

Robust Human Pose Estimation using multiple multimodal visual sensors]Robust Human Pose Estimation using multiple multimodal visual sensors

## **Abstract**



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Research question . . . . .	3
1.2	Process Pipeline . . . . .	5
<b>2</b>	<b>Data Processing</b>	<b>7</b>
2.1	Developed Software . . . . .	9
2.2	Stream pre-processing . . . . .	11
2.2.1	Multiple Cameras . . . . .	11
2.2.2	Recording session set-up . . . . .	11
2.3	Data acquisition . . . . .	15
2.3.1	Data format . . . . .	15
2.3.2	Recording process . . . . .	19
2.4	Data population . . . . .	21
2.5	Data Evaluation . . . . .	23
2.6	Data Augmentation . . . . .	25
<b>3</b>	<b>Model development</b>	<b>27</b>
3.1	Model training . . . . .	29
3.2	Model evaluation . . . . .	31
<b>4</b>	<b>Experiment</b>	<b>33</b>
<b>5</b>	<b>Results</b>	<b>35</b>
<b>6</b>	<b>Conclusion</b>	<b>37</b>
6.1	Future work . . . . .	39



# List of Figures

1-1	Process Pipeline with all steps . . . . .	5
2-1	FESD GUI . . . . .	9
2-2	Pinhole camera model . . . . .	16
2-3	Recording GUI marked in Red . . . . .	19



# List of Tables





# Listings

2.1	Example of session metadata . . . . .	16
-----	---------------------------------------	----



# Chapter 1

## Introduction

- Introduce human pose estimation
- Introduce SilverFit and their goal
- Introduce the problem (rewrite research question)

The code of this thesis is available on GitHub <sup>1</sup>.

---

<sup>1</sup><https://github.com/LeonardoPohl/FESD>



## 1.1 Research question

In this thesis, we try to find out if it is possible to develop a method that can tell if a joint produced by human pose estimation is potentially faulty. We try to achieve this by building a dataset using RGBD cameras from different angles.

This project includes both the creation of the dataset as well as the data population with estimated human pose data and the training and evaluation of a model for fault estimation. The dataset contains different challenging scenarios, which make human pose detection more error-prone. These challenges include but are not limited to; lighting, background, clothing, accessories attached to the wrist and ankles, and proximity of the limbs to objects. We found that these have the largest effect on the performance of human pose estimation.



## 1.2 Process Pipeline

The whole process of fault estimation can be seen as a pipeline. We start at the most basic starting block, the camera streams, and end at the most complex block, the fault estimation. The pipeline is shown in Figure 1-1. The pipeline consists of seven steps, which are described in more detail in the following sections. The steps are **(I)** Stream Pre-Processing, **(II)** Data Acquisition, **(III)** Data Population, **(IV)** Data Post-Processing/Evaluation, **(V)** Data Augmentation, **(VI)** Model Training, and **(VII)** Model Evaluation. The results of each step are used as input for the next step.

We further divided the process into a data processing and model development phase. The data processing phase is the first five steps of the pipeline. The model development phase is the last two steps of the pipeline.

In the next chapters, we give a basic overview of the whole process. We go into more detail in the following sections.

