# Avatar: Inverse Kinematics for Hand Control

**Lab Report: Lab Cognitive Robotics – Institute for Compute Science VI**
**Autonomous Intelligent Systems, 2020**

Marie-Theres Schier, Baraa Hassan,
Under Supervision: Chrisitian Lenz

An Avatar Robot is a robot, which is being controlled from a remote location using a human operator, such that it can be used to reach dangerous areas, that are unsafe for human beings. Furthermore, it can be used for remote sensing, to let humans be able to sense a remote environment based on the robot's hardware and sensors. By allowing control over the robot hand fingertips, the person who is manipulating the Avatar Robot acquires a lot of controlling power upon the remote location. In this lab report, we present our work on bringing the movements of the robotic hand closer to the controlling hand movements made by the human operator.

## 1 Introduction

In a situation like the current time - Corona Pandemic time - the need for an Avatar Robot - to do some tasks contact-less - is increasing, because there is no guarantee that doing these tasks in the current time in a normal way - by contacting other people - is safe.
For instance, doctors and medical staff inside a hospital could interact with Corona patients remotely through an Avatar Robot, without being at the risk of getting infected. Additionally, products, medication, and food can be delivered using an Avatar Robot and in this way contribute to eliminating the contact between the delivery person and the client to further suppress the spread of the Pandemic.
Sequentially, the need for controlling the Avatar Robot's hand fingertips is growing up as well, since this functionality is included in most of the tasks that use the advantage of having an Avatar Robot.
Therefore, our project focus was controlling the Avatar Robot's hand fingertips, to ease the life of humanity and to participate - with what we have learned - in solving issues caused by the current Pandemic. Accordingly, we employed the Inverse Kinematics to attain our goal.

# 2 Hardware

Our Hardware is composed of a Sense Glove, a Schunk Hand, and 2 UR5e arms. The Sense Glove is held by one UR5e arm and connected to our software. The Schunk Hand is lifted by one UR5e arm and receives its commands through communication with the software.

## 2.1 Schunk Hand

The Schunk Hand (fig. 1) has 9 DoF, so there are only 9 active joints that control the motion of the 5 fingertips of the hand, while all the other joints are just mimicking those 9 DoF joints, to achieve the goal pose of the fingertips.

Figure 1: The Schunk hand [2].

## 2.2 Sense Glove

The Sense Glove(fig. 2) has 20 DoF, such that there are 20 active joints that one can move freely, which means we need to map this flexible moving hand to the strict robotic hand (20 DoF to 9 DoF).

Figure 2: Each finger has 4 active joints.

# 3 Problem Definition

Our main goal was to able to successfully control the Schunk Hand with the Sense Glove in such a manner, that a specific finger pose, done by the human operator on the Sense Glove, is being mirrored by the Schunk Hand.
Especially difficult poses are finger poses where the fingertips come close together, such as trying to pinch a small object between thumb and index finger, because this requires that the resulting fingertip positions have to be calculated correctly. If that would not be the case, the fingertips would not come close and therefore the Schunk Hand would not be able to pick up a small object (e.g. a chess figurine).

## 3.1 Problem Challenges

One challenge was the discrepancy in movement ability between Sense Glove and Schunk Hand: We want to - precisely - control the Schunk Hand, which has only 9 DoF, with the Sense Glove, which has 20 DoF.
Therefore, the challenge was to find a correct mapping, so that the relative fingertip position of the glove matches that of the robotic hand, despite the mentioned difference in the number of active joints.
Because of different link lengths of the two hands, it is not enough to simply map the respective joint angles onto the Schunk Hand.
Another challenge we faced, was the existence of mimic joints in the Schunk Hand, although consisting of only 9 active joints - which can be directly controlled by sending angle values to the respective joints - the Schunk Hand actually includes more movable joints, which are only mimicking other joints. This complete kinematic model can be seen in detail in figure 3.

