Grasp Synthesis for Human-Virtual Character Handovers

Master Thesis

Eva Linssen
3902749

Supervisors:
Dr. ir. A. Frank van der Stappen
Dr. Zerrin Yumak

Utrecht University
Department of Information and Computer Sciences

Daily Supervisors:
Ing. Hugo Habets
Ir. Njin-Zu Chen MTD.

Philips Research

March 1, 2018

Abstract

We create grasps for a human-to-virtual-character handover. This is a grasp synthesis problem with the added requirement of avoiding the human hand, while aiming for natural and comfortable looking output grasps. Grasp synthesis is mainly used in the robotics domain for real life applications. Here the forces applied by the robot hand must be calculated through a precise physics model in order to correctly predict their real-world consequences. As our situation is a virtual one, we can approximate the physics requirements. Our approach is based on random sampling of hand placements in proximity of the object and then closing the fingers around the object. We analyse the parameters used in our method and constrain their ranges in order to shorten the execution time. We have made a significant improvement with respect to the naive version of the algorithm (which does not restrict the ranges of the parameters): the final version of our algorithm finds a valid grasp in 3.6% of the time taken by our naive version.

Contents

Chapter 1

Introduction

In this thesis we study the virtual scenario in which the hand of a user presents an object to a virtual character: a human-virtual character handover. The scene plays out in VR where the user can walk around and interact with the environment with a set of virtual hands. The handover is one of the interactions between the human and the virtual character that can take place. During a human-virtual character handover, an object can be transferred either from the human to the virtual character or in the other direction. We focus on finding grasps for the first situation, where the human presents the object to the virtual character. This means that we have to take the human's hand into account when finding a grasp for the virtual character on the object.

Our setting is a surgery room. We need to find grasps on the items that are typically found in such a room. Such items are, for example, surgery instruments or controllers of the surrounding machinery. There are many different operations that can be done in an operation room and each of them makes use of different types of surgery instruments. We have observed that surgery instruments can be split into roughly three groups: the scissors, the knifes and the braces. The scissors are long and thin with two ears at one end where the surgeon controls them. The knifes are long and thin and sharp on one end. The braces have all kinds of shapes, and are used to keep the skin and organs in place during deep operations. We represent the scissors and knifes with objects like a pair of scissors and a scalpel. As the braces exist in a large variety of shapes we could not find a good shape to represent them. Other objects that could occasionally be handed over in a surgery room would be remote controls of the surrounding machinery. The mentioned objects have a few properties in common: they are small and light and made of non-deformable material. In the context of the surgery room it seems reasonable that parts of the object must remain sterile. The objects are used during operations where hygiene is an issue. Rules with respect to sterility are stricter for the parts of the object that come in direct contact with open wounds of the patient. Touching these parts of the object is forbidden in general, and thus also during the handover. Additionally, the need to keep objects sterile means that the object must not accidentally fall on the floor during handovers. So both participants in the handover should have a stable grasp on the object before the giver releases it.

1.1 Research Questions

We are seeking a method to generate a grasp for a handover between a user and a virtual character. Grasp synthesis for robot arms in the real world is a problem that has received considerable

attention. However, the methods used in this domain are not efficient enough to deal with our interactive scenario. This means we need an online grasp synthesis method. Note that if the grasp synthesis takes too long the user will perceive this as lag and get frustrated. The time frame in which the virtual character lifts its arm can be used to calculate the precise grasp, but this still gives us fairly little time. This means we need our method to be fast. We want the output grasps to be human-like and realistic, as our virtual character is humanoid and has human hands. It is hard to decide what exactly constitutes as a 'realistic' grasp as it is a fairly subjective term. So we must find a way to objectively validate the realism of our grasps.

Thus our research will thus encompass the following questions:

- How do we find grasping hand poses appropriate for the scenario of a human-virtual character handover?
- How do we model 'realism' in order to produce natural-looking grasps?
- Can we find the grasps fast enough to allow our grasp synthesis method to be used in an interactive environment?

1.2 Contribution

As far as we know there is hardly any work on human-virtual character handovers. Much of the research concerning handovers between humans and physically existing robots focusses on the behaviour of the robot rather than grasp synthesis. While we found a few papers about grasping for robot-to-human handovers, we did not find any that study the human-to-robot handovers.

Our grasp synthesis method is inspired by grasp synthesis methods that pick up loose (free floating) objects. Our hand model takes into account the anatomy of the hand and ergonomy, with the goal that our output grasps look natural to other humans. We add avoidance of the human hand and avoidance of touching certain places on the object as extra requirements for our output. Lastly we propose an evaluation measure based on comfort of the grasp.

To validate our that our grasps immobilize the object, we take notions from the robotic domain. In robotics it is important that the grasps immobilize the object. The grasps are validated to conform to force closure and evaluated on grasp quality so that output grasp are as robust as possible. This is not necessary in our virtual environment, and as such we contribute an immobility validation method that measures whether a grasp *approximately* immobilizes the object. It is both a faster calculation and it is easier to find appropriate force distributions.

The robots for which the robotic grasp synthesis methods are designed often have much simpler hands than humans do. A robot can have a four fingered gripper with one joint per finger, where humans have five fingers with three joints each. This makes finding grasping hand poses for the human hand much harder, simply because its pose has a higher number of degrees of freedom. We model the human and its closing motion to find realistic hand poses in the high dimensional search space. The output of robotic grasp synthesis methods are often precision grasps (touching the object only with the fingertips). As we find poses by closing the hand, our method is also able to output power grasps (which also include other parts of the fingers and hand palm in the grasp).

1.3 Thesis Structure

The rest of this thesis will be structured as follows. In Chapter 2 we discuss background information that is useful to understand the problem. We will talk about handovers in general, but also specifically handovers between humans and robots. We also discuss the anatomy of the hand, common grasp types and grasp immobility analysis. In Chapter 3 we discuss existing work about grasp synthesis. The existing work comes from the robotics domain, but also from the animation domain. In the animation domain they automate generation of smooth motion paths for a human character that manipulates an object. In Chapter 4 we explain our approach to the problem. First we introduce our general intuition for making natural grasps. Then we model the hand and its closing motion. We give pseudo-code for our grasp synthesis method. In Chapter 5 we test an implementation of our method with the goal of improving execution times. We do this by finding better values and constraints for the parameters involved. In Chapter 6 we conclude on what we accomplished and discuss future work.

Chapter 2

Background

Before we start on grasp synthesis methods, we discuss some topics that are useful to have knowledge of. Section 2.1 explains human behaviour during handovers. Section 2.2 goes into aspects that need consideration when modelling robot-behaviour during a human-robot handovers. This thesis only covers a small aspect of the robot-behaviour, namely the making of grasp poses. Sections 2.1 and 2.2 are discussed mainly for contextual information. They do not necessarily need to be read to understand the material covered in the remainder of this work. Sections we do recommend to read are Sections 2.3, 2.4, 2.5 and 2.6, as this information is used in our model. Section 2.3 explains the anatomy of the human hand and its allowed motion. Section 2.4 discusses types of grasps commonly used by humans and Section 2.5 discusses ergonomy with respect to grasping. Section 2.6 explains a physics model this is used in analysing whether or a grasp can immobilize an object.

2.1 Human Handover Behaviour

A handover takes place when one person hands an object over to another person. It is a cooperation task: the participants of the handover must establish together that a handover will take place, and how it will happen. During a handover there is constant communication between the two participants. The fact that humans can do handovers without much thought implies that there is an underlying procedure that humans follow. Cakmak et al. [5], [6] and Strabala et al. [32], [33] have done several studies on human-human handovers in order to find the procedure that humans instinctively use during a handover task. Strabala et al. split up the process in a physical and a social-cognitive channel as seen in Figure 2.1.

In the social-cognitive channel, the humans signal each other about the *what*, *when* and *where* of the handover. The 'what' decides which object to be handed over. This is established before the handover process, for example through speech. The 'when' and 'where' get signalled through body language during respectively the approach and reach stage. This body language is for example body orientation and eye contact.

The physical channel consists of an *approach*, *reach* and *transfer* stage. During the approach phase the bodily distance between the giver and receiver is decreased. One or both walks towards the other. To accommodate for the handover ready to take place, the giver's grasp on the item leaves space open for the receiver to grasp onto. When they are close enough, the reaching phase starts: they move their hands towards a centre point where the giver's hand will meet the held

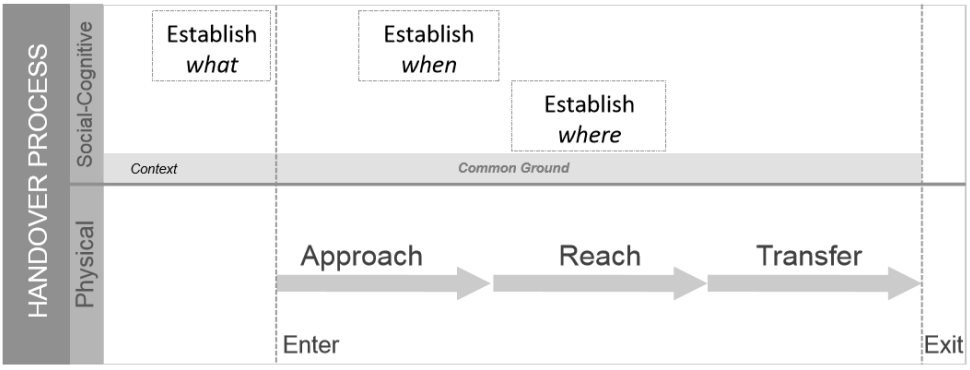


Figure 2.1: The handover procedure depicted as a physical and socio-cognitive channel. Shown is how the timing of those channels interlace: the 'what' is decided beforehand, the 'when' is established during the approach stage, the 'where' gets decided during the reaching stage, after which the object gets transferred from the giver to the receiver. The image is from Strabala et al. [33].

out object. Usually people start lifting their hand at 3m distance. This is the distance where the reaching phase ends and the transfer phase begins. The receiver sees the object and finds a grasp on the object. At that point both humans have a grasp on the object. During the transfer phase the object changes owner; the giver releases the object after he or she has made sure that the receiver has a good grasp and the object will not fall when they let go.

The context of the handover is also important. When people know and like each other, there is more allowance for proximity. Between two strangers more distance is kept between bodies as to not reach into personal space. Two strangers would also avoid touching each other and each others' hands as that seems too intimate.

Note that the topic of this paper is just finding a grasp for the virtual character on an object presented by a a human. We calculate a grasp for the exact moment between the reach and transfer stage, which is only a fraction of the time of the complete handover. The next section is about re-purposing human body language onto robots to improve the smoothness of human-robot handovers. This information is presented here only for context and to show how complex the process of a handover is.

2.2 Human-Robot handovers

The target of most research done on handovers is improving interaction between a human and a physical robot. This interaction is smoother if the robot can signal through 'body language'. The robot is not always humanoid, so the body language signals need to be designed differently. As the roles of giver and receiver have different characteristics, robot-to-human handovers have different requirements than human-to-robot handovers.

In this section we discuss several components associated with body language in handovers between robots and humans. During a handover, a giver cooperates with a receiver to make the object change hands. During cooperation, the participants need to be aware of each others intentions. Humans can accomplish handover tasks without use of speech; they signal their intentions to each other using body language.

Eye-Contact

Many papers report that eye gazing is an important cue during the handover action. Moon et al. [25] observe that during the reaching phase 68% of the human givers look at the position where the handover will take place. Another 18% look up to meet eyes with the receiver just before the transition stage. They let a robot imitate these two types of gazing behaviour and testes this during robot-to-human handovers. They compare the gazing behaviours to a behaviour that does not include gazing. They found that the gazing behaviour improves the speed in which the handover concludes and that the testers find the lookup behaviour (where the robot would look up just before the transition stage) the most natural of the three.

Navigation

During the handover process, the robot and human need to approach each other. Research has been done about finding robot trajectories that are safe and predictable to the human [15], [31], [19]. They try to copy human approach behaviour learned from user studies onto the robot, so that the human can better predict their actions and feel safer.

Glasauer et al. [12] inspect the velocity profile used in human-robot handovers. Among industrial robots the general velocity profile is typically trapezoid-shaped. The humanoid velocity profile is bell-shaped, which generates what is usually called a 'minimum jerk' motion trajectory. Glasauer et al. report that humans can better estimate the handover location when the robot uses a humanoid velocity profile, which lets the handover proceed smoother. This is especially the case when the robot also looks humanoid. Kulić and Croft [19] find a safe path for a robot to a human in order to allow for interaction: an interaction is likely to take place quite close to the human where a robot's quick (walking) motions can be dangerous for the human. They base their 'danger criteria' heuristic on the robot's inertia, distance to the human and the force of impact (would a collision with the human take place). Kajikawa et al. [15] find an approach path for a human-to-robot handover where the human and robot share the burden of navigating: when the human starts walking, they estimate the position where the robot and human would 'meet in the middle' and calculate a path towards it. Sisbot et al. [31] take into account human comfort when calculating paths for a robot. They consider personal space, visibility and the surprise effect when a robot suddenly appears from behind a close obstacle. Koay et al. [17] did a user study to find the preferred approach direction and position for a non-humanoid robot to bring a human an object. In their findings the robot is allowed to come quite close to the users (an average approach distance of 67cm). The preferred direction was to the front and slightly to the right.

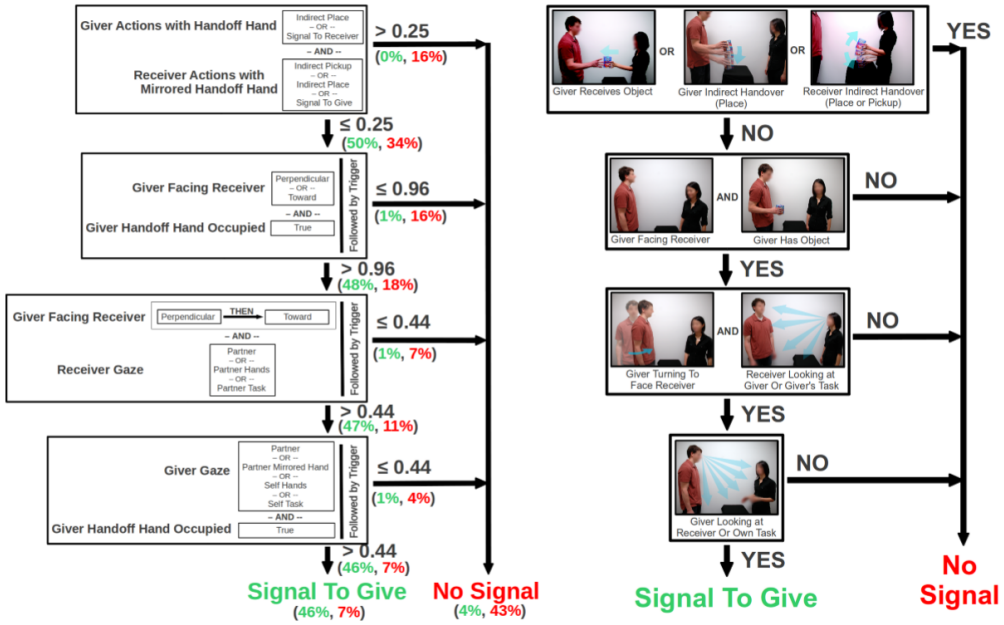
Cooperation

A handover is a cooperation task. During cooperation multiple individuals share the burden of the task, while having only information from their point of view. An individual can only know for sure what he or she plans to do. They do not have perfect world knowledge and the behaviours of others has to be predicted. Humans are fairly unpredictable beings, but there are patterns that they follow. The difficulty in human-robot interaction lays in modelling these patterns correctly. Specific cooperation tasks can be modelled through such patterns. Research has been done about cooperation between humans and robots. For example, Koppula et al. [18] trained a robot to assist a human in several household tasks like preparing food. They recorded video footage of a human

individually completing the task. Then they used machine learning to teach the robot what the human's next step in completing the task would be.

Understanding initiative

In a human-to-robot handover, noting will happen if the robot does not understand that the human wants to give it something. Strabala et al. [32] used the findings from their user studies to make a classifier to help the robot understand the human's body language in the context of initialisation of a handover. During a user study they showed users images of two humans; the users had to decide whether one human had the intention of giving an item to the second human. They trained a classifier (a decision tree) on this data, so that a robot might understand the body language of a potential giver. In Figure 2.2 this decision tree and its interpretation is shown.



(a) Sequence Feature Decision Tree

(b) Interpreted Decision Tree

Figure 2.2: A classifier trained from human perception data gained from a user study. The classifier is used to understand whether or not a human giver wants to give an object to receiver. In (a) the decision tree is shown, in (b) its used features are depicted with images for better understanding of their implications. Image is from Strabala et al. [32].

One of the features used is whether or not the potential giver is holding an object. However, for real world robots it is difficult to see if a human has an object in his or her hand. Due to limitations of computer vision methods, only pre-known objects that do not deform during handling can be noticed. Object occlusion due to the hand being around the object is also a problem. Pan et al. [26] tried to work around this by selecting features only from the skeleton pose of the human. They automatically extracted features from motion capture data (of both handover and non-handover actions) using a Support Vector Machine (SVM). The set of the most successful features were selected using a Random Forest method. Their robot observed human motion using Microsoft Kinect version 2 sensors, which can also extract a human skeleton pose. They use the SVM to classify the intentions of the human based on their body language.

2.3 The Human Hand

In this section we will discuss the anatomy of the hand, and how to build a virtual hand from this information. Because our grasp synthesis takes into account how 'comfortable' grasps are, we will also need to know some basics of ergonomy in the context of object-holding.

Anatomy

The human hand is a complex structure. The hand contains a total of 27 bones. Figure 2.3 shows its structure. The 8 carpal bones connect the hand to the rest of the arm and enable rotation of the hand around the wrist. In the palm there are 5 metacarpal bones. The digits connect to the palm at the heads of the metacarpals, where the knuckles lie.

The hand has 5 digits: 4 fingers and a thumb. Each digit connect to the palm with their proximal phalanx. The phalanx at the end of each digit is called the distal phalanx. The fingers have 3 phalanges: between the proximal and distal phalanx lies the intermediate phalanx. The thumb does not have an intermediate phalanx. The phalanges are connected through joints. From wrist to fingertip, there are the carpometacarpal joint (CMC), the metacarpophalangeal joint (MCP), the proximal interphalangeal joint (PIP) and the distal interphalangeal joint (DIP).

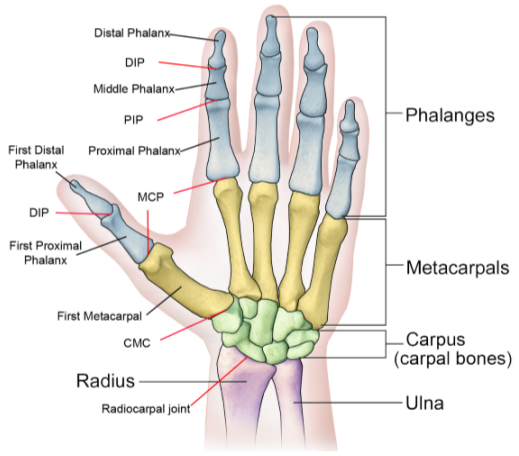


Figure 2.3: The bones of the forearm, wrist and hand. Acronyms: CMC - Carpometacarpal joint, MCP - Metacapophalangeal joint, PIP - Proximal interphalangeal joint, DIP - Distal interphalangeal joint. Taken from Figure 2 in [36] and shown with original caption.

The rotation of a joint is caused by contraction of muscles. The bending of the fingers is controlled by long muscles that originate from the forearm and run through the hand to connect to distal and intermediate phalanges. The way the tendons connect, in combination with the shape of the bones at the PIP and DIP joints, allows the fingers to bend in only one direction. Bending there joints for angles higher than roughly 100 degrees is not possible as the flesh of the lower phalanx is in the way. Bending a finger backwards is not possible without external help (for example when pulling it backwards with your other hand). Small muscles that originate from the palm control subtle sideways rotations around the CMC and MCP joints. Thus the MPC joints can rotate more freely than the PIP and DIP. The MPC can also bend slightly backwards, where the PIP and DIP cannot do that without outside help. All muscles together create a fair but limited range of motion for the fingers.

Virtual Hand

A virtual hand is a hand that exists in a virtual environment. Its visible part consists of a mesh. To animate a mesh we have to rig a skeleton into the mesh. In animation context a skeleton is a tree of links, or 'bones', which are connected by joints. By rigging the skeleton to the mesh, we connect (parts of) the mesh to the bones: Instead of animating the original mesh, we can animate the skeleton and the mesh's vertices will follow.

To rig a hand we have to place the tree of bones such that we can make the mesh move just like a real hand. It seems logical to look back at the anatomical structure of the hand and the positions of the joints inside of it. We describe the range of motion that bones can have based on the hand's anatomy. The carpal bones enable wrist motion in a real hand, but their individual motion is not visible from the outside. The metacarpals of the fingers have a very limited range of motion and so their movement is also barely visible from the outside of the hand. Because they have so little influence, we choose not to actually animate these bones. The motion of the phalanges of the hand is visible from the outside and does need to be animated. The fingers are chains of three bones, just like the three phalanges in their hand anatomy. The thumb has only two phalanges, but is a special case as it can rotate fairly freely around its MPC joint. So for the thumb we add the metacarpal to its bone chain. We end up with the following structure for representing hand poses: the wrist is the root of the skeleton, the fingers are four 3-long bone chains, originating form the positions of the knuckles and the thumb is a 3-long bone chain that originates somewhere on the thumb-side of the wrist. We will group the five 3-long bone chains under the name *digit chains*. In Figure 2.4 we a hand mesh rigged as such.

Figure 2.4: A rigged hand. The root of the skeleton is the wrist. From the root originate five chains of bones depicting the digits. The bones that we animate (the phalanges and the thumb metacarpal) are outlined in blue. The black outlined bones (several wrist and metacarpal bones) do not move during the animation; the joints between two static bones are considered rigid.

This hand model consists of 27 degrees of freedom (DoF). We need a position and orientation for the wrist (6DoF) and a rotation for the base of the thumb (3DoF). The base of each finger can rotate around two axes (8DoF). Each digit has a second and third joint that can rotate around one axis (10DoF). This high number of DoFs in a small place makes it difficult to avoid self-collisions. Models with other numbers of DoF exist; The number of DOFs can be lowered to 15DoF based on, among other things, interdependencies of the finger joints [20]. Our model has 21DoF, since we use for each finger the dependency of the rotation between the of the DIP and the PIP (or MCP for the thumb). Wheatland et al. [36] give a good overview of the biomechanical structure of the hand, and which of its properties are used to model it.

We have now described hand anatomy and how this inspires the build of a virtual hand. This information is used to build a model for making realistic hand poses in Section 4.4.

2.4 Grasp Types

In the previous subsection we have seen how a real hand is built and how its structure inspired the build of a virtual hand. We have also discussed how the rotation of some joints are constrained by the anatomy of the hand. There are still many poses that a hand is capable of attaining. Not all of the poses are grasps, and not all grasps are as comfortable or even possible. We want our grasps to be recognised as natural by humans. As such, we will discuss the types of grasps that humans commonly make in this section.

Feix et al. [10] collected papers categorizing types of grasps and made a general taxonomy for grasp types that are common for human hands. They state that a grasp (fitting their taxonomy) is every static hand posture with which an object can be held securely with one hand, irrespective of the hand orientation. As such, their taxonomy fits with the grasps our method should make, as our goal is to make stable grasps.

The first distinction that is made in grasps is to place them in one of the categories 'power', 'intermediate' or 'precision'. A power grasp encases an object with enough of the hand that the object cannot be moved with respect to the hand without loosening the grasps. The power grasp group includes many grasps where the object is pushed against the palm by the rest of the hand. A precision grasp is a grasp where the object can be moved with precision with respect to the hand. Many precision grasps hold onto the object through only the tips of the fingers. The intermediate grasps are a category between the power and precision grasps, as not all grasps are easily classified as just power or precision. In an intermediate grasp the object *can* be moved with respect to the hand, but in a limited manner. A second distinction is made for describing which part of the hand opposes the fingers. Sometimes this is the 'palm'. Other times it is the 'side' of the hand or fingers, or the 'pad' of the thumb. A third distinction that is made, and which Feix et al. contributed as a new distinction in grasp taxonomies, is the orientation of the thumb. The thumb can be opposite of the palm and fingers, which they call 'abducted'. Otherwise the thumb stands next to the palm, where it can either push the object to the side of the hand, or simply be out of the way of the grasp.

The three distinctions make for 17 subcategories of grasps, which are split up into at most 5 grasp types per subcategories. Figure 2.5 shows the 34 grasp types in Feix et al.'s taxonomy, organized per subcategory.