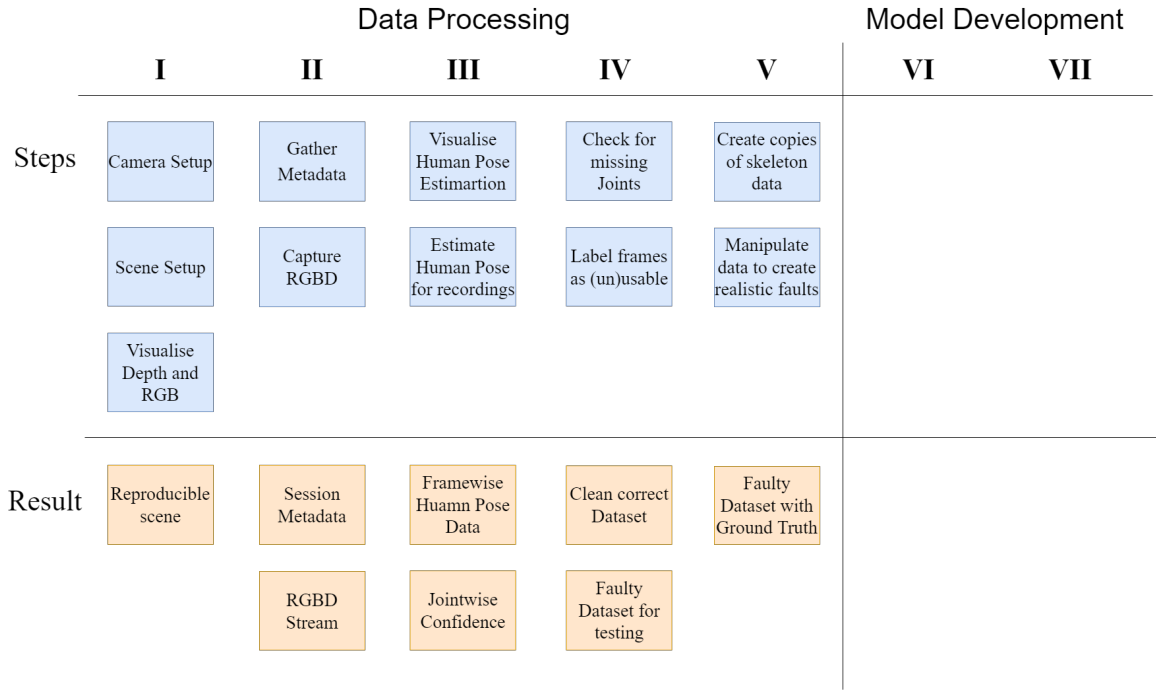


Figure 1-1: The whole Process pipeline with all the steps, which are marked in blue, and the results of each step, which are marked orange. The results of the steps are used as input for the next step. The steps are described in more detail in the following sections. The steps are: **(I)** Stream Pre-Processing, **(II)** Data Acquisition, **(III)** Data Population, **(IV)** Data Post-Processing/Evaluation, **(V)** Data Augmentation, **(VI)** Model Training, and **(VII)** Model Evaluation.





## Chapter 2

# Data Processing

In this chapter, we discuss the data processing steps that are required to prepare the data for the model development process. Firstly, we address different session parameters and how they might influence the data. Secondly, we explain the data acquisition process and how the data is stored in files such that it can be used in the future. Thirdly, we discuss the data population process in which the human pose data is extracted from the raw data. To ensure the quality of the dataset we then evaluate the data by filtering invalid skeletons and marking data points as valid or invalid. Finally, we discuss the data augmentation process in which the data is augmented to increase the size of the dataset.



## 2.1 Developed Software

*UNSURE Should I write about the software, explain the OpenGL implementation, the ImGui GUI and so on? TODO Change Screenshots to light mode to be consistent with the rest of the thesis (can wait until screenshots are final)*

