DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

A Metric for Hand Comfort/Discomfort Evaluation: Towards Expressivity in Spatial Control

Jonas Mayer

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics: Games Engineering

A Metric for Hand Comfort/Discomfort Evaluation: Towards Expressivity in Spatial Control

Eine Metrik zur Comfort- und Discomfortevaluation der Hand: Hin zur Aussagekraft räumlicher Steuerung

Author:	Jonas Mayer
Supervisor:	Dr. Nicholas Katzakis
Advisor:	Prof. Gudrun Klinker
Submission Date:	Submission date

I confirm that this bachelor's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, Submission date   Jonas Mayer

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Motivation

Over the last two decades human-computer interaction in desktop computer environments has evolved .... resulting in the keyboard-mouse input standard. Using a WIMP interface and multiple keyboard macros users can complete a variety of tasks efficiently and effectively. However, in interaction contexts such as virtual reality and human-robot interaction which become more and more relevant, traditional interaction techniques are not a suitable solution. Speech is an intuitive way of interacting, even though it by nature requires the user to talk constantly which is not optimal for longer periods of time. Additionally, speech interaction is limited when it comes to contexts such as commanding a robot to execute different household tasks or playing a real-time strategy game in virtual reality, where spacial navigation is important. In such cases, pointing with hands is a much more efficient and intuitive solution. By using different hand postures different commands can be issued, which enhances interaction expressiveness similarly to key macros in a traditional environment. The main challenge for hand postures has been fighting physical forces that cause different symptoms like fatigue or discomfort. The latter are generally known to limit user experience and precision [**short1999precision**] in the task environment.

When designers create a hand posture interaction environment, they have to take these physical factors and hand posture comfort and discomfort into account. One goal is to choose an optimal set of hand postures that will ensure maximum task performance and comfort for the user while creating a minimum of discomfort. Especially for larger sets of commands that becomes challenging as there are no straight forward metrics that allows to compare hand postures quickly [**naddeo2015proposal**]. So either a costly user study has to be organized to get objective feedback, or the designers have to rely on their own subjective impressions and assumptions.

Therefore, the main goal of this bachelor thesis is to support the creation and evaluation of a hand posture vocabulary for efficient pointing-based posture interaction.

1.2 Study Goals

The approach taken to solve the problem stated above, is based on the hypothesis that hand posture comfort and discomfort do affect the users performance in a specific task and obviously the user experience. Based on this, two main goals were formulated.

1.2.1 Create a Metric for Hand Posture Comfort/Discomfort

The objective was to create a metric, that allows designers to get a quick and objective evaluation of a hand posture regarding Comfort and Discomfort. The metric should be used to compare similar hand postures and to rule out bad ones directly.

For the creation of the metric state of the art comfort and discomfort models were taken into consideration. Looking at the human hand's complex anatomy, multiple influential factors for hand posture comfort and discomfort were identified. In order to compute a concrete metric value in real time, the identified factors were implemented in a Unity 3D environment. Finally the metrics correctness was verified in a user study.

1.2.2 Show the Metric's Influence on User Performance

While the correlation of Comfort/Discomfort and user experience is rather trivial, the influence of Comfort/Discomfort on user performance was only indicated for specific different contexts. Therefore the objective was to show this correlation in the context of hand posture comfort/discomfort as measured by the metric. This was also targeted in a user study.

This bachelor thesis aims to support the creation and evaluation of a hand posture vocabulary for efficient pointing-based posture interaction. For that we propose a hand posture comfort/discomfort metric that allows for quick and objective hand posture evaluation. We combined state of the art comfort/discomfort models with current hand anatomy and ergonomics knowledge to create models that can predict hand comfort and discomfort given a specific posture. Based on our model we created a naive metric, which we improved in a second step using data from a user study. Finally another user study was used to validate our metric and to show the impact of comfort and discomfort on performance in a hand pointing task.

2 Related Work

2.1 Nikolas Schneider

Being its follow-up work, this Bachelor Thesis was highly influenced by the work of Nikolas Schneider. In his Bachelor Thesis Nikolas Schneider compared three different hand postures, namely a pointing, spiderman and a pinching posture in terms of precision and performance in a 3D-pointing task.

For that, he conducted a user study, where the participants had to perform a target shooting test in a Unity 3D environment using one of the mentioned postures. For that purpose the participants wore an AR-Rift giving them augmentations of the targets to shoot and the shooting direction, indicated by a laser beam. The beam's origin and direction were computed differently for each hand posture. In order to track the participant's hand, a metal construction featuring a Leap Motion Controller and ART marker was strapped to the forearm. Applying a KNN algorithm on the data gained from the Leap, the user's hand posture was determined in order to make sure, the participant would hold the posture throughout the test. Right after the target shooting, users were asked to rate their experience with the hand postures regarding perceived accuracy and comfort.

The results showed that generally the pointing posture performed best, followed by the spiderman posture and finally the pinching posture. The questionnaire revealed similar results for user perception, indicating a connection between user comfort and performance.

