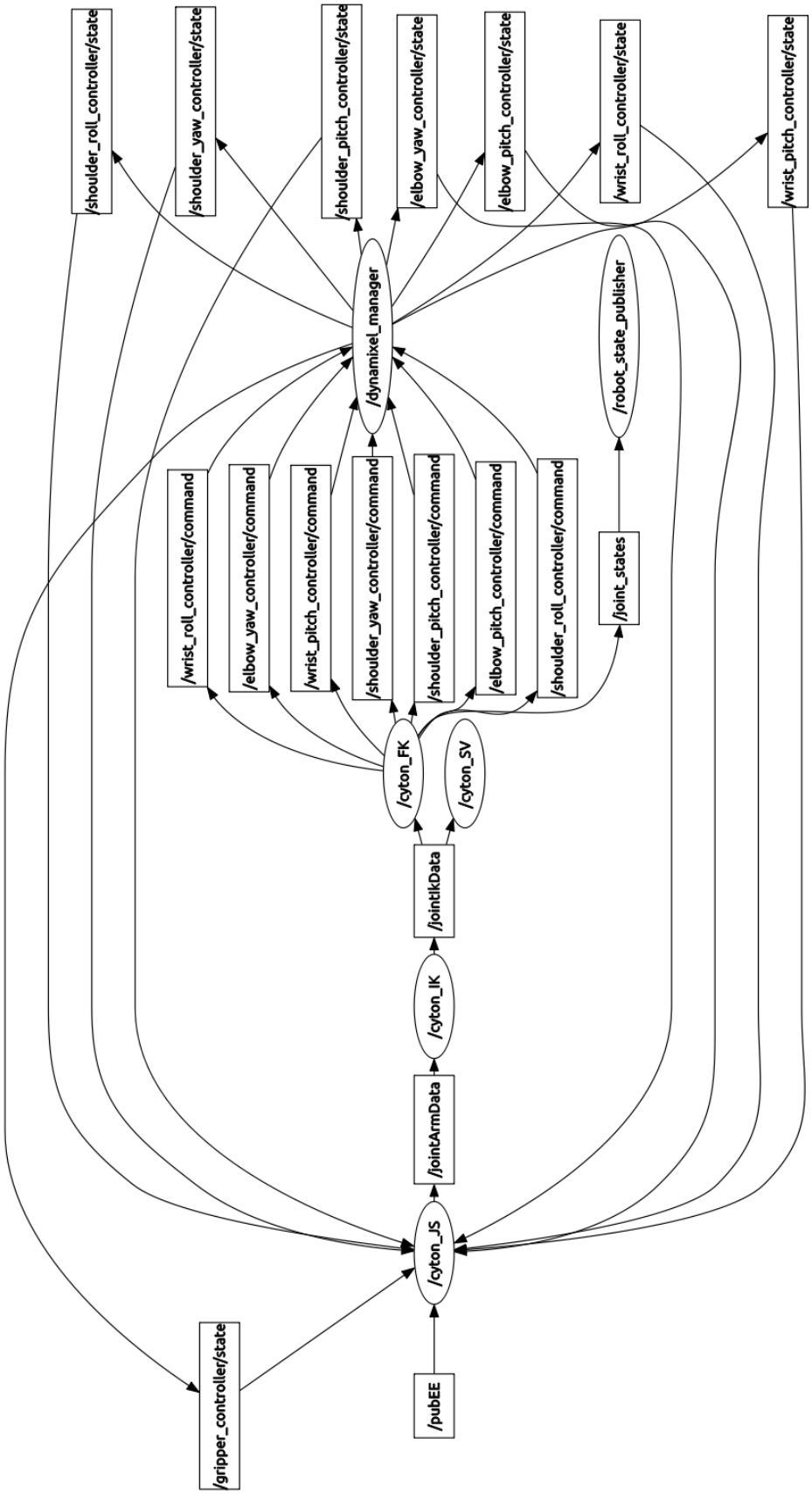


Figure 5.7: Synchronized inverse kinematics control structure.

- **Random Targets for Reachability Test:**

  This experiment is done in order to evaluate the reachability of robotic arm to any random position in a free mode (*i.e.*, the end-effector path is not planned). This test is done at different frequency of operation. Figure 5.8 presents one of the test that is conducted at a rate of 1 Hz in order to allow the end-effector to reach the desired position.

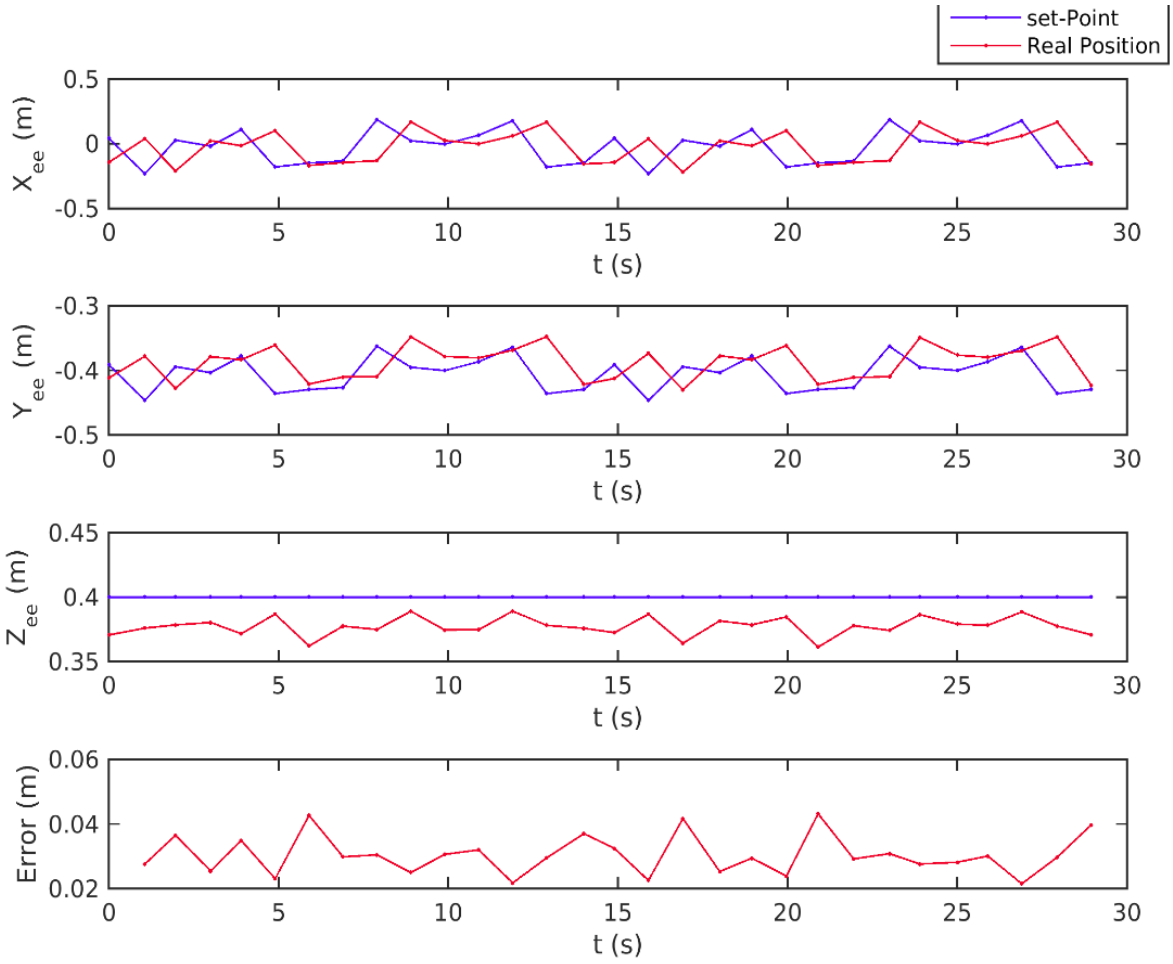


Figure 5.8: Cyton Gamma 1500 arm response to target random position.

From Figure 5.8, it is evident that the robotic arm takes significant amount of time to reach the goal position. In Figure 5.8, the real position and desired position of end-effector in X, Y, Z coordinate respectively, and finally the associated error in Cartesian space is presented. The error associated with this reachability test is found to be 0.030 ± 0.006 meter. This means that, the arm takes time to reach the desired goal, but the associated error is acceptable considering manipulator arm and controller imperfections.

- **Maximum Velocity in Cartesian Space:**

As stated in last part that, the arm is capable of reaching a goal position but, it is important to analyze end-effector velocity in Cartesian space of robotic arm, considering joint velocity constraints. Figure 5.9, presents the analysis of end-effector set-point at any time-instant and real position of end-effector at that time-instant. It can be seen that the arm takes significant amount of time to reach the desired position in all the three-axes. This experiment is carried out at a rate of 0.2 Hz, in order to allow the arm to reach the desired position without considering time as a constraint. In this case, maximum velocity for all the working joints are imposed, so as to get exact behavior of arm and faster movement.

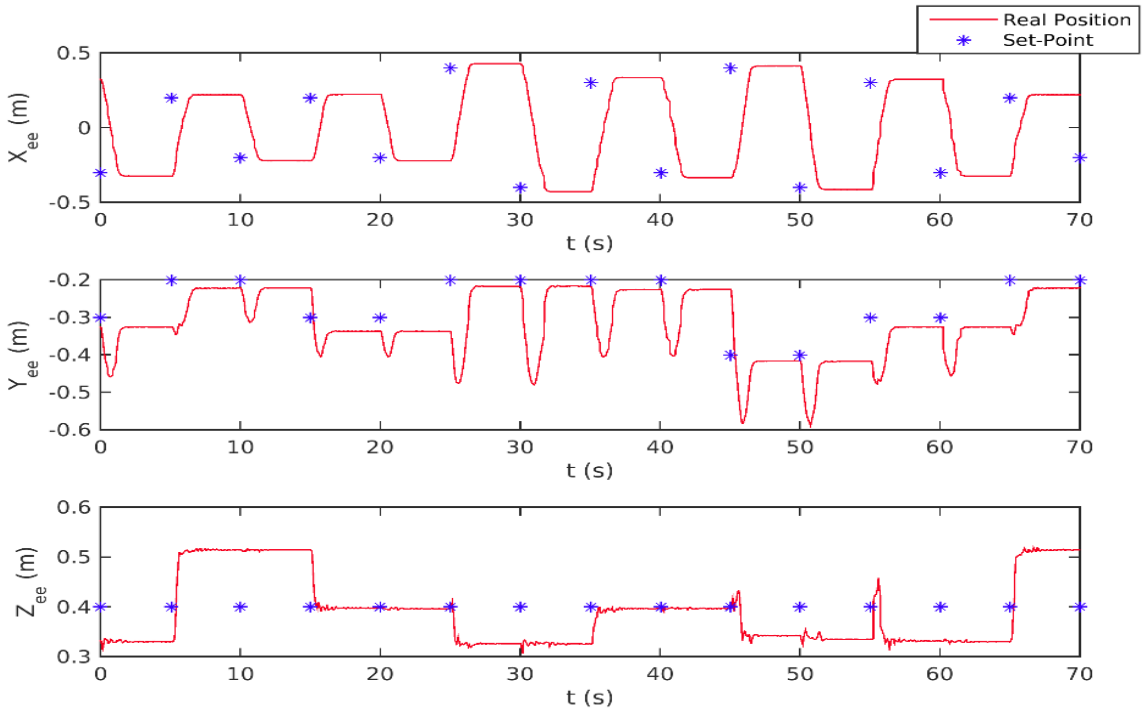


Figure 5.9: Analysis of maximum velocity in Cartesian space of Cyton Gamma 1500 arm.

Table 5.2 presents the velocities associated for different test points and time taken to reach goal position. From this experiment, it can be concluded that the arm can move in Cartesian space at a rate of 0.44308 ± 0.052933 meter/second. considering these specification, ball-catching task can be performed both in simulation and real-time scenario for further analysis.

Comparative Analysis of Manipulator Arm Motion Control Methods

Two methods are implemented for manipulator arm motion control. These methods of arm motion control have their own advantages and disadvantages according to several factors. Table 5.3 contain some of the experimentally observed factors that affect arm motion control.

Table 5.2: Analysis of maximum velocity attained by Cyton Gamma 1500 in Cartesian space.

Test Number	Initial Point(x,y,z)	Final Point(x,y,z)	Displacement (ms)	Time Taken (s)	End-effector velocity (m/s)
1	0.30, -0.30, 0.40	-0.30, -0.30, 0.40	0.60	1.30	0.46
2	-0.30, -0.30, 0.40	0.20, -0.20, 0.40	0.51	1.25	0.41
3	0.20, -0.20, 0.40	-0.20, -0.20, 0.40	0.40	1.00	0.40
4	-0.20, -0.20, 0.40	0.20, -0.30, 0.40	0.41	1.18	0.35
5	0.20, -0.30, 0.40	-0.20, -0.30, 0.40	0.40	1.00	0.40
6	-0.20, -0.30, 0.40	0.40, -0.20, 0.40	0.61	1.35	0.45
7	0.40, -0.20, 0.40	-0.40, -0.20, 0.40	0.80	1.75	0.46
8	-0.40, -0.20, 0.40	0.30, -0.20, 0.40	0.70	1.55	0.45
9	0.30, -0.20, 0.40	-0.30, -0.20, 0.40	0.60	1.42	0.42
10	-0.30, -0.20, 0.40	0.40, -0.40, 0.40	0.73	1.60	0.45
11	0.40, -0.40, 0.40	-0.40, -0.40, 0.40	0.80	1.50	0.53
12	-0.40, -0.40, 0.40	0.30, -0.30, 0.40	0.71	1.33	0.53

Table 5.3: Comparison of manipulator arm motion control methods.

Arm motion with 7 DOF	Arm motion with 3 DOF
Runs at 5 Hz	Runs above 30 Hz
Not easy to implement	Easy to implement
Performance is affected by inverse kinematics calculation rate	Performance is not affected by inverse kinematics calculation rate
Arm is more flexible with more DOF	Less flexible in comparison to 7 DOF
Inverse kinematics is calculated using numerical method	Inverse kinematics is calculated analytically
Not suitable for ball-catching application with available resource	Suitable for ball catching application
Suitable for low dynamic operations	Suitable for highly dynamic application

CHAPTER 6

Experimental Results

The purpose of this chapter is to present the experiments conducted, the principal results obtained and the major conclusions in terms of overall system's performance. First, to perform real experiments a calibration process to align the arm and the sensor is necessary and the implemented procedures are described in this chapter. Second, the vision and the robot arm systems are evaluated separately to measure the performance of the proposed solutions. Finally, a set of experiments are conducted using depth sensor raw-data (i.e., ball in real flight), taken from measurements in the real scenario, combined with a custom simulator (*MATLAB*®). The main goal is to evaluate the overall system with the integration of vision and robot motion in a simulation environment, but based on vision prerecorded data. From the very beginning of the project, it was assumed that more important than the rate of success with the real robot, it would be preferable to analyze and evaluate the conditions and range of open parameters that are necessary to perform a successful ball catching task. These open parameters are time of flight, maximum height attained by ball, initial velocity, inclination angle and distance of subject from the depth sensor.

