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

Abstract

Contents

List of Figures

List of Tables

Listings

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 [1].

[1] `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.

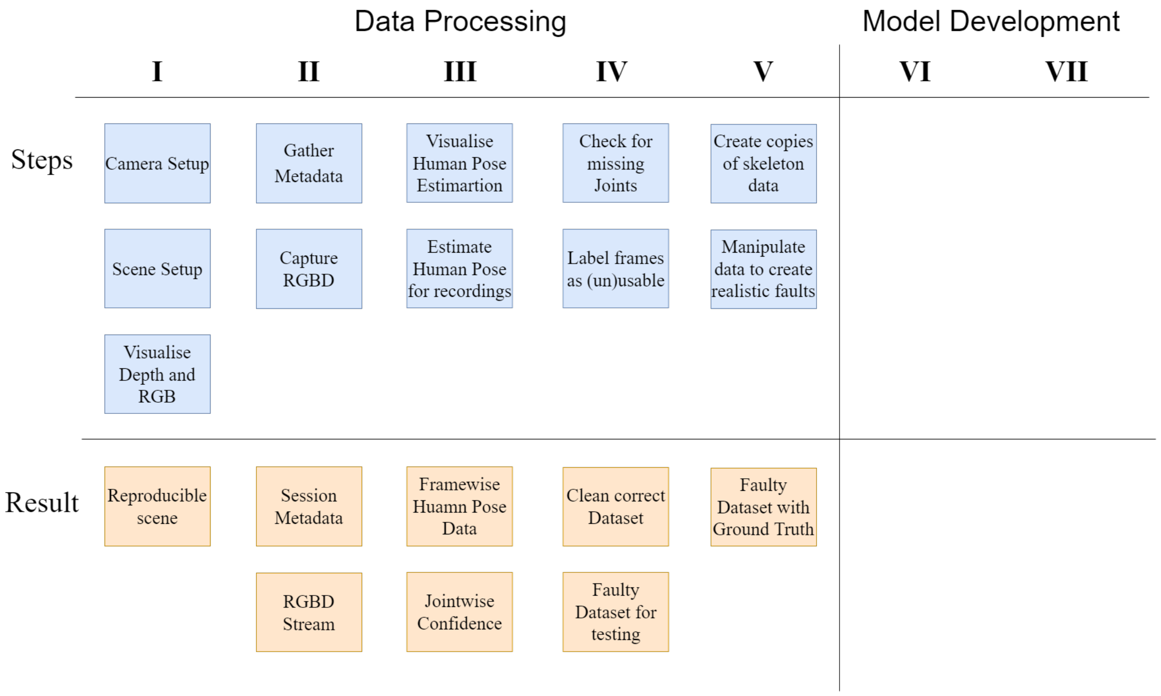

Data Processing
Model Development
I
II
III
IV
V
VI
VII
Steps
Camera Setup
Gather Metadata
Visualise Human Pose Estimartion
Check for missing Joints
Create copies of skeleton data
Scene Setup
Capture RGBD
Estimate Human Pose for recordings
Label frames as (un)usable
Manipulate data to create realistic faults
Visualise Depth and RGB
Result
Reproducible scene
Session Metadata
Framewise Huamn Pose Data
Clean correct Dataset
Faulty Dataset with Ground Truth
RGBD Stream
Jointwise Confidence
Faulty Dataset for testing

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.