2.2 Comfort and Discomfort

As the main goal was to create a quantitative metric for hand posture comfort and discomfort evaluation, it was crucial to understand state of the art concepts of comfort and discomfort and to have a look at similar approaches taken to create comfort metrics.

In their editorial Vink et al. [**vink2012editorial**] give a good overview over current comfort and discomfort definitions, different models explaining the origin of both. They state that even though there has been much research on comfort and discomfort, the results are generally ignored in practical design contexts due to their broad theoretical scope. Concluding they express the importance of further research in order to generate applicable models and metrics for concrete body parts.

Fagarasanu et al. [**fagarasanu2004measurement**] discovered that limbs in neutral postures showed a significantly lower muscle activity, indicating higher perceived comfort. Apostolico et al. [**apostolico2014postural**] defined the term "Range of Rest Posture" (RRP), a angular range for articular joints where the joint can be seen as statistically in rest. They further measure the RRP for multiple human joints and express its importance for evaluation of postural comfort. Based on this Naddeo et al. [**naddeo2015proposal**] used a neural network to generate a concrete metric for postural comfort based on RRP. Therefore they compare user comfort ratings of certain joint postures with the measured distance to the RRP. They further also described other potential influential factors to take into account when evaluating comfort.

Short et al. [**short1999precision**] conducted a user study to investigate the so called precision hypothesis. The results indicated that generally more comfortable posture generate a higher precision in pointing tasks. This effect is magnified, when the targets become smaller.

subfig

3 Theoretical Foundation

3.1 Comfort and Discomfort Definitions

Before actually creating a metric, it is obviously crucial to have an exact definition of the terms comfort and discomfort. In this bachelor thesis comfort and discomfort will be referred to as described by Vink et al. [**vink2012editorial**].

In their paper comfort is defined as "pleasant state or relaxed feeling of a human being in reaction to its environment". Therefore comfort is a positive emotional state in reaction to the environment highly dependent on emotions and expectation. Comfort is generally related to "luxury, feeling relaxed or being refreshed".

Discomfort on the other hand is defined as "an unpleasant state of the human body in reaction to its physical environment". Physical stress is the main cause of discomfort, a negative state of the body. Discomfort is often felt in the form of fatigue, stiffness and pain and can in extreme cases even lead to injury.

It is important to keep in mind, that comfort and discomfort are in fact not two opposing sides on one scale. They much more are two independent factors influencing the overall well being in different ways, somewhat similar to Herzberg's motivation-hygiene theory. The absence of discomfort does not automatically result in comfort and vice versa.

An example for the importance of this differentiation can be found when choosing the softness of foams for mattresses or seats. While softer materials will continuously increase perceived comfort, having too soft foams will result in reduced postural support, leading to higher stress on muscles and tendons and finally causing discomfort symptoms like stiffness or back pain.

3.2 Hand Comfort and Discomfort Metric Components

Looking at the hand's anatomy the following four components were determined to be most influential on comfort and discomfort based on the definitions from above. In the following section the comfort and discomfort components will be explained and a brief idea for computation will be given.

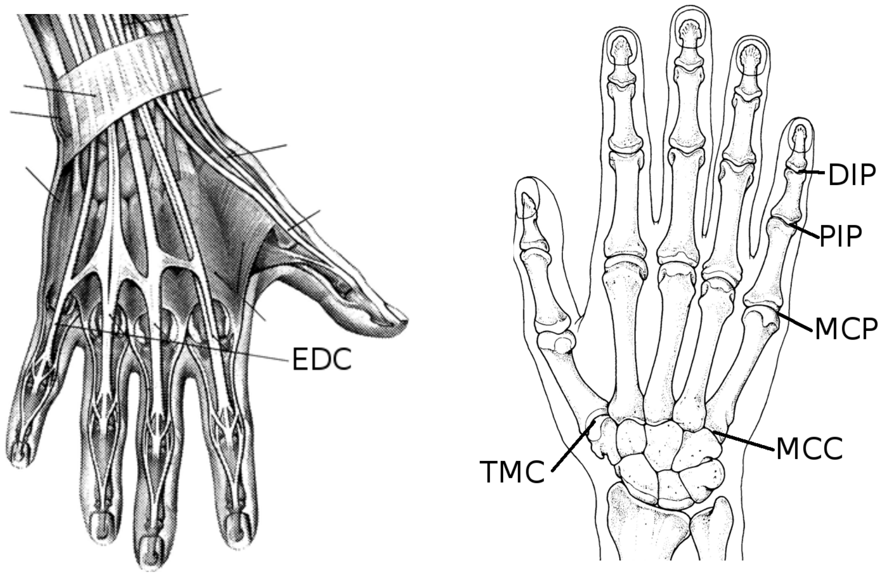

Figure 3.1: Hand Anatomy

3.2.1 Deviation from Range of Rest Posture

The **Range of Rest Posture (RRP)** component is based on the work of Apostolico et al. [**apostolico2014postural**]. They define the "Rest Posture" of a human joint as a posture, where involved muscles are completely relaxed or strain is minimized. When in Rest Posture, maximum comfort is perceived in that particular joint. Thus perceived comfort should decrease when deviating from the RRP. Due to anatomical differences between different humans, it makes more sense to look at the so called "Range of Rest Posture", a range of angles for an articular joint, where the joint "can be considered statistically in rest".

