DESIGN AND IMPLEMENTATION OF GRAPHIC USER INTERFACE FOR CROSS-MODALITY SENSOR SIMULATION

DESIGN AND IMPLEMENTATION OF GRAPHIC USER INTERFACE FOR CROSS-MODALITY SENSOR SIMULATION

By LIANG XU, M.Eng.

A Thesis Submitted to the School of Graduate Studies in Partial Fulfillment of the Requirements for the Degree Master of Engineering

McMaster University

MASTER OF ENGINEERING (2022)

Hamilton, Ontario, Canada (Computing and Software)

TITLE:	DESIGN AND IMPLEMENTATION OF GRAPHIC USER INTERFACE FOR Cross-Modality Sensor Simulation
AUTHOR:	Liang Xu M.Eng. (Computer Science and Engineering), McMaster University, Hamilton, Canada
SUPERVISOR:	Dr. Zheng Rong
NUMBER OF PAGES:	vi, 40

Acknowledgement

I would like to express my gratitude to my supervisor Dr. Rong Zheng for her guidance, comments and suggestions throughout my graduate study and the project. I would also like to thank Yujiao Hao who designed the CROMOSim pipeline and implemented the initial scripts.

Table of Contents

List of Figures

Abstract

With the advancement of wearable devices and micro-electromechanical systems (MEMS), inertial measurement unit (IMU) sensors have been widely adopted in commodity devices[1]. Hao et al. designed a cross-modality sensor simulator named CROMOSim that simulates IMU sensor data to address the data scarcity problem. However, the CROMOSim pipeline is not user-friendly and has a long learning curve. Therefore, designing a graphical user interface (GUI) that allows users to access the CROMOSim pipeline easily is essential for improving learnability and usability. One option is to develop a standalone application with GUI for the pipeline. However, installing such an executable application usually takes much effort to prepare a suitable running environment. Instead, in this project, we design a web application that features a web front-end and CROMOSim as a back-end. Considering the maintenance and extensibility, we implement the web front-end with JavaScript and implement the back-end service application using Spring framework. To use the web application, a user needs to upload a video containing a single subject with target movement. A mesh model and the corresponding motion sequence will be generated based on the input video. The user can zoom in and out and rotate the mesh model, and then select a desired virtual sensor location. Data from the virtual sensor will be simulated that match the subject's movements in the video. To evaluate the performance of the implementation, we have conducted a user survey and scene-based usability tests. Test results show that our application for CROMOSim satisfies the main usability goals, but there are still learnability improvements to be made.

Chapter 1

Introduction

Tracing the development of inertial measurement units (IMUs) since 19^{th} century, it was initially used for navigation and some other large-scale applications[2]. In recent years, with the build-out of the MEMS technology, IMUs sensor has been introduced to the area of daily usage. Human activity recognition (HAR), which utilizes IMUs sensor data to monitor human motions and analyze patterns, is an essential application of IMUs in wearable and mobile devices[3]. However, the amount of sensor data required to train HAR model is ordinarily large, making it challenging to collect in uncontrolled environments[4]. For the purpose of tackling data scarcity problem, Hao et al. designed a cross-modality sensor simulator named CROMOSim that simulated high-fidelity virtual IMU sensor data from motion capture systems or monocular RGB cameras.

1.1 Motivation

Hao et al. presented a practical pipeline that simulated IMU sensor readings at arbitrary on-body positions. This pipeline takes a monocular camera video as input, and generates SMPL-represented global motion and body shape based on the input video. Using SMPL model, the simulator takes sensor placements and orientation as input, and predicts virtual IMU readings in the sensor coordinates frame as the final output. One of the critical functions of this pipeline is to extract global human motion sequences from a video. Nonetheless, this function requires several external methods and too much effort to set up the environment. On the other hand, the whole pipeline is too complex to be implemented in a few scripts, resulting in a longer learning curve than expectation. The motivation of our project is to improve usability and reduce the effort required for learning CROMOSim, by wrapping up the pipeline into an easy-to-use tool and implementing a web-based graphical user interface. Synchronously, a well-designed GUI is needed to provide a visual representation of model data and helps users to interact with the model directly.

1.2 Related work

To solve the data scarcity problem, many frameworks have been developed to simulate IMU sensor data from monocular RGB videos. However, these frameworks are usually time-consuming to set up a running environment. In addition, existing frameworks do not include any graphic user interface, which are user-unfriendly and always require much effort to learn and use. ZeroNet provides a solution to extract 3D finger pose and location from videos and estimate corresponding acceleration and

orientation[5]. In [6], the author presents a method to identify humans in videos, extract their 2D poses, and then generate synthetic IMU data (acceleration and gyro norms) using a regression model. Unfortunately, the aforementioned two techniques merely focus on the 3D poses of the human/finger, while the global motion problem is simply ignored by assuming a fixed camera scene. Similar to our approach, in [7], the author introduced a pipeline to extract whole-body human activities (including 3D poses and movements in the whole scene) from a moving camera video and simulate virtual sensor data. However, in this pipeline, the depth map and camera ego-motion estimation are divided into two steps, which require precise prediction to avoid distorted 3D motions. On the other hand, when extracting accelerometry data, the use of skeleton body representation discounts factor considerations such as body mass, device movement and skin friction.

1.3 Goal

This project aims to provide an efficient graphic user interface to access Hao et al.'s cross-modality sensor simulator using video from a monocular RGB camera. Our project intends to be a portable and lightweight application that allows users to generate a mesh model from appropriate video and select a desired sensor location from the model. Our project presents a web application with two main components, including a web page and a web service application. The web page is implemented with HTML, CSS and React JavaScript, which is expected to be easy for target user to learn and use. After accessing our web page successfully, users are required to upload a video and estimation results from robust consistent video depth method to generate a mesh model. Users need to choose one or more vertices or faces in the

mesh model to represent the location of a IMU sensor. Virtual IMU sensor data for all frames will be generated based on the input sensor location, and it will be available for download as the final result.

1.4 Organization of the Report

This report documents the design of the web application and the techniques used. The rest of the report is organized into five chapters. Chapter 2 describes the main procedures and methods used in our application to generate a 3D mesh model from the given video, including global trajectory and body pose estimation. Chapters 3 and 4 provide some critical design decisions as well as implementation details for both back-end and front-end respectively. The following chapter summarizes the usability test methodology that has been performed for the web application. Then we describe test results and analyze potential improvement in user experience. Finally, the report ends with a conclusion summarizing the implementation of our web application and several improvements we could conduct for future work.

Chapter 2

Mesh Model and Movement

As described in CROMOSim pipeline, the whole system could be divided into three function modules:

- Input data processing module: In this module, it takes a monocular camera video as input, extracts human pose and global displacement from the video, and converts them into human motion sequence represented by SMPL model [8].
- Human body module: This module visualizes the SMPL motion sequence, generated by the previous module. This module allows users to interact with the visualized mesh model and identify desired body locations.
- Sensor simulator module: This module takes the specified sensor location as input and simulates IMU sensor data in the sensor coordinates frame.