In our model, validity of a grasp is calculated based on a set of rules. However, the human eye might not always agree that a valid output grasp is visually convincing, or 'natural'. We could

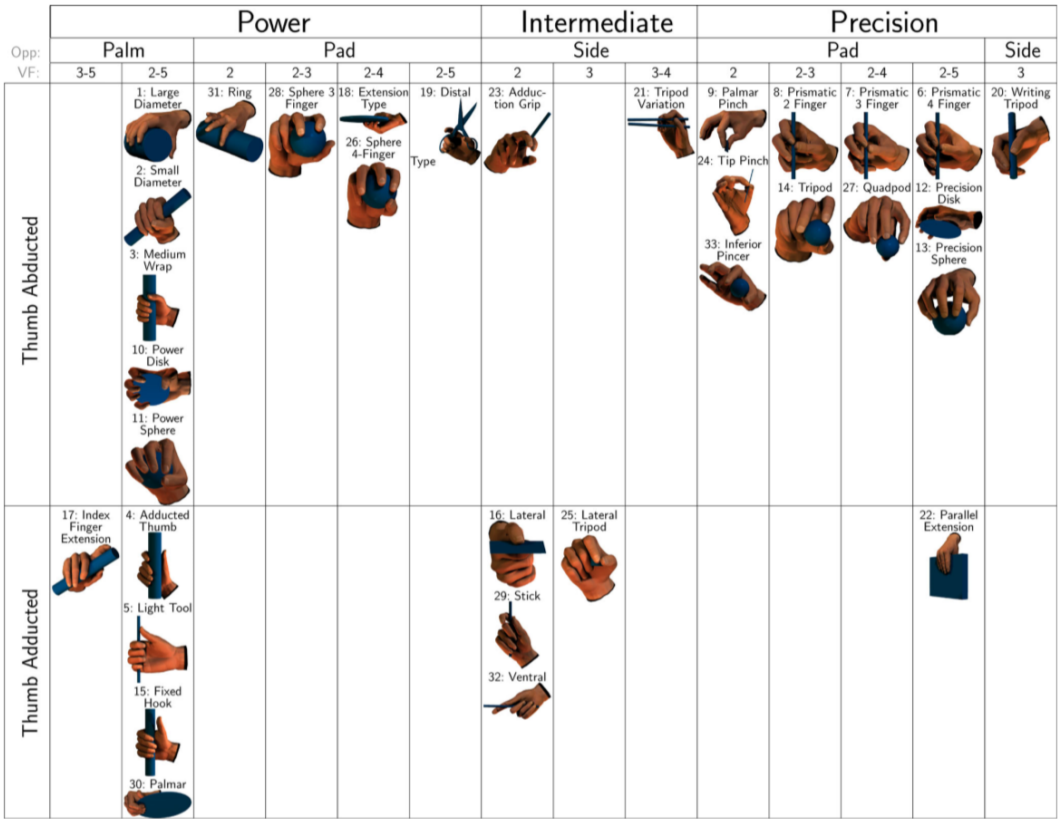


Figure 2.5: Taxonomy of grasp types, from the work of Feix et al. [10]. In the columns the distinction between power/intermediate/precision grasps is made first. These columns are split further between which part of the hand opposes the fingers in the 'Opp' row. 'VF' stands for 'Virtual Finger' and describes which fingers are used on the other side of the opposition (not all fingers are always included). The numbers in that row are the indices of the fingers, where 1 is the thumb and 5 is the little finger. The distinction abduction/adduction is made in the last two rows.

use this taxonomy to visually verify the grasps in order to see how good the output of the grasp synthesis method is. As it contains the common grasps used by humans, we propose that if we can classify a grasp as one of Feix et al.'s types, it is natural.

2.5 Ergonomy

We would like the grasps of our output to look natural, and thus look like a human would choose them. It is difficult to find an objective measure for how likely it would be that a human would make a certain grasp. We assume that humans would most likely choose a grasp that is 'comfortable'. Much research is being done in the domain of ergonomy. This domain includes, among other things, designing objects so that they can be held while causing minimum discomfort to the body. Objects can be designed, in both shape and weight distribution, such that they lie comfortable in the hand. Research in the other direction, finding hand poses that are comfortable on an arbitrary object, has not been done to the best of our knowledge. We did find some works about the amounts of forces that can be applied by the finger. This is what we base our comfort measure on.

Didomenico and Nussbaum [8] find a 'peak strength' of grasps by measuring the sum of forces applied on contacts after increasing the applied force for one second. As such they find the biggest force that would be used in tasks that are done often and repeatedly, like pushing buttons. They measure the peak force for one fingered pushing and pulling motion, where they measured force for only one contact. The average peak strength for these motions is $50N$. The average peak strength for a 'chuck' grasp, which seems similar to a Prismatic 3 Finger in the taxonomy in Subsection 2.4, is averagely $80N$. We also learn that for a 'palmar' grasp, similar to a Palmar Pinch in the taxonomy, the peak strength applied is averagely $54N$. Their 'grip' grasp (a power grasp) found peak strengths of $360N$. This high value is caused by the high number of contacts that are part of the power grasp. We use the peak forces we have just seen to see if there are force sizes as applied at contacts that would not be realistic. The palmar grasp contains 2 finger tips and the chuck contains 3. From which we predict that each contact on a finger can apply a peak force of about $27N$. Note that even though the pushing and pulling motions have only one contact, they have much higher peak force than the what we predicted for one contact previously. We suspect that these motions get additional force from the arm and the body, where the two precision grasps get their forces mostly through finger muscles. Didomenico and Nussbaum investigated the maximum force a grasp can apply in quickly repeated tasks. It would also be useful to find out how force is divided among fingers in human grasps.

We found the work of Radwin et al. [27], which investigates the forces used by the individual fingers. In a 'five-fingered pinch' grasp, similar to a Distal in the taxonomy, the force distribution between the fingers seems to be 33% for the index finger, 33% for the middle finger, 17% for the ring finger and 17% for the little finger. The ring and little finger together apply about as much force as the index or middle finger do individually. Note that the amount of force the thumb applies has not been specified. We assume that it can apply about as much force as the index or middle finger, as the thumb is not particularly stronger than either fingers; its usefulness lies mostly in its range of motion. From this we assume that a comfortable distribution of forces over the fingers would be approximately 25% for the thumb, 25% for the index finger, 25% for the middle finger, 12.5% for the ring finger and 12.5% for the little finger. We will use this assumption to propose an evaluation measure based on 'comfort' in Section 4.6.

2.6 Grasp Immobility Analysis

In our grasp synthesis method we will have to decide objectively whether a given grasp can realistically hold an object stable. This subsection we will discuss how to analyse a grasp on an object for immobility. We will use a running example of a box held between two the thumb and index finger. To analyse whether it is immobilized we need to model the forces applied by the finger on the object. As our grasp is a hand pose in a virtual context, we do not actually know where forces apply and what their size could be. So we have to find a model for simulating this physics first.

Let us first discuss the forces that are applied on the object. We assume that only gravity and contact forces apply on the object. There is a gravity force F_g on the object, anchored at the centre of mass (CoM) of the object. Gravity points down and has the size $9.81m$, where m is the mass of the object. Next to gravity there are contact forces of the hand onto the object, which are applied at contact points. From each contact point a contact force F gets applied by the hand on the object. The contact force can be split up in a normal component F_n and a friction component F_s. The normal force F_n is perpendicular to the plane of contact. The force of static friction F_s is parallel to the plane. In a frictionless model it is assumed that $F_s = 0$; as a consequence the contact force is equal to F_n.

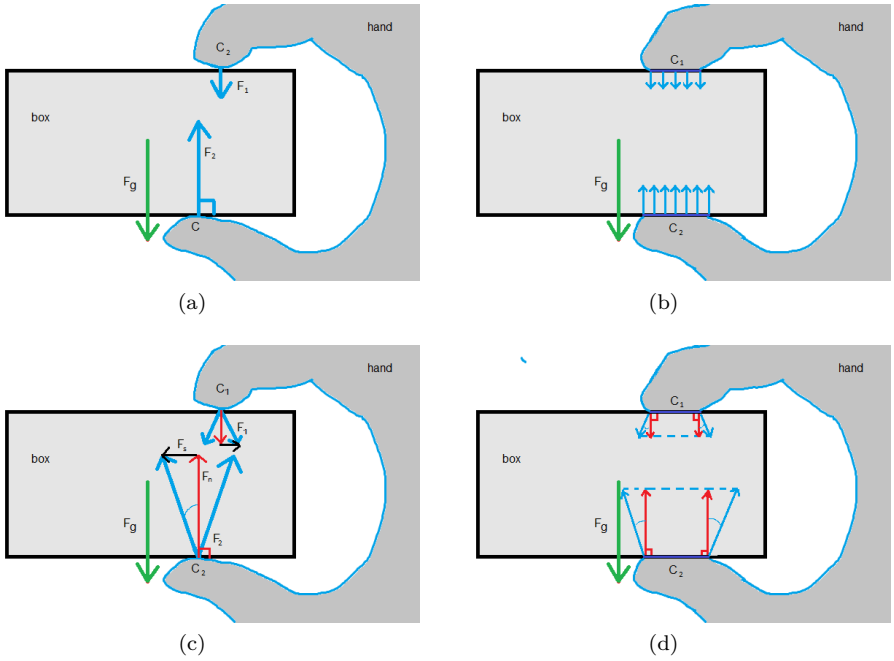


Figure 2.6: Different ways of modelling the contact forces for a 2-fingered grasp on a box in 2D. In (a) we treat the contacts as frictionless contact points. As such the contact forces F are equal to the normal forces. (b) A contact is seen as a frictionless contact area C. This can be modelled by small set of contact points sampled from the area C. Is is common to choose points on the boundary of C. In (c) we have contact point *with* friction. The static friction force F_s is applied parallel to the object surface and can have a length of maximally $|\mu_s F_n|$. This makes that any force inside the friction cone with half-angle $cos^{-1}(mu_s)$ is a valid contact force. (d) The contacts are treated as contact areas while also modelling friction: the friction forces applied on points on the boundary of C point away from the centre of C. This allows us to model the variety of possible contact forces that confirm to static friction, while keeping the total number of contact forces limited.

The boundary of a volume, its surface, is a continuous set of points. When two volumes 'touch', there exists a non-empty set of points that is the intersection of the points on the surfaces of the two objects. This set of points contains all *contact points*. Note that is does not have to be a connected set: there can be multiple disconnected continuous point sets. One such set is called a *contact area*. When a contact area is small enough, it is often represented using 1 contact point. However, the hand is a soft body, which means that when a finger touches the objects surface the finger deforms itself to fit against the object' surface: the size of the contact area is too big to capture in just 1 contact point. A way to overcome this is to represent the contact area by a discrete set of contact points, as in Figure 2.6b. It is common to find points for this small set from the boundary points of the area.

For a stable grasp we want the object to be at rest and immobilized. When the object is at rest the sum of forces and the sum of torques must be 0, from which follows that the sum of wrenches applied on the object must also be 0. A wrench w is defined as $W = [F, p \times F]$ where F is a contact force and p is the position of the (contact) point measured relative to the object's centre of mass. A common approach to check whether a given set of contacts immobilizes an object the *force closure* analysis [28]. In a force closure analysis we investigate whether the wrenches can withstand external forces and torques. If this is the case, it is implied that the internal forces block movement in all directions. Force closure exists when the convex hull of all wrenches applied on the object contains the origin. It has been proven that for a set of d-dimensional wrenches, d+1 wrenches are sufficient to immobilize the object. So for a 2D case where wrenches are 3-dimensional, 4 wrenches are necessary. For a 3D case a wrench is 6-dimensional and thus we need at least 7 wrenches to immobilize an object. In the situation in Figure 2.6a we modelled the contact forces as frictionless contacts points; as such there are only 2 wrenches and proving force closure is impossible. However, by adding more complexity to the modelling of contact forces (by either seeing them as a contact area or by adding friction) proving it becomes possible as the number of wrenches rises.

We have already explained that a contact force consists of a normal force and a friction force. Without friction, the contact force is equal to the normal force and thus perpendicular to the contact surface. There are two types of friction: static and dynamic friction. Static friction prevents slipping from happening between two objects' surfaces. This is the type of friction that applies on immobilized objects. Dynamic (or kinetic) friction applies on objects that slip relatively to each other. The Coulomb model states that static friction can only become as big as $|\mu_s F_n|$, in which F_n is the normal force and μ_s is the coefficient of static friction. Static friction turns into dynamic friction when the net external force becomes bigger than $\mu_s F_n$. The value of μ_s depends on the materials of the two objects and can be found empirically. There is no dependence on the size of the contact area or the roughness of the surface.

Our objects are held and thus we assume that there is only static friction and no dynamic friction. The magnitude of static friction can range anywhere between 0 and $|\mu_s F_n|$. Its direction can be any direction as long as it is parallel to the surface. This means that there is a continuous set of possible friction forces per contact points. Adding friction into the force closure analysis would mean that we need to check it for all possible versions of the friction force. Because there are an infinite number of possible friction forces, we represent them by a smaller set. A contact force F lies within a cone along the normal with half angle $\theta = cos^{-1}(\mu_s)$. The set of possible contact forces is continuous within the cone, so the wrenches that represent them are also continuous is wrench space. As none of the non-boundary wrenches influence the shape of the convex hull, they can be left out of the representing set. So we can represent a contact point with friction by a set

of wrenches on the boundary of the friction cone. In our example with the two fingered grasp on a box in Figure 2.6c, one contact point is represented by the 2 wrenches at extremes of the 2D cone.

Many grasp synthesis methods prove their grasp as valid by finding a set of forces for which force closure exists. They start from contact points and their surface normals and search for forces that uphold static friction. While sampling methods can find forces by sampling the friction cone, the methods that use optimization cannot: they need a parametrized formulation of the problem. Note that the sum of any two vectors inside the friction cone is a vector that also lies inside the cone. So any contact force conforming to static friction can be found by summing k vectors on the boundary of the friction cone as:

$$F_c = \sum_{i=1}^{k} \lambda_i F_i \tag{2.1}$$

where F_i is a force vector on the boundary of the friction cone and any $\lambda_i > 0$. It is common to find the vectors F_i by sampling a small set of vectors uniformly across the boundary of the friction cone. For simplicity we approximate the friction cone with four vectors (so $k = 4$), the directions of which are equally distributed around the cone. They call this approximation of the friction cone a 'friction pyramid'.

When combining contact areas with friction, the number of wrenches becomes high very fast. The higher the number of wrenches, the more difficult it becomes to find a set of force magnitudes for which the wrench sum is 0. We can reduce the number of wrenches per contact area by only taking the extremes. Instead of using one cone per friction force, we choose a mid point in the contact area, and use only the friction force F_i that points away from this mid point. This way we only have one wrench per contact point. This is shown in Figure 2.6d); We will use both friction and contact areas to model the contact forces for a 3D immobility analysis in Section 4.5.

In this chapter we have discussed human handover behaviour, which we needed to understand the requirements of handover grasps. We discussed it specifically in a context of the human-robot handovers. From this we have seen that there are many aspects to take into account to make the human and robot work together smoothly during the handover. We have then discussed the anatomy of the hand, from which we modelled a virtual hand and learned about the ranges of motion of the hand. Then we discussed common grasp types and ergonomy. Lastly we discussed how to model contact forces such that we can analyse whether or not the contacts of a grasp immobilize the object.

Chapter 3

Related Work

Finding a grasp for a (robotic) hand on an object has been a subject of research for a long time. In industrial settings this is used to pick up and move objects with robot arms. Some of those robots also interact with humans, for example when they hand out objects to the human. In Section 3.1 we discuss grasp synthesis methods that grasp on free-floating objects. We discuss grasp synthesis methods specifically for handovers in Section 3.2. Another domain that has researched object grasping is the animation domain. It seeks to automate the making of animation for the complex structure that is the hand during interaction with objects. We discuss these techniques and their usefulness to our goal shortly in Section 3.3.

3.1 Grasp Synthesis for Robots

Robotic grasp synthesis methods can be divided into two main groups. The first group uses inverse kinematics to find a grasp. This type of method creates contact points on the object surface and then finds hand configurations for which the fingers reach those points. The second group uses forward kinematics to find a grasp. They make a hand configuration and find the contact points on the positions where the hand touches the object surface. Our method is a forward kinematics type method.

A grasp that is to be used in the real world must be able to hold the object immobile; as such all grasp synthesis methods we found check their output for at least the force closure property. The sampling methods among them all evaluate grasps with respect to the *grasp quality* measure from Ferrari and Canny[11]. Grasp quality is measured as the radius of the biggest sphere with a centre at the origin that can fit inside of the convex hull. Thus for positive grasp quality values the origin must lie inside of the convex hull of the wrenches induced by the fingers. This means that the grasp quality implies force closure as well.

Inverse Kinematics

The first type of grasp synthesis methods use inverse kinematics (IK) to find a grasping hand pose. This means that they calculate the contact points on the object surface before finding a corresponding hand configuration. In most cases these methods create precision grasps, where only the fingertips will touch the object. For precision grasps the contact points on the finger are easy to define as they are part of a hand model. The contact points on the object surface are a

little more difficult to determine. This is because the forces applied to the object by the grasp should keep the object stable.

Borst et al. [4] describe a sampling method to finding contact points on the object surface for a 4-fingered gripper. For each sample they shoot 4 rays in different directions from a point between two opposing surfaces. The points that the rays hit become the contact points. They discard the samples where contact points lie too close together or are close to a surface that supports the object (like the table that the object stands on). Then they search for a hand configuration for which the fingers reach the contact points and avoid collision with the object. Their sampling method is relatively simple, but it is fairly specific to the shape of their robotic hand, a 4-fingered gripper. Their method might not translate well to hand models that might have a different number of fingers or alternative finger joint restrictions.

A solution to this could be to use the method of Rosales et al. [29]. They take a set of contact points as input and find a set of solution ranges for the configurations of the fingers. Finding any configuration for which the contact points and fingertip match up is a difficult problem. Fingers cannot move independent from each other and collision among the fingers, as well as with the object, needs to be avoided. They formulate the problem as two high-dimensional non-linear optimizations, finding the minimum and maximum solution to the same target function. This way they find two vectors describing the minimum and maximum corners of a high dimensional box.

They split this box up and recalculate the minimum and maximum for the sub-boxes iteratively, until their size is below a predefined threshold. So the output is either an empty box (when there is no solution) or a set of boxes he contain the possible solutions. The advantage of this is that a high number of possible solutions is found. It can also be applied to any type of hand, though this requires the user to model constraints of the new hand into the non-linear system. Additionally, between all solutions within the same box there exists a continuous path. This means that continuous motion/animation of the hand with respect to the object can be abstracted from the box. Disadvantage is that this method depends on other methods to supply contact points that fulfil the force closure requirement. This means that this method has to be combined with a method checking for force closure. Finding solutions for non-linear optimization is very difficult though, and this method uses it repeatedly. From this we conclude that this method can probably not run in realtime.

Harada et al. [14] manually create grasps for a set of bounding boxes of various shapes. They call these grasps 'pregrasps'. Suppose a grasp can hold a primitive shape like a box. Then it seems a reasonable assumption that similar grasps must exist that can hold objects similar to that primitive. During their grasp synthesis process they first find a bounding box that contains (part of) that object. Then they select a pregrasp for which these object bounds fit inside the bounding box used in the pregrasp. By their definition, the pregrasp's box is bigger than the object box, they can sample for the object box's position inside of the pregrasp box. They sample for contact positions by projecting the precalculated contacts onto the object's box. A candidate grasp is made by solving IK for the fingers such that each finger can reach its selected contact positions. Lastly, they measure grasp quality for a set of candidate grasps. They speed up checking for the force closure property by approximating it first with a much faster method, before measuring it with its precise (and slow) version. This speeds up their grasp synthesis process, because the fast calculation can root out most bad grasps. They use the slow force closure computation only for promising grasps.

Forward Kinematics

The second type of grasp synthesis methods uses forward kinematics (FK) to find a grasping hand pose. This means that they pose the fingers first and then derive contacts from these poses.

Miller et al. [24] use FK in their grasp synthesis method. Their method starts by creating a grasp on a primitive shape which is similar to the shape of the object. For each primitive shape they support (spheres, cylinders, boxes and cones) they create a hand configuration called a 'preshape'. While their idea of using a preshape is similar to Harada et al.'s pregrasp [14], Miller et al. evolve their preshape into a full grasp differently: they use FK instead of IK to find a final grasp pose. Miller et al.'s sample grasp consists of the chosen preshape, a position and orientation describing the configuration of the robot hand's palm. The positions and orientations of the sample grasps are generated using a set of rules. Sample grasps that are invalid, for example if they overlap the object, are discarded. For the remaining grasps they close the fingers around the object and then determine the grasp quality for the contact points where the fingertips ended up. Then the best grasp is chosen. A downside is that they have to decompose the object shape into a set of primitives by hand. Our method find grasps similarly, by placing an opened hand close to the object and closing the fingers.

Goldfeder et al. [13] extended on this work by automating the decomposition into primitives. They build a three of decompositions; at each node they split a part of the object that was previously captured with one primitive into two primitives. As such, the nodes further away from the root capture the object's shape more precisely than those closer to the root. A preshape is sampled on an individual primitive in a node, then a grasp is planned on the set of primitives that lie in the node. As such, their search space is a lot less complex then it would be if they used the whole object shape; this is what makes their method fast. Because this is a method meant for real world robots they still have to evaluate their samples for the force closure property, slowing the process down considerably.

The key problem of finding a grasp is finding a way to approach the object with the hand. Ciocarlie and Allen [7] let a user supply ways of approaching objects interactively. In a real world set-up, they find the position and orientation of a user's hand. From this hand configuration, sample grasps are made and improved using Simulated Annealing. The user is responsible for pulling the optimization out of local optima. A target function is used to guide the fingers around the object, but not onto the object surface. Simulated annealing mutates the current best sample to find a better one; this could lead to grasp samples clipping through the object surface if their parent sample is touching the surface. In a concurrent loop, the better grasp samples get optimized (the fingertips are put onto the object surface) and evaluated for force closure. The idea to have a user guide the grasp optimization is interesting, but is more useful for pre-creating grasps that are to be used at a later time.

Saut and Sidobre [30] precalculate a big set of grasps for an object to be selected in real-time. At runtime they sort this set of grasps on its appropriateness in a certain context. A set of points on the surface of the object are uniformly sampled (while saving information about local properties of the surface) and stored in a Kd tree. Secondly, the volume of points that the fingers can reach and touch, or the 'work space' of the fingers, is calculated. This volume is approximated as a set of spheres that lie strictly inside of the workspace. Around the object, a number of frames is sampled. The frames are evaluated: if the workspace spheres can intersect a surface point and the local surface information at this point is appropriate, then the object can be grasped. Force closure

and collision avoidance is tested and unfit frames are discarded. Then the grasps are ordered on a robustness measure, which is based on the local surface information at the contact points. At runtime the most robust grasp that works for the current situation is chosen. Saut and Sibore mention the possibility to use their method in handovers. Their method might even work in our case: most of the grasp creation is done offline, and at runtime a grasp gets chosen that does not overlap the opposing hand. This would fit our need for a fast method, as well as our need to find realistic grasp, as each accepted grasp would be applicable for use in the real world. However, only samples are kept that conform to force closure. In the case that the presenting hand occupies much space around the object, it might well be that all the possible grasps get rejected because they overlap with the presenting hand. For our problem it might be better to search for a smaller set of sample grasps that work especially for the given handover context.

3.2 Grasp Synthesis for Human-robot handovers

There are also methods from the robotics domain that find grasps specifically for use during handovers. For handovers in which a human and a robot participate, there are two handover directions. The object can be handed over by the robot to the human in a robot-to-human handover. Here, the robot has to find a grasp on the object while leaving space for the human to grab on as well. The method of Lopez-Damian et al. [23] does this by cutting the object geometry in two, and finding two robot grasps on the two object-pieces. Kim et al. [16] do more or less the same, except they specify two opposing surfaces for both robot hands to grab onto first. Both methods assume that the human can grasp the object if a grasp can be found for a second robot hand. This is not an unreasonable assumption: humans are more dexterous and creative than robots in making grasps. During a human-to-robot handover, a human takes the role of the giver and a robot is the receiver. This is a more difficult case as the robot has to sense that the human is initiating the handover action. Additionally, the robot has no influence on how the human holds onto the object. Here the challenge for the robot is to find a grasp that avoids the human hand. Most research about human-to-robot handovers focusses on recognising signals from the human (see Section 2.2). We have not found any grasp synthesis methods that are specified to human-to-robot handovers.

Edsinger and Kemp [9] show that humans intuitively hand an object so that the receiver can easily grasp it. As robotic hands are more limited than human hands in their movement, this speeds up the handover. By making use of this human behaviour, the process of identifying a good reach configuration for the robot hand can be simplified. The human will help solve this problem intuitively from their perspective. The intuitiveness behind this robot-helping holdout might imply that this also happens among humans; a human's holdout configuration might depend both on object functionality and the hand pose of the receiver. Their simplification of the reaching pose of the robot hand might thus also work for our virtual character. However, their method takes a lot of preprocessing, which makes it more difficult to enter new objects quickly. Next to that, as our objects are fairly tiny, many of the precreated grasps will be rejected at runtime because the presenting hand is in the way. This method could be extended on to be more appropriate for handovers. Checking that every grasp can be paired with at least one other grasp from the output set without overlapping, ensures that there is free space to place the hand in. This would increase the length of time spent pre-processing. However, this does still not guarantee that there is a grasp available in response to every human-made holdout grasp. It might be better to make a smaller grasp set that is targeted to the space that is actually available for grasping.

3.3 Grasp Synthesis for Hand Animation

In the field of animation, robotic grasp synthesis methods also find their use. Animation of hands manipulating objects manually is time consuming due to the high DoF of the hands. Hand motion synthesis methods generate animation of the hands automatically. This also includes manipulation motion, in which the hand is in contact with objects. When it comes to manipulation motions, several techniques split the process up into a number of phases [2], [3], [38]. These phases usually are variation on the three actions *approach*, *actuation* and *release*. Actuation is the phase in which the object is in contact with the hand and is being manipulated. Zhoa et al. [38] split up the approach phase in a *reaching* and a *closing* phase. Having a grasp on the object implies that the object is manipulated.

Sueda and Pai [34] reviewed several methods of hand motion synthesis methods using musculoskeletal simulation. According to them, torque joint models are the most useful for manipulation tasks, because they give direct control of the skeleton. Other techniques (passive muscles, moment arm models, dynamic strands, volumetric models) are not useful for manipulation tasks due to the high complexity of the hand.