When looking at postures involving multiple joints, Naddeo et al. [**naddeo2015proposal**] state that comfort can be determined by combining the comfort values of the single joints.

In our case, we considered the human hand to have one RRP for each finger joint in a non-resting position with the palm facing downwards, resulting in a range of relaxed hand postures (Figure ???) , where the comfort is maximized. For a particular hand posture denoted by "x" the RRP component *RRP(x)* can be computed by determining the angular distances to the RRP for every joint and adding them up.

3.2.2 The Inter Finger Angles

As it can be seen in Figure 3.1 the hand has a very compact and highly connected system of muscles, tendons and soft tissue that limits the individual movement of fingers. The fingers, excluding the thumb, share **most** of their flexor and extendor muscles. However minor individual flexion and extension of adjacent fingers is still possible due to finger tendons originating from different areas of the muscles. In the case of the *Extensor digitorum communis* (EDC in Figure 3.1) the finger tendons are even interconnected on the back of the hand.

In addition to this only three principal nerves serve the muscles of the hand, which makes it even harder for the motory system to fully differentiate between the individual fingers.

In conclusion of this, hand postures with high bending differences between the fingers should not only cause physical stress on both tendons and muscles, but also cognitive stress. This is due to the human trying to achieve and hold a complex posture with limited cognitive and physical means. This can lead to severe discomfort, which is manifested in cramping up the hand and pain. For a specific hand posture the inter finger angle component *IFA(x)* can be computed, by computing the total angular bending differences of adjacent fingers and adding them up.

3.2.3 Finger Hyperextension

As highlighted by LaViola [**laviola1999survey**] **hyperextension** (Figure 3.2), "*puts more strain on the* [metacarpophalangeal] *joints and tendons than the hand is accustomed to*" [**laviola1999survey**]. Consequently extending the fingers beyond their natural range of motion creates noticeable amounts of discomfort. Even though this might seem redundant to the deviation from RRP on first sight, hyperextension takes a special position as it causes considerably more discomfort, compared to a full flexion of the fingers and compared to what the deviation from RRP would suggest. The hyperextension component *HE(x)* can be computed by simply finding all hyperextended fingers and adding up the hyperextension angle of the MCP.

3.2.4 Finger Abduction

Finger **abduction** (Figure 3.2) is the act of fanning out the fingers using the *interosseus* muscles located within the hand. High abduction also causes stress on the MCP joints (Figure 3.1), the muscles and tendons involved as well as the soft tissue in between the fingers.

Abduction was taken into consideration as a discomfort factor, analogue to the hyperextension, because full abduction creates substantially more discomfort than full

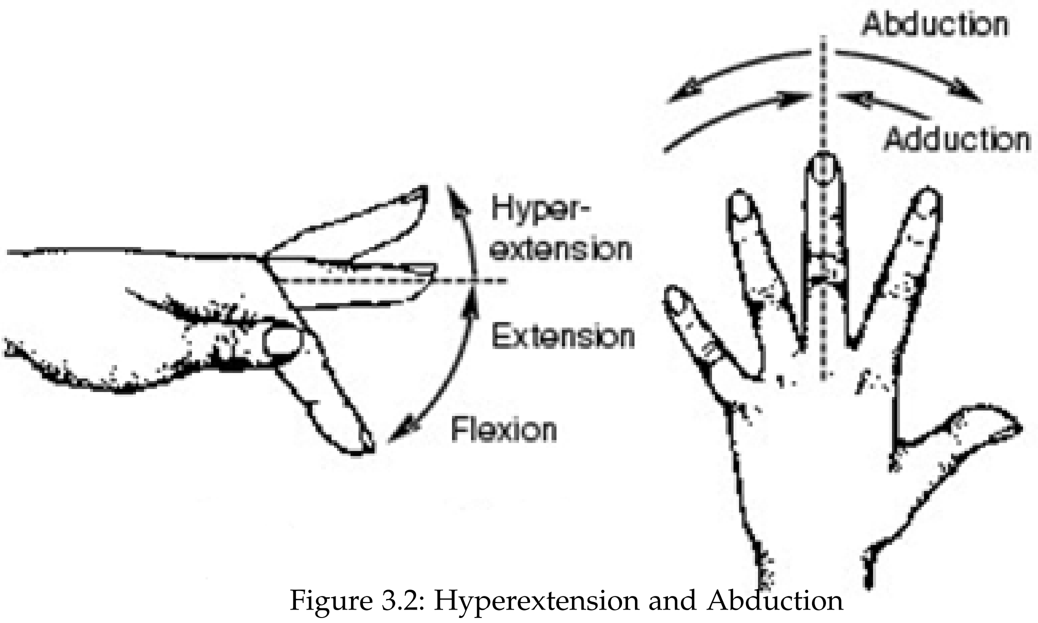

Figure 3.2: Hyperextension and Abduction

adduction (Figure 3.2). By summing all absolute abduction angles of the fingers, the finger abduction component *FA(x)* can be computed.

