UNIVERSITY OF
LIVERPOOL

COMP390
2023/24

Computer Vision and AR for Endovascular Intervention

Student Name: [Yulin Huang]

Student ID: [201676465]

Supervisor Name: [Anh Nguyen]

DEPARTMENT OF COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Anh Nguyen, for his invaluable guidance and support throughout my research. His insights and expertise have been fundamental to the success of this work.

Special thanks also go to Tudor Jianu for his advice and contributions that greatly enhanced this project.

I would like to extend my heartfelt thanks to my family. Their unwavering support and love have been my strength and motivation throughout this journey.

I am also immensely grateful to my girlfriend, Lucy Zhao, for her understanding and encouragement during the course of my thesis. Her support has been a cornerstone of my personal and academic journey.

Furthermore, I wish to acknowledge the critical role played by open-source libraries in the realization of this project. I extend my thanks to the developers of OpenCV[1] and SciKit-Surgery Augmented Reality[2] for providing essential tools and references that significantly contributed to my project.

This work would not have been possible without the collective support and encouragement from each of these individuals and resources. Thank you.

UNIVERSITY OF
LIVERPOOL

COMP390 2023/24

Computer Vision and AR for Endovascular Intervention

DEPARTMENT OF COMPUTER SCIENCE

University of Liverpool
Liverpool L69 3BX

Contents

Abstract

This project is divided into two parts, Part1: Real-world Model Interaction and Tracking and Part2: Endovascular Intervention Simulation.

Part1 includes Real-Time Video Capture and Overlay, which could overlay simulated models on real-world images, such as those captured from live camera or uploaded video. In addition, ArUco marker detection in video frames can be performed and the models can be tracked according to the markers. This process was implemented through the OpenCV library and the SciKit-Surgery Augmented Reality[2] library in combination with Pyside6.

The goal of the second part was to create an Endovascular Intervention simulation tool for use in Augmented Reality (AR) and Mixed Reality (MR) environments. This part was developed through Unity combined with the Microsoft Mixed Reality Toolkit-Unity (MRTK) development framework.

Statement of Ethical Compliance

Data Category: A
Participant Category: 0
I confirm that I have read the ethical guidelines and will follow them during this project. Further details can be found in the relevant sections of this proposal.

1 Introduction & Background

This project is dedicated to the development of an open-source simulator of Endovascular Intervention that can be deployed in AR/VR devices, which is designed with two main parts: Part1: Real-world Model Interaction and Tracking and Part2: Endovascular Intervention Simulation.

Part1 focuses on the superposition of the model and the real-world scene in the augmented reality environment, and the positioning in the virtual scene, while Part2 focuses on the endovascular intervention simulation, with an emphasis on the interaction of the Catheter and the Aorta walls during the simulation process, and their adaptation to AR/VR devices.

The background research for this project is focused on the feasibility and implementation of Endovascular Intervention simulator in VR and AR systems, such as HoloLens. Also for the interaction between the catheter and the aorta during endovascular manipulation. Attention is also given to methods for overlaying display and virtual scenes in AR/VR, and research is desired on performing localisation in virtual environments relative to the display environment.

In the past, physics simulations, such as collisions and deformation of objects, were often simulated using the Mujoco platform. MuJoCo Physics Engine are lauded for their ability to orchestrate complex simulations, especially in virtual spaces where high-fidelity interaction is critical[3]. The integration with Unity, further amplifies its capabilities. The combined strengths of MuJoCo and Unity form a comprehensive platform, melding the robust physics simulation of MuJoCo with Unity's proficiency in real-time rendering and interaction[4].

In addition, the emergence of magnetic resonance-safe endovascular robotic platforms and the development of endovascular catheterised robotic systems equipped with magnetically-controlled haptic force feedback demonstrate the direction of the industry towards perceptible force sensing and hyper-realistic feedback in training scenarios.[5], [6]

The expansion into augmented reality (AR) for surgical applications has also been a significant point of interest in recent literature. The feasibility and benefits of using Microsoft HoloLens for holographic augmented reality in endovascular surgery have been explored, with findings supporting the device's potential for clinical application[7]. Similarly, studies on wearable AR navigation systems for surgical telementoring based on Microsoft HoloLens indicate that such technology could revolutionize surgical education and remote guidance[8]. The efficiency of marker

tracking for surgical navigation using both mono and stereo vision in Microsoft HoloLens underscores the device's applicability to the medical field[9]. Additionally, the Microsoft HoloLens 2 has been analyzed extensively for its use in medical and healthcare contexts, showing promising prospects for future advancements[10].

About Specific Terminology or Acronyms In order that the general computer science reader can accurately understand any specific terminologies or acronym that appears in this paper, a glossary has been created at the end of the paper and hyperlinks have been used wherever a specific terminology or acronym appears in the text, which can be clicked on to jump to the corresponding section of the glossary.
Hyperlink to glossary

2 Design & Implementation

2.1 Part1: Real-world Model Interaction and Tracking

In this part, I used OpenCV[1] and SciKit-Surgery Augmented Reality[2] libraries for image processing and model tracking. Qt[11] is used to design the graphical interface, and VTK[12] to build a overlay window[13]. OpenCV and SciKit-Surgery libraries help me process images and track Aruco Marker[14] within video steaming. Qt allows me to create a user interface that ensures user could more easily change multiple settings, which can enhance the user experience. VTK and SciKit-Surgery Augmented Reality library are the management of overlay and multi-layer video rendering in my project. An ArUco Marker Generator is also included, which enables the user to generate different ArUco Markers and save them. This section will detail the system's design, focusing on System Components and Organization, Data Structures and Algorithms, User Interface Design, and Design Notation and Diagrams.

2.1.1 Design

1. **System Components and Organization**
   The project is structured into three primary components, each responsible for distinct functionalities within the system. Here's a detailed breakdown of these components and their organization:

   (a) **Frontend - User Interface**
   The system's user interface is developed using Qt, which is a framework that enables the creation of graphically applications. The main class controlling the UI is *Overlay_and_Tracking.py*, which serves as

the central hub for user interactions and display functionalities. This class manages the overlay of models on video streaming and provides interactive buttons for users to control various settings, such as model color, video source, models uploading and changing, or adjusting ArUco marker types and sizes, and more.

(b) **Backend - Helper Classes**
The backend is composed of various helper classes, each set to handle specific tasks:

- *Image Capture:* The *video_source.py* class handles the acquisition of video streams from various sources, including live cameras and recorded media. It is responsible for configuring camera settings, initializing video capture, and video frame cropping to adapted to screen size. This component ensures the reliability and stability of video feed intake.
- *Model Loading:* Managed by *model_loader.py*, this component is important for the model loading and initializing. It loads ".stl" model files from external files, sets up texture mapping, and prepares the models for real-time overlay. The class also checks for errors in model data to prevent crashes or rendering issues during operation. It also optimizes the structures of model data to enhance rendering efficiency and reduce memory overhead.
- *Model Overlay:* The *overlay_window.py* is central to integrating 3D models with live or recorded video. With the help of VTK, this module could set up a multi-layered rendering environment where each layer can independently handle elements like video backgrounds, 3D models, or some GUI overlays(In the future, maybe.).
- *Transform Management:* The *transform_manager.py* class provides a method for managing 4x4 transformation matrices crucial for spatial adjustments of models in 3D space. It stores and retrieves transformations efficiently. And allow it for dynamic modifications of object orientations and positions.
- *ArUco Marker Tracking:* Functionality provided by *arucotracker.py* includes detecting and decoding ArUco markers from the video stream using OpenCV. This module calculates position and orientation of detected marker, and handle the spatial position data for model tracking.

(c) **Additional Component - ArUco Marker Generator**
An independent component in the system is the ArUco Marker Generator, managed by *Aruco_Generator.py*. This tool allows users to

select and visualize different ArUco markers. Users can also save these markers as separate image files.

Organization:

The system's architecture is designed to easy maintenance and scalability. The modular nature of the helper classes allows for isolated development and testing, which enhances the system's robustness and flexibility. This organization simplifies development and testing and enables the integration of additional functionalities in the future with minimal disruption to the existing system.

2. **User Interface Design**

In this section, I will show the user interface design of the two main applications of my Part1. Both applications use the Python-based Qt package for user GUI design. I will also provide a brief description of the Main menu and the ArUco Marker Generator interface.

(a) **Main Menu(Overlay and Tracking)**

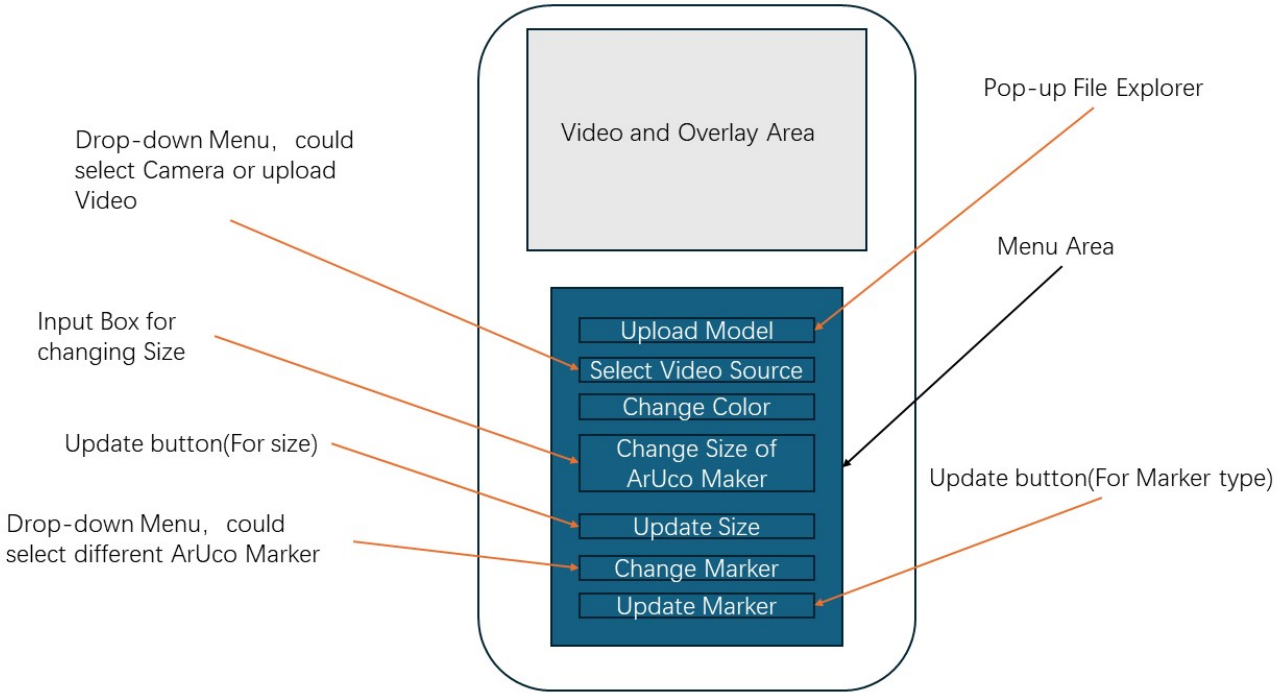

Figure 1: Design of Overlay and Tracking Main Menu

The main menu of Overlay and Tracking is the central hub of the application, providing users with access to all the functionalities. The Main Menu consists of the Video and Overlay Area, which displays the video image and overlay of the model, and the Menu Area, which is responsible for setting various parameters. The menu includes buttons for model uploading, video source selection, ArUco marker settings, and model color adjustments.

When you click on the "Upload Model" button, a file explorer will pop up to select the folder where you want to upload the model. The program will read two files in the folder, one is the model file in ".stl" format, and the other is a file named "colours.txt". the first field in the txt file is the file name of the model, and the last three fields are The first field in the txt file is the file name of the model, and the last three fields are the RGB data of the default colours of the model (the model colours can be changed in the application interface after uploading). After uploading the model will be overlayed onto the video display as Overlay. The second button is the video source selection button, when clicked, a drop-down menu with three options will appear, "Camera1", "Camera2" and "Upload Video Camera1", "Camera2" and "Upload Video". The app will default to "Camera1" as the video source when it is initially launched. Selecting "Camera1" or "Camera2" will directly switch to the corresponding video source. After selecting the "Upload Video" option, an "Upload Video File" button will appear at the bottom. Clicking on the "Upload Video File" button will start the file explorer, you can select ".mp4", ".avi", ".webm" format video for uploading. After the video has been uploaded the tracking of Overlay and ArUco markers will start automatically if there is already an uploaded model present. Similarly, the video source can be switched back to the Camera input by selecting Camera1. After uploading the model, you can change the colour of the model by clicking the "Change Model Color" button.

To ensure the accuracy of ArUco marker tracking, you need to modify the "ArUco Dictionary" to the corresponding dictionary of the marker you are using and click "Update ArUco "button. You also need to change the "ArUco Marker Size", which represents the size of the real ArUco marker in millimetres, and click the "Update Marker Size" button when you are done.

(b) **ArUco Generator**

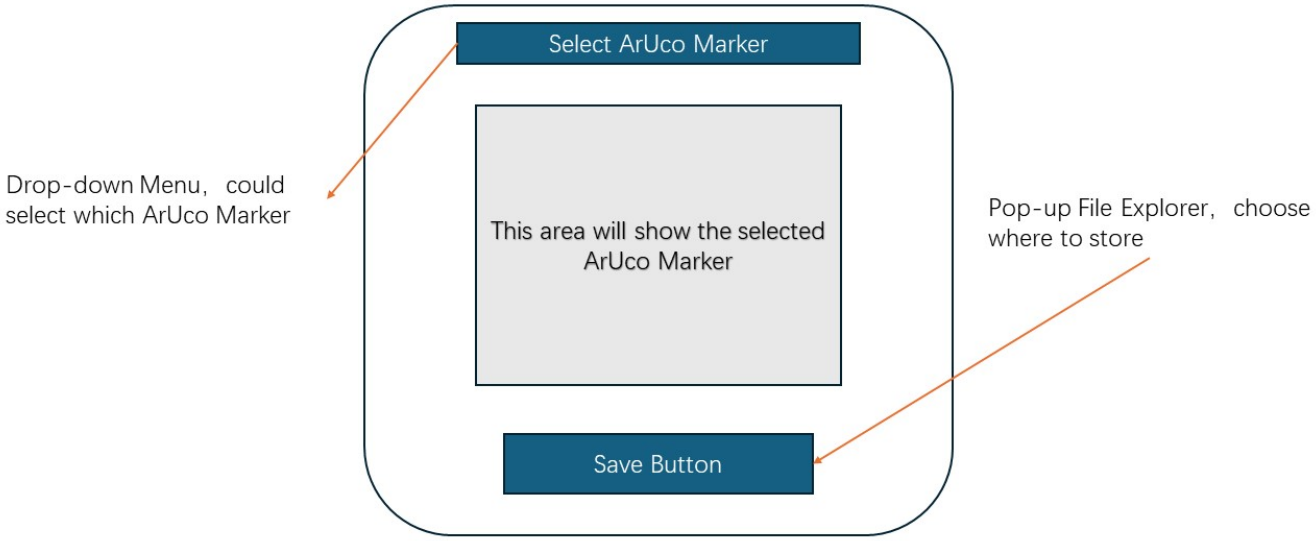

Figure 2: Design of ArUco Generator

Figure 3: ArUco Generator

The ArUco Generator interface allows users to select different ArUco markers from a predefined list and visualize them in real-time. The interface provides a clear preview of the selected marker and allows users to save the marker as an image file for later use. The design is simple and user-friendly, with intuitive controls for easy marker customization.

You can first select the ArUco marker you want to use by launching the drop-down menu via "Select Aruco" and the selected ArUco marker will be displayed on the screen in real time. After that, you can click the "Save Marker" button to start the file explorer to save the marker, the file name will be set to the corresponding dictionary of the ArUco marker by default.

3. **Design Notation and Diagrams**

   (a) **Use Case Diagrams**

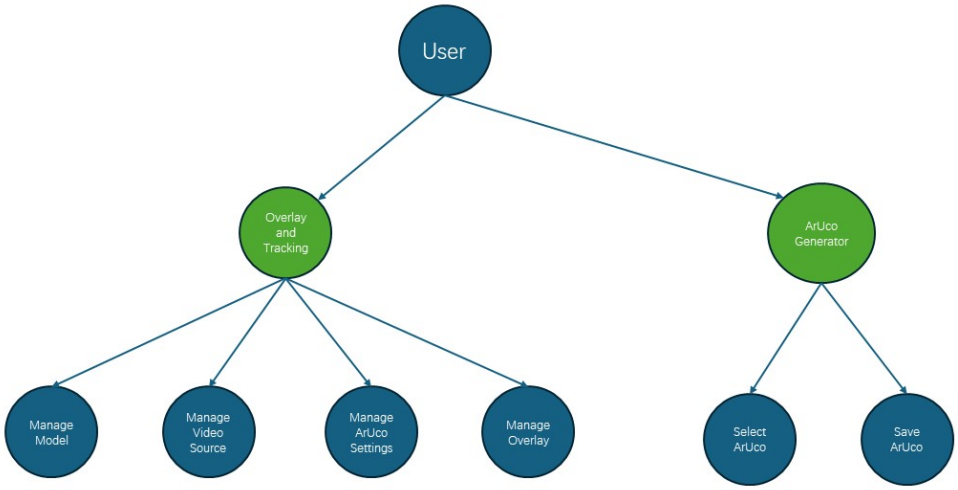

Figure 4: Use Case Diagrams

   (b) **Data Flow Diagrams**

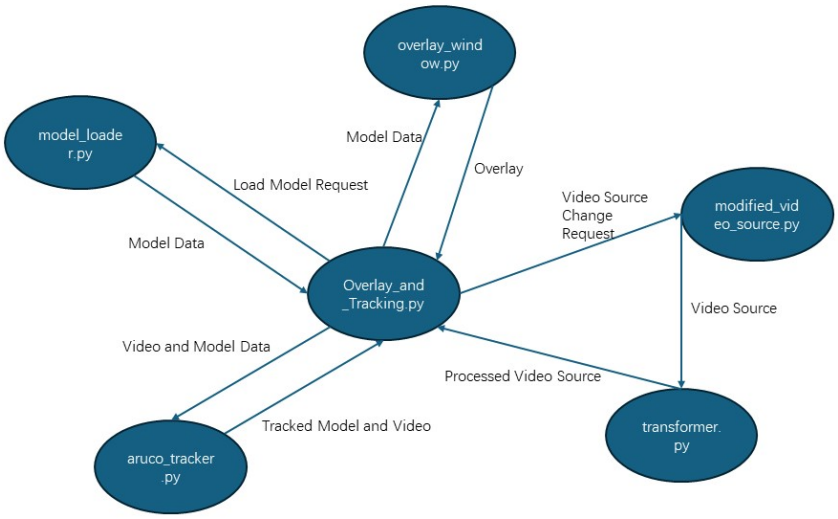

Figure 5: Data Flow Diagrams

2.1.2 Implementation

1. **Backend Implementation**

   (a) **Pre-processing for video capture and upload**

   This part is mainly controlled by *video_source.py* and is combined with a front-end Qt-based user interface to manage the core video processing functions. A robust video source management and processing system is implemented in *video_source.py* through the *TimestampedVideoSource* class and *VideoSourceWrapper* class. The system allows the user to interactively control the selection of video sources through a graphical interface, and it ensures accurate video stream processing.