Manipulation motion synthesis techniques for hands are mostly physical-based [22], [21], [37]. There are also data-driven physical-based techniques [2], [38]. These techniques, however, require either a target grasp pose or the object's desired trajectory as input, both of which we lack. An exception is Zhao et al. [38], which only requires a grasp type and the object mesh as input. Their motion capture database contains of animations of picking up 10 base object geometries, each with 10 types of grips. The geometry of the object to be manipulated gets analysed to find a best match within the set of base objects. Given a grasp type, an animation can be created by following a path through poses in the example data. This whole path is then optimized. Since it is a physical simulation, they also optimize the grasp quality of the grasp: the forces applied by the fingers need to keep the object stable. However, just like other manipulation motion synthesis methods, this method 'renders' a motion to generate a complete animation as output. Because in our case the object is presented by the human in real-time, we cannot use methods of this type.

This chapter we have seen several grasp synthesis methods. Some use IK to find finger poses, others use FK. The sampling methods for grasp synthesis for robot hands are not designed for interactive use: they use grasp quality to validate and rank the grasps on, which is a time expensive operation and as such not suitable to our situation. Saut and Sidobre's method [30] is an exception, as it pre-calculates a set of grasps and ranks them for usefulness at realtime. However, the shape of the presenting hand in our handover context is only decided at runtime. The presence of the presenting hand might invalidate all pre-calculated grasps. So it might be better to find grasps at run-time, where we can guide the search away from grasps that collide with the presenting hand. The methods used in the animation domain are optimization based and find not only a grasp but a complete motion path for the hand. As they compute a full motion path, they cannot be used in an interactive scene. Thus these methods are not useful in our context.

The common drawback of grasp synthesis methods is that they use grasp quality, a time-consuming calculation. Grasp quality is useful as it both implies force closure and ranks the grasps on immobilizing ability. However, in our context, we need grasps to be believable more than we need them to be physically immobilizing. Our method does *not* use grasp quality, but an approximated force closure analysis to validate our grasps. We rank them with a 'comfort' measure based on ergonomy. As such our method is faster and thus more appropriate for use in real-time

than most existing grasp synthesis methods. In the next chapter we will explain our own method for grasp synthesis for handovers.

Chapter 4

Methodology

In this chapter we will first select a type of synthesis method in Section 4.1. In Section 4.2 we list some aspects of realism we want our output to conform to. Then, in Section 4.2 we explain the intuition behind our grasp synthesis method and expand it into an algorithm. In Sections 4.4 and 4.5 we outline the details of our algorithm. In Section 4.6 we propose an evaluation measure that might rank the output grasps on quality.

4.1 Method Type

In the previous chapter we have seen that there are several general types of methods for finding grasps on non-deformable objects. One option for solving our problem would be to use a data-driven method. This includes using motion capture data of a hand picking up objects to construct grasps for a given object. Data-driven methods are better suited to fullbody motion synthesis than to hand motion specifically. Markers on hands need to be placed very close together in a small space which makes it difficult to distinguish between them. Additionally, in certain poses (for example a fist) the hand obstructs the visibility of markers from the cameras, resulting in missing data. The difficulties in capturing motion of hands is a reason for us to seek a different approach. Another option is to use an optimization method. Here the pose requirements are formulated as an optimization problem. However, due to the many degrees of freedom in the hand and the requirement that no overlap should occur this ends up as a high dimensional constraint non-linear problem, which is hard to find solvers for. Incorporating the forbidden space into this optimization might also prove a problem. Iterative optimization algorithms are prone to local optima and are too slow for interactive use. Following this the optimization method is not viable for us.

The option that we study is a sampling method. Sampling methods generate a set of samples and evaluate them to find the best sample among them. This is one way to deal with optimizing high dimensional problems; this approach does not create optimal grasps, but it can find one that is relatively good (with respect to the rest of the samples). In general, the larger the sample set the better the solution that will be obtained. Sampling is well suited to real time problems as long as the grasp generation and evaluation can be done relatively quick. Additionally, one can tune the number of samples to balance the need for speed with the quality of the output.

A sampling method consists of three building blocks: generation, validation and evaluation. A set of samples is generated. Each sample gets validated for a number of conditions. All the valid samples are evaluated and ranked: the best sample is given as the output. This thesis will focus

on the generation and validation of grasps. Though we have an idea in mind for evaluating grasps (see Section 4.6), we did not implement or test it due to time constraints.

4.2 Algorithm Outline

We have stated in Section 1.1 that we are looking for grasps that take hold of an object presented by a human user in a realistic manner. We will first define a number of aspects that together form our notion of 'realistic'. After this we will explain how we include these aspects in our model. Then we show the main outline of our algorithm.

We define four aspects in which our grasp should be realistic. Firstly, the pose of the hand should look *anatomically plausible*. This mostly means that the angles of the finger joints should lie within the human range. The second aspect of a realistic grasp is that it should look like it holds the object *immobile*. In the context of the surgery room our objects should not fall as they will loose their sterility that way. In the real world this would imply the need for a firm grip on the object. In our virtual scenario we can just define the object's position with respect the virtual hand. However, to make it look believable that the object stays in place in the virtual hand, its grasp needs to look like it immobilizes it. A third aspect of a realistic grasp would be that we must *avoid deep collision* of the virtual hand into the space occupied by the object and user hand. In the virtual world it is in principle possible for several meshes to occupy the same space. However, when big parts of the meshes overlap, it looks unrealistic to the user as intersection of hand and object is impossible in real life. With 'deep' collision we refer to two meshes overlapping such that, visually, it cannot be explained by imagining that (one of) the meshes deform(s) to allow space for the other. In our case, the hand deforms when touching the object, as we assume our objects to be rigid. A 'superficial' collision would look like a touch. Though we allow superficial collision (which allows that we find contacts), we will not allow deep collision in our grasp output. Our fourth aspect of realism is specific for our handover scenario: we allow for parts of the object that may not be touched. For example, our surgery instruments must remain sterile. Another example is the sharp edge of the scalpel, where a touch may result in cutting the contacting parts of the hand. A less dire example would be grasping on the buttons of the remote and accidentally pushing them. This way we disallow grasps that are unbelievable as fingers touch the object where a human would not touch them.

The intuition behind our grasp synthesis algorithm is simple: we approach the object with an open hand and when near enough we close the fingers. When we say a hand is 'open(ed)', we mean that its fingers are stretched. The hand is 'closed' when the fingers are bent. The hand can be closed around an object, like in a power grasp. Our method of simply closing the hand around an object imitates how a human would make a grasp. Humans do not exactly plan the pose of their hand during grasp, nor do they think heavily on the exact location of the contacts. They start with a general idea for a grasp and shape their hand to the object while bending the fingers. The preshape of the hand, as Miller et al. [24] call it, depends on the shape of the object. As we want our method to work in general for small and light objects, we start at all times with an opened hand as a preshape. Then we simulate bending the fingers until they touch the object. This bending simulation ensures that the resulting grasp is at least anatomically possible. It also makes for a good way of finding finger poses that both touch the object but do not penetrate it too deep: these are the types of finger poses we desire to find. We observe that in a human hand the phalanges rotate around the joints in a relatively constant speed when the fingers are closing

(and the object does not block movement yet). We take this into account in our model. After simulating the hand closing and finding touching finger poses, we validate the resulting grasp: we check the existence of sufficient contacts, the lack of deep collision, the positions of the contacts and if the grasps immobilizes the object. This amounts to a 5-step algorithm, for which the steps are named and described as follows:

Step 1 Wrist Placement: the wrist is placed somewhere around the object, such that there exists a pose for which the hand can touch the object.

Step 2 Finger Posing: we start with an open hand and close the fingers around the object. We check that there is at least one contact on the thumb and one on a finger.

Step 3 Collision Validation: we check that the hand does not penetrate too far into already-occupied space (the object and human hand).

Step 4 Contact Validation: we check that the contacts do not lie on the predefined forbidden surfaces of the object.

Step 5 Immobility Validation: we check if there exists a set of forces as applied by the contacts for which we can oppose the gravity on the object.

In the function TrySynthesizeGrasp in Algorithm 1 we show the 5-step algorithm through pseudo-code. Names of subfunctions that are printed in all-caps font will be expanded on in Algorithms 2-10, which are discussed in the remainder of this chapter. First we find a wrist placement (line 2), after which we find touching poses for the fingers and check if there are sufficient contacts (lines 3 and the first part of line 6). Then we check that the hand does not collide with the object or presenting hand (second and third part of line 6). We check that the contacts do not lie on forbidden surfaces (fourth part of line 6). Lastly we analyse if the found grasp can immobilize the object (fifth part of line 6). If any of the checks fail, the found grasp is invalid (lines 7 and 8). If we passed all validation steps, we output a valid grasp (line 11). Algorithm TrySynthesizeGrasp can fail, which means that we are not guaranteed a valid output. We attempt making grasps until a valid one is found.

In the following sections we will more precisely describe the 5 steps of which our algorithm consists. In Section 4.3 we describe the Wrist Placement step of our algorithm. In Section 4.4 we describe the Finger Posing step of our algorithm. We explain the hand (motion) model we use for finding anatomically plausible hand poses. In Section 4.5 we explain the three algorithm steps related to the validation of the output grasps: the Collision, Contact and Immobility Validation steps. In the last section of this chapter, Section 4.6 we propose a way to evaluate the grasps that our algorithm outputs.

Algorithm 1 TrySynthesizeGrasp

Input: Graspable object objectToGrasp, hand presentingHand
Output: If a pose is outputted, this pose is valid

```
1: Generation of a grasp
2: wristFrame ← PlaceWristFrame(objectToGrasp)
3: pose, contacts ← CalculateHoldPoseAndContacts(wristFrame, objectToGrasp)
4:
5: Validation of this grasp
6: if !GraspingContactsExist(contacts)
   or CollidesWithMesh(pose, objectToGrasp.Mesh)
   or CollidesWithMesh(pose, presentingHand.Mesh)
   or CollidesWithForbiddenSurface(contacts, objectToGrasp)
   or IsImmobilizingGrasp(contacts, objectToGrasp) then
7:    This grasp is invalid
8:    return null
9: end if
10:
11: return valid grasp
```

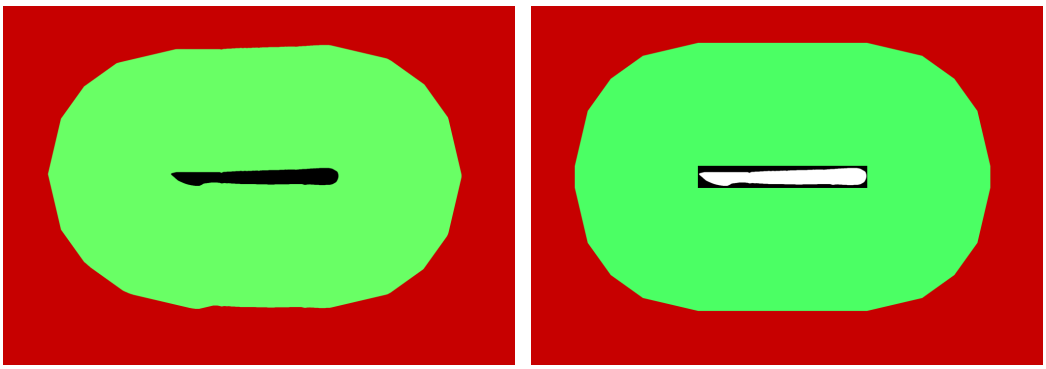
4.3 Wrist Placement

In the previous subsection we explained our intuition for finding grasping poses on objects. We have said that we would place the hand 'near the object', and then close the fingers. In this section we will describe how we find a placement for the wrist.

In function PlaceWristFrame in Algorithm 2 we give pseudo-code for placing the wrist. We know that only for wrist placements that are near enough to the intended object the fingers can touch the object after closing them. However, placing the wrist too close to the object often has the consequence that the virtual hand penetrates into the object. The position should not lie inside of the object as this instantly invalidates the grasp (the hand overlaps the object). As we discussed at the start of this chapter, we see the absence of deep collision between the hand and occupied space as one of our aspects of a realistic grasp. Thus, the chosen positions lie outside of the object, but within a distance where the hand should be able to touch the object.

Within the hand, the tip of a stretched middle finger has the maximal distance d from the wrist. The value of d depends on the hand mesh used. In our hand rig we have $d = 22$cm. The wrist must be placed with a distance d from the object to be able to touch fingers to the object. So there is a volume around the object of which we are certain that it contains the wrist positions of all valid grasps. We call this the *reachable volume* of the object. We intend to place the wrist at a position inside this volume. However, calculating the exact bounds of the reachable volume entails doing calculations on a mesh. Checking if a point lies inside this volume is an expensive operation as we must calculate the distance to the mesh. We prefer to approximate the volume with two primitive shapes, for which we can easily check if points lie inside. The objects we consider (the remote, scalpel and scissors) have bounding boxes that fit reasonably well around these objects volumes. They fit at least well enough that we can say that placing the wrist at any position inside their bounds has a high probability to cause the hand to penetrate onto the object. So instead of finding a point in a mesh volume we approximate this volume with two bounding shapes: one box B that contains the object (and is approximating the object-shape) and a larger capsule-like shape C that encases the reachable space. B consists of a centre point O and 3 orthogonal vectors X, Y and Z that originate from the central point. The lengths of the vectors represent the size of

the bounding box. All points $\{(O + uX + vY + wZ) \in \mathbb{R}^3 \mid u, v, w \in [-1, 1]\}$ lie inside the box. B is assumed to be known in our algorithm and should thus be defined before running it. In our implementation we use a library to calculate B, but it can also be made manually. We construct $C = B \oplus S$, where S is a sphere of radius d centred at the origin. In Figure 4.1 we show both the reachable volume and its approximation for a scalpel in 2D. The two volumes look very alike. This is caused by how well the bounding box fits around the object. The space in the bounding box that is not filled by the object's volume is thin with respect to d. As such the reachable volume of the object's surface lacks the concavities that the object's surface has.



(a) (b)

Figure 4.1: A 2D example of our way of approximating the reachable volume of a scalpel. (a) The reachable space (green) contains all points within a distance d of the scalpel (black). (b) The approximation of the reachable space (green) lies maximally a distance d away from the bounding box (black) of the scalpel. Note how similar the two green volumes in (a) and (b) are.

We randomly generate a wrist position at a point p inside of C. Since per definition $B \subset C$ we might generate a p that lies inside of B. We retry point generation when it lies inside of the object box B. The rotation of the wrist is chosen randomly. We have defined the reachable volume of the object to contain all positions for which a valid placement can be found. As $C - B$ approximates this space fairly well for object that have wet fitted bounding boxes, we know that $C - B$ contains the majority of positions for which valid placement can be found. However, this space also contains many invalid placements. In Chapter 5 we will experiment with parameters for decreasing the volume from which we choose the wrist placements. We also argue for a few measures that conciser whether a placement is promising or not, in order to prune out bad placements early. This should increase the probability that the output grasp is valid. We will experiment with constraining rotation choices for this same reason.

Algorithm 2 PlaceWristFrame

Input: Graspable object objectToGrasp
Output: A frame for the wrist

```
objectBox ← bounding box B that fits tightly around the object
objectCapsule ← gets the volume C of points that lie maximally d away from objectBox

position ← random position in objectCapsule - objectBox
rotation ← random rotation

wristframe ← (position, rotation)
return wristframe
```

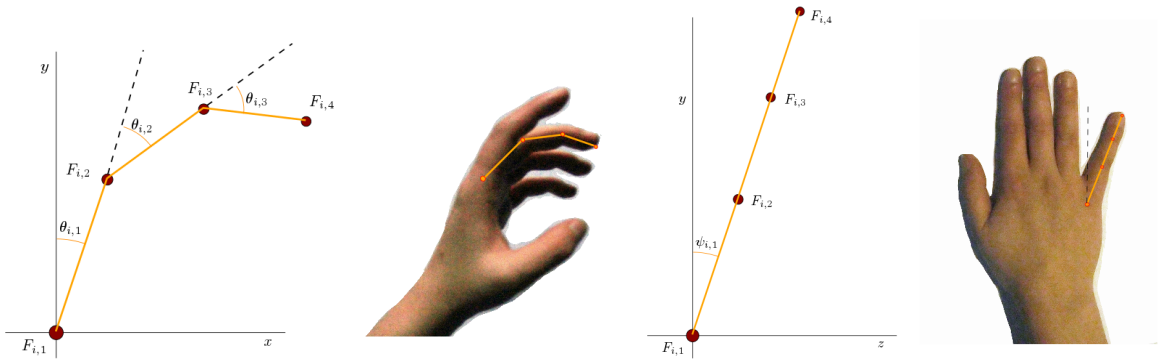
4.4 Model for Finger Bending Motion

In the previous subsection we have explained how we place the wrist such that it is near the object. Now we will explain how we find finger poses such that they touch the object. In order to simulate the bending of the fingers we must make a model for the closing movement of the fingers. This model must be linked to the shape of our hand rig. The hand rig consists of a free base (the wrist), from which originate five digit chains (the five fingers). The digit chains are 3-linked robots with a fixed bases that are placed relatively to the wrist. From the rig we can extract the relative positions of these fixed bases. We also know the lengths of the bones. These variables are static during motion of the hand, but might have different values for different types of hand meshes.

In Section 2.3 we have discussed how the muscles allow the fingers to move. During hand motion, the bones in the hand rotate around the joints. The fingers do not move completely independent from each other. In a real hand, which consists of many tiny muscles and bones in a small space, the motion of one finger has some influence on other parts of the hand. For example, the little finger usually does not move completely independent from the ring finger. Additionally, a finger cannot move into the space that a neighbouring finger already occupies. However, modelling those subtle interdependencies of motion between fingers is not an easy task. Thus we model motion of the digits as if they move independent from each other. Our intuition was to close the fingers around the object. However, since we threat the fingers as independent, we close the fingers independent from each other.

First let us discuss some terminology for clarity. The *front* of the hand is the side that contains the palm. A vector that points 'forward' with respect to the hand points perpendicularly out of the palm. A vector that points 'upwards' with respect to the hand points in the direction of the stretched fingers. The front of a finger or phalanx is the side where the finger pads lie, just like the palm which also lies on the front-side. We will now start with the modelling of one finger (note that thumb is a special case and will be treated later). The *base joint* is placed at the position of the knuckle, which equals the MCP joint for all fingers except for the thumb. The *second joint* and *third joint* correspond to the PIP and DIP joints of the finger respectively. We use the terms base, second and third instead of using the abbreviated names of the joints, because the equivalent joints in the thumb have different abbreviated names.

A Finger is a chain of 3 joints originating from the wrist frame F_0. For a finger i, the frames $F_{i,1}$, $F_{i,2}$ and $F_{i,3}$ are placed at the base, second and third joint respectively. The frames are placed such that the phalanx aligns with the local +y axis. A finger can bend forward for all three joints, and rotate sideways for the base joint. We model these rotations as angles relative to a default orientation. The default orientation of the finger joint is such that it correspond to a stretched finger. To bend the finger forward, we rotate a joint θ degrees along the local z-axis on the joint. We will refer to this bend direction as a *pitch angle*. To rotate sideways we ψ degrees around the local x-axis. To this bend direction we will refer as the *yaw angle*. Both directions of rotation are shown in Figures 4.2a and 4.2b respectively. The remaining rotational axis is the y-axis. Rotation around this axes would be described by the roll angle. However, the range of motion that is possible for normal fingers around this axes is neglectable; we thus keep the roll angle fixed.



(a) (b)

Figure 4.2: Modelling of a finger x. (a) Bending the finger forward along the z-axis, the left hand has positive pitch angles for all its joints. (b) Bending the finger sidewards along the x-axis, the right hand has a positive yaw angle for the little finger

The thumb is a little different from the other fingers, though we treat it almost the same in our model. The base joint of the thumb is the CMC, the second joint is the MCP and the third joint is the DIP. The thumb is a special digit as its base joint *can* rotate around the y-axes. It is, however, very difficult to model for which roll-yaw combinations a pose would be considered 'physically plausible'. We find it easier to model the thumb with just two rotatable axes: we set the roll to a constant 0 and vary only the pitch and yaw angle. The main difference from a normal finger in our model is that the allowed yaw range is relatively large.

So for any digit i we have a pitch angle for each joint, and a yaw angle for the base joint. We number the digits $[0, 1, 2, 3, 4]$, with 0 being the thumb and 4 the little finger. A finger pose of finger i would be described through four angles, a pitch for each joint and the base yaw angle. However, there is a dependency between the pitch angles of the PIP and the DIP so that $\theta_{i,3} = \frac{2}{3}\theta_{i,2}$ [1]. We use this dependency to simplify the finger pose so that we have only 3 variables per finger. A pose would thus consist of a pitch and a yaw angle for the base joint, and a pitch angle for the second joint. We thus describe a finger pose $q_i = [\theta_{i,1}, \psi_{i,1}, \theta_{i,2}]$.

A finger can bend about 90 degrees forward, and a limited number of degrees to the side. The thumb's base lies inside the palm and has a smaller reach forwards, which is is compensated by its large sidewards range. These considerations are summarized in Table 4.1, where we show the constraints to our finger pose space.

Finger	Range of base yaw angle ψ_1	Range of base pitch angle θ_1	Range of second pitch angle θ_2
Thumb	$[0, 45]$	$[0, 60]$	$[-45, 45]$
Index	$[-5, 5]$	$[0, 90]$	$[0, 90]$
Middle	$[-5, 5]$	$[0, 90]$	$[0, 90]$
Ring	$[-5, 5]$	$[0, 90]$	$[0, 90]$
Little	$[-5, 5]$	$[0, 90]$	$[0, 90]$

Table 4.1: The allowed ranges of the variable joint angle for each finger, in degrees.

The digit chains have their base placed in a fixed position relative to the wrist. The wrist frame F_0 originates at the wrist of the hand rig. As a frame is a 6 dimensional point and the finger poses are 15 variables, a complete configuration for a hand pose is a 21 dimensional vector.

If we know the default orientation of a finger and the lengths and widths of the phalanges, we can approximate the volume where the flesh of a posed finger resides. This is useful when finding contacts between the object and the hand and collision checking. If we were finding a grasp to use in a real-world situation, we would have to be very precise in finding contacts and collisions. Simulating this, we would have to find collisions and contacts are found by intersecting the meshes of the hand and object. Comparing meshes is a fairly time consuming operations. As we do not require this level of precision, we approximate the surface of the mesh of the hand by a set of line segments. During hand-closing motion, the fingers bend forward. As such we expect that contacts will lie on the front of the fingers. We check if these line segments intersect with the mesh of the object to find the contacts.

Let us formulate how we find the approximate volume of a finger through line segments. Suppose a phalanx j of finger i has a length $L_{i,j}$ and diameter $W_{i,j}$. Then its approximate volume can be described as the set of local points :

$$V_{i,j} = \{(x, y, z) \in \mathbb{R}^3 \mid 0 \leq y \leq L_{i,j} \text{ and } 0 \leq \sqrt{x^2 + z^2} \leq \frac{1}{2} W_{i,j}\}. \tag{4.1}$$

This describes each phalanx j as a cylinder of height $L_{i,j}$ and radius $\frac{1}{2}W_{i,j}$. Note that as $V_{i,j}$ is a cylinder and thus a convex shape, any line segment between any two points in $V_{i,j}$ does not intersect any space outside of the cylinder. In Subsection 4.5 we use this formulation again to check for deep collisions.

Note that $V_{i,j}$ consists of positions *local* to frame $F_{i,j}$. So before we can check for deep collisions, we must transform the two endpoints of the line segments to global coordinates. We know the configuration of the free base of the hand as wrist frame $F_0 = (p_0, r_0)$. We also know the default local position and rotation vectors of the fixed base of each finger i relative to its parent frame. To find the global position of a point p that is local to $F_{i,4}$ (the distal phalanx) we calculate:

$$T(p_0)R(r_0)T(p_{i,1})R(r_{i,1})T(p_{i,2})R(r_{i,2})T(p_{i,3})R(r_{i,3})T(p_{i,4})R(r_{i,4})p \tag{4.2}$$

where $T(position)$ and $R(rotation)$ are translation and rotation matrices respectively. We show values for the local positions $p_{i,j}$ and local rotations $r_{i,j}$ of joints with respect to an earlier joint in the chain as in Table 4.2. We rewrite equation 4.2 in a recursive formulation. The position p local to the frame $F_{i,x}$ would be transformed to global coordinates through the equation:

$$G_{i,x}(p) = \begin{cases} G_{i,x-1}[T(p_{i,x})R(r_{i,x})p] & \text{if } x > 0 \\ T(p_0)R(r_0)p & \text{otherwise} \end{cases} \tag{4.3}$$

This function shows the relation that a joint has with the earlier joints in the digit chain: it finds the position with respect to the local frame, then transforms it by the position and orientation this frame has with respect to an earlier frame. We recurse until we get to the wrist frame, which is the root of the chain, and output the now global coordinate. (Note that this function only expects input for $x \in 0, 1, 2, 3$ as the digit chains are only 4 frames long.)

