Collision detection and proximity sensors in 3D simulations of mechanical systems

Huijgens, J.

Award date:
2015

Eindhoven University of Technology

Master Thesis

Collision detection and proximity sensors in 3D simulations of mechanical systems

Supervisor TU/e:
Dr. Andrei Jalba

Author:
Jesper Huijgens

Supervisors TNO:
Dr. Carmen Bratosin
&
Dr. Bart Theelen

Department of Mathematics and Computer Science
Visualization

Eindhoven University of Technology

November 2015

Collision detection and proximity sensors in 3D simulations of mechanical systems

by Jesper Huijgens

Abstract

In this thesis we discuss two challenges in 3D simulations of certain mechanical systems. Firstly we discuss collision detection, secondly we discuss how to simulate proximity sensors. The discussed and proposed methods and algorithms are evaluated and implemented in SimulationTool, a tool developed by Philips Healthcare to simulate interventional X-Ray machines.

For collision detection we describe the GJK-algorithm. To increase the performance of collision detection we split the process in a broad phase and narrow phase. In the broad phase we check for an intersection between the Bounding Volumes (BVs), volumes that entirely contain an object. We test for an intersection between the two objects only if the BVs collide.

To model proximity sensors we propose methods for two types of sensors: a distance sensor and a capacitive sensor. The distance sensor is comparable to a parking sensor in a car. The model for the distance sensor computes the shortest distance to any object in its conic sensing volume.

Capacitive sensors generate an electric field surrounding the sensor. Any presence or displacement of nearby conductive objects changes the electric field, which results in a change in the output voltage of the sensor. In this thesis we describe and validate a model to simulate this sensor. Based on the output voltages, the interventional X-Ray machines may be restricted in their movement in certain directions. The proposed model approximates the capacitance of objects close to the sensor. Based on the capacitance of these objects the model predicts the output of the sensor.

Acknowledgements

This thesis is the result of eight months of hard work. A lot of people have helped me through these months and a few of them I want to thank explicitly. First of all I would like to thank dr. Andrei Jalba. Even though he is no expert on the interventional X-Ray machines developed by Philips Healthcare, he always directed me in the right direction or came up with a solution for my challenges.

Secondly I want to thank dr. Carmen Bratosin and dr. Bart Theelen. Carmen for introducing me to the project and pointing me in the right direction. I would like to thank Bart for everything he has done for me during the project. Beside the weekly meetings that kept me on track, he always managed to help me with any of my problems. Bart was the one that usually suggested totally different but great solutions to problems I was trying to tackle for over a week.

Furthermore, I want to thank Sjoerd Leeuwenberg. Sjoerd helped me out with the the experiments and gave me tons of advise and information about the bodyguard on the interventional X-Ray machines.

Lastly, I take this opportunity to thank my family and friends for the patience and support during the project. The person I like to thank the most is my girlfriend. Without her endless patience and support, this eight months would have been totally different.

Thanks everybody!
Jesper Huijgens

Contents

List of Figures

Abbreviations

BV	**B**ounding **V**olume
OBB	**O**riented **B**ounding **B**ox
AABB	**A**xis **A**ligned **B**ounding **B**ox
GJK	**G**ilbert **J**ohnson **K**eerthi
FEM	**F**inite **E**lement **M**ethod
FOV	**F**ield **O**f **V**iew

1 Introduction

Interventional X-Ray machines are widely used in radiology. Radiology is an imaging technique useful to reveal the internal structure of an object, such as bones in a human body. Although the technique was discovered in the late 1800s, further development of interventional X-Ray machines is still a core activity for Philips Healthcare. One of the machines developed by Philips Healthcare is shown in Figure 1.1.

FIGURE 1.1: An interventional X-Ray machine developed by Philips Healthcare

It is needless to mention that the development costs of these kind of machines are immense. To support the development of said machines, Philips Healthcare is developing a simulation tool called SimulationTool. SimulationTool is being developed to virtually simulate interventional X-Ray machines. A user can load a certain interventional X-Ray machine into SimulationTool, and via a user interface, the user can control the movements of the machine similarly to the physical machine. Figure 1.2 shows an example of

SimulationTool with the machine in Figure 1.1 loaded into the scene. By using SimulationTool, an approximation of a machine's expected behavior can be observed without constructing an expensive prototype.

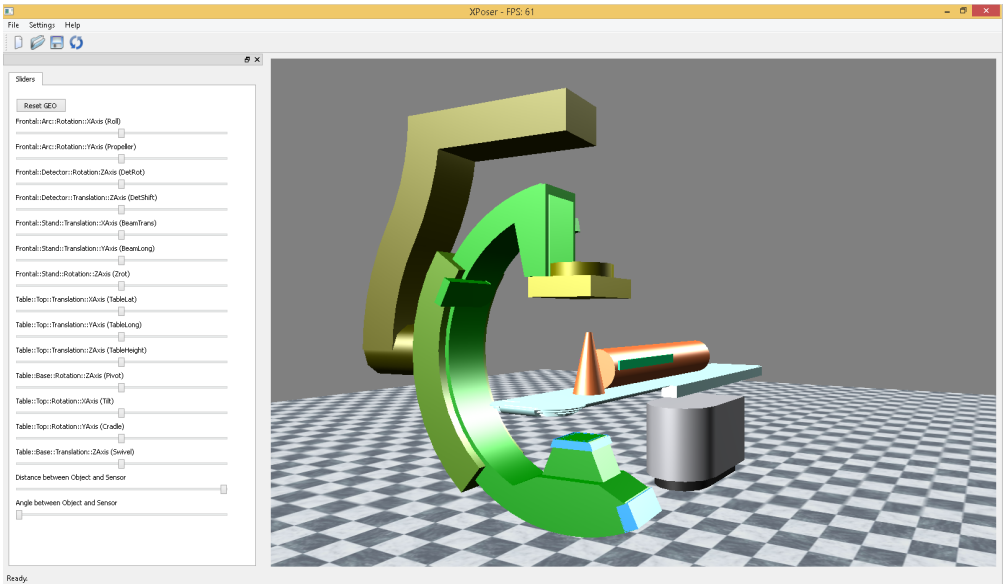

FIGURE 1.2: An example of SimulationTool simulating an interventional X-Ray machine developed by Philips Healthcare

The machine in SimulationTool may be seen as a model of an interventional X-Ray machine consisting of 3D objects, such as the table top or the scanner connected to the arm. Each of these 3D objects is a polyhedron constructed of triangles. The objects may have several degrees of freedom to move or rotate. For example, the table top can be lowered or raised but the table top can also be rotated. All the objects are managed by a scene graph to preserve spatial relations, i.e. if the table top is lowered, the patient on top of the table lowers as well. The machines do not simply instantly move or stand still, the objects may have velocity and acceleration. SimulationTool is implemented with a physics engine that simulates these accelerations. Additional information about the scene graph can be found in Appendix A.

1.1 Problem description

The simulation is made as realistically as possible to mimic the behavior of a physical machine. Yet, not the entire behavior of the physical machines is captured by SimulationTool. The physical machine cannot move through other objects, and in the worst case, the machine collides with an object in the room, like the table or patient. The version of SimulationTool that was available for this project lacked the feature to handle collisions of the machine with any other object in the scene. The machines in the simulation could move through objects without stopping or notifying the user.

Furthermore, a collision between the machine and an object (e.g. a patient or doctor) should be avoided at all costs. To prevent collisions, Philips Healthcare geared the interventional X-Ray machines with capacitive proximity sensors. These sensors generate an electric field surrounding the sensor. Any presence or displacement of nearby conductive objects alter the electric field, which results in a change in the output voltage of the sensor. Based on the output voltages, the machine may be restricted in its movement in certain directions.

The goal of this project is to get SimulationTool two steps closer to the behavior of the physical machines. The first step is enhancing it with collision detection. Normally, a simulation would be implemented with collision handling. Collision handling consists of two parts, collision detection and the response to the detected collision. Note that detecting a collision does not imply that the movement of the machine during the simulation is stopped (this is a response), the software is just informed of the collision. In SimulationTool we are only interested in the detection part.

The second step is simulating proximity sensors. The output of the simulated sensors can be used in an algorithm (similar to the algorithm used in a physical machine) to restrict the movements of the machine in certain directions. With the simulation of proximity sensors a collision prevention algorithm may be tested in SimulationTool instead of a physical interventional X-Ray machine. The former is likely a less costly alternative to the latter.

1.2 Summary of results

During this project two considerable contributions were added to SimulationTool: SimulationTool was extended with collision detection and with models to simulate proximity sensors.

For collision detection the GJK-algorithm [Ber99] was implemented. To increase the performance of collision detection, objects are first tested in a fast broad phase. In the broad phase we test for intersections between the bounding volumes of the objects. Objects passing the broad phase are further tested for collisions in the narrow phase.

Models for two types of proximity sensors were developed to support collision prevention. Firstly, a model for a distance sensor that is comparable to an infrared or ultrasonic proximity sensor was made. The model is used to measure the shortest distance between the sensor and any object present in a conic volume (the space visible to the sensor).

Secondly a model that captures the behavior of Philips Healthcare's capacitive sensors was made. The model is thoroughly tested and validated against the proximity sensors on a real interventional X-Ray machine. The model in SimulationTool can be fine-tuned such that a good approximation is made of the output of the sensor in the physical machine.

The model for distance sensors could be used in several experimental simulations to find out if a distance sensor would be more suitable for collision prevention than a capacitive sensor. The advantage of a distance sensor is that it measures the shortest distance between the sensor and the closest object, whereas a capacitive sensor is influenced by the distance, size and material of nearby objects. The disadvantage is that a distance sensor is restricted by its sensing volume: any nearby objects outside of this area are not observed. In contrast, a capacitive sensor would observe these objects. Furthermore, anything irrelevant covering the sensor (e.g. a cable, a cover or some fluid) may result in detrimental outputs. The use of a distance sensor is thus not a realistic option for Philips Healthcare. For this reason, we focus more on the capacitive sensor in this thesis.

Thirdly, we validated the capacitive sensor model. We conducted several experiments, using the actual capacitive sensor, and performed the same experiment in SimulationTool

and compared the results. With the experiments we investigate the influence of the distance, size and orientation of nearby objects.

1.3 Structure of this thesis

In Section 2 we discuss how collision detection is implemented using the GJK-algorithm. We describe several Bounding Volumes that can increase the performance of the GJK-algorithm in Section 3. We explain the distance sensor in Section 4 and the capacitive sensor in Section 5. We validate the model for the capacitive sensor and the used methods in Section 6. In Section 7 we summarize the project, discuss some limitations in the contributions to SimulationTool and list directions for future work.

2 | Collision detection

The first goal of the project was to implement collision detection in SimulationTool. In this section we describe how the GJK-algorithm is used to test for intersections between objects in SimulationTool. Note that we did not implement a version of the GJK-algorithm that computes the shortest distance between two objects.

First we review literature about collision detection, but specially about the GJK-algorithm in Section 2.1. Preliminaries are mentioned in Section 2.2. In Section 2.3 we describe how the GJK-algorithm works. Pseudo code of the GJK-algorithm can be found in Section 2.4. The results of the GJK-algorithm are shown in Section 2.5.

2.1 Related work

Collision detection is a widely discussed topic in the literature of computer science. Several distance/collision algorithms and optimizations are devised and well discussed. Since there are various algorithms and approaches, detecting collisions is not the problem, however, choosing the right algorithm and approach to detect collisions is the problem. S. Kockara reviewed the most common intersection detection algorithms in [KHI+07]. Kockara clearly states that pair-wise testing the primitive segments of two objects is too exhaustive to do every step in a simulation.

The GJK-algorithm (Gilbert-Johnson-Keerthi) [GJK88] [Ber99] is a commonly used algorithm for collision detection. Given two objects, the GJK algorithm finds the closest distance between the convex hulls of these objects. Both Kockara in [KHI+07] and G. Zachmann in [Zac00] describe the GJK-algorithm as an efficient simplex-based algorithm that computes the shortest distance between two objects. The algorithm is known for its fast convergence, especially for the simpler task of finding whether two objects collide or not (instead of finding the shortest distance). However, there are some caveats. The algorithm may be prone to rounding errors due to floating point arithmetics. A second downside of the algorithm is that it operates on the convex hulls of the objects and not on the actual geometry of concave objects [Ber99] [Cou01]. Most objects in SimulationTool

are convex, making the GJK-algorithm a very convenient algorithm. Only one object is concave but this concave object is easily split up in a few convex objects, making the GJK-algorithm applicable to the whole scene in SimulationTool. In other simulations it may not be so trivial to split a concave object into convex objects. In [MG09] an algorithm is described that decomposes a concave object in a set of convex objects. An alternative is creating a set of convex shapes that, combined, approximate the concave object.

2.2 Preliminaries

The GJK-algorithm operates on a specific combination of the vertices of the two objects, called the Minkowski difference [KHI+07].

The Minkowski difference of two point sets A and B, denoted as $A \Theta B$, is the subtraction of each point in B from each point in A.

Definition 2.1. Minkowski difference of point sets A and B:

$$A \Theta B = \{\mathbf{x} - \mathbf{y} : \mathbf{x} \in A, \mathbf{y} \in B\}$$

Figure 2.1 illustrates the Minkowski difference of objects A and B.

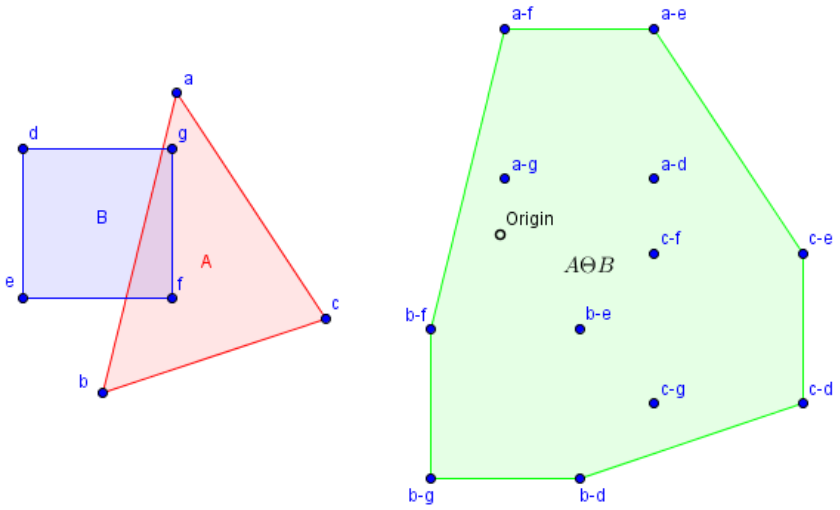

FIGURE 2.1: The Minkowski difference $A\Theta B$ (Green) of vertex sets A (Red) and B (Blue)

Two properties of the Minkowski difference are the following:

Property I. The shortest Euclidean distance between A and B, is equivalent to the shortest Euclidean distance from $A\Theta B$ to the origin. i.e.

$$\min_{a\in A, b\in B} d(a,b) = \min_{c\in A\Theta B} d(c,O)$$

Property II. Objects A and B collide if and only if the origin is contained by the minkowski difference. i.e.

$$O \in A\Theta B \Leftrightarrow A \text{ and } B \text{ collide}$$

2.3 Design

Using the Minkowski difference makes collision detection relatively easy. To determine the shortest distance between two objects, we can compute the shortest distance of the Minkowski difference to the origin (Property I).

To determine whether two objects are colliding, we can determine if the origin is enveloped by the Minkowski difference (Property II). In this thesis we only discuss a boolean variant of the GJK algorithm that tests if two objects are colliding or not, thus only Property II is used.

Computing the complete Minkowski difference can be an intensive task, depending on the number of vertices of the objects. Luckily we do not need to compute the entire Minkowski difference, we only have to visit several points on the boundary of the convex hull of the Minkowski difference, called support points.

2.3.1 Support points

Suppose we have point set P, then the support point $\mathbf{s} \in P$ along direction $\mathbf{v}$ is the furthest point in P along direction $\mathbf{v}$. In Figure 2.2 we can see an illustration of finding support point of A in direction $\mathbf{v}$. If one would stand inside A, point b is the furthest point in direction $\mathbf{v}$.

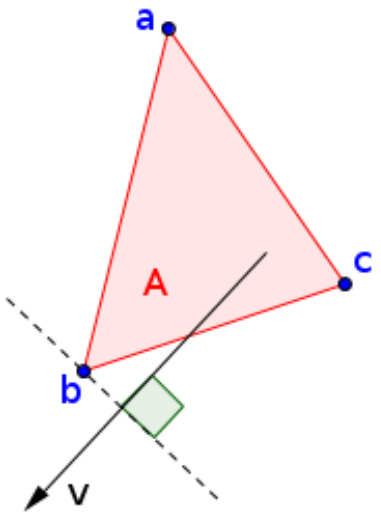

FIGURE 2.2: Illustrating the support point of A in the direction of $\mathbf{v}$

For notation we denote a support point in point set P along direction $\mathbf{v}$ as $Supp(P, \mathbf{v})$. The support point $\mathbf{s}$ of a point set P, the point furthest (seen from any point in the P)

in direction $\mathbf{v}$, can be found via the projection of each point in P on $\mathbf{v}$:

$$\mathbf{s} = Supp(P, \mathbf{v}) = \underset{\mathbf{p} \in P}{\operatorname{argmax}}\, \mathbf{p} \cdot \mathbf{v}$$

Looking at the definition of the Minkowski difference, we can observe that $Supp(A\Theta B, \mathbf{v}) = Supp(A, \mathbf{v}) - Supp(B, -\mathbf{v})$: The furthest point in A along direction $\mathbf{v}$, minus the furthest point in B along the opposite direction $-\mathbf{v}$. Figure 2.3 illustrates the definition of the support point $Supp(A\Theta B, \mathbf{v})$. The furthest point in shape A along direction $\mathbf{v}$ is b, the furthest point in shape B along direction $-\mathbf{v}$ is g and indeed, $Supp(A\Theta B, \mathbf{v})$ is $b - g$.

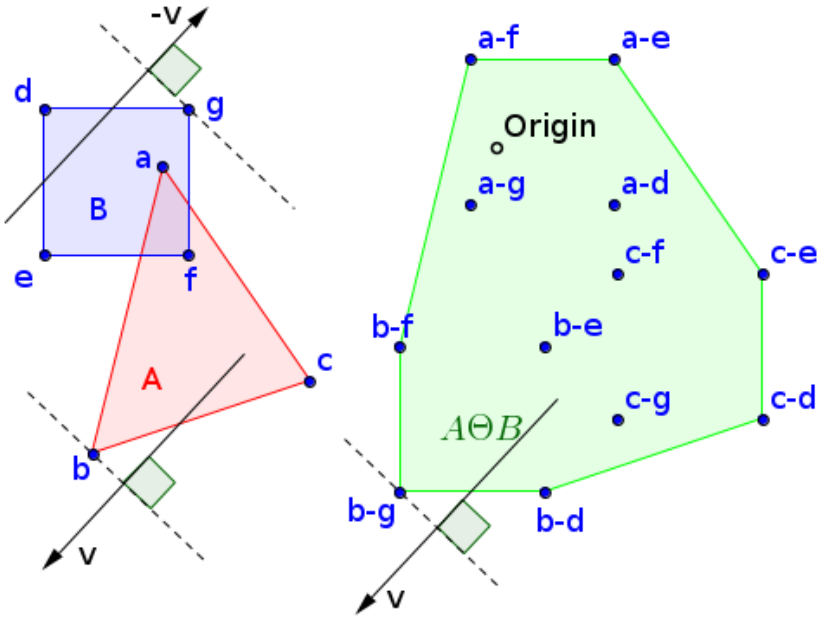

FIGURE 2.3: Illustrating the definition of the support point $Supp(A\Theta B, \mathbf{v})$, b is the support point of A, g of B and $b - g$ of $A\Theta B$

Note that to find a support point in the Minkowski difference we do not need to compute the total Minkowski difference.

2.3.2 The GJK-algorithm

When given two objects, the GJK-algorithm constructs a simplex S that is as close as possible to the origin, by using support points. That is because the shortest distance from the final simplex to the origin is the shortest distance between the two given objects.

Note that in the case of a collision, S will eventually envelop the origin. In $\mathbb{R}^2$ a simplex is a triangle, in $\mathbb{R}^3$ it is a tetrahedron.

For clarification, in Figure 2.2, the simplex of support points $b-g$, $a-e$ and $c-e$ could be a final simplex. This simplex envelops the origin, indicating that shape A and B are indeed colliding.

Initially S is a simplex consisting of an arbitrarily chosen support point. With each iteration of the algorithm, a new support point s is added to S. From $S \cup \{s\}$ a new simplex is computed: a subset of $S \cup \{s\}$ that is closest to the origin. In all cases, s is part of the new simplex, therefore the new simplex is closer to the origin, or at least just as close.

Depending on the variant of the GJK-algorithm, the algorithm may have a different termination condition. If the algorithm is tailored to approximate the distance between two objects, the algorithm updates the simplex until the new support point is not significantly closer (and still in the same direction) to the origin than the current simplex. A variant tailored to test if objects either collide or not may terminate earlier. Both variants terminate when the simplex envelops the origin, however, the latter may terminate when a separating plane is found [Ber99]. A separating plane is found when the new support point is not beyond the origin, as seen from the current simplex. The support point is the furthest point from the simplex towards the origin. If that point did not pass the origin then there is no point beyond the origin and therefore no simplex exists that envelops the origin. Such a termination may occur much earlier than with the former variant of the algorithm. This faster termination condition is implemented in SimulationTool.

2.4 Implementation

In this section we describe the high level implementation. We give pseudo code of the structure of the algorithm. The algorithm relies on two procedures. The first is the procedure to find a support point in direction $\mathbf{v}$. The second procedure is to check if the simplex contains the origin after a new support point is added, if it does contain the origin, the algorithm terminates. If not, the procedure updates the simplex to the

smallest subset of the current simplex that is closest to the origin and it updates the search direction $\mathbf{v}$. The last procedure is in more detail explained in Appendix C.