   - *Video Source Configuration and Management:*
     The *TimestampedVideoSource* class uses the OpenCV library's, emphcv2.VideoCapture method to initialise the video capture from the camera number or file path. It can support basic video capture functionality or provide the ability to validate the size of the video stream to ensure that the resolution is compatible with the system.

If the resolution size provided by the setting is valid, the system sets the resolution accordingly. If the resolution is not valid, the system uses the camera's settings by default. If the requested resolution option is not supported by the camera, the system throws an error, which effectively prevents runtime problems that may result from unsupported video configurations and increases system stability and reliability.

- *Synchronisation and Timestamping:*
  The *TimestampedVideoSource* class also serves to synchronise the video stream with the modelled overlay, embedding a timestamp with the current date and time at each frame captured. Each time a video frame is successfully read, the *datetime.datetime.now()* method gets the current system time. This timestamp is then converted to a string or other format and stored or passed along with the video frame, ensuring that each frame has an exact timestamp. In Augmented Reality (AR) or Virtual Reality (VR) systems, quasi-timestamping this timestamp ensures alignment between the 3D model and the actual environment.With such a timestamping feature, *TimestampedVideoSource* can ensure the synchronisation of the video frames for the subsequent synchronisation of the model overlay.
- *Handles multiple video sources:*
  The *VideoSourceWrapper* class manages multiple video sources, allowing the addition of camera inputs or the uploading of video files, and verifying their existence and compatibility. A centralised method for releasing all video sources is provided in this class to ensure that resources that are no longer needed are managed and cleaned up in a timely manner. Each video source is encapsulated in an instance of the *TimestampedVideoSource* class, which allows each source to be controlled and processed independently. This feature ensures that the system load is at an appropriate level.
- *Retrieve and display video frames:*
  Retrieval of frames is achieved by system calls to the *read()* method of each video source, which is implemented through OpenCV's *cv2.VideoCapture.read()* . This method captures a single frame from the video source and checks to see if it was successfully captured. If successfully captured, the frame is tagged using the current timestamp, ensuring that each frame has an accurate time reference. When processing multiple video sources, the system ensures synchronisation between frames. Synchronisation of frames

is achieved by using the same frame rate setting for all sources and matching frames from different sources with timestamps. This allows for smoothness and synchronisation in the case of multiple video source inputs.

pseudocode for the *video_source.py*:

```
Class TimestampedVideoSource:
    Initialize(source_num_or_file, optional dims):
        Try:
            Open video source with cv2.VideoCapture(source_num_or_file)
        If source not opened:
            Raise RuntimeError("Failed to open Video camera")

        If dimensions provided:
            Validate and set video resolution:
                Check if dimensions are integers and >= 1
                Set resolution using cv2.VideoCapture properties
                If resolution setting fails:
                    Raise ValueError("Requested resolution not supported")

        Initialize empty frame array based on video dimensions
        Initialize timestamp

    Read():
        Capture frame and current time using cv2.VideoCapture.read()
        If frame captured successfully:
            Update timestamp with current datetime
        Return frame, timestamp

    Release():
        Release video source using cv2.VideoCapture.release()

    Update source(new_source):
        Release current source
        Reinitialize with new source

Class VideoSourceWrapper:
    Initialize():
        Create an empty list for storing video sources

    Add camera(camera_number, optional dims):
        Validate camera input (check if camera number is valid)
        Add camera source with optional dimensions to the list

    Add file(filename, optional dims):
        Validate if filename exists and is a valid file
        Add file source with optional dimensions to the list
```

```
42
43      Get next frames():
44          For each source in the list:
45              If source is opened:
46                  Read frame from source
47              Collect and return all frames
48
49      Release all sources():
50          For each source in the list:
51              Release source
```

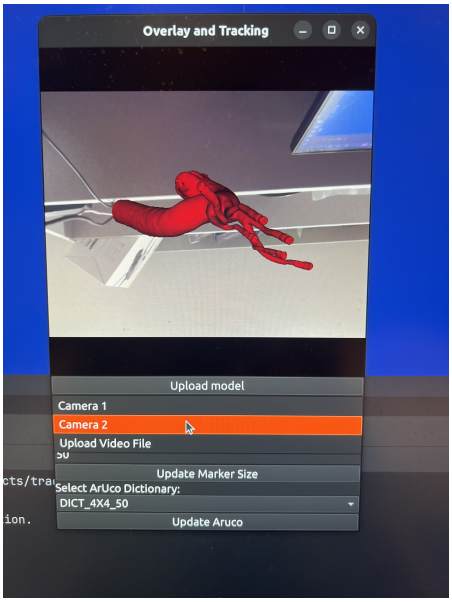

Figure 6: Video Source Switching

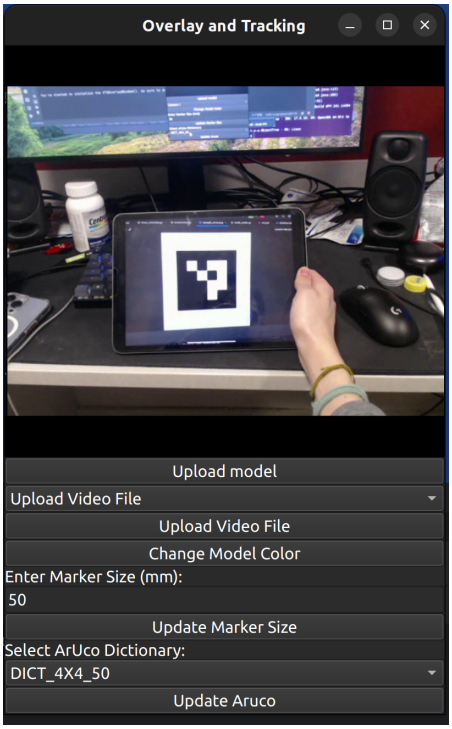

Figure 7: Video Upload

(b) **Image Processing Algorithms in Multi-Layer Video Rendering** In a video rendering system, video frames need to be dynamically managed to create complex visual effects, such as models and real-time video overlays in this project. This also involved real-time adjustments to the Alpha Blending[15] and video. A greyscale image with alpha blending of the RGBA stream was used to precisely control transparency and layering effects to enable the superimposition of a layer's frame (e.g. the model) onto the original frame[16]. Ensuring overlay accuracy and visual fidelity is crucial for applications such as augmented reality[17].

In addition, it is necessary to update and align the video frames to the appropriate layers by adjusting the data range according to the size of the incoming video frames. This incorporation of real-time processing improves the continuity of the video image by preventing visual interruptions caused by frame misalignment[18].

I also introduced Adaptive Exponential Smoothing into the multi-layer video rendering system (*Overlay_and_Tracking.py*). This enhances image processing, such as when processing video streams involving com-

plex dynamic scenes[19]. By dynamically adjusting its smoothing parameter (Alpha), Adaptive Exponential Smoothing (AES) is able to more accurately adapt to changes in content within a video frame, such as lighting adjustments, scene switching, or object movement, and can reduce visual jitter and blurring due to rapid changes [19].

Compared with the traditional Exponential Moving Average, the adaptive feature of AES has a greater advantage. Exponential Moving Average (EMA), although fast in processing and low in computational cost, may not be able to adequately adapt its fixed smoothing parameters to real-time changes in the video content in the face of complex scene variations, thus affecting the final image quality[20]–[22]. In a multilayer rendering system, combining AES for real-time video transmission and dynamically adjusting the smoothing parameters according to the content differences between the previous and previous frames can maintain the continuity and naturalness of the visual effects, especially when dealing with moving objects and changing backgrounds. In addition, AES's also better handles scenes with large lighting variations, maintaining the balance of colors and shades of light and dark [23].

I have similarly experimented with Complex Exponential Smoothing, and while it excels in handling data with clear trends and cyclical variations, its application in video rendering systems may not be as straightforward and effective as AES. Because Complex Exponential Smoothing (CES) is designed to provide a more comprehensive understanding of the multiple influences on the data [24], [25], its use in non-predictive applications may lead to overly complex processing and increased computational burden.

Therefore, AES is ultimately used in video rendering systems to respond more directly to real-time changes in video content, reduce visual jitter, and improve the viewing experience.

To summarize, AES can improve image stability and visual quality, as well as enhance the system's responsiveness to environmental changes. By intelligently adjusting processing parameters, AES helps to ensure high efficiency while also adapting to visual jitter or lighting changes that may be encountered with ArUco marker tracking.

Code for the AES:

```python
class SmoothedTransform:
    def __init__(self, alpha, min_alpha=0.01, max_alpha=0.9):
        """
        Initializes the adaptive smoothing transform class.

        Parameters:
        alpha (float): The initial smoothing factor.
        min_alpha (float): The minimum allowable value for alpha
        to prevent it from becoming too low.
        max_alpha (float): The maximum allowable value for alpha
        to prevent it from becoming too high.
        """
        self.alpha = alpha
        self.min_alpha = min_alpha
        self.max_alpha = max_alpha
        self.transform = None
      def adjust_alpha(self, new_transform):
            """
            Adjusts the smoothing factor alpha based on the
            difference between the new transform and the
            current transform to
            better adapt to recent data changes.

            Parameters:
            new_transform (float): The new data point used to
            update the transform.
            """
            if self.transform is not None:
            # Calculate the absolute difference between the current
            # and new transforms
            error = abs(new_transform - self.transform)
            # Dynamically adjust alpha based on the error,
            # inversely scaling it
            self.alpha = max(self.min_alpha, min(self.max_alpha,
            1 / (1 + error)))

      def update(self, new_transform):
            """
            Updates the current transform with a new data point using
            adaptive exponential smoothing.

            Parameters:
            new_transform (float): The new data point to incorporate
            into the smoothed data.

            Returns:
            float: The updated transform value.
            """
```

```
            if self.transform is None:
            # If no transform has been set yet, initialize it
            # with the new transform
            self.transform = new_transform
            else:
            # Adjust alpha based on the new data point
            self.adjust_alpha(new_transform)
            # Apply the adjusted alpha to compute
            # the new smoothed transform
            self.transform = self.alpha * new_transform +
            (1 - self.alpha) * self.transform
            return self.transform
```

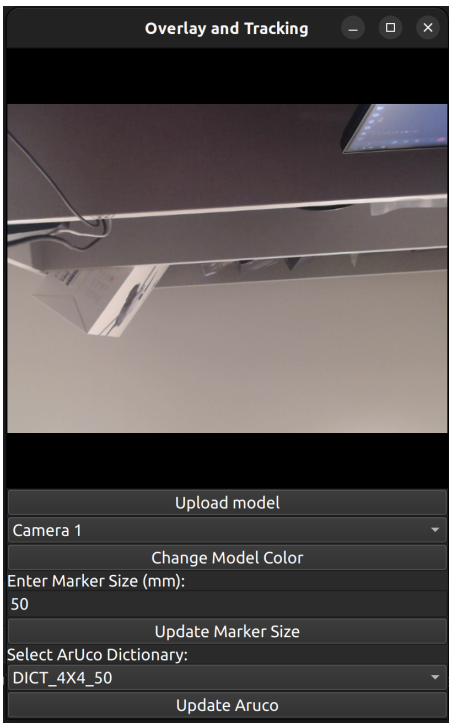

Figure 8: Overlay and Tracking Main Menu(Without Overlay Activated)

(c) **Model Rendering and Overlay**
It is mainly implemented by *overlay_window.py*, in which the *VTKOverlayWindow* class combines the functionality of VTK, Qt and OpenCV for real-time video streaming and model overlay.

- *Multi-layer rendering settings for video and models:*
  When initialising the *VTKOverlayWindow* class, the VTK library's *vtkRenderWindow* and *vtkRenderer* are used to build a multi-layered

rendering environment[26]. This multi-layer architecture allows independent control of content rendering on each layer, where the bottom layer is typically used to display live or pre-recorded video, while the top layer is used to add 3D models and other GUI elements[27], [28]. Such a layer setup ensures proper visual overlay between the rendered elements. The *SetNumberOfLayers* method is used to define the number of layers in the rendering window, and the *SetLayer* method is used to assign each renderer to a specific layer. This layering system ensures that the video stream and 3D models are correctly displayed and overlaid in the final output[26].

Code for the multi-layer rendering settings:

```
# Create a rendering window and set multiple layers
renderWindow = vtk.vtkRenderWindow()
renderWindow.SetNumberOfLayers(3)

# Create renderers for each layer
videoRenderer = vtk.vtkRenderer()
videoRenderer.SetLayer(0)  # Bottom layer for video
modelRenderer = vtk.vtkRenderer()
modelRenderer.SetLayer(1)  # Middle layer for 3D models
uiRenderer = vtk.vtkRenderer()
uiRenderer.SetLayer(2)  # Top layer for UI components

# Add renderers to the rendering window
renderWindow.AddRenderer(videoRenderer)
renderWindow.AddRenderer(modelRenderer)
renderWindow.AddRenderer(uiRenderer)
```

- *Import and format conversion of video streams:*
  In order to implement VTK overlay, the video frames captured by OpenCV need to be converted into a format that VTK can handle. Here the video frames are converted by using VTK's *vtkImageImport* method. The conversion process requires setting the correct data types and sizes and updating the memory pointers to the video frames to ensure that the video stream data captured from OpenCV can be integrated into the VTK rendering process. This ensures synchronisation and integrity of the video stream before and after conversion without causing video lag in VTK.[29], [30]

Code for the video stream conversion:

```
# Initialize video capture
cap = cv2.VideoCapture(0)

```

```
# Read a frame from OpenCV and import it into VTK
ret, frame = cap.read()
vtkImage = vtk.vtkImageData()
vtkImage.SetDimensions(frame.shape[1], frame.shape[0], 1)
vtkImage.SetScalarTypeToUnsignedChar()

# Create a VTK image importer and connect it to the image data
imageImporter = vtk.vtkImageImport()
imageImporter.SetInputData(vtkImage)
imageImporter.SetImportVoidPointer(frame.data, frame.nbytes)
imageImporter.Update()
```

- *Dynamic rendering and adjustment of 3D models:*
  Rendering of 3D models is achieved by using components such as VTK's *vtkActor* and *vtkPolyDataMapper*. These components support loading models from external files (e.g. STL format) and allow transformations such as translation, scaling, rotation, etc. to be applied to the model. These transformations of the model can be adjusted in real time to support dynamic user interaction and visual modifications[31]. The model can be rendered in the VTK overlay by using the *vtkPolyDataMapper* map the model data to geometry and set the actor using the *vtkActor* and *AddActor* method to add the model to the corresponding rendering layer. Then can use the *SetPosition* and *SetOrientation* method to adjust the model position and orientation to match the video[32].The function of modifying the position of the model will play an important role in section (f) Marker Detection and Tracking.

Code for the dynamic rendering and adjustment of 3D models:

```
# Load a 3D model from a file
modelReader = vtk.vtkSTLReader()
modelReader.SetFileName("model.stl")

# Map the model data to geometry
mapper = vtk.vtkPolyDataMapper()
mapper.SetInputConnection(modelReader.GetOutputPort())

# Create an actor for the model and add it to the renderer
actor = vtk.vtkActor()
actor.SetMapper(mapper)
modelRenderer.AddActor(actor)

# Adjust model position and orientation to match the video
# The position data can be varied depending on
```

```
# tracking data in (f) Marker Detection and Tracking
actor.SetPosition(10, 0, 0)
actor.SetOrientation(0, 90, 45)
```

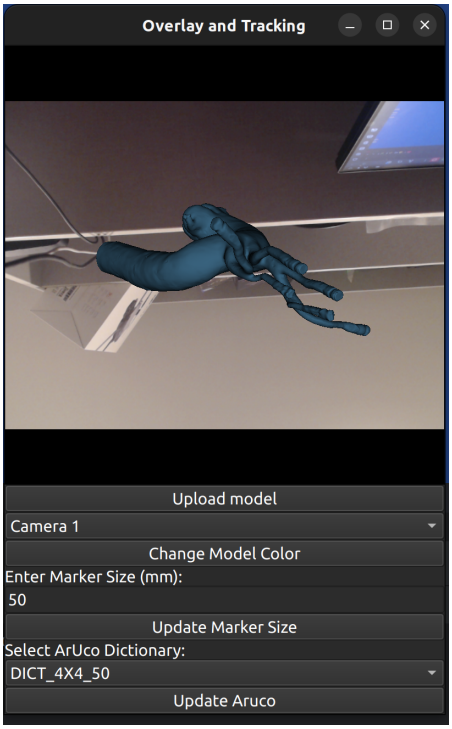

Figure 9: Overlay and Tracking Main Menu(With Overlay Activated)

(d) **Optimization of Visual Effects and Depth Processing:**
To enhance the visual quality of rendering, VTK's Depth Stripping technology can be enabled. Depth peeling is suitable for rendering transparent or semi-transparent objects and effectively solves visual artefacts and incorrect transparency problems during rendering. Through layer-by-layer rendering and compositing, Depth Stripping ensures visual correctness and is suitable for use in complex overlay or multi-object scenes.[32], [33] The *UseDepthPeelingOn* method is used to enable Depth Stripping, and the *SetMaximumNumberOfPeels* and *SetOcclusionRatio* methods are used to set the maximum number of peels and the occlusion ratio, respectively, to control the rendering quality and performance.