We have now formulated how we find positions in and on the fingers. Let us take a step back at how we can use this for finding finger poses in our algorithm. We have explained that we find

Joint	phalanx index j	local position $p_{i,j}$	local rotation $r_{i,j}$
Base joint	1	default	default $+(\theta_{i,1}, \psi_{i,1}, 0)$
Second joint	2	$(0, L_{i,2}, 0)$	$(\theta_{i,2}, 0, 0)$
Third joint	3	$(0, L_{i,3}, 0)$	$(\frac{2}{3}\theta_{i,2}, 0, 0)$
finger tip	4	$(0, L_{i,4}, 0)$	$(0, 0, 0)$

Table 4.2: Definitions of translation and rotation vectors used to do the forward kinematics of phalanges in Equation 4.3. The rotations are defined as Euler rotations. These are the local coordinates of a joint with respect to its parent joint.

a finger pose by bending the finger until it hits the object's surface. We have written pseudo-code for this in Algorithm 4. In this pseudo-code we have already placed the wrist, so we know where the finger frame is. We sample a yaw angle within its allowed range. Then we set the base pitch angle and second pitch angle to their minimum value (lower ends of the ranges of pitch angles, see Table 4.1). The third pitch angle depends on the second pitch angle. During hand-closing motion, the fingers bend forward. As such we expect that contacts will lie on the front of the fingers. We increase the pitch angles stepwise, and at every step we cast 3 *touch segments* across the front of the phalanges. When one of these line segments hits the object, we find a contact at that point. The touch segment for a phalanx j on finger i is defined as the line segment AB with $A = G_{i,j}(0, 0, W_{i,j})$ and $B = G_{i,j}(0, L_{i,j}, W_{i,j})$. Remember that the +z direction was the forward direction of a frame placed at a joint and +y the upward direction. So the touch segments lie on the front of the fingers as in Figure 4.3.

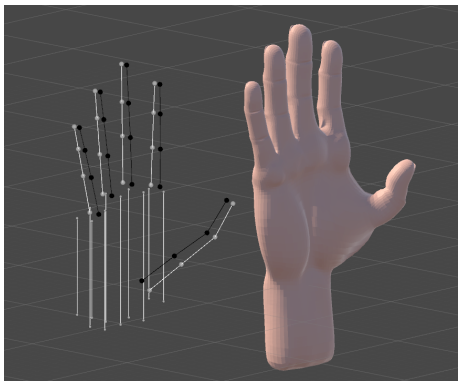


Figure 4.3: The collision segments (white) and touch segments (black) of our hand model next to the posed hand mesh. The collision segments lie under the surface of the hand and the touch segments of the finger lie on the front of the fingers.

During the closing of a finger, all three pitch angles increase at the same time at a constant rate. However, the speed at which the pitch angles increase is not the same for each joint; we have already said that we use the ratio $\frac{2}{3}$ as a dependency between the pitch angles of the second and third joints. From observation during the closing of a human hand, we saw that there is also a relation between the pitch angles of base and the second joint. We call this the *bend ratio*. We introduce this as a variable of our search space as follows: for each finger i we have pitch angles $\theta_{i,1}$ and $\theta_{i,2}$. Bend ratio ρ_i describes the relation between the two angles as $\theta_{i,2} = \rho_i \theta_{i,1}$. To make sure the second joint does not bend backwards we need $\rho_i > 0$. We assume values near 1 work best as we have observed that more extreme ratios in real life finger poses are unnatural: when they happen they are often cause by the object being in the way of one of the angles, and not caused by

natural motion. To allow this into our closing model we keep iterating on the second pitch angle after a contact point is found on the proximal phalanx. Of course in this case we set the increase per step of the base pitch angle to 0.

After we have placed the wrist, we have to find poses for the fingers such that they touch the object. In the function CalculateHoldPoseAndContacts in Algorithm 3 we find a pose for all five fingers. At the same time we find and save the contacts that accompany the found pose. Function FindHittingFingerPose in Algorithm 4 is called for each of the fingers. In this algorithm, we try finger poses and analyse them for (the position of) contacts. We start from what we assume to be a non-touching finger pose (the finger is stretched) and iteratively increase the pitch angles. As such, when we find a pose that touches the object, there is a low probability that this pose deeply collides with the object at the same time; in the previous iteration the finger did not touch and we have only bent the finger slightly further, so the intersection between the finger and object must be small. We assume that the intersection is small enough that it is not evident to the viewer that the finger is intersecting the object. We use a step size (parameter angleStep in Algorithm 4) of 5 degrees.

Algorithm 3 CalculateHoldPoseAndContacts

Input: wristFrame, object
Output: A hand pose and a list of contacts between the hand and object

```
contacts ← empty list
fingerPoses ← empty list
for each finger f do
    fingerPoseF, contactsF ← FindTouchingFingerPose(f, wristFrame, object)
    contacts.Add(contactsF)
    fingerPoses.Add(fingerPoseF)
end for
pose ← (wristFrame, fingerPoses)

return pose, contacts
```

Algorithm 4 FindTouchingFingerPose

Input: finger f, wristFrame, object
Output: A finger pose and a list of contacts between this finger and the object

```
contactsF ← empty list
fingerPoseF ← FingerOpenedPose()
bendRatio ← {0.7, 1, 1.42}

for all ratio ∈ bendRatios do
    for angle = 0; angle < ½π; angle = angle+angleStep do
        testPose ← (angle, fingerPoseF.baseYaw, ratio∗angle)
        touchSegments ← line segments placed across the front of the phalanges for testPose
        for segment ∈ touchSegments that intersect with the object mesh do
            contactsF.add(object.mesh.Intersection(segment))
        end for
        if contactsF is not empty then
            return (testPose, contactsF)
        end if
    end for
end for

return fingerPoseF, contactsF
```

We find contacts from finger poses by intersecting line segments on the surfaces of the fingers with the object's mesh. In Figure 4.2 we show a model of a finger. We have seen that if we have the lengths and width of the phalanges of the fingers, we can find approximate positions of the ends of line segments that lie on the skin of the fingers. We find contacts by intersecting the line segments with the object. The actual surface of the hand is a continuous set of points. It impossible to cover the surface of the hand with enough line segments to find every contact point. We have to compromise between execution speed and the precision of the found contacts. As such, we find only line segments at the front of the finger as in Figure 4.3, using the assumption that most contacts lie there because the hand closes around the object. To use more touch segments means to take more time checking for contacts. Consequence is that visibly more convincing contact points lying on the side of the finger are not registered as contacts.

For a hand pose to be a grasp we need there to be contacts. As we have found the contact while making the pose, we can very easily verify the existence of contacts. This is included in function GraspingContactsExist in Algorithm 5. On the taxonomy of grasp types we saw grasps that all have at least 2 touching fingers. Also, apart from the fixed hook type, all grasps make use of the thumb for contact. We deem the fixed hook type more appropriate for casually holding an object and less for presenting it to someone else. As such we do not feel the fixed hook would be appropriate for use during a handover. Ruling out the fixed hook as a handover grasp gives us the opportunity to require to always have at least a contact on both the thumb and at least one other finger. In Section 5.4 we use this requirement to speed up the Finger Posing step in our algorithm. As we have already found the contacts in the previous step, we can just check if our list of contacts contains at least 2 contacts, with at least one on a finger and at least one on the thumb.

Algorithm 5 GraspingContactsExist

Input: contacts
Output: Do the thumb and fingers both have at least one contact?
1: thumbTouches ← **true** if there is at least 1 contact at the thumb, otherwise **false**
2: fingerTouches ← **true** if there is at least 1 contact at any finger, otherwise **false**
3:
4: **return** thumbTouches **and** fingerTouches

4.5 Grasp Validation

We have explained how we find a pose for the hand in the last sections. Now we describe how we validate the pose. We want our grasps to avoid deep collisions with other meshes, to avoid touching the forbidden surfaces of the object and we want a grasp that could physically immobilize the object.

Collision Validation

In Subsection 4.4 we have described that our method of finding finger poses should avoid the fingers colliding with the object. However, that is only true for the fingers in most cases; if the starting pose of the finger (with, for the fingers, a bend angle of 0) collides with the object, this is initially seen as a contact even though the finger collides deeply with the object. Additionally, the palm can still collide with the object and we have not yet checked that the pose does not intersect the presenting hand. So we have to verify that the resulting hand pose does not collide deeply with the object or presenting hand.

The function CollidesWithMesh in Algorithm 6 checks whether the receiving hand overlaps a mesh (the object or presenting hand). As in the contact finding, we define a set of line segments based on our hand model. This time we find these segments in the inside of the hand model and not on the surface. We make a set of line segments through the palm, underneath the surface of the hand mesh. We manually found the positions of their endpoints local to the wrist frame. At runtime we use the Equation 4.3 to transform the local positions of the palm segments to global coordinates. We also make line segments through the centres of the phalanges. The line segments through the palm and phalanges together are called the *collision segments*. These line segments are shown in Figure 4.4. To check if the hand in a certain pose collides with a mesh we check whether any of the collision segments intersect with (or are contained by) the mesh. If so, we deem the grasp invalid. The way we choose the placement of these line segments allows the hand to penetrate into the mesh up to about half a centimetre. We can afford to do this as the hand is a soft body: it deforms slightly on contact. Allowing the hand to penetrate slightly into the object would look like a touch to the human eye.

Algorithm 6 CollidesWithMesh

Input: pose, mesh
Output: Does the hand intersect the given mesh?

```
fingerSegments ← calculate line segments through the phalanges of the fingers for this pose
palmSegments ← calculate line segments through the palm for this pose

for all segment ∈ fingerSegments ∪ palmSegments do
    if mesh.IntersectsOrContains(segment)) then
        return true
    end if
end for

return false
```

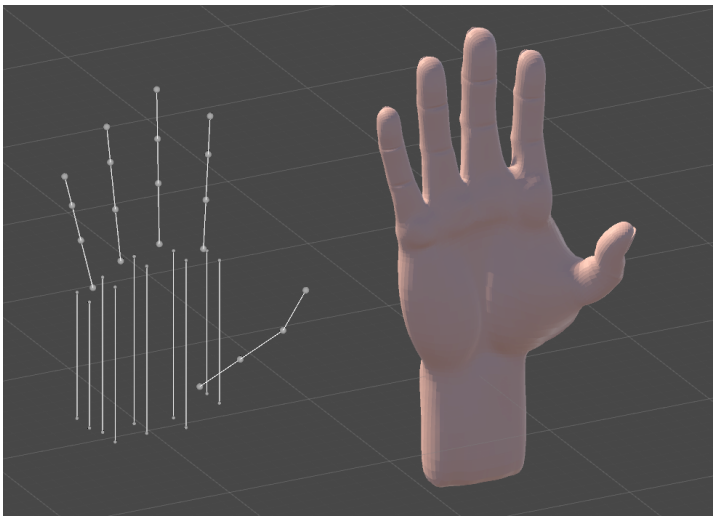


Figure 4.4: On the right we show the mesh of the posed hand. On the left we show the 25 line segments that we use for doing collision validation: 5 segments across both sides of the palm (placed manually) and 1 segment through the centre of each phalanx.

Contact Validation

In function CollidesWithForbiddenSurface in Algorithm 7 we check whether our grasp stays away from the forbidden surfaces. Our objects are triangulated meshes where all triangles have an index. When we find a contact, which is a point on the mesh, we also save the index of the triangle that it lies on. The forbidden surfaces on our objects consist of a list of triangle indices. To check whether the hand hits a forbidden surface, we compare the contacts triangle index to the list of forbidden indices. In Algorithm 7 we compare the triangle indices of all contacts of a grasp with with indices of the forbidden triangles. If there is an overlap between the two groups the grasp touches forbidden surfaces; this means the grasp is invalid. Note that we cannot completely avoid that the hand touches a forbidden surface, because of the impreciseness of our contact finding method. Contacts that are visual to the eye (but invisible to the algorithm) can still touch the forbidden surfaces.

Algorithm 7 CollidesWithForbiddenSurface

Input: contacts, object
Output: Do the contacts hit any forbidden object surface?

```
1: for all contact ∈ contacts do
2:   if object.IsForbiddenSurface(contact.hitSurface) then
3:     return true
4:   end if
5: end for
6:
7: return false
```

Immobility Validation

At this point in our algorithm we have found a wrist placement, finger poses and contact positions. We have also checked that the pose does not collide with the object and presenting hand and that the contacts lie on the allowed surfaces. The last validation step is to check whether our grasp can immobilize the object. To analyse whether or not an object is immobilized, we need to know what the forces applied on the object are. However, we do not know which forces are applied. We only have a set of contacts; We have to determine the forces exerted through these contacts ourselves. In Section 2.6 we have explained how we can model the forces at contacts while taking friction into account. First we will show how to model a set of unit sized wrenches at the contacts. We show how to find a set of force magnitudes. Next, we show how to analyse whether a set of general wrenches immobilizes the object. Finally, we show how we use this in our Immobility Validation algorithm step.

We start from a set of contact points $c \in C$ and the normals of the surfaces they lie on. We want to see the hand as a deformable shape, as in real life the skin of the fingers deforms when touching an object. So we must transform the contact points into contact areas. We chose to change the contact points into circle-shaped contact areas. Contact areas are modelled through a small set of representing contact point that lie on the boundary of the area. Normally, adding friction to this would create a friction cone for each of the representing points. For a circle shaped area we can represent the multiple friction cones with on cut-off friction cone, which is an approximation of the union of the multiple friction cones. We show this in Figure 4.5.

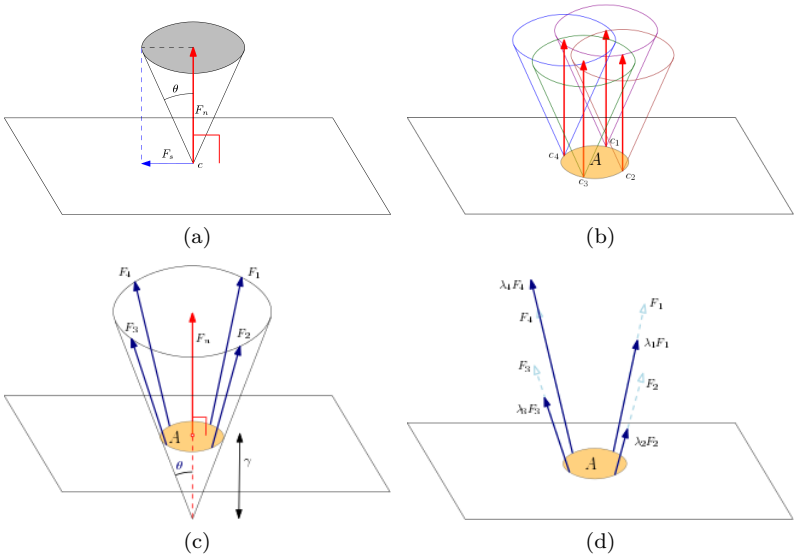


Figure 4.5: (a) The friction cone for a contact point c. F_n is the normal force, F_s is the force of static friction and θ is the half-angle of the cone. Any force starting from c and ending on a point in the grey circle is a valid contact force. (b) A contact area A can be seen as a set of contact points. It is modelled by choosing a small set of contact points on the boundaries of the area. (c) If the contact area is circle shaped, we can find contact forces originating from the boundary of this contact circle by using a cut-off friction cone: we make a friction cone at the centre of the contact circle and move it back along the surface normal by γ. (d) Suppose the forces F found in (c) have a magnitude of 1. We apply a force distribution $L = [\lambda_1, ..\lambda_4]$ by multiplying the scalars λ with the unit forces.

In Figure 4.5a we show the friction cone for one contact point c. In figure 4.5b we show what happens when we expand the contact point to a circle with c as its centre: It is now modelled by multiple friction cones. In Figure 4.5c we visualize the cut-off friction cone. It is created by pulling the friction cone at contact point c back along the surface normal by γ. All vectors on the boundary of the cut-off friction also lie on the boundary of the friction cones in 4.5b. Note that we are using only 4 vectors to model a contact area with friction. While this keeps the amount of wrenches in a grasp low, we do not know how well this approximates the contact areas with friction in reality.

We have explained in Section 2.6 that in a 3D situation, where wrenches are 6-dimensional, we need at least 7 wrenches to find force closure. In Feix et al's taxonomy (see Section 2.4) we saw that all grasps use at least 2 fingers. That means we have at least 2 contacts for each grasp. If we need at least 7 wrenches and have at least 2 contacts, then we need at least 4 wrenches per contact. So modelling friction with 4 wrenches per contact is sufficient to allow for two-contact grasps. We model our contact areas as cut-off friction pyramids. Then one contact area c is characterized as wrench set $W_c = [W_{c,1}, W_{c,2}, W_{c,3}, W_{c,4}]$, where all $|W_{c,i}| = 1$. This process is written down as pseudo-code in Algorithm 8. In function FourUnitWrenchesFromContact we build 4 wrenches from a contact point and surface normal. In lines 2-5 we move the friction cone back along the surface normal. In lines 6 and 7 we make the 4 vectors on the boundary of the friction cone. (Together these 4 vectors make up the friction pyramid.) We use the friction coefficient of $\mu_s = 0.34$ between leather and metal for all our objects, as it comes as close to our contacts lying between skin and metal or plastic as we could find. The half-angle of the friction cones is calculated as $\theta = cos^{-1}(\mu_s)$. In lines 9-14 we build the unit sized wrenches that make up the cut-off friction pyramid.

Algorithm 8 FourUnitWrenchesFromContact

Input: contact
Output: Set of 4 wrenches with unit sized forces, anchored around the contact point.

```
wrenches ← empty list
hit ← contact.Position
normal ← contact.Surface.Normal
γ ← a small value
coneOrigin ← hit - γ*normal
coneHalfAngle ← halfangle of the friction cone
forceDirections ← 4 vectors on the boundary of the friction cone, uniformly spread

for all direction ∈ forceDirections do
    anchor ← coneOrigin + γ∗direction
    force ← direction.normalized
    wrench ← (force, anchor × force)
    wrenches.Add(wrench)
end for
return wrenches
```

A grasp has a set of contact points C. For each contact we find a set W_c of 4 unit wrenches with function FourUnitWrenchesFromContact. For each contact $c \in C$ we find a set of force magnitudes $L_c = [\lambda_{c,1}..\lambda_{c,4}]$. When the sum of all wrenches applied on the object is 0 we know that the object is at rest. All wrenches on our object are the wrenches at the contacts and also the gravity wrench. So in order for the object to be at rest in our grasp, equation 4.4 must hold.

$$(\sum_{c \in C} \sum_{i=1}^{4} \lambda_{c,i} W_{c,i}) + W_g = 0 \tag{4.4}$$

where W_g is the gravity wrench on the object. Interesting is that this formulation also implies force closure. A set of wrenches W satisfies the force closure property when the origin lies inside of the convex hull of W. Suppose we have a set of points P such that its convex hull H where $O \notin H$. We sum the points in P to make point $S = \sum_{p \in P} p$. Then we make a plane l on which line xS with scalar x lies, that divides the wrench space into two sides. Because of how S is defined, we know that either all point P lie on l, or we have at least one point on each side of l. Suppose we would like to make new points set $P' = P \cap \{q\}$ and new convex hull H' of P'. We can reason that for any $q \in \{xS \mid x < 0\}$ we must have $O \in H'$. This is because we know that per definition O lies on l as xS goes through the origin. We also know that we have at least one point in each side of l in P. Per definition, the point $-S$ lies on plane l and thus for $q = -S$ the convex hull H' includes the origin. In Figure 4.6 we visualize this proof with a 2D example. This proof should also work in cases with a higher number of dimensions.

Suppose we have a grasp with contact wrenches W where the object is at rest. The full set of wrenches on the object is $W \cap \{W_g\}$. As the object is at rest we have $[\sum_{w \in W} w] + W_g = 0$. From this follows that the convex hull of $W \cap \{W_g\}$ must the origin and thus force closure exists. Either the convex hull of W contains the origin, or we relate back the theory we visualize in Figure 4.6. As $-W_g = \sum_{w \in W} w$, it follows that adding to W any wrench on line $k = -xW_g$ for $x < 0$ would cause new the convex hull to include the origin. As W_g lies on k per definition, we know that the convex hull of $W \cap \{W_g\}$ must contain the origin. Thus, if the object is at rest with a grasp with contact wrenches W and gravity wrench W_g, the force closure attribute holds for this grasp.

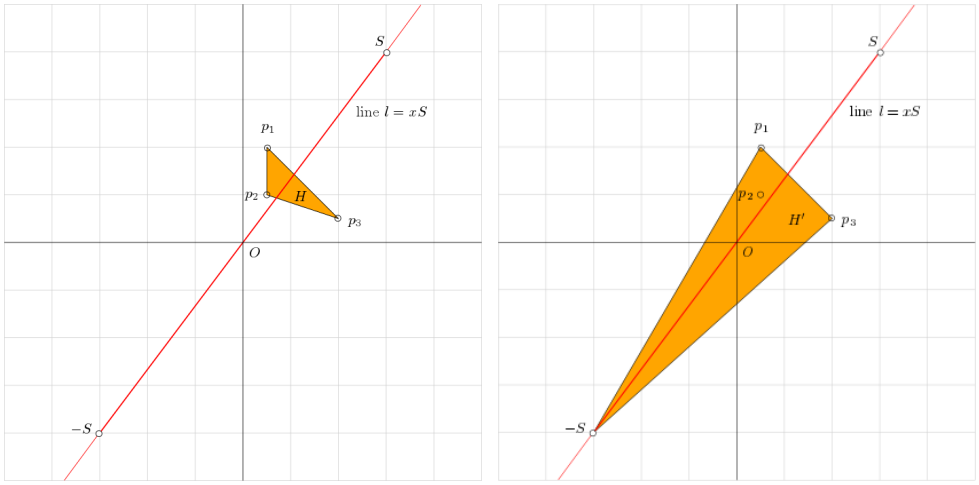


Figure 4.6: A 2D visualisation of our observation that the origin must lie inside the convex hull of a set of points after we add the negative sum of the point set to it. (a) We show a point set $P = [p_1, p_2, p_3]$ with convex hull H. Point S is the sum of all points in P. We define plane l on which line xS with scalar x lies (note that in 2D, 'plane' l is equal to xS). (b) We make the convex hull H' of point set $P' = P \cap \{-S\}$. Note that convex hull H' contains the origin.

Our goal for finding force magnitudes must thus be to have the resulting wrenches sum to negative gravity. We have explained in Section 4.1 that we cannot use optimization within our method. So we use sampling to find a set of force magnitudes. It will be very hard to find a set of force magnitudes such that we perfectly oppose gravity. However, as our grasp are to be used in a virtual environment, we do not require precise force closure of the object. Instead, we approximate it. We desire a set of contact wrenches to be sum to a wrench is equal to the negative gravity wrench. In other words, the sum of contact wrenches must *oppose* the gravity wrench. To perfectly oppose gravity the angle between the summed contact wrench and the gravity wrench should thus be 180 degrees. We revise Equation 4.4 to allow a little more room in the following equation:

$$Angle(\sum_{c \in C} \sum_{i=1}^{4} \lambda_{c,i} W_{c,i}, -W_g) < \alpha \tag{4.5}$$

α is an acceptance threshold. We use $\alpha = 5$ degrees in our experiments. After we find a force distribution that has a small enough angle with respect to the perfect gravity opposition wrench (namely $-W_g$), we scale the force distribution so that its summed magnitude matches $|W_g|$. If Equation 4.5 holds for a given grasp and a choice of force magnitudes, then we are not guaranteed that the grasp has the attributes that the object is perfectly at rest or that force closure applies. However, as we allow only a small deviation of 5 degrees, we are assured that the grasp is at the very least very close to both attributes. The distance between $-W_g$ and $\sum_{w \in W} w$ is small, which bring the object close to being at rest. The distance between O and the closest face of the convex hull is either 0 or very small, which brings us close to force closure.

Algorithm 9 IsImmobilizingGrasp

Input: contacts, objects
Output: Can a set of forces be found for these contacts such that it (approximately) immobilizes the object?

```
1:  unitWrenches ← empty set
2:  for contact ∈ contacts do
3:      unitWrenches.Add(FourWrenchesFromContact(contact))
4:  end for
5:
6:  wrenchSets ← empty list
7:  for i in range(0,100) do
8:      forceMagnitudes ← ForceDistributionsOverContacts(contacts)
9:      wrenchSets.Add(forceMagnitudes*unitWrenches)
10: end for
11:
12: bestWrenches ← ArgMin_{W∈wrenchSets} (Angle(Sum(W), -object.gravityWrench))
13: bestWrenches ← ScaleToGravitySize(bestWrenches)
14:
15: WrenchSum ← Sum(bestWrenches)
16: angle ← Angle(WrenchSum, -object.gravityWrench)
17: isImmobilizing ← angle < α
18: return isImmobilizing
```

The function IsImmobilizingGrasp in Algorithm 9 outlines the Immobility Validation step through pseudo-code. We first find 4 unit wrenches at each contact in lines 1-4. We create 100 different distributions of force magnitudes for the unit wrenches in lines 6-10. In lines 12 we find the set of force magnitudes for which the unit wrenches best oppose the gravity wrench. In line 13 we scale this force distribution so that its summed magnitudes is equal to the magnitudes of the gravity wrench.

In lines 15-17 we check if the force distribution (which was the best of a set) actually opposes the direction of gravity such that Equation 4.5 holds. We calculate the angle between negative gravity and the summed wrench again. We allow an angle threshold of 5 degrees, for bigger angles we deem the grasp invalid. We return in line 18 whether the best force distribution was good enough to immobilize the object.

So we have explained how we find the unit wrenches, and how we find out whether a set of force magnitudes combined with the unit wrenches can oppose gravity. We have yet to explain how we find a set of force magnitudes. In Section 2.5 we explained how a comfortable distribution of force over the fingers means approximately 25% for the thumb, 25% for the index finger, 25% for the middle finger, 12.5% for the ring finger and 12.5% for the little finger. Because we want our grasp to be natural, we have the constraints that the forces should be applicable by human fingers. We saw in Section 2.3 that each contact is able to apply about 27N (or carry about $3kg$). Since our objects are specified to be small and light, the probability that we would need to use *more* force than this is very small. As such we do not actually place a limit on the maximum force. When we decide if a wrench set immobilizes the object based on its *angle* against the gravity wrench, we do not bother with making realistic force magnitudes in this step. We scale them back to realistic magnitudes at the end of Algorithm 9.