6.1 Ball Catching Scenario and Calibration Process

To validate depth sensor data as a robust data, it is necessary to evaluate it using some ground truth data. For this dissertation work, system validation was performed using a Vicon system. Vicon system [15] [44] is best known for its accurate motion capture mechanism, where 8 infrared cameras are simultaneously capturing movement of an object through placed markers. For the purpose of comparison of depth sensor and Vicon data.

6.1.1 Ball Catching Scenario

An overview of all the elements of the ball catching experiment (robotic arm, depth sensor, subject launching the ball, in which the respective coordinate frames are represented) is presented in Figure 6.1. The distance between ground and the origin of robotic arm is 1.62 m. Depth sensor is placed just above robotic arm. It can be seen that, the robotic arm is in a hanging state.

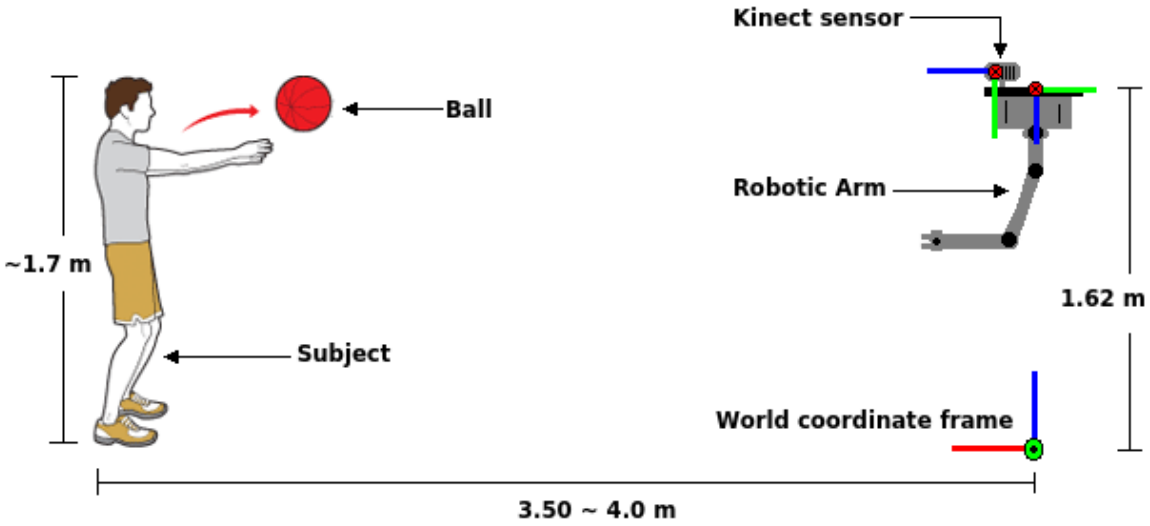


Figure 6.1: Ball catching scenario. Note: The 3D axes shown in RGB refer to XYZ in a sequential manner. In addition to that, the cross symbol denotes axis negative direction towards user, and dot symbol denotes axis positive direction towards user.

Below are presented some of the constraints and observations according to the implemented vision system using depth sensor.

- The prediction of the flying trajectory starts once the ball leaves the subject's hand (the ball must be isolated from any other object). This corresponds to an approximate distance of about 4.0 3.5 meter from the robot and depth sensor.
- The average flying time for several experiments conducted with the depth sensor is found to be 0.933 ± 0.107 second, considering throw from different angle and different height. This average time of flight is calculated using several experimental measurements obtained from implemented vision system.
- The average time period for which the depth sensor can see the ball from several experimental measurements found to be 0.682 ± 0.061 second, considering throw from different angle and different height.

6.1.2 Calibration Process

For the ball catching task, the Kinect-sensor and the robot arm must be calibrated, i.e., their positions and orientations must be well-characterized. The pose of the

camera must be known relative to the robot's arm in order to be able to perform the task precisely and to allow a rigorous off-line analysis as well. Not only the depth sensor has to be calibrated with respect to the manipulator arm, but the sensors, depth sensor and Vicon system, have to be calibrated to each other. Otherwise, the evaluation of the ball detection algorithm could fail when combining data from the two sensors for analysis purposes.

In robotics applications, different coordinate systems can be used to define where robots, sensors, and other objects are located. In general, the location of an object in 3-D space is defined by its position and orientation with respect to a certain frame of reference. There are multiple possible representations for these quantities, some of which are specific to certain applications. Translation and rotation are alternative terms for position and orientation. There are several methods that can used for calibration between different sensors or sensor and a robotic arm.

Transformation between one coordinate frame to another basically can be defined in two ways, 1) Using Translation and Rotation representation or 2) using Quaternions. For this particular dissertation work, two different systems that are to be calibrated to get transform from one to another or the other way around. These two different systems are Kinect-Vicon for the purpose of comparison and Kinect-Cyton for ball catching application. The two calibration process for these two systems are presented below:

Depth sensor-Vicon

The Vicon is one of the most accurate system for motion capture purposes and is used as ground truth to evaluate robustness of depth sensor data. In this case, to get the transformation matrix between depth sensor and Vicon system, three Vicon IR markers are placed on different parts of the depth sensor as shown in Figure 6.2.

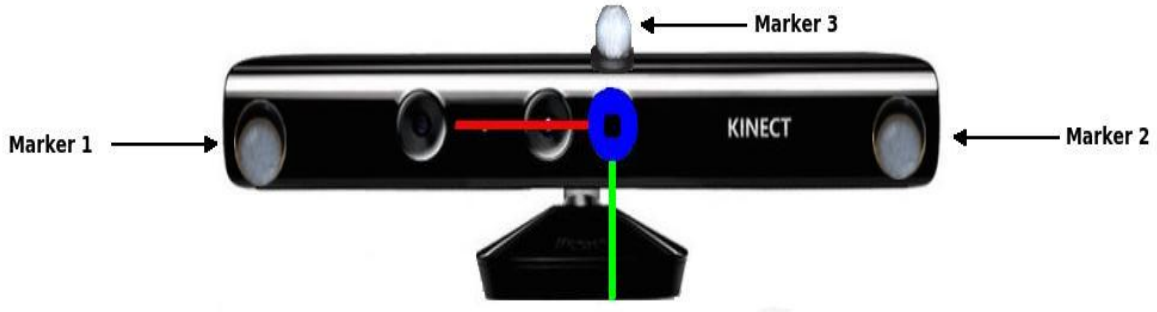


Figure 6.2: Markers over Kinect sensor for calibration. Note: The 3D axes shown in RGB refer to XYZ in a sequential manner. In addition to that, the cross symbol denotes axis negative direction towards user, and dot symbol denotes axis positive direction towards user.

Coordinates of these markers are obtained with respect to Vicon system frame of reference. This position and orientation subsequently used to get transform between

these two systems and are presented by expression 6.1 and 6.2.

$$VTK = \begin{bmatrix} 0.83141 & 0 & -0.55565 & 1.2421 \\ 0.55565 & 0 & 0.83141 & -1.4319 \\ 0 & -1 & 0 & 1.1009 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.1}$$

$$KTV = \begin{bmatrix} 0.83144 & 0.55562 & 0 & -0.23711 \\ 0 & 0 & -1 & 1.1009 \\ -0.55562 & 0.83144 & 0 & 1.8807 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.2}$$

Where, VTK is the transform matrix from Vicon frame to depth sensor frame and KTV is the transform matrix from depth sensor frame to Vicon frame. Table 6.1 presents an analysis of robustness of depth sensor data with respect to Vicon system measurements.

Table 6.1: Kinect sensor and Vicon system calibration analysis.

Test number	**Actual distance from Kinect (mm)**	**Distance given by Kinect (mm)**	**Error (mm) without Kinect correction (% Error)**	**Error (mm) with Kinect correction (% Error)**
01	2910.4	2882.8	110.83 (3.81)	27.597 (0.95)
02	3111.4	3072.2	124.6 (4.00)	39.241 (1.26)
03	3400	3362.4	129.57 (3.81)	37.606 (1.10)
04	2511.1	2491.5	92.469 (3.68)	19.587 (0.78)
05	2764.5	2736.3	110.15 (3.98)	28.132 (1.01)
06	3201.8	3151.3	141.39 (4.41)	50.479 (1.57)

From Table 6.1, it is evident that after enforcing depth sensor correction factor error in depth sensor data reduces significantly. Ball positions are recorded by placing the ball at different positions using Vicon IR marker. These positions are used for comparative analysis after using transform from depth sensor frame to Vicon system frame of reference. Ball position is recorded by both the systems simultaneously. These correction factors are defined as follows:

- Calculated center of ball is actually on the surface of ball and not exactly in the center of ball. This error can be reduced by adding an offset in depth data of depth sensor. This correction factor is used by adding the information of ball's radius to the depth sensor depth data.
- Depth sensor has an internal error of 2.5 cm per meter in z-axis of depth optical frame. This correction factor is enforced by adding a compensation factor of 2.5 cm per meter of the depth sensor depth data.

Kinect-Cyton

The coordinate frames of the depth sensor and the Robotic arm system for interception of the flying-ball trajectories must be calibrated so as to get transformation matrix from one frame to another. From Figure 6.3, it can seen that the depth sensor is placed just above robotic manipulator. To get transformation between depth sensor and robotic manipulator, approximation method (*i.e.*, by using a measurement tape) is used. By measuring the translations along all three axes and getting rotation around all three axes, transform is computed. Transform between these two frames are represented by expression 6.3 and 6.4.

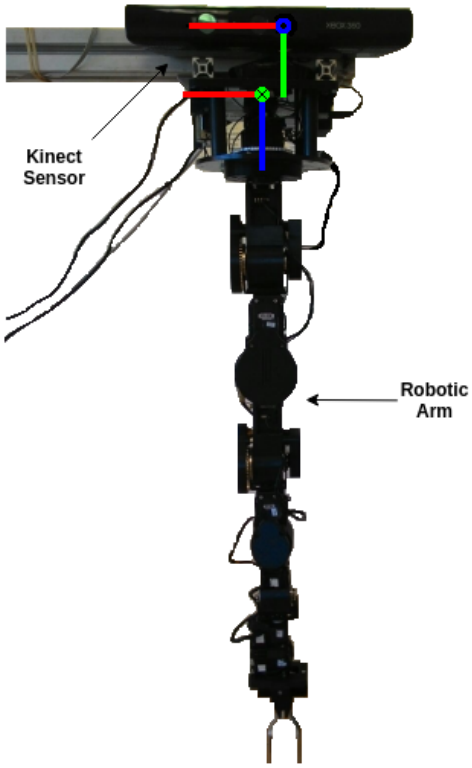


Figure 6.3: Depth sensor to Robotic Arm Transform. Note: The 3D axes shown in RGB refer to XYZ in a sequential manner. In addition to that, the cross symbol denotes axis negative direction towards user, and dot symbol denotes axis positive direction towards user.

$$KTR = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -0.065 \\ 0 & 1 & 0 & -0.055 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

$$RTK = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0.055 \\ 0 & 1 & 0 & -0.065 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.4}$$

Where, KTR is the transformation matrix from depth sensor frame to Robot frame as presented in expression 6.3, and RTK is the transformation matrix from Robot frame to Kinect frame as presented in expression 6.4. Coordinate frames for this transform are presented in Figure 6.3.

6.1.3 Comparative Analysis of Kinect and Vicon Data-set

After Kinect-Vicon calibration and considering Vicon as ground truth, a comparative analysis on trajectories given by depth sensor and Vicon system can be done. Vicon data is being recorded at a rate of 4 times to depth sensor rate. These data are synchronized to get exact position of ball with respect to both the systems. The synchronization is done in space and not in time.

For the purpose of synchronization both the systems (*i.e.*, depth sensor based application and Vicon system) are started at any instant (*i.e.*, not related to each other) and ball catching experiment data are recorded. To synchronize these data, before ball catching experiment, pendulum like swinging task is established. This swinging task helped greatly for synchronizing depth sensor and Vicon system data. Figure 6.4, presents the swing-state of ball, that is used for synchronization of depth sensor and Vicon system data. The peak position of ball from both the data sets are considered for synchronization by finding a proper delay.

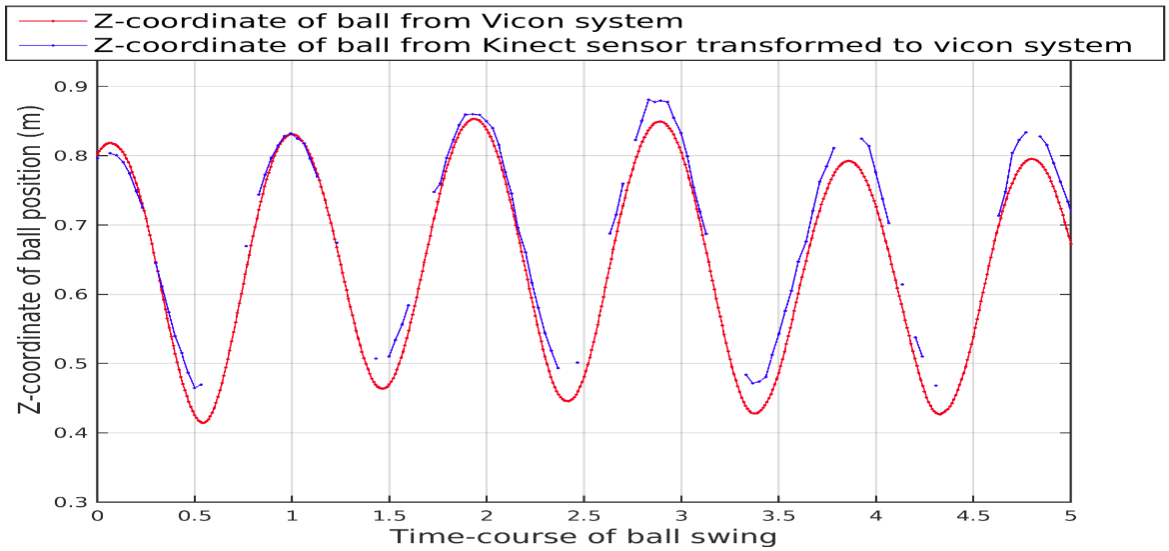


Figure 6.4: Synchronization of depth sensor based application and Vicon system.

Flying ball trajectories are recorded in both the systems simultaneously and is as shown in Figure 6.5. It can be seen that, the depth sensor based trajectory has less number of sample with respect to Vicon system and there is an offset-error, which is systematic in nature.