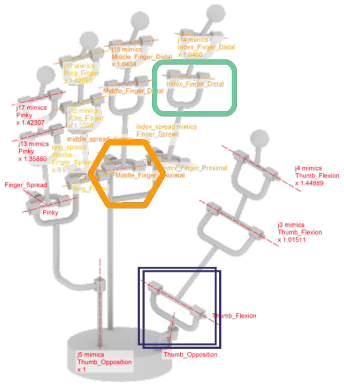

Figure 3: The joints with colored shapes are examples of the active joints (9 DoF joints) and the joints above them are the joints mimicking them [2].

### 3.2 Our Solution

In our solution, we started by ignoring the mimic joints in our Inverse Kinematics logic, but the result was far from what we were expecting. Then, we tested a new solution by taking the average value for the active joints according to the fingertip and other mimic joints, that are mimicking these active joints, as explained in the example below. We will talk about the resulted performance in details in the performance section.

**For example:**
let **X1** be an active joint, **X2** and **X3** joints that mimicking **X1**(their poses depend on **X1**'s pose):

X1 has value according to the fingertip pose (Active Joint)
$X2 = offset + X1 * factor$ (Mimic Joint)
$X3 = offset + X1 * factor$ (Mimic Joint)

$X1_0 = IK(fingertippose)$
$X1_1 = (X2 - offset)/factor$
$X1_2 = (X3 - offset)/factor$
Then: $X1 = (X1_0 + X1_1 + X1_2)/3$

## 4 Inverse Kinematics

In order to tackle the challenge described in the problem definition section, we used Inverse Kinematics and gradient descent. Figure 4 shows an exemplary kinematic chain, which consists of links $L_i$ and joints $q_i$ (with $i \in \{1, 2, 3, 4\}$) and the end-effector - the part, which is interacting with the environment.
The basic idea of Inverse Kinematics is to calculate the joint angles $q_i$ which leads to a given end-effector pose $(x, y, \theta)$; therefore being the direct opposite of Forward

Kinematics, where the goal is to compute the end-effector pose according to given joint angles.
As it can be easily understood, Inverse Kinematics turns out to be more difficult, because given the end-effector pose, several joint angle configurations are possible, which lead to that specific pose.

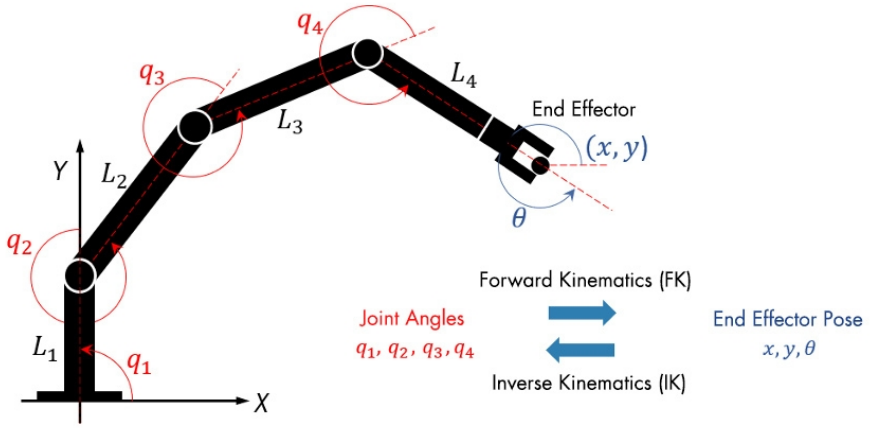


Figure 4: Kinematic model consisting of links $L_i$ and joints $q_i$; with an explanation of Forward and Inverse Kinematics [3].

### 4.1 Employing IK in our Solution

For our work, we used IK to calculate the joint angles for the Schunk Hand. Given the goal fingertip positions by previous FK computation on the Sense Glove input, we calculate the Jacobian matrix for each finger separately. Then we take the pseudo-inverse of the matrix to further compute a small change in joint angles, so that the fingertip of the Schunk Hand further moves towards the goal fingertip position.
In a loop, we apply these small changes to the joint angles, and repeatedly checking if the current pose and the goal pose are close enough.