Code for Optimization of Visual Effects and Depth Processing:

```
# Enable depth peeling for better transparency handling
modelRenderer.UseDepthPeelingOn()
modelRenderer.SetMaximumNumberOfPeels(100)
modelRenderer.SetOcclusionRatio(0.1)
```

(e) **Precise Control of Camera and Viewing:**
During the implementation of model overlay using VTK, it is important to maintain an accurate overlay between the 3D model and the background video stream. When using different cameras, deviations in the model overlay may occur due to different camera parameters such as focal length, projection, and position. In order to make the model appear accurately at the right position in the background video, the parameters of the camera should be modified. By getting the camera's focal length, position and projection parameters to modify the variables in video processing, the viewpoint between the 3D model view and the video stream background can be consistently matched when using a camera with different characteristics. It might be necessary to additionally adjust the cropping range and field of view of the camera to ensure the correct range of visual output. The *vtkCamera* class is built to control the camera settings. Method like *SetPosition*, *SetFocalPoint*, and *SetViewUp* are used to adjust the camera's position, focus point, and view direction, respectively.[34], [35]

Code for Precise Control of Camera and Viewing:

```
# Configure the camera to match the video view
camera = modelRenderer.GetActiveCamera()
camera.SetPosition(0, 0, 100)
camera.SetFocalPoint(0, 0, 0)
camera.SetViewUp(0, 1, 0)
```

(f) **Marker Detection and Tracking**

- *Loading Calibration Data:*
  Obtaining accurate camera calibration data before performing the spatial localisation of the model is necessary. It is first necessary to load the projection matrix and distortion coefficients of the camera in (e) Precise Control of Camera and Viewing, which have been acquired and stored in that section. This data is then read to calibrate the subsequent image processing and marker recognition so that it accurately reflects the actual camera viewing angle. The use of different calibration data affects the model's spatial localisation function and marker tracking accuracy.

Pseudocode for the Loading Calibration Data:

```
Function _load_calibration(textfile):
    Read 'projection_matrix' from
    '(e) Precise Control of Camera and Viewing'
    Read 'distortion' from
    '(e) Precise Control of Camera and Viewing'
    Return projection_matrix, distortion
```

- *Initializing Video Source and Tracker Configuration:*
  Afterwards, the camera is initialised and the required tracking parameters are set according to the camera configuration obtained in the first step. This includes setting the video source selected by the user (e.g., camera or video file) and configuring the properties of the video stream (e.g., resolution and frame rate.). In addition, the associated rigid body is set up based on the acquired configuration information and the rigid body is used to compute its position and orientation based on the detected markers.

Pseudocode for the Initializing Video Source and Tracker Configuration:

```
Method initialize_video_capture(video_source, configuration):
    If video_source is not 'none':
        self._capture = cv2.VideoCapture(video_source)
        If "capture properties" in configuration:
            For each property in configuration["capture properties"]:
                self._capture.set(cv2 property, value)
```

- *Load ArUco dictionary:*
  This section is used to load the dictionary corresponding to the ArUco token selected by the user. This dictionary represents the index corresponding to each ArUco. The ArUco dictionary is obtained and returned to OpenCV for tracking specific ArUco tokens.

Pseudocode for the Load ArUco dictionary:

```
Function get_aruco_dictionary(configuration):
    aruco_dict =
    cv2.aruco.getPredefinedDictionary(cv2.aruco.DICT_ARUCO_ORIGINAL)
    Return aruco_dict
```

- *Real-Time Marker Detection using OpenCV:*
  In the *detect_markers* method, detection of ArUco markers from a given video frame will be implemented. Firstly, the dictionary corresponding to the specific ArUco selected by the user is obtained

from the previous section. Next, the default detection parameters are created using OpenCV's ArUco module. The method will identify the markers in the video frame by calling the *cv2.aruco.detectMarkers* function. This function accepts three variables: the input video frame, the ArUco dictionary and the detection parameter. If the function detects markers in that video frame, it will mark the *marker_corners* not null, and then it will use the *cv2.aruco.drawDetectedMarkers* function to draw those markers on the frame for visual observation (and optionally, not mark them). Finally, the method returns the corner positions of the detected markers *marker_corners* and the *ID marker_ids* of the corresponding markers, providing positional data for the model tracking in later sections.

Pseudocode for the Real-Time Marker Detection and Data Capture:

```
Method detect_markers(frame):
    ar_dict = get_aruco_dictionary(configuration)
      # Default parameters
    parameters = cv2.aruco.DetectorParameters_create()
    marker_corners, marker_ids, _ =
    cv2.aruco.detectMarkers(frame, ar_dict, parameters=parameters)
    If marker_corners are not empty:
      # Optional: for visualization
        cv2.aruco.drawDetectedMarkers(frame, marker_corners)
    Return marker_corners, marker_ids
```

- *Estimate the pose of detected markers:*
  In the *estimate_pose* method, OpenCV's ArUco module is used to estimate the pose of the detected markers, which includes their rotation vector (RVEC) and translation vector (TVEC). The marker corners *marker_corners* identified from the video frames in the previous section are first checked to see if they are empty. If it is not null, it means that the marker detection was successful before the pose estimation is allowed. For the detected markers, the *cv2.aruco.estimatePoseSingleMarkers* function is used to calculate the pose of the ArUco markers relative to the camera. This function takes as input the corner points of the markers obtained from the previous sections, the dimensions of the markers, the projection matrix of the camera, and the distortion coefficients. The method returns RVEC and TVEC, which are vectors describing the 3D position and orientation of the ArUco marker relative to

the camera. If you need to visualise the pose, you can use the *cv2.aruco.drawAxis* function to draw the marker's axes in the video frame.

Pseudocode for the Estimate the pose of detected markers:

```
Method estimate_pose(marker_corners, marker_ids):
    If marker_corners are not empty:
        For each marker in marker_corners:
            rvec, tvec, _ =
            cv2.aruco.estimatePoseSingleMarkers(marker_corners,
            marker_size, self._camera_projection_matrix,
            self._camera_distortion)
            # Optional: for visualization
            cv2.aruco.drawAxis(frame, self._camera_projection_matrix,
            self._camera_distortion, rvec, tvec, length_of_axis)
        Return rvec, tvec
```

- *Start Tracking and Outputting Marker Pose Data:*
  In the *start_tracking* method, the current state is checked to see if it is "ready", and if it is confirmed to be "ready", the system state is set to "tracking". If the state is "ready", the system state will be set to "tracking", and then it will start to process the consecutive frames from the video source selected by the user. The video frames are read through the *self._capture.read()* method in a loop, and if it succeeds in acquiring the frame, the processing will continue; if it fails, the tracking will be terminated. For each video frame, the system first detects and identifies the ArUco markers in the frame by calling the *detect_markers(frame)* method, which returns the marker's corner point and identifier. Subsequently, the use of the *estimate_pose(marker_corners, marker_ids)* method is used to compute the 3D poses of these markers, obtaining the markers' RVEC and TVEC, and, finally, the system stores these pose results for model tracking.

Pseudocode for the Start Tracking and Outputting Marker Pose Data:

```
MMethod start_tracking():
    If self._state == "ready":
        self._state = "tracking"
        While self._state == "tracking":
            ret, frame = self._capture.read()
            If not ret:
                Break  # End tracking if video ends or error occurs
            marker_corners, marker_ids = detect_markers(frame)
```

```
            rvec, tvec = estimate_pose(marker_corners, marker_ids)
            Process and store marker pose data
```

- *Update the position of the model:*
  In the *update_model_position* method, the system reads a frame from the video source and detects and tracks the ArUco markers in the frame. If the markers are detected, the system updates the position of the model based on the marker's pose. The model's position is updated by applying the RVEC and TVEC obtained from the marker pose estimation to the model's transformation. The model's position is then updated in the VTK rendering window to reflect the new position based on the marker's pose. This process is repeated for each frame to ensure that the model's position is continuously updated based on the marker's pose, allowing the model to follow the movement of the markers in the video stream.

Pseudocode for the Update the position of the model:

```
Method update_model_position:
    Read a frame from video source into 'image'
    If frame read is successful:
        Detect and follow ArUco markers in 'image'
            If markers detected:
                  Update model position based on marker pose
        Render the overlay window with updated model position
```

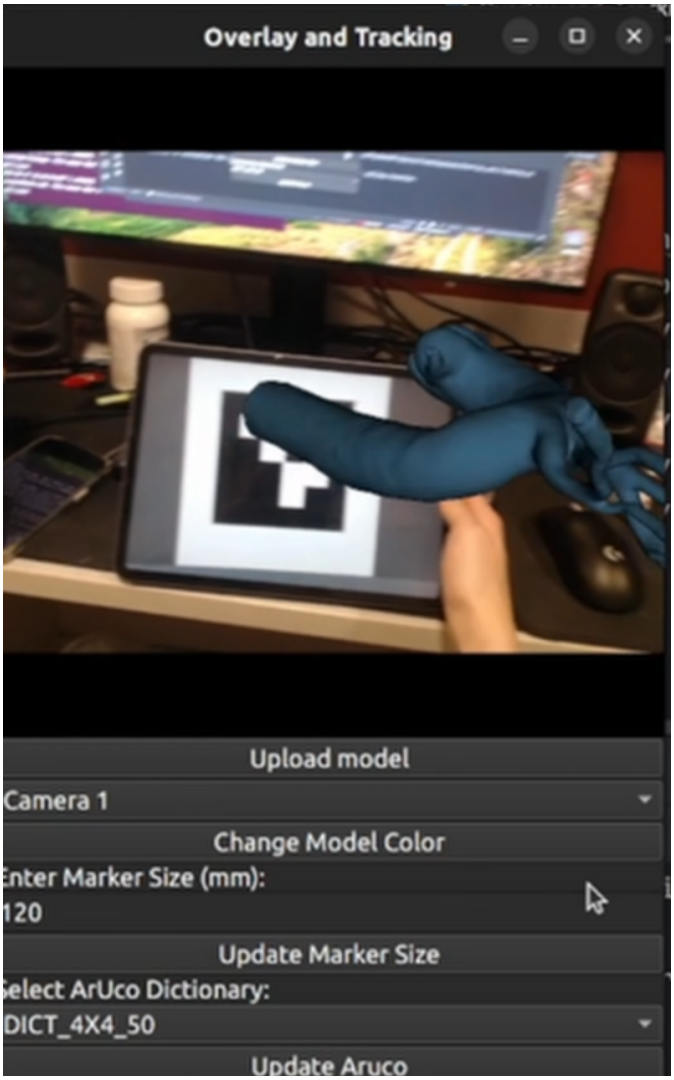

Figure 10: Overlay and Tracking

(g) **Model Color change**
In order to implement the model colour change functionality, it is necessary to reuse the model upload functionality mentioned in the (a) Main Menu (Overlay and Tracking) section. Before performing a model colour modification it will first check if a model exists in the VTK Overlay. If not, a warning box will be pushed to prompt the user to upload the model first. If the model is already loaded in VTK Overlay, when the model colour modification button is clicked in the Qt window, a colour picker dialog box will be opened in the Qt window allowing the user to select a colour. After the user selects a colour, a new model

loader will be created and the model and colour will be reloaded using the model upload function mentioned in the (a) Main Menu (Overlay and Tracking) section. Once the model has been re-uploaded with the newly selected colours, update the VTK Render window to re-render the model Overlay to show the model with the modified colours.

Pseudocode Code for the Model Color change:

```
Method change_model_color:
    Check if the model directory has been loaded
    If no models are loaded:
        Display a warning dialog, "Please upload models first"
        Return

    Open a color picker dialog
    If a valid color is chosen:
        Convert the color to RGB format
        Create a model loader with the model directory and RGB color
        Add the loaded models to the VTK rendering window
        Re-render the window to update the color
```

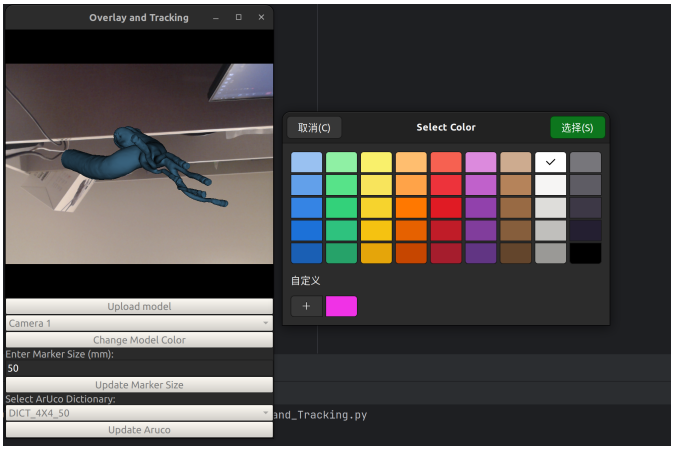

Figure 11: Colour Picker Dialog Box

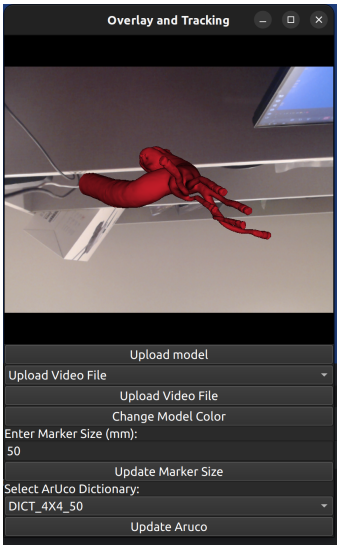

Figure 12: Color Changed Model

(h) **ArUco Marker Settings**

The ArUco parameters are set up by setting up a text box for entering the size of the markers and a drop-down menu for selecting the ArUco lexicon.

Firstly, a label and a text input box are created in the Qt window, allowing the user to enter the size of the ArUco markers (in millimetres) and update the settings with a button. After clicking on the "Update Marker Size" button, the system will read the size value entered by the user, update the settings, and then restart via the Start Tracking and Outputting Marker Pose Data section. restarting the tracking thread to apply the new settings.

Second, a drop-down menu was also created for the user to select the different ArUco dictionaries they use, as well as a button to update the selected dictionary. Once the user has selected a dictionary via the drop-down menu, clicking on the "Update ArUco" button triggers the update of the dictionary, which works in the same way as updating the tag size.

Pseudocode Code for the ArUco Marker Settings:

```
1 Method setup_marker_size_ui:
2     Create label "Enter Marker Size (mm):"
```

```
    Add label to layout

    Create input field for marker size
    Set default text of input field to "50"
    Add input field to layout

    Create button "Update Marker Size"
    Connect button click event to method update_marker_size
    Add button to layout

Method update_marker_size:
    Read value from marker size input field
    Convert input value to float and store in marker size configuration
    Reinitialize ArUco tracker with updated configuration
    Start the ArUco tracker

Method setup_dictionary_ui:
    Create label "Select ArUco Dictionary:"
    Add label to layout

    Create dropdown for dictionary selection
    Populate dropdown with dictionary options:
        'DICT_4X4_50', 'DICT_4X4_100', etc.
    Add dropdown to layout

    Create button "Update Aruco"
    Connect button click event to method update_dictionary
    Add button to layout

Method update_dictionary:
    Get selected dictionary from dropdown
    Update ArUco dictionary in configuration with selected item
    Reinitialize ArUco tracker with updated configuration
    Start the ArUco tracker

```

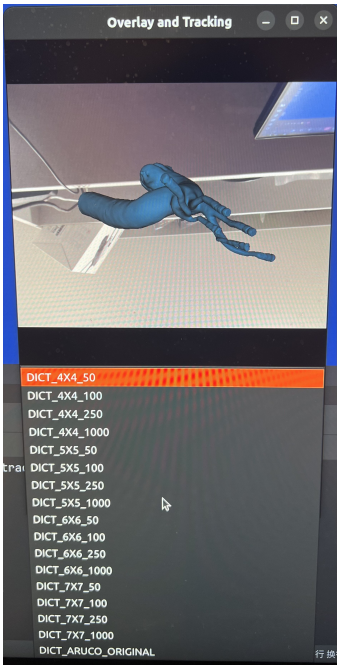

Figure 13: ArUco Dictionary Selection

(i) **ArUco Generator**
The ArUco Marker Generator application is the same as Overlay and Tacking, and is implemented by combining the ArUco module of OpenCV with a Qt graphical interface.

The application first creates a user interface with a drop-down menu that allows the user to select from a variety of predefined ArUco dictionaries. After the user selects a dictionary, OpenCV's *aruco.getPredefinedDictionary* method is called to retrieve the corresponding dictionary, and then the *aruco.drawMarker* method is used to generate markers of the corresponding size and dictionary. After that, the generated marker image is converted to QImage and QPixmap and displayed in the Qt interface.

In addition, the ArUco marker saving function is also implemented, allowing the user to save the generated markers locally as an image file via the standard file dialogue box. When user clicks the "Save Marker" button, it will trigger the *save_marker* method. This method will open a file save dialogue box, allowing the user to select the file storage path and format. Upon user confirmation, the marker will be generated again using the current user-selected dictionary configuration to ensure

that the latest generated image is saved, and the *cv2.imwrite* method will be used to write the image to the specified folder with the default ArUco Dictionary filename.