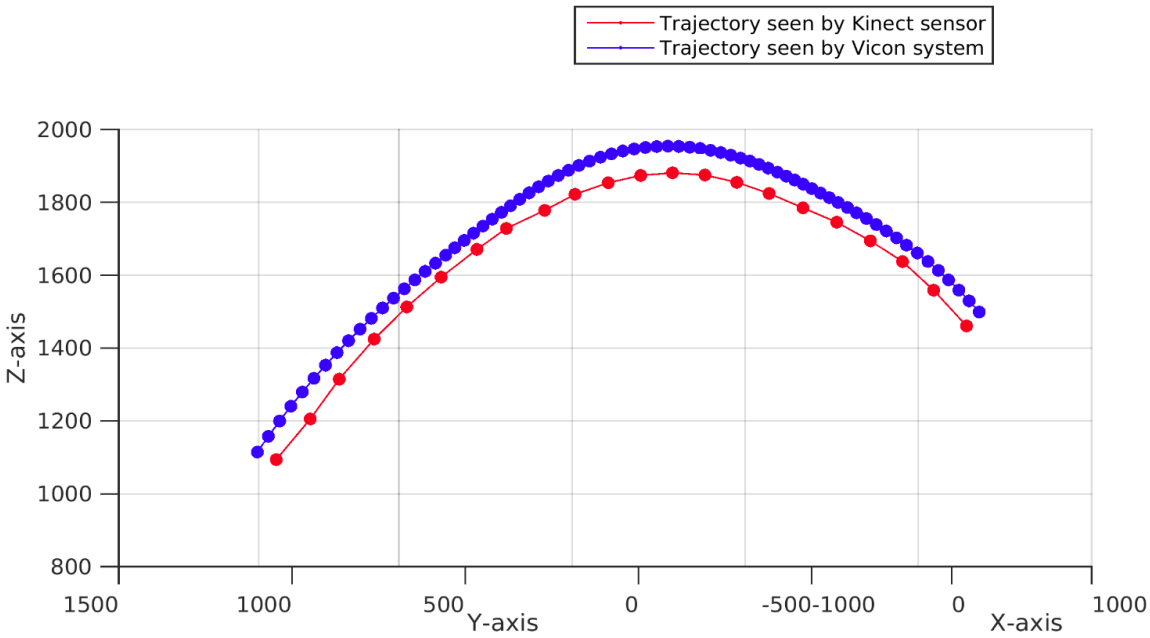


Figure 6.5: Ball trajectory seen by depth sensor and Vicon system.

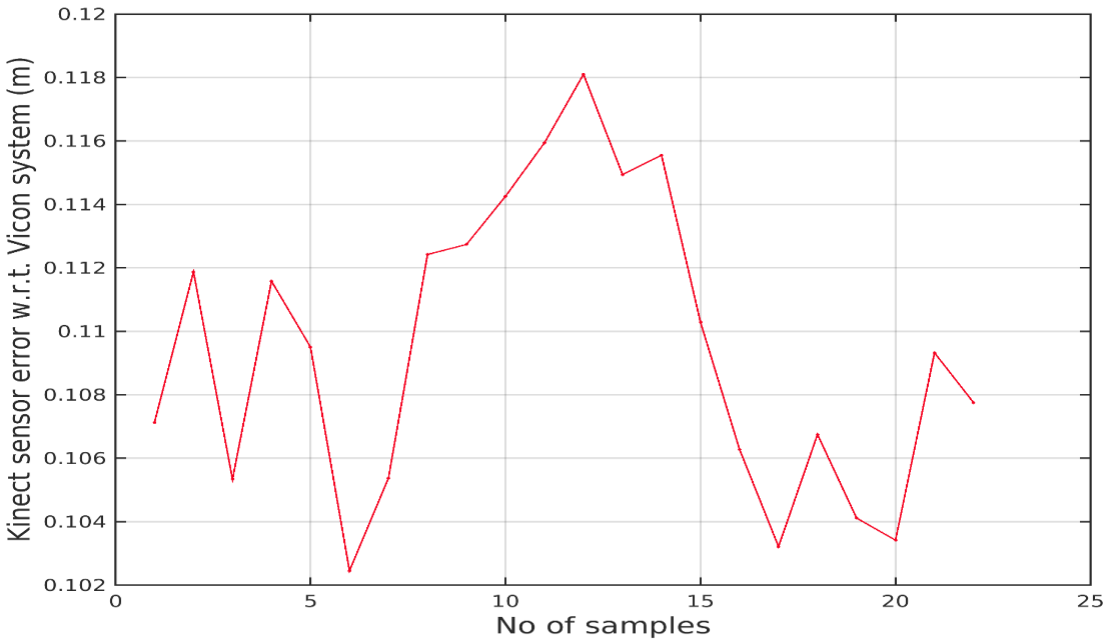


Figure 6.6: Error associated with depth sensor with respect to Vicon system.

The offset error associated with depth sensor data with respect to Vicon system can be seen in Figure 6.6. From Figure 6.5, it can be seen that the data from depth sensor is robust, but there is a systematic offset between trajectory seen by depth sensor and Vicon system. Associated error is 0.10947 ± 0.00463 meter, and it can be compensated in real-time. With all the available experimental data, it can be concluded that the vision system with depth sensor works fairly well. In addition to that, it is also observed that the number of measurements given by depth sensor is fair enough for trajectory estimation at an early stage (*i.e.*, when ball starts flying). From now on-wards depth sensor based application will be used for experimental analysis of ball trajectory estimation with two different methods stated earlier (*i.e.*, polynomial approximation, and Kalman filter based estimation).

6.2 Predicting the Ball Trajectory

For the purpose of ball trajectory and landing point prediction, two different methods are implemented (*i.e.*, polynomial approximation, and Kalman filter based estimation). These methods of prediction are to be analyzed according to experimental results. The analysis of estimation methods will be done using real data-sets that are recorded with implemented vision system. Following subsections contain some of the experimentally obtained results related to ball trajectory, time of flight and landing point estimation.

6.2.1 ROS Implementation (Nodes and Topics)

The application of ball trajectory prediction along with landing point over a certain plane is implemented in ROS platform, and the corresponding ROS graph is presented in Figure 6.7. This application is implemented using point cloud based ball detection and tracking method. The node detection_node is implemented for ball detection and tracking using point-cloud data. The node estimation_node receives ball coordinates from detection_node in real-time and estimates the trajectory using polynomial approximation or Kalman filter and then publishes estimated landing point. But, this estimated landing point is in depth sensor's reference frame. To transform this estimated point to robotic manipulator frame, transform_node takes this estimated landing point and transforms to robotic manipulator's reference frame using expression 6.3. The node cyton_EE takes this estimated point in robotic manipulator's reference frame and treats it as the goal position for ball catching task.

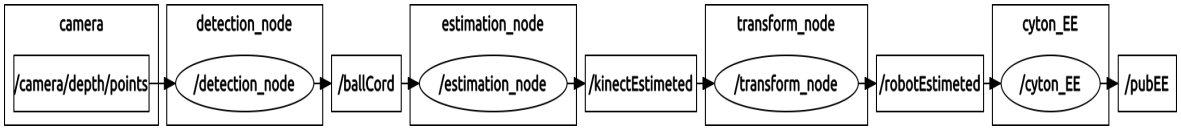


Figure 6.7: Ball trajectory prediction nodes and topics.

6.2.2 Evaluation of the Estimation Methods

As stated earlier, two estimation methods are implemented. A plane of fixed height is considered for ball's landing plane, where the robot will be trying to catch the ball. Figure 6.8 presents a basic visualization of real-world scenario, showing the landing plane of fixed height that is used to estimate ball's trajectory and it's landing point.

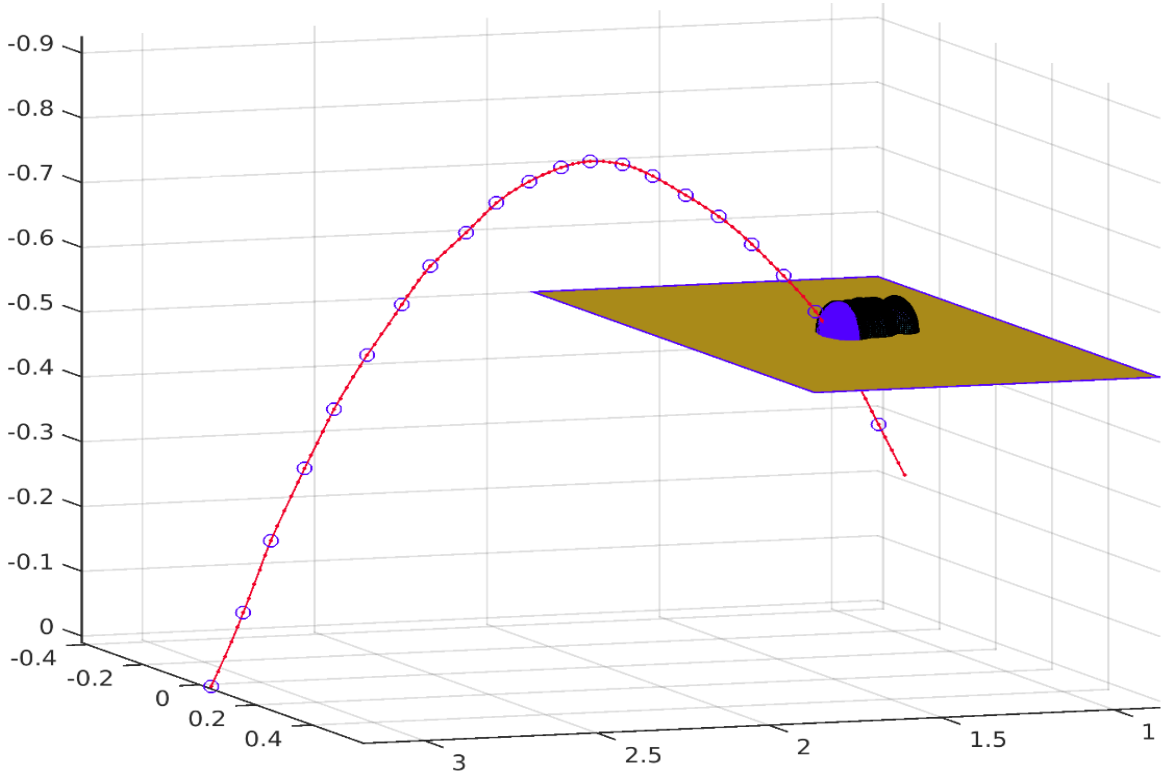


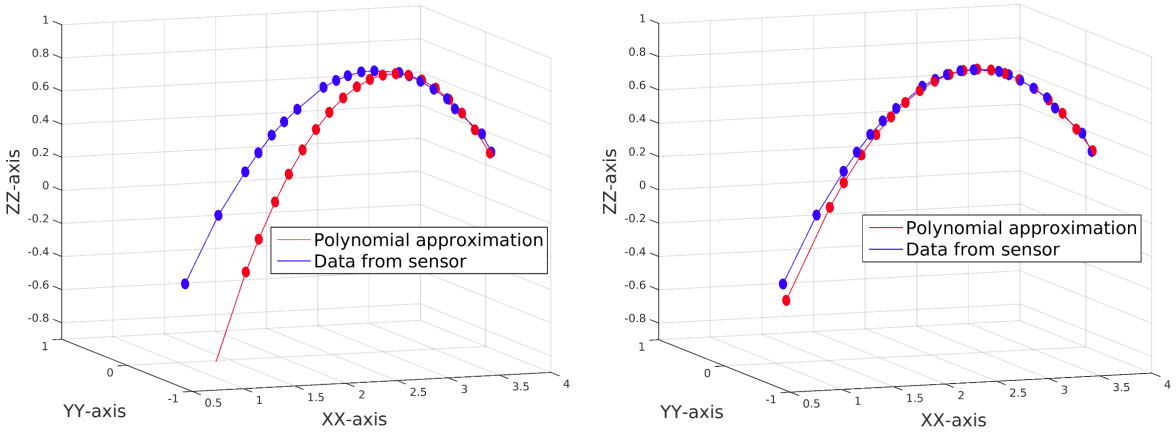
Figure 6.8: Intersection plane of trajectory.

It can be seen that, the estimated point over the landing plane changes. More details about estimation are explored in the following section.

Polynomial Approximation

Estimated trajectory from polynomial approximation can be seen in Figure 6.9. It can be seen that as the number of samples increases, estimation becomes better and better. But, this method takes more time to give a better estimation and leaves less time for actuation. This method solely depends upon few past samples, and if the samples are good enough then the estimation will be really good. But if past samples are not uniform then the estimation will not be good. Figure 6.9a and 6.9b, present estimated trajectory of ball from past 8 and 12 samples respectively.

Figure 6.10, shows the associated error with respect to polynomial approximation. A closer look at the Figure 6.10 gives better understanding about error associated to this method of trajectory estimation.



(a) Polynomial approximation with 8 samples. (b) Polynomial approximation with 12 samples.

Figure 6.9: Trajectory estimation with polynomial approximation.

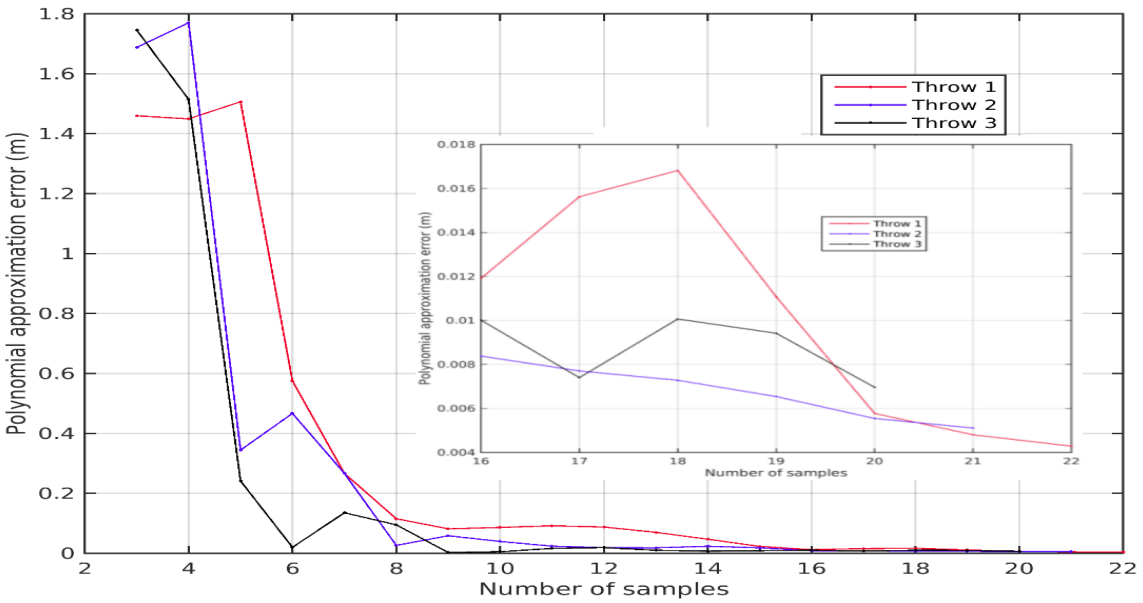
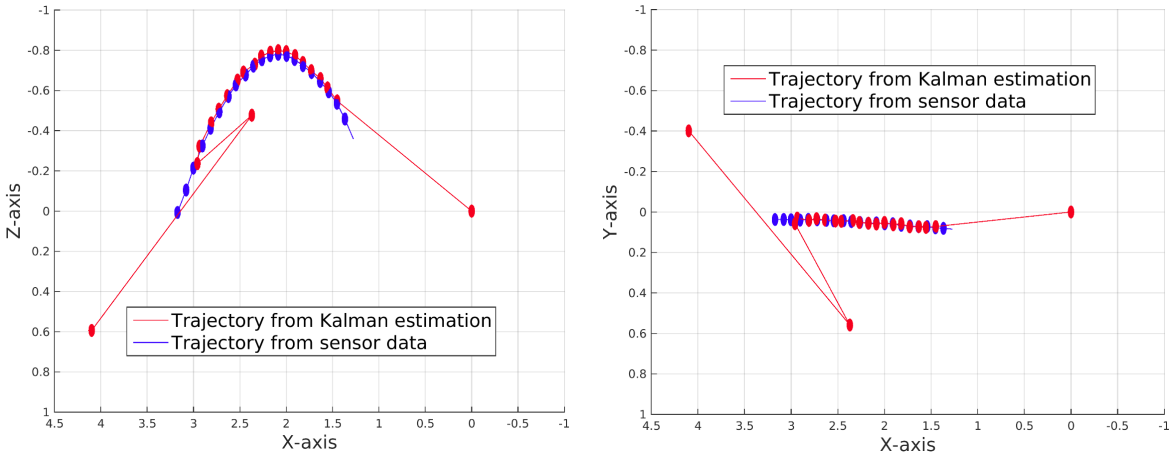


Figure 6.10: Error associated with polynomial approximation.