In Algorithm 10 we find a set of force magnitudes L_c for all contacts $c \in C$. First, in lines 1 and 2 we find the magnitude of force we want to apply per finger; this is the comfortable force distribution (as explained above) with noise added. In lines 5-11 we divide the force magnitudes per finger over the different contacts. If a finger has n contacts, each of these contacts gets $\frac{1}{n}$ of

the force magnitude that was reserved for that finger. Then we find four scalars in the range $[0, 1]$ that sum to 1 together. One contact is linked to a set of 4 unit sized wrenches: we divide the reserved force magnitude for this contact over the 4 unit wrenches with these 4 scalars. In line 12 we return the found distribution of force magnitudes.

Algorithm 10 ForceDistributionsOverContacts(contacts)

Input: contacts
Output: A set of wrenches applied at the contacts

```
1: perfectDistribution ← [25,25,25,12.5,12.5]
2: noisyDistribution ← apply random noise to perfectDistribution
3:
4: forceMagnitudes ← empty set
5: for contact ∈ contacts do
6:     n ← NumberOfContactsOnFinger(contact.finger, contacts)
7:     fingerTotalForceSize ← noiseDistribution[contact.finger]
8:     contactForceSize ← fingerForceSize /n
9:     contactDistribution ← four values [0, 1] that sum to 1 together
10:    forceMagnitudes.Add(contactForceSize∗contactDistribution)
11: end for
12: return forceMagnitudes
```

4.6 Evaluation Measure

In the previous sections we have explained our complete method for synthesizing grasps. This contains the generation and validation parts of a sampling method. In this section we propose an evaluation method. Evaluation of the samples is needed to rank the sample grasps so that the best among them can be used as output. There is an accepted evaluation measure: the grasp quality, originally defined by Ferrari and Canny [11]. It measures how robust a grasp is. This measure approaches the term 'realistic grasp' from a physics point of view: how much force would minimally be needed to push the object out of the grasp? It is a useful measure as it implies the existence of force closure for the grasp. However, it relies on the computation of a 6D convex hull, which is a complex calculation. In the grasp synthesis methods that use Ferrari and Canny's grasp quality to evaluate grasp samples, about half the execution time is taken up by evaluating the grasps.

In the real world, having a grasp that can withstand large external forces while staying immobilized is useful. Real-life robots must be careful with the object they are holding. Our problem exists in a virtual world, though, where we can ignore laws of physics to a certain extend; A considerable violating of the laws of physics will be noticeable and thus unrealistic. However, if we want to make a grasp that *looks* realistic, we do not need to go as far as calculating the smallest destabilizing force; we just need the approximated force closure property, which we have already covered in our formulation of a valid grasp. So we will not use Ferrari and Canny's grasp quality as the evaluation function.

We are considering to use a 'comfort' measure for it. For each grasp we have found a force distribution over the fingers such that it opposes gravity. There must be distributions of forces that are preferable to others; for example those in which the forces are distributed very unevenly over the fingers. Another consideration to make is the inherent strength of the fingers. In Section 2.5 we have seen a force distribution that we think is comfortable. It combines the ring and little finger into one virtual finger and then distribution the force evenly across these '4' fingers. As explained in Algorithm 10, we find force distributions that lie close to this distribution. Better grasps should lie closer to the perfect comfortable distribution.

So our evaluation measure should determine the distance between a given force distribution and the perfect comfortable one. We must also include in the formulation that not all finger have contacts and that some have multiple contacts. Suppose that we have set $b = [b_1, .., b_5]$, where b_i is 1 when finger i has a contact and 0 when it does not. Suppose that $D = [0.25, 0.25, 0.25, 0.125, 0.125]$ is the perfect force distribution vector. We define $D' = \frac{1}{Z} bD$, where $Z = \sum_{i=1}^{5} b_i D_i$, which means that the sum of values in D' is 1. Now we define f_i as the sum of the force magnitudes applied at finger i in the actual distribution. Then the evaluation measure E is:

$$E = \sum_{i=1}^{5} (D'_i - f_i)^2 \tag{4.6}$$

Chapter 5

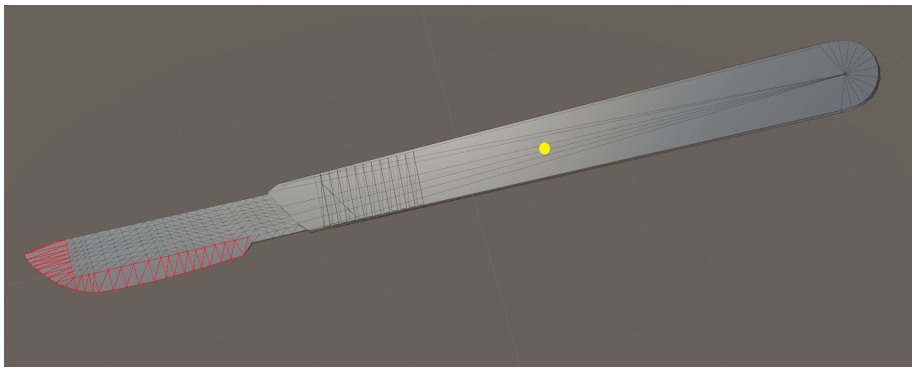
Experiments

In Chapter 4 we have presented our method of grasp synthesis during handovers. We have explained how we generate grasps and how we examine the validity of the grasps. Sampling methods tend to be slow due to the large search space and constraints on valid grasp poses. Our goal is to lower the execution time of valid grasp search for our grasp synthesis method. This goal can achieved by decreasing the generation time of samples and by increasing the probability of finding successful samples. We established the parameters that are involved in our grasp sampling method. In this chapter we conduct experiments where we constrain the parameters in order to decrease the number of attempts we need before finding a valid grasp. Next to that we discuss changes to our algorithm that lead to a decrease in execution time. We discuss these improvements to our method in the order of the steps in our algorithm.

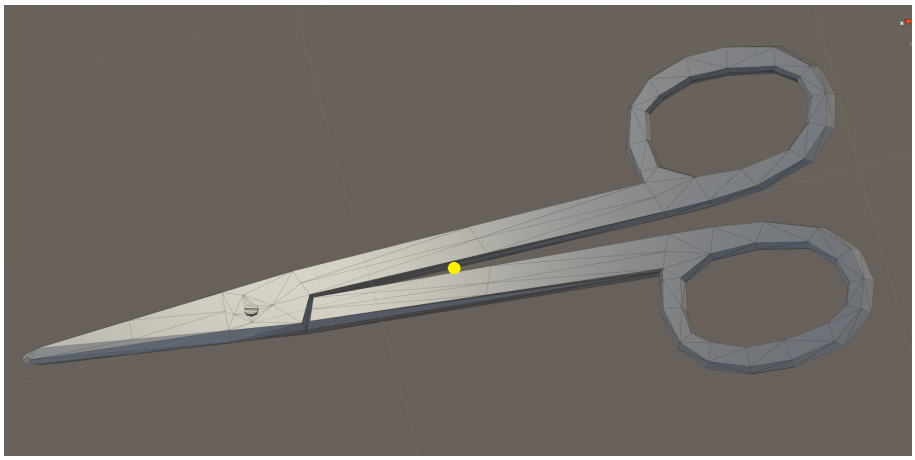
In Section 5.1 we explain the general set-up of our experiments. We do this experiment several times and compare the results across different versions of our algorithm. In Section 5.2 we show the results of the naive version of our algorithm. In Sections 5.3 to 5.6 we discuss the modifications to our algorithm with the goal of improving its execution speed.

5.1 Set-up

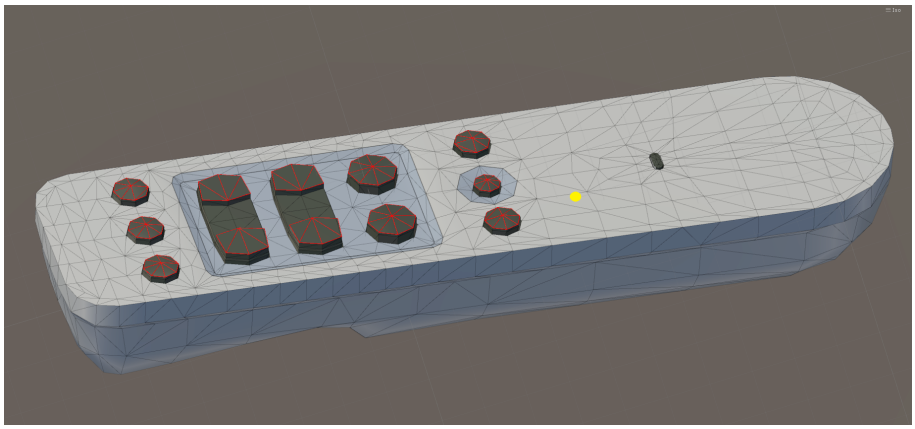
Let us first describe the general set-up we use to investigate the output of our method. We have implemented our method in Unity [35]. In Chapter 1 we explained that our method is to be used in the context of a surgery room. The objects we want to hand over in this room are surgery instruments and possibly remote controls. So we test our method on such object types. We selected three objects: a pair of scissors, a scalpel and a remote control. Figure 5.1 shows the triangle meshes we used to represent the surface of those objects. We have also stated that some parts of the objects should not be touched. In Figure 5.1 the forbidden surfaces are shown as red outlined triangles. Our method validates the grasps by, among other things, checking if a force distribution can be found to counter the gravity on the object. We assume the gravity force is applied at the centre of mass (CoM) of the object. The size of the gravity force depends on the mass of the object. Thus we need to define a mass and CoM position for the objects. We base these definitions on the masses and CoM's of real world scissors, remotes and scalpels. We weighed these objects to find a mass. For finding the CoM we balanced the objects and then considered their symmetry axes to guess a reasonable position for the CoM. This object information is shown in Table 5.1.



(a)



(b)



(c)

Figure 5.1: The triangulated meshes of the chosen objects. Their forbidden surfaces are shown as triangles with red outlines. A yellow dot is placed on the position of the CoM. (a) shows the mesh of the scalpel. Its forbidden surfaces lie on the edge of the blade, as the blade needs to be kept sterile and touching it can be painful. (b) shows the mesh of the scissors. We did not give the scissors forbidden surfaces as the sharp edges are already protected by the scissors being shut. (c) shows the mesh of the remote. Forbidden surfaces include the buttons, which should not be accidentally pushed during a handover.

Object	Mass (g)	Description of CoM	Description of forbidden surface	Triangle count of mesh
Scissors	50	Centre of the object	None	572
Scalpel	20	Centre of the heft	Edge of the blade	1453
Remote	130	Off-centre, towards the end of the remote where the batteries lie	Buttons	2203

Table 5.1: Properties of the objects used in the experiments.

In our experiments we will go through the steps of our method and investigate per step where we can constrain ranges of parameters. We ran the experiments on a laptop with Windows 10 Home, 64-bit installed. The laptop has an Intel Core i7-5700HQ CPU 2.70GHz, 8Gb RAM and an NVidia GeForce GTX 960M video card. As our grasp synthesis is to be used in the context of human-virtual character handovers, we test our synthesis method on several different 'holdouts'. We use the term 'holdout' for a configuration of an object and hand, in which the hand is holding the object and presenting it towards a receiver. We used our posed hand mesh as a stand-in for the presenting hand.

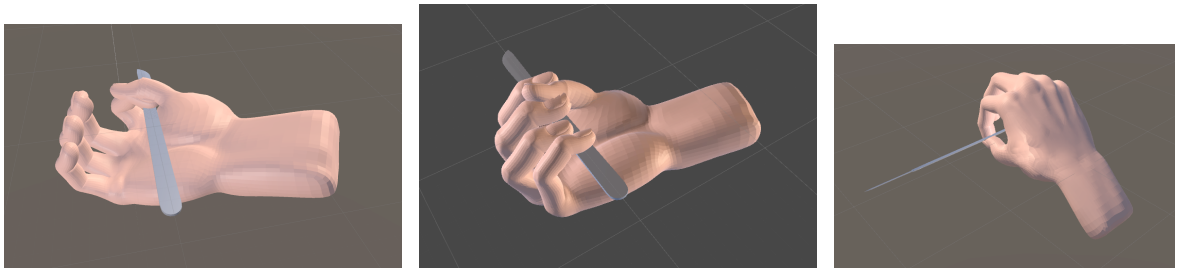
We made five holdouts per object. Out of the five holdouts, two were configured manually after real world examples (assuring their naturalness) and three were made with our grasp synthesis method. We call the manually created holdouts 'Holds' while we call the synthesized holdouts 'GenHolds'. We aimed to cover the diversity in holdouts by choosing holdouts with differing grasp types (see Section 2.4). We show the holdouts for the scalpel, remote and scissors in Figures 5.2, 5.3 and 5.4 respectively. In the figures' captions we state their grasp types and argue why we classified them as such. Figure 2.5 shows the original examples of these grasp types in Feix' et al.-s [10] grasp taxonomy.

In the remainder of this chapter we will make a number of *grasp sets* to show our findings. Every finding that improves our algorithm will be used for generating the next grasp set. As such we improve our method stepwise.

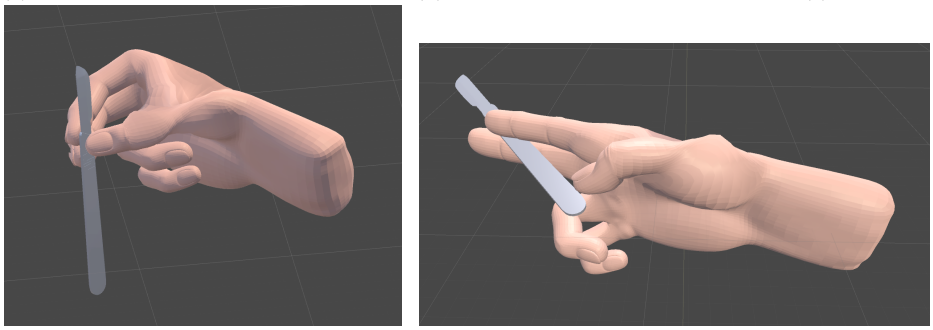
A grasp set is made by synthesizing a set of grasps for each of the fifteen holdouts. We run our algorithm until we find 25 valid grasps per holdout. The valid grasps are saved. We do not save all failed grasps attempts; there were so many that it became a problem to store them all. Instead, we only store the first 100 failed grasp attempts for each holdout. We use this relatively small set of failed grasps to represent the huge set we were unable to store.

When saving a grasp we administer its pose, its contacts and its force distribution. (For failed grasps the list of contacts and/or forces is empty.) Additionally, we administer the time we spent for each step and the measured properties. We first investigate which properties valid grasps have that invalid grasps lack. If we know the specific properties that only valid grasps have we can stop making a grasp early. This way we do not waste time executing the later algorithm steps for grasps that lack the required properties. For valid grasps we also administer the time and number of attempts we need before we find the valid grasps.

We thus generate and save 25 valid and 100 invalid grasps for all 15 holdouts: the full grasp set contains 1875 grasps. The 1500 invalid grasps will be referred to as the *negative* grasp set and the 375 valid grasps will be called the positive grasps set. We use the negative grasps to see which types of failures happen in the many attempts that our algorithm takes before a valid grasp

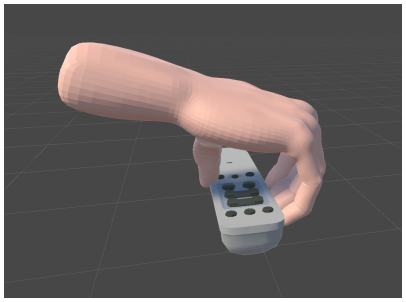


(a) Scalpel GenHold1 (b) Scalpel GenHold2 (c) Scalpel GenHold3



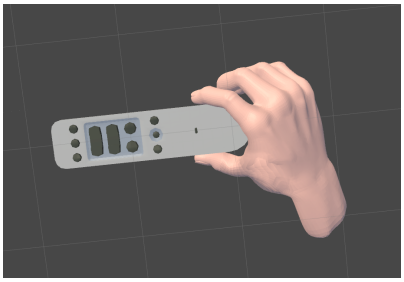
(d) Scalpel Hold1 (e) Scalpel Hold2

Figure 5.2: Five holdout poses presenting the Scalpel to the receiver. The GenHolds were synthesized by our algorithm, the Holds were made manually and imitate real world holds on the object. (a) GenHold1, classified as a Ring, as the thumb and index finger bend around the object from opposite directions. Contacts lie at the bottom and tip of the index finger and the intermediate phalanx of the thumb. (b) GenHold2, classified as a Small Diameter as the fingers push the object into the palm, while the thumb opposes the fingers. (c) GenHold3, classified as a Palmar Pinch as the contacts lie on the pads on the distal phalanges of the thumb and index finger. (d) Hold1, classified as a Prismatic 2 Finger as the contacts lie on the tips of the thumb and two fingers. (e) Hold2, classified as an Adduction Grip because the contacts lie on the sides of the fingers.

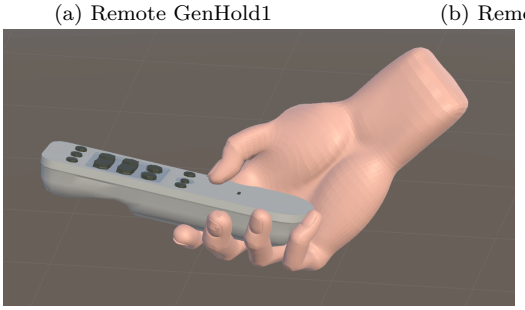


(a) Remote GenHold1

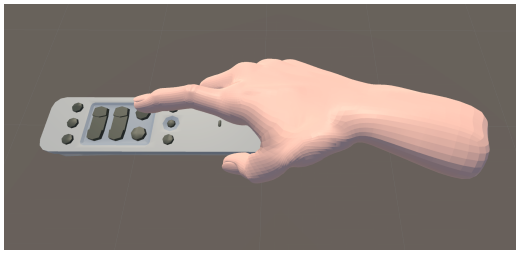
(b) Remote GenHold2



(c) Remote GenHold3

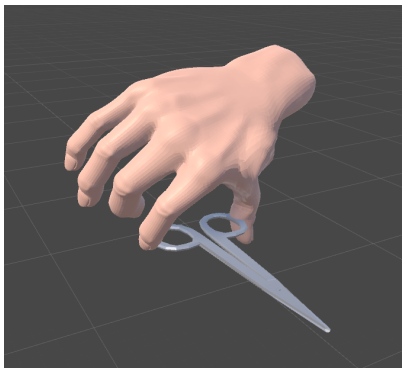


(d) Remote Hold1

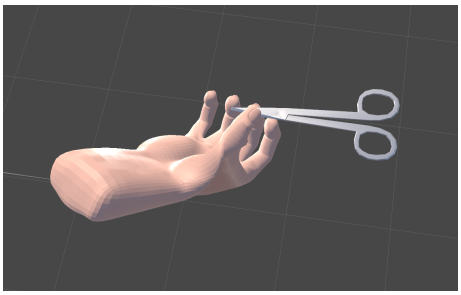


(e) Remote Hold2

Figure 5.3: Five holdout poses presenting the remote to the receiver. The GenHolds were synthesized by our algorithm, the Holds were made manually and imitate real world holds on the object.(a) GenHold1, classified as a Prismatic 4 Finger as the contacts lie clearly on the finger tips and the fingers stand opposite of the thumb. (b) GenHold2, classified as Medium Wrap as it looks like the finger push the object into the palm and the thumb opposes the fingers. (c) GenHold3, classified as a Ring as the thumb and finger are shaped against the rounded end of the reciever. (d) Hold1, classified as a Light Tool as all four fingers are used and the adducted thumb has a contact on its fingertip. (e) Hold2, classified as a Index Finger Extension as the index is stretched and the thumb is adducted while the remaining fingers push the object against the palm.

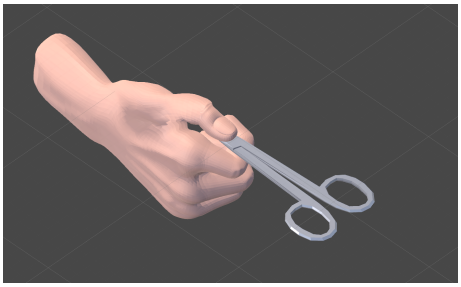


(a) Scissors GenHold1

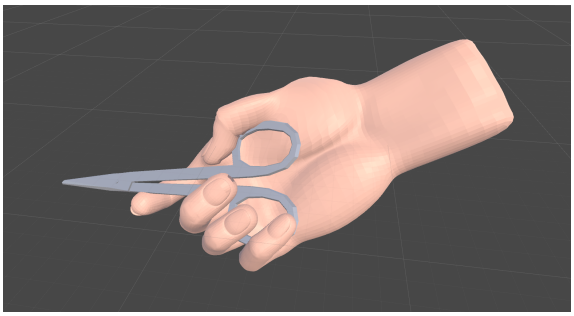


(b) Scissors GenHold2

(c) Scissors GenHold3



(d) Scissors Hold1



(e) Scissors Hold2

Figure 5.4: Five holdout poses presenting the Scissors to the receiver. The GenHolds were synthesized by our algorithm, the Holds were made manually and imitate real world holds on the object.(a) GenHold1, classified as a Palmar Pinch as the contacts lie on the distal pads (pads of the distal phalanges) of the thumb and index finger. (b) GenHold2, classified as Prismatic 2 Finger as the contacts lie on the distal pads of an abducted thumb, the index and middle fingers. (c) GenHold3, classified as Writing Tripod as the contacts lie on the distal pad of the index finger, the side of the middle finger. (d) Hold1, classified as a Lateral as an adducted thumb pushes the object against the side of the index finger. (e) Hold2, classified as an Adducted Thumb as the full inside of the adducted thumb touches the object while the fingers push the object into the palm.

is found. In our method we validate a grasp by checking that it conforms to a set of conditions. There are thus multiple different reasons for which a grasp can be invalid. We make a distinction between the different invalid grasps by referring to them with their *output type*, using the names in Table 5.2. For example, when we talk about a 'NoContact' grasp, we mean a grasp that is invalid because it does not have any contacts. Note that 'Valid' is also an output type.

Name of Output Type	Description
NoContacts	We do not find any contacts with any digit
NoThumb	We do not find any contacts on the thumb
NoThumbEarlystop	This groups the NoContact and NoThumb grasps together to improve execution speed: The thumb does not touch the object (which triggers a failure condition), so we do not have to try posing the fingers either.
NoFingers	We find no contacts on any of the finger
PalmCollision	The palm collides into the presenting hand or object
FingerCollision	A finger collides into the presenting hand or object
UnwantedContact	A contact lies on a forbidden surface
NotImmobilizing	We did not find a force distribution that could oppose gravity.
Valid	This grasp attempt did *not* trigger any failure conditions

Table 5.2: Names and description of the output types that our algorithm outputs when the grasp attempts triggers a failure condition.

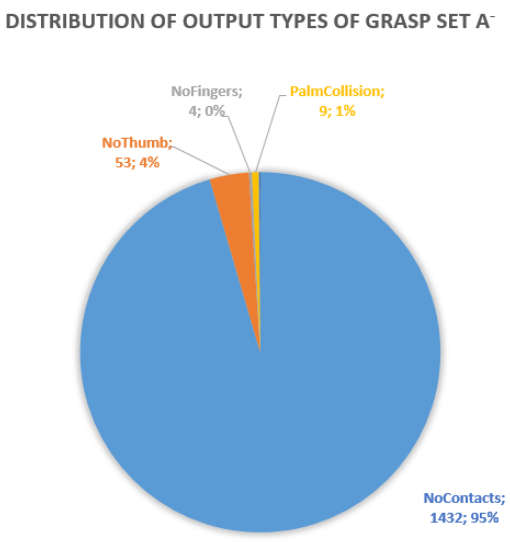
5.2 Naive synthesis

We start our experiments by making a set of grasps with the naive version of our algorithm. For each holdout that we showed in the previous section, we run our grasp synthesis method until we find 25 valid grasps. This is the positive grasp set A^+. Next we run the method again until we find 100 invalid grasps for each holdout; these 1500 invalid grasps are negative grasp set A^-. The grasp set A is made from combining A^- and A^+. A part of the results is shown in Figures 5.5 to 5.7.

In Figure 5.5a we show for each holdout the amount of attempts and time required to generate one valid grasp. We see that there are big differences between objects, but also between the individual holdouts. However, we can observe several trends. We observe that the scissors are fairly easy to validly grasp, whereas the scalpel and especially the remote are hard to find good grasps for. We took a closer look at the data to see if the number of triangles of the mesh differs a lot. This could affect the time used in the Finger Posing and Collision Validation steps. It turns out not to be the case: there are no significant differences in execution times for these steps between the different objects. So we reason that the remote has a lot more volume than both the scissors and scalpel and therefore more easily leads to collisions. We found support for this reasoning by looking back in grasp set A^- to see at which objects the Palm- and FingerCollision grasps happen; from the 10 collision grasps we found in grasp set A^-, 9 happened on the remote, spread about evenly across its five holdouts. We check collisions by intersecting a set of line segments with the posed hand. For objects with more volume there is a higher probability that (one of) these segment will intersect with the object. The remote is has the most volume, which makes it more likely that a collision happens. The scalpel and scissors are fairly thin objects and thus the probability that

Object/ Holdout	Average Number of Attempts	Average Total Time (ms)
Remote		
GenHold1	515,593	147,335
GenHold2	2,214,064	642,242
GenHold3	139,860	40,525
Hold1	1,971,821	661,714
Hold2	134,359	48,915
Total Remote	**995,139**	**308,146**
Scalpel		
GenHold1	380,771	95,681
GenHold2	161,882	41,362
GenHold3	5,529	1,415
Hold1	1,206,705	340,163
Hold2	16,541	6,075
Total Scalpel	**354,286**	**96,939**
Scissors		
GenHold1	10,649	3,022
GenHold2	3,890	1,095
GenHold3	20,506	5,718
Hold1	10,453	3,145
Hold2	51,689	18,660
Total Scissors	**19,437**	**6,328**
End Totaal	**456,287**	**137,138**

(a)



(b)

Figure 5.5: (a) Average number of attempts and average total time needed to make grasp set A^+ (b) The distribution of output types for the 1500 invalid grasps in grasp set A^-. The labels read 'Output type; number of occurrences; occurring percentage'.

the collision segments intersect with them is fairly low. It could be that the collision segments miss the object due to its thin shape when there is in fact a collision. It is possible that this leads to false positives in the Collision Validation step. We looked through the grasps in A^+ to see if the scalpel and/or scissors show more grasps that visually collide with the object than the remote. This is not the case. So we adjust our reasoning slightly: as the objects are thin, it is unlikely that we find a wrist placement for which both the palm collides with the object *and* the fingers touch. We see less Palm- and FingerCollision grasps for the thin objects because they are already falsified by a lack of sufficient contacts.

