ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Bachelor Thesis

Augmented Reality for Surgical Navigation using HoloLens

Student:	Fabian Schneider, 12-913-703
Professor:	Prof. Marc Pollefeys, ETH Zürich, D-INFK
Supervisors:	Prof. Orçun Göksel, ETH Zürich, D-ITET
	Dr. Philipp Fürnstahl, Universitätsklinik Balgrist, Universität Zürich
	Dr. Farrukh Iqbal Sheikh, ETH Zürich, D-ITET
Hand in date:	09.09.2017
Document version:	1.1

Abstract

A common surgical procedure in orthopedic spine surgery is the so called spinal fusion. Part of the spinal fusion procedure is the placement of pedicle screws into a human vertebra. For this task surgeons currently rely on either pre-manufactured, user-specific guides or marker based tracking for intraoperative navigation. It is highly desired to find new methods for intraoperative navigation that are freed from those drawbacks. Independent head-mounted augmented reality devices are one of the promising technologies that could be applied in this clinical setting. In this work we will make use of the Microsoft HoloLens, which is one of the most prominent current stand-alone augmented reality devices.

Our main contribution is a method for overlaying a 3D model of a vertebra onto a static real vertebra. To achieve this we devised a simple method for defining a point-cloud of vertebra landmarks using the Microsoft HoloLens. This is done by shooting virtual rays through the target landmarks from multiples positions and angles using a point and click mechanism. For each landmark the rays passing through it give rise to a intersection problem of askew 3D lines. We can solve it in a least-square sense yielding the desired point as its solution. Repeatedly applying this method for all landmarks generates a point cloud. Together with the corresponding 3D points predefined on the 3D model of the vertebra we can instantiate the two point clouds as an absolute orientation problem. We find the optimal transform which overlays the 3D model over the user defined set of points using Horn's closed-form solution of absolute orientation using unit quaternions.

We evaluated our method's accuracy and precision and express those as mean and variance of error distance measurements. In a first experiment we can show that our method is precise and accurate in defining a point with an error distance of 0.42 ± 0.15 mm from a virtual target point. I.e. we can repeatedly define points that are very close to the desired target. These measurements are done within the HoloLens' internal coordinate system. A second experiment shows that we couldn't get a similar accuracy or precision with respect to a real target. Measurements were done using scene captures from three orthogonal angles showing the real target and the virtual point. The average error distances measured on those 2D projections range from 6.8 mm to 8.7 mm. A systematic error in the experiment cannot be ruled out without any doubt, but we are confident that the accuracy of the method is mainly driven by the accuracy with which the device scans and understands its surroundings. A last experiment shows that

our method is quicker than a standard method for placing the vertebra 3D model. The standard method uses common hand gestures seen in HoloLens applications to drag and rotate a 3D model into place. The target is a virtual copy of the same vertebra 3D model.

The method can be used to establish a spatial correspondence between a 3D scan of an object an its real instance. This can be a basis for several medical or non-medical applications where information must be shown in a way that is taking the 3D structure of the real object into account.

Contents

1 Introduction

Augmented reality is defined as 'images produced by a computer and used together with a view of the real world' by the Cambridge English dictionary. This already is enough information for us to understand why augmented reality became part of popular culture so quickly and is becoming ever more important. It is a step towards the fusion of the virtual or artificial with the real world. Information processing systems are running and calculating at speeds that are unimaginable and became a loyal servant to us humans in many fields of application. Still there is a huge bottleneck in the information flow between the device and the human. Of course this is mostly due to the human nature that we cannot keep up with a modern computer in most computational tasks but sometimes this bottleneck is also narrowed by the way information is presented to us. E.g. visual information output is traditionally presented on a flat screen which is a more or less unchanged paradigm since many decades, despite the fact that we are equipped with all sensory needed to have a 3-dimensional experience and perception of the world we live in. For many tasks and types of information like reading and editing texts it still seems reasonable to display this information on a 2D screen but when dealing with 3D information that stems from the real world just in front of us it is obviously not always desired to have that information presented as a 2D projection since we basically lose a dimension. Another great disruption in information flow can be seen in the fact that we sometimes want additional information for something that we are looking at but have to turn our heads towards a screen to absorb this information. Let's further think of this by studying the information flow from the moment a patient is being scanned by a CT or MRI machine in a hospital to the time at which the doctor performs the surgical procedure on said patient. First we reconstruct the 3D information from parts of the body we don't have direct visual contact to. Then the information is inspected by a doctor as a series of images or as a 3D model on a 2D screen. This is already done at a considerable temporal distance from the moment when the patient was scanned and from a spatial distance to patient. The surgeon might add information like an operative plan in preparation to the surgery. Then the surgeon performs the surgery and constantly switches its visual focus between a screen showing the information and the real patient in front of him. One quickly sees that the information was projected, deviated and took a prolonged path which might be unnecessary for some applications. With head-mounted augmented reality devices we have means of displaying virtual information right into our line of sight where they belong. This closes the gap between human sensory and computers a bit more. For this to work such a device must first

of all establish a correspondence between the world around us and the virtual coordinate system it operates in. We've recently seen devices been brought to market or being in development which have reasonable size and weight but still sufficient computing power embedded into them for them to be operated as a stand-alone augmented reality device. One of these is the Hololens (Microsoft, Redmond CA, United States). We used the HoloLens in this work, but there are also devices produced or developed by other companies. The independence of these devices from external processing devices as well as the impressive spatial understanding these devices have render them attractive for usage in other fields besides military or entertainment. Especially medical applications are of interest, since using such a device could potentially improve the flow of information by presenting 3D data as 3D models where they actually belong to.

1.1 Motivation

Current concepts of surgical navigation have considerable drawbacks. Two concepts for intraoperative navigation are established in orthopedic surgery, namely surgical patient-specific guides and marker-based navigation systems. Surgical guides are custom-tailored and additively manufactured to fit the anatomy of a specific patient. Besides cost, a main drawback of the technique is the limited flexibility during the surgery. As the function of a guide must be defined preoperatively, the surgeon has no possibility to adapt the navigation plan online. Marker-based navigation systems offer more intraoperative flexibility, but they also have technical drawbacks. Optical tracking requires a direct line of sight between the observed objects and the camera system. Moreover, the interface to the surgeon often entails standard input/output devices, which is cumbersome and requires dedicated personnel. These considerable drawbacks of both systems motivate research to develop new navigation techniques.

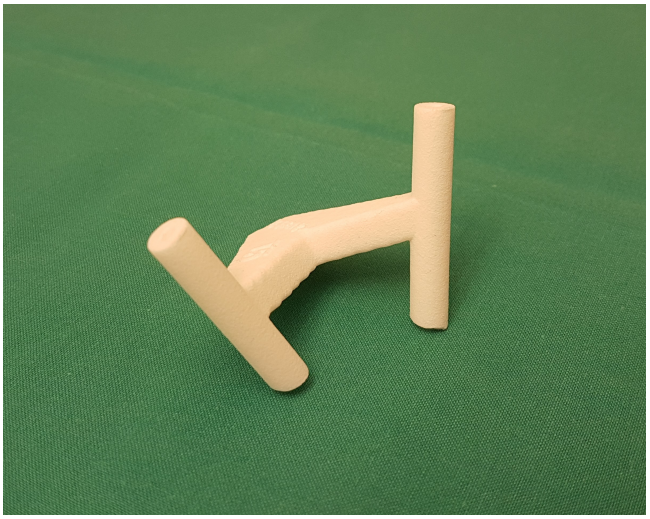

Figure 1: A 3D printed patient-specific guide for pedicle screw placement

1.2 Related Work

The first head-mounted 3D displays can be dated back to 1968, when Ivan Sutherland proposed a first such device [1]. It took many years after until medium-sized computers became sufficiently powerful to drive such a display with appropriate 3D content. A paper from Vogt et al. dated 2003 [2] presents a augmented reality system using a head-mounted device that just merely outperformed a predecessor system that was run by three networked SGI workstations. 'The system has a compelling real-time performance with 30 frames/second, displaying stereoscopic augmented video views with XGA resolution' is proudly stated in the abstract of that paper. Augmented reality for surgical navigation is a very active topic of research nowadays, but it doesn't surprise that most important work cannot be found before the 1990s. Because accuracy is the most important constraint in a clinical setting, augmented reality applications are as good as their capability of reliably registering 3D models with real world objects. We see that a lot of work from the 1990s and 2000s that involves augmented reality in medicine uses different approaches in displaying the information to the surgeon, but most importantly one notices that stand-alone image based registration is not found. Most methods enhance the registration of 3D models using additional tracking devices and markers, sometimes even directly integrated in the on site imaging device. A good source for references is the paper from Harders et al. from 2007 [3].
Clinical studies show significant advantages measured in speed and accuracy

of performed surgical tasks by using see-trough augmented reality devices [4]. Recent studies assess the feasibility and accuracy of pedicle screw placement using augmented reality surgical navigation provided by a hybrid operation theatres [5] or propose augmented reality surgical navigation using ultrasound-assisted registration for pedicle screw placement [6]. Marker-free and and purely stereo camera based registration is still rarely seen. A promising paper by Wang et al. [7] shows a 3D image overlay accuracy of 0.71 mm using a custom built stereo camera system and is completely marker-free.
One notable survey study on HoloLens in operating theatres is from Vasallo et al. [8] which states that 'our results show promise of this device's potential for intraoperative clinical use.' after testing the device and it's accuracy and stability under realistic clinical circumstances. By inspecting the internet one quickly finds that many institutes and companies at least work with head-mounted augmented reality devices in a clinical setting, e.g. [9], [10].

Figure 2: A hybrid operating theatre manufactured and sold by Philips

1.3 Outline

First we give a brief introduction into HoloLens, the Unity based API and explain some terminology. For the interested reader there is a explanation of Horn's closed-form solution of absolute orientation using unit quaternions and how it is useful in our context. Then we show our contributions, mainly the holographic application that enables the user to define a set of points

corresponding to prescribed landmarks. It is further described how we then register the 3D model using that set of user defined points. Finally we evaluate the method's precision and accuracy. We will use the most common definition of precision and accuracy. Precise is the method, if we can repeatedly get the same results with small variance. Accuracy is expressed by the mean of our measurements being very close to the desired or expected true value.

2 Background

In this section we want to present all the information and terminology needed for the reader to understand our work. We assume the reader has knowledge of basic linear algebra and computer graphics.

2.1 HoloLens

The Microsoft HoloLens is a standalone device with all computing power it needs for a convincing augmented reality experience already built-in. The device has limited computing power and battery size due to its cordless and portable design. It is equipped with a Intel 32 bit architecture 'Cherry Trail' SoC, which integrates CPU and GPU on the same silicon die. Alongside the SoC a custom-built co-processor called the Holographic Processing Unit (HPU) is used which is dedicated for processing all camera and sensor input. For both chips there are no details about clock frequencies and cache sizes available. The memory consists of 2GB RAM and 64GB flash storage. The battery typically lasts 2-3 hours when the device is actively used. Built-in sensors are one depth camera, four environment understanding cameras, one inertial measurement unit, one ambient light sensor. All of these sensors are not accessible through an API and are fed to the HPU. Additionally there is a 2MP RGB front facing camera which can be programmatically accessed and four microphones which are input to the speech recognition. Connectivity is provided with Bluetooth 4.1 LE and Wi-Fi 802.11ac [11]. The HoloLens comes with a optional Clicker device but the main interaction is done without any additional tools. The two main inputs beside position and viewing direction are voice and hand gesture recognition. The whole augmented reality experience is dependent on the illusion that a virtual 3D object keeps its position and orientation relative to the real world and is transformed according to the the user's movement. For this illusion to emerge, we need to know how the user moves through space and in which direction his head is pointing. Most other virtual reality or augmented reality devices use external tracking devices to achieve this, e.g. infra-red emitters on the head mounted device itself and multiple external infra-red cameras. As for the HoloLens, all of its sensors sit on the user's head and it is completely independent of external tracking. Therefore inferring the device's position and orientation in space becomes a problem that is closely related to camera pose estimation and SLAM, etc. Microsoft hasn't released any details on what algorithms are used, which of course is part of their intellectual property. The HPU estimates the camera pose and even reconstructs a 3D mesh of the surroundings. On top of that we get a coordinate system which is scaled to match the

distances of the real world around us. This coordinate system serves as our world coordinate system for rendering and is called the spatial coordinate system.

Figure 3: HoloLens advertisement by Microsoft

2.1.1 Holograms

In Microsoft's terminology, 3D objects rendered in a augmented reality environment such that they smoothly blend into the real world are called holograms. The term hologram shouldn't be confused with the concept of 3D imagery being described and imprinted on a 2D surface.

2.1.2 Spatial coordinate system

We see that the HoloLens has outstanding spatial perception, but it is still to show if it is accurate enough for medical applications. The spatial coordinate system is a core concept in the HoloLens API and the device's position and orientation are expressed in coordinates relative to this system, as well as any 3D object we wish to place in world around us. This scaled coordinate system matches the floating point values it operates in with the actual real world distances. I.e. if the point that represents the device was translated in the spatial coordinate system by a vector of magnitude 1.0f, we then know that we actually travelled approximately 1 meter in the real world. We tried to figure out where the origin of the spatial coordinate system is using a small 3D object rendered at the origin. From this we can quickly

see that the spatial coordinate system has its origin roughly at the point in real world space where the device has physically been at the moment the application is started. The origin and hence the coordinate system can switch its position and the unit scales of the coordinate system may be redefined at any time. This is due to the constant improvement and readjustment of the HoloLens' understanding of the world. We can observe drifting objects, i.e. they sometimes suddenly shift or pop into another position, whenever the coordinate systems changes [12].