The high level pseudo code, Algorithm 2.1, is rather straightforward. Given point sets A and B, we begin with an empty simplex S and an arbitrary direction vector $\mathbf{v}$. With each iteration we compute a new support point. Whenever, seen from the current simplex, the new support point is not on the other side of the origin, a simplex that envelops the origin does not exist: a separating plane is found and we may terminate. Otherwise we check whether the new simplex (including the new support) envelops the origin or not and update the simplex and the direction vector.

Algorithm 2.1 GJK-algorithm(Pointset A, Pointset B)

```
S = ∅ {The simplex}
v ⇐ "Arbitrary vector" {The direction vector}
while true do
  {Find new support}
  s ⇐ Supp(AΘB, v)
  if s · v ≤ 0 then
    {A separating plane is found}
    return false
  if CheckAndUpdateSimplex(s, S, v) then
    return true
```

2.5 Result

In SimulationTool all objects that should not collide are tested for collisions. In Figure 2.4 we can clearly see that, in the right image, the scanner and the head of the patient are colliding. We inform the user of an intersection by showing a red exclamation mark.

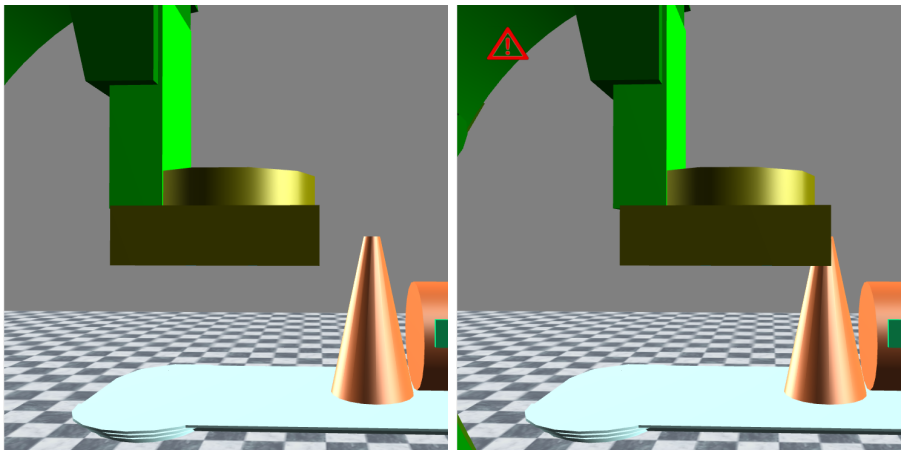

FIGURE 2.4: Left: no collision. Right: a collision of the scanner and the head of the patient. The red exclamation mark indicates that there is a collision.

3 Bounding volumes

Even without using separating planes, the GJK-algorithm as described in the previous section is fast but applying any of the two variants to a large number of objects with complex geometries may result in a major performance penalty. An efficient approach for increasing performance of collision detection is by splitting the process into a broad phase and a narrow phase [KHI+07]. During the broad phase, obviously disjoint objects can be identified in an attempt to prune tests in the narrow phase. In the narrow phase, only objects that passed the broad phase are tested for collisions. Of course, splitting the process is only useful if the broad phase is much faster than the narrow phase.

An often used technique for the broad phase is testing for collisions between the bounding volumes of two objects. A Bounding Volume (BV) is a volume that entirely envelops an object [Got00]. If two BVs are not colliding, then the objects are not colliding either. However, having two colliding BVs does not imply that the corresponding objects are in contact as well. If the BVs collide, the objects pass the broad phase and are tested for an actual collision during the narrow phase.

In Section 3.1 we discuss three different BVs and their properties. In Section 3.2 algorithms are provided to compute an Axis Aligned Bounding Box (AABB) or an Oriented Bounding Box (OBB). The results of the BVs are shown in Section 3.3.

3.1 Design

Any kind of shape could be chosen as a BV, as long as the shape entirely contains the more complicated object. The commonly used bounding volumes are the Axis-Aligned Bounding Boxes (AABB) and the sphere [Got00], but other volumes like Oriented Bounding Boxes (OBB), cylinders or the convex hull could also be suitable in certain situations. The AABB and the OBB are implemented in SimulationTool. In the following sections we discuss the sphere, the AABB, the OBB and how to compute an AABB or OBB. Most of the information in this Section is discussed by Gottschalk

in [Got00]. However, for the OBB some improvements/corrections are made in the equations to calculate the covariance matrix of an object.

3.1.1 Bounding Spheres

One of the most simple bounding volumes is the bounding sphere. A bounding sphere is (not necessary, but ideally) the smallest sphere enveloping an object. An example of a bounding sphere is shown in Figure 3.1

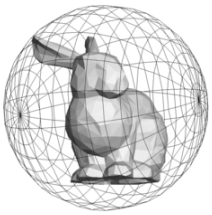

FIGURE 3.1: An example of a bounding sphere enveloping the Stanford bunny. Source: www.mathforum.org

A bounding sphere could be represented by its center point $\mathbf{c}$ and radius r [Got00]. With these parameters the sphere envelops the following region R:

$$R = \{(x, y, z)^T \quad | \quad (x - \mathbf{c}_x)^2 + (y - \mathbf{c}_y)^2 + (z - \mathbf{c}_z)^2 < r^2\}$$

A disadvantage of the bounding sphere is that it possibly contains a much bigger region than the object itself. On the other hand, intersection tests with bounding spheres are particularly simple. Two bounding spheres bs_1 and bs_2 with center points $\mathbf{c_1}$ and $\mathbf{c_2}$ respectively and radii r_1 and r_2 respectively intersect iff

$$(\mathbf{c_2}_x - \mathbf{c_1}_x)^2 + (\mathbf{c_2}_y - \mathbf{c_1}_y)^2 + (\mathbf{c_2}_z - \mathbf{c_1}_z)^2 \leq (r_1 + r_2)^2$$

3.1.2 Axis Aligned Bounding Boxes

Another variant of bounding volumes is the Axis Aligned Bounding Box (AABB). An AABB is the smallest possible axis aligned box enveloping the object, meaning that the

box is aligned with the axis of the coordinate system. An example of a AABB is given in Figure 3.2.

FIGURE 3.2: An example of an AABB enveloping the Stanford bunny. Source: www.mathforum.org

An AABB could be represented in multiple ways.
One representation could be the lower bound and upper bound for each axis: l_x, l_y, l_z, u_x, u_y and u_z. The region R is specified as follows:

$$R = \{(x, y, z)^T \quad | \quad l_x \leq x \leq u_x \ \wedge \ l_y \leq y \leq u_y \ \wedge \ l_z \leq z \leq u_z\}$$

An alternative but commonly used representation is a corner point $\mathbf{p}$ and the lengths of each edge d_x, d_y and d_z. Then the region R is specified by the following formula:

$$R = \{(x, y, z)^T \quad | \quad \mathbf{p}_x \leq x \leq \mathbf{p}_x + d_x \ \wedge \ \mathbf{p}_y \leq y \leq \mathbf{p}_y + d_y \ \wedge \ \mathbf{p}_z \leq z \leq \mathbf{p}_z + d_z\}$$

A third representation is similar to the previous but slightly different: a center point $\mathbf{c}$ is given and the half widths along each axis w_x, w_y and w_z (half of the width of the AABB in each direction). With these parameters R is specified as follows:

$$R = \{(x, y, z)^T \quad | \quad |\mathbf{c}_x - x| \leq w_x \ \wedge \ |\mathbf{c}_y - y| \leq w_y \ \wedge \ |\mathbf{c}_z - z| \leq w_z\}$$

The intersection test for AABBs is like the bounding spheres relatively simple. Assume we have two AABBs, $aabb_1$ and $aabb_2$, represented by their relative center point $\mathbf{c_1}$ and $\mathbf{c_2}$ and their relative half widths w_{1x}, w_{1y}, w_{1z} and w_{2x}, w_{2y}, w_{2z}. Then $aabb_1$ and $aabb_2$ intersect iff

$$|\mathbf{c_2}_x - \mathbf{c_1}_x| \leq w_{1x} + w_{2x} \ \wedge \ |\mathbf{c_2}_y - \mathbf{c_1}_y| \leq w_{1y} + w_{2y} \ \wedge \ |\mathbf{c_2}_z - \mathbf{c_1}_z| \leq w_{1z} + w_{2z}$$

3.1.3 Oriented Bounding Boxes

An often better fitting bounding box, compared to the AABB, is the Oriented Bounding Box (OBB). An OBB is not bound to the axis of the Cartesian coordinate system, they may have an arbitrary orientation. An example of an OBB next to an AABB is shown in Figure 3.3.

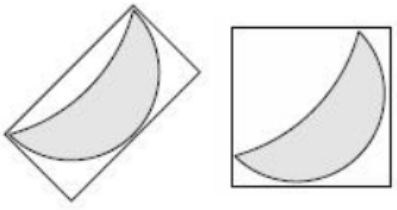

FIGURE 3.3: An example of an OBB next to an AABB in $\mathbb{R}^2$. Source: cse.csusb.edu

Figure 3.3 clearly shows why OBBs are more suitable compared AABBs: the OBB is a much tighter volume.

To represent an OBB we could use one of the three discussed representations for an AABB and additionally a representation for the orientation. For this we use three mutually orthogonal unit vectors $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$. From here and further in this report we use the third representation of an AABB. We represent an OBB with a center point $\mathbf{c}$, half widths w_1, w_2 and w_3 and the three direction vectors $\mathbf{v_1}$, $\mathbf{v_2}$ and $\mathbf{v_3}$. The region R is specified as follows:

$$R \;=\; \{\mathbf{c} + a\, w_1\, \mathbf{v_1} + b\, w_2\, \mathbf{v_2} + c\, w_1\, \mathbf{v_2} \quad | \quad a, b, c \in [-1, 1]\}$$

The intersection tests for OBBs are not as simple as for the previous BVs. For intersection tests one would have to use one of the many collision detection algorithms.

3.1.4 Finding an Oriented Bounding Box based on a covariance matrix

We saw that an OBB can fit an object more compact than an AABB. However, we do not know a suitable orientation for the bounding box beforehand. We somehow have to fit the bounding box compactly around the shape of the object.

In 1985, Joseph O'Rouke [O'R85] proposed an algorithm that computes the minimal box enveloping an object (an OBB). Although the bounding box is of minimal size, the algorithm is rather complicated. Firstly, one has to construct a so called Gaussian sphere for the given object. Then, for all pairs of edges, a box has to be rotated around the object until the smallest box is found. The algorithm runs in $O(n^3)$ time, where n is the number of vertices (points) of the object.

Luckily, a faster approximation algorithm exists that runs in only $O(n)$ time, where n is the number of triangles of the object, which was devised by Gottschalk [Got00]. Gottschalk describes three fitting algorithms based on the statistical spread in the geometry of the object. The algorithms are each based on a different property of the geometry. The algorithms approximate the directions of the shape of the object, particularly the direction with the most spread and the direction with the least spread. The first proposed algorithm approximates the shape of the object based on the spread of the vertices, another algorithm is based on the spread of the triangles of the object, a third algorithm uses the convex hull of the object. In this thesis we only discuss the first two of the three.

The shape of a set of points can be approximated with a covariance matrix C and a centroid $\mathbf{m}$, just like how a normal Gaussian curve describes the distribution of points in one dimension. For objects in $\mathbb{R}^n$, matrix C is a symmetric $n \times n$ matrix. In general, the eigenvectors of the matrix are mutually orthogonal and we use these vectors as the orientation of the OBB.

3.1.4.1 Covariance matrix based on the distribution of vertices

The straight forward approach of calculating the covariance matrix C is by inspecting the distribution of the vertices of the object. After all, the object is defined by the vertices and triangles of the object. Suppose that the object consists of n points $\mathbf{p^1}, \mathbf{p^2}, ..., \mathbf{p^n}$. Then C_{ij} (element (i, j) of C) is defined as follows:

$$\begin{aligned} \mathrm{C}_{ij} = \mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_j) &= \mathrm{E}[\mathbf{x}_i \mathbf{x}_j] - \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j] \\ &= \frac{1}{n}\sum_{k=1}^{n} \mathbf{p}_i^k \mathbf{p}_j^k \; - \; \frac{1}{n}\sum_{k=1}^{n} \mathbf{p}_i^k \; \frac{1}{n}\sum_{k=1}^{n} \mathbf{p}_j^k \end{aligned}$$

where $\mathbf{x}$ is a multivariate random variable (random vector) and $\mathrm{E}[\mathbf{x}_i]$ the expectation of the ith coordinate of $\mathbf{x}$. All the detailed derivations to compute the covariance matrix based on the distribution of vertices, step by step, can be found in Appendix B.1.

3.1.4.2 Covariance matrix based on the distribution of triangles

A problem with the covariance matrix based on only the vertices is that a skewed (not uniformly spread) vertex set will influence the result of the covariance matrix. For example, a small part of an object that contains half of all the vertices will contribute to half of the covariance matrix. A solution would be to somehow weight the vertices, such that a cluttered part of an object does not bias C. This weighting may be done by using the areas of the triangles: smaller triangles contribute less to the covariance matrix than larger triangles.

The definition for the covariance matrix using weighted triangles is as follows:

$$\begin{aligned} \mathrm{C}_{ij} =& \mathrm{E}[\mathbf{x}_i\mathbf{x}_j] - \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j] \\ =& \frac{1}{A^M}\sum_{k=1}^{n}\frac{2A^k}{24}\left(9\mathbf{m}_i^k\mathbf{m}_j^k + \mathbf{p}_i^k\mathbf{p}_j^k + \mathbf{q}_i^k\mathbf{q}_j^k + \mathbf{r}_i^k\mathbf{r}_j^k\right) \\ & - \frac{\sum_{k=1}^{n}A^k\mathbf{m}_i^k}{A^M}\ \frac{\sum_{k=1}^{n}A^k\mathbf{m}_j^k}{A^M} \end{aligned}$$

where,

- Superscript k refers to the kth triangle.
- Superscript M refers to all n triangles, also called the model (Model).
- A is the area of either a triangle or model (identified with superscript).
- $\mathbf{a}_i$ denotes the ith coordinate of $\mathbf{a}$.
- $\mathbf{p}^k$, $\mathbf{q}^k$ and $\mathbf{r}^k$ are the vertices of the kth triangle.
- $\mathbf{m}^k$ is the centroid of the kth triangle.

All the detailed derivations to compute the covariance matrix based on the distribution of triangles, step by step, can be found in Appendix B.2.

3.1.4.3 Fitting an Oriented Bounding Box

Now we have defined two techniques to calculate a covariance matrix, we find Cs eigenvectors $\mathbf{v^1}$, $\mathbf{v^2}$ and $\mathbf{v^3}$. The eigenvector(s) corresponding to the largest eigenvalue corresponds to the direction with the most co-variability, i.e. the direction with most spread. Since the covariance matrix is symmetric (by its definition), eigenvectors $\mathbf{v^1}$, $\mathbf{v^2}$ and $\mathbf{v^3}$ corresponding to distinct eigenvalues λ_1, λ_2 and λ_3 respectively are orthogonal to each other. The proof can be found in Appendix B.3. We use the normalized vectors of $\mathbf{v^1}$, $\mathbf{v^2}$ and $\mathbf{v^3}$ as the orientation of the OBB.

To find the dimensions of the OBB, we project each point $\mathbf{p}^k$ on each of the vectors and find the extremes (lower extreme $\mathbf{l}$ and upper extreme $\mathbf{u}$) along each direction:

$$\mathbf{u}_i = \max_{1 \leq k \leq n} (\mathbf{v^i} \cdot \mathbf{p}^k)$$
$$\mathbf{l}_i = \min_{1 \leq k \leq n} (\mathbf{v^i} \cdot \mathbf{p}^k)$$

where n is the number of points of the object.
The half width $\mathbf{w}_i$ of the OBB along axis i of the OBB is given by:

$$\mathbf{w}_i = \frac{1}{2}(\mathbf{u}_i - \mathbf{l}_i)$$

The center $\mathbf{c}$ is given by:

$$\mathbf{c} = \sum_{i=1}^{3} \frac{1}{2}(\mathbf{l}_i + \mathbf{u}_i)\mathbf{v^i}$$

We now have fitted an OBB defined by $\mathbf{c}$, $\mathbf{v^1}$, $\mathbf{v^2}$, $\mathbf{v^3}$ and the half widths $\mathbf{w}$.

3.1.4.4 Exceptional Cases

In Section 3.1.4.3 we discussed that the eigenvectors are mutually orthogonal which is only the case with distinct eigenvalues. However, in some cases not all eigenvalues are distinct. We then say that the geometric multiplicity of an eigenvalue λ is greater than one, meaning that more than one eigenvectors correspond to eigenvalue λ. In this case two eigenvectors corresponding to λ are not necessarily orthogonal. The dimension of the eigenspace associated to λ is greater than one, i.e. the eigenspace is not defined by one corresponding eigenvector, but multiple linear independent eigenvectors defining a

space of higher dimension. These cases occur with objects that have equal spread in several directions. Such as a circle, a cylinder or a sphere.

Figure 3.4 clearly illustrates such a case: $\mathbf{v^2}$ and $\mathbf{v^3}$ have the same spread along the vectors but $\mathbf{v^2}$ and $\mathbf{v^3}$ are not orthogonal. A solution to this problem is finding the pair of non orthogonal eigenvectors and rotate one of the two vectors in such a way that it is orthogonal with the other two eigenvectors. Note that rotating one vector or the other has no influence on the size of the resulting bounding box, the spread of the vertices along both vectors is the same.

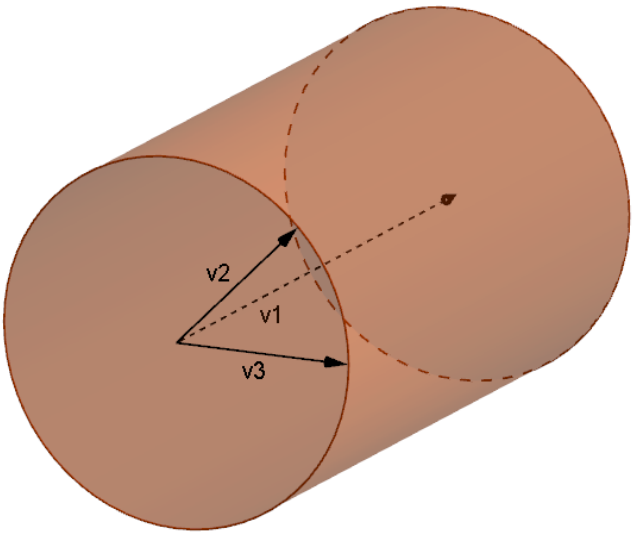

FIGURE 3.4: An example of a exceptional case: A cylinder with $\mathbf{v^2}$ and $\mathbf{v^3}$ not orthogonal

3.2 Implementation

In SimulationTool, both the AABB and the OBB are implemented. The explanation and pseudo code to compute the BVs is given in the following two sections.

3.2.1 Axis aligned bounding boxes

The algorithm to compute the AABBs of the objects in SimulationTool is rather straight forward. Given the point set of an object, we find the minimum and the maximum

coordinates: min_i and max_i along each axis i. The x coordinate of the center point of the AABB equals $\frac{1}{2}(max_x + min_x)$. The half width of the AABB along the x-axis equals $\frac{1}{2}(max_x - min_x)$. The same holds for the y and z axis. The pseudo code of the algorithm is in Algorithm 3.1.

Algorithm 3.1 computeAABBs()

```
let O be all the objects in SimulationTool
for each object o ∈ O do
  let P be the point set of o
  let c be the center point of the AABB of o
  let w_i be the half width of the AABB of o for the ith axis

  {For each point in P calculate the lower and upper bound along each axis}
  min_x, min_y, min_z ⇐ ∞
  max_x, max_y, max_z ⇐ −∞
  for all p ∈ P do
    for each axis i do
      {Check if it is new lower bound or upper bound}
      if p_i < min_i then
        min_i ⇐ p_i
      if p_i > max_i then
        max_i ⇐ p_i

  {Compute c and w based on the lower and upper bounds}
  for each axis i do
    w_i ⇐ 1/2 (max_i − min_i)
    c_i ⇐ 1/2 (max_i + min_i)
```

3.2.2 Oriented bounding boxes

The algorithm to compute the OBBs of all objects in SimulationTool follows the same structure, but is a bit more complex. First the orientation is found by computing the eigenvectors of the covariance matrix of an object. How to compute the covariance matrix is described in Section 3.1.4. Now we do not find the lower and upper bound along the Cartesian axes, but along the just computed directions of the orientation. From the directions and the lower and upper bounds the center point and the half widths are computed. The pseudo code of the algorithm is in Algorithm 3.2.

Algorithm 3.2 computeOBBs()

```
let O be all the objects in SimulationTool
for each object o ∈ O do
  let P be the point set of o
  let c be the center point of the OBB of o
  let v^1, v^2 and v^3 be the orientation of the OBB of o
  let w_i be the half width of the OBB of o for each direction v^i

  cov ⇐ computeCovarianceMatrix(P)
  let v^1, v^2 and v^3 be the eigenvectors of cov
  check for and fix exceptional cases (non-orthogonal eigenvectors)

  {For each point in P calculate the lower and upper bound along each direction}
  min_1, min_2, min_3 ⇐ ∞
  max_1, max_2, max_3 ⇐ −∞
  for all p ∈ P do
    for each v^i do
      {Check if it is new lower bound or upper bound}
      if v^i · p < min_i then
        min_i ⇐ v^i · p
      if v^i · p > max_i then
        max_i ⇐ v^i · p

  {Compute c and w based on the orientation and the lower and upper bounds}
  c_i ⇐ Σ_{i=1}^{3} ( 1/2 (max_i + min_i) v^i )
  for each direction i do
    w_i ⇐ 1/2 (max_i − min_i)
```

3.3 Result

As mentioned above, SimulationTool is implemented with AABBs and with OBBs. In Figure 3.5 we can clearly see the differences between the two BVs. The difference is very clear in the arc shapes. The OBBs are tighter around the objects.