## 5 Software

### 5.1 Software's Pipeline

The pipeline (fig. 5) starts with the Sense Glove doing an action with the fingertips (evoked by the movement of the human operator). Second, the Hand Control software computes the fingertips poses (the Forward Kinematics) for the Sense Glove. Third, the software computes the Inverse Kinematics for the Avatar Robot's hand fingertips. Finally, the software sends the command to the Avatar Robot's hand joints to mimic the motion of the Sense Glove action.

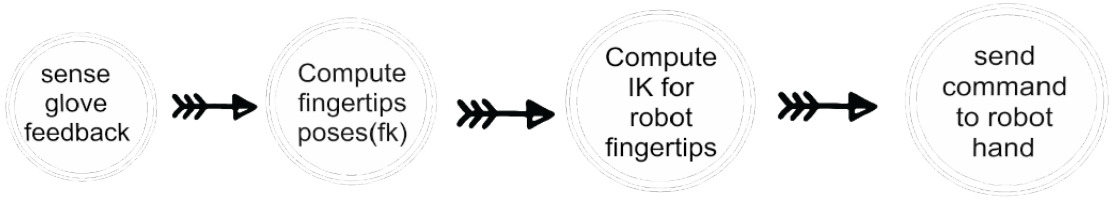


Figure 5: Software's pipeline.

## 5.2 Used Libraries

### 5.2.1 ROS

The Robot Operating System (ROS) framework is a framework to ease the development of robotic software. It is the main framework that our code uses. It is playing the role of an operating system - but it is not an actual operating system - where our Hand Control robotic software is being built and run using its set of tools and libraries.

ROS processes are represented as a graph that includes mainly Nodes (graph's nodes), Topics (edges), and other components. Nodes act as the running process inside ROS. They are the logic part, in where we implemented the Hand Control and the Inverse Kinematics logic. They are the executable files that interact with each other by exchanging data - called messages - anonymously, where no node knows the source or the destination of those messages.
The main node that initiates the ROS framework is called the "roscore" node. To be able to use the ROS framework and run your executable nodes, you should run the "roscore" node at the beginning.

Topics are communication channels between Nodes to share or receive data. The name of the topic is unique in the same namespace. Hence nodes can receive messages shared by other nodes with listening/subscribing to a certain topic. Additionally, they can share messages to other nodes by publishing/sharing to a certain topic [9].

### 5.2.2 Moveit

The Robot Model is formed of links and joints connecting these links. The model is being described in detail with information about the position, rotation, etc. in the Unified Robot Description Format (URDF) file.
The model has a reference frame for the whole model (in our sense glove model is the frame for the link named base_link) and there is a frame for each link and each joint. The reference frame is the coordinates that explain the transformation of all the links and joints according to the same point, other local frames are coordinates that explains the transformation of the links and joints locally according to itself(its previous

transformation).
MoveIt Is the library used for Motion Planning, Manipulation, Inverse Kinematics, etc. It contains classes that represent the robot model and the robot state, in addition to functions that are used for Motion Planning, Manipulation, Inverse Kinematics, and other tasks.
Moveit reads the URDF file and offers you data structures that contain data about the connected links and joints, and about those frames in the current state, then by using Moveit you are able to compute the fingertip new transformation according to given changes in the joint angles(Forward Kinematics) or to compute the change in the joint angles according to the given change in the fingertip transformation(Inverse Kinematics)[4].

## 5.3 Used tools

### 5.3.1 catkin

Is the build tool for ROS packages, which uses Cmake.txt and package.xml files as the package metadata. In case you want to run a node(executable file) in your software - that depends on ROS framework -, then you need to build the package containing this node using the Catkin tool[1].

### 5.3.2 roslaunch

A tool that offers you launching a launch file - with one command - that may contain one node or more or another much complex lunch script[6]

### 5.3.3 rosmon

A tool that offers the same as roslaunch but with a clean and modern console UI[7].