Kalman Filter Based Estimation

The main idea on which Kalman filter works [32] is, 1) error correction and 2) reduction of effect of noise over sensor measurement. This method of estimation works well if the model is well designed and states are initialized with proper values. Figure 6.11, shows estimation based on Kalman filter and it estimates next state with all the past information unless it is reset to initialization state. The next estimated state can be used to compute the landing point. From Figure 6.11, it is evident that the initial estimation is far away from real-trajectory. But, after addition of some more samples, the estimation becomes better.

Error associated with Kalman filter based estimation cab seen in Figure 6.12.



(a) Estimated trajectory with Kalman filter (side-view).

(b) Estimated trajectory with Kalman filter (top-view).

Figure 6.11: Trajectory estimation with Kalman filter.

From Figure 6.12, it can be seen that as more samples are added, associated error reduces almost exponentially but, at a faster rate. A closer look at the Figure 6.12 gives better understanding about error associated to this method of estimation.

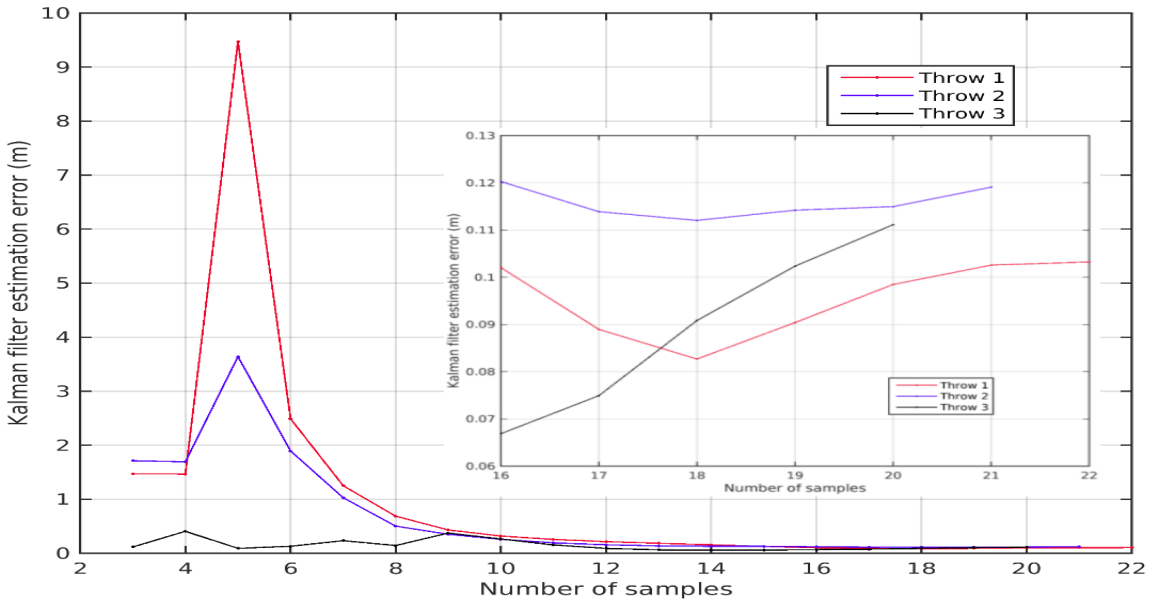


Figure 6.12: Error associated with Kalman filter based estimation.

Comparative Analysis of Polynomial Approximation and Kalman Filter Based Estimation

Considering above two estimation methods and in the scope of this dissertation work, there is another way of thinking is that, if a high performance system (*i.e.*, high acquisition frame rate, faster computation capability, and highly dynamic manipulator arm) is available, polynomial approximation based estimation can be used. In that case more time can be given for estimation and less time for actuation. Figure 6.13, shows a comparative look at the associated error with respect to Kalman filter based estimation, as well as polynomial approximation. With Kalman filter, next state can be estimated and then corrective action can be taken, which is not considered in the case of polynomial approximation.

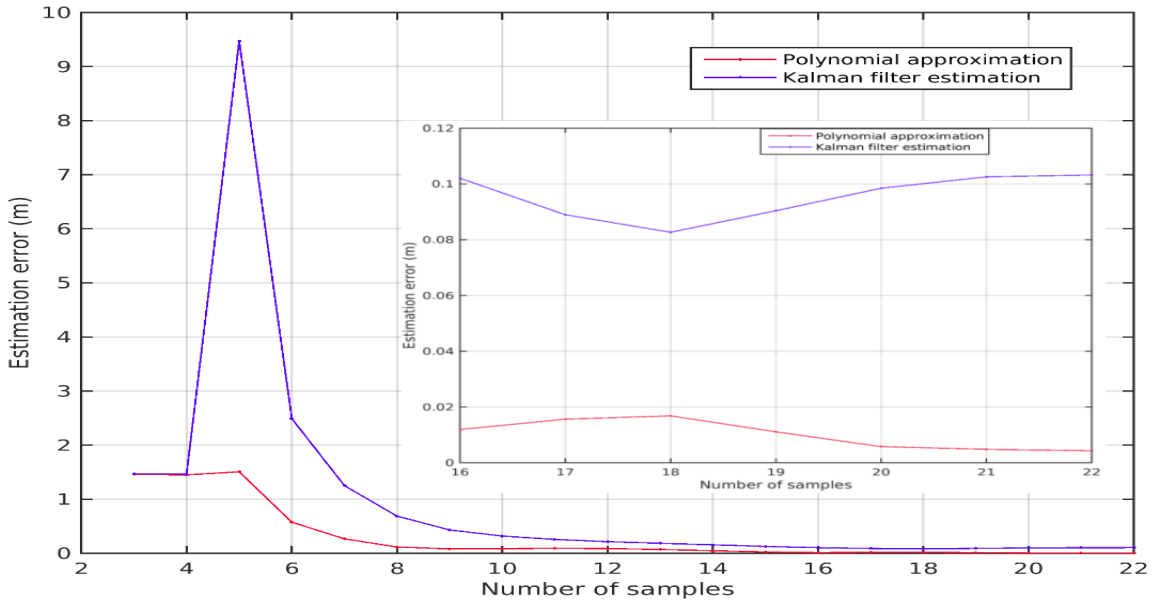


Figure 6.13: Comparative error analysis of estimation methods.

According to the error analysis results, it can be concluded that, both the estimation methods works fairly well. But, a comparative analysis will provide more detailed conclusion. From Figure 6.13, it can be seen that the estimation error reduces with time almost exponentially as more samples are added for both the methods. There is one observation worth of attention is that Kalman filter based estimation has more error that polynomial approximation method. Below are presented some conclusions related to experimental observation of estimation methods.

1. Polynomial approximation method works fairly well compared to Kalman filter based estimation, if and only if past sample vales are considered and not other factors such as: error correction, sensor imperfection, noise in sensor measurements. Its worth of mention that, this comparison is not related to real ball trajectory but, related to depth sensor trajectory.

2. On the other hand, Kalman filter works well but, the error in estimation is significantly higher than polynomial approximation method, when comparison is not related to real ball trajectory but, related to depth sensor trajectory.

3. From 6.12, it is evident that the error associated with Kalman filter is approximately 10 cm in all the observations. But, from Figure 6.6, it can be seen that the error associated with depth sensor with respect to Vicon system is also 0.109 ± 0.005 meter. This means that the Kalman filter based estimation method compensates noisy measurements given by depth sensor.

4. From above observations, it can be concluded that the Kalman filter based estimation works better than polynomial approximation. Kalman filter based estimation gives real position of ball, by compensating noise and other sensory imperfections.

5. By doing a fine tune of co-variance matrices, Kalman filter model can be used without any compensation and can be termed as a reliable method of estimation over polynomial approximation. A fine tuned model will reduce the estimation time and will give exact estimation of ball's landing point at an early stage of estimation process.

6.3 Computer Simulation of the Ball Catching Task

Given the high complexity of the ball catching task, a computer simulation that attempts to model the real situation is an invaluable tool for analyzing how the system works and changes behavior according a set of parameters. Rapid testing, validation and performance optimization can be carried out in simulation before any implementation in the real hardware. To address this need, a ball-catching simulator that represents the operation of the overall system over time was developed in MATLAB®. This simulator integrates the kinematic models of the Cyton manipulator arm with real data-sets of the depth sensor previously recorded. The focus of this section is on the description of the computer experiments conducted and the insights gained into the system's functioning. It shows the effects of alternative conditions and courses of action, as well as strategies for successful ball catching.

6.3.1 Spatiotemporal Conditions for Successful Ball Catching

Catching a flying ball is a behavioral task with time constraints for a successful completion, requiring a trade-off between the time allocated to perception and action. In general, a longer perception may reduce the uncertainty in perceptual estimates. However, a longer perception phase leaves less time for action, which results in less precise movements. This subsection aims to analyze the conditions for a successful catching by changing Spatiotemporal parameters and assuming the maximum attainable velocity in the robot's joints. More specifically, this study focuses on the

impact of the trade-off between the amount of time allocated to perception and the amount of time remaining for action, assuming ideal parabolic trajectories generated by the computer.

Work-space

The analysis of trade-off between the time allocated to perception and action can only be done for a specific work-space where the manipulator arm will be operating. Figure 6.14, shows the work-space considered for this analysis, at a particular plane (*i.e.*, 0.45 meter in manipulator arm's z-axis). The blue portion represents the near by area of manipulator arm's end-effector and can be considered as the easily accessible region for manipulator arm in a limited available time.

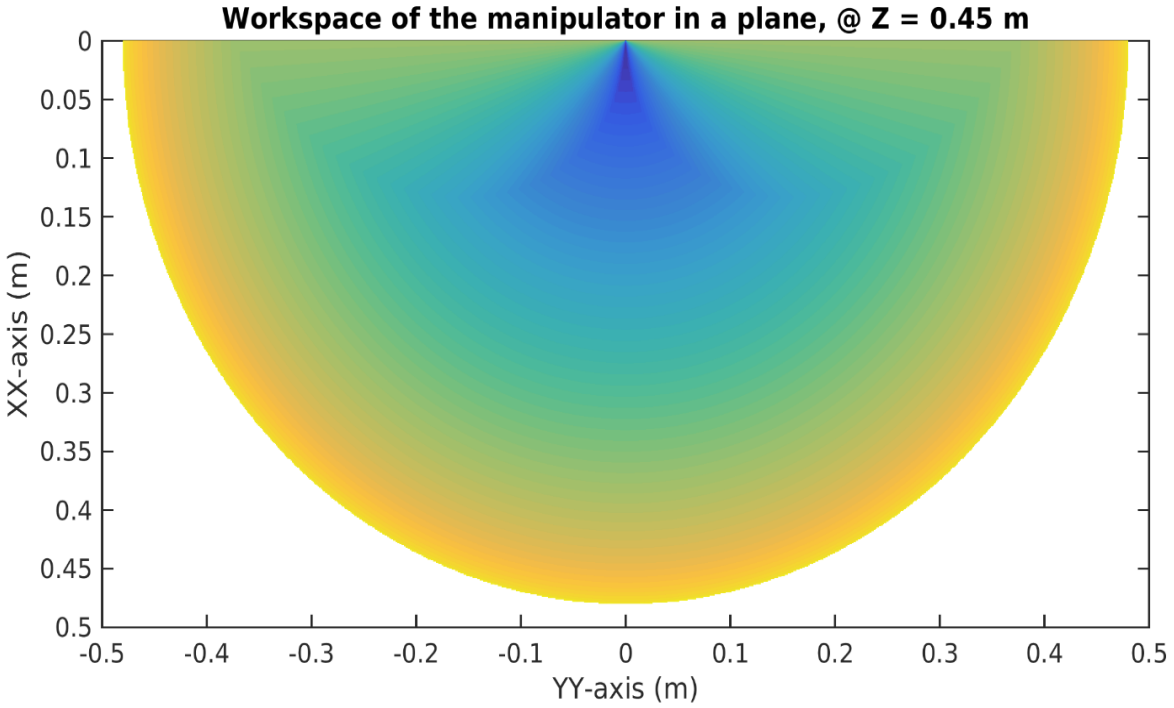
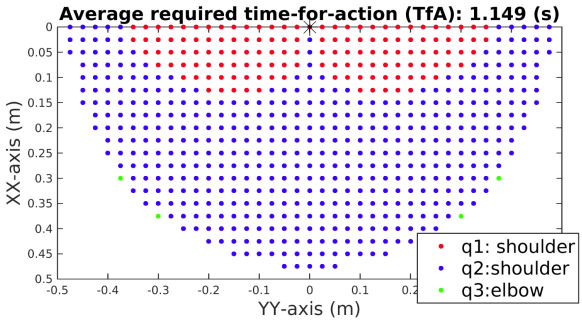


Figure 6.14: Work-space for spatiotemporal analysis, this a particular case where the end-effector of manipulator is placed at (0.00, 0.00, 0.45).

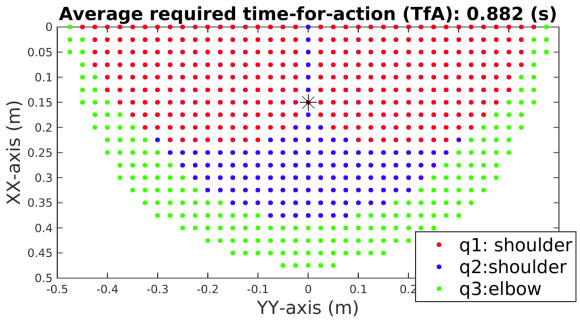
Analysis of Joint Limiting the Time-for-action (TfA)

The robot's joint limiting the time-for-action (TfA), when the end-effector is required to move from a predefined initial position to the target positions defined by a grid of uniformly distributed points inside the work-space (the horizontal plane is defined for a height above the ground of 1.17 meter) can be analyzed considering different cases.