This chapter focuses on the input data processing module, which introduces the techniques and frameworks used to extract human pose and global motion from the

video.

As shown in Fig.2.1, there are two sub-processes in the input data processing module to extract the human motion sequence from the video. Firstly, we estimate the global displacement and rotation of the human body. After that, 3D human poses are estimated and represented with an SMPL model. The SMPL-represented result is combined with the global displacement in first step to generate a global human motion sequence, which will be visualized to users in next module.

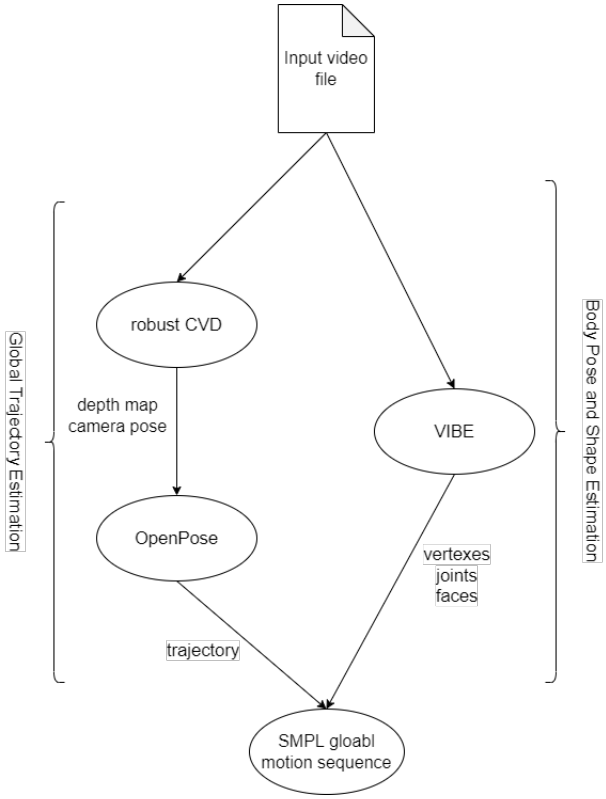

Fig. 2.1: Motion sequence extraction

2.1 Global Trajectory Estimation

The estimation of human trajectory in global coordinates is one of the essential procedures to extract global human motion sequences from a input video. In this project, we implement this procedure utilizing the robust consistent video depth estimation method (Robust CVD) and OpenPose[9][10].

However, the execution time of the CVD method is longer than expected. Considering the scale of our project, we adopt the CVD method as an external service. Users need to execute the CVD method separately and upload the results to our project as input. The whole trajectory estimating procedure is implemented as an executable script with Python. It takes the results of robust CVD as input, applies OpenPose to the torso centre locations in frame coordinates, and returns the torso centre locations in global coordinates as a text file.

The specific steps are as follows:

1. Take input video and CVD results from the user and store them in a local working directory.
2. Convert the video into frames and downscale each frame.
3. Convert downscaled frames into video and apply OpenPose to obtain body key points in each frame.
4. Load raw torso centre locations in video frames and filter out these locations with low confidence scores.

5. Load depth map and camera positions to estimate trajectory in a global coordinate for each frame.

6. Apply Kalman filter to smooth resulting trajectory

7. Re-construct the data format to fit requirements of mesh model canvas.

8. Store the results as a text file in the working directory.

2.2 Body Pose and Shape Estimation

Estimating the pose and shape of human body is another essential element to evaluate the global human motion sequence. We adopt Video Inference for Body Pose and Shape Estimation method (VIBE) in our project to directly generate in-place 3D human poses regardless of global trajectory[11]. This method generates 3D human pose and shape estimation represented by a skinned multi-person linear model (SMPL).

2.2.1 SMPL Model

SMPL is a skinned vertex-based model that is represented by a triangle mesh, which could be divided into a vertex list and a face list. The vertex list contains all required vertices to form the model, while the face list indicates how vertices are connected together to represent the human skin with triangles. As mentioned in the related work section, a skeleton body model is inadequate in pose representation which will discount factor considerations such as skin friction. Meanwhile, wearable IMU sensors are always attached to human skin instead of the skeleton. Thus, compared with the skeleton model, SMPL model provides a better representation and also helps to

achieve more accurate estimation.

2.2.2 Estimation Procedures

Execution results of VIBE method are stored as a serialized pickle file (.pkl) in the working directory. We only consider two aspects from the result file. Firstly, in order to estimate body pose sequence, the vertex locations and mesh face composition of each frame are required. Vertex data stored in the result file is organized into a 2-Dimension array without face information. Mesh face data is extracted from the default model file and reconstruct vertex data. Secondly, as an add-on feature, 3D joint locations are extracted from the result file to visualize some key joints of the human body. These key joints will serve as a reference for the user to select sensor locations.

For body shape estimation, it is assumed that people's body shapes will not change within a short period of time. Instead of estimating specific body shapes, a average neutral SMPL model is used to avoid unnecessary computation.

The specific steps are as follows:

1. Take input video from the user and store it in the local working directory.
2. Execute VIBE script on the command line on the input video.
3. Load execution results which are stored in the working directory.
4. Extract vertex data and joint locations from the results.
5. Identify and fill empty frames.

6. Re-construct data format to fit the requirements of mesh model canvas.

7. Store results as a text file in the working directory.

Chapter 3

Web Application Back-end Design and Implementation

In consideration of portability, we wrap CROMOSim pipeline into a web application, which allows the users to access the project virtually through a front-end web page while file storage and computations take place in the back-end service server.

The number of services included in back-end application is relatively small, and a small number of concurrent users are expected accordingly. We make the decision to design the back-end application with a Monolith architecture in that all services are coupled into a single instance. This architecture contributes to simplify the deployment and testing process. We also apply the Spring framework in our project to simplify the complex development process by offering technologies such as Aspect-oriented programming and Dependency injection.

As shown in Fig3.1, the proposed back-end application consists of 4 components,

including API controllers, business interfaces, business service processes and data transfer objects. When users access our project, the service requests will be sent from the web client to the back-end application through the API controller. Once received, it will be converted into a data transfer object and sent to the business service process indirectly through a business interface. The result will be computed in the business service process, which is invisible to the client and returned through an HTTP response.