3.3 Naive Metrics

Now that the different components most influential to comfort and discomfort are known and assumed to be computable (concrete implementation given in section 4.4), it is important to think about how to combine the different factors into one metric value.

A naive approach would be to simply sum up the single comfort and discomfort components for the whole hand. In order to compensate potential differences in intensity scales of the components, it makes sense to balance the component values with weighting coefficients $(c_{RRP}, c_{IFA}, ...)$. The concrete weighting coefficients can be estimated and verified by experimental testing. When assigning the different components to either comfort or discomfort, following functions for a hand posture denoted by "x" can be derived.

$$Comfort(x) = c_{RRP} \cdot RRP(x)$$

$$Discomfort(x) = c_{IFA} \cdot IFA(x) + c_{HE} \cdot HE(x) + c_{FA} \cdot FA(x)$$

The resulting metric for comfort has a minimum value of 0, where user perceived comfort is expected to be minimized. Higher metric values are due to larger distances to the RRP and therefore correspond to lower comfort values.

The discomfort metric, due to the nature of its components, also has a minimum value of 0, where user perceived discomfort is minimized. Higher amounts of discomfort are to be expected, when either high inter finger angles, hyperextension or high abduction are performed, resulting in higher discomfort metric values.

3.4 Improved Metrics

Even though the naive metrics contain the basic causes of comfort and discomfort, they still lack deeper consideration for the anatomical differences between the fingers. The severity of this problem can be experienced when for example comparing the perceived discomfort of the standard pointing posture using the index finger to pointing with the ring finger. The concept of improvement is straight forward: instead of applying the metrics to the whole hand, we consider the contributions from individual fingers and weight them with importance coefficients.

$$Comfort(x) = c_{RRPindex} \cdot RRP(index) + c_{RRPmiddle} \cdot RRP(middle) + ...$$

$$Discomfort(x) = c_{IFAindex} \cdot IFA(index) + c_{IFAmiddle} \cdot + ... + c_{HEindex} \cdot HE(index) + ...$$

In our case we have five comfort values and a total of twelve discomfort values, as the discomfort metric components neglect the thumb.

However, finding the exact weighting coefficients turns out to be rather tricky, as they are generally unknown and due to their number hard to estimate. In order to solve this problem, data from a user study will be processed by a machine learning algorithm in order to compute the best fitting coefficients. In our case, the problem can be reduced to a curve fitting problem that can be solved with for example the least-squares algorithm.

4 Implementation

This chapter will give some details about the implementation, the implementation process, the problems that occurred an how they were solved.

4.1 Basic Setup

4.1.1 Unity

For the implementation of the hand tracking, the metric computation and the user study, Unity 3D 5.3 was chosen as a programming environment. Technically Unity 3D is a powerful game engine, that can be used for 3d or 2d applications and games. It allows users to easily create scenes with different objects whose behavior can be specified with `C#` scripts and are rendered automatically. Due to its complex input manager, it allows the user to receive input from a multitude of input devices and supports most unconventional interaction devices like HMDs and also the Leap Motion Controller. In addition a built-in UI-Manager allows for quick prototyping of user interfaces.

4.1.2 Leap Motion Controller

Similar to Nikolas Schneider's setup, a Leap Motion Controller was used for the actual hand tracking. The Leap Motion Controller, or simply Leap is an optical hand tracker using three IR-LEDs and two IR-Cameras to record the users hand. By providing an SDK and a Unity package, integration into Unity 3D works seamlessly. With the new Leap Orion SDK, the Leap allows for accurate and robust finger tracking in different lighting conditions and usage contexts. The Leap was chosen due to its easy integration combined with the solid tracking performance, that is unrivaled for hands-free hand trackers. Instead of attaching it to the user's arm with a metal construction, potentially causing discomfort to the user, it was simply placed on a desk.

4.1.3 Further Project Configuration

As using the AR-Rift did not seam to benefit the project substantially, it allowed to set up a clean 3D Unity project. The only imported assets were the Leap Motion Unity core assets which provide useful tools for interacting with the recorded hands. All work was based on the Leap sample scene with two basic hand objects.

