**Smart-glass based**

**Remote Guidance System**

Software Architecture Design Document

Group 21

Lyndon Prado - 9740783

Tingcong Jimmy Li - 100017000

Keagan Foster - 101609822

Ayub Khan - 100667654

Dineth Gunawardena - 100862158

Kosala Edirisinghe - 101265981

Krishna Adhikari - 4953193

Liam Pan -101106174

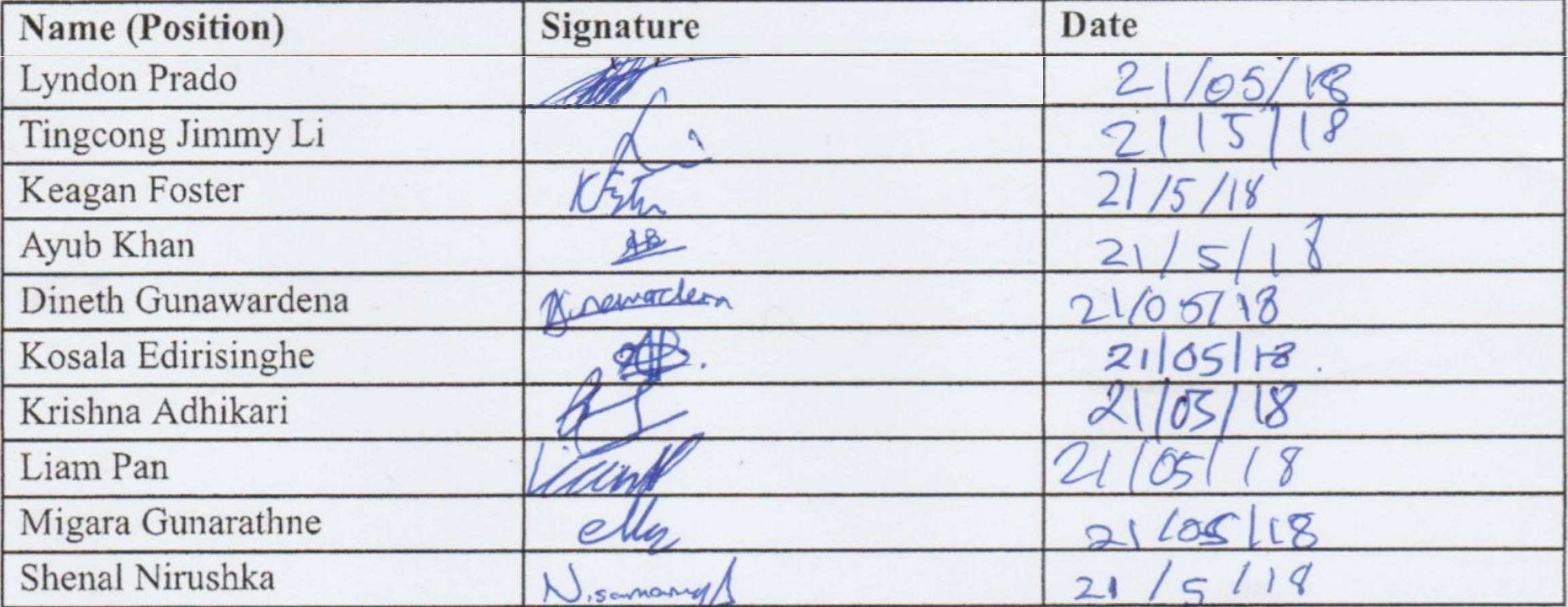
Migara Gunarathne - 101002269

Shenal Nirushka – 101054998

**Table 1. Document Change Control**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Authors** | **Summary of Changes** |
| 1.0 | 08/05/2018 | Lyndon Prado | Initial Draft  Initial Executive Summary  Problem Analysis  Initial Architecture design  CRC cards |
| 2.0 | 20/05/2018 | Tingcong Jimmy Li | Modified architecture design to higher level design  Created Deployment diagram and description  Created Component diagram and description  Created Package diagram and description  Updated CRC cards |
| 2.1 | 24/05/2018 | Lyndon Prado | Added Conclusion  Modified Executive Summary  Added Justifications  Added Patterns  Added Extensions |