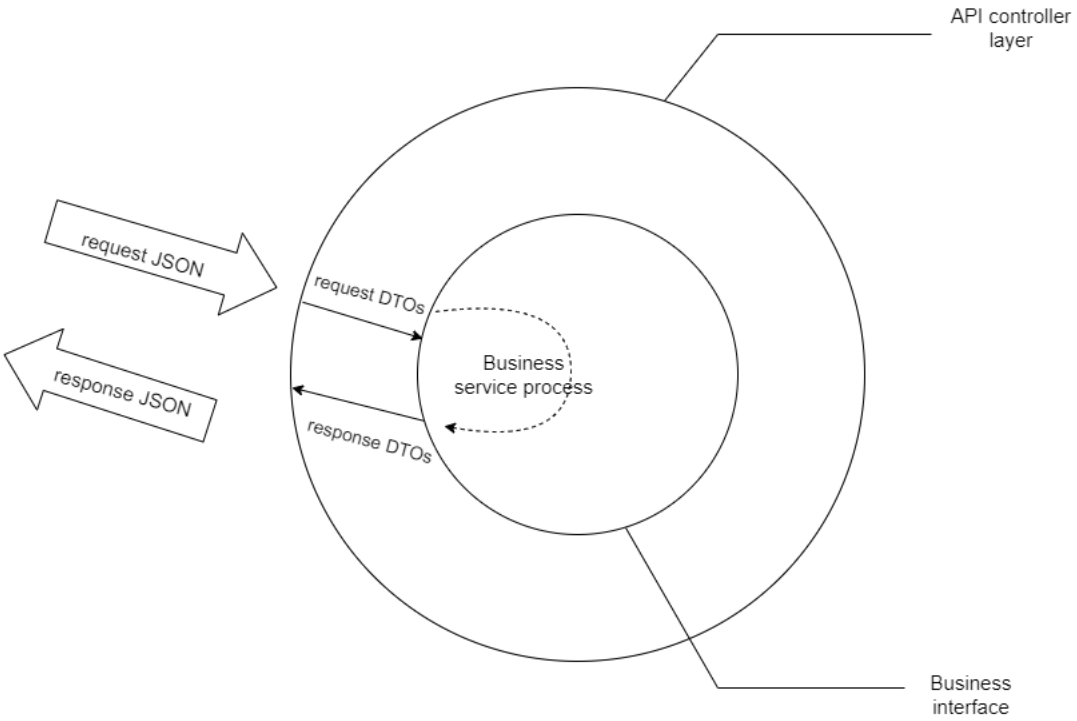

Fig. 3.1: Back-end components overview

3.1 Modules and System Overview

The back-end service application is designed using Spring framework and implemented with Java. The detailed implementation could be divided into four modules. Each module works independently while they share the same working directory for result storage. The relationship and interaction between modules and external system are summarized in Fig3.2

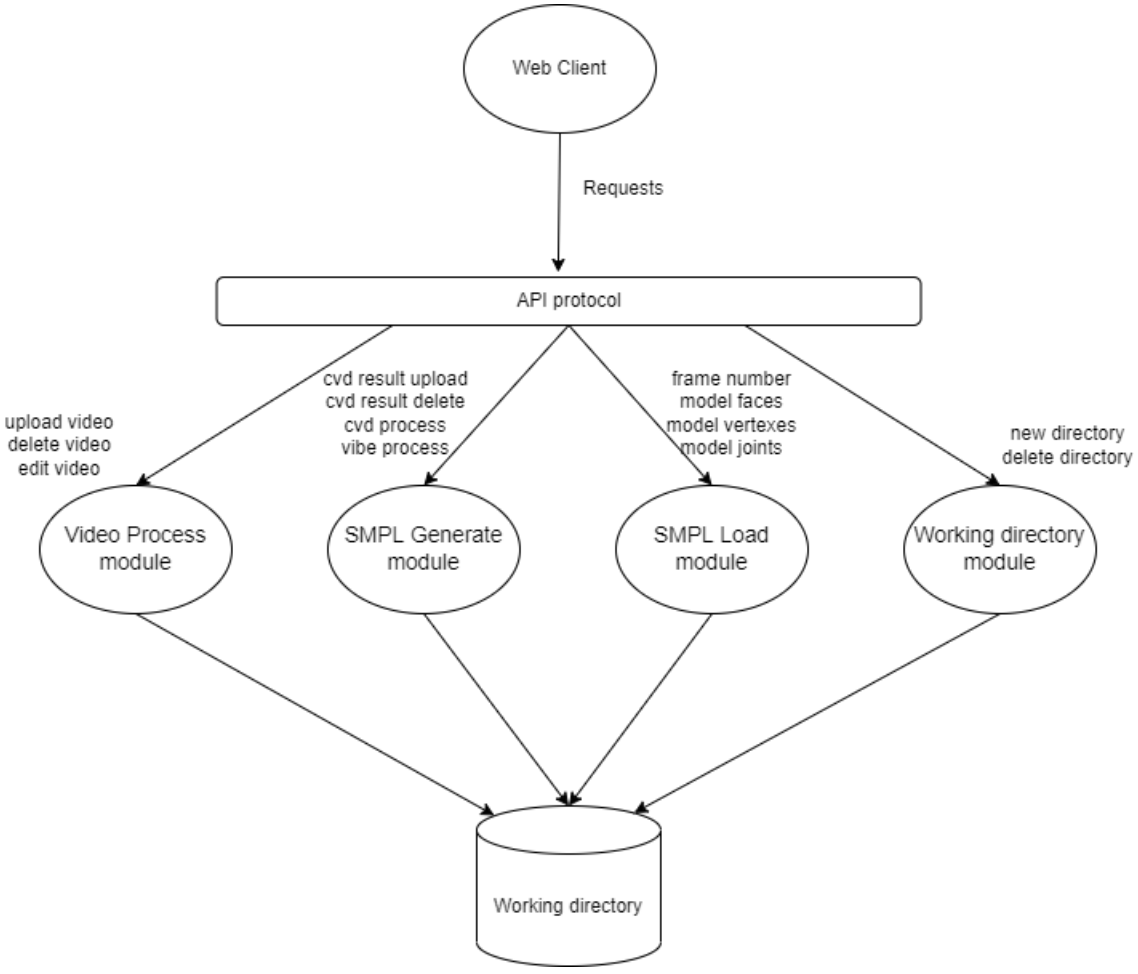

Fig. 3.2: Back-end module architecture design

- Video Process module:

This module provides video-related services to web clients. These services include trimming the input video, storing and deleting the video in the working directory.

- SMPL Generate module:

  This module allows the client to perform model-generating procedures based on the input video in working directory. It also provides services of uploading or deleting the results of robust CVD method, which is required by the generation procedures.

- SMPL Load module:

  This module handles the requests for model data, including the number of frames, mesh model faces, vertices and joints for a given frame index.

- Working directory module:

  This module provides directory-related services to avoid directory conflicts, such as checking the availability of directory ID and creating or deleting the working directory when users enter or leave the web client.

3.2 Data Storage and Multi-user Handling

Multi-user handling is another vital consideration requiring concern in this project. As one of the functional requirements, this project obtains input files from the users and stores them in the working directory of server. To avoid data conflicts between different clients, their HTTP requests and working directories need to be separated from each other.

Our project handles multiple clients by using threads from the Spring framework. A thread will be created for each HTTP request received from web clients. Every request will be executed in its own thread, separated from the others. However, threads safety is not a concern for this project since file storage is also separated, which means that threads from different clients do not share the same working directory.

The working directories are managed dynamically. When the user enters the web page and uploads a required video, the web client will send the first HTTP request to the back-end service application. Such a request will generate a random number and create a working directory named after that generated number. The first response sent back to the client will also include this randomly generated number, and the client will use it in later requests to indicate its working directory. When the user leaves or refreshes the web page, the working directory will be deleted.