### 5.3.4 rosbag

Is a command-line tool, that gives you the ability to record and playback data. It gives you the ability to record the messages in topics that are published by current running nodes, as well it offers you playback this file - from the command-line - and those data will be published to their target topics for other running nodes, without the need of running any extra nodes to publish this data. The data are being recorded/playback in a file with bag extension[5].

### 5.3.5 rviz

Is a tool for visualization purposes it contains many plugins that are used to visualize and simulate your model in 3D[8].

### 5.4 Our Contribution

1. Creating the Sense Glove model.
2. Creating the simulation file.
3. Controlling the hand by mapping the joints.
4. Pipeline with the Sense Glove's forward kinematics.
5. Inverse kinematic computation on the Schunk Hand.
6. Computing the performance of the Hand Controller.

### 5.5 Software Topics Graph

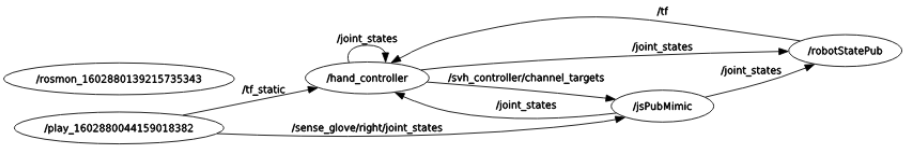


Figure 6: Software's Topics Graph.

## 6 Results

The photos in figure 7 display our results. They show how we were able to control the robotic hand with the Sense Glove. As can be seen, it was possible to pinch/grab objects, such as a charger plug and a chess piece, noting that it was - as can be expected - easier to grab a rectangular shape (such as the plug) than a small round object.
Figure 8 additionally shows the models of the Sense Glove and the Schunk Hand, also demonstrating how the Schunk Hand reacts according to the initial Sense Glove movements.

## 7 Performance

| After 200 iterations of the bag file | | | |
|---|---|---|---|
| Type of Error | Thumb | Index | Middle |
| Min. Deviation | 0.000217967 | 0.000743657 | 0.000694107 |
| Avg. Deviation | 0.00338044 | 0.00247985 | 0.00191389 |
| Max. Deviation | 0.01 | 0.00598231 | 0.00655697 |