Note that the 'averages' in Figure 5.5a are not very precise, with 25 valid grasps per holdout. Looking closer at the data of grasp set A^+ we see that there are big differences between holdouts in the required number of attempts. Apparently, our naive algorithm needs between the 1 and $8,5$ *million* attempts before finding a valid grasp. The very high or low outliers that occasionally happen have a real influence on the averages.

The overall probability of success (100%/average number of attempts) is $100\%/456287 = 0.00022\%$. Needless to say, there is only a very small chance for a grasp attempts to turn into a valid grasp. In later sections we will use the average number of attempts and the average total time as a benchmark on whether or not parameter constraints speed up our algorithm.

In Figure 5.5b, we show the distribution of output types across the invalid grasps generated with the naive algorithm. We observe that there are very few grasp attempts of type NoFingers, PalmCollision, FingerCollision and NotImmobilizing and none of the type UnwantedContact. 95% of the invalid grasps are NoContacts grasps. This means that in barely any of the grasps the hand is close enough to the object to touch it. This is no surprise, as we find positions in a placement volume such that a position can lie at most 22cm away from the object. For the positions with relatively high distances it will be difficult to find both a thumb and finger contact as the fingers are too far away from the object. Secondly, the naive algorithm randomly selects an orientation,

which means that the hand can also point away from the object. The result is that most of the time we output a NoContact grasp, for which the validation steps are not executed. As such, we suspect that the average duration of time spent in the validation steps does not play a major role in the overall running time for finding one valid grasp.

In Figure 5.6 we denote the time spent in each step, per output type. We leave out the output types that barely occur as their low number means the calculated averages are unrepresentative of them. We see that the Wrist Placement takes up only a very small part of the total time, even when it had a 5% contribution to the total time in Figure 5.7. This is because Wrist Placement is executed at every grasp attempt. The Finger Posing step is the most time consuming of our algorithm steps, as it is also executed for every grasp attempts. That explains why it takes up 90% of the total time as we saw in Figure 5.7. For grasp attempts that end as NoContact grasp the execution time of the Finger Posing step is relatively short. This is because we stop posing a finger early when its base joint is further away from the object than its finger length. In many of the NoContact grasps the fingers were too far away from the object to reach it. We assume this is the reason that the Finger Posing step takes a fairly short time for this output type. In Section 5.4 we will reduce the time taken by the Finger Posing step. Lastly, we observe that a serious amount of time is spent on the Immobility Validation step. In the Immobility Validation step we try 100 combinations of forces exerted through the contacts to counter gravity. It is expected that all these wrench related calculations take a lot of time. In Section 5.6 we propose a modification to the algorithm to speed up this step.

In Figure 5.7 we added up the time spent per algorithm step off the grasps in grasp set A^-. 90% of the time spent in our algorithm is used during the Finger Posing step. The Wrist Placement step takes up 5% of the spent time, with which it takes up a significantly higher percentage of the total time than any of the validation steps. This is simply because the Wrist Placement step is executed at every grasp attempt, where the validation steps only get executed every so often.

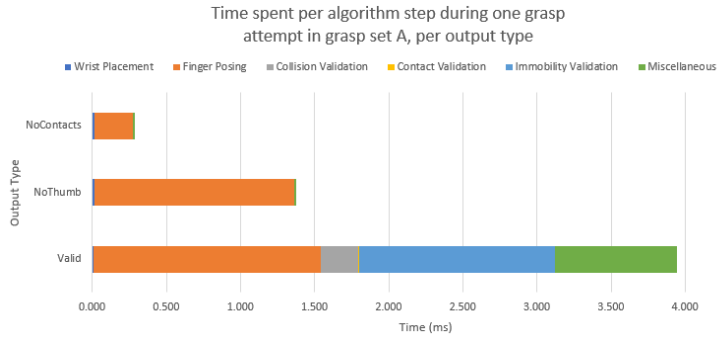


Figure 5.6: The average distribution of time spent per algorithm step for the grasp attempts in the grasp set A, per output type. 'Miscellaneous' includes mostly administration of properties of grasp attempts used only in context of the experiment. Note that grasp set A only includes a few grasp attempts of the types NoFingers, PalmCollision, FingerCollision and NotImmobilizing, which makes their averages unreliable. As such these output types were left out of this graph.(The Contact Validation step takes about 0.001ms when executed; as such it is barely visible in this graph.)

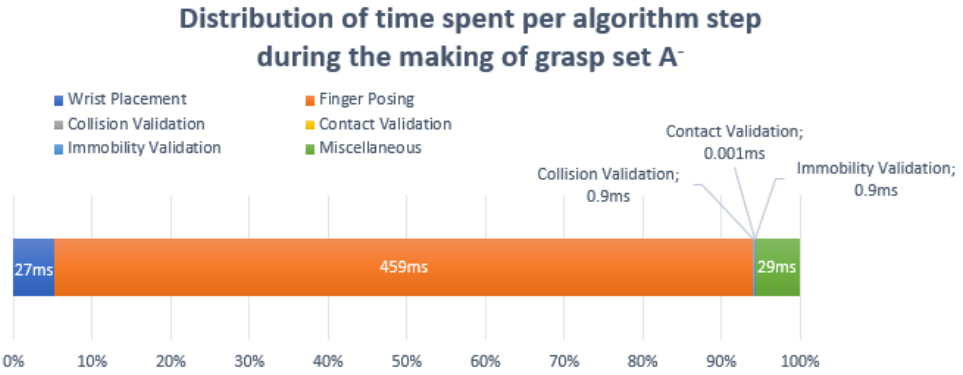


Figure 5.7: The time we spent in total during the making of the 1500 invalid grasps in grasp set A^-.

5.3 Wrist Placement

The first step of our grasp sampling method is the Wrist Placement step. This step has a major influence on the resulting output: it places the palm (which can cause collisions) and as such determines the reach of the fingers (which limits the space for which we can search for contacts). In this section we will define a set of measures that describe properties of wrist placements. We will use these measures to limit our wrist placement search, in hope of improving the output in terms of both runtime and success probability. First we define measures which relate wrist placement to the object. These measures can also be used for making grasps on free-floating objects. Later we define an additional measure which relates the wrist placement to the presenting hand.

Relative to the object

In Section 5.2 we saw that many grasp attempts did not find any contacts to begin with. This was caused by the naive way of placing the wrist: even when the hand was placed close enough to the object, we saw that most orientations point the hand away from the object, leaving it unable to touch it. So, it is our aim to identify a range of wrist placements (position and orientation) that are more likely to produce valid grasps. Generating a wrist placement and checking if a placement has certain properties is a matter of a few vector calculations. As such it is a relatively cheap operation. We make wrist placements by generating them within the space we defined in Section 4.3, and prune out placements that look unpromising. This way, we do not *directly* constrain the range of the search space for wrist placements. Instead, we constrain the ranges of a set of measures which describe the wrist placement. The 'unpromising' wrist placements have measure values that lie outside the allowed ranges. The boundaries of the allowed ranges are added as placement parameters to our algorithm.

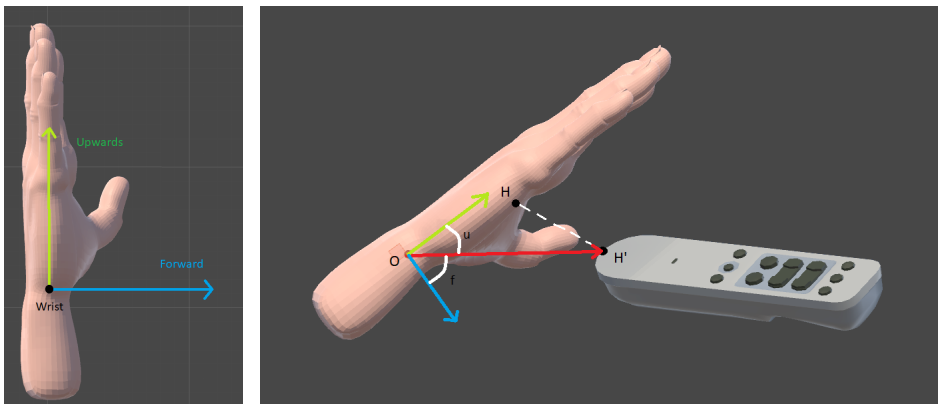
In the algorithm, we find a position first and then rotate the wrist, which places the hand relative to the object. Because of this our measures must describe a relation between a wrist placement and the object. For object related wrist placements we consider the following measures:

Wrist Distance: The distance from the wrist position O to the position O' on the object closest to O. We sample for wrist positions between 0 and 22cm away from the object.

Palm Distance: The distance from the palm position H to the position H' on the object closest to H. We do not sample this value directly: its value depends on the placement and rotation of the wrist. As such it lies between 0 and about 30 cm.

Upwards Wrist Angle: The angle between the upward vector of the wrist and the vector $H' - O$. It is a 3D angle and as such has values between 0 and 180 degrees.

Forward Wrist Angle: The angle between the forward vector of the wrist and the vector $H' - O$. It is a 3D angle and as such has values between 0 and 180 degrees.



(a) (b)

Figure 5.8: (a) The upward vector (green) and forward vector (blue) in the hand (b) We show how we measure the upward and forward wrist angles in a grasp: we find wrist position O and the position H' on the object that is closest the palm position H. Then we find the angles between the upward (green) and forward (blue) vectors in the hand with vector $H' - O$ (shown in red). The upward and forward angles are the angles u and f in the figure respectively.

We visualize the Upwards and Forwards Wrist Angles in Figure 5.8. The upward and forward vectors of the wrist are shown in Figure 5.8a. In Figure 5.8b we give an example of how we measure the Upwards and Forwards Wrist Angles in a grasp. We calculated the values of these measures for the grasps in the grasp set A. In Figure 5.9 we show the occurring measure values for these four measures. The figures show a clear preference of valid grasps for a part of the domain of the measures. So for these measures is it easy to set the range boundary parameters. The distance between the palm and the object is at most 8cm for valid grasps, while the values of invalid grasps are spread across the whole domain. The distance between the palm and the object is at least 0.75 cm and at most 12 cm. For the upward wrist angle, valid grasps have values of at most 55 degrees. The values of the forward wrist angle of valid grasps lies between 35 and 95 degrees. The grasps with values outside of these ranges are all invalid.

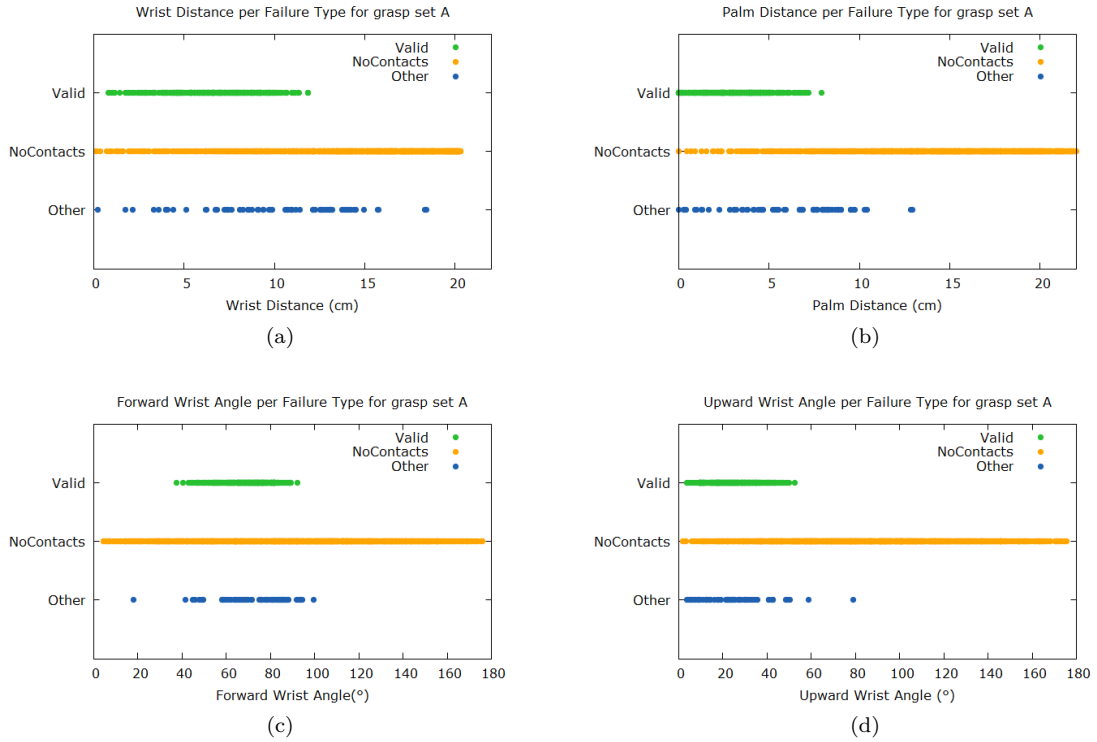


Figure 5.9: The values for the object related measures for grasps in grasp set A. (a) Values of the wrist distance measure for grouped per output type (b) Values of the palm distance measure (c) Values of the forward wrist angle measure (d) Values of the upward wrist angle measure (e) the upward wrist angle set out against the wrist distance (f) the upward wrist angle set out against the forward wrist angle. The green dots are the valid grasps, the orange depict dots the NoContact grasps, the dark blue dots are the grasps of the remaining output types.

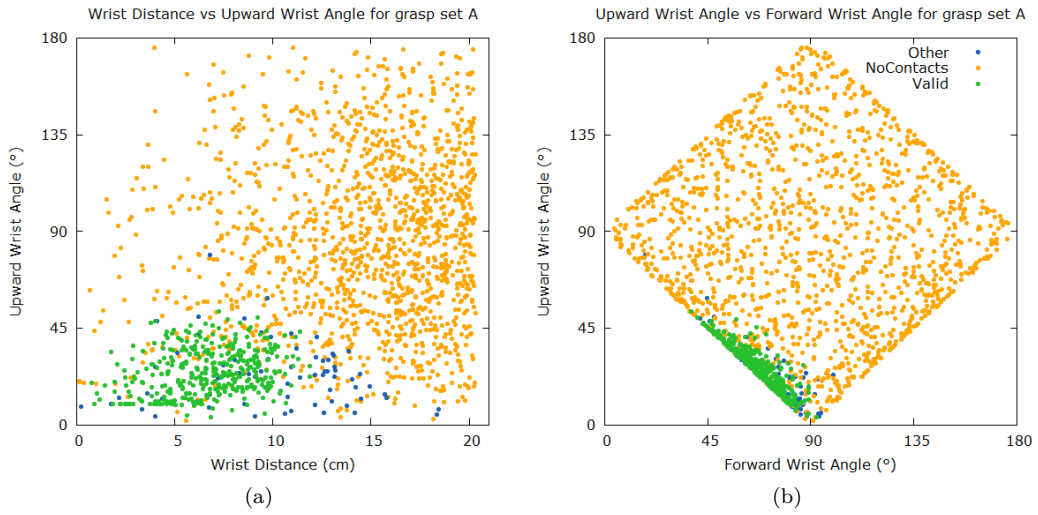


Figure 5.10: Two pairs of object related measures set out against each other for grasps from grasp set A. (a) The upward wrist angle set out against the wrist distance (b) The upward wrist angle set out against the forward wrist angle. The green dots are the valid grasps, the orange depict dots the NoContact grasps, the dark blue dots are the grasps of the remaining output types.

Next we made graphs in which we set out the different measure against each other. Not all combinations led to further insights. We show two of the combination in Figure 5.10. In Figure 5.10b we plot the forward wrist angle (on the x-axis) against the upward wrist angles(on the y-axis). In this graph we see that the most valid grasps lie close to the line $y = 95 - x$ for $x = [35, 95]$. Looking closer, we see that all valid grasps lie at $x + y \leq 115$. We want to take advantage of this by pruning out the grasps that have $x + y > 115$. So we add an extra measure:

Wrist Angle Sum This is the sum of forward and upwards wrist angles. As such it has a theoretical range of 0 and 360 degrees. Note, however, that the upward and forward vectors in the wrist have the same origin and are perpendicular to each other. Thus, their summed angle compared to one point will always be at least 90 degrees and at most 270 degrees. This is visible in the diamond shape of the point cloud in Figure 5.10b.

So we have five object related measures. We have seen in Figures 5.9 and 5.10b that the values of these measures for valid grasps lie in a subset of the range in which all grasps lie. We set the allowed ranges of the object related measures such that all valid grasps lie inside these ranges. An obvious consequence of imposing these ranges is that other valid grasps (not satisfying these constraints) will no longer be found. However, as we have not seen any grasps that lie outside of these bounds, the probability of valid grasps existing outside of these boundaries is fairly low. The boundaries of the allowed ranges as displayed in Table 5.3.

Measure Name	Minimum Value	Maximum Value
Wrist distance	0.75 cm	12 cm
Palm distance	0 cm	8 cm
Upwards wrist angle	0 degrees	55 degrees
Forward wrist angle	35 degrees	95 degrees
Wrist angle sum	90 degrees	115 degrees

Table 5.3: The allowed ranges for the object related measures.

Algorithm 11 ModdifiedPlaceWristFrame

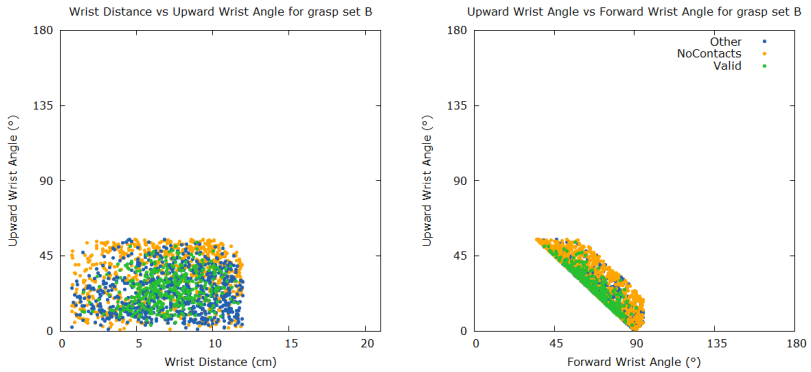
Input: Graspable object objectToGrasp
Output: A frame for the wrist

objectBox ← bounding box B that fits snugly around the object
objectCapsule ← gets the volume C of points that lie maximally d_{mtip} away from objectBox
position ← a point within objectCapsule
rotation ← random rotation
placement ← (position, rotation)
while placement.position lies on the outside of objectBox
or OutsideAllowedRange_WristDistance(CalculateWristDistance(placement))
or OutsideAllowedRange_PalmDistance(CalculatePalmDistance(placement))
or OutsideAllowedRange_UpwardsWristAngle(CalculateUpwardsWristAngle(placement))
or OutsideAllowedRange_ForwardsWristAngle(CalculateForwardWristAngle(placement))
or OutsideAllowedRange_WristAngleSum(CalculateWristAngleSum(placement))
do
  position ← a point within objectCapsule
  rotation ← random rotation
  placement ← (position, rotation)
end while
return placement

We have designed a set of measures for which we selected allowed ranges such that we prune out unpromising wrist placements early. We call the boundaries of these ranges *placement parameters*. To include the placement parameters in our algorithm, we change the wrist placement algorithm (originally as in Algorithm 2) to the pseudo-code in Algorithm 11. To see how well this improves our algorithm we generate grasp set B with the modified version of our algorithm. It is constructed like we explained in Section 5.1 with 25 valid and 100 invalid grasps per holdout. Negative grasp set B^- contains 1500 failed grasp attempts, and positive grasp set B^+ contains 375 valid grasps. Grasp set B is the grasp set of B^- and B^+ combined.

In Figure 5.10 we see that dots depicting valid grasps lie compactly together. The dots of invalid grasps are more uniformly spread across the graph. In grasp set A 95% of the failed grasp attempts were of the type NoContact. Our measures were chosen to improve the probability of finding promising wrist placements. As such, we expect that we will see less NoContact grasps in this version of our algorithm. In Figure 5.11 we show two graphs with the upwards wrist angle plotted against the wrist distance and forward wrist angle respectively. Comparing Figure 5.10 and Figure 5.11 we see that placing the allowed measure ranges cuts out a lot of the unpromising wrist placements. These graphs show very clearly how effective placing boundaries around the search space can be: the point clouds are limited to a very small part of the space taken up in the original graph.

In Figure 5.12b we show the distribution of output types for grasp set B. This can be compared with the distribution of output types in grasp set A (Figure 5.5b). We see that the use of placements parameters to prune the wrist placement has worked as intended: we have reduced the percentage of failed grasp that is of type NoContact from 95% to 48%. This is also visible in the benchmarks of grasp set B^+ shown in Figure 5.12a. When we compare this with the benchmarks we had for the grasp set A (Figure 5.5a) we see that we accelerated the search for valid grasps. This is especially true for the scissors and scalpel. These objects are smaller than the remote with respect to the hand's size and are thus harder to find contacts for. Where the probability of success is 0.00022% for grasp set A, we have increased the probability to 0.0033%, which is 15 times better. The total time spend finding a valid grasp has been reduced from 137s to 52s.

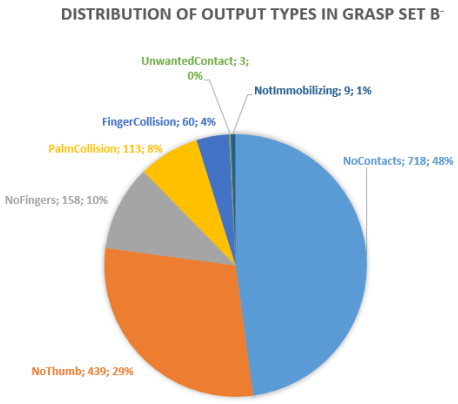


(a) (b)

Figure 5.11: Two pairs of object related measures set out against each other for grasps from grasp set B. The green dots are the valid grasps. The orange depict dots the NoContact grasps. The dark blue dots are the grasps of the remaining output types. (a) The upward wrist angle plotted against the wrist distance (b) The upward wrist angle plotted against the forward wrist angle.

Object/ Holdout	Average Number of Attempts	Average Total Time (ms)
Remote		
GenHold1	16,615	27,840
GenHold2	187,703	320,232
GenHold3	5,788	10,047
Hold1	183,251	326,334
Hold2	9,598	17,349
Total Remote	**80,591**	**140,360**
Scalpel		
GenHold1	9,174	15,592
GenHold2	3,214	5,445
GenHold3	175	302
Hold1	31,628	57,190
Hold2	1,572	2,917
Total Scalpel	**9,152**	**16,289**
Scissors		
GenHold1	318	541
GenHold2	261	444
GenHold3	713	1,201
Hold1	607	1,037
Hold2	3,907	7,035
Total Scissors	**1,161**	**2,052**
End Total	**30,302**	**52,900**

(a)



(b)

Figure 5.12: (a) Average number of attempts and average total time needed to make grasp set B^+ (b) The distribution of output types for the 1500 invalid grasps in grasp set B^-. The labels read 'Output type; number of occurrences; occurring percentage'.

Relative to the presenting hand

We have improved our algorithm by pruning placements that are unpromising, either because they are too far away from the object or because they orientate the hand such that fingers cannot reach the object. Another consideration when placing the wrist is its placement relative to the presenting hand. In handovers between two humans the two participants will probably stand opposite of each other during the handover. This implies that the two hands will be placed on two opposite sides of the object: these are the placements that are most easily reachable while standing opposite of each other. In this case it is intuitively least likely that the two hands will be touching.

In our method, placing the receiving hand too close to the presenting hand often causes grasps to be invalid. So we want to also guide the wrist placement with respect to the presenting hand. We describe a measure for this as follows:

Palm-CoM-Palm Angle: The angle in the XY plane between the two vectors from the CoM to the two hand palms. This angle can lie between 0 and 180 degrees.

The angle between the two palms and the CoM is initially a 3D angle. However, our goal with this measure is to have the two hands approach the object from two different directions on the ground plane: two people attached to the hands stand on opposing sides of the object on the ground plane, to evade standing in each other's personal space. For that reason we project the positions of the palms and the CoM on the XY plane and measure their angle.

Our intuition is that the vectors from the CoM to the two palms point in opposing directions for valid handovers. In Figure 5.13 we see that this is more or less the case for grasp set B^+: the two palm-vectors make a blunt angle (> 90 degrees) for most of the valid grasp attempts. However, there are still a number of grasps for which this angle is sharp. We explain that by the fact that these grasps were generated while only avoiding hand collisions. In real world handovers the two participants also have bodies for which they avoid collisions. We did not take the rest of the body into account in our algorithm. However, with intended use of our algorithm in mind (a human and a virtual character in a VR setting) we feel that constraining the palm-CoM-palm angles to the range of $[90, 180]$ degrees would be beneficial. We loose some valid grasps (in a search space area where success probability is small) in order to increase the practicality of our output.