4.2 Hand Model

In order to make different computations with hands, considerations have to be made about how to represent a hand posture in a virtual environment. In the field of free hand computer interaction, two different hand models are commonly used: angle-based and point-based hand models. The point-based hand model describes a hand posture as a set of 6 6DOF (position and orientation) points. 5 of these represent the finger tips, one represents the palm. In the angle-based hand model, a hand posture is described by the hand's joint angles. Depending on the literature, the joints in the hand have a total of 22 [**su1994logical**] or 23 DOFs [**laviola1999survey**], differentiating in the DOFs of the *trapeziometacarpal* joint (TMC, Figure 3.1). Even though the two models are almost interchangeable, the angle-based model was chosen, as it makes most computations of metric components trivial. A common simplification for angle-based hand models is to neglect the 2 DOFs, given by the *metacarpocarpal* (MCC, Figure 3.1) of the forth and fifth digit. This is done for example by the Leap, because the MCCs hardly move and are therefore are not noticeable for most applications. In conclusion, the implementation was based on a 21DOF angle based hand model, as it can be seen in Figure ????.

The goal for the unity implementation was to create a universally usable and flexible class structure for hand postures, thus the `AngleBasedHandModel` class was created. The class has a reference to a thumb object of class `AngleBasedThumbModel` and four enumerated fingers of class `AngleBasedFingerModel`. In addition the hands position and rotation are stored for debugging. The class also provides a function to calculate the mathematical Euclidean distance to another hand posture, a function to interpolate between two hand postures, a toString function and multiple functions for loading and storing hand postures as CSV.

The `AngleBasedFingerModel` (and `AngleBasedThumbModel`) stores their 4 (and 5) DOFs as float angles. To ease computation the rotation of the MCP (and TMC) is redundantly stored as quaternion. Both classes also implement the same functions as `AngleBasedHandModel` and `AngleBasedFingerModel` additionally provides a function for computing the total flexion of a finger (Figure ???). The three classes are fully serializable to facilitate simple saving and loading of hand postures. In order to obtain the hand posture data from the the Leap in realtime, a `HandObserver` script was attached to each Leap hand objects. The `HandObserver` extracted the hand posture data from the Leap and updated an `AngleBasedHandModel` object while taking the handedness into account.

4.3 Hand Posture Classification

In order to make sure, that the participant would hold the hand posture during the user study, some kind of hand posture classification system had to be implemented. While there are multiple different methods, that can be used for hand posture classification, the k-nearest neighbors (k-NN) algorithm has proven to be both suitable for hand posture classification and comparatively easy to implement.

The k-NN algorithm is a simple machine learning algorithm that can be applied to a multitude of problems with an n-dimensional feature vector. The basic idea is to test the membership of an object using training data. The membership of the vector is determined by the majority of it's k nearest neighbors.

In the case of hand posture classification, we have a 21-dimensional feature vector according to our hand model. The implementation here was based on Nikolas Schneider's implementation, but was refactored to be more versatile in use. Basically the k-NN implementation has two main objectives:

1. Data Collection
2. Actual Posture Classification

For **data collection** a set of training data, consisting of feature vectors with their classification has to be obtained. This was implemented by the `PostureDataHandler` class. The `PostureDataHandler` takes care of the training data, implemented as a list of `TrainingUnit` objects and provides functions to add and delete `TrainingUnits`. A `TrainingUnit` consists of a `AngleBasedHandModel` object that is classified with a `Posture` enumerator. To achieve data persistence, the `PostureDataHandler` saves the data to the hard drive after every modification and automatically loads it when instantiated.

A TrainingManager scene was created that allows the user to manipulate the training data using a GUI. The user can inspect `TrainingUnits`, that are visualized with a `OutputHand` built from Unity primitives. The `TrainingUnits` are grouped by `Posture` and can be deleted individually or as group. In order to add `TrainingUnits` the user has to choose the desired `Posture`, form the hand posture over the Leap and press a button.

The **actual posture classification** classifies a hand posture based on the k nearest neighbors. For that, the distances of the hand posture to all training units has to be computed and the k closest have to be selected. In this case, k is set as the square root of the total training data size.

In the implementation this is handled by the `ThreadedKNN` class. As the name suggest, the classification is done in parallel to the main program using threads. Each frame the `ThreadedKNN` starts a new thread which receives the `AngleBasedHandModel` that needs to be classified. The thread receives a list of `PoseCompareObjects` from the `PostureDataHandler` containing the Euclidian distance of every `TrainingUnit` to the `AngleBasedHandModel` and its classifying `Posture`. The thread then sorts the list by distance and counts the occurrences of the different `Postures` among the k nearest `PoseCompareObjects`. The `AngleBasedHandModel` is then classified as the most frequent `Posture` and the result is written back to the `ThreadedKNN` object.

In the context it was used, the k-NN implementation could differentiate between 10 postures with 50 recorded training samples each. The hand posture classification was performed on two hand robustly in real time without noticeable performance impact or lag.

4.4 Hand Posture Metrics

Now that we have a representation for hand postures, the metric can be implemented. For that a `Comfort` and a `Discomfort` class were created. Both contain functions for computing the total metric value, the metric components for the whole hand and for the single fingers as well as functions for outputting the finger values to a .CSV file.

