# iGPS Based Motion Control of a Robotic Manipulator using Robot Operating System

*A Project Report*

*submitted by*

**ALAP KSHIRSAGAR**

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

**MASTER OF TECHNOLOGY**
**IN**
**MECHANICAL DESIGN**


**DEPARTMENT OF MECHANICAL ENGINEERING**
**INDIAN INSTITUTE OF TECHNOLOGY MADRAS.**

**May 2017**

# THESIS CERTIFICATE

This is to certify that the thesis titled **iGPS Based Motion Control of a Robotic Manipulator using Robot Operating System**, submitted by **Alap Kshirsagar**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology in Mechanical Design**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

**Dr. Sourav Rakshit**
Project Guide
Assistant Professor
Dept. of Mechanical Engineering
IIT-Madras, Chennai
India, 600036

**Dr. B.V.S.S.S. Prasad**
Head of the Department
Dept. of Mechanical Engineering
IIT-Madras, Chennai
India, 600036

Date : 9 May 2017

Place : Chennai

# ACKNOWLEDGEMENTS

I would like to express my sincere gratitude towards Dr. Sourav Rakshit and Dr. Burkhard Corves for giving me the opportunity to pursue this joint-project work under their guidance. Also I would like to thank the selection committee for providing me the DAAD-IIT Master's Sandwich Scholarship.

This project would not have been successful without the mentor-ship and valuable inputs of Jorge De La Cruz. I would also like to thank Tim Detert, Tobias Haschke and all other IGM-COAR group members for their constant help and contribution in this work. I truly enjoyed working in this group and will be indebted to all of you eternally.

Last but not the least, I would like to thank my parents, brother and friends. They are the greatest blessings I have been given in this life. I could never do enough to sufficiently repay the unconditional love and support they have constantly bestowed upon me.

Alap Kshirsagar

# ABSTRACT

KEYWORDS: Indoor GPS; Robot Operating System; Robot Manipulator Control

Robotic manipulators have become the key components of industrial automation. The accuracy of robotic manipulator's end effector determines the quality of products and dictates versatility of the system. Usually, the motion of the end effector is estimated and controlled by using the encoder feedback only. But the resulting end effector trajectory is inaccurate due to link deformations, joint flexibilities, external vibrations, non-linearities of the gear mechanisms, manipulator dynamics etc. To overcome these effects, the use of external sensor for direct measurement of end effector's position and orientation was proposed decades ago. Since then a lot of work has focussed on computer vision based end effector control. But vision feedback is difficult to incorporate into the control loop due to low sampling rate and delays.

In this work, a new technology called iGPS is used to get accurate measurements of end effector's position and orientation. Indoor-GPS or iGPS is a cutting edge metrology system which allows monitoring several sensors simultaneously and has accuracy up to $\pm 0.1mm$ for 3D positions. Also the latency for the iGPS system is lower than other systems. Thus iGPS can be used for improving the accuracy of robotic manipulators. In recent years, Robot Operating System has revolutionized the software development process for novel robotics algorithms and applications. This open source framework facilitates modularity, customizability, extensions to multi-robot systems and global collaborations. Therefore, in this work ROS is chosen as the platform for establishing communication between different components of the system and developing software for iGPS feedback based robot motion control.

Three different control structures are presented for including the iGPS feedback in the motion control system of a robotic manipulator. The motion control system consists of motion planner, joint controller and the joint states publisher. The first approach includes iGPS feedback before the motion planning stage. The second approach involves

iGPS correction in the joint state trajectory and the third approach includes iGPS feedback in the joint control loop through the joint states publisher. The last two approaches require transformation from end-effector co-ordinates to joint states. These transformed joint states can be fused with the encoder data to reduce error and overcome difficulties arrising due to missing data. The use of Kalman Filter for fusion of multifrequency and delayed sensor data is described. The structures for iGPS feedback based robot motion control are implemented in the ROS framework and tested in a robot simulator 'Gazebo' and also on physical 'Universal Robot-5' manipulator. Results of conducted experiments show that the error in end-effector trajectory can be reduced to 25% using the iGPS feedback.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| **IITM** | Indian Institute of Technology Madras |
| **RWTH** | Rheinisch-Westfälische Technische Hochschule |
| **IGM** | Institut für Getriebetechnik und Maschinendynamik |
| **COAR** | Center of Advanced Robotics |
| **ROS** | Robot Operating System |
| **ROS-I** | Robot Operating System Industrial |
| **API** | Application Programming Interface |
| **UR5** | Universal Robots-5 |
| **LRP** | Long Reach Probe |
| **PCE** | Position Calculation Engine |
| **DoF** | Degrees of Freedom |
| **1-D** | One Dimensional |
| **GPS** | Global Positioning System |
| **iGPS** | Indoor GPS |
| **URDF** | Unified Robot Description Format |
| **SRDF** | Semantic Robot Description Format |
| **XML** | Extensible Markup Language |
| **OMPL** | Open Motion Planning Library |
| **FCL** | Flexible Collision Library |
| **ACM** | Allowed Collision Matrix |
| **IPTP** | Iterative Parabolic Time Parametrization |
| **KDL** | Kinematics and Dynamics Library |
| **KF** | Kalman Filter |
| **IK** | Inverse Kinematics |
| **RMS** | Root Mean Square |

# NOTATION

| | |
|---|---|
| $T_i^d$ | Desired end effector pose |
| $J_i^d$ | Desired joint state vector |
| $P_i^d$ | Desired end effector position vector |
| $R_i^d$ | Desired end effector orientation matrix |
| $T_i^a$ | Actual end effector pose |
| $J_i^a$ | Actual joint state vector |
| $P_i^a$ | Actual end effector position vector |
| $R_i^a$ | Actual end effector orientation matrix |
| $T_i^{d*}$ | Corrected target end effector pose |
| $J_i^{d*}$ | Corrected target joint state vector |
| $P_i^{d*}$ | Corrected target end effector position vector |
| $R_i^{d*}$ | Corrected target end effector orientation matrix |
| $\Delta_i$ | Error in end effector pose |
| $\delta_i$ | Error in joint state vector (also used for sensor delay) |
| $x_k$ | State of the system $k$ |
| $\hat{x}_k^-$ | Priori state estimate at step $k$ |
| $A$ | Matrix that relates the previous state and current state |
| $B$ | Matrix that relates the previous control input to current state |
| $u_k$ | Control input at step $k$ |
| $w_k$ | Process noise at step $k$ |
| $z_k$ | Sensor measurement at step $k$ |
| $v_k$ | Sensor noise at step $k$ |
| $P_k^-$ | Priori estimate error covariance at step $k$ |
| $Q$ | Process noise covariance |
| $K_k$ | Kalman gain |
| $H$ | Matrix that relates the state to sensor measurement |
| $R$ | Measurement noise covariance |
| $\hat{x}_k$ | Posteriori state estimate at step $k$ |

| | |
|---|---|
| $P_k$ | Posteriori estimate error covariance at step $k$ |
| $I$ | Identity matrix |
| $Z$ | Zero matrix |
| $f$ | Frequency |
| $T^{iGPS}$ | Pose in iGPS world frame |
| $R^{iGPS}$ | Orientation in iGPS world frame |
| $P^{iGPS}$ | Position in iGPS world frame |
| $[\tau]_{iGPS}^{ur5}$ | Transformation matrix between iGPS world frame and UR5 world frame |
| $T^{UR5}$ | Pose in UR5 world frame |
| $J$ | Jacobian matrix |

# CHAPTER 1

# Introduction

This chapter will provide an overview of the project, problem statement, motivation behind the research work and literature review. In the last section, the organization of rest of the thesis will be presented.

## 1.1 Overview and Problem Statement

In this era of automation, robots have brought about a new wave of industrial revolution. Robotic manipulators have replaced manual labour in a varied range of applications. Today's robotic manipulators are capable of performing repetitive tasks with a high efficiency. This has increased the output while reducing the manufacturing costs.

End effectors of industrial manipulators have to follow specific trajectories or reach destination points with high accuracy and speed. The motion of the end effector is usually estimated from the joint state feedback by using a robot model. There are several ways of improving the accuracy of a robot manipulator. A lot of research has been done on improving the accuracy of robotic manipulation either by using advanced control algorithms for accurately controlling individual joints or by accurately modelling the kinematic and dynamic parameters of the physical robot. However, almost all of the control approaches use only the joint state feedback from encoders attached to motors.

Most of the industrial robots use gear mechanisms to drive each joint. Due to these gear mechanisms of high reduction ratio, every joint becomes a flexible joint. Also each link of the robot is usually modelled as a rigid body. But in reality there are link deformations due to gravitational load and applied load (in the form of objects attached to the end effector). Thus the actual trajectory followed by the end-effector does not match accurately with the desired trajectory. Other factors that also contribute to the inaccuracy are external vibrations, non-linearities of the gear mechanisms, manipulator dynamics etc.

One of the proposed solutions in literature to counter these physical effects is the direct feedback of the end-effector's position and orientation (referred to as 'pose' hereafter for simplicity) [1]. Researchers have explored various alternatives for direct measurement of the end-effector's pose. Some of these include cameras, laser trackers, ultrasonic range finders, inertial sensors and position sensitive detectors. Computer vision based method is recognized as a promising choice. But due to low sampling rate and additional delays of the vision system, it is difficult to incorporate the vision measurements in the joint control loop [5].

In recent years, a metrology system called 'iGPS' or 'indoor GPS' has been proven to be useful for obtaining 'real-time' six degree of freedom (6DoF) data about pose of the end-effector or any other point on robot [6]. The accuracy of iGPS system is $200\mu m$ for 3D positions [7]. Also the sampling rate of iGPS system is much higher than that of a vision based system. This enables the use of iGPS measurements of end effector's pose in the motion control loop of a robotic manipulator.

Traditionally, the robotic manipulators have been controlled using the motion control packages supplied by the manufacturer. These packages are proprietary and differ drastically from manufacturer to manufacturer. Thus the software development carried out for a particular manipulator cannot be easily extended to other manipulators. Also the OEM motion control packages have their own set of operations which are not customizable.

The use of Open Source platform for robot software development facilitates modularity, customizability, extensions to multi-robot systems and global collaborations. In recent years, Robot Operating System (ROS) has become a de facto standard platform for open source software development in robotics [8]. Researchers from all over the world are developing ROS packages and tools for rapid development of novel robotics applications. ROS-Industrial project is extending the advanced capabilities of ROS software to manufacturing.

The different components of the system are shown in Fig. 1.1. The problem statement for this research work is: how can we use iGPS to improve the accuracy of robot motion in a ROS framework?

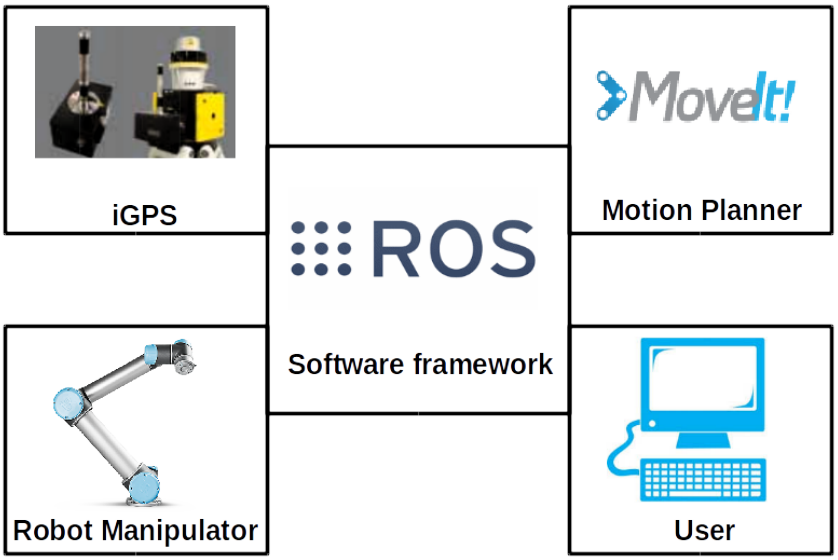



Figure 1.1: iGPS based control of a robot manipulator in ROS

## 1.2 Motivation

The advances in computational power and robot architectures have opened new avenues of manufacturing applications. One such novel application is 3D printing. This rapid prototyping technique has revolutionized the manufacturing industry. Apart from conventional uses of 3D printing such as printing spare parts, scale models and art, 3D printing is also being used by medical industry for prostheses and even human body parts. Researchers have proposed the use of multiple degree of freedom robotic manipulators for 3D printing [9]. The benefits include reduced printing time and increased versatility. However the robot motion accuracy needs to be improved for realizing these benefits. The use of advanced control algorithms relies on the accuracy of kinematic model of the robot. Improving the robot motion accuracy through direct measurement of end-effector will enable use of 3D printing to make final products ready for the market.

Another application which requires high positioning accuracy of robot manipulators is manipulation of small deformable objects like thin cables and wires. Researchers have developed various algorithms to obtain trajectories for manipulating deformable objects to desired configurations from different initial configurations [10]. However the objects are not exactly manipulated as desired because the actual trajectory followed by the end-effector is inaccurate due to the joint flexibilities, link deformations, vibrations

and other external disturbances.

In advanced manufacturing systems, multiple robots collaborate with each other, and sometimes even with humans, to accomplish complex tasks. While today's industrial robots have very good repeatability, the absolute accuracy of co-operating robots is not so good. Different factors such as hardware and software limitations, manufacturing tolerances, payload effects, compliance, elasticity and thermal effects contribute to this inaccuracy. External metrology systems like iGPS provide a way of compensating for these errors and improving cooperative robot positioning [11].

Open source software development for external sensor feedback based control of robot manipulators will allow global collaborations, applications to different tasks and extensions to multi-robot scenarios. Also the modularity and customizability of the software will enable comparison between different algorithms for individual modules as well as for the overall approach.

## 1.3 Project Goals

The primary objective of this project is to use iGPS feedback for accurate motion control of robotic manipulator in a ROS framework. This can be divided into following sub-goals:

1. Devise algorithms for iGPS feedback based control of robotic manipulators
2. Develop ROS packages for integration of iGPS in robot motion control system
3. Compare accuracy of iGPS and computer vision system
4. Evaluate end-effector feedback based control algorithms in robot simulator
5. Evaluate performance of the iGPS based control algorithms on industrial manipulator

## 1.4 Literature review

The idea of using direct measurement of end-effector's pose in robot motion control loop was proposed decades ago [1]. In their work, the authors have described two algorithms for end-effector sensor feedback control as shown in Fig.1.2. In the first

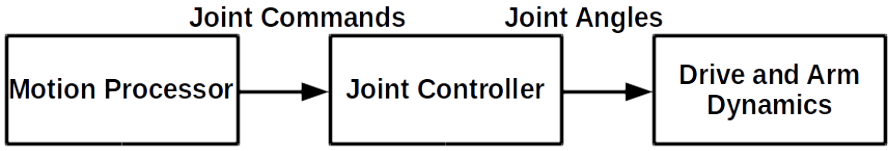
algorithm (Fig.1.2b), the sensor feedback is given to the motion processor. The motion processor then sends the corrective commands to the joint control module. The second approach (Fig.1.2c) uses the end-effector pose feedback directly in the joint control module by transforming it from end-effector co-ordinates to joint angle co-ordinates. Since this work, a lot of research has been done on exploring different sensors for direct measurement of end-effector's pose.

The desired features of a metrology system which can be used for applications of robots in industrial automation are: high accuracy and large workspace volume. Also the system should be capable of measuring all six degrees of freedom, preferably of multiple points or features. Several approaches have been developed for such metrology systems which can be broadly catergorized into two types [12]:

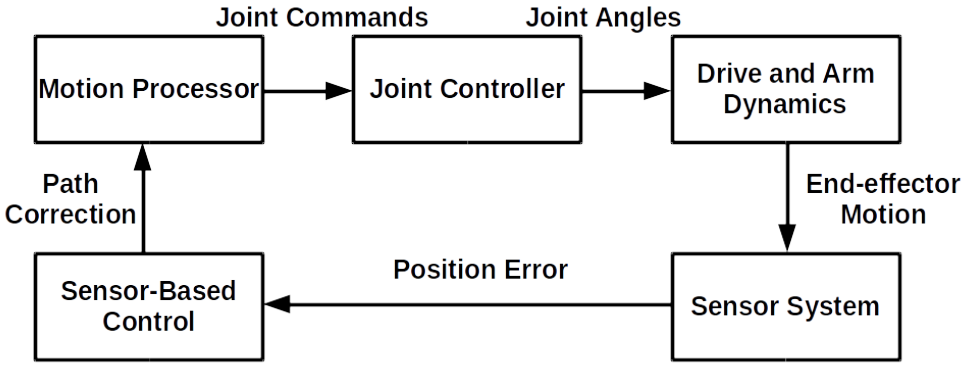
1. Centralized Systems: Stand-alone systems that can work independently
2. Distributed Systems: Systems with multiple measuring stations that have to work co-operatively

The most popular example of centralized system is a laser tracker which can track only individual points. The laser trackers pose a risk of interruption of production process due to loss of laser line visibility. Also for tracking multiple degrees of freedom, data fusion of several laser trackers is required. Computer vision based measuring systems are the most popular distributed systems for many scientific and industrial applications. They are capable of tracking several objects in the visible space of the cameras. However, the accuracy of camera depends heavily upon the number of pixels and distance of the object from camera. If the distance is large, the markers or tracked features on objects also need to be larger in size. This poses difficulties in extending camera systems for applications in large production halls and assembly lines.

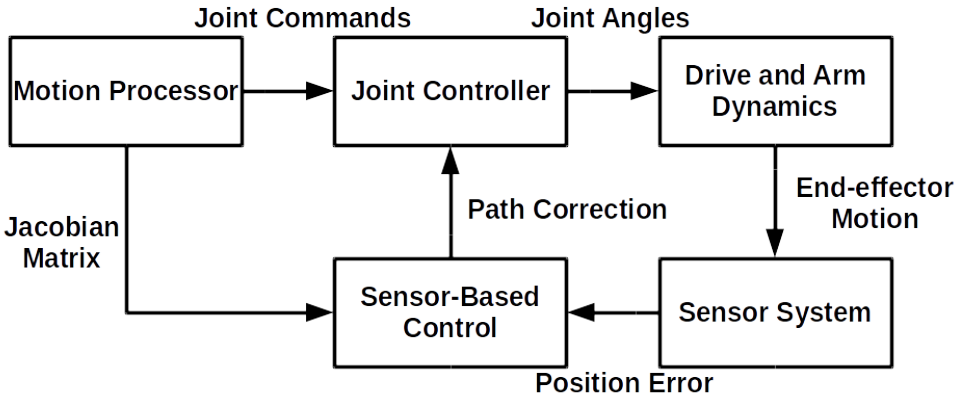
Indoor GPS (iGPS) is a scalable measurement concept that converts a workspace into a metrology enabled work volume. The operation of an iGPS system is comparable to a global positioning system (GPS), but it is designed for industrial applications on a facility-wide scale. GPS satellites are replaced by infrared iGPS laser transmitters that activate a measurement field as large as an entire room or facility. iGPS features accuracies that are roughly a hundred thousand times higher than consumer GPS systems. Tools, parts, probes and equipment, such as AGVs, can be equipped with iGPS receivers that are tracked by the transmitters. As such, positions can be measured or dynamically




(a) Open loop




(b) Motion Processor loop




(c) Axis Processor loop

Figure 1.2: Alternative structures for robot end effector control (Adapted from [1])

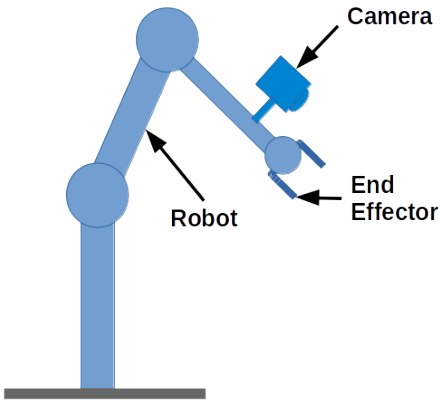
tracked with a high degree of confidence in order to provide accurate metrology data or to facilitate tracking and positioning applications. To obtain accurate positioning, receivers need line-of-sight from minimum 2 transmitter fields. Adding more transmitters enlarges the measurement area, improves robustness by guaranteeing line-of-sight, and further increases measurement accuracy. iGPS is a flexible metrology solution that offers full freedom of operation for indoor and outdoor use. The iSpace systems are standardized configurations built with iGPS components focusing on ease of installation and use [7].

Computer vision based motion control of robotic manipulators, commonly known as 'visual servoing' [13], has been researched extensively. The two main fundamentally different configurations that exist for a visual servoing system are [14]:

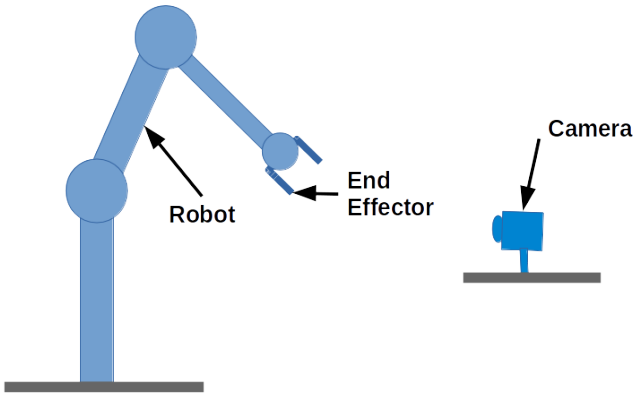
1. Eye-in-hand
2. Eye-to-hand or stand-alone

In a eye-in-hand configuration [15] [16], the camera is attached to the end-effector of robotic manipulator and moves along with it, as shown in Fig.1.3a. Whereas in a eye-to-hand configuration, as shown in Fig.1.3b, the camera is fixed on the ground and tracks the motion of end effector [17], [18][19]. The major drawback of vision based approach is the slow sampling rate which makes it difficult to include this feedback into the joint control loop running at a much higher frequency [5].

External metrology systems offer an alternative to the vision based approaches. These include coordinate measuring machines, laster trackers, iGPS, ultrasound systems etc.[20]. The iGPS system offered by Nikon can be used to track multiple sensors simultaneously with an accuracy of $200\mu m$ [7]. Wang et al. [21] determined that for speeds below 10 cm/s, the iGPS is capable of producing relatively repeatable tracking data. Schmitt et al. [6] tested various layouts of iGPS transmitters and concluded that the 'Standard layout' advised by the manufacturer provided best measurement results. Norman et. al [22] used the iGPS metrology system for absolute positioning and movement of the cooperating robots. Schmitt et al. [23] implemented a very basic control loop for accurate motion of robot manipulator at slow speeds (0.1m/s) and also proposed the use of kalman filter to extend this method to higher speeds. Norman et al. [11] presented a validation of iGPS technology for co-operative robot positioning by comparing with the ballbar measurements.




(a) Eye-in-hand




(b) Eye-to-hand

Figure 1.3: Visual Servoing system configurations

But all the efforts done so far for iGPS based control of robotic manipulators have used commercial KUKA motion control packages. This poses limitations on the modularity and customizability of the control approaches. Also it is difficult to extend this work to robots of different manufacturers.