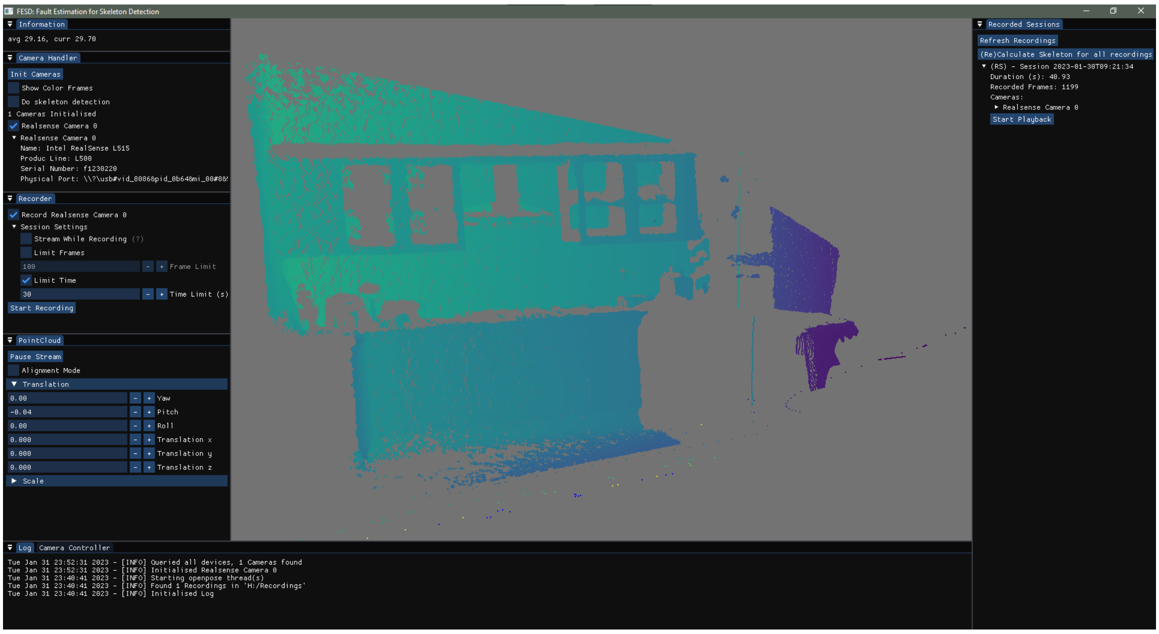
Chapter 2

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)*



avg 29.16, curr 29.70

□ Show Color Frames
□ Do skeleton detection
1 Cameras Initialised
☑ Realsense Camera 0
▼ Realsense Camera 0
Name: Intel RealSense L515
Produc Line: L500
Serial Number: f1230220
Physical Port: \\?\usb#vid_8086&pid_0b64&mi_00#88

☑ Record Realsense Camera 0
▼ Session Settings
□ Stream While Recording (?)
□ Limit Frames
100 − + Frame Limit
☑ Limit Time
30 − + Time Limit (s)

□ Alignment Mode
0.00 − + Yaw
-0.04 − + Pitch
0.00 − + Roll
0.000 − + Translation x
0.000 − + Translation y
0.000 − + Translation z

```
Tue Jan 31 23:52:31 2023 - [INFO] Queried all devices, 1 Cameras found
Tue Jan 31 23:52:31 2023 - [INFO] Initialised Realsense Camera 0
Tue Jan 31 23:40:41 2023 - [INFO] Starting openpose thread(s)
Tue Jan 31 23:40:41 2023 - [INFO] Found 1 Recordings in 'H:/Recordings'
Tue Jan 31 23:40:41 2023 - [INFO] Initialised Log
```

▼ (RS) - Session 2023-01-30T09:21:34
Duration (s): 40.93
Recorded Frames: 1199
Cameras:
▶ Realsense Camera 0