Pseudocode Code for the ArUco Generator:

```
Class ArucoGenerator:
    Method __init__:
        Initialize GUI components
        Set initial ArUco dictionary to DICT_4X4_50
        Call init_ui to set up the user interface

    Method init_ui:
        Set window title to "ArUco Marker Generator"
        Create vertical layout for widgets

        Create dropdown for ArUco dictionary selection
        Add ArUco dictionary options to dropdown
        Connect dropdown change event to update_dictionary method
        Add dropdown to layout

        Create label to display marker image
        Add label to layout

        Create button "Save Marker"
        Connect button click event to save_marker method
        Add button to layout

        Generate initial marker

    Method update_dictionary:
        Read selected dictionary name from dropdown
        Update ArUco dictionary using selected name
        Generate new marker based on updated dictionary

    Method generate_marker:
        Get predefined dictionary using current ArUco dictionary setting
        Generate marker image for marker ID 0 with size 400x400 pixels
        Call show_marker with generated marker image

    Method show_marker(marker_image):
        Convert marker image to QImage with Grayscale format
        Convert QImage to QPixmap
        Set QPixmap on image label for display

    Method save_marker:
        Get current dictionary name from dropdown
        Set default filename based on selected dictionary and .png extension
```

```
        Open save file dialog with default filename and filter for PNG
        If file path is provided:
            Generate marker image again to ensure current settings are used
            Save marker image to file using OpenCV

Start Point:
    Initialize QApplication
    Create instance of ArucoGenerator
    Show window
    Execute QApplication
```

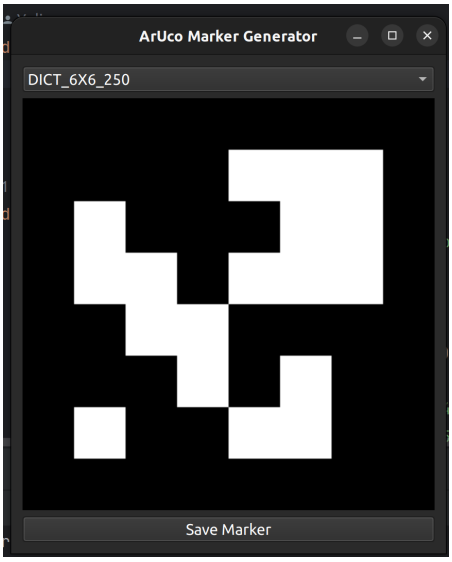

Figure 14: ArUco Generator Dictionary Changed

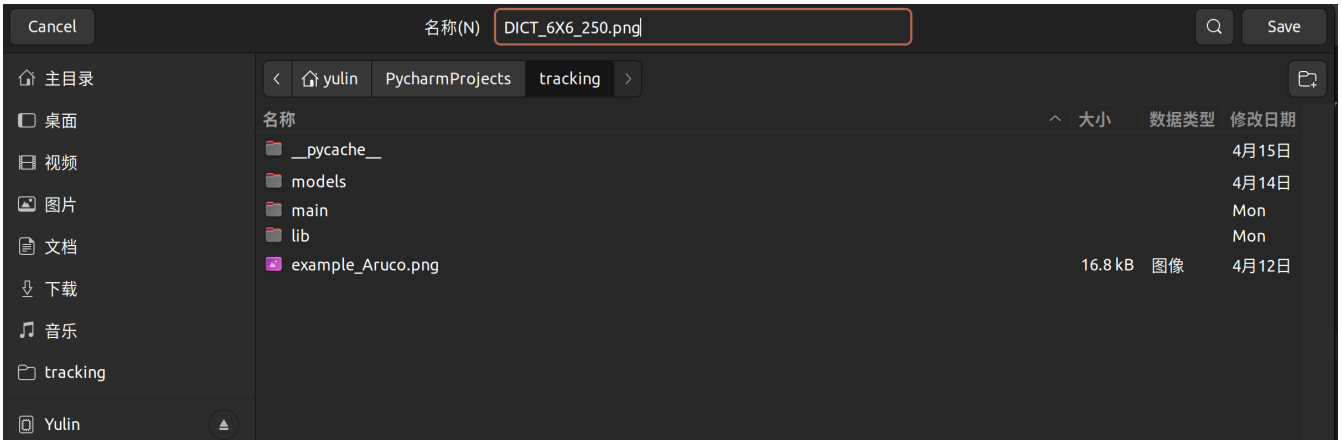

Figure 15: ArUco Marker Saved

2. **Frontend Implementation**
   In this project, the front-end Qt interface is implemented through the PySide6 framework. The various controls on the interface are organised by creating a vertical layout manager, QVBoxLayout. The margins and spacing of the layout were set to zero for the sake of indirection and to maximise space usage.

   The interface includes several features and key components that have been implemented in the previous section Backend Implementation: a drop-down menu that allows the user to select different video sources, a video upload button, an upload button that enables the user to upload a 3D model file, a colour selection button to adjust the display colour of the model, a text box to set the size of the actual ArUco icon and a drop-down menu for selecting the ArUco dictionary.

   These components are connected to the appropriate functional functions through signalling and slotting mechanisms to ensure that user actions trigger the appropriate response[36]. For example, the selection of a video source is done via a drop-down menu, and a function is called to update the video source used by the system when the selection is changed. Similarly, the colour change of a model is done by opening the colour dialog, and when the user selects a new colour, the system applies this colour to the 3D model.

   In addition, a VTK display window is integrated to display the video stream and the 3D model, which is adapted to the user's operating system to en-

sure compatibility. Due to the high extensibility of the Qt interface, new components and features can be easily added to the interface in the future.

Code for the Frontend Implementation:

```
class QtWidget(QWidget):
    def __init__(self, video_source, dims=None):
        super().__init__()
        # Setup the layout for the widget
        self.color_button = None
        self.layout = QVBoxLayout(self)
        self.layout.setContentsMargins(0, 0, 0, 0)
        self.layout.setSpacing(0)

        # Initialize the VTK overlay window, conditionally based on the OS
        init_vtk_widget = platform.system() != 'Linux'
        self.vtk_overlay_window =
        VTKOverlayWindow(offscreen=False, init_widget=init_vtk_widget)
        self.layout.addWidget(self.vtk_overlay_window)

        # Initialize the video source
        self.video_source = TimestampedVideoSource(video_source, dims)

        # Set up a timer to update the view periodically
        self.timer = QTimer()
        self.timer.timeout.connect(self.update_view)
        self.update_rate = 30
        self.model_dir = None
        # Setup additional controls
        self.setup_upload_button()
        self.setup_video_source_controls()
        self.setup_color_change_button()
```

2.2 Part2: Endovascular Intervention Simulation

Our project combines machine learning with AR/VR [5], [37] to create an open-source simulator for autonomous cannulation in endovascular operations. This project try to offer a high-fidelity model of the catheter and aorta, integrating an advanced endovascular robot. It is designed to provide real-time force sensing and has shown, through prelimi- nary tests, to effectively mimic real-world robot behaviours[38]. Furthermore, we aim to deploy CathSim on VR devices like HoloLens or Meta Quest[39], enhancing its accessibility and utility for medical professionals.

2.2.1 Design

1. **Rope Design**
   Early in the design process, I considered two different approaches to Rope design and did an initial practice and comparison of them.

   Firstly, I used a Rope constructed from a slender cylinder cut into multiple sub-cylinders, each of which is equivalent to a node of a rope, and used Unity C# scripts to control each node to enable the bending and elongation of the rope and to try and make this rope give feedback on the forces exerted by the outside world.

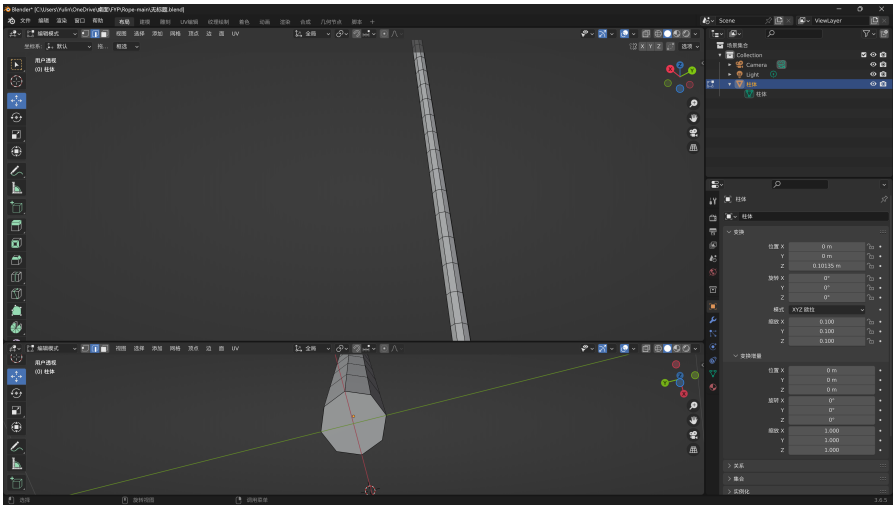

Figure 16: Rope Design in Blender

In the second approach, I attempted to splice multiple identical and separate cubes into a Rope using Unity C# scripting directly. The bending and stretching of the rope was achieved by controlling each cube and their gaps directly through the script.

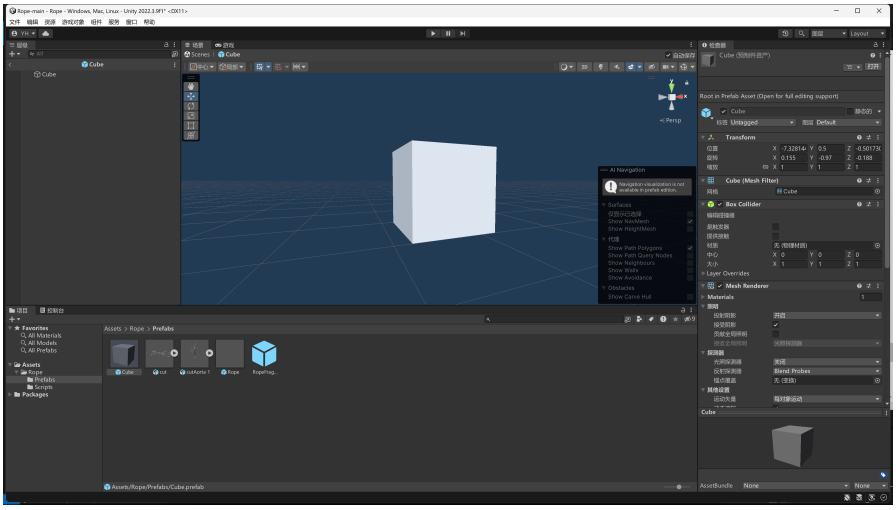

Figure 17: Cube Fragement

Either way, there were a lot of serious and difficult to locate and solve problems in the early stages of the project, and both implementations had major flaws and loopholes.

2. **Model Design**
   The model used for this project was derived from a distribution made by Supervisor at the beginning of the project, with all members using the same Aorta model. In addition to the Aorta model, we have the Catheter model and the Catheter controller model.

   In order for Rope to have access to the Aorta model, the Aorta model needs to be modified using Blender. By optimising the number of triangles that the Aorta has, and by cutting and opening up the ends of the blood vessels in the Aorta model, the Rope can be inserted into the Aorta model without any problems.

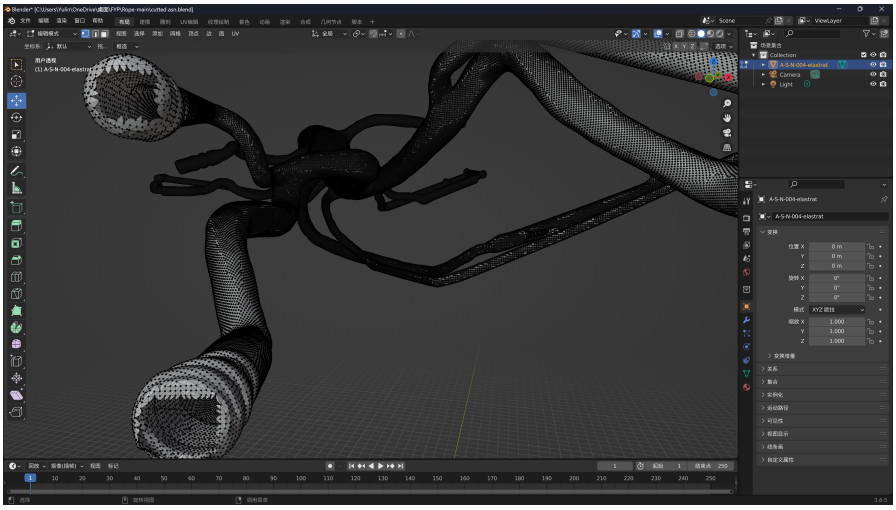

Figure 18: Cut Aorta Model

3. **Endovascular Intervention Simulation Design**
   Endovascular Intervention Simulation is the core part of Part2. It was planned to control the Rope and Catheter in some way, including movements such as rotation, forward, backward and bending. This would allow the Rope to enter the Aorta model and move under the control of the user, with force feedback (e.g. bending, etc.) when it touches the walls of the Aorta. When touching the Aorta wall it should also follow the path of the Aorta deeper under the control of the user, in which case the Rope should bend with the curve of the inner wall of the Aorta and remain inside the Aorta model without escaping.

   In the original design, it was planned to use a slider to control the forward and backward motion movement of Rope and Catheter. However, this control method has serious flaws, Rope can only move back and forth but not bend and rotate under the control of this method. So this control method was abandoned at the early stage of design.

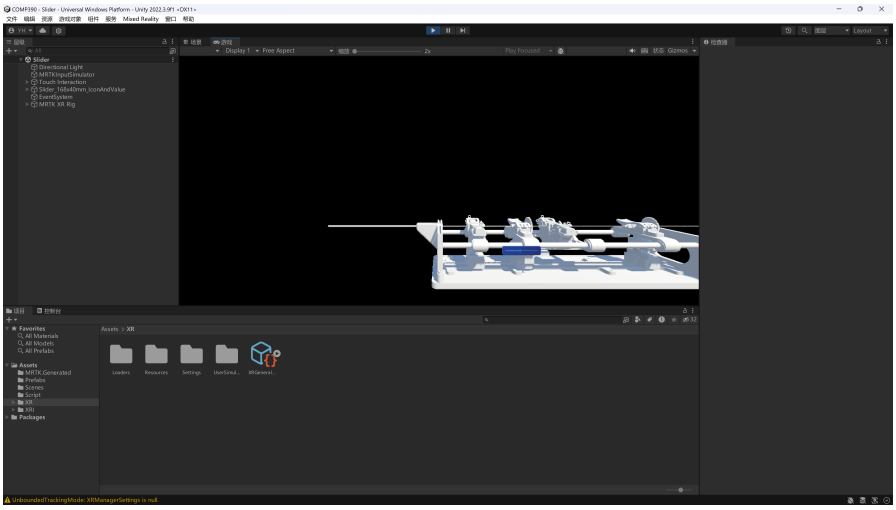

Figure 19: Slider Control Design

After that, we tried to use a cylinder to control Rope under the suggestion of Supervisor. Compared with the slider, the cylinder can control the Rope's forward, backward and rotation at the same time. However, since we are conducting the development of AR/VR project and the ultimate goal is to deploy it on AR/VR related devices, using a cylinder may not be convenient for users to operate it while using virtual reality devices. The design and implementation was previously discarded.

In the end, we chose to use MRTK for development. This development framework can provide a simulated palm for direct control of the Rope. Simulated finger pinching can be performed through the handle of the VR/AR device. Thus, the Rope can be simultaneously controlled with forward and backward motion, rotation and bending.

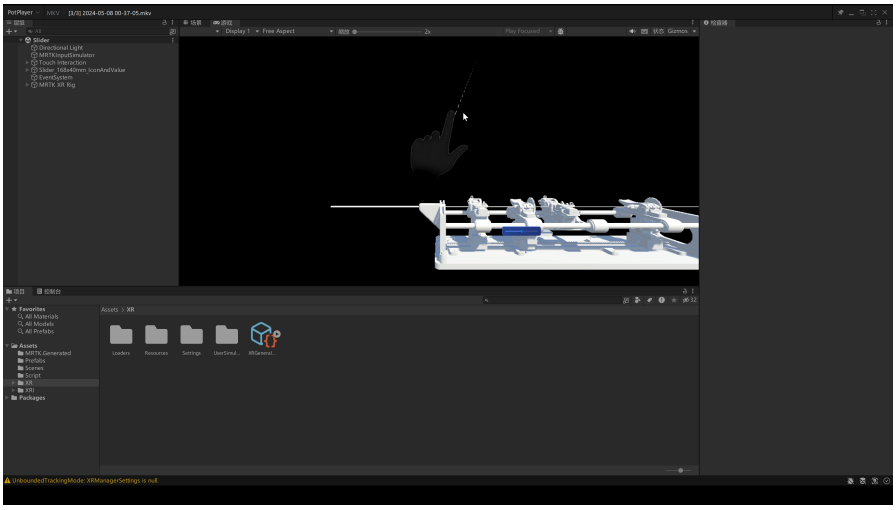

Figure 20: Virtual Hand Control

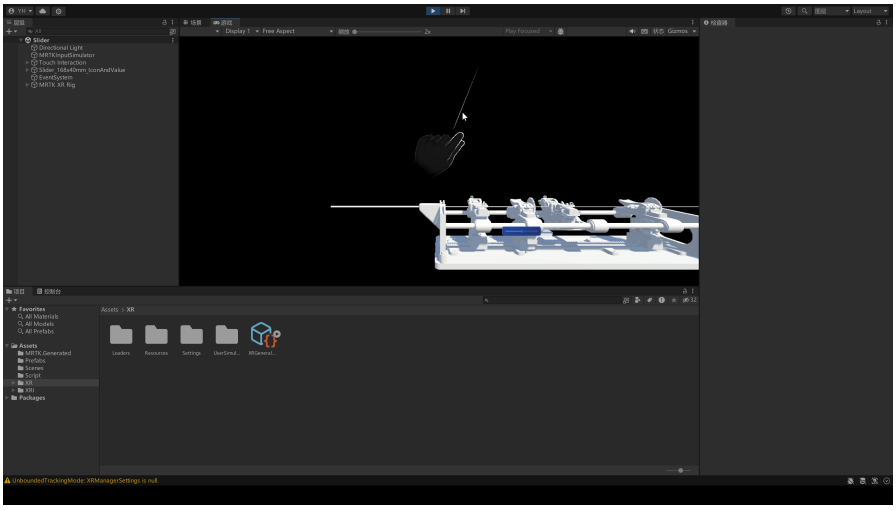

Figure 21: Finger Pinching

4. **User Interface Design**
   In the initial design of the project, I designed two different user interfaces, one for users using a flat display and one for users using VR/AR. In the interface designed for users using flat displays, the interaction is mostly realised using buttons, whereas the interface designed for VR/AR users takes into account factors such as interaction in virtual environments and mostly uses a virtual interface for interaction.