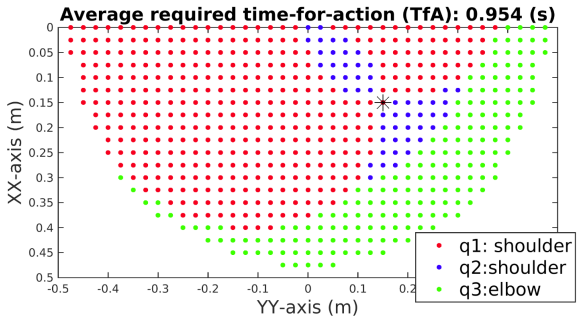
Figure 6.15, presents the joint limiting the time-for-action (TfA), where the different colored portion represent different joint of manipulator arm (*i.e.*, red colored



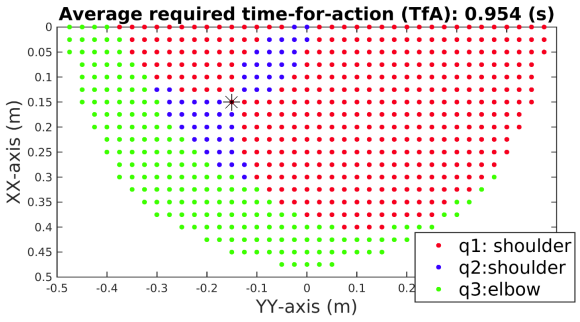
(a) End-effector initial position is at (0.00, 0.00, 0.45).



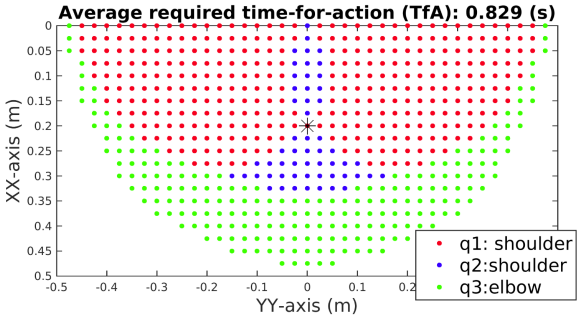
(b) End-effector initial position is at (0.15, 0.00, 0.45).



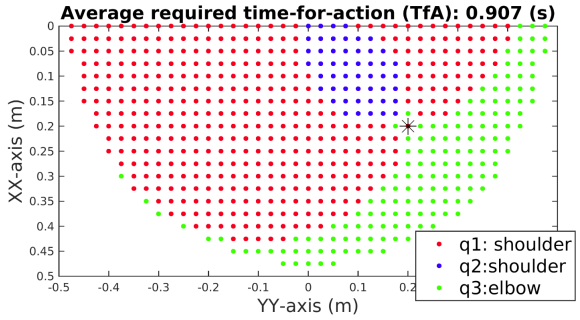
(c) End-effector initial position is at (0.15, 0.15, 0.45).



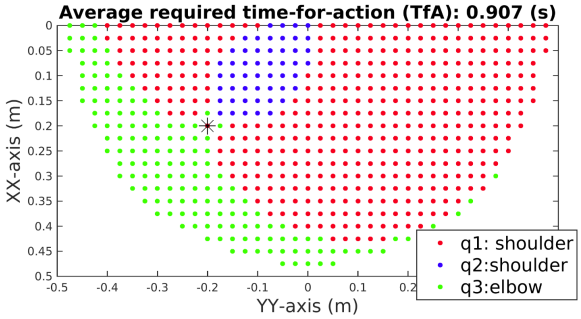
(d) End-effector initial position is at (0.15, -0.15, 0.45).



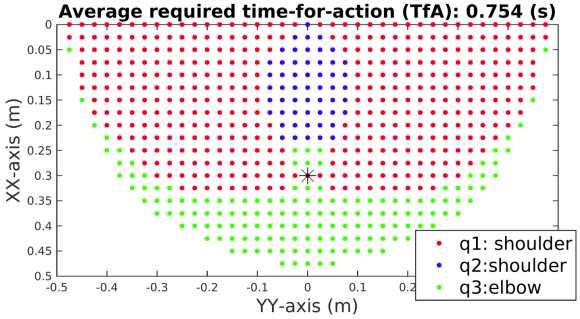
(e) End-effector initial position is at (0.20, 0.00, 0.45).



(f) End-effector initial position is at (0.20, 0.20, 0.45).



(g) End-effector initial position is at (0.20, -0.20, 0.45).



(h) End-effector initial position is at (0.30, 0.00, 0.45).

Figure 6.15: Representation of the robot's joint limiting the time-for-action (TfA).

area for shoulder roll joint, blue colored area for shoulder pitch joint, and green colored area for elbow joint,) in terms of prominent joint for successful catching. The blue dots signifies that, if the estimated ball position is in those area, then the joint 2 (*i.e.*, shoulder pitch joint) requires maximum time for successful catching. Similarly, joint 1 and joint 3 will be taking maximum time, if the predicted ball position lies in red and green area for successful catching respectively. Sub-figures in Figure 6.15, represent different joint limiting condition for different initial end-effector position. It can be seen from these figures that, the allocated area for joints are different for all the cases except mirror positions about $y = 0$ axis.

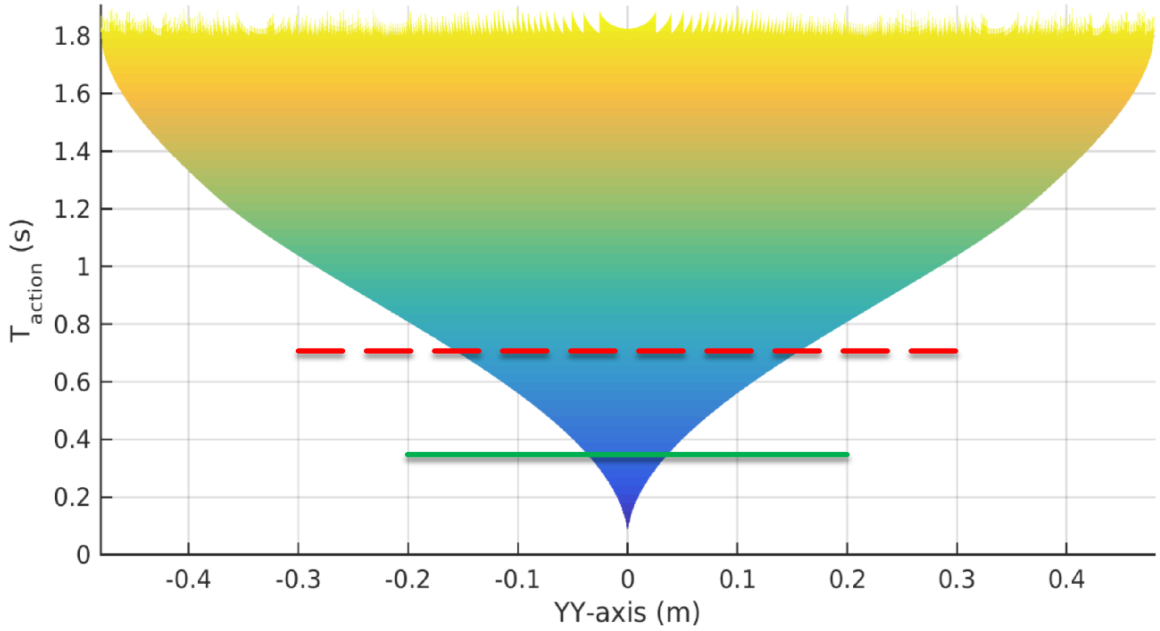
Table 6.2, shows the overall analysis related to time-for-action (TfA), considering different initial end-effector position. It can seen that the $T_{\text{mean action}}$ (s) decreases proportionally with respect to distance from $(0.0, 0.0, 0.45)$. It is because of the reduction in distance to be covered by end-effector on an average basis to any goal position inside work-space.

Table 6.2: Analysis of time-for-action (TfA) with respect to initial end-effector position.

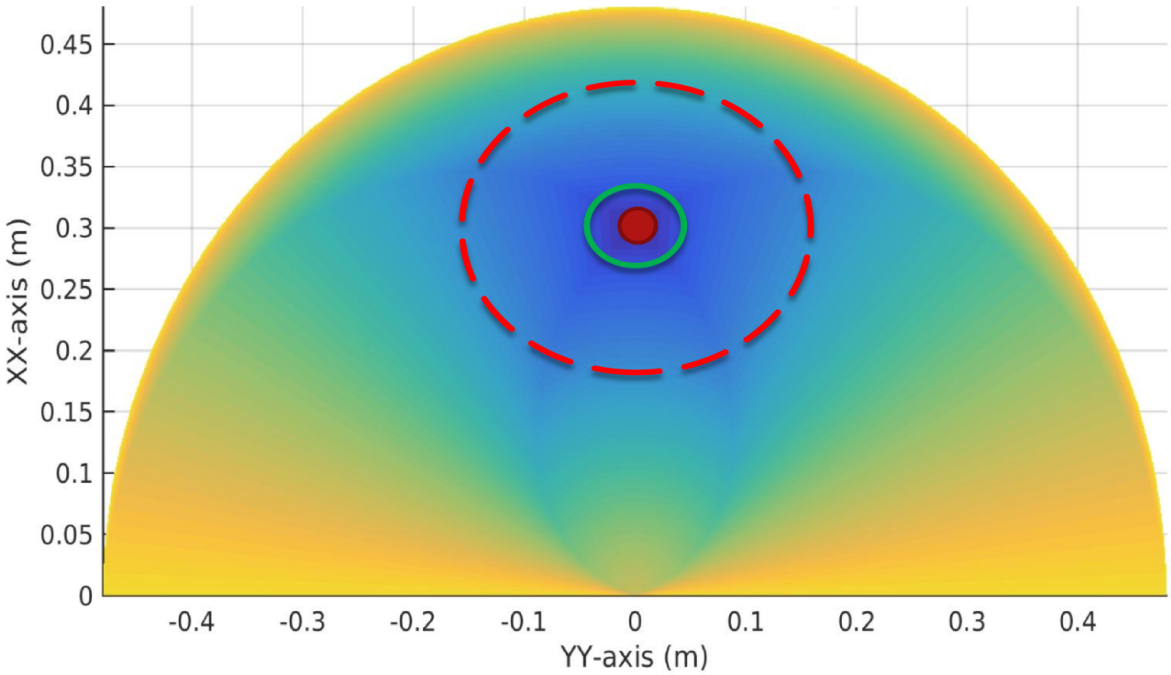
End-effector initial position	**Distance (m) from** $(0.0, 0.0, 0.45)$	$T_{\textbf{mean action}}$ **(s)**	$T_{\textbf{mean action left}}$ **(s)**	$T_{\textbf{mean action right}}$ **(s)**
0.00, 0.00, 0.45	0.0	1.1489	1.1493	1.1493
0.15, 0.00, 0.45	0.15	0.8825	0.8830	0.8830
0.15, 0.15, 0.45	0.2121	0.9542	1.2681	0.6408
0.15, -0.15, 0.45	0.2121	0.9542	0.6408	1.2681
0.20, 0.00, 0.45	0.20	0.8295	0.8299	0.8299
0.20, 0.20, 0.45	0.2828	0.9065	1.2524	0.5612
0.20, -0.20, 0.45	0.2828	0.9065	0.5612	1.2524
0.30, 0.00, 0.45	0.30	0.7536	0.7539	0.7539

The time $T_{\text{mean action}}$ (s), signifies the average time taken by the manipulator arm to reach any point inside work-space. Whereas, $T_{\text{mean action left}}$ (s) and $T_{\text{mean action right}}$ (s), denotes the average time required by the manipulator to move the end-effector from initial position to reach any of the point in left side and right side of work-space respectively.

The Figure 6.16, presents the the robot's capability in terms of time-for-action (TfA), when the end-effector is required to move from the initial position (*i.e.*, this is a particular case with end-effector initial position at $(0.30, 0.0, 0.45)$) to the predicted goal position. It can be seen that, the analysis of time-for-action (TfA) is in 2 dimensional space, assuming the manipulator need to operate over a fixed plane.



(a) Time-for-action (TfA) Vs displacement (side-view). The red dotted line signifies the maximum time available for action, if time of flight is completely given for action to catch a flying ball. The green line signifies the time available for action, assuming half of the time of flight is given for perception to catch a flying ball.



(b) Work-space for Time-for-action (TfA) Vs displacement (top-view). The red dotted circle signifies the area that can be covered by manipulator, if time of flight is completely given for action to catch a flying ball. The green circle signifies the time available for action, assuming half of the time of flight is given for perception to catch a flying ball. The red dot signifies initial end-effector position.

Figure 6.16: Available time-for-action (TfA) Vs displacement.

Figure 6.16a, presents the side view of time-for-action (TfA) analysis. Where as Figure 6.16b, shows the top view of relation between TfA and displacement by end-effector of manipulator arm. This shows that the time-for-action (TfA) is inversely proportional to the displacement made by end-effector. This is done by trajectory planning, in order to make the end-effector follow the liner path to reach the desired position. If there is more time available for action then the end-effector can travel more distance. Considering the average time of flight described in section 6.1.1 as 0.9335 ± 0.10696 second, it is very difficult for this manipulator arm to cover a distance more than 20 cm. There are exceptions in every case and in this will be covered in next section of this chapter. It must be noted that, there is time for perception and action, and if all the time is left for action then there is no time left for perception. For successful catching, there must be a trade-off between perception and action time.

Impact of Maximum Height Attained by Ball Over Time-for-action (TfA) and End-effector Displacement

As mentioned in section 4.1, that the maximum height attained by ball affects time-of-flight (ToF). This subsection is about an overall analysis of relation between maximum height attained by ball and it's time-of-flight (ToF).

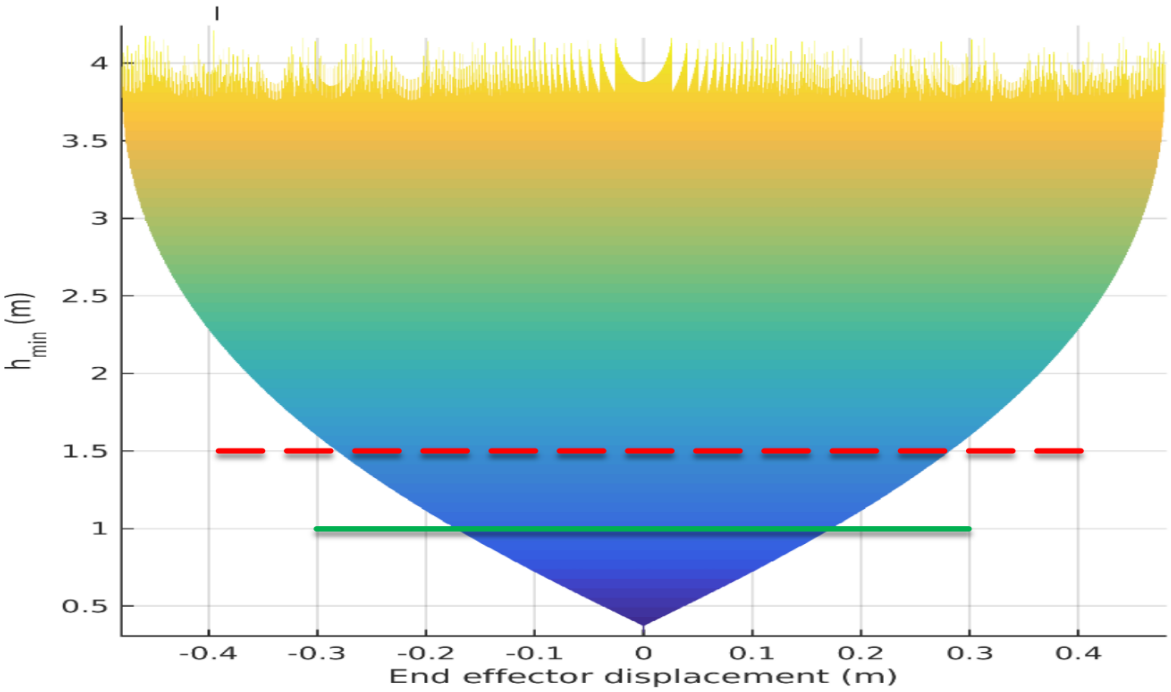


Figure 6.17: Minimum required height for successful displacement of end-effector (side-view). The red dotted line signifies the maximum time available for action, if time of flight is completely given for action to catch a flying ball. This is considered taking experimental setup in to consideration. But, 1 m height is found to be optimal for proper throw.