## 1.5 Outline of Thesis

This chapter has given an introduction to the thesis. The rest of the thesis is organized as follows. The next Chapter explains the individual components of the system shown in Fig.1.1 (iGPS, ROS, robot manipulator UR5 and motion planner MoveIt!). In Chapter 3, the algorithms for iGPS feedback based robot motion control are described in detail. Also the method of using Kalman Filter for fusion of multi-frequency and delayed data is explained. Chapter 4 describes the experimental set-up consisting of UR5 and iGPS system along with the ROS framework. Also the ROS implementation of each of the algorithm is explained. Then the results of tests conducted in robot simulator and on physical robot are presented in Chapter 5. Finally Chapter 6 consists of the conclusions and possible applications of this work. Also some directions for future research are presented.

# CHAPTER 2

# Description of System Components

This chapter will describe different components of the overall system shown in Fig.1.1. These include the iGPS technology, UR-5 manipulator, Robot Operating System and motion planning framework MoveIt!.

## 2.1 iGPS Technology

The iGPS is a type of distributed metrology system developed by Nikon Metrology. It offers a high measurement accuracy [upto 0.2mm] [6] with a large workspace. This system is capable of measuring position and orientation of several points in real time. Theoretically the number of points it can track simultaneously and independently is unlimited. Also it can automatically regain connection after a line-of-sight disruption. Other features of iGPS system include modularity, scalability, reconfigurability, portability and ease of handling and installation [24]. Currently iGPS system is used for applications in aircraft assembly and ship building with tasks including alignment, inspection, robot calibration, actuator metrolog and tracking [21].

### 2.1.1 Components of iGPS

There are three main components of iGPS:

1. Transmitters
2. Detectors or Probes
3. Position Calculation Engine or Main Server

#### Transmitters

The transmitters make up the constellation of 'reference points' similar to the Geosynchronus satellites in the Global Positioning System (GPS). They send two laser planes

Figure 2.1: iGPS transmitter (fixed)

in a frequency of 40Hz that can be recognized by the detectors. The transmitter just sends out the signals without higher logic or task. In our workshop, there are 5 fixed transmitters (one fixed transmitter is shown in Fig.2.1) and one mobile transmitter that can be used on a tripod. The mobile one is very useful to get access to some hidden places in the workshop. Three of the transmitters are tilted about 10 degrees to widen the space for mobile units on the floor. All the transmitters have power switches to make it much easier to turn them on and off. The transmitters need to run for at least 15 minutes before starting the calibration.

### Detectors/Probes

Several detectors/probes are provided with the iGPS system for different applications.

#### i6 Probe

The i6 Probe (shown in Fig.2.2) is the most important tool for the setup of frames and coordinate systems. It contains 4 detectors and therefore it can detect all 6 Degrees-of-freedom (DoF). It is basically a mounting point for different tips that can be used to clarify the position of points or objects with respect to given coordinate systems. The common workflow of the i6 Probe is strongly connected to Serveyor (software on main server) for step-by-step setups of coordinate systems or frames. The i6 Probe has its own power switch and a toggle button to interact with Surveyor. The battery status is shown directly at the battery case. It has to be made sure that the correct tip is selected in the frame setup. Otherwise incorrect coordinates will be measured and the results will be wrong even if the workflow is correct.

(a) Front view (b) Back view

(c) Tip Tools (d) Tips (e) Tip Callibrator

Figure 2.2: i6 Probe

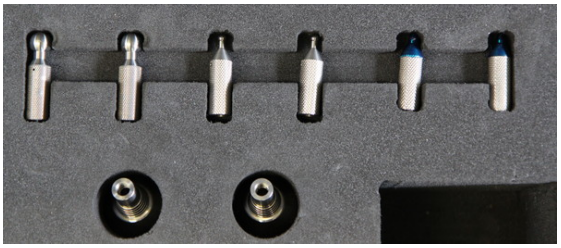
There are 3 different tips for the i6 Probe and Nikon provides two tools for mounting those tips. One can hold the tip and the other one can be attached to the probe. With opposite movements they can be mounted properly. All the tips are pre-calibrated by Nikon and can be selected within the frame configurations of the i6 Probe. If the results of a tip are getting worse it is possible to re-calibrate this tip. Therefore a special tool in surveyor is provided to move the tip within the tool calibrator. It is NOT recommended by the manufacturer to re-calibrate their tips. Normally the calibration can last for the whole lifetime of a tip. Only if a tip gets damaged by hitting hard surfaces it becomes necessary to re-calibrate them. The calibration routine for tips is more useful for self-made tips, if this is desired and necessary.

**i6 Long Reach Probe**

The i6 Long Reach Probe (LRP) (shown in Fig.2.3) is the elder brother of the i6 Probe. It also contains 4 detectors and can detect all 6 spatial DoF. The distances between the detectors are much wider than the ones from the i6 Probe and this is the biggest advantage for calibration. Due to this the effect of measurement errors become less important and so the calibration results are much better. Therefore the i6 LRP is always set as the frame for Bundle Collection by default. It is recommended to always

(a) Front view

(b) Back view



(c) Tips

Figure 2.3: i6 LRP Probe


(a) Front view


(b) Back view

(c) i5 Base

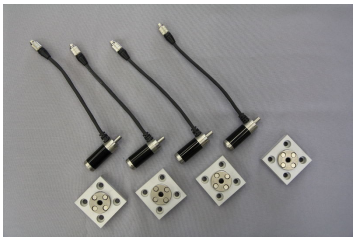
(d) i5 Mount

Figure 2.4: i5 Probe

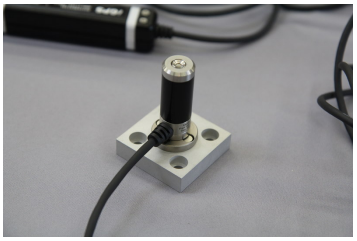
take the i6 LRP for the calibration routine. The PCE of the i6 LRP can be turned on at the black/yellow box by sliding the little button at the bottom. The LRP is also pre-calibrated by Nikon and any changes in calibration state can affect the absolute accuracy of iGPS. Beside the calibration options of the LRP it is also possible to measure certain points within the workspace as with the i6 Probe. On the bottom frame a tip holder can be mounted to achieve this functionality. There are multiple different tips for the LRP (shown in Fig.2.3c that are already set up within Surveyor to choose from.

**i5 Probe**

The i5 is a probe (shown in Fig.2.4) which can be used to measure 5 DoF. It contains 2 detectors that allow the detection of x, y, z positions and the two rotational axes. Only



(a) Single Detectors



(b) Single Detector in Mount

Figure 2.5: Single Detectors with Mounts

the rotation for the vertical axes cannot be detected by this device. Thus for 6-DoF measurements with i5's we need to create a frame of frames with a minimum of two i5's. The battery of the i5 is secured by a hooked button. Every i5 has its on power button. It is placed underneath the bottom detector and looks like a little grey dot.

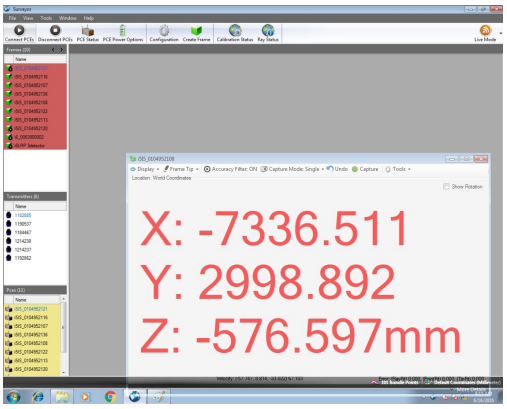
To be flexible in mounting the i5 at specific measurement places, every i5 has its own mounting plate. Every plate can be mounted by 4 screws and they contain magnets to ensure the full contact between the i5 and the plate. The centred position is ensured by a vertical pin in the mounting plate that needs to be fixed by the grub screw at the side.

**Single Detectors**

The single detectors (shown in Fig.2.5) are very useful for custom frames that need to be small and specific. For measurement tasks where two or three i5s cannot be used, a set of 4 single detectors can be considered. The single detectors need to be wired properly with a PCE and certain amplifiers. The system is marked explicitly with only one solution of wiring up all the connections. Similar to the i5's the single detectors also have their own mounting plates. Both plates are similar except that the mounting plates for the single detectors don't lock the rotation around the vertical axis. However this is not important because a single detector cannot detect this DoF.

**Main Server (iSpace) and Position Calculation Engine (Surveyor)**

Surveyor is the software provided by Nikon for communicating with other iGPS components. It is made to setup differant configurations of the system, to show frame positions, to calibrate the system etc. The user interface of Surveyor software is shown in Fig.2.6a. iSpace (shown in Fig.2.6b) is the main server which runs the Surveyor




(a) Surveyor software user interface


(b) iSpace (Main Server)

Figure 2.6: Main Server/PCE

software.

### 2.1.2 iGPS working principle

There are three main steps in the measurement algorithm of iGPS [11][25]. First step consists of calibration of the system to define a reference co-ordinate frame. This is done through an internal calibration procedure known as 'Bundle adjustment'.Through this procedure the relative locations of the transmitters, the rotation speeds and spin axis biases for each transmitter are stored in the system. The transmitters emit two laser beams at slightly different speeds between 40-50 hz. The beams are separated in horizontal plane by 90 degrees and inclined to the vertical plane by 30 degrees. Also each transmitter emits an IR strobe at the beginning of every second rotation. Each sensor in the workspace receives these signals and sends the time of arrivals to the Position Calculation Engines. The resolution of arrival times is 20ns and this determines the position measurement accuracy. Using these arrival times a ray can be defined from transmitter to the sensor. The elevation (theta) of the ray is determined by the time difference between two laser beams striking the sensor. For higher elevation angles, the time difference is more and for lower elevation angles the time difference is less. The azimuth angle is determined by the time difference between the reference strobe signal and the midpoint of the two laser beams. These elevation and azimuth angles are calculated for each detected transmitter. Then the position of the sensor with respect to the co-ordinate frame defined in the calibration procedure is obtained by using the

method of triangulation.

### 2.1.3 Accuracy of iGPS

For optimum performance it is recommended to keep the distance between sensors and transmitters between $3m$ to $35m$ and guarantee individual sensors to be within line of sight of at least 4 transmitters. The measurement error given by the manufacturers of the initial version of iGPS system based on a fairly comprehensive mathematical model with the combined standard unvertainty expanded to a $95\%$ confidence level ($2\sigma$) is $0.25mm$ [26]. Further, the claimed accuracy for overall system performance stands at $169\mu m$ for $15 \times 15 \times 1.5m$ working space and $315\mu m$ for a $30 \times 30 \times 3m$ working space with a network of six transmitters arranged in a box configuration [27]. These accuracy statements are valid under recommended operating conditions and with two second averaging of the measurements which may not be possible in all industrial environments.

The static tests conducted by several researchers have shown the 3D positioining error to be between $0.05 - 0.25mm$ and expanded coordinate uncertainity ($2\sigma$) of $\pm 1 - 1.1mm$ [24] [25] [28]. The dynamic tests conducted by [21] indicated that for speeds below $10cm/s$, the iGPS is capable of producing relatively repeatable tracking data. However, as speed is increased, the tracking accuracy degrades. At $1m/s$, the mean tracking error can be in the order of $3 - 4mm$. [25] have reported that for path tracking purpose the deviations are less than $0.3mm$ but with a delay of $0.3ms$.

## 2.2 Universal Robot-5 Manipulator

The Universal Robot-5 (UR5), shown in Fig.2.7, is a lightweight 6 DoF robot developed by Universal Robot group. UR5 is ideal for automating low-weight processing tasks like picking, placing and testing. It is easy to program and fast to set-up. It has been used widely in industry for different tasks ranging from assembly to painting, from packing to polishing, from injection moulding to welding and even 3D printing. In this section, the components of the UR5 manipulator, its specifications, workspace and communication methods will be described.

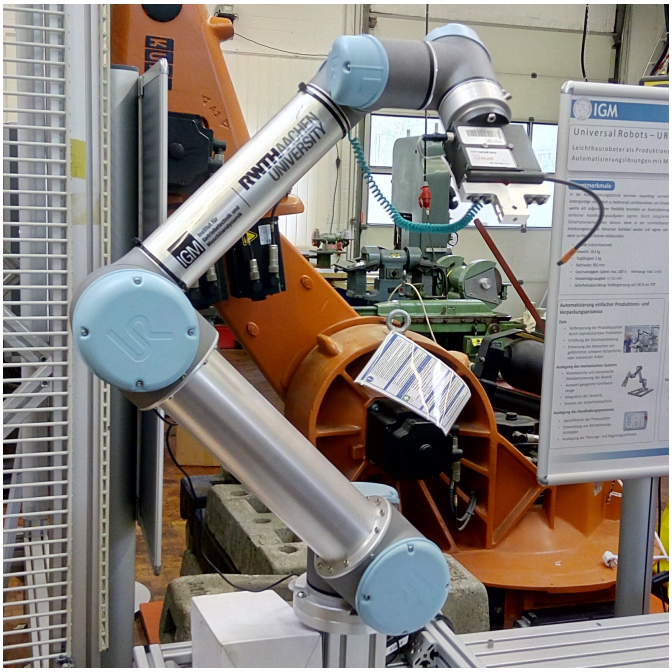



Figure 2.7: UR5 Manipulator

### 2.2.1 Components of UR5

The UR5 manipulator consists of 6 joints: shoulder pan, shoulder lift, elbow, wrist-1, wrist-2 and wrist-3. The base link is used to mount the robot on a table or on ground. A small shoulder link connects the shoulder pan and lift joints. Upper arm link is the longest link which connects the elbow joint and shoulder joint. The forearm link connects the elbow joint and wrist joints. Small wrist links connect the three wrist joints. An external end-effector such as a gripper can be attached to the wrist-3 joint using a mounting hub. The dimensions of each link are shown in Fig.2.8.

### 2.2.2 Specifications of UR5

Table 2.1 lists the important technical specifications of UR5 given by the Universal Robots [2]

### 2.2.3 Workspace of UR5

The workspace of the UR5 robot extends $850mm$ from the base joint [2] as shown in Fig.2.9. It is important to consider the cylindrical volume directly above and directly below the robot base when a mounting place for the robot is chosen. Moving the tool

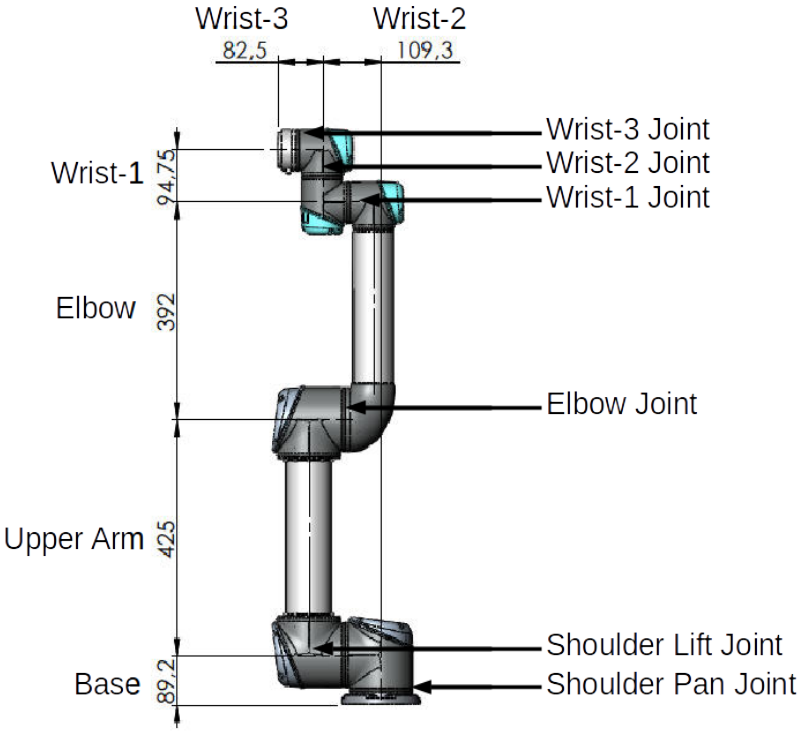
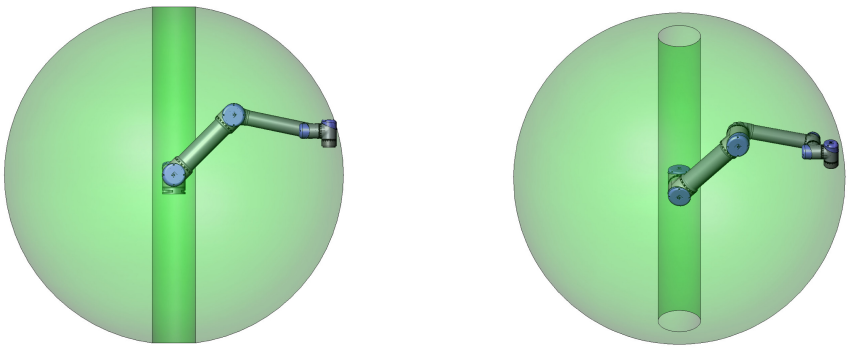



Figure 2.8: Components of UR5

| | |
|---|---|
| Weight | 18.4 kg |
| Payload | 5 kg |
| Reach | 850 mm |
| Joint ranges | $\pm 360°$ on all joints |
| Speed | Joint: Max $180°$/sec Tool: 1m/sec. |
| Repeatability | $\pm 0.1mm$ |
| Footprint | $\phi 149mm$ |
| Degrees of freedom | 6 rotating joints |
| Control box size (WxHxD) | 475 mm x 423 mm x 268 mm |
| Communication | TCP/IP 100 Mbit: IEEE 802.3u, 100BASE-TX Ethernet socket & Modbus TCP |
| Power consumption | Approx. 200 watts using a typical program |
| Materials | Aluminium, ABS plastic, PP plastic |
| Operating Temperature | $0 - 50°C$ |
| Power supply | 100-240V AC, 50-60 Hz |

Table 2.1: Technical specifications of UR5 manipulator



(a) Front (b) Tilted

Figure 2.9: UR5 workspace (Figure courtesy [2])

close to the cylindrical volume should be avoided if possible, because it causes the joints to move fast even though the tool is moving slowly, causing the robot to work inefficiently and the conduction of the risk assessment to be difficult.

### 2.2.4 Communication with UR5

The Universal Robot can be controlled a three different levels: The Graphical User-Interface (GUI) Level, the Script Level and the C-API Level [29]. PolyScope, shown in Fig.2.10, is the graphical user interface (GUI) to operate the robot, run existing robot programs or create new ones [30]. PolyScope runs on the touch sensitive screen attached to the control box. It is possible to connect a mouse and a keyboard to the controller box or the teach pendant. Using PolyScope, individual joints can be moved directly in the 'move joints mode' or the robot can be moved physically by hand in the 'teach mode' . It also has a 'Emergency Stop Button' to halt the robot motion in case of apparent collision or undesired motion.

There are several possilble ways of controlling the UR5 on Script Level like the URScript programming language, Matlab or ROS. In this research, due to the reasons that will be explained in Section 2.3.1 we have chosen to use ROS for Script Level control of UR5.

The third level of control i.e. C-API Level controller can be compiled on the robot itself. However this method is not recommended due to safety concerns.

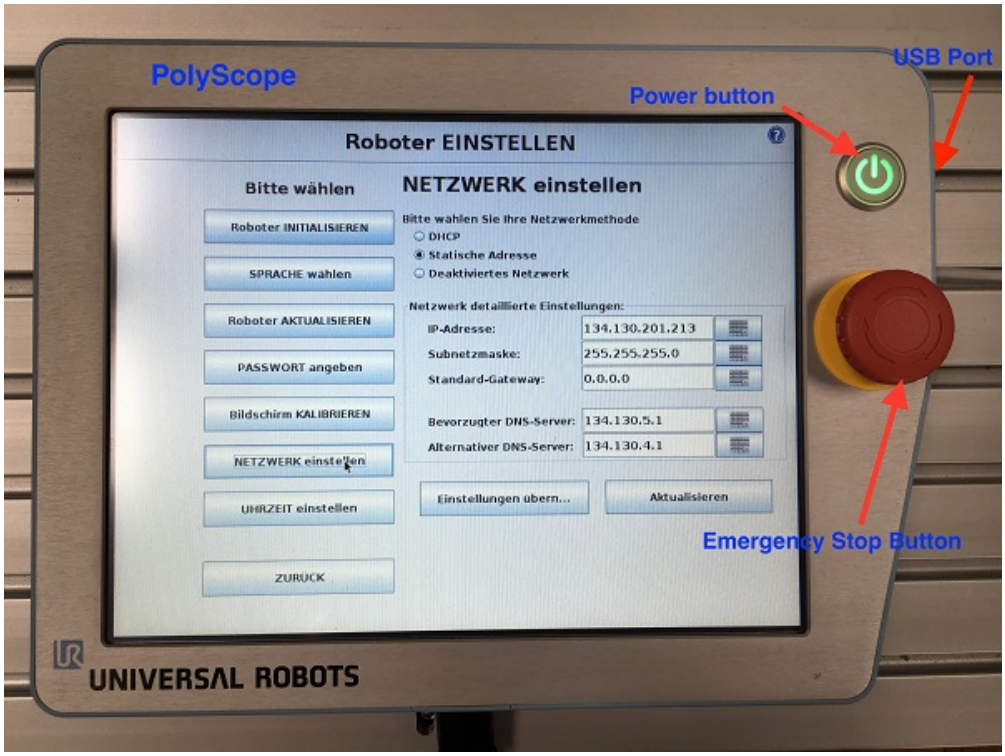



Figure 2.10: Graphical User Interface: PolyScope

## 2.3 Robot Operating System

Robot Operating System (ROS) is not an 'Operating System' in the traditional sense of process management and scheduling [31], but ROS is an open-source software framework providing operating-system like capabilities for controlling robots. It provides an advanced programming environment for controlling low-level hardware. In recent years, the robotics community has made tremendous progress in software development for robots using ROS.

The official description of ROS is [32]: *ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. ROS is similar in some respects to 'robot frameworks,' such as Player, YARP, Orocos, CARMEN, Orca, MOOS, and Microsoft Robotics Studio.*

### 2.3.1 Why ROS?

The main advantages of ROS are modularity, scalability, software reuse, distributed computing, language independence and rapid testing [8]. These are explained below:

- **Modularity**: Using ROS, the robot's software can be divided into small standalone parts that co-operate to accomplish a high level objective.
- **Scalability**: ROS is appropriate for large runtime systems and for large development processes
- **Software Reuse**: Robotics community has developed efficient implementations of many important robotic algorithms such as navigation, motion planning, mapping etc. in ROS, which can be used directly for developing new ideas. This results in a rapid learning curve and less time is spent on re-inventing the wheels.
- **Distributed Computing**: In many robotic tasks, several robots and even humans (through desktops/laptops) have to collaborate and communicate with each other. Sometimes a single robot itself can have multiple computers which individually focus on specific tasks. ROS provides very simple and efficient mechanisms for communication between multiple processes.
- **Language Independence**: ROS framework is easy to implement in any modern programming language such as Python, C++, Lisp, Java and Lua.
- **Rapid Testing**: In many cases, working with physical robots is time consuming and sometimes not possible due to lack of resources. Using ROS we can test high-level algorithms by replacing the low-level control and hardware with simulators. Also the sensor data from actual robots can be recorded and replayed to test various algorithms for processing that data.

### 2.3.2 ROS Architechure

ROS is based on a graph architecture with the fundamental elements: nodes, topics, messages and services [31] as shown in Fig.2.11.

Nodes are the small software blocks or modules where all the processing takes place. All nodes run simultaneously and mostly independent of each other. A robotic system will usually consist of multiple nodes. For example, one node for controlling the motor drivers, one node for handling the encoders, one node for camera feedback, one node for path planning etc. A ROS node is written with the use of a ROS client library, such as *roscpp* or *rospy*.

