T. USLU

IZMIR KATIP CELEBI UNIVERSITY

2021

IZMIR KATIP CELEBI UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEVELOPMENT OF A SURGICAL NAVIGATION PROCEDURE FOR COCHLEAR MICROROBOT OPERATIONS

M.Sc. THESIS

Tuğrul USLU

Department of Mechanical Engineering

Thesis Advisor:

Assoc. Prof. Dr. Erkin GEZGİN

JANUARY 2021

IZMIR KATIP CELEBI UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEVELOPMENT OF A SURGICAL NAVIGATION PROCEDURE FOR COCHLEAR MICROROBOT OPERATIONS

M.Sc. THESIS

Tuğrul USLU
Y180217005
0000-0002-2154-9268

Department of Mechanical Engineering

Thesis Advisor: Assoc. Prof. Dr. Erkin GEZGİN

JANUARY 2021

İZMİR KATİP CELEBİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KOKLEAR MİKROROBOT OPERASYONLARI İÇİN BİR CERRAHİ NAVİGASYON YÖNTEMİNİN GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Tuğrul USLU
Y180217005
0000-0002-2154-9268

Makine Mühendisliği Ana Bilim Dalı

Tez Danışmanı: Doç. Dr. Erkin GEZGİN

OCAK 2021

Tuğrul USLU a **M.Sc.** student of **IKCU Graduate School Of Natural And Applied Sciences**, successfully defended the thesis entitled "**DEVELOPMENT OF A SURGICAL NAVIGATION PROCEDURE FOR COCHLEAR MICROROBOT OPERATIONS"** which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :

Assoc. Prof. Dr. Erkin GEZGİN
İzmir Kâtip Çelebi University

Jury Members :

Asst. Prof. Dr. Duygu ATCI
İzmir Kâtip Çelebi University

Asst. Prof. Dr. Özgün SELVİ
Çankaya University

Date of Defense : 15.01.2021

To my family and friends

FOREWORD

First and foremost, I would like to thank my advisor, Erkin GEZGİN, who kindled my passion for robotics, and offered me excellent guidance with patience and respect since my undergraduate years. Also I would like to thank all of my labmates at the Medical Robotics Laboratory at the İzmir Katip Çelebi University for all the support they provided me.

This thesis was supported by The Scientific and Technological Research Council of Turkey. Project No: 218E055

January 2021

Tuğrul USLU

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

DEVELOPMENT OF A SURGICAL NAVIGATION PROCEDURE FOR COCHLEAR MICROROBOT OPERATIONS

ABSTRACT

By the help of technological advances such as the development of computer and imaging technologies, surgical navigation has started to be used rapidly in medical literature. It is one of the recent methods used to track surgical tools inside the operation volume. In light of vital advantages as operation safety, minimally invasive application compatibility, reduction of operation times and reduced post operation complication risks, surgical navigation has been rapidly adopted throughout the relevant fields. Considering these, this study proposes both creation of a surgical navigation system utilizing free and open source software and a low cost motion tracking system, and a verification method to check validity of navigation system calibration. Also this study discussed feasibility of an analytical approach to one of the most important steps of surgical navigation as registration and presented a real world example of distortions might be caused by discussed analytical approach. Throughout the study, various tools related to surgical navigation and a solid model with known dimensions are designed and manufactured. Designed tool's precision then tested by performing calibration procedures and error values of these calibrations are examined. Then solid model was placed inside a capture volume in which the motion cameras are able to provide position measurements with help of designed tools. Carrying out point based registration method with least squares approach by taking necessary measurements, relation between the model reference and measurement space reference was calculated by means of a transformation matrix. This procedure is also applied in a capture volume with known broken calibration and on a solid model with movement compensation provided by previously designed tools. Later in the study an industrial robot manipulator arm was integrated to the navigation system and its implementation was verified with an navigation application utilizing designed solid model.

Keywords: medical robotics, surgical navigation, point based registration, motion capture

KOKLEAR MIKROROBOT OPERASYONLARI İÇİN BİR CERRAHİ NAVİGASYON YÖNTEMİNİN GELİŞTİRİLMESİ

ÖZET

Bilgisayar ve görüntüleme teknolojilerinin gelişmesiyle medikal alanda hızla kullanılmaya başlayan cerrahi navigasyon, operasyon hacminin takibinin sağlanmasında kullanılan güncel medikal yöntemlerden biridir. Uygulama esnasında cerrah ve hastaya sağladığı, operasyon güvenliği, minimal invaziv uygulama uyumluluğu, operasyon sürelerinin ve uygulama sonrası komplikasyon risklerinin azaltılması gibi önemli avantajlar ışığında cerrahi navigasyon ile takip, ilgili alanlarda hızlı bir şekilde benimsenmiştir. Bu çalışma, özgür ve açık kaynak yazılımları ve düşük maliyetli hareket takip sistemi içeren bir cerrahi navigasyon sistemi, ve navigasyon sisteminin geçerliliğini denetleyecek bir doğrulama yönteminin oluşturulmasını içermektedir. Ayrıca bu çalışma da cerrahi navigasyonun en önemli adımlarından biri olan eşleştirme prosedürünün analitik çözüm yöntemi ile gerçekleştirilmesinin uygulanabilirliği tartışılmış ve tartışılan analitik çözüm yönteminin gerçek bir uygulamada oluşturabileceği bozulmalar gösterilmiştir. Çalışma boyunca, cerrahi navigasyon sırasında kullanılmak üzere çeşitli araçlar ve ölçüleri bilinen bir katı model tasarlanmış ve üretilmiştir. Tasarlanan araçların hassasiyeti üzerlerinde kalibrasyon prosedürleri uygulanarak bulunan hata değerleri ile incelenmiştir. Daha sonra ise katı model hareket yakalama kameralarının takip edebildiği ilgili uygulama hacmine yerleştirilmiş, ve ilgili ölçümler yapılarak en küçük kareler yaklaşımı ile nokta tabanlı eşleştirme yöntemi uygulanmış, cerrahi navigasyon için gerekli olan gerçek model referansı ile hareket yakalama kameralarının yer aldığı ölçüm uzay referansı arasındaki ilişki bulunmuştur. Bu prosedür hareket yakalama kameralarının kalibrasyonunun bozuk olduğu bilinen bir uygulama hacminde, ayrıca katı modelin hareketlerinin tasarlanan araçlar yardımıyla kompanse edildiği bir uygulama hacminde uygulanmıştır. Çalışmanın sonraki aşamasında bir endüstriyel robot, oluşturulan cerrahi navigasyon sistemine entegre edilmiş ve tasarlanan katı model yardımı ile entegrasyonun başarısı bir cerrahi navigasyon uygulamasıyla doğrulanmıştır.

Anahtar Kelimeler: medikal robotik, cerrahi navigation, nokta bazlı eşleştirme, hareket yakalama

1. INTRODUCTION

The most important fact that increases success of surgical operations is the quality of visual feedback the doctor is getting from the workspace. In open surgical operations naked eye vision of the surgeon can be enhanced by surgical lamps and eyeglasses with loupes to catch details that can be missed with naked eye in natural lighting conditions. However in minimally invasive operations like laparoscopic surgeries, due to workspace being a closed space, these solutions are rendered ineffective. In these operations, visual feedback to the doctor is provided by endoscopic cameras positioned in the workspace. However some factors as loss of depth in 2d vision, camera/monitor resolution limits, physically unreachable positions by endoscopic cameras and need to clean optics due to internal bleeding and other bodily fluids indicates weaknesses of the endoscopic cameras. By the help of advancing technology high resolution endoscopic systems started to be used in surgical operations but these systems cannot provide permanent solutions to every flaw listed above. In modern times surgical navigation methodologies started to appear in many scientific studies to provide doctor with visual feedback in minimally invasive operations.

Basically, positions and orientations of surgical tools are tracked with optical or electromagnetic methods in real time and streamed to virtual environment in surgical navigation methods where visual data obtained with surgical vision technologies from related workspace visualized during operation in virtual environment with or without support of augmented reality. Thus virtual feedback from the workspace can be provided to operating surgeon. Some factors effecting operation efficiency might be listed as,

- Motion capture hardware used during operation and its official software provided by manufacturer,
- Calibration procedure included motion capture software to calculate positions and orientations of tracked rigid bodies,
- Data types provided by software to users and ability to collect these data in a processable way,
- Ability to calculate relations between surgical workspace, virtual environment and surgical tool reference coordinate systems with collected data.

1.1 Motion Capture Systems

Navigating any object through space with precision and accuracy, the object's position and orientation in space must be known. There are many approaches to measure a body's position and/or orientation in space. Most common types of motion capture systems utilized in surgical navigation can be categorized by methods they use to achieve results and can basically be listed as electromagnetic (also can be called magnetic), ultrasonic, and optical.

Electromagnetic method uses one generated electromagnetic field for each independent axis on space (three for spatial space). A receiver placed onto the desired object can measure sum of these fields and this sum can be used to determine position of this receiver in space. Ultrasonic systems use sound waves as medium of motion capturing. These sound waves are generally outside of human hearing frequency range for ultrasonic systems and emitted directly into an area of interest by transmitters. Echoes of these sound waves as they travel thorough area of interest can be observed by receivers and utilized in calculating desired positions and/or orientations. Optical motion tracking systems utilize light waves as medium of motion capture and can be categorized as marker-based and markerless systems. Markerless optical motion capture systems utilize heavy image processing in their calculations. Although these systems are convenient as they do not require extra equipment like markers, they lack in precision compared to marker-based optical systems. Marker-based optical motion capture systems depend on special equipment called markers to track an objects position and/or orientation in space. These markers vary in methodologies of which these marker-based system use in their calculations. Most common marker types are pattern type and sphere type markers. Pattern type markers are made of asymmetric patterns shown in figure 1.1 and generally used in augmented reality applications. Motion tracking with sphere type markers utilizes active (light-emitting) or passive (light-reflecting) spheres in position and/or orientation calculations. Passive spheres are generally coated with a reflective material and externally illuminated to provide better visibility for motion capture cameras. At least two motion capture cameras are required for calculating position spheres in space.

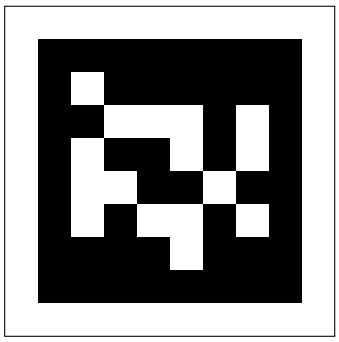

Figure 1.1. Example of Pattern Marker.

Motion capturing with optical systems with sphere markers starts with calibrating motion capture cameras. Calibration calculates visual distortion in motion capture cameras and relative positions and orientations of each motion capture camera with respect to each other. From each camera images are captured simultaneously and fed into image processing to calculate centroid of projection of sphere in screen area of each camera as shown in figure 1.2. If local distortions of motion capture cameras are low, centroids of sphere projections can be assumed as centroids of spheres.

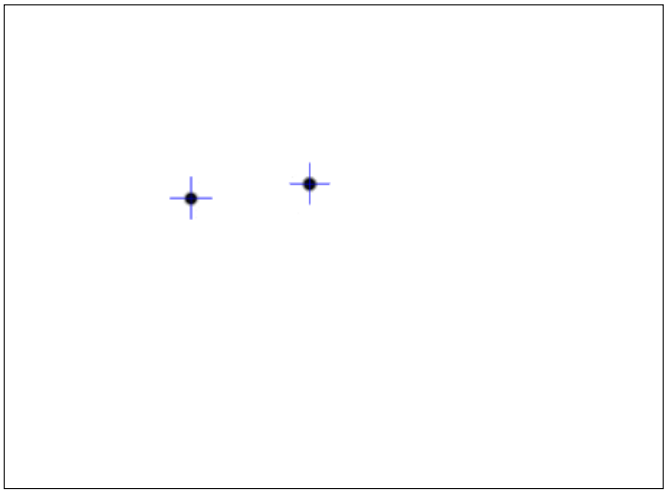

Figure 1.2. Visual Taken From a Motion Tracking Camera While Two Marker Sphere Centroids Being Detected.

From this point, lines to centroid positions can be constructed from local camera references while keeping visual distortions calculated in calibration phase in mind. Two constructed lines from different motion capture cameras $l_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T + p \begin{bmatrix} a & b & c \end{bmatrix}^T$ and $l_2 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T + p \begin{bmatrix} d & e & f \end{bmatrix}^T$ can be placed in space by utilization of relative position and orientations of motion capture cameras utilizing ${}^{C1}_{C2}\boldsymbol{R}$ rotation matrix and ${}^{C1}_{C2}\boldsymbol{t}$ translation as shown in equation 1.1, also calculated in cali-

bration phase. Any collision of lines within a tolerance is a detected position (Figure 1.3).

$$\boldsymbol{l}_2^1 = -{}^{C^1}_{C^2}\boldsymbol{t} + p({}^{C^1}_{C^2}\boldsymbol{R}\begin{bmatrix} d & e & f \end{bmatrix}^T + {}^{C^1}_{C^2}\boldsymbol{t}) \tag{1.1}$$

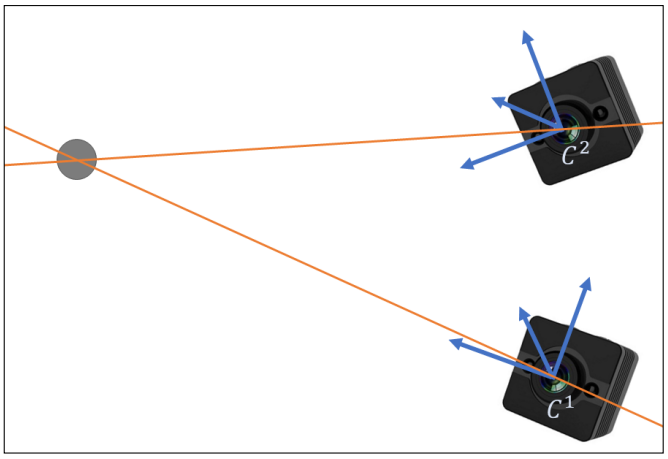

Figure 1.3. Sphere Marker Position Detection.

Two lines from one local camera frame will only collide at origin of local camera frame which is a trivial solution. This is the reason why at least two (stereo) motion capture cameras are required to measure position of a sphere marker in space.

1.2 Medical Usage

Medical procedures especially in head region of patient requires precision and non-invasiveness that in fact difficult for an human to provide. On the other hand, robotics and surgical navigation can provide precise, accurate, non-invasive operations. Before the advent of computers, a neurosurgical procedure named stereotaxy was developed. Stereotaxy requires the exact localization and targeting of intracranial structures for the placement of electrodes, needles, or catheters. Initially, this problem was addressed using anatomical drawings as an atlas for intracranial target planning and with the help of mechanical head frames attached to the patient's skull. The planned target could then be transferred onto the actual intraoperative patient setup [1]. Later medical imaging methods such as Computer Tomography (CT) and Magnetic Resonance Imaging (MRI) used for third dimensional mapping of the parts of the patient.

Surgical navigation can be achieved by different approaches, most commonly by using electromagnetism, ultrasound waves or by using light waves, to capture and relay patient's anatomy and the surgeon's precise movements in relation to the patient, to

computer monitors in the operating room. By using light waves, optical motion capturing can be done using cameras and special selected points. By tracking positions and orientations of patient and surgical tools, tools' relative position and orientation can be transmitted in virtual space.

In order to use surgical navigation methods effectively in surgical operations relations between different reference coordinate systems (patient, motion capture system, surgical tool reference coordinate systems etc.) should be known with precision. Accordingly in related literature many different approaches has been suggested. Arun, Huang and Bolstein [2] have described how the relation between two coordinate systems may be calculated using least squares method. The writers have taken advantage of a point cloud, every element of which known according to both coordinate systems, in their work. Hong and friends [3] have developed a robust method of calculating a transformation that describes relation between workspace and virtual environment by using fiducial points taken from surface of the body of patient. In their future work [4] they reduced surgical navigation registration error by the help of an ultrasonic based motion capture system to add fiducial points found inside of the patient body to their recommended method. Same writer and friends have successfully performed operations like inner ear surgery [5], chest surgery [6] and bone surgery [7] with related methodology. Fitzpatrick, West and Maurer [8, 9] have estimated target registration error (TRE) values from known fiducial registration error (FRE) or fiducial localization error (FLE) values.

Although there exist limited literature regarding with the advances of surgical navigation, its usage continually increases by the technological advances and ongoing researches.

This thesis tries to focus on the utilization of low cost motion capture systems for surgical navigation in operations that requires high precision as cochlear microrobot operations. Throughout the thesis, methodologies to relate various reference frames inside the operation workspace were discussed and implemented to the motion capture system by using mostly open source software. Also a design of mock-up model was proposed in order to verify the calibration of the motion capture cameras. At the end of the thesis, using the acquired knowledge, 6 degree of freedom robot manipulator was implemented to the setup and its usage was verified for future macro-micro surgical navigation purposes.

2. SURGICAL NAVIGATION SYSTEM

In scope of this work, low cost Optitrack motion capture system is planned to be used for real-time acquisition of position and orientation data of any arbitrary rigid bodies in space. System includes three Optitrack V100R2 infrared cameras to be used for navigation in measurement space and official software (Motive Tracker). Time coupled tracking data of rigid bodies obtained by official software processing visual data being streamed from the cameras (Figure 2.1).

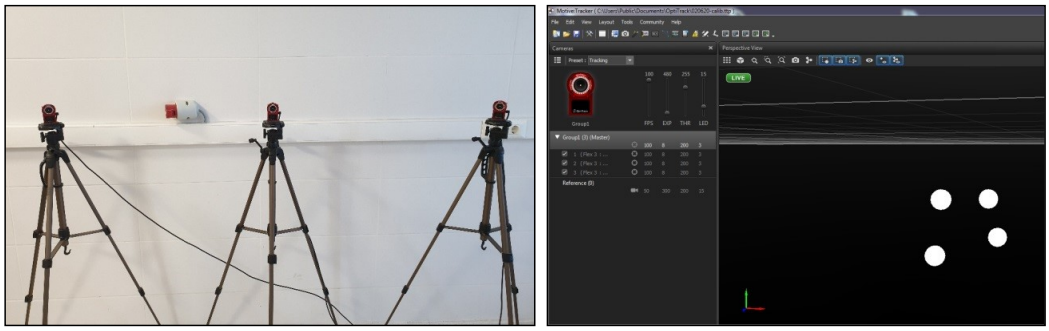

Figure 2.1. Optitrack V100R2 Motion Capture Cameras and Motive Tracker Software.

Motion capture cameras whose important technical properties are presented in table 2.1 are able to track positions of infrared reflective spheres passively reflecting infrared light and positioned in measurement space.

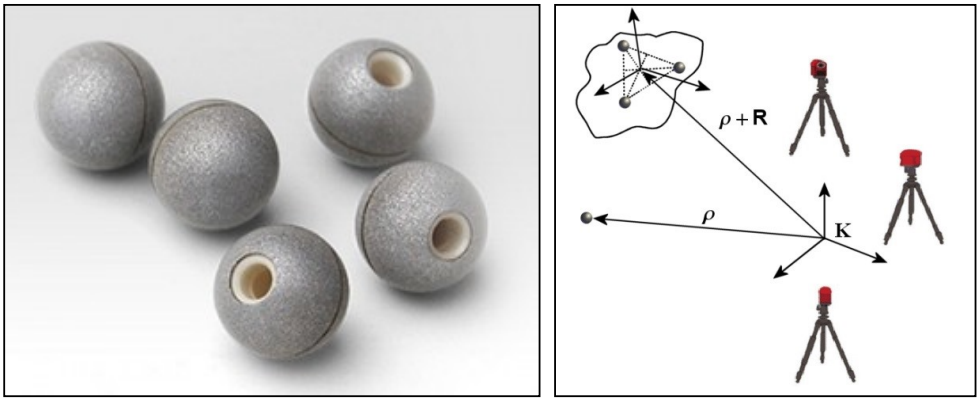

Figure 2.2. Passive Infrared Reflective Spheres and Requirements of Tracking.

Table 2.1. Motion Capture Cameras Technical Specifications.

Optitrack V100R2		
Visual Sensor	**Lenses, Filter and Light Source**	**Other**
Pixel Size: $6\mu m \times 6\mu m$	Lens: $4.5mm$ F1.6	Dataport: USB 2.0
Sensor Size: $4.5mm \times 2.88mm$	Horizontal Field of View: $64°$	Power: 5V 490mA
Resolution 640×480 (0.3 MP)	Vertical Field of View: $35°$	Dimensions: $45.2mm \times 36.6mm \times 74.7mm$
Frames per Second: $25, 50, 100$	Filter: $800nm$ Infrared longpass	Weight: 0.1kg
Delay: $10ms$	Light Source: 26 LED, $850nm$ Infrared	-
Snapshot Speed: $20\mu s$ minimum	-	-

At this point, while one infrared reflective sphere is enough to track position of any arbitrary point with respect to tracking system reference (K), orientation and position of an arbitrary rigid body can only be calculated by at least three infrared reflective spheres positioned in an asymmetrical arrangement (Figure 2.2).

Important elements required for performing tracking like power management, synchronization and computer connection of the cameras are managed by Optihub 2 center hub. Center hub and related schematic of connection are shown in figure 2.3

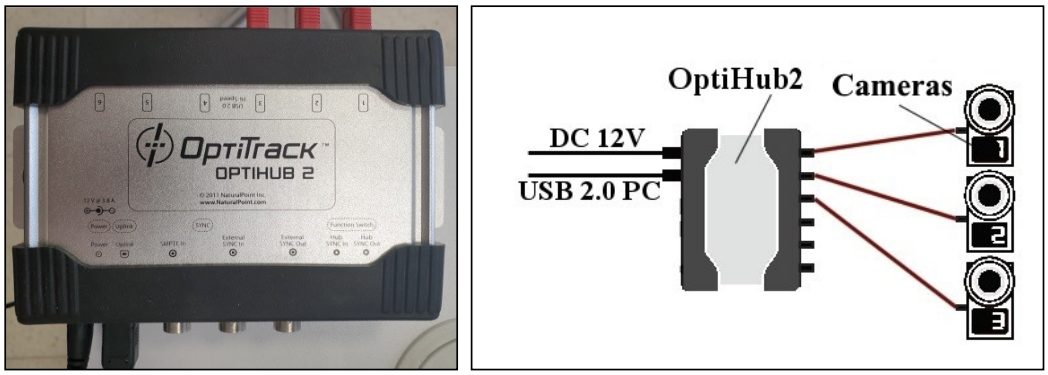

Figure 2.3. Optihub2 and Tracking System Connections Schematic

In scope of this work, official software of utilized tracking system, Optitrack Motive Tracker version 1.7.2 is used for calculating spatial positions of passive infra reflective sphere and positions and orientations of rigid bodies described by three or more in-

frared reflective spheres by processing visual data being acquisitioned from cameras. Related software is also have the ability to stream these position an orientation data to third party applications in real time through NatNet protocol by providing a module and an application programming interface (API). Details of the usage of this data will be given in throughout the thesis.

3. TRACKING SYSTEM CALIBRATION PROCEDURE

The tracking system can track infrared reflective sphere positions in three dimensional workspace by utilization of at least two motion capture cameras. In classical surgical navigation systems, motion capture cameras providing stereo vision are fixed in a rigid frame. Due to this fact, related systems can be merchandised pre-calibrated by respective manufacturers (Figure 3.1).

Figure 3.1. Classical Surgical Navigation Systems (NDI, Atracsys).

But within the scope of the thesis, as opposed to the classic merchandised solutions, motion capture cameras of optical tracking system are placed in environment by the user. This provides user with the big advantage of adjusting relative positions of motion capture cameras to provide better fit workspace, however vision distortion, and position and orientation between cameras are not known during setup of the optical tracking system. Due to this fact, before usage, a procedure of calibration must be applied to the optical tracking system, finding relations between cameras and their optical distortions.

Optical tracking system's official software Optitrack Motive Tracker provides software calibration solution of Optitrack's optical tracking system by help of some specialized hardware requirements. A metal tool containing three infrared reflective spheres whose positions are precisely known relative to each other is provided as hardware requirement of calibration procedure. Relative distances between infrared reflective spheres can be changed to other predefined distances on same or different tool also provided by manufacturer to better fit the desired workspace (Figure 3.2).

Figure 3.2. Optitrack Calibration Tool CW-500.

During calibration procedure, calibration tool should be gently moved in workspace in such orientations that infrared reflective spheres on the tool can be in front of the motion capture cameras. Whole workspace should be scanned and sampled with movements of the calibration tool while ensuring all infrared reflective spheres are visible for each motion capture camera (Figure 3.3).

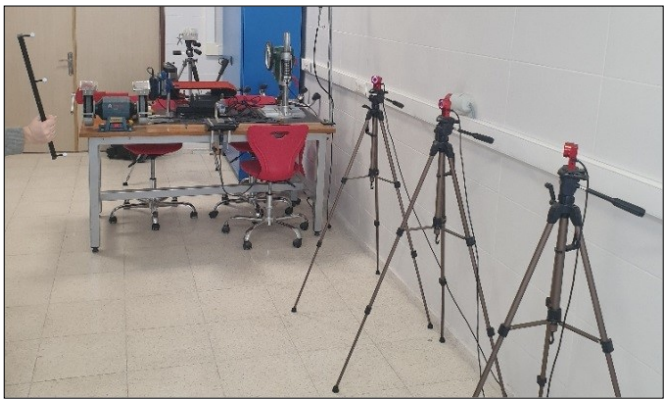

Figure 3.3. Application of Calibration on Motion Tracking Cameras.

Each motion capture camera collects visual samples in process to calculate their related position, orientation and optical distortion. two dimensional visual data collected from each motion capture cameras are utilized together with the known distances of the infrared reflective spheres on the calibration tool in creation of a workspace where all relations of the cameras are known by the software with a determined error value (Figure 3.4).

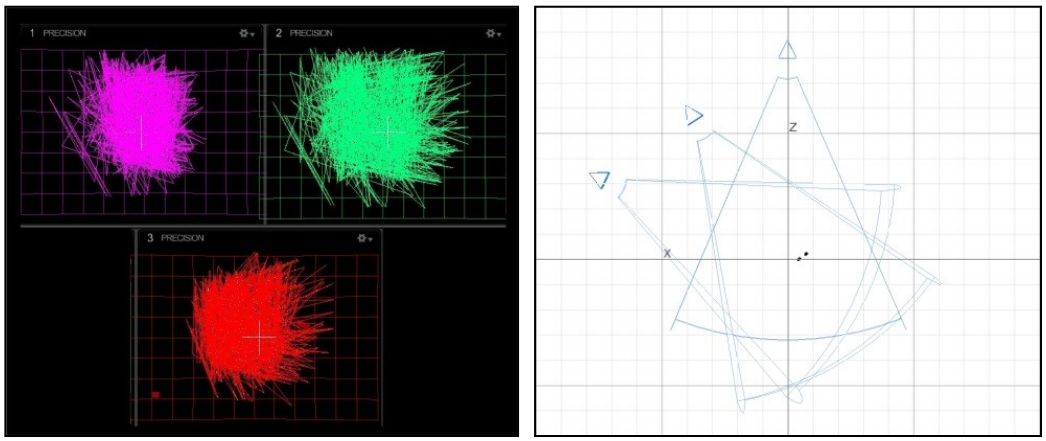

Figure 3.4. Camera Lens Distortions and Third Dimensional Motion Capture Volume.

Software outlines all the results obtained from calibration process and presents to the user. Figure 3.5 shows sample calibration outline performed in laboratory.