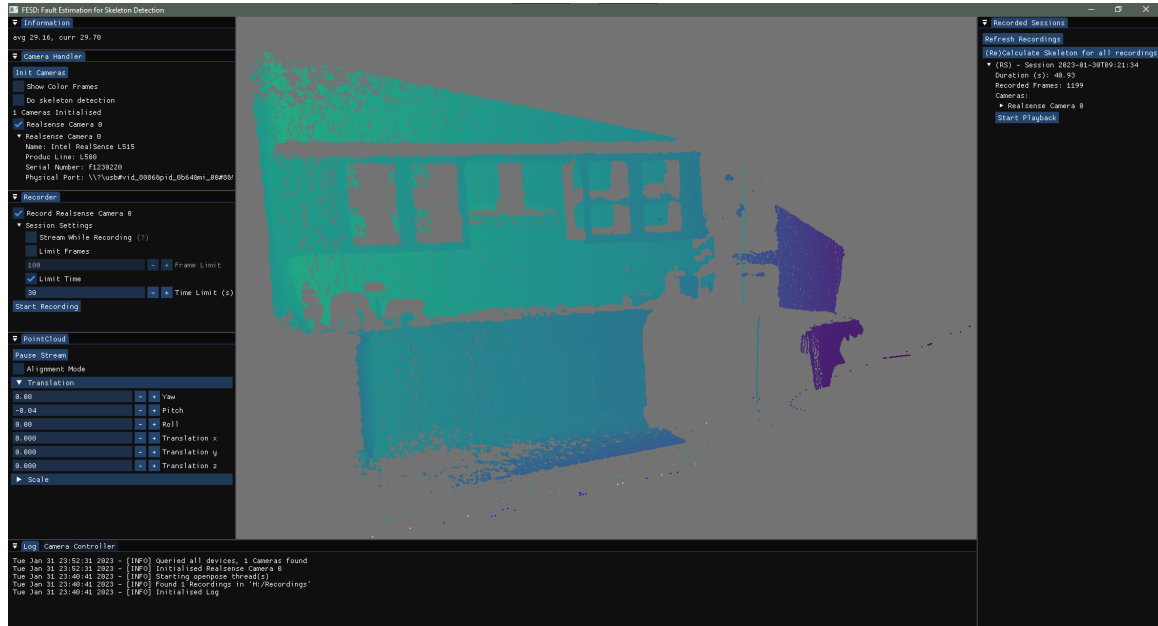


Figure 2-1: A screenshot of the FESD GUI streaming a point cloud. The GUI is used to record and visualize the data, and to playback recordings to validate the data. The GUI is written in C++ using the ImGui framework. The Pointcloud is visualised using OpenGL and a glsl shader.



## 2.2 Stream pre-processing

To get the best possible results we need to make sure that the cameras are set up in exactly the intended way.

### 2.2.1 Multiple Cameras

We use multiple cameras to increase the accuracy of the results. We use two cameras to record the same scene from two different angles. This way we can compare the results from the two cameras and make sure that the results are consistent. We also use multiple cameras to record the same scene from different heights and angles.

***UNSURE** Should I write about it if Im not going to do it? Its quite interesting how the synchronisation might work and how the pointclouds can be synchronised. I already did a lot of research on it but if Im not going to implement it then this might not be the best point to do it.*

### 2.2.2 Recording session set-up

We consider different environmental setups to increase the significance of the results. The following session parameters are considered:

#### Lighting

RGBD cameras function with infrared light therefore is the lighting of a scene essential. We found that direct sunlight interferes with some RGBD cameras more than others based on the infrared range that is used. Since the exact sunlighting is not controllable we choose to make it as optimal as possible to improve reproducibility. Therefore, we choose a room with no sunlight but we do include artificial light to reduce any damage that might occur to visibility issues.

#### Relative Camera Position

At SilverFit, cameras are attached above a screen at a height of 180cm facing downward at around 20 deg. To form a more general model, we will experiment with different setups and angles. We experiment with six different setups in total. Three setups from different

angles (20 deg, 0 deg, 340 deg) at two different heights (180cm, 120cm). The different setups can be seen in figure TODO.

**TODO** *Add figure with different setups.*

**UNSURE** *We Develop a functionality that lets us determine the exact height and orientation of the camera. We do this by detecting the floor and thereby calculating the height of the camera and the angle at which it is pointing downward. We can also detect if the camera is not completely straight and therefore might influence the results.*

## **Sitting or standing**

From experience, we know that detecting the joints correctly is influenced by the position of the participant. This is especially true for the difference between a sitting and a standing patient. Human pose detection is in general more reliable if the patient is standing, due to reduced occlusion. We record each scenario sitting and standing.

## **Clothing and ankle and wrist attachments**

Clothing can have a similar effect on the efficacy of HPE as lighting. If the participant is wearing black pants infrared light will be absorbed rather than reflected leading to 'blind spots' in the legs. Since the legs are already more unreliable than the rest of the body, these blind spots can negatively affect HPE.

Since SilverFit develops games for rehabilitation, the supervising physiotherapist might choose to attach weights to the ankles and/or wrists to increase the effectiveness of the exercise. We therefore also include attached and held weights to simulate difficult situations.

## **Background**

The background of the scene can have a significant effect on the results. We, therefore, record the same scenario with and without a visible background, i.e. a wall is behind the participant or there is no wall within the maximum sensor range (6m).

## **Crampedness of the Environment**

The Crampedness of the scene increases the number of false positives of HPE. We, therefore, record the same scenario with and without clutter. We consider clutter to be any object

that is not a part of the participant’s body. However, clutter is quite objective and therefore we will not be able to define it in a universally applicable way.

### **Distance to the camera**

Games developed by SilverFit have a calibration step where the participant is asked to stand at a certain distance from the camera. We, therefore, record the scenario at that specific distance. This ensures that noise introduced by the depth sensor has little effect on the results. The participant is positioned 2 meters away from the camera. *UNSURE, ask someone at SilverFit.*





## 2.3 Data acquisition

The second phase of the pipeline is data acquisition. After we have set up the camera according to the session parameters we can start recording. We set a timer for 30s and record the RGB and Depth streams from the camera. We also record the camera intrinsics, which we will use to recreate the point cloud at a later stage. We record the data in a folder structure that is defined by the session parameters.