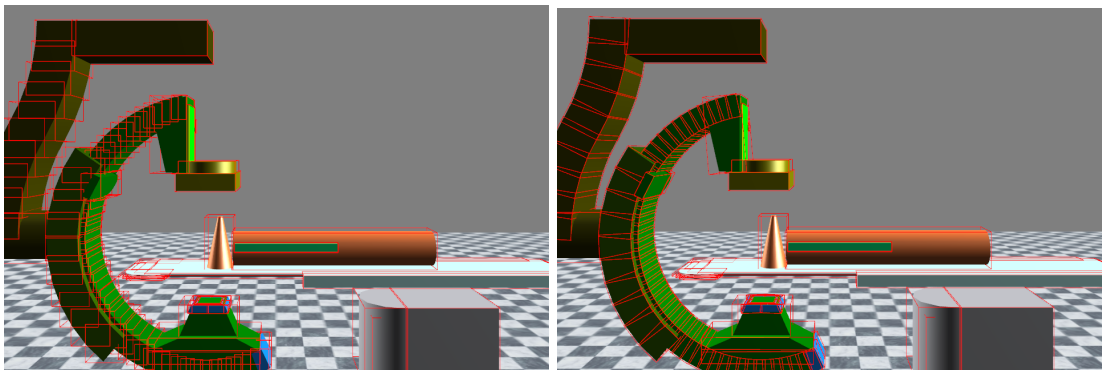

FIGURE 3.5: Left: a figure of SimulationTool using AABBs; Right: a figure of SimulationTool using OBBs

In SimulationTool most of the objects are initially placed axis aligned, so that for most objects the AABB is already a good BV.

In Figure 3.6 we see how BVs can increase the performance of collision detection.

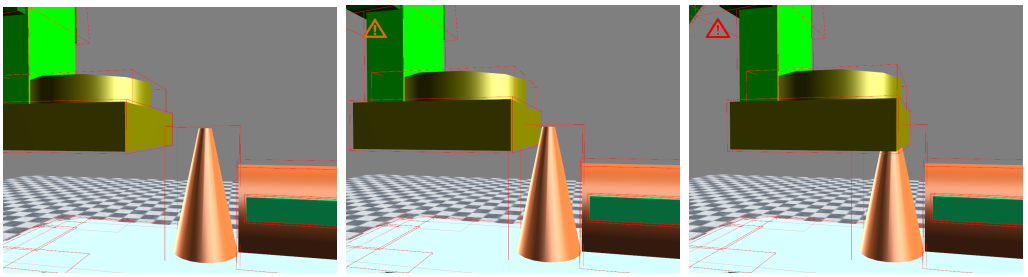

FIGURE 3.6: Left: no collisions occur in this case. Center: only the bounding boxes collide in this case. Right: a collision of the scanner with the head of the patient. The orange exclamation mark indicates that two or more bounding boxes are colliding, but no objects. The red exclamation mark indicates that there is a collision between two or more objects.

In the left figure none of the objects and none of the BVs are colliding, in this case only the BVs are tested for a collision, the objects do not pass the broad phase. In the middle figure the BVs are colliding, in this case the objects are tested for collision in the narrow phase. The orange exclamation mark indicates that only the BVs are colliding. In the right figure the objects are colliding, the collision is identified in the narrow phase. In SimulationTool, for the majority of the objects the BVs do not collide, so only a few objects will be tested for intersection per iteration. This is why BVs can increase the performance of collision detection.

If performance is still an issue in a simulation, one could consider hierarchies of BVs. Instead of only wrapping objects in a BV, pairs of BVs can be wrapped in a BV as well. One can keep wrapping BVs until only one root BV is left. If two BVs **A** and **B**, containing multiple other BVs, do not collide, then the BVs contained by **A** do not collide with BVs contained by **B**. This approach may increase the performance if a scene contains a lot of BVs.

4 Distance sensor

The first proximity sensor that we discuss is the distance sensor. The distance sensor may be compared to a parking sensor in a car: the sensor measures the distance of the closest object. However, it measures the objects in a sensing volume. Everything outside this volume can not be observed. For the distance sensor we use a conic sensing volume. Note that for the distance inside the cone we do not use the GJK-algorithm, however, we do use the GJK-algorithm to check if an object is inside the cone or not.

Literature and work related to distance sensors is reviewed in Section 4.1, the design of the sensor is described in Section 4.2. The distance sensors working in SimulationTool are shown in Section 4.3.

4.1 Related work

Literature describing how to simulate distance sensors, as a sensor that senses in a conic volume, is hard to find. However, literature that relates to properties of cones and intersection tests with cones is easier to find. Wolfram Alpha [Weib] lists a lot of properties of and equations for cones, very interesting is the conic sections obtained by cutting a cone with a plane [Weic]. The point on a conic section that is closest to the sensor, is in general a point of interest in simulating a distance sensor, because this point is the closest point visible to the sensor. However, a triangle does not necessarily entirely cut a cone, it may only intersect with a part of the cone.

M. Held [Hel97] describes a method to test for intersection of a triangle and a cone based on the mathematical equation of a cone, the method used requires an upright cone, with the sensor point on the origin. Held explains that this can be achieved for every cone by rotating and translating a whole scene, such that the cone is in the correct position and orientation.

A method described in [Ebe08] does not require an upright cone with the sensor point on the origin. A property described in [Ebe08] uses the field of view (FOV) of a cone, the

angle of the sensor. Based on the angle between the line from a given point to the sensor and the direction vector of the sensor, we can determine if the point is within the FOV or not. Further, [Ebe08] also describes a method to do an intersection test of a triangle and a cone using the previous property. The methods in both [Hel97] and [Ebe08] are adopted in SimulationTool to find the closest point in a conic sensing volume.

A distance sensor as in SimulationTool is also implemented in a virtual robot experimentation platform called v-rep developed by Coppelia Robotics [cop]. The distance sensor is also a sensor that computes the closest point and distance of an object visible in a sensing volume. A clear example of this sensor is shown in [vRe].

4.2 Design

In SimulationTool the distance sensor is modeled as a cone defined by four arguments. The cone has an angle α and a range d, the cone may be located at an arbitrary position $\mathbf{p}$ and has a likewise arbitrary orientation $\mathbf{v}$. Figure 4.1 shows such a sensor with the parameters in $\mathbb{R}^2$.

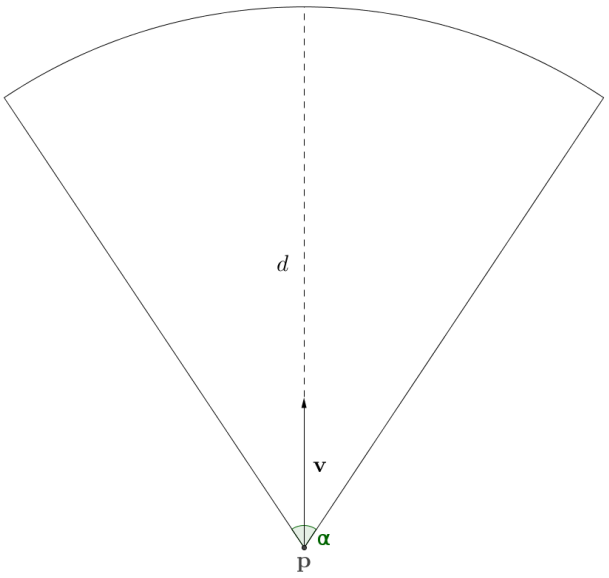

FIGURE 4.1: A sensor parameterized by α, d, $\mathbf{p}$ and $\mathbf{v}$.

The problem to solve is finding the point on a target object that is closest to the sensor and that lies in the cone (the sensing volume). From now on denoted as the closest point.

The GJK-algorithm algorithm does not solve this problem, however, the GJK-algorithm can be used to quickly verify if an object intersects the cone at all. Figure 4.2 shows a clear example where point **B** is closer to the sensor, whereas only point **A** is visible to the sensor. The Figure clarifies that a regular distance algorithm is not applicable for this task.

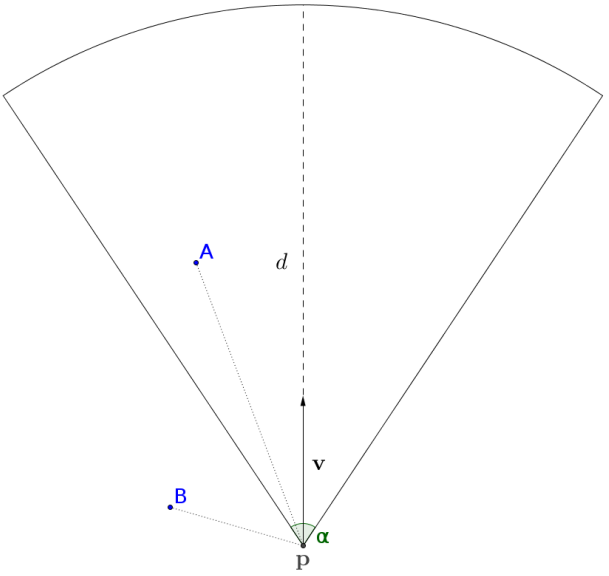

FIGURE 4.2: A sensor that senses **A**, although it is further away then **B**. The distance sensor senses only objects inside its sensing volume.

As said before, all objects in SimulationTool are built of triangles. To find the closest point on an object, we have to find the closest visible point on each triangle. In the set of found points we have to search for the one with the shortest distance to the sensor.

A first attempt to find the closest point on a triangle, is finding the closest point on the plane spanned by a triangle. A plane cutting a cone forms a conic section [Weic]. The shape of a conic section may be a circle, ellipse, parabola or a hyperbola, depending on the angle between the plane and **v** and α. The four distinguishable conic sections are shown in Figure 4.3.

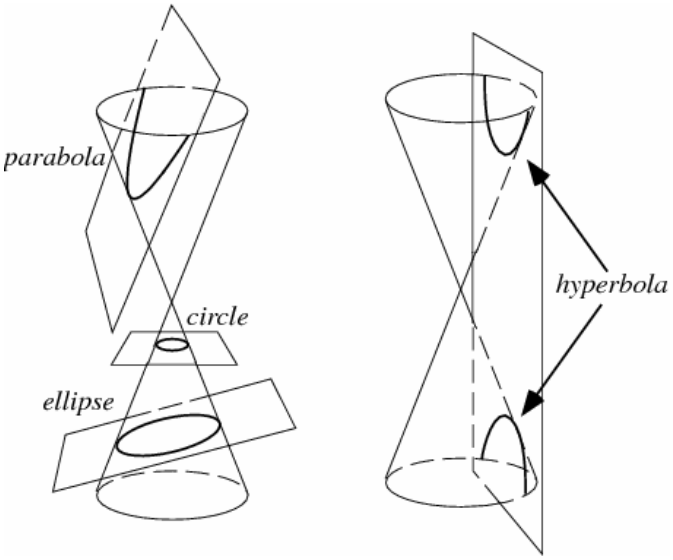

FIGURE 4.3: The four distinguishable conic sections. Source: [Weic]

On the left of Figure 4.4, we can see that the closest point, **Q**, is from the sensor directly to the plane, along the normal of the plane (from the sensor perpendicular to the plane). For the circular conic section this is always the case, but for other conic sections this may or may not be the case.

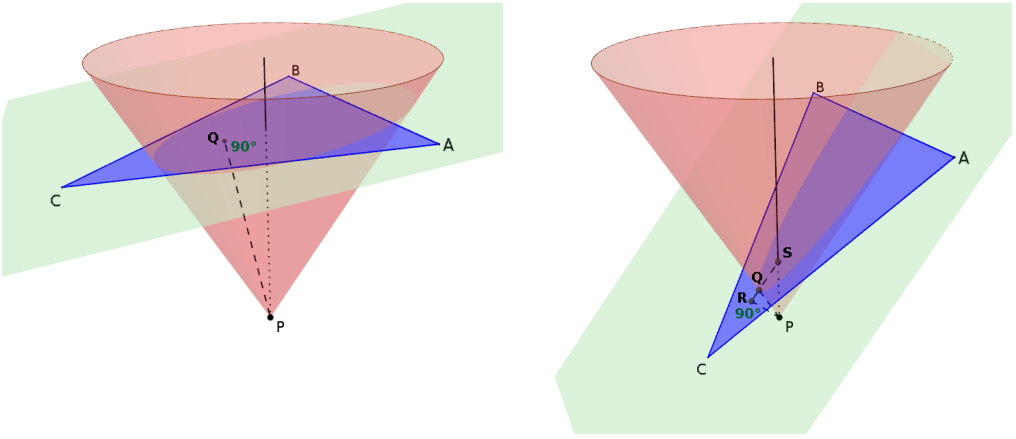

FIGURE 4.4: Left: The closest point is **Q**, from **P** perpendicular to **ABC**; Right: **R** is the closest point from **P** to **ABC**, but **Q** is the closest visible point from **P**.

On the right of Figure 4.4 we see a elliptical conic section with the point on the plane closest to **P** being **R**. However, **R** is outside the sensing volume, the closest visible point is point **Q**.

For all cases where the closest point on the cutting plane is outside the sensing volume, we have to find the closest point on the border of the conic section. We find that point by shooting a ray in the cutting plane, from the closest point on the plane (**R** in Figure 4.4), straight towards the center of the cone (**S** in Figure 4.4). The first intersection point (**Q** on the right in Figure 4.4) of the ray and the cone is the closest point on the conic section.

However, we are not looking for the closest point on a plane or conic section, but we are looking for the closest point on a triangle. For the closest point on a triangle we can distinguish five cases:

The closest point on the triangle may be

1. the closest point on the cutting plane, only valid if this point is in the interior of the triangle and inside the sensing volume. The case is clarified in Figure 4.5.

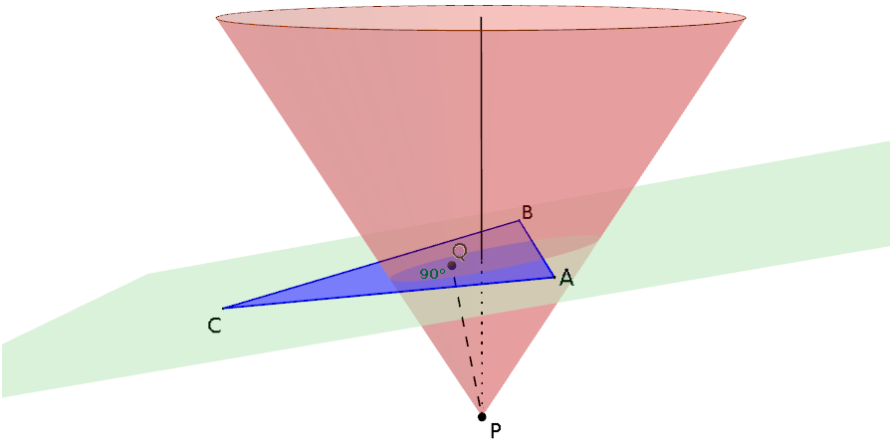

FIGURE 4.5: Case 1: Seen from the sensor, **Q** is the closest point on the triangle. **Q** is from **P** perpendicular to the cutting plane. Note that triangle **ABC** is entirely on the green plane.

2. the closest point on the border of the conic section, only valid if this point is in the interior of the triangle. The case is clarified in Figure 4.6.

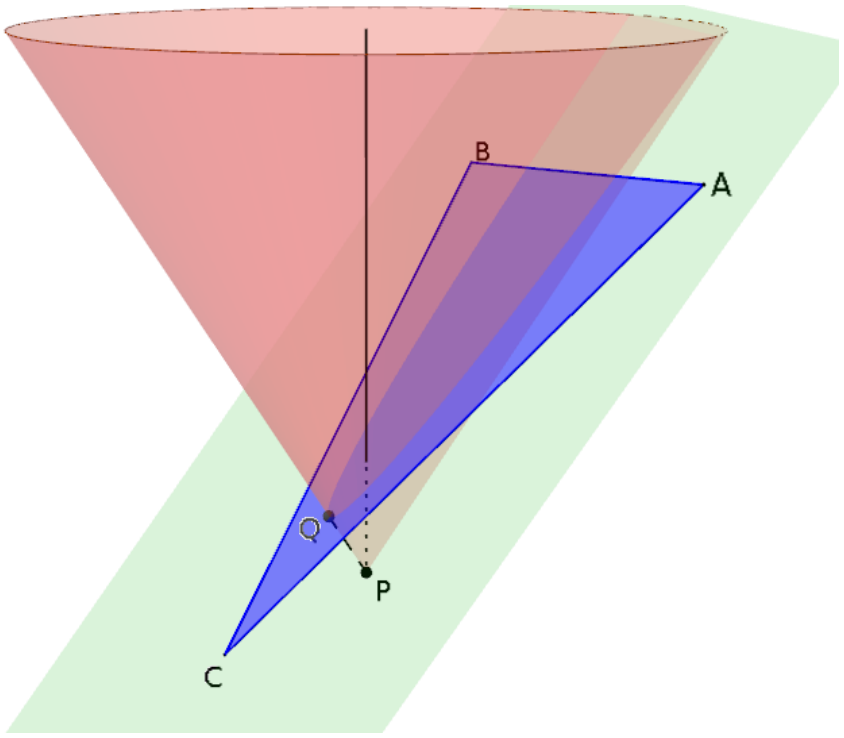

FIGURE 4.6: Case 2: Seen from the sensor, **Q** is the closest point on the triangle. **Q** is on the border of the conic section, closest to **P**. Note that triangle **ABC** is entirely on the green plane.

3. the closest point where an edge of the triangle intersects with the conic section.

   The case is clarified in Figure 4.7.

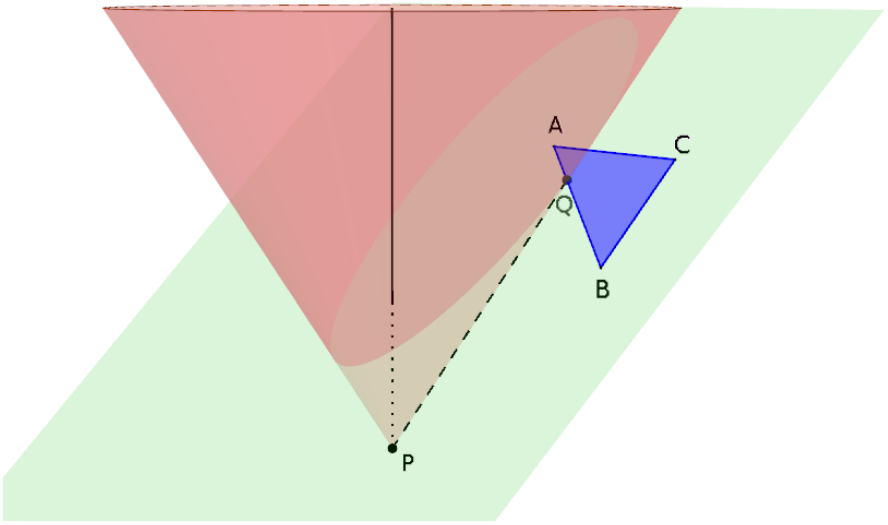

FIGURE 4.7: Case 3: Seen from the sensor, **Q** is the closest point on the triangle. **Q** is the intersection point of the cone and edge **AB**. Note that triangle **ABC** is entirely on the green plane.

4. from the sensor perpendicular to one of the edges of the triangle. The case is clarified in Figure 4.8.

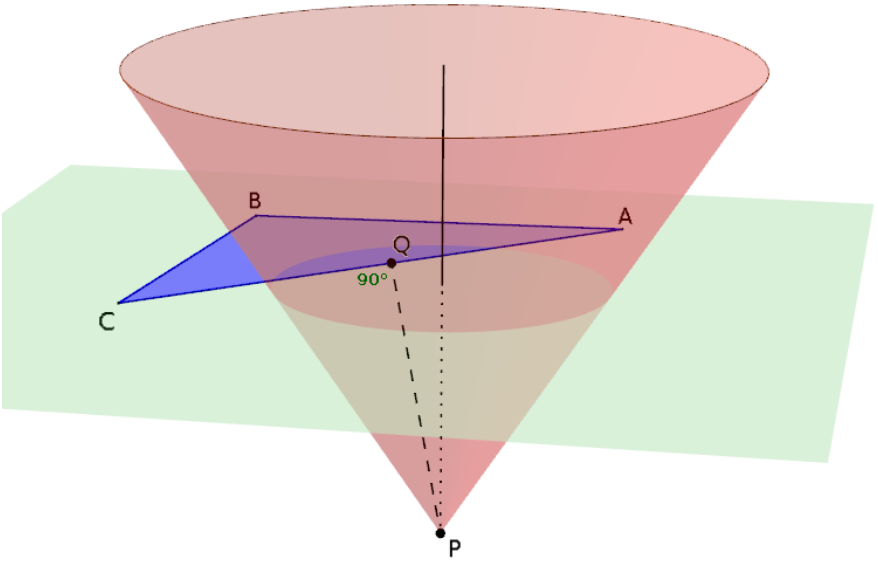

FIGURE 4.8: Case 4: Seen from the sensor, **Q** is the closest point on the triangle. **Q** is from **P** perpendicular to edge **AC**. Note, from **P** to **Q** is not perpendicular to triangle **ABC**. Note that triangle **ABC** is entirely on the green plane.

5. from the sensor directly to one of the vertices of the triangle. The case is clarified in Figure 4.9.