Nodes communicate with other through messages published on Topics. A message is a data structure which can also be an array of messages nested arbitrarily deep. Nodes

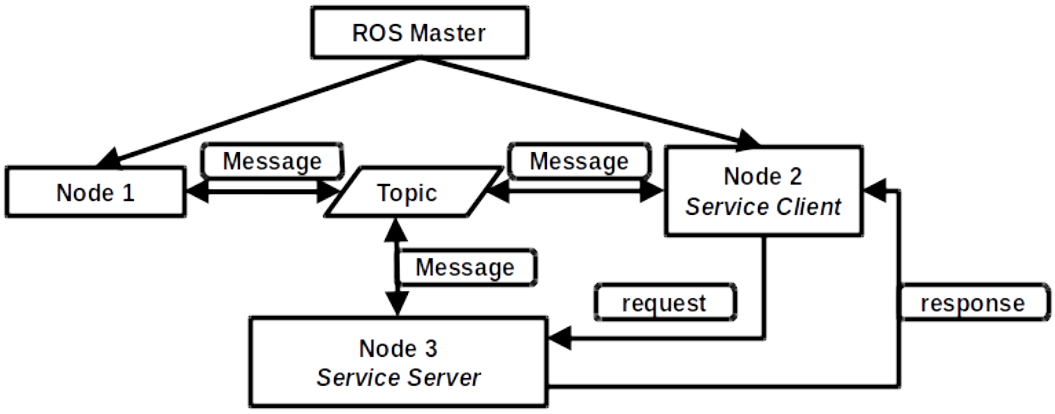



Figure 2.11: Fundamental elements of ROS

can publish (send) messages to a topic or subscribe (receive) messages from a topic. A single node can publish or subscribe to multiple topics. Also a single topic can have multiple publishing or subscribed nodes. In general the nodes are not aware of other publishers/subscribers of topics.

In addition to messages, ROS nodes can get information in the form of 'parameters' stored on a central 'parameter server' . This is useful for storing and fetching configuration parameters of the robot system which mostly remain constant throughout the operation.

Another mode of communication between nodes is called as 'service' . In this routine, a server node advertises a 'service' with a unique name. Client nodes can send a message to the server node as a 'request' and await the 'response' (again in the form of a message) from the server node. Thus this is 'need-based' interaction between nodes whereas the topics have 'broadcast' routine scheme.

The ROS Master node provides naming and registration services to the rest of the nodes in the ROS system. It tracks publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they communicate with each other peer-to-peer.

For starting multiple nodes, including the master, at the same time, a 'launch file' can be created. The collection of executable and supporting files which perform a certain high-level processing task is called as a 'package'. All ROS software is organized

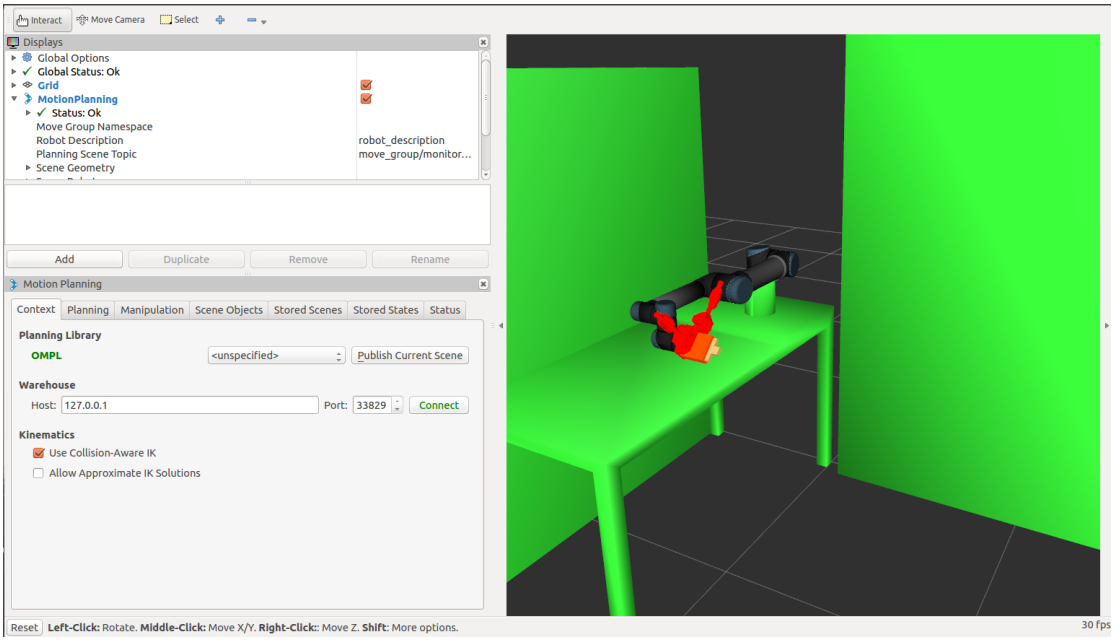



Figure 2.12: ROS visualization (rviz) Interface

in packages and ROS provides several commands for installing new packages and interacting with installed packages.

### 2.3.3 Additional ROS tools

#### Visualizing Data

For displaying sensor data and state information from ROS, a 3D visualizer called rviz (ROS visualization), shown in Fig.2.12, is often used. The robot's current configuration can be visualized on a virtual model of the robot. Also live representations of sensor values published over ROS Topics such as camera data, infra-red distance measurements, sonar data etc. can be displayed in rviz.

#### Simulation with Gazebo

As mentioned in Section2.3.1, ROS allows us to separate the high level algorithm implementations from the low level controllers and drivers on physical robot. So a real robot can be replaced by a powerful simulator such as Gazebo (shown in Fig.2.13) in which the characteristics of the robot and its environment can be modelled. Using Gazebo we can simulate the interactions between the world and the robot in the same way as the physical system. This makes it possible to rapidly design robots, test algo-

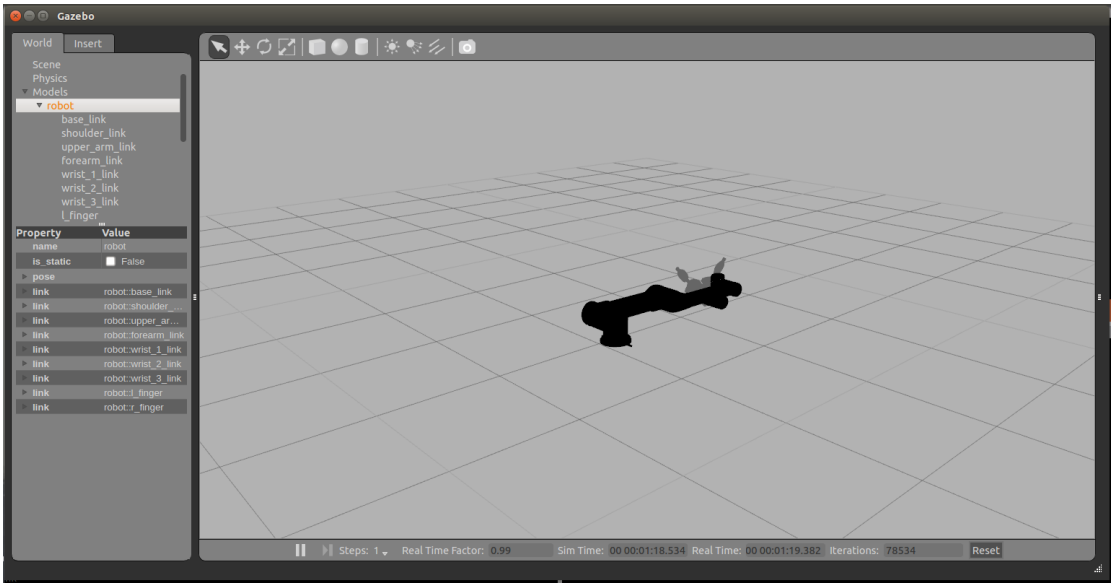



Figure 2.13: Gazebo (Robot Simulator) Interface

rithms, perform regression testing and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. This free high-fidelity simulator has a robust physics engine, high-quality graphics and convenient programming and graphical interfaces.

### ROS-Industrial

ROS capabilities are being extended to robotic applications in manufacturing automation through the ROS-Industrial (ROS-I) project [33]. This open source project includes interfaces for common industrial manipulators, grippers, sensors and device networks with ROS. An international consortium of industry and research members supports the ROS-I project. Some of the current members include industry parteners such as ABB Robotics, Boeing, Bosch, BMW, John Deere, Siemens and academic partners such as Fraunhofer IPA, NTU Singapore, SwRI, UT Austin, TU Delft etc.

ROS Industrial combines the ROS high-level functionality with the low-level reliability and safety of an industrial robot controller. It provides an easy interface to apply cutting-edge academic research to industrial applications by using a common ROS architechure. ROS-I provides Unified Robot Description Formats (URDFs), interface libraries (drivers) and other capabilities (packages) that are useful for manufacturing automation. The Unified Robot Description Format (URDF) is the standard ROS XML representation of the robot model (kinematics, dynamics, sensors).

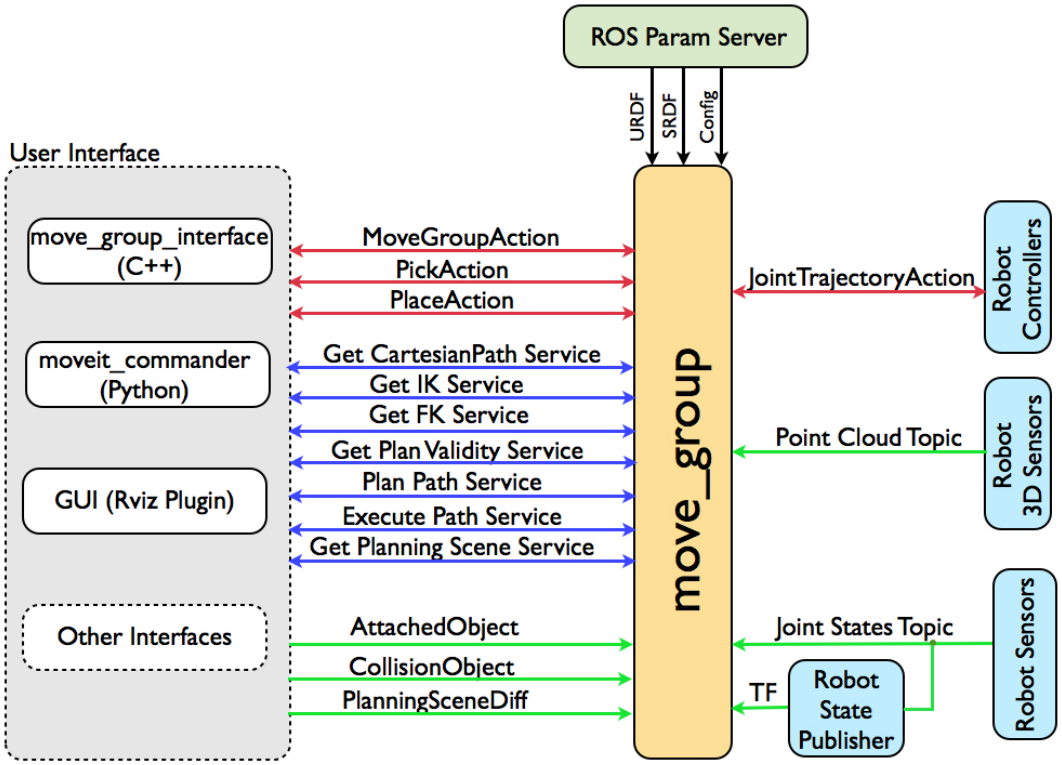



Figure 2.14: MoveIt! system architecture (Figure courtesy [3])

## 2.4 Motion planning framework: *MoveIt!*

The role of a motion planner is to find a feasible and collision free trajectory from current configuration to desired configuration of the robot. 'MoveIt!' is one such state-of-the-art motion planner for manipulation tasks which is widely used in the robotics community. This open source platform can be integrated with the ROS framework and used for evaluating new robot designs, developing advanced robotics applications and building integrated robotics products for industrial, commercial, R&D and other domains. MoveIt! incorporates the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation.

### 2.4.1 MoveIt! System Architecture

Fig.2.14, shows the high level architecture of the *move_group* node. This node is the primary node provided by MoveIt!. It puts all the individual pieces together and provides a set of ROS services and actions for the users to use. The various components of the system architecture are explained below [3]:

1. **User Interface**: There are several possible ways for accessing the actions and services provided by *move_group* node. The most common ways include *move_group_interface* package in C++, *moveit_commander* package in Python or using the Motion Planning plug-in to rviz GUI.

2. **Configuration**: The URDF and SRDF for the robot can be passed to the *move_group* node using the *robot_description* and *robot_description_semantic* parameters respectively on the ROS param server. Other configuration parameters including joint limits, kinematics, motion planning, perception and other information can be stored in the config directory of the corresponding MoveIt! config package for the robot

3. **Robot interface**: The communication between *move_group* node and the physical robot or simulator model is done through ROS topics and actions. The current state of the robot i.e. information about each joint value of the robot, is obtained by subscribing to the */joint_states* topic. Multiple publishers can publish to this topic even with only partial information about the robot state. The joint state publisher has to be implemented on each physical robot or the simulator model. The global information about the robot's pose and other frame transforms are obtained from the ROS TF library through the */tf* topic. The robot needs to have a *robot_state_publisher* node for publishing TF information on */tf* topic. A ROS action interface called *FollowJointTrajectoryAction* is used by *move_group* node to communicate with the controllers on the robot. The server for this action service has to be running on the robot side and *move_group* will act as a client to talk to this controller action server. The representation of the world and any objects carried by the robot (considered to be rigidly attached to the robot) are maintained by *move_group* node using the Planning Scene Monitor.

*move_group* is structured in a modular way. There are separate plugins for individual capabilities like pick and place, kinematics, motion planning. The plugins are configurable using ROS through a set of ROS yaml parameters and through the use of the ROS pluginlib library.

### 2.4.2 Motion Planners

MoveIt! supports many different types of planners such as Open Motion Planning Library (OMPL), Stochastic Trajectory Optimization for Motion Planning (STOMP), Search-Based Planning Library (SBPL), Covariant Hamiltonian Optimization for Motion Planning (CHOMP) etc. The default motion planner integrated in MoveIt! Is OMPL (Open Motion Planning Library). This open-source motion planning library primarily implements randomized motion planners. OMPL has no concept of a robot thus MoveIt! Configures OMPL and provides the back-end for OMPL to work with the problems in Robotics.

### 2.4.3 Collision Checking

The motion planner has to account for self-collisions and collisions of the robot with the environment (including any objects mounted on/picked up by the robot). The collision checking in MoveIt! is mainly carried out using the FCL package and is configured inside a Planning Scene using the *CollisionWorld* object. Different types of objects such as meshes, primitive shapes (boxes, cylinders, cones, spheres and planes), octomap etc. are supported for collision checking in MoveIt!. The computational complexity of collision checking operation is reduced by using the *Allowed Collision Matrix (ACM)* which encodes a binary value corresponding to the possibility of collision between pairs of bodies. If the bodies are far away then this value will be set to 1 and collision check will not be done between these bodies. The motion planner also has to obey user specified kinematic constraints like position/orientation constraints on links, joint constraints etc. MoveIt! offers all these constraints along with the option of additional constraints with a user-defined callback.

### 2.4.4 Time Parametrization

The motion planners usually do not associate any timing information with the generated paths. Thus some post-processing has to be done in order to time parametrize the kinematic trajectories for velocity and acceleration values accounting for the maximum velocity and acceleration limits imposed on individual joints. MoveIt! supports various algorithms for post-processing a kinematic trajectory to add timestamps and velocity/acceleration values. However, at present only *Iterative Parabolic Time Parametrization* is available in MoveIt! which is used by default in the Motion Planning Pipeline.

### 2.4.5 Kinematics

Due to MoveIt!'s modular design, different kinematics plugins can be used with it. The default inverse kinematics plugin for MoveIt! is configured using the *Kinematics and Dynamics Library (KDL)*[34] numerical jacobian-based solver. But other numerical IK plugins like *trac-IK*[35] and *LMA* (Levenberg-Marquardt) can be used. Users can also develop their own analytical IK plugins for MoveIt! using the *IKFast* module [36].

## 2.5 Summary

In this Chapter the different components of overall system for iGPS feedback based motion control of robotic manipulator were described. iGPS (indoor GPS) system can be used for applications of robotics in industrial automation as it offers high accuracy and large workspace volume. Other advantages of iGPS include modularity, scalability, reconfigurability, portability and ease of handling and installation. There are several detectors/probes provided with the iGPS system which can be used for different applications. The detectors receive the laser planes sent by the transmitters and a Position Calculation Engine calculates the position of detector using triangulation. Universal Robot-5 (UR5) manipulator can be used for evaluating the control structures for iGPS feedback based motion control. This lightweight 6DoF robot is ideal for automating low-weight processing tasks. Robot Operating System (ROS) is an open source platform which can be used for Script Level control of UR5 manipulator. ROS architecture is suitable for modularity, scalability, software reuse, distributed computing, language independence and rapid testing. Robotics researchers have developed additional capabilities using ROS such as rviz (for visualizing data), Gazebo (for robot simulation), ROS-Industrial (for interfacing with common industrial robots). MoveIt! is a state-of-the-art open source motion planner which can be used to obtain collision free trajectories that obey user defined constraints. The system architecture of MoveIt! is modular and it supports different plug-ins for motion planning, kinematics, collision checking and time parametrization.

# CHAPTER 3

# Algorithms for iGPS feedback based robot motion control

The general structure of the robot motion control system, without direct feedback of end effector's position and orientation (pose), is shown in Fig.3.1.

The desired pose of the end effector is given by the user in Cartesian space. The motion planner performs interpolations and inverse kinematic transformations to generate a joint space trajectory that will move the robot from current configuration to the desired configuration. A 'trajectory' is different from a 'path' as it includes time parametrization and takes into account the joint velocity/acceleration limits and any other user specified constraints at the joint level. The motion planner tries to generate a trajectory which is free from self-collisions and collisions with other objects in the environment of the robot. If the motion planner is successful in generating such a trajectory, then the target joint space values are sent to the joint controllers. The joint controllers perform the closed loop control of individual motors/actuators with feedback from position/velocity sensors such as encoders attached to the joints. The drive commands from joint controllers are sent to the motors/actuators which move the physical robot or robot's model in a simulator. The joint states feedback from encoders is provided by a joint states publisher.

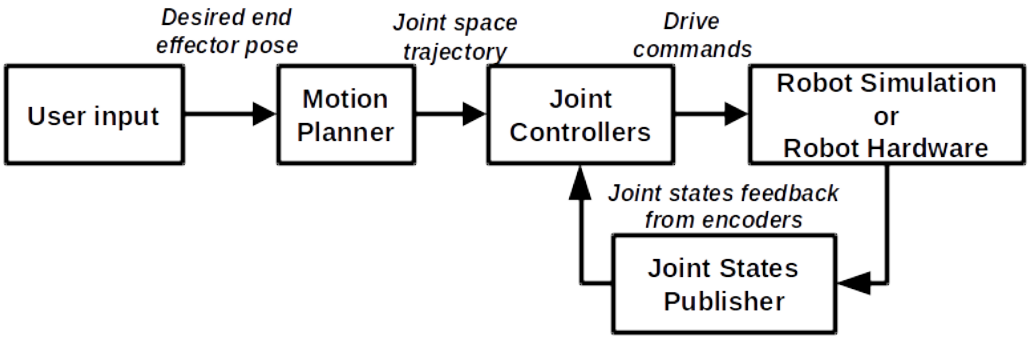



Figure 3.1: Robot motion control system

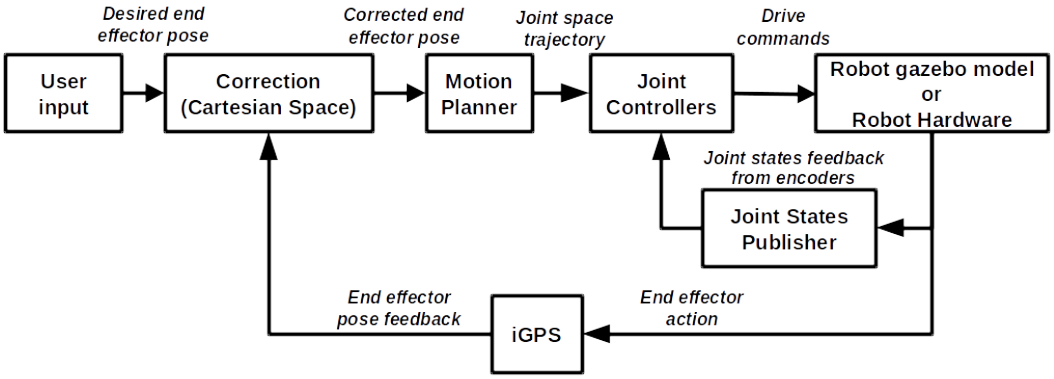



Figure 3.2: Correction before motion planning stage

Three possible ways of including the iGPS feedback in this robot motion control system are:

1. Correction before motion planning stage
2. Correction in joint space trajectory
3. Feedback in joint control loop

These three approaches are explained in detail below.

## 3.1 Correction before motion planning stage

In this approach, the pose of robotic manipulator's end effector obtained from external sensor is used to correct the desired end effector pose given to the motion planner. As shown in Fig.3.2, the motion planner then uses the corrected desired end effector pose to generate a new joint space trajectory. The joint controllers then perform the closed loop control with feedback from joint encoders and drive the robot through the desired configurations.

One possible way of making corrections before motion planning stage is illustrated in Fig.3.3. Here the user input is given in the form of a sequence of desired end effector poses. After reaching each target pose, the end effector pose measurement from iGPS is taken into account.

Let $T_1^d, T_2^d .....T_n^d$ be the sequence of 'n' desired end effector poses given by the user. The actual poses of end effector as measured by iGPS be $T_1^a, T_2^a .....T_n^a$. Let

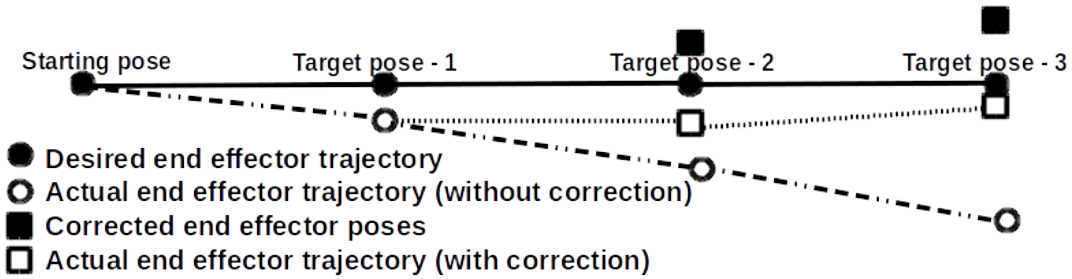



Figure 3.3: Illustration of motion planning stage correction method

$T_1^{d*}, T_2^{d*}.....T_n^{d*}$ be the sequence of corrected end effector poses after applying the correction before motion planning stage. Each pose $T_i \in \mathbb{R}^{4\times4}$ is a homogeneous transformation matrix consisting of a position vector $P_i \in \mathbb{R}^{3\times1}$ and an orientation matrix $R_i \in \mathbb{R}^{3\times3}$ as shown in Eq.3.1.

$$[T_i]_{4x4} = \left[\begin{array}{c|c} [R_i]_{3x3} & [P_i]_{3x1} \\ \hline [0]_{1x3} & [1]_{1x1} \end{array}\right] \tag{3.1}$$

At each goal state '$i$', the error ($\Delta_i$) between measured end effector pose and desired end effector pose is given by Eq3.2

$$\Delta_i = \begin{cases} 0 & \text{if } i = 1 \\ (T_i^{d*})^{-1} T_i^a & \text{if } i > 1 \end{cases} \tag{3.2}$$

The next target pose is modified to compensate for error in the current pose.

$$T_{i+1}^{d*} = T_{i+1}^{d} \Delta_i \tag{3.3}$$

Another possible way of compensating for error in the current pose is to issue a new target pose by adding the error in previous target pose, but the first method is preferred as it doesn't increase the number of target poses.