Chapter 4

Web Application Front-end Design and Implementation

The front-end web page is another essential part of this project, which provides a graphic user interface allowing users to access project functions through HTTP requests. We make a design decision to implement the web page with React.JS, which is a component-based JavaScript framework. React.JS provides a modular structure that makes code maintainable and also flexible for adding new features for future works. At the same time, React.JS uses the virtual document object model (DOM) interface, which is beneficial to ensure the rendering performance of mesh models when visualizing the motion sequence.

The basic wire-frame design of the GUI could be found in Fig. 4.1. The whole web page is composed of 4 components. At the top of the page is a title bar with a logo, the name of the current page and a help button. On the left-hand side of the page is a sidebar containing some general functions, e.g., back to the home page, reset

the current page, and move forward or backward. In the centre of the page, there is the main window that will display the selected video, SMPL model or processing progress according to the current step. At the bottom of the window, there is a tool panel that offers all available functions for current progress.

Fig. 4.1: Web page wireframe layout overview

4.1 Modules and System Overview

This project summarises CROMOSim pipeline and divides the whole procedure of the client side into three steps, including video upload, model generation and model visualization. By using the component-based framework that depends on JavaScript,

the whole web page is organized into a tree structure. An overview of web page architecture could be found in Fig. 4.2. The root node represents the main window which has three child nodes representing three components, respectively. The components will be displayed in the main window one at a time according to the current progress.

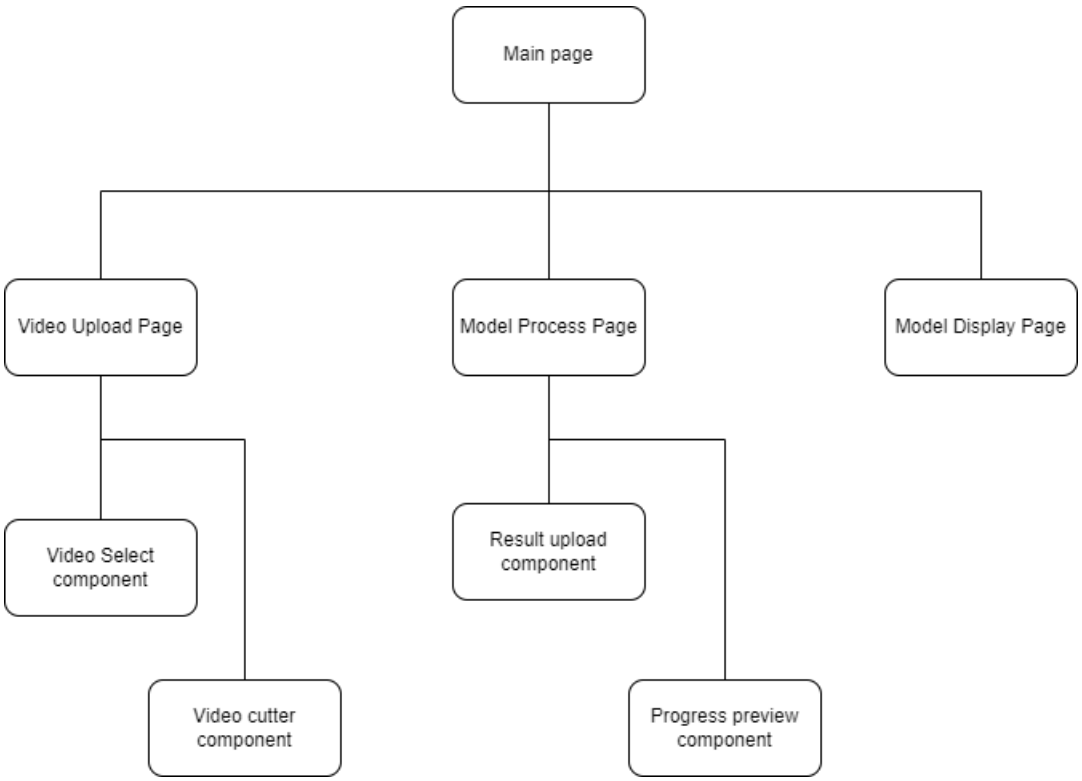

Fig. 4.2: Web page structure design

- Video Upload Component:

  This component offers all video-related functions, including uploading a video and trimming the uploaded video. This component could be further divided into two sub-components:

  1. As shown in Fig. 4.3, the Video Upload sub-component is displayed on the main window when the user enters the web application. It includes a

button that allows the user to open a file dialog box, browse and select an MP4 file to upload.

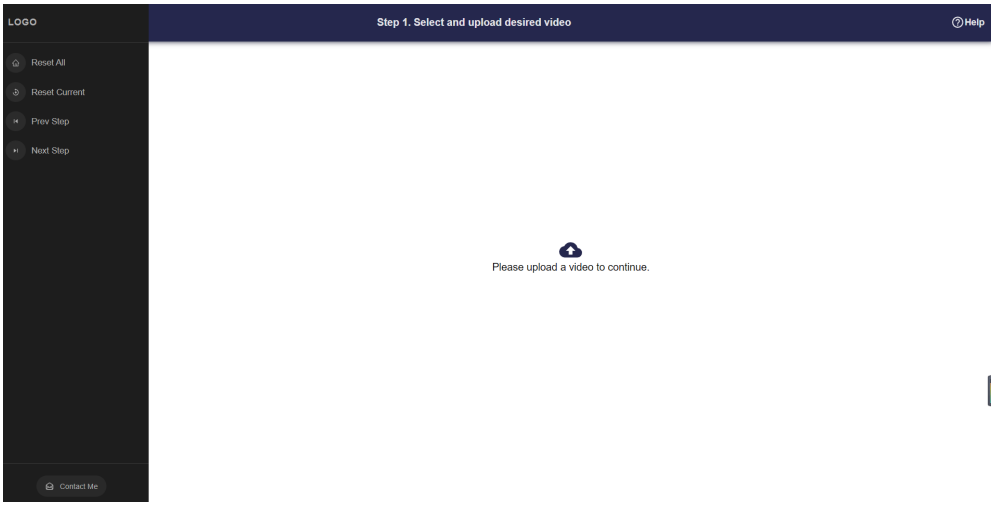

Fig. 4.3: Video upload sub-component

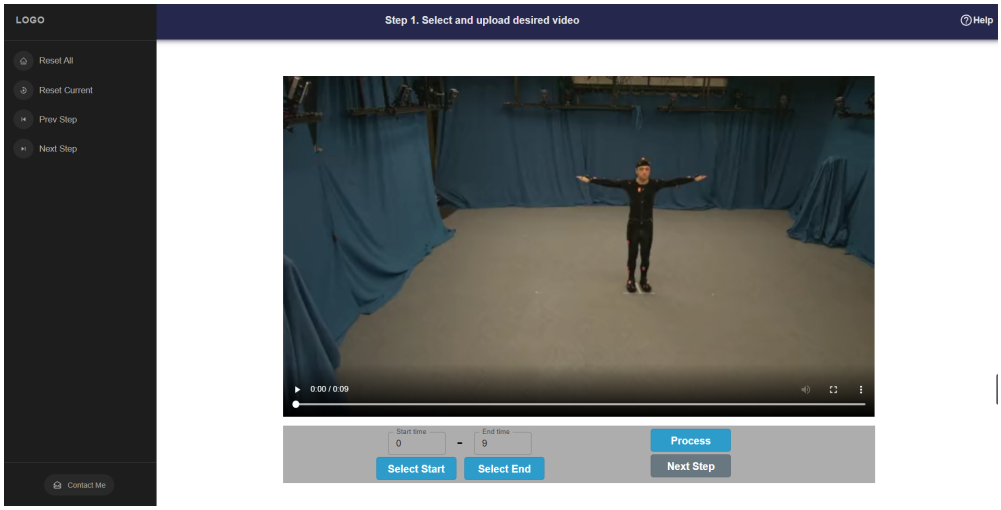

Fig. 4.4: Video trimmer sub-component

2. The Video Trimmer sub-component automatically appears when a video is selected as shown in Fig. 4.4. This sub-component offers a preview of the uploaded video and allows the user to trim video by selecting both the

start and end times. Once completed, the user can either reset the current step or move forward to the next component.