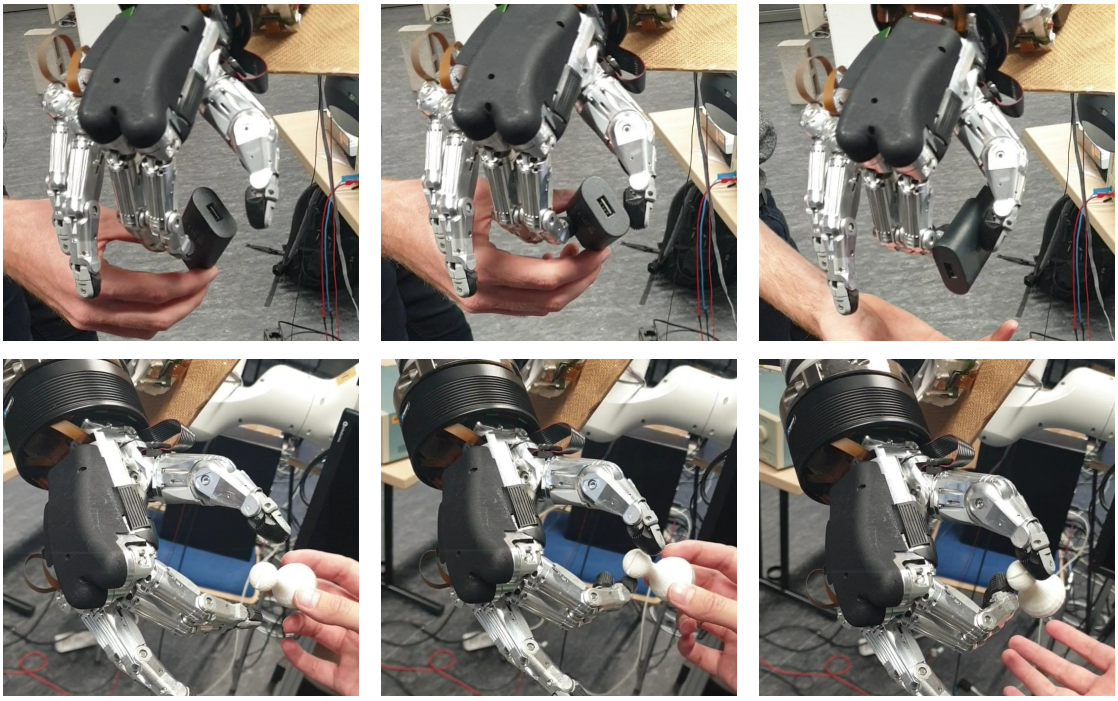

Figure 7: Taken photos that show the result of our work: The first row shows the robotic hand grabbing a charger plug. The second row shows the robotic hand pinching a chess piece. The hand is being controlled by the movements with the Sense Glove.

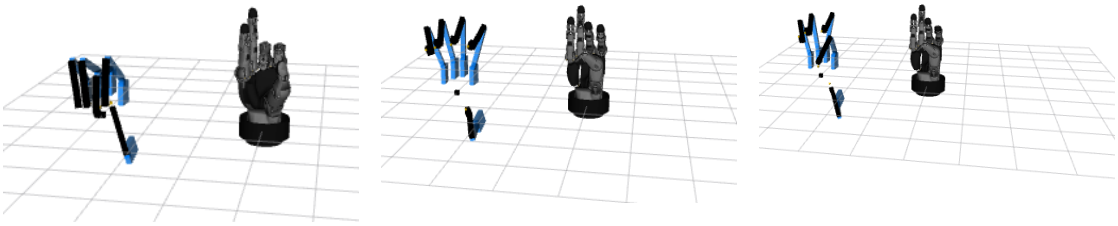

Figure 8: The pictures show the Sense Glove model in blue and the Schunk Hand model in black. We used these models to check how the glove movements transfer onto the robotic hand.

## 8 Conclusion

We were able to successfully employ Inverse Kinematics to calculate the fingertip positions for the Schunk Hand. With the working model, we could transfer this into the real world and could move the Schunk Hand with the Sense Glove. We managed to pinch objects between thumb and index finger, although only for a rather short amount of

time and with some effort on the side of the Sense Glove.
An existing problem at the current time is the difference in the workspace between the Sense Glove and the Schunk Hand, which also caused the described difficulties in easily picking up small objects.
There are still positions that can be reached with the Sense Glove, but not by the Schunk Hand, because the two hands differ in regards to their link lengths and also the number of (active) joints.

## 9 Future Work

Future work includes facing the problem of the two different workspaces, which has been addressed in the last section. To solve this issue, we would need to further improve the mapping from the sense glove values to the ones on the Schunk Hand.
For that, the workspaces of both hands have to be explored in a quantitative way, so that the respective limits can be taken into account when calculating the Schunk Hand angle values.

## References


[1] catkin tool. URL `http://wiki.ros.org/catkin`.

[2] Schunk Robotic Hand. URL `http://wiki.ros.org/schunk_svh_driver`.

[3] Inverse Kinematics. URL `https://de.mathworks.com/discovery/inverse-kinematics.html`.

[4] Moveit library. URL `https://moveit.ros.org/`.

[5] rosbag tool. URL `http://wiki.ros.org/rosbag`.

[6] rosluanch tool. URL `http://wiki.ros.org/roslaunch`.

[7] rosmon tool. URL `http://wiki.ros.org/rosmon`.

[8] rviz tool. URL `http://wiki.ros.org/rviz`.

[9] Stanford Artificial Intelligence Laboratory et al. Robotic operating system. URL `https://www.ros.org`.