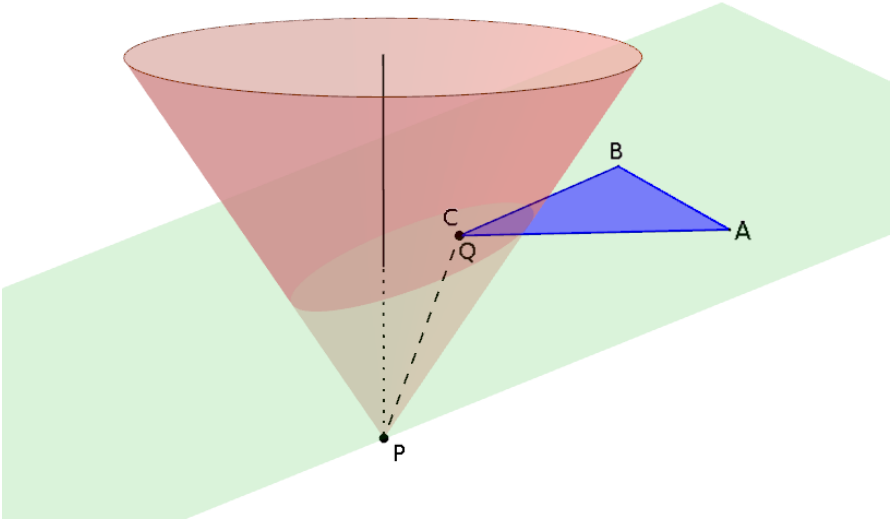

FIGURE 4.9: Case 5: Seen from the sensor, **Q** is the closest point on the triangle. **Q** is corner point **C** of **ABC**. Note that triangle **ABC** is entirely on the green plane.

Case 1, 4 and 5 are relatively easy to solve. Case 1 is the distance from P to the projection of P onto the triangle. Case 4 is the distance from P to the projection of P onto the closest edge of the triangle. Case 5 is simply the distance from P to the closest corner of the triangle.

For case 2 and 3 we need an algorithm that finds the closest intersection point of a line segment (or line) and a cone. We describe two methods to find this intersection point. The first is based on the mathematical equation of a cone, the second is based on the field of view of the sensor.

4.2.1 Method based on equation of a cone

The equation for the points on the border of the cone in Figure 4.1, is as follows [Weib]:

$$x^2 + y^2 = \tan(\frac{\alpha}{2})^2 z^2 \Leftrightarrow \tag{4.1}$$

$$x^2 + y^2 - \tan(\frac{\alpha}{2})^2 z^2 = 0 \tag{4.2}$$

Each intersection point of a line and a cone satisfies this equation.

Further, we have a parametric equation for a line in $\mathbb{R}^3$:

$$\mathbf{p} + t\mathbf{d} \tag{4.3}$$

where $\mathbf{p}$ is an arbitrary point on the line, $\mathbf{d}$ the direction vector and t a scalar. By varying t we can visit each point on the line. An edge e of a triangle, with begin point $\mathbf{a}$ and end point $\mathbf{b}$ can be written as a parametric equation as well:

$$\mathbf{a} + t\mathbf{d_e} = \mathbf{a} + t(\mathbf{b} - \mathbf{a}) \tag{4.4}$$

where $0 <= t <= 1$.

If edge e intersects a cone, then a value for t between 0 and 1 exists such that the parametric equation 4.4 expresses the intersection point. We do not know the exact value for t yet, but we do know that the point resulting from the parametric equation is on the boundary of the cone, therefor, equation 4.2 must hold for that point. We have two equations and one unknown so we can solve this for t.

Let the intersection point, called $\mathbf{q}(t)$, be defined as follows:

$$\mathbf{q}(t) = \mathbf{a} + t\mathbf{d_e} \tag{4.5}$$

$\mathbf{q}(t)$ is on the boundary of the cone if and only if $\mathbf{q}(t)$ satisfies equation 4.2:

$$\mathbf{q}(t)_x^2 + \mathbf{q}(t)_y^2 - \tan(\frac{\alpha}{2})^2\mathbf{q}(t)_z^2 = 0 \tag{4.6}$$

We substitute $\mathbf{q}(t)$ with its definition in Equation 4.5:

$$(\mathbf{a} + t\mathbf{d_e})_x^2 + (\mathbf{a} + t\mathbf{d_e})_y^2 - \tan(\frac{\alpha}{2})^2(\mathbf{a} + t\mathbf{d_e})_z^2 = 0 \Leftrightarrow$$
$$(\mathbf{a}_x + t\mathbf{d_e}_x)^2 + (\mathbf{a}_y + t\mathbf{d_e}_y)^2 - \tan(\frac{\alpha}{2})^2(\mathbf{a}_z + t\mathbf{d_e}_z)^2 = 0$$

Which is equivalent to:

$$\left(\mathbf{d_e}_x^2 + \mathbf{d_e}_y^2 - \tan(\frac{\alpha}{2})^2\mathbf{d_e}_z^2\right)t^2 +$$
$$2\left(\mathbf{a}_x\mathbf{d_e}_x + \mathbf{a}_y\mathbf{d_e}_y - \tan(\frac{\alpha}{2})^2\mathbf{a}_z\mathbf{d_e}_z\right)t +$$
$$\left(\mathbf{a}_x^2 + \mathbf{a}_y^2 - \tan(\frac{\alpha}{2})^2\mathbf{a}_z^2\right) = 0$$

Which is of the form $at^2 + bt + c$ and therefore it can be solved with the *abc*-formula. In general, the *abc*-formula results in a t_1 and a t_2. The computed values for t_1 and t_2 have to lie in the interval $[0, 1]$. If t_1 or t_2 is not in this interval, then the corresponding intersection point is not between $\mathbf{a}$ and $\mathbf{b}$. To compute the actual intersection points we simply fill in the values for t_1 and t_2 in Equation 4.5.

This method solves the problem, but comes with a few caveats: this only works for an upright cone that starts in the origin. A sensor in SimulationTool can be located and oriented arbitrarily. Using this method enforces rotation and translation of the scene, such that the sensor is an upright cone starting in the origin.

4.2.2 Method based on the field of view of the cone

This method has the advantage that it can be used for a sensor with arbitrary position and orientation, without translating and rotating the whole scene, which can result in improved performance.

This method is not based on the equation of a cone, but rather on the fact that the angle between the line from the sensor to a point on the boundary of the cone and the direction of the sensor, $\mathbf{v}$, must be equal to $\frac{\alpha}{2}$. This relation is clearly illustrated in Figure 4.10.

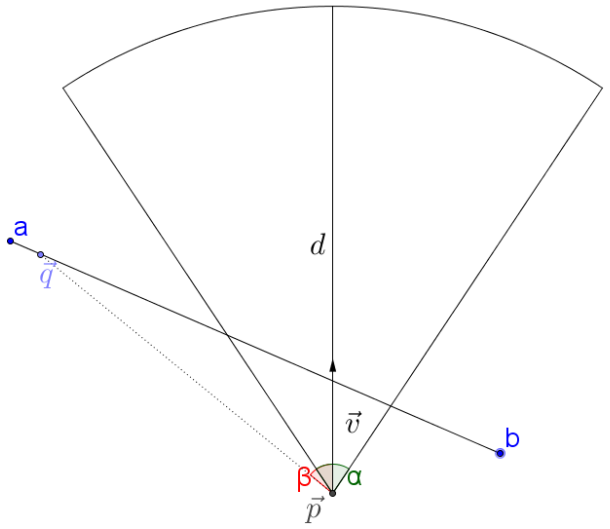

FIGURE 4.10: Point $\mathbf{q}$ is an intersection point only if $\beta = \frac{\alpha}{2}$.

In the figure we see an edge, with begin point $\mathbf{a}$ and end point $\mathbf{b}$, that intersects the sensor cone. Further we see an arbitrarily chosen point $\mathbf{q}$ on edge $\mathbf{ab}$. $\mathbf{q}$ would be intersecting the cone if and only if $\beta = \frac{\alpha}{2}$ [Ebe08].

We can compute the angle α between two arbitrary vectors $\mathbf{u_1}$ and $\mathbf{u_2}$ with the inner product,

$$\mathbf{u_1} \cdot \mathbf{u_2} = ||\mathbf{u_1}||\ ||\mathbf{u_2}|| \cos(\alpha) \tag{4.7}$$

To evaluate angle β in Figure 4.10, we can use equation 4.7:

$$\mathbf{v} \cdot (\mathbf{q} - \mathbf{p}) = ||\mathbf{v}||\ ||\mathbf{q} - \mathbf{p}|| \cos(\beta) \tag{4.8}$$

If $\mathbf{v}$ is unit length, Equation 4.8 becomes:

$$\mathbf{v} \cdot (\mathbf{q} - \mathbf{p}) = ||\mathbf{q} - \mathbf{p}|| \cos(\beta) \tag{4.9}$$

To avoid calculating the norm (an expensive square root calculation) of $\mathbf{q} - \mathbf{p}$ we use the quadratic variant of Equation 4.9:

$$(\mathbf{v} \cdot (\mathbf{q} - \mathbf{p}))^2 = ||\mathbf{q} - \mathbf{p}||^2 \cos^2(\beta) \tag{4.10}$$

Note that solving Equation 4.10 also finds solutions for the mirrored cone, an intersection point of the cone in direction -$\mathbf{v}$. This can easily be checked by verifying that:

$$\mathbf{v} \cdot (\mathbf{q} - \mathbf{p}) \geq 0 \tag{4.11}$$

In the same manner as solving Equation 4.6 we can solve Equation 4.10 for t. $\mathbf{q}$ is again a point on the edge from $\mathbf{a}$ to $\mathbf{b}$ as in Equation 4.5, substituting the definition of $\mathbf{q}$ in Equation 4.10 gives:

$$(\mathbf{v} \cdot (\mathbf{q} - \mathbf{p}))^2 = ||\mathbf{q} - \mathbf{p}||^2 \cos^2(\beta) \Leftrightarrow \tag{4.12}$$

$$(\mathbf{v} \cdot (\mathbf{q}(t) - \mathbf{p}))^2 = ||\mathbf{q}(t) - \mathbf{p}||^2 \cos^2(\beta) \Leftrightarrow \tag{4.13}$$

$$(\mathbf{v} \cdot ((\mathbf{a} + t\mathbf{d_e}) - \mathbf{p}))^2 = ||(\mathbf{a} + t\mathbf{d_e}) - \mathbf{p}||^2 \cos^2(\beta) \Leftrightarrow \tag{4.14}$$

$$(\mathbf{v} \cdot ((\mathbf{a} + t\mathbf{d_e}) - \mathbf{p}))^2 - ||(\mathbf{a} + t\mathbf{d_e}) - \mathbf{p}||^2 \cos^2(\beta) = 0 \tag{4.15}$$

Which can be rewritten to ([Ebe08]):

$$\left((\mathbf{v} \cdot \mathbf{d_e})^2 - ||\mathbf{d_e}||^2 \cos^2(\beta)\right) t^2 +$$
$$2\left((\mathbf{v} \cdot \mathbf{d_e})(\mathbf{v} \cdot (\mathbf{a} - \mathbf{p})) - (\mathbf{d_e} \cdot (\mathbf{a} - \mathbf{p})) \cos^2(\beta)\right) t +$$
$$\left((\mathbf{v} \cdot (\mathbf{a} - \mathbf{p}))^2 - ||\mathbf{a} - \mathbf{p}||^2 \cos^2(\beta)\right) = 0$$

This is again of the form $at^2 + bt + c = 0$ and thus can be solved with the *abc*-formula. Since we are looking for intersection points, we simply substitute β with $\frac{\alpha}{2}$. Just as the method in Section 4.2.1, the computed values for t_1 and t_2 have to lie in the interval $[0, 1]$. If t_1 or t_2 is not in this interval, then the corresponding intersection point is not

between **a** and **b**. To compute the actual intersection points we simply fill in the values for t_1 and t_2 in Equation 4.5.

4.3 Result

In Figure 4.11 we see the visualization of the distance sensors.

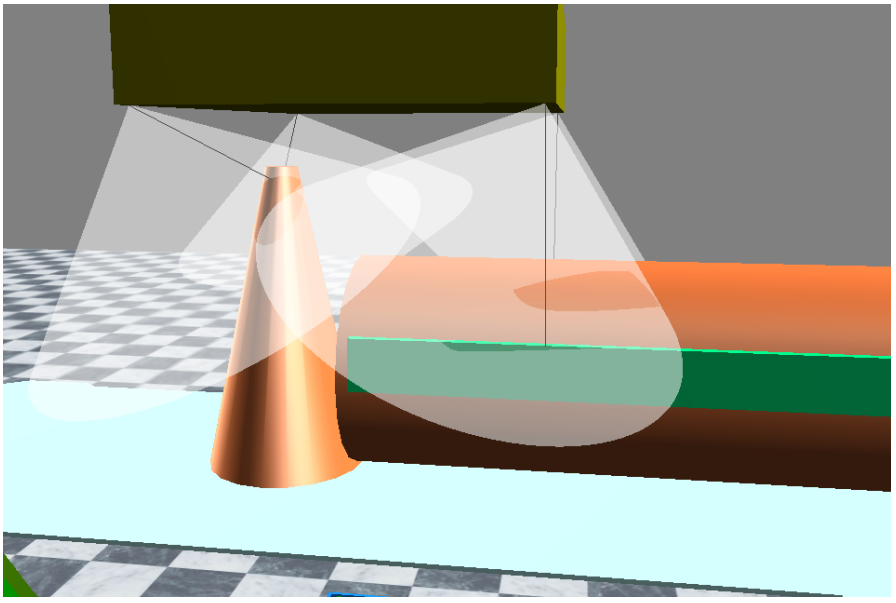

FIGURE 4.11: For this study, we chose a case of having a distance sensor at each corner.

For this study, we chose a case of having a distance sensor at each corner. The black lines point from the sensors to the closest visible point respectively. The two left sensors found the closest points on the head of the patient. The closest points for the two right sensors are on the arms of the patient.

In multiple situations, the distance sensor would be a very interesting sensor for Philips Healthcare. However, in a lot of the situations, there is a high risk that the sensor gets obscured, which results in unusable output voltages. For example, a protection cover can be placed around the machine for hygienic reasons. This cover may obscure the sensors. Further, any cable or fluid, or other obstacle in front of the sensor may hinder the reliability of the sensor as well. For these reasons the interventional X-Ray machines are equipped with capacitive sensors.

5 | Capacitive sensor

As explained, the interventional X-Ray machines developed by Philips Healthcare are equipped with capacitive sensors. REMOVED DUE TO CONFIDENTIALITY. The bodyguard is a hood placed around the X-Ray scanner. Figure 5.1 shows a real bodyguard.

FIGURE 5.1: The bodyguard. Normally this hood is placed around the X-Ray scanner.

The sensors in the bodyguard sense the proximity of nearby conductive objects. Such a sensor generates an electric field surrounding the sensor and any disturbance in this field, due to the presence of conductive objects, changes the output voltage of the sensor. A schematic overview of where one of the x sensors is located is shown in Figure 5.2.

FIGURE 5.2: Illustration of the bodyguard of an interventional X-Ray machine

In SimulationTool a capacitive sensor consists of x sensor plates, the x plates are shown in Figure 5.3. Note that in the real bodyguard a capacitive sensor is one connected piece.

FIGURE 5.3: The three sensor plates that construct one capacitive sensor in SimulationTool

When detecting an object in its proximity, the task of the model for the capacitive sensor is to predict output of the real capacitive sensor in a similar situation. In SimulationTool, each object is a polyhedron consisting of triangles.

Thus, the task is to predict the output of the real sensor when given a polyhedron near the sensor like in Figure 5.4.

FIGURE 5.4: A polyhedron near the capacitive sensor.

To explain how the output can be approximated for a polyhedron, we first discuss how to approximate the output with only one triangle in front of only one sensor plate, like the situation shown in Figure 5.5.

FIGURE 5.5: A triangle near a capacitive sensor plate.

In Figure 5.6 we see the front view of a sensor.

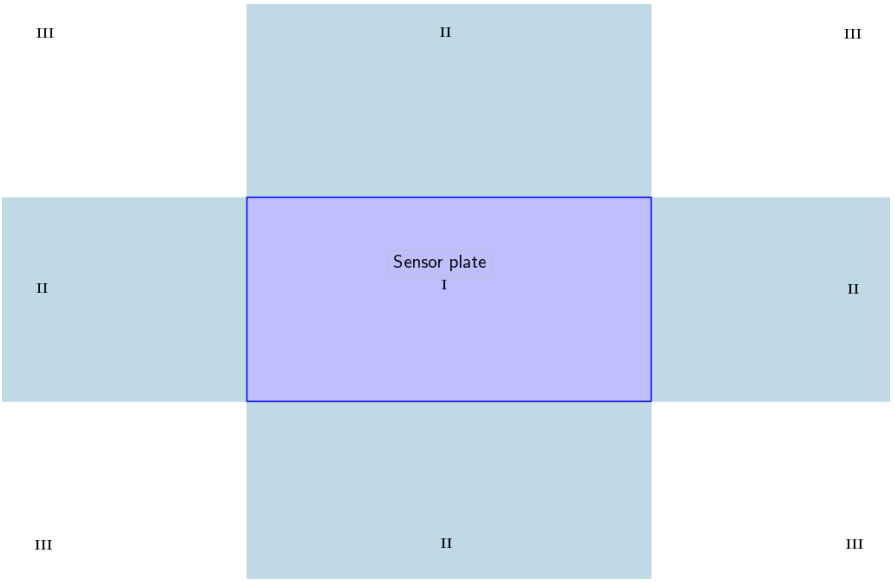

FIGURE 5.6: The three distinguishable regions around a sensor plate.

We can observe that a triangle can be in three regions. The regions are distinguished because the distance from a point to the sensor is different in each region:

Region I the shortest distance from a point to the sensor is directly to the plate.

Region II the shortest distance from a point to the sensor is to the nearest edge of the plate.

Region III the shortest distance from a point to the sensor is to the nearest corner of to the plate.

These distances are used to approximate the capacitance of a triangle. In each region another approximation method is used.

The structure of this chapter is as follows: First we discuss related work and some preliminaries in Sections 5.1 and 5.2. In Sections 5.3.1, 5.3.2 and 5.3.3 we explain how to approximate the capacitance of a triangle in region I, II or III respectively. In Section 5.3.4 an approach is given to approximate the capacitance of an object. Based on the capacitances of all the objects a method to compute the output voltages is given in Section 5.3.5. Lastly, the results and limitations are discussed in Section 5.4.

5.1 Related work

A less discussed topic is capacitive proximity sensors for the protection of people near machines. An even less discussed topic is capacitive sensors in simulations. N. Karlsson [KJ93] describes a capacitive sensor that detects presence of a human near a machine, by placing a sending antenna on the floor and a receiving antenna on the ceiling. When a person is too close to the machine, the electric field between the antennae is disrupted, which is detected by the system. The capacitive sensors in the interventional X-Ray machines consist of only a sending antenna and not a receiving antenna. Despite this difference, the principle of a capacitive sensor is clearly described by N. Karlsson. The electric circuit described by N. Karlsson is similar to the circuit used for the interventional X-Ray machines. A similar set up is described by C. Jiang in [JLSC91]. [JLSC91] is an overview of machine guarding techniques protecting people nearby machines. Jiang describes the capacitive sensor as a sensor with high visibility, but with the disadvantage that the system requires regular calibration and maintenance.

The previously-mentioned two papers describe the capacitive sensor as a sensor to detect the presence of a person, i.e. the sensor is not used to measure the distance to a person. R.N. Aguilar [ARM] describes a capacitive human detection system that estimates the distance between a person and a sensor. More interesting and related to Philips Healthcare's interventional X-Ray machines is that they estimate the location of a person and not just the distance. A main difference between the setup and the sensors developed by Philips Healthcare is that the latter are close to the body and operate on shorter distances. The setup in [ARM] always operates on the entire human body, not only on a part of the body.

Furthermore, Y. Xiang [Xia06] proposed a model to precisely calculate the capacitance of two non-parallel plates. Since it is a mathematical model, it is applicable in a simulation. J.M. Bueno [BBCPCI11] shows that the Finite Element Method (FEM) is applicable to calculate the capacitance of two non-parallel plates. To verify that FEM is indeed applicable, the results are compared to the model devised by Xiang [Xia06]. The approach of applying FEM in [BBCPCI11] is further clarified in [Hos15]. They describe two inclined plates as a capacitor. One plate is split up in very small beams, each beam is considered as a parallel plate to the other plate. The method results in a integral

that approximately solves for the capacitance of the capacitor formed by the two plates. This approach is similar to the method implemented in SimulationTool. The objects in SimulationTool are all built out of triangles. In SimulationTool a similar method is applied to the triangles that are close to a sensor. For the triangles in SimulationTool we can devise a similar integral. In one case, the integral can be solved analytically, in other cases we discretize the integral.

5.2 Preliminaries

5.2.1 Barycentric coordinates

To describe a point on a triangle in SimulationTool we use a rearrangement of the barycentric coordinates [Weia] to describe an arbitrary point $\mathbf{b}$ on the surface.

Assume we have a triangle defined by its three corner points $\mathbf{p_0}$, $\mathbf{p_1}$ and $\mathbf{p_2}$. Let the edge from $\mathbf{p_0}$ to $\mathbf{p_1}$ be $\mathbf{u} = \mathbf{p_1} - \mathbf{p_0}$ and the edge from $\mathbf{p_0}$ to $\mathbf{p_2}$ be $\mathbf{v} = \mathbf{p_2} - \mathbf{p_0}$. With the barycentric coordinates λ_0 and λ_1 we can describe each point $\mathbf{b}$ on the triangle as follows:

$$\begin{aligned}\mathbf{b}(\lambda_0, \lambda_1) &= \mathbf{p_0} + \lambda_0 \left(\mathbf{p_1} - \mathbf{p_0}\right) + \lambda_1 \left(\mathbf{p_2} - \mathbf{p_0}\right) \\ &= \mathbf{p_0} + \lambda_0 \, \mathbf{u} + \lambda_1 \, \mathbf{v}\end{aligned}$$

where $\lambda_0, \lambda_1 \in [0, 1] \ \wedge \ \lambda_0 + \lambda_1 \leq 1$.