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

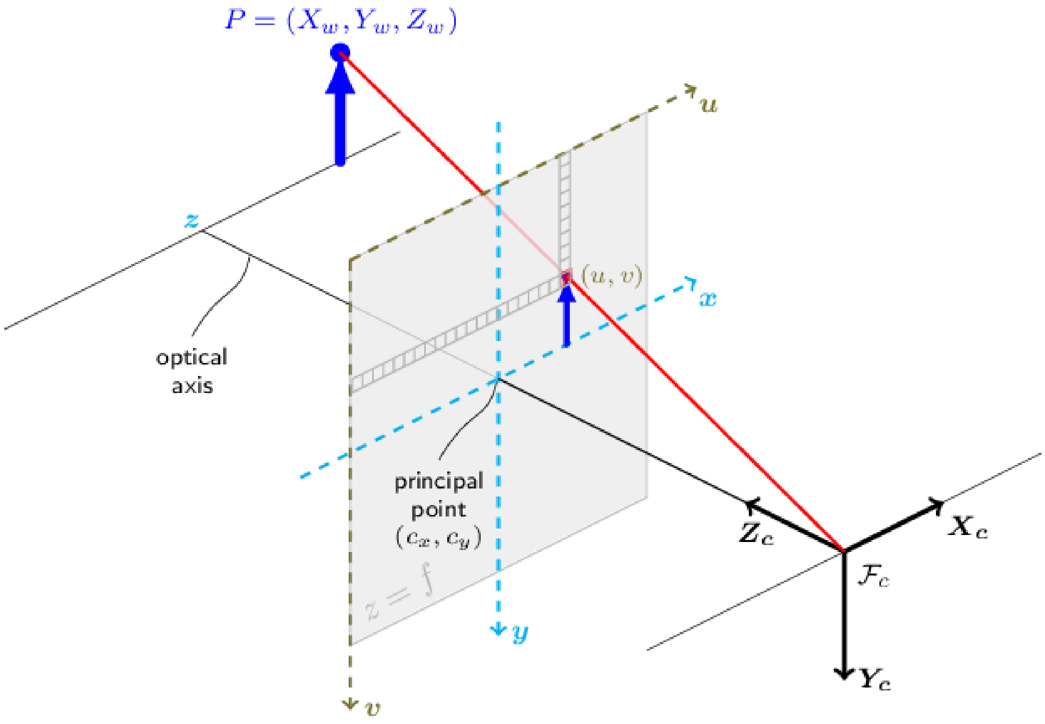


Figure 2-2: The pinhole camera model showing the principal point. The principal point is the point where the optical axis intersects the image plane. The field of view is the angle between the optical axis and the image plane.

translation are stored as Euler angles[1] 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

[1] 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.

```json
{
  "Cameras" :
  [
    {
      "Cx" : 314.26,
      "Cy" : 239.46,
      "FileName" : "Session_2023 01 30T09.21.34_Realsense_Camera_0.bag",
      "Fx" : 459.77,
      "Fy" : 459.83,
      "MeterPerUnit" : 0.00025,
      "Name" : "Realsense Camera 0",
      "Type" : "Realsense"
    }
  ],
  "DurationInSec": 40.0,
  "Name": "Session 2023 01 30T09:21:34",
  "RecordedFrames": 1200,
  "Rotation": {
      "Roll": 0.0,
      "Pitch": 0.0,
      "Yaw": 0.0
  },
  "Translation": {
      "X": 0.0,
      "Y": 0.0,
      "Z": 0.0
  },
  "Session Parameters": {
    "Sitting": true,
    "Background close": true,
    "Cramped": false,
    "Dark Clothing": true,
    "Holding Weight": false,
    "Ankle Weight": false,
    "Height": 1.8,
    "Angle": 20.0
  }
}
```

Realsense Cameras

We record Realsense Cameras using the librealsense SDK provided by Intel. Using the SDK we have access to the Hight-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[2].

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[3]. 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[4].

Orbbecc Astra Cameras

To read the depth stream of the Orbbecc Astra camera we use the OpenNI2 API[5]. The OpenNI2 API is a cross-platform API that allows us to access the depth stream of the Orbbecc 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[6].

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[7].