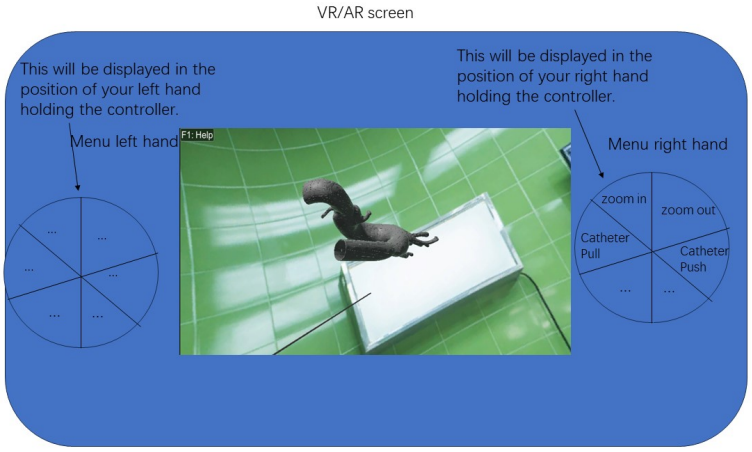

Figure 22: AR/VR User Interface

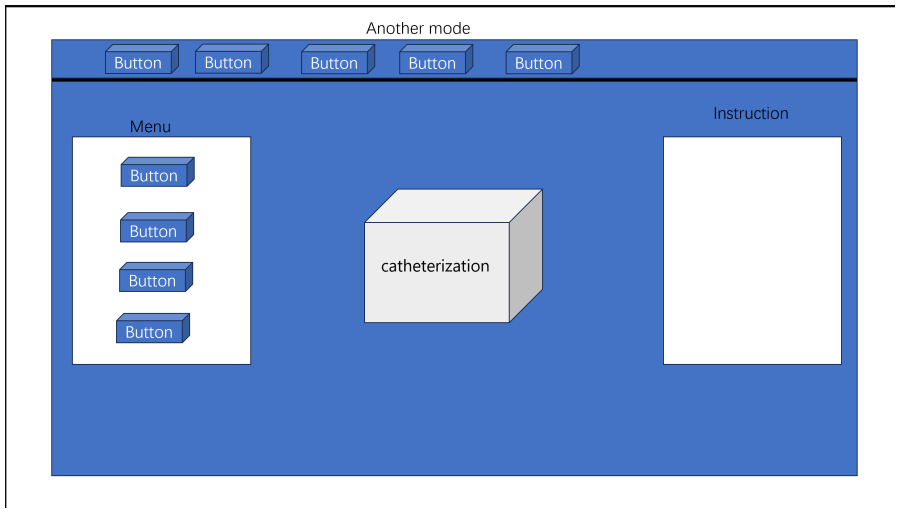

Figure 23: Flat Display User Interface

At this stage, the user interface is still in the design phase, as the core Endovascular Intervention Simulation component is still not up to the level of design that it was at the beginning of the project.

2.2.2 Implementation

1. **Framework Implementation**
   The MRTK development framework needs to be built in Unity first at the

start of the project.

First you need to download the Mixed Reality Feature Tool for Unity and select the corresponding Unity project and the necessary feature packs in the tool. The tool will then automatically add the required MRTK development packages to the project[40].

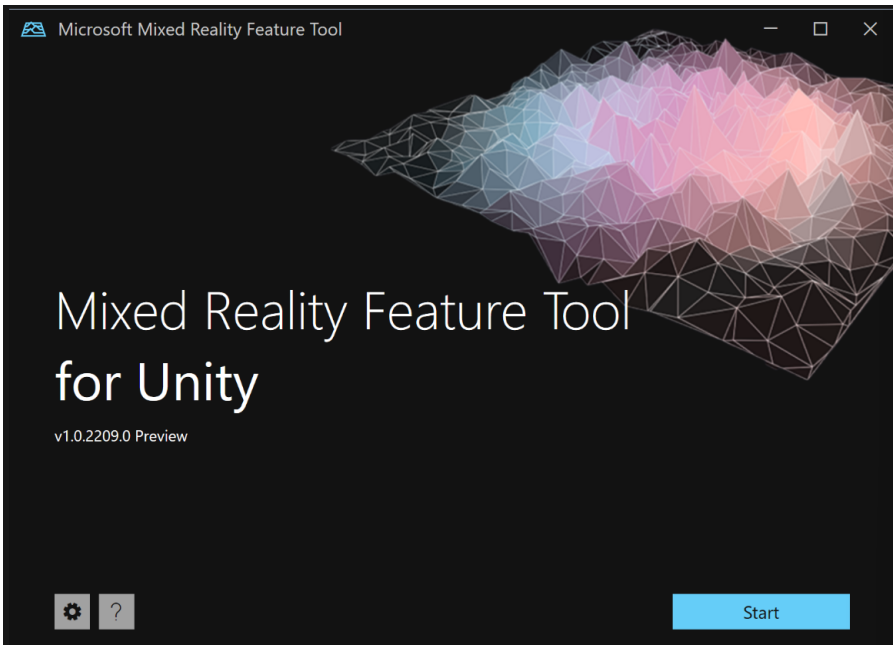

Figure 24: Mixed Reality Feature Tool for Unity[40]

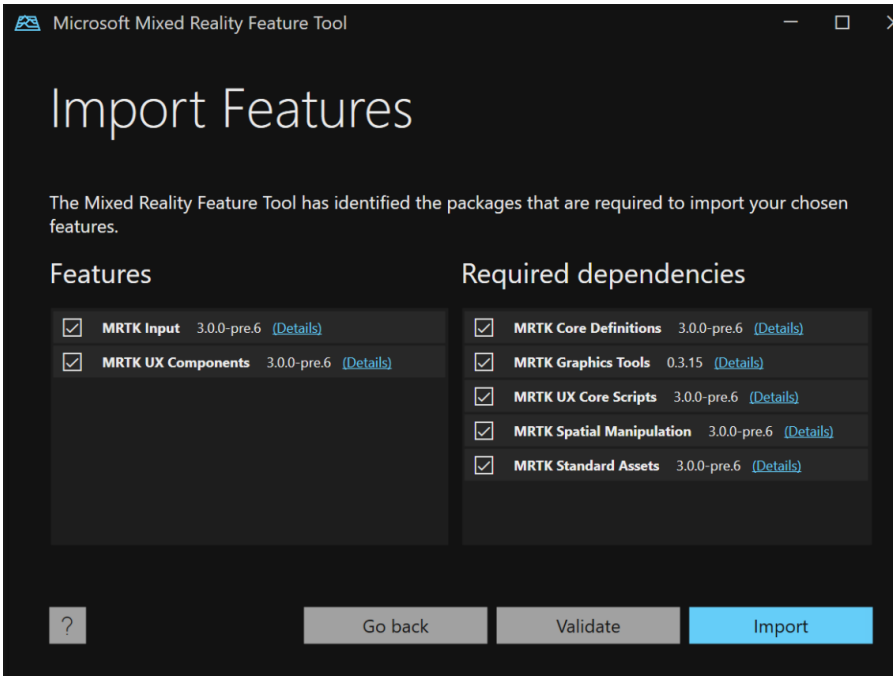

Figure 25: Necessary Feature Packs[40]

In order to use this toolkit in the project, it needs to add the components *MRTK XR Big* and *MRTKInputSimulator* to the Asset for Unity. These components provide the necessary functionality to develop in HoloLens, such as hand simulation.

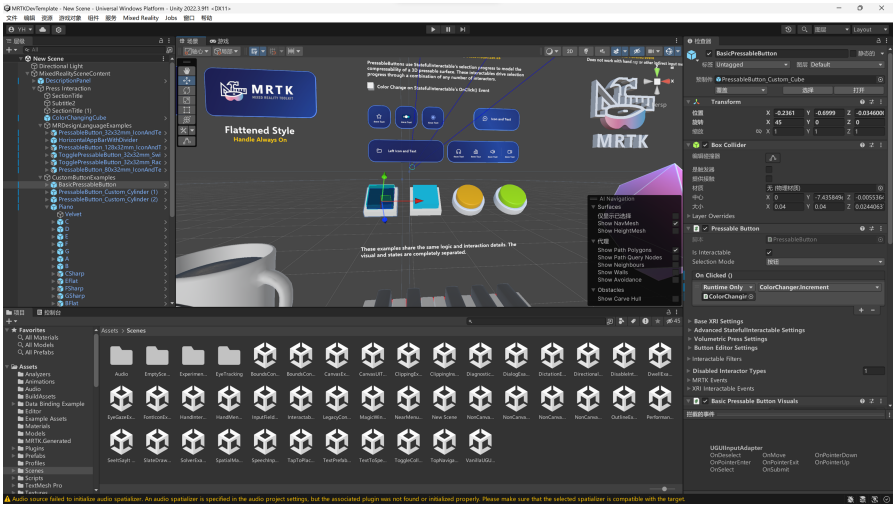

Figure 26: MRTK3 Framework Example[40]

2. **Rope Implementation**
   As described in the Rope construction method described in the Rope Design section, the Rope construction was ultimately implemented using a method of splicing multiple Cubes using Unity C# scripts.

   First create a Cube as a rope *fragment prefab* and add the *BoxCollider* component to give it a collision volume. It also needs to have the *Rigidbody* component added to it to give it properties such as gravity and force impact so that it can react to gravity and external forces. After that, the Rope is generated and controlled directly by the C# script where the *RopeController* class resides. A dynamic rope is modelled by physical and visual components.

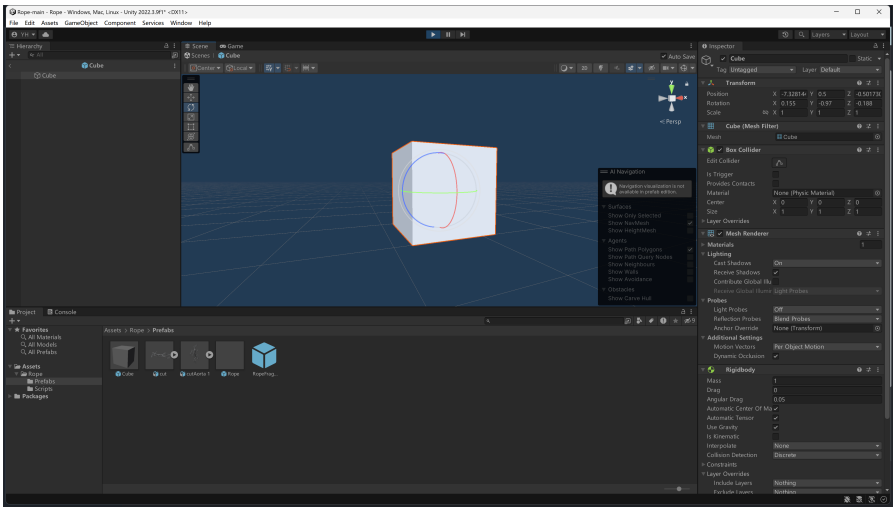

Figure 27: Fragment Prefab

The class uses *fragmentPrefab* to define each segment of the rope (using the previously created Cube), and *fragmentCount* to control the total length of the rope, using the *interval* variable to define the spatial distance between segments. They can all be modified in Unity editor configuration. In the script's Start method, it first initialises an array of *GameObject* to store the rope segments (all cloned from the same Cube). Each segment is then instantiated and placed one by one in a loop to form a linear structure. For each fragment that is not the first, a *FixedJoint* component is added to it. A *FixedJoint* is a special type of physical joint that is used to fix two objects together in terms of their physical behaviour so that they behave as a single unit in the simulation environment[41]. When a FixedJoint is used to connect two objects, the two objects remain relatively unchanged

in position and orientation, simulating an effect of being pinned or welded together. This is later connected to the *Rigidbody* component created by the previous fragment, achieving physical adhesion between the fragments.

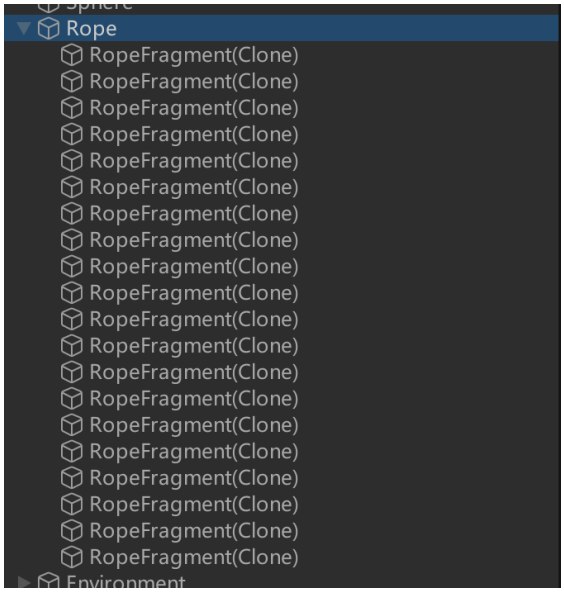

Figure 28: Fragment Prefab Connected

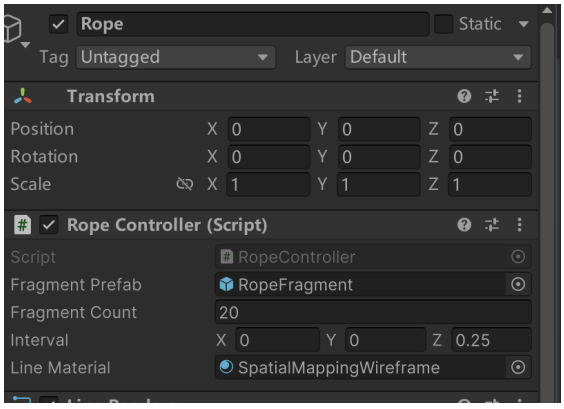

Figure 29: Rope Settings

Additionally, the same configuration that handles the initial bending of the rope is implemented in the script, whereby if a segment needs to have a bend applied, a trigonometric function is used to determine the offset of each segment by calculating the bending angle and the spacing of the segments of

the rope. Specifically, it converts the total bending angle to radians for each segment, and then calculates to obtain offsets in the X and Z directions to visually realise the bending effect of the rope.

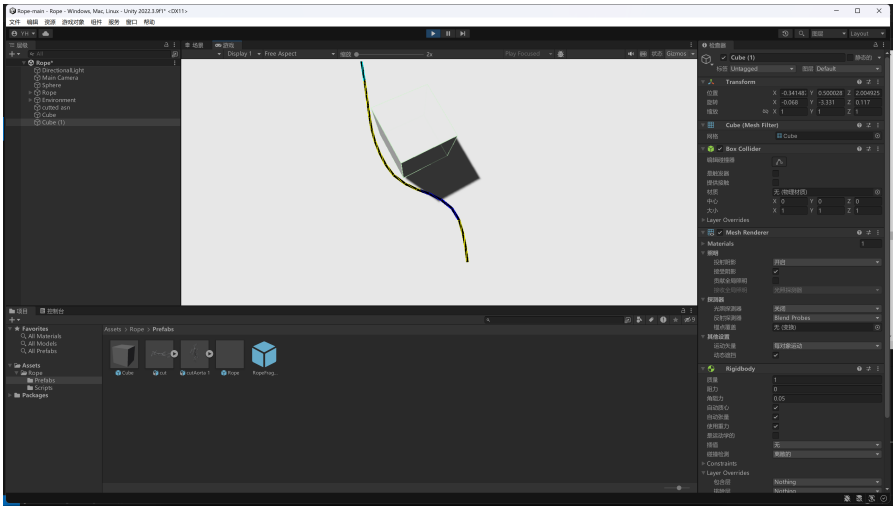

Figure 30: Rope Bending

Also, in order to simulate the real Endovascular Intervention, the anterior end of the Rope is bent to a certain extent. By setting *bendAngle* and *segmentToBend*, some segments of the anterior end of the Rope can be bent to achieve a more realistic simulation.

Finally, to enhance the visual representation of the rope, a *LineRenderer* component is added to the script that draws a continuous line between the rope segments. This can also be adjusted in the Unity editor by configuring properties such as material, line width and colour to ensure that the rope is visually appealing. In the *LateUpdate* method, the script dynamically updates the position of each point of the *LineRenderer* based on the real-time physical position of the segments to ensure that the line is updated as the segments move, ensuring visual continuity and dynamic response.