### 2.3.1 Data format

An important part of data acquisition is the description of the way the data is stored in the file system. This is essential for any future use of the data and therefore we need to make sure that the data is stored in a way that is easy to understand and easy to use. We store in general two to five files per session depending on the camera configuration.

#### Session Metadata

Every session contains a "SESSION\_NAME.json" file that contains the session parameters and camera metadata. The session name is automatically generated based on the starting time of the session, this way we can make sure that the session name is unique every time and we can also have an idea of which recording is the most recent without looking at the contents of the file.

**Camera metadata** The camera metadata contains the camera intrinsics, which we will use to recreate the point cloud at a later stage. The camera intrinsics are the field of view of the depth camera in the horizontal and vertical direction, and the principal point in the horizontal and vertical direction. The field of view is the angle between the optical axis and the image plane. The principal point is the point in the image where the optical axis of the camera intersects the image plane. The principal point and the field of view are explained in Figure 2-2.

**Camera orientation** Additionally, to the camera intrinsics we store the relative rotation and translation between the cameras if multiple cameras are used. The rotation and

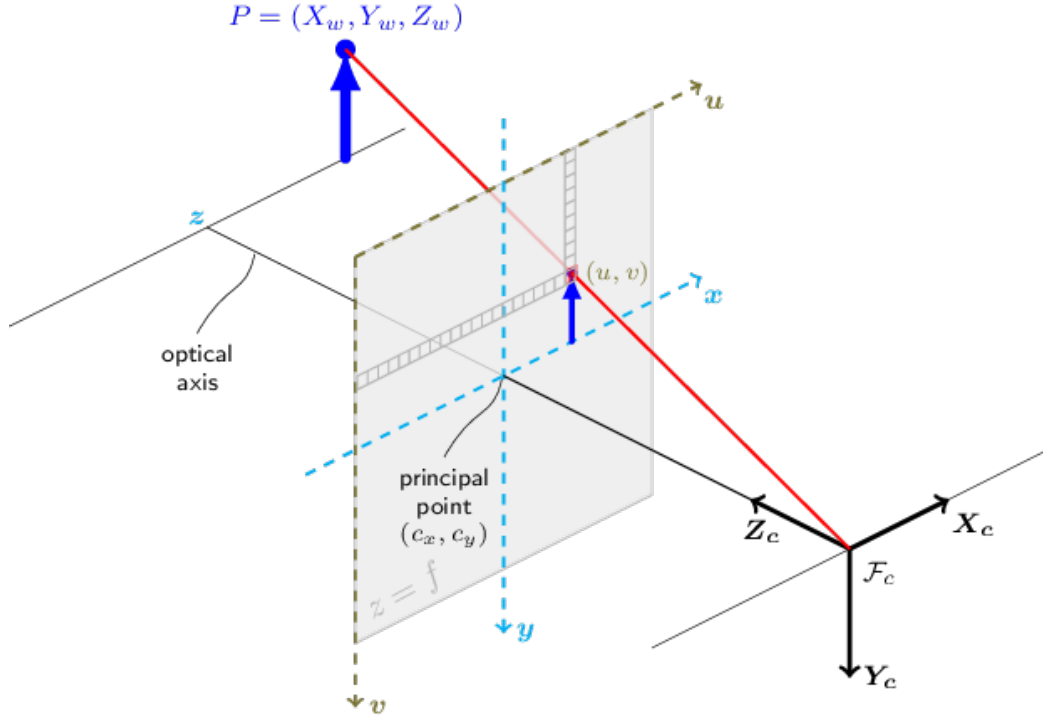


Figure 2-2: The pinhole camera model showing the principal point. The principal point is the point where the optical axis intersects the image plane. The field of view is the angle between the optical axis and the image plane.

translation are stored as Euler angles<sup>1</sup> and a vector respectively [1]. The rotation is the rotation of the camera relative to the second camera in the system. The translation is the translation of the camera concerning the second camera in the system.