- Model Process Component:

  This component takes three zip files from users as input and sends requests to the back-end server to generate the SMPL mesh model. The model-generating procedure can be split into four sub-steps, and the progress is visualized on the main window. This component could also be divided into two sub-components as follows:

  1. The Result Upload sub-component is similar to the video select sub-component as shown in Fig. 4.5. It allows the user to select and upload results from robust CVD method. This sub-component also includes descriptions of all desired files.

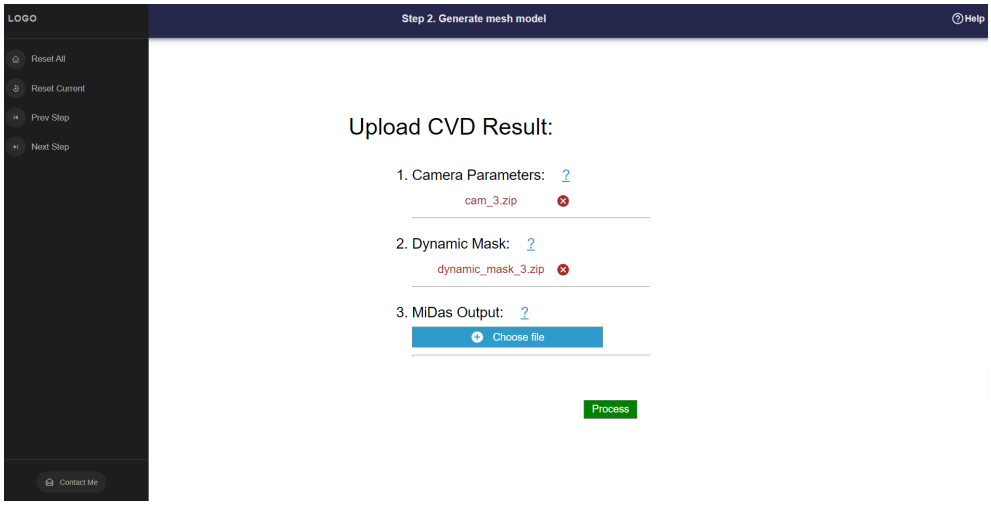

Fig. 4.5: Result upload sub-component

2. Fig. 4.6 presents the Progress Preview sub-component, which provides an overview of model-generation steps and indicates the status of each step, i.e. completed, pending, in progress or failure. Only after all steps have been completed successfully can users move forward. The failure of any step means that the upload files from the client are improper, and the user will be instructed to re-execute the result upload step.

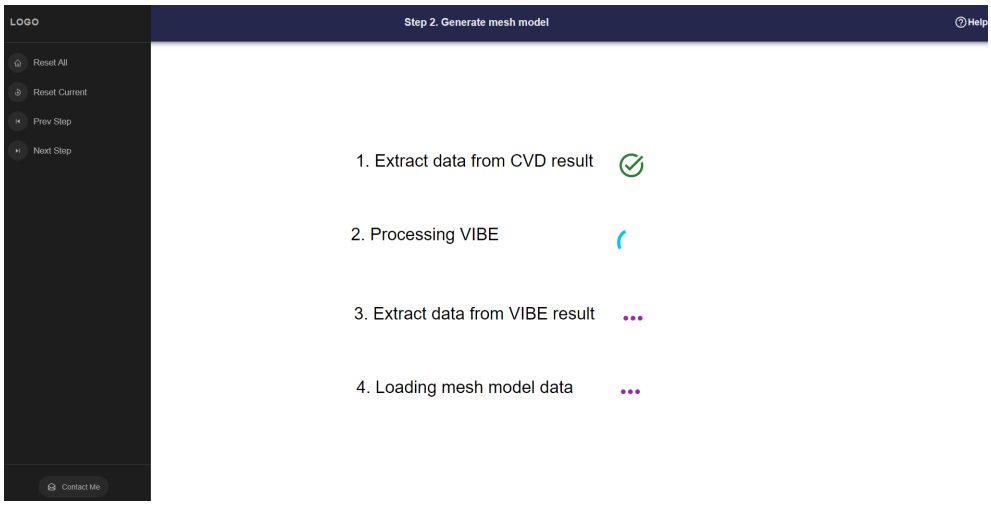

Fig. 4.6: Progress Preview sub-component

- Model Display Component:

  Once the mesh model data is loaded successfully, the Model Display Component will be displayed in the main window as shown in Fig. 4.7. This component includes a 3D canvas that visualizes the mesh model and its global motion sequence. On the canvas, users are able to control position and angle of the camera by dragging the mouse. They could also double-click a vertex or face on canvas to place a coloured sphere or a coloured triangle, indicating that the vertex or face is selected as a virtual location for IMU sensor. This component

also visualizes the location of human key joints as an option working as the reference for selecting a desired sensor location.

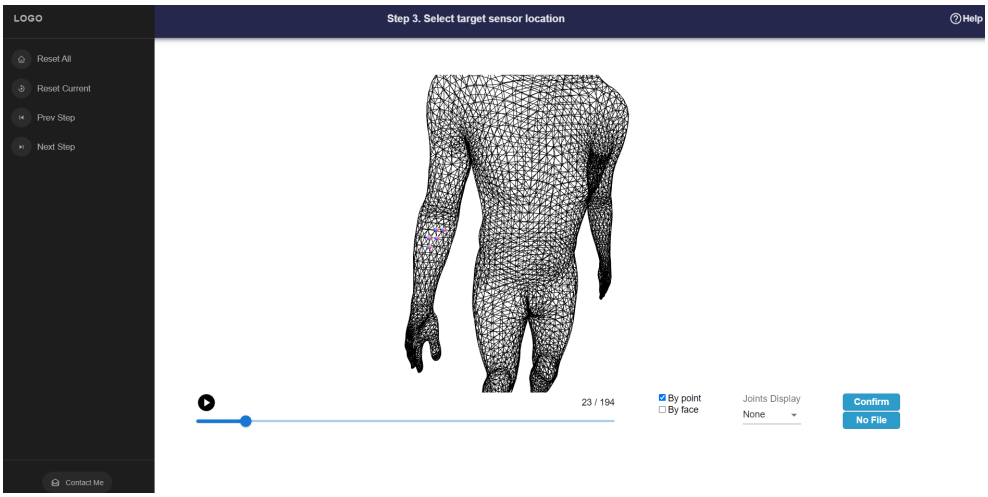

Fig. 4.7: Model Display Component

4.2 SMPL Model Rendering

Rendering SMPL mesh model on the 3D canvas is an essential part of the front-end web page. In this part, mesh model data is received from the back-end web application through HTTP responses. Users will have a preview of the mesh model with motion sequences, and they can rotate the mesh model by controlling the camera with simple mouse actions.