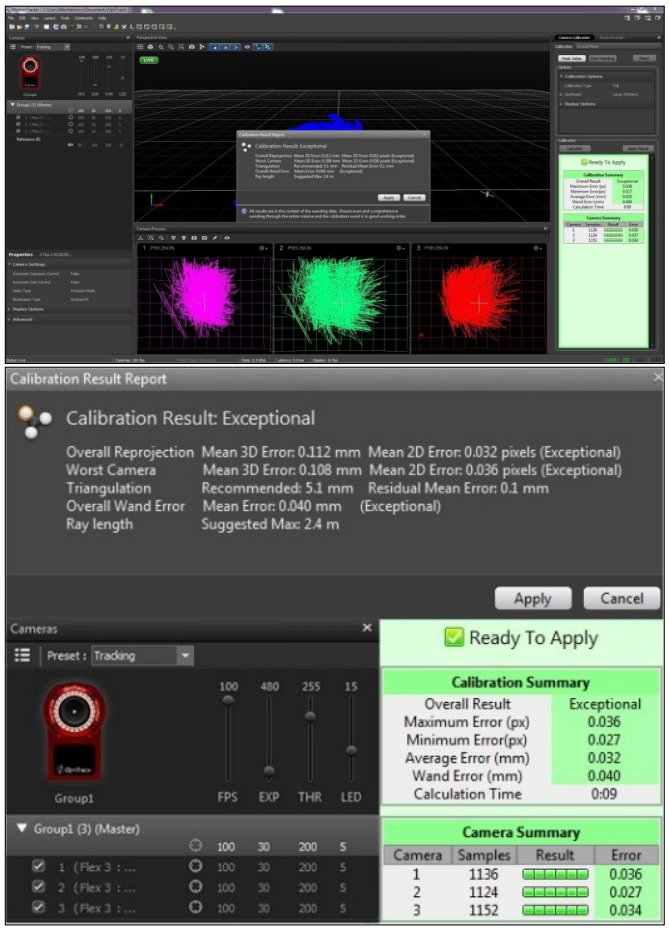

Figure 3.5. Result Screen of Tracking System Calibration.

It should be pointed out that the three dimensional motion tracking workspace reference (tracking system reference) is arbitrarily decided by the the software and every calibration relations are defined by this reference (K). Motive Tracker software allows user to define tracking system reference by using a official hardware although this method is not used in the thesis for sake of not adding extra uncertainties (Figure 3.6).

Figure 3.6. Tracking System Reference Defining Frame.

4. DEFINITIONS OF DATA AND STREAMING

In scope of the thesis, tracking systems official software Optitrack Motive Tracker 1.7.2 is utilized as a base of measurement of workspace tracking data. Related software, in real-time, processes visual two dimensional images acquired from calibrated motion capture cameras to calculate infrared reflective sphere positions defined by tracking system reference in the workspace. By creating a point cloud with asymmetric configuration of three or more infrared reflective spheres, real time orientation data alongside the center of point cloud position can also be acquired by defining related geometry as a rigid body in the official software (Figure 4.1).

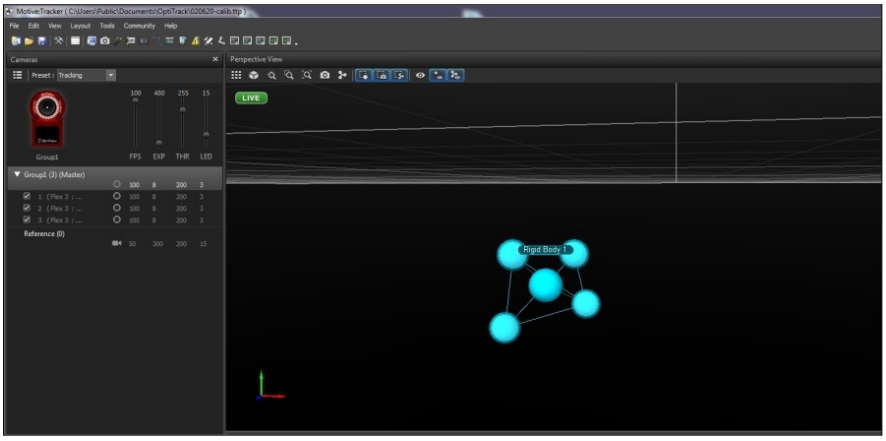

Figure 4.1. An Example of A Rigid Body Defined Within Software.

Although Optitrack Motive Tracker is a proprietary software, It allows real time streaming of the data calculated inside software to third party applications with NatNet module. NatNet module send packets of data over network using NatNet protocol. NatNet protokol enables streaming of infrared reflective sphere positions, defined rigid body positions, and orientations and positions of infrared reflective spheres that defines the rigid body. While it is not possible to easily distinguish between infrared reflective spheres, sphere that is defined in a rigid body can be distinguished by related rigid body's label (Figure 4.2).

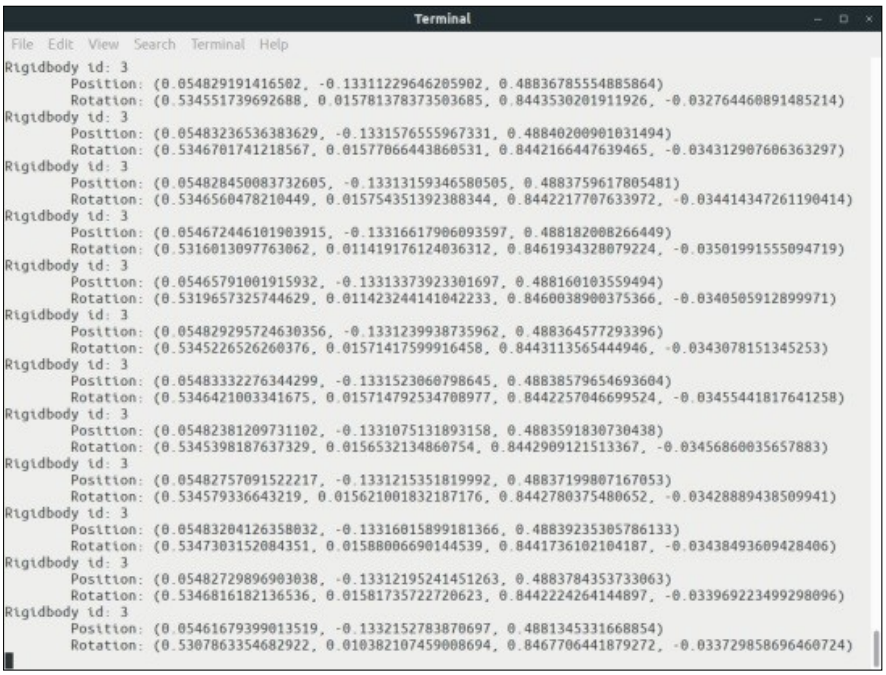

Figure 4.2. Data Stream Thorough NatNet.

In surgical navigation performed in scope of this thesis, free and open source software 3D Slicer [10] figure 4.3 is used in parallel to the Motive Tracker. 3D Slicer is mainly specialized for medical image informatics, image processing, and three-dimensional visualization, also can be extented to be utilized in surgical navigation using SlicerIGT [11] extension.

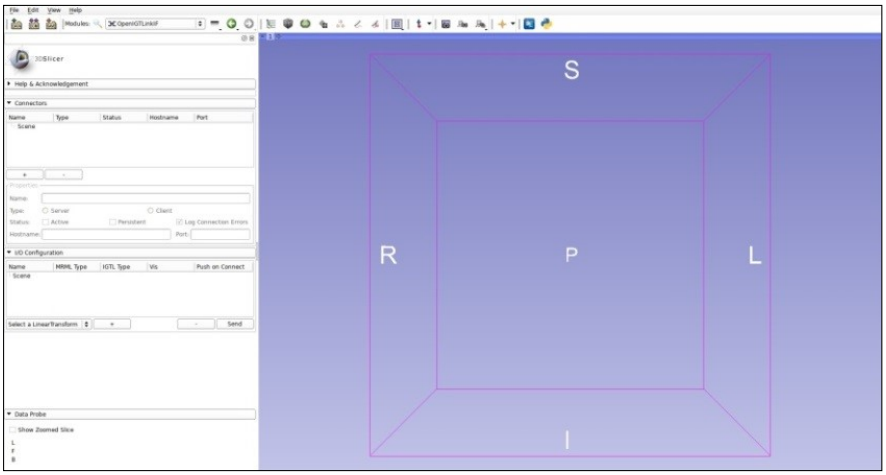

Figure 4.3. 3D Slicer Software.

NatNet protocol is not supported by 3D Slicer. Due to this fact 3D Slicer is not able to get the data being streamed from Motive Tracker directly. With this reason also free and open source software Plus ToolKit [12], a collection of many applications and li-

braries for data collection, processing, calibration and real time positioning, vision and other sensor data collection, is used for establishing communication between 3D Slicer and Motive Tracker. Plus Toolkit software can translate data being streamed from Motive tracker to OpenIGTLink protocol, which can be used in 3D Slicer software with SlicerIGT extension (Figure 4.4).

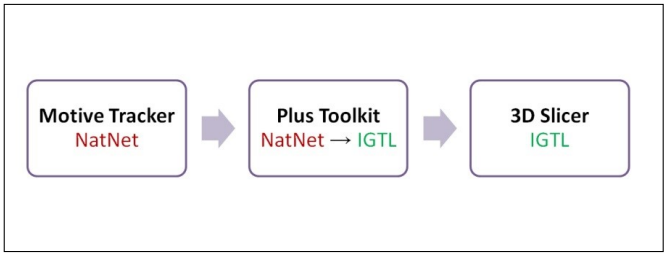

Figure 4.4. Data Stream Between Motive Tracker and 3D Slicer Software.

Plus ToolKit only officially supports streaming from Motive Tracker versions 1.10.3 and above. Since Motive Tracker software at the laboratory is version 1.7.2, source code of the Plus ToolKit is modified and made able to understand data encoded in NatNet protocol used by related version of the Motive Tracker.

5. REGISTRATION OF ANY ARBITRARY REFERENCE WITH TRACKING SYSTEM REFERENCE

Previous sections declares that after calibration procedure, tracking system reference is created automatically by Motive Tracker or manually by user using a official hardware tool. Measurement data taken from motion tracking system are given relative to the tracking system reference in surgical navigation applications. Related reference system is the most basic property of the created third dimensional motion tracking volume. This volume includes a local volume shown in the figure 5.1. Each action to be tracked is performed in this local volume which has its own reference.

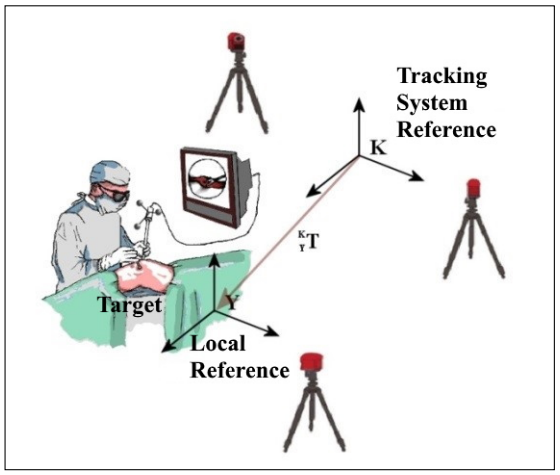

Figure 5.1. An Application of Surgical Navigation.

In surgical navigation application, the target with local reference is positioned inside third dimensional motion tracking volume with a specific orientation. Since surgical navigation application is executed on related target, a registration procedure should be performed on local system reference (Y) and tracking system reference (K), calculating a transformation matrix (${}^{K}_{Y}\boldsymbol{T}$) in order to provide visual feedback operator surgeon. In this way, any rigid body (surgical tool etc.) used by surgeon that is tracked with respect to tracking system reference can be described with respect to target reference. In scope of this thesis, analytical approach and least squares approach are considered for registration procedure.

5.1 Need for Registration

In order to explain why registration is needed for surgical navigation a basic navigation scenario is constructed as follows, two identical objects can be arbitrarily placed in space, one of these objects can thought to be real object while other one is it's virtual image. The only difference between these objects are the real object is described by R reference coordinate frame and Virtual Object is described by V reference coordinate frame (Figure 5.2)

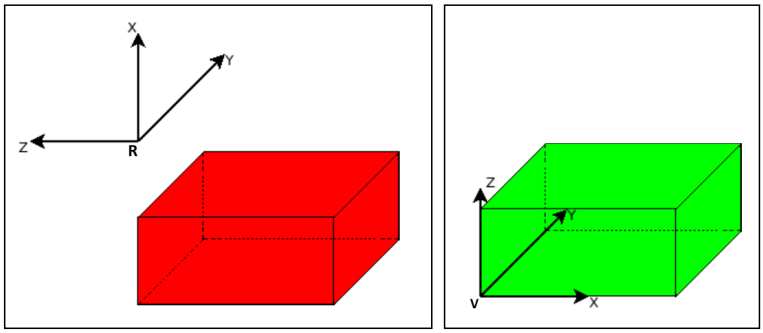

Figure 5.2. Real Object (Patient / Workspace) and Virtual Object (Image).

If a probe is taken to an arbitrary P point on real object, on R space it can be seen touching P point on real object. However in V space, probe will be seen far away from P' point which is the corresponding point of the P point on virtual object. If R and V space are viewed together, the error can be clearly seen in figure 5.3 and a observer in V space cannot tell which point the probe is touching on real object.

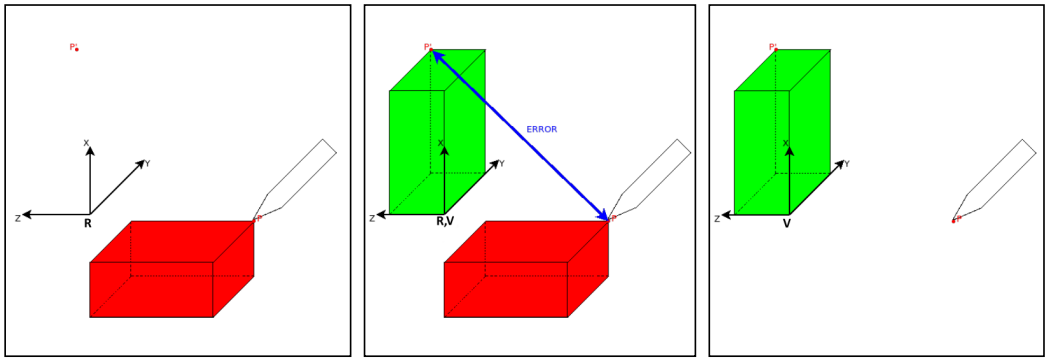

Figure 5.3. View in R Space, R and V Space, V Space Before Registration.

By applying a C transformation to V space, Any arbitrary P point in R space can be mapped to it's corresponding P' point in V space. Objective of registration is finding the C transformation that relates R and V space with minimal error.

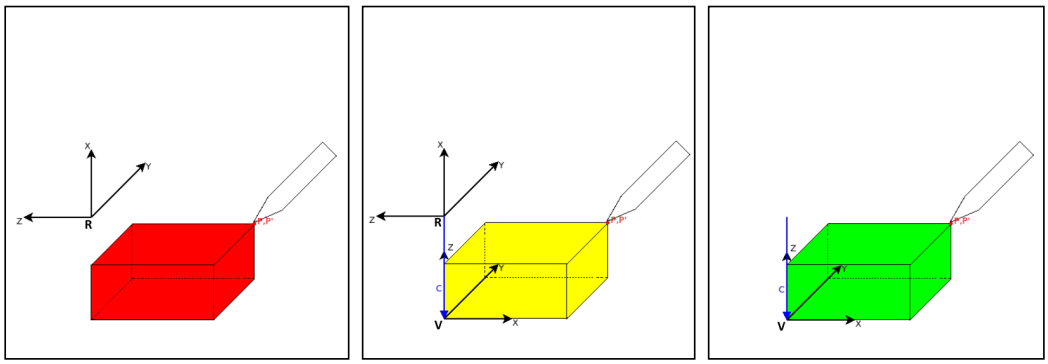

Figure 5.4. View in R Space, R and V Space, V Space After Registration.

5.2 Analytical Approach

Relation of an arbitrary point with respect to two reference systems is shown in equation 5.1.

$$\begin{bmatrix} {}^{K}\boldsymbol{\rho} \\ 1 \end{bmatrix} = {}^{K}_{Y}\boldsymbol{T} \begin{bmatrix} {}^{Y}\boldsymbol{\rho} \\ 1 \end{bmatrix} \tag{5.1}$$

In equation 5.1, ${}^{K}\boldsymbol{\rho} = \begin{bmatrix} {}^{K}\rho_x & {}^{K}\rho_y & {}^{K}\rho_z \end{bmatrix}^T$ and ${}^{Y}\boldsymbol{\rho} = \begin{bmatrix} {}^{Y}\rho_x & {}^{Y}\rho_y & {}^{Y}\rho_z \end{bmatrix}^T$ are position vectors defining related points with respect to K and Y reference systems. ${}^{K}_{Y}\boldsymbol{T}$ transformation matrix includes ${}^{K}_{Y}\boldsymbol{R} = \begin{bmatrix} {}^{K}\hat{\boldsymbol{x}}_Y & {}^{K}\hat{\boldsymbol{y}}_Y & {}^{K}\hat{\boldsymbol{z}}_Y \end{bmatrix}$ rotation matrix that defines relative orientation between references and ${}^{K}\boldsymbol{\rho}_Y = \begin{bmatrix} {}^{K}\rho_Y x & {}^{K}\rho_Y y & {}^{K}\rho_Y z \end{bmatrix}^T$ translation vector that defines relative distance vector between origins of references. (Equation 5.2)

$${}^{K}_{Y}\boldsymbol{T} = \begin{bmatrix} & {}^{K}_{Y}\boldsymbol{R} & & {}^{K}\boldsymbol{\rho}_Y \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

As seen on equation 5.2, ${}^{K}_{Y}\boldsymbol{T}$ transformation matrix consist of twelve parameters, nine of these are included in ${}^{K}_{Y}\boldsymbol{R}$ rotation matrix and rest of these are from ${}^{K}\boldsymbol{\rho}_Y$ translation vector. In order to ${}^{K}_{Y}\boldsymbol{T}$ transformation matrix to be numerically expressed, all of these parameters must be calculated. Due to this reason twelve independent equations including these parameters are needed. Using unit vectors in ${}^{K}_{Y}\boldsymbol{R}$, which are orthogonal with respect to each other, six equations can be obtained. (Equation 5.3)

$$\begin{aligned} {}^{K}\hat{\boldsymbol{x}}_Y.{}^{K}\hat{\boldsymbol{x}}_Y = 1, \quad {}^{K}\hat{\boldsymbol{y}}_Y.{}^{K}\hat{\boldsymbol{y}}_Y = 1, \quad {}^{K}\hat{\boldsymbol{z}}_Y.{}^{K}\hat{\boldsymbol{z}}_Y = 1 \\ {}^{K}\hat{\boldsymbol{x}}_Y.{}^{K}\hat{\boldsymbol{y}}_Y = 0, \quad {}^{K}\hat{\boldsymbol{x}}_Y.{}^{K}\hat{\boldsymbol{z}}_Y = 0, \quad {}^{K}\hat{\boldsymbol{y}}_Y.{}^{K}\hat{\boldsymbol{z}}_Y = 0 \end{aligned} \tag{5.3}$$

Six more equations that are still needed can be obtained by placing position vectors in equation 5.1, the position vectors are extracted from two arbitrary points in space whose positions are known with respect to both references. To summarize briefly, in theory two arbitrary points from workspace and whose positions vectors are known with respect to both references are enough to calculate relation between reference systems. (Figure 5.5)

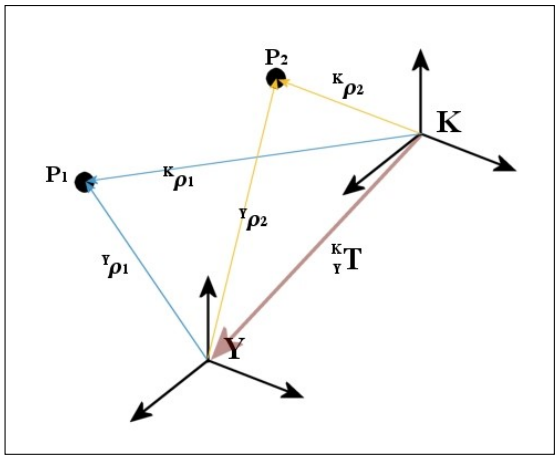

Figure 5.5. Determining Relation Between Reference Systems (2 Points).

But rotating one or both of the references according to the axis defined by utilized arbitrary points will not affect related point positions. Due to this reason there exists infinite solutions for desired ${}^{K}_{Y}\boldsymbol{T}$ transformation matrix provided by stated theoretical approach. Even through every possible solution will perfectly match two utilized points, in a real application, calculation of ${}^{K}_{Y}\boldsymbol{T}$ transformation matrix using only two points will cause high errors in navigation stage where any other point is transformed to local reference using equation 5.1 due to orientation uncertainty. These errors can be reduced by eliminating orientation uncertainty by using more points.

For the sake of utilizing more arbitrary points in calculation of ${}^{K}_{Y}\boldsymbol{T}$ transformation matrix, six equations obtained from ${}^{K}_{Y}\boldsymbol{R}$ rotation matrix properties (equation 5.3) will not be used. In this condition, to calculate related transformation matrix, number of independent equations should be increased back to twelve. Therefore four arbitrary points in space whose positions are known with respect to both references $({}^{K}\boldsymbol{\rho}_i, {}^{Y}\boldsymbol{\rho}_i \;\; i = 1, 2, 3, 4)$ should be utilized in equation 5.1 (Figure 5.6).

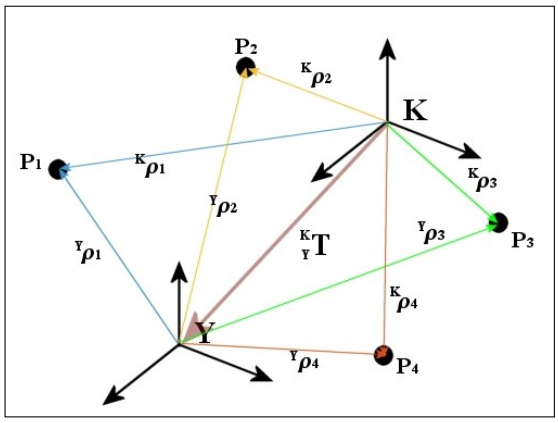

Figure 5.6. Determining Relation Between Reference Systems (4 Points).

At this point, presented analytical solution assumes all points taken from workspace and utilized in equation 5.1 are measured with respect to both reference systems without any error. However in real applications measurement errors are unavoidable due to uncertainties of manufacturing, motion capturing and human input so unfortunately usability of this methodology with high precision is low. Uncertainties in measurement and not using constraint equations obtained from orthogonality of ${}^{K}_{Y}\boldsymbol{R}$ rotation matrix for sake of utilizing four arbitrary points will cause vector components of calculated rotation matrix to deviate from being unit vector. Thus related rotation matrix will have non unity determinant. This fact will certainly cause distortions (shear and zoom) in registration and, by extension, in workspace (Figure 5.7).

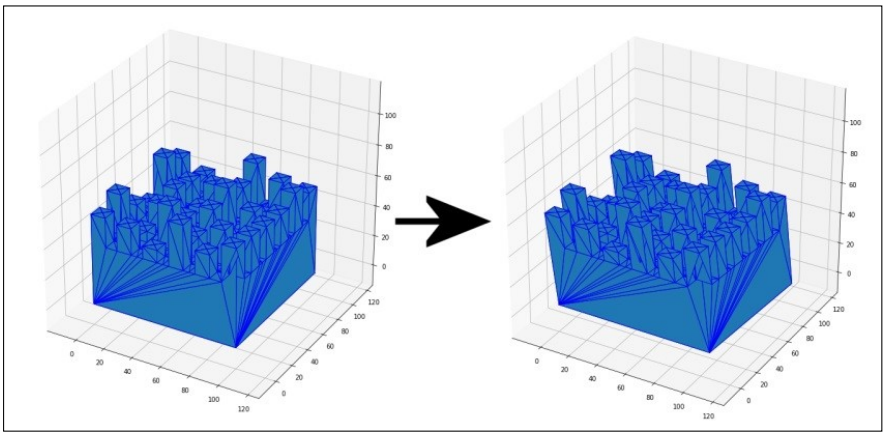

Figure 5.7. Distortion Caused by Analytic Solution in Real Applications.

In light of this, least squares approach is decided to be utilized instead of analytical approach due to the factors discussed.

5.3 Least Squares Approach

Although relation of a random point cloud to any arbitrary two reference systems is described in equation 5.1, It is not suitable to use same equation a point cloud taken from an environment with measurement uncertainties. In this case, relation between two position sets (${}^{K}\boldsymbol{\rho}_i, {}^{Y}\boldsymbol{\rho}_i \quad i = 1, 2, ..., n$), obtained from n points measured from both references, is defined by equation 5.4

$$\begin{bmatrix} {}^{K}\boldsymbol{\rho}_i \\ 1 \end{bmatrix} = {}^{K}_{Y}\boldsymbol{T} \begin{bmatrix} {}^{Y}\boldsymbol{\rho}_i \\ 1 \end{bmatrix} + \begin{bmatrix} \boldsymbol{e}_i \\ 0 \end{bmatrix} \quad i = 1, 2, ..., n \tag{5.4}$$

where $\boldsymbol{e}_i$ is error vector caused by uncertainties in measurement environment. If equation 5.4 is rearranged by separating transformation matrix into rotation matrix and translation vector equation 5.5 will be formed.

$${}^{K}\boldsymbol{\rho}_i = {}^{K}_{Y}\boldsymbol{R}\,{}^{Y}\boldsymbol{\rho}_i + {}^{K}\boldsymbol{\rho}_Y + \boldsymbol{e}_i \quad i = 1, 2, ..., n \tag{5.5}$$

Rearranging equation 5.5, error vector can be defined as (5.6)

$$\boldsymbol{e}_i = {}^{K}\boldsymbol{\rho}_i - {}^{K}_{Y}\boldsymbol{R}\,{}^{Y}\boldsymbol{\rho}_i - {}^{K}\boldsymbol{\rho}_Y \quad i = 1, 2, ..., n \tag{5.6}$$

Objective of approach is to used for determining relation between references, least squares method, is to minimize square sum of error vectors (equation 5.7) by obtaining eligible rotation matrix ${}^{K}_{Y}\boldsymbol{R}$ and translation vector ${}^{K}\boldsymbol{\rho}_Y$.

$$\Sigma \boldsymbol{e}_i^2 = \sum_{i=1}^{n} \left\| {}^{K}\boldsymbol{\rho}_i - {}^{K}_{Y}\boldsymbol{R}\,{}^{Y}\boldsymbol{\rho}_i - {}^{K}\boldsymbol{\rho}_Y \right\|^2 \tag{5.7}$$

If the rotation matrix and translation vector that provides minimum sum of error squares value is shown as ${}^{K}_{Y}\boldsymbol{R}'$ and ${}^{K}\boldsymbol{\rho}_Y{}'$ respectively, centroid of ${}^{K}\boldsymbol{\rho}_i \quad i = 1, 2, ..., n$ position set (${}^{K}\overline{\boldsymbol{\rho}}$) and centroid of the position set ${}^{K}\boldsymbol{\rho}_i{}' \quad i = 1, 2, ..., n$ (${}^{K}\overline{\boldsymbol{\rho}}'$) obtained from equation 5.8 are equal to each other [2].

$${}^{K}\boldsymbol{\rho}_i{}' = {}^{K}_{Y}\boldsymbol{R}'\,{}^{Y}\boldsymbol{\rho}_i + {}^{K}\boldsymbol{\rho}_Y{}' \quad i = 1, 2, ..., n \tag{5.8}$$

Definitions of related parameters are shown in equation 5.9

$$
\begin{aligned}
{}^{K}\overline{\boldsymbol{\rho}} &= {}^{K}\overline{\boldsymbol{\rho}}' \\
{}^{K}\overline{\boldsymbol{\rho}} &= \frac{1}{n}\sum_{i=1}^{n} {}^{K}\boldsymbol{\rho}_i \\
{}^{K}\overline{\boldsymbol{\rho}}' = \frac{1}{n}\sum_{i=1}^{n} {}^{K}\boldsymbol{\rho}_i{}' &= {}^{K}_{Y}\boldsymbol{R}' \, {}^{Y}\overline{\boldsymbol{\rho}} + {}^{K}\boldsymbol{\rho}_Y{}' \\
{}^{Y}\overline{\boldsymbol{\rho}}' &= \frac{1}{n}\sum_{i=1}^{n} {}^{Y}\boldsymbol{\rho}_i{}'
\end{aligned}
\tag{5.9}
$$

At this point by demeaning positions sets, two parameters shown in equation 5.10 will be obtained

$$
\begin{aligned}
{}^{K}\boldsymbol{q}_i &= {}^{K}\boldsymbol{\rho}_i - {}^{K}\overline{\boldsymbol{\rho}} \\
{}^{Y}\boldsymbol{q}_i &= {}^{Y}\boldsymbol{\rho}_i - {}^{Y}\overline{\boldsymbol{\rho}}
\end{aligned}
\tag{5.10}
$$

Then equation 5.7 can be rewritten using these parameter as equation 5.11

$$
\Sigma \boldsymbol{e}_i^2 = \sum_{i=1}^{n} \left\| {}^{K}\boldsymbol{q}_i - {}^{K}_{Y}\boldsymbol{R} \, {}^{Y}\boldsymbol{q}_i \right\|^2 \tag{5.11}
$$

Consequently, problem of least squares approach is reduced to

- calculation of ${}^{K}_{Y}\boldsymbol{R}'$ rotation matrix that minimizes equation 5.11,
- calculation of ${}^{K}\boldsymbol{\rho}_Y{}'$ translation vector using found ${}^{K}_{Y}\boldsymbol{R}'$ rotation matrix.