The 'Correction before motion planning stage' approach does not require any transformation from end-effector co-ordinates to joint space co-ordinates before making the correction. However, the disadvantage of this approach is that a new motion plan has to be generated after each correction, which may add delays to the movement of robotic

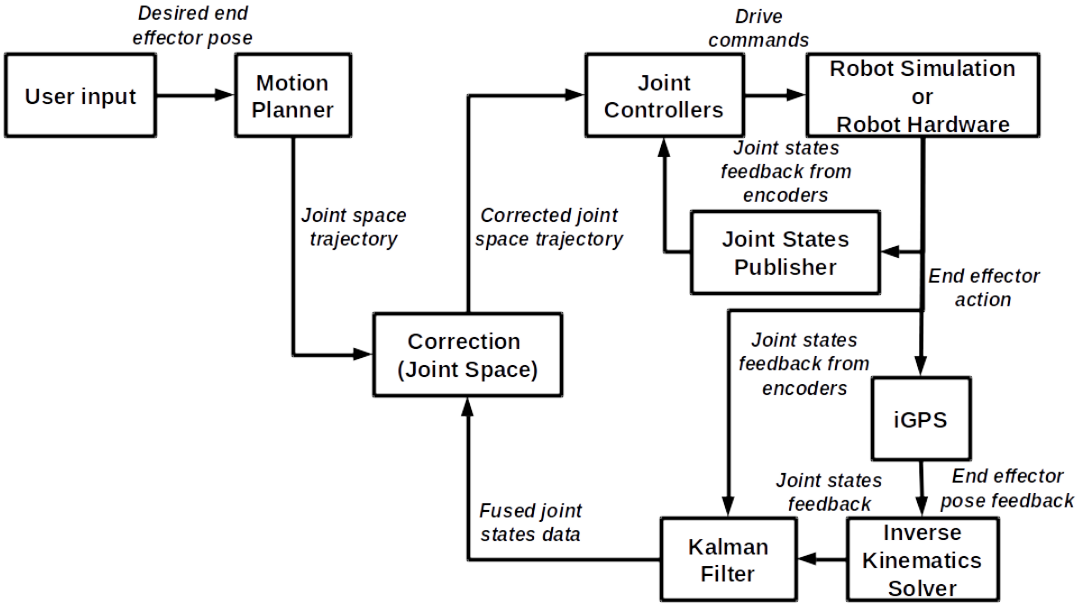



Figure 3.4: Correction in joint space trajectory with Kalman Filter

manipulator. Also it does not correct the end effector pose during the execution of a motion plan. Thus the trajectory followed by end-effector between two target poses will be inaccurate.

## 3.2 Correction in joint space trajectory

In this algorithm, the correction is done after the motion planning stage but before the joint control stage, as shown in Fig.3.4. The joint space trajectory generated by motion planner consists of target joint positions/velocities/accelerations along with time parametrization. At each target joint state, the end effector pose feedback from iGPS is taken into account. The sensor signal is converted from end effector coordinates to joint states using inverse kinematics (IK) transform. The publishing rate of iGPS is 40Hz whereas the joint trajectory goal states are published at 10Hz. The inverse kinematics solver may have inaccuracies in the solutions. Also if there is a loss of line-of-sight between the iGPS probe and transmitters, we will not get any feedback and the robot motion will not be corrected. To overcome these difficulties, the use of a Kalman Filter for fusion of encoder data and iGPS data is proposed. It will be explained in Section 3.4.

Let $J_1^d$, $J_2^d$ ..... $J_n^d$ be the sequence of 'n' joint states calculated by the motion planner

after converting the desired end-effector trajectory to joint space trajectory. The actual joint states from IK transformed data of iGPS be $J_1^a, J_2^a .....J_n^a$. Let $J_1^{d*}, J_2^{d*} .....J_n^{d*}$ be the sequence of end effector poses after applying the correction before motion planning stage.

At each joint goal state 'i' the error ($\delta_i$) between actual joint states and desired joint states is given by Eq.3.4.

$$\delta_i = \begin{cases} 0 & \text{if } i = 1 \\ J_i^d - J_i^a & i > 1 \end{cases} \tag{3.4}$$

The next target joint state vector is modified to compensate for error in the current joint states.

$$J_{i+1}^{d*} = J_{i+1}^d + \delta_i \tag{3.5}$$

## 3.3 Feedback in joint control loop

An alternative approach for end effector feedback is shown in Fig.3.5. In this approach, the iGPS feedback is given to the joint states publisher. The sensor signal is converted from end-effector coordinates to joint states and given to the joint controllers as inputs instead of the encoder feedback through the joint states publisher.

The publishing rate of iGPS is 40Hz whereas the joint control loop runs at 125Hz. At this high sampling rate, the delay due to Inverse Kinematics Solver results in incorrect joint states being published by the joint states publisher. Also if there is a loss of line-of-sight between the iGPS probe and transmitters, the joint controllers will not get any feedback and the robot motion will stop. To overcome these difficulties, again the use of a Kalman Filter is proposed for fusion of encoder data and iGPS data.

As shown in Fig.3.6, the encoder data (published at 125Hz) is fused with joint state values obtained from iGPS feedback (published at 40Hz) using the inverse kinematics solver. Any additional delay due to inverse kinematics solver is also taken into account. The fused data is published at 125Hz through the joint state publisher and given as input to the joint controllers. The algorithm for using Kalman Filter for fusion of multi-frequency and delayed sensor data will be explained in Section 3.4.

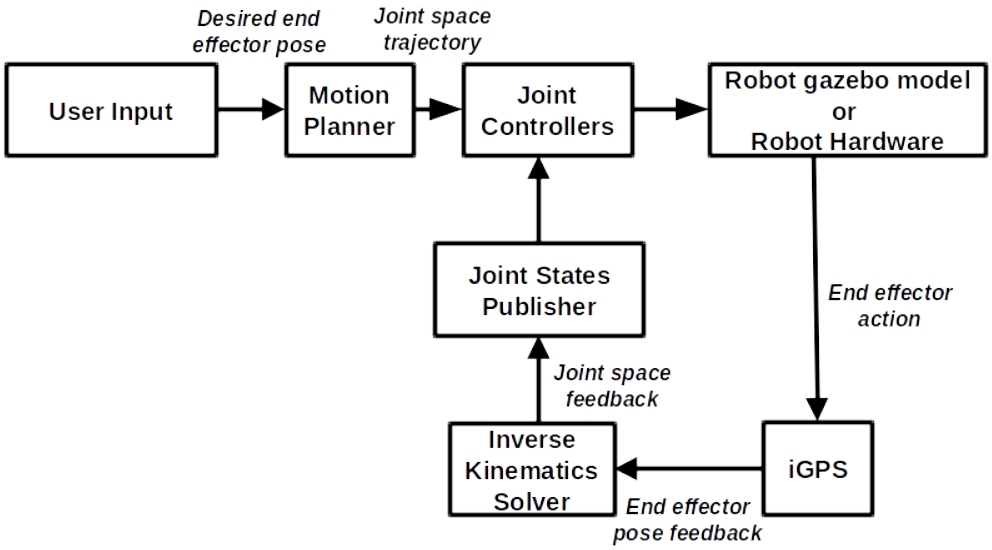



Figure 3.5: Feedback in joint control stage

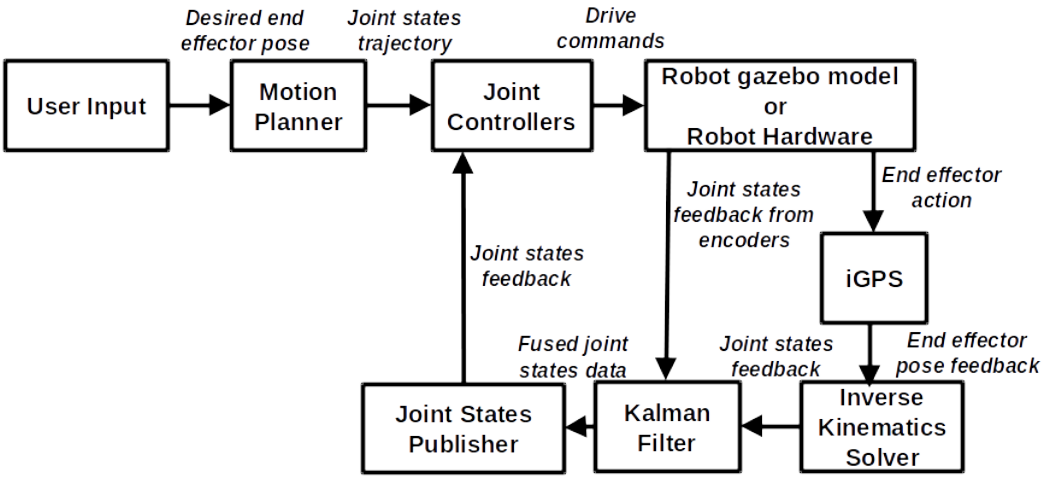



Figure 3.6: Feedback in joint control stage with Kalman Filter

In this approach, the motion plan has to be generated only once and there are no delays in the movement of robotic manipulator. Also, the frequency of the joint control loop is much higher than the motion planning loop thus this approach provides potential for higher bandwidth of the closed loop system. But this method is difficult to implement because it requires the transformation from end-effector coordinates to joint states at a very high rate. Also in some industrial robotic manipulators, the joint controllers and joint states publishers are designed by the manufacturer and not accessible for modifications. In these cases the joint controllers directly use the encoder feedback only and there is no way of supplying corrected joint values as inputs to the joint controllers.

The advantage of this approach is that the required publishing rate of fused joint states is much lower than the joint control loop frequency. Also there is no need to generate a new motion plan at each target point. This approach removes the drawbacks of previous two approaches and is easily implementable on all robotic manipulators, including those who have inbuilt joint controllers and joint states publisher.

## 3.4 Kalman filter for fusion of multi-frequency and delayed sensor data

In the control structures described in Section 3.2 and Section 3.3, the iGPS measurements have to be converted from end effector coordinates to joint states. This requires an inverse kinematics solver which may add delay to the system. Also the desired loop-rate of joint states publisher is higher than the sampling rate of the sensor. Furthermore, in actual implementation we need a method to filter the iGPS data and compensate for any delays or missing data. To overcome these difficulties, we propose the use of Kalman Filter for obtaining the optimal state estimate by combining the encoder data with the end-effector sensor (iGPS) data. The iGPS measurements are first converted to joint states and then fused with the joint state data available from encoders. This allows us to extend the discrete Kalman filter for fusion of this multi-frequency and delayed sensor data.

### 3.4.1 Discrete Kalman Filter

Kalman Filter, as the name suggests, was invented by R.E. Kalman in 1960 [37]. It is a recursive mathematical formulation for obtaining optimal state estimates of a linear system. Optimality here implies that the mean of the squared error between true value and estimated value is minimized. Kalman filter is used for many applications including filtering noisy signals and generating non-observable states. It supports estimation of past, present and future states even without knowing the precise nature of the modelled system. Since Kalman Filter involves solving a linear system of equations it has the ability to calculate the state estimates in real time. Another advantage of the Kalman Filter is that knowledge of a long state history is not necessary since the filter only uses the immediate last state and a covariance matrix that defines the probability of the state being correct. The Discrete Kalman Filter formulation is described below (adapted from [38]).

The linear stochastic differential equation of a discrete-time controlled process is given by:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{3.6}$$

Here the $n \times n$ matrix $A$ relates the state at the previous time step $k-1$ to the state at the current time step $k$, in the absence of either a driving function or process noise. The $n \times l$ matrix $B$ relates the optimal control input $u \in \mathbb{R}^l$ to the state $x \in \mathbb{R}^n$.

The sensor measurement $z \in \mathbb{R}^m$ is related to the state of the system by equation:

$$z_k = Hx_k + v_k \tag{3.7}$$

Here the $m \times n$ matrix $H$ is the model of the sensors and relates the state to the sensor measurement $z_k$. The matrices $A$,$B$ and $H$ are assumed to be constant throughout the process. The random variables $w_k$ and $v_k$ correspond to the process and sensor noise respectively. They are assumed to be independent of each other, white and with zero mean and covariances $Q$ and $R$ respectively. The process noise covariance matrix $Q$ and sensor noise covariance matrix $R$ are assumed to be constant throughout the process.

The mathematical formulation of the Kalman Filter consists of two parts: Time update equations (prediction) Eq.3.8, Eq.3.9 and Measurement update equations (cor-

rection) Eq.3.10, Eq.3.11, Eq.3.12.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.8}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{3.9}$$

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{3.10}$$

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{3.11}$$

$$P_k = (I - K_k H)P_k^- \tag{3.12}$$

The variables in these equations are summarized in Table 3.1.

| | |
|---|---|
| $\hat{x}_k^-$ | Priori state estimate at step $k$ |
| $A$ | Matrix that relates the previous state and current state |
| $B$ | Matrix that relates the previous control input to current state |
| $u_k$ | Control input at step $k$ |
| $P_k^-$ | Priori estimate error covariance at step $k$ |
| $Q$ | Process noise covariance |
| $K_k$ | Kalman gain |
| $H$ | Matrix that relates the state to measurement |
| $R$ | Measurement noise covariance |
| $\hat{x}_k$ | Posteriori state estimate at step $k$ |
| $P_k$ | Posteriori estimate error covariance at step $k$ |
| $I$ | Identity matrix |

Table 3.1: Description of variables in Discrete Kalman Filter equations

The time update equations are used to calculate the state and error covariance estimates for the next time step based on the current state and error covariance estimates. These values are called 'priori estimates'. The measurement update equations are used to incorporate a new sensor measurement to 'correct' the priori estimates and obtain improved 'posteriori' estimates of states and error covariance. Thus the time update equations are predictor equations and the measurement update equations are corrector equations. The final estimation algorithm consists of these predictor-corrector parts as shown in Fig.3.7.

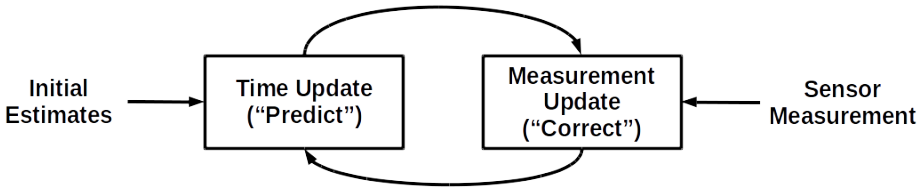



Figure 3.7: Discrete Kalman Filter algorithm

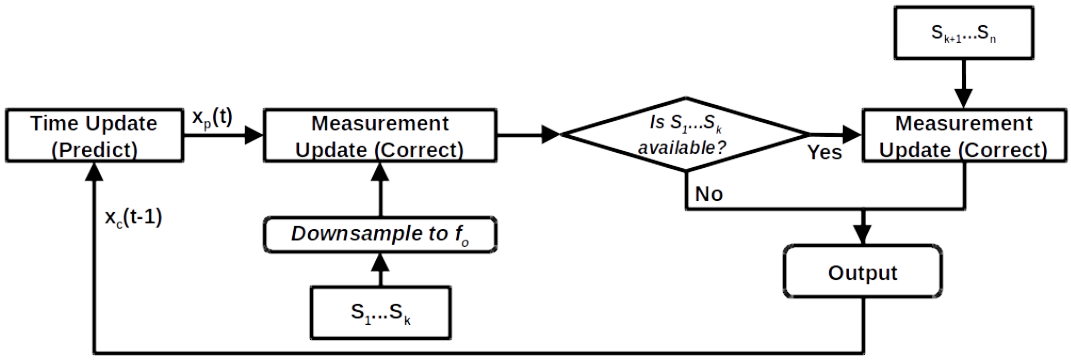



Figure 3.8: Multi-frequency data fusion using Kalman Filter

### 3.4.2 Multifrequency data fusion using Kalman Filter

The Discrete Kalman Filter can be extended to incorporate multiple sensor measurements with different sampling rates [39][40][41]. Let the state of the system be measured by $n$ sensors $S_1$, $S_2$,..$S_k$, $S_{k+1}$..$S_n$ with sampling frequencies $f_1$, $f_2$ ... $f_k$, $f_{k+1}$,... $f_n$. The desired output frequency of the Kalman Filter estimates be $f_o$. Without loss of generality, the sensor frequencies can be arranged in a decreasing order.

$$f_1 \geq f_2 \geq ....f_k \geq f_o \geq f_{k+1} \geq ... \geq f_n \tag{3.13}$$

The sensors are divided into two groups: Group-1 consists of sensors $S_1$..$S_k$ with frequencies greater than the desired output frequency and Group-2 consists of $S_{k+1}$...$S_n$ with frequencies lower than the desired output frequency. For sensors in Group-1, the measurements are down-sampled to $f_o$ and the update equations (correction) are applied at each time-step. For sensors in Group-2, the measurement update is applied whenever the next sensor reading becomes available. The algorithm is summarized in Fig.3.8.

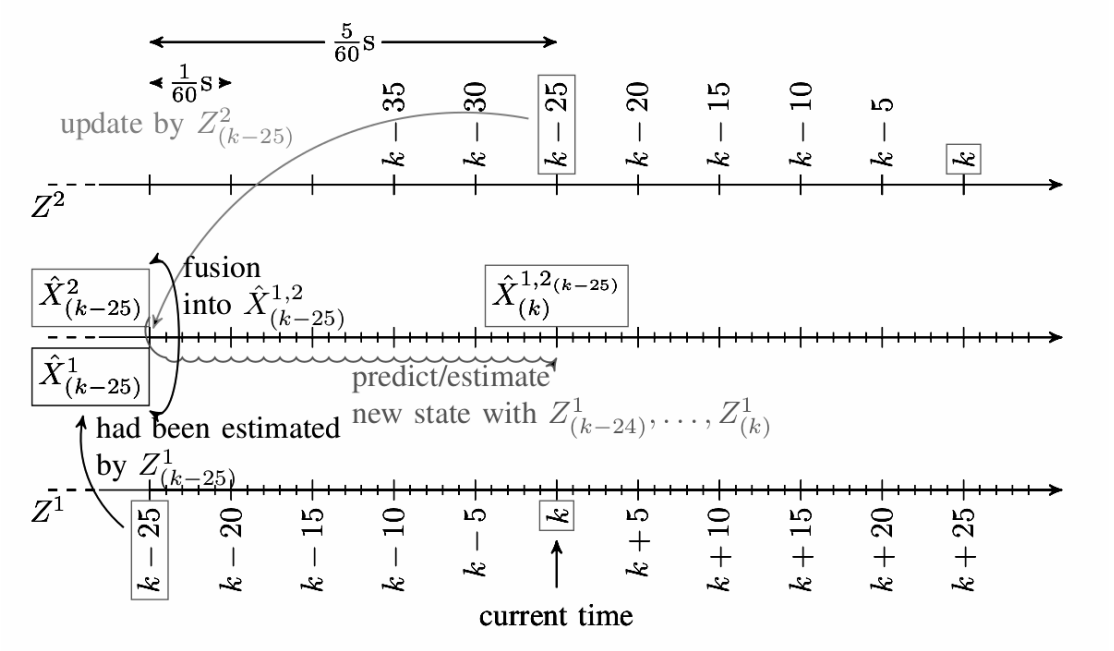



Figure 3.9: Classical method for fusion of delay and non-delay data (Figure courtesy [4])

### 3.4.3 Fusion of multi-frequency and delayed sensor data using Kalman Filter

In some cases, the sensor measurements may have delays due to processing time. Various methods have been developed to use Kalman Filter for sensor data with delays. The common approach [42] is to go back to the time step for which measurement data is available at the current time step and then re-filter again from that time to the current time using all the available sensor data. This approach is shown in Fig.3.9.

For multifrequency and delayed data fusion, the above approach can be combined with the approach described in section 3.4.2. Let the state of the system be measured by $n$ sensors $S_1$, $S_2$, .... $S_n$ with sampling frequencies $f_1$, $f_2$..... $f_n$ and measurement delays $\frac{\delta_1}{f_o}$, $\frac{\delta_2}{f_o}$...$\frac{\delta_n}{f_o}$ where $f_o$ is the desired output frequency of the filter. Again the sensors can be divided into two groups Group-1 and Group-2 as described in section 3.4.2. Without loss of generality we arrange the sensors with increasing order of corresponding delays:

$$0 \leq \delta_1 \leq \delta_2 \leq ..... \leq \delta_n \tag{3.14}$$

For a sensor $S_k$ if the measurement frequency $f_k$ is greater than $f_o$ then at each step $t$, update $t - \delta_k$ step with current sensor reading and apply Kalman Filter prediction/cor-

rection from $t - \delta_k$ to $t$ with all available inputs at each step. If $f_k < f_o$, then at step $t$ when sensor data is available, update $t - \delta_k$ step with current sensor reading and apply Kalman Filter prediction/correction from $t - \delta_k$ to $t$ with all available inputs at each step. Thus we only need to store at max $\delta_n$ number of readings of each sensor.

## 3.5 Summary

The robot motion control system consists of a motion planner, joint controller and joint states publisher. To include the direct end-effector pose feedback from iGPS in the motion control system, three possible approaches were discussed in this chapter. The first approach involves correction before the motion planning stage. This may add delays to the robot motion since a new motion plan has to be generated after every correction. Also it does not ensure accurate trajectory of end effector between two desired poses. In the second approach, iGPS feedback is transformed from end-effector co-ordinates to joint states using an inverse kinematics solver. Then the correction is applied in the joint space trajectory after the motion planning stage but before the joint control stage. In the third approach, the feedback is given to the joint controllers directly, after inverse kinematics transform. This approach can be used for following the trajectory accurately but it is difficult to implement. To improve the robustness of above approaches, the use of Kalman Filter for fusion of encoder and iGPS data was proposed. The formulation of Discrete Kalman Filter was described and extended to fusion of multifrequency delayed sensor data.

# CHAPTER 4

# Experimental Setup for iGPS feedback based control of UR5 manipulator using ROS

As described in Chapter2, the UR5 is a 6 DoF robot arm developed by Universal Robots group. Universal Robots group has been working on an experimental package which is released through the ROS-Industrial project. In this project, the basic functionalities included in the *universal_robot* stack [43] are used and more functionalities are added to integrate the UR5 in a collaborative environment with the iGPS feedback. The entire framework is shown in Fig.4.1. The algorithms described in Chapter3, are implemented in the ROS framework for experimental validation. Even though the UR5 is chosen as a test-bed, the implementation can be easily extended to any other industrial robot due to the modular nature of our code. This chapter will first describe the UR5 manipulator control procedure using ROS without iGPS feedback. Then the implementation of each of the three algorithms for iGPS feedback based control of UR5 using ROS will be explained.