**Table 2. Document Sign Off**



# Table of Contents

[Table of Contents 2](#_Toc515041686)

[1 Problem Analysis 5](#_Toc515041687)

[1.1 Goals 5](#_Toc515041688)

[1.2 Assumptions 5](#_Toc515041689)

[1.3 Simplifications 5](#_Toc515041690)

[2 Candidate Architecture 6](#_Toc515041691)

[2.1 Overview 6](#_Toc515041692)

[2.1.1 Deployment Diagram 6](#_Toc515041693)

[2.1.1.1 Justification 6](#_Toc515041694)

[2.1.2 Component Diagram 7](#_Toc515041695)

[2.1.2.1 Justification 8](#_Toc515041696)

[2.1.2.1.1 Vuzix 8](#_Toc515041697)

[2.1.2.1.2 Server 8](#_Toc515041698)

[2.1.3 Package Diagram 8](#_Toc515041699)

[2.1.4 Justification 9](#_Toc515041700)

[2.1.4.1 User Interface Package 9](#_Toc515041701)

[2.1.4.1.1 Media Input 9](#_Toc515041702)

[2.1.4.1.2 Media Interface 9](#_Toc515041703)

[2.1.4.1.3 Media output 9](#_Toc515041704)

[2.1.4.1.4 User Inputs 9](#_Toc515041705)

[2.1.4.2 Communication Package 9](#_Toc515041706)

[2.1.4.2.1 Communication Interface 10](#_Toc515041707)

[2.1.4.2.2 Input processor 10](#_Toc515041708)

[2.1.4.3 Processing package 10](#_Toc515041709)

[2.1.4.3.1 Output processor 10](#_Toc515041710)

[2.1.4.3.2 Video processing 10](#_Toc515041711)

[2.1.4.3.3 Input processing 10](#_Toc515041712)

[2.2 Component, Responsibility, Collaborator (CRC) cards 11](#_Toc515041713)

[2.2.1 Vuzix M100 Smart Glasses 11](#_Toc515041714)

[2.2.4 Server 13](#_Toc515041715)

[3 Quality 16](#_Toc515041716)

[3.1 Design Patterns 16](#_Toc515041717)

[3.1.1 Model View Controller 16](#_Toc515041718)

[3.1.2 Client - Server Architecture Pattern 17](#_Toc515041719)

[3.2 Extensions 17](#_Toc515041720)

[3.3 Conclusion 17](#_Toc515041721)

Executive Summary

The current state of remote training and guidance is not in a state that is effective in comparison to in person training. In most cases, either the person being trained must travel do a different location to be trained, or the instructor that is allocated to train the individual must go to the location of that said individual. This becomes a bigger issue when trainees are in multiple locations, making this method of training tedious and costly.

The solution to reduce the effort and cost in remote guidance, whilst still maintaining training efficiency and efficacy, is to simulate in person training, whilst maintaining the possibility for the instructor to be in a different location as the trainee. To simulate this condition, we have researched effective ways that maintain high levels of learning, and we found that having the first-person perspective of the instructor, regarding the task being completed, was an effective way in completing and learning to complete that specific task.

Our solution involves integrating a smart glass Vuzix m100 as a platform to prototype our solution, to prove that this concept is effective. A hand gesture recognition program will be written to interface with the smart glass platform, so that the instructor can send first perspective instructions to the operator/trainee. First a video live recording of the operator/trainee’s environment will be streamed to the instructor so that the instructor has the first person’s perspective of the operator. Once the Instructor has a view of what the operator is seeing, at the same time in real time, the instructor’s own smart glasses will be recording his/her own environment, but the difference here is that the instructor’s own view will be processed so that only the hands of the instructor, is extracted from the video stream. The hands that are extracted are then overlaid on top of the operators view and sent back to the operator. This is so that the operator can see what the instructor wants him/her to focus on. Both the instructor and the operator will both have a first person’s perspective view of each other’s view of their own environment.