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?*

[2] `https://github.com/IntelRealSense/librealsense`

[3] `http://wiki.ros.org/Bags`

[4] `http://wiki.ros.org/rosbag/Code%20API`

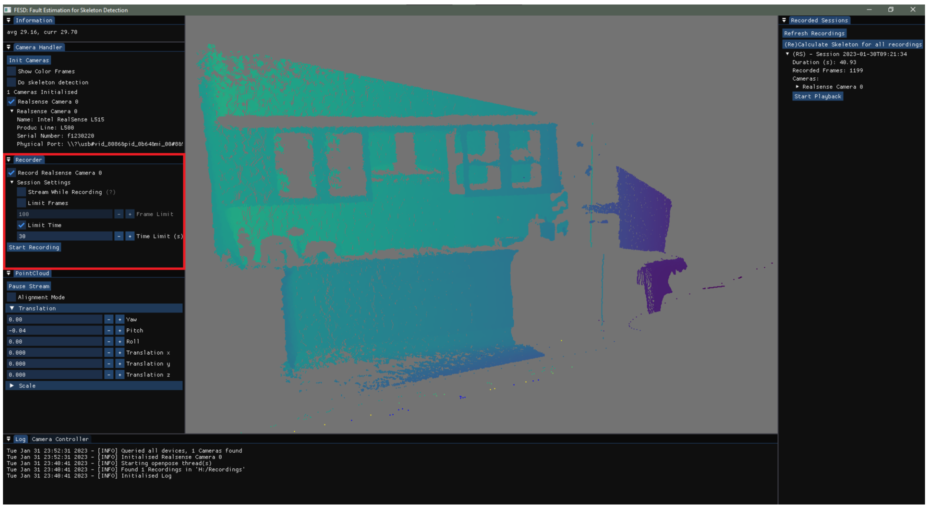
[5] `https://structure.io/openni`

[6] `https://github.com/structureio/OpenNI2`

[7] `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[8]. 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 beend 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.



```
avg 29.16, curr 29.70

□ Show Color Frames
□ Do skeleton detection
1 Cameras Initialised
☑ Realsense Camera 0
▼ Realsense Camera 0
    Name: Intel RealSense L515
    Produc Line: L500
    Serial Number: f1230220
    Physical Port: \\?\usb#vid_8086&pid_0b64&mi_00#8&

☑ Record Realsense Camera 0
▼ Session Settings
    □ Stream While Recording (?)
    □ Limit Frames
    100          - + Frame Limit
    ☑ Limit Time
    30           - + Time Limit (s)

□ Alignment Mode
0.00         - + Yaw
-0.04        - + Pitch
0.00         - + Roll
0.000        - + Translation x
0.000        - + Translation y
0.000        - + Translation z

▼ (RS) - Session 2023-01-30T09:21:34
  Duration (s): 40.93
  Recorded Frames: 1199
  Cameras:
   ▶ Realsense Camera 0
```

```
Tue Jan 31 23:52:31 2023 - [INFO] Queried all devices, 1 Cameras found
Tue Jan 31 23:52:31 2023 - [INFO] Initialised Realsense Camera 0
Tue Jan 31 23:40:41 2023 - [INFO] Starting openpose thread(s)
Tue Jan 31 23:40:41 2023 - [INFO] Found 1 Recordings in 'H:/Recordings'
Tue Jan 31 23:40:41 2023 - [INFO] Initialised Log
```

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 .

[8] From 30 FPS, without pointcloud to 15 FPS with pointcloud

2.4 Data population

JUST AS REFERENC *To achieve the highest framerate, we calculate the skeleton based on the recorded data and add it to the dataset in a separate step. The RGB stream is used to create the dataset for the skeleton detection and the depth stream is used to create the point clouds for the calculation of global skeleton points. We store both the local 2D coordinates in accordance with the image used for the skeleton detection, as well as the global 3D coordinates based on the aligned point clouds. Additionally, OpenPose provides us with a confidence score for each joint.*

DECISION *I decided to switch to NuiTrack, it is closer to silverfit and I think a better choice. Openpose poses more problems than it solves*

TODO *Explain what is meant by Data Population (skeleton detection).*

2.4.1 Human Pose Estimators

OpenPose

TODO *Give rough overview of Openpose and how the projection might have worked.*

NuiTrack

To utilise skeleton data, as well as the human silhouette in their games, SilverFit utilises the NuiTrack SDK. ***TODO*** *Explain what NuiTrack is and does and how it works.*

2.4.2 Human Pose Estimation

Human pose estimation is not trivial in terms of resource usage. Calculating the human pose while recording would mean a significant reduction of the recorded frames. Therefore, we calculate the skeleton seperatly for each frame of the recording.