**Session parameters** The session parameters are the same as the ones defined in the section "Stream pre-processing". The user enters the session parameters before starting the recording. Most of the parameters are boolean values that indicate whether the user is sitting, wearing dark clothing, etc. The height and angle parameters are the height of the camera to the floor and the angle of the camera relative to the orientation of the user as explained in the previous section.

An example of the Session metadata can be seen in Listing 2.1.

Listing 2.1: Example of the Session metadata with a single Realsense Camera which was recorded for 40 seconds at around 30 frames per second resulting in 1200 frames. Some

<sup>1</sup>Technically we are storing the rotation with the Tait–Bryan notation, i.e. x-y-z or yaw-pitch-roll, rather than the classic Euler notation. However, the name Euler angle is more commonly used and understood.

values have been changed to increase readability.

---

```
1 {
2   "Cameras" :
3   [
4     {
5       "Cx" : 314.26,
6       "Cy" : 239.46,
7       "FileName" : "Session_2023-01-30T09.21.34_Realsense_Camera_0.bag",
8       "Fx" : 459.77,
9       "Fy" : 459.83,
10      "MeterPerUnit" : 0.00025,
11      "Name" : "Realsense Camera 0",
12      "Type" : "Realsense"
13    }
14  ],
15  "DurationInSec": 40.0,
16  "Name": "Session 2023-01-30T09:21:34",
17  "RecordedFrames": 1200,
18  "Rotation": {
19    "Roll": 0.0,
20    "Pitch": 0.0,
21    "Yaw": 0.0
22  },
23  "Translation": {
24    "X": 0.0,
25    "Y": 0.0,
26    "Z": 0.0
27  },
28  "Session Parameters": {
29    "Sitting": true,
30    "Background close": true,
31    "Cramped": false,
32    "Dark Clothing": true,
33    "Holding Weight": false,
34    "Ankle Weight": false,
35    "Height": 1.8,
36    "Angle": 20.0
37  }
38 }
```

---

## Realsense Cameras

We record Realsense Cameras using the librealsense SDK provided by Intel. Using the SDK we have access to the High-Level Pipeline API which allows us to stream the camera feed and access the camera intrinsics. This High-Level Pipeline API allows us to record the RGB and Depth streams from the Realsense camera. The SDK automatically synchronises the Depth and RGB stream as well as the motion sensors, which we do not use since our

camera is static. The librealsense SDK is available on GitHub<sup>2</sup>.

The Recordings are stored in a ROS bag file. A ROS bag file is a file format for storing ROS messages. The ROS bag file format is a container format that stores multiple messages in a single file. The ROS bag file format is described in detail in the ROS wiki<sup>3</sup>. The ROS bag file format is a container format that stores multiple messages in a single file. In our case, the important messages are the camera intrinsics, which allow us to create a virtual Realsense Camera from the recording, the RGB stream, and the Depth stream. However, other messages are also stored and can be accessed using the ROS Bag API<sup>4</sup>.

## Orbbec Astra Cameras

To read the depth stream of the Orbbec Astra camera we use the OpenNI2 API<sup>5</sup>. The OpenNI2 API is a cross-platform API that allows us to access the depth stream of the Orbbec Astra camera. The OpenNI API is no longer being developed by PrimeSense and has been renamed to OpenNI2 to avoid confusion with the OpenNI API. The OpenNI2 API is available on GitHub<sup>6</sup>.

Using the OpenNI2 API we can also record the depth stream to a file. The depth stream is stored as a .ONI file. The .ONI file format is a proprietary format that is not documented. However, the OpenNI2 API provides a .ONI file reader that allows us to access the depth stream.

Sadly, the OpenNI2 API does not provide a way to access the RGB stream. Therefore, we use the OpenNI2 API to access the depth stream and OpenCV to access the RGB stream. The RGB stream is stored as a .AVI file. The .AVI file format is a container format that stores multiple video streams in a single file. The .AVI file format is described in detail in the Microsoft documentation<sup>7</sup>.