4.4.1 Comfort

The only component, that is computed in `Comfort` is the distance to the RRP. Theoretically every DOF in the hand has an own RRP, that can be determined through experiments as done by Apostolico et al. [**apostolico2014postural**]. However this turns out to be a costly and lengthy process, that had exceeded the focus of this thesis. Therefore some simplifications had to be made:

1. The RRP would not be determined for every RRP separately but the whole hand simultaneously.
2. The RRP would not be defined as a continuous range but as a discrete set of samples.

In conclusion of this, the RRP would be defined as a set of relaxed hand postures. To implement this, we simply used the 50 samples of the "idle" `Posture` stored in the `PostureDataHandler` to define our RRP. A correct implementation of the distance to the RRP would have been to compute the 21-dimensional bounding volume of the RRP, test a hand posture for collision with the volume and calculate the minimum distance to the volume. Again as a simplification, only the minimum Euclidean distance to any sample of the RRP set was computed. Strictly seen, a hand posture within the RRP could have an RRP value $\neq 0$, but early tests showed the error to be negligibly small. The result were two functions, one that calculated the minimum distance to the RRP for the whole hand, one that calculated it for every single finger individually.

4.4.2 Discomfort

In `Discomfort` the three discomfort components were computed: inter finger angles, hyperextension and abduction.

For the whole-hand computation of inter finger angles, initially only the absolute flexing differences between the fingers were added up. However, it showed early that the anatomical differences between the individual fingers were so severely influencing the inter finger angle discomfort, that they already had to be compensated in the naive metric. Most of all, the ring finger showed to be incapable of having large flexion differences to its adjacent neighbors. However this only stood out when the ring finger had to stick out between its neighbors, not when it was between them. Therefore a ring finger bonus was added, that was computed as follows:

```
  Mathf.Abs((fingers[middle].getTotalFlexion() - fingers[ring].getTotalFlexion()
)- (fingers[ring].getTotalFlexion() - fingers[pinky].getTotalFlexion()));
```

This way, the ring bonus only occurred, when the ring finger stuck out between it's neighbors. The ring bonus was multiplied with a weighting coefficient, that was estimated to 1.3.

For computing the single-finger values, index and pinky received the angle differences to their only neighbors while the middle and ring finger values were computed similarly to the ring bonus.

The computation of the hyperextension was more straight forward. Hyper-extended fingers due to the nature of our hand model have negative extension angles in the MCP. Consequently the single finger hyper extension values are the absolute extension angle of the MCP if the finger is hyper-extended otherwise 0.

Due to the angle-based hand model, computing the single finger abduction component, comes down calculating the absolute of the fingers MCP abduction angle.

For both hyperextension and finger abduction the whole hand metric value is simply the sum of the single finger values.

To get the total naive discomfort value, the 3 whole hand values were weighted with their importances and afterward added up.

4.5 Random Hand Generator

One possible weak spot, when comparing and examining hand postures is the process of choosing the hand postures to compare. In the context of this thesis, as many diverse hand postures as possible had to be compared in order to get universally valid picture. To solve this, one possible approach is to randomly generate hand postures. For this, a `RandomHandGenerator` was created. The class provides a function that returns a random `AngleBasedHandModel`. Internally the function again chooses from 4 different random hand generators randomly. The different functions are: `createRandomRandom()`, `createRandomFromSaved()`, `createRandomFromSavedMorph()`, `createRandomProcedural()`.

The `createRandomRandom()` function was the first method to be created. It uses maximum and minimum joint angle values, defining the Range of Motion (ROM) [**apostolico2014postura**]. For each individual finger an individual flexion and abduction within the predefined ROM is computed.

In the `createRandomFromSaved()` a hand posture is composed by randomly picking finger postures from the training data. For every finger a random TrainingUnit is picked from the PostureDataHandler and that specific finger is taken over by the new random hand. This way, many interesting hand posture combinations can be created of the 10 different training data postures.

`createRandomFromSavedMorph()` also works using the training data. For the creation of a random hand, two random `TrainingUnits` are again picked. The resulting hand is created, by interpolating the two hand postures, with the weight varying between the fingers.

Finally the `createRandomProcedural()` had a set of predefined finger states, similar as described in the paper of Su et al. [**su1994logical**]. For the thumb the states `Under`, `Downward`, `Aligned` and `Sideways` were defined, the remaining fingers had the possible states `HyperExtended`, `Flat`, `Fist` and `ForwardFist`. Each of the hand's fingers is assigned one state randomly and a random global abduction value is generated, defining the total fanning of the fingers, before assembling the actual hand posture from predefined finger configurations corresponding to the finger states.

To guarantee a homogenous distribution of comfort and discomfort among the randomly generated hand postures, a random range of possible naive comfort/discomfort metric values for the hand posture is defined beforehand. The random hand generation process is repeated until the calculated naive metric value of the hand posture lies within the specified range.