Given steps below, detailed derivation described in [2], should be followed to obtain ${}^{K}_{Y}\boldsymbol{R}'$ rotation matrix and ${}^{K}\boldsymbol{\rho}_Y{}'$ translation vector that relates position sets $({}^{K}\boldsymbol{\rho}_i, {}^{Y}\boldsymbol{\rho}_i \;\; i = 1, 2, ..., n)$ obtained from measuring n points in space.

- Centroids ${}^{K}\overline{\boldsymbol{\rho}}$ and ${}^{Y}\overline{\boldsymbol{\rho}}$ are calculated from position sets expressed by different references ${}^{K}\boldsymbol{\rho}_i$ and ${}^{Y}\boldsymbol{\rho}_i \;\; i = 1, 2, ..., n$.
- 3x3 $\boldsymbol{H}$ matrix is calculated by utilizing calculated centroids and position sets obtained from equation 5.10 (equation 5.12)

$$
\boldsymbol{H} = \sum_{i=1}^{n} {}^{Y}\boldsymbol{q}_i \, {}^{K}\boldsymbol{q}_i{}^{T} \tag{5.12}
$$

- Singular value decomposition is performed on calculated $\boldsymbol{H}$ matrix (equation 5.13).

$$\boldsymbol{H} = \boldsymbol{U}\boldsymbol{\lambda}\boldsymbol{V}^T \tag{5.13}$$

- ${}^K_Y\boldsymbol{R}'$ rotation matrix is calculated by using equation 5.14.

$${}^K_Y\boldsymbol{R}' = \boldsymbol{V}\boldsymbol{U}^T \tag{5.14}$$

- ${}^K\boldsymbol{\rho}_Y{}'$ translation vector is calculated by using equation 5.15.

$${}^K\boldsymbol{\rho}_Y{}' = {}^K\overline{\boldsymbol{\rho}} - {}^K_Y\boldsymbol{R}'\,{}^Y\overline{\boldsymbol{\rho}} \tag{5.15}$$

As a result of related procedure ${}^K_Y\boldsymbol{T}$ transformation matrix that minimizes square sum of error can be easily constructed using ${}^K_Y\boldsymbol{R}'$ rotation matrix and ${}^K\boldsymbol{\rho}_Y{}'$ translation vector.

6. VERIFICATION OF TRACKING SYSTEM CALIBRATION

As can be seen, motion tracking system, region of surgical operation, virtual environment and obtaining relations between related coordinate frames with minimal error are the most important factors in performing surgical navigation procedures effectively. The transformations relating these coordinate frames are obtained by performing registration on measured position sets of fiducial points from different reference frames. Errors in registration, surface in transformations relating references and directly effect precision of navigation. Basically these errors are produced from calibration of motion tracking system and pointer tool, consistency between real model (workspace), where operation being performed on and virtual model where navigation being performed on, and right selection of fiducial points by operator. At this point a preoperation verification methodology should be created for minimizing uncertainties produced from related factors or determining main source of errors. By considering calibration of motion tracking system is closest factor to navigation hardware, focus of the created methodology should be verification of the calibration results. Accordingly a mock-up model is manufactured to be used in calibration verification of Optitrack V100R2 motion capture cameras.

6.1 Reference Mock-Up Model and Verification Methodology

A rigid mock-up model with markings on known fiducial point coordinates defined from own reference system should be used for verification of tracking system calibration. Within scope of created methodology and procedure of verification, positions of markings on the rigid mock-up model will be measured from tracking system reference by utilizing a pointer tool and motion tracking system software. (Figure 6.1)

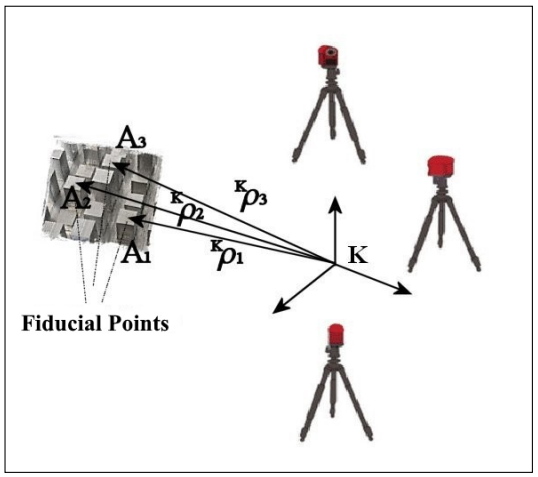

Figure 6.1. Measurement of Fiducial Points with Respect to Tracking System Reference.

From this point, half of obtained points and their precisely known related pair's coordinates on mock-up model will be used in registration with least squares approach and transformation matrix that provides minimal registration error. Utilizing obtained transformation matrix, square mean error of registration will be calculated and so registration error (Fiducial registration error - FRE) will be found. After this stage, rest of the obtained points and their respective pairs on mock-up model will be used for calculating target registration error (TRE) using same transformation matrix obtained before. Both error values being in accepted interval will point to the fact of calibration of tracking system is sufficient.

Rapid prototyping methodology (3D printing) is used for manufacturing first design iteration of mock-up model. Figure 6.2 shows first mock-up model designed and manufactured.

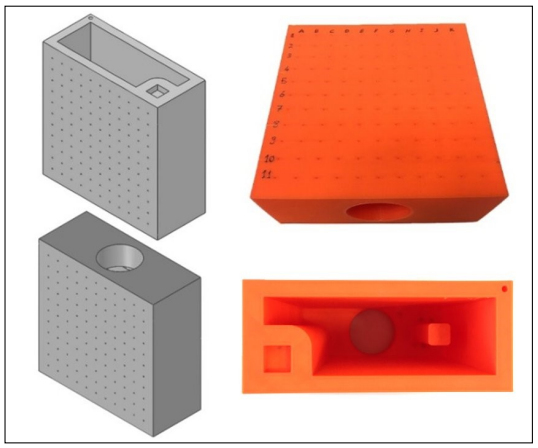

Figure 6.2. Reference Mock-up Model (First Prototype).

There are 11×11 fiducial point markings on two sides of the mock-up model. Also cylindrical slot for holder and connection point for rigid body markers are added into the design. However, in the mock-up model designed using 3D model design, It has been observed that there are cracks and warpings induced by rapid prototyping system and the measurement uncertainties caused by these. The relevant mechanical defects shown in figure 6.3 prevented the produced mock-up prototype model from being used as a calibration mock-up reference.

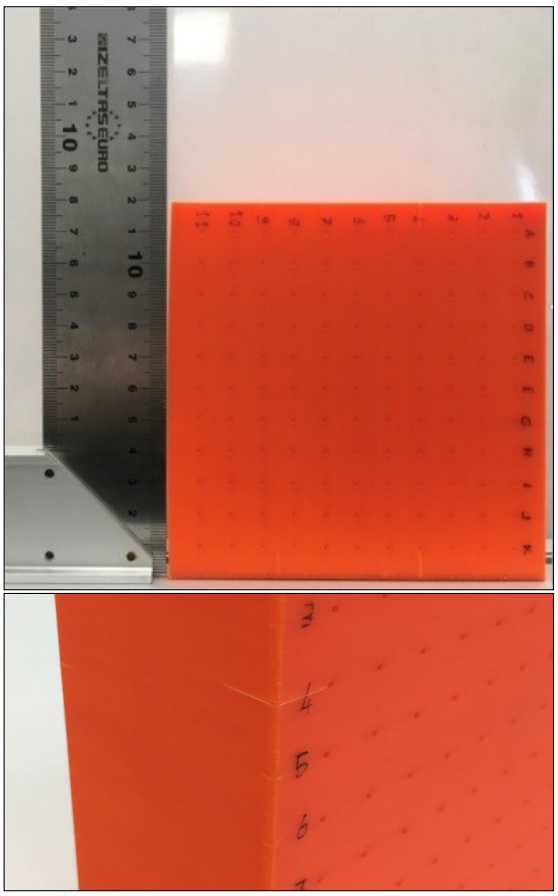

Figure 6.3. Warping and Cracks on First Prototype of Mock-up Model.

Also an unpredictable design error stands out in same prototype. There are enough number of fiducial points to be used in registration but these fiducial points are placed in only two planes. Procedure of calibration verification will be carried out in a three dimensional volume. Dependence of two planes serves as a disadvantage for obtaining meaningful fiducial (FRE) and target (TRE) registration errors. At this point It is decided to make changes in related design.

Second designed prototype mock-up model is shown in figure 6.4. Fiducial points are placed on towers of differing lengths for avoiding having all fiducial points on

same plane. This configuration enabled fiducial point selection from different planes. Also small cross sections used for minimizing mechanical defects caused from rapid prototyping system.

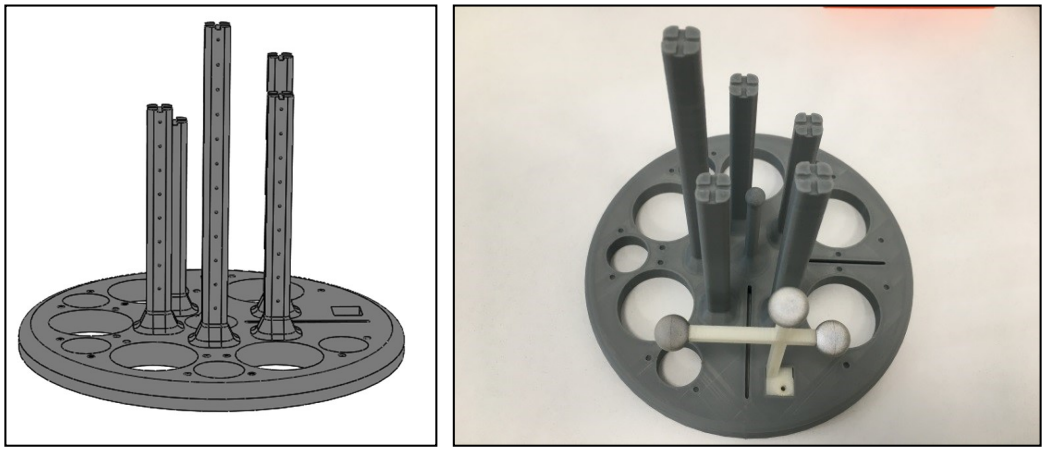

Figure 6.4. Reference Mock-up Model (Second Prototype).

Cylindrical slots for holder and connection point for rigid body markers can be clearly seen in figure. Although no visible mechanical defects were observed on prototype mock-up model, during registration deflection of the towers were observed.

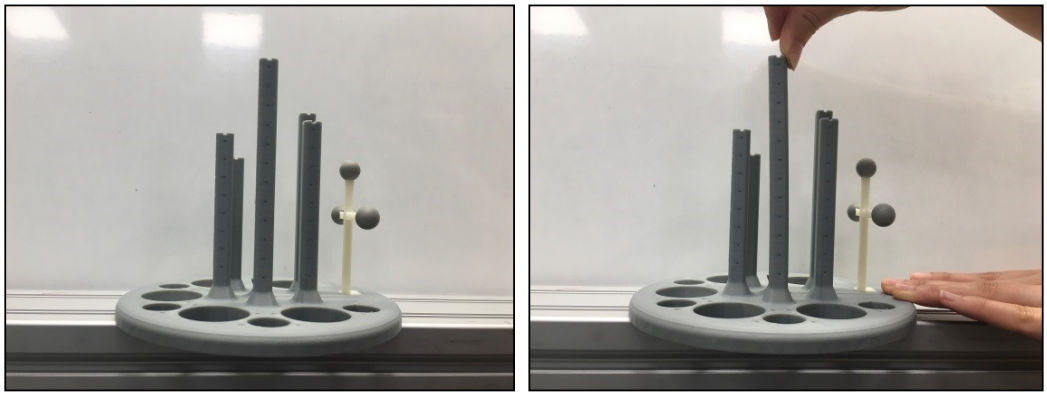

Figure 6.5. Observed Deflection of Second Prototype of Mock-up Model.

As it can be clearly seen from figure 6.5, occurred deflection proves that the design is prone to user caused measurement uncertainties during verification procedure. Due to this reason second manufactured mock-up prototype was decided to be not suitable as a calibration mock-up reference.

Additional uncertainties are introduced to verification methodology, independent of design, in both prototyping stages. In both reference mock-up prototypes, rapid prototyping methodology was utilized for manufacturing. For interest of reducing uncertainties caused from manufacturing and usage of reference mock-up model, Güngör

Makine GM MILL CNC 3-axis processing system is decided to be utilized in manufacturing instead of rapid prototyping system. In this context, based on previous designs, related reference mock-up model is updated to latest design.

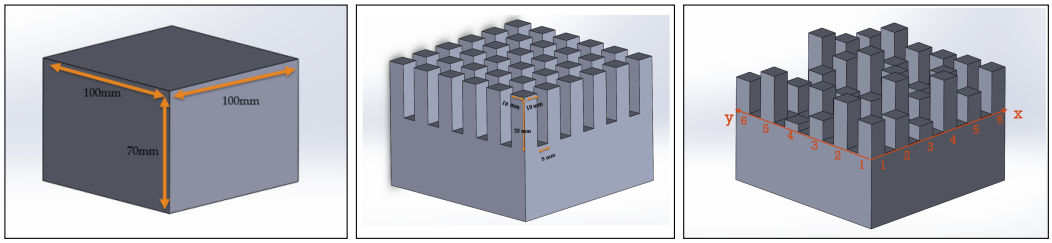

Figure 6.6. Reference Mock-up Model (Third Prototype).

three dimensional design procedure of reference mock-up model can be seen in figure 6.6. In total 6×6 towers with dimensions of $10 \times 10 \times 30$ mm are obtained from a rigid body with dimensions $100 \times 100 \times 70$ by creating five equally spaced canals on both x and y axis and from surface of the rigid body, with 30 mm depth and 8 mm width. In this way centers of top surfaces of towers can be used as fiducial point locations. Each tower height should be differentiated without depending on a specific parameter to avoid problem of having all fiducual points on same plane. Accordingly heights of each tower decided by creating a randomized matrix of shape 6×6 with each element of which being an integer between 1 and 30. table 6.1 shows value of each randomized element with their x and y axis rank as shown in figure 6.6.

Siemens SinuTrain emulation software, which is fully compatible with Siemens Sinumeric 828D controller, is utilized in programming and emulation stages of CNC based manufacturing of designed reference mock-up model (Figure 6.7). Possible problems that might arise in part manufacturing were observed and manufacturing risks were reduced to minimum by the usage of emulation software with same CNC software as controller.

Table 6.1. Reference Mock-up Model Tower Heights.

Rank	**1**	**2**	**3**	**4**	**5**	**6**
1	19	17	8	29	15	24
2	26	7	28	13	22	23
3	11	16	12	22	8	21
4	4	18	10	28	17	20
6	26	15	17	20	6	26
7	18	11	5	28	20	18

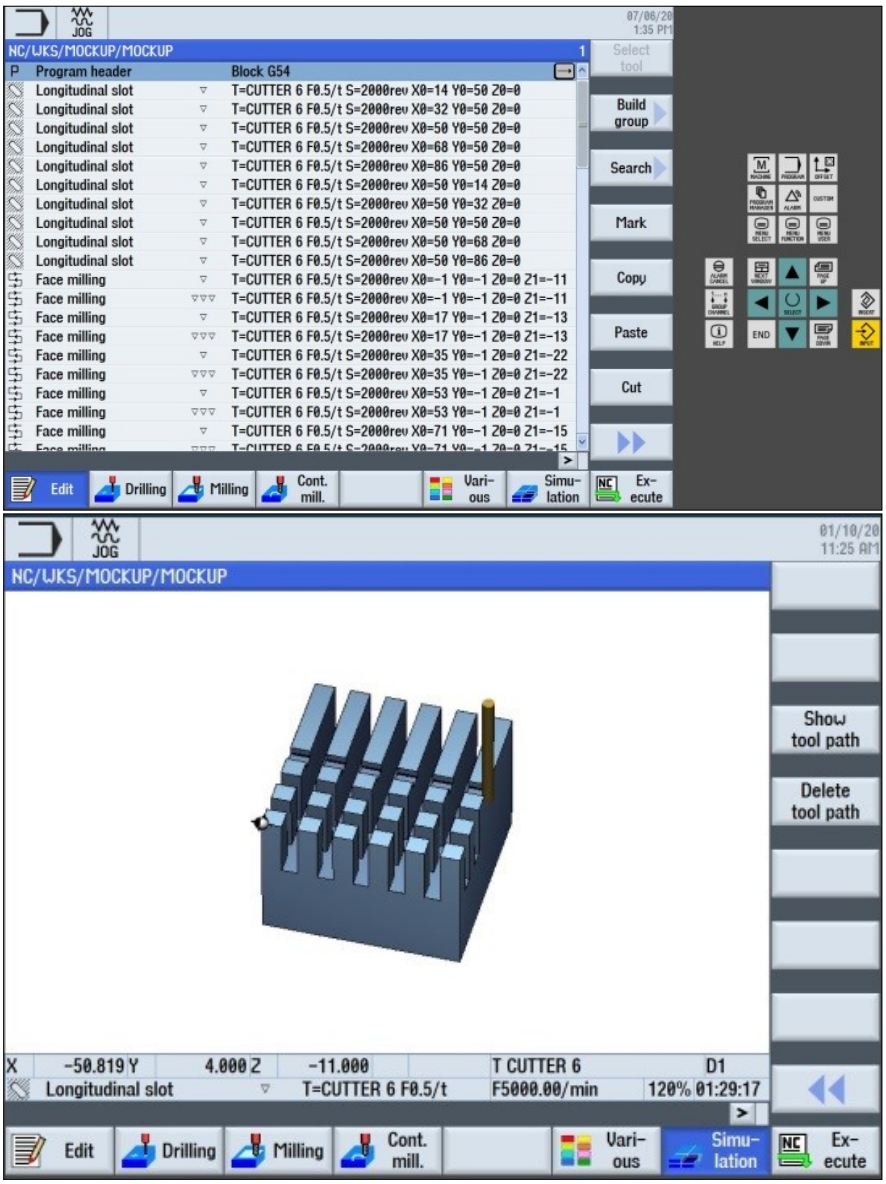

Figure 6.7. Siemens SinuTrain Coding and Emulation Interface.

An aluminum block is selected to be used as base material of reference mock-up model considering ease of processing, resistance to corrosion and low density. Programming that done in Siemens Sinutrain emulation software was transferred to Siemens Sinunumeric 828D controller, and manufacturing process is started after fixing bulk part to CNC machine (Figure 6.8-6.9). CNC parameters used in manufacturing process were given on table 6.2.

Table 6.2. CNC Manufacturing Parameters.

Parameter	Specification
Cutting Tool	M6 End Mill
Lathe Speed	2000 rotations/minute
Feed Speed	0.5 mm/tooth
Tolerance	0.001 mm

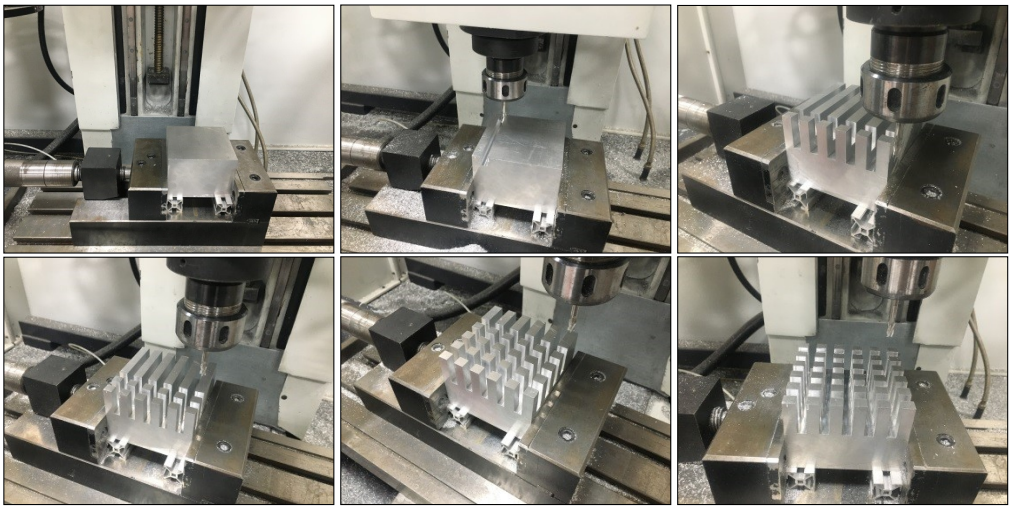

Figure 6.8. Grooving Parallel to X and Y axis.

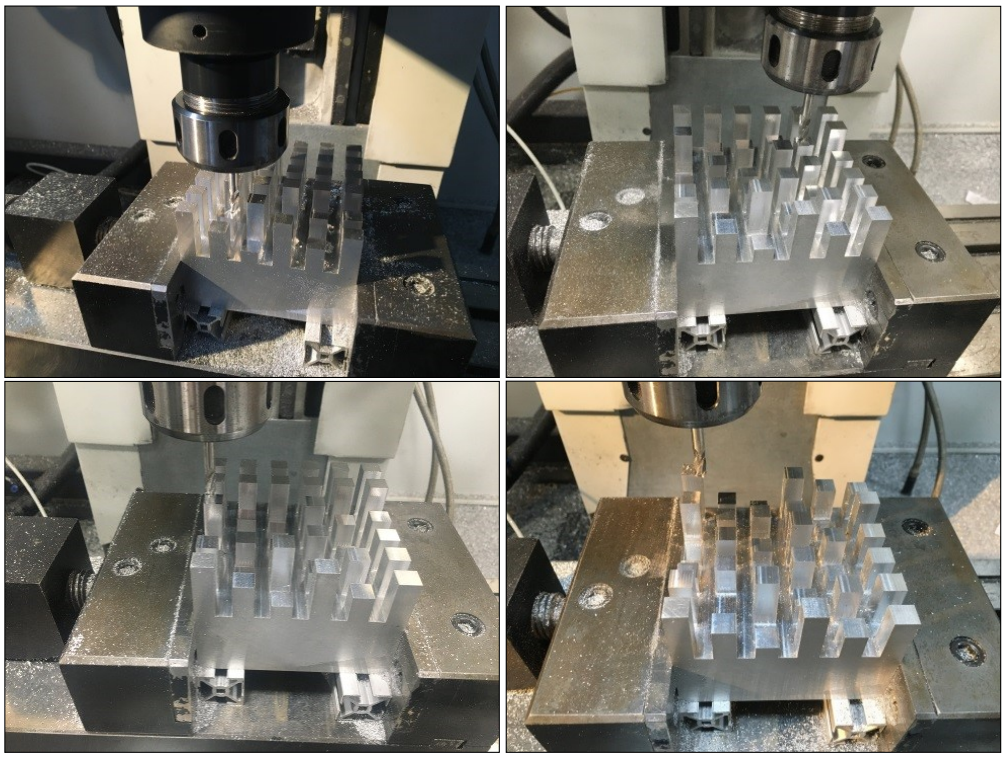

Figure 6.9. Adjusting Tower Heights.

After completing design and manufacturing process of reference mock-up model, without taking model from CNC (without shifting coordinate frame), markings of depth 0.025 mm were created on geometrical middle points of top surfaces of towers with random heights to allow comfortable position measurement with pointer tool considering geometrical middle points of top surfaces of towers are fiducial locations. M1.6X4 centering drill bit is utilized in creating these markings and manufacturing procedure of reference mock-up model is completed (Figure 6.10).

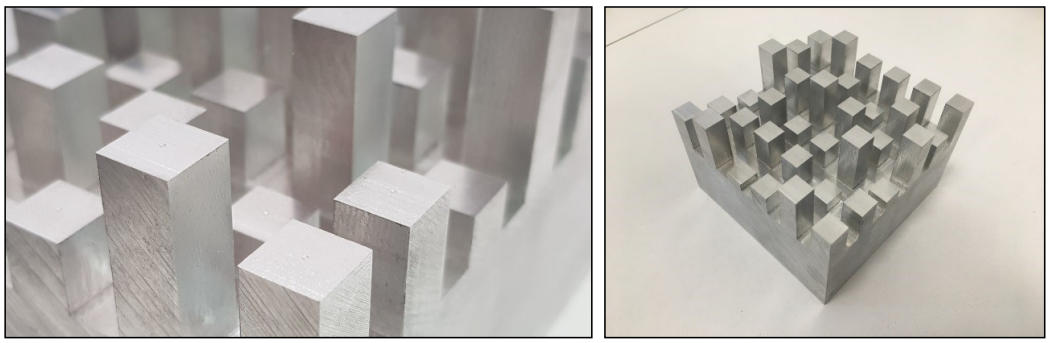

Figure 6.10. Markings Created on Tower Top Surfaces and Fully Done Reference Mock-up Model.

6.2 Design of Rigid Body Markers

Tracking of any rigid body in operation workspace by motion tracking system requires rigid body markers. Rigid body markers is a geometrical tool containing three or more passive reflecting or active light emitting spheres (Figure 6.11) in a asymmetric arrangement used for position tracking of points in optical motion tracking systems (motion capture cameras), like of which utilized in this thesis. By fixing rigid body marker on a rigid body, real time position and orientation tracking of related rigid body can be performed. Basically this procedure can be summarized as adding a reference system on any rigid body that is desired to be tracked. Software creates reference systems automatically for each rigid body marker by processing data obtained from tracking system. Tracking system is sensing light originated from light emitting (active) or light reflecting (passive) spheres positioned on rigid body marker in a special geometry.

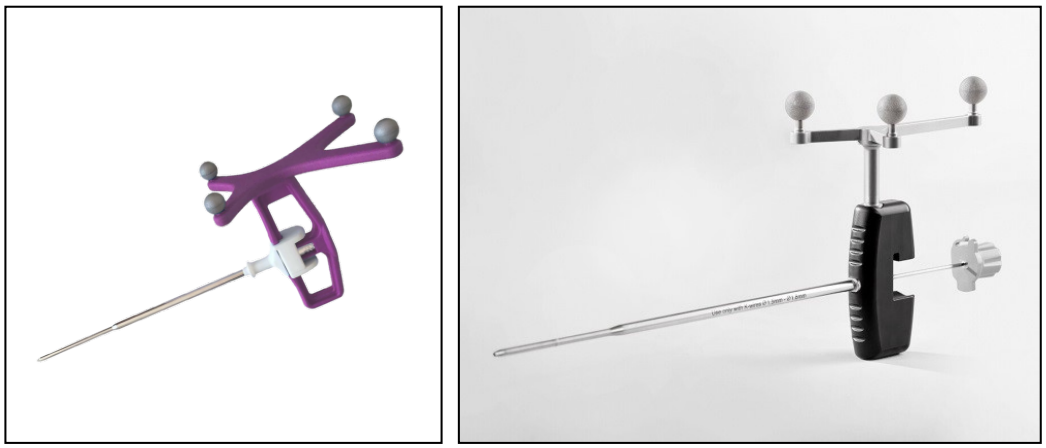

Figure 6.11. Rigid Body Markers on Several Pointer Tools and Infrared Reflective Spheres (IZI Medical, BRAINLAB).

As noted in calibration verification methodology, fiducial point positions of mock-up model positioned inside workspace must be measured with respect to tracking system reference to be able to verify calibration. In accordance to this process, two different rigid body markers should be used for tracking both mock-up model and pointer tool (Figure 6.12).