Integrating a function f over the surface of a triangle T using barycentric coordinates is particularly easy [Got00]:

$$\begin{aligned}\int_T f(\mathbf{b}) \, \mathrm{d}A \ &= \mathrm{S} \int_0^1 \int_0^{1-\lambda_1} f(\mathbf{b}(\lambda_0, \lambda_1)) \ \mathrm{d}\lambda_0 \mathrm{d}\lambda_1 \\ &= ||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} f(\mathbf{p_0} + \lambda_0 \, \mathbf{u} + \lambda_1 \, \mathbf{v}) \ \mathrm{d}\lambda_0 \mathrm{d}\lambda_1\end{aligned}$$

where S is the surface element, which equals two times the area of the triangle and $\mathbf{b}$ is the barycentric parametrization of point on the surface.

5.2.2 Projections

To compute the shortest distance from a point $\mathbf{b}$ on a triangle to a sensor plate we project the point on the plate, lets call the projection point $\mathbf{b}'$. The shortest distance is the distance from the $\mathbf{b}$ to $\mathbf{b}'$. Figure 5.7 illustrates a projection in $\mathbb{R}^3$ seen from above.

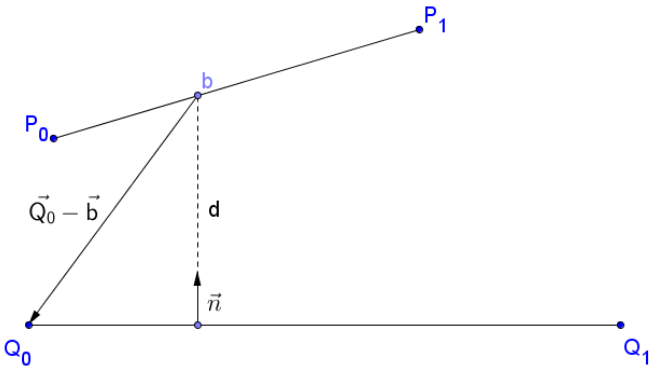

FIGURE 5.7: Projection in $\mathbb{R}^3$ seen from above.

$\mathbf{q_0}$ and $\mathbf{q_1}$ are two corner points of the sensor plate.$\mathbf{n}$ is the normal vector of the plate: the unit vector perpendicular to the line. d is the distance between $\mathbf{b}$ and $\mathbf{b}'$, the number we are looking for. We can compute d as follows [Weie]:

$$d \;=\; (\mathbf{q_0} - \mathbf{b}) \cdot -\mathbf{n} \;=\; (\mathbf{b} - \mathbf{q_0}) \cdot \mathbf{n}$$

where $\cdot$ is the dot / inner product.
With Barycentric coordinates we get:

$$\begin{aligned} d &= (\mathbf{b} - \mathbf{q_0}) \cdot \mathbf{n} \\ &= (\mathbf{b}(\lambda_0, \lambda_1) - \mathbf{q_0}) \cdot \mathbf{n} \\ &= (\mathbf{p_0} + \lambda_0 \mathbf{u} + \lambda_1 \mathbf{v} - \mathbf{q_0}) \cdot \mathbf{n} \\ &= \mathbf{p_0} \cdot \mathbf{n} + \lambda_1 \, \mathbf{u} \cdot \mathbf{n} + \lambda_0 \, \mathbf{v} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n} \end{aligned}$$

5.2.3 Properties of capacitances

Law of capacitance

For two parallel plates we have the following formula to calculate the capacitance

C between the plates:
$C = \frac{\varepsilon \cdot A}{d}$
where ε is the dielectric constant, A is the area of the plates and d is the distance between the plates.

Law of parallel capacitances

Multiple capacitances connected in parallel may be added to calculate the total capacitance:

$C = C_1 + C_2 + ... + C_{n-1} + C_n$

Voltage divider

Two electrical components connected in series, as in Figure 5.8, can act as a voltage (potential) divider.

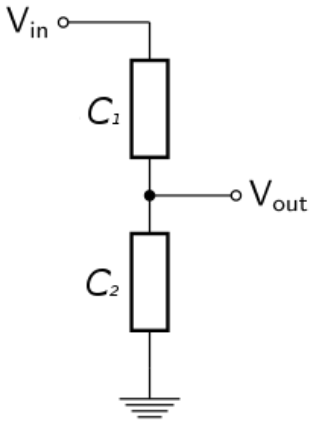

FIGURE 5.8: Simple circuit of a voltage divider

Two capacitors, C_1 and C_2, connected in series act as a voltage divider with the following behavior:
$V_{out} = \frac{C_1}{C_1 + C_2} V_{in}$, where V_{in} is the input voltage and V_{out} the voltage across capacitor C_2.

5.3 Design

As mentioned above, J.M. Bueno [BBCPCI11] shows that the Finite Element Method (FEM) is applicable to calculate the capacitance of two non-parallel plates. Figure 5.9 shows an example of two non-parallel plates.

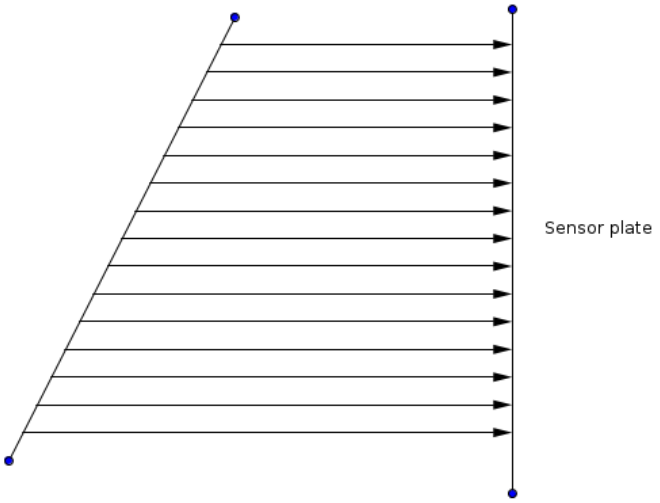

FIGURE 5.9: Example of a triangle in front of a sensor plane.

J.M. Bueno describes an integral over the area of the plates that accurately approximates the capacitance of the two plates. Figure 5.10 illustrates what the integral means. The left plate is split up in two plates. By treating the two smaller plates as parallel to the larger plane, we can approximate the capacitance. If we keep splitting the plates up, we end up with infinitely small parallel plates. By using the law of parallel capacitances, we can add up the capacitances of all small plates to approximate the total capacitance between the two plates in Figure 5.9.

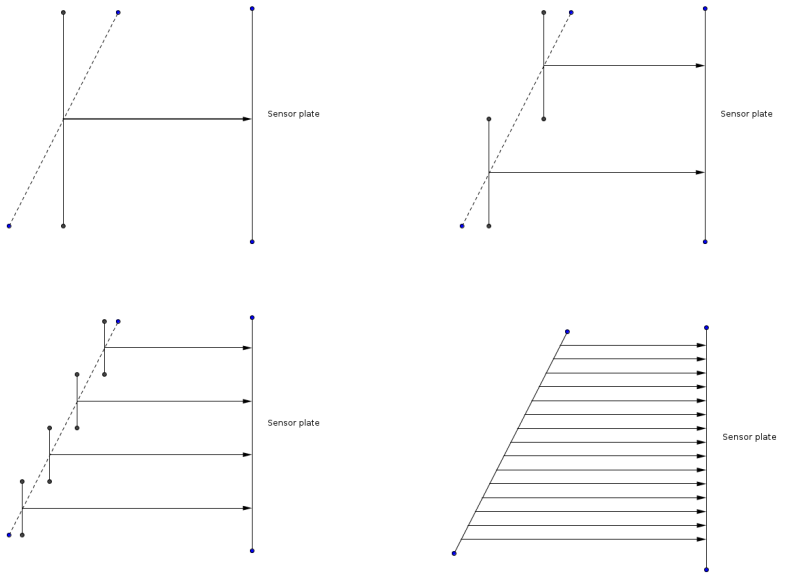

FIGURE 5.10: Using infinite many small parallel plates, we can approximate the capacitance of the two large inclined plates.

For SimulationTool we follow a similar approach, however, we do not integrate over plates but over triangles.

5.3.1 Capacitance of a triangle in front of a sensor plate

Assume a sensor plate that is defined by the four points $\mathbf{q_0}$, $\mathbf{q_1}$, $\mathbf{q_2}$ and $\mathbf{q_3}$. Vector $\mathbf{n}$ is the unit vector perpendicular to the rectangle called the normal vector of the sensor.

Now, assume an arbitrary triangle in front of the sensor plate that is defined by the three points $\mathbf{p_0}$, $\mathbf{p_1}$ and $\mathbf{p_2}$. Edge $\mathbf{u} = \mathbf{p_1} - \mathbf{p_0}$ which is the vector from $\mathbf{p_0}$ to $\mathbf{p_1}$, edge $\mathbf{v} = \mathbf{p_2} - \mathbf{p_0}$ which is the vector from $\mathbf{p_0}$ to $\mathbf{p_2}$.

Figure 5.11 illustrates such triangle in front of the sensor plate.

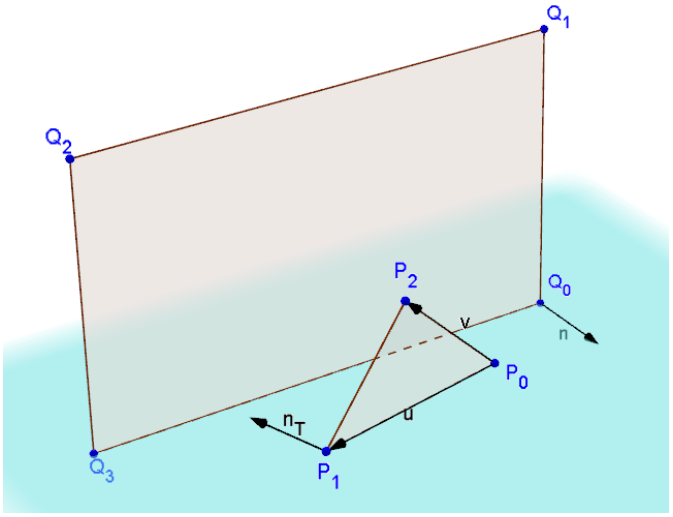

FIGURE 5.11: Example of a triangle in front of a sensor plane.

Combining the law of capacitance and the law of parallel capacitances allows us to approximate the capacitance of triangle T by integrating over Ts surface:

$$C = \int \mathrm{d}C = \int_T \frac{\varepsilon}{\delta d}\,\mathrm{d}A = \varepsilon \int_T \frac{1}{\delta d}\,\mathrm{d}A$$

where δd is the distance function, i.e. the shortest distance from a point on the triangle to the sensor plate. Note that the distance to the sensor plate differs per point on the triangle.

Integration using Barycentric coordinates leads to

$$C = \varepsilon\,||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\delta d}\,\mathrm{d}\lambda_0 \mathrm{d}\lambda_1$$

As explained in Section 5.2, to compute the shortest distance we have to compute the distance from the point to its projection on the plate.

$$C = \varepsilon\,||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\mathbf{p_0} \cdot \mathbf{n} + \lambda_1\,\mathbf{u} \cdot \mathbf{n} + \lambda_0\,\mathbf{v} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n}}\,\mathrm{d}\lambda_0 \mathrm{d}\lambda_1$$

Substituting $a = \mathbf{u} \cdot \mathbf{n}$, $b = \mathbf{v} \cdot \mathbf{n}$ and $c = \mathbf{p_0} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n}$ gives:

$$C = \varepsilon \, ||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{a\lambda_1 + b\lambda_0 + c} \, \mathrm{d}\lambda_0 \mathrm{d}\lambda_1$$
$$= \frac{\varepsilon \, ||\mathbf{u} \times \mathbf{v}||}{b} \Big(\frac{b(a+c)}{a(a-b)} \, ln(a+c) \; - \; \frac{b+c}{a-b} \, ln(b+c) \; + \; \frac{c}{a} \, ln(c) \Big)$$

How this integral is solved is explained step by step in Appendix D.

5.3.2 Capacitance of a triangle next to a sensor plate

Here we discuss how to compute the capacitance of a triangle next to a sensor plate. This situation differs from the situation in Section 5.3.1 because the triangle is located in another region. In this situation the triangle is located in region II, the marked region in Figure 5.12.

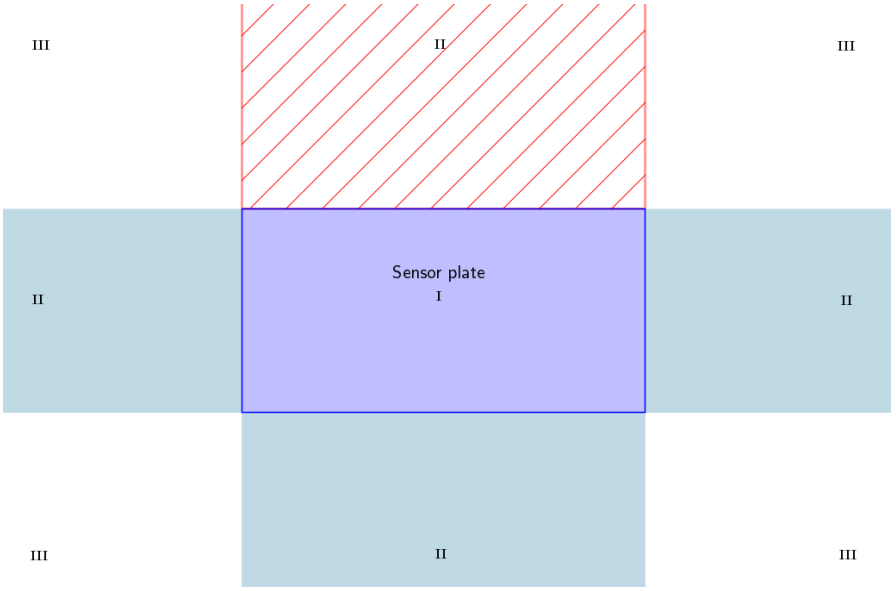

FIGURE 5.12: Illustrating region II. The shortest distance from a point on a triangle in region II to the sensor is to straight to the edge of the sensor plate.

Now the integral differs from the integral in Section 5.3.1 because the shortest distance from a point **b** on the triangle to the sensor plate is not calculated by projecting **b** on the plate. I.e., we have to use a different distance function. We have to compute the shortest distance from **b** to the edge of the sensor plate. The equation to compute the shortest distance from a point **b** to the edge of the sensor plate with end points $\mathbf{q_0}$ and

$\mathbf{q_1}$ is as follows [Weid]:

$$d = \frac{||(\mathbf{b} - \mathbf{q_0}) \times (\mathbf{b} - \mathbf{q_1})||}{||\mathbf{q_1} - \mathbf{q_0}||}$$

where $\times$ is the cross product.

Analytically solving

$$C = \varepsilon \, ||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\delta d} \, \mathrm{d}\lambda_0 \mathrm{d}\lambda_1$$

with the new distance function does not seem to be possible. However, we can solve the integral numerically. To do so we have to discretize the integral. We can discretize the integral by splitting the triangle in 4 smaller but equal sized triangles. We keep repeating the process until we see no significant improvement in the approximation for the capacitance. How to split the triangle is illustrated in Figure 5.13.

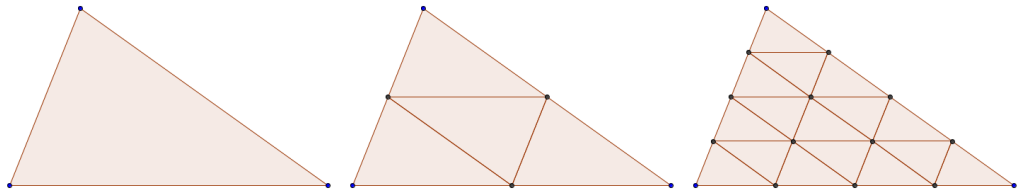

FIGURE 5.13: Process of splitting a triangle in equally sized similar triangles.

We chose this method of splitting the triangles, because it results in a balanced spread and equally-sized similar triangles. In the end we have a set S of 4^n of these triangles, where n is the number of times we split the triangles.

For each triangle S_i with centroid $\mathbf{m_i}$ we can approximate the distance:

$$d_i \approx \frac{||(\mathbf{m_i} - \mathbf{q_0}) \times (\mathbf{m_i} - \mathbf{q_1})||}{||\mathbf{q_1} - \mathbf{q_0}||}$$

Now we can approximate C using:

$$\begin{aligned} C &= \varepsilon \, ||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\delta d} \, \mathrm{d}\lambda_0 \mathrm{d}\lambda_1 \\ &\approx \frac{\varepsilon \, ||\mathbf{u} \times \mathbf{v}||}{2 \cdot 4^n} \sum_{i=1}^{4^n} \frac{1}{d_i} \\ &\approx \frac{\varepsilon \, ||\mathbf{u} \times \mathbf{v}||}{2 \cdot 4^n} \sum_{i=1}^{4^n} \frac{||\mathbf{q_1} - \mathbf{q_0}||}{||(\mathbf{m_i} - \mathbf{q_0}) \times (\mathbf{m_i} - \mathbf{q_1})||} \end{aligned}$$

5.3.3 Capacitance of a triangle diagonally to a sensor plate

Lastly, we discuss how to compute the capacitance of a triangle diagonally to a sensor plate. This situation differs from both the situation in Section 5.3.1 and the situation in Section 5.3.2 because the triangle is located in region III, the region marked in Figure 5.14.

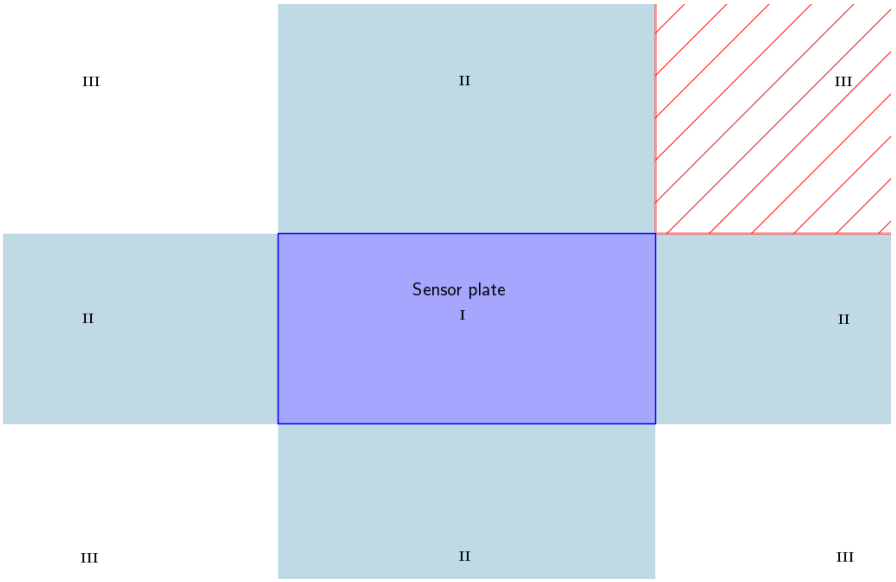

FIGURE 5.14: Illustrating region III. The shortest distance from a point on a triangle in region III to the sensor is to straight to the corner of the sensor plate.

Now the integral differs again because the shortest distance from a point $\mathbf{b}$ on the triangle to sensor plate is straight to the corner of the sensor plate. The distance from $\mathbf{b}$ to corner point $\mathbf{q_0}$ (in $\mathbb{R}^3$) is as follows:

$$d = ||\mathbf{q_0} - \mathbf{b}|| = \sqrt{\sum_{i=1}^{3} (\mathbf{q_0}_i - \mathbf{b}_i)^2}$$

Again, analytically solving

$$C = \varepsilon \, ||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\delta d} \, \mathrm{d}\lambda_0 \mathrm{d}\lambda_1$$

with the new distance function does not seem to be possible.

We numerically integrate in a similar fashion as in Section 5.3.2. We split up triangle T in a set S of 4^n small, equally sized similar triangles, where n is again the number of times we split the triangles. For each triangle S_i with centroid m_i we can approximate the distance:

$$d_i \approx ||\mathbf{q_0} - \mathbf{m_i}|| = \sqrt{\sum_{j=1}^{3} (\mathbf{q_0}_j - \mathbf{m_i}_j)^2}$$

Now we can approximate C using:

$$\begin{aligned} C &= \varepsilon \, ||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\delta d} \, \mathrm{d}\lambda_0 \mathrm{d}\lambda_1 \\ &\approx \frac{\varepsilon \, ||\mathbf{u} \times \mathbf{v}||}{2 \cdot 4^n} \sum_{i=1}^{4^n} \frac{1}{d_i} \\ &\approx \frac{\varepsilon \, ||\mathbf{u} \times \mathbf{v}||}{2 \cdot 4^n} \sum_{i=1}^{4^n} \frac{1}{||\mathbf{q_0} - \mathbf{m_i}||} \end{aligned}$$

5.3.4 Capacitance of an object

Now we know how to approximate the capacitance of a triangle, we can approximate the capacitance of an object. If an object is charged, the charges repel each other and distribute over the surface of the object. Therefore, to approximate the capacitance of an object, we can approximate the capacitance of the surface of the object. The objects in SimulationTool are polyhedra constructed by triangles, these triangles describe the surface of the object. By the law of parallel capacitances, we can approximate the capacitance of the object by adding the capacitances of these triangles.

Let the triangles be the set $T = \{T_1, T_2, ..., T_n - 1, T_n\}$. The total capacity can be computed as follows:

$$C_{Total} = \sum_{i=1}^{n} C_{T_i}$$

For each capacitive sensor, we can compute the total capacitance of all objects. From this total capacitance per sensor we can compute the output voltages.

5.3.5 From total capacitances to output voltage

In an interventional X-Ray machine of Philips Healthcare, a circuit like the simplified voltage divider in Figure 5.8 is used, where V_{out} is the measured output , C_1 is a couple capacitor and C_2 is the capacitance of the capacitive sensor.