2.1.3 Spatial mapping

As described in the last two sections the device has an understanding of its surroundings. The HoloLens reconstructs the environment and gives us access to a 3D triangle mesh representing it. This mesh is continually updated and may be read through the Unity based APIs easily. The quality and detail of the 3D mesh is not comparable to the 3D information a laser scan would deliver. But nevertheless we get an detailed understanding of the room's size and of all surfaces like table tops, the floor, etc. Yet we don't get detailed 3D information of more complex geometries. This is mainly due to the limited computation power and limited accuracy of the sensory input. One may expect this to improve in the future when high performance sensors shrink and the computational effort for a more detailed reconstruction is feasible. Very conveniently we may get a classification of surfaces into floor, ceiling, walls and platforms. A platform is any horizontal surface between floor and ceiling, e.g. tables and seats. This can drive applications that automatically place objects onto adequate surfaces [13]. The spatial map also drives the physical interaction of 3D objects with the real world, e.g. a 3D ball falling from the table and bouncing off the floor. In Unity the spatial map is just another physics engine enabled 3D object and a physics collider is attached to it. It is also the source for realistic object occlusion. If a 3D object is placed under a table, it gets occluded accordingly if we look at the scene from the top. This is just a common z-culling process but contributes a lot to the user's immersion [14].

2.1.4 Gaze

By far the most important input method is the actual head movement and head orientation. The forward viewing direction is also called the gaze direction. Most prominently the gaze is used in conjunction with a cursor object. The cursor itself can be at a fixed distance in front of the user or wherever the gaze intersects with a 3D object, i.e. the spatial map or any 3D model

that was placed in the environment. This mechanism makes it easy to point at and select objects [15].

2.1.5 Gestures and hand tracking

Simple hand gestures are recognized by the device's cameras. This implies that we are restricted to gesture with the arm stretched out such that the hand is in the camera's line of sight. Simple gestures which are supported by the API are the bloom gesture and the select gesture. The bloom gesture is performed by starting with a fist and then opening the hand by spreading all fingers. The select gesture is similar to a mouse click, but it is merely the basic component for composite gestures. It is performed by stretching out in a loose fist and tapping down the index finger which intially points up. Combining gaze input and the select gesture, we get a simple but effective point and click mechanism. But more complex gestures can be recognized by the system as well. The tap gesture is a select gesture followed by a release gesture, which is just pointing the finger back up. This is equivalent to a normal left mouse click in most HoloLens contexts. The hold gesture is a press gesture without a release. Combine the hold gesture with a hand movement and we get so called manipulation and navigation gestures. Manipulation gestures translate hand movements one to one, where as navigation gestures can be compared to a virtual joystick with a bounded range in all three directions [16].

2.1.6 Voice recognition

Another powerful input method for the HoloLens is voice recognition. HoloLens is enabled with Microsoft's speech recognition engine. The Unity API for voice recognition is astonishingly easy to use but an internet connection is mandatory. For keyword recognition one defines in code string constants of sentences to recognize and hook them up with callbacks. If one chooses words and sentences with more than 2 or 3 syllables a very good recognition rate can be expected. During our work with the HoloLens we didn't encounter problems with voice recognition. There is also the possibility to define SRGS grammars [17] using XML files. This enables us to define our own semantic interpretation of predefined sentence structures. In a HoloLens application a grammar recognizer object will then notify us if an object of our SRGS grammar has been recognized and we get all the information needed to act accordingly. As for now, only English is supported and recognized [18], [19].

2.2 Absolute orientation and Horn's solution

As we will see, a crucial part of this work is the step of finding a transform which overlays a 3D model of a vertebra over its real world instance, i.e. the actual human vertebra. We translate this into the task of attaining two sets of corresponding points and then finding the desired transform that matches the two point clouds as the solution to an instance of the absolute orientation problem. We will briefly explain coarlsy in which context and how we get the two point clouds. Then we describe the theory that underlies Horn's method for solving absolute orientation.

The 3D model exhibits predefined landmark points which are to be found and defined by the user in the real world. More bluntly said, we want to tell the HoloLens for each predefined landmark where it actually is to be found in the real world.
A use case scenario looks as follows. The patient's vertebra was scanned, e.g. using MRI, and we obtained a high quality 3D model of the vertebra. The surgeon enters the operating theatre and the patient's vertebra is exposed in a way that the predefined landmark points are directly visible on the bone. Using the HoloLens and our holographic application the surgeon will find those real landmark points one after another and triangulate a corresponding point in the spatial coordinate system. The application will show on a 3D model of the vertebra which landmarks need to be found and in what order this should be done. How this point definition exactly is done will be explained in section 3.7. After this procedure we will have a complete set of user defined points that correspond to the predefined landmarks. Those user defined points and predefined landmarks on the 3D model have a pairwise correspondence, i.e. we know which user defined point belongs to which 3D model landmark. Of course we assume here, that the user didn't make any mistakes and strictly stuck to the order of point definition given by the application. Since those two point clouds are two representation of the same set of points, we now may ask ourselves how one could find the transform that brings both together. More specifically, we want the 3D model of the vertebra to move towards the set of user defined points which represent the real vertebra. We will now explain how this can be done and start off with a simple instance of the problem. After that we explain the more realistic case where the two point clouds have a point correspondence but don't represent the same mathematical point due to measurement errors. We will see, that this exactly is what we are dealing with in our context and how this can be solved for an optimal transform. Good educational material on this topic has been created by Ziv Yaniv [20].

Now let's denote the set of n points which were defined by the user as $\boldsymbol{P_r} \in \mathbb{R}^{3\times n}$, the right set of points, and the n given landmarks on the 3D model as $\boldsymbol{P_l} \in \mathbb{R}^{3\times n}$, the left set of points. Since the pairing of the points in $\boldsymbol{P_r}$ and $\boldsymbol{P_l}$ is known, there exists a closed-form solution to the problem of finding a rigid transform T such that $\boldsymbol{P_r} = T(\boldsymbol{P_l})$. The problem is easy to solve if we could assume that the point pairs $\boldsymbol{p_{l,i}}, \boldsymbol{p_{r,i}} \in \mathbb{R}^{\mathbf{3}}$ for $i = 1..n$ each define the same point just relative to two different coordinate systems. The solution to this simpler problem requires only three point pairs and is constructed as follows:

Let $\boldsymbol{p_{l,1}}, \boldsymbol{p_{l,2}}, \boldsymbol{p_{l,3}}$ and $\boldsymbol{p_{r,1}}, \boldsymbol{p_{r,2}}, \boldsymbol{p_{r,3}}$ be the coordinates of the three points in the left and the right coordinate systems. For point set we construct a new orthogonal coordinate system which will yield a transform to the standard basis coordinate system. We can then combine them to get a transform from the right basis to the left basis and vice versa by transposing the resulting basis transform.

We show the construction mathematically:
1. Choose $\boldsymbol{p_{l,1}}$ as the origin of the new coordinate system for the left point set (analogous for right point set).
2. Construct new $\boldsymbol{x_l}$ axis:

$$\boldsymbol{x_l} = \frac{\boldsymbol{p_{l,2}} - \boldsymbol{p_{l,1}}}{\|\boldsymbol{p_{l,2}} - \boldsymbol{p_{l,1}}\|}$$

3. Construct new $\boldsymbol{y_l}$ axis:

$$\boldsymbol{y'_l} = (\boldsymbol{p_{l,3}} - \boldsymbol{p_{l,1}}) - [(\boldsymbol{p_{l,3}} - \boldsymbol{p_{l,1}}) \cdot \boldsymbol{x_l}]\boldsymbol{x_l}$$

$$\boldsymbol{y_l} = \frac{\boldsymbol{y'_l}}{\|\boldsymbol{y'_l}\|}$$

4. Construct new $\boldsymbol{z_l}$ axis:

$$\boldsymbol{x_l} \times \boldsymbol{y_l}$$

5. Construct rotation matrices for the left and right coordinate system:

$$\boldsymbol{R_l} = [\boldsymbol{x_l}, \boldsymbol{y_l}, \boldsymbol{z_l}], \quad \boldsymbol{R_r} = [\boldsymbol{x_r}, \boldsymbol{y_r}, \boldsymbol{z_r}]$$

6. Contruct rotation matrix that transforms from the new left coordinate

system to the new right coordinate system:

$$\boldsymbol{R} = \boldsymbol{R_r} \boldsymbol{R_l^T}$$

7. Construct translation between the new coordinate systems:

$$\boldsymbol{t} = \boldsymbol{p_{r,1}} - \boldsymbol{R} \boldsymbol{p_{l,1}}$$

With $\boldsymbol{R}$ and $\boldsymbol{t}$ we have the desired rigid transform and we can easily construct the 4x4 rotation and translation matrix for homogeneous 3D coordinates. This simple instance of our problem helps us understand what we are trying to achieve. Our realistic problem is a bit different, since the two sets of points may stem from two measurements that each inherently introduce some errors to the coordinates. In other words, there may not exist a transform T such that $\boldsymbol{P_r} = T(\boldsymbol{P_l})$. What we can do is define an error score for the rigid transform and find the transform T which minimizes it.

Mathematically:
The error distance between two corresponding points after applying a rigid transform is

$$\boldsymbol{e_i} = \boldsymbol{p_{r,i}} - \boldsymbol{R} \boldsymbol{p_{l,i}} - \boldsymbol{t}$$

and we seek to minimize the sum of squared error of all point pairs:

$$\sum_{i=1}^{n} \|\boldsymbol{e_i}\|_2^2$$

We now simplify the point representations by subtracting the centroids (mean points) from the coordinates. This is done for each set of points independently:

$$\boldsymbol{\mu_l} = \frac{\sum_{i=1}^{n} \boldsymbol{p_{l,i}}}{n}, \quad \boldsymbol{\mu_r} = \frac{\sum_{i=1}^{n} \boldsymbol{p_{r,i}}}{n}$$
$$\boldsymbol{p'_{l,i}} = \boldsymbol{p_{l,i}} - \boldsymbol{\mu_l}, \quad \boldsymbol{p'_{r,i}} = \boldsymbol{p_{r,i}} - \boldsymbol{\mu_r}$$

The error now becomes:

$$\boldsymbol{e_i} = \boldsymbol{p'_{r,i}} - \boldsymbol{R} \boldsymbol{p'_{l,i}} - \boldsymbol{t'}, \quad \boldsymbol{t'} = \boldsymbol{t} - \boldsymbol{\mu_r} + \boldsymbol{R} \boldsymbol{\mu_l}$$

Let's expand the sum of squared errors using the new definition of $\boldsymbol{e_i}$:

$$\begin{aligned}\sum_{i=1}^{n} \|\boldsymbol{e_i}\|_2^2 &= \sum_{i=1}^{n} \|\boldsymbol{p'_{r,i}} - \boldsymbol{R}\boldsymbol{p'_{r,i}} - \boldsymbol{t'}\|_2^2 \\ &= \sum_{i=1}^{n} \|\boldsymbol{p'_{r,i}} - \boldsymbol{R}\boldsymbol{p'_{l,i}}\|_2^2 - 2\boldsymbol{t'} \cdot \sum_{i=1}^{n} [\boldsymbol{p'_{r,i}} - \boldsymbol{R}\boldsymbol{p'_{l,i}}] + n\|\boldsymbol{t'}\|_2^2\end{aligned} \tag{1}$$

Because $\sum_{i=1}^{n} p_{l,i} = n\mu_l$ we have

$$\sum_{i=1}^{n} \boldsymbol{p'_{l,i}} = \sum_{i=1}^{n} (\boldsymbol{p_{l,i}} - \boldsymbol{\mu_l}) = \sum_{i=1}^{n} \boldsymbol{p_{l,i}} - \sum_{i=1}^{n} \boldsymbol{\mu_l} = n\boldsymbol{\mu_l} - n\boldsymbol{\mu_l} = 0$$

Which makes the middle term irrelevant.
Because the last term is minimized when setting $\boldsymbol{t'} = \boldsymbol{0}$ we get:

$$\boldsymbol{t} = \boldsymbol{\mu_r} - \boldsymbol{R}\boldsymbol{\mu_l}$$

This means that the translation is just given by subtracting the right centroid by the rotated left centroid. We are left to minimize the first term with respect to the rotational transform $\boldsymbol{R}$. We expand the first term:

$$\sum_{i=1}^{n} \|\boldsymbol{p'_{r,i}} - \boldsymbol{R}\boldsymbol{p'_{r,i}}\|_2^2 = \sum_{i=1}^{n} \|\boldsymbol{p'_{r,i}}\|_2^2 - 2\sum_{i=1}^{n} \boldsymbol{p'_{r,i}} \cdot \boldsymbol{R}\boldsymbol{p'_{l,i}} + \sum_{i=1}^{n} \|\boldsymbol{R}\boldsymbol{p'_{l,i}}\|_2^2$$

One can see that $\sum_{i=1}^{n} \|\boldsymbol{p'_{r,i}}\|_2^2$ and $\sum_{i=1}^{n} \|\boldsymbol{R}\boldsymbol{p'_{l,i}}\|_2^2$ are both constant with respect to the rotational transform, i.e. they are functions of the point sets only. Note that $\boldsymbol{R}$ being a length preserving transform implies the equality $\sum_{i=1}^{n} \|\boldsymbol{R}\boldsymbol{p'_{l,i}}\|_2^2 = \sum_{i=1}^{n} \|\boldsymbol{p'_{l,i}}\|_2^2$ and hence the value of $\sum_{i=1}^{n} \|\boldsymbol{R}\boldsymbol{p'_{l,i}}\|_2^2$ truely is independent of $\boldsymbol{R}$. It follows that maximizing $-2\sum_{i=1}^{n} \boldsymbol{p'_{r,i}} \cdot \boldsymbol{R}\boldsymbol{p'_{l,i}}$ minimizes $\sum_{i=1}^{n} \|\boldsymbol{p'_{r,i}} - \boldsymbol{R}\boldsymbol{p'_{r,i}}\|_2^2$.