## 4.1 Motion control of UR5 manipulator using ROS

The *universal_robot* package is a part of the ROS-Industrial program [43]. It provides ROS nodes for communication with Universal's industrial robot controllers (*ur_driver*)

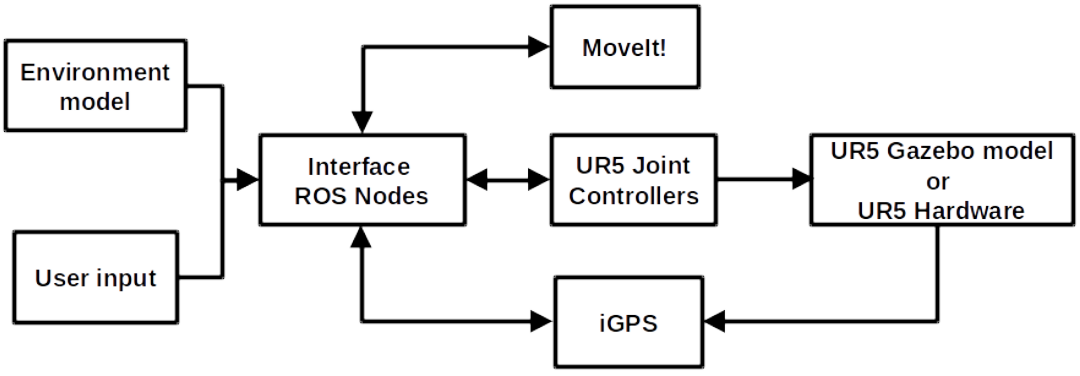



Figure 4.1: iGPS feedback based control of UR5 manipulator using ROS

| Joint | Position Limits $(rad)$ | Velocity Limits $(rad/sec)$ |
|---|---|---|
| Shoulder Pan | $-\pi$ to $\pi$ | 3.15 |
| Shoulder Lift | $-\pi$ to $\pi$ | 3.15 |
| Elbow | $-\pi$ to $\pi$ | 3.15 |
| Wrist-1 | $-\pi$ to $\pi$ | 3.2 |
| Wrist-2 | $-\pi$ to $\pi$ | 3.2 |
| Wrist-3 | $-\pi$ to $\pi$ | 3.2 |

Table 4.1: UR5 joint limits

and URDF models for various robot arms and associated MoveIt! packages. The researchers at 'Center of Advanced Robotics (COAR)' group at Institut fÃijr Getriebetechnik und Maschinendynamik, RWTH Aachen (where this project work was carried out) have developed an interface (*move_group_interface*) between the *move_group* node and *ur_driver* node to extend MoveIt!'s capabilities to UR5. It should be noted that MoveIt! is only able to solve the motion planning request for joint limited case of the robot. The joint limits are listed in Table4.1.

One of the most important things in path planning is to avoid collisions with objects which could appear in the workspace of the robot. MoveIt! can perform the collision checks given that the environment model is provided to it. COAR researchers have developed the following architecture to simplify and integrate the object detection and avoidance process in the path planning process.

The main problem has been divided into two parts, fixed objects and random objects. In the first case, a node is created to declare different kinds of fixed objects in the workspace. Another node similar to the previous one is created for random objects. This node maps the new objects added to the workspace or attached to the manipulator with high frequency, so that these objects are integrated into the object avoidance procedure.

Another added capability of *move_group_interface* at COAR includes the *gripper_action* service. This is especially useful for pick/place tasks using a simple gripper with 2 fingers. The fingers of the gripper can be commanded to move to any desired position within the joint range.

There are three possible modes of user inputs that can be given to control the UR5 manipulator using *move_group_interface*:

1. PoseMoveInterface: The desired end effector position and orientations (poses) can be specified by the user in a .txt file. The *move_group_interface* will read one pose at a time and provide it to MoveIt! for motion planning. If the motion plan

request is successful, then *ur_driver* will be command the actual robot or Gazebo model to the desired pose. After completing this action, the next target pose will be attempted and this process will continue till the last user defined pose is reached. In this case, the path followed by end-effector between two poses cannot be specified and it may not be linear.

2. JointMoveInterface: In this mode, the user directly specifies the target angular positions for each joint. The *move_group_interface* will read one set of values at a time and provide it to MoveIt! for motion planning. If the motion plan request is successful, then *ur_driver* will command the motors of actual robot or Gazebo model to the desired angular values. After completing this action, the next set of target angular positions will be attempted and this process will continue till the last set of user specified joint values.

3. WaypointsMoveInterface: The user can specify a Cartesian path that the end-effector of UR5 should follow. This is done by specifying a list of way-points along the desired path. The initial pose (start state) does not need to be added to the way-point list but adding it can help with visualizations. The *move_group_interface* will read all the way-points at once and supply the list to MoveIt! for motion planning. MoveIt! will attempt to find a Cartesian path through those way-points. If the motion plan request is successful, then *ur_driver* will command the actual robot or Gazebo model to follow the desired trajectory. In this case, the path followed by end-effector is linear between two way-points.

## 4.2 UR5 motion control with iGPS feedback using ROS

For direct measurement of end-effector's pose using iGPS, two i5 probes are mounted on the gripper attached to the wrist-3 joint of UR5. Each i5 probe is able to provide 5DoF data about the end-effector's pose and therefore a frame-of-frames is constructed using two i5 probes in the Surveyor system for getting 6DoF measurements. The experimental set-up is shown in Fig.4.2.

A ROS node was developed to publish the co-ordinate frames of iGPS probes and transmitters to the */tf* topic. The reference frame for these values is set up by using the i6 probe. The *move_group_interface* node listens to this topic for getting the end-effector's pose. This direct feedback of end-effector's pose is used for accurate motion control of UR5 manipulator by using the three algorithms described in Chapter3.

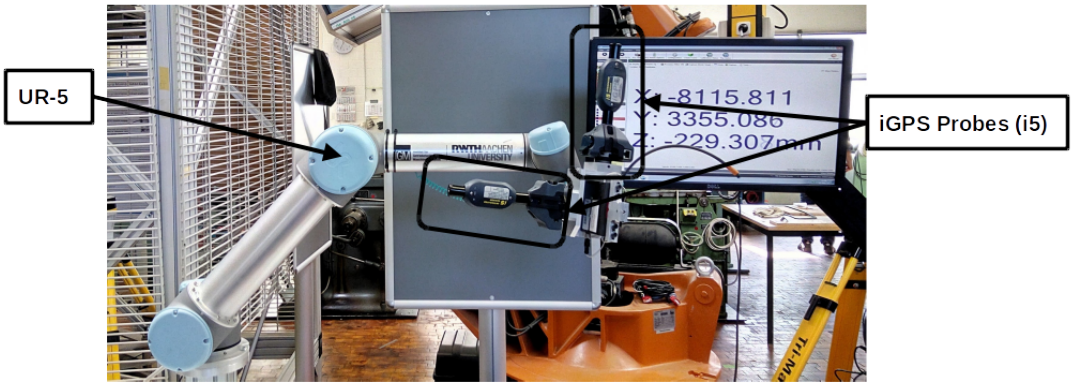



Figure 4.2: Experimental setup with UR5 manipulator and two i5 probes

### 4.2.1 Correction before motion planning stage

As explained in Section3.1, the first approach for iGPS based accurate motion control of UR5 manipulator involves correction before the motion planning stage. The pose values given by i5 frame-of-frames are converted from iGPS world frame to the UR5 world frame using the transformation matrix $[\tau]^{ur5}_{iGPS} \in \mathbb{R}^{4\times4}$. Let the pose reported by i5 probes in iGPS world frame be $T^{iGPS} \in \mathbb{R}^{4\times4}$ consisting of a position vector $P^{iGPS} \in \mathbb{R}^{3\times1}$ and an orientation matrix $R^{iGPS} \in \mathbb{R}^{3\times3}$ as shown below:

$$T^{iGPS} = \left[\begin{array}{c|c} [R^{iGPS}]_{3x3} & [P^{iGPS}]_{3x1} \\ \hline [0]_{1x3} & [1]_{1x1} \end{array}\right] \tag{4.1}$$

Then the pose in UR5 world frame $T^{ur5}$ is given by:

$$T^{ur5} = [\tau]^{ur5}_{iGPS} T^{iGPS} \tag{4.2}$$

The UR5 is commanded using the PoseMoveInterface to move through a sequence of poses. At each destination pose, the error between actual pose values given by iGPS and the desired pose, is calculated by using Eq.3.2 and substituting $T^a_i = T^{ur5}$. The next target pose is compensated for this error by using Eq.3.3.

### 4.2.2 Correction in joint trajectory

In this approach, the iGPS correction is done after the motion planning stage but before the joint control stage. The iGPS values are converted from end-effector coordinates to joint states using inverse kinematics solvers. Two inverse kinematics solvers were tested in this approach:

1. *ur_kinematics*: This is an analytical IK solver developed by Hawkins [44] specifically for the UR manipulators using the closed form inverse kinematics solutions. However, this solver requires the end effector pose in base link frame of UR5 which is difficult to obtain using the iGPS probes. Also this solver gives all possible solutions without considering the joint limits listed in Table4.1.
2. *trac_ik*: Trac-IK [35] is an inverse kinematics solver developed by Traclabs that gives more reliable solutions than common available open source IK solvers. This faster, significantly more reliable replacement for KDL's pseudoinverse Jacobian solver works for any robot chain defined in a URDF file. Also it is able to give IK solutions within the joint limits specified in URDF. It has different modes of solutions:
   - *speed* : Returns first IK solution
   - *distance* : Returns closest IK solution to seed values
   - *manip1* : Returns solution that maximizes $\sqrt{det(J * J^T)}$
   - *manip2* : Returns solution that minimizes $cond(J) = |J| * |J^{-1}|$

We use the *distance* mode and give the encoder values as the seed values. The IK solver runs for the full cycle and gives the closest solution to these seed values. This data is fused with the encoder data using a Kalman Filter. The output frequency of the Kalman Filter depends on the time interval between subsequent joint trajectory states generated by MoveIt!. Usually the interval is between 0.1s to 0.2s. So the output frequency of 10-20 Hz is sufficient for this approach to work.

This algorithm can be used for both *PoseMoveInterface* and *WaypointsMoveInterface*. In both cases, the motion plan consists of target joint positions/velcities/accelerations along with time- from-start. At each joint trajectory point, the error between actual values from Kalman Filter output and target values given by MoveIt!, is calculated using Eq.3.4. The next target joint trajectory set-point is compensated for this error using Eq.3.5. The corrected joint trajectory values are published on the *FollowJointTrajectory* action server. The joint trajectory controller performs interpolations between those joint trajectory points using linear (if only joint positions are specified) or cubic (if joint

positions and velocities are specified) or quintic (if joint positions, velocities and accelerations are specified) splines and executes the trajectory as well as the robot hardware allows.

### 4.2.3 Feedback in joint control loop

In this approach, the iGPS feedback has to be given directly to the joint controllers. For this the iGPS values are transformed from end effector frame to joint co-ordinate frames using one of the inverse kinematics solvers listed in Section 4.2.2. The joint control loops run at $125Hz$ and the encoder data is also published at the same frequency but the iGPS values are published at $40Hz$. So we use a Kalman Filter for fusion of multifrequency data with delays as explained in Section3.4.2. *move_group_interface* node then publishes the fused values to a topic */corrected_joint_states*. *move_group* node of MoveIt! is configured to use these values instead of the values published on */joint_states* topic by the encoders. However, the inbuilt robot joint controllers cannot be modified to use these corrected values instead of the encoder values. Thus this approach cannot be tested on the actual UR5 robot.

## 4.3 Summary

The interface between Universal Robot's industrial robot controllers and motion planner MoveIt! was developed to control UR5 manipulator using ROS. Additional capabilities include gripper action and environment model for collision avoidance. The robot can be commanded in end-effector co-ordinates mode, joint co-ordinates mode or cartesian trajectory mode. Experimental set-up was built using two iGPS probes for getting 6DoF measurements of UR5 manipulator's end-effector. Further, the three algorithms for iGPS feedback based accurate control of UR5 were implemented in ROS framework. For transformation between end-effector co-ordinates to joint states, two different inverse kinematics solvers were included in the code. The analytical IK solver ur_kinematics is faster and accurate but does not consider joint limits and requires end-effector pose in base link frame. Also it is only available for Universal's robots. The other IK solver trac_ik is more versatile as it can be used for any URDF and it obeys the specified joint limits.

# CHAPTER 5

# Results

In this chapter, the results of experiments conducted for evaluating the ROS code for 'iGPS feedback based robot motion control' on UR5 model in Gazebo and physical UR5 robot are presented. First, comparison between the accuracy of iGPS measurement and a camera system will be shown. Then the results of Kalman Filter for fusion of multi-frequency and delayed data will be described. Finally the comparison between end-effector trajectory with and without iGPS correction will be presented.

## 5.1 Comparison of iGPS and Camera

As explained in Section1.4, vision based approaches have been extensively researched and there are several open source implementations of state-of-the art object tracking algorithms. ViSP (Visual Servoing Platform) is one such library that allows prototyping and developing applications using visual tracking and visual servoing techniques developed by Inria Lagadic team [45]. We used the visp_auto_tracker ROS package that wraps model-based trackers provided by ViSP visual servoing library [46] to get the 3D position and orientation of a stationary object. A QRcode of Flash code pattern as shown in Fig.5.1a, was attached to the object. 'Microsoft Kinect for xbox 360' sensor (shown in Fig.5.1b) was used for vision feedback. ROS packages libfreenect [47] and freenect_launch [48] were used to access the Kinect sensor data.


(a) QR-code for automatic object tracking (b) Microsoft Kinect Xbox 360 sensor

Figure 5.1: Object tracking using camera

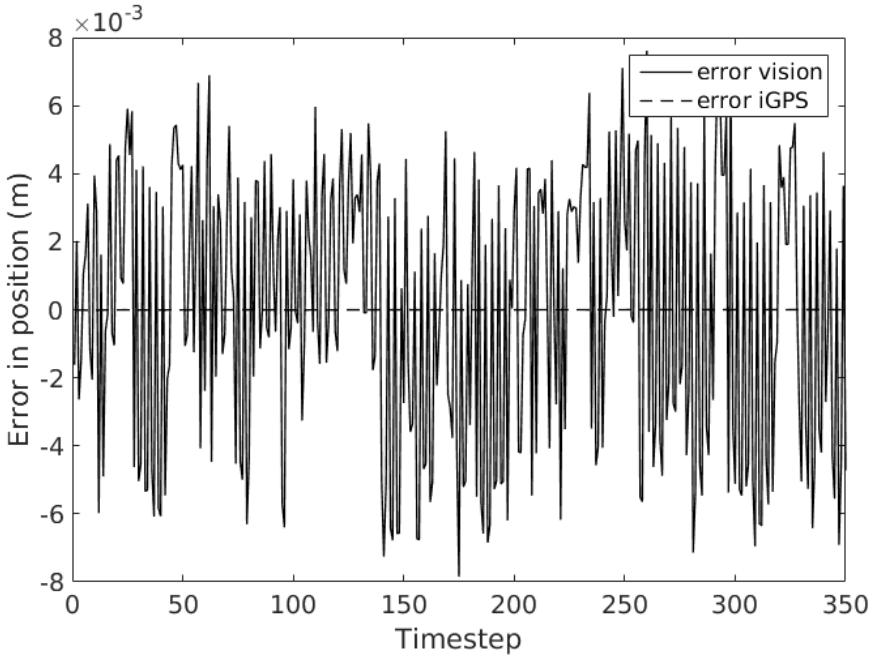



Figure 5.2: Static position measurement error: Comparision between iGPS and Kinect

| Parameter | iGPS | Kinect |
|---|---|---|
| Frequency | 40-50Hz | 10-30Hz |
| Range | 3-35m | 0.8-6m |
| RMS error | 0.2mm | 7.9mm |

Table 5.1: Comparison between iGPS and Kinect

The visp_auto_tracker is able to automatically detect the object using the QR code shown in Fig5.1a. The detection allows to initialise the model-based trackers. In case the tracking is lost, a new detection is performed that will be used to re-initialize the tracker. However, through our experiments it was found that this re-initialization takes place only when the object is placed at a certain distance from the camera. So whenever the tracking was lost, the object had to be taken closer to the camera for re-initialization of tracking algorithms.

The static position measurement errors of Kinect and iGPS are shown in Fig.5.2. The results are tabulated in Table5.1. Along with the re-initialization issue, Kinect based tracking method performs poorly in comparison to iGPS in terms of static position measurement accuracy, range and measurement frequency.

| Parameter | Value | Parameter | Value |
|---|---|---|---|
| A | I | B | Z |
| H | I | Q | 1e-2*I |

Table 5.2: Input parameters for Kalman filter tests (I=Identity matrix; Z = Zero matrix)

| Parameter | Value | Parameter | Value | Parameter | Value |
|---|---|---|---|---|---|
| $R_{sensor1}$ | 1e-2*I | $\delta_{sensor1}$ | 0 | $f_{sensor1}$ | $f_{output}$ |

Table 5.3: Sensor properties for discrete Kalman filter test (I=Identity matrix)

## 5.2 Kalman Filter Tests

The formulation for fusion of multifrequency and delayed sensor data using Kalman Filter, as described in Section3.4, was tested on manually generated random sensor data in Matlab. Then the formulation was implemented in ROS and the Kalman Filter was tuned for UR5 encoders and IK solutions obtained from iGPS data. This section will present the results of these tests. Table5.2 lists the values of input paramters used for Kalman Filter tests.

### 5.2.1 Discrete Kalman Filter

Kalman Filter code was tested on readings from a single sensor. Fig.5.3 shows the plot of sensor data, true value and Kalman filter output. Table5.3 lists the sensor parameters.

### 5.2.2 Fusion of multifrequency sensor data using Kalman Filter

Code for multi-frequency data fusion using Kalman Filter, was tested to fuse the data from three sensors measuring the same true value with different sampling frequencies. Fig.5.4 shows the plot of input sensor data, true value and fused KF output. Table5.4

| Parameter | Value | Parameter | Value | Parameter | Value |
|---|---|---|---|---|---|
| $R_{sensor1}$ | 1e-2*I | $\delta_{sensor1}$ | 0 | $f_{sensor1}$ | $f_{output}$ |
| $R_{sensor2}$ | 1e-2*I | $\delta_{sensor2}$ | 0 | $f_{sensor2}$ | $0.5 * f_{output}$ |
| $R_{sensor3}$ | 1e-2*I | $\delta_{sensor3}$ | 0 | $f_{sensor3}$ | $0.25 * f_{output}$ |

Table 5.4: Sensor properties for multi-frequency data fusion using Kalman filter test (I=Identity matrix)

| Parameter | Value | Parameter | Value | Parameter | Value |
|---|---|---|---|---|---|
| $R_{sensor1}$ | 1e-2*I | $\delta_{sensor1}$ | 0 | $f_{sensor1}$ | $f_{output}$ |
| $R_{sensor2}$ | 1e-2*I | $\delta_{sensor2}$ | $5/f_o$ | $f_{sensor2}$ | $0.5 * f_{output}$ |
| $R_{sensor3}$ | 1e-2*I | $\delta_{sensor3}$ | $10/f_o$ | $f_{sensor3}$ | $0.25 * f_{output}$ |

Table 5.5: Sensor properties for multi-frequency and delayed data fusion using Kalman filter test (I=Identity matrix)

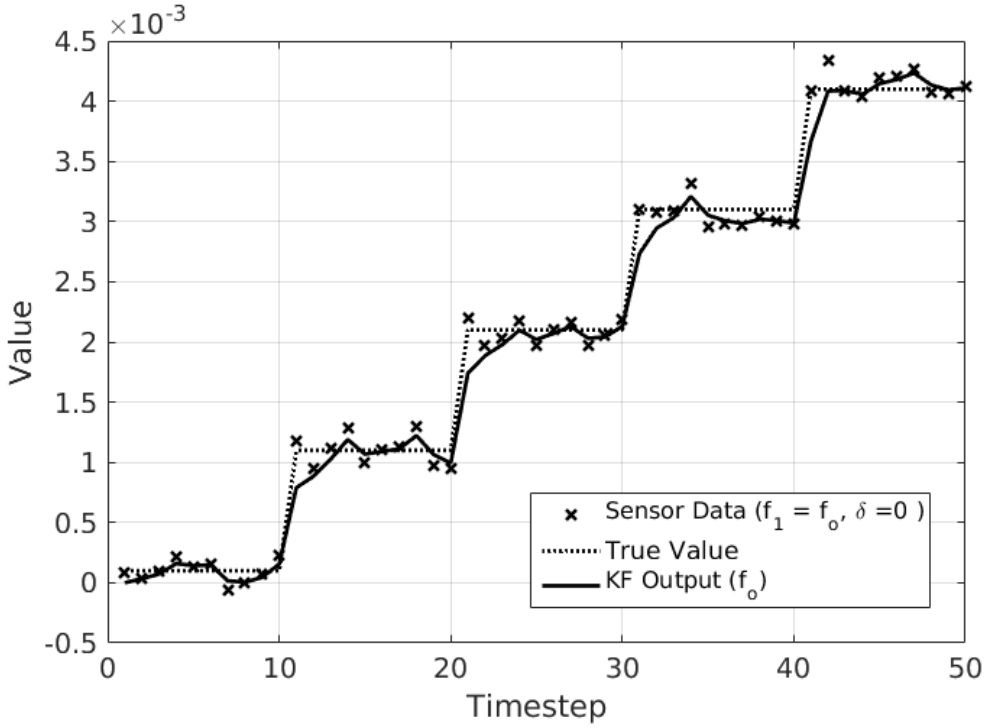



Figure 5.3: Discrete Kalman Filter

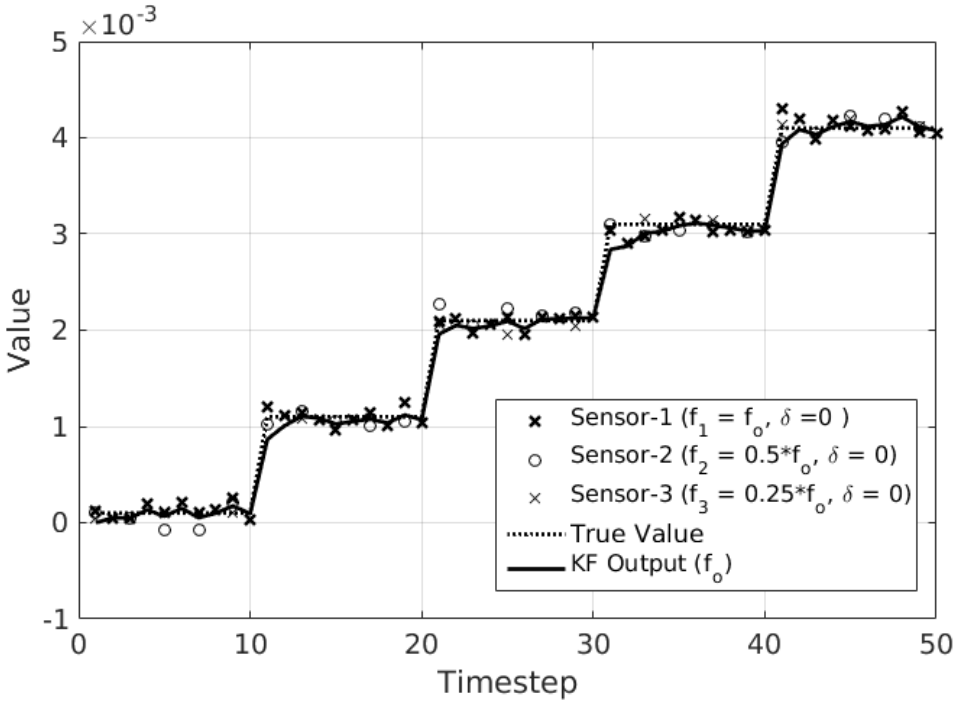



Figure 5.4: Multifrequency sensor data fusion using Kalman Filter

lists the sensor properties.