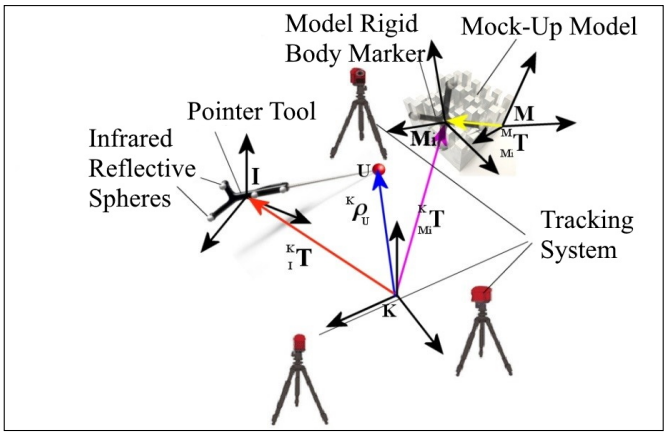

Figure 6.12. Utilization of Markers During Calibration Verification.

Calibration verification and measurement process is clearly shown in figure 6.12. Positions and orientations of marker references that is assembled to pointer tool (I) and mock-up model (Mi) can be measured by tracking system software with respect to tracking system reference (K). Transformation matrices ${}^{K}_{I}\boldsymbol{T}$ and ${}^{K}_{Mi}\boldsymbol{T}$ can be easily obtained. Pointer tip reference (U), which is known with respect to pointer tool reference, can be described with respect to tracking system reference by the utilization of related transformation matrices (Equation 6.1).

$$\begin{bmatrix} {}^{K}\boldsymbol{\rho}_U \\ 1 \end{bmatrix} = {}^{K}_{I}\boldsymbol{T} \begin{bmatrix} {}^{I}\boldsymbol{\rho}_U \\ 1 \end{bmatrix}$$
$$\begin{bmatrix} {}^{Mi}\boldsymbol{\rho}_U \\ 1 \end{bmatrix} = {}^{K}_{Mi}\boldsymbol{T}^{-1} \begin{bmatrix} {}^{K}\boldsymbol{\rho}_U \\ 1 \end{bmatrix} \tag{6.1}$$

Thereby pointer tip point can be utilized for measuring fiducial point positions on markings of mock-up with respect to mock-up rigid body marker reference (Mi). Since geometry structure of mock-up model is precisely known with respect to model reference (M), transformation matrix ${}^{M}_{Mi}\boldsymbol{T}$ can be found utilizing least squares approach discussed in section 5.3. Fiducial (FRE) and target (TRE) registration errors between measured and real mock-up fiducial point positions can be calculated using this transformation matrix.

Geometrical precision of pointer tip and distinguishability between rigid body markers by system software during tracking are crucial. Uncertainties will certainly arise in verification and registration procedures in case of geometrical defect of pointer tip or interference of rigid body markers by system software. For this reason, pointer tip precision, which is important for verification and navigation efficiency, is enhanced bt the usage of T0416 precise cylindrical shafted dead center with integrateable rigid body marker.

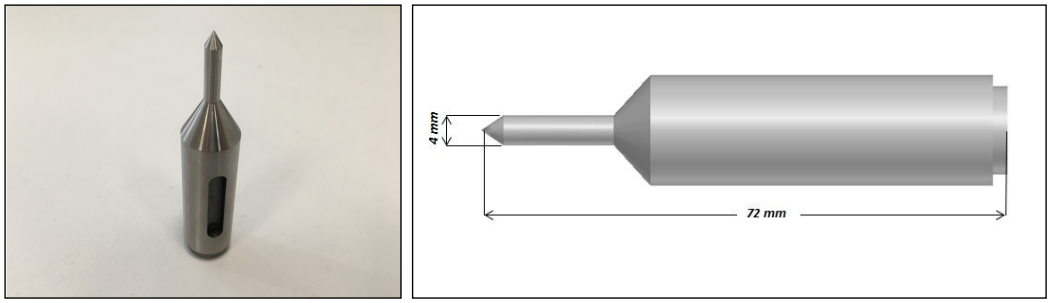

Figure 6.13. T0416 Precise Cylindrical Shafted Dead Center Which is Used as a Part of Measurement tool.

Tracking multiple rigid bodies simultaneously in workspace by tracking system requires each geometry created by infrared reflective spheres, positioned on each defined rigid body marker, to be different for all possible rigid body poses. Rigid body markers with similar geometrical positioning of reflective spheres cannot be distinguished from each other by tracking system. Therefore geometries of rigid body markers should be designed with this fact in mind. In literature, mostly marker designs presented by track-

ing system manufacturers are being used, and there are a limited number of works that are published on distinguishable marker designing. In this thesis, rigid body markers are designed according to specified constraints of related works [13, 14].

General marker design is shown in figure 6.14. Total number of reflective spheres is n and each reflective sphere defined as $\boldsymbol{m}_i \quad i = 1, 2, \ldots, n$. Each possible pair of reflective spheres creates a segment ($\boldsymbol{d}_i$), and total number of segments (S)

$$S = \frac{n(n-1)}{2} \tag{6.2}$$

is calculated using equation 6.2.

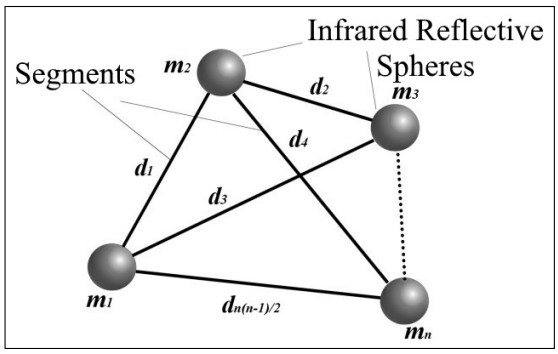

Figure 6.14. General Geometry of Rigid Body Markers.

Before continuing into marker design, design constraint of minimum (d_{min}) and maximum (d_{max}) lengths of segments should be decided. Tracking system accuracy increases as segment lengths increases however considering limited workspace caused by the nature of operations, maximum length should be kept within the constraints of work. For each tracking system, considering reflective spheres and differentiating segments, minimum length difference (Δd_{min}) and minimum angle difference ($\Delta \alpha_{min}$) parameters were provided. Accordingly procedure steps given below should be followed for marker design.

- Total number of segments calculated utilizing equation 6.2 considering number of reflective spheres to be used.
- Segment length are chosen considering determined limits $d_{min} \leq d_i \leq d_{max} \quad i = 1, 2, \ldots, S$.
- Check each difference of length between segments are not shorter than segment minimum length difference (Δd_{min}) $|d_i - d_j| \geq \Delta d_{min} \quad i, j = 1, 2, \ldots, S \;\; \& \;\; i \neq j$.

Designed rigid body marker is able to be defined to tracking system after the procedure. Same procedure steps should be followed for any additional rigid body marker desired to be used in tracking system. However additional procedure steps for two rigid body markers should be applied for tracking system to distinguish these rigid body markers.

- List segment line lengths of both designed markers $d_{1i},\ d_{2j}\ \ i = 1, 2, \ldots, S_1,\ j = 1, 2, \ldots, S_2$.
- Similar segments are found between markers by utilizing hardware segment minimum length difference (Δd_{min}) $|d_{1i} - d_{2j}| \geq \Delta d_{min}\ \ i = 1, 2, \ldots, S_1\ \ j = 1, 2, \ldots, S_2$.
- Considering segment pairs on same marker, angle between segments are calculated for both markers $\theta_{1ij} = \angle\, d^b_{1i}, d^b_{1j}\ ,\ \theta_{2ij} = \angle\, d^b_{2i}, d^b_{2j}\ \ i, j = 1, 2, \ldots, p\ \ \&\ \ i \neq j$
- Check absolute difference between calculated pairs are not smaller than hardware segment minimum angle difference ($\Delta\alpha_{min}$) $|\theta_{2ij} - \theta_{1ij}| \geq \Delta\alpha_{min}$

Rigid body markers designed by this procedure are distinguishable by the tracking system in the same workspace.

Optimal number of reflective spheres for single plane (every reflective sphere are positioned on a single plane) markers is between three and six. There can be seen a decrease of target registration error (TRE) by increasing number of reflective spheres on a marker however marker design also gets more complicated as number of reflective spheres on the tool increases. Most significant decrease of registration error is obtained by increasing number of reflective spheres to four from three [13]. Considering existing marker designs in literature and specifications of Optitrack V100R2 infrared cameras, constraints of marker designs are decided and shown in table 6.3.

Table 6.3. Marker Design Constraints.

Total Infrared Reflective Sphere Number (n)	**Minimum Segment Length** (d_{min})	**Maximum Segment Length** (d_{max})	**Hardware Segment Minimum Length Difference** (Δd_{min})	**Hardware Segment Minimum Angle Difference** ($\Delta\alpha_{min}$)
4	50 mm	100 mm	5 mm	2°

Accordingly rigid body markers for both pointer tool and mock-up model are designed by utilizing related procedure with design constraints (Figure 6.15) and prototypes of rigid body markers are manufactured (Figure 6.16-6.17).

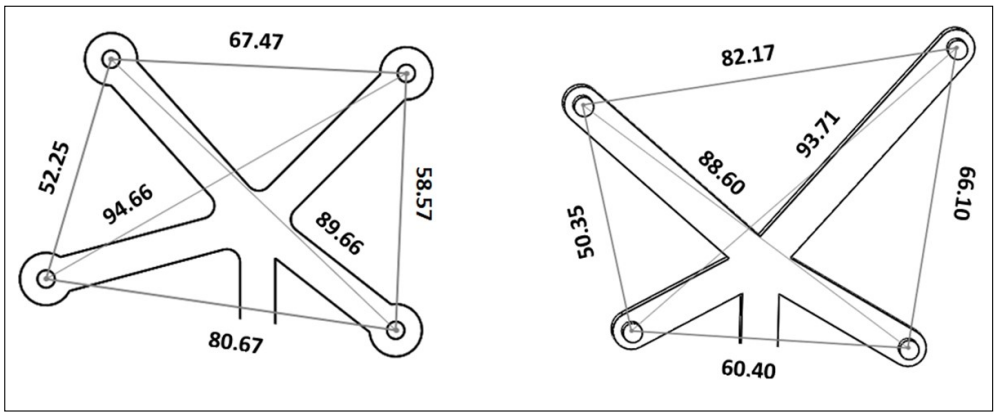

Figure 6.15. Designed Pointer Tool and Mock-up Model Marker Dimensions (mm).

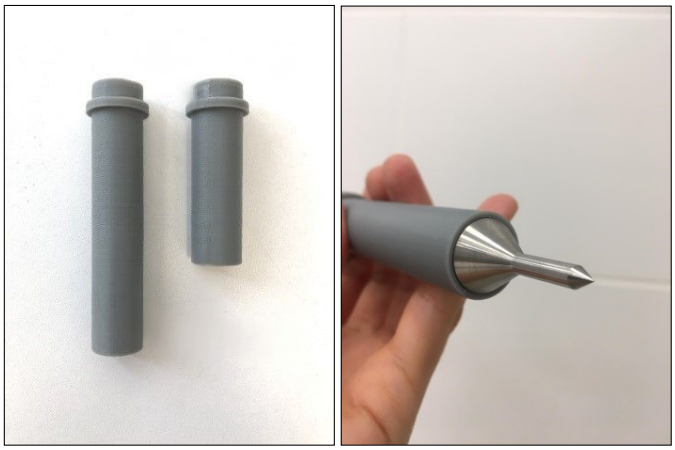

Figure 6.16. Some Measurement Tip Holder Parts of Pointer Tool.

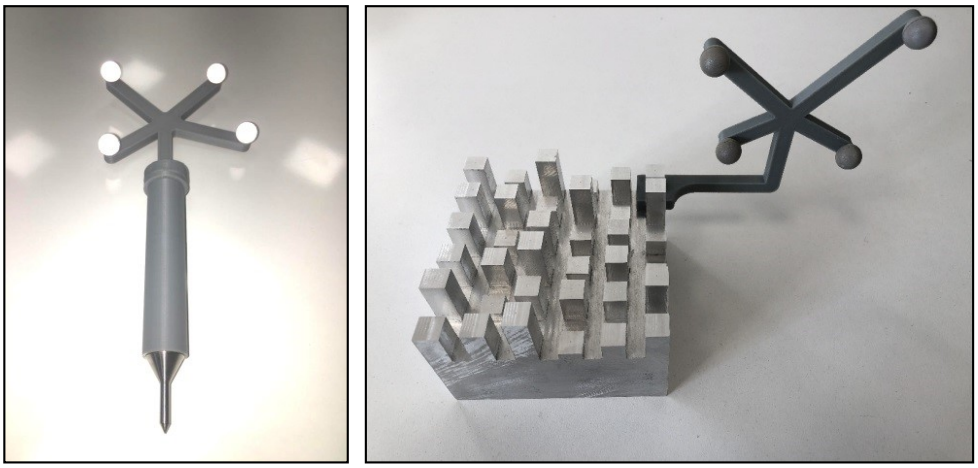

Figure 6.17. Prototypes of Pointer tool Marker and Mock-up Model Marker.

6.3 Calibration of Pointer Tool

If designed markers and mock-up model are to be used in tracking system verification procedure, tip position of pointer tool, which will be utilized in position measurement, is need be known with respect to pointer tool rigid body marker reference.

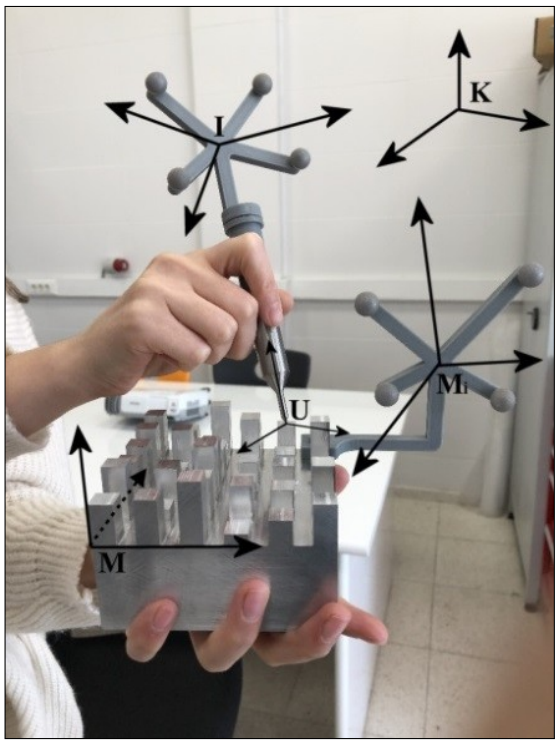

Figure 6.18. Demonstration of Taking Measurement From Mock-up Model.

Mock-up fiducial point position measurement is shown in figure 6.18, also previously mentioned need of defined pointer tip position can be seen. Transformation matrices ${}^{K}_{I}\boldsymbol{T}$ and ${}^{K}_{Mi}\boldsymbol{T}$ are known since designed markers are distinguishable, definable by tracking system.

However the transformation matrix ${}^{K}_{U}\boldsymbol{T}$ that defines position of the pointer tip and orientation of pointer tool with respect to the tracking system reference (K), which represents relation between pointer tip point reference (U) and tracking system reference, should be known in order to measure positions of fiducial points on mock-up model and visualize pointer tool in virtual environment correctly during surgical navigation. To calculate related transformation matrix,

$$ {}^{K}_{U}\boldsymbol{T} = {}^{K}_{I}\boldsymbol{T}\,{}^{I}_{U}\boldsymbol{T} \tag{6.3} $$

equation 6.3 can be utilized. But transformation matrix ${}^{I}_{U}\boldsymbol{T}$ that defines relation be-

tween pointer tool rigid body marker reference (I) and pointer tip point reference (U) should be known. Related transformation matrix can be found by the utilization of two different methods.

First method is based on precise knowledge of geometry parts of pointer tool assembly (rigid body marker, dead center). Considering centroid point of the reflective spheres of rigid body marker is recognized as the origin point of reference, tracking system defines reference orientation as same as tracking system reference (K) orientation at the point of definition (Figure 6.19)

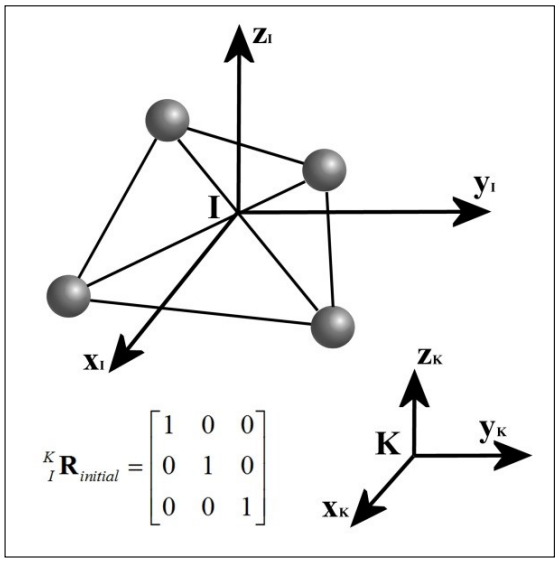

Figure 6.19. Orientation of Rigid Body Marker at Point of Definition.

This property allows to define orientation of the pointer tool by simply holding tool in a specific orientation when tracking system defines the rigid body marker of the pointer tool. Knowing orientation, pointer tip point position can be obtained from known geometry of pointer tool. However user error in orientating pointer in related orientation and tracking system's measurement uncertainty in defining rigid body marker will greatly affect precision or complicate calculation process of obtained ${}^{I}_{U}\boldsymbol{T}$ transformation matrix. Therefore ${}^{I}_{U}\boldsymbol{T}$ transformation matrix that relates pointer tool rigid body marker reference (I) and pointer tip point reference (U) is calculated utilizing, second method, pointer tool calibration. Basis of related procedure consists of two stages, which are pivot calibration and spin calibration. Both of these stages are executed utilizing SlicerIGT extension of 3D Slicer software.

6.3.1 Spin calibration

Spin calibration objective is to describe the z_U axis of the pointer tip point reference (U) with respect to pointer tool reference (I). Cylindrical measurement tip spin axis (m) is coincident with z_U axis. Utilizing this fact the ${}^{I}_{U}\boldsymbol{R}$ rotation matrix between pointer reference and pointer tip point reference can be calculated. Spin calibration ensures that orientation of pointer tool is consistent with the orientation of pointer tool in virtual environment. pointer tip point position with respect to pointer reference is the most important parameter for performing measurement on a point with pointer tool. During spin calibration, pointer tool is rotated along cylindrical measurement tip spin axis with minimal deviation, and orientation information during rotation is sampled by SlicerIGT (Figure 6.20).

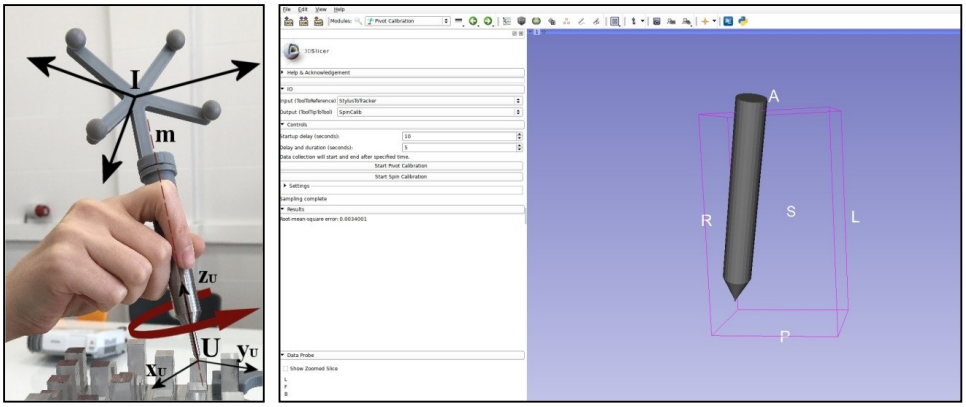

Figure 6.20. Rotation of Pointer Tool Along Spin Axis and 3D Slicer Visualization.

Sample orientations are collected from pointer tool reference and after collection of samples in a set amount of time SlicerIGT calculates spin axis of the cylindrical measurement tip with a root mean square (RMS) error produced from axis deviations and/or geometrical defects in pointer tool. Spin axis is calculated as a unit vector with respect to tracking system reference (K). Since ${}^{K}_{I}\boldsymbol{T}$ transformation matrix is known, calculated unit vector can be easily described by pointer tool reference (I). (Equation 6.4)

$$\begin{bmatrix} {}^{I}\hat{\boldsymbol{z}}_U \\ 1 \end{bmatrix} = {}^{K}_{I}\boldsymbol{T}^{-1} \begin{bmatrix} {}^{K}\hat{\boldsymbol{z}}_U \\ 1 \end{bmatrix} \tag{6.4}$$

x and y axis placement of pointer tool tip point reference are insignificant since pointer tool model in virtual environment is assumed to be symmetrical in x and y axis. At this point ${}^{I}_{U}\boldsymbol{R} = \begin{bmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & {}^{I}\hat{\boldsymbol{z}}_U \end{bmatrix}$ rotation matrix is obtained with compatible values to ${}^{I}\hat{\boldsymbol{z}}_U$ axis. After this step, pointer tip point position with respect to pointer tool reference

(I) should be calculated with pivot calibration to construct ${}^{I}_{U}\boldsymbol{T}$ transformation matrix.

6.3.2 Pivot calibration

Objective of pivot calibration is to determine pointer tip point position with respect to pointer tool reference (I) which is the origin point of pointer tip reference (${}^{I}\boldsymbol{\delta}_{U}$). Since pivot calibration is directly related to the point measurement procedure, precision of calibration is very important for both navigation and verification stages. During pivot calibration, pointer tool is moved like a platform of spherical mechanism where position of pointer tip point is fixed. SlicerIGT samples orientation of pointer tool reference with respect to tracking system reference during this movement (Figure 6.21).

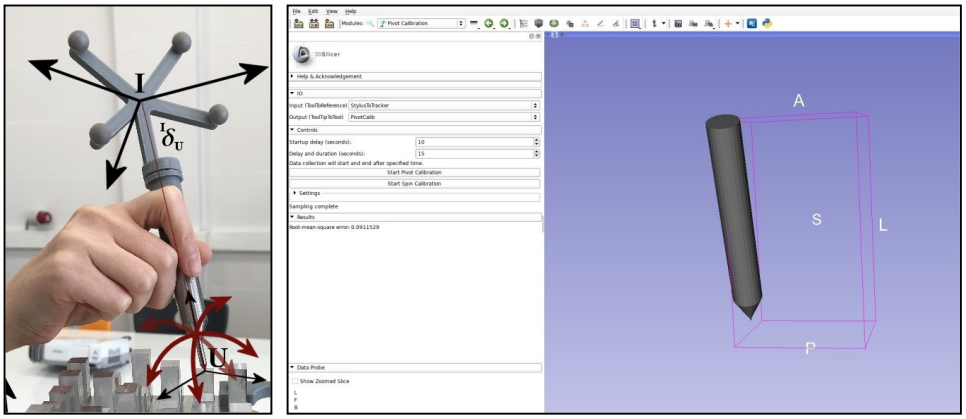

Figure 6.21. Spherical Movement of Pointer Tool with Pointer Tip as Isopoint and 3D Slicer Visualization.

Assuming pointer tool as a rigid body, unchanging position on pointer tool reference with respect to tracking system reference during calibration movement should be pointer tip point position. Samples collected are utilized by SlicerIGT in calculating a point with least root mean square (RMS) value considering position change. Similar to section 6.3.1, calculated vector can be easily described by pointer tool reference (I) since ${}^{K}_{I}\boldsymbol{T}$ transformation matrix is known. (Equation 6.5)

$$\begin{bmatrix} {}^{I}\boldsymbol{\delta}_{U} \\ 1 \end{bmatrix} = {}^{K}_{I}\boldsymbol{T}^{-1} \begin{bmatrix} {}^{K}\boldsymbol{\delta}_{U} \\ 1 \end{bmatrix} \tag{6.5}$$

As mentioned before, main objective of pointer tool calibration is to find ${}^{I}_{U}\boldsymbol{T}$ transformation matrix that describes relation between pointer tool reference (I) and pointer tip reference (U). Thereby ${}^{K}_{U}\boldsymbol{T}$ transformation matrix required in verification procedure can be calculated utilizing equation 6.3. At this point ${}^{I}_{U}\boldsymbol{T}$ transformation matrix can be constructed utilizing both spin and pivot calibration results. (Equation 6.6)

$$ {}^{I}_{U}\boldsymbol{T} = \begin{bmatrix} & {}^{I}_{U}\boldsymbol{R} & & {}^{I}\boldsymbol{\delta}_{U} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.6} $$

6.4 Design Improvements of Pointer Tool

As seen in figure 6.22, total of three parts are used in first prototype construction of pointer tool. These parts are precise measurement tip, measurement tip holder and rigid body marker.

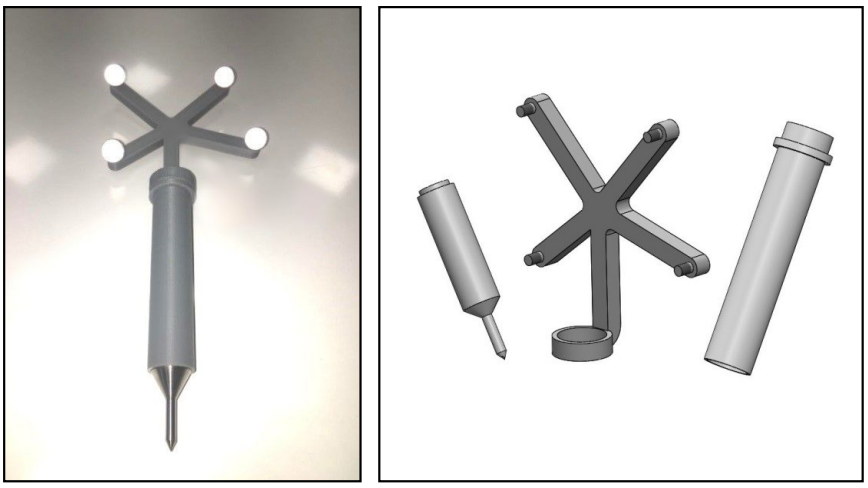

Figure 6.22. First prototype of Pointer Tool and Separate Parts.

However this design suffered from unwanted deformations and joint gaps. These facts undermined rigidity of the pointer tool during measurement and navigation. At this point a new design is utilized to keep pointer tool more resistant to deflections caused by user and reduce the distance between rigid body marker and pointer tip. Keeping this distance short ensures positional error of pointer tip caused by orientation noise of the rigid body marker will be lower. New modified design can be seen in figure 6.23. Measurement tip holder and rigid body marker are merged into a single part with objective of also supporting rigid body marker with the shaft of precise measurement tip.

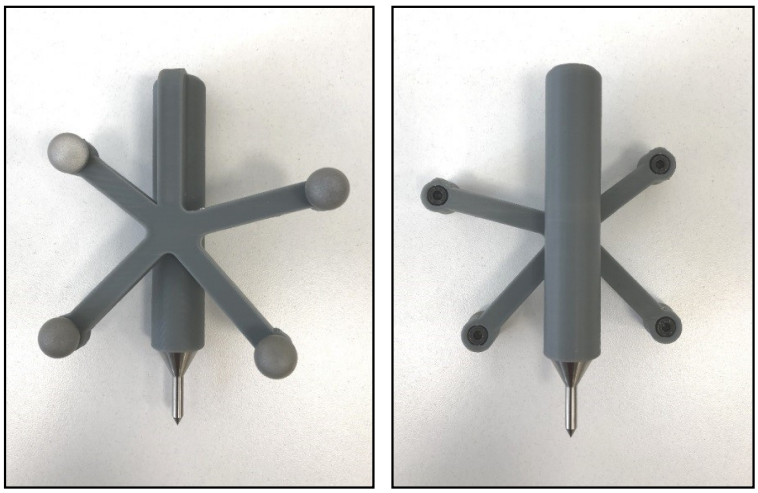

Figure 6.23. Updated Design of Pointer Tool.