4.6 User Study Tests

For investigating the main questions of the thesis, a test environment had to be created. In order to verify and to improve the metric, the idea was to let participants rate different hand postures and compare the ratings with the computed metric values. Additionally, the objective was to show the impact of comfort and discomfort on precision and performance in different tasks, for that a target shooting test and a line tracing test were implemented. The user study was designed for a seated participant in front of a desk, with the elbow of the dominant hand resting on the edge. For interaction a controller and a monitor were placed on the desk. Additionally a Leap Motion controller was attached to the desk.

4.6.1 Hand Posture Evaluation

For the hand posture evaluation, a `UserStudyComfortEvaluation` scene was created. To begin the test, a random hand posture was generated and visualized using the `OutputHand` known from the `TrainingManager` scene. Afterwards the participant had to imitate the hand posture and rate the hand posture on an intuitive comfort/discomfort scale ranging from 0 (extremely uncomfortable) to 10 (extremely comfortable). If the participant was unable to mimic the hand posture due to its complexity, the posture was supposed to be rated with a 0. After confirming his or her choice, the results were logged to a .CSV file. For that, the naive and improved metric components were computed and logged together with the participants rating and name. In addition the whole randomly generated hand was logged, in order to be able to do further calculations later on if necessary.

Before the first hand posture evaluation the participant was shown two different randomly generated hand postures, to get a reference on what to expect in terms of comfort and discomfort. The first hand posture was supposed to be a relaxed one that could be rated with a 10, therefore with a metric value close to 0. The second hand posture should represent an extremely uncomfortable hand posture resulting in a 0 rating, therefore the metric value should be as high as possible.

4.6.2 Precision and Performance Tests

Before the target shooting and the line tracing test. The participant was again shown a randomly generated hand posture, that was meant to be tested. Again the user had to mimic the hand posture with the dominant hand and again the user had to rate the hand. After rating, the subject had to hold his hand over the Leap while holding the posture. In order to track the hand posture during the tests, the hand posture had to be learned by the k-NN. For that the Leap recorded 50 samples of the hand and the hand posture data was saved as `TrainingUnits` by the `PostureDataHandler`.

After that the actual tests started. If the participants broke the hand posture they were kindly asked to correct it. During both tests the participants were shown a visualization of the given hand posture in the bottom right corner as a reminder. They had to perform different tasks using their dominant hand while maintaining the hand posture. For that, the subjects were shown a minimalistic representation of their hand position that was recorded with the Leap. In favor of comparability, the pointing direction was universally set independent of the hand posture as the forward direction of the hand base, and was visualized with a laser beam.

4.6.3 Target Shooting Test

The target shooting test was chosen to test the performance impact in context of quick punctual pointing. This specific test was chosen due to its similarity to real world tasks, such as giving a robot quick spatial commands or commanding units to perform some action somewhere in space in a RTS game. To implement this, the `UserStudyTargetShooting` scene was created.

The scene featured an `OutputHand` visualizing a total of 18 targets which appeared individually in a random order. The participant then had to aim down the individual targets with their hand and shoot them with the press of a button. Once a target was hit, the distance of the hit to the target's center was computed as a metric for accuracy. The accuracy value, the time to hit as well as the different metric components and the whole hand were then logged in a .CSV file.

When the participants broke the hand posture, shooting was locked, until the hand posture was corrected.

4.6.4 Line Tracing Test

The line tracing test was chosen due to its similarity to real world tasks such as drawing a route to be traversed by a robot or patrolled by units in a RTS. For this, the `UserStudyLineTracing` scene was created.

In the test, the participant was shown a blue curve on a drawing plane, that had to be traced. For that the user had to use the ray, to draw his own line on the drawing plane as close to the given curve as possible by holding a controller button. If the participant broke the hand posture, the drawing was interrupted, if the user released the drawing button, the test was ended after a certain timeout. After the test was ended, the total accuracy was calculated and was logged together with the total task completion time, hand posture metric values and the whole hand in a .CSV file.

The curves were implemented as discrete sets of `LinePoints` that were connected to their neighbors with line segments. In order to get a smooth curve 12 random points were generated on the drawing plane and a bezier curve was fitted through them. The bezier curve was then discretized in 75 `LinePoints`. For drawing the line, a `LinePoint` was set at the starting point and a second was moved with the ray on the drawing plane. As soon as the second `LinePoint` moved too far away from the previous `LinePoint`, it was fixed to its position and a new `LinePoint` was added in and moved further instead.

The accuracy could be determined by calculating the average minimum distance of every drawn `LinePoint` to the given curve defined by its `LinePoints`. The same was done the other way around and added up to the total line accuracy value.

4.6.5 The User Study Management

In order to automate the user study process, a `UserStudyDevScreen` scene was added in. The scene showed a GUI, that can be used configure a whole user study for one participant. The experimenter can set the name of the participant, his handedness and choose a `Pose` to be overwritten by the users hand recordings. Further it can be selected which of the three user study test have to be performed and how often these will be repeated for different hand postures.

After confirming the setting, the user study will start with the desired number of hand posture evaluations and automatically switch to the precision and performance tests. After finishing the complete user study, the participant is shown a ending screen, thanking him for participating in the study and giving him the opportunity to play around with the standard Leap hand visualization.