The Figure 6.17, presents the the effect of maximum height attained by ball during flight over the time-for-action (TfA) and time-for-perception (TfP), when the end-effector is required to move from the initial position to the predicted goal position.

In this case an analysis is done to validate the condition of minimum height required for successful movement of end-effector from initial position (*i.e.*, this is a particular case with end-effector initial position at $(0.30, 0.0, 0.45)$) to the predicted goal position. Of course the height attained by ball decides time available for perception and action. It can be said that, if the ball travels with more height, it has higher time of flight and accordingly the end-effector can make larger displacement. This shows that the time-for-action (TfA) and time-for-perception (TfP) together depends directly over maximum height attained by ball. This can be understood in a different way that, if the ball attains a higher position during flight, it will have longer time of flight. Longer time of flight will be helpful for the perception and action process. Hence, if there is more time available for action then, the end-effector can travel more distance. Form Figure 6.17, it can be seen that the end-effector can only displace 25 cm approximately considering the available experimental setup with depth sensor. Figure 6.18, presents top view of work-space for height related analysis.

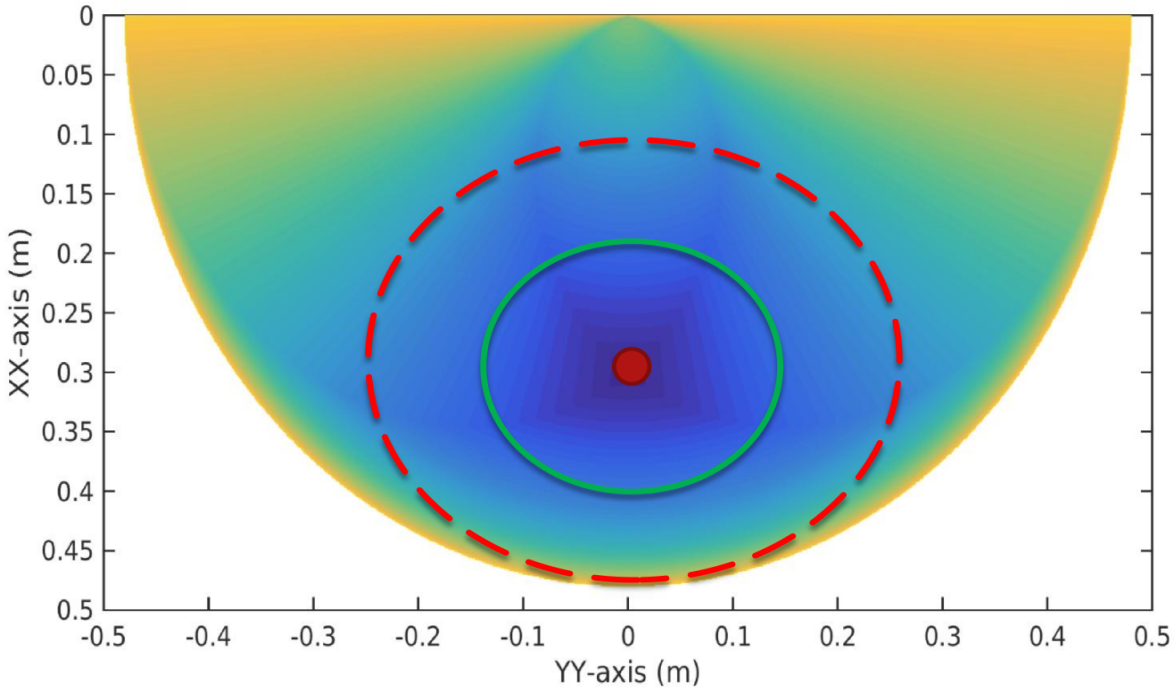


Figure 6.18: Work-space for Minimum required height for successful displacement of end-effector (bottom-view). The red dotted circle signifies the area that the manipulator can cover, if time of flight is completely given for action to catch a flying ball. The green circle signifies the optimal area that the manipulator can cover considering a proper throw. The red dot signifies initial end-effector position.

Impact of Number of Kinect Samples Over Time-for-action (TfA) and End-effector Displacement

In line with the analysis of time-for-action (TfA), minimum height required for successful catching and possible end-effector displacement, it is also necessary to analyze the effect of number of samples over time-for-action (TfA) and end-effector displacement. Figure 6.19, presents the the robot's overall analysis of relation between number of depth sensor samples and end-effector displacement, when the end-effector is required to move from the initial position (*i.e.*, this is a particular case with end-effector initial position at $(0.30, 0.0, 0.45)$) to the predicted goal position.

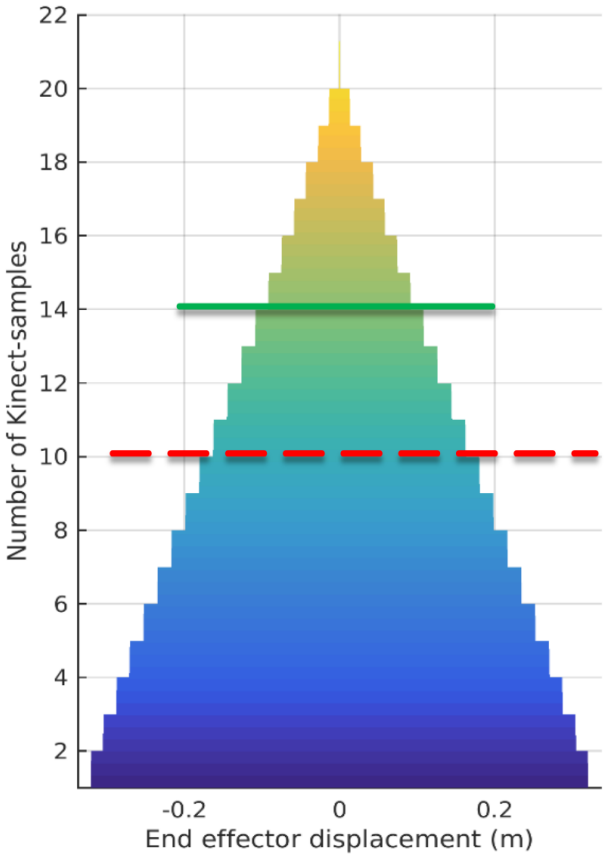


Figure 6.19: Number of Kinect sample Vs displacement (side-view). The red dotted line signifies the minimum number of Kinect sample required to get a proper estimation. But, 14 or more number of sample is found to be optimal which is denoted by green line.

From Figure 6.19, it is evident that the end-effector can cover more distance, if the manipulator arm receives a command to move to the predicted position with less samples. Again, there is conflict between time-for-action (TfA) and time-for-perception (TfP). This conflict can only be solved if and only if time-for-action (TfA) and time-for-perception (TfP) together is less than the time of flight (ToF), in order to achieve a successful catching task. Figure 6.20, presents the analysis of displacement

in a fixed plane related to the number of samples. If more time is spent on estimation then, there is less time left for action.

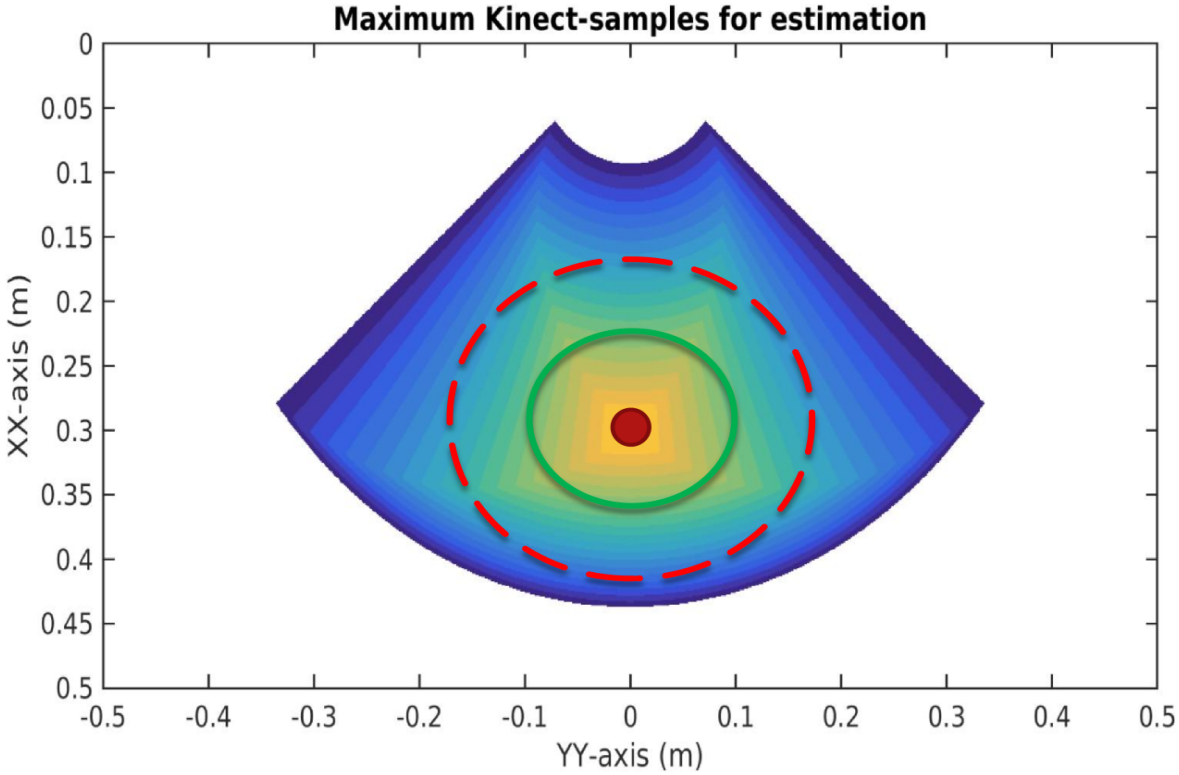


Figure 6.20: Number of Kinect sample Vs displacement (top-view). The red dotted circle signifies the area that the manipulator can cover considering an estimation with 10 samples. The green circle signifies the area that the manipulator can cover considering an estimation with 14 samples. The red dot signifies initial end-effector position.

6.3.2 Simulated Robot Motion Based on Real Vision Data

The preceding subsection presented a general study focusing on conditions for successful ball catching. The study included the evaluation of three performance measures: (i) the time required to move the arm's end-effector to a grid of reachable target points using the maximum joint velocities (time-for-action, TfA), (ii) the required maximum height that the ball must reach to ensure proper time of flight for a given time-for-perception (TfP) and (iii) the available number of Kinect samples for trajectory estimation for a given time-of-flight (ToF). Here, time-for-perception is defined as the total time for which the difference between two successive estimates of the catching point is less than a given threshold. All these measures were computed assuming that the ball catching task requires trading-off the time allocated to perception and action and, most importantly, that only one is possible at a given instant. This means that the more task time is spent sensing, the less task time remains

for robot's movement.

In this subsection, a ball catching task is simulated to demonstrate a specific case in which success is achieved (empirical validation), but considering that moving the robot's end-effector is possible during the perceptual estimates. Two additional assumptions are imposed in the computer experiment: First, it is assumed that the height of the catching plane (i.e., the horizontal plane where the intersection between the ball and the arm's end-effector should occur) is the same as the ball's initial height (i.e., the first sample taken from the depth sensor). Second, the initial position of the robot's end-effector is given by the (x, y, z) coordinates, (0.2, 0.0, 1.0) (m). The trajectory of the flying ball was selected from a large set of throws recorded with the depth sensor in the real scenario with random initial positions and velocities. An off-line data analysis revealed average values for the time-of-flight and time-for-perception (TfP) using Kalman estimation.

After the ball is launched, the robot's end-effector moves immediately to a new target as the prediction of the final catching point improves over time. This intermediate target is selected assuming that a good estimate about the direction of the horizontal component of the velocity vector (i.e., the vertical plane where the ball travels) can be obtained initially, for example, based on the hand's motion of the subject launching the ball. More specifically, the end-effector of the robot moves to the target point that minimizes the time-for-action (TfA), from the set of points belonging to the intersection line of the vertical plane with the work-space area. Here, additive noise was added to this estimate associated with noise and measurement errors.

Then, the predicted catching point is fed as the final target to the robot motion module when the difference between two successive estimations is less than 2 cm. Even so, the robot motion is not activated in two circumstances. The first occurs whenever the target end-effector position lies outside the robot's reachable volume. Second, it is possible that the robot cannot reach the target at the desired time. Given the initial arm configuration and taking into account the Cyton arm's physical limits (i.e., maximum joint velocities and accelerations), the inverse kinematics algorithm solution may allow estimating the time needed to bring the end-effector to the target.

Figure 6.21 shows the simulation results in the form of three snapshots of the robot and ball motions. The snapshots are taken for the initial arm configuration and ball location (left), at the instant in which the perception system provides the estimation of the catching point (middle) and at the final catching configuration (right). Figure 6.22 illustrates the time evolution of the robot's joints, marking the instants in which the catching point is estimated and that when the robot successfully caught the ball.

It is worth mentioning that robot's joints move with the maximum attainable velocity, but synchronous motion is not required. Finally, the robot's end-effector trajectory according to ball measurements and estimation of catching point is shown in Figure 6.23.