### 5.2.3 Fusion of multi-frequency and delayed sensor data using Kalman Filter

Code for fusion of multi-frequency and delayed sensor data using Kalman Filter, was tested to fuse the data from three sensors measuring the same true-value with different sampling frequencies. In addition, two of the sensors gave the output with different delays. Fig.5.5 shows the plot of input sensor data, true value and fused KF output. Table5.5 lists the sensor properties.

### 5.2.4 Kalman Filter tuning for UR5 and iGPS

The measurement noise covariances $R_{encoder}$ of the UR5 encoders were measured in a static configuration. Similarly, the covariances of IK solutions of iGPS measurements $R_{igps}$ were measured in a static configuration of UR5. The value of process noise covariance $Q$ was then tuned to get best possible filter performance by keeping the measurement noise covariances constant. It was observed that for a constant $R$, reduc-

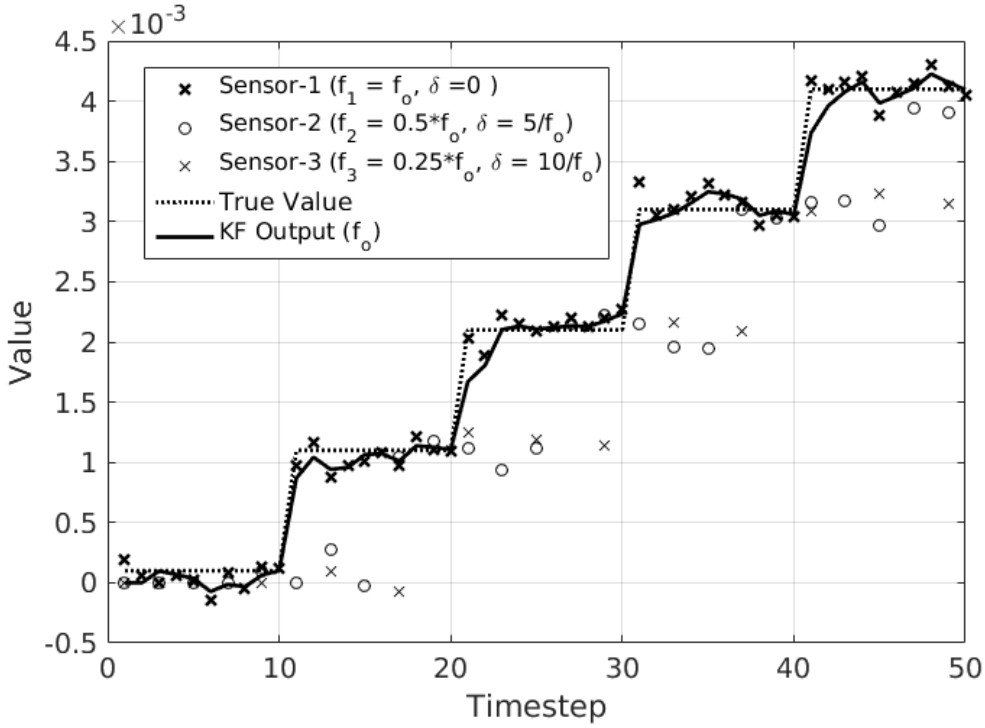



Figure 5.5: Multifrequency and delayed sensor data fusion using Kalman Filter

| Parameter | Value | Parameter | Value | Parameter | Value |
|---|---|---|---|---|---|
| $R_{encoder}$ | 1e-9 | $\delta_{encoder}$ | 0 | $f_{encoder}$ | 125Hz |
| $R_{igps}$ | 1e-6 | $\delta_{igps}$ | 0 | $f_{igps}$ | 40Hz |

Table 5.6: Properties of UR5 encoder data and iGPS data (IK solutions)

ing $Q$ reduces the output covariance but increased the filter response time. Increasing the value of $R$ for a constant $Q$ reduced the output covariance but increased the filter response time. Table 5.6 lists the parameters of UR5 encoders and iGPS (IK solutions).

Fig.5.6 and Fig.5.7 show the Kalman Filtered output of joint angle and velocity measurements reported by encoder on UR5. Fig.5.8 shows the Kalman filtered output of joint angles obtained from IK solver using the iGPS measurements of end-effector's pose. Fig.5.9 shows the fused output of iGPS and encoder data. (In all these cases the value of $Q$ is $1e$-06)

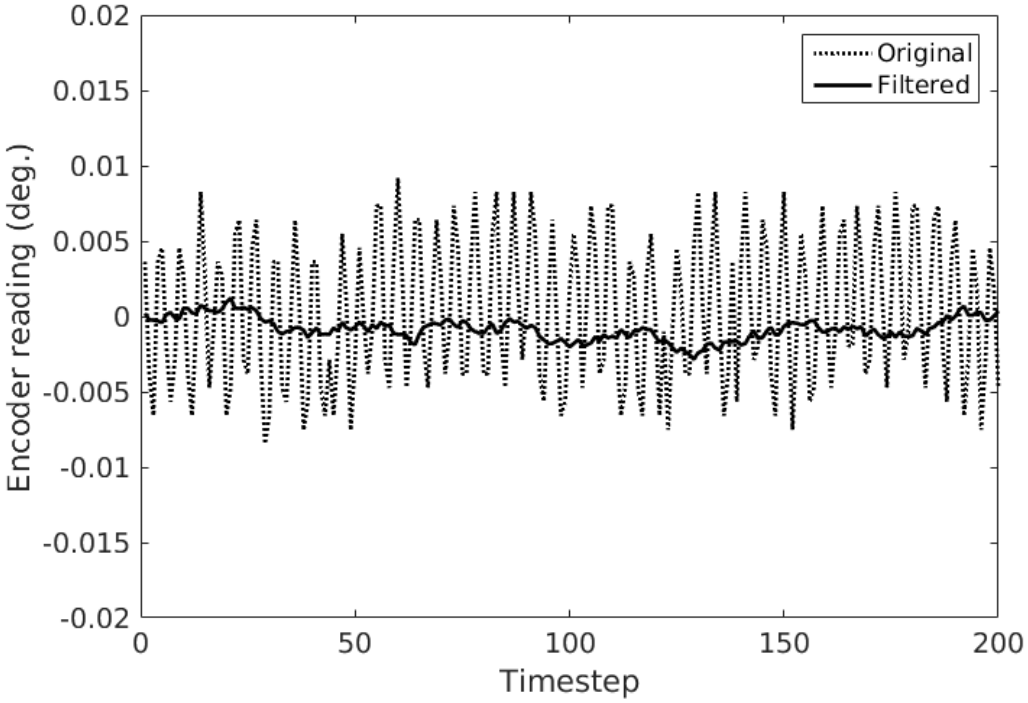



Figure 5.6: Kalman Filter output of encoder joint angle measurements

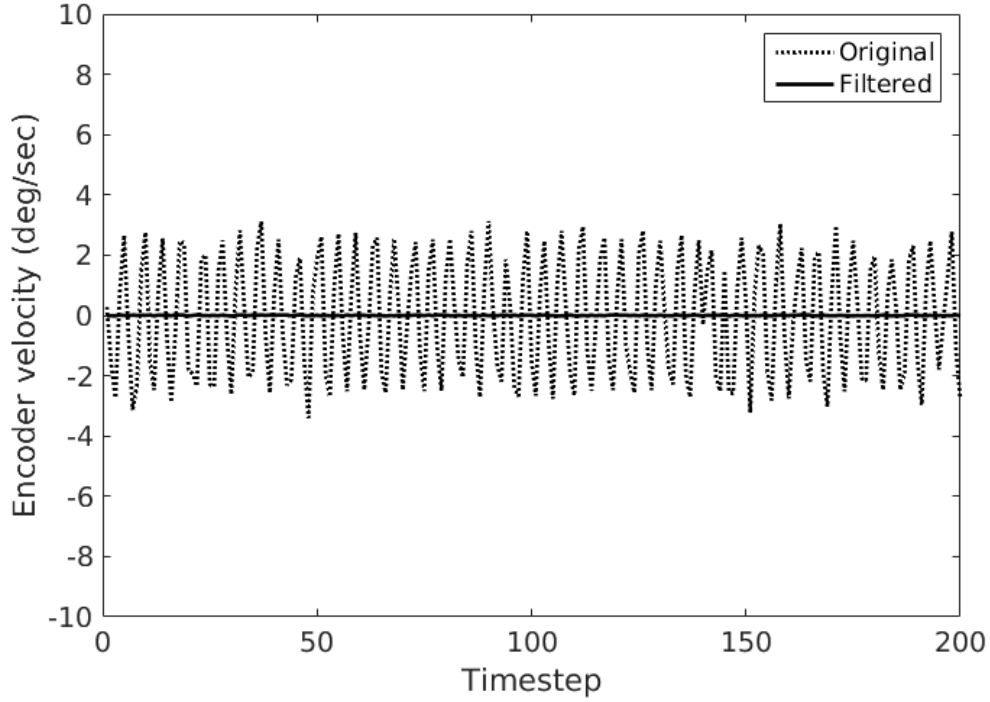



Figure 5.7: Kalman Filter output of encoder joint velocity measurements

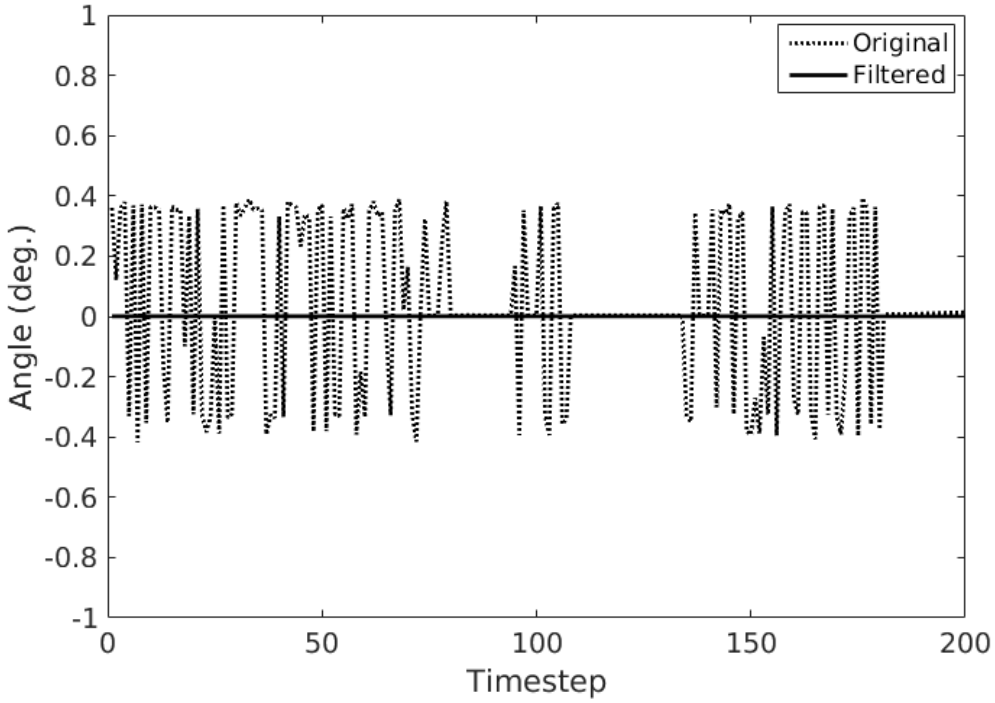



Figure 5.8: Kalman Filter output of IK joint angle solution of iGPS measurements

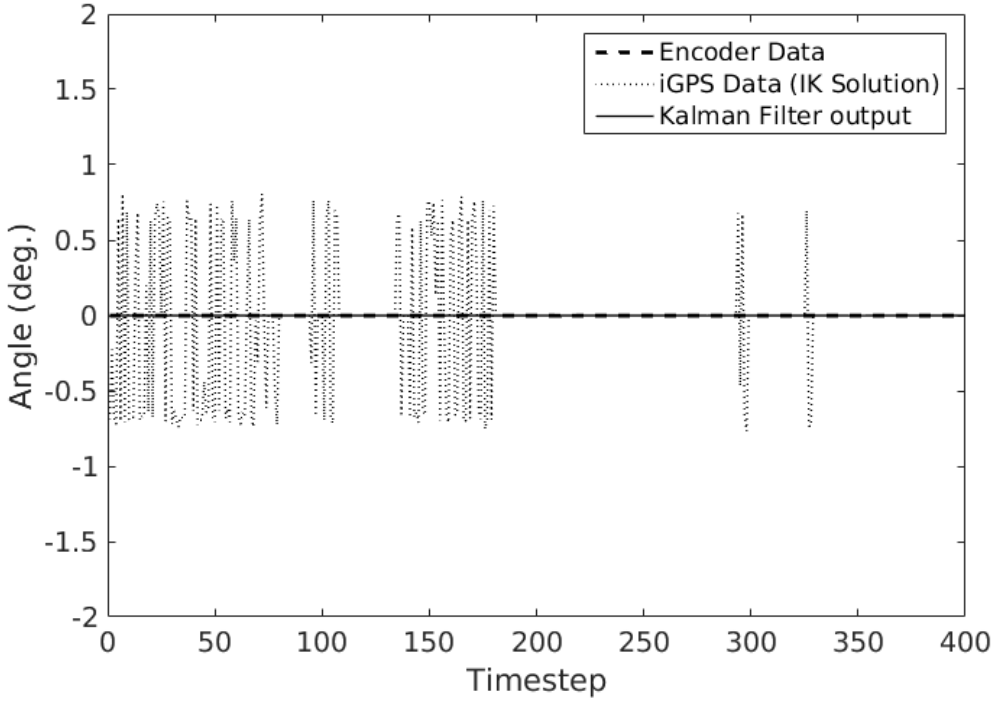



Figure 5.9: Kalman Filter output of iGPS and encoder data fusion

| | Maximum error (mm) | RMS error (mm) |
|---|---|---|
| Without correction | 1.33 | 0.89 |
| With correction | 0.39 | 0.18 |

Table 5.7: Error in Trajectory (Gazebo UR5 model) : Correction before motion planning stage algorithm

| | Maximum error (mm) | RMS error (mm) |
|---|---|---|
| Without correction | 7.75 | 4.13 |
| With correction | 3.31 | 2.11 |

Table 5.8: Error in Trajectory (Physical UR5) : Correction before motion planning stage algorithm

## 5.3 Evaluation of iGPS feedback correction before motion planning stage algorithm

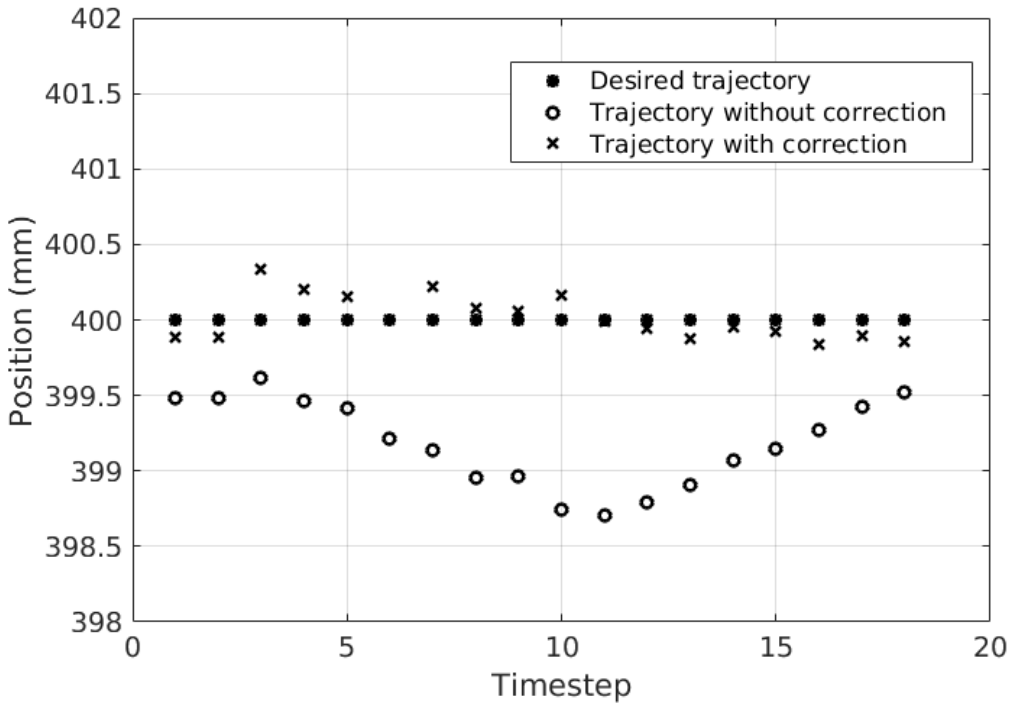
This algorithm was explained in Section 3.1. For testing this algorithm, UR5 manipulator was programmed to pass through a sequence of poses along a straight line (using PoseMoveInterface explained in Section 4.1). First the code was tested on UR5 model in Gazebo simulator and then the tests were conducted on the actual UR5 set-up.

#### UR5 Gazebo model

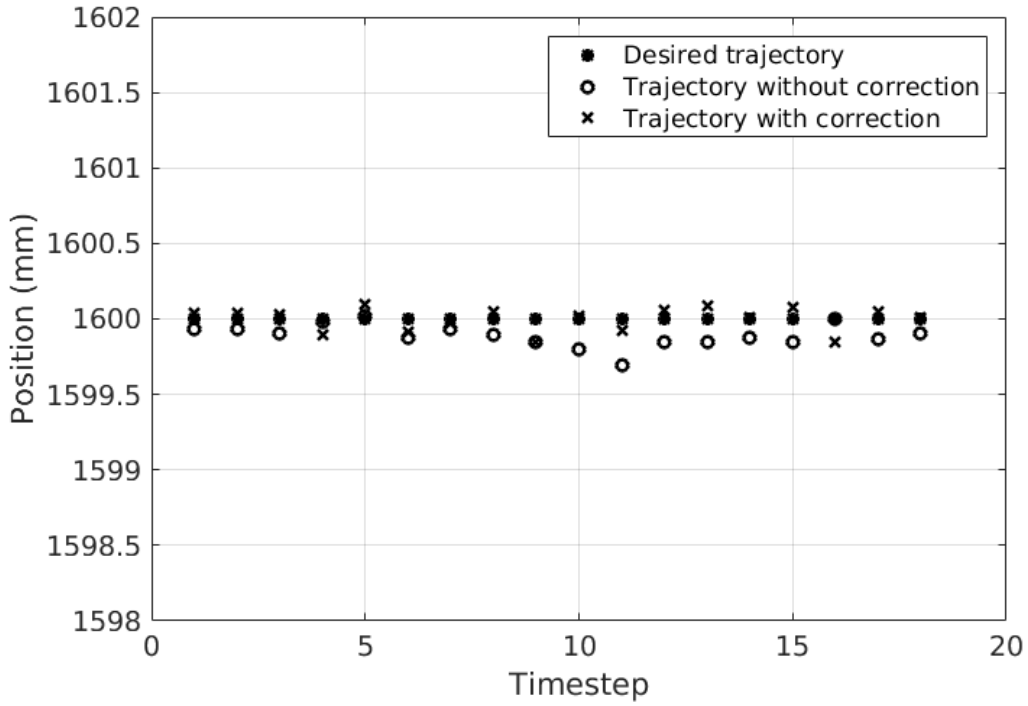
For simulation in Gazebo, the end-effector's pose was obtained using the $get\_link\_state$ service routine. Fig.5.10 shows the plots of end-effector trajectory with and without direct end-effector pose measurement correction before motion planning stage. Table 5.7 summarizes the maximum and root-mean-squared (RMS) error in end-effector trajectory.

#### Physical UR5 Robot

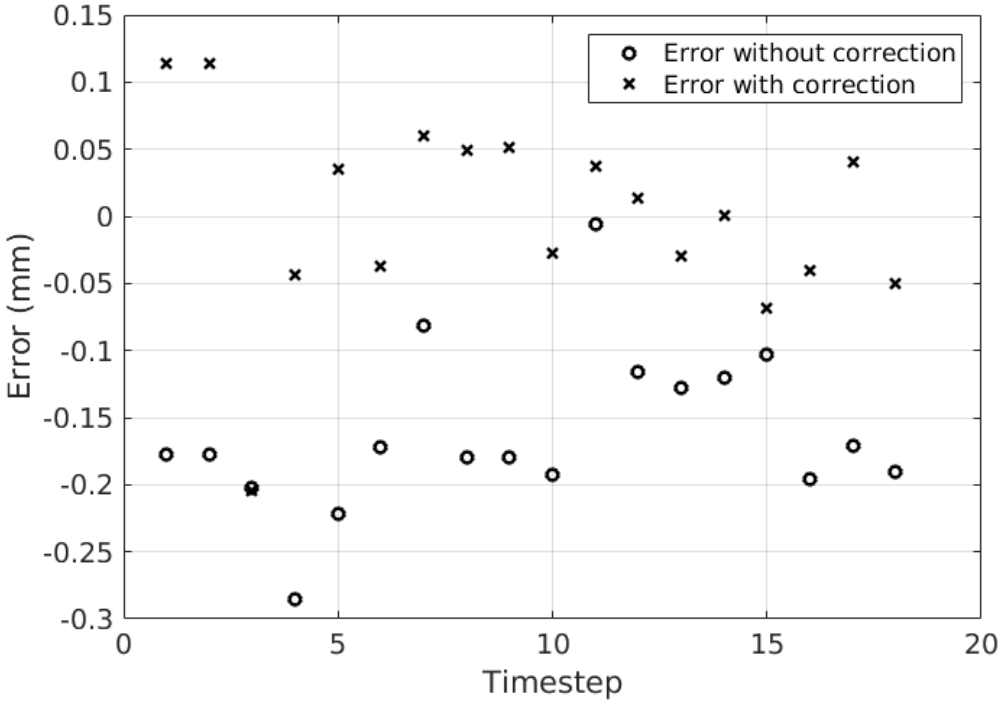
Fig.5.11 shows the plots of UR5 end-effector trajectory with and without iGPS end-effector pose measurement correction before motion planning stage. Table 5.8 summarizes the maximum and root-mean-squared (RMS) error in end-effector trajectory.