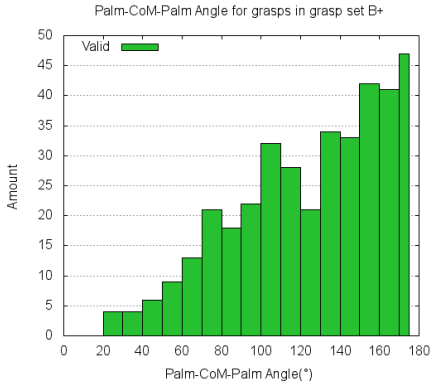
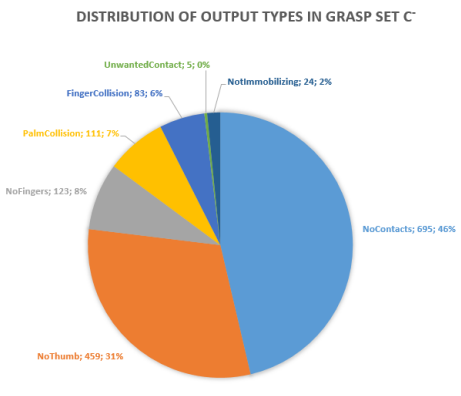


Figure 5.13: Histogram of angles between the two palms and the CoM of the object on the XY plane for the grasp set B^+.

We add the allowed palm-CoM-palm angle range as an extra condition to the wrist placement code. Note that we *add it* to the object related placements parameters. We place it as an extra condition to line 6 in Algorithm 11; as it is only one extra line of code, we do not show pseudo-code including this modification. To see how this impacts the output of our algorithm we generate grasp set C with the latest modified version of our algorithm, in which we guide wrist placement with respect to both the object and presenting hand. It is constructed like we explained in Section 5.1, consisting of 25 valid and 100 invalid grasps per holdout. Negative grasp set C^- contains 1500 failed grasp attempts and positive grasp set C^+ contains 375 valid grasps. Grasp set C is the grasp set of C^- and C^+ combined. Results for this grasp set are shown in Figure 5.14. Figure 5.14b shows the distribution of output types within grasp set C^-. When comparing this to the distribution of in B^- (see Figure 5.12b), we see that the latest modification to our algorithm has not significantly changed the distribution of output types. In Figure 5.14a we show the benchmarks for grasp set C^+. The average number of attempts needed is reduced from 30302 to 15931, which means we cut the number of attempts in half by adding the palm-CoM-palm range. The execution time is reduced from 53s to 28s.

Object/ Holdout	Average number of Attempts	Average Total Time (ms)
Remote		
GenHold1	13,287	22,340
GenHold2	87,560	148,187
GenHold3	5,557	9,583
Hold1	78,176	139,334
Hold2	5,170	9,437
Total Remote	**37,950**	**65,776**
Scalpel		
GenHold1	14,597	25,440
GenHold2	2,521	4,387
GenHold3	170	304
Hold1	27,451	50,205
Hold2	690	1,334
Total Scalpel	**9,086**	**16,334**
Scissors		
GenHold1	265	470
GenHold2	94	166
GenHold3	294	504
Hold1	483	829
Hold2	2,380	4,409
Total Scissors	**704**	**1,276**
End Total	**15,913**	**27,795**

(a)



(b)

Figure 5.14: (a) Average number of attempts and average total time needed to make grasp set C^+ (b) The distribution of output type for the 1500 invalid grasps in grasp set C^-. The labels read 'Output type; number of occurrences; occurring percentage'.

5.4 Finger Posing

In the previous section we have shown the influence the guided wrist placement on the execution time of our algorithm. We have proposed 6 measures that we use to guide the search for promising placements. Five of those measures relate to the placement relative to the object, the last measure relates to placement relative to the presenting hand. We have seen that the use of these measures improves both the probability of success and the execution time of our algorithm. In this section we further reduce the execution time of our method. We showed in Section 5.2 that during the construction of a grasp set most of the time was spent on the Finger Posing step. This is one of the algorithm steps that is executed during *every* grasp attempt. If we reduce the execution time of this step, we could significantly improve the execution time of our method.

We explained our method of finding touching finger poses in Section 4.4. Finger poses are found by systematically trying out poses and finding contacts. In order to reduce execution time we would like to reduce the number of poses for which we check for contacts. The system with which we find poses has a number of parameters: the allowed ranges if the yaw and pitch angles, the set of tried bend ratios and the step size for increasing the bend angle. In this section we will first discuss the yaw angles. Next we propose a modification to the finger posing algorithm that lets us stop posing early for grasps that have no thumb contacts. Lastly, we discuss the pitch angles and investigate limiting the number of tried poses by selecting the bend ratios differently.

Yaw angles

One of the parameters in the finger posing step is the yaw angle. The yaw angle is the angle which a finger can bend sidewards from the base joint. This range of motion fingers have in a real hand for moving sidewards is limited. We currently allow the yaw angles of the fingers to lie in the small range $[-5, 5]$ degrees. One could argue that for simplicity's sake we could make this value a constant. Figure 5.15 shows that in valid grasps the side angles of the index finger are spread uniformly across the allowed range. We saw similar results for the other fingers. The thumb has a much larger allowed range for yaw angles. The yaw angles for which we find touching thumb poses are as spread across the allowed range as we saw for the index finger. We feel that we cannot justify making the side angles a fixed value for the finger. As such we do not make changes in our algorithm with respect to the yaw angles.

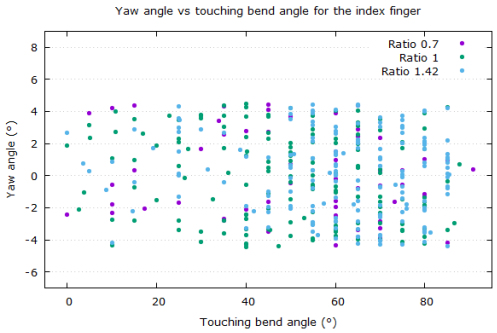


Figure 5.15: Graph were we plot the yaw angle against the bend angle for which touching poses are found for the index finger, made from the grasp set D^+. The dots are coloured per bend ratio; the purple dots have bend ratio 0.7, green dots have bend ratio 1 and blue dots have bend ratio 1.42.

Early check for lack of thumb contacts

We have explained in Section 4.4 that we assume that a valid grasp has a contact on at least the thumb and one finger. In the original Finger Posing algorithm (Algorithm 3) we find a pose *for each finger*, after which we check whether there are sufficient contacts. If we lack a thumb contact, we output a NoThumb grasp. In Figure 5.14b we see that 31% of the failed grasps in the current version of our algorithm (where we guide wrist placement) are of the type NoThumb. So we will probably be able to save time if we skip the finding of poses for the remaining fingers after we already know that the thumb has no contacts. A consequence of stopping after checking the thumb, is that we do not know whether contacts could have been found for the remaining fingers. Thus we do not know whether the output type would have been of type NoContact or NoThumb. As such we add the new output type 'NoThumbEarlystop' for the grasps that are falsified in this manner.

We modify the Finger Posing pseudo-code (originally found in Algorithm 3) to make Algorithm 12. In it, we stop finger posing early when there is no thumb contact. It is constructed like we explained in Section 5.1; with 25 valid and 100 invalid grasps per holdout. Negative grasp set D^- contains 1500 failed grasp attempts, and positive grasp set D^+ contains 375 valid grasps. Grasp set D is the grasp set of D^- and D^+ combined. In Figure 5.16a we see the average number of attempts needed to find one valid grasp. We compare these results with the benchmarks of the previous version of our algorithm (Figure 5.14a), in which we guide the placement, but without the early stop. We observe that the number of attempts has not changed significantly A second observation, when comparing benchmark results between grasp sets C and D, is the reduction of average total times. On average the total time is reduced from 28s to 13s. In Figure 5.16b we show the change this algorithm improvement has on the distribution of output types: the NoThumb and NoContact grasps are now grouped together under the name NoThumbEarlystop.

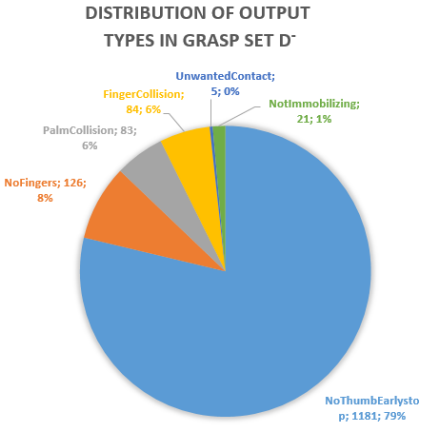
Algorithm 12 ModifiedCalculateHoldPoseAndContacts

Input: wristPlacement, object
Output: A hand pose and a list of contacts between the hand and object

```
contacts ← empty list
fingerPoses ← empty list
for f = 0; f < 5; f = f + 1 do
    fingerPoseF, contactsF ← FindTouchingFingerPose(f, wristPlacement, object)
    contacts.Add(contactsF)
    fingerPoses.Add(fingerPoseF)
    if f == 0 and contacts is empty then
        This grasp is invalid
        return null
    end if
end for
pose ← (wristPlacement, fingerPoses)

return pose, contacts
```

Object/ Holdout	Average Number of Attempts	Average Total Time (ms)
Remote		
GenHold1	11,964	9,806
GenHold2	90,882	77,770
GenHold3	4,186	3,605
Hold1	71,477	50,009
Hold2	3,986	2,611
Total Remote	**36,499**	**28,760**
Scalpel		
GenHold1	17,350	11,319
GenHold2	2,316	1,538
GenHold3	118	90
Hold1	24,933	14,910
Hold2	457	245
Total Scalpel	**9,035**	**5,621**
Scissors		
GenHold1	250	148
GenHold2	133	106
GenHold3	464	327
Hold1	194	140
Hold2	2,433	1,086
Total Scissors	**695**	**361**
End Total	**15,410**	**11,581**

(a)



(b)

Figure 5.16: (a) Average Number of Attempts and average total time needed to make grasp set D^+ (b) The distribution of output type for the 1500 invalid grasps in grasp set D^-. The labels read 'Output type; number of occurrences; occurring percentage'.

Bend Angles and Ratios

Earlier in this section we have reduced the number of poses we check for contacts by stopping early when the thumb does not find a contact. This has reduced the average time of finding a valid grasp to 13s. Still, most time in our grasp synthesis method is spent executing the Finger Posing step. In order to decrease the number of checked poses further we could do three things: we could lower the number of bend ratios, constrain the allowed range of the pitch angles or increase the step size. We have already argued why increasing the step size is a bad idea: this would increase penetration of the finger into the object and thus increase the amount of FingerCollision grasps. We will thus investigate the other two options.

First we investigate the bend ratios and the bend angles for which contacts are found. The use of these parameters is shown in Algorithm 4. The bend angle is the angle which we increase stepwise until a touching pose is found. The base pitch angle is equal to the bend angle. The bend ratio is multiplied with the bend angle to find the value of the second pitch angle. We selected the three bend ratios 0.7, 1 and 1.42 for our experiments. The whole range of the bend angles is iterated through once for each of the bend ratios. We output the first touching pose we find and as such not always try all bend angles or ratios. When no contact is found, however, we have tried all of them. The invalid grasps that do not have sufficient contacts (NoContact, NoThumb and NoFingers) take up 85% of the invalid grasps found in grasp set C. If we can reduce either the range of bend ratios or the number of bend ratios we check, we can decrease the execution time for the Finger Posing step.

We want to see whether we can further constrain the allowed ranges for bend angle and reduce the number of pose checks this way. We look at the successful bend angles in a previous grasp set. Note that the grasp set we use for this is grasp set C (where we only improved on the Wrist Placement step) and not grasp set D (where we also stop early after a lack of thumb contacts). We felt that, as we investigate per finger which bend angles and ratios are successful, it would be a waste to forget the finger poses that would touch the object when the thumb does not have a contact. These finger poses however, are not administrated for grasp set D by design. As such we investigate the bend angles and ratios through grasp set C.

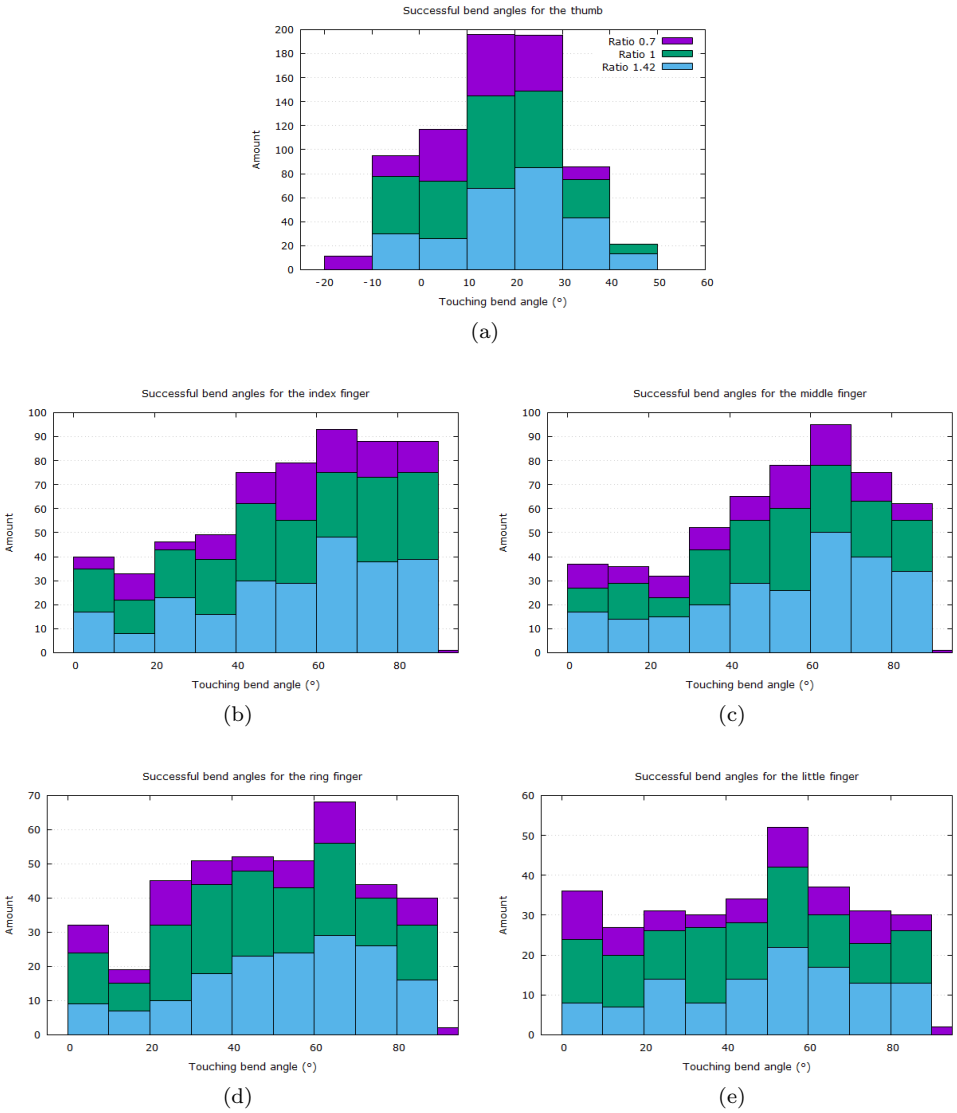


Figure 5.17: Five histograms (one for each finger) showing the bend angles for which a touching pose for a finger was found in grasp set C. The histograms are binned per 10 degrees. The colours on the bars show the bend ratio that was used (purple is ratio 0.7, green is ratio 1 and blue is ratio 1.42). (a) shows the histogram for the thumb, (b) shows the histogram for the index finger (c) shows the histogram for the middle finger (d) shows the histogram for the ring finger (e) shows the histogram for the little finger.

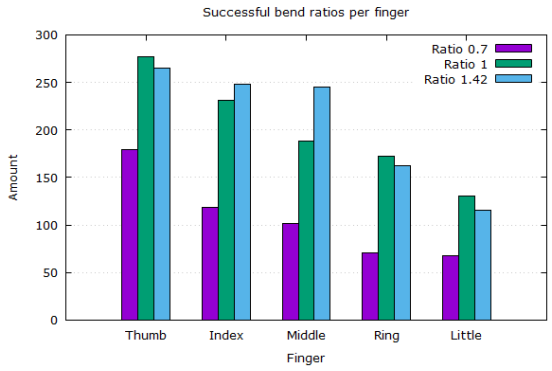


Figure 5.18: Number of times a bend ratio was used in the grasp set C.

Figure 5.17 shows histograms for the number of touching finger poses per bend ratio, grouped by bend angle in bins of 10 degrees. We observe that none of the histogram bins are empty, which means that the full range is used. The thumb has most of its successful bend angles around the centre of the allowed range. For the remaining fingers most successful bend angles lie on the higher side of the range. The graphs show a vague preference for certain bend angles. However, too many touching poses are found at bend angle values around the extremes of the range. We cannot justify constraining the allowed ranges for bend angles of the fingers. We could argue about the usefulness of slightly reducing the allowed range of the thumb. However, that by itself would not provide us with much of a time decrease. Thus we do not think that constraining the allowed bend angle ranges would reduce the execution time much.

We have explained that we cannot decrease the number of pose checks by further constraining the range of allowed bend angles. Let us now investigate whether we can decrease the number of bend ratios that we try contact finding for.

In Figure 5.18 we show the number of times a bend ratio finds a touching pose. We observe that the bend ratio 0.7 is less often successful at finding a contact than the ratios 1 or 1.42. This is especially true for the four fingers. The ratios 1 and 1.42 are equally as good at finding touching poses. We suspect that wrist placements exist for which a finger could have found a touching pose with each bend ratio. In this situation, the first bend ratio we try determines the output pose. As the reach of the fingers does not differ that much between the different bend ratios we suspect that we could remove at least one of the three from the selected set. To test which one that would be, we take all the grasps in grasp set C^+ and recalculate the touching poses with all three ratios. We copy the wrist placement and yaw angles from the old grasp and try to find a touching pose with all three bend ratios. The result of this is shown in Figure 5.19. In this figure we show per finger the number of touching poses we would *not* find when the old bend ratio is replaced with a new one. We observe that for the fingers we would lose barely any contact when we replace the old ratio 0.7 with either of the ratios 1 or 1.42. As such, we suspect that we can remove the ratio 0.7 from the set of tried bend ratios of the fingers. This would reduce the execution time of the Finger Posing step by up to a third. We will test out this suspicion later in this section.

Instead of removing a bend ratio from the set, we can also just use only one bend ratio per finger. The areas that a finger can reach do not differ much between bend ratios; the areas of reach of the three bend ratios overlap each other. This implies that for some wrist placements, a finger could find a contact for each of the bend ratios. Additionally, this implies that if the first tried bend ratio does not find a contact, there is only a small probability that a different ratio does find a contact. One could therefore reason that trying all three ratios is a waste of time: it could be better to try only one ratio and declare the finger as contactless when this one ratio does not find a touch on the object. Now we need to find out which ratio that should be. We could choose randomly from the allowed ratio set, or we could set one fixed value.

In the previous paragraphs we have proposed two ways that could reduce the time we spend Finger Posing: One way is to remove the 0.7 ratio from the set of tried ratios for the fingers. Secondly, we could try only one ratio for finding a contact, either by choosing randomly or by making it a fixed value. A third option would be to combine the two options. We conduct an experiment to find the best way of selecting the tried bend ratios. We modify the current version of our algorithm (where we stopped early after a lack of thumb contacts) to select the set of tried bend ratios alternatively. We try a few different manners of selection: 'try all bend ratios', 'try all bend ratios for but ignore 0.7 from the fingers', 'select one bend ratio randomly', 'select one bend

Number of Lost Contacts when redoing the Finger Posing with a single Bend Ratio				
Finger	**Lost on New Ratio 0.7**	**Lost on New Ratio 1.0**	**Lost New Ratio 1.42**	**Total Lost**
Thumb				
Old Ratio 0.7	-	25	27	**52**
Old Ratio 1	37	-	30	**67**
Old Ratio 1.42	46	52	-	**98**
Lost on Thumb	**83**	**77**	**57**	
Index Finger				
Old Ratio 0.7	-	0	6	**6**
Old Ratio 1	45	-	31	**76**
Old Ratio 1.42	49	30	-	**79**
Lost on Index Finger	**94**	**30**	**37**	
Middle Finger				
Old Ratio 0.7	-	2	3	**5**
Old Ratio 1	15	-	14	**29**
Old Ratio 1.42	37	23	-	**60**
Lost on Middle Finger	**52**	**25**	**17**	
Ring Finger				
Old Ratio 0.7	-	1	1	**2**
Old Ratio 1	14	-	9	**23**
Old Ratio 1.42	18	11	-	**29**
Lost on Ring Finger	**32**	**12**	**10**	
Little Finger				
Old Ratio 0.7	-	0	1	**1**
Old Ratio 1	6	-	1	**7**
Old Ratio 1.42	11	8	-	**19**
Lost on Little Finger	**17**	**8**	**2**	
Total Contacts lost on All	**278**	**152**	**123**	
Total Contacts Lost on Fingers	**195**	**75**	**66**	

Figure 5.19: The amount of contacts that was lost when recalculating finger poses with each of the three ratios. The lower numbers for the middle, ring and little finger are also caused by those finger having a lower number of contacts in grasp set C+.

ratio randomly but ignore 0.7 from the fingers' and 'use only ratio 1' or 'use only ratio 1.42'. The first of this list is the unmodified bend ratio selection method; it was added as the control group to which we compare the other groups.

We tested the different options for bend ratio selection and show the results in Figure 5.20. Overall, we see that all options show an improvement to the algorithm's execution time. For the scissors (which already require a very low number of grasp attempts), improvement by any of the options is barely significant. The remote has a few holdouts for which finding grasps is hard, which means there is more room for improvement. The average improvement across all objects is thus mostly influenced by improvements to grasp finding for the remote. Trying only one ratio has a strange influence on the output of the scalpel: its needed number of attempts goes up (by, on average, $\frac{2}{3}$ of the old numbers), but the execution time is reduced by about a $\frac{1}{5}$. As we are looking for ways to improve the execution time, this is not necessarily a bad thing. We can afford to attempt to make a grasp more often if it means that the overall execution time is reduced.

	Benchmark results for various ways of selecting the tried bend ratios in the Finger Posing step											
	(a) Try All Bend Ratios		(b) Try All Bend Ratios, but ignore ratio 0.7 for the fingers		(c) Try 1 Random Bend Ratio		(d) Try 1 Random Bend Ratio, but ignore 0.7 for the fingers		(e) Use only ratio 1.0		(f) Use only ratio 1.42	
Object/ holdout	Average Number of Attempts	Total Time (ms)	Difference in Number of Attempts	Difference in Total Time (ms)	Difference in Number of Attempts	Difference in Total Time (ms)	Difference in Number of Attempts	Difference in Total Time (ms)	Difference in Number of Attempts	Difference in Total Time (ms)	Difference in Number of Attempts	Difference in Total Time (ms)
Remote												
GenHold1	13,380	10,665	-1,655	-3,716	-2,927	-6,523	-1,859	-6,068	1,523	-4,726	-5,815	-7,615
GenHold2	108,079	86,844	-5,051	-22,002	-28,174	-51,339	-33,225	-53,497	2,904	-35,295	-25,866	-46,229
GenHold3	3,617	2,952	1,397	192	-66	-1,395	84	-1,327	1,213	-824	415	-760
Hold1	94,775	62,776	-28,749	-29,570	-56,816	-50,065	-47,207	-46,784	-20,497	-37,854	-44,498	-45,551
Hold2	4,831	2,999	1,662	35	151	-1,478	8	-1,502	1,152	-1,166	-1,189	-1,865
Total Remote	**44,936**	**33,247**	**-6,479**	**-11,012**	**-17,566**	**-22,160**	**-16,440**	**-21,836**	**-2,741**	**-15,973**	**-15,390**	**-20,404**
Scalpel												
GenHold1	16,336	10,314	-124	-2,795	9,209	-2,911	3,638	-4,525	8,542	-3,082	16,896	-573
GenHold2	2,738	1,735	-72	-490	2,586	-193	1,105	-619	618	-748	1,835	-387
GenHold3	124	86	-18	-32	88	-20	23	-39	56	-29	19	-39
Hold1	23,317	13,410	1,943	-2,876	21,373	-1,918	28,235	-103	36,080	1,883	18,132	-1,429
Hold2	641	323	-17	-92	354	-99	289	-109	213	-130	315	-102
Total Scalpel	**8,631**	**5,174**	**342**	**-1,257**	**6,722**	**-1,028**	**6,658**	**-1,079**	**9,102**	**-421**	**7,440**	**-506**
Scissors												
GenHold1	300	175	-42	-63	-1	-94	39	-82	144	-55	35	-84
GenHold2	157	119	-17	-37	-16	-61	26	-49	31	-47	25	-47
GenHold3	383	265	-15	-74	230	-62	133	-98	370	-23	100	-104
Hold1	204	141	17	-25	133	-29	13	-68	114	-37	64	-49
Hold2	2,953	1,285	-321	-447	1,298	-469	353	-649	1,910	-364	634	-585
Total Scissors	**800**	**397**	**-76**	**-129**	**329**	**-143**	**113**	**-189**	**514**	**-105**	**172**	**-174**
End Total	**18,122**	**12,939**	**-2,071**	**-4,133**	**-3,505**	**-7,777**	**-3,223**	**-7,701**	**2,292**	**-5,500**	**-2,593**	**-7,028**

Figure 5.20: Results of more carefully choosing the set of tried bend ratios in the Finger Posing step. We made 50 in stead of 25 valid grasps per holdout to gain more reliable averages. Column (a) shows the benchmark for the control group (constructed with the algorithm without a change tot the bend ratio selection method). Column (b), (c), (d) and (e) show the *difference* of specific ratio selection in respect to column (a). In these columns negative numbers mean improvements; we coloured big increased numbers in red and big decreased numbers in green and to reflect this. For column (b) we tried all bend ratios for the thumb and ratios 1 and 1.42 for the fingers. In column (c) we chose one of the three ratios randomly. For Column (d) we chose one ratio randomly while ignoring the ratio 0.7 for the fingers. In columns (e) and (f) we use only the ratio 1 and the ratio 1.42 respectively.