Finding this transform $\boldsymbol{R}$ is in essence what is known as absolute orientation. We are using Horn's famous closed-form solution of absolute orientation [21] which uses unit quaternions to represent the rotational transform. At this point we won't explain quaternions in depth but just remind ourselves that they are four-dimensional objects, a unit quaternion $\boldsymbol{q} = [cos\frac{\theta}{2}, \boldsymbol{r}sin\frac{\theta}{2}]$ represents a rotation by θ radians around a rotational axis $\boldsymbol{r}$ and has the property $\boldsymbol{q} \cdot \boldsymbol{q} = 1$. The rotation defined by $\boldsymbol{q}$ is performed on a point $\boldsymbol{p}$ as $\boldsymbol{q} * \boldsymbol{p} * \overline{\boldsymbol{q}}$. Here $*$ is the quaternion multiplication and $\overline{\boldsymbol{q}}$ is the conjugate

quaternion of $\boldsymbol{q}$. Furthermore, $\boldsymbol{p}$ and all points in the two points sets are now 3D vectors embedded in pure quaternion, i.e. the real part of the quaternion is 0 and the imaginary part is equal to the 3D vector we want to rotate.
Now we may rewrite

$$-2\sum_{i=1}^{n} \boldsymbol{p'_{r,i}} \cdot \boldsymbol{R p'_{l,i}} = -2\sum_{i=1}^{n} \boldsymbol{p'_{r,i}} \cdot (\boldsymbol{q} * \boldsymbol{p'_{l,i}} * \overline{\boldsymbol{q}}) = -2\sum_{i=1}^{n} (\boldsymbol{q} * \boldsymbol{p'_{l,i}} * \overline{\boldsymbol{q}}) \cdot \boldsymbol{p'_{r,i}} = -2\sum_{i=1}^{n} (\boldsymbol{q} * \boldsymbol{p'_{l,i}}) \cdot (\boldsymbol{p'_{r,i}} * \boldsymbol{q})$$

We used the fact that For the next step, we need to remind ourselves, that quaternion multiplication can also be expressed as a matrix-vector product:

$$\boldsymbol{p'_{r,i}} * \boldsymbol{q} = \begin{bmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & -z'_{r,i} & y'_{r,i} \\ y'_{r,i} & z'_{r,i} & 0 & -x'_{r,i} \\ z'_{r,i} & -y'_{r,i} & x'_{r,i} & 0 \end{bmatrix} \boldsymbol{q} = \boldsymbol{P'_{r,i}} \boldsymbol{q}$$

$$\boldsymbol{q} * \boldsymbol{p'_{l,i}} = \boldsymbol{q} \begin{bmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & -z'_{r,i} & -y'_{r,i} \\ y'_{r,i} & -z'_{r,i} & 0 & x'_{r,i} \\ z'_{r,i} & y'_{r,i} & -x'_{r,i} & 0 \end{bmatrix} = \overline{\boldsymbol{P'_{l,i}}} \boldsymbol{q}$$

We use this to rewrite the dot product of two quaternion products as the dot product of two matrix-vector products. Note that we drop the factor of -2 for simplicity:

$$\begin{aligned} \sum_{i=1}^{n} (\boldsymbol{q} * \boldsymbol{p'_{l,i}}) \cdot (\boldsymbol{p'_{r,i}} * \boldsymbol{q}) &= \sum_{i=1}^{n} (\overline{\boldsymbol{P'_{l,i}}} \boldsymbol{q}) \cdot (\boldsymbol{P'_{r,i}} \boldsymbol{q}) \\ &= \sum_{i=1}^{n} (\overline{\boldsymbol{P'_{l,i}}} \boldsymbol{q})^{\boldsymbol{T}} (\boldsymbol{P'_{r,i}} \boldsymbol{q}) \\ &= \sum_{i=1}^{n} \boldsymbol{q}^{\boldsymbol{T}} \overline{\boldsymbol{P'_{l,i}}}^{\boldsymbol{T}} \boldsymbol{P'_{r,i}} \boldsymbol{q} \\ &= \boldsymbol{q}^{\boldsymbol{T}} \sum_{i=1}^{n} (\overline{\boldsymbol{P'_{l,i}}}^{\boldsymbol{T}} \boldsymbol{P'_{r,i}}) \boldsymbol{q} \end{aligned} \tag{2}$$

Since all matrices $\overline{\boldsymbol{P'_{l,i}}}^{\boldsymbol{T}} \boldsymbol{P'_{r,i}}$ are symmetric, a sum of such matrices is also symmetric. Let $\boldsymbol{N} = \sum_{i=1}^{n} \overline{\boldsymbol{P'_{l,i}}}^{\boldsymbol{T}} \boldsymbol{P'_{r,i}}$ and as mentioned, $\boldsymbol{N}$ is symmetric. Hence there will be a eigenvalue decomposition of $\boldsymbol{N}$ that yields four real eigenvalues $\lambda_1, ..., \lambda_4$ with $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq \lambda_4$ and corresponding four orthogonal unit eigenvectors $\boldsymbol{v_1}, ..., \boldsymbol{v_4}$, such that:

$$\boldsymbol{N}\boldsymbol{v_i} = \boldsymbol{\lambda_i}\boldsymbol{v_i}, \quad \text{for } i = 1, ..., 4$$

The eigenvectors span the 4D space and we may interpret them as 4D vectors. Since they are a basis, any quaternion and especially the unit quaternion $\boldsymbol{q}$ can be represented as a linear combination of the eigenvectors $\boldsymbol{v_1}, ..., \boldsymbol{v_4}$, i.e. for some real coordinates $\alpha_1, ..., \alpha_4$:

$$\boldsymbol{q} = \sum_{i=1}^{4} \alpha_i \boldsymbol{v_i}$$

The eigenvectors are orthogonal, i.e. $\boldsymbol{v_i} \cdot \boldsymbol{v_i} = 1$ for i = 1,...,4. Additionally remember that $\boldsymbol{q}$ is a unit quaternion. This gives us:

$$\boldsymbol{q} \cdot \boldsymbol{q} = \sum_{i=1}^{4} \alpha_i^2 = 1$$

Then, because $\boldsymbol{v_1}, ..., \boldsymbol{v_4}$ are eigenvectors:

$$\boldsymbol{N}\boldsymbol{q} = \boldsymbol{N} \sum_{i=1}^{4} \alpha_i \boldsymbol{v_i} = \sum_{i=1}^{4} \alpha_i \lambda_i \boldsymbol{v_i}$$

And all together:

$$\boldsymbol{q^T}\boldsymbol{N}\boldsymbol{q} = \sum_{i=1}^{4} \alpha_i^2 \lambda_i$$

As λ_1 is the largest eigenvalue and $\sum_{i=1}^{4} \alpha_i^2 = 1$ the expression $\boldsymbol{q^T}\boldsymbol{N}\boldsymbol{q}$ cannot become larger than λ_1. This can be seen, when we assume all eigenvalues are equal:

$$\boldsymbol{q^T}\boldsymbol{N}\boldsymbol{q} \leq \sum_{i=1}^{4} \alpha_i^2 \lambda_1 = \lambda_1 \sum_{i=1}^{4} \alpha_i^2 = \lambda_1$$

What we now immediatly see, is that the quadratic form $\boldsymbol{q^T}\boldsymbol{N}\boldsymbol{q}$ is maximized when we chose $\boldsymbol{q}$ to be $\boldsymbol{v_1}$, i.e. $\boldsymbol{q}$ has coordinates $\alpha_1 = 1$ and $\alpha_2 = \alpha_3 = \alpha_4 = 0$ with respect to the eigenvector basis.

3 Registering a 3D model with a set of user-defined points

3.1 Development environment

For programming, a system with Microsoft Windows 10 64-bit, Microsoft Visual Studio 2015 Enterprise, Unity Personal v5.5.2f1 [22] and Blender v2.78 [23] was used. A linear algebra library called AlgLib v3.10.0 [24] and many parts from the Unity HoloToolkit v1.5.5.0 [25] were incorporated into the mixed reality application.

Unity is a game development platform that is tailored and optimized to only provide functions and data structures needed for game programming and no general mathematical functions E.g. there is a `Matrix4x4` data type which is used to represent rigid transformations but there is no data type which represents a generic matrix. There is also no implementation of singular value decomposition. Therefore we had to incorporate a numerical library AlgLib [24] which is available as C# source code. Unity uses C# as a scripting language but for compilation it uses the Mono C# compiler and its implementation of a subset of the .NET framework. Therefore a class library must comply to the Mono subset of the .NET framework. Additionally we extensively used the Unity HoloToolkit which is a set of Unity scripts and assets that aid the development of mixed reality apps for HoloLens. Visual Studio 2015 has an option to assemble the library to a 'Unity 3.5 subset' which we did in the case of AlgLib. The Unity project then compiles into a UWP (Universal Windows Platform) project which has to be opened in Visual Studio. With UWP Microsoft brought a new platform which allows development for all Windows 10 devices including Xbox and HoloLens. The final step is to build this Visual Studio project as a x86 release build and deploy it to the HoloLens.

3.2 3D printed vertebra as target object

A common task in spine surgeries is the placement of pedicle screws. Since the human vertebra exhibits a lot of good visual features due to its complex geometry, it was chosen as an exemplary target. Balgrist CARD provided a 3D print of a vertebra. Throughout this work we used this as our target object. The corresponding 3D mesh was also made available. We drilled a hole in the 3D printed vertebra to mount it on a wooden board cutout using common M4 bolts and nuts. See figure 4.

Figure 4: The 3D printed vertebra is mounted on a wooden stand

3.3 Pre-processing the 3D model

The graphics processing unit of the HoloLens is fairly limited compared to desktop grade graphics chips. All we know about the SoC is that it is a Intel 'Cherry Trail' SoC. Therefore we may expect that the graphics processor has similar performance to those found in off-the-shelf Intel Atom X chips. This implies that we cannot expect to use the same models and shaders that we would use for high fidelity 3D applications that run on much faster hardware. It is also very important to have the application running at a constant 60 frames per second. Otherwise the augmented reality experience crumbles down because the rendered objects start to judder [26][27]. Therefore we preprocessed the 3D model of a vertebra we got supplied by Balgrist CARD. The vertex count was initially 20'472 and was reduced to 1'654. The face count was 33'094 and was reduced to 3'308. This was done using Blender. No vertex shaders are applied and a standard Unity pixel shader is used. Using the initial high fidelity models had a performance impact, such that an application showing two of them in the user's field of view would let the frame rate drop as low as 32 frames per second.

3.4 Unity scene structure

A mixed reality application for HoloLens is most conveniently prepared with Unity and the HoloToolkit for Unity. The HoloToolkit on the one hand sets up the target platform, the rendering quality settings etc. of the Unity project, on the other hand it configures the scene's main camera. The camera's near clipping plane is set to a distance of 0.45 m. Microsoft recommends to use a near clipping plane at 0.85 m. The optimal distance to holograms should be between 1.25 m and 5 m to avoid conflicts between eye accommodation and eye convergence. Operating at smaller distances can lead to discomfort and eye strain. We chose to work at smaller distances, since in clinical settings the objects of interest are closer than 0.85 m to the user [27]. Here, like in other 3D graphics applications, the camera is the object which defines the viewing projection, the viewing frustum and the position from which the scene is rendered [28]. We added a parent `GameObject` to the scene. This object contains all the main code in its associated Unity script. We use a spatial anchor for this parent object and all our vectors and models are children of this parent object. This has the advantage that even though the spatial coordinate system might change over time, we always have the vertebra models, rays and points at a fix position relative to each other. Without this our calculations would fail quickly since objects tend to move a bit relative to each other inside the spatial coordinate system. We added all the manager objects from the HoloToolkit [25] that are needed for gesture input, voice recognition and spatial mapping. The `InputManager` class provides an interface to the input system of the HoloLens as a observer pattern. We can listen on input events and register methods as input handlers. The `GazeManager` provides convenience functions, e.g. we can ask it to return the `GameObject` that we are currently looking at. A `GazeStabilizer` is registered to the `GazeManager` and its main purpose is to smooth the gaze vector using a simple Lerp function. This smoothed version gaze vector is driving the `BasicCursor` and reduces the jitter that is introduced by the user's subtle head movements. We used a `SpeechManager` which is a basic keyword recognizer. In the `SpeechManager` Unity script we add all desired keywords and pair them with a corresponding method to call. We also make use of the `SpatialMappingManager` which allows us to control the rendering of the spatial map. Besides the manager objects and scripts there is one static light source in the scene and a `HoloLensCamera`. This light source is needed to illuminate the models, otherwise they will stay dark in color.

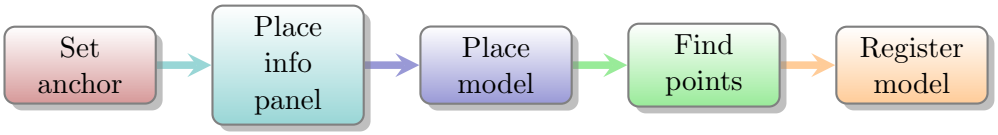

Figure 5: Main steps using our application

3.5 User workflow

Figure 5 shows the main work flow a user goes through when using our application. The very first step is to select a point close to the working space. We achieve this by showing the spatial map to the user and letting the user choose a point on the mesh using gaze and the air tap gesture. We transform our application's main `GameObject` to this position and add a spatial anchor to it. Then the user has to place an info panel in a similar way. Its purpose is to show information to the user, which is not otherwise visually conveyed by the holograms. After this initial setup phase we proceed by letting the user place a hologram, which shows the landmarks that the user should find as highlighted areas on a 3D model of a vertebra. Finally the user has to find those landmark points in the real world. As soon as all landmarks are found and the corresponding points are defined, the hologram is translated and rotated by point cloud registration. In the next sections we describe some of these steps in more detail.

3.6 Showing information to the user

One convenient method often used in other augmented reality applications to present information to the user comes in form of a panel which displays text and graphics. The panel rotates in all directions such that it always faces the user and hosts the information. Letting an object rotate such that it always faces the user is called 'billboarding' in Microsoft's terminology. The panel's surface is perpendicular to the viewing direction which makes all displayed text clearly readable. This is depicted in figure 6.