6.5 Calibration and Verification Studies

Firstly, spin calibration was performed to start tracking system verification methodology of proposed procedure after completing design and manufacturing of required tools. The most important consideration at this point is the fact that since the pointer tool is rigid, performing pointer tool calibration once is enough as long as no shape modification performed on pointer tool or rigid body marker of pointer tool is not redefined in tracking system software. ${}^{I}\boldsymbol{\delta}_{U}$ translation vector and ${}^{I}_{U}\boldsymbol{R}$ rotation matrix will still be valid after any change on workspace. First calibration of pointer tool should be performed on a tracking system with known validity. Accordingly, Optitrack V100R2 motion capture cameras are calibrated with official software before the procedure and ensured constancy of positions of motion capture cameras during procedure.

Spin calibration is performed using KUKA KR6 R900 SIXX industrial robot considering difficulty of user performing spin calibration of pointer tool while manually keeping spin axis stable by hand. By contrast pivot calibration only requires stability of single point (pointer tip point) during motion and accordingly can be more easily performed by hand. A fixture part is designed to match spin axis of pointer tool with the joint axis of the last joint of industrial manipulator. After the assembly, spin calibration is completed by only rotating related axis (Figure 6.24).

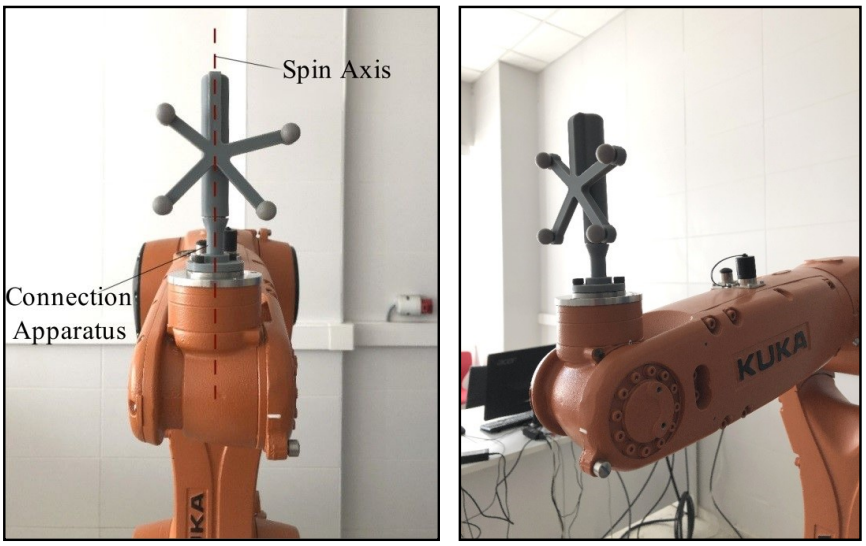

Figure 6.24. Pointer Tool and Macro Manipulator Spin Calibration Assembly.

Result of spin calibrations performed by the manipulator and the hand manually for 10 tries are shown in table 6.4. As can be seen in table, tries with lowest root mean square (RMS) values were extracted by unit spin axis calculation of SlicerIGT software was obtained by utilizing the manipulator.

Table 6.4. Marker Rotation Calibration Results.

Test Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Mean**
Manuel Calibration RMS Value (10^{-3} **mm)**	4.21	1.94	4.03	4.05	4.76	3.75	2.63	3.57	2.44	3.13	3.45
Manipulator Assisted Calibration RMS Value (10^{-3} **mm)**	2.26	2.37	2.12	2.69	2.61	2.77	2.39	2.26	2.35	2.80	2.46

As mentioned before, Pointer tool should perform a spherical movement while keeping pointer tip as center and fixed point during pivot calibration. Executing this movement with a manipulator requires knowledge of position of pointer tip in connection fixture and ability to describe distance of this point to isopoint of the manipulator. Since designed prototype is manufactured by using rapid prototyping, the uncertainties of dimensions and distances are inevitably effect calibration. Due to those reasons pivot calibration is decided to be performed by hand by the help of a manufactured basic apparatus with a stitch to block displacement of pointer tip (Figure 6.25).

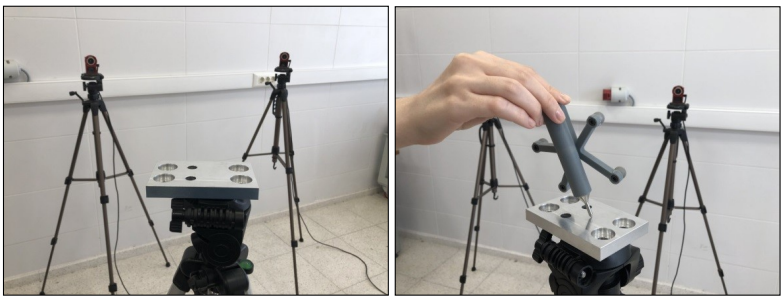

Figure 6.25. Apparatus Manufactured for Pivot Calibration and Process of Calibration.

Results of 10 trials of pivot calibration were shown in table 6.5. As can be seen in related table, low values of root mean square (RMS) errors indicate that pivot calibration of pointer tool is successfully performed.

Table 6.5. Marker Pivot Calibration Results.

Test Number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**Mean**
Manuel Calibration RMS Value (10^{-3} mm)	72.00	78.05	68.54	60.50	92.59	79.08	90.43	94.33	79.31	64.42	77.92

${}^{I}_{U}\boldsymbol{T}$ transformation matrix was obtained by pointer tool calibrations and next stage, verification of tracking system calibration, was commenced. At this point, without interrupting present calibration of tracking system, mock-up model was placed in measurement space and all measurements were taken from marks of 0.025 mm depth on geometrical middle points of top surfaces of towers, which are determined to be fiducial points (Figure 6.26).

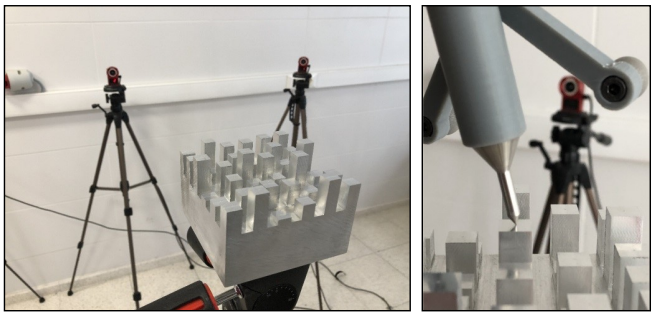

Figure 6.26. Placement of Mock-up Model and Process of Measurement.

During procedure, fiducial points were measured by collecting 100 position samples from each point and averaging the samples. This was done to reduce system uncertainties in calculating fiducial registration error (FRE) and target registration error (TRE) values. A program was written using Python programming language [Appendix A] to perform measurement/verification stages and related algorithm is shown in figure 6.27.

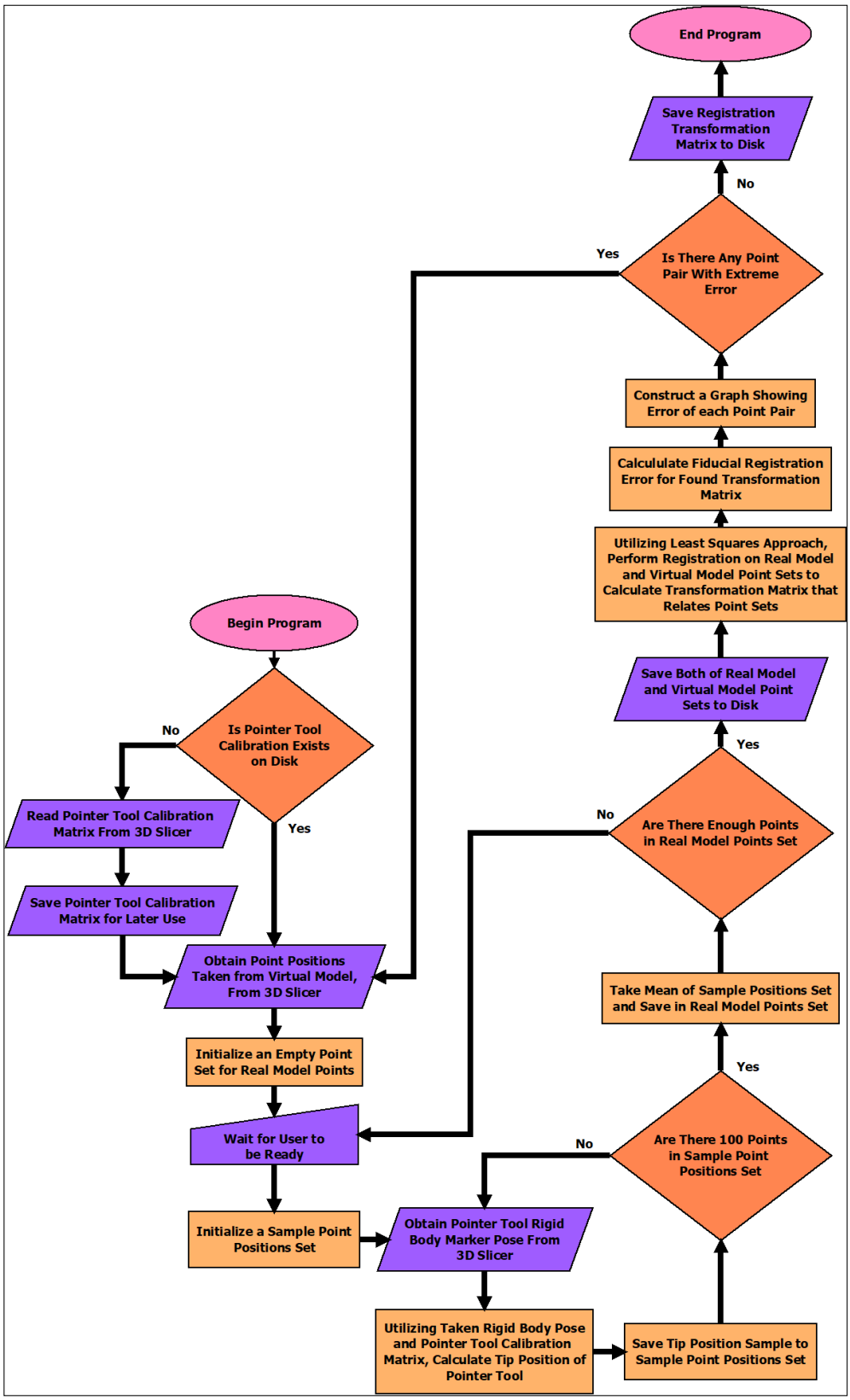

Figure 6.27. Algorithm of Software Created for Measurement and Verification Studies.

In first phase of the study, mock-up model was assumed to be fixed in position and orientation and fiducial point positions on mock-up model are measured with respect to tracking system reference. All measured fiducial point positions (${}^{K}\boldsymbol{\rho}_{U}^{i} \quad i = 1, 2, 3, \ldots, 36$) were utilized together with corresponding and precisely known fiducial point positions with respect to mock-up model reference (${}^{M}\boldsymbol{\rho}_{U}^{i} \quad i = 1, 2, 3, \ldots, 36$) to perform registration with least squares methodology described in section 5.3 and ${}^{M}_{K}\boldsymbol{T}$ transformation matrix that gives least square error was obtained. Thus related fiducial registration error (FRE) was calculated. Afterwards same procedure was repeated with only random half of the measured fiducial point positions and ${}^{M}_{K}\boldsymbol{T}$ transformation matrix was recalculated. Unused fidual point's positions with respect to model reference (${}^{M}\boldsymbol{\rho}_{U}^{j} \quad j = 1, 2, 3, \ldots, 18$) were calculated utilizing equation 6.7.

$$\begin{bmatrix} {}^{M}\boldsymbol{\rho}_{U}^{j} \\ 1 \end{bmatrix} = {}^{M}_{K}\boldsymbol{T} \begin{bmatrix} {}^{K}\boldsymbol{\rho}_{U}^{j} \\ 1 \end{bmatrix} \tag{6.7}$$

Results obtained from equation 6.7 were compared with corresponding known fiducial point positions with respect to model reference and from the distance between each pair target registration error (TRE) was calculated. Results of performed verification were shown in table 6.6 for five tries.

Table 6.6. Results of Verification Study (Fixed Mock-up Model).

Test Number		**1**	**2**	**3**	**4**	**5**
36 Fiducial Points FRE = TRE (10^{-3} **mm**)		200.63	209.25	230.28	236.64	236.27
A	**18 Fiducial Points FRE** (10^{-3} **mm**)	206.81	203.69	243.57	178.01	207.88
	18 Fiducial Points TRE (10^{-3} **mm**)	201.89	220.83	218.78	304.72	274.75
	36 Fiducial Points TRE (10^{-3} **mm**)	204.37	212.43	231.51	249.54	243.62
B	**18 Fiducial Points FRE** (10^{-3} **mm**)	202.49	210.75	244.52	204.98	227.00
	18 Fiducial Points TRE (10^{-3} **mm**)	210.31	211.48	224.10	268.48	254.42
	36 Fiducial Points TRE (10^{-3} **mm**)	206.44	211.11	234.53	238.85	241.10
C	**18 Fiducial Points FRE** (10^{-3} **mm**)	158.27	199.50	205.12	173.83	233.12
	18 Fiducial Points TRE (10^{-3} **mm**)	257.74	232.69	266.51	297.80	250.63
	36 Fiducial Points TRE (10^{-3} **mm**)	213.87	216.73	237.80	243.82	242.03

As can be seen in table, acceptable low values of calculated errors indicates tracking system calibration is sufficient. Also three different random fiducial point selections (Experiment A, B, C) is shown in table 6.6. Although fiducial registration error (FRE) and target registration error (TRE) calculated with the half of fiducial points can be lower then overall fiducial registration error (FRE) of all 36 fiducial points, target registration error (TRE) calculated with all 36 fiducial points is always higher than overall fiducial registration error (FRE). This fact indicates validity of verification study.

In second stage of study, mock-up rigid body marker was deployed to take measurements on mobile mock-up model (Figure 6.28). Following a similar procedure, fiducial point positions measured with respect to mock-up rigid body marker reference (${}^{Mi}\boldsymbol{\rho}_U^k \quad k = 1, 2, 3, \ldots, 36$) were utilized together with precisely known fiducial point positions (${}^{M}\boldsymbol{\rho}_U^k \quad k = 1, 2, 3, \ldots, 36$) to calculate ${}_K^M\boldsymbol{T}$ transformation matrix that gives least square error using registration methodology. Thus fiducial registration error (FRE) is calculated.

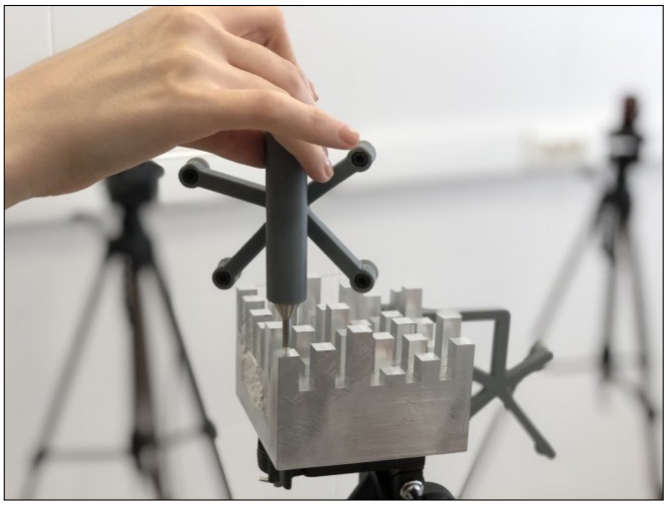

Figure 6.28. Taking Measurement on Mobile Mock-up Model.

Shown results for five trials in table 6.7, when analyzed, show consistency with previous experiment.

Table 6.7. Results of Verification Study (Mobile Mock-up Model).

Test Number		**1**	**2**	**3**	**4**	**5**
36 Fiducial Points FRE = TRE (10^{-3} **mm)**		295.44	255.84	232.30	307.73	248.30
A	**18 Fiducial Points FRE** (10^{-3} **mm)**	299.94	225.23	215.95	299.88	233.47
	18 Fiducial Points TRE (10^{-3} **mm)**	297.33	293.06	275.45	324.04	273.21
	36 Fiducial Points TRE (10^{-3} **mm)**	298.64	261.36	247.49	312.19	254.12
B	**18 Fiducial Points FRE** (10^{-3} **mm)**	280.06	248.97	249.90	267.09	235.75
	18 Fiducial Points TRE (10^{-3} **mm)**	317.09	279.89	216.14	359.89	270.04
	36 Fiducial Points TRE (10^{-3} **mm)**	299.15	264.88	233.63	316.91	253.48
C	**18 Fiducial Points FRE** (10^{-3} **mm)**	321.97	216.44	196.77	318.37	261.53
	18 Fiducial Points TRE (10^{-3} **mm)**	271.90	309.08	268.80	307.21	245.91
	36 Fiducial Points TRE (10^{-3} **mm)**	297.99	266.81	235.55	312.84	253.84

Related procedure was repeated on fixed mock-up model in a environment with intentionally broken tracking system calibration to prove validity of suggested calibration verification procedure. Results were shown in table 6.8 for five trials.

Table 6.8. Results of Verification Study (Broken Tracking System Calibration).

Test Number		1	2	3	4	5
36 Fiducial Points FRE = TRE (10^{-3} mm)		2845.4	2750.7	2786.6	2303.5	2175.1
A	**18 Fiducial Points FRE (10^{-3} mm)**	3348.2	2752.2	3292.8	2703.1	2258.2
	18 Fiducial Points TRE (10^{-3} mm)	2621.2	2784.6	2374.5	1913.7	2110.3
	36 Fiducial Points TRE (10^{-3} mm)	3006.7	2768.5	2870.6	2341.9	2185.5
B	**18 Fiducial Points FRE (10^{-3} mm)**	3311.8	2491.6	1987.5	2519.3	2247.4
	18 Fiducial Points TRE (10^{-3} mm)	2561.3	3052.4	3451.0	2159.5	2132.8
	36 Fiducial Points TRE (10^{-3} mm)	2960.4	2786.2	2816.0	2346.3	2190.9
C	**18 Fiducial Points FRE (10^{-3} mm)**	1947.6	2058.1	1876.3	1830.0	2220.6
	18 Fiducial Points TRE (10^{-3} mm)	3583.7	3444.4	3542.4	2870.3	2150.2
	36 Fiducial Points TRE (10^{-3} mm)	2884.1	2837.2	2834.5	2407.0	2185.7

Inspecting values shown in related table, it can be seen that substantially higher error values were obtained from the environment with broken tracking system calibration. In turn, this fact supports validity of verification procedure.

7. REGISTRATION BETWEEN TRACKING SYSTEM MEASUREMENT SPACE AND MACRO MANIPULATOR WORKSPACE

In scope of this thesis, six degrees of freedom KUKA KR6 R900 SIXX serial robot manipulator was deployed in surgical navigation study. ${}^{K}_{R}\boldsymbol{T}$ transformation matrix that relates manipulator reference (R) and tracking system reference (K) should be calculated for successful utilization of serial robot manipulator (Figure 7.1).

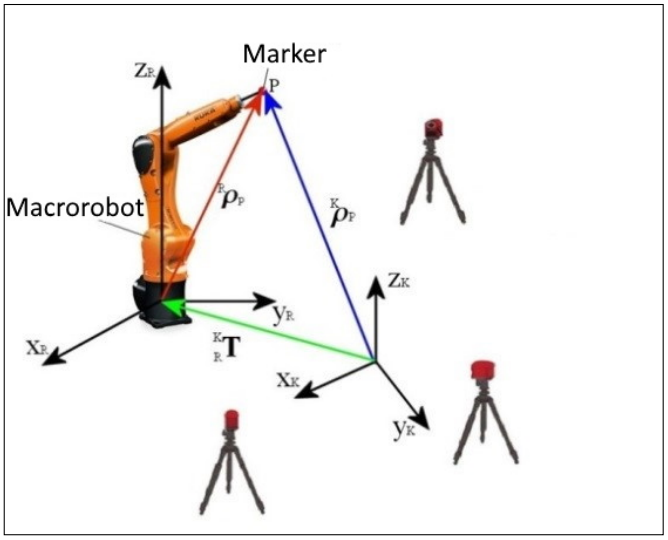

Figure 7.1. Robot Manipulator and Tracking System References.

In this context least squares methodology (theoretical background can be seen in section 5.3) was decided to be utilized.

Related procedure requires point position sets described with respect to both references ${}^{K}\boldsymbol{\rho}_{Pi}$ and ${}^{R}\boldsymbol{\rho}_{Pi}$ $\quad i = 1, 2, 3, \ldots, n$ to be constructed. With this purpose in mind an apparatus, containing infrared reflective sphere on tip point, was designed to be assembled on the last joint axis of serial robot manipulator. Position of the infrared reflective sphere on apparatus then can be tracked by tracking system (Figure 7.2).

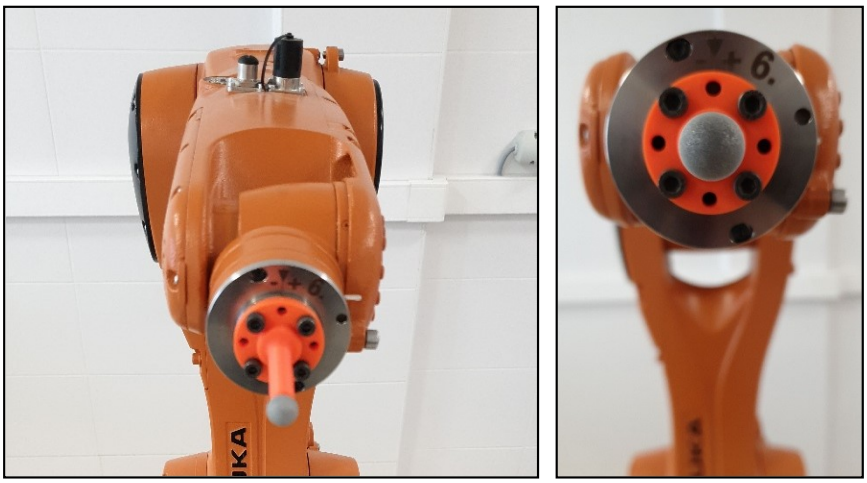

Figure 7.2. Robot Manipulator with Designed Apparatus.

In constructing point position sets, steps described below were followed.

- Moving apparatus tip point to an arbitrary position with respect to manipulator reference.
- After movement is completed, measuring position of infrared reflective sphere with motion capture cameras with respect to tracking system reference

Official controller of serial robot manipulator (KRC4) was utilized to give movement to the apparatus tip in order to position it to an arbitrary point with respect to manipulator reference. Inverse and forward kinematic solutions of 6 degree of freedom robot manipulator are embedded inside official KUKA system software. Thus forward kinematic solution of end effector pose with respect to manipulator reference can be read after system actuators are driven via controller. In addition, it is sufficient to send the end effector pose to the system software as an alternative. Thus, the necessary actuator angles will be calculated by the official system software using integrated inverse kinematic solution and manipulator movement will be realized.

In the first trial of the study, robot axes was moved manually by the help of the controller to random positions in workspace. Related positions of the infrared reflective sphere at the tip of apparatus was obtained with respect to both references by the help of Motive Tracker and KUKA system software. However this method was proved to be infeasible due to reasons of being slow and not suitable for collection of a substantial number of points, and difficulty of constraining taken positions in a workspace. Later, it was decided to take point sets from a cuboid volume that encompasses head volume of an adult person and inside space that is accessible to manipulator (Figure 7.3).

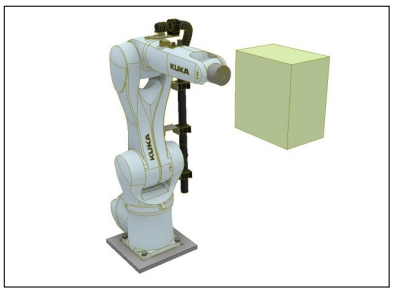

Figure 7.3. Volume Created for Determining Point Position Set.

As shown in figure 7.3, cuboid volume with dimensions of $219.2 \times 144.8 \times 232$ mm is positioned in workspace of manipulator. For positioning apparatus tip, a $12 \times 12 \times 12$ three dimensional cubic grid of points, with a total of 1728 points, were fitted inside the cuboid volume with a homogeneous distribution. Total volume then divided into 36 sub-volumes and inside each sub-volume, order of apparatus tip arrivals at the points were randomized to avoid any bias that might be caused by the sequence during point collection. Also, by randomizing point orders within sub-volumes instead of whole volume, average distance traveled between two points by apparatus tip was shortened and in turn average time required for each position measurement was reduced. Path followed by apparatus tip is shown in figure 7.4, path inside each sub-volume is shown with different color.

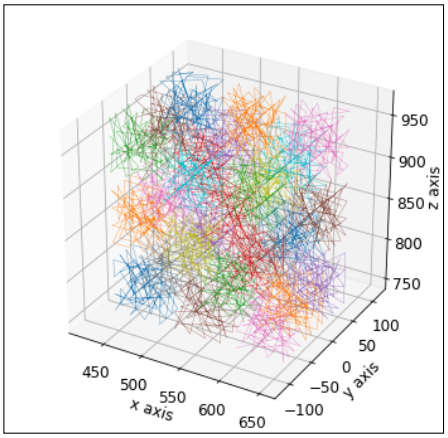

Figure 7.4. Path Followed by Manipulator with Apparatus During Measurement of Points.

Similar to study on mock-up model, 100 position samples were collected by tracking system at each point. In ${}^{K}_{R}\boldsymbol{T}$ transformation matrix calculation procedure, related point's position with respect to tracking system reference is defined as average of these position samples. The procedure calculating relation between manipulator reference (R) and calibrated tracking system reference (K) was implemented as a software (Ap-

pendix B) utilizing open-source robotics framework Robot Operating System (ROS) and performed automatically by implemented software. Required communication between software in process of sending target positions to KUKA system software was constructed by utilization of open-source KukaVarProxy [15] software.

After $\{{}^{K}\boldsymbol{\rho}_{Pi}\}$ and $\{{}^{R}\boldsymbol{\rho}_{Pi}\}$ $i = 1, 2, 3, \ldots, 1728$ point position sets were constructed, utilizing equations 5.10 - 5.15, ${}^{K}_{R}\boldsymbol{T}$ transformation matrix that gives least square error was calculated. Since position uncertainty of utilized manipulator (± 0.03 mm) is much lower than measurement uncertainty of tracking system (± 0.3 mm), it was assumed that manipulator is moved to defined point without error. Errors (e_i) were calculated between each point position pair in sets of point positions known with respect to manipulator reference (${}^{R}\boldsymbol{\rho}_{Pi}$) and point positions that were measured with respect to tracking reference and transformed to manipulator reference with ${}^{K}_{R}\boldsymbol{T}$ transformation matrix (${}^{K*}\boldsymbol{\rho}_{Pi}$) (Equation 7.1)

$$\begin{bmatrix} {}^{K*}\boldsymbol{\rho}_{Pi} \\ 1 \end{bmatrix} = {}^{K}_{R}\boldsymbol{T} \begin{bmatrix} {}^{R}\boldsymbol{\rho}_{Pi} \\ 1 \end{bmatrix} \qquad (7.1)$$
$$e_i = |{}^{K}\boldsymbol{\rho}_{Pi}| - |{}^{K*}\boldsymbol{\rho}_{Pi}|$$

Gradient of errors (mm) at each point can be clearly seen in figure 7.5.

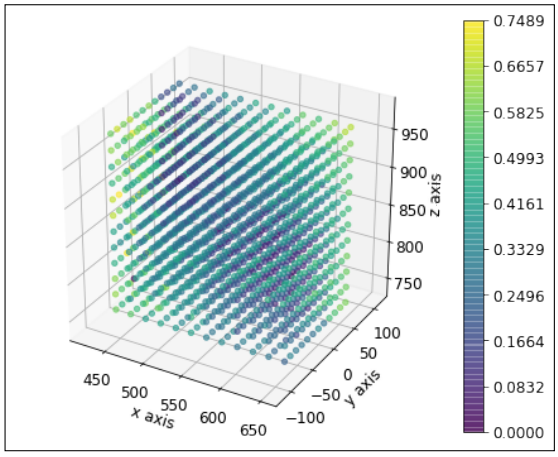

Figure 7.5. Gradient of Error at Points of Point Position Set.

Fiducial registration error (FRE), which is equal to mean root square of errors of positions calculated using ${}^{K}_{R}\boldsymbol{T}$ transformation matrix, was calculated to be 0.350 mm. This indicates feasibility of surgical navigation with possessed hardware and within defined volume.