In column (b) we show that by only ignoring ratio 0.7 for the fingers the execution time is decreased by about $\frac{1}{3}$. At the same time, the number of attempts has barely changed. Column (c) shows us that randomly selecting one ratio decreases both the number of attempts and the execution time on average. For this column we see that the numbers for the remote are reduced with relatively big amounts. We also see that the amount of attempts for Hold1 and (to a lesser extend) GenHold1 of the scalpel have increased with a significant amount, although their average execution times are slightly decreased. Overall, this column shows the best results. Interestingly enough, adding the two options used for (b) and (c) together is actually slightly *less* effective than just using the second option. As the numbers are not that far apart we reason this is caused by the unstable averages again. We had hoped to avoid this by doubling the number of valid grasps per holdout. Column (d) and (e) show what happens when we fix the bend ratio to values 1.0 and 1.42 respectively. Only using ratio 1 does not work well: finding touching poses for the scalpel is most difficult for this bend ratio selection. We also see a relatively small decrease of execution time in comparison with other bend ratio selection methods. Using only bend ratio 1.42 seems promising however: its reduction of execution time is similar to the option 'select one randomly' and its sister option 'select one randomly ignore 0.7 for the fingers'. The options from columns (c), (d) and (f) have similar benchmark results; any of the options will improve the algorithm. We prefer to select a bend ratio randomly, as this method has slightly better benchmark results. Note that the differences are small enough to be caused by the unstable averages again. Choosing the option of column (c) over the options from columns (d) and (f) is arbitrary. We modify our algorithm to reflect the new bend ratio selection method in Algorithm 13.

We rename the positive grasp set we found to make column (c) for Figure 5.20 to grasp set E^+. In Figure 5.21a we shown its benchmark results. When comparing these benchmarks with the benchmarks from grasp set D^+, we see that our latest algorithm modification (selection of bend ratios) has not influenced the number of needed attempts. We do see however, that the average total time for finding one valid grasp attempt has been reduced from 11s to 5s.

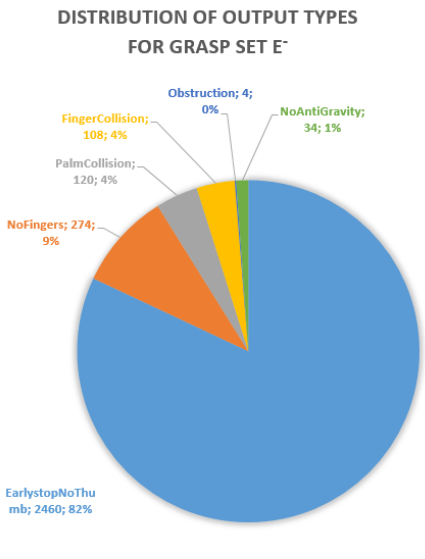
Algorithm 13 ModifiedFindTouchingFingerPose

Input: finger f, wristFrame, object
Output: A finger pose and a list of contacts between this finger and the object

```
contactsF ← empty list
fingerPoseF ← FingerOpenedPose()
bendRatios ← {0.7, 1, 1.42}

ratio ← randomly select one of the values in bendRatios
for angle = 0; angle < 90; angle = angle+angleStep do
    testPose ← (angle, fingerPoseF.baseYaw, ratio∗angle)
    touchSegments ← calculate line segments over the front of this finger's phalanges for testPose

    for segment ∈ touchSegments that intersect with the object do
        contactsF.add(IntersectObject(segment))
    end for
    if contactsF is not empty then
        return (testPose, contactsF)
    end if
end for

return fingerPoseF, contactsF
```

Object/ holdout	Average Number of Attempts	Average Total Time (ms)
Remote		
GenHold1	10453	4141
GenHold2	79905	35504
GenHold3	3551	1557
Hold1	37958	12712
Hold2	4982	1521
Total Remote	**27370**	**11087**
Scalpel		
GenHold1	25545	7403
GenHold2	5324	1542
GenHold3	211	65
Hold1	44691	11492
Hold2	995	224
Total Scalpel	**15353**	**4145**
Scissors		
GenHold1	300	80
GenHold2	142	58
GenHold3	613	204
Hold1	337	112
Hold2	4251	816
Total Scissors	**1128**	**254**
End Total	**14617**	**5162**

(a)



(b)

Figure 5.21: (a) Average number of attempts and average total time needed to make grasp set E^+ (b) The distribution of output type for the 1500 invalid grasps in grasp set E^-. The labels read 'Output type; number of occurrences; occurring percentage'.

Next, we investigate whether changing the selection of bend ratios has influenced the distribution of output types. We generate a grasp set E^-, which consists of 100 failed grasp attempts per holdout. We show its output distribution in Figure 5.21b. We see a surprising effect when comparing the output distribution of this grasp set with the output distribution from grasp set D^- (Figure 5.16b): selecting the bend ratio randomly has slightly decreased the percentage of Palm- and FingerCollision grasps in our negative grasp set, in favour of NoThumbEarlystop and NoFinger grasps. This is not strange: we sacrifice contacts that can only be found with the other two ratios.

In this section we investigated how we could reduce the number of finger poses checked for contacts in order to reduce execution time of the Finger Posing step. One improvement was made by stopping early when no thumb contacts were found. A second improvement was made by further reducing the number of bend ratios for which we try to find a touching pose. After these improvements to our algorithm the average time for finding one valid grasp has been reduced to 5s.

5.5 Collision & Contact Validation

In the previous section we have made improvements to the Finger Posing step in order to decrease the execution time of our algorithm. In this section we will discuss why we cannot speed up the Collision Validation step without losing precision. Our model of identifying noticeable overlap between the grasping hand and another mesh (object/presenting hand) is already quite limited. As explained in Section 4.5, we create a number of segments to represent the volume of the hand and check them for intersections with the mesh. We use 20 line segments for the palm and 3 segments per finger to model the volume that the hand occupies. This is imprecise and allows for penetration of about half a centimetre. For the scalpel (the thinnest object in our object set) we saw that our collision model does not always realise that the scalpel is stabbed right through

the palm. This happens rarely; we have reasoned before that PalmCollision grasps have wrist placements that place the fingers out of reach of the object. We fear that further decreasing the number of line segments we check for intersections during the Collision Validation step will increase the number of false valid grasps, where the object does not simply penetrate slightly through the skin, but goes through the hand palm or a finger.

After the Collision Validation step comes the Contact Validation step, in which we check that our contacts do not lie on forbidden surfaces. As we discussed in Section 4.5, we check whether contacts lie on forbidden surfaces by comparing two sets of triangle indices. One set, the set of forbidden triangle indices, is predefined. The second set is found by collecting from the contacts the indices of the triangles that these contacts touch. We saved the forbidden surface indices in a hash-table, for which we can check in $O(1)$ whether it contains a certain value. We check for each contact's triangle index whether or not it is in the hash-table. The number of contacts is never quite high: we can find 1 contact per phalanx and there are 15 phalanges. While a maximum of 15 contacts is possible, we have not seen grasps that have more than 6 contacts. We feel this contact validity check is already quite time efficient.

5.6 Immobility Validation

The last step of our algorithm is the Immobility Validation step. While we do not often get the chance to execute it (as not many grasps pass the validation checks of previous steps) it is one of the steps that takes a relatively high amount of time.

Right now, we sample 100 force distributions, rank them on their ability to oppose gravity and deem the grasp valid if the best force distribution passes the acceptance threshold. That is how robotics algorithms choose a strong grip: by selecting the best one out of a bigger set. We saw in our data, however, that choosing the best of 100 might not be necessary: an average of 18 satisfactory force distributions is found in a set of 100 tries. We already approximate the gravity opposing wrench and are not going to actually use the forces we calculate for a grasp. In the robotic domain these forces are used to steer to power of motored joints and the force distribution should be optimal. However, we only use the force sets for validation of our grasps. We require at least one force distribution to be better than the acceptance threshold. So we change our Immobility Validation code to be as in Algorithm 14.

Algorithm 14 ModifiedIsImmobilizingGrasp

Input: contacts, objects
Output: Can a set of forces be found for these contacts that is immobilizing?

```
for i in range(0,100) do
    forceDistribution ← ForceDistributionsOverFingers(contacts))
    Wrenches ← DistributionToWrenches(contacts, forceDistribution)
    if WrenchesOpposeGravity(Wrenches) then
        Wrenches ← ScaleToGravitySize(bestWrenches)
        return true
    end if
end for

return false
```

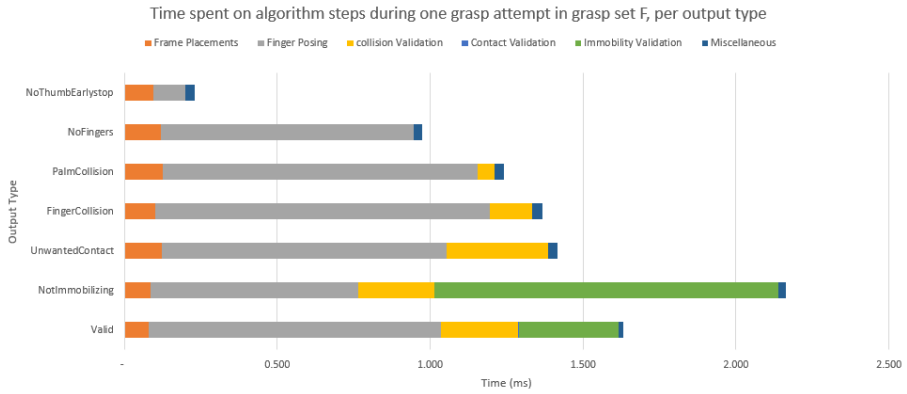


Figure 5.22: Times spent on one grasp attempt per output type in the grasp set F. See how the valid grasps spend much less time in the Immobility Validation step than the NotImmobilizing grasps. (The Contact Validation step takes about 0.001ms when executed; as such it is barely visible in this graph.)

To see how well this improves our algorithm we generate grasp set F with the latest version of our algorithm, in which we deem a grasp valid after finding one gravity opposing force distribution. Grasp set F is constructed just like the previous grasp sets, from 25 valid and 100 invalid grasps per holdout. Negative grasp set F^- contains 1500 failed grasp attempts, and positive grasp set F^+ contains 375 valid grasps. Grasp set F is the grasp set of F^- and F^+ combined. We compare the benchmark results of grasp set F^+ with the benchmark of the previous grasp set E^+. There is no improvement visible in our benchmark. This is to be expected however: we only stop executing the Immobility Validation step early when we find a *valid* grasp. The data in our benchmark per definition only contains one occurrence of a valid grasp. The effect is however visible when we compare the durations of time steps per output type as in Figure 5.22. There is almost a full millisecond difference between the NotImmobilizing and Valid output type. This does not significantly influence the overall speed of our algorithm. It would make more of a change if more of our grasps would reach the last step of our algorithm.

In this chapter we have investigated for every step of our grasp synthesis algorithm whether it could be improved. For the first step, the Wrist Placement step, we improved the success probability. We introduced placement parameters so that we could prune the most unpromising wrist placements. For 6 measures we found the allowed ranges; five of them were object related, the last of them was related to the presenting hand. We improved the Finger Posing step by reducing the step time. Without reducing the success probability we introduced an early stop that was triggered when the thumb lacked contacts. Our next improvement was to select the bend ratio randomly, instead of trying all three ratios. This improved the execution time further, but caused a slight reduction of success probability. We discussed the Collision and Contact Validation steps, but did not find any way to improve on them. Lastly, we improved the execution time of the Immobility Validation. However this steps is not executed often enough for this improvement to influence the total time much.

We have increased the probability of success from 0.00022% (one in 450000 for the naive algorithm) to about 0.0066% (one in 15000 for the final algorithm). This means we have improved the success probability by a factor 30. We have also improved the average time we need for finding one valid grasp from 137s to 5s. The final version of our algorithm finds a valid grasp in 3.6% of the execution time of our naive algorithm.

Chapter 6

Conclusions and Future Work

In this thesis we describe a grasp synthesis method for grasping an object that is held out by a human user. Our goal was for the grasps to be believable to the human in this context. We incorporated realism in our model by introducing four aspects of realism: anatomy, deep collision avoidance, forbidden surface avoidance and immobility. We let the grasps be anatomically plausible by the definition of our hand pose space: we set boundaries around the dimensions of the pose that match the range of motion of the human hand. We made the poses for the fingers by closing them stepwise around the object. A result of this is that, in contrast to other grasp synthesis methods, our algorithm can also make power grasps. Usually, grasp synthesis methods have contacts only at the fingertips, which means they only output precision grasps. We avoid making contacts on forbidden surfaces mostly by deeming such grasps invalid. This is a simple way of avoiding them. It works for us because the forbidden surfaces of our objects occupy only a very small part of the total object surface. Thus we saw very few grasps that were invalid because there were contacts on the forbidden surfaces. To analyse the whether our grasps look believably immobilizing, we modelled the grasp as a physics problem with friction and soft bodies. For a grasp to be realistic, the grasp must apply wrenches on the object such that the object is at rest and force closure applies. Instead of checking whether our grasps satisfy this exactly, we allow grasp that are approximately at rest and satisfy approximate force closure. We are allowed to do this without repercussions as in our virtual environment we do not actually use the forces outside of validation. We placed 4 wrenches around each contact, such that they make the shape of a cut-off friction pyramid. We then attempted 100 times to find a force distribution over the wrenches that, together with gravity, approximately conform to force closure. We sampled the force magnitudes such that each finger had a similar magnitude sum.

We have also seen that there are considerable differences in execution time between objects and individual holdouts. GenHold2 on the Remote (Figure 5.3b) was particularly hard to find a grasp for. The same goes for Hold1 on the Remote (Figure 5.3d) and GenHold1 and Hold1 on the Scalpel (Figures 5.2a and 5.2d). In these four holdouts, the presenting hand blocks off about half of the object's surface with its presence. We suspect that because of this we spend many attempts colliding with the presenting hand. In the remaining difficult holdout the scalpel dangles from between three fingers, which means most of the normals on the surface were perpendicular to the direction of gravity. We think that it was hard to find a force distribution for which the sum would still oppose gravity. These difficult holdouts made our data slightly unpredictable. The probability of success in one grasp attempt is so small that we do not find stable averages: a few

particularly bad outliers could be the cause of the big differences between benchmark results of two positive grasp sets. Datasets of 25 per holdout might have been too small to give a good overview of averages. That is also why we chose to use 50 valid grasps per holdout instead of 25 for the data in Figure 5.20. The high average numbers of the remote made them difficult to compare with the improvements we made for the number of the scissor-holdouts, which were generally very low.

Our original algorithm was not fast enough to work in an interactive environment: it took between 6 and 300 seconds to find one valid grasp, with an average of 137s. We looked at most of the parameters in our algorithm to see if we could find values for them that reduce the execution time of our algorithm. The final version of our algorithm finds a valid grasp in 5s on average. This is a lot faster than the average time the naive version of our algorithm takes. Further improvement can be made by adding more placement measures. With the 6 placement measures we rule out wrist placements that are unpromising. There are probably different measures we can still try for finding better wrist placements. Another option is to look at the implementation itself. Our code is not particularly optimized, so in that regard execution time reduction can probably be found. Currently we run the implementation on a single thread. However, the sampling process includes generating many samples, which are constructed independent of each other. We could speed up the generation process by constructing multiple samples at once using multi-threading. Unity's physics library, which we use for the collision checking, does not allow multi-threading. Switching to another engine might lead to better results.

6.1 Future Work

There are three aspects of our method that we want to investigate further.

The first aspect we want to investigate is the quality of our output with respect to realism. As the grasp taxonomy contains the common types of grasps, we propose to visually verify the grasps by classifying them with Feix et al.-s [10] taxonomy. The grasps that are unclassifiable are not human-like and thus unrealistic. While these grasp look invalid to human user, they are valid according to our objective assessment of realism; they are our method's false valid grasps. We are interesting the percentage of false valid in our positive grasp sets. Additionally, we want to know whether the improvements to our algorithm changed the percentage of false valid grasps.

The second aspect we are interested in investigating is the 'comfort' measure we proposed for evaluating the grasps. With the comfort measure, we would favour grasps with force magnitudes that are evenly distributed across the touching fingers. A good evaluation method could root out the unnatural looking grasps that our method produces. We are interested in seeing whether this evaluation measure would rank a false valid grasp after a true valid grasp. As we did not have time to test our comfort measure we do not know how effective it could have been.

The last aspect we want to further investigate is the execution time of our algorithm when used in its intended context. Our initial goal was to use our grasp synthesis method for a real-time application in Virtual Reality. However, the current execution time does not allow us to reach that level. Note that the computer we tested our implementation on does not contain hardware strong enough to support VR. We would have liked to know our algorithm's execution time on a computer that can run VR applications.

Bibliography

[1] Buryanov Alexander and Kotiuk Viktor. "Proportions of hand segments". In: *Int. J. Morphol* 28.3 (2010), pp. 755–758.

[2] Sheldon Andrews and Paul G Kry. "Goal directed multi-finger manipulation: Control policies and analysis". In: *Computers & Graphics* 37.7 (2013), pp. 830–839.

[3] Yunfei Bai, Kristin Siu, and C Karen Liu. "Synthesis of concurrent object manipulation tasks". In: *ACM Transactions on Graphics (TOG)* 31.6 (2012), p. 156.

[4] Ch Borst, Max Fischer, and Gerd Hirzinger. "A fast and robust grasp planner for arbitrary 3D objects". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 3. IEEE. 1999, pp. 1890–1896.

[5] Maya Cakmak, Siddhartha S Srinivasa, Min Kyung Lee, Jodi Forlizzi, and Sara Kiesler. "Human preferences for robot-human hand-over configurations". In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*. IEEE. 2011, pp. 1986–1993.

[6] Maya Cakmak, Siddhartha S Srinivasa, Min Kyung Lee, Sara Kiesler, and Jodi Forlizzi. "Using spatial and temporal contrast for fluent robot-human hand-overs". In: *Proceedings of the 6th international conference on Human-robot interaction*. ACM. 2011, pp. 489–496.

[7] Matei T Ciocarlie and Peter K Allen. "Hand posture subspaces for dexterous robotic grasping". In: *The International Journal of Robotics Research* 28.7 (2009), pp. 851–867.

[8] Angela Didomenico and Maury A Nussbaum. "Measurement and prediction of single and multi-digit finger strength". In: *Ergonomics* 46.15 (2003), pp. 1531–1548.

[9] Aaron Edsinger and Charles C Kemp. "Human-robot interaction for cooperative manipulation: Handing objects to one another". In: *Robot and Human interactive Communication, 2007. RO-MAN 2007. The 16th IEEE International Symposium on*. IEEE. 2007, pp. 1167–1172.

[10] Thomas Feix, Javier Romero, Heinz-Bodo Schmiedmayer, Aaron M Dollar, and Danica Kragic. "The grasp taxonomy of human grasp types". In: *IEEE Transactions on Human-Machine Systems* 46.1 (2016), pp. 66–77.

[11] Carlo Ferrari and John Canny. "Planning optimal grasps". In: *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. IEEE. 1992, pp. 2290–2295.

[12] Stefan Glasauer, Markus Huber, Patrizia Basili, Alois Knoll, and Thomas Brandt. "Interacting in time and space: Investigating human-human and human-robot joint action". In: *RO-MAN, 2010 IEEE*. IEEE. 2010, pp. 252–257.

[13] Corey Goldfeder, Peter K Allen, Claire Lackner, and Raphael Pelossof. "Grasp planning via decomposition trees". In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE. 2007, pp. 4679–4684.

[14] Kensuke Harada, Kenji Kaneko, and Fumio Kanehiro. "Fast grasp planning for hand/arm systems based on convex model". In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 1162–1168.

[15] Shinya Kajikawa, Takaki Okino, Kohtaro Ohba, and Hikaru Inooka. "Motion planning for hand-over between human and robot". In: *Intelligent Robots and Systems 95.'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*. Vol. 1. IEEE. 1995, pp. 193–199.

[16] Jinsul Kim, Jihwan Park, Yong K Hwang, and MJ Lee. "Advanced grasp planning for handover operation between human and robot". In: *2nd International Conference on Autonomous Robots and Agents*. 2004, pp. 13–15.

[17] Kheng Lee Koay, Emrah Akin Sisbot, Dag Sverre Syrdal, Mick L Walters, Kerstin Dautenhahn, and Rachid Alami. "Exploratory Study of a Robot Approaching a Person in the Context of Handing Over an Object." In: *AAAI spring symposium: multidisciplinary collaboration for socially assistive robotics*. 2007, pp. 18–24.

[18] Hema S Koppula, Ashesh Jain, and Ashutosh Saxena. "Anticipatory planning for human-robot teams". In: *Experimental Robotics*. Springer. 2016, pp. 453–470.

[19] Dana Kulić and Elizabeth A Croft. "Safe planning for human-robot interaction". In: *Journal of Field Robotics* 22.7 (2005), pp. 383–396.

[20] John Lin, Ying Wu, and Thomas S Huang. "Modeling the constraints of human hand motion". In: *Human Motion, 2000. Proceedings. Workshop on*. IEEE. 2000, pp. 121–126.

[21] C Karen Liu. "Dextrous manipulation from a grasping pose". In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 59.

[22] C Karen Liu. "Synthesis of interactive hand manipulation". In: *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association. 2008, pp. 163–171.

[23] Efrain Lopez-Damian, Daniel Sidobre, Stephane DeLaTour, and Rachid Alami. "Grasp planning for interactive object manipulation". In: *Proc. of the Intl. Symp. on Robotics and Automation*. 2006.

[24] Andrew T Miller, Steffen Knoop, Henrik I Christensen, and Peter K Allen. "Automatic grasp planning using shape primitives". In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Vol. 2. IEEE. 2003, pp. 1824–1829.

[25] AJung Moon, Daniel M Troniak, Brian Gleeson, Matthew KXJ Pan, Minhua Zheng, Benjamin A Blumer, Karon MacLean, and Elizabeth A Croft. "Meet me where i'm gazing: how shared attention gaze affects human-robot handover timing". In: *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*. ACM. 2014, pp. 334–341.

[26] Matthew KXJ Pan, Vidar Skjervøy, Wesley P Chan, Masayuki Inaba, and Elizabeth A Croft. "Automated detection of handovers using kinematic features". In: *The International Journal of Robotics Research* (2017).

[27] Robert G Radwin, Seoungyeon Oh, Todd R Jensen, and John G Webster. “External finger forces in submaximal five-finger static pinch prehension”. In: *Ergonomics* 35.3 (1992), pp. 275–288.

[28] Elon Rimon and Joel Burdick. “On force and form closure for multiple finger grasps”. In: *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*. Vol. 2. IEEE. 1996, pp. 1795–1800.

[29] Carlos Rosales, Josep M Porta, Raúl Suarez, and Lluis Ros. “Finding all valid hand configurations for a given precision grasp”. In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 1634–1640.

[30] Jean-Philippe Saut and Daniel Sidobre. “Efficient models for grasp planning with a multi-fingered hand”. In: *Robotics and Autonomous Systems* 60.3 (2012), pp. 347–357.

[31] Emrah Akin Sisbot, Rachid Alami, Thierry Siméon, Kerstin Dautenhahn, Michael Walters, and Sarah Woods. “Navigation in the presence of humans”. In: *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*. IEEE. 2005, pp. 181–188.

[32] Kyle Strabala, Min Kyung Lee, Anca Dragan, Jodi Forlizzi, and Siddhartha S Srinivasa. “Learning the communication of intent prior to physical collaboration”. In: *RO-MAN, 2012 IEEE*. IEEE. 2012, pp. 968–973.

[33] Kyle Strabala, Min Kyung Lee, Anca Diana Dragan, Jodi Lee Forlizzi, Siddhartha Srinivasa, Maya Cakmak, and Vincenzo Micelli. “Towards seamless human-robot handovers”. In: *Journal of Human-Robot Interaction* 2.1 (2013), pp. 112–132.

[34] Shinjiro Sueda and Dinesh K. Pai. “Hand simulation models in computer graphics”. In: (2014).

[35] Unity. *Unity User Manual*. 2017. URL: https://docs.unity3d.com/Manual/index.html (visited on 03/02/2017).

[36] Nkenge Wheatland, Yingying Wang, Huaguang Song, Michael Neff, Victor Zordan, and Sophie Jörg. “State of the art in hand and finger modeling and animation”. In: *Computer Graphics Forum*. Vol. 34. 2. Wiley Online Library. 2015, pp. 735–760.

[37] Yuting Ye and C Karen Liu. “Synthesis of detailed hand manipulations using contact sampling”. In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 41.

[38] Wenping Zhao, Jianjie Zhang, Jianyuan Min, and Jinxiang Chai. “Robust realtime physics-based motion control for human grasping”. In: *ACM Transactions on Graphics (TOG)* 32.6 (2013), p. 207.