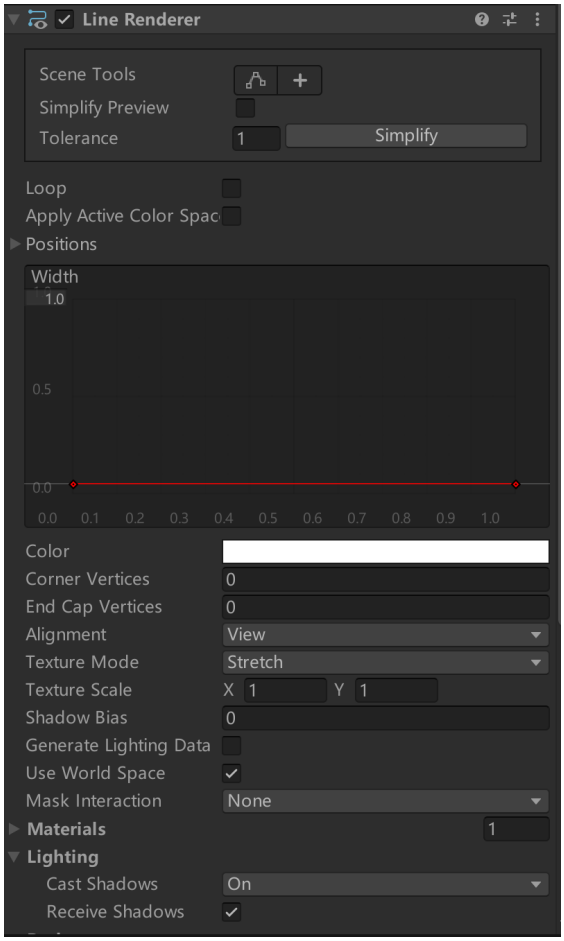

Figure 31: Line Renderer

Pseudocode for the Rope Implementation:

```
using UnityEngine;

namespace Rope
{
    public class RopeController : MonoBehaviour
    {
        [SerializeField] private GameObject fragmentPrefab;
        [SerializeField] private int fragmentCount = 80;
        [SerializeField] private Vector3 interval = new Vector3(0f, 0f, 0.25f);
        [SerializeField] private Material lineMaterial;

        private GameObject[] fragments;
        private float[] xPositions;
        private float[] yPositions;
```

```
15          private float[] zPositions;
16          private CatmullRomSpline splineX;
17          private CatmullRomSpline splineY;
18          private CatmullRomSpline splineZ;
19          private int splineFactor = 4;
20
21          void Start()
22          {
23              fragments = new GameObject[fragmentCount];
24              var position = Vector3.zero;
25              // The desired total bend angle
26              var bendAngle = 30f;
27              // Number of segments over which to apply the bend
28              var segmentToBend = 0;
29
30              for (var i = 0; i < fragmentCount; i++)
31              {
32                  fragments[i] = Instantiate(fragmentPrefab,
33                  position, Quaternion.identity, transform);
34
35                  if (i > 0)
36                  {
37                      var joint =
38                      fragments[i].AddComponent<FixedJoint>();
39                      joint.connectedBody =
40                      fragments[i - 1].GetComponent<Rigidbody>();
41                  }
42
43                  // Calculate the bend for the initial segments
44                  if (i < segmentToBend)
45                  {
46                      // Convert the total angle into radians
47                      // and distribute it across the specified segments
48                      var radiansPerSegment =
49                      Mathf.Deg2Rad * (bendAngle / segmentToBend);
50                      // Use interval.z as the base distance between segments
51                      var offsetX = Mathf.Sin(radiansPerSegment * i) *
52                      interval.z;
53                      var offsetZ =
54                      (1 - Mathf.Cos(radiansPerSegment * i)) * interval.z;
55
56                      // Apply the calculated offset
57                      position.x += offsetX;
58                      position.z += offsetZ;
59                  }
60                  else
61                  {
62                      position += interval;
63                  }
```

```
64              }
65
66              var lineRenderer = GetComponent<LineRenderer>();\
67              // Make sure a material is assigned
68              lineRenderer.material = lineMaterial;
69              lineRenderer.startWidth = 0.05f;
70              lineRenderer.endWidth = 0.05f;
71              lineRenderer.positionCount = fragmentCount;
72              lineRenderer.startColor = Color.white;
73              lineRenderer.endColor = Color.white;
74          }
75
76          void LateUpdate()
77          {
78              var lineRenderer = GetComponent<LineRenderer>();
79
80              for (int i = 0; i < fragmentCount; i++)
81              {
82                  lineRenderer.SetPosition(i, fragments[i].transform.position);
83              }
84          }
85      }
86  }
```

3. **Model Implementation**
   To give the model a collision volume, you need to add *Collider* to it. There are two types of *Collider* in Unity, *Box Collider* and *Mesh Collider*.
   Box Collider is a simple collider component that is usually applied to objects with regular geometry, such as cubes, rectangles, and so on. This collider has a low computational cost due to its simple geometry and is suitable for scenarios that deal with a large number of object collisions. Its main advantages are high computational efficiency and low computational cost. [42]

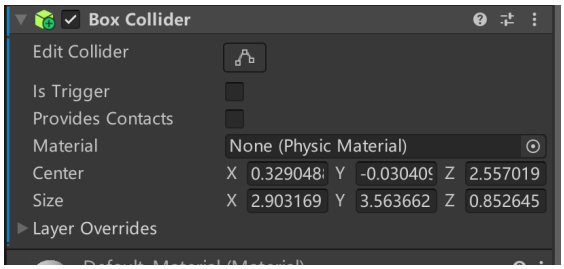

Figure 32: Box Collider

Mesh Collider, on the other hand, is a collider component that can handle

complex model shapes, and is suitable for objects with irregular shapes and a high level of detail, such as Aorta. since it creates collision zones based on the real mesh of the model, it can provide very accurate collision detection, but this also means a higher computational cost. [43]

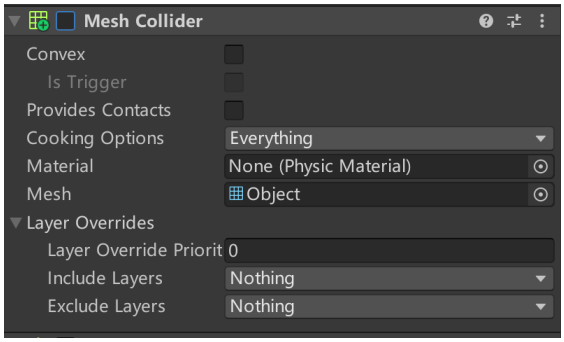

Figure 33: Mesh Collider

In order to achieve more accurate interactions between Rope and Aorta, *Mesh Collider* is used to achieve more detailed interactions, applying *Mesh Collider* to the Aorta model and applying Aorta's mesh can achieve more accurate interactions. The image below shows the difference between using *Box Collider* and *Mesh Collider*, and it is clear that using *Box Collider* does not allow Rope to enter the Aorta model.

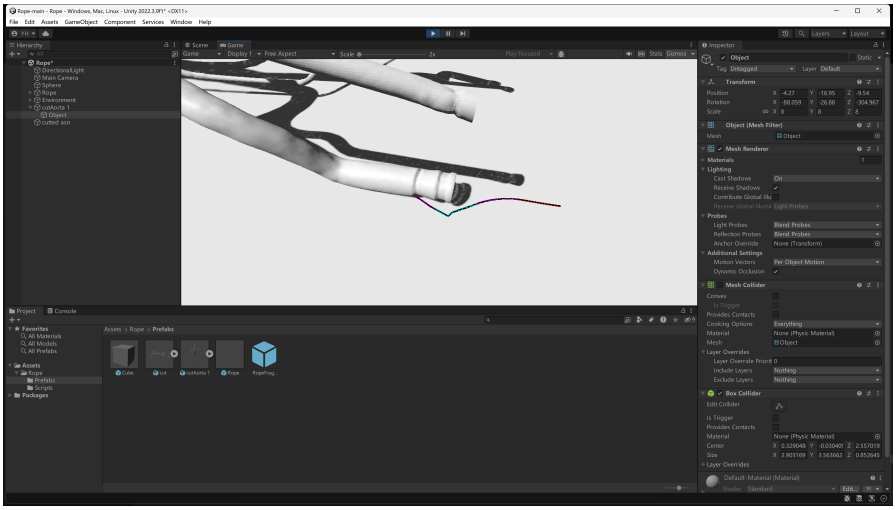

Figure 34: Using Box Collider

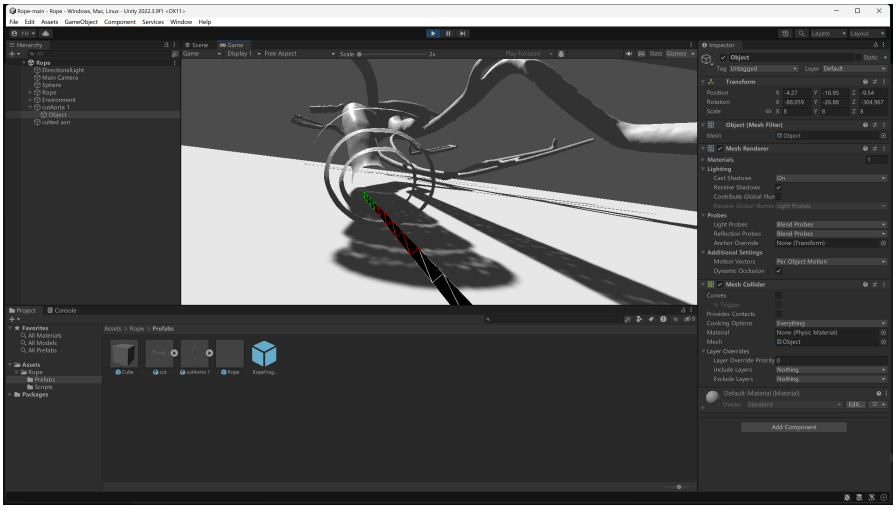

Figure 35: Using Mesh Collider

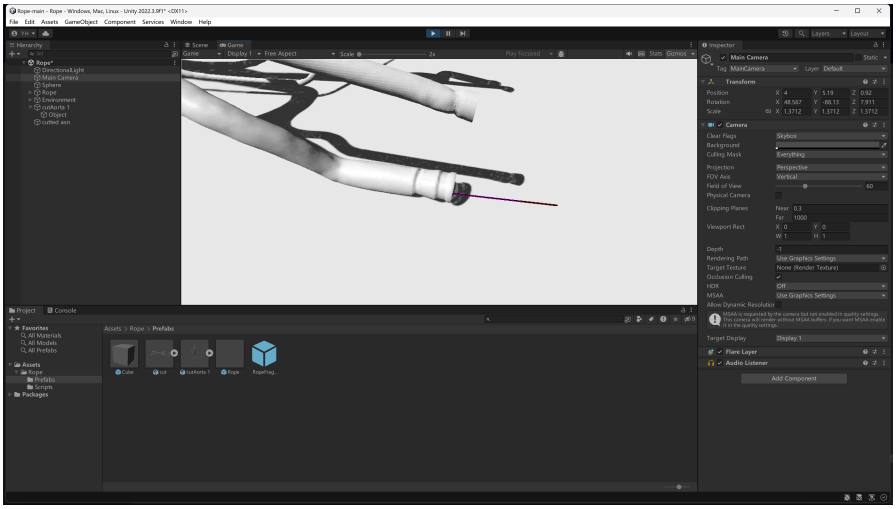

Figure 36: Using Mesh Collider

Although the basic interaction between Rope and Aorta model is achieved, there are still many problems at this stage. For example, overstretching the Rope and exceeding its elastic limit will cause the Rope to become confused. Moreover, it is not possible to make the Rope deeper into the Aorta at this stage, and the problem of the Rope escaping from the Aorta during the insertion process may also occur.

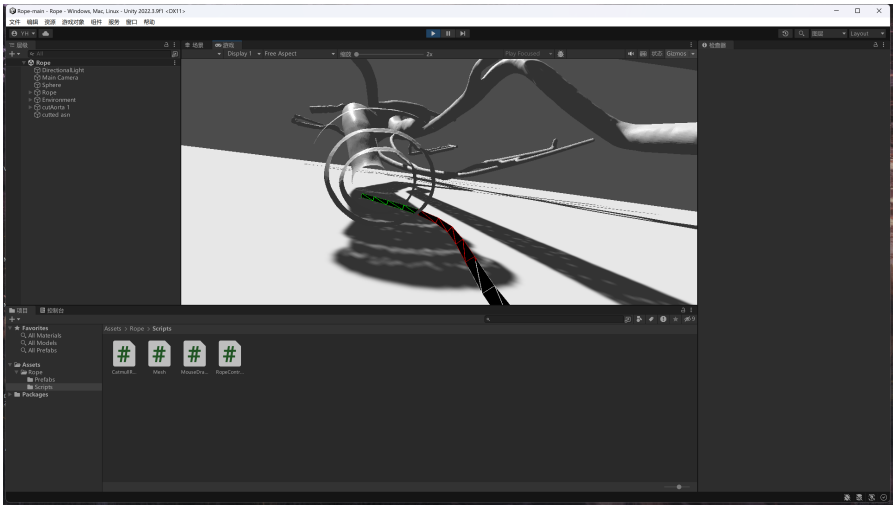

Figure 37: Rope Insertion Out

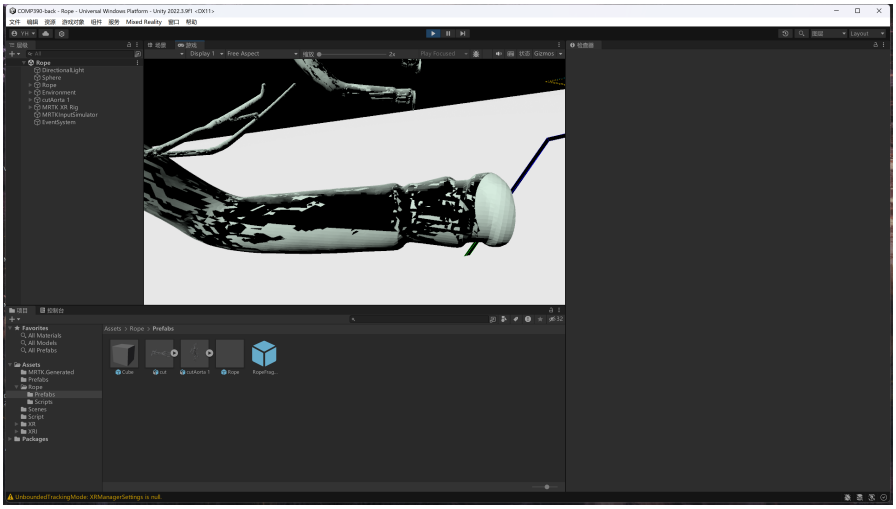

Figure 38: Rope Jittering

4. **Current Stage**
   After completing all the parts, using the features of the MRTK, it is then possible to simulate with the virtual hand, which can be used to directly control the various postures of the Rope. Similarly, the same effect (except for rotation) can be achieved by using the mouse to click and drag the Rope directly.

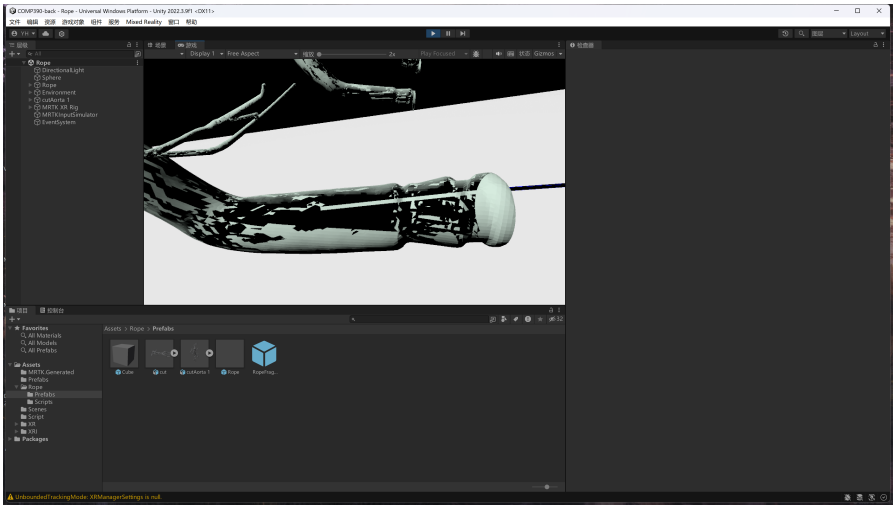

Figure 39: Initialize

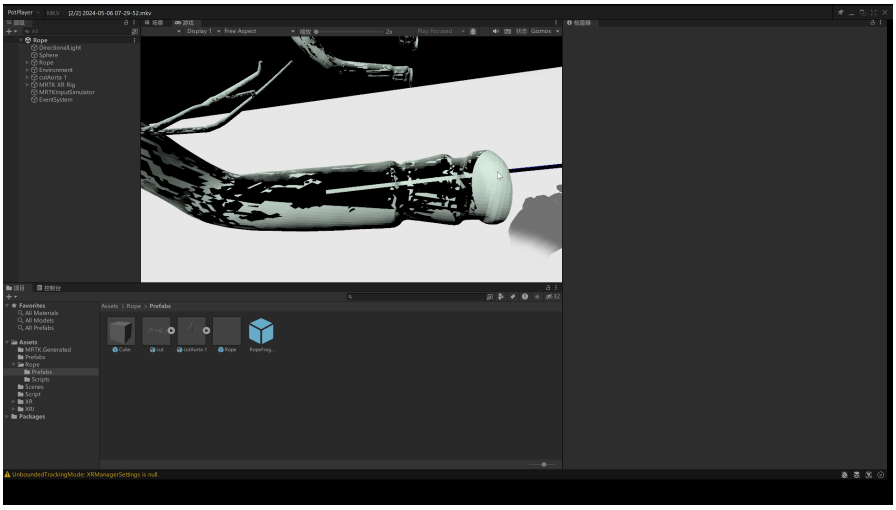

Figure 40: Insertion

3 Testing & Evaluation

3.1 Part1: Real-world Model Interaction and Tracking

3.1.1 Testing

To test the functionality of Part1, a unit test was created using the *unittest* library[44].

- **Initialization Test:**
  The test ensures that the video source and video frame parameters are set correctly when the class is instantiated. This test verifies that the constructor correctly receives and applies the incoming video source and corresponding parameters.

- **Start Test:**
  Tests if the *start* method starts the timer correctly. By simulating the timer's *isActive* method to return False and re-calling the *start* method, you can check if the timer is set to the normal start frequency.