7.1 Compensation for Possible Apparatus Length Uncertainty

In section 7, apparatus length between robot manipulator end effector base and centroid of sphere apparatus was manually measured with a precise caliper ruler and each position in $\{{}^{K}\boldsymbol{\rho}_{Pi}\}$ $i = 1, 2, 3, \ldots, 1728$ point position set were visited utilizing this measured length as an offset in a fixed orientation. $\{{}^{R}\boldsymbol{\rho}_{Pi}\}$ $i = 1, 2, 3, \ldots, 1728$ point position set constructed with this procedure will valid as long as centroid position of sphere apparatus is perfectly known. In case of this measurement being imperfect, $\{{}^{R}\boldsymbol{\rho}_{Pi}\}$ point position set will drift in direction of fixed orientation set in position measurement procedure. This possibility of error should be considered.

Like in section 7, a $6 \times 6 \times 6$ three dimensional cubic grid of points, with a total of 216 points, were fitted inside of cuboid volume, that encompasses head volume of an adult person and inside space that is accessible to manipulator, with a homogeneous distribution. Total volume then divided into 8 sub-volumes and inside each sub-volume, order of apparatus tip arrivals at the points are randomized. As key consideration, each position in $\{{}^{R}\boldsymbol{\rho}_{Pi}\}$ $i = 1, 2, 3, \ldots, 216$ point position set was also given a set of Euler angles for each point as $\begin{bmatrix} \pm\frac{\pi}{36} & \pm\frac{\pi}{36} & \pm\frac{\pi}{36} \end{bmatrix}$ in radians. Then 100 position samples were collected by tracking system at each point. Related point position with respect to tracking system reference was defined as average of these position samples. After related position measurements were completed, utilizing equations 5.10 - 5.15, ${}^{K}_{R}\boldsymbol{T}$ transformation matrix that gives least square error was calculated.

Errors (e_i) were calculated utilizing equation 7.1. Gradient of errors (mm) at each point was constructed and is shown in figure 7.6

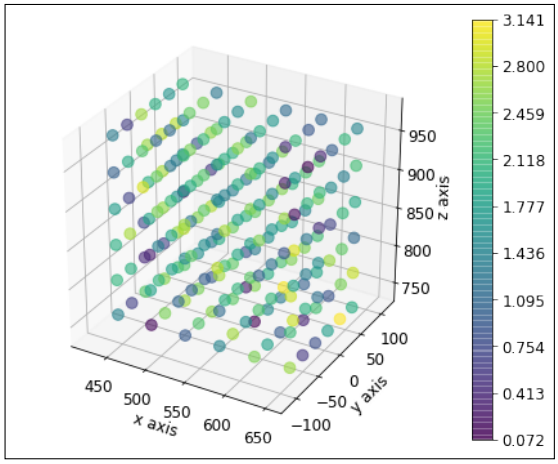

Figure 7.6. Gradient of Error at Points of Point Position Set with Euler Angles.

Fiducial registration error (FRE) was calculated to be 1.945 mm, which is considerably higher than fiducial point error found in section 7. Reason of this is the fact that Euler angles given to each point broke homogeneity of error caused by fixed angles which in turn caused drift in measured positions.

The fact of fiducial registration error (FRE) was found to be considerably higher points to possibly of there was some uncertainty in manual apparatus length measurement. In response to this fact, an algorithm was developed and implemented as a Python program to compensate this uncertainty [Appendix C]. The program written can emulate different apparatus lengths and perform error calculation as each Euler angle given to each point position is known. The algorithm of the written program can be given as,

- Emulate each apparatus length and calculate fiducial registration error (FRE) from a given range,
- Fit a polynomial to fiducial registration error (FRE) with respect to emulated apparatus length data,
- Calculate emulated apparatus length with minumum fiducial registration error (FRE) as effective length of apparatus from fitted polynomial.

Graph of raw data is given in figure 7.7,

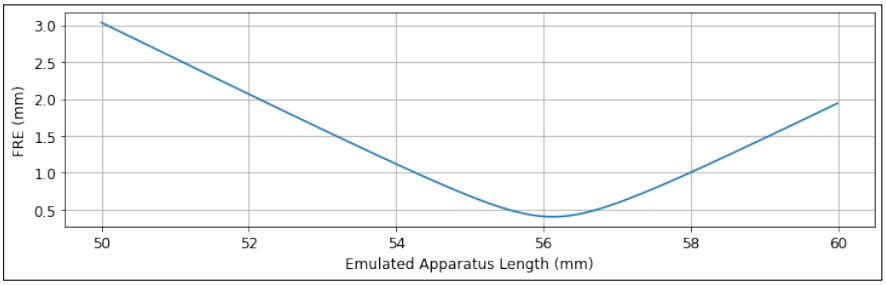

Figure 7.7. Fiducial Registration Error (FRE) with Respect to Emulated Apparatus Length.

Sixth order polynomial $2.628 \times 10^{-5}x^6 - 9.151 \times 10^{-3}x^5 + 1.321x^4 - 1.013 \times 10^2x^3 + 4.354 \times 10^3x^2 - 9.941 \times 10^4x + 9.426 \times 10^5$ was fitted to raw data. Graph of raw data overlapped by fitted polynomial is given in figure 7.7,

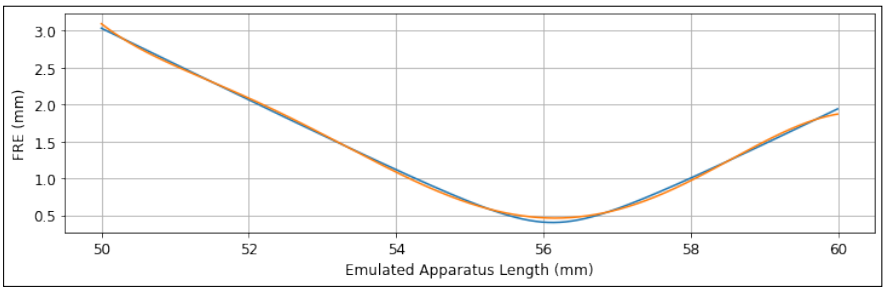

Figure 7.8. Fiducial Registration Error (FRE) with Respect to Emulated Apparatus Length Overlapped by Fitted Polynomial.

At this point, effective length of apparatus was found to be 56.156 mm from roots of the polynomial. Gradient of error shown in figure 7.9 is found by inserting effective length to gradient of error calculation procedure.

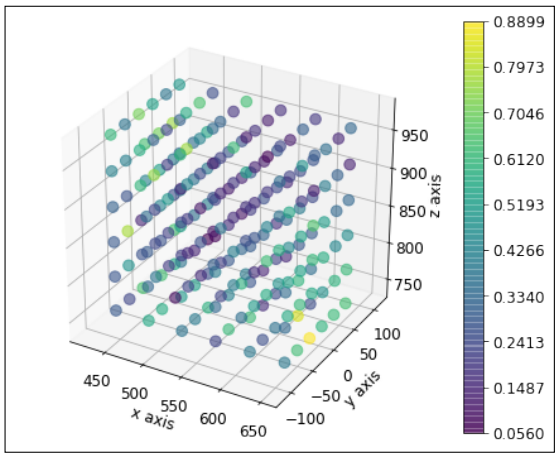

Figure 7.9. Gradient of Error at Points of Point Position Set.

Fiducial registration error (FRE), calculated with effective length, was found to be 0.403 mm, which is comparable to the fiducial registration error (FRE) found in section 7. Thus the method proposed for the compensation of length uncertainty that may occur during tool tip measurements was proved to be effective.

8. HARDWARE VERIFICATION

At this point, studies done in previous sections were utilized in performing navigation on mock-up model with robot manipulator. Firstly, a part to be assembled to the end effector of robot manipulator was designed (figure 8.1). Main criteria considered in this design is safety. Related part was designed to be easily breakable in situations of undesirably high forces that might be applied by robot manipulator to mock-up model due to few millimeters of navigation error.

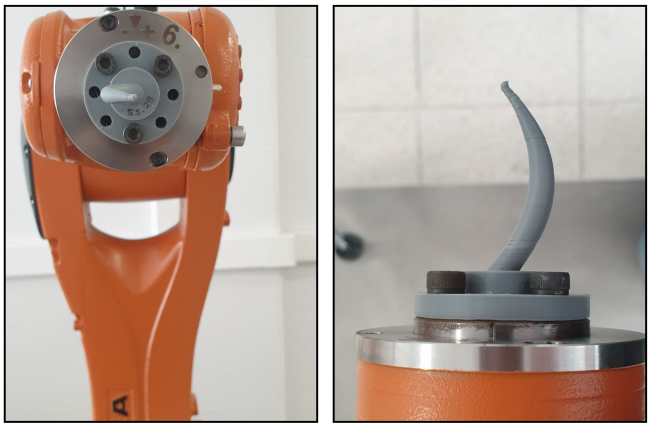

Figure 8.1. Robot Manipulator with Designed Part.

Navigation procedure was implemented as a Robot Operating System (ROS) node in Python programming language [Appendix D], interfacing with 3D Slicer (via ros_igtl_bridge [16]), Optitrack Motive Tracker (via Plus Toolkit, ros_igtl_bridge) and KUKA KR6 R900 SIXX industrial robot (via KukaROSOpenComm [17]) (Figure 8.2).

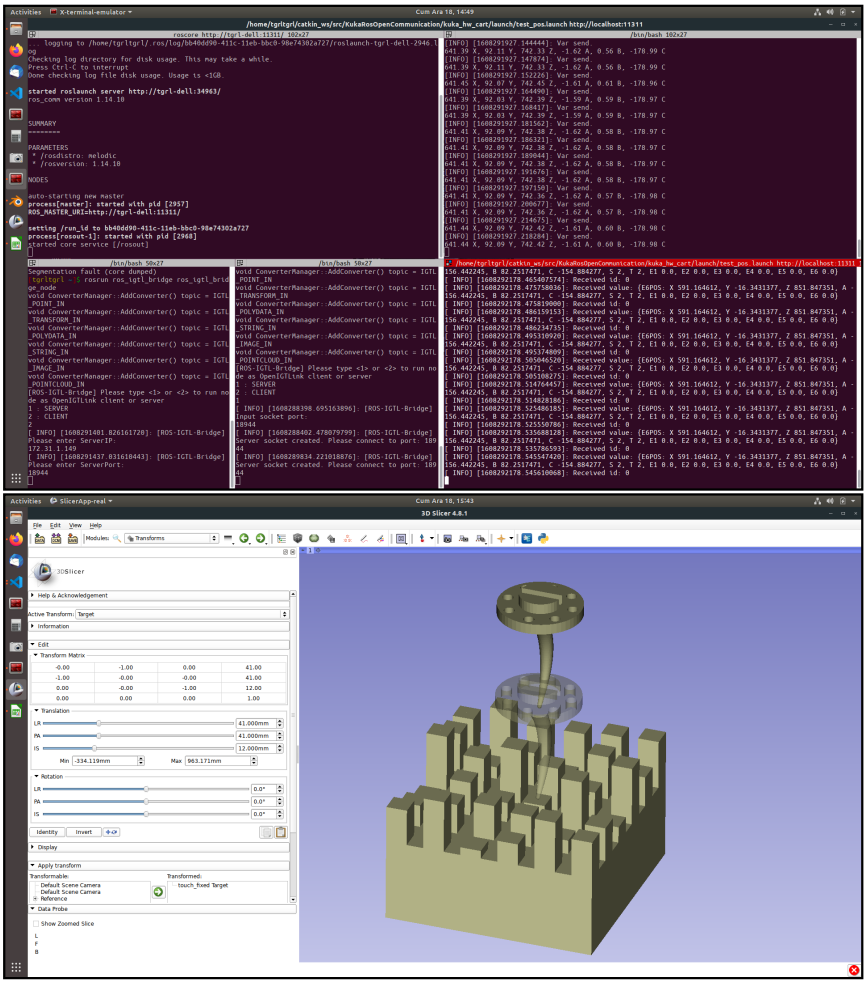

Figure 8.2. ROS and 3D Slicer Interface.

Written ROS node utilizes calibration data calculated in sections 6.3, 6.5 and 7.1 as it streams robot manipulator end effector position and orientation to 3D Slicer in real-time. Opaque virtual model of the end effector represents current position and orientation of robot end effector. Path planning was done by operator with 3D Slicer software. Target orientation and position of robot end effector was visualized as half transparent visual model of robot end effector. Operator is able to manipulate target position and orientation with respect to mock-up model reference as desired (Figure 8.3).

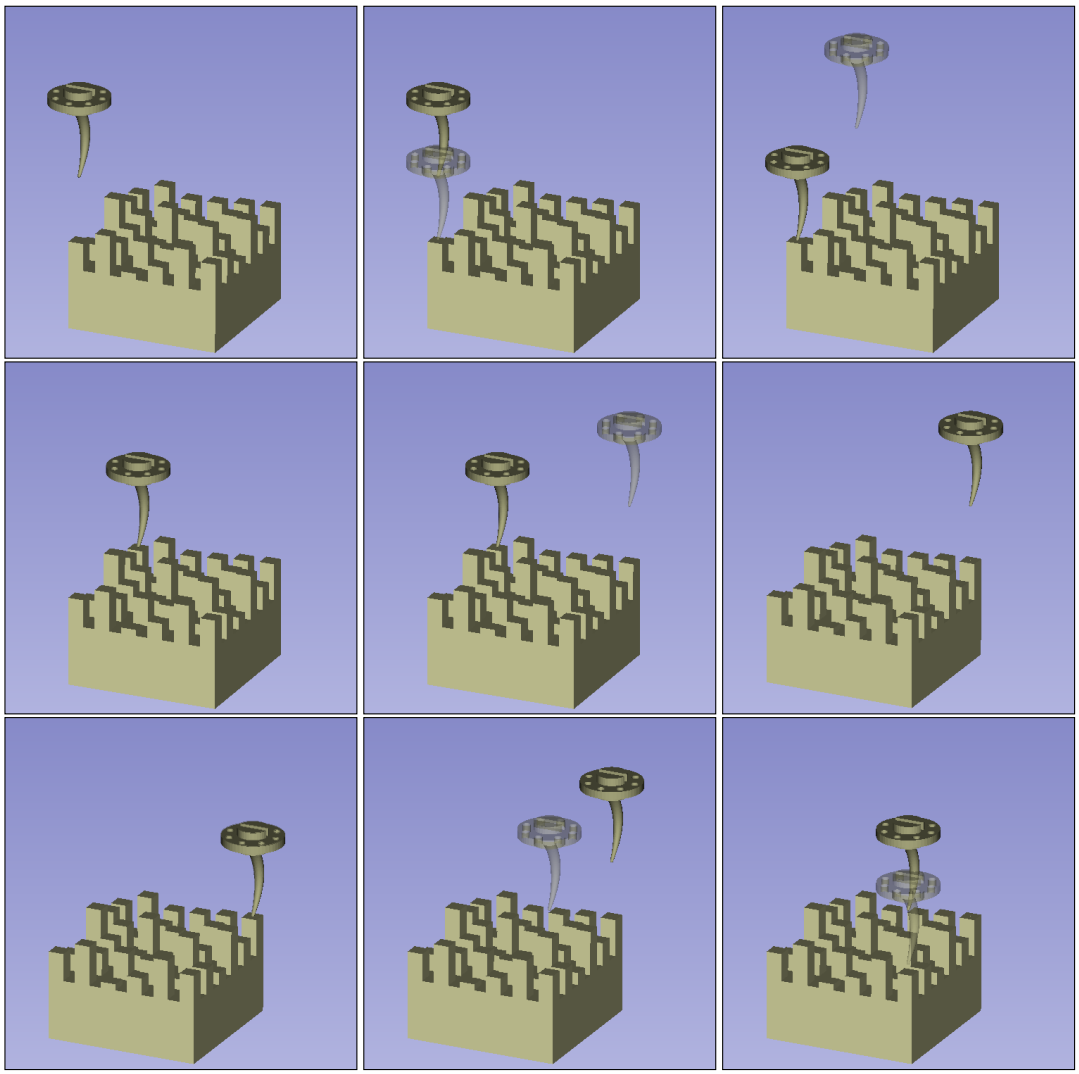

Figure 8.3. Visuals From Virtual Environment at Some Key Moments.

Target position and orientation data is continuously being streamed to written ROS node, in turn written ROS node sends these positions and orientations to robot manipulator to be executed as movements. In figure 8.4, each row show execution of movement to corner fiducial point positions of mock-up model. Result of each movement is shown as close-up images in last column. As can be seen in figure 8.4, results of hardware verification procedure were promising.

Figure 8.4. Corresponding Movements of Robot Manipulator.

9. CONCLUSIONS

Throughout the thesis, a low cost motion tracking system was utilized together with free and open source software to create a surgical navigation system which was aimed to be effective in robotic surgical operations especially targeted for cochlear microrobot operations. Facts like,

- low error values found at each step,
- ability to stream position and orientation data of tools and workspace in virtual space,
- successful integration of a macro manipulator to the system,
- successful path planning performed in 3d Slicer,
- and also successful execution of this path my macro manipulator

indicates that designed surgical navigation system gives promising results to be used in cochlear microrobot operations. It is clear that low error values in tables 6.4-6.7 suggests successful design and manufacturing of mock-up model and pointer tools. This fact already set validity of verification methodology described in section 6.

Also as seen in the close-up images of the final hardware verification procedure from figure 8.4, it is clear that operator was able to move end effector of robot manipulator to desired positions successfully. This fact clearly points to the success of finding required relation between robot manipulator reference (R) and tracking system reference (K), that mapped motion capturing workspace to robot manipulator workspace; pointer tool reference (I) and pointer tool rigid body marker reference (U), which effects fiducial point position measurement with respect to tracking system reference (K) and directly calibration of patient (in this case mock-up model); mock-up rigid body marker reference (Mi) and mock-up model reference (M), which described fiducial point positions with respect to tracking system reference (K).

In future works, described and verified surgical navigation methodology will be applied to a constructed cochlea model and a graphical user interface will be designed for better usability of the proposed navigation system. Also an end-effector for robot arm might be designed for measuring target registration error values during surgical navigation application shown in figures 8.3 and 8.4. These error values might include

error from geometrical defects of model which was assumed to be non-existent during this study.

REFERENCES

1. Mezger U, Jendrewski C, and Bartels M. Navigation in surgery. Langenbeck's Archives of Surgery 2013 Feb; 398:501–14. DOI: 10.1007/s00423-013-1059-4

2. Arun KS, Huang TS, and Blostein SD. Least-Squares Fitting of Two 3-D Point Sets. IEEE Transactions on Pattern Analysis and Machine Intelligence 1987 Sep; PAMI-9:698–700. DOI: 10.1109/tpami.1987.4767965

3. Cho B, Oka M, Matsumoto N, Ouchida R, Hong J, and Hashizume M. Warning navigation system using real-time safe region monitoring for otologic surgery. International Journal of Computer Assisted Radiology and Surgery 2012 Nov; 8:395–405. DOI: 10.1007/s11548-012-0797-z

4. Jeon S, Park J, Chien J, and Hong J. A hybrid method to improve target registration accuracy in surgical navigation. Minimally Invasive Therapy & Allied Technologies 2015 Mar; 24:356–63. DOI: 10.3109/13645706.2015.1020555

5. Hong J, Matsumoto N, Ouchida R, Komune S, and Hashizume M. Medical Navigation System for Otologic Surgery Based on Hybrid Registration and Virtual Intraoperative Computed Tomography. IEEE Transactions on Biomedical Engineering 2009 Feb; 56:426–32. DOI: 10.1109/tbme.2008.2008168

6. Tomikawa M, Hong J, Shiotani S, Tokunaga E, Konishi K, Ieiri S, Tanoue K, Akahoshi T, Maehara Y, and Hashizume M. Real-Time 3-Dimensional Virtual Reality Navigation System with Open MRI for Breast-Conserving Surgery. Journal of the American College of Surgeons 2010 Jun; 210:927–33. DOI: 10.1016/j.jamcollsurg.2010.01.032

7. Cho HS, Park YK, Gupta S, Yoon C, Han I, Kim H-S, Choi H, and Hong J. Augmented reality in bone tumour resection. Bone & Joint Research 2017 Mar; 6:137–43. DOI: 10.1302/2046-3758.63.bjr-2016-0289.r1

8. Fitzpatrick JM, West JB, and Maurer CR. Derivation of expected registration error for point-based rigid-body registration. *Medical Imaging 1998: Image Processing*. Ed. by Hanson KM. SPIE, 1998 Jun. DOI: 10.1117/12.310824

9. Fitzpatrick JM, West JB, and Maurer CR. Predicting error in rigid-body point-based registration. IEEE Transactions on Medical Imaging 1998; 17:694–702. DOI: 10.1109/42.736021

10. Fedorov A, Beichel R, Kalpathy-Cramer J, Finet J, Fillion-Robin J-C, Pujol S, Bauer C, Jennings D, Fennessy F, Sonka M, Buatti J, Aylward S, Miller JV, Pieper S, and Kikinis R. 3D Slicer as an image computing platform for the Quantitative Imaging Network. Magnetic Resonance Imaging 2012 Nov; 30:1323–41. DOI: 10.1016/j.mri.2012.05.001

11. Ungi T, Lasso A, and Fichtinger G. Open-source platforms for navigated image-guided interventions. Medical Image Analysis 2016 Oct; 33:181–6. DOI: 10.1016/j.media.2016.06.011

12. Lasso A, Heffter T, Rankin A, Pinter C, Ungi T, and Fichtinger G. PLUS: Open-Source Toolkit for Ultrasound-Guided Intervention Systems. IEEE Transactions on Biomedical Engineering 2014 Oct; 61:2527–37. DOI: 10.1109/tbme.2014.2322864

13. Brown A, Uneri A, and Silva TD. Design and validation of an open-source library of dynamic reference frames for research and education in optical tracking. Journal of Medical Imaging 2018 Feb; 5:1. DOI: 10.1117/1.jmi.5.2.021215

14. Pintaric T and Kaufmann H. A rigid-body target design methodology for optical pose-tracking systems. *Proceedings of the 2008 ACM symposium on Virtual reality software and technology - VRST '08*. ACM Press, 2008. DOI: 10.1145/1450579.1450594

15. IMTS and Peloux L du. KukaVarProxy. Available from: https://github.com/ImtsSrl/KUKAVARPROXY [Accessed on: 2020 Dec 21]

16. Frank T, Krieger A, Leonard S, Patel NA, and Tokuda J. ROS-IGTL-Bridge: an open network interface for image-guided therapy using the ROS environment. International Journal of Computer Assisted Radiology and Surgery 2017 May; 12:1451–60. DOI: 10.1007/s11548-017-1618-1

17. Kahveci A. KukaRosOpenCommunication. Available from: https://github.com/AytacKahveci/KukaRosOpenCommunication [Accessed on: 2020 Dec 21]