Before entering Model Display Component, this application has a procedure to preload all mesh model data into the web page. This step ensures the data integration when displaying the mesh model and could also prevent unexpected latency when playing the model motion sequence. As mentioned in the back-end application design, mesh

model data is separated into four parts, including frame number, mesh model faces, mesh model vertices and joints. The model face data is an array of faces composed of 3 indexes of vertex, and model vertex data is an array of vertices represented by a tuple of x, y and z positions. The vertex and face data are received separately and combined instead of being combined before receiving. We made this design decision because the face data is identical for all frames, while only vertex data changes for different frames. When the number of frames is large, separating faces from mesh model data can ensure the loading performance.

We use the Three.js library for drawing 3D mesh models in the browser. To simulate the virtual way of human eye, we use the camera based on the perspective projection provided by the library. We also enable orbital control, which allows the user to translate the camera and adjust the angle. The input video always includes a large number of frames, so motion sequences are visualized to be similar with a video. We simply replace the old vertex data with the new one for each frame. We also update the position of the selected location for the virtual IMU sensor and notify canvas to update.

4.3 Point and Face Selection

To obtain the final simulator result, the user needs to select several vertices or faces in the mesh model to indicate the IMU sensor location. However, the mesh model and the mouse belong to different coordinate systems, and users cannot select the model vertex directly. Thus, we need to project all model vertices into the camera's normalized device coordinate (NDC) space and allow users to highlight the vertex or

face closest to the mouse position.

We first transform the vertex into the camera's matrix world to project a model vertex. Given all required data as shown below:

$$v = \Big[x, y, z\Big], M_{iw}, M_p$$

where v is a model vertex, M_{iw} and M_p represent 4×4 inverse world matrix and projection matrix provided by Three.js library respectively. As shown in Eq.(4.1) and Eq.(4.2), we first apply the inverse of the world matrix to the vertex's vector to get a new vector in the camera world.

$$\Big[r_1, r_2, r_3, r_4\Big] = \Big[x, y, z, 1\Big] \times M \tag{4.1}$$

$$v_{new} = \Big[r_1/r_4,\ r_2/r_4,\ r_3/r_4\Big] \tag{4.2}$$

After getting the vertex in the camera's matrix world, we apply the projection matrix to the result of the previous step using the same equation. The result of this step is the screen position of the vertex closest to the user's mouse.

We compute the Euclidean distance between the mouse position and each vertex to highlight a point. The vertex with the closest distance will be selected and highlighted with a coloured sphere:

$$\min_{\forall v \in V_p} \sqrt{(v.x - m.x)^2 + (v.y - m.y)^2} \tag{4.3}$$

In regard to the faces, we could not directly calculate the Euclidean distance in a 2D screen coordinate system. Instead, we first go through every face and identify faces with the mouse position in them using the triangle area as shown in Eq.(4.4) and Eq.(4.5).

$$area(v_1, v_2, v_3) = (v_1.x - v_3.x) \times (v_2.y - v_3.y) - (v_2.x - v_3.x) \times (v_1.y - v_3.y) \quad (4.4)$$

where v_1, v_2 and v_3 are the vertices of a triangle.

$$area(v_a, v_b, v_c) = area(v_m, v_a, v_b) + area(v_m, v_b, v_c) + area(v_m, v_a, v_c) \quad (4.5)$$

where v_m is the mouse position, and v_a, v_b and v_c is the vertices of a face. For each selected face, we compute the sum of the Euclidean distance between the mouse position and each face vertex. The face with the largest sum of distance will be highlighted using a colored triangle and three colored spheres in each vertex:

$$\min_{\forall f \in F_p} \sum_{\forall v \in f} \sqrt{(v.x - m.x)^2 + (v.y - m.y)^2} \quad (4.6)$$

Chapter 5

Usability Test

Usability test aims to measure the usability goals of the GUI implemented for CROMOSim and also identify potential usability issues. The format of this test is unmoderated and asynchronous. During the test, any improper use of this function by participants will not be monitored, and the test will be recorded for later analysis.

Participants in this usability test are five students around the age of 25, including two females and three males. All participants have related engineering backgrounds and have experience with coding or IMU sensors, which matches our target demographic. To access the GUI of CROMOSim, participants need to use a web browser (Chrome is recommended). On the other hand, screen capture software is required to record participants' performance during tests.

5.1 Methodology

There are two parts to the usability test, including three scenario tasks and four post-test survey questions. The scenario tasks and survey questions are the same for all participants and the detailed test procedure is shown as follows:

1. Participants will be provided with test scenarios, instructions, and sample input files.
2. Participants will perform task scenario #1.
3. Participants will perform task scenario #2.
4. Participants will perform task scenario #3.
5. Participants will answer the post-test survey questions.
6. Participants will have a post-test interview with us if necessary.

We expect the whole test procedure should take about 20 to 30 minutes for participants to complete, including application processing time.

Three test scenarios are listed below which will be provided to participants during the test.

- Scenario #1:
    1. Enter the web page and upload sample video "video_1.mp4"
    2. Set the start and end times to 1 and 9, respectively, and then move forward.
    3. Upload sample CVD results: "cam_1.zip", "dynamic_mask_1.zip", and "output_1.zip" and then move forward.

4. Review generated mesh model and select five connected vertices (three connected faces) to indicate the sensor location.
5. Download the results and close the page.

- Scenario #2:

1. Enter the web page and upload sample video "video_2.mp4"
2. Move forward without indicating start and end times.
3. Upload sample CVD result: "cam_2.zip", "dynamic_mask_2.zip" and "output_2_incorrect.zip".
4. Delete "output_2_incorrect.zip" and upload "output_2.zip" and then move forward.
5. Review generated mesh model and select five connected vertices (3 connected faces) to indicate the sensor location.
6. Download the result and close the page.

- Scenario #3:

1. Enter the web page and upload sample video "video_3.mp4"
2. Move forward without indicating start and end times.
3. Upload sample CVD results: "cam_3.zip", "dynamic_mask_3.zip", and "output_3.zip" and then move forward.
4. Cancel model generation step by "reset all" and exit the page.

After collecting responses from the participants, we will review the video and extract key information from the video for analysis. The measurement of usability performance is based on the following metrics:

- Performance of each task (in seconds).
- Number of errors made per task. (An error will be considered to occur when a user attempts the wrong user action.)
- Completeness of each scenario.