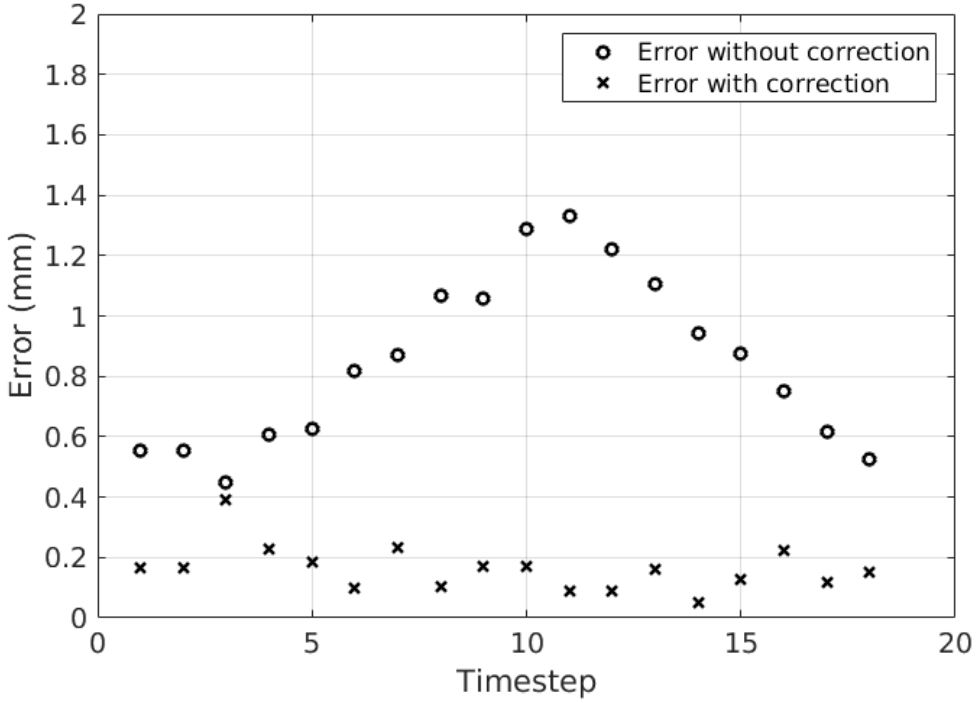

(a) End effector position along x-axis




(b) End effector position along z-axis

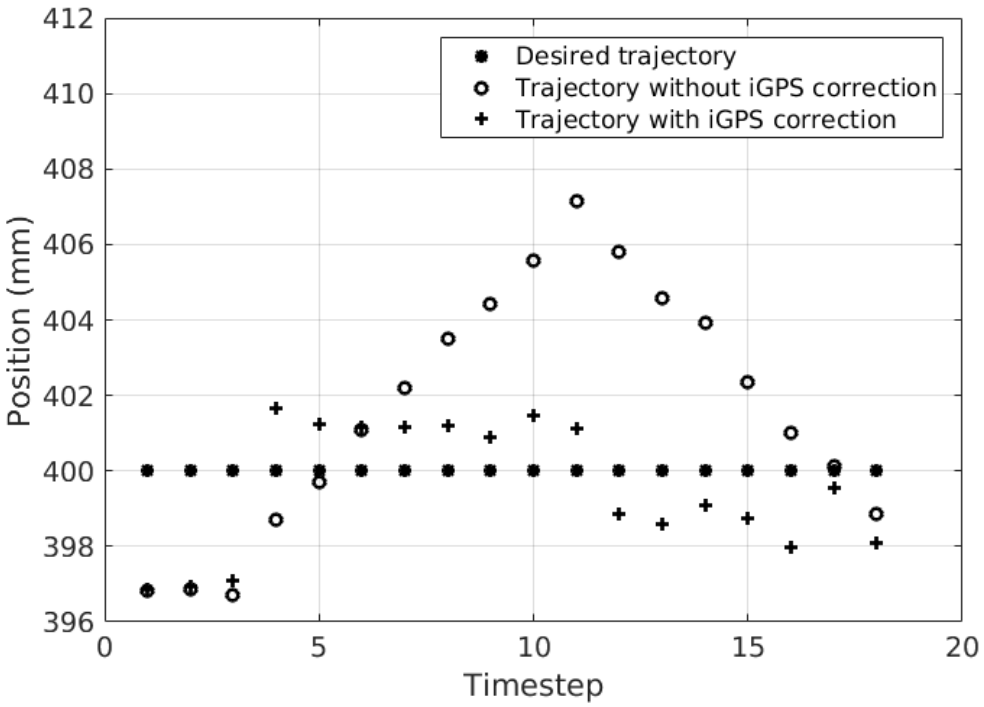



(c) Error in end effector position along y-axis

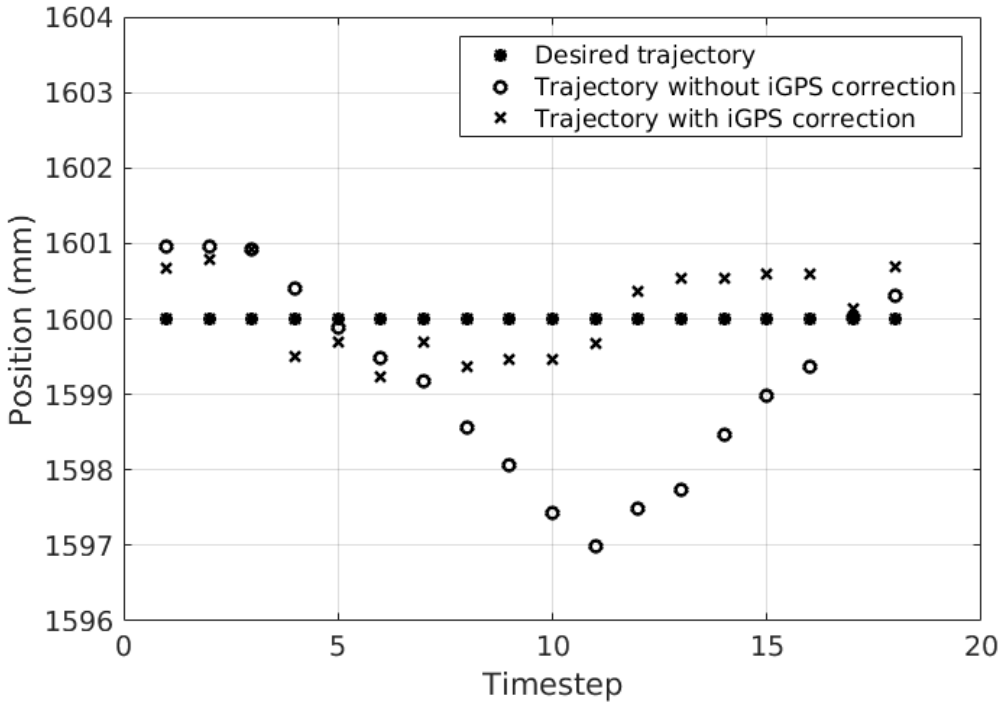



(d) Total error in end effector position

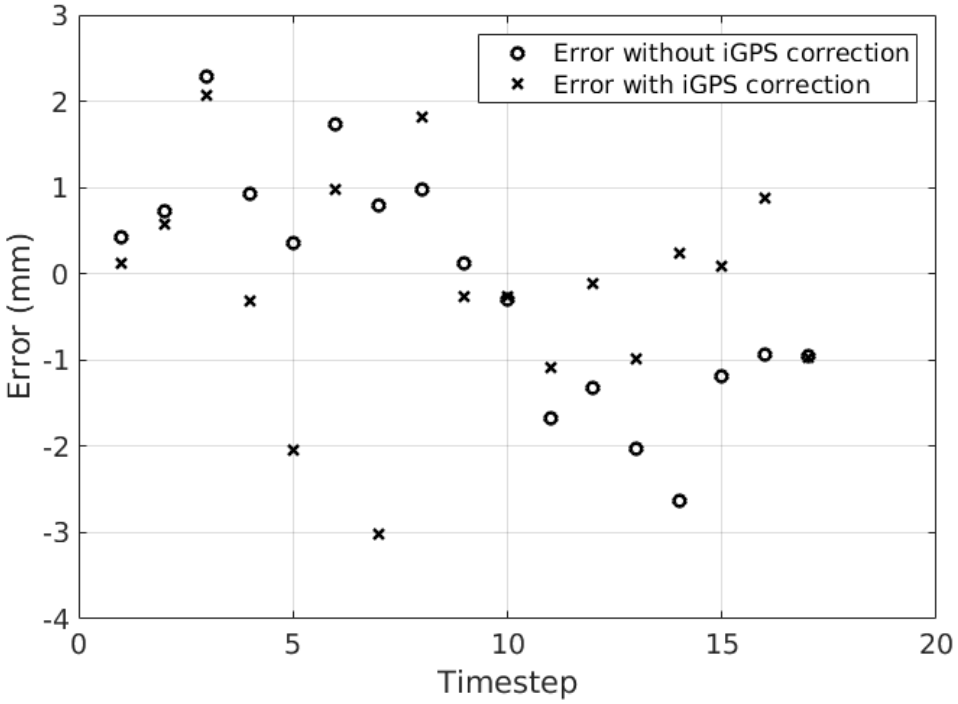
Figure 5.10: Gazebo simulation (Correction before motion planning stage) : 1-D end effector trajectory



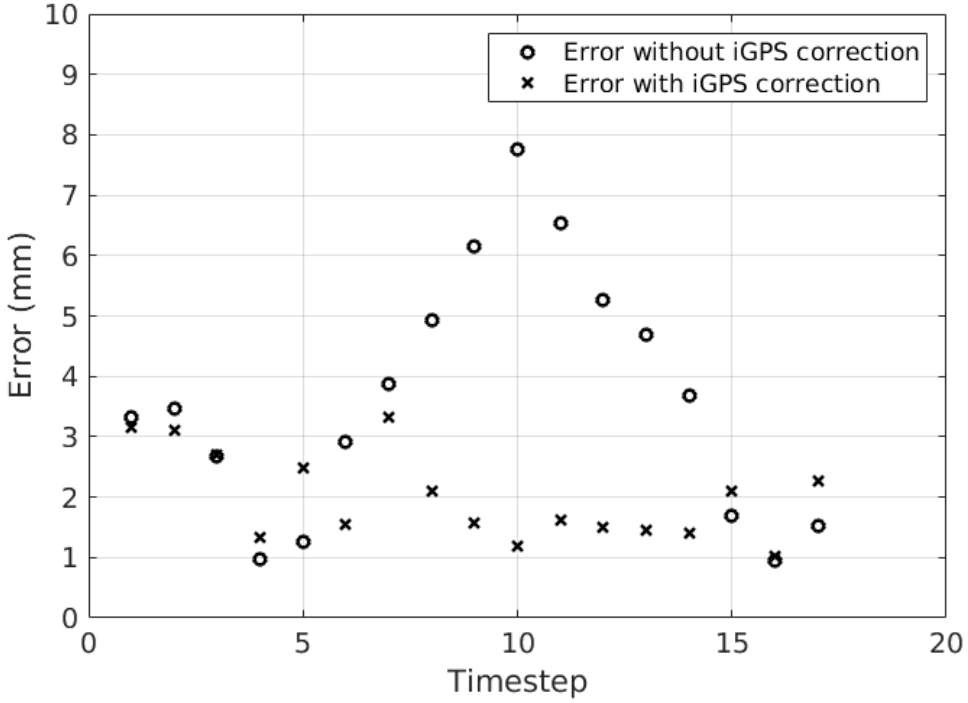

(a) End effector position along x-axis




(b) End effector position along z-axis




(c) Error in end effector position along y-axis




(d) Total error in end effector position

Figure 5.11: UR5 test (iGPS Correction before motion planning stage) : 1-D end effector trajectory

| | Maximum error (mm) | RMS error (mm) |
|---|---|---|
| Without correction | 0.33 | 0.19 |
| With correction | 0.25 | 0.15 |

Table 5.9: Error in Trajectory (Gazebo UR5 model) : Correction in joint state trajectory algorithm

| | Maximum error (mm) | RMS error (mm) |
|---|---|---|
| Without correction | 8.69 | 4.9 |
| With correction | 2.97 | 1.16 |

Table 5.10: Error in Trajectory (Physical UR5) : Correction in joint state trajectory algorithm

## 5.4 Evaluation of iGPS feedback correction in joint space trajectory algorithm

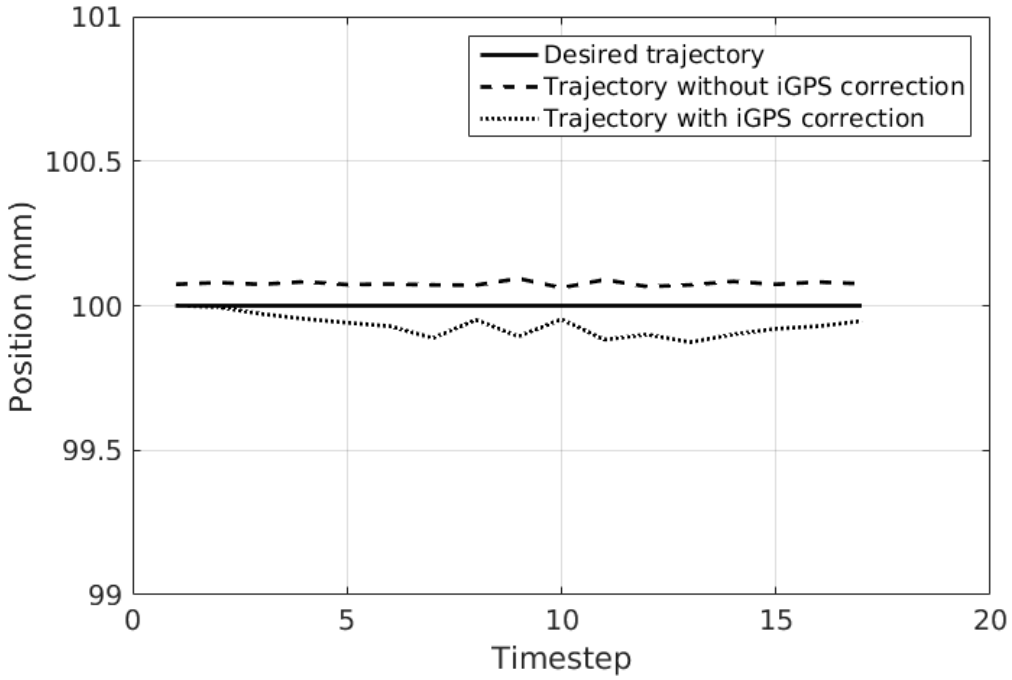
This algorithm was explained in Section 3.2 and the ROS implementation was explained in Section 4.2.2. For testing this algorithm, UR5 manipulator was programmed to traverse along a straight line trajectory (using WaypointsMoveInterface mode explained in Section 4.1). First the code was tested on UR5 model in Gazebo simulator and then the tests were conducted on the actual UR5 set-up.

#### UR5 Gazebo model

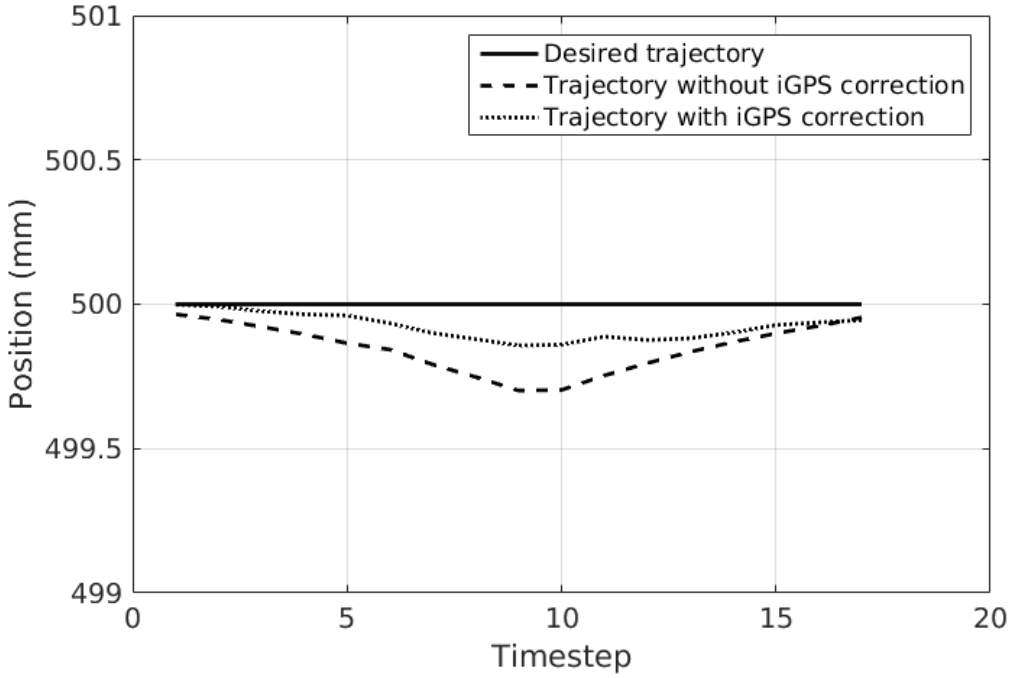
For simulation in Gazebo, the end-effector's pose was obtained using the $get\_link\_state$ service routine. Fig.5.12 shows the plots of end-effector trajectory with and without direct end-effector pose measurement correction in joint trajectory (after IK transform and fusion with encoder data). Table 5.9 summarizes the maximum and root-mean-squared (RMS) error in end-effector trajectory.

#### Physical UR5 Robot

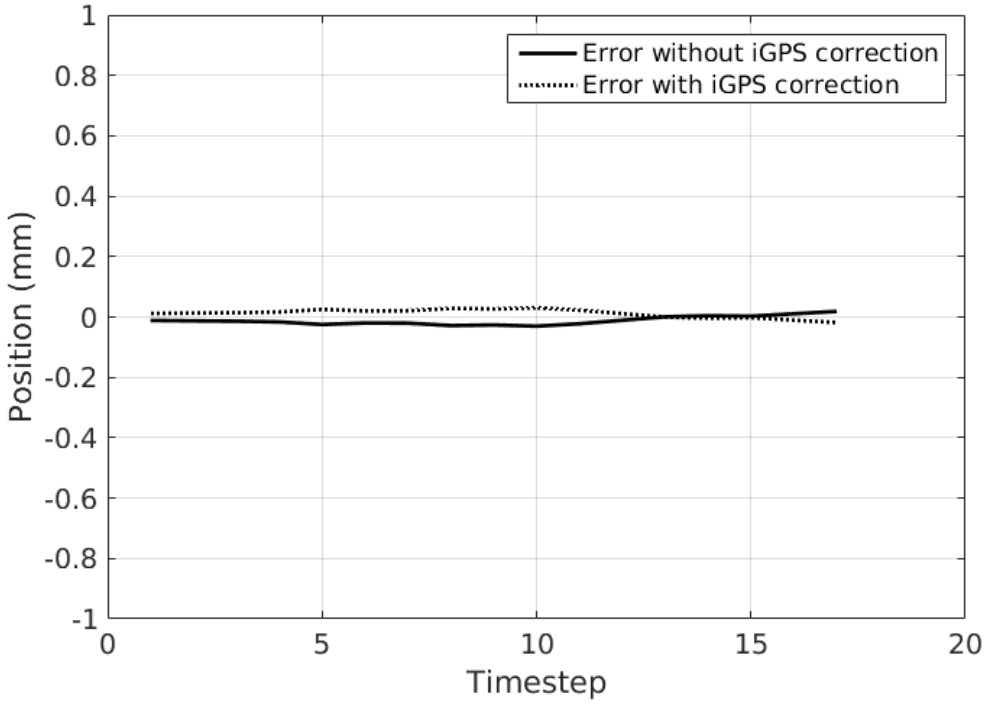
Fig.5.13 shows the plots of UR5 end-effector trajectory with and without iGPS end-effector pose measurement correction in joint trajectory. Table 5.10 summarizes the maximum and root-mean-squared (RMS) error in end-effector trajectory.



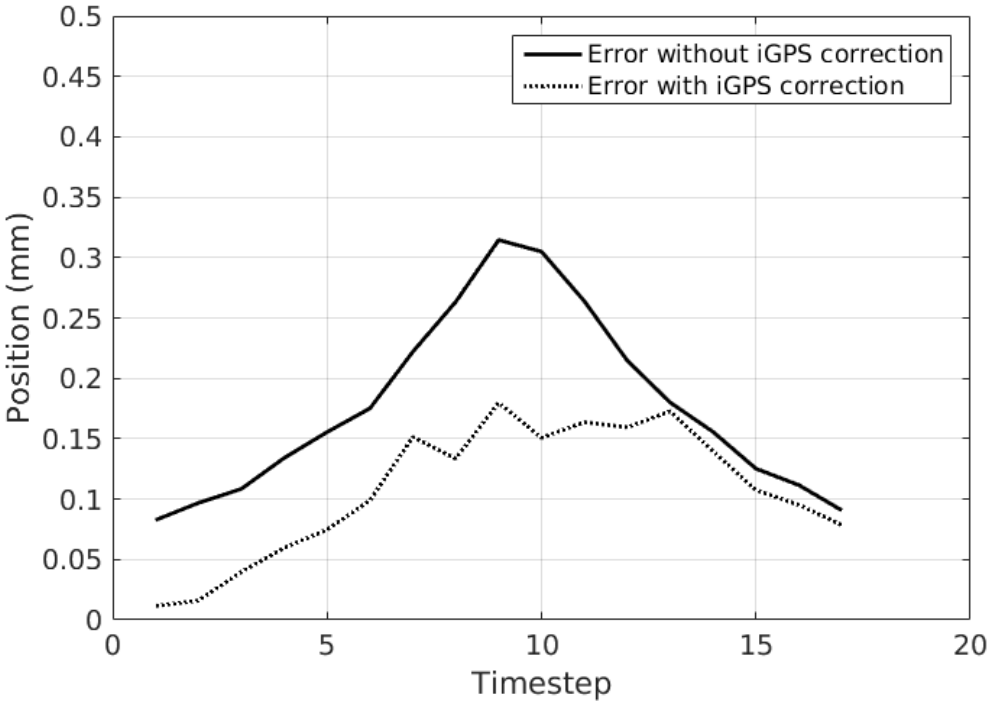

(a) End effector position along x-axis




(b) End effector position along z-axis

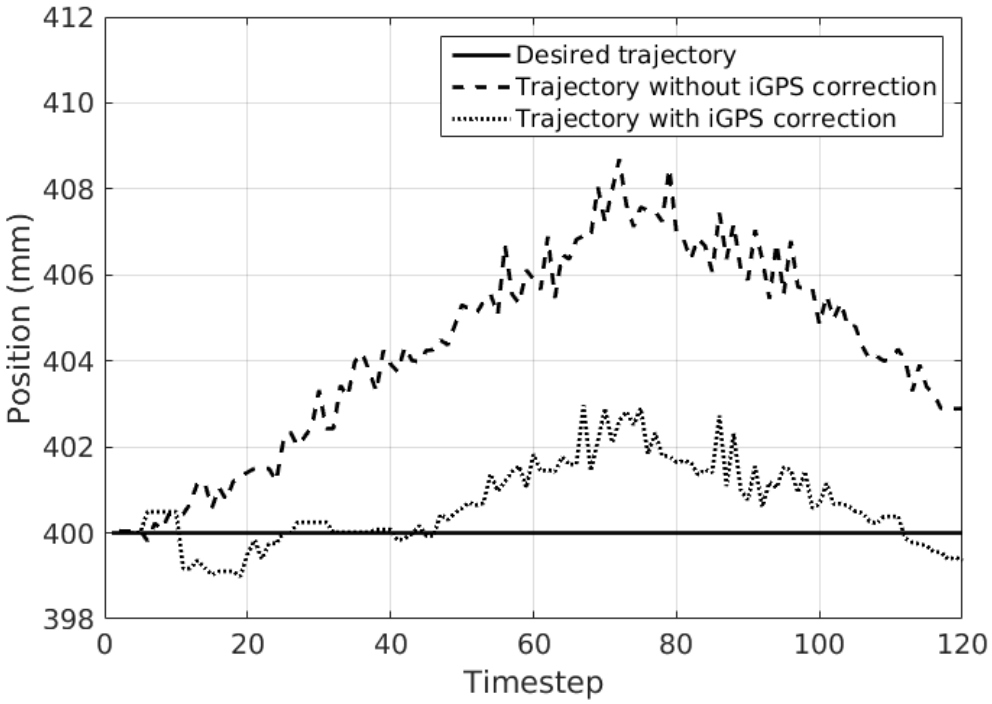



(c) Error in end effector position along y-axis

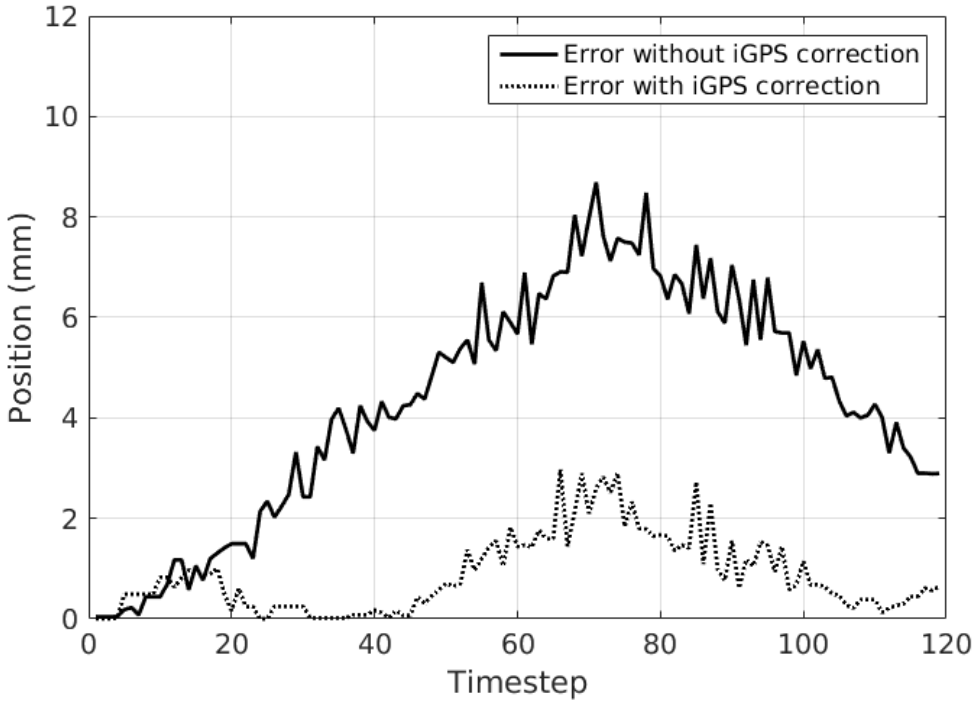



(d) Total error in end effector position

Figure 5.12: Gazebo simulation (joint state trajectory correction) : 1-D end effector trajectory