- **Stop Test:**
  Tests that the *stop* method effectively stops the timer, and stops the video capture.

- **Update Video Source Controls Test:**
  Tests that the video source selector and video upload button are set correctly. This includes checking that the video source selector is of type *QComboBox* and that the upload video button is visible in its initial state.

- **Update View Test:**
  Detecting whether the screen can be refreshed properly ensures that the model position is instantly refreshed when ArUco marker tracking is performed.

- **Teardown Test:**
  At the end of the test, ensure that you exit the *QApplication* instance appropriately to avoid memory leaks or other resource hogging issues.

Code for the Testing:

```
import unittest
from PySide6.QtWidgets import QApplication
from unittest.mock import MagicMock
import sys

# Import your BaseWidget class and other necessary modules
from your_module import BaseWidget, TimestampedVideoSource, VTKOverlayWindow

class TestBaseWidget(unittest.TestCase):
    def setUp(self):
        # Create an QApplication instance because QWidget requires it for initialization.
        self.app = QApplication(sys.argv)
        # Mock the real video source and dimensions
        self.video_source = MagicMock()
```

```
        self.dims = (640, 480)
        # Instantiate BaseWidget
        self.widget = BaseWidget(video_source=self.video_source, dims=self.dims)

    def test_initialization(self):
        # Test if initialization correctly sets up the video source and dimensions
        self.assertIsInstance(self.widget.video_source, TimestampedVideoSource)
        self.assertEqual(self.widget.video_source.source, self.video_source)
        self.assertEqual(self.widget.video_source.dims, self.dims)

    def test_start(self):
        # Test if the timer starts running after calling the start method
        self.widget.timer.isActive = MagicMock(return_value=False)
        self.widget.start()
        self.widget.timer.start.assert_called_once_with(1000.0 / self.widget.update_rate)

    def test_stop(self):
        # Test if the timer stops after calling the stop method
        self.widget.stop()
        self.assertTrue(self.widget.timer.stop.called)

    def test_update_video_source_controls(self):
        # Test if video source controls are set up correctly
        self.assertIsInstance(self.widget.video_source_selector, QComboBox)
        self.assertTrue(self.widget.upload_video_button.isVisible())

    def test_update_view(self):
        # Ensure that update_view throws NotImplementedError
        with self.assertRaises(NotImplementedError):
            self.widget.update_view()

    def tearDown(self):
        # Properly quit the QApplication instance
        self.app.quit()

if __name__ == '__main__':
    unittest.main()
```

3.1.2 Evaluation

- **Functional Evaluation:**
  The part achieves features such as real-time video capture, model overlays and ArUco marker tracking that were requested during the design phase. Additional features such as model colour modification, video source switching, ArUco parameter setting and ArUco generator have been added. The part has been successfully implemented and tested, and the functionality is stable and reliable.

- **Performance Evaluation:**
  After performance evaluation, this section can be used normally in most cases, but if the uploaded video frame rate is too high, the resolution is too high or the video is too large, it may cause the uploaded video interface to lag. In this case it may also cause video stuttering and poor tracking within the application interface.

- **User Experience Evaluation:**
  The app's application interface is simple and logical, with relatively easy instructions and operations for various functions. However, when deploying the application, you may need to install the required open source libraries and configure the environment.

- **Maintainability and scalability evaluation:**
  The application was developed using Pyside 6 for the front-end, and individual features were added as widgets with a high degree of maintainability and extensibility. The core functions of video capture, model tracking, and video source switching were split and organised into multiple classes for easy maintenance and upgrading afterwards.

3.2 Part2: Endovascular Intervention Simulation

3.2.1 Testing

To test the functionality of Part2, a unit test was created using the *Nunit* library[45].

1. **Test environment setup and cleanup:**

   - **SetUp method:**
     Before starting a new test using *Nunit*, first build the test environment using the *SetUp* method first. This creates a new GameObject object and adds the RopeController component to it for use in subsequent tests.
   - **TearDown method:**
     Clean up the test environment by executing the *TearDown* method after each test is completed. This destroys the GameObject object created in the *SetUp* method and ensures that the environment is reset after each test is run, avoiding the need for different tests to interact with each other.

2. **Test Case Description:**

- **Testing the correct number of segments:**
  This verifies that the specified number of fragments are created correctly when the *Start* method in *RopeController* is called.This test checks that the fragment array is the preset length of 80 to ensure that the fragments are initialised correctly.
- **Test the correctness of the segment position:**
  This test further checks that the position of each fragment is set correctly after the *Start* method is called. This test ensures that each fragment is placed in the correct position by verifying the incremental position of each fragment on the z-axis (the default is 0.25 units of length per interval).
- **Testing LineRenderer position updates:**
  After executing the *LateUpdate* method, it will be verified that the position array of the *LineRenderer* component has been updated and matches the current position of the fragment. This test checks that visual elements, such as line rendering, respond correctly to background data changes.

```
using NUnit.Framework;
using UnityEngine;
using UnityEngine.TestTools;

namespace Rope.Tests
{
    public class RopeControllerTests
    {
        private RopeController _ropeController;
        private GameObject _ropeControllerGameObject;

        [SetUp]
        public void SetUp()
        {
            _ropeControllerGameObject = new GameObject();
            _ropeController = _ropeControllerGameObject.AddComponent<RopeController>();
        }

        [TearDown]
        public void TearDown()
        {
            Object.Destroy(_ropeControllerGameObject);
        }

        [Test]
        public void Start_WithValidParameters_CreatesCorrectNumberOfFragments()
        {
```

```
28              _ropeController.Start();
29
30              Assert.AreEqual(80, _ropeController.fragments.Length);
31          }
32
33          [Test]
34          public void Start_WithValidParameters_CreatesFragmentsWithCorrectPositions()
35          {
36              _ropeController.Start();
37
38              for (int i = 0; i < _ropeController.fragments.Length; i++)
39              {
40                  Assert.AreEqual(i * 0.25f,
41                  _ropeController.fragments[i].transform.position.z);
42              }
43          }
44
45          [Test]
46          public void LateUpdate_WithValidParameters_UpdatesLineRendererPositions()
47          {
48              _ropeController.Start();
49              _ropeController.LateUpdate();
50
51              var lineRenderer = _ropeController.GetComponent<LineRenderer>();
52
53              for (int i = 0; i < _ropeController.fragments.Length; i++)
54              {
55                  Assert.AreEqual(_ropeController.fragments[i].transform.position,
56                  lineRenderer.GetPosition(i));
57              }
58          }
59      }
60  }
```

3.2.2 Evaluation

4 Project Ethics

I have read and abide by the University's ethical guidelines[46]. The project did not involve direct interaction with human participants during the design, implementation or evaluation phases. An extensive review of the project scope and methodology confirmed that no personal data was collected, analyzed or used. In addition, all activities were within the scope of activities permitted by our ethical guidelines. It was verified with the project supervisor that no customized activities required separate ethical approval. Therefore, there are no other ethical issues involved in this project.

5 Conclusion & Future Work

In this project Part1: Real-world Model Interaction and Tracking, real-time video capture and model overlay, as well as model detection and tracking of ArUco markers, were achieved by combining the OpenCV library with the SciKit-Surgery augmented reality library and Pyside6. In the future, it is planned to add various features to facilitate operation in VR/AR environments, and to further explore the deep integration of Part1 and Part2 and deployment in various AR/VR devices. The project will also be explored for remote deployment for tele-surgery or remote training.

However, in Part2: Endovascular Intervention Simulation, the project faced significant challenges, which led to delays in the project schedule and incomplete final results. Despite these challenges, the basic requirements of the project were eventually completed through multiple designs and evaluations during the course of the project. The Endovascular Intervention Simulation for HoloLens was developed using the Unity engine in conjunction with the Microsoft Mixed Reality Toolkit-Unity (MRTK) development framework, and although it did not achieve all of the functionality that was designed in the early stages of the project, it was still a valuable development experience. Although not all of the features designed at the beginning of the project were implemented, this project was still a valuable development experience. In the future, we plan to continue to improve the unfinished features, further optimise the interaction between the core Aorta and Rope, and try more physical simulation methods to achieve better simulation effects. At the same time, it is hoped to add Rope's force feedback function and more suggestive and helpful functions. There are also plans to expand this project to all kinds of VR/VR devices, not limited to Microsoft Hololens, and to make it more accessible to the public.

In the future, it is expected that this project will be further improved to realise a complete open source endovascular intervention simulation tool, which will not only become a powerful teaching and training tool, but also be able to play a greater role in medical practice.

6 BCS Criteria & Self-Reflection

This section will be used to state that my project met the six outcomes expected by the Chartered Institute of Information Technology[47]. I will focus on illustrating an ability to self-manage a significant piece of work and the critical self-evaluation component.

6.1 An Ability to Apply Practical and Analytical Skills during the degree programme.

The project has demonstrated the practical and analytical skills I have learnt during my time at university. Throughout the degree I have gained a deeper understanding of programming languages such as Python, C# , Java and C and have gradually begun to experiment with them. The theoretical and practical foundations of these languages have been key in enabling me to achieve the complex functionality required for development and realisation projects. For example, in the Part1: Real-world Model Interaction and Tracking section of my project, which was written entirely in Python, there was a high level of theoretical and practical demand for the Python language. In my Part2: Endovascular Intervention Simulation, I needed to acquire and apply knowledge such as the application of Unity and the development and application of the C# language that I had learnt in my degree programme. These technical skills were acquired and refined through a careful learning process and were directly applied to the project, which dealt with the development of a real model interaction and tracking system and the development of a Unity-based Endovascular Intervention Simulation.

In developing Part1: Real-world Model Interaction and Tracking and Part2: Endovascular Intervention Simulation, I have also made extensive use of the Artificial Intelligence, Game Development and Computer Vision knowledge that I have learnt on my degree course.

For Part1: Real-world Model Interaction and Tracking, I utilised the techniques learnt in the Computer Vision course to process the images and used the OpenCV package usage learnt in the course to implement the tracking of the ArUco markers. By utilising the image processing and tracking capabilities of OpenCV, accurate model interaction in complex environments is carried out in practice.

In Part2: Endovascular Intervention Simulation, I used my knowledge of game development to develop a Unity project, using Blender to modify and optimise the model, and applying Unity techniques to ensure that Rope interacts with the blood vessels.

This project, dedicated to the development of a realistic simulation used for endovascular interventions in a virtual reality environment, emphasised my ability to integrate practical skills and theoretical insights, demonstrating a deep understanding of the technical and theoretical aspects I have learnt during the course.

Overall, this project clearly demonstrated my ability to apply the analytical and

practical skills acquired during my degree programme. It also demonstrated my understanding and use of complex programming techniques and frameworks, as well as my ability to use multidisciplinary knowledge to cross-cut problem solving. Through this project, I have accomplished the ability to translate my learning into practical applications in the real world.

6.2 Innovation and/or Creativity

There are some innovations in the field of medical simulation technology in my project, especially Part2: Endovascular Intervention Simulation, The aim of Part2 is to create one of the few open source endovascular intervention simulation projects in the field to make training tools more accessible to the medical community. The project combines traditional surgical simulation with augmented reality/virtual reality technology to make surgical simulation procedures visual and easy to practice. Compared to traditional simulations that are limited to 2D screen displays, this approach uses virtual reality to present surgical simulations in 3D space, which not only enhances the realism of the simulation and interactions, but also allows users to experience and understand the steps involved in the surgery more clearly by allowing them to interact with the simulated environment in a more intuitive and natural way.

The project uses an open source framework (MRTK) and the integration of Augmented Reality/Virtual Reality (AR/VR) technology to provide a practical and innovative application for educational tools in the medical field.

6.3 Synthesis of Information, Ideas, and Practices

This project integrates development tools and theoretical principles from different fields to design the open source Endovascular Intervention Simulation to provide a convenient tool for medical surgery simulation or surgical training.

In the first part of the project, Real-world Model Interaction and Tracking, open source development tools such as OpenCV, VTK and SciKit-Surgery were used. The use of these powerful tools allows me to design user-friendly graphical interfaces or to enhance the model rendering capabilities and real-world model tracking capabilities of my application. OpenCV provides a rich set of image processing tools to help me perform complex image processing and tracking, while VTK and SciKit-Surgery provide powerful tools for medical impacts, such as multilayer image rendering and overlays. This can help me combine tracking capabilities with augmented reality to create a more interactive virtual reality system for users. This section combines knowledge, tools and ideas from various fields to create a

high quality solution.

Part 2: Endovascular Intervention Simulation Developing an application on Microsoft Hololens using MRTK translates the theoretical knowledge I have learnt in my school course such as C# and Unity development into a practical solution. The development utilised model modification and knowledge related to Unity development, C# development, etc. to transform Endovascular Intervention Simulation, which is traditionally limited to a flat display, into a virtual reality simulation with immersive, interactive features. This part of the development demonstrates how AR application development techniques and Unity development can be combined to provide a virtual reality surgical simulation with multiple functionalities.

Both parts of the project exemplify how technical and theoretical knowledge from the fields of computer vision, artificial intelligence and software development can be applied to create effective and innovative medical training tools.

6.4 Meeting a Real Need in a Wider Context

Both parts of the project, Part1: Real-world Model Interaction and Tracking and Part2: Endovascular Intervention Simulation, address some of the broader needs in the medical field.

For Part1: Real-world Model Interaction and Trac, current market systems usually lack user-friendly graphical interfaces, and features such as model colour, selection of different ArUco markers, and resizing are lacking or incomplete, which can cause some degree of difficulty for users. For Part1 the project adds a graphical interface and provides a variety of modifiable parameters to optimise these shortcomings, making the software less difficult to use and better adapted to the needs of a wide range of scenarios.

Part2 considers the lack of open source endovascular intervention simulation in the market and the fact that most existing simulation tools are limited to 2D planar presentation and cannot meet the complex 3D visual and operational needs. The aim is to develop an open source platform that supports immersive 3D simulation, AR/VR and other functions. The system can support 3D simulation in AR/VR (deployed in Microsoft Microsoft Hololens), and by lowering the barrier to use through more intuitive and simple controls, it can be used in the future to allow healthcare professionals or non-professionals alike to experience or learn surgical skills. This simulation tool can not only be used for professional training, but also meets the need for telemedicine services that can provide remote diagnosis and treatment in the future.

Overall, it is planned that these two components will be combined in future work, which can meet the needs for simulation of surgical training simulation for simplicity, remote operation, and 3D highly experiential simulation. In the future it may be possible to expand into more areas to meet a wider range of needs, such as providing an immersive experience of Endovascular Intervention Simulation for lay people.

6.5 An Ability to Self-Manage a Significant Piece of Work

In my project I demonstrated the ability to self-manage a significant piece of work, but it was partially flawed. The project consisted of two widely differing parts, which added to the difficulty of managing the project as a whole, and in order to keep both of the major parts of the project accurately planned and executed, I used a variety of tools and methods to ensure that the project ran smoothly.

Firstly, I used a number of time management tools to map out the timeline of the project, including key milestones, time required, and deadlines for each progress module. I created some Gantt charts and schedules to help me monitor the progress of the project and try my best to make sure that the tasks in each phase are completed on time. In this way I could get a clear picture of the overall progress of the project and adjust the plan as much as possible in time to cope with possible delays. However, as this was my second time working on a larger volume project (the last time was COMP208 Group Project), I was not able to be very perfect in creating the schedule and Gantt chart, and some mistakes were made.

Secondly, in terms of project management, I used GitHub to maintain version control of the project and writing between team members. I used GitHub to effectively track code updates and backups, as well as to enable team members to view the latest progress of the project in real time and provide feedback. GitHub's version control and backup features have many times saved errors caused by mistakes, effectively avoiding many accidents. Using this open source platform has helped me to manage a major task and increase the efficiency of multi-person collaboration.

In addition, in order to control the development progress and quality, I also hold weekly progress meetings with my team members and supervisor to report the progress of this week's work and discuss and plan the next work. In these meetings, I can get sufficient feedback to help me modify and optimise my previous work, and make reasonable planning and arrangement for the next work. This regular reporting and discussion has ensured that the project has developed ac-

cording to the set objectives.

I also focus on stage-by-stage problem analysis and risk management during project implementation. Whenever the project progresses to a certain stage, I will review the previous work, check and improve any possible problems in the completed work, and make sure that the previous work is accurate before proceeding to the next stage1. This is a good way to ensure that I make fewer mistakes when managing a large volume of work.

Whilst I have adopted a variety of methods during the project management process to ensure that the project is executed efficiently and to a high quality as planned, I still have some shortcomings in my ability to self-manage a significant piece of work. For example, although I produced a Gantt chart and schedule to monitor the progress of the project, at the beginning of the project I did not properly consider the time required for some parts and the difficulties I may have encountered, for example, I encountered great difficulties in carrying out the initial design and import of the Rope in Part2: Endovascular Intervention Simulation, etc., and the rate of progress was not as fast as expected. This resulted in the project progressing at a much slower pace than expected and led to the project being put on hold for some time. In addition, although we had weekly meetings, some of them were of minimal effect. These meetings usually took place when the project was experiencing some major difficulties, and in these meetings the solutions to the problems and the planning for the next phase of the project progress were not discussed very effectively.

In addition, I had problems with teamwork when using GitHub for project management. Poor documentation, different operating systems used by each member, conflicting versions of various software packages, and GitHub's file size limitations for uploading files caused many difficulties for team members when sharing through GitHub.

Overall, this project demonstrated my ability to self-manage a significant piece of work, but it also demonstrated my shortcomings in some of these areas. This project gave me a great opportunity to optimise my ability to manage projects, such as time planning skills and communication with team members, as well as making me realise what I need to learn and improve in project management.

6.6 Critical Self-Evaluation of the Process

In my projects, Part1: Real-world Model Interaction and Tracking and Part2: Endovascular Intervention Simulation, although both have been accomplished, there

have been many challenges and difficulties in the development process, and so far there have been some shortcomings. Through in-depth critical self-evaluation, I was able to comprehensively analyse the successes and shortcomings of the project, as well as gain experience and lessons learned.