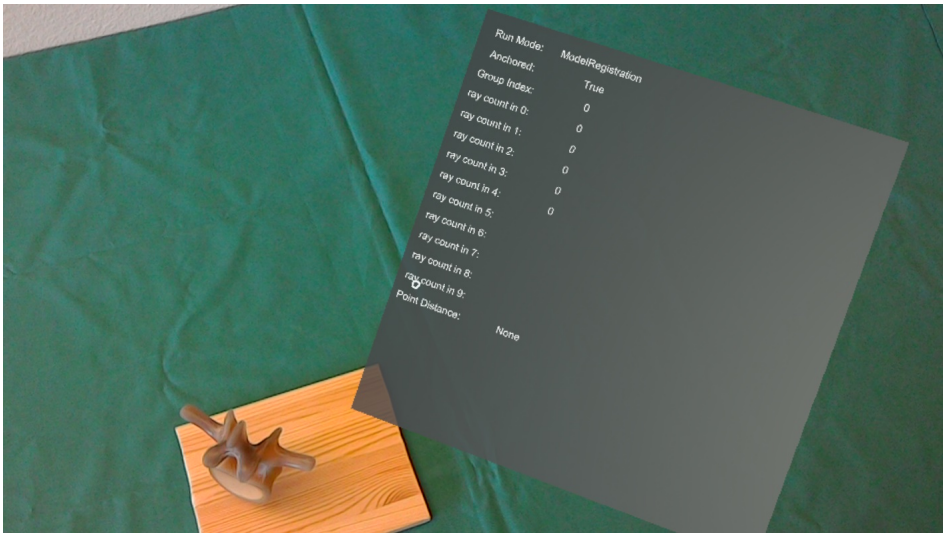

Figure 6: An info panel showing crucial information

3.7 Defining a single point

The first approach to define points that correspond to given landmarks, was to use the spatial reconstruction. For this we intersected the ray defined by the gaze vector and the camera position with the spatial map. The spatial map was accessed through the `GazeManager` object. It automatically performs a `RayCast.Intersect` whenever we look at the spatial map. Quickly we found that the reconstruction has a low fidelity and intersecting this mesh yields points that are not representative for the object at hand. Figure 7 shows a such reconstruction. So we abandoned this approach soon after

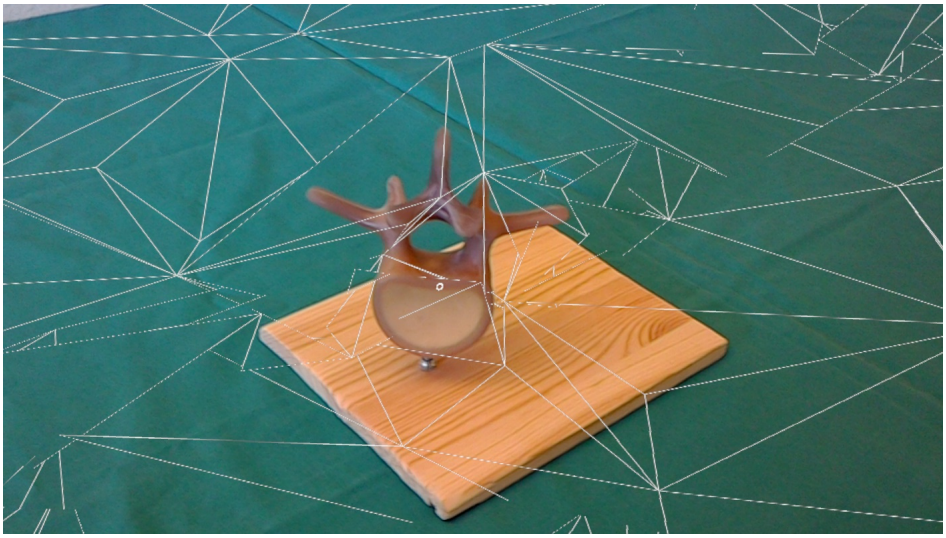

Figure 7: Spatial map of the vertebra 3D print

and went for defining points freely, without any information from the spatial map. Hence one important component of this work is the process of defining an arbitrary point very accurately in our spatial coordinate frame. The goal is to be able to find the coordinates of predefined landmarks on the real world object in the spatial coordinate system and hence establish a correspondence to the real world landmark. This will be important for the registration step in which we use Horn's formulation to find the transform which brings the 3D model into the right spot.

If one thinks of how a point can be defined in terms of other mathematical objects, one quickly thinks of the intersection of multiple lines. In two dimensions every pair of lines that is not parallel must intersect which yields a point. In three dimensional euclidean space two lines must be coplanar and not be parallel in the plane they span to intersect in a point.

The device position and the viewing direction define a ray in 3D space.

Hence whenever the user looks at the target point we get a ray that passes through the target point and originates at the device's location. Repeat this from multiple locations and viewing angles and we have an instance of the fore-mentioned intersection problem. But the key assumption here is that those lines all intersect in one point. This is almost certainly not the case since the user introduces a lot of noise into position and viewing direction and cannot reliably hit the same point from multiple directions. Let alone the fact that even in a controlled setup lines probably wouldn't intersect in the same point numerically.

We can overcome this flaw by using a least-squares solution to the line intersection problem. Following formulations are inspired by and excerpted from this nice write-up [29].
Let $\boldsymbol{a} \in \mathbb{R}^3$ a point (in our case the HoloLens position), $\boldsymbol{n} \in \mathbb{R}^3, \|\boldsymbol{n}\|_2 = 1$ a direction vector (in our case the viewing direction). This defines a line

$$\boldsymbol{l} = \boldsymbol{a} + t \cdot \boldsymbol{n}, \quad -\infty < t < \infty$$

We define the distance between a point and a line by the distance of the point to the point obtained by projecting it orthogonal onto the line. Mathematically we can define a distance function:

$$\begin{aligned} D(\boldsymbol{p}; \boldsymbol{a}, \boldsymbol{n}) &= \|(\boldsymbol{a} - \boldsymbol{p}) - ((\boldsymbol{a} - \boldsymbol{p})^T \boldsymbol{n})\boldsymbol{n}\|_2^2 \\ &= \|(\boldsymbol{a} - \boldsymbol{p}) - \boldsymbol{n}\boldsymbol{n}^T(\boldsymbol{a} - \boldsymbol{p})\|_2^2 \\ &= \|(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)(\boldsymbol{a} - \boldsymbol{p})\|_2^2 \\ &= (\boldsymbol{a} - \boldsymbol{p})^T(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)^T(\boldsymbol{a} - \boldsymbol{p}) \\ &= (\boldsymbol{a} - \boldsymbol{p})^T(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)(\boldsymbol{a} - \boldsymbol{p}) \end{aligned} \tag{3}$$

using the idempotence of the orthogonal projector $(\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}^T)$ in the last equation.

Now we have sampled k such pairs of points $\boldsymbol{a_i}$ and direction vectors $\boldsymbol{n_i}$ for $i = 1, ..., k$.

$$D(\boldsymbol{p}; \boldsymbol{A}, \boldsymbol{N}) = \sum_{i=1}^{k} D(\boldsymbol{p}; \boldsymbol{a_i}, \boldsymbol{n_i}) = \sum_{i=1}^{k} (\boldsymbol{a_i} - \boldsymbol{p})^T(\boldsymbol{I} - \boldsymbol{n_i}\boldsymbol{n_i}^T)(\boldsymbol{a_i} - \boldsymbol{p})$$

So minimizing this function yields the point p with minimal squared distance to all lines. This is expressed by the following constraint on the partial derivative of $D(\boldsymbol{p}; \boldsymbol{A}, \boldsymbol{N})$ with respect to $\boldsymbol{p}$:

$$\frac{\partial D}{\partial \boldsymbol{p}} = \sum_{i=1}^{k} -2(\boldsymbol{I} - \boldsymbol{n_i}\boldsymbol{n_i}^T)(\boldsymbol{a_i} - \boldsymbol{p}) = 0$$

which immediately is transformable to the system

$$\boldsymbol{Rp} = \boldsymbol{q}, \boldsymbol{R} = \sum_{i=1}^{k}(\boldsymbol{I} - \boldsymbol{n_i}\boldsymbol{n_i}^T), \boldsymbol{q} = \sum_{i=1}^{k}(\boldsymbol{I} - \boldsymbol{n_i}\boldsymbol{n_i}^T)\boldsymbol{a_i}$$

We can now calculate our least-squares solution as

$$\boldsymbol{p} = \boldsymbol{R}^{-1}\boldsymbol{q}$$

It's highly probable that the system is regular since this is only not the case if we had parallel lines. In application this expression is computationally affordable to solve directly, since we are working with 3D vectors and at most four pairs of points and direction vectors per point we want to find. Our method is applied as soon as we have at least two lines.

The implementation as a C# method looks as follows:

Listing 1: Least-squares solution to line intersection

```
1  private void CalculateLeastSquaresPoint()
2  {
3          .
4          .
5          .
6
7      Matrix n = new Matrix(3, index);
8
9      for (int i = 0; i < index; i++)
10     {
11         Vector3 direction = rays[i].GetDirection();
12         n[0, i] = direction.x;
13         n[1, i] = direction.y;
14         n[2, i] = direction.z;
15     }
16
17     Matrix R = Matrix.ZeroMatrix(3, 3);
18     Matrix q = Matrix.ZeroMatrix(3, 1);
19
20     Matrix temp;
21     Matrix ni;
22     Matrix ai;
23     Matrix _q;
24     for (int i = 0; i < index; i++)
25     {
```

```
26          temp = Matrix.IdentityMatrix(3, 3);
27          ni = n.GetCol(i);
28          ai = new Matrix(3, 1);
29          Vector3 point = rays[i].GetPoint();
30          ai[0, 0] = point.x;
31          ai[1, 0] = point.y;
32          ai[2, 0] = point.z;
33          temp = temp - ni * Matrix.Transpose(ni);
34          R = R + temp;
35          _q = temp * ai;
36          q = q + _q;
37      }
38
39      Matrix sol = R.SolveWith(q);
40
41      pointLeastSquaresSolution = new Vector3((float)sol[0,
            0], (float)sol[1, 0], (float)sol[2, 0]);
42
43          .
44          .
45          .
46
47  }
```

From the HoloLens we simply read out its position using `Vector3 pos = Camera.main.transform.position`, its viewing direction using `Vector3 dir = Camera.main.transform.forward`, and calculate and save these as point and direction vector of our ray. The rays are visualized as a thin green line segment in space and the solution to the least-squares line intersection is shown as a small yellow sphere, see figure 8. The user can shoot arbitrary rays before he adds the current ray to the set of rays constituting one point. By saying 'Add Ray' this is achieved. One can delete points, rays, switch between point groups, etc. by voice command.

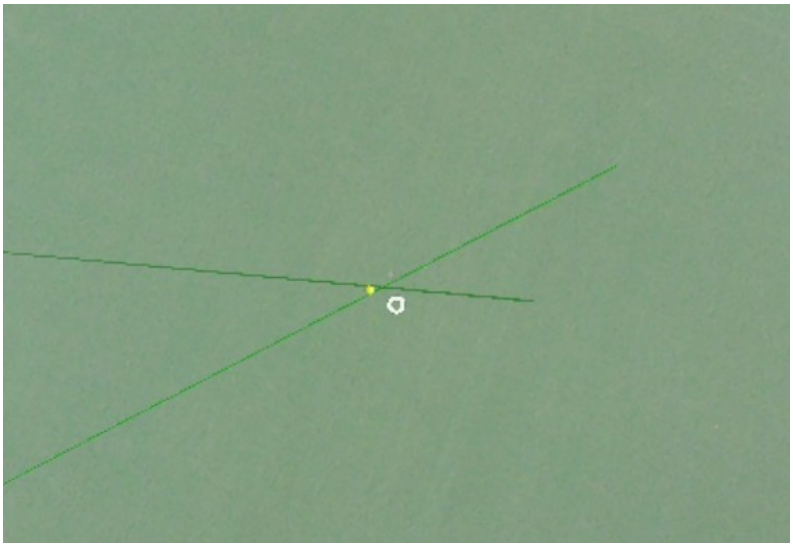

Figure 8: Visualizing rays (green) and the least-squares solution (yellow point)

3.8 Overlaying the model

At this point we have defined a point one after another, each time given a landmark to which it should correspond. Hence we have a known point correspondence between landmark points and user defined points. We now want to overlay the 3D model using its predefined landmark points, we call them the 'source point cloud', over the real world object which is represented by the user-defined points, which we will call 'the target point cloud'. The class `HornPointPair` is defined to hold such a corresponding pair of `Vector3` points. In section 2.2 we've explained how one can derive the notion of a best solution to this problem by defining an error function and how we then may obtain the actual rotation and translation by minimizing it. A base implementation of Horn's algorithm was supplied by Balgrist CARD. The function is normally part of an internal library and was adopted and partially re-written to support Unity's data types. We named it `HornAbsoluteOrientation` and the its code can be found in Appendix A. We defined classes `HornInput` and `HornOutput` which encapsulate arguments and result of the C# method `HornAbsoluteOrientation`. `HornInput` is a list of `HornPointPair` objects plus some convenience functions. As of now, this class is obsolete but at the beginning it contained more functionality than a standard list. This is all we need to solve the absolute orientation problem. Since Horn's solution is expressed as rota-

tion about the centroid of the source point cloud, we must be careful not to rotate the 3D model about the origin of the local 3D model coordinate system. Therefore a `HornOutput` object consists of the source centroid represented as a `Vector3`, the rotational `Quaternion` and a `Vector3` translation. We are free to rotate an object in Unity around any point and axis using `gameObject.transform.RotateAround(centerOfRotation, axis, angle)`, where `centerOfRotation` and `axis` are of type `Vector3` and `angle` is a `float`. This rotation is done inside the 3D models `Update` method and it is guarded by a boolean variable that indicates if we have a new result from `HornAbsoluteOrientation`. We solve absolute orientation whenever the user gives the command 'Fit Model' and check beforehand if there are at least three point pairs. The solution is then handed to the object holding the 3D model and the boolean flag is set to indicate that a new rotation and translation can be performed on the 3D model. We made the assumption that the 3D model was defined using a high quality scan, in our case of a real vertebra. Hence the model being defined in a coordinate system which matches the real world scale, and working within the spatial coordinate system of the HoloLens, we can assume that the target and the 3D model already match in scale. If not, this could be easily ensured during the scanning process. The result after overlaying the model can be seen in figure 9.