APPENDIX A

surgical_navigation_node.py

```
#!/usr/bin/env python
########################################
#The ROS module for surgical navigation.  #
########################################
#Author: Tugrul Uslu                     #
#Email: tugrul.uslu@ikc.edu.tr           #
#       Izmir Katip Celebi University    #
#          Mechanical Engineering        #
########################################

from __future__ import division, absolute_import, print_function, unicode_literals

import threading
import sys
import os

import numpy as np
import rospy # pylint: disable=E0401
#from rospy.numpy_msg import numpy_msg # pylint: disable=E0401
from ros_igtl_bridge.msg import igtltransform, igtlpoint, igtlpointcloud # pylint:
↪  disable=E0401
from geometry_msgs.msg import Point, Transform # pylint: disable=E0401
from surgical_navigation.msg import kuka_var # pylint: disable=E0401

from quaternion import quat2mat, mat2quat
from euler import decompose_affine, compose_affine, inverse_affine, euler2mat,
↪  mat2euler

if sys.version_info.major < 3:
    input = raw_input
    range = xrange

# Constants
REF_TO_TRACKER_NAME = "ReferenceToTracker"
STY_TO_TRACKER_NAME = "StylusToTracker"
REF_CALIB_TRANS_NAME = "RefCalib"
STY_CALIB_TRANS_NAME = "StyCalib"
REF_CALIB_TRANS_FILE_NAME = "reference_calib_trans.csv"
STY_CALIB_TRANS_FILE_NAME = "stylus_calib_trans.csv"
ENV_CALIB_TRANS_FILE_NAME = "environment_calib_trans.csv"

###############
## UTILITIES ##
###############

def tuple2ndarray(point):
    """
    """
    P = np.ones((4,1))
    P[:3,0] = point
    return P
```

```
50
51
52  def ndarray2tuple(P):
53      point = tuple(P[:3,0])
54      return point
55
56
57  def transform2mat_and_trans(transform):
58      """
59      """
60      t = np.array((transform.translation.x,
61                    transform.translation.y, transform.translation.z))
62      q = np.array((transform.rotation.w, transform.rotation.x,
63                    transform.rotation.y, transform.rotation.z))
64      R = quat2mat(q)
65      return R, t
66
67
68  def affine2igtltransform(A, name):
69      R, t = decompose_affine(A)
70      q = mat2quat(R)
71      msg = igtltransform()
72      msg.name = name
73      msg.transform.translation.x, msg.transform.translation.y,
        ↪  msg.transform.translation.z = t
74      msg.transform.rotation.w, msg.transform.rotation.x, msg.transform.rotation.y,
        ↪  msg.transform.rotation.z = q
75      return msg
76
77
78  def convert_point(P, *transforms):
79      """
80      returns;
81          P; Converted point, ndarray shape 4,1
82      """
83      for transform in transforms:
84          P = np.matmul(transform, P)
85      return P
86
87  def filter_mean(point_samples ,n_slice=10):
88      """
89      point_samples; np.array shape 3,i,n
90      """
91      _, i, n = point_samples.shape
92      step = n // n_slice
93      point_samples_f = np.zeros((3, i, n_slice))
94      for j in range(n_slice):
95          temp = point_samples[:,:, j*step:(j+1)*step]
96          temp = np.mean(temp, axis=2)
97          point_samples_f[:,:,j] = temp
98      return point_samples_f
99
100
101 #############
102 ## CLASSES ##
103 #############
104
```

```
105
106 class Surgical_Navigation(object):
107     """
108     """
109     ## Setup ##
110     def __init__(self):
111         """
112         """
113         # Variables to hold transformations
114         self.env_calib_trans = np.loadtxt(ENV_CALIB_TRANS_FILE_NAME)
115         self.ref_calib_trans = np.eye(4)
116         self.sty_calib_trans = np.eye(4)
117         self.sty2tracker = np.eye(4)
118         self.ref2tracker = np.eye(4)
119         # ###
120         # List to hold target points
121         self.targetpoint_lt = list()
122         # ###
123         # Desired end effector transformation
124         self.end_eff_transform = np.eye(4)
125         # ###
126         # Initialize system as uncalibrated
127         self.sys_is_calibrated = False
128         # ###
129         # Initialize ROS
130         rospy.init_node("surgical_navigation_node", anonymous=True)
131         rospy.on_shutdown(self.terminate)  # exit gracefully
132         # ###
133         # Create the Publishers
134         self.pub_kuka = rospy.Publisher("kuka_var_stream", kuka_var, queue_size=10)
135         self.pub_pointcloud = rospy.Publisher("IGTL_POINTCLOUD_OUT", igtlpointcloud,
            ↪  queue_size=10)
136         self.pub_transform = rospy.Publisher("IGTL_TRANSFORM_OUT", igtltransform,
            ↪  queue_size=10)
137         # ###
138         # try loading calibration matrices from disk
139         try:
140             self.ref_calib_trans = np.loadtxt(REF_CALIB_TRANS_FILE_NAME)
141             self.sty_calib_trans = np.loadtxt(STY_CALIB_TRANS_FILE_NAME)
142             rospy.loginfo("Calibration matrices are loaded from memory")
143             # If successfull, system assumed to be calibrated
144             self.sys_is_calibrated = True
145         except IOError as error:
146             # These calib files do not exist so we ask them from the user
147             # If failure, system is assumed to be uncalibrated
148             rospy.loginfo("Calibration files for Reference or Stylus are not found.")
149         # ###
150         # Inform user that setup is done
151         rospy.loginfo("Setup Done.")
152
153     def terminate(self):
154         """
155         Terminate gracefully
156         """
157         pass
158
159     ## Callbacks ##
```

```
160     def transform_callback(self, data):
161         """
162         """
163         R, t = transform2mat_and_trans(data.transform)
164         A = compose_affine(R, t)
165         if not self.sys_is_calibrated:
166             # System is not calibrated
167             # Ask user for calibration matrices
168             print("Please send the calibration matrices.")
169             if data.name == REF_CALIB_TRANS_NAME:
170                 # Save the transform for later use and hold it
171                 self.ref_calib_trans = A
172                 np.savetxt(REF_CALIB_TRANS_FILE_NAME, self.ref_calib_trans)
173                 # Inform user
174                 rospy.loginfo(REF_CALIB_TRANS_NAME+" is saved.")
175             elif data.name == STY_CALIB_TRANS_NAME:
176                 # Save the transform for later use and hold it
177                 self.sty_calib_trans = A
178                 np.savetxt(STY_CALIB_TRANS_FILE_NAME, self.sty_calib_trans)
179                 # Inform user
180                 rospy.loginfo(STY_CALIB_TRANS_NAME+" is saved.")
181             else:
182                 rospy.logwarn(
183                     """Please only send the calibration transforms.
184                     This warning might be caused by wrong naming of
    ↪  transformations."""
185                     )
186             if not ((np.allclose(self.ref_calib_trans, np.eye(4))) or
                ↪  (np.allclose(self.sty_calib_trans, np.eye(4)))):
187                 # Since both calib matrice holder variables are not none system
                    ↪  assumed to be calibrated
188                 self.sys_is_calibrated = True
189                 # Inform user
190                 rospy.loginfo("Calibration is completed.")
191         else:
192             # System is calibrated
193             if data.name == REF_TO_TRACKER_NAME:
194                 self.ref2tracker = A
195             elif data.name == STY_TO_TRACKER_NAME:
196                 self.sty2tracker = A
197             else:
198                 rospy.logwarn(
199                     """Please only send the Stylus's or Reference's transforms.
200                     This warnign might be caused by wrong naming of
    ↪  transformations."""
201                     )
202
203     def point_callback(self, data):
204         """
205         placeholder
206         """
207         self.targetpoint_lt.append((data.pointdata.x, data.pointdata.y,
            ↪  data.pointdata.z))
208
209     def string_callback(self, data):
210         """
211         """
```

```
212         print(data.name)
213         print(data.string)
214 
215     ## Tasks ##
216     def task_robot_move_to_ref(self, Pref):
217         """
218         Pref: point according to reference frame
219         """
220         rTs = self.ref2tracker
221         sTk = self.env_calib_trans
222         Pk = convert_point(Pref, rTs, sTk)
223         return Pk
224 
225 
226     def calibrate_patient(self, n_sample=100, n_point=None): #WIP
227         """
228         """
229         rospy.sleep(10)
230         if n_point == None:
231             n_point = len(self.targetpoint_lt)
232         if n_point == 0:
233             print("No target points")
234             return
235         #pub = rospy.Publisher("IGTL_POINT_OUT", igtlpoint, queue_size=10)
236         calib_points = np.zeros((3, n_point, n_sample))
237         r = rospy.Rate(90)
238         counter = 0
239         while True:
240             #cmd = input("Press Enter to get a point")
241             os.system("spd-say 'Proceed to next point'")
242             rospy.sleep(5)
243             os.system("spd-say 'Starting'")
244             #if not cmd == "":
245             #    break
246             rospy.sleep(0.10)
247             Pr_samples = np.zeros((4, 1, n_sample), dtype=float)
248             for i in range(Pr_samples.shape[2]):
249                 r.sleep()
250                 T_Pm = np.matmul(self.sty2tracker, self.sty_calib_trans,)
251                 T_Pr = np.matmul(inverse_affine(self.ref2tracker), T_Pm)
252                 Pr = T_Pr[:,3][np.newaxis].T
253                 Pr_samples[:,:,i] = Pr
254             os.system("spd-say 'Done'")
255             rospy.sleep(1)
256             print("Point X=%.2f, Y=%.2f, Z=%.2f collected."%tuple(np.mean(Pr_samples,
            ↪  axis=2)[:3,0]))
257             #calib_points_lt.append(tuple(np.mean(Pr_samples, axis=0)[:3,0]))
258             calib_points[:, counter, :] = np.squeeze(Pr_samples)[:3,:]
259             counter += 1
260             if counter == n_point:
261                 X = np.array(self.targetpoint_lt).T
262                 Y = np.mean(calib_points, axis=2)
263                 np.savetxt("target_points.csv", X)
264                 np.savetxt("comp_collected_points.csv", Y)
265                 os.system("spd-say 'All done'")
266                 break
267 
```

```
268     def collect_points(self, n_point, n_sample=100):
269         """
270         """
271         points_array = np.zeros((3, n_point))
272         r = rospy.Rate(90)
273         counter = 0
274         try:
275             while True:
276                 cmd = input("Press Enter to get a point")
277                 if not cmd == "":
278                     break
279                 rospy.sleep(0.5)
280                 P_samples = np.zeros((3, n_sample), dtype=float)
281                 for i in range(n_sample):
282                     sty2tracker = self.sty2tracker
283                     T_Pm = np.matmul(sty2tracker, self.sty_calib_trans)
284                     Pm = T_Pm[:3,3]
285                     P_samples[:,i] = Pm
286                     r.sleep()
287                 P_mean = np.mean(P_samples, axis=1)
288                 points_array[:,counter] = P_mean[:3]
289                 print("Point X=%.2f, Y=%.2f, Z=%.2f collected."%(tuple(P_mean[:3])))
290                 counter += 1
291                 if counter == n_point:
292                     np.savetxt("collected_points.csv", points_array)
293                     break
294         except KeyboardInterrupt:
295             return
296 
297     def publish_calib(self):
298         msg_sty_calib = affine2igtltransform(self.sty_calib_trans,
            ↪  STY_CALIB_TRANS_NAME)
299         msg_ref_calib = affine2igtltransform(self.ref_calib_trans,
            ↪  REF_CALIB_TRANS_NAME)
300         self.pub_transform.publish(msg_sty_calib)
301         self.pub_transform.publish(msg_ref_calib)
302 
303     ## ROS listener setup ##
304     def run_listener(self):
305         """
306         """
307         rospy.Subscriber("IGTL_TRANSFORM_IN", igtltransform,
308                          self.transform_callback)
309         rospy.Subscriber("IGTL_POINT_IN", igtlpoint, self.point_callback)
310         rospy.spin() # Comment out ros.spin() if publisher is running so program
            ↪  doesnot prematurely close
311 
312     ## ROS publisher setup ##
313     def run_publisher(self):
314         """
315         """
316         #print(end_eff_transform)
317         rate = rospy.Rate(20)
318         while not rospy.is_shutdown():
319             #print("publisher running")
320             R, t = decompose_affine(self.end_eff_transform)
321             x, y, z = t
```

```
            a, b, c = [-156.44, 82.25, -154.88]
            #print("calling publish")
            self.pub_kuka.publish((x,y,z,a,b,c))
            rate.sleep()

    def publish_single(self, T):
        if not rospy.is_shutdown():
            #print("publisher running")
            R, t = decompose_affine(T)
            x, y, z = t
            a, b, c = [-156.44, 82.25, -154.88]
            #print("calling publish")
            self.pub_kuka.publish((x,y,z,a,b,c))

    def run(self):
        """
        """
        self.run_listener()
        #self.run_publisher()
        # run listener in another thread

    def run_thread(self):
        thread_1 = threading.Thread(target=self.run)
        thread_1.start()

    def command_interface(self):
        """
        """
        print("Command Line Interface\n\nCommands;\n\tlist\n\tclean\n\tkeep <Target
        ↪  index>\n\tgoref <Target index>\n\tcalcref <Target index>\n\texit\n")
        while True:
            cmd = input(">> ")
            cmd_lt = cmd.split(" ")
            if cmd_lt[0] == "list":
                for i, point in enumerate(self.targetpoint_lt):
                    print(i, "\b.\t", "X=%.2f, Y=%.2f, Z=%.2f"%point)
            elif cmd_lt[0] == "clean":
                self.targetpoint_lt = list()
            elif cmd_lt[0] == "keep":
                if not len(cmd_lt) == 2:
                    print("usage: 'keep <Target index>")
                else:
                    try:
                        target_index = int(cmd_lt[1])
                        self.targetpoint_lt = self.targetpoint_lt[-target_index:]
                    except ValueError as error:
                        rospy.logerr("Invalid target index\n%s"%error)
            elif cmd_lt[0] == "goref":
                if not len(cmd_lt) == 2:
                    print("usage: 'goref <Target index>")
                else:
                    try:
                        target_index = int(cmd_lt[1])
                        # Assume intend to fix the point to the ref
                        Pr = tuple2ndarray(self.targetpoint_lt[target_index])
                        Pk = self.task_robot_move_to_ref(Pr)
                        print(Pk[:3,0])
```

```
                        t = Pk[:3,0]
                        self.end_eff_transform = compose_affine(np.eye(3), t)
                        self.publish_single(self.end_eff_transform)
                    except ValueError as error:
                        rospy.logerr("Invalid target index\n%s"%error)
            elif cmd_lt[0] == "calcref":
                if not len(cmd_lt) == 2:
                    print("usage: 'calcref <Target index>")
                else:
                    try:
                        target_index = int(cmd_lt[1])
                        Ps = tuple2ndarray(self.targetpoint_lt[target_index])
                        Pr = convert_point(Ps, inverse_affine(self.ref2tracker))
                        point = ndarray2tuple(Pr)
                        self.targetpoint_lt.append(point)
                    except ValueError as error:
                        rospy.logerr("Invalid target index\n%s"%error)
            # ### WIP
            elif cmd_lt[0] == "calib_patient":
                self.calibrate_patient(100, n_point=36)
            elif cmd_lt[0] == "send_calib":
                self.publish_calib()
            elif cmd_lt[0] == "collect_points":
                if not len(cmd_lt) == 3:
                    print("usage: 'collect_points <# of points> <# of samples>")
                else:
                    self.collect_points(int(cmd_lt[1]), int(cmd_lt[2]))
            elif cmd_lt[0] == "debug":
                """
                """
            elif cmd_lt[0] == "exit":
                rospy.signal_shutdown("doit")
                break
            elif cmd_lt[0] == "":
                pass
            else:
                print("Unknown command")


if __name__ == "__main__":
    navi = Surgical_Navigation()
    navi.run_thread()
    navi.command_interface()
```

quaternion.py

```python
#!/usr/bin/env python
########################################
#The ROS module for Surgical navigation   #
#calculations                             #
########################################
#Author: Tugrul Uslu                      #
#Email: tugrul.uslu@ikc.edu.tr            #
#       Izmir Katip Celebi University     #
#          Mechanical Engineering         #
########################################
"""
Performing rotations with quaternions;
P' = q * P * conj(q)
P is a vector represented with (x, y, z, k) where k = 0.
Let q represent a a rotation along axis (x,y,z) with a rotation
angle of a, then;
q = cos(a/2) + i (x*sin(a/2)) + j(y*sin(a/2)) + k(z*sin(a/2))

Combining quaternions rotations;
let, q1 is the first rotation, q2 is second rotation,
in absolute frame of reference; q2*q1
in rotating objects frame of reference; q1*q2

Conjugate of a quaternion;
q = ix + jy + kz + w, then;
conj(q) = - ix - jy - kz + w
"""

from __future__ import division, absolute_import, print_function, unicode_literals

import numpy as np


class Quat(object):
    """
    """

    def __init__(self, param):
        """
        param; iterable, (a, b, c, d)
            where q = a + i b + j c + k d
        """
        self.param = param

    @property
    def conj(self):
        """
        """
        a, b, c, d = self.param
        param_conj = (a, -b, -c, -d)
        return Quat(param_conj)

    def __mul__(self, other):
        a, b, c, d = self.param
        e, f, g, h = other.param
```

```
56          a_new = a*e - b*f - c*g - d*h
57          b_new = b*e + a*f + c*h - d*g
58          c_new = a*g - b*h + c*e + d*f
59          d_new = a*h + b*g - c*f + d*e
60          param_new = (a_new, b_new, c_new, d_new)
61          return Quat(param_new)
62
63      def __iter__(self):
64          return self.param
65
66
67  def quat2mat(quat):
68      """
69      quat; (w, x, y, z) iterable
70      returns;
71      mat; size 3x3 2d numpy array
72      """
73      w, x, y, z = quat
74      mat = np.array(
75          [[1 - 2*(y**2 + z**2), 2*(x*y - w*z), 2*(w*y + x*z)],
76           [2*(x*y + w*z), 1 - 2*(x**2 + z**2), 2*(y*z - w*x)],
77           [2*(x*z - w*y), 2*(w*x + y*z), 1 - 2*(x**2 + y**2)]]
78      )
79      return mat
80
81
82  def mat2quat(mat):
83      """
84      mat; size 3x3 2d numpy array
85      returns;
86      quat; size 4 1d numpy array
87      references;
88
    ↪ http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/
89      """
90      tr = np.trace(mat)
91      if (tr > 0):
92          S = np.sqrt(tr+1.0) * 2  # S=4*a
93          a = 0.25 * S
94          b = (mat[2, 1] - mat[1, 2]) / S
95          c = (mat[0, 2] - mat[2, 0]) / S
96          d = (mat[1, 0] - mat[0, 1]) / S
97      elif ((mat[0, 0] > mat[1, 1]) & (mat[0, 0] > mat[2, 2])):
98          S = np.sqrt(1.0 + mat[0, 0] - mat[1, 1] - mat[2, 2]) * 2  # S=4*b
99          a = (mat[2, 1] - mat[1, 2]) / S
100         b = 0.25 * S
101         c = (mat[0, 1] + mat[1, 0]) / S
102         d = (mat[0, 2] + mat[2, 0]) / S
103     elif (mat[1, 1] > mat[2, 2]):
104         S = np.sqrt(1.0 + mat[1, 1] - mat[0, 0] - mat[2, 2]) * 2  # S=4*c
105         a = (mat[0, 2] - mat[2, 0]) / S
106         b = (mat[0, 1] + mat[1, 0]) / S
107         c = 0.25 * S
108         d = (mat[1, 2] + mat[2, 1]) / S
109     else:
110         S = np.sqrt(1.0 + mat[2, 2] - mat[0, 0] - mat[1, 1]) * 2  # S=4*d
111         a = (mat[1, 0] - mat[0, 1]) / S
```

```
112             b = (mat[0, 2] + mat[2, 0]) / S
113             c = (mat[1, 2] + mat[2, 1]) / S
114             d = 0.25 * S
115         return np.array([a, b, c, d])
```

euler.py

```python
#!/usr/bin/env python
########################################
#The ROS module for Surgical navigation   #
#calculations                             #
########################################
#Author: Tugrul Uslu                      #
#Email: tugrul.uslu@ikc.edu.tr            #
#       Izmir Katip Celebi University     #
#          Mechanical Engineering         #
########################################

from __future__ import division, absolute_import, print_function, unicode_literals

import numpy as np


def decompose_affine(A):
    """
    assumes orthogonolity
    A; size 4x4 2d numpy array
    returns;
    R; size 3x3 2d numpy array
    t; size 3 1d numpy array
    """
    assert A.shape == (4, 4)
    R = A[:3, :3]
    t = A[:3, 3]
    assert is_orthogonal(R)
    return R, t


def compose_affine(R, t):
    """
    R; size 3x3 2d numpy array
    t; size 3 1d numpy array
    returns;
    A; size 4x4 2d numpy array
    """
    assert R.shape == (3, 3)
    assert t.shape == (3,)
    A = np.eye(4)
    A[:3, :3] = R
    A[:3, 3] = t
    return A


def inverse_affine(A):
    """
    """
    assert A.shape == (4, 4)
    R, t = decompose_affine(A)
    t_ = t[np.newaxis].T
    t_ = -np.matmul(R.T, t_)
    t_ = np.reshape(t_, (3,))
    A_inv = compose_affine(R.T, t_)
```

```
    return A_inv


def is_orthogonal(mat, tolerance=1e-6):
    """
    mat; 2d numpy array
    """
    res = np.matmul(mat, mat.T)
    should_be_zero = np.linalg.norm(np.eye(res.shape[0]) - res)
    result = should_be_zero <= tolerance
    if not result:
        print("Orthogonality error: %f is higher than tolerance"%should_be_zero)
    return result


def euler2mat(xyz_lt, order="zyx"):
    """
    xyz_lt; list of angles in rads
    order; order of matmul, "xyz" or "zyx"
    """
    x, y, z = xyz_lt
    Rx = np.array(
        [[1, 0, 0],
         [0, np.cos(x), -np.sin(x)],
         [0, np.sin(x), np.cos(x)]]
    )
    Ry = np.array(
        [[np.cos(y), 0, np.sin(y)],
         [0, 1, 0],
         [-np.sin(y), 0, np.cos(y)]]
    )
    Rz = np.array(
        [[np.cos(z), -np.sin(z), 0],
         [np.sin(z), np.cos(z), 0],
         [0, 0, 1]]
    )
    if order == "xyz":
        R = np.matmul(Ry, Rz)
        R = np.matmul(Rx, R)
    elif order == "zyx":
        R = np.matmul(Ry, Rx)
        R = np.matmul(Rz, R)
    else:
        raise ValueError
    return R


def mat2euler(R, order="zyx"):
    """
    references;
    https://www.learnopencv.com/rotation-matrix-to-euler-angles/
    transforms3d
    """

    if order == "xyz":
        i, j, k = 2, 1, 0
        parity = True
```

```
    elif order == "zyx":
        i, j, k = 0, 1, 2
        parity = False
    else:
        raise ValueError

    assert is_orthogonal(R)
    sy = np.sqrt(R[i, i]**2 + R[j, i]**2)
    is_singular = sy < 1e-6

    if not is_singular:
        teta_x = np.arctan2(R[k, j], R[k, k])
        teta_y = np.arctan2(-R[k, i], sy)
        teta_z = np.arctan2(R[j, i], R[i, i])
    else:
        teta_x = np.arctan2(-R[j, k], R[j, j])
        teta_y = np.arctan2(-R[k, i], sy)
        teta_z = 0

    if parity:
        teta_x, teta_y, teta_z = -teta_z, -teta_y, -teta_x

    return (teta_x, teta_y, teta_z)
```

test_registration.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# # Testing Registration

# In[ ]:


from os.path import join, expanduser
from math import sqrt

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d, art3d
import matplotlib.animation as animation

#from registration import point_register_basic, point_register_weighted,
↪  apply_transformation, calc_errors
from point_register import point_register, calc_errors, apply_transformation
import data_pairs

font = {'family' : 'DejaVu Sans',
        'size'   : 12}

matplotlib.rc("font", **font)

def set_equal_aspect(ax, data):
    # Create cubic bounding box to simulate equal aspect ratio
    max_range = np.array([data[0,:].max()-data[0,:].min(),
    ↪  data[1,:].max()-data[1,:].min(), data[2,:].max()-data[2,:].min()]).max()
    Xb = 0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][0].flatten() +
    ↪  0.5*(data[0,:].max()+data[0,:].min())
    Yb = 0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][1].flatten() +
    ↪  0.5*(data[1,:].max()+data[1,:].min())
    Zb = 0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][2].flatten() +
    ↪  0.5*(data[2,:].max()+data[2,:].min())
    for xb, yb, zb in zip(Xb, Yb, Zb):
        ax.plot([xb], [yb], [zb], 'w')


# In[ ]:


data_pairs.print_pair_indexes(0,10)


# In[ ]:


# select pair
X_path, Y_path = data_pairs.data_path_pairs[2]

# Load points from cvs files
X = np.loadtxt(X_path)[:3,:]
```

```
Y = np.loadtxt(Y_path)[:3,:]


# In[ ]:


# Remove broken points
points2remove = []
X = np.delete(X, points2remove, axis=1)
Y = np.delete(Y, points2remove, axis=1)


# ## Performing registration with all point pairs

# In[ ]:


# Point registration method to be used

R, t = point_register(X, Y)

#print(f"Registration R = {R}")

# Transform the points
Xt = apply_transformation(X, R, t)

# Save Transformation matrix
T = np.eye(4)
T[:3,:3] = R
T[:3,-1] = t
home_path = expanduser("~")
np.savetxt(join(home_path, "calib_trans.csv"), T)

# Calculate Errors
rms_all, fre_mat_all = calc_errors(X, Y, R, t)

print(f"RMS = {rms_all}")
print(f"Max FRE = {fre_mat_all.max()}")
print(f"Min FRE = {fre_mat_all.min()}")


# In[ ]:


# Plot the points
fig = plt.figure(figsize=[6,6])
ax = fig.add_subplot(111, projection="3d")
ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")

#ax.scatter(X[0],X[1],X[2], c="b")
ax.scatter(Y[0],Y[1],Y[2], c="g")
ax.scatter(Xt[0],Xt[1],Xt[2], s=10, c="r")

set_equal_aspect(ax, Y)
plt.show()
```

```
108
109
110 # In[ ]:
111
112
113 # Plot fre heatmap
114 fig = plt.figure(figsize=[7.5,6])
115 ax = fig.add_subplot(111, projection="3d")
116 ax.set_xlabel("x axis")
117 ax.set_ylabel("y axis")
118 ax.set_zlabel("z axis")
119
120 pos = ax.scatter(Y[0],Y[1],Y[2], c=fre_mat_all.flatten(), s=75/4, alpha=0.6)#,
    ↪  cmap="cool")
121
122 set_equal_aspect(ax, Y)
123
124 # Create colorbar
125 ticks = np.linspace(fre_mat_all.min(), fre_mat_all.max(), 10)
126 plt.colorbar(pos, ticks=ticks)
127 plt.show()
128
129
130 # ## Registration with selected N point pairs
131
132 # In[ ]:
133
134
135 # Select N
136 N = 35
137
138 # Sort point indexes by FRE
139 best_sorted = np.argsort(fre_mat_all)
140 #print(best_sorted)
141
142 ### Sample registration points selections ###
143 # Take best N points
144 regist_indexes = best_sorted[:N]
145
146 # Take worst N points
147 #regist_indexes = best_sorted[-N:]
148
149 # Take first N points
150 #regist_indexes = np.array(range(4))
151
152 # Select registration points
153 #regist_indexes = np.array([27,  6,  7, 32, 11, 35,  3, 13,  0, 33, 22,  4,  1, 17,
    ↪  8, 28, 34, 19])
154
155 # Select registration points randomly
156 #indexes = np.arange(X.shape[1])
157 #np.random.shuffle(indexes)
158 #regist_indexes = indexes[:N]
159 #del indexes
160 ##########################################
161
162 # Get target points that are not used in registation
```

```
target_indexes = np.delete(np.array([i for i in range(X.shape[1])]), regist_indexes)

# Perform registration
R, t = point_register(X[:,regist_indexes], Y[:,regist_indexes])

# Transform the points
Xt = apply_transformation(X, R, t)

# Save Transformation matrix
T = np.eye(4)
T[:3,:3] = R
T[:3,-1] = t
#np.savetxt("mTk.csv", T)

# Calculate Errors
rms_all, _ = calc_errors(X, Y, R, t)
rms, fre_mat = calc_errors(X[:,regist_indexes], Y[:,regist_indexes], R, t)

print(f"Registration points n = {regist_indexes.shape[0]}")
print(f"Registration RMS = {rms}")
print(f"RMS for all points = {rms_all}")
print(f"Max FRE = {fre_mat.max()}")
print(f"Min FRE = {fre_mat.min()}")


# In[ ]:


# Calculate TRE
rms_tre, tre_mat = calc_errors(X[:,target_indexes], Y[:,target_indexes], R, t)

print(f"Target points n = {target_indexes.shape[0]}")
print(f"TRE RMS = {rms_tre}")
print(f"TRE mean = {np.mean(tre_mat)}")
print(f"Max TRE = {tre_mat.max()}")
print(f"Min TRE = {tre_mat.min()}")


# In[ ]:


# Plot the points
fig = plt.figure(figsize=[6,6])
ax = fig.add_subplot(111, projection="3d")
ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")

#ax.scatter(X[0],X[1],X[2], c="b")
ax.scatter(Y[0],Y[1],Y[2], c="g")
ax.scatter(Xt[0],Xt[1],Xt[2], s=10, c="r")

set_equal_aspect(ax, Y)
plt.show()


# In[ ]:
```

```
220
221
222 # Plot FRE heatmap
223 fig = plt.figure(figsize=[7.5,6])
224 ax = fig.add_subplot(111, projection="3d")
225 ax.set_xlabel("x axis")
226 ax.set_ylabel("y axis")
227 ax.set_zlabel("z axis")
228
229 pos = ax.scatter(Y[0,regist_indexes],Y[1,regist_indexes],Y[2,regist_indexes],
230                  c=fre_mat.flatten(), s=75, alpha=0.6, cmap="cool")
231
232 set_equal_aspect(ax, Y)
233
234 # Create colorbar
235 ticks = np.linspace(fre_mat.min(), fre_mat.max(), 10)
236 plt.colorbar(pos, ticks=ticks)
237 plt.show()
238
239
240 # In[ ]:
241
242
243 # Plot TRE heatmap
244 fig = plt.figure(figsize=[7.5,6])
245 ax = fig.add_subplot(111, projection="3d")
246 ax.set_xlabel("x axis")
247 ax.set_ylabel("y axis")
248 ax.set_zlabel("z axis")
249
250 pos = ax.scatter(Y[0,target_indexes],Y[1,target_indexes],Y[2,target_indexes],
251                  c=tre_mat.flatten(), s=75, alpha=0.6, cmap="cool")
252
253 set_equal_aspect(ax, Y)
254
255 # Create colorbar
256 ticks = np.linspace(tre_mat.min(), tre_mat.max(), 10)
257 plt.colorbar(pos, ticks=ticks)
258 plt.show()
259
260
261 # In[ ]:
262
263
264 # Plot ALL heatmap
265 fig = plt.figure(figsize=[7.5,6])
266 ax = fig.add_subplot(111, projection="3d")
267 ax.set_xlabel("x axis")
268 ax.set_ylabel("y axis")
269 ax.set_zlabel("z axis")
270
271 Y_all = np.hstack((Y[:,regist_indexes], Y[:,target_indexes]))
272 all_mat = np.hstack((fre_mat, tre_mat))
273
274 pos = ax.scatter(Y_all[0,:],Y_all[1,:],Y_all[2,:], c=all_mat.flatten(), s=75,
    ↪  alpha=0.6, cmap="cool")
275
```

```
276 set_equal_aspect(ax, Y)
277
278 # Create colorbar
279 ticks = np.linspace(all_mat.min(), all_mat.max(), 10)
280 plt.colorbar(pos, ticks=ticks)
281 plt.show()
```

point_register.py

```python
#!/usr/bin/env python3
########################################
#Registration library                  #
########################################
#Author: Tugrul Uslu                   #
#Email: tugrul.uslu@ikc.edu.tr         #
#      Izmir Katip Celebi University   #
#         Mechanical Engineering       #
########################################

import numpy as np


def apply_transformation(X, R, t):
    """
    X: ndarray, shape (3,n)
        Points in reference space
    R: ndarray, shape (3,3)
        Rotation matrix
    t: ndarray, shape(3,)
        Translation Vector
    Returns:
    Xt: ndarray, shape (3,n)
        Points in transformed reference space
    """
    assert X.shape[0] == 3
    Xt = R @ X
    Xt = Xt + np.repeat(t[np.newaxis].T, X.shape[1], axis=1)
    return Xt


def calc_errors(X, Y, R, t):
    """
    Calculate fiducial registration error
    X: ndarray, shape: 3 rows, i columns
        Points in reference space
    Y: ndarray, shape: 3 rows, i columns
        Points in target space
    R: ndarray, shape (3,3)
        Rotation matrix
    t: ndarray, shape (3,)
        Translation Vector
    Returns;
    rms: float
        Mean fiducial registration error
    fre_mat: ndarray, shape: shape (n,)
        fiducial registration error at each point
    """
    assert X.shape == Y.shape
    # Number of points
    N = X.shape[1]
    # Apply transformation
    Xt = apply_transformation(X, R, t)
    # Find distance between points (FRE for each point)
    fre_mat = np.linalg.norm(Xt - Y, axis=0)
```