***ISSUE:** currently the playback of the .AVI file is only possible at a specific framerate, which is set at the beginning of the recording session. This poses a substantial issue regarding synchronisation. Shoul I write about this?*

---

<sup>2</sup><https://github.com/IntelRealSense/librealsense>

<sup>3</sup><http://wiki.ros.org/Bags>

<sup>4</sup><http://wiki.ros.org/rosbag/Code%20API>

<sup>5</sup><https://structure.io/openni>

<sup>6</sup><https://github.com/structureio/OpenNI2>

<sup>7</sup>[https://docs.microsoft.com/en-us/previous-versions/ms779636\(v=vs.85\)](https://docs.microsoft.com/en-us/previous-versions/ms779636(v=vs.85))

### 2.3.2 Recording process

Once the scene is set and the pointclouds have been aligned the user can start the recording. Firstly, there are pre recording settings such as a frame and/or time limit in seconds, which allows accurate time and/or frame constraints to create equal recordings. Secondly, the user can select whether or not to display the pointcloud while recording. This allows the user to see the pointcloud in real-time and adjust the recording accordingly, however, this leads to a substantially reduced framerate<sup>8</sup>. These can be set in the GUI as shown in Figure 2-3. Finally, the user can configure the session parameters described earlier. After the recording has been started by the user a preconfigured countdown will be started, allowing the user to get into position in time. Once the countdown has finished the recording will start. The recording will stop once the time or frame limit has been reached. The recording will also stop if the user presses the stop button.

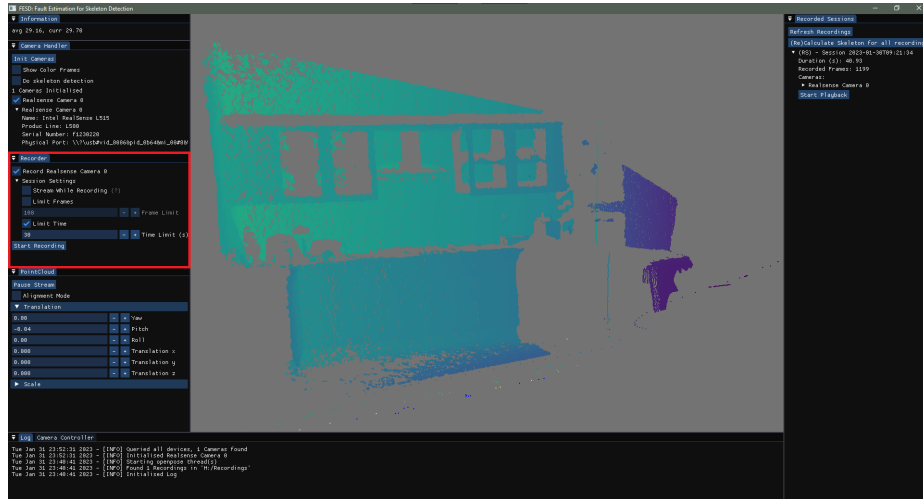


Figure 2-3: Recording GUI marked in Red. The user can set the recording parameters here such as the time and frame limits and whether or not the .

<sup>8</sup>From 30 FPS, without pointcloud to 15 FPS with pointcloud



## 2.4 Data population





## 2.5 Data Evaluation



## 2.6 Data Augmentation

This phase allows us to create a large amount of data with a controlled amount of faults. This is important since we want to be able to train a model that can detect faults in the data. The augmented data is stored in a separate file so that we can use it to train the model and compare it to the original manually checked ground truth.



## Chapter 3

# Model development



## 3.1 Model training





## 3.2 Model evaluation



## Chapter 4

# Experiment

- Cameras and how we used them
- Setup check (is the setup correct?)
- The recording process
- What Exercise was chosen and why
- Data size (not that important but still good to give perspective)
- How many seconds and frames were recorded in total
- How much data had to be discarded due to missing ground truth
- How much data was used for training and how much for testing



## Chapter 5

# Results

- Influence of different session parameters
- Time of recording
- Time of skeleton tracking

Here we present the results of our experiments. We first present the results of the experiments with the different network layouts. We then present the results of the experiments with the different input data. We also present the results of the experiments with the different error metrics. Finally, we present the results of the experiments with the different data augmentation techniques.



## Chapter 6

# Conclusion

In Conclusion, ...





## 6.1 Future work

- More stability in software (it is actually quite stable already but adding tests and ensuring adequate error handling would make it enterprise grade)
- Save everything in a single rosbag file and write a rosbag reader
- Multithreaded Streaming and recording (would probably not work since memory would be bottle neck)
- More data more variation in session parameters
- Different exercises
- Joint Reconstruction
- Attempt to fix error
- Import model using ONNX
- Use trained model in SilverFit
- Train model with different HPE algorithms and use model to decide when to use which