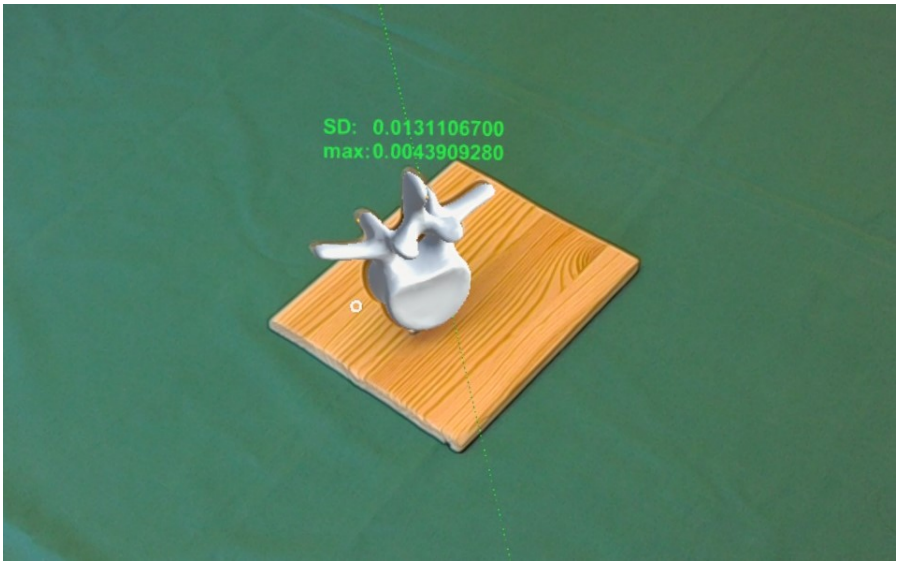

Figure 9: Model was overlayed using point cloud registration

3.9 Visual feedback

One of the main advantages of augmented reality applications is the ability to show information with visual context where they really belong and where they are of greater meaning to the user, i.e. right next to the real world object you are interacting with. We used a simple but effective concept of rotating 3D models towards the user's position. This is called 'billboarding' in Microsofts augmented reality terminology. On top of the vertebra model we placed a 3D text model which is always facing the user and displays information of interest expressing the distance between the virtual target and the 3D model we want to overlay. These are the sum of all point pair distances and the largest distance between a single point pair. We mentioned 'billboarding' before when we showed the info panel in section 3.6. See figure 21 in section 4.3.

Another visual cue is the cursor which is at fixed distance in viewing direction at a point 40 cm away from the user. The cursor is actually a small donut shaped 3D object rendered in white color. Its center opening serves as our aiming sight. An issue with this approach was that it is very difficult to hit a point if the cursor has another focal distance as the object your are looking at, e.g. the cursor is at 40 cm distance but the target is at 55 cm distance. When this is the case, one cannot align the cursor opening with the target on both eyes. Trying to do so using only one eye and one of the stereo images results in rays that miss the target point by far. This problem is severe when the cursor has a self adjusting distance, e.g. when it is rendered wherever the gaze hits the spatial map. By fixing the cursor's distance the user is able to move relative to the target, such that cursor distance and target distance match up and aiming with both eyes becomes easier again. The first approach to solve this was to disable one display and align the ray with the center of the display that is still running. We couldn't find a way to display the cursor only on one side, and I couldn't find the correct transform that shifts the origin of the ray towards the physical location of the eye behind the working display.

For shooting rays we display the angle between the direction vector of other rays and the current viewing direction from which a freshly added ray is derived. This helps to quickly guide the user towards shooting rays that form nearly orthogonal angles. For this see figure 10 in section 4.1.1.

4 Evaluation

To evaluate our method we came up with a few experiments. First of all we want to measure if the precision of the ray casting method described in section 3.7 increases when increasing the angles between rays. Additionally it shows how big the error distance to a target is for the first time. For this we used a virtual target. All measurements of this experiment were done in the spatial coordinate system. The second experiment shows how accurate one can be relative to a real world target. For this we constructed a setup that allows for controlled measurements of the scene using the front-facing camera. The last experiment measures the time needed to complete the task of finding five landmarks and overlaying a corresponding 3D model over another identical 3D model which is placed as a virtual target.

4.1 Precision and dependence on angles of our method

4.1.1 Description

From intuition we suppose that rays coming from almost orthogonal directions will yield the points with the least error distance to the actual target compared to rays with a small angle between them.

To test this we shot rays through a virtual target point which was represented by the 3D model of a sphere with a diameter of 2 mm with the target point as its center. The diameter of 2 mm is somewhat arbitrary and was chosen after trial and error. A bigger size makes it more difficult to actually hit the center since one has to estimate if a ray that pierces the sphere would go through the center or not. Still the size is big enough to be clearly visible. We chose a virtual target over a real target because the effects of degrading tracking quality or sudden changes in the coordinate system aren't impacting the experiment negatively if we let the underlying system relocate the target together with all other objects. The target was placed in a open space such that it is possible to move around it freely and especially bend over it with ease. Additionally we displayed the angle between rays for the user to be able to shoot rays with a desired angle between them. The code which achieves this can be seen in listing **??** in Appendix B and a screen shot of the scene can be seen in figure 10.

For any ray we used as many retries as needed to hit the target close to its center and to establish the prescribed angles between the rays. A ray is then finally accepted and added by the voice command "Add ray" and the user may proceed with placing the next ray. The rational behind this, is to exclude outliers which naturally occurr when trying to hit a target. Main

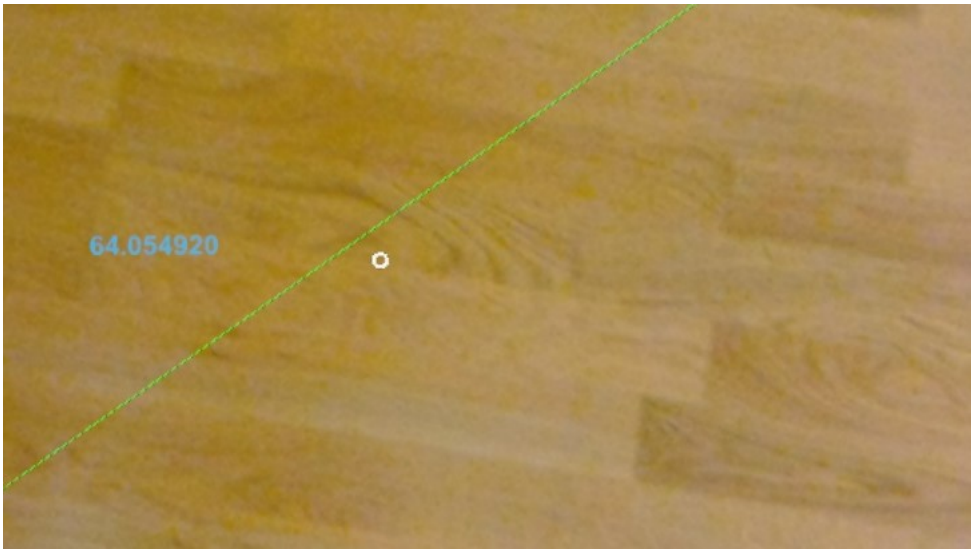

Figure 10: A green ray, in blue font the angle between it and the current viewing direction

reasons for those outliers are sudden head movements or uncomfortable head positions as well as blunt mistakes when operating the app. We refrained from incorporating any additional feedback besides showing the angles between rays, e.g. indicating a hit by showing whether the ray intersects the small sphere or not. The sphere should merely act as a fake target and it would be unrealistic for a real target to indicate a hit. Working with a real world target, the user has to judge by itself if the ray is passing through the target accurately and we want the same behaviour in this experiment. After all rays are placed as prescribed by the experiment, we calculate the least-squares solution using the method described in section 3.7. Finally the distance of the solution to the target center is measured. The first part of the experiment was conducted using two rays at angles of 30, 45, 60 and 90 degrees. The experiment was repeated five times for every angle configuration. The device was calibrated and the surroundings thoroughly scanned before the actual experiment.

4.1.2 Results

Table 1 shows the measurement results, the mean and standard deviation per set of five measurements with equal angle restriction is calculated.

Figure 11 shows the mean and standard deviation of every measurement set plotted against angle.

From a set of measurements with three rays, see table 2, we couldn't see

Angle [°]	d [mm]	μ	σ
30	2.722590 2.667547 2.784677 1.101444 1.095197	2.074291	0.891900
45	1.283587 1.453487 1.240328 0.428739 0.840534	1.049335	0.413349
60	0.840232 0.320893 0.522181 0.791713 0.374047	0.569813	0.237129
90	0.583310 0.428525 0.567517 0.285552 0.253294	0.423639	0.153546

Table 1: Measurements with 2 lines per point

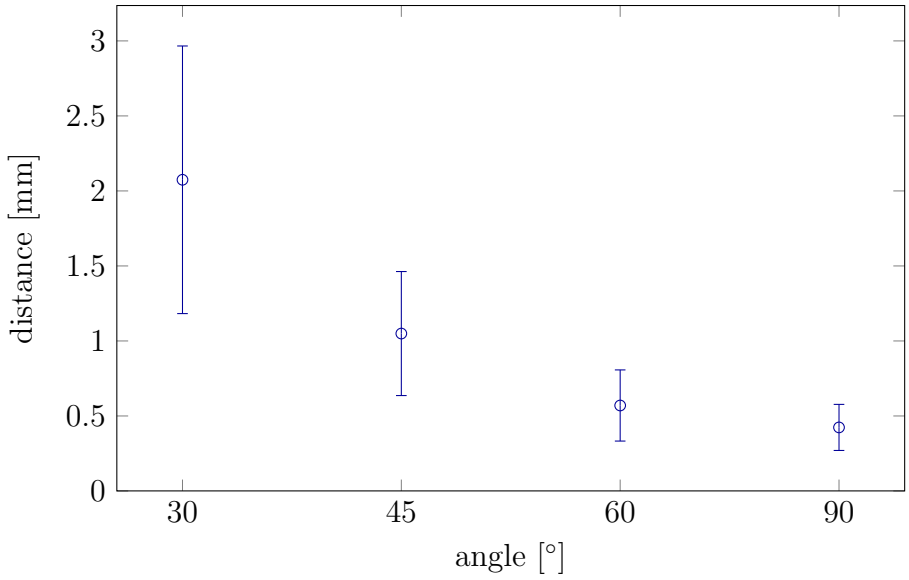

Figure 11: Mean and standard deviation

Angle [°]	d [mm]	μ	σ
90	0.509633 0.296323 0.329511 0.129577 0.881475	0.429304	0.286527

Table 2: Measurements with 3 lines per point

any improvements. The mean is about the same, the standard deviation is a bit higher than the results with two rays and 60° or 90° angles.

It is worthy to note that we can achieve sub-millimeter precision using our method. We can also clearly see how the angle affects the error distance noticeably. The bigger the angle becomes, the more precisely the points are defined. We conclude that one should at least have 60° angles between rays, since from there on improvements are small. Going from 45° to 60° nearly halves the error distance, but going from 60° to 90° only leads to a minor improvement. Additionally we should remark that it was the most difficult to comply to the 90° constraint, since one has to be able to bend over a lot and view the scene from more extreme angles. This might not always be possible.

4.2 Single point accuracy

4.2.1 Description

In this experiment we want to measure how accurate a virtual point is compared to the a real word target. The distance between the real world target and the virtual point is measured relative to a real world scale by capturing the scene through the HoloLens' front-facing camera. The target is a corner of a 3D printed cube with an edge length of 100mm. For this experiment we've built a small measure box which has a checker pattern on the inside on two adjacent sides. The box and the target object can be seen in figure 12. For the checker pattern we used a 1mm graph paper that was found online [30]. The box was carefully crafted to ensure that it doesn't introduce errors when measuring distances in the post processing step of the experiment. Therefore we also used a machinist square to ensure 90° angles. Inside the box we placed the cube justified with the lower inside corner of the box. The top side and the sides opposite to the scales are open such that we can capture the scene in the box. The main purpose of the checker pattern is to make it easier to setup the HoloLens' for capture by aiding with aligning the camera's viewing direction exactly with an edge of the cube. One can verify a good alignment by checking that the edges which are perpendicular to the viewing direction of the camera are perfectly aligned with the checker pattern. Another reason for the checker pattern is the ability to detect and account for lens distortions in post processing by calibrating the camera, if the need for this arises. Additionally the edges of the cube were marked with black color to make them visually more distinguishable from the rest of the cube in a well lit environment. This is on the one hand important for the fore mentioned setup and alignment of the device for capturing but on the other hand the target corner is merely visible when aiming if left plain white. The blackened edges also help in post processing, since we use the knowledge that

For the measurements the scales I used in ImageJ were 3.0368 pixels/mm, 3.1010 pixels/mm and 2.8833 pixels/mm for front, side and top view respectively.