The skeletons is stored ... ***TODO*** *Explain how the skeleton is stored.*

2.5 Data Evaluation

Next, we evaluate the recorded data and especially the detected skeleton. We focus specifically on the joints with a low confidence value.

We discard frames with lacking pose data from the training dataset, they are not usable to train a model. We might still use it for the testing phase. If we notice that the dataset is getting too small, we might re-record some sessions.

QUESTION *Should I discard frames with limited joints?*

Additionally, while recording multiple people might be wrongly detected. However, in our experiments we only consider single person recordings. Therefore, we can discard other people from the data.

The pseudo code for the data evaluation process can be seen in Listing **??**. Most of the checks mentioned, such as `selectInvalidPeople` in Line 9 or `checkJointValidity` in Line 14 happen manually. However, the data processing is done automatically by the code to redue human error.

Listing 2.2: Pseudo code for data evaluation

```
def data_evaluation(recording):
  invalid_frames = []
  for frame in recording:
    if not frame:
      invalid_frames.append(frame)
      continue
    else:
      if len(frame.people) > 1:
        invalid_people = frame.selectInvalidPeople()
        frame.remove(invalid_people)

      for joint in frame.people[0].skeleton:
        if joint.confidence < CONFIDENCE_THRESHOLD:
          checkJointValidity(joint)

  if len(invalid_frames) > INVALID_LIMIT:
    return False

  recording.replace(invalid_frames, Null)

  return True
```

2.6 Data Augmentation

Once the data is cleaned and we filter recordings with too many missing joints, we can augment the data to simulate a larger amount of data with the ability to create faulty scenarios controlled. One major fault is the seemingly random detachment of the joint to a side. This especially affects the legs and arms, therefore we will have a bias toward limbs with this augmentation.

Furthermore, we randomly move the joints with low confidence. Another fault is the disappearance of joints. We use the same bias as with the random detachment, i.e. we take the limb bias, as well as the confidence into consideration.

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.

TODO *Create some screenshots of the augmented data from the different methods.*

Chapter 3

Model development

While there could be multiple approaches to fault estimation, we have chosen to use a deep learning approach. The reason for this is that deep learning has shown to be very successful in many different fields, such as image classification, object detection, and image segmentation. The reason for this is that deep learning can learn the features of the data by itself, without the need for manual feature extraction. This is especially useful in our case, as we have a large amount of data, but we do not know which features are important for the fault estimation.

Other possible solutions could be to use rule-based systems, which use inverse kinematics, or use frame-to-frame joint comparison to detect discreptancies, however, these are quite limited and might result in either too many false positives or false negatives. Furthermore, these rules, such as the frame-to-frame joint comparison, are not always applicable to all types of movements, and therefore might not be able to detect all types of faults in all cases.

3.1 Model training

Using this enlarged dataset, we can train a Neural Network to recognise faults in the data. We use the depth data as input, the skeleton data as input, and a combination of both as input. We also experiment with different network layouts, such as a fully connected network, a convolutional network, and a combination of both. We use the augmented data to train the model and the manually checked ground truth to validate the model. We use the validation data to determine the best model and the best network layout. We use the best model to predict the faults in the data.

3.2 Model evaluation

Finally, we evaluate our model by calculating different error metrics such as the mean absolute error, the mean squared error, and the root mean squared error. We also calculate the accuracy of the model, which is the percentage of correctly predicted faults. We also calculate the precision and recall of the model. The precision is the percentage of correctly predicted faults out of all predicted faults. The recall is the percentage of correctly predicted faults out of all faults in the data. We also calculate the F1 score, which is the harmonic mean of the precision and recall. The F1 score is a good indicator of the overall performance of the model.

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)
- Evaluation process evaluation, how many recordings had to be discarded, how many frames were invalid, and so on
- 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 generally usable) (I also have a lot of ideas for this but I will not write them here, maybe ill name some in the report)
- Save everything in a single rosbag file and write a custom rosbag reader
- Multithreaded Streaming and recording (would probably not work since memory would be bottle neck, especially to a single file this would be a problem)
- More data more variation in session parameters
- Different exercises
- Joint Reconstruction
- Attempt to fix detected 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