Bilkent University

Department of Computer Engineering

Senior Design Project

MoveIt, Indoor Manipulation : 3D Semantic Reconstruction, Display and Manipulation System

High Level Design Report

Barış Can

Faruk Oruç

Mert Soydinç

Pınar Ayaz

Ünsal Öztürk

Supervisor: Associate Professor Selim Aksoy, Ph.D.

Jury Members: Assistant Professor Shervin Rahimzadeh Arashloo,Ph.D.

Assistant Professor Hamdi Dibeklioğlu, Ph.D.

High Level Design Report

December 31, 2019

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Contents

High Level Design Report

MoveIt, Indoor Manipulation : 3D Semantic Reconstruction, Display and Manipulation System

1. Introduction

The popularity of smartphones rises daily. While they have many uses, it can be argued that taking and sharing photos are among the most widespread uses of today's incredibly powerful smartphones. Photos capture a moment in nature or an indoor scene, and they all have visual and spatial information of the objects contained within them. It is possible to extract this information, and more information can be extracted as one adds additional angles and positions from which the original scene is viewed. The amount of information one can extract is directly proportional to the extra number of angles and positions, and the quality of these images. This information then may be used to model the environment captured by these images in 3D. This 3D recreation of the room will be fully visualized in a VR environment that enables the user to freely move, scale, and rotate objects in the 3D reconstruction of the room using a VR Headset.

1.1. Purpose of the system

MoveIt: Indoor Manipulation aims to bring our homes into virtual reality by recreating 3D indoor scenes using smartphone camera. With MoveIt, a quick scan of a room will bring the objects contained within this room into VR where they can be freely moved, interacted with, and manipulated. MoveIt allows the users to visualise the new look of their room with the new locations of the objects without actually moving said objects.

MoveIt is also a helpful tool that enables the user's aesthetic ability by allowing them to redecorate their room with ease while also providing functional help such as object boundary and collision detection. Users are not limited to their own objects to place. They can download other objects uploaded by other users and place these objects in their rooms.

1.2. Design goals

1.2.1. Usability

The MoveIt application should have a user-friendly interface that would enable different users from various age and knowledge groups to use the application with ease, and it should not take more than 30 minutes for a user to get accustomed to the interface. It should include a user manual, an instructions page and YouTube videos explaining the users the key concepts of the application, such as how to record a room and how to move objects around. These tutorials should not take more than two minutes for a user to complete. Users also should be able to move around objects with natural controls that are easy to perform and comes as natural, and a user should not spend more than five seconds on average to perform a given operation on an object.

1.2.2. Reliability

3D reconstruction of the objects in a room must be as detailed as possible and the application should be able to recreate obscure areas of the objects with high accuracy by making correct assumptions. The gaps in the reconstructed textures should not be more than 10% of the visible surfaces of the scene, and the artifacts due to the reconstruction should not yield an error more than 10% in terms of volume of the individual object. Different components of the application should work together harmoniously. To elaborate, MoveIt will have a smartphone application as well as a background side where 3D reconstruction will take place. Since the user interacts with the smartphone application, the background computation times should be optimized as much as possible to guarantee responsiveness. The computation for the reconstruction should not take more than five minutes. When the reconstructed objects are manipulated by the user (e.g. moved around), the previous place of the object as well as the new one should be altered and displayed correctly by the application. The only tolerable error should be the floating point representation rounding-off errors.

1.2.3. Security

The application should be able to protect the data that the user uploads to the system such as recordings or just their email addresses since these data could be classified as personal. Since the personal videos will not be kept in the user's storage, a secure hashing scheme must be used, and this hashing scheme should be considered as "having minor weaknesses" in the worst case [5].

1.2.4. Smartphone Friendliness

The users are expected to use smartphones as an interface to interact with the MoveIt application. Therefore, the application should not be very taxing on smartphones and the power usage as well as the mobile data usage should be optimized to ensure user satisfaction. The user should be able to use the application for at least two hours before the battery of the smartphone runs out. The reconstructed scene should run smoothly.

1.2.5. Scalability

Since the computations will be done on a remote server, it might require too much computational power if several users send their videos at the same time. In order to solve this, either computational power of the server should be increased by hiring a server with double the number of cores and double the amount of memory. Once scaling up costs exceed 16 times the initial server rent, distributed versions of the algorithms to be developed should be introduced, and multiple servers should be used to carry out the computations. In addition, problems in the system caused by an increase in people must be addressed quickly.

1.2.6. Availability

The system should be available for the users and function at all times. But since server failures are inevitable, we are aiming for an SLA level of 99.9% (Yearly 8h46m of downtime)[6].

1.3. Definitions, acronyms, and abbreviations

Semantic Segmentation: The operation of linking each pixel of an image to a class label. In our project, this corresponds to linking the pixels of the detected furniture in the image to the correct label of that furniture.

3D Reconstruction: The process of capturing the shape and appearance of real objects. In our project, we are trying to reconstruct the geometry of captured indoor scenes in 3D accurately.

Semantic Label: The label assigned to the pixels in an image after performing semantic segmentation. For example, the label ‘chair’ assigned to the pixels in the image that belong to a chair.

Geometric 3D Mesh: The structural build of a 3D model consisting of polygons. 3D meshes use reference points in X, Y and Z axes to define shapes with height, width and depth. In our project, these will be used to construct a 3D model for the detected objects from the indoor scenes.

Texture Analysis: Refers to the characterization of regions in an image by their texture content.

VR Scene Rendering: Producing a view of a scene for humans in a VR environment that aims to be as close to the actual scene in reality as possible by using the 3D data. This will be performed to display the 3D reconstructed indoor scenes in our project.

1.4. Overview

MoveIt: Indoor Manipulation aims to develop an application which will semantically reconstruct the geometry of indoor areas, such as a living room or a kitchen, in 3D by using videos, live scans/feed obtained from the camera of a mobile device. The users will need to create an account either by our sign up screen or their Google account. These accounts will contain users’ email addresses, their objects and their scenes. After they scan, the 3D reconstruction process will produce a 3D scene and will label the reconstructed objects and geometry semantically by attaching a semantic label to each object. It will also allow the users of the application to manipulate the scene by allowing the users to move, rotate, scale, and deform the geometry of the objects. Texture obtained from the live feed will also be mapped to the corresponding meshes. The application will store the 3D reconstruction information along with the texture, label, and mesh information of a given object on a database for later data analytics and machine learning purposes, and will be open to the public as a labelled set of 3D objects, which may be suitable for supervised learning purposes. However, these geometric meshes will be added to the database if and only if the user gives permission to do so, and these meshes will be stored anonymously. If they decide to upload their furniture' meshes, they can also tag them as they wish.