Future extensibility has also been designed into the proposed solution so that more complex functionalities can be implemented. For example, object recognition capabilities for object selection to improve intuitiveness of the product, and, hand gesture recognition and controls, to have options controlled by hand gestures, instead of using the conventional menu items or physical buttons.

This document covers the high-level system architecture for the problem analysis, deployment diagram component diagram and package diagram; and component, responsibility collaborator cards. It does not attempt to show implementations of the candidate components.

# 1 Problem Analysis

## 1.1 Goals

The creation of a platform where and instructor can communicate to other operators/trainees remotely, without compromising the efficiency and efficacy of their work. This instructor must be able to communicate with an operator in a distant location as if it were simulating a real situation where the instructor is in one country and respective operator is in another country.

The platform must be able to maintain connection speeds and keep real time video feeds under testing in remote areas.

## 1.2 Assumptions

The following assumptions has been made to create a complete object design:

* The prototype will have the operator device and instructor device hard coded to simplify the design
* The communication between two smart glasses will use the network connection of a hot spot/server. Namely another device such as a phone or laptop, and it will have the image/video processing on that external device
* When the object recognition is implemented, no database will be used to store the definitions, but a neural network model will be pre-trained on a separate computer, to reduce the resources used on the prototype devices.

## 1.3 Simplifications

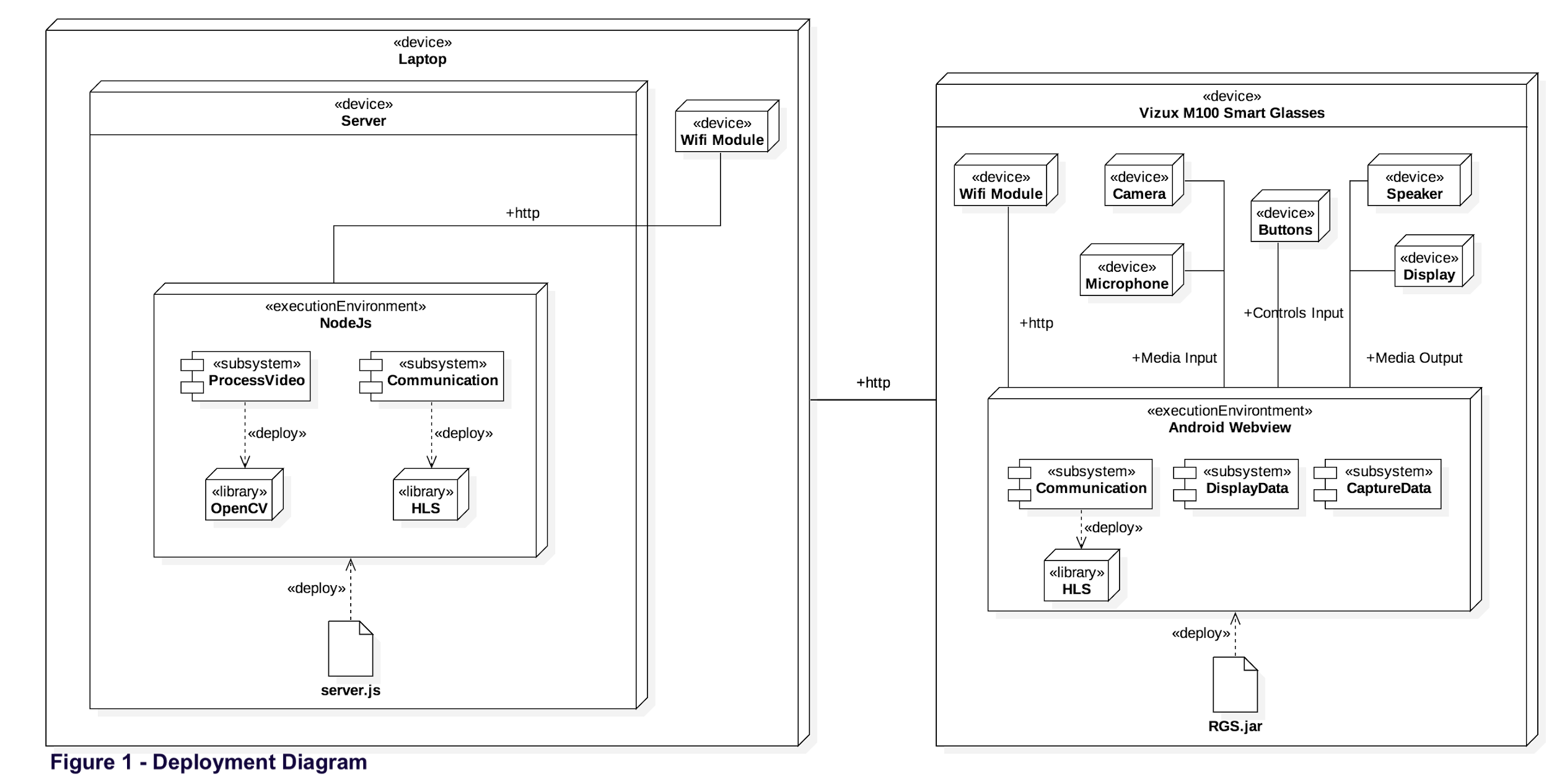
The smart glass remote guidance-based guidance system will only appear to implement the remote guidance features using video streaming to users of the system, internally it is simplified in the following ways:

* Roles are hardcoded and there is no ability to changes roles, unless resetting the whole device
* In this prototype demonstration, only two devices are used, one instructor and one operator. The prototype will not have the capabilities to connect to multiple devices (instructors will not be able to connect to multiple operators and multiple operators will not be able to choose the instructor because they will be hardcoded).
* As this is only a demonstration, security will not be looked into to a great degree. As connections will be hardcoded.
* Demonstration will happen in a closed environment i.e. having a local server to test the platform, instead of connecting through the internet and/or cloud to test the remote capabilities.
* We will be using libraries to handle the network communication between the users and the server, therefore the architecture of the network communication can be treated as a black box/subsystem.

# 2 Candidate Architecture

## 2.1 Overview

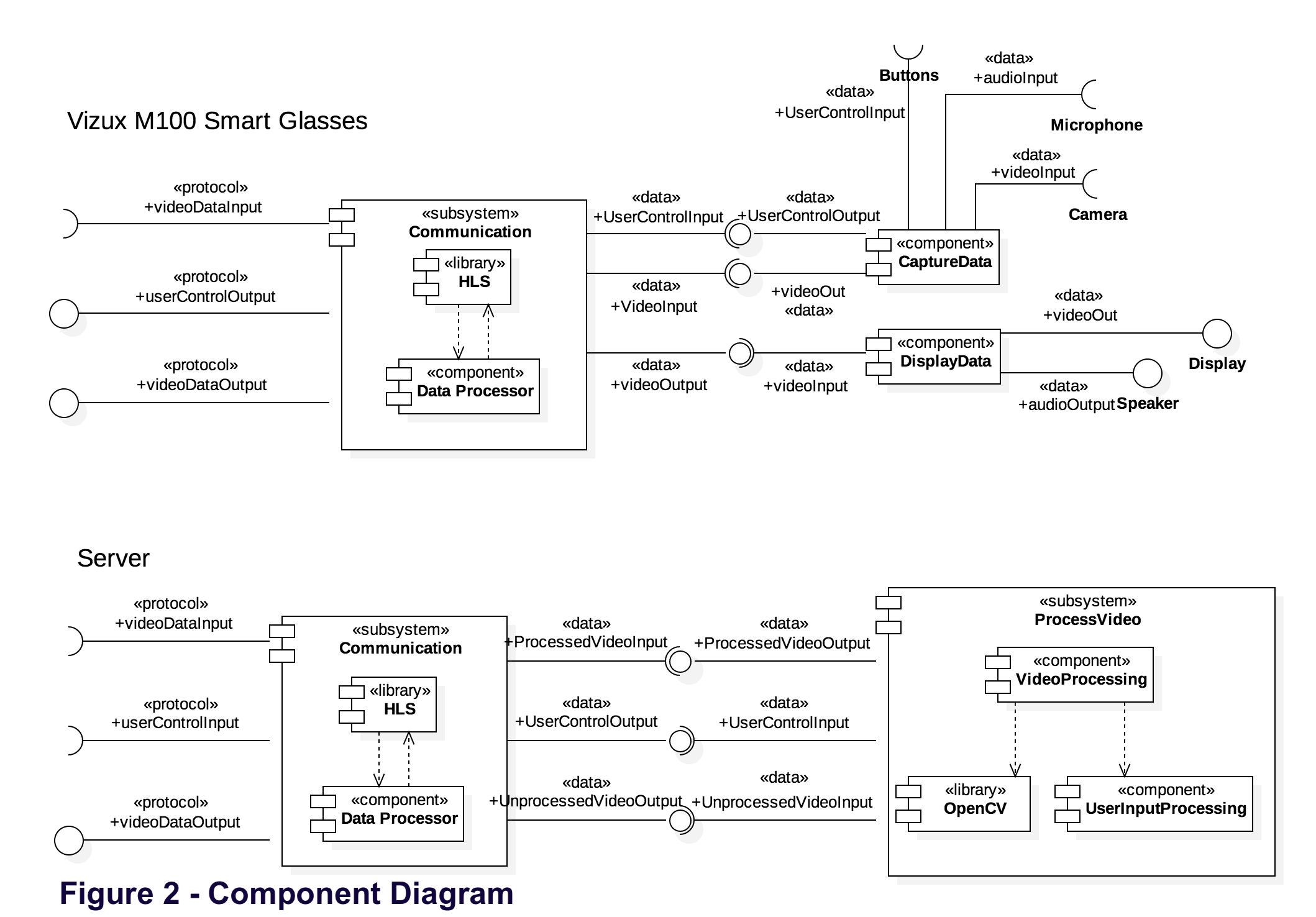
### 2.1.1 Deployment Diagram



#### 2.1.1.1 Justification

The deployment diagram shown in Figure 1, describes the dependencies and order of how the system is set up. In this case, there are two initialising files, one on the server and one on the client(s) (vuzix glasses). Server.js initializes the server on the laptop and the RGS.js initialises the system on the Vuzix glasses. The laptop is used to host the local server that will be used for the two Vuzix clients to connect to. The server is needed, so that the two glasses can communicate to each other, through the server. The server is also needed so that the overhead of the video processing is removed from the glasses and the video processing is done on the server instead (this is just one of many solutions, but this is the solution that the team has chosen to go with). Within the server device, the server.js file is used to initialise and deploy NodeJS. NodeJS is the execution environment where sub-systems that handle the video processing and the network communications, are set up. ProcessVideo subsystem then deploys and uses OpenCV to process the video streaming. The communication subsystem within the execution environment is used to handle the network communication between the clients and the server, which is done by using HLS (Http Live Streaming). HLS is used for cross device communication. The communication between the vuzix glasses and the server is facilitated by their respective Wi-Fi modules. On the client end, the Android environment on the vuzix is deployed with the RGS.js artifact (remote guidance system.js). The execution environment contains three subsystems for handling communication, data capture and data display. The communication subsystem of the android WebView also deploys HLS for cross device communication. CaptureData subsystem communicates with device buttons, device camera and device microphone. DisplayData subsystem communicates with device display and device speaker.