```
    # RMS error
    rms = np.sqrt(np.mean(fre_mat**2))
    return rms, fre_mat


def point_register(X, Y):
    """
    X: ndarray, shape (3,n)
    Y: ndarray, shape (3,n)
    returns;
    R: ndarray, shape (3,3)
    t: ndarray, shape (3,)
    """
    assert (X.shape == Y.shape) and (X.shape[0] == 3)
    # Demean point sets
    Xc = X - np.repeat(np.mean(X, axis=1, keepdims=True), X.shape[1], axis=1)
    Yc = Y - np.repeat(np.mean(Y, axis=1, keepdims=True), Y.shape[1], axis=1)
    # Calculate SVD of covariance
    U, _, Vh = np.linalg.svd(Xc @ Yc.T)
    # Calculate R rotation and t translation
    R = Vh.T @ np.diag([1, 1, np.linalg.det(Vh.T @ U)]) @ U.T
    t = np.mean(Y, axis=1) - R @ np.mean(X, axis=1)
    return R, t
```

APPENDIX B

kuka_ws_scan_node.py

```
#!/usr/bin/env python
########################################
#Author: Tugrul Uslu                   #
#Email: tugrul.uslu@ikcu.edu.tr        #
#      Izmir Katip Celebi University   #
#         Mechanical Engineering       #
########################################

from __future__ import division, absolute_import, print_function, unicode_literals
import sys
import os.path
import time
from threading import Thread

import numpy as np
import rospy  # pylint: disable=E0401
from std_msgs.msg import Float64MultiArray  # pylint: disable=E0401
from sensor_msgs.msg import JointState  # pylint: disable=E0401

from NatNetClient import NatNetClient


if sys.version_info.major < 3:
    input = raw_input
    range = xrange

### CONSTANTS ###
POINTS_ARRAY_NAME = "points_array_14092020-0802.csv"
MOTIVE_POINTS_NAME = "motive_points_%s.csv"%(time.strftime("%d%m%Y-%H%M",
↪  time.gmtime()))

SELF_IP = "172.31.1.150"
MULTICAST_IP = "239.255.42.99"


class NatNetInter(object):  # TODO natnetinter can be a ros pkg in itself
    """
    """

    def __init__(self, server_ip, mcast_grup):
        """
        """
        self.__natnetclient = NatNetClient(serverIPAddress=server_ip,
                                           multicastAddress=mcast_grup,
                                           natNetStreamVersion=(2, 7, 0, 0))
        self.status = dict()

    def setup_point_collector(self, sample_n=100, filename=MOTIVE_POINTS_NAME):
        """
        """
        if os.path.isfile(os.path.join(os.path.expanduser("~"), filename)):
```

```
51              # Reconstruct point_lt var
52              motive_points = np.loadtxt(filename)
53              self.point_lt = [motive_points[:, i]
54                               for i in range(motive_points.shape[1])]
55          else:
56              self.point_lt = list()
57          self.__natnetclient.newFrameListener = self.__point_collector
58          self.file_path = os.path.join(os.path.expanduser("~"), filename)
59          self.sample_n = sample_n
60          self.counter = sample_n + 1  # set counter 0 to start collecting
61          self.point_sample_container = np.zeros((3, sample_n))
62          # set initial status for point collecting
63          self.status["is_point_collector_setup"] = True
64          self.status["is_collecting"] = False
65          self.status["is_waiting"] = False
66          self.status["progress_to_next"] = False
67
68      def start_point_collector(self):
69          """
70          """
71          if self.status["is_collecting"]:
72              pass
73          else:
74              rospy.loginfo("Starting point collecting.")
75              try:
76                  assert self.status["is_point_collector_setup"] == True
77              except KeyError:
78                  print(
79                      "To start collecting points, setup_point_collector method must be
                        ↪  called first")
80                  return
81              except AssertionError:
82                  print("Point Collecting Disabled.")
83                  return
84              self.status["is_collecting"] = True
85              self.counter = 0
86
87      def __point_collector(self, unlabeledMarkersCount, unlabeledMarkerPos):
88          """
89          """
90          #rospy.loginfo("unlabeledMarkersCount: %d"%unlabeledMarkersCount)
91          if self.status["is_collecting"]:
92              if (self.counter < self.sample_n) and (unlabeledMarkersCount != 1):
93                  rospy.logwarn(
94                      "Warning, unlabeled marker count is not one. Remove the
                        ↪  additional marker.")
95                  self.counter = 0
96              else:
97                  pos = unlabeledMarkerPos[0]
98                  if self.counter < self.sample_n:
99                      self.point_sample_container[:, self.counter] = pos
100                     self.counter += 1
101                 elif self.counter == self.sample_n:
102                     rospy.loginfo("Successfully collected points.")
103                     point = np.mean(self.point_sample_container, axis=1)*1000
104                     rospy.loginfo("Collected Point %.2f X, %.2f Y, %.2f Z" %
105                                   tuple(point))
```

```
106                     self.point_lt.append(point)
107                     np.savetxt(self.file_path, np.array(self.point_lt).T)
108                     self.status["progress_to_next"] = True
109                     self.counter += 1
110 
111     def run(self):
112         """
113         """
114         self.__natnetclient.run()
115 
116     def stop(self):
117         """
118         """
119         self.__natnetclient.stop()
120 
121 
122 class KukaWSScan(object):
123     """
124     For local rotation, order is C -> B -> A
125     For global rotation, order is A -> B -> C
126     """
127 
128     def __init__(self):
129         """
130         """
131         # Initialize ROS
132         rospy.init_node("kuka_ws_scan_node")
133         # Create command publisher
134         self.command_pub = rospy.Publisher("position_trajectory_controller/command",
135                                            Float64MultiArray,
136                                            queue_size=10)
137         # Get target points from memory
138         try:
139             path = os.path.join(os.path.expanduser("~"), POINTS_ARRAY_NAME)
140             self.points_array = np.loadtxt(path)
141         except IOError:
142             msg = "Cannot find points_array file. "
143             rospy.logerr(msg)
144             rospy.signal_shutdown(msg)
145             raise
146         # ###
147         # Initialize natnetclient
148         self.natnetinter = NatNetInter(SELF_IP, MULTICAST_IP)
149         self.natnetinter.setup_point_collector()
150         self.natnetinter.run()
151         # ###
152         # Call self.terminate on exit
153         rospy.on_shutdown(self.terminate)
154         # point index counter
155         self.point_index = len(self.natnetinter.point_lt)
156         if self.point_index == self.points_array.shape[1]:
157             msg = "There are no more points to collect.\nQuitting"
158             rospy.loginfo(msg)
159             rospy.signal_shutdown(msg)
160 
161 
162     def all_close(self, goal, actual, tolerance=0.001):
```

```
163         """
164         Convenience method for testing if a list of values are within a tolerance of
    ↪ their counterparts in another list
165         @param: goal       A list of floats, a Pose or a PoseStamped
166         @param: actual     A list of floats, a Pose or a PoseStamped
167         @param: tolerance  A float
168         @returns: bool
169         """
170         allclose = True
171         for index in range(3):
172             if abs(actual[index] - goal[index]) > tolerance:
173                 allclose = False
174         return allclose
175 
176     def joint_states_callback(self, joint_states_msg):
177         """
178         """
179         joint_states = joint_states_msg.position
180         if self.point_index < self.points_array.shape[1] and
        ↪  self.all_close(joint_states, self.points_array[:3, self.point_index]):
181             #rospy.loginfo("Starting point collection.")
182             self.natnetinter.start_point_collector()
183 
184     def run_listerner(self):
185         """
186         """
187         rospy.Subscriber("joint_states", JointState,
188                          self.joint_states_callback)
189         rospy.spin()
190 
191     def terminate(self):
192         """
193         """
194         self.natnetinter.stop()
195 
196     def run(self):
197         """
198         """
199         listener_thread = Thread(target=self.run_listerner)
200         listener_thread.start()
201 
202         rospy.sleep(5)
203 
204         rate = rospy.Rate(20)
205         point_n = self.points_array.shape[1]
206         try:
207             while not rospy.is_shutdown():
208                 if self.point_index == (point_n) and not
                ↪  self.natnetinter.status["is_waiting"]:
209                     msg = "All points are completed"
210                     rospy.loginfo(msg)
211                     rospy.signal_shutdown(msg)
212                 elif self.natnetinter.status["progress_to_next"] and self.point_index
                ↪  <= (point_n-1):
213                     rospy.loginfo("Stepping.")
214                     self.point_index += 1
215                     self.natnetinter.status["progress_to_next"] = False
```

```
216                     self.natnetinter.status["is_collecting"] = False
217                     self.natnetinter.status["is_waiting"] = False
218                 elif not self.natnetinter.status["is_waiting"]:
219                     rospy.loginfo(
220                         "====== Processing point number %d =======" %
                            ↪  self.point_index)
221                     point = self.points_array[:, self.point_index]
222                     rospy.loginfo("Point %.2f X, %.2f Y, %.2f Z, %.2f A, %.2f B, %.2f
                        ↪  C" %
223                                   tuple(point))
224                     target = Float64MultiArray()
225                     #target.data = (point[0], point[1], point[2],
226                     #               0.0, -90.0, 180.0)
227                     target.data = (point[0], point[1], point[2],
228                                    point[3], point[4], point[5])
229                     self.command_pub.publish(target)
230                     rospy.loginfo("send.")
231                     self.natnetinter.status["is_waiting"] = True
232                 else:
233                     rate.sleep()
234             except rospy.ROSInterruptException:
235                 rospy.signal_shutdown("Keyboard Interruption")
236 
237     def debug(self):
238         rospy.sleep(1)
239         if not rospy.is_shutdown():
240             target = Float64MultiArray()
241             target.data = (600.0, 0.0, 900.0, 0.0, -90.0, 180.0)
242             self.command_pub.publish(target)
243         rospy.sleep(1)
244 
245 
246 if __name__ == "__main__":
247     kuka = KukaWSScan()
248     kuka.run()
```

APPENDIX C

find_endeff_length.ipynb

```
#!/usr/bin/env python
# coding: utf-8

# # Find Kuka's Effective End Effector Length

# In[ ]:


from os.path import join, expanduser
from math import sqrt

import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d, art3d
import matplotlib.animation as animation

from euler import decompose_affine, compose_affine, inverse_affine, euler2mat,
↪  mat2euler
from point_register import point_register, calc_errors, apply_transformation
import data_pairs

font = {'family' : 'DejaVu Sans',
        'size'   : 12}

matplotlib.rc("font", **font)

def set_equal_aspect(ax, data):
    # Create cubic bounding box to simulate equal aspect ratio
    max_range = np.array([data[0,:].max()-data[0,:].min(),
    ↪  data[1,:].max()-data[1,:].min(), data[2,:].max()-data[2,:].min()]).max()
    Xb = 0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][0].flatten() +
    ↪  0.5*(data[0,:].max()+data[0,:].min())
    Yb = 0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][1].flatten() +
    ↪  0.5*(data[1,:].max()+data[1,:].min())
    Zb = 0.5*max_range*np.mgrid[-1:2:2,-1:2:2,-1:2:2][2].flatten() +
    ↪  0.5*(data[2,:].max()+data[2,:].min())
    for xb, yb, zb in zip(Xb, Yb, Zb):
        ax.plot([xb], [yb], [zb], 'w')


# In[ ]:


# select pair
home_path = expanduser("~")
database_path = join(home_path, "Database")
X_path, Y_path = join(database_path, "points_array_14092020-0802.csv"),
↪  join(database_path, "motive_points_14092020-0807.csv")
#X_path, Y_path = join(database_path, "points_array_04092020-1150.csv"),
↪  join(database_path, "motive_points_04092020-1158.csv")
```

```
#X_path, Y_path = join(database_path, "points_array_03092020-1205.csv"),
↪  join(database_path, "motive_points_03092020-1258.csv")

# Load points from cvs files
X = np.loadtxt(X_path)[:3,:]
Y = np.loadtxt(Y_path)[:,:]
ABC = np.loadtxt(X_path)[3:,:]


# In[ ]:


# Remove broken points
points2remove = []
X = np.delete(X, points2remove, axis=1)
Y = np.delete(Y, points2remove, axis=1)
ABC = np.delete(ABC, points2remove, axis=1)


# In[ ]:


# Plot the points
fig = plt.figure(figsize=[6,12])
ax = fig.add_subplot(211, projection="3d")
ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")
ax.scatter(Y[0],Y[1],Y[2], c="g")
set_equal_aspect(ax, Y)

ax = fig.add_subplot(212, projection="3d")
ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")
ax.scatter(X[0],X[1],X[2], c="b")
set_equal_aspect(ax, X)

plt.show()


# In[ ]:


# Point registration method to be used

R, t = point_register(X, Y)

#print(f"Registration R = {R}")

# Transform the points
Xt = apply_transformation(X, R, t)

# Save Transformation matrix
T = np.eye(4)
T[:3,:3] = R
T[:3,-1] = t
```

```
np.savetxt("calib_trans.csv", T)

# Calculate Errors
rms_all, fre_mat_all = calc_errors(X, Y, R, t)

print(f"RMS = {rms_all}")
print(f"Max FRE = {fre_mat_all.max()}")
print(f"Min FRE = {fre_mat_all.min()}")


# In[ ]:


# Plot fre heatmap
fig = plt.figure(figsize=[7.5,6])
ax = fig.add_subplot(111, projection="3d")
ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")

pos = ax.scatter(X[0],X[1],X[2], c=fre_mat_all.flatten(), s=75, alpha=0.6)#,
↪  cmap="cool")

set_equal_aspect(ax, X)

# Create colorbar
ticks = np.linspace(fre_mat_all.min(), fre_mat_all.max(), 10)
plt.colorbar(pos, ticks=ticks)
plt.show()


# In[ ]:


h_tool = 60
h_start = 50
h_end = 60
h_step = 0.01


# In[ ]:


h_array = np.arange(h_start, h_end, h_step) - h_tool
rms_array = np.zeros(h_array.shape[0])
for i,h in enumerate(h_array):
    # Calculate rotations
    Rs = np.array([euler2mat(np.deg2rad([c, b, a]), order="zyx") for a,b,c in ABC.T])
    # Construct translation vector
    t = np.array([0, 0, h])
    # Apply transformations to the points
    Xh = np.array([Rs[j,:,:] @ t + X[:,j] for j in range(Rs.shape[0])]).T
    # Point registration method to be used
    R, t = point_register(Xh, Y)
    # Transform the points
    Xt = apply_transformation(Xh, R, t)
    # Calculate Errors
```

```
157         rms_all, _ = calc_errors(Xh, Y, R, t)
158         # Store RMS value
159         rms_array[i] = rms_all
160 
161     print(h_array[rms_array.argmin()]+h_tool)
162     print(rms_array.min())
163 
164 
165     # In[ ]:
166 
167 
168     # Plot RMS per h
169     fig = plt.figure(figsize=[12,6])
170     ax = fig.add_subplot(111)
171     ax.plot(h_array+h_tool, rms_array)
172     ax.set_aspect(1)
173     ax.grid()
174     plt.show()
175 
176 
177     # In[ ]:
178 
179 
180     def fit_data(data_x, data_y, deg=3, num=100):
181         coeffs = np.polyfit(data_x, data_y, deg=deg)
182         print(f"""Polynomial coefficients = {coeffs}""")
183         def y_poly(coeffs,x):
184             y = 0
185             order = len(coeffs) - 1
186             for i ,coeff in enumerate(coeffs):
187                 y += coeff*x**(order-i)
188             return y
189         data_x_fit = np.linspace(data_x.min(), data_x.max(), num=num)
190         data_y_fit = np.array([y_poly(coeffs,x) for x in data_x_fit])
191         data_fitted = np.vstack((data_x_fit, data_y_fit))
192         return coeffs, data_fitted
193 
194 
195     coeffs, data_fitted = fit_data(h_array+h_tool, rms_array, deg=6,
    ↪  num=h_array.shape[0])
196     dcoeffs = np.polyder(coeffs)
197     droots = np.roots(dcoeffs)
198     print(f"""Roots of Derivative = {droots}""")
199 
200     # Plot RMS per h
201     fig = plt.figure(figsize=[12,6])
202     ax = fig.add_subplot(111)
203     ax.plot(h_array+h_tool, rms_array)
204     ax.plot(data_fitted[0], data_fitted[1])
205     ax.set_aspect(1)
206     ax.grid()
207     plt.show()
208 
209 
210     # In[ ]:
211 
212 
```

```
# Manually select h value from the roots
# Effective tool length is;
h_ = 56.15622603
h = h_ - h_tool

# Calculate rotations
Rs = np.array([euler2mat(np.deg2rad([c, b, a]), order="zyx") for a,b,c in ABC.T])
# Construct translation vector
t = np.array([0, 0, h])
# Apply transformations to the points
Xh = np.array([Rs[j,:,:] @ t + X[:,j] for j in range(Rs.shape[0])]).T
# Point registration method to be used
R, t = point_register(Xh, Y)
# Transform the points
Xt = apply_transformation(Xh, R, t)
# Calculate Errors
rms_all, fre_mat_all = calc_errors(Xh, Y, R, t)

# Save Environment Calibration
T = np.eye(4)
T[:3,:3] = R
T[:3,-1] = t
T = inverse_affine(T)
np.savetxt(join(home_path, "environment_calib_trans.csv"), T)

print(f"RMS = {rms_all}")
print(f"Max FRE = {fre_mat_all.max()}")
print(f"Min FRE = {fre_mat_all.min()}")


# In[ ]:


# Plot the points
fig = plt.figure(figsize=[6,6])
ax = fig.add_subplot(111, projection="3d")
ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")

#ax.scatter(X[0],X[1],X[2], c="b")
ax.scatter(Y[0],Y[1],Y[2], c="g")
ax.scatter(Xt[0],Xt[1],Xt[2], s=10, c="r")

set_equal_aspect(ax, Y)
plt.show()


# In[ ]:


# Plot fre heatmap
fig = plt.figure(figsize=[7.5,6])
ax = fig.add_subplot(111, projection="3d")
ax.set_xlabel("x axis")
ax.set_ylabel("y axis")
ax.set_zlabel("z axis")
```

```
270
271 pos = ax.scatter(X[0],X[1],X[2], c=fre_mat_all.flatten(), s=75, alpha=0.6)#,
    ↪  cmap="cool")
272
273 set_equal_aspect(ax, X)
274
275 # Create colorbar
276 ticks = np.linspace(fre_mat_all.min(), fre_mat_all.max(), 10)
277 plt.colorbar(pos, ticks=ticks)
278 plt.show()
279
280
281 # In[ ]:
```

APPENDIX D

slicer_surnav_node.py

```
#!/usr/bin/env python
########################################
#The ROS module for surgical navigation   #
#with 3d Slicer.                          #
########################################
#Author: Tugrul Uslu                      #
#Email: tugrul.uslu@ikcu.edu.tr            #
#       Izmir Katip Celebi University      #
#          Mechanical Engineering          #
########################################

from __future__ import division, absolute_import, print_function, unicode_literals

import threading
import sys

import numpy as np
import rospy  # pylint: disable=E0401
# from rospy.numpy_msg import numpy_msg # pylint: disable=E0401
from ros_igtl_bridge.msg import igtltransform, igtlpoint, igtlpointcloud  # pylint:
↪  disable=E0401
from geometry_msgs.msg import Point, Transform  # pylint: disable=E0401
from std_msgs.msg import Float64MultiArray  # pylint: disable=E0401
from sensor_msgs.msg import JointState  # pylint: disable=E0401

from quaternion import quat2mat, mat2quat
from euler import decompose_affine, compose_affine, inverse_affine, euler2mat,
↪  mat2euler

if sys.version_info.major < 3:
    input = raw_input
    range = xrange

### CONSTANTS ###

# ROS Constants
NODE_NAME = "slicer_surnav_node"

# Transforms
REF_TO_TRACKER_NAME = "ReferenceToTracker"
STY_TO_TRACKER_NAME = "StylusToTracker"
REF_CALIB_TRANS_NAME = "RefCalib"
STY_CALIB_TRANS_NAME = "StyCalib"
CURRENT_POSE_NAME = "Current"
TARGET_TRANS_NAME = "Target"
REF_FINAL_POSE = "Reference"
STY_FINAL_POSE = "Stylus"

# Filenames
REF_CALIB_TRANS_FILE_NAME = "reference_calib_trans.csv"
STY_CALIB_TRANS_FILE_NAME = "stylus_calib_trans.csv"
```

```
50  ENV_CALIB_TRANS_FILE_NAME = "environment_calib_trans.csv"
51
52  ### UTILITIES ###
53
54
55  def tuple2ndarray(point):
56      """
57      """
58      P = np.ones((4, 1))
59      P[:3, 0] = point
60      return P
61
62
63  def ndarray2tuple(P):
64      """
65      """
66      point = tuple(P[:3, 0])
67      return point
68
69
70  def transform2mat_and_trans(transform):
71      """
72      """
73      t = np.array((transform.translation.x,
74                    transform.translation.y,
75                    transform.translation.z))
76      q = np.array((transform.rotation.w, transform.rotation.x,
77                    transform.rotation.y, transform.rotation.z))
78      R = quat2mat(q)
79      return R, t
80
81
82  def affine2igtltransform(A, name):
83      """
84      """
85      R, t = decompose_affine(A)
86      q = mat2quat(R)
87      msg = igtltransform()
88      msg.name = name
89      msg.transform.translation.x, msg.transform.translation.y,
        ↪  msg.transform.translation.z = t
90      msg.transform.rotation.w, msg.transform.rotation.x, msg.transform.rotation.y,
        ↪  msg.transform.rotation.z = q
91      return msg
92
93
94  def load_calib():
95      """
96      """
97      to_load = [ENV_CALIB_TRANS_FILE_NAME,
98                 REF_CALIB_TRANS_FILE_NAME,
99                 STY_CALIB_TRANS_FILE_NAME]
100     loaded_lt = []
101     for filename in to_load:
102         try:
103             loaded_lt.append(np.loadtxt(filename))
104         except IOError as err:
```

```
105                 loaded_lt.append(None)
106                 rospy.logwarn("%s, None type placed instead." % err)
107         return loaded_lt
108 
109 
110 def run_in_thread(func):
111     def wrapper(*args):
112         thread = threading.Thread(target=func, args=args)
113         thread.start()
114         rospy.loginfo("Function %s started in another thread." % func.__name__)
115     return wrapper
116 
117 
118 ### FUNCTIONS ###
119 
120 
121 def terminate():
122     """
123     """
124     rospy.signal_shutdown("Shutting down.")
125 
126 
127 def init_ros():
128     """
129     Initialize ROS
130     """
131     rospy.init_node(NODE_NAME, anonymous=True)
132     rospy.on_shutdown(terminate)   # exit gracefully
133 
134 
135 def convert_point(P, *transforms):
136     """
137     returns;
138         P; Converted point, ndarray shape 4,1
139     """
140     for transform in transforms:
141         P = np.matmul(transform, P)
142     return P
143 
144 
145 # @run_in_thread
146 def run_listeners(listener_data_lt):
147     """
148     listeners_lt: 2d list
149         [[msg_name, msg_type, calback_func], [...]]
150     """
151     try:
152         for listener in listener_data_lt:
153             rospy.Subscriber(*listener)
154             rospy.loginfo("Starting %s listener." % listener[0])
155         rospy.spin()
156     except KeyboardInterrupt:
157         return
158 
159 
160 def set_publishers(publisher_data_lt, queue_size=10):
161     """
```

```
162         publishers_lt: 2d list
163             [[msg_name, msg_type], [...]]
164         """
165         publishers_lt = []
166         for publisher in publisher_data_lt:
167             publishers_lt.append(rospy.Publisher(
168                 *publisher, queue_size=queue_size))
169             rospy.loginfo("Created %s publisher." % publisher[0])
170         return publishers_lt
171 
172 
173     ### TASKS ####
174 
175 
176     def basic_navigation():
177         """
178         """
179         rospy.loginfo("Starting basic navigation task.")
180         # Load calib data
181         calib_lt = load_calib()
182         # Set publishers
183         publisher_data_lt = [["position_trajectory_controller/command",
        ↪  Float64MultiArray],
184                              ["IGTL_POINTCLOUD_OUT", igtlpointcloud],
185                              ["IGTL_TRANSFORM_OUT", igtltransform]]
186         command_pub, pointcloud_pub, transform_pub = set_publishers(
187             publisher_data_lt)
188 
189         # ###
190         # Set transform holders
191         REFERENCE = 0
192         STYLUS = 1
193         transforms_lt = [np.eye(4), np.eye(4)]
194 
195         # ###
196         # Set Callbacks
197         def transform_callback(data):
198             """
199             """
200             R, t = transform2mat_and_trans(data.transform)
201             A = compose_affine(R, t)
202             # Handle ref trans
203             if data.name == REF_TO_TRACKER_NAME:
204                 try:
205                     env_calib = calib_lt[0]
206                     ref_calib = calib_lt[1]
207                     temp = np.matmul(env_calib, A)
208                     transforms_lt[REFERENCE] = np.matmul(temp, ref_calib)
209                 except ValueError:
210                     transforms_lt[REFERENCE] = A
211                 reference_msg = affine2igtltransform(
212                     transforms_lt[REFERENCE], REF_FINAL_POSE)
213                 transform_pub.publish(reference_msg)
214             # ###
215             # Handle sty trans
216             elif data.name == STY_TO_TRACKER_NAME:
217                 try:
```

```
                env_calib = calib_lt[0]
                sty_calib = calib_lt[2]
                temp = np.matmul(env_calib, A)
                transforms_lt[STYLUS] = np.matmul(temp, sty_calib)
            except ValueError:
                transforms_lt[STYLUS] = A
            stylus_msg = affine2igtltransform(
                transforms_lt[STYLUS], STY_FINAL_POSE)
            transform_pub.publish(stylus_msg)
        # ###
        # Handle target trans
        elif data.name == TARGET_TRANS_NAME:
            target = np.matmul(transforms_lt[REFERENCE], A)
            R, t = decompose_affine(target)
            c, b, a = np.rad2deg(mat2euler(R, order="zyx"))
            target_msg = Float64MultiArray()
            target_msg.data = (t[0], t[1], t[2], a, b, c)
            command_pub.publish(target_msg)
            rospy.loginfo(
                "Var send.\n%.2f X, %.2f Y, %.2f Z, %.2f A, %.2f B, %.2f C" %
                ↪  target_msg.data)
        # ###
        # Handle ref calib trans
        elif data.name == REF_CALIB_TRANS_NAME:
            # Save the transform for later use and hold it
            ref_calib = A
            np.savetxt(REF_CALIB_TRANS_FILE_NAME, ref_calib)
            calib_lt[1] = ref_calib
            # Inform user
            rospy.loginfo(REF_CALIB_TRANS_NAME+" is saved.")
        # ###
        # Handle ref calib trans
        elif data.name == STY_CALIB_TRANS_NAME:
            # Save the transform for later use and hold it
            sty_calib = A
            np.savetxt(STY_CALIB_TRANS_FILE_NAME, sty_calib)
            calib_lt[2] = sty_calib
            # Inform user
            rospy.loginfo(STY_CALIB_TRANS_NAME+" is saved.")
        # ###
        # Handle unknown
        else:
            rospy.logwarn(
                "Unexpected transformation received.\nThis warning might be caused by
                ↪  wrong naming of transformations.")

    def joint_states_callback(data):
        """
        """
        joint_states = data.position
        a, b, c = joint_states[3:]
        R = euler2mat(np.deg2rad([c, b, a]), order="zyx")
        t = np.array(joint_states[:3])
        A = compose_affine(R, t)
        current_pose_msg = affine2igtltransform(A, CURRENT_POSE_NAME)
        transform_pub.publish(current_pose_msg)

```

```
273     # ###
274     # Start listeners
275     listener_data_lt = [["IGTL_TRANSFORM_IN", igtltransform, transform_callback],
276                         ["joint_states", JointState, joint_states_callback]]
277     run_listeners(listener_data_lt)
278
279
280 if __name__ == "__main__":
281     init_ros()
282     basic_navigation()
```

CURRICULUM VITAE

Name and Surname: Tuğrul USLU
Place and Date of Birth: MANİSA 04.12.1995
Email: tugruluslu5483@gmail.com
ORCID: 0000-0002-2154-9268

Conference Papers:

USLU T, ÇETİN L, and GEZGIN E. Preliminary Study of a Surgical Navigation with Point Based Registration Method. 3. International Conference on Medical Devices 2020