The estimation of the flying ball can only start when the ball is visible and ends

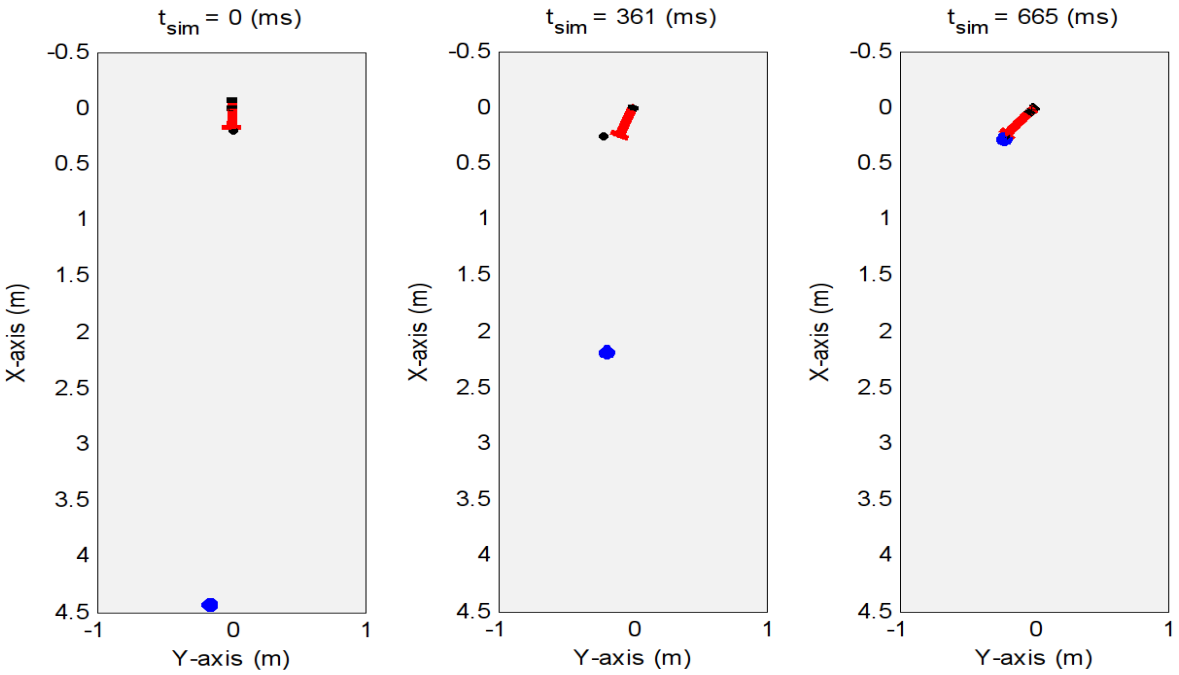


Figure 6.21: Snapshots of the robot and ball motions taken from the simulator: initial arm configurations and ball location (left); instant at which the perception system provides an estimation of the catching point (middle); final catching configuration corresponding to the (x, y, z) coordinates, (0.25,- 0.22, 1.0) (m) (right)

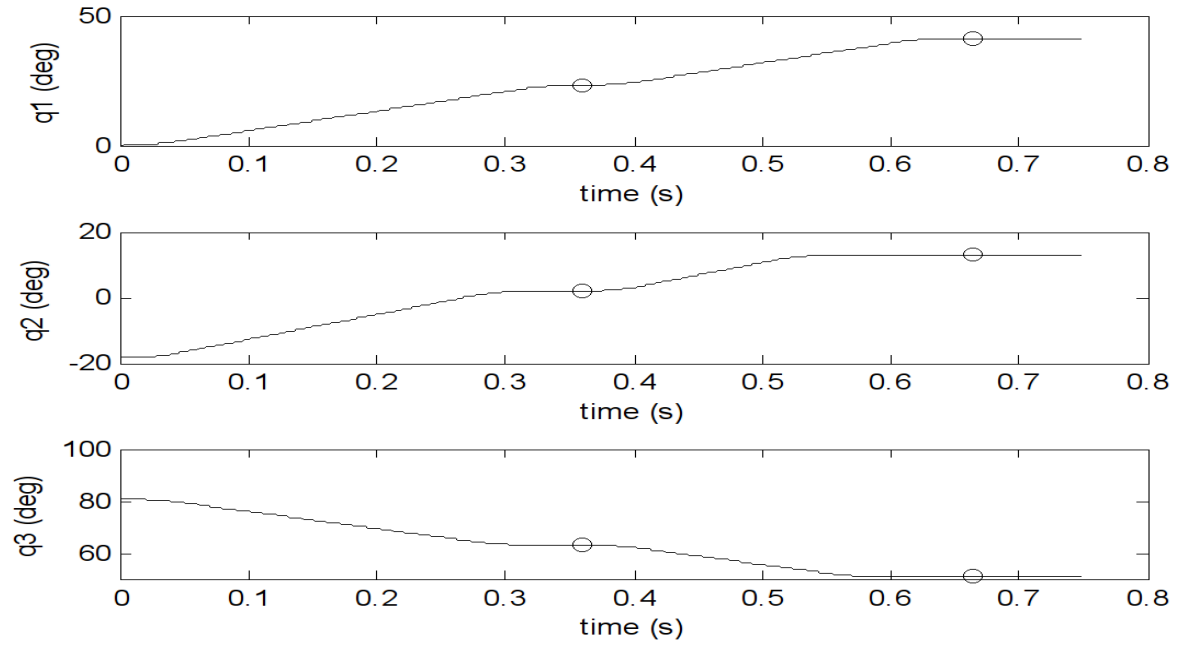


Figure 6.22: Time courses of the joint angular displacements, marking the instants in which the catching point is estimated and that when the robot successfully caught the ball.

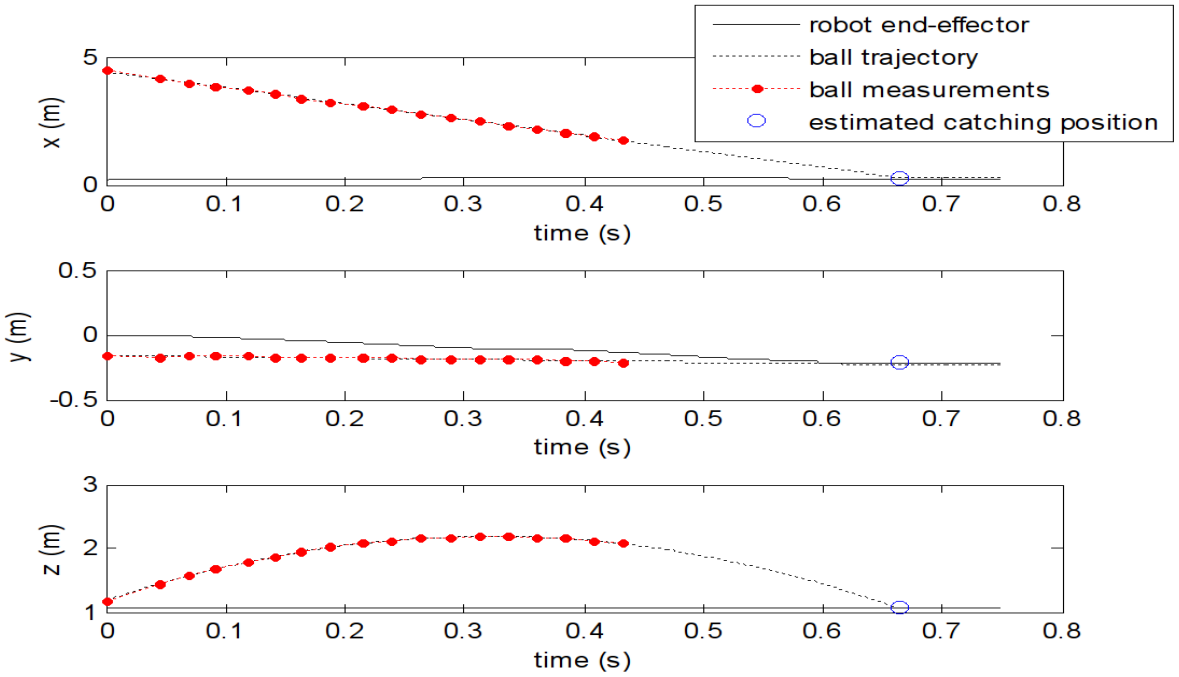


Figure 6.23: Trajectories generated by the robot's end-effector in response to the estimated catch point.

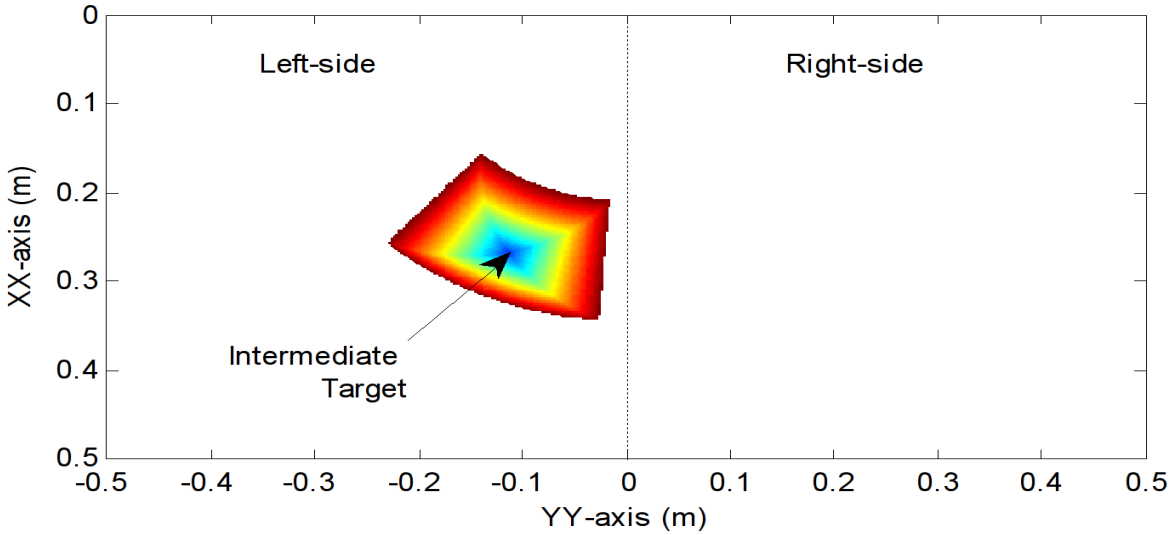


Figure 6.24: Reachable space where the Cyton arm is able to intercept the flying ball if the end-effector is located at the intermediate target when the prediction of the catching point and time is available.

when the ball leaves the volume (*i.e.*, voxel grid) previously defined. This corresponds to distance about 3.0 m from the robot. As a final evaluation of the possible success rate when using this strategy for planning the catching motion, it was calculated the total area, measured in the catching plane, where the ball could land and

the robot is able to intercept the flying ball. Given the intermediate target calculated in this particular case, the area is about 10 % of the total available work-space area, when considering only the left-side, as shown in Figure 6.24.

6.4 Final Remarks

The preceding sections presented an experimental analysis of studied methods for ball catching application. This includes calibration process, trajectory estimation, spatiotemporal conditions for successful ball catching, and computer based simulation of ball catching task for feasibility test. In line with the studied topics, some observations are presented below.

- Calibration of depth sensor and Vicon system was necessary for validation of measurements given by depth sensor, in order to explore the sensory imperfections. In addition to that, calibration of depth sensor and Cyton Gamma 1500 manipulator arm is also necessary for integration of developed vision system and manipulator arm motion control system. This gave an insight to depth sensor and implemented vision system algorithm accuracy.
- A comparative analysis of estimation methods was necessary for selection of better one among them, in order to get better estimation in real-time. This analysis is also carried out considering sensory noise and imperfections. Kalman filter based estimation gave better result compared to polynomial approximation method.
- For successful ball catching, it was necessary to study the spatiotemporal conditions for this work. It is worth of mention that, this study provided some result about suitable conditions for successful ball catching task. It was also an important aspect to consider how (time-for-action, TfA), time-for-perception (TfP), and time-of-flight (ToF) are related or should be related for successful ball catching task. The resulted condition is that, there should be enough time left from time-of-flight (ToF) for (time-for-action, TfA) after spending some time for perception.
- Computer based simulation scenario is used for validation of developed systems, considering manipulator arm constraints and taking real-data from depth sensor. It was observed that, successful ball catching is possible but only in some cases.

Integrated System for Real-time Ball Catching Task

The integrated application of ball catching task is implemented in ROS platform and corresponding ROS graph is presented in Figure 6.25. This an integrated application and consists both vision system and arm motion control applications.

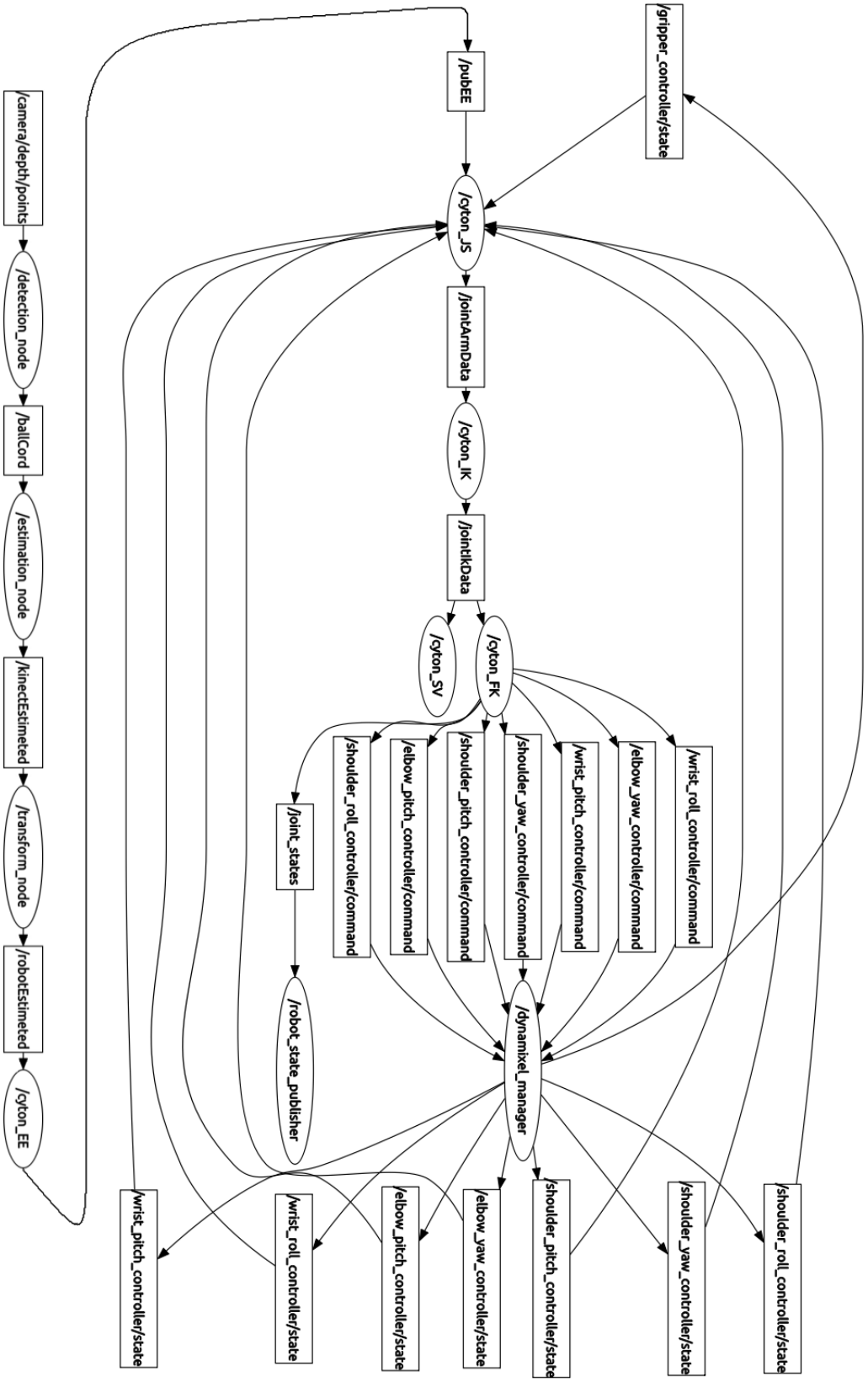


Figure 6.25: Integrated system control structure for ball catching application.

CHAPTER 7

Conclusions

This final chapter is dedicated to discuss the significance of the main results achieved, to present the final conclusions of the work and perspectives of future developments.