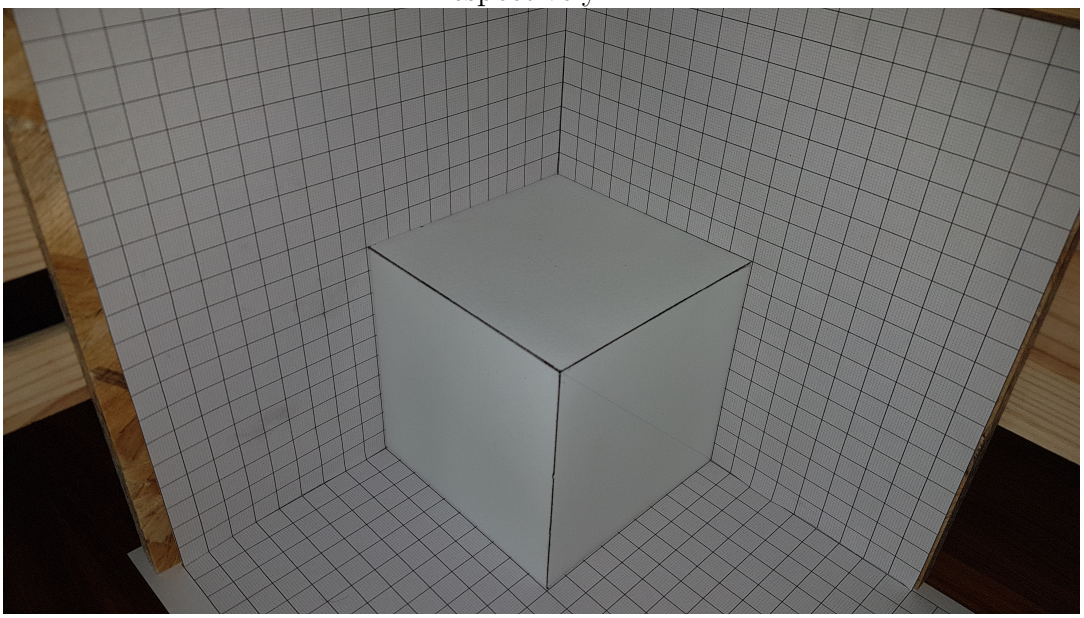

Figure 12: The 3D printed cube justified inside the measure box

the edge is 100mm in length to set the scales for measuring distances in the captures. For capturing we've also 3D printed the spectator view mounting rack using the STL files provided by Microsoft [31], assembled it using standard M4 and M6 bolts and nuts, see figure 13 and finally screwed it on top of a professional tripod for attaching the HoloLens to it, see figure 14. This ensures stability throughout the captures. We did ten measurements using three rays each with angles between 60° and 90°. It turned out to be very difficult to conduct this experiment. When the device is fit into the mounting rack, it is not possible to completely look through the stereo displays and aim correctly. So the idea was to use the live video feed which can be accessed through the web interface of the HoloLens. The problem with this approach is, that the frame rate of the application drops to 30 frames per second. This leads to unstable holograms and this would introduce a source of error which is not realistic for a normal usage of the device [27][26]. We ended up writing some code which made it possible to make multiple measurements and then display them one after another for capture. Again this is done by voice command. Before the experiment was done, the HoloLens was again calibrated and the room was thoroughly scanned after the application started. How a capture looks like, can be seen in figure 19. The final step of the experiment

Figure 13: The HoloLens is positioned for capture using the mounting rack and a tripod

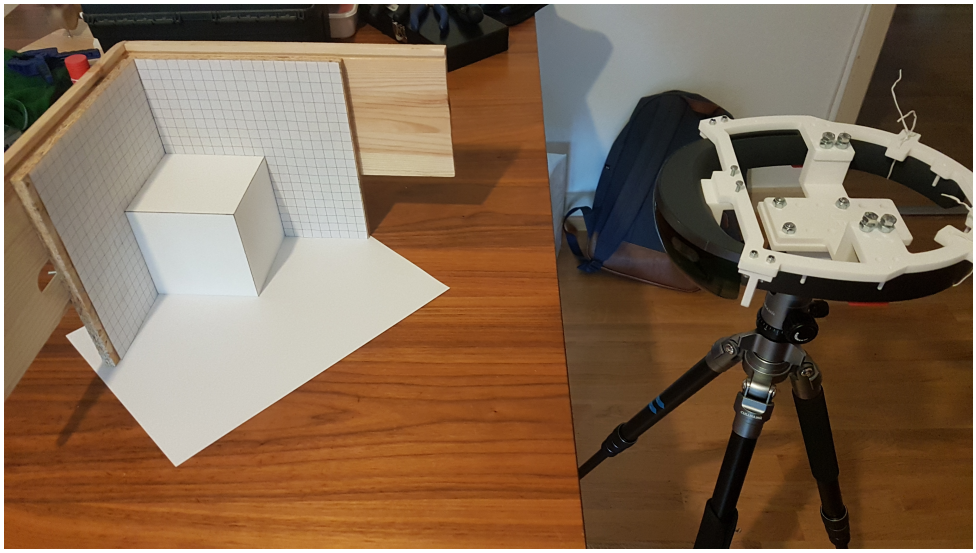

Figure 14: The 3D printed cube justified inside the measure box

View	(μ_x, μ_y) [(cm,cm)]	(σ_x, σ_y) [(cm,cm)]
Front	(0.1768, 0.7038)	(0.1463, 0.3363)
Side	(0.6469, 0.2032)	(0.1246, 0.2285)
Top	(0.6060, 0.6242)	(0.1254, 0.1103)

Table 3: Mean and standard deviation of each of the three measurement sets

was the act of measuring the real world distances between the actual target corner and where the virtual point is perceived. This is done using ImageJ [32], a program written in Java for image processing and analysis. By first measuring the length of one of the cube's edges that is nearest to the camera, ImageJ conveniently sets a pixels per mm scale which we can then use for following measurements. For the measurements the scales I used in ImageJ were 3.0368 pixels/mm, 3.1010 pixels/mm and 2.8833 pixels/mm for front, side and top view respectively. Of course one should not use the scales of the checker pattern for setting the scale, since the point is roughly 10cm away from it. Then we always measured the distance of the point to the cube's corner in two perpendicular directions which are given by the visible edges that join at the corner.

4.2.2 Results

Some minor coordinate system shifts and jitters occurred during the ray casting stage of the experiment between point definitions. The shifts were subjectively conceivable but we weren't prepared and able to measure those. Nevertheless the shifts were small and the set of points still seemed to be closely around the target corner. So we continued the experiment ignoring this observation, which still occurred after some retries. A severe issue was that after re-positioning the HoloLens for capture, the virtual points had shifted by a much bigger amount. This could be confirmed by peeking through the stereo displays without using the live video feed provided on the device portal web interface. This is the final situation which we measured. It is known that enabling live video feed degrades the tracking performance because it caps the frame rate of the device at 30 frames per second [27]. We made sure that the live feed was enabled to check the position only when the device was slightly moved. Figures 15, 16 and 17 show the measurements made using the captures from three different angles. The figures 18, 19 and 20 show the corresponding captures which were used. These are the first captures of each set of ten captures. Table 3 show the means and standard deviations of the three measurement sets.

First of all, the points are all close together, i.e. the standard deviation

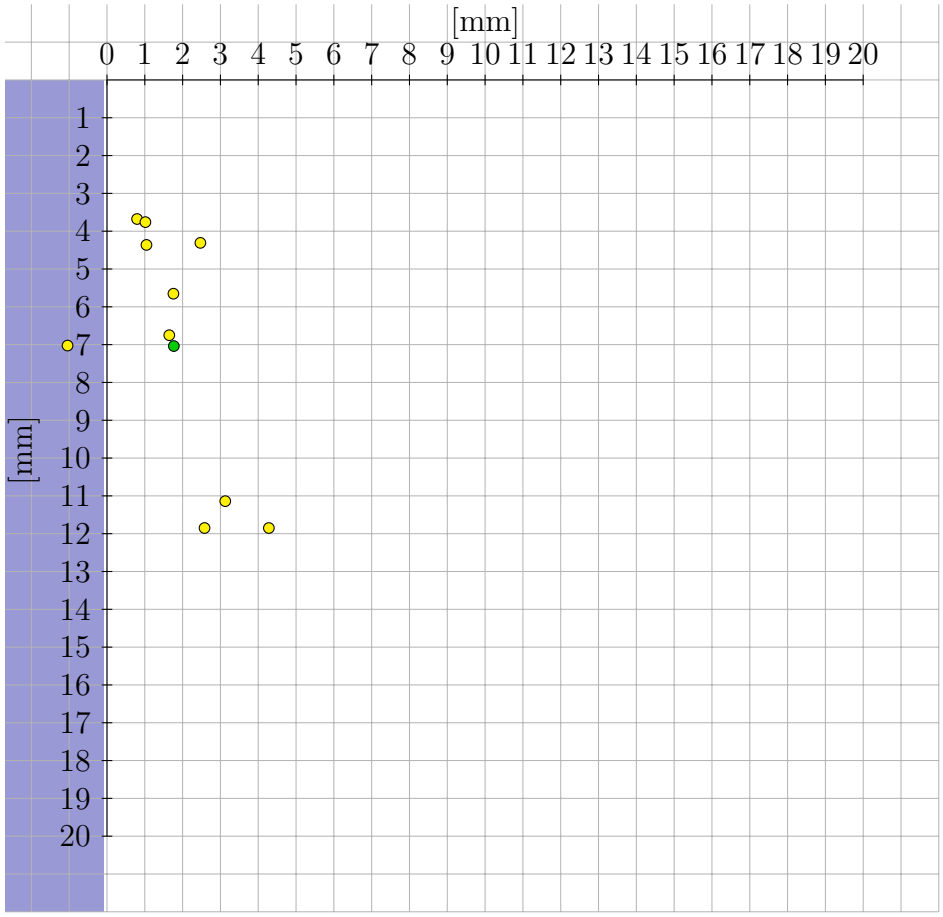

Figure 15: Measurements of front view, measurements in yellow, mean in green

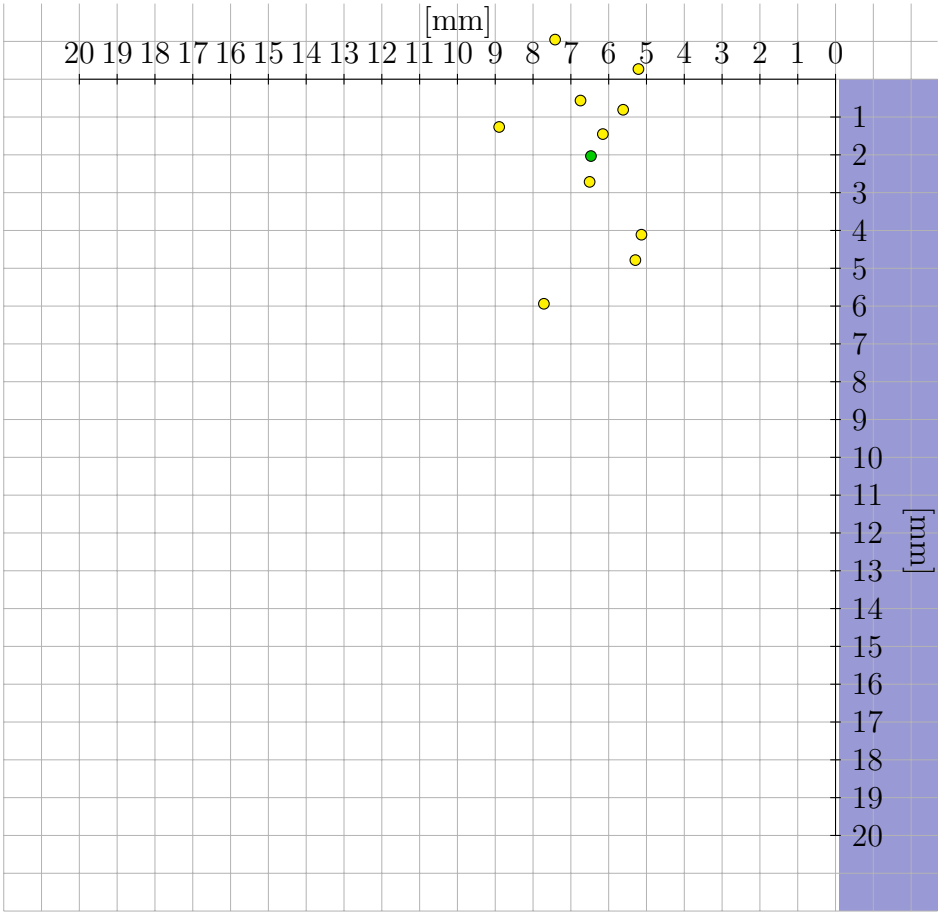

Figure 16: Measurements, side view

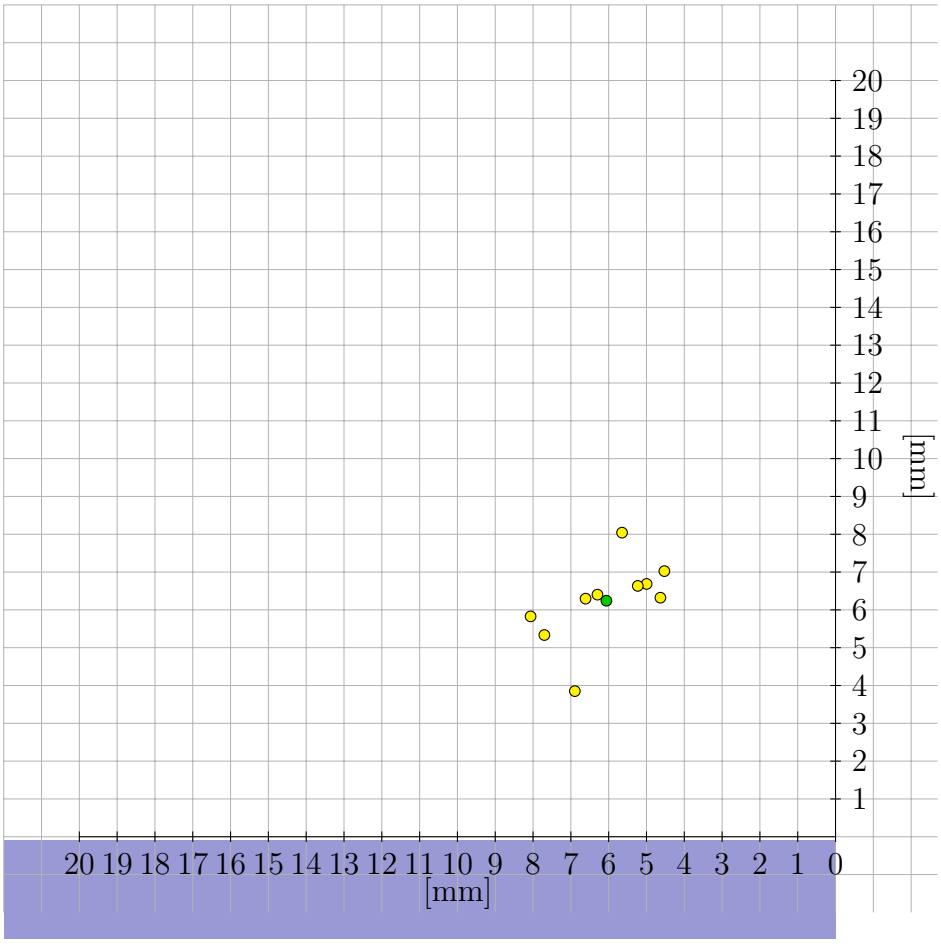

Figure 17: Measurements, top view

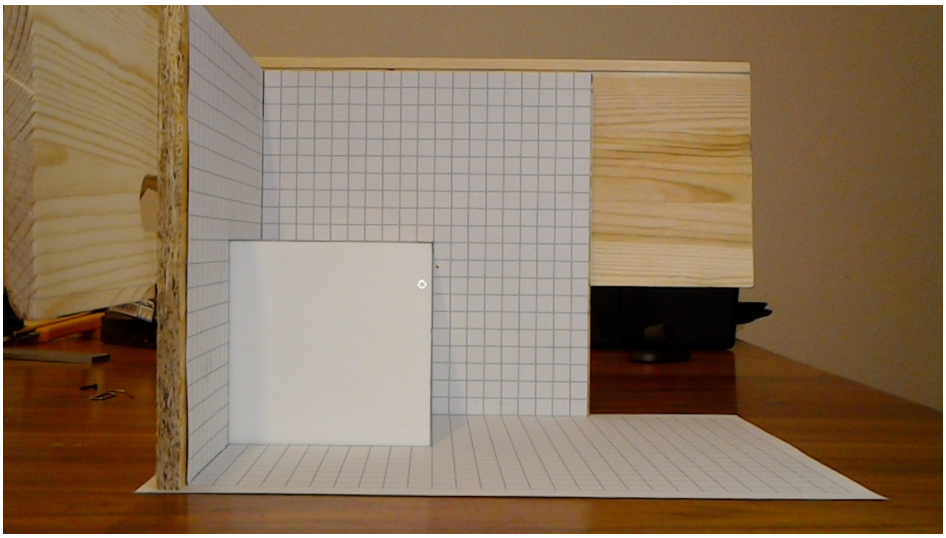

Figure 18: Capture, front view

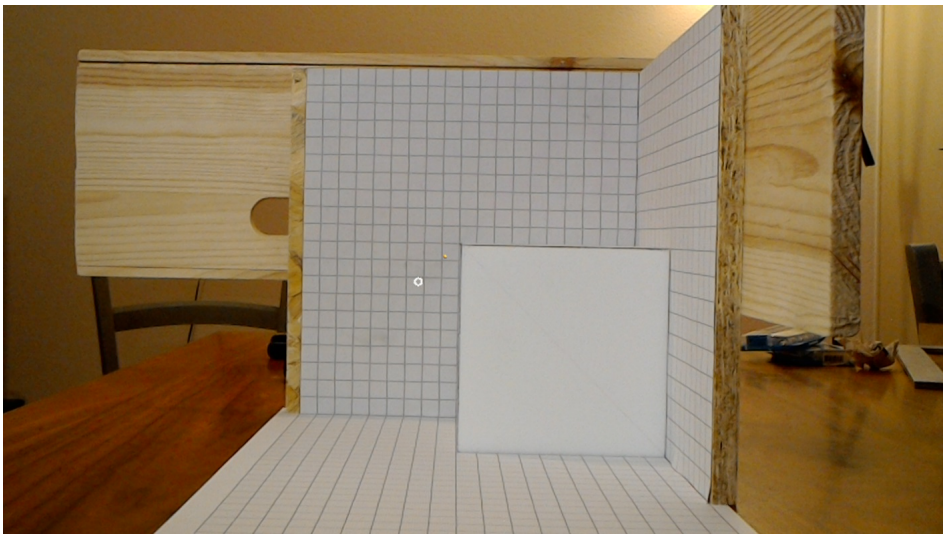

Figure 19: Capture, side view

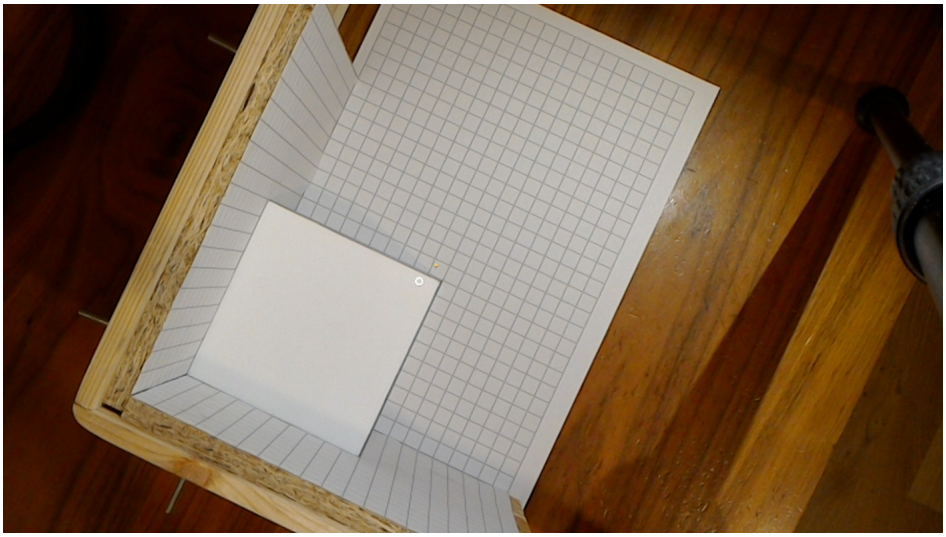

Figure 20: Capture, top view

is small, which indicates high precision of the ray casting method. But as we can see, the mean point is way off the actual target (Again see table 3). This shows the inaccurate spatial mapping in this case.

I refrained from doing more experiments of the same kind since this single set of measurements already makes it clear that accuracy is not guaranteed in this simple setup. Furthermore in this experiment we are able to measure with fairly high accuracy how far the point is off the target, but we didn't have the means to control all the interesting factors that could influence the hologram stability. Hence we are not in a position to make any statements about how or why exactly this inaccuracy arose in our experiment. Without knowledge of what actually caused this offset, one set of measurements is still enough to show that a sub-millimeter accuracy is not guaranteed even though a complete scan of the room was performed. Our guess so far is, that lighting and the amount of good visual features in the surroundings have the biggest influence on hologram stability. These external parameter were not controlled in this experiment. Assuming that the HoloLens works with some kind of SLAM which relies on SIFT features or similar methods that rely on visual landmarks, one can imagine that a lack of contrast and distinctive visual cues are affecting the device's accuracy. A candidate for causing disruptive hologram stability is also the amount of reflective surfaces and materials that are transparent to IR light. As the depth sensor works with IR light, windows are causing holes in the spatial map. The light of course just passes through the window glass. It also cannot be ruled out, that activating the live video feed for fine grained positioning of the device

had an influence.

4.3 Timing a simple navigation task

4.3.1 Description

This experiment shows how much time is needed for completing the task of defining five points according to five given landmarks and fit the white 3D model over an exact copy of the same model in red which acts a target. The target has a corresponding set of landmarks defined. These landmark pairs are actually used for calculating the error distances and are predefined. Of course the model fitting is done using the user defined points. In preparation for the experiment a point for the spatial anchor is chosen by gazing at the spatial map of the working area, then the info panel, the 3D model with the landmarks and the red target are placed. The timer starts after the preparation as soon as the user is ready. This is again done by voice command. The timer ends after all landmarks were found and the voice command for fitting the model is given. It was subjectively decided whether the points are defined well or not, since we don't know the actual distance of a virtual point to a real world target when using this method on a real world object. So we didn't use the information of distance while defining the points, even though it would be available. Finally we measured the summed distance of all point pairs, the maximal distance among all point pairs and of course the time. In figure 21 we can see how the user sees the scene after the fit. For comparison we measured the same metrics for a simplistic interaction method using the manipulation gesture to drag and rotate the vertebra. The code for this was derived from an online source [33]. This base line experiment had an time limit of 150 seconds, because one could take way longer until a relatively good fit is achieved. Of course this is a super simplistic method which could be further improved. Our method was evaluated first, so the time limit for the base line experiment was set at one of the larger time values we got from the experiment involving our method. These experiments were not designed to show a dependency of the error distances on the time taken for task completion. For each method we did six measurements.

4.3.2 Results

As one can see from tables 4, 5, 6 and 7, our method is significantly more precise compared to standard gestures used in holographic applications.

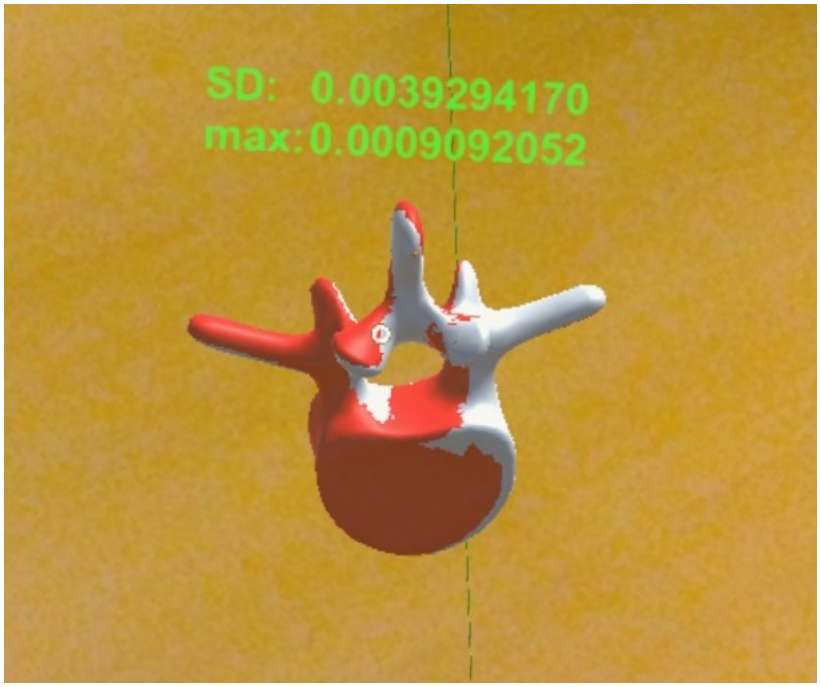

Figure 21: The white vertebra is fitted onto the red target vertebra. The summed distances of all point pairs and the maximum distance between any point pair is displayed

Time [sec]	Summed distance [mm]	Maximal point pair distance [mm]
150	79.557	28.151
150	41.1872	13.9122
150	39.4954	9.1234
150	21.0492	5.4344
150	35.5707	8.2462
150	39.7712	14.6332

Table 4: Drag and rotate using gestures

Time [sec]	Summed distance [mm]	Maximal point pair distance [mm]
154.86	2.7207	0.7710
105.64	2.4072	0.7824
116.58	3.8122	1.2730
161.59	2.9345	0.7879
127.29	3.1683	0.8787
137.44	4.4431	1.2898

Table 5: Our method

SD [mm] μ	SD [mm] σ	Max [mm] μ	Max [mm] σ
42.7718	19.4891	13.2501	8.0937

Table 6: Mean and standard deviation of the drag and rotate method

SD [mm] μ	SD [mm] σ	Max [mm] μ	Max [mm] σ
3.2477	0.75312	0.8631	0.4083

Table 7: Mean and standard deviation of the drag and rotate method

5 Conclusions

We could show that our method is precise assuming static targets. Points can reliably be defined with a distance smaller than a millimeter from the target. The accuracy relative to real world targets is rather big. This renders it unusable with the HoloLens for medical appliances as is, but I'm confident that this accuracy gets better when we see more sophisticated augmented-reality devices. A point cloud that consists of such points can be used as the target point cloud for registration which yields a good and precise fit of two models.

6 Future work

It was very difficult to measure the accuracy of our method, and it seems to be difficult in general. There are a lot of variables influencing the device's capability of maintaining a stable transform between its own coordinate system and the world around it. It is still left to find out what those main influences are and which one has the most impact. There is a lot more that could be tested in an experimental setting, especially concerning user experience of our method. Additionally there is a lot that can be done after a 3D model was registered. This is merely a stepping stone for basic surgical navigation and further visualizations, educational applications etc. can be built on top of it.

Appendices

A

Implementation of Horn's closed-form solution

Listing 2: Horn's method

```
1  private HornOutput HornAbsoluteOrientation(HornInput input
       )
2      {
3
4          //Use the technique desribed by Berthold K.P. Horn
               "Closed-form solution of absolute orientation
               using unit quaternions"
5
6
7          Vector3 sourceCentroid = new Vector3();
8          Vector3 targetCentroid = new Vector3();
9          int size = input.GetSize();
10         Vector3[] sourcePoints = new Vector3[size];
11         Vector3[] targetPoints = new Vector3[size];
12
13         for (int i = 0; i < size; i++)
14         {
15             HornPointPair pp = input.GetPointPair(i);
16             sourcePoints[i] = pp.source;
17             targetPoints[i] = pp.target;
18             sourceCentroid += pp.source;
19             targetCentroid += pp.target;
20         }
21
22         sourceCentroid /= size;
23         targetCentroid /= size;
24
25         //Check if only one point, if so, stop
26         if (sourcePoints.Length == 1)
27         {
28             Quaternion rot = new Quaternion();
29             rot.eulerAngles = new Vector3(0, 0, 0);
30             Vector3 trans = targetCentroid - sourceCentroid
                   ;
31             return new HornOutput(sourceCentroid, rot,
                   trans);
32         }
```

```
33
34          //Now define a matrix M
35          var m = new double[3, 3];
36          for (int i = 0; i < size; i++)
37          {
38              Vector3 source = sourcePoints[i] -
                    sourceCentroid;
39              Vector3 target = targetPoints[i] -
                    targetCentroid;
40              for (int k = 0; k < 3; k++)
41                  for (int l = 0; l < 3; l++)
42                      m[k, l] += source[k] * target[l];
43          }
44
45          //Then define matrix N
46          var n = new double[4, 4];
47          n[0, 0] = +m[0, 0] + m[1, 1] + m[2, 2];
48          n[1, 1] = +m[0, 0] - m[1, 1] - m[2, 2];
49          n[2, 2] = -m[0, 0] + m[1, 1] - m[2, 2];
50          n[3, 3] = -m[0, 0] - m[1, 1] + m[2, 2];
51          n[0, 1] = (n[1, 0] = m[1, 2] - m[2, 1]);
52          n[0, 2] = (n[2, 0] = m[2, 0] - m[0, 2]);
53          n[0, 3] = (n[3, 0] = m[0, 1] - m[1, 0]);
54          n[1, 2] = (n[2, 1] = m[0, 1] + m[1, 0]);
55          n[1, 3] = (n[3, 1] = m[2, 0] + m[0, 2]);
56          n[2, 3] = (n[3, 2] = m[1, 2] + m[2, 1]);
57
58          //Compute the eigenvalues of N
59          var eigenvalues = new double[4];
60          var eigenvectors = new double[4, 4];
61          if (!alglib.smatrixevd(n, 4, 1, true, out
                eigenvalues, out eigenvectors))
62          {
63              Debug.Log("No EVD");
64              throw new Exception("Cannot find eigenvalues");
65          }
66
67          //Compute the quaternion. This depends on weather
                the stuff is colinear
68          var quaternion = new double[4];
69          if (Math.Abs(eigenvalues[0] - eigenvalues[1]) <
                double.Epsilon || size == 2)
70          {
71              //Find a point that is not identical to the
                    first point
72              var indexDiffers = -1;
73              for (var i = 1; i < size; i++)
74              {
```

```
75                  if (Math.Pow(Vector3.Distance(sourcePoints[
                        i], sourcePoints[0]), 2) < double.
                        Epsilon)
76                      continue;
77                  if (Math.Pow(Vector3.Distance(targetPoints[
                        i], targetPoints[0]), 2) < double.
                        Epsilon)
78                      continue;
79                  indexDiffers = i;
80                  break;
81              }
82              Debug.Log("We found indexDiffers = " +
                    indexDiffers.ToString());
83              if (indexDiffers == -1)
84              {
85                  Debug.LogError("indexDiffers = -1");
86                  throw new Exception("All points are equal")
                        ;
87              }
88
89              //This is colinear
90              var ds = Vector3.Normalize(sourcePoints[
                    indexDiffers] - sourcePoints[0]);
91              var dt = Vector3.Normalize(targetPoints[
                    indexDiffers] - targetPoints[0]);
92
93
94              // take dot & cross product
95              var cross = Vector3.Cross(ds, dt);
96              var w = Vector3.Dot(ds, dt);
97              double r = Vector3.Magnitude(cross);
98              var theta = Math.Atan2(r, w);
99
100             //Construct quaternion
101             quaternion[0] = Math.Cos(theta / 2);
102             if (Math.Abs(r) > double.Epsilon)
103             {
104                 r = Math.Sin(theta / 2) / 2;
105                 quaternion[1] = cross[0] * r;
106                 quaternion[2] = cross[1] * r;
107                 quaternion[3] = cross[2] * r;
108             }
109             else
110             {
111                 Debug.LogError("Abs(r) <= double.Epsilon...
                        have to check this");
112                 double[] dtA = { dt.x, dt.y, dt.z };
113                 double[] dsA = { ds.x, ds.y, ds.z };
114                 double[] nil = null;
```

```
115                 Perpendicular(dsA, ref dtA, ref nil);
116                 r = Math.Sin(theta / 2);
117                 quaternion[1] = dt[0] * r;
118                 quaternion[2] = dt[1] * r;
119                 quaternion[3] = dt[2] * r;
120             }
121         }
122         else
123         {
124             //This is only in the case when the points are
                    not coplanar
125             quaternion[0] = eigenvectors[0, 3];
126             quaternion[1] = eigenvectors[1, 3];
127             quaternion[2] = eigenvectors[2, 3];
128             quaternion[3] = eigenvectors[3, 3];
129         }
130
131         return new HornOutput(
132             sourceCentroid,
133             new Quaternion((float)quaternion[1], (float)
                    quaternion[2], (float)quaternion[3], (float)
                    quaternion[0]),
134             new Vector3(targetCentroid[0] - sourceCentroid
                    [0], targetCentroid[1] - sourceCentroid[1],
                    targetCentroid[2] - sourceCentroid[2])
135             );
136
137     }
```

B

Showing angles between rays

Listing 3: Displaying angles between rays

```
1 public void ShowAimingAngle(Vector3 newDirectionVector)
2         {
3             Vector3 point =
                  CalculateLeastSquaresPointAiming(
                  newDirectionVector);
4             if (index == 1)
5             {
6                 Vector3 dir0 = rays[0].GetDirection();
7                 float angle = Vector3.Angle(dir0,
                      newDirectionVector);
8                 //Vector3 position = point - dir0 * 0.05f;
9                 Vector3 position = Camera.main.transform.
                      position + Camera.main.transform.forward
```

```
                    .normalized * 0.5f + Vector3.Cross(
                    Camera.main.transform.forward, Camera.
                    main.transform.up).normalized * 0.05f;
10                  textAngleAiming[0].transform.position =
                    position;
11                  textAngleAiming[0].GetComponent<TextMesh>()
                    .text = angle.ToString("F6");
12                  textAngleAiming[0].SetActive(true);
13              }if(index == 2)
14              {
15                  Vector3 dir0 = rays[0].GetDirection();
16                  float angle0 = Vector3.Angle(dir0,
                    newDirectionVector);
17                  //Vector3 position0 = point - dir0 * 0.05f;
18                  Vector3 position0 = Camera.main.transform.
                    position + Camera.main.transform.forward
                    .normalized * 0.5f + Vector3.Cross(
                    Camera.main.transform.forward, Camera.
                    main.transform.up).normalized * 0.05f;
19                  textAngleAiming[0].transform.position =
                    position0;
20                  textAngleAiming[0].GetComponent<TextMesh>()
                    .text = angle0.ToString("F6");
21                  textAngleAiming[0].SetActive(true);
22
23                  Vector3 dir1 = rays[1].GetDirection();
24                  float angle1 = Vector3.Angle(dir1,
                    newDirectionVector);
25                  //Vector3 position1 = point - dir1 * 0.05f;
26                  Vector3 position1 = Camera.main.transform.
                    position + Camera.main.transform.forward
                    .normalized * 0.5f + Vector3.Cross(
                    Camera.main.transform.forward, Camera.
                    main.transform.up).normalized * -0.05f;
27                  textAngleAiming[1].transform.position =
                    position1;
28                  textAngleAiming[1].GetComponent<TextMesh>()
                    .text = angle1.ToString("F6");
29                  textAngleAiming[1].SetActive(true);
30              }
31
32          }
```

Listings

List of Figures

List of Tables

References

[1] Ivan E. Sutherland. "A Head-mounted Three Dimensional Display". In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). San Francisco, California: ACM, 1968, pp. 757–764. DOI: 10.1145/1476589.1476686. URL: http://doi.acm.org/10.1145/1476589.1476686.

[2] S. Vogt et al. "A high performance AR system for medical applications". In: *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.* Oct. 2003, pp. 270–271. DOI: 10.1109/ISMAR.2003.1240715.

[3] Matthias Harders, Gerald Bianchi, and Benjamin Knoerlein. "Multimodal Augmented Reality in Medicine". In: *Universal Access in Human-Computer Interaction. Ambient Interaction: 4th International Conference on Universal Access in Human-Computer Interaction, UAHCI 2007 Held as Part of HCI International 2007 Beijing, China, July 22-27, 2007 Proceedings, Part II*. Ed. by Constantine Stephanidis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 652–658. ISBN: 978-3-540-73281-5. DOI: 10.1007/978-3-540-73281-5_70. URL: https://doi.org/10.1007/978-3-540-73281-5_70.

[4] Joerg Traub et al. "Towards a Hybrid Navigation Interface: Comparison of a Slice Based Navigation System with In-situ Visualization". In: *Proceedings of MIAR 2006*. LNCS. Shanghai, China: Springer, Aug. 2006, pp. 364–372.

[5] Adrian Elmi-Terander et al. "Surgical Navigation Technology Based on Augmented Reality and Integrated 3D Intraoperative Imaging: A Spine Cadaveric Feasibility and Accuracy Study". In: *Spine (Phila Pa 1976)* 41.21 (Nov. 2016). 27513166[pmid], E1303–E1311. ISSN: 0362-2436. DOI: 10.1097/BRS.0000000000001830. URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5113235/.

[6] Longfei Ma et al. "Augmented reality surgical navigation with ultrasound-assisted registration for pedicle screw placement: a pilot study". In: *International Journal of Computer Assisted Radiology and Surgery* (Aug. 2017). ISSN: 1861-6429. DOI: 10.1007/s11548-017-1652-z. URL: https://doi.org/10.1007/s11548-017-1652-z.

[7] J. Wang et al. "Augmented Reality Navigation With Automatic Marker-Free Image Registration Using 3-D Image Overlay for Dental Surgery". In: *IEEE Transactions on Biomedical Engineering* 61.4 (Apr. 2014), pp. 1295–1304. ISSN: 0018-9294. DOI: 10.1109/TBME.2014.2301191.

[8] Reid Vassallo et al. *Hologram stability evaluation for Microsoft HoloLens*. 2017. DOI: 10.1117/12.2255831. URL: http://dx.doi.org/10.1117/12.2255831.

[9] Scopsis Medical. *Spine Surgery Using Scopis Holographic Navigation and Microsoft HoloLens*. URL: https://www.youtube.com/watch?v=xvbWE4OsKxY (visited on 06/25/2017).

[10] Jason Odom. *Two Doctors Simplifyu Spinal Surgery with the HoloLens*. URL: https://hololens.reality.news/news/two-doctors-simplify-spinal-surgery-with-hololens-0175101/ (visited on 06/03/2017).

[11] Microsoft. *HoloLens hardware details*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/hololens_hardware_details (visited on 08/25/2017).

[12] Microsoft. *Coordinate systems*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/coordinate_systems (visited on 08/25/2017).

[13] Microsoft. *Spatial mapping in Unity*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/spatial_mapping_in_unity (visited on 08/25/2017).

[14] Microsoft. *Spatial mapping*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/spatial_mapping (visited on 08/25/2017).

[15] Microsoft. *Gaze*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/gaze (visited on 08/25/2017).

[16] Microsoft. *Gestures*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/gestures (visited on 08/25/2017).

[17] Microsoft. *Speech Recognition Grammar Specification Version 1.0*. URL: https://www.w3.org/TR/speech-grammar/ (visited on 08/25/2017).

[18] Microsoft. *Frequently asked questions*. URL: https://www.microsoft.com/en-us/hololens/faq (visited on 08/25/2017).

[19] Microsoft. *Voice input in Unity*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/voice_input_in_unity (visited on 08/25/2017).

[20] Ziv Yaniv - School of Engineering and Jerusalem - Israel Computer Science - The Hebrew University. *Point-based rigid registration*. URL: http://yanivresearch.info/writtenMaterial/pbrr.pdf (visited on 08/24/2017).

[21] Berthold K. P. Horn. "Closed-form solution of absolute orientation using unit quaternions". In: *J. Opt. Soc. Am. A* 4.4 (Apr. 1987), pp. 629–642. DOI: 10.1364/JOSAA.4.000629. URL: http://josaa.osa.org/abstract.cfm?URI=josaa-4-4-629.

[22] Unity Technologies. *Unity - Game Engine*. URL: https://unity3d.com/de (visited on 01/10/2017).

[23] Blender Foundation. *Blender.org*. URL: https://www.blender.org/download (visited on 02/05/2017).

[24] ALGLIB Project. *AlgLib Free Edition*. URL: http://www.alglib.net/translator/re/alglib-3.11.0.csharp.gpl.zip (visited on 03/25/2017).

[25] Microsoft. *HoloToolkit-Unity*. URL: https://github.com/Microsoft/HoloToolkit-Unity/tree/master/External/Unitypackages (visited on 05/10/2017).

[26] Microsoft. *Performance recommendations for HoloLens apps*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/performance_recommendations_for_hololens_apps (visited on 04/05/2017).

[27] Microsoft. *Hologram stability*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/hologram_stability (visited on 04/05/2017).

[28] Microsoft. *Camera in Unity*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/camera_in_unity (visited on 04/05/2017).

[29] Johannes Traa. *Least-Squares Intersection of Lines*. URL: http://cal.cs.illinois.edu/~johannes/research/LS_line_intersect.pdf (visited on 04/02/2017).

[30] State University of Campinas (Brasil) - Faculdade de Tecnologia. *A4 1mm Square Graph Paper*. URL: http://www.ft.unicamp.br/~lfavila/TT113EB104/folha_milimetrada.pdf (visited on 06/05/2017).

[31] Microsoft. *Spectator view*. URL: https://developer.microsoft.com/en-us/windows/mixed-reality/spectator_view (visited on 05/23/2017).

[32] U.S.A. National Institutes of Health. *ImageJ*. URL: https://imagej.nih.gov/ij/index.html (visited on 06/23/2017).

[33] Bill McCrary. *HoloToolkit - Simple Drag/Resize/Rotate - Bill's Blog*. URL: https://www.billmccrary.com/wp-content/uploads/2017/03/SimpleDragResizeRotate.unitypackage.zip (visited on 06/15/2017).

Acknowledgements

I'd like to thank my supervisors Prof. Pollefeys, Prof. Göksel, Dr. Fürnstahl and Dr. Sheikh for their tremendous support and patience with all my questions. I'd also like to thank Mr. Hecht from Balgrist CARD for all the support with 3D printing. This is also the perfect opportunity to thank all my relatives, friends and loved ones for their trust in me and their support, not only during the making of this thesis, but over the past years as an ETH student as well.

Thank you!

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Augmented Reality for Surgical Navigation using HoloLens

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Schneider	Fabian

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 1.7.2017	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.