5.2 Test Results and analysis

According to the methodology we introduced previously, we received recorded videos and survey results from each participant. We summarized test results in Table 5.1 shown following:

Table 5.1: Summary Table of Task Performance and Error Counts

	Task Name	Expected Task Performance	Average Task Performance	Errors count
Scenario 1	Upload video	15 s	17 s	0
	Trim video	30 s	76 s	12
	Upload CVD result	30 s	56 s	2
	Select sensor location	100 s	119 s	15
	Generate and download result	10 s	10 s	0
Scenario 2	Upload video	15 s	14 s	0
	Upload CVD result	30 s	37 s	1
	Delete CVD result	10 s	24 s	4
	Select sensor location	100 s	104 s	2
	Generate and download result	10 s	9 s	0
Scenario 3	Upload video	15 s	12 s	0
	Upload CVD result	30 s	34 s	0
	Cancel and exit	5 s	11 s	0

5.2.1 Scenario #1

1. It took a long time for the majority of participants to set start and end times correctly, and they made many errors. One possible reason is that there are not enough hints on the user interface to instruct what users can do on the current page. On the other hand, some participants failed to notice the "Help" button in the right-top corner.

2. Two participants failed to perform "upload zip file" correctly. They moved forward before the file was uploaded successfully. One possible reason is that there is no indicators for file uploading progress, and users is allowed to move forward without completing the current step.

3. Three participants made tiny errors while controlling the mesh model and selecting sensor locations. It can be seen from the observation that most participants did not use the "Help" feature properly, but preferred to try it by themselves. Including some small hints near the 3D canvas might reduce the number of incorrect actions.

5.2.2 Scenario #2

1. Two participants failed to perform "delete zip file re-upload" correctly. They moved forward before deleting or uploading successfully. Similar to the second observation in scenario #1, there is no indicators for file deleting and uploading procedures, and the user is allowed to move forward without completing the current step.

2. One participant still made a few errors while controlling the mesh model and

selecting sensor locations. She used the touchpad while performing the task, and we can notice that the touchpad was incompatible with the model-interaction features.

5.2.3 Scenario #3

1. One participant took a relatively long time to find the "reset all" button for cancelling the model generation step. One possible reason is that the "reset all" button is located on the "sidebar", and it might be hard for some users to connect the sidebar with the current progress page. Including a "cancel" button on the main window may help to improve performance.

5.2.4 Post-test Survey and Interview

1. The first question asks participants to rate their confusion while performing video uploading and trimming tasks. As shown in Fig. 5.1, most of the participants (4 out of 5) felt quite confused (above medium), and only one participant felt a little confused. During the interview, three participants suggested adding new features that allow them to set start and end times directly with the keyboard, and 1 participant suggested adding instructions close to the related buttons.

2. The second question asks participants to rate their confusion while performing zip file uploading and model generation. Similar to the result of the first question, most of the participants (3 out of 5) felt quite confused (above medium), one participant felt a little confused, and one participant felt not confused. During the interview, participants mentioned that they feel uncertain about

the upload and generation progress. They suggested including a progress bar or some indicators in the main window to estimate the time left.

3. The third question requires participants to rate efficiency while selecting sensor location on the mesh model. As a result, shown in Fig. 5.3, two participants thought this function is ineffective (below medium), and the other three believed it is effective (above medium). Through conversations with the participants, we noticed that the mesh model looks messy and participants needed to move their mice very carefully to select desired vertices. On the other hand, some participants expected more available actions to interact with the model, such as resetting the camera position and clearing all selected vertices.

4. The last question asks participants to rate the efficiency of the "help" feature. Three of the participants thought this function is not effective (below medium), and the other two participants believed it is effective (above medium). During the interview, two participants asked for a "help" page with more details, for example, using a video instead of images to visualize instructions and proper actions.

Post-survey questions and the responses from participants are provided in Fig. 4.1 - 4.4 in Appendix A.

Chapter 6

Conclusion and Future work

In this report, we provided the design and implementation of a web application for CROMOSim pipeline. This application comprised two components: a front-end web page as the graphic user interface and a back-end service application. Users could access CROMOSim virtually through our application without any installation steps.

According to the usability test performed previously, our application for CROMOSim reduced the efforts required for learning and using, which satisfied main requirements of the pipeline. However, there are still some improvements that could be conducted for future work:

- For the front-end web page, error messages are provided for some inappropriate user actions. As shown in the usability test result, users are sometimes confused by error messages and feel inconfident in using such an application. We will improve the error handling for future work, identify all potential user errors and add suggestions for fixing problems. On the other hand, we will enhance the appearance of the graphic user interface to improve users' confidence in our

application.

- For the back-end service application, a video is taken from the user and stored in the server's local working directory. However, for file uploading, we only restrict available file types without any security scanning. Since the uploaded files have potential risks to the back-end server, as part of future work, we will add a security framework to scan any input files and ensure the security of our back-end server.

- As part of the pipeline, we need to estimate the global trajectory from input video using robust CVD method. The execution time of this method is unexpected long (about 6 hours for 10 seconds video), which will increase the burden on back-end server. This application includes robust CVD method as an external service. Users must execute robust CVD on their devices or online platforms and upload execution results as input to our application. In the future work, it is expected to find a lightweight replacement for robust CVD method and incorporate it to the application to reduce the required input from the user.

Bibliography

[1] M. Tanenhaus, D. Carhoun, T. Geis, E. Wan, and A. Holland. Miniature IMU/INS with optimally fused low drift MEMS gyro and accelerometers for applications in GPS-denied environments. In *Proceedings of the 2012 IEEE/ION Position, Location and Navigation Symposium*, pages 259–264, 2012.

[2] Norhafizan Ahmad, Raja Ariffin Raja Ghazilla, Nazirah M Khairi, and Vijayabaskar Kasi. Reviews on various inertial measurement unit (IMU) sensor applications. *International journal of signal processing systems*, 1(2):256–262, 2013.

[3] Jindong Wang, Yiqiang Chen, Shuji Hao, Xiaohui Peng, and Lisha Hu. Deep learning for sensor-based activity recognition: A survey. *CoRR*, abs/1707.03502, 2017.

[4] Yonatan Vaizman, Katherine Ellis, and Gert R. G. Lanckriet. Recognizing detailed human context in-the-wild from smartphones and smartwatches. *CoRR*, abs/1609.06354, 2016.

[5] Yilin Liu, Shijia Zhang, and Mahanth K. Gowda. When video meets inertial sensors: Zero-shot domain adaptation for finger motion analytics with inertial

sensors. *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, 2021.

[6] Vitor Fortes Rey, Peter Hevesi, Onorina Kovalenko, and Paul Lukowicz. Let there be IMU data: Generating training data for wearable, motion sensor based activity recognition from monocular RGB videos. UbiComp/ISWC '19 Adjunct, page 699–708, New York, NY, USA, 2019. Association for Computing Machinery.

[7] HyeokHyen Kwon, Catherine Tong, Harish Haresamudram, Yan Gao, Gregory D. Abowd, Nicholas D. Lane, and Thomas Ploetz. IMUTube: Automatic extraction of virtual on-body accelerometry from video for human activity recognition. *CoRR*, abs/2006.05675, 2020.