4.7 Stumbling Blocks

During implementation, a variety of unexpected problems occurred and different concepts turned out to not work in the context as expected. The following section describes different stumbling blocks that can commonly be found in similar contexts or were severe threats to the project.

4.7.1 Serialization and Normalization of Vector3 and Quaternions

Since a `BinaryFormatter` was used for saving and loading the training data in the `PostureDataHandler`, all saved objects had to be serializable. For this `TrainingUnit` had to be serializale, consequently a `AngleBasedHandModel` and a `AngleBasedFinger-` and `-ThumbModel` had to serializable. As `floats` and `Enums` are both serializable, the only problems were `Vector3` and `Quaternion`, the standard Unity classes, that for some reason are not serializable. The only solution for this was to create a custom struct for vectors and quaternions.

Thus, the `sVector3` and `sQuaternions` were created. Both `structs` basically implement their unserializable counterparts, in both cases operators were defined to enable implicit typecasting to the unserializable form and back. For logging, both `structs` also received additional functions for printing and parsing into .CSV files.

When extracting the MCP and TMC angles (Figure [**fig:anatomytotal**]) of the fingers, it was necessary to compute the relative rotations of the finger joints compared to their parents. The first idea, was to use the different functions of the Unity `Transform` class, but it turned out, that this class was unable to be instantiated without being a component of a game object. Therefore direct quaternion multiplication was used, as defined in the standard Unity class. After a while it became clear, that the standard quaternion multiplication would not automatically normalize the result, which caused several bugs when trying to do further computations with the `Quaternions`. Obviously the standard unity functions do not check if a `Quaternion` is normalized before doing further computations.

The bug was even harder to track, as it could not be visualized due to the `Transform` class automatically normalizing a `Quaternion` when applied to it as rotation.

As a consequence of this, the `sQuaternion` also received a function for normalization and length computation, as the standard `Quaternion` class implements neither.

Finally, creating a ambidextrous hand posture classification environment required to be able to transform a left handed hand posture to a right handed one and vice versa. Thus, functions for mirroring `sQuaternions` around different axes had to be implemented as `Quaternion` did not provide any.

4.7.2 Leap Tracking

The Leap Motion Orion SDK was a big step forward for the tracking of the Leap. It enabled tracking of awkward postures such as pinching, that were not possible to track before, it improved the finger pose estimation for occluded fingers and overall massively improved the range and robustness at different lighting conditions. However, due to its intended field of use, the Leap tracking is focused on quick and fluent hand and finger tracking, that provides the precision needed to create a reasonable visual parity. However, for contexts that require high precision, the Leap struggles to deliver usable data.

While this characteristic of the Leap was no problem for the hand posture classification, it lead to unacceptable impact for pointing, especially for unconventional hand postures. Using the ray to aim down targets or to trace a line magnified the inaccuracies of the hand tracking, resulting in accuracy loss in the user studies. Because the tracking inaccuracies exceeded the expected accuracy loss by hand postures, an alternative hand tracking solution had to be found.

For precise hand tracking a optical outside-in solution was used, the ART system. Using four cameras the system can detect a marker attached to the back of the user's hand. By attaching a marker to the user study desk, the ART hand position could be calibrated to the same coordinate system as used by the leap. The tracking data of the ART was implemented to only be used for hand rotation and position in the target shooting and line tracing task.

The result was a much higher tracking precision that allowed meaningful user accuracy observation.

However the need for the user to wear a hand tracker yielded yet another problem. The tracker's black rubber strap, wrapped around the user's hand was visible for the Leap. In some cases this inferred with the hand detection, causing the Leap to detect a small closed fist in the lower stump of the hand. However, this problem could be solved by wrapping occluding the strap with something with a similar IR signature as skin. For the user study this was realized by wrapping a paper towel around the strap.

4.7.3 Hand Posture Detection

The hand posture detection as implemented by the k-NN algorithm was supposed to differentiate between a relatively small set of predefined hand postures. Through the addition of the random hand generator, samples of random hand postures were added with a "custom" `Posture` to the training data set. Problems occurred, when the randomly generated posture was similar or identical to a predefined hand posture by coincidence. This caused problems in the user studies, when the predefined posture was detected instead of the custom one and the user was notified to correct his hand posture even though he had the right hand posture.

As a consequence the misuse of the k-NN was ended, instead only the minimum Euclidean distance to the "custom" `TrainingUnits` is computed, similar as it is computed for the RRP component. If the minimum distance is smaller than a certain threshold, the hand is considered to hold the hand posture.

5 User Studies

A total of two user studies were conducted. The user studies were held in the Games-Lab on TUM Campus Garching. The participants were compensated with beverages and/or sweets. Beforehand the participants were informed about the aim of the study as well as their specific objectives.

5.1 First User Study

aa

5.2 Early Analysis and its Consequences

5.3 Second User Study

6 Results and Discussion

6.1 Results

aa

6.2 Discussion

List of Figures

List of Tables