The main purpose of the application is to allow the users to manipulate an indoor scene without actually having to add, move, or alter the objects in the scene. For instance, if the user wishes to furnish a room, the user is able to use this application to reconstruct the geometry of the room and the current objects within the room as a 3D scene in VR. The application will allow the necessary facilities to manipulate the positions, rotations, scales, and the geometry of the scene, while also allowing the user to place other objects in this scene. These new objects can be fetched from the database that the application keeps or may be supplied by the user in the form of a 3D mesh and texture. Objects can be queried and retrieved from the database using semantic labels or uploaders' specified tags. As an example, the user will be able to request a 3D mesh representation of a chair, and if available, the application will present a set of chairs that the user can add to the scene present in the database. A user will not be able to upload objects directly to the database, and user-provided objects will only be stored locally. Another relevant feature is to save a scene with a given set of objects, geometry, and the texture locally on a device for later access and modification.

Another purpose, though not the main purpose of this application is to create a publicly available dataset of labelled 3D objects and textures corresponding to these objects. This dataset might be used to study various aspects of 3D reconstruction, texture analysis, and the relationship between these and the semantic labels provided alongside the objects. The dataset may also be used for supervised learning purposes, and the more the app is used, the larger the dataset will be.

As for the platforms and the hardware on which the application will be deployed, MoveIt: Indoor Manipulation will rely on many other frameworks and paradigms. Retrieval of live scans/feed will depend on Android infrastructure; the texture analysis, texture to geometry mapping, 3D reconstruction, semantic segmentation of the live scan and semantic labelling of meshes will be carried out on a remote server with sufficient hardware. The models used for the semantic segmentation of images will be based on pre-trained convolutional neural networks, and for the 3D reconstruction and texture mapping step, popular frameworks such as OpenCV will be used [8]. Once a model of the scene is created, the user may choose to have this scene rendered on a VR headset, or on a personal computer. The user will be able to control and manipulate the scene using VR controllers, or keyboard and mouse.

2. Current software architecture

Below, current software architectures similar to MoveIt will be discussed.

2.1. Hololens

- It is mixed reality technology that allows users to perform several actions such as playing games, changing the objects in the room, or get some holographic statistics [2].
- Users can interact with objects in the room via a VR headset.
- VR headset scans the room, performs semantic segmentation and users can grab and move objects, rotate them, scale them.
- Users can design hologram objects.

2.2. The Sims

- The Sims is a 3D life simulation game series that includes a mechanic to furnish houses as the user wishes. This furnishing mechanic is the part of this game that MoveIt takes inspiration from. [3]
- Players can build rooms and houses as they wish and furnish these rooms with the objects that the game provides.
- Players select the object from the objects menu which is divided by various categories, and place the object into the room. They can rotate the objects, change their colors and designs and scale them.

- Players can look at the rooms in any angle they prefer in third person perspective.

2.3. Roomplaner

- It is a mobile application that is similar to sims in a way that users can plan a virtually created room[4].
- Users can change the size of the room, add windows or doors and see those changes in 3D.
- Uses IKEA dataset (objects).
- Users can rotate, scale and change the color of the objects.
- Users can see the size of the objects from the air and a map of the room or house. This makes the application more realistic and useful for designers.

2.4. IKEA

- Users can make a scene for their room, select furniture from IKEA database and place them into the AR scene.
- Users can rotate or scale furniture that you placed.
- Users can search for furniture from categories and check their prices from IKEA.
- There are several restrictions such as the room should be well lit, or the floor should be free of clutter.
- You can share your place with social apps like Whatsapp or Instagram [1].

3. Proposed software architecture

The following parts are the software architecture details of the proposed system.

3.1. Overview

In the following sections, proposed software architecture of MoveIt is discussed. Subsystem decomposition is explained in detail and the subsystem decomposition diagram is provided. In the following section, hardware and software mappings are given for the application. In addition, persistent data management methods, application's access and security, its global software control and its boundary conditions such as starting, terminating and failure scenarios are explained thoroughly.

3.2. Subsystem decomposition

The MoveIt software architecture is divided into four main subsystems, namely the Client, Server, Renderer, and 3D Semantic Reconstruction subsystems. The Client and Server subsystems are based on Client-Server architecture, and their main purpose is to provide a software backbone for the transmission, storage, and processing of data. These subsystems allow the users to send their data to the server, have it processed and stored, and retrieve a semantic 3D reconstruction of their rooms in a streamlined and efficient manner. The separation between the client and the server incurs in an enhancement in privacy and safety measures, as the server manages user connections in a peer oblivious way from the perspective of the client, i.e. the users are semantically unaware of the other users currently connected to the server.

The Server subsystem and its deployment on a separate and a computationally powerful machine, relative to mobile devices and personal computers, also yields an increase in the efficiency of MoveIt, as processing the data on a server capable of performing feature extraction, geometry processing, and parametric mapping of textures and displacements in a parallelized manner. The server also maintains privacy and safety through making the database involved in the storage of meshes and textures to be unavailable to outside users, as a result of the encapsulation present in the design of the subsystems.

The Client and the Server subsystems communicate through a TCP interface, using a MoveIt specific data format encapsulated in a Data object. The data received or sent through a connection is preprocessed according to application specific file/data headers, which is later converted to conventional formats. E.g. the client creates a connection to the server through the services provided by the Server subsystem, and sends the relevant bytes to the server through the connection. The connection blindly passes the bytes to the server along with the application specific header describing the context of the data, and through the services provided by a data decoder, the data is decoded according to the hardcoded headers, and then converted into commercially available formats.

MoveIt also makes use of the MVC pattern for its Renderer subsystem in an event based manner. A thread continuously polls for the inputs arriving to the subsystem, which are then passed to a controller subsystem, which issues modification requests to the relevant subcomponents, which in turn update the abstract representation of 3D models, and in the next render loop, the updated models are rendered, hence changing the view, in accordance to the MVC pattern.

The 3D Semantic Reconstructor does not use any particular design pattern. It provides the logic for geometry processing and has wrapper classes for representing pre-trained neural networks for the purposes of semantic segmentation. The semantic segmentation of successive frames in the video provided by the user is used to estimate depth information and semantic tags for the reconstruction. This subsystem combines these two nested subsystems to produce a final 3D semantic reconstruction of the indoor scene. The relevant software for this subsystem runs exclusively on the server, and the Server subsystem also persists the reconstructions by writing the necessary data to a database. The reconstructed meshes and related parametric texture maps are communicated back to the client in the same manner the client sends the video to the server: the server, through the connection object between the client and the server, sends the data as bytes, and an application specific header for decoding purposes. The data decoder on the client side decodes the data via the header and converts the data into a commercially available format, e.g. .obj files for meshes and hdr files for parametric texture/displacement maps.

Client subsystem also provides the UI for the actual usage of the app, along with several OS and file system utilities for the storage and management of scenes and videos of the user. The subsystem provides different wrappers and UI software for different hardware: one for VR, one for PC, and one for Android devices. These wrappers and user interfaces also provide the necessary facilities for rendering and scene manipulation through the services offered by the Renderer subsystem.

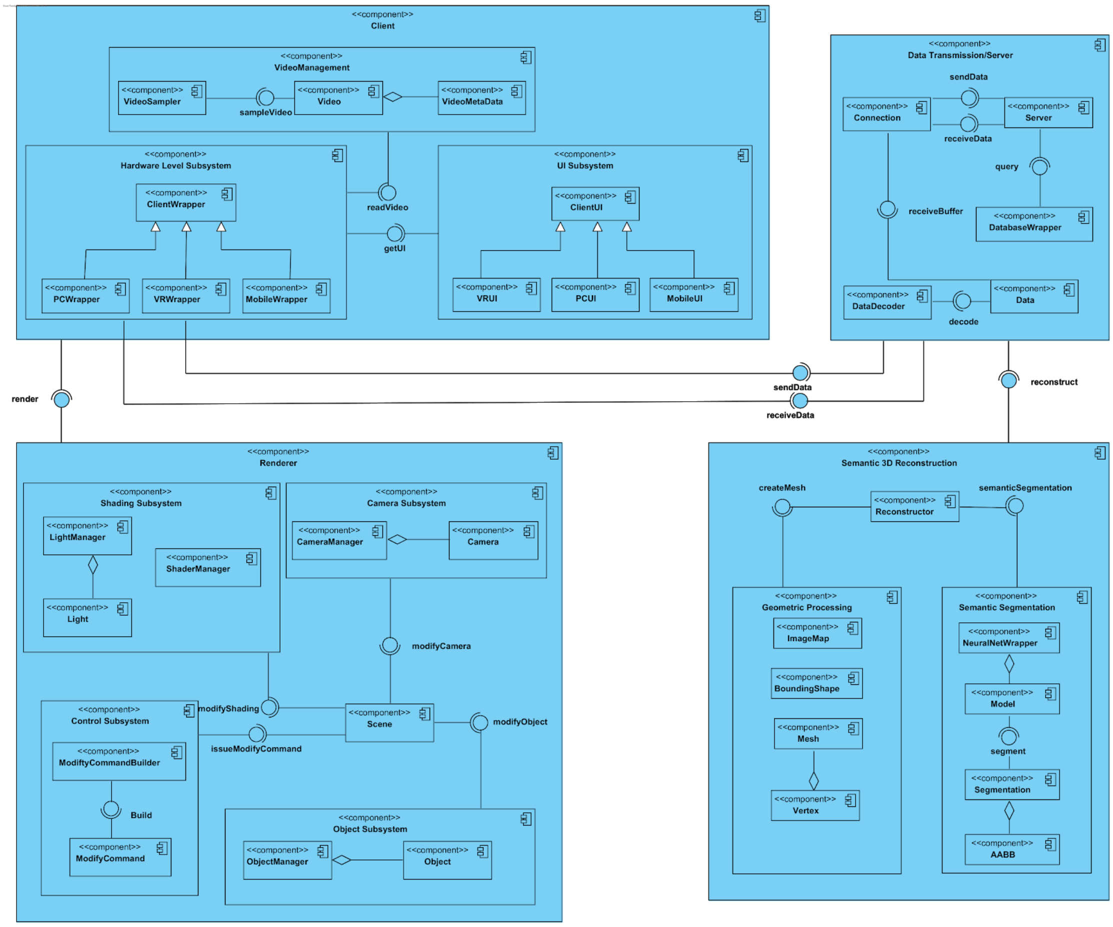

Fig. 1. Subsystem Decomposition Diagram

For a higher quality version, please visit https://imgur.com/YcFGrwc .

3.3. Hardware/Software mapping

There are three types of clients: a Mobile Client that runs on a mobile device and Main Client and VR Client that run on PC. The Mobile Client will use the Android operating system, which will be used for sending collected data to the main server. Main Client will run on a Windows operating system, it will use any browser to access the data stored in the database and it will run on a Unity environment. The VR Client will also use a Unity environment but it will use the 3D components of Unity. All clients will use a TCP/IP connection to communicate with the server. The persistent data will be stored in a MySQL database. MoveIt's main server will run on a Windows machine which uses Microsoft's Azure API.

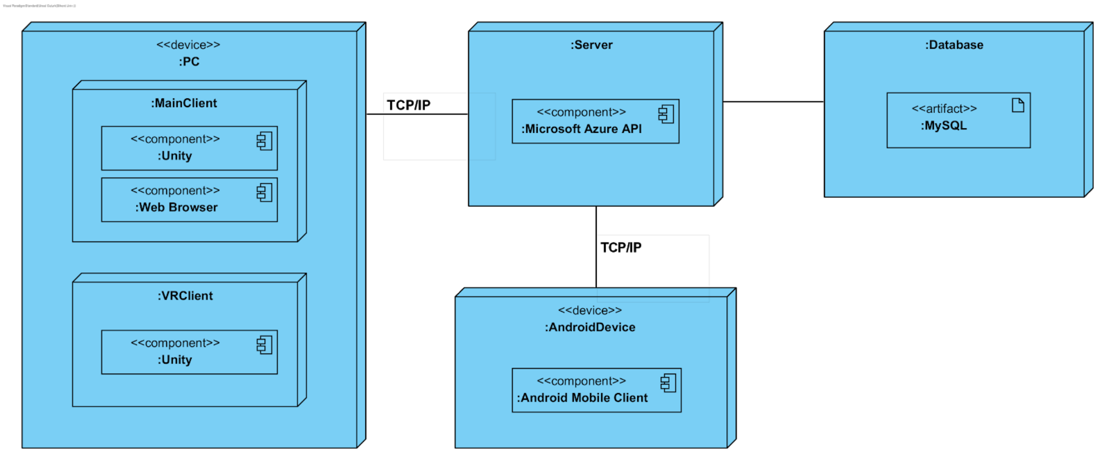

Fig. 2. Deployment Diagram

3.4. Persistent data management

Data storage is a significant part of the working model of MoveIt. Many of its essential components requires persistent data storage. An efficient and secure way to store data is required for keeping user related information such as user ID, password and email address. The system also needs to be able to save uploaded items in order for these items to be used by other users. In order to achieve this, a relational database using MySQL will be used by the main server of the system to store the mentioned data.

On top of this, individual user's scanned items and environment will be stored on their side in order to maximize their privacy.

3.5. Access control and security

The system will use various methods to maximize the security of its stored and accessed data. In order to do this, systems database will use several hashing methods to encrypt its content. Registration to the MoveIt system will require an email address validity of this email address will be checked by an email verification. Users will be able to determine and change their user ID and password only after this verification step is passed. In order to log in to the system, users will be required to enter their ID and password.

Any data entered into the system by a user will only remain in their devices and it not be shared with other users. However, an option to upload scanned data to the database is given to the users. This will allow users to share their scanned objects and scenes with others. This uploaded objects and scenes will not be associated with the uploaders account in order to protect the user's privacy.

3.6. Global software control

MoveIt uses a centralized main server to control its entire system. This server acts as an event-driven control system which responds to the user's inputs and produces events based on them. SOme notable events are as follows.

When a user logs into the system, their credentials are checked for validity and the access is granted if the inputs match the database values.

When the user uploads their scanned objects to the database, the main server will process this object into a more compact form and will store it in the database.

When the user browses the object workshop, the server queries the database and presents several randomly selected objects to the user. Then the user can search for specific types of objects which prompts the server to search for such objects in the database. The main server then presents the found objects to the user.

3.7. Boundary conditions

3.7.1. Starting the Application

In order to use MoveIt, the user needs to download the Android application from Google Play Store and install it to an Android based device. The user then needs to create an account. They can sign up with a Google account or with an email address. Once the user has an account, they can sign in with their credentials and after giving the required permissions, they can start using the application. For the VR application, users can download the application from our website and start using it with their credentials.

3.7.2. Terminating the Application

The user can log out of the application by clicking the logout button in the sliding panel on the left side of the screen.

3.7.3. Failure in the Application

In case of a crash in the client, if the user uploads some meshes to workshop or reconstructing a scene, the process will be terminated and the user will not lose any data. Since reconstruction and uploading requires internet connection, lack of connection will also cause failures. In addition, if the server machine crashes due to overloading, processes will also be terminated.

4. Subsystem services

4.1. Client

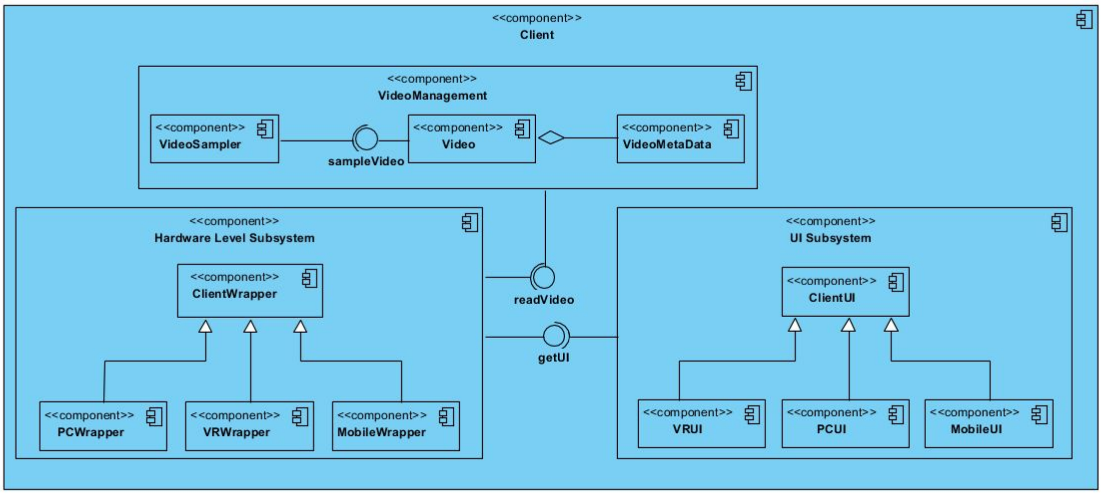

Fig. 3. Client Subsystem

4.1.1. VideoManagement

VideoSampler: Samples the uploaded videos.

Video: Contains the actual video data uploaded by the users.

VideoMetaData: Handles all the metadata related to the uploaded user videos such as their names, resolutions, dimensions etc.

4.1.2. Hardware Level Subsystem

ClientWrapper: Provides an abstraction for Client operations. Subclasses implement hardware specific code for file I/O, UI rendering, controls, control polling, and network interfaces.

PCWrapper: Provides hardware specific code for MoveIt operations on PC.

VRWrapper: Provides hardware specific code for MoveIt operations on VR headsets.

MobileWrapper: Provides hardware specific code for MoveIt operations on mobile devices, in particular Android.

4.1.3. UI Subsystem

ClientUI: This component is the parent component of all the user interface components.

VRUI: Provides the user interface functionality for the VR client.

PCUI: Provides the user interface functionality for the PC client.

MobileUI: Provides the user interface functionality for the Mobile application client.

4.2. Renderer

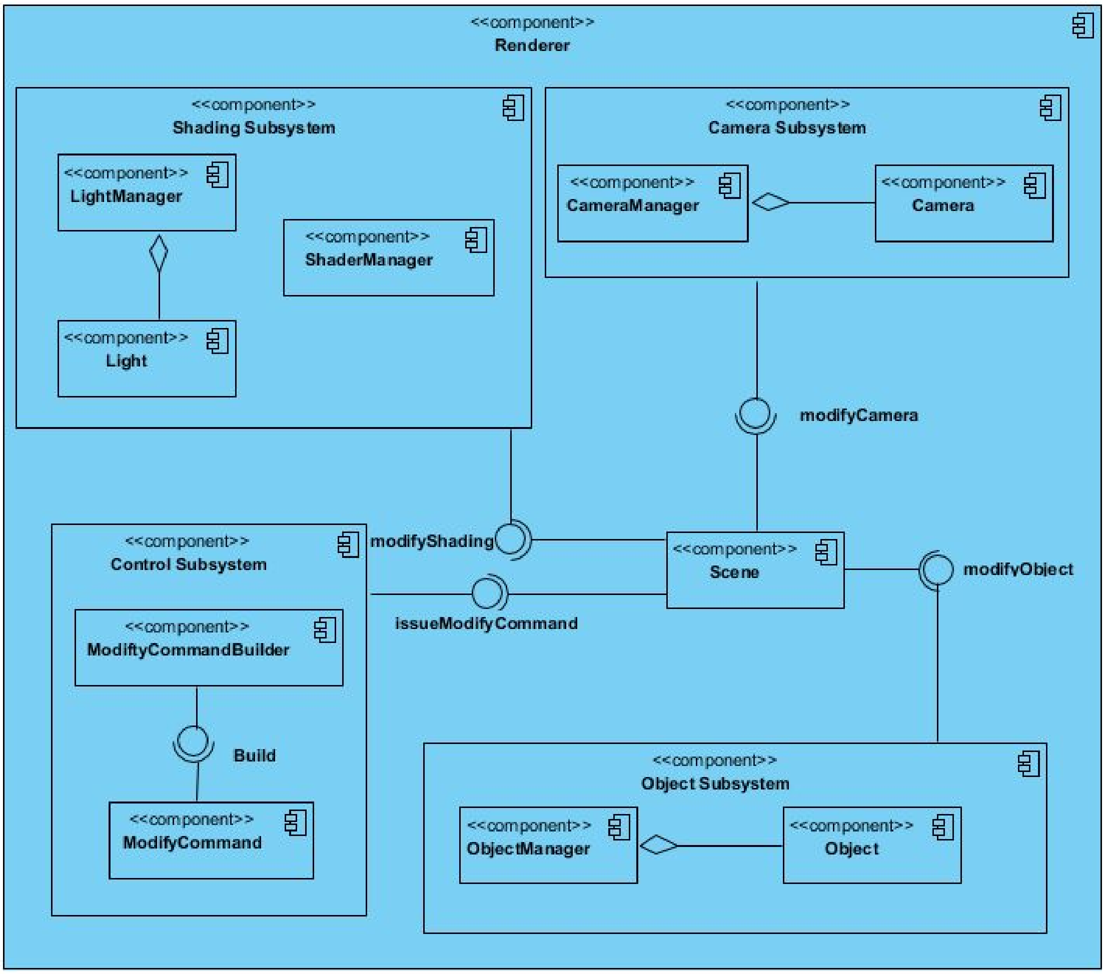

Fig.4. Renderer Subsystem Diagram

4.2.1. Shading Subsystem

LightManager: Contains the light map of the scene. Used to add light sources to the scene or remove light sources from the scene.

ShaderManager: Contains the shader map and manages the shader programs for the scene.

Light: Represents the light in a given scene. It contains position, intensity and color information.

4.2.2. Camera Subsystem

CameraManager: Manages the interactions between the camera and the scene.

Camera: Used to view the scene. It contains the camera position, aperture size, focal length and its projection matrix.

4.2.3. Control Subsystem

ModifyCommandBuilder: Used to easily generate instances of ModifyCommand with a specific command.

ModifyCommand: A wrapper object encapsulating operations on the geometry of the scene or meshes. Used to rotate, scale, slice, replicate, shear, translate objects.

4.2.4. Object Subsystem

Object Manager: Contains all the objects. Used to pass commands to objects and store the objects.

Object: Representation of a real object. It contains the object mesh and texture information.

Scene: Representation of a real scene. It contains an object manager to keep track of the object in it. It can also modify the objects in the scene, light and camera settings. It also contains a render function.

4.3. Server

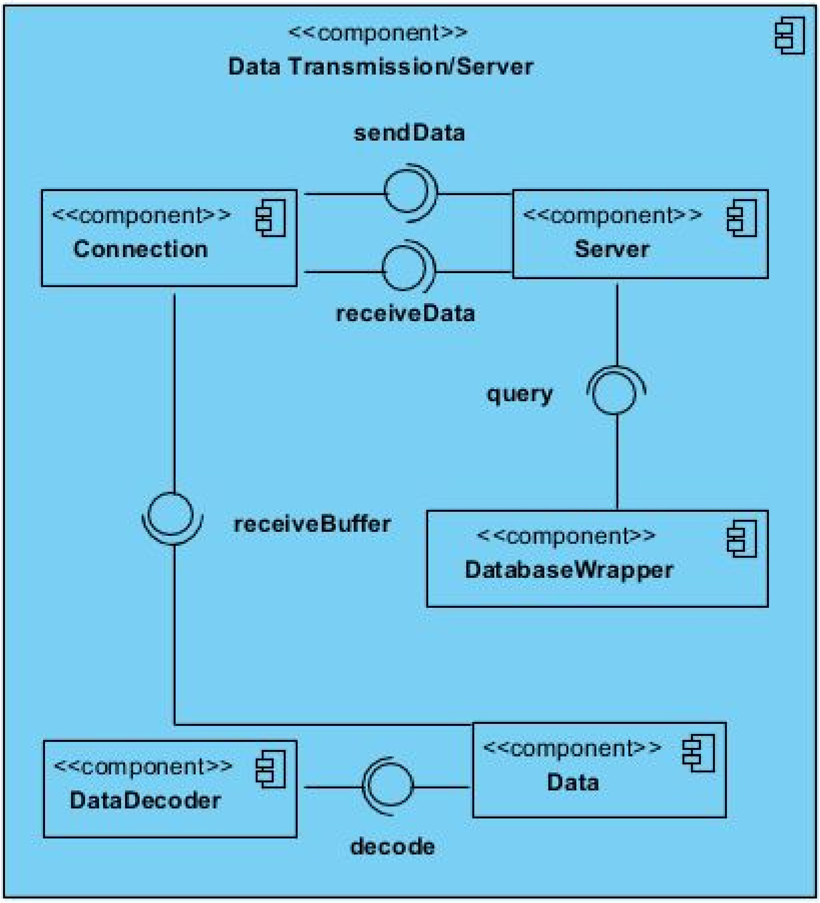

Fig.5. Data Transmission/Server Subsystem Diagram

Connection: Represents the connection between a Server and a Client object. Establishes the connection between these two using a Socket. Allows the transfer of raw bytes of data.

Server: Main server of the application. It is used as a bridge between the reconstructor, database and client.

DatabaseWrapper: Bridge between the server and database. Handles the data connection between them.

DataDecoder: Decodes the raw bytes received through a connection. Has hard-coded instructions to decode application specific headers for particular objects and files. Produces instances of the Data class.

Data: Provides a neat representation of data for application specific purposes. Has fields to determine the header to be used if the data is to be sent over a network, along with file type and metadata.

4.4. Reconstructor

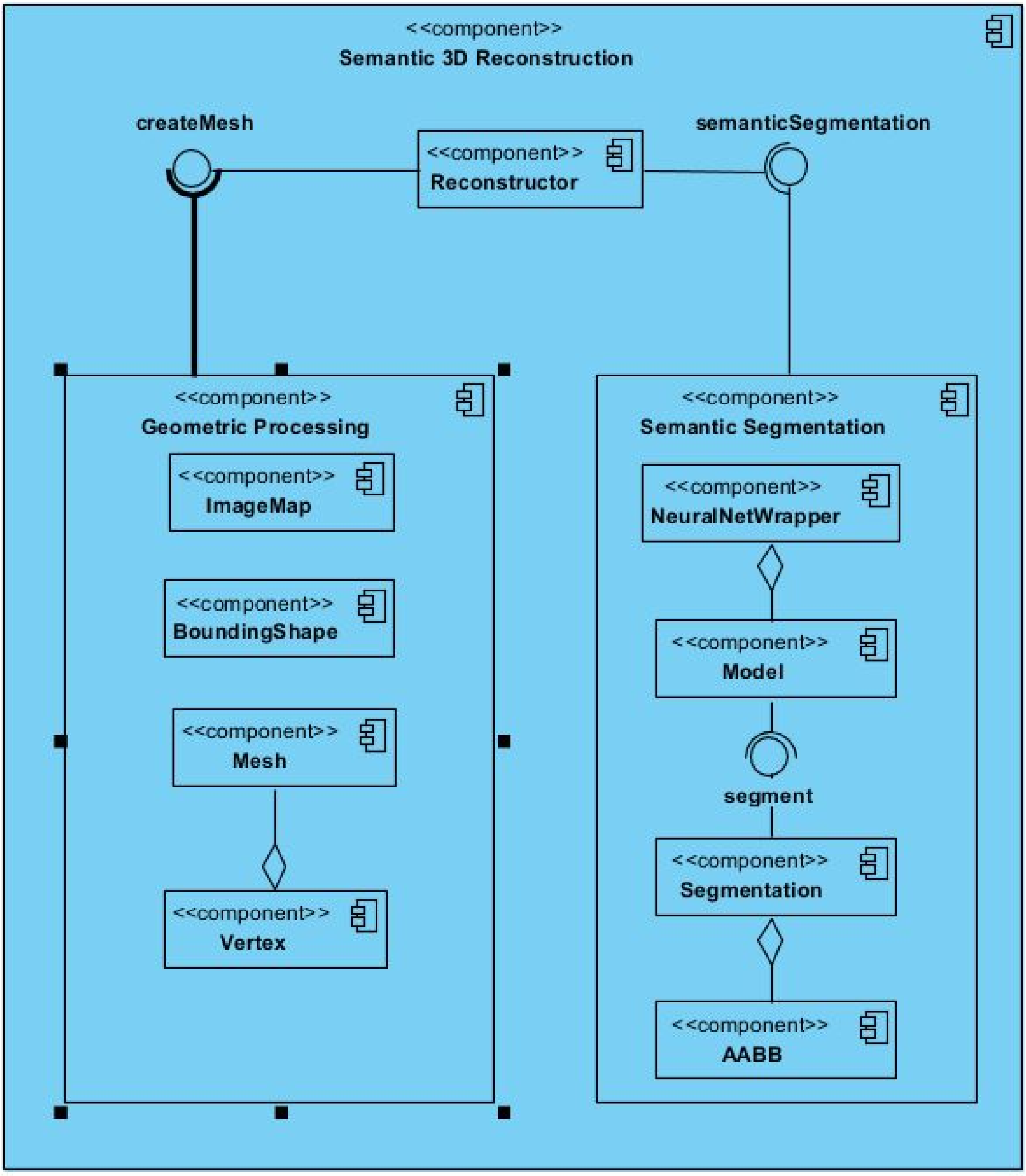

Fig.6. Semantic 3D Reconstruction Subsystem Diagram

4.4.1. Geometric Processing

ImageMap: ImageMap is an image containing the texture map, displacement map, bump map, or any kind of parametric map that can be represented as a 2D image.

BoundingShape: It contains the bounding shape of an object. Used to determine collisions with other objects.

Mesh: Provides an application specific representation of a 3D mesh using index and vertex sets

Vertex: Provides an application specific representation of a 3D vertex in Cartesian coordinates.

4.4.2. Semantic Segmentation

NeuralNetWrapper: Used to communicate between the neural net model and the reconstructor.

Model: Neural net model that produces the segmentations.

Segmentation: Output of the model. It contains the guesses produced by the model for a given image.

AABB: Represents the boundaries of the model’s guesses.

5. New Knowledge Acquired and Learning Strategies Used

All of our current research so far is gathered through online research and some testing on Unity. We particularly used papers from the Technical University of Munich for the 3D reconstruction stage. We are currently focusing on two different papers in order to find ways to recreate the 3D structure of the room and to find boundaries for objects given multiple frames.

In order to recreate the 3D structure of the rooms, we use “Efficient Online Surface Correction for Real-time Large-Scale 3D Reconstruction”[16] as reference. We learnt the logic behind the 3D reconstruction through our research and we are trying to adapt them to our project. We created some basic meshes by extracting geometrical features such as height, width, and depth. We also worked on texture analysis on some objects by extraction of texture information.

We created some scenes in Unity using already created meshes and textures. Using these scenes, we did some experiments on variables such as camera angles, lighting and moving the camera. We also did some research about how we can convert our scenes from mobile application into VR on Unity.

To find boundaries for household objects given a series of frames, we studied the paper “Real-Time Dense Geometry from a Handheld Camera”[17] for guidance. The contents of this paper greatly match our own goals as they also aim to create a 3D depth map for images and reconstruct them using this data from a set of images.

6. Client-Server Class Diagram

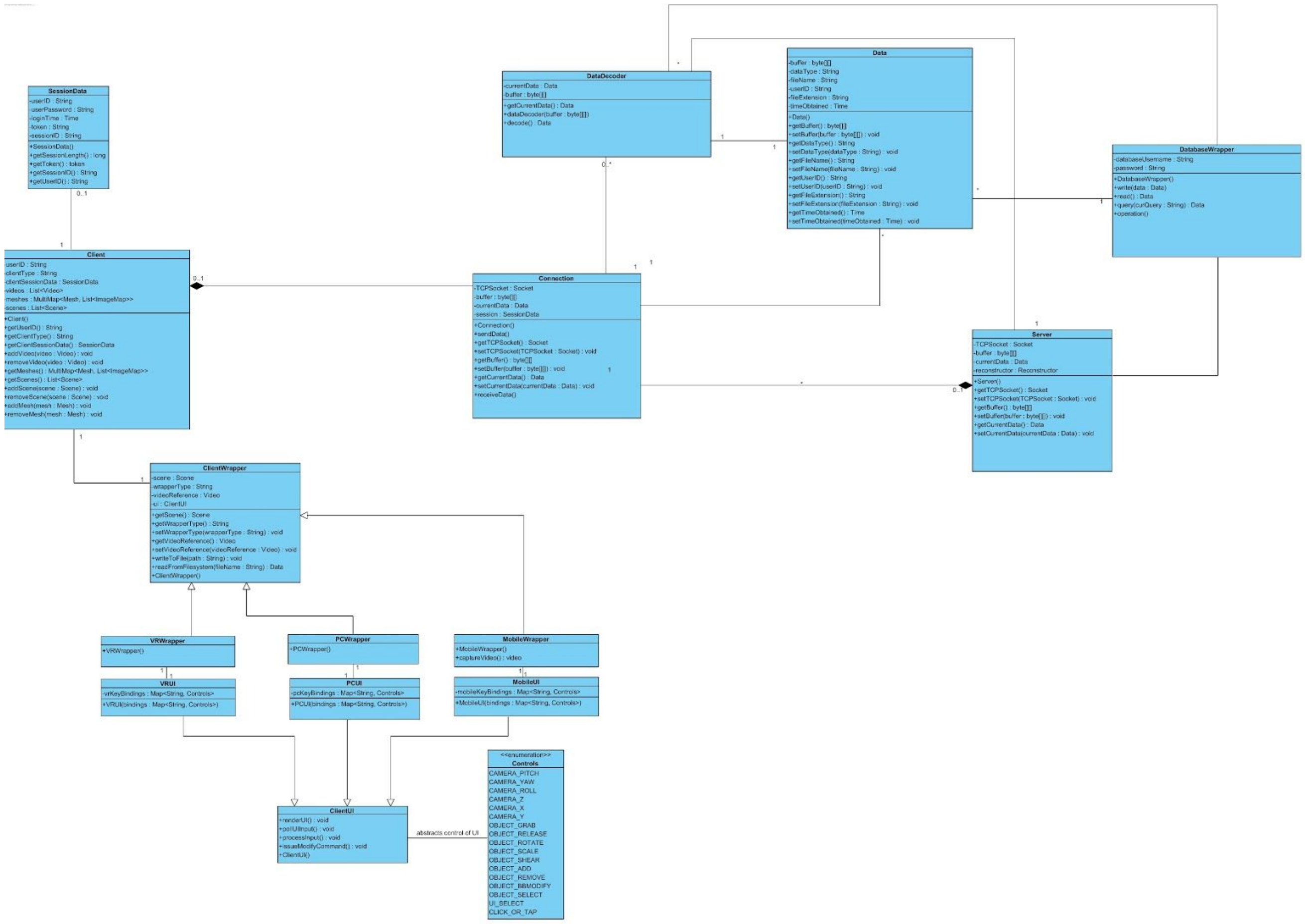

Fig.7. Client-Server Class Diagram

7. Revised Class Diagram

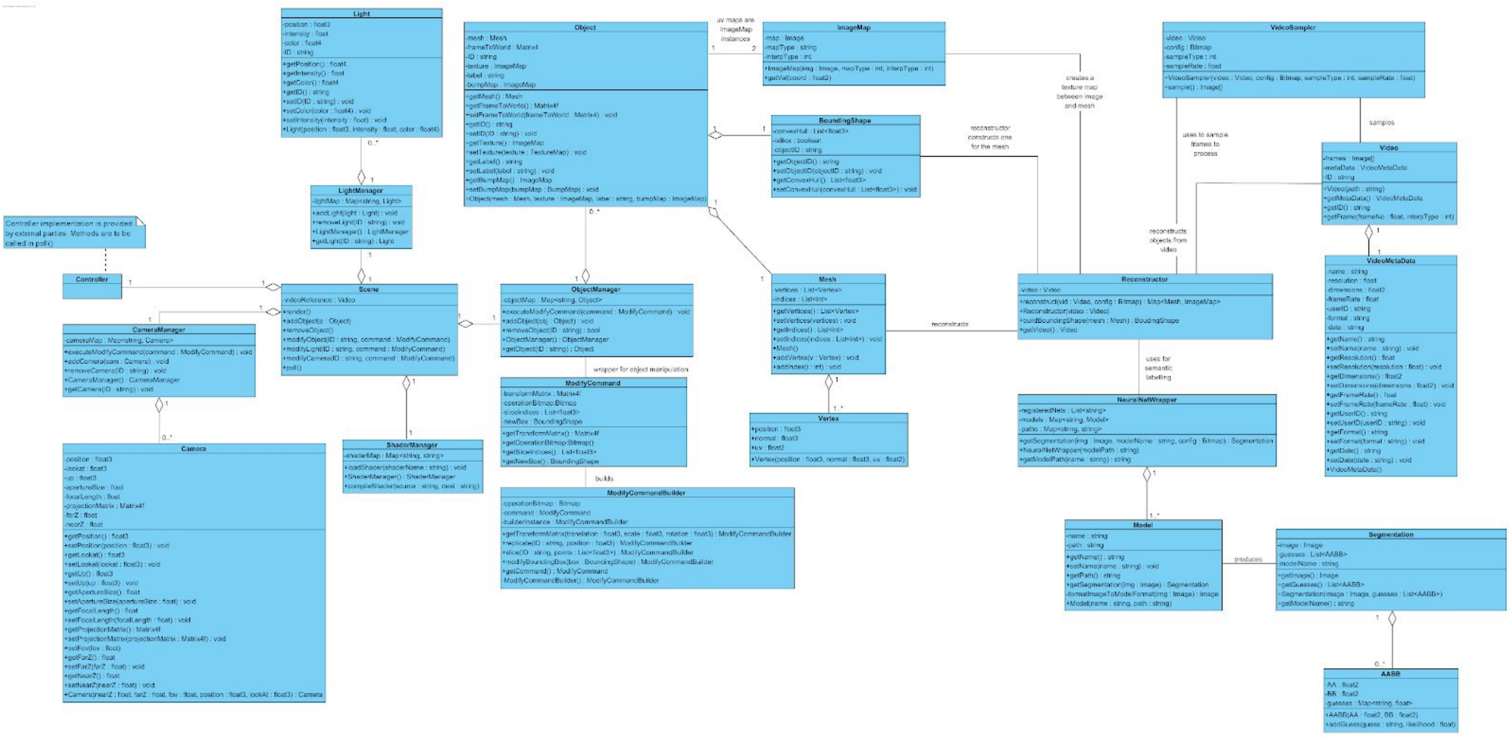
Light
-position : float3
-intensity : float
-color : float4
-ID : string
+getPosition() : float4
+getIntensity() : float
+getColor() : float4
+getID() : string
+setID(ID : string) : void
+setColor(color : float4) : void
+setIntensity(intensity : float) : void
+Light(position : float3, intensity : float, color : float4)
LightManager
-lightMap : Map<string, Light>
+addLight(light : Light) : void
+removeLight(ID : string) : void
+LightManager() : LightManager
+getLight(ID : string) : Light
Controller implementation is provided by external parties. Methods are to be called in poll()
Controller
Scene
-videoReference : Video
+render()
+addObject(o : Object)
+removeObject()
+modifyObject(ID : string, command : ModifyCommand)
+modifyLight(ID : string, command : ModifyCommand)
+modifyCamera(ID : string, command : ModifyCommand)
+poll()
CameraManager
-cameraMap : Map<string, Camera>
+executeModifyCommand(command : ModifyCommand) : void
+addCamera(cam : Camera) : void
+removeCamera(ID : string) : void
+CameraManager() : CameraManager
+getCamera(ID : string) : void
Camera
-position : float3
-lockat : float3
-up : float3
-apertureSize : float
-focalLength : float
-projectionMatrix : Matrix4f
-farZ : float
-nearZ : float
+getPosition() : float3
+setPosition(position : float3) : void
+getLockat() : float3
+setLockat(lockat : float3) : void
+getUp() : float3
+setUp(up : float3) : void
+getApertureSize() : float
+setApertureSize(apertureSize : float) : void
+getFocalLength() : float
+setFocalLength(focalLength : float) : void
+getProjectionMatrix() : Matrix4f
+setProjectionMatrix(projectionMatrix : Matrix4f) : void
+setFov(fov : float)
+getFarZ() : float
+setFarZ(farZ : float) : void
+getNearZ() : float
+setNearZ(nearZ : float) : void
+Camera(nearZ : float, farZ : float, fov : float, position : float3, lockAt : float3) : Camera
ShaderManager
-shaderMap : Map<string, string>
+loadShader(shaderName : string) : void
+ShaderManager() : ShaderManager
+compileShader(source : string, dest : string)
Object
-mesh : Mesh
-frameToWorld : Matrix4
-ID : string
-texture : ImageMap
-label : string
-bumpMap : ImageMap
+getMesh() : Mesh
+getFrameToWorld() : Matrix4f
+setFrameToWorld(frameToWorld : Matrix4) : void
+getID() : string
+setID(ID : string) : void
+getTexture() : ImageMap
+setTexture(texture : TextureMap) : void
+getLabel() : string
+setLabel(label : string) : void
+getBumpMap() : ImageMap
+setBumpMap(bumpMap : BumpMap) : void
+Object(mesh : Mesh, texture : ImageMap, label : string, bumpMap : ImageMap)
uv maps are ImageMap instances
ImageMap
-map : Image
-mapType : string
-interpType : int
+ImageMap(img : Image, mapType : int, interpType : int)
+getVal(coord : float2)
BoundingShape
-convexHull : List<float3>
-isBox : boolean
-objectID : string
+getObjectID() : string
+setObjectID(objectID : string) : void
+getConvexHull() : List<float3>
+setConvexHull(convexHull : List<float3>) : void
reconstructor constructs one for the mesh
creates a texture map between image and mesh
ObjectManager
-objectMap : Map<string, Object>
+executeModifyCommand(command : ModifyCommand) : void
+addObject(obj : Object) : void
+removeObject(ID : string) : bool
+ObjectManager() : ObjectManager
+getObject(ID : string) : Object
wrapper for object manipulation
ModifyCommand
-transformMatrix : Matrix4f
-operationBitmap:Bitmap
-sliceIndices : List<float3>
-newBox : BoundingShape
+getTransformMatrix() : Matrix4f
+getOperationBitmap:Bitmap()
+getSliceIndices() : List<float3>
+getNewBox() : BoundingShape
builds
ModifyCommandBuilder
-operationBitmap : Bitmap
-command : ModifyCommand
-builderInstance : ModifyCommandBuilder
+getTransformMatrix(translation : float3, scale : float3, rotation : float3) : ModifyCommandBuilder
+replicate(ID : string, position : float3) : ModifyCommandBuilder
+slice(ID : string, points : List<float3>) : ModifyCommandBuilder
+modifyBoundingBox(box : BoundingShape) : ModifyCommandBuilder
+getCommand() : ModifyCommand
-ModifyCommandBuilder() : ModifyCommandBuilder
Mesh
-vertices : List<Vertex>
-indices : List<int>
+getVertices() : List<Vertex>
+setVertices(vertices) : void
+getIndices() : List<int>
+setIndices(indices : List<int>) : void
+Mesh()
+addVertex(v : Vertex) : void
+addIndex(i : int) : void
Vertex
+position : float3
+normal : float3
+uv : float2
+Vertex(position : float3, normal : float3, uv : float2)
reconstructs
VideoSampler
-video : Video
-config : Bitmap
-sampleType : int
-sampleRate : float
+VideoSampler(video : Video, config : Bitmap, sampleType : int, sampleRate : float)
+sample() : Image[]
uses to sample frames to process
samples
reconstructs objects from video
Video
-frames : Image[]
-metaData : VideoMetaData
-ID : string
+Video(path : string)
+getMetaData() : VideoMetaData
+getID() : string
+getFrame(frameNo : float, interpType : int)
VideoMetaData
-name : string
-resolution : float
-dimensions : float2
-frameRate : float
-userID : string
-format : string
-date : string
+getName() : string
+setName(name : string) : void
+getResolution() : float
+setResolution(resolution : float) : void
+getDimensions() : float2
+setDimensions(dimensions : float2) : void
+getFrameRate() : float
+setFrameRate(frameRate : float) : void
+getUserID() : string
+setUserID(userID : string) : void
+getFormat() : string
+setFormat(format : string) : void
+getDate() : string
+setDate(date : string) : void
+VideoMetaData()
Reconstructor
-video : Video
+reconstruct(vid : Video, config : Bitmap) : Map<Mesh, ImageMap>
+Reconstructor(video : Video)
+buildBoundingShape(mesh : Mesh) : BoudingShape
+getVideo() : Video
uses for semantic labelling
NeuralNetWrapper
-registeredNets : List<string>
-models : Map<string, Model>
-paths : Map<string, string>
+getSegmentation(img : Image, modelName : string, config : Bitmap) : Segmentation
+NeuralNetWrapper(modelPath : string)
+getModelPath(name : string) : string
Model
-name : string
-path : string
+getName() : string
+setName(name : string) : void
+getPath() : string
+getSegmentation(img : Image) : Segmentation
-formatImageToModelFormat(img : Image) : Image
+Model(name : string, path : string)
produces
Segmentation
-image : Image
-guesses : List<AABB>
-modelName : string
+getImage() : Image
+getGuesses() : List<AABB>
+Segmentation(image : Image, guesses : List<AABB>)
+getModelName() : string
AABB
-AA : float2
-BB : float2
-guesses : Map<string, float>
+AABB(AA : float2, BB : float2)
+addGuess(guess : string, likelihood : float)

Fig.8. Revised Class Diagram

For a higher quality version, please visit https://imgur.com/a/PrCVE7i .

8. References

[1] “IKEA Place,” *Google Play'de Uygulamalar*. [Online]. Available: https://play.google.com/store/apps/details?id=com.inter_ikea.place&hl=tr. [Accessed: 02-Nov-2019].

[2] “Microsoft HoloLens: Mixed Reality Technology for Business,” *Microsoft HoloLens | Mixed Reality Technology for Business*. [Online]. Available: https://www.microsoft.com/en-us/hololens. [Accessed: 02-Nov-2019].

[3] “Sims,” *EA Games*, 14-Oct-2019. [Online]. Available: https://www.ea.com/games/the-sims. [Accessed: 02-Nov-2019].

[4]“Room Planner: Home Interior & Floorplan Design 3D - Apps on Google Play,” *Google*. [Online]. Available: https://play.google.com/store/apps/details?id=com.icandesignapp.all&hl=en_US. [Accessed: 02-Nov-2019].

[5] *Lifetimes of cryptographic hash functions*. [Online]. Available: https://valerieaurora.org/hash.html. [Accessed: 09-Nov-2019].

[6] R. Eisele, “SLA Uptime Calculator: How much downtime is 99.9%,” *SLA Uptime Calculator: How much downtime is 99.9% • Open Source is Everything*. [Online]. Available: https://www.xarg.org/tools/sla-uptime-calculator/. [Accessed: 02-Nov-2019].

9. Website

https://barisc22.github.io/MoveIt/ MoveIt website