7.1 Results Discussion

The main objective of this work was to develop computational algorithms to control the Cyton Gamma 1500 robotic arm to catch a flying ball. This work was divided into four fundamental parts: first, to study the possible scenario for ball catching application, second, vision system that is to be used for this ball catching task with available resources, third, the need of low level control of the robotic arms that allow to control them using different approaches, fourth, study of spatiotemporal conditions for successful ball catching task and finally the development of a software architecture for simulation of ball catching application considering the real data-set from implemented vision system.

Calibration of different associated system was necessary for evaluation of performance of vision system and for the purpose of integration of developed vision system with manipulator arm motion control system. It is observed that the implemented vision system is robust enough to be used for ball catching task.

It is also observed that the depth sensor works fairly well in the range of 0.8 meter to 4 meter. Out side of this range, error increases significantly and this creates problem in estimation. Depth sensor's field of view is limited and that puts a constraint on distance of ball throwing task.

Implementation of Kalman filter for estimation is found to be suitable with some advantageous feature and it helped in solving the core problem: noise and sensory

imperfection in its measurements.

To make the implemented vision system a more robust one, human skeleton can be added so as to get an idea of the ball's landing vertical plane at an early stage from the user's hand movements. This will reduce the time-for-action (TfA) and subsequently more time can be given for a better perception.

During the course of implementation of motion control algorithm, it was noticed that the ROS package for 7 DOF configuration is not suitable for ball catching task because of its sluggish nature. It runs at a rate of $4 \sim 5$ Hz. To solve this problem, the configuration of manipulator arm is reduced to 3 DOF. This 3 DOF based configuration solved the problem of algorithm execution rate to a great extent.

Manipulator arm was tested for performance evaluation under different circumstances. This provided an overall idea about the dynamic capability of manipulator arm for the evaluation of feasibility of the ball catching task.

In terms of analysis of spatiotemporal conditions, it is observed that if the height of thrown ball is more then, the time-of-flight (ToF) increases. Increase in time-of-flight (ToF) provides more time for time-for-action (TfA) and time-for-perception (TfP). In addition to that, for better estimation more no of uniform samples are required, this in turn increases time-for-perception (TfP) and makes ball catching task difficult. This affects time-for-action (TfA) directly. If better measurement is available at the very beginning then, the robot can move to exact position at an early stage.

Considering all the constraints along with manipulator arm limitations in terms of maximum joint velocity and joint displacement range, it is found that exist some cases where successful ball catching task is possible.

7.2 Final Conclusion

Globally, the results obtained are acceptable and relevant, even if some refinements, improvements and extensions are required to improve the system's performance. Considering this, following are some of the conclusions can be drawn:

1. Implemented vision system is robust enough to be used for ball catching task.
2. A finely tuned Kalman filter is suitable for estimation, this estimation can be used for ball catching task.
3. Simulation of implemented vision system based data can be used for analysis of spatiotemporal conditions.
4. ROS provides an upper-hand in these kind works where distributed control is necessary considering implementation, control and management of several processes.
5. Implemented system is suitable for ball catching task, but only under some circumstances.

6. Finally, the vision system performance was satisfactory and the robotic arm also performed well during performance evaluation.

7.3 Future Work

During the implementation of this project, it was noticed that the system could be improved by addressing to new features or improving the implemented ones. Given the current state of development, the perspectives of future work points in the following directions:

1. Kinect version 2 can be used for establishment of better vision system.
2. Fine tuning of implemented Kalman filter can be done to improve the estimation process.
3. System can be modified for a robust control structure with automatic unwanted error avoidance feature.
4. A new and better robotic manipulator should be used for this kind of task, because this arm is not rigid enough for highly dynamic applications.
5. Human skeleton and point cloud data can be used simultaneously to get a better idea about ball's landing point as soon as possible.
6. Better calibration method can be implemented to get transform between two coordinate frames in order to reduce error.
7. Before going to any kind of real-time implementation, evaluation of developed system can be done in virtual environment. In ROS, rviz and Gazebo can be used for simulation purpose
8. New available version of ROS can be used for this application, as the catkin based build system still exists.
9. If possible, stereo vision system with better features can be used. This will help in increasing distance between subject and manipulator arm, subsequently increased time-of-flight (ToF) is possible.

References

[1] Michael A. Goodrich and Alan C. Schultz. Human-robot interaction: A survey. *Foundation and Trends in Human-Computer Interaction*, 1(3):203 – 275, 2007.

[2] Jorn Malzahn, Anh Son Phung, and Torsten Bertram. A multi-link-flexible robot arm catching thrown balls. In *7th German Conference on Robotics, Proceedings of ROBOTIK 2012*, Munich, Germany, May 2012. VDE.

[3] Jens Kober, Matthew Glisson, and Michael Mistry. Playing catch and juggling with a humanoid robot. Technical report, Disney Research, Pittsburgh, PA 15213, USA, 2012.

[4] Jens Kober, Matthew Glisson, and Michael Mistry. Playing catch and juggling with a humanoid robot. @ONLINE available. `https://www.youtube.com/watch?v=83eGcht7IiI`, December 2015.

[5] Institute of Robotics and German Aerospace Center Mechatronics. Rollin' justin, dlr. @ONLINE available. `http://www.dlr.de/rm/en/desktopdefault.aspx/tabid-5471/`, December 2015.

[6] Oliver Birbach and Udo Frese. Estimation and prediction of multiple flying balls using probability hypothesis density filtering. *IEEE/RSJ International Conference on Intelligent Robots & Systems*, 2011.

[7] Berthold Bauml, Thomas Wimbock, and Gerd Hirzinger. Rollin' justin robot catches balls tossed in its direction. @ONLINE available. `https://www.youtube.com/watch?v=R6pPwP3s7s4`, December 2015.

[8] Seungsu Kim, Ashwini Shukla, and Aude Billard. Catching objects in flight, lasa, epfl. @ONLINE available. `https://www.youtube.com/watch?v=24TCUZISIdU`, December 2015.

[9] Aude Billard and Daniel Grollman. Robot learning by demonstration, scholarpedia, 8(12):3824. @ONLINE available. http://www.scholarpedia.org/article/Robot_learning_by_demonstration, December 2015.

[10] Kuka Robots. Kuka lbr iiwa, @ONLINE available. http://www.kuka-lbr-iiwa.com/, December 2015.

[11] Michael Brading, Kenneth Salsman, Manjunath Somayaji, and Aptina Imaging. Using 3d sensors to bring depth discernment to embedded vision apps. @ONLINE available. http://www.embedded.com, February 2016.

[12] Georg Batz, Arhan Yaqub, Haiyan Wu, Kolja Kuhnlenz, and Martin Buss Dirk Wollherr. Dynamic manipulation: Nonprehensile ball catching. In *18th Mediterranean Conference on Control & Automation*, Marrakech, June 2010. IEEE.

[13] Jwu-Sheng Hu, Ming-Chih Chien, Yung-Jung Chang, Yen-Chung Chang, Shyh-Haur Su, Jwu-Jiun Yang, and Chen-Yu Kai. A robotic ball catcher with embedded visual servo processor. In *International Conference on Intelligent Robots & Systems*, Taipei, October 2010. IEEE.

[14] Seungsu Kim, Elena Gribovskaya, and Aude Billard. Learning motion dynamics to catch a moving object. In *10th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, Marrakech, December 2010. IEEE.

[15] Vicon Motion Systems Ltd. Motion capture systems, @ONLINE available. http://www.vicon.com/, April 2016.

[16] Robai Corporation. Cyton gamma 1500, @ONLINE available. http://www.robai.com/robots/robot/cyton-gamma-1500/, October 2015.

[17] Microsoft Corporation. Kinect for xbox 360, @ONLINE available. http://www.xbox.com/en-US/xbox-360/accessories/kinect, October 2015.

[18] Inc. (OSRF) Open Source Robotics Foundation. Robot operating system, @ONLINE available. http://www.ros.org/, October 2015.

[19] Itseez team. Opencv (open source computer vision), @ONLINE available. http://opencv.org/, November 2015.

[20] ROBOTIS INC. Robotis e-manual v1.27.00, mx-28t / mx-28r, @ONLINE available. http://support.robotis.com/en/product/dynamixel/mx_series/mx-28.htm, December 2015.

[21] ROBOTIS INC. Robotis e-manual v1.27.00, mx-64t / mx-64r, @ONLINE available. http://support.robotis.com/en/product/dynamixel/mx_series/mx-64.htm, December 2015.

[22] Robai Corporation. Cyton gamma 1500 arm specifications, @ONLINE available. `www.robai.com/assets/Cyton-Gamma-1500-Arm-Specifications_2015.pdf`, November 2015.

[23] Andrew Davison. Kinect open source programming secrets, @ONLINE available. `http://fivedots.coe.psu.ac.th/~ad/jg/nui16/index.html`, November 2015.

[24] Mattia Avancini. *Using Kinect to emulate an Interactive Whiteboard*. PhD dissertation, 2012.

[25] Rethink Robotics. rviz (ros visualization), @ONLINE available. `http://sdk.rethinkrobotics.com/wiki/Rviz`, March 2016.

[26] D. Halliday, R. Resnick, and J. Walker. *Fundamentals of Physics*. Wiley Publications, 9th edition, 2010.

[27] William Woodall. Converting between ros images and opencv, @ONLINE available. `http://wiki.ros.org/cv_bridge/Tutorials/`, December 2015.

[28] Nicolas Burrus. Kinect calibration, @ONLINE available. `http://nicolas.burrus.name/index.php/Research/KinectCalibration`, April 2016.

[29] Paulo Dias, Joao Silva, Rafael Castro, and Antonio J. R. Neves. *Detection of Aerial Balls Using a Kinect Sensor*. Springer International Publishing, May 2015.

[30] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, New York, ninth dover printing, tenth gpo printing edition, 1964.

[31] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. 1979.

[32] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, Chapel Hill, NC, USA, 1995.

[33] Mohinder S. Grewal and Angus P. Andrews. *Kalman Filtering: Theory and Practice with MATLAB*. Wiley-IEEE Press, 4th edition, 2014.

[34] Samprit Chatterjee and Ali S. Hadi. *Regression Analysis by Example*. Wiley Publications, 5th edition, 2012.

[35] Thomas S. Shores. *Applied Linear Algebra and Matrix Analysis*. Springer-Verlag New York, 2007.

[36] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[37] G. M. Siouris, Guanrong Chen, and Jianrong Wang. Tracking an incoming ballistic missile using an extended interval kalman filter. *IEEE Transactions on Aerospace and Electronic Systems (Volume:33 , Issue: 1)*, 1997.

[38] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. MIT Press, Cambridge, MA, USA, 1st edition, 1982.

[39] P.J. McKerrow. *Introduction to Robotics*. Electronic Systems Engineering Series. Addison-Wesley Pub (Sd), May 1991.

[40] Ioan A. Sucan and Sachin Chitta. Moveit!, @ONLINE available. `http://moveit.ros.org`, April 2016.

[41] Open Robot Control Software. Kinematic and dynamic solvers, @ONLINE available. `http://www.orocos.org/kdl/UserManual/kinematic_solvers`, April 2016.

[42] Inc. (OSRF) Open Source Robotics Foundation. Robot model, @ONLINE available. `http://wiki.ros.org/robot_model`, April 2016.

[43] Lentin Joseph. *Mastering ROS for Robotics Programming*. Packt Publishing, December 2015.

[44] Faculty of Human Sciences. 8-camera vicon passive marker system, @ONLINE available. `http://www.psy.mq.edu.au/me2/index.php/about/`, April 2016.

Appendices

APPENDIX A

Camera Parameters

This part of appendix contains some of the most important information about camera that are used either in ROS or in another platform where ever vision system is used. This `sensor_msgs` is that message which defines meta information for a camera.

A.1 sensor_msgs/CameraInfo Message

It should be in a camera namespace on topic "`camera_info`" and accompanied by up to five image topics named:

1. image_raw - raw data from the camera driver, possibly Bayer encoded.
2. image - monochrome, distorted.
3. image_color - color, distorted.
4. image_rect - monochrome, rectified.
5. image_rect_color - color, rectified.

The image_pipeline contains packages (image_proc, stereo_image_proc) for producing the four processed image topics from image_raw and camera_info.. The image_geometry package provides a user-friendly interface to common operations using this meta information. If you want to, e.g., project a 3d point into image coordinates, we strongly recommend using image_geometry. If the camera is not calibrated, the matrices D, K, R, P should be left zeroed out. In particular, clients may assume that K[0] == 0.0 indicates of a non-calibrated camera. The meaning of the camera parameters are described in detail at:

http://www.ros.org/wiki/image_pipeline/CameraInfo

Image acquisition info is defined as follows: Time of image acquisition, camera coordinate frame ID, where Header timestamp should be acquisition time of image, Header frame_id should be optical frame of camera, origin of frame should be optical center of camera, +x should point to the right in the image, +y should point down in the image, +z should point into the plane of the image. Camera calibration Parameters are those values that are usually fixed during camera calibration. Their values will be the same in all messages until the camera is re-calibrated. Note: self-calibrating systems may ”re-calibrate” frequently.

The internal parameters can be used to warp a raw (distorted) image to:

1. An un-distorted image (requires D and K)
2. A rectified image (requires D, K, R)

The projection matrix P projects 3D points into the rectified image. The image dimensions with which the camera was calibrated. Normally this will be the full camera resolution in pixels and is declared as:

1. uint32 height
2. uint32 width

Intrinsic camera matrix for the raw (distorted) images is the one which . Projects 3D points in the camera coordinate frame to 2D pixel coordinates using the focal lengths (fx, fy) and principal point (cx, cy). K is a 3x3 row-major matrix of float64[12] type. K is defined as follows:

$$K = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix}$$

By convention, Projection/camera matrix matrix specifies the intrinsic (camera) matrix of the processed or rectified image. That is, the left 3×3 portion is the normal camera intrinsic matrix for the rectified image. It projects 3D points in the camera coordinate frame to 2D pixel coordinates using the focal lengths (fx^{τ}, fy^{τ}) and principal point (cx^{τ}, cy^{τ}) - these may differ from the values in K. Projection matrix is defined as follows:

$$P = \begin{bmatrix} fx^{\tau} & 0 & cx^{\tau} & Tx \\ 0 & fy^{\tau} & cy^{\tau} & Ty \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Operational Parameters define the image region actually captured by the camera driver. Although they affect the geometry of the output image, they may be

changed freely without re-calibrating the camera. Region of interest (sub-window of full camera resolution), given in full resolution image coordinates. A particular ROI always denotes the same window of pixels on the camera sensor, regardless of other settings. The default setting of roi (all values 0) is considered the same as full resolution (roi.width = width, roi.height = height).

A.2 sensor_msgs/PointCloud2 Message

This message holds a collection of N-dimensional points, which may contain additional information such as normals, intensity, etc. The point data is stored as a binary blob, its layout described by the contents of the "fields" array. The point cloud data may be organized 2d (image-like) or 1d (un-ordered). Point clouds organized as 2d images may be produced by camera depth sensors such as stereo or Kinect.

Time of sensor data acquisition, and the coordinate frame ID (for 3d points) are defined by header and frame. 2D structure of the point cloud. If the cloud is unordered, height is 1 and width is the length of the point cloud. Other parameters such as if the data is dense or bigendian or not are declared.