In SimulationTool we implemented a model for the voltage divider as well. The voltage divider uses 2.5 Volt as input voltage. In the model for the capacitive sensor, the couple capacitor and the dielectric constant are adjustable to fine tune the model.

Given the total capacitance of a sensor, we can predict the output voltage using the following formula:

$$V_{out} = \frac{C_{couple}}{C_{couple} + C_{Total}} V_{in}$$

5.4 Result

The capacitive sensor cannot be visualized as clearly as the distance sensor. But, in Figure 5.15 we see x of the x capacitive sensors on the bottom of the bodyguard. In the top right we see the output of the sensors.

FIGURE 5.15: Figure of SimulationTool with x capacitive sensors.

A low output voltage is the result of a high capacitance. One of the sensors is right above the head of the patient. The output of this sensor is shown by the fourth bar in the corner of the screen. A close object results in a high capacitance. When the sensor is close to the human body, the presence of the body increases the capacitance

and thus lowers the output voltage. Another sensor is further away from the body, the capacitance is not as high as with the first sensor. The behavior of the model for the capacitive sensor is further tested and validated in Section 6.

Limitations

In the real bodyguard, the different capacitive sensors influence each other. A conductive object near one of the sensors results in a potential difference with the other sensors. The effect is illustrated in Figure 5.16.

FIGURE 5.16: Illustrating the potential difference between two sensors.

In the current model of SimulationTool the capacitive sensors are handled separately, i.e. there is no interaction between two capacitive sensors.

Furthermore, the model in SimulationTool assumes objects to be very conductive and well grounded. However, not all objects near the real sensor are very conductive or well grounded.

FIGURE 5.17: Currently, the model assumes that objects are very conductive and well grounded. There is no distinction between different materials.

For example the apple in Figure 5.17 may have a totally different conductivity than a human body. Even different human bodies can have a different conductivity. The conductivity will eventually have some influence on the output of the sensor.

For both the potential difference between two sensors and the different materials, future work is needed.

6 Experiments and validation of the capacitive sensor

To validate the methods and the model for Philips Healthcare's capacitive sensor we conducted several experiments, each to evaluate a specific property of the model. In Section 6.1 we discuss the accuracy of numerically solving the integral over the surface of a triangle to compute its capacitance. In Section 6.2 we see the accuracy of a fine-tuned model. The influence of the size of an object is discussed in Section 6.3. Lastly, in Section 6.4 we validate the accuracy of the model applied to non-parallel plates.

Note that in the graphs in this section we do not plot the actual output of the sensor, we plot how much the output differs from the output of the sensor if no objects are nearby. I.e., if the output of the sensor, with no objects nearby, is $2500V$, and with an object nearby $2000V$, we plot $500V$. This difference is denoted as *delta output* in the graphs.

6.1 Experiment I: Accuracy of discretizing the integral over the surface of a triangle

To validate the accuracy of the discretization of the integrals in Sections 5.3.2 and 5.3.3, we conducted an experiment for which the capacitance can be approximated both analytically and numerically. This is only possible if a triangle is in front of the sensor (in Region I in Figure 5.6). Therefore, in SimulationTool, we placed a plate in an arbitrary orientation in front of a sensor plate. The setup is shown in Figure 6.1.

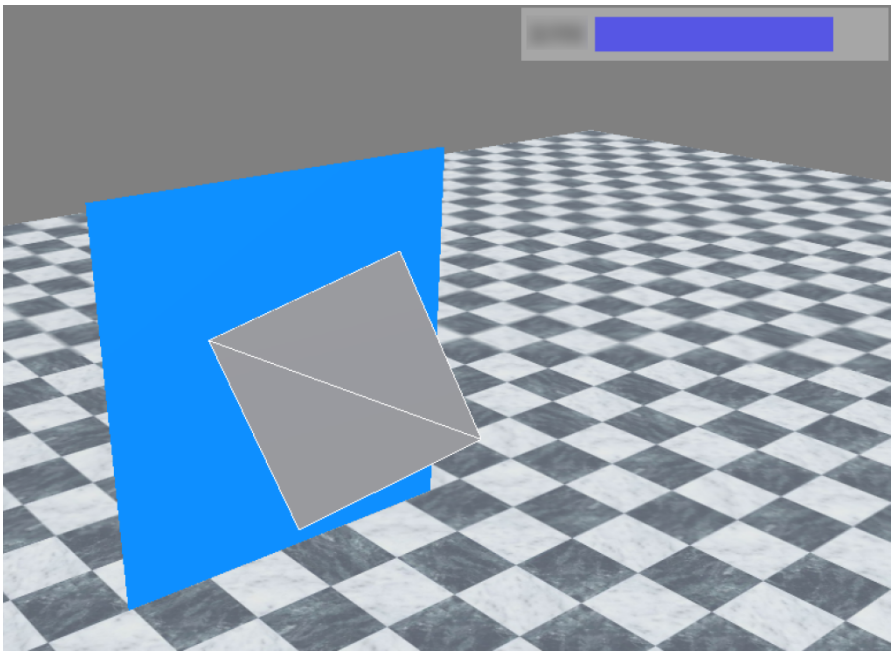

FIGURE 6.1: Setup in SimulationTool. The two gray oriented triangles are in total in front of the blue sensor plate.

Analytically we computed the real capacitance of the sensor. Thereafter, we iteratively split up the triangles forming the plate. At each iteration we approximated the capacitance of the sensor numerically. The results are in Figure 6.2.

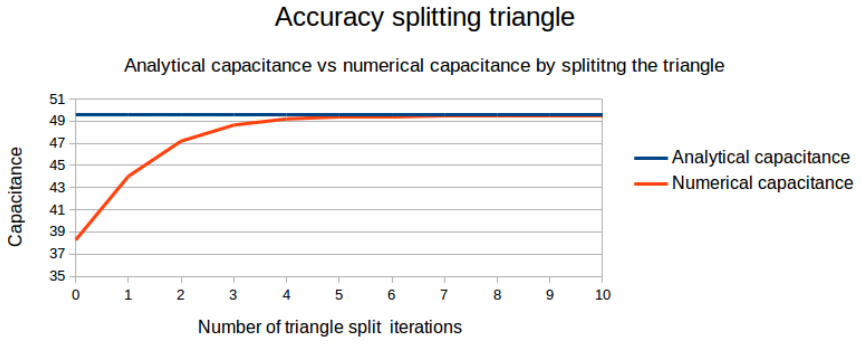

FIGURE 6.2: Accuracy of discretizing the integral over the area of a triangle by means of iteratively splitting the triangle

We see that in the first few iterations the approximation is quite below the real capacitance. But after splitting up the triangle only four or more times, the difference is negligible. For a precise and accurate simulation we recommend to split triangles at

least four times. For better computational performance we recommend a lower number of splits.

6.2 Experiment II: fine tuning the model for one plate

In this experiment we examine how accurately we can fine-tune the model given a reference curve. This reference curve is the curve of an actual measurement on the physical bodyguard. To create the curve we use the measurement tool shown in Figure 6.3.

FIGURE 6.3: The tool used to measure both distance to the sensor and the output of the sensor simultaneously

With this tool we can simultaneously measure the output of the sensor and the distance between the plate and the sensor. Firstly we place the plate (attached to the tool) tightly against the sensor, then, while measuring, we slowly move the measuring tool backward. The process is illustrated in Figure 6.4. We started with the plate at distance $d = xcm$ and moved it backward until the plate is hardly observable by the sensor around $d = xcm$.

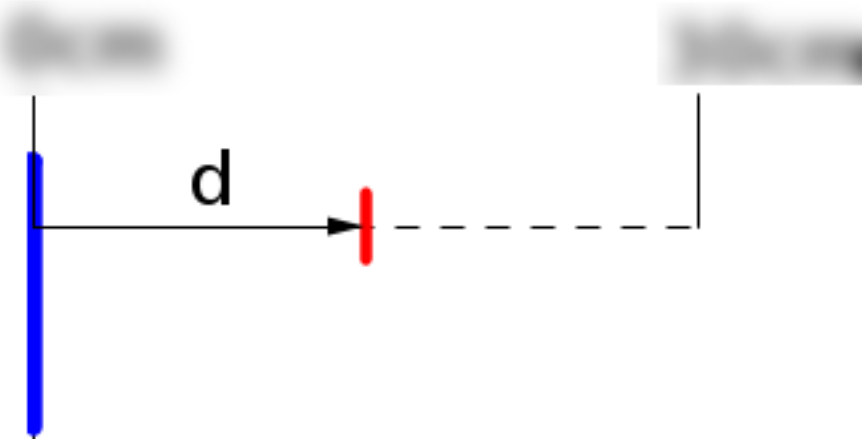

FIGURE 6.4: Illustrating experiment II, a plate is moved from $x$cm to $x$cm away from the sensor.

This procedure results in curves like the curve in Figure 6.5.

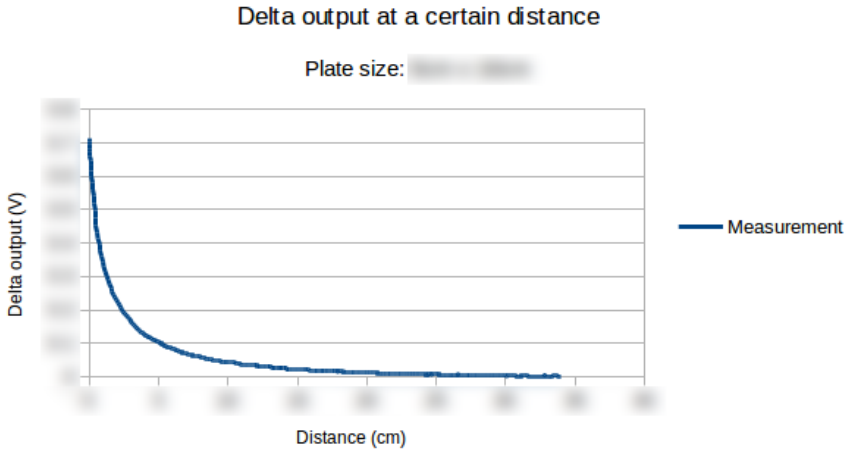

FIGURE 6.5: The delta output (not actual output) as a function of increasing distance. A curve like this is used as a reference curve in following experiments.

For this experiment, we examine if we can fine-tune the model such that the result of the same procedure performed in SimulationTool closely fits the reference curve.

The result of Experiment II with a plate of size $x$cm by $x$cm is shown in Figure 6.6.

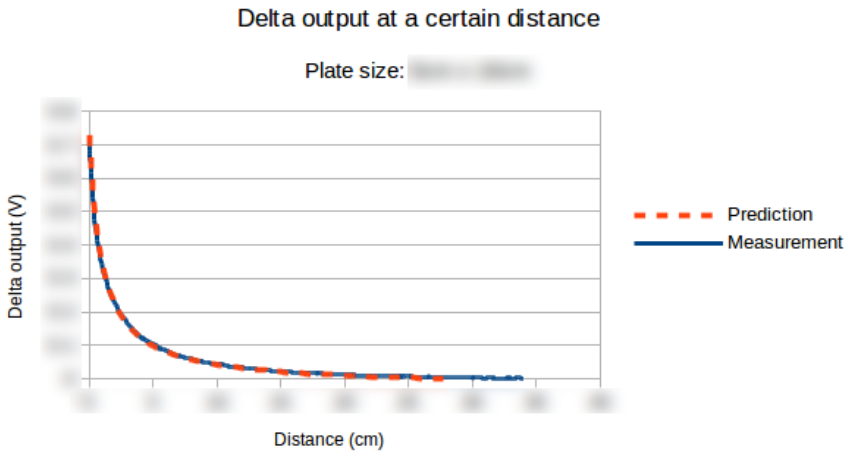

FIGURE 6.6: The fine-tuned model predicts the capacitive sensor accurately

The blue line, the reference curve, represents the actual measured data from the physical sensor. The outputs from SimulationTool, the orange striped line, precisely follows the reference curve. These accurate predictions were also observable with the other two plates. For this reason we only included the result of the plate of size $x$cm by $x$cm.

For configurations with a parallel plate in front of the sensor we can say that if the model is properly fine-tuned, the model can predict the output of the physical sensor very accurately.

6.3 Experiment III: varying plate sizes

During the conduction of the previously-described experiment the goal was to fine-tune the model given a reference curve. The goal of this experiment is to examine the influence of the size of the plate. Therefore we calibrate the model using the reference curve of a plate of $x$cm by $x$cm and see how the model performs if we make changes in the setup. In SimulationTool we perform the procedure described in the previous experiment with different plate sizes and compare the results with the corresponding reference curves. For the experiment we have 3 available plate sizes, $x$cm by $x$cm, $x$cm by $x$cm and $x$cm by $x$cm, the plates are shown in Figure 6.7.

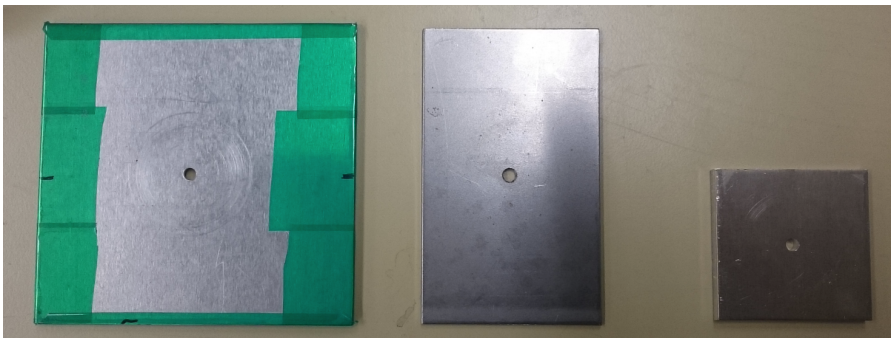

FIGURE 6.7: Different plate sizes used during experiment III

An example of this experiment is: calibrate the model for a plate of $x$cm by $x$cm, perform the previously described procedure in SimulationTool with a plate of $x$cm by $x$cm and compare the result with the reference curve of a plate of $x$cm by $x$cm.

Figure 6.8 shows the result of a plate size of $x$cm by $x$cm. The blue line is the actual measurement of a plate of $x$cm by $x$cm: the reference curve obtained with the previously explained procedure. The orange line is the prediction of SimulationTool. SimulationTool is slightly below the actual measurement. The greatest difference is at $x$cm, the difference is less than 24%, after $x$cm it is less than 5%.

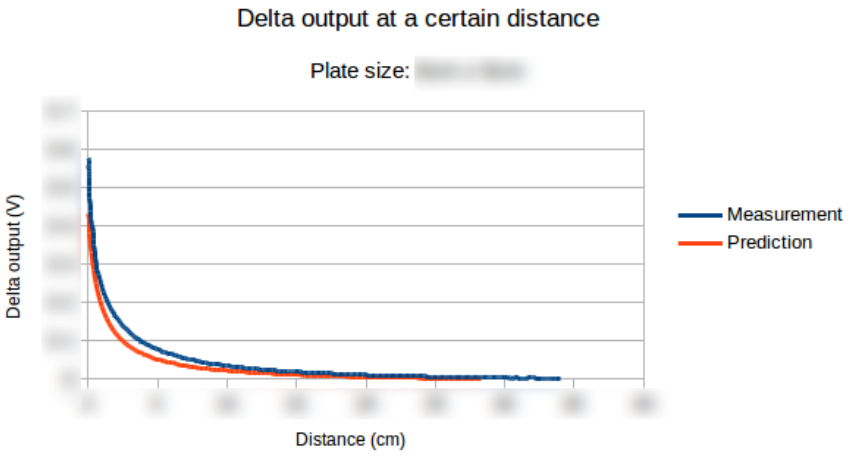

FIGURE 6.8: Measurements and predictions for a parallel plate of $x$cm by $x$cm with the model fine-tuned for a plate of $x$cm by $x$cm

Figure 6.9 shows the result of a plate of size $x$cm by $x$cm. Again, the blue line is the reference curve and the orange line the prediction of SimulationTool. The prediction is

slightly above the reference curve. With 10%, the greatest difference is at $x$cm, after $x$cm the difference is less than 6%.

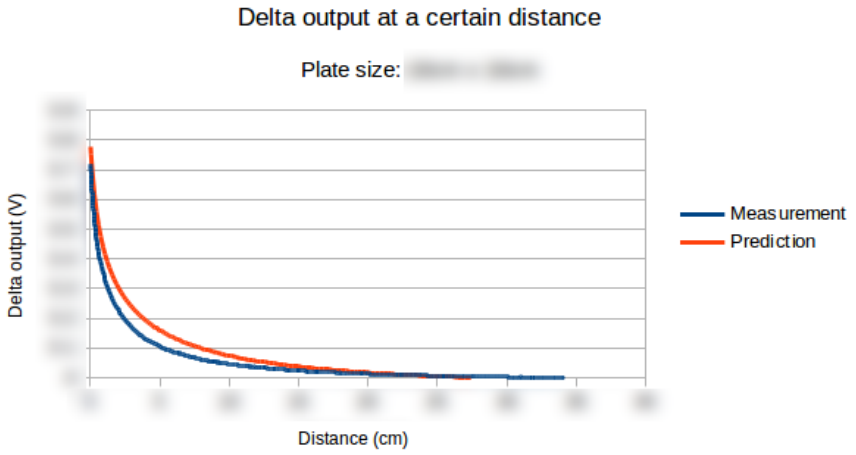

FIGURE 6.9: Measurements and predictions for a parallel plate of $x$cm by $x$cm with the model fine-tuned for a plate of $x$cm by $x$cm

The predictions are again accurate, but not as accurate as in Section 6.2. It is notable that the prediction with a bigger plate is too high and the prediction with a smaller plate is too low. The capacitance seems to be non-linearly dependent on the plate size. In perfect circumstances, the capacitance should grow linearly with the size of a plate, which is how the model in SimulationTool is implemented. The differences may be due to side effects like the fringe effect [Xia06]; another cause could be a relation between the total size of the objects and the total size of the sensor. Future work is needed to identify the cause and/or relation of this dependency.

6.4 Experiment IV: varying angle

For the last experiment we do not vary the distance or the size of the plates, we pivot a plate around on of its sides at a fixed distance d from the sensor. The experiment is illustrated in Figure 6.10.

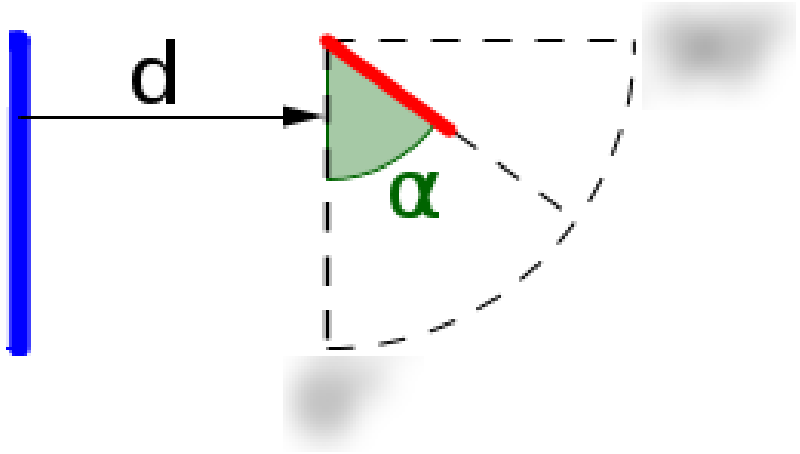

FIGURE 6.10: Illustrating experiment IV, a plate is pivoted around one edge to measure the influence of the angle between the plate and the sensor

The goal of the experiment is to validate that the effect of an inclined object in SimulationTool is almost equal to the effect of an inclined object in front of the physical sensor. To conduct the experiment we extended the measurement tool in Figure 6.3 with a tool created with LEGO (which has a high dielectric constant) shown in Figure 6.11.

FIGURE 6.11: The tool used to vary the angle between the plate and the sensor

With the measurement tool we can fix the distance, with the second tool we can pivot a plate around its side. The result is comparable to the reference curves, but with the angle on the $x - axis$ instead of the distance. An example of such a curve is shown in Figure 6.12.

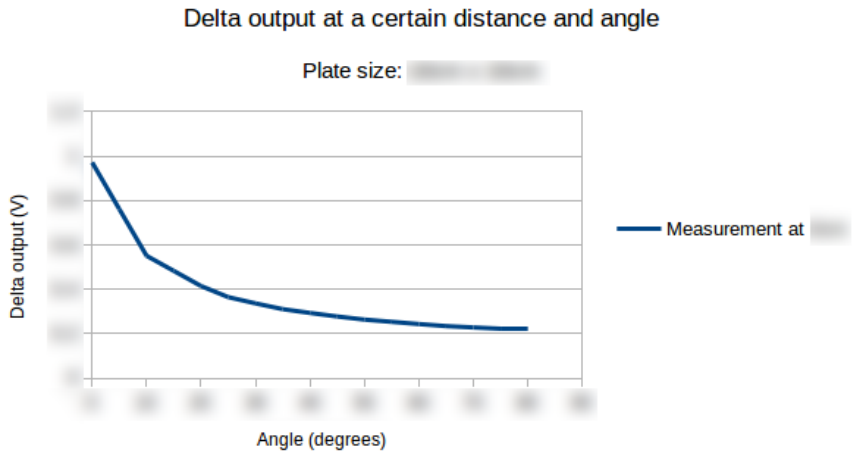

FIGURE 6.12: Example of a result of experiment IV

For the experiment we calibrate the model for the correct plate size and perform the exact same pivoting procedure in SimulationTool, we can compare the results with actual measurements like the measurements in Figure 6.12. We conducted the experiment for a plate of size $x$cm by $x$cm and $x$cm by $x$cm at distances of $x$cm, $x$cm, $x$cm and $x$cm from the sensor.

In Figure 6.13 we see the result of the plate of size $x$cm by $x$cm.