[8] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J. Black. SMPL: A skinned multi-person linear model. *ACM Trans. Graphics (Proc. SIGGRAPH Asia)*, 34(6):248:1–248:16, October 2015.

[9] Johannes Kopf, Xuejian Rong, and Jia-Bin Huang. Robust consistent video depth estimation. 2021.

[10] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. *CoRR*, abs/1812.08008, 2018.

[11] Muhammed Kocabas, Nikos Athanasiou, and Michael J. Black. VIBE: Video inference for human body pose and shape estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Appendix A

Post-survey questions and responses

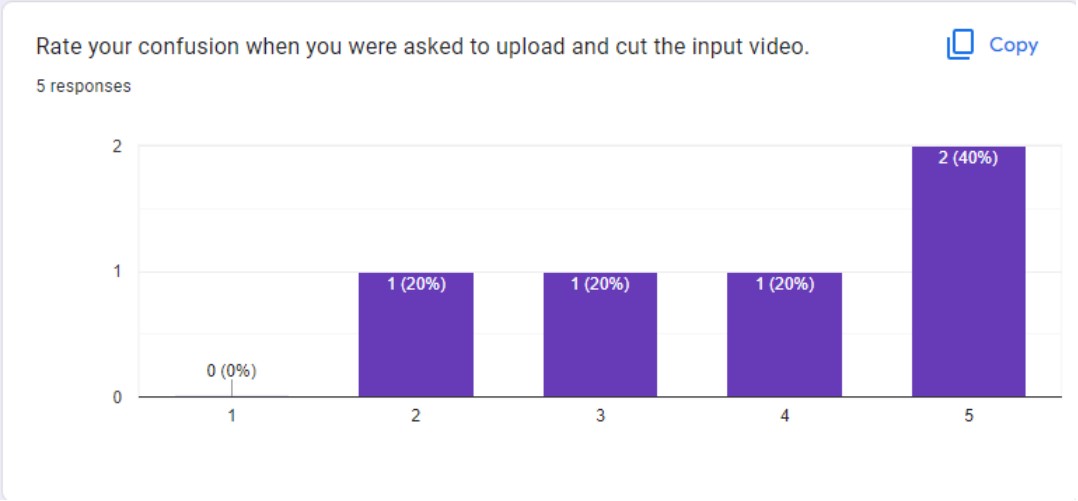

Fig. 4.1: Post-test survey question #1

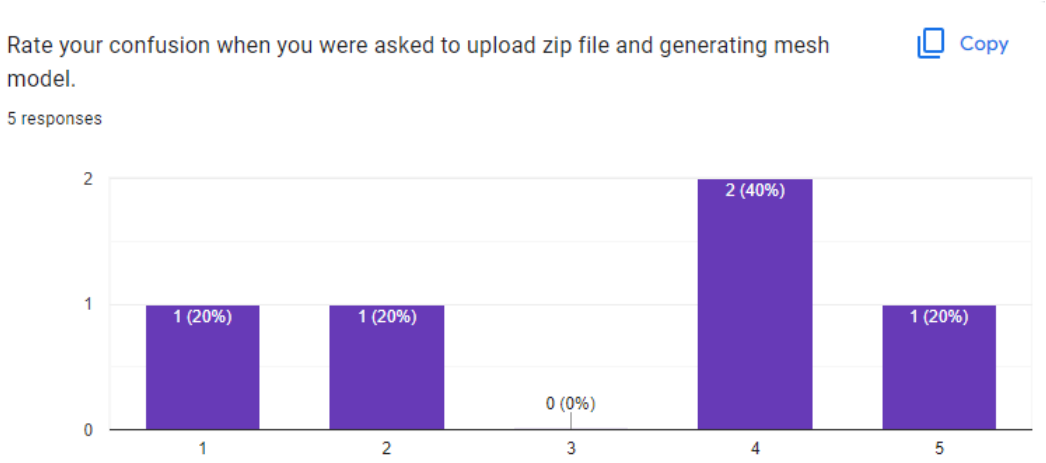

Fig. 4.2: Post-test survey question #2

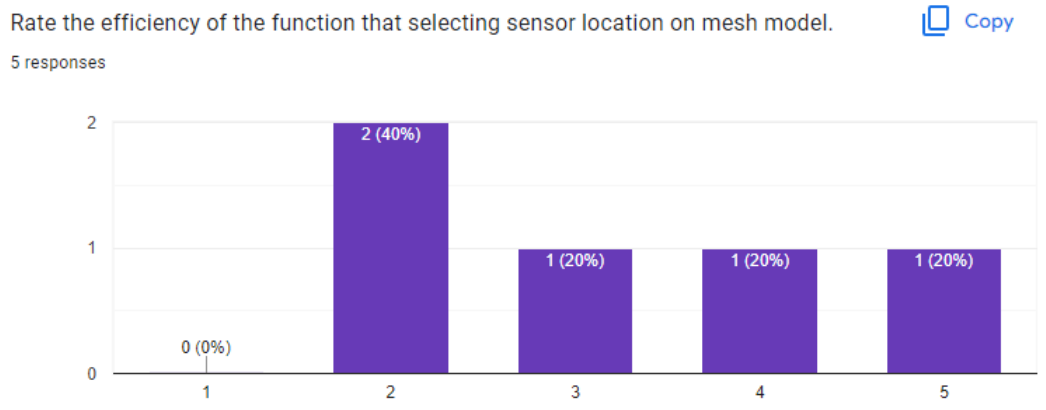

Fig. 4.3: Post-test survey question #3

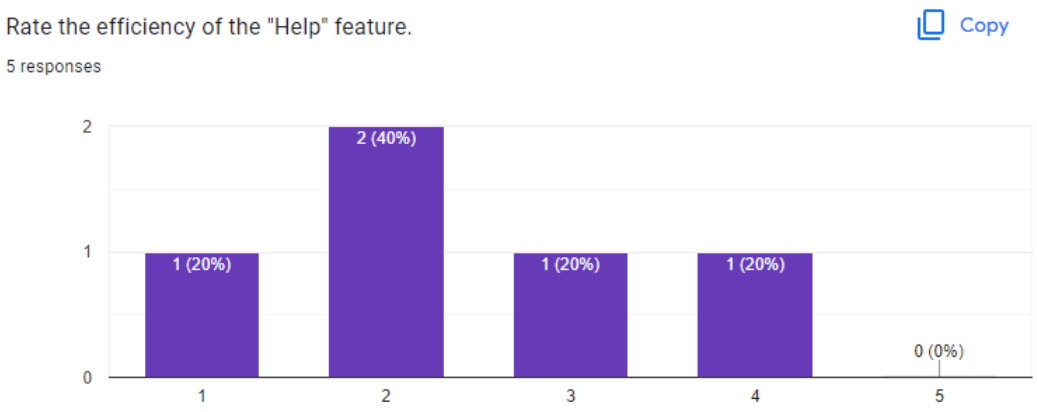

Fig. 4.4: Post-test survey question #4