First of all, the functionality of my project Part1 is relatively complete, which can effectively implement real-time model overlay and ArUco marker tracking, and complete the basic function of my plan. At the same time, I also added extra features such as ArUco icon parameter tuning and ArUco icon generator. However, there are some technical limitations in this part, mainly in platform compatibility. Currently, the system only runs on the Linux platform and has not yet implemented support for other operating systems such as Windows or macOS, nor has it been able to complete deployment on VR/AR devices such as HoloLens. This limitation may have impacted the project's widespread adoption. In this regard, I believe that my Part1 met the requirements of my plan, but still needs to be improved and extended in terms of compatibility.

For Part2, the development process encountered significant technical challenges, especially during the stages of designing the Rope and developing the method of interaction between the Rope and the vessel wall. These technical issues led to delays in the development progress and the final product implemented only the most basic functionality and did not achieve the level of Rope-vessel wall interaction required at the beginning of the design. This difficulty stemmed from my lack of skill in using development tools such as Unity and Blender, and my underestimation of the complexity of the interaction logic of physical simulation and model interaction. Nonetheless, the development process has greatly strengthened my technical skills in 3D modelling, Unity development and physics simulation simulation.

By critically reflecting on and analysing these issues, I realised that I should plan better in the upfront technical assessment and time management phases when undertaking future project management. This includes analysing in detail the technical difficulty and time required for each development phase during the project planning stage, as well as being prepared in advance to deal with unforeseen circumstances as they occur. In addition, it is also important to communicate more with team members and supervisors at the technical level during the project to accelerate the speed of breaking through the development challenges.

Overall, the development process of this project was full of difficulties and challenges, but it also strengthened my technical level, project management skills and

the ability to solve unknown problems. Through this project, I was able to improve my professional skills in a variety of different technical areas, as well as develop my ability to effectively self-manage and work in a team on projects with complex environments. Through this critical self-assessment and reflection, I was able to better identify and improve on my shortcomings in my work and prepare myself for the greater challenges I may face in the future.

References

[1] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[2] S. Thompson, T. Dowrick, M. Ahmad, *et al.*, "SciKit-Surgery: Compact Libraries for Surgical Navigation," *International journal of computer assisted radiology and surgery*, vol. 15, no. 7, pp. 1075–1084, 2020. DOI: 10.1007/s11548-020-02180-5.

[3] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033. DOI: 10.1109/IROS.2012.6386109.

[4] A. Juliani, V.-P. Berges, E. Vckay, *et al.*, "Unity: A general platform for intelligent agents," *ArXiv*, vol. abs/1809.02627, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52185833.

[5] D. Kundrat, G. Dagnino, T. M. Y. Kwok, *et al.*, "An mr-safe endovascular robotic platform: Design, control, and ex-vivo evaluation," *IEEE Transactions on Biomedical Engineering*, vol. 68, no. 10, pp. 3110–3121, 2021. DOI: 10.1109/TBME.2021.3065146.

[6] X. Li, S. Guo, P. Shi, X. Jin, and M. Kawanishi, "An endovascular catheterization robotic system using collaborative operation with magnetically controlled haptic force feedback," *Micromachines*, vol. 13, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:247827780.

[7] C. Mialhe, A. Chaudhuri, J. Raffort, and F. Lareyre, "Feasibility of the application of holographic augmented reality in endovascular surgery using microsoft hololens head-mounted display.," *Annals of vascular surgery*, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:235675708.

[8] P. Liu, C. Li, C. Xiao, *et al.*, "A wearable augmented reality navigation system for surgical telementoring based on microsoft hololens," *Annals of Biomedical Engineering*, vol. 49, pp. 287–298, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:219328694.

[9] A. Thabit, W. J. Niessen, E. B. Wolvius, and T. van Walsum, "Evaluation of marker tracking using mono and stereo vision in microsoft hololens for surgical navigation," in *Medical Imaging*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:247914326.

[10] A. Palumbo, "Microsoft hololens 2 in medical and healthcare context: State of the art and future prospects," *Sensors (Basel, Switzerland)*, vol. 22, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252891043.

[11] The Qt Company, *Qt - cross-platform software development for embedded and desktop*, [Online; accessed 29-April-2024], 2024. [Online]. Available: https://www.qt.io.

[12] W. Schroeder, K. Martin, and B. Lorensen, *The Visualization Toolkit (4th ed.)* Kitware, 2006, ISBN: 978-1-930934-19-1.

[13] M. Sun and S. Wu, "A software development of dicom image processing based on qt, vtk and itk," in *2013 IEEE International Conference on Medical Imaging Physics and Engineering*, 2013, pp. 231–235. DOI: 10.1109/ICMIPE.2013.6864541.

[14] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, 590–596 vol. 2. DOI: 10.1109/CVPR.2005.74.

[15] Wikipedia contributors, *Alpha compositing — Wikipedia, the free encyclopedia*, [Online; accessed 1-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Alpha_compositing&oldid=1220212606.

[16] C. Liu, Y. Liang, and W. Wen, "Fire image augmentation based on diverse alpha compositing for fire detection," in *2022 15th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*, 2022, pp. 1–6. DOI: 10.1109/CISP-BMEI56279.2022.9979846.

[17] A. Settimi, J. Gamerro, and Y. Weinand, "Augmented-reality-assisted timber drilling with smart retrofitted tools," *Automation in Construction*, vol. 139, p. 104 272, 2022, ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2022.104272. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580522001455.

[18] S. Wang and J. Zhang, "Research and implementation of real-time render optimization algorithm based on gpu," *Journal of Physics: Conference Series*, vol. 2136, no. 1, p. 012 059, Dec. 2021. DOI: 10.1088/1742-6596/2136/1/012059. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/2136/1/012059.

[19] K. Dang, J. Yang, and J. Yuan, "Adaptive exponential smoothing for online filtering of pixel prediction maps," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 3209–3217. DOI: 10.1109/ICCV.2015.367.

[20] S. Ekern, "Adaptive exponential smoothing revisited," *The Journal of the Operational Research Society*, vol. 32, no. 9, pp. 775–782, 1981, ISSN: 01605682, 14769360. [Online]. Available: http://www.jstor.org/stable/2581393 (visited on 05/01/2024).

[21] J. W. Taylor, "Smooth transition exponential smoothing," *Journal of Forecasting*, vol. 23, no. 6, pp. 385–404, 2004. DOI: https://doi.org/10.1002/for.918. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/for.918. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/for.918.

[22] Investopedia, *Exponential moving average (ema) - definition*, https://www.investopedia.com/terms/e/ema.asp, Accessed: 2024-05-01, 2024.

[23] P.-L. Hsieh, C. Ma, J. Yu, and H. Li, "Unconstrained realtime facial performance capture," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1675–1683. DOI: 10.1109/CVPR.2015.7298776.

[24] Vinay, *Complex exponential smoothing (ces)*, https://medium.com/@vinay1996/complex-exponential-smoothing-ces-63d1c6ddfc78, Accessed: 2024-05-01, 2018.

[25] I. Svetunkov, N. Kourentzes, and J. K. Ord, "Complex exponential smoothing," *Naval Research Logistics (NRL)*, vol. 69, no. 8, pp. 1108–1123, 2022. DOI: https://doi.org/10.1002/nav.22074. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nav.22074. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nav.22074.

[26] J. Tan, J. Chen, Y. Wang, L. Li, and Y. Bao, "Design of 3d visualization system based on vtk utilizing marching cubes and ray casting algorithm," in *2016 8th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC)*, vol. 02, 2016, pp. 192–197. DOI: 10.1109/IHMSC.2016.153.

[27] S. Sun, J. He, and L. Ma, "3d visualization system for medical image based on vtk and ok series image board," in *2011 International Conference on Computer Science and Service System (CSSS)*, 2011, pp. 951–954. DOI: 10.1109/CSSS.2011.5974642.

[28] Y. Wang, W. Wan, X. Zhou, S. Mao, and HuiLi, "Local transparency technique for heart model from vtk," in *IET International Communication Conference on Wireless Mobile and Computing (CCWMC 2011)*, 2011, pp. 139–142. DOI: 10.1049/cp.2011.0863.

[29] vtk2jbean, *Vtkimageimport documentation*, https://jbeanvtk.sourceforge.net/apidocs/vtk/vtkImageImport.html, Accessed: 2024-05-05.

[30] *Vtkimageimport class reference*, https://vtk.org/doc/nightly/html/classvtkImageImport.html, Accessed: 2024-05-05.

[31] L. Xiaoqi, J. Dongzheng, Z. Baohua, and R. Xiaoying, "Study and implementation for interactive cutting of 3d medical image based on vtk," in *Proceedings of 2011 Cross Strait Quad-Regional Radio Science and Wireless Technology Conference*, vol. 2, 2011, pp. 1351–1354. DOI: 10.1109/CSQRWC.2011.6037214.

[32] W. Yang, Z. Zeng, and Y. Bai, "3d reconstruction system on coarse aggregate microfabric based on vtk," in *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 4, 2009, pp. 85–88. DOI: 10.1109/ICICISYS.2009.5357714.

[33] A. Vacanti and D. DeMarle, *VTK Technical Highlight: Dual Depth Peeling*, https://www.kitware.com/vtk-technical-highlight-dual-depth-peeling/, Accessed: 2024-05-05, Oct. 2016.

[34] *Vtk examples for camera in c++*, https://examples.vtk.org/site/Cxx/Visualization/Camera/, Accessed: 2024-05-05.

[35] *Documentation for vtkcamera*, https://www.geologie.uni-freiburg.de/root/manuals/vtkman/vtkCamera.html, Accessed: 2024-05-05.

[36] G. Lazar and R. Penea, *Mastering Qt 5: Create stunning cross-platform applications using C++ with Qt Widgets and QML with Qt Quick*. Packt Publishing Ltd, 2018.

[37] A. Ranne, Y. Velikova, N. Navab, and F. R. y Baena, "Aiareseg: Catheter detection and segmentation in interventional ultrasound using transformers," *ArXiv*, vol. abs/2309.14492, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:262826706.

[38] T. Jianu, B. Huang, M. E. M. K. Abdelaziz, *et al.*, *Cathsim: An open-source simulator for endovascular intervention*, 2022. arXiv: 2208.01455 [cs.RO].

[39] Meta Quest, *Meta quest development - unity documentation*, https://developer.oculus.com/documentation/unity/, Accessed: 2023-11-02, 2023.

[40] Microsoft, *Welcome to mixed reality feature tool*, Microsoft Docs, Accessed: 2024-05-01, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/welcome-to-mr-feature-tool.

[41] *Fixed joint*, Unity Documentation, Accessed: yyyy-mm-dd, 2024. [Online]. Available: https://docs.unity3d.com/Manual/class-FixedJoint.html.

[42] *Box collider*, Unity Documentation, Accessed: yyyy-mm-dd, 2024. [Online]. Available: https://docs.unity3d.com/Manual/class-BoxCollider.html.

[43] *Mesh collider*, Unity Documentation, Accessed: yyyy-mm-dd, 2024. [Online]. Available: https://docs.unity3d.com/Manual/class-MeshCollider.html.

[44] Python Software Foundation, *Unittest – unit testing framework*, https://docs.python.org/3/library/unittest.html, Accessed: 2023-05-01, 2023.

[45] NUnit Development Team, *Nunit: A unit-testing framework for all .net languages*, https://nunit.org/, Accessed on 2023-05-01, 2023.

[46] University of Liverpool, *Comp390 honours year project 2023-24 ethical guidance*, Accessed: 2023-11-03, 2023. [Online]. Available: https://student.csc.liv.ac.uk/internal/modules/comp390/2023-24/ethics.php.

[47] BCS, *Guidelines on course accreditation: Information for universities and colleges*, Accessed: 2024-05-01, Jan. 2020. [Online]. Available: https://www.bcs.org/media/11ofljxo/course-accreditation-guidelines.pdf.

[48] "Aruco markers." (2021), [Online]. Available: https://fab.cba.mit.edu/classes/865.21/people/zach/arucomarkers.html (visited on 04/29/2024).

[49] Blender Foundation, *Blender*, https://www.blender.org, Accessed on: 2024-05-01, 2024. [Online]. Available: https://www.blender.org.

[50] Wikipedia contributors, *Blender (software) — Wikipedia, the free encyclopedia*, [Online; accessed 1-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Blender_(software)&oldid=1220122561.

[51] Wikipedia contributors, *Github — Wikipedia, the free encyclopedia*, [Online; accessed 3-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=GitHub&oldid=1221852262.

[52] Wikipedia contributors, *Microsoft hololens — Wikipedia, the free encyclopedia*, [Online; accessed 3-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Microsoft_HoloLens&oldid=1221749039.

[53] Microsoft, *Microsoft hololens*, https://www.microsoft.com/opencv_libraryen-us/hololens, Accessed: 2024-05-01, 2024.

[54] Wikipedia contributors, *Qt (software) — Wikipedia, the free encyclopedia*, [Online; accessed 29-April-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Qt_(software)&oldid=1219937369.

[55] Wikipedia contributors, *Pyside — Wikipedia, the free encyclopedia*, [Online; accessed 7-May-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=PySide&oldid=1214569279.

[56] The Qt Company, *Pyside6: Python bindings for the qt cross-platform application framework*, Accessed: 2024-05-01, 2024. [Online]. Available: https://pypi.org/project/PySide6/.

[57] Wikipedia contributors, *Opencv — Wikipedia, the free encyclopedia*, [Online; accessed 29-April-2024], 2024. [Online]. Available: https://en.wikipedia.org/w/index.php?title=OpenCV&oldid=1208982530.

Glossary

Adaptive Exponential Smoothing Adaptive exponential smoothing models are designed to improve performance by letting the smoothing parameter vary according to the most recent forecasting accuracy. [20], [21] 14

AES Adaptive Exponential Smoothing 15

Alpha Blending In computer graphics, alpha compositing or alpha blending is the process of combining one image with a background to create the appearance of partial or full transparency. It is often useful to render picture elements (pixels) in separate passes or layers and then combine the resulting 2D images into a single, final image called the composite. Compositing is used extensively in film when combining computer-rendered image elements with live footage. Alpha blending is also used in 2D computer graphics to put rasterized foreground elements over a background.[15] 14

Aruco Marker ArUco markers are 2D binary-encoded fiducial patterns designed to be quickly located by computer vision systems. ArUco marker patterns are defined by a binary dictionary in OpenCV, and the various library functions return pattern IDs and pose information from scanned images.[48] 4

Blender Blender is a free and open-source 3D computer graphics software tool set used for creating animated films, visual effects, art, 3D-printed models, motion graphics, interactive 3D applications, virtual reality, and, formerly, video games. Blender's features include 3D modelling, UV mapping, texturing, digital drawing, raster graphics editing, rigging and skinning, fluid and smoke simulation, particle simulation, soft body simulation, sculpting, animation, match moving, rendering, motion graphics, video editing, and compositing[49], [50]. 59

CES Complex Exponential Smoothing 15

Complex Exponential Smoothing Complex exponential smoothing is a time series forecasting method that combines exponential smoothing with trend and seasonality. It is a variant of the standard exponential smoothing method, which is a simple technique for smoothing out data by using a weighted average of past observations.[24] 15

EMA Exponential Moving Average 15

Exponential Moving Average An exponential moving average (EMA) is a type of moving average (MA) that places a greater weight and significance on the most recent data points. The exponential moving average is also referred to as the exponentially weighted moving average. An exponentially weighted moving average reacts more significantly to recent price changes than a simple moving average simple moving average (SMA), which applies an equal weight to all observations in the period.[22] 15

GitHub GitHub is a developer platform that allows developers to create, store, manage and share their code. It uses Git software, providing the distributed version control of Git plus access control, bug tracking, software feature requests, task management, continuous integration, and wikis for every project. Headquartered in California, it has been a subsidiary of Microsoft since 2018. It is commonly used to host open source software development projects. As of January 2023, GitHub reported having over 100 million developers and more than 420 million repositories, including at least 28 million public repositories. It is the world's largest source code host as of June 2023.[51] 62

Microsoft Hololens Microsoft HoloLens is an augmented reality (AR)/mixed reality (MR) headset developed and manufactured by Microsoft. HoloLens runs the Windows Mixed Reality platform under the Windows 10 operating system. Some of the positional tracking technology used in HoloLens can trace its lineage to the Microsoft Kinect, an accessory for Microsoft's Xbox 360 and Xbox One game consoles that was introduced in 2010[52], [53] 58, 61

MRTK Microsoft Mixed Reality Toolkit-Unity 2, 38, 40, 60, 61

OpenCV OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly for real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage, then Itseez (which was later acquired by Intel). The library is cross-platform and licensed as free and open-source software under Apache License 2. Starting in 2011, OpenCV features GPU acceleration for real-time operations.[54] 2, 4

PySide6 PySide is a Python binding of the cross-platform GUI toolkit Qt developed by The Qt Company, as part of the Qt for Python project. It is one of the alternatives to the standard library package Tkinter. Like Qt, PySide is free software. PySide supports Linux/X11, macOS, and Microsoft Windows. The project can also be cross compiled to embedded systems like Raspberry Pi, and Android devices.[55], [56] 33

Qt Qt (pronounced "cute" or as an initialism) is cross-platform application development framework for creating graphical user interfaces as well as cross-platform applications that run on various software and hardware platforms such as Linux, Windows, macOS, Android or embedded systems with little or no change in the underlying codebase while still being a native application with native capabilities and speed.[57] 4

RVEC rotation vector 23–25

TVEC translation vector 23–25

VTK The Visualization Toolkit (VTK) is an open-source, freely available software system for 3D computer graphics, modeling, image processing, volume rendering, scientific visualization, and 2D plotting. It supports a wide variety of visualization algorithms and advanced modeling techniques, and it takes advantage of both threaded and distributed memory parallel processing for speed and scalability, respectively.[12] 4