### 2.1.2 Component Diagram



#### 2.1.2.1 Justification

The component diagram shown in Figure 2, describes at a high level the interactions between components and subsystems. Interactions between the Vuzix Smart Glasses and the Server are represented by their respective Communication subsystems. Data captured by the Vuzix are output to the Server which in turn outputs processed data to the device.

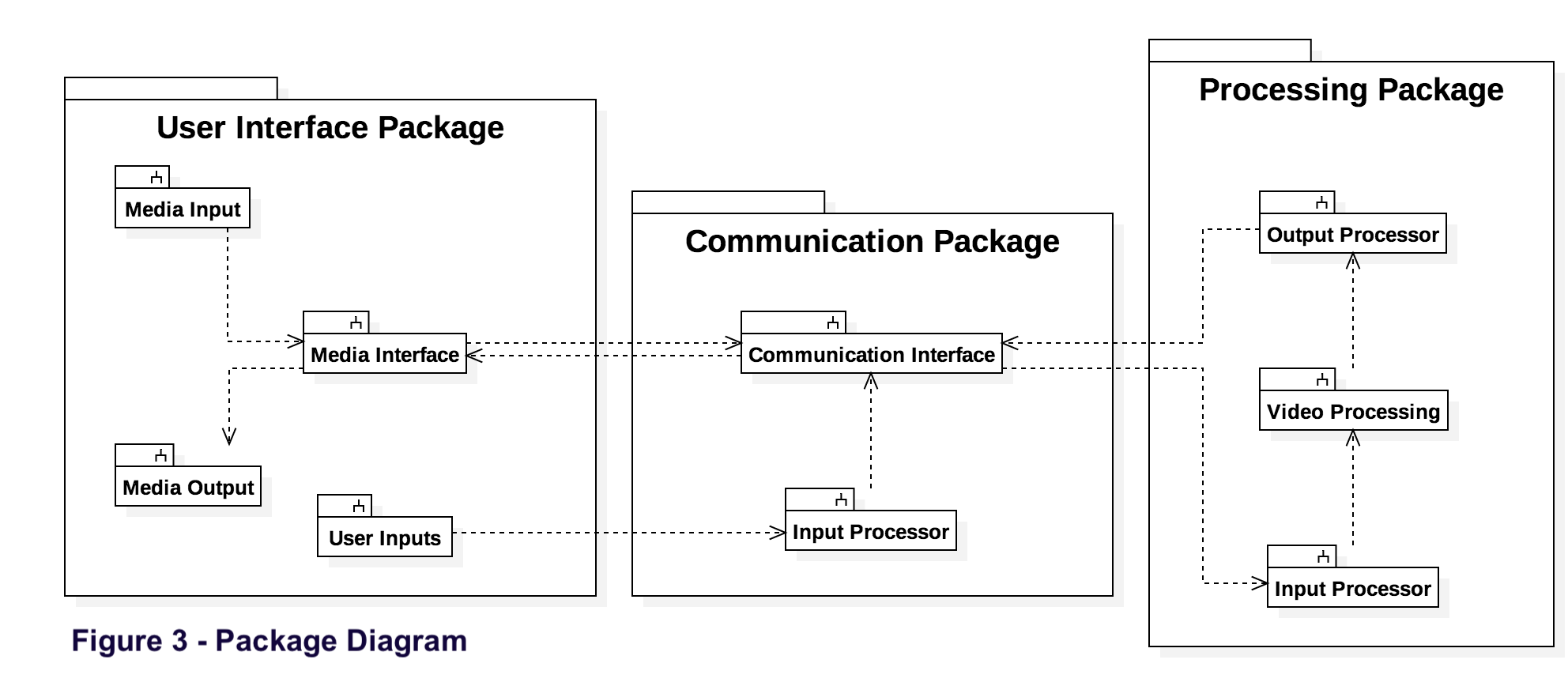
##### 2.1.2.1.1 Vuzix

The components within the communication subsystem on the Vuzix Smart Glass requires captured inputs from the Vuzix microphone and camera to provide output to the Server. The microphone and camera data are processed by the CaptureData component. The communication subsystem provides output for the Vuzix display and speaker through the DisplayData component, which the communication subsystem receives from the server. A HLS library is contained within the communication subsystem to enable cross device communication to and from the server.