(a) End effector position along x-axis




(b) Error in end effector position along x-axis

Figure 5.13: UR5 test (iGPS Correction in joint state trajectory) : 1-D end effector trajectory

## 5.5 Summary

The accuracy, measurement frequency and range of iGPS was shown to be much better than a vision sensor like Kinect. The implementation of Kalman Filter was tested for all the three scenarios: single sensor, multiple sensors with different sampling rates, multiple sensors with different sampling rates and delays. Then the results of filter parameter tuning for UR5 encoder measurements and IK solutions of iGPS feedback were presented. Evaluation of two algorithms for iGPS feedback based accurate motion control of UR5 robot was carried out. The experiments were conducted on the robot model in Gazebo simulator and also on the physical set-up. The third algorithm (feedback in joint control loop) could not be tested due to the inbuilt joint controllers of UR5 which could not be modified to use the corrected joint state feedback. The experiments on UR5 showed that first algorithm reduced the RMS error in end-effector trajectory to $40\%$ and the second algorithm reduced the same to $25\%$.

# CHAPTER 6

# Conclusions and Future Work

This chapter will summarize the results and conclusions of this research work. Also a few directions for possible future extensions will be described.

## 6.1 Overall system

This work was the first attempt at experimental validation of iGPS feedback based accurate motion control of robot manipulator in a ROS framework. For accurate robot manipulator control, the method of using direct feedback of end-effector's position and orientation using external sensor has been proposed decades ago. But so far most of the research has focussed on computer vision based approaches. In this work, a large scale metrology system called 'iGPS' was used for direct measurement of end effector's position and orientation. iGPS was shown to be better than 'Kinect (vision sensor)' in terms of accuracy, measurement frequency and range. The previous research on iGPS feedback based motion control has involved only simple controller implementation using the robot motion control packages provided by OEMs. Also the performance was limited due to latency and lower accuracy of initial versions of the iGPS system. In this work, the advanced version of iGPS system was used and the software development was carried out in the open source framework of Robot Operating System. The advantages of our software development approach include modularity, customizability, ease of extension to multi-robot systems and collaborative development.

## 6.2 Algorithms

To include the direct end-effector pose feedback from iGPS in the motion control system of a robot manipulator, three possible approaches were described. The first approach involved iGPS feedback correction before the motion planning stage. In this

approach, a new motion plan had to be generated after every correction. Also it did not ensure accurate trajectory of end effector between two desired poses. In the second approach, iGPS feedback was transformed from end-effector co-ordinates to joint states using an inverse kinematics solver. Then the correction was applied in the joint space trajectory after the motion planning stage but before the joint control stage. In the third approach, the iGPS feedback was given to the joint controllers directly (after inverse kinematics transform). To improve the robustness of above approaches, the use of Kalman Filter for fusion of encoder and iGPS data was proposed. The formulation of Discrete Kalman Filter was described and extended to fusion of multi-frequency delayed sensor data.

## 6.3 Implementation

Robot Operating System framework was chosen for implementing the algorithms due to it's features like modularity, scalability, software reuse, distributed computing, language independence and rapid testing. A lightweight 6DoF robot manipulator Universal Robot-5 (UR5) was used for evaluating the control structures. The C++ API of a state-of-the-art open source motion planner MoveIt! was used to obtain collision free trajectories that obey user defined constraints. The ROS interface between Universal Robot's industrial robot controllers and motion planner MoveIt! was developed to control UR5 manipulator. Additional capabilities like gripper action and environment model for collision avoidance were included in the interface. Three modes of operation were developed. In the first mode, the sequence of desired end-effector poses could be specified. In the second mode, the sequence of desired joint states could be specified. In the third mode, a Cartesian trajectory could be specified in the form of way-points.

A ROS node was developed for getting the iGPS frames from Position Calculation Engine (PCE). The three structures for iGPS feedback based control were implemented in the ROS framework. For transformation between end-effector co-ordinates and joint states, two different inverse kinematics solvers were included in the code. The analytical IK solver ur_kinematics was found to be faster and accurate but it did not consider joint limits specified in the URDF and required end-effector pose in the base link frame. Also it could only be used for Universal's robots. Thus another IK solver trac_ik was

included in the code which was found to be more versatile. It could be used for any robot and it obeyed the specified joint limits in URDF. Even though the overall implementation was done for UR5 robot, it can be easily extended to any other robot or multi-robot systems.

## 6.4 Experimental validation

Experimental set-up was built using two iGPS probes for getting 6DoF pose measurements of UR5 manipulator's end-effector. The ROS code was first tested on UR5 model in the robot simulator Gazebo and then tested on the actual robot. The first approach, 'Correction before motion planning' added delays to the robot motion since a new motion plan had to be generated after each correction. This approach reduced the positioning error (RMS) to $40\%$. The second approach involved the iGPS feedback (after inverse kinematics transformation and fusion with encoder data using KF) in the joint space trajectory calculated by the motion planner. This approach reduced the RMS error in trajectory to $25\%$. The third algorithm i.e. feedback at joint control loop could not be tested since UR5 had it's own joint controllers which could not be modified. Tables 6.1 and 6.2 summarize the results of experiments.

| Control structure | Maximum error (mm) | RMS error (mm) |
|---|---|---|
| No end-effector pose feedback (Fig.3.1) | 1.33 | 0.89 |
| Correction before motion planning (Fig.3.2) | 0.39 | 0.18 |
| Correction in joint space trajectory (Fig.3.4) | 0.25 | 0.15 |

Table 6.1: Results of simulation in Gazebo

| Control structure | Maximum error (mm) | RMS error (mm) |
|---|---|---|
| No iGPS feedback (Fig.3.1) | 8.69 | 4.90 |
| Correction before motion planning (Fig.3.2) | 3.31 | 2.11 |
| Correction in joint space trajectory (Fig.3.4) | 2.97 | 1.16 |

Table 6.2: Results of experiments with UR5

## 6.5 Possible applications

There are several applications where the improved accuracy of robotic manipulator will be beneficial. One such application is 3D printing with multi-DoF robotic arms. The quality of final product of this rapid prototyping technique can be improved by reducing the error between desired and actual end-effector trajectories. The iGPS feedback based accurate robot manipulator control techniques can be used for this purpose. Another application of this work can be in the robotic manipulation of small deformable objects like thin cables and wires. Tasks like routing electrical cables can be accomplished accurately with the help of iGPS feedback. Since the software development is done in a ROS framework, the work can be easily extended to multi-robot systems for collaborative tasks.

## 6.6 Future improvements

The algorithm for iGPS feedback in the joint control loop as explained in Section 3.3 can be tested on custom built robot manipulator. In our present code, only the position control of the end effector is implemented. In future, the velocity and force control of end-effector can be incorporated. Also advanced control strategies like model predictive control can be implemented in the joint control loop using the ros_control package. There is a lot of scope of improvement in the motion planner MoveIt!. Currently only Iterative Parabolic Time Parametrization is used for obtaining trajectories. Other time parametrization algorithms like Time Optimal Trajectory Generation can be implemented in the MoveIt! C++ API.

# APPENDIX A

# Instructions to use the code

## A.1 Pre-requisites

- Ubuntu 14.04 LTS
- ROS-Indigo (http://wiki.ros.org/indigo/Installation)
- Gazebo `http://gazebosim.org/download`
- ROS-Industrial's *universal_robots* package `http://wiki.ros.org/universal_robot`
- moveit package `https://github.com/ros-planning/moveit/tree/indigo-devel`
- *geometric_shapes* package `https://github.com/ros-planning/geometric_shapes`

*moveit_commander* and *geometric_shapes* packages must be downloaded in the *catkin_ws/src* folder along with the our packages. For further information about configuration of ROS environment and *catkin* workspace set-up procedures please go to the following link: `http://wiki.ros.org/ROS/Tutorials/`

Installation procedure for *geometric_shapes*:

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ros-planning/geometric_shapes
$ cd geometric_shapes
$ git checkout indigo-devel
```

Installation procedure for *moveit_commander*:

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ros-planning/moveit_commander
$ git checkout indigo-devel
```

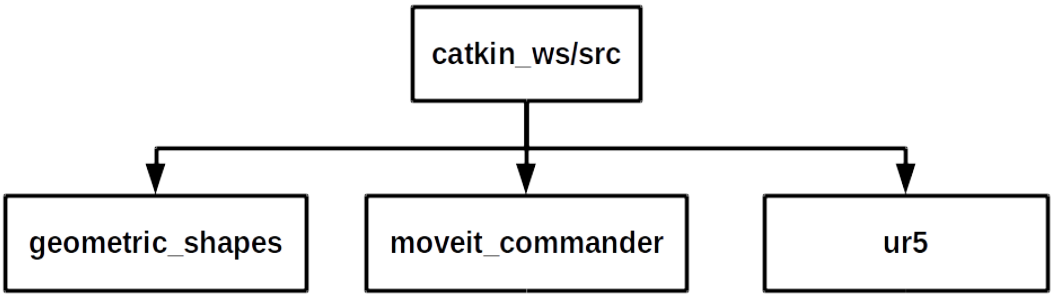



Figure A.1: Directory tree

The tree directory is shown in Fig.A.1.

After cloning all the packages in *src* folder, verify that the /ur5/source_files/*main.sh* file has the correct extension for the *setup* file (*.bash* or *.zsh*). Also, verify that the IP address given in this file matches the one of the Computer on which this code will be run.

## A.2 Git branches in *ur5* folder

We have used git for version control of our software. A very long but also very good introduction into every level of git can be found here `https://www.atlassian.com/git/tutorials/what-is-version-control`. The Git branches of UR5 code are listed below with a brief description of each branch.

1. *master* : Stable branch for controlling UR5 without iGPS feedback
2. *igps_path_stage_correction* : Development branch for controlling UR5 with iGPS correction before motion planning stage
3. *igps_joint_trajectory_correction* : Development branch for controlling UR5 with iGPS correction in joint space trajectory
4. *igps_joint_control_loop* : Development branch for iGPS feedback in joint control loop

## A.3 Running simulation in Gazebo

Once the package has been cloned into the *src* folder in the *catkin* workspace, execute these commands in the Linux terminal (Terminator is recommended for this procedure):

```
$ cd ~/catkin_ws
$ source ~/catkin_ws/src/ur5/source_files/main.sh
$ cd build
$ cmake ../src
$ make
$ cd ..
$ catkin_make
```

These commands will build the catkin workspace which is needed for running the IGM-UR5 package. Next, execute the following commands:

```
$ source ~/catkin_ws/src/ur5/01_ROS_Code/source_files/main.sh
$ roslaunch igm_collision main.launch
```

Open a new terminal or split that one you are currently working on (in Terminator: Shift+Ctrl+O ) and type the following commands:

```
$ source ~/catkin_ws/src/ur5/source_files/main.sh
$ rosrun igm_collision move_group_interface_collision
```

By default the IGM-UR5 package comes with the internal configuration for a simulated model of the UR5 (Gazebo model). Therefore, in order to send a trajectory to the moveit node, the *sim_path_trajectory* file has to be executed in a new terminal or window:

```
$ source ~/catkin_ws/src/ur5/source_files/main.sh
$ rosrun igm_collision sim_path_publisher
```

## A.4 Running the IGM-UR5 package on a real UR5 robot

If the package is going to be used on a real robot, the following arguments must be added when launching the main.launch file:

```
$ source ~/catkin_ws/src/ur5/source_files/main.sh
$ roslaunch igm_collision main.launch sim:=false robot_ip
   :=134.130.201.213 reverse_port:=50001
```

Once the desired poses have been added to the poses2.txt file, the *move_group_interface_collision* and the *path_publisher* nodes have to be launched into the ROS network. Therefore, in new terminal windows execute the following commands:

Terminal 1

```
$ source ~/catkin_ws/src/ur5/source_files/main.sh
$ rosrun igm_collision move_group_interface_collision
```

Terminal 2

```
$ source ~/catkin_ws/src/ur5/source_files/main.sh
$ rosrun igm_collision path_publisher
```

## A.5 User input

There are three possible modes of user input that can be specified in the code as explained in Section 4.1. For switching between different modes, the relevant portion in the *main* function of move_group_interface_collision.cpp has to be commented out, before compiling the code. We are working on incorporating a parameter to automatically select the mode while starting the node itself.

The *sim_path_publisher* node (simulation) or *path_publisher*(actual UR5) node will publish the desired poses included in the file poses2.txt on the */wished_path* topic. If this file name has to be altered, the public variable *Pose.sourceFile* must be changed in the C++ code *sim_path_publisher.cpp* or *path_publisher.cpp* in the directory: */ur5/01_ROS_Code/igm_collision/src/*

By default the poses included in the poses2.txt represents a specific end-effector pose in term of 3D position and quaternions for rotation.

x y z rx ry rz w

# REFERENCES


[1] M. Good and L. Sweet. Structures for Sensor-Based Robot Motion Control. In *American Control Conference, 1984*, pages 23–31, San Diego, USA, June 1984.

[2] Universal Robots. User Manual UR5/CB3, 2014.

[3] I. Sucan and S. Chitta. Moveit! http://moveit.ros.org, April 2017.

[4] S. Pornsarayouth and M. Wongsaisuwan. Sensor fusion of delay and non-delay signal using Kalman Filter with moving covariance. In *IEEE International Conference on Robotics and Biomimetics*, pages 2045–2049. IEEE, February 2008.

[5] S. Jeon, M. Tomizuka, and T. Katou. Kinematic Kalman Filter (KKF) for Robot End-Effector Sensing. *Journal of Dynamic Systems, Measurement, and Control*, 131(2):1–8, 2009.

[6] R. Schmitt, S. Nisch, A. Schonberg, F. Demeester, and S. Renders. Performance evaluation of iGPS for industrial applications. In *International Conference on Indoor Positioning and Indoor Navigation*, pages 1–8. IEEE, September 2010.

[7] Nikon Metrology. iSpace Large volume metrology , tracking and positioning. www.nikonmetrology.com/iSpace_EN, 2010.

[8] J. O'Kane. *A Gentle Introduction to ROS*. Independently published, October 2013.

[9] M. de Gier. *Control of a Robotic Arm: Application to on-surface 3D-printing*. Master of science thesis, Delft University of Technology, 2015.

[10] T. Tang, C. Liu, W. Chen, and M. Tomizuka. Robotic manipulation of deformable objects by tangent space mapping and non-rigid registration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2689–2696, 2016.

[11] A. Norman, A. Schönberg, Igor A. Gorlach, and R. Schmitt. Validation of iGPS as an external measurement system for cooperative robot positioning. *International Journal of Advanced Manufacturing Technology*, 64(1-4):427–446, 2013.

[12] D. Maisano, J. Jamshidi, F. Franceschini, P. Maropoulos, L. Mastrogiacomo, A. Mileham, and G. Owen. A comparison of two distributed large-volume measurement systems: the mobile spatial co-ordinate measuring system and the indoor global positioning system. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 223(5):511–521, May 2009.

[13] G. Agin. *Real time control of a robot with a mobile camera*. Technical note. SRI International, 1979.

[14] D. Kragic and H. Christensen. Survey on visual servoing for manipulation. Technical report, Computational Vision and Active Perception Laboratory, 2002.

[15] N. Papanikolopoulos and C. Smith. Computer vision eye-in-hand issues during botic tasks. In *International Conference on Robotics and Automation*, volume 3, pages 2989–2994 VOl 3. IEEE, 1995.

[16] R. Basri, E. Rivlin, and I. Shimshoni. Visual homing: Surfing on the epipoles. *International Journal of Computer Vision*, 33(2):117–137, 1999.

[17] N. Hollinghurst and R. Cipolla. Uncalibrated stereo hand-eye coordination. *Image and Vision Computing*, 12(3):187–192, 1994.

[18] J. Chen, A. Behal, D. Dawson, and Y. Fang. 2.5D visual servoing with a fixed camera. In *Proceedings of American Control Conference*, volume 4, pages 3442–3447. IEEE, 2003.

[19] Y. Zhang and M. Mehrandezh. Visual Servoing of a 5-DOF Mobile Manipulator using a Catadioptric Vision System. In *Canadian Conference on Electrical and Computer Engineering*, page 671906. International Society for Optics and Photonics, October 2007.

[20] R. Mautz. Overview of current indoor positioning systems. *Geodezija ir Kartografija*, 35(1):18–22, 2009.

[21] Z. Wang, L. Mastrogiacomo, F. Franceschini, and P. Maropoulos. Experimental comparison of dynamic tracking performance of iGPS and laser tracker. *International Journal of Advanced Manufacturing Technology*, 56(1-4):205–213, September 2011.

[22] A. Norman, A. Schönberg, I. Gorlach, and R. Schmitt. Cooperation of Industrial Robots with Indoor-GPS. *Proceedings of the International Conference on Competitive Manufacturing*, pages 215–224, 2010.

[23] R. Schmitt, A. Schoenberg, and B. Damm. Indoor-GPS based robots as a key technology for versatile production. In *41st International Symposium on Robotics (ISR) and 6th German Conference on Robotics (ROBOTIK)*, pages 1–7, 2010.

[24] F. Franceschini, M. Galetto, D. Maisano, L. Mastrogiacomo, and B. Pralio. *Distributed Large-Scale Dimensional Metrology*. Springer London, London, 2011.

[25] C. Depenthal and J. Schwendemann. IGPS - A NEW SYSTEM FOR STATIC AND KINEMATIC MEASUREMENTS. *Optical 3-D Measurement Techniques IX*, I:131–140, 2009.

[26] ArcSecond. Constellation 3D-i: error budget and spec- ifications.White Paper 063102. Arc Second, Inc., Dulles, VA, 2002.

[27] Metris. iGPS: data sheet v1.3 (DS-iGPS-140807). Metris HQ, Leuven, 2007.

[28] J. Muelaner, Z. Wang, J. Jamshidi, and P. Maropoulos. Verification of the indoor GPS system by comparison with points calibrated using a network of laser tracker measurements. *Proceedings of the 6th CIRP-Sponsored International Conference on Digital Enterprise Technology*, 66(Flack 2001):607–619, 2010.

[29] Universal Robots. The URScript Programming Language, 2015.

[30] Univeral Robots. PolyScope Manual, 2012.

[31] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng. ROS: an open-source Robot Operating System. In *ICRA Workshop on Open Source Software*, 2009.

[32] Open Source Robotics Foundation. ROS. http://wiki.ros.org/ROS/Introduction, April 2017.

[33] ROS-Industrial Consortium. ROS-Industrial. http://rosindustrial.org/, April 2017.

[34] Orocos.org. Kinematics and dynamics library (kdl). http://www.orocos.org/kdl, April 2017.

[35] P. Beeson and B. Ames. TRAC-IK: An open-source library for improved solving of generic inverse kinematics. In *Proceedings of the IEEE RAS Humanoids Conference*, Seoul, Korea, November 2015.

[36] OpenRave. Ikfast: The robot kinematics compiler. http://openrave.org/docs/0.8.2/openravepy/ikfast/, April 2017.

[37] R. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35, 1960.

[38] G. Welch and G. Bishop. An introduction to the kalman filter, 2006.

[39] R. Cristi and M. Tummala. Multirate, multiresolution, recursive Kalman filter. *Signal Processing*, 80(9):1945–1958, 2000.

[40] L. Dong-Jun and M. Tomizuka. Multirate optimal state estimation with sensor fusion. *Proceedings of the 2003 American Control Conference, 2003.*, 4:2887–2892, 2003.

[41] H. Durrant-whyte. Multi Sensor Data Fusion. http://www.acfr.usyd.edu.au/pdfs/training/multiSensorDataFusion/dataFusionNotes.pdf, 2006.

[42] X. Lu, H. Zhang, W. Wang, and K. Teo. Kalman filtering for multiple time-delay systems. *Automatica*, 41(8):1455–1461, 2005.

[43] S. Edwards, S. Glaser, K. Hawkins, W. Meeussen, and F. Messmer. universal_robot stack. http://wiki.ros.org/universal_robot, April 2017.

[44] K. Hawkins. Analytic Inverse Kinematics for the Universal Robots. http://hdl.handle.net/1853/50782, 2013.

[45] É. Marchand, F. Spindler, and F. Chaumette. ViSP for visual servoing: A generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, 12(4):40–52, December 2005.

[46] F. Novotny. visp_auto_tracker. http://wiki.ros.org/visp_auto_tracker, April 2017.

[47] H. Martin, J. Blake, K. Machulis, and OpenKinect community. libfreenect. http://wiki.ros.org/libfreenect, April 2017.

[48] P. Mihelich and P. Khandelwal. freenect_launch. http://wiki.ros.org/freenect_launch, April 2017.