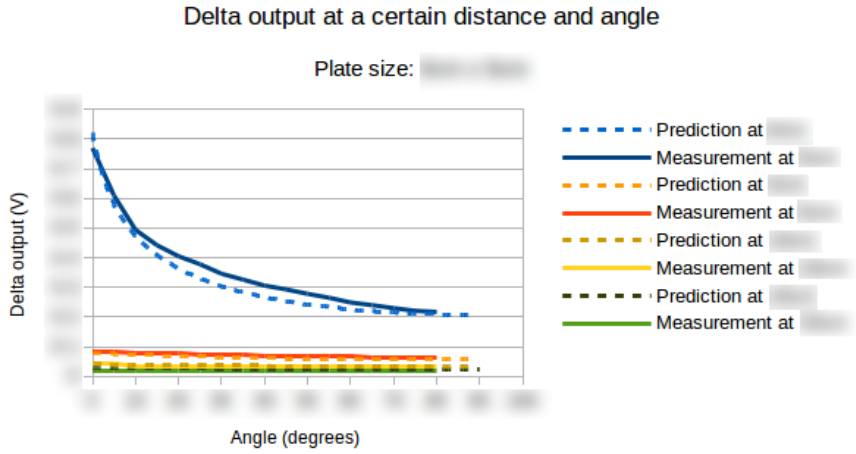

FIGURE 6.13: Measurements and predictions of a non-parallel plate of $x$cm by $x$cm with the model fine-tuned for a parallel plate

We see that the accuracy is rather good. Note that the measurements are made per x degrees, so we have to interpolate between these points. The highest variation is at 0cm, where the sensor is very sensitive.

In Figure 6.14 we see result of the plate of size $x$cm by $x$cm.

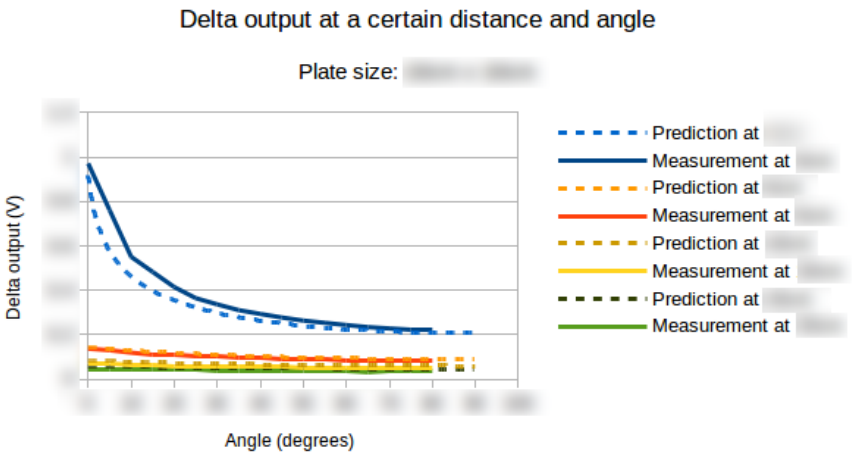

FIGURE 6.14: Measurements and predictions of a non-parallel plate of $x$cm by $x$cm with the model fine-tuned for a parallel plate

The results for a plate of x cm by x are almost identical to the results of a plate of $x$cm by $x$cm. Again the largest deviation is at a distance of $x$cm.

Based on these results we can say that the model captures the behavior of inclined objects in front of the sensor quite well. The largest deviations are due to the different objects sizes, and probably not their orientations.

7 | Conclusion and future work

In this thesis we covered two topics: collision detection and simulating proximity sensors in 3D simulations of mechanical systems. The discussed and proposed methods and algorithms are implemented in SimulationTool, a tool implemented by Philips Healthcare to simulate interventional X-Ray machines.

For collision detection we described the GJK-algorithm. The GJK-algorithm is an efficient algorithm to test for intersections between objects. To increase the performance of collision detection we split the process in a broad phase and narrow phase. In the broad phase we check for an intersection between the Bounding Volumes (BVs), the volumes that entirely contain the object. We only test for an intersection between the two objects if the BVs collide.

To simulate proximity sensors in 3D simulations we proposed methods to simulate two types of proximity sensors: a distance sensor and a capacitive sensor. The distance sensor is comparable to a parking sensor in a car. For multiple reasons, the distance sensor is not a realistic option for Philips Healthcare to equip on their interventional X-Ray machines. Instead they use capacitive sensors. These sensors generate an electric field surrounding the sensor. Any presence or displacement of nearby conductive objects changes the electric field, which results in a change in the output voltage of the sensor. Based on the output voltages, the interventional X-Ray machines may be restricted in their movement in certain directions. In this thesis we described and validated a model to simulate these sensors. The model approximates the capacitances of objects close to the sensor. Based on the capacitances of these objects the model predicts the output of the sensor.

The model is accurate, but there is still room for improvement. The model seems to miss a connection between the size of the objects and the size of the sensor. In future work this connection should be investigated.

As a next step, the effect of different materials and levels of conductivity should be researched. The current model is tested and validated for highly conductive and grounded objects.

Lastly, the effect of the potential difference between two capacitive sensors should be studied. An object that is close to a capacitive sensor changes the charge on that sensor. This change in charge results in a potential difference between that and adjacent sensors. The output voltages of these sensors drop due to the potential difference.

A | Details of the scene graph in SimulationTool

In SimulationTool objects are built up by primitives, called shapes. The shapes are attached to at most one link in a linkage. A shape has its own position and rotation relative to its parent link. The following primitives are available in SimulationTool: Cube, Cuboid, Arc, Plane, Cone and a Cylinder.

Shapes composing an object are all attached to a link. To translate or rotate such an object, we now do not need to translate or rotate each shape of the object: only the link has to be translated or rotated.

Links reside in a tree of links. Each link tree has one root, and each link may have several child links. A tree is used to construct a representation of connected objects, such as the X-Ray scanner, which is a combination of multiple moving and/or rotating arms and the scanner itself. Thus, a single link tree represents a part of the scene, not the complete scene.

A linkage is a set of link trees. This set of link trees represents the whole scene that is being simulated. The linkage represents all the connected components in the scene. In general, this is a tree for the scanner connected to the arms and a tree for the patient and the table. But more trees could be present in a linkage, such as a tree for a doctor or equipment in the operation room.

B | Derivations for Oriented Bounding Boxes

B.1 Distribution of vertices

The derivation for the covariance matrix regarding the distribution of the vertices of an object is as follows.

Suppose that the object consists of n points $\mathbf{p^1}, \mathbf{p^2}, ..., \mathbf{p^n}$ in space. Then C_{ij} (element (i, j) of C) is defined as follows:

$$\begin{aligned}
\mathrm{C}_{ij} = \mathrm{Cov}(\mathbf{x}_i, \mathbf{x}_j) &= \mathrm{E}[(\mathbf{x}_i - \mathrm{E}[\mathbf{x}_i])(\mathbf{x}_j - \mathrm{E}[\mathbf{x}_j])] \\
&= \mathrm{E}[\mathbf{x}_i\mathbf{x}_j - \mathbf{x}_i\mathrm{E}[\mathbf{x}_j] - \mathrm{E}[\mathbf{x}_i]\mathbf{x}_j + \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j]] \\
&= \mathrm{E}[\mathbf{x}_i\mathbf{x}_j] - \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j] - \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j] + \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j] \\
&= \mathrm{E}[\mathbf{x}_i\mathbf{x}_j] - \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j] \\
&= \mathrm{E}[\mathbf{x}_i\mathbf{x}_j] \; - \; \mathbf{m}_i\mathbf{m}_j \\
&= \frac{1}{n}\sum_{k=1}^{n} \mathbf{p}_i^k\mathbf{p}_j^k \; - \; \frac{1}{n}\sum_{k=1}^{n} \mathbf{p}_i^k \; \frac{1}{n}\sum_{k=1}^{n} \mathbf{p}_j^k
\end{aligned}$$

where $\mathbf{x}$ is a multivariate random variable (random vector), $\mathrm{E}[\mathbf{x}_i]$ the expectation of the ith coordinate of $\mathbf{x}$ and $\mathbf{m}$ the mean of all points.

B.2 Distribution of triangles

A problem with the covariance matrix based on only the vertices is that a skew vertex set will influence the result of the covariance matrix. A region wise small part of an object that consists of half of the vertices contribute to half of the covariance matrix. A solution would be to somehow weight the vertices, such that a cluttered part of an object does not bias C. This weighting may be done with the areas of the triangles: smaller triangles contribute less to the covariance matrix.

Before buckling down into the formula to evaluate the covariance matrix, a clear definition for notation is needed:

- Superscript k refers to the kth triangle.
- $\mathbf{a}_i$ denotes the ith coordinate of $\mathbf{a}$.
- Superscript M refers to all n triangles, also called the model (Model).
- P^k refers to the domain of integration of triangle k.
- $\mathbf{m}^k$ is the mean point of the kth triangle.
- The three vertices of triangle k are denoted as $\mathbf{p}^k$, $\mathbf{q}^k$ and $\mathbf{r}^k$.
  The kth triangle is parameterized by:

$$x^k(s,t) = \mathbf{p}^k + s\mathbf{u}^k + t\mathbf{v}^k, \text{ with } s \in [0,1] \text{ and } t \in [0, 1-s]$$

  where $\mathbf{u}^k = \mathbf{q}^k - \mathbf{p}^k$ and $\mathbf{v}^k = \mathbf{r}^k - \mathbf{p}^k$.
- The surface integral of a function $f(s,t)$ over the surface of a triangle k is defined as

$$\int_{P^k} f \mathrm{d}A = ||\mathbf{u} \times \mathbf{v}|| \int_0^1 \int_0^{1-s} f(s,t) \mathrm{d}t \mathrm{d}s$$

The definition for the covariance matrix is the same as before:

$$\mathrm{C}_{ij} = \mathrm{E}[\mathbf{x}_i \mathbf{x}_j] - \mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j]$$

where

$$\mathrm{E}[\mathbf{x}_i] = \frac{\int_M \mathbf{x}_i \mathrm{d}A}{\int_M \mathrm{d}A}$$

$$\mathrm{E}[\mathbf{x}_j] = \frac{\int_M \mathbf{x}_j \mathrm{d}A}{\int_M \mathrm{d}A}$$

$$\mathrm{E}[\mathbf{x}_i \mathbf{x}_j] = \frac{\int_M \mathbf{x}_i \mathbf{x}_j \mathrm{d}A}{\int_M \mathrm{d}A}$$

The integral over the domain M can be considered as the sum of integrals over the surfaces of the n triangles, which simplifies the equation above:

$$\begin{aligned}
\int_M \mathrm{d}A &= \sum_{k=1}^{n} \int_{P^k} \mathrm{d}A \\
&= \sum_{k=1}^{n} \int_0^1 \int_0^{1-s} ||\mathbf{u}^k \times \mathbf{v}^k|| \mathrm{d}t\mathrm{d}s \\
&= \sum_{k=1}^{n} ||\mathbf{u}^k \times \mathbf{v}^k|| \int_0^1 \int_0^{1-s} \mathrm{d}t\mathrm{d}s \\
&= \sum_{k=1}^{n} ||\mathbf{u}^k \times \mathbf{v}^k|| \int_0^1 (1-s)ds \\
&= \sum_{k=1}^{n} ||\mathbf{u}^k \times \mathbf{v}^k|| \frac{1}{2} \\
&= \sum_{k=1}^{n} A^k \\
&= A^M
\end{aligned}$$

Substituting the integral in the previous equations gives:

$$\mathrm{E}[\mathbf{x}_i] = \frac{1}{A^M} \int_M \mathbf{x}_i \mathrm{d}A$$

$$\mathrm{E}[\mathbf{x}_j] = \frac{1}{A^M} \int_M \mathbf{x}_j \mathrm{d}A$$

$$\mathrm{E}[\mathbf{x}_i \mathbf{x}_j] = \frac{1}{A^M} \int_M \mathbf{x}_i \mathbf{x}_j \mathrm{d}A$$

We can solve the first equation as follows:

$$
\begin{aligned}
\mathrm{E}[\mathbf{x}_i] &= \frac{1}{A^M}\int_M \mathbf{x}_i \mathrm{d}A \\
&= \frac{1}{A^M}\sum_{k=1}^{n}\int_{P^k} x_i^k \mathrm{d}A \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\int_0^1\int_0^{1-s} x_i^k \mathrm{d}t\mathrm{d}s \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\int_0^1\int_0^{1-s} \mathbf{p}_i^k + s\mathbf{u}_i^k + t\mathbf{v}_i^k \mathrm{d}t\mathrm{d}s \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\int_0^1 (1-s)\mathbf{p}_i^k + (1-s)s\mathbf{u}_i^k + \frac{1}{2}(1-s)^2\mathbf{v}_i^k ds \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\left(\frac{1}{2}\mathbf{p}_i^k + \frac{1}{6}\mathbf{u}_i^k + \frac{1}{6}\mathbf{v}_i^k\right) \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\left(\frac{1}{2}\mathbf{p}_i^k + \frac{1}{6}(\mathbf{q}_i^k - \mathbf{p}_i^k) + \frac{1}{6}(\mathbf{r}_i^k - \mathbf{p}_i^k)\right) \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\frac{1}{6}\left(\mathbf{p}_i^k + \mathbf{q}_i^k + \mathbf{r}_i^k\right) \\
&= \frac{1}{A^M}\sum_{k=1}^{n}\frac{1}{2}||\mathbf{u}^k\times\mathbf{v}^k||\frac{1}{3}\left(\mathbf{p}_i^k + \mathbf{q}_i^k + \mathbf{r}_i^k\right) \\
&= \frac{1}{A^M}\sum_{k=1}^{n} A^k\mathbf{m}_i^k
\end{aligned}
$$

$\mathrm{E}[\mathbf{x}_j]$ can be solved analogously to solving $\mathrm{E}[\mathbf{x}_i]$. Solving $\mathrm{E}[\mathbf{x}_i\mathbf{x}_j]$ is a bit more complex.

$$
\begin{aligned}
\mathrm{E}[\mathbf{x}_i\mathbf{x}_j] &= \frac{1}{A^M}\int_M \mathbf{x}_i\mathbf{x}_j \mathrm{d}A \\
&= \frac{1}{A^M}\sum_{k=1}^{n}\int_{P^k} \mathbf{x}_i^k\mathbf{x}_j^k \mathrm{d}A \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\int_0^1\int_0^{1-s} \mathbf{x}_i^k\mathbf{x}_j^k \mathrm{d}t\mathrm{d}s \\
&= \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\int_0^1\int_0^{1-s} f(s,t) \mathrm{d}t\mathrm{d}s
\end{aligned}
$$

where

$$\begin{aligned} f &= \mathbf{x}_i^k \mathbf{x}_j^k \\ &= (\mathbf{p}_i^k + s\mathbf{u}_i^k + t\mathbf{v}_i^k)(\mathbf{p}_j^k + s\mathbf{u}_j^k + t\mathbf{v}_j^k) \\ &= \mathbf{p}_i^k\mathbf{p}_j^k + s\mathbf{p}_i^k\mathbf{u}_j^k + t\mathbf{p}_i^k\mathbf{v}_j^k + s\mathbf{u}_i^k\mathbf{p}_j^k + s^2\mathbf{u}_i^k\mathbf{u}_j^k + st\mathbf{u}_i^k\mathbf{v}_j^k + t\mathbf{v}_i^k\mathbf{p}_j^k + st\mathbf{v}_i^k\mathbf{u}_j^k + t^2\mathbf{v}_i^k\mathbf{v}_j^k \\ &= \mathbf{p}_i^k\mathbf{p}_j^k + s(\mathbf{p}_i^k\mathbf{u}_j^k + \mathbf{u}_i^k\mathbf{p}_j^k) + t(\mathbf{p}_i^k\mathbf{v}_j^k + \mathbf{v}_i^k\mathbf{p}_j^k) + st(\mathbf{u}_i^k\mathbf{v}_j^k + \mathbf{v}_i^k\mathbf{u}_j^k) + s^2\mathbf{u}_i^k\mathbf{u}_j^k + t^2\mathbf{v}_i^k\mathbf{v}_j^k \end{aligned}$$

Solving the inner integral with respect to t gives

$$\begin{aligned} \int_0^{1-s} f(s,t)dt =& (1-s)\mathbf{p}_i^k\mathbf{p}_j^k + (1-s)s(\mathbf{p}_i^k\mathbf{u}_j^k + \mathbf{u}_i^k\mathbf{p}_j^k) + \frac{1}{2}(1-s)^2(\mathbf{p}_i^k\mathbf{v}_j^k + \mathbf{v}_i^k\mathbf{p}_j^k) + \\ & \frac{s}{2}(1-s)^2(\mathbf{u}_i^k\mathbf{v}_j^k + \mathbf{v}_i^k\mathbf{u}_j^k) + s^2(1-s)\mathbf{u}_i^k\mathbf{u}_j^k + \frac{1}{3}(1-s)^3\mathbf{v}_i^k\mathbf{v}_j^k \end{aligned}$$

With this result solving the outer integral with respect to s gives

$$\begin{aligned} \int_0^1 \int_0^{1-s} f(s,t)\mathrm{d}t\mathrm{d}s =& \frac{1}{24}\Big(12\mathbf{p}_i^k\mathbf{p}_j^k + 4(\mathbf{p}_i^k\mathbf{u}_j^k + \mathbf{u}_i^k\mathbf{p}_j^k) + 4(\mathbf{p}_i^k\mathbf{v}_j^k + \mathbf{v}_i^k\mathbf{p}_j^k) + \\ & 1(\mathbf{u}_i^k\mathbf{v}_j^k + \mathbf{v}_i^k\mathbf{u}_j^k) + 2\mathbf{u}_i^k\mathbf{u}_j^k + 2\mathbf{v}_i^k\mathbf{v}_j^k\Big) \end{aligned}$$

In this equation we can substitute $\mathbf{u}^k = \mathbf{q}^k - \mathbf{p}^k$ and $\mathbf{v}^k = \mathbf{r}^k - \mathbf{p}^k$:

$$\begin{aligned} \int_0^1 \int_0^{1-s} f(s,t)\mathrm{d}t\mathrm{d}s &= \frac{1}{24}\Big(2\mathbf{p}_i^k\mathbf{p}_j^k + \mathbf{p}_i^k\mathbf{q}_j^k + \mathbf{p}_i^k\mathbf{r}_j^k + \\ &\qquad 2\mathbf{q}_i^k\mathbf{q}_j^k + \mathbf{q}_i^k\mathbf{p}_j^k + \mathbf{q}_i^k\mathbf{r}_j^k + \\ &\qquad 2\mathbf{r}_i^k\mathbf{r}_j^k + \mathbf{r}_i^k\mathbf{p}_j^k + \mathbf{r}_i^k\mathbf{q}_j^k\Big) \\ &= \frac{1}{24}\Big((\mathbf{p}_i^k + \mathbf{q}_i^k + \mathbf{r}_i^k)(\mathbf{p}_j^k + \mathbf{q}_j^k + \mathbf{r}_j^k) + \\ &\qquad \mathbf{p}_i^k\mathbf{p}_j^k + \mathbf{q}_i^k\mathbf{q}_j^k + \mathbf{r}_i^k\mathbf{r}_j^k\Big) \\ &= \frac{1}{24}\Big(9\mathbf{m}_i^k\mathbf{m}_j^k + \mathbf{p}_i^k\mathbf{p}_j^k + \mathbf{q}_i^k\mathbf{q}_j^k + \mathbf{r}_i^k\mathbf{r}_j^k\Big) \end{aligned}$$

With this equation we can solve $\mathrm{E}[\mathbf{x}_i\mathbf{x}_j]$:

$$\begin{aligned}
\mathrm{E}[\mathbf{x}_i\mathbf{x}_j] =& \frac{1}{A^M}\sum_{k=1}^{n}\int_{P^k}\mathbf{x}_i^k\mathbf{x}_j^k\mathrm{d}A \\
=& \frac{1}{A^M}\sum_{k=1}^{n}||\mathbf{u}^k\times\mathbf{v}^k||\int_0^1\int_0^{1-s}f(s,t)\mathrm{d}t\mathrm{d}s \\
=& \frac{1}{A^M}\sum_{k=1}^{n}2A^k\int_0^1\int_0^{1-s}f(s,t)\mathrm{d}t\mathrm{d}s \\
=& \frac{1}{A^M}\sum_{k=1}^{n}\frac{2A^k}{24}\Big(9\mathbf{m}_i^k\mathbf{m}_j^k+\mathbf{p}_i^k\mathbf{p}_j^k+\mathbf{q}_i^k\mathbf{q}_j^k+\mathbf{r}_i^k\mathbf{r}_j^k\Big)
\end{aligned}$$

Combining the three equation results in the value of the element in C:

$$\begin{aligned}
\mathrm{C}_{ij}=\mathrm{E}[\mathbf{x}_i\mathbf{x}_j]-\mathrm{E}[\mathbf{x}_i]\mathrm{E}[\mathbf{x}_j] =& \frac{1}{A^M}\sum_{k=1}^{n}\frac{2A^k}{24}\Big(9\mathbf{m}_i^k\mathbf{m}_j^k+\mathbf{p}_i^k\mathbf{p}_j^k+\mathbf{q}_i^k\mathbf{q}_j^k+\mathbf{r}_i^k\mathbf{r}_j^k\Big) \\
&- \frac{1}{A^M}\sum_{k=1}^{n}A^k\mathbf{m}_i^k\ \ \frac{1}{A^M}\sum_{k=1}^{n}A^k\mathbf{m}_j^k
\end{aligned}$$

B.3 Prove of orthogonality of eigenvectors in a symmetric matrix

Since the covariance matrix is symmetric (by its definition), eigenvectors $\mathbf{v^1}$ and $\mathbf{v^2}$ corresponding to distinct eigenvalues λ_1 and λ_2 respectively are orthogonal to each other. This is proven as follows:

$$\begin{aligned}
\lambda_1(\mathbf{v^1}\cdot\mathbf{v^2}) &= (\lambda_1\mathbf{v^1})\cdot\mathbf{v^2} = (\mathrm{C}\mathbf{v^1})\cdot\mathbf{v^2} \\
&= (\mathrm{C}\mathbf{v^1})^T\mathbf{v^2} = (\mathbf{v^1}^T\mathrm{C}^T)\mathbf{v^2} \\
&= \mathbf{v^1}^T(\mathrm{C}^T\mathbf{v^2}) = \mathbf{v^1}\cdot(\mathrm{C}^T\mathbf{v^2}) \\
&= \mathbf{v^1}\cdot(\mathrm{C}\mathbf{v^2}) = \mathbf{v^1}\cdot(\lambda_2\mathbf{v^2}) \\
&= \lambda_2(\mathbf{v^1}\cdot\mathbf{v^2}) \\
\lambda_1(\mathbf{v^1}\cdot\mathbf{v^2}) = \lambda_2(\mathbf{v^1}\cdot\mathbf{v^2}) \wedge \lambda_1\neq\lambda_2 &\Rightarrow \mathbf{v^1}\cdot\mathbf{v^2}=0
\end{aligned}$$

$\mathbf{v^1}\cdot\mathbf{v^2}=0$ is the definition of orthogonality of $\mathbf{v^1}$ and $\mathbf{v^2}$.

C | Details of the implementation of the GJK-algorithm in SimulationTool

C.1 CheckAndUpdateSimplex

The structure of the GJK-algorithm uses only a few lines of code. However, the algorithm relies on the function *CheckAndUpdateSimplex*. This procedure that checks and updates the simplex and the direction vector is a bit more complicated, we discuss the different cases (simplex of 1, 2, 3 or 4 points) one by one. The naming in the figures differs a bit from the pseudo code. In each of the figures, **A** is the new support point, **O** the origin. **B**, **C** and **D** represent the points that already were in the simplex, where **B** was added most recently, and **D** is the longest in the simplex.

Point Updating a simplex consisting of 1 point is trivial. The new point (**A**) stays in the simplex, it's the very first point. The new support direction is from **A** towards the origin.

Update:

$\mathbf{B} \Leftarrow \mathbf{A}$

$\mathbf{v} \Leftarrow -\mathbf{A}$

We have not yet found a simplex that contains the origin, therefore we return *false*.

Line segment A simplex of two points is a line segment. Figure C.1 shows an example of such a line segment.

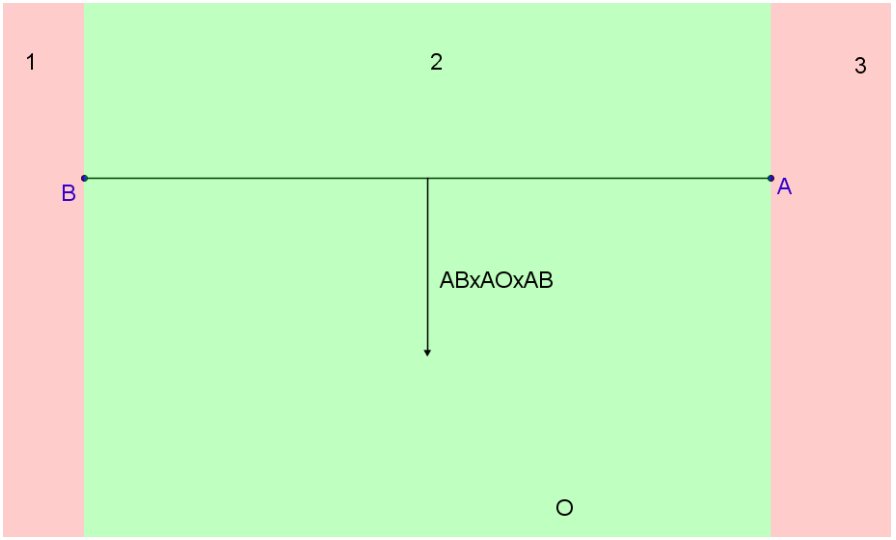

FIGURE C.1: Line segment **AB** as simplex.

If we have line segment **AB** as simplex, we know two things for sure: The origin cannot lie in region 1. Point **A** was added as a result of looking for the furthest point towards the direction of the origin from point **B**. So the origin cannot lie to the left of **AB**.
The origin cannot lie in region 3 either. From **B**, **A** was the furthest point in the direction of the origin. If the origin would be located in region 3 a separating axis would have been found (the axis through **A**) and the algorithm would terminate.
Only region 2 is left. Both **A** and **B** are part of region 2, so both of the points stay in the simplex. The new search direction has to be perpendicular to line segment **AB** and towards the origin, which can be calculated via a double cross product: $\mathbf{AB} \times \mathbf{AO} \times \mathbf{AB}$.
Update:
$\mathbf{C} \Leftarrow \mathbf{B}$
$\mathbf{B} \Leftarrow \mathbf{A}$
$\mathbf{v} \Leftarrow \mathbf{AB} \times \mathbf{AO} \times \mathbf{AB}$
We have not yet found a simplex that contains the origin, therefore we return *false*.

Triangle A simplex of three points is a triangle. Figure C.2 shows an example of such a triangle.

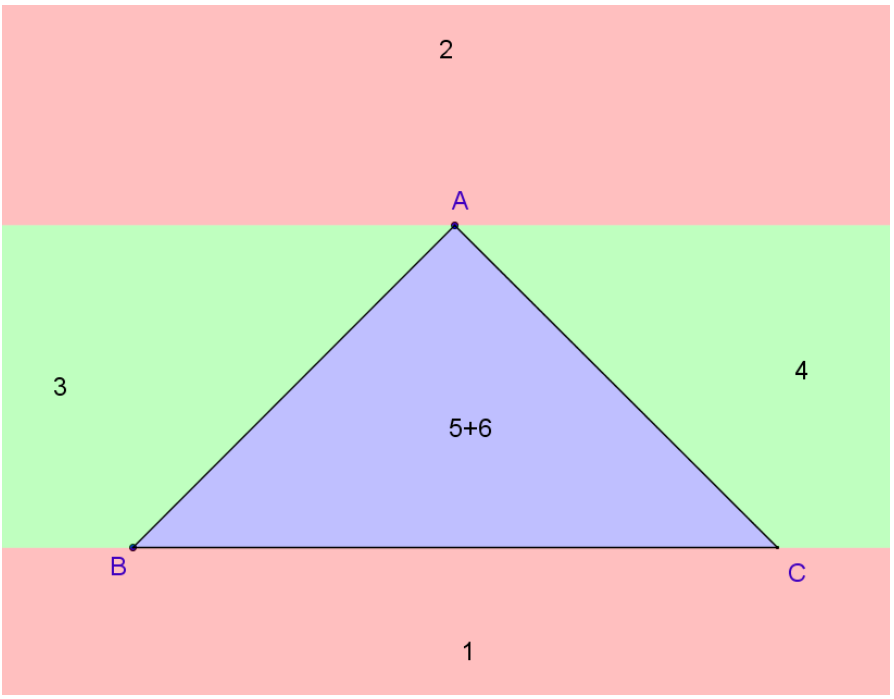

FIGURE C.2: Triangle **ABC** as simplex.

The triangle case is comparable to the line segment. We first distinguish a number of regions where to origin could be located, then for each region we discuss how to update the simplex and the direction vector. Just as with the line segment, we do not have to look at region 1 and 2, for the same reasons.

To distinguish region 3, 4, 5 and 6 we need the normal **n** of triangle **ABC**, the normal of a triangle is the cross product of 2 edges counterclockwise. So in this case normal $\mathbf{n} = \mathbf{AB} \times \mathbf{AC}$.

Region 3

If the origin is located in region 3, then the origin is to the left of line segment **AB**. The origin is to the left of **AB** if and only if $(\mathbf{AB} \times \mathbf{n}) \cdot \mathbf{AO} > 0$. If this is the case, we update the simplex comparable to a line segment, **A** and **B** make space for the new support and the direction vector is perpendicular to the edge and towards the origin.

Update:

$\mathbf{C} \Leftarrow \mathbf{B}$

$\mathbf{B} \Leftarrow \mathbf{A}$

$\mathbf{v} \Leftarrow \mathbf{AB} \times \mathbf{AO} \times \mathbf{AB}$

We have not yet found a simplex that contains the origin, therefore we return *false*.

Region 4

If the origin is located in region 4, then the origin is to the left of line segment **AC**. The origin is to the left of **AC** if and only if $(\mathbf{n} \times \mathbf{AC}) \cdot \mathbf{AO} > 0$. If this is the case, we update the simplex comparable to a line segment, **A** makes space for the new support (**C** stays the same) and the direction vector is perpendicular to the edge and towards the origin.

Update:

$\mathbf{B} \Leftarrow \mathbf{A}$

$\mathbf{v} \Leftarrow \mathbf{AC} \times \mathbf{AO} \times \mathbf{AC}$

We have not yet found a simplex that contains the origin, therefore we return *false*.

Region 5+6

If the origin is not located in region 3 or 4, it must be above or below triangle **ABC**. Whether the origin is above or below **ABC** can be tested via the normal. The origin is above the triangle if and only if $\mathbf{n} \cdot \mathbf{AO} > 0$. If not, then the origin is below the triangle. If the origin is above **ABC** we just shift **A**, **B** and **C**. The new search direction is the normal of **ABC**. Otherwise we shift the points, but also have to flip the order of the points. The new search direction is the normal again, but in the oposite direction.

Update:

if $\mathbf{n} \cdot \mathbf{AO} > 0$ **then**

  $\mathbf{D} \Leftarrow \mathbf{C}$

  $\mathbf{C} \Leftarrow \mathbf{B}$

  $\mathbf{B} \Leftarrow \mathbf{A}$

  $\mathbf{v} \Leftarrow \mathbf{n}$

else

  $\mathbf{D} \Leftarrow \mathbf{B}$

  $\mathbf{B} \Leftarrow \mathbf{A}$

  $\mathbf{v} \Leftarrow -\mathbf{n}$

We have not yet found a simplex that contains the origin, therefore we return *false*.

Tetrahedron The first time that we have a simplex with 4 points, a tetrahedron, is the time that we might have found an intersection. Recall that an intersection is found

when the origin is enveloped by a simplex, the tetrahedron is the only simplex (in $\mathbb{R}^3$) that can envelope the origin. Figure C.3 shows an example of a tetrahedron simplex.

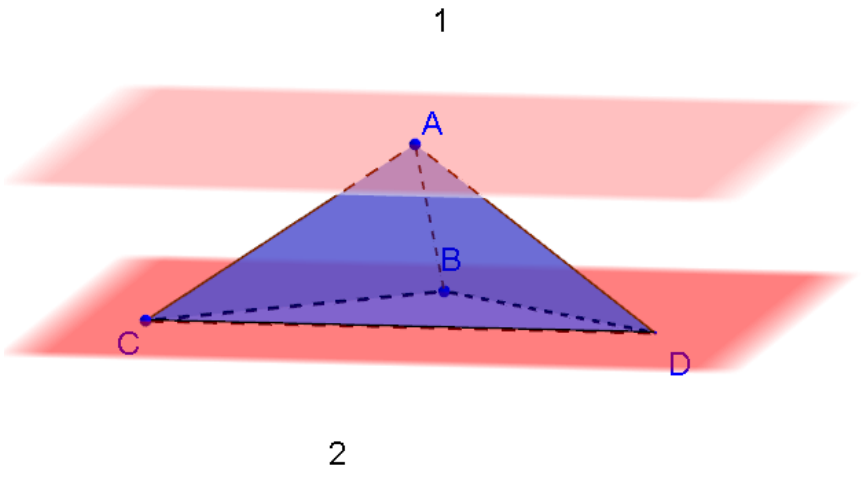

FIGURE C.3: Tetrahedron **ABCD** as simplex.

Just like the line segment and the triangle, the origin is not in region 2, we searched for the origin on the other side of the bottom red plane. The origin could be in region 1, but then the algorithm would have terminated because of the visible separating plane (separating axis in $\mathbb{R}^3$).

In Figure C.3 no further distinguishable regions are shown to avoid clutter. However, initially we define 4 more regions: One region if the origin is in front of a triangle, for each of the three triangles **ACD**, **ADB** and **ABC** and a last region if the origin is inside the tetrahedron, respectively identified as region 2, 3, 4 and 5.
Regions 2, 3 and 4 are handled analogously to each other, therefor we briefly discuss region 4 and 5.

Region 4

The origin is in front of triangle **ABC** if and only if $(\mathbf{AB} \times \mathbf{AC}) \cdot \mathbf{AO} < 0$. Then the origin can be in 3 regions, comparable to the triangle case. The origin can be to the left of **AB**, to the right of **AC** or right in front of **ABC**. These cases are already captured in Section C.1.

Region 5

For region 5 there is not much to discuss, we found the origin to be inside the tetrahedron.

I.e., in this case we found an intersection of the two objects and return *true*.

The origin is inside the tetrahedron if and only if

$$(\mathbf{AC} \times \mathbf{AD}) \cdot \mathbf{AO} \leq 0 \quad \wedge \quad (\mathbf{AD} \times \mathbf{AB}) \cdot \mathbf{AO} \leq 0 \quad \wedge \quad (\mathbf{AB} \times \mathbf{AC}) \cdot \mathbf{AO} \leq 0$$

D | Solving the equations for proximity sensors

In Section 5 we encountered an integral that approximates the capacitance of a triangle right in front of a sensor plate. The integral is quite complicated to solve. The details on how to solve this integral is listed here step by step. Halfway the integral is transformed using Barycentric coordinates, for more information of this process see Section 5.2.

$$
\begin{aligned}
C &= \int \mathrm{d}C \\
&= \int_T \frac{\varepsilon}{\delta d}\,\mathrm{d}A \\
&= \varepsilon \int_T \frac{1}{\delta d}\,\mathrm{d}A \\
&= \varepsilon\,||\mathbf{u}\times\mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\delta d}\,\mathrm{d}\lambda_0\mathrm{d}\lambda_1 \\
&= \varepsilon\,||\mathbf{u}\times\mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{(\mathbf{p_0}+\lambda_1\mathbf{u}+\lambda_0\mathbf{v}-\mathbf{q_0})\cdot\mathbf{n}}\,\mathrm{d}\lambda_0\mathrm{d}\lambda_1 \\
&= \varepsilon\,||\mathbf{u}\times\mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{\mathbf{p_0}\cdot\mathbf{n}+\lambda_1\,\mathbf{u}\cdot\mathbf{n}+\lambda_0\,\mathbf{v}\cdot\mathbf{n}-\mathbf{q_0}\cdot\mathbf{n}}\,\mathrm{d}\lambda_0\mathrm{d}\lambda_1
\end{aligned}
$$

Substituting $a = \mathbf{u}\cdot\mathbf{n}$, $b = \mathbf{v}\cdot\mathbf{n}$ and $c = \mathbf{p_0}\cdot\mathbf{n} - \mathbf{q_0}\cdot\mathbf{n}$ gives:

$$
\begin{aligned}
C &= \varepsilon\,||\mathbf{u}\times\mathbf{v}|| \int_0^1 \int_0^{1-\lambda_1} \frac{1}{a\lambda_1+b\lambda_0+c}\,\mathrm{d}\lambda_0\mathrm{d}\lambda_1 \\
&= \varepsilon\,||\mathbf{u}\times\mathbf{v}|| \int_0^1 \Big[\,\frac{1}{b}\ln(a\lambda_1+b\lambda_0+c)\,\Big]_0^{1-\lambda_1}\,\mathrm{d}\lambda_1 \\
&= \frac{\varepsilon\,||\mathbf{u}\times\mathbf{v}||}{b} \int_0^1 \ln(a\lambda_1+b(1-\lambda_1)+c) - \ln(a\lambda_1+c)\,\mathrm{d}\lambda_1 \\
&= \frac{\varepsilon\,||\mathbf{u}\times\mathbf{v}||}{b} \int_0^1 \ln((a-b)\lambda_1+b+c) - \ln(a\lambda_1+c)\,\mathrm{d}\lambda_1 \\
&= \frac{\varepsilon\,||\mathbf{u}\times\mathbf{v}||}{b} \Big[\,\frac{a\,((a-b)\lambda_1+b+c)\,ln((a-b)\lambda_1+b+c)\;-\;(a-b)\,(a\lambda_1+c)\,ln(a\lambda_1+c)}{a\,(a-b)}\,\Big]_0^1 \\
&= \frac{\varepsilon\,||\mathbf{u}\times\mathbf{v}||}{b} \Bigg(\Big(\,\frac{a\,(a+c)\,ln(a+c)\;-\;(a-b)\,(a+c)\,ln(a+c)}{a\,(a-b)}\,\Big) - \\
&\qquad \Big(\,\frac{a\,(b+c)\,ln(b+c)\;-\;(a-b)\,(c)\,ln(c)}{a\,(a-b)}\,\Big)\Bigg) \\
&= \frac{\varepsilon\,||\mathbf{u}\times\mathbf{v}||}{b} \Big(\,\frac{b(a+c)}{a(a-b)}\,ln(a+c)\;-\;\frac{b+c}{a-b}\,ln(b+c)\;+\;\frac{c}{a}\,ln(c)\,\Big)
\end{aligned}
$$

Undoing the substitution for a, b and c gives:

$$C = \frac{\varepsilon \, ||\mathbf{u} \times \mathbf{v}||}{\mathbf{v} \cdot \mathbf{n}} \left(\frac{\mathbf{v} \cdot \mathbf{n} \, (\mathbf{u} \cdot \mathbf{n} + (\mathbf{p_0} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n}))}{\mathbf{u} \cdot \mathbf{n} \, (\mathbf{u} \cdot \mathbf{n} - \mathbf{v} \cdot \mathbf{n})} \, ln(\mathbf{u} \cdot \mathbf{n} + (\mathbf{p_0} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n})) - \right.$$
$$\frac{\mathbf{v} \cdot \mathbf{n} + (\mathbf{p_0} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n})}{\mathbf{u} \cdot \mathbf{n} - \mathbf{v} \cdot \mathbf{n}} \, ln(\mathbf{v} \cdot \mathbf{n} + (\mathbf{p_0} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n})) +$$
$$\left. \frac{(\mathbf{p_0} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n})}{\mathbf{u} \cdot \mathbf{n}} \, ln((\mathbf{p_0} \cdot \mathbf{n} - \mathbf{q_0} \cdot \mathbf{n})) \right)$$

Bibliography

[ARM] RN Aguilar, M Roelofsz, and GCM Meijer. Capacitive human-detection systems with auto-calibration.

[BBCPCI11] JM Bueno-Barrachina, CS Canas-Penuelas, and S Catalan-Izquierdo. Capacitance evaluation on non-parallel thick-plate capacitors by means of finite element analysis. *Journal of Energy and Power Engineering*, 5(4):373–378, 2011.

[Ber99] Gino van den Bergen. A fast and robust gjk implementation for collision detection of convex objects. *Journal of Graphics Tools*, 4(2):7–25, 1999.

[cop] Coppelia robotics gmbh. http://www.coppeliarobotics.com/contact.html. Last visited on 13/05/2015.

[Cou01] Murilo G Coutinho. *Dynamic simulations of multibody systems*. Springer, 2001.

[Ebe08] David Eberly. Intersection of a triangle and a cone, 2008.

[GJK88] Elmer G Gilbert, Daniel W Johnson, and S Sathiya Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, 4(2):193–203, 1988.

[Got00] Stefan Gottschalk. *Collision queries using oriented bounding boxes*. PhD thesis, The University of North Carolina at Chapel Hill, 2000.

[Hel97] Martin Held. Erita collection of efficient and reliable intersection tests. *Journal of Graphics Tools*, 2(4):25–44, 1997.

[Hos15] Ranav Hosangadi. Capacitance of two non parallel plates. http://physics.stackexchange.com/questions/148283/capacitance-of-two-non-parallel-plates, 2015. Last visited on 13/05/2015.

[JLSC91] Bernard C Jiang, Andrew YH Lio, N Suresh, and Otto SH Cheng. An evaluation of machine guarding techniques for robot guarding. *Robotics and autonomous systems*, 7(4):299–308, 1991.

[KHI+07] Sinan Kockara, Tansel Halic, K Iqbal, Coskun Bayrak, and Richard Rowe. Collision detection: A survey. In *Systems, Man and Cybernetics, 2007. ISIC. IEEE International Conference on*, pages 4046–4051. IEEE, 2007.

[KJ93] Nils Karlsson and J-O Jarrhed. A capacitive sensor for the detection of humans in a robot cell. In *Instrumentation and Measurement Technology Conference, 1993. IMTC/93. Conference Record., IEEE*, pages 164–166. IEEE, 1993.

[MG09] Khaled Mamou and Faouzi Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3501–3504. IEEE, 2009.

[O'R85] Joseph O'Rourke. Finding minimal enclosing boxes. *International journal of computer & information sciences*, 14(3):183–199, 1985.

[vRe] V-rep proximity sensor. https://www.youtube.com/watch?v=3w3zzVIr3kI. Last visited on 13/05/2015.

[Weia] Eric W. Weisstein. Barycentric coordinates. From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/BarycentricCoordinates.html. Last visited on 09/2/2015.

[Weib] Eric W. Weisstein. Cone. From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/Cone.html. Last visited on 09/2/2015.

[Weic] Eric W. Weisstein. Conic section. From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/ConicSection.html. Last visited on 09/2/2015.

[Weid] Eric W. Weisstein. Point-line distance–3-dimensional. From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/Point-LineDistance3-Dimensional.html. Last visited on 09/2/2015.

[Weie] Eric W. Weisstein. Point-plane distance. From MathWorld—A Wolfram Web Resource. http://mathworld.wolfram.com/Point-PlaneDistance.html. Last visited on 09/2/2015.

[Xia06] Yumin Xiang. The electrostatic capacitance of an inclined plate capacitor. *Journal of electrostatics*, 64(1):29–34, 2006.

[Zac00] Gabriel Zachmann. *Virtual reality in assembly simulation-collision detection, simulation algorithms, and interaction techniques.* PhD thesis, Zachmann, Gabriel, 2000.