##### 2.1.2.1.2 Server

Components that enables communication to and from the Server to Vuzix Smart Glass is contained within the Communication subsystem, its inputs and outputs are coupled with the Communication subsystem on the Vuzix Smart Glasses. Input data to the server are processed by the ProcessVideo subsystem, based on the required specific of the user inputs to provide processed output to the Vuzix Smart Glasses for display. This communication subsystem also contains the HLS library to enable cross device communication to and from the Server to the smart glasses.

### 2.1.3 Package Diagram



### 2.1.4 Justification

#### 2.1.4.1 User Interface Package

This package contains all subsystems that the user(s) can directly interact with. It contains two subsystems that handles the input and output of visual data, and a subsystem for handling user inputs.

##### 2.1.4.1.1 Media Input

This package contains the subsystems that handle the interfacing of hardware microphone and the camera. Having a separate module for the media input allows to simplify troubleshooting, such that if either the microphone or the camera were to fail, then their functionality won’t affect the system (it may affect the usability of the system). This subsystem outputs raw media inputs to the Media Interface subsystem.

##### 2.1.4.1.2 Media Interface

Media Interface contains subsystems that handles the incoming raw data from the microphone and the camera and outputs encoded data to the communication interface. This subsystem also handles the incoming data transmission from the communication interface, the data then output to the Media Output interface. Having a module that focuses on handling the data transmissions between incoming and outgoing data, allows for the prioritization of data, ultimately using the bandwidth of the data transmission more efficiently. It also allows for asynchronous transmission, because the media interface can run on a separate thread, therefore increasing transmission speeds.

##### 2.1.4.1.3 Media output

Once data is received from the communication interface, the respective visual data will be represented on the display and the respective audio data will be reproduced on the Vuzix speaker system. This package will contain subsystems to handle this. Having a separate module for the media output allows to simplify troubleshooting, such that if either the speaker of the screen was to fail, then their functionality won’t affect the system as a whole (it may affect the usability of the system).

##### 2.1.4.1.4 User Inputs

This package subsystems will deal with the physical button functionality of the Vuzix glasses. It will convert the raw button press signals to a format to be transmitted to the communication interface. It will be sent on a separate channel to that of the audio and video streaming channel, so that the controls will still be responsive, even though the streaming freezes (if the streaming does freeze)

#### 2.1.4.2 Communication Package

The communication interface package will handle all the network communications between the glasses and the server. It will prioritize the most important data to be sent first to optimize the speed of transmission, and efficiently uses bandwidth. Since the system will be Client to Server to Client, the communication interface will handle the communication between the clients and the servers so that the intended clients is talking to the intended server and vice versa. It also ensures that enough of bandwidth is shared between the ingoing and outgoing data being transfers to and from clients via the server.

##### 2.1.4.2.1 Communication Interface

This sub-package of the communication interface handles the audio and visual data transmission between the two glasses.

##### 2.1.4.2.2 Input processor

This sub-package of the communication interface handles the button presses to modify settings, this is on a separate channel to that of the sub-package communication Interface, so that it is still responsive if the sub-package communication interface goes down.

#### 2.1.4.3 Processing package

This sub-package handles the image and video processing of the video streaming. It will also handle the hand gesture recognition, as the hand gesture recognition needs the video streaming to work. It will also handle the snapshot and Sketching with the hand gestures, and in future implementation, it will also handle object recognition.

##### 2.1.4.3.1 Output processor

This sub-package handles the video transmission to the communication interface after the video stream has been processed

##### 2.1.4.3.2 Video processing

This sub-package handles the video processing and the image processing (when snapshot is taken for sketching)

##### 2.1.4.3.3 Input processing

This sub-package handling the incoming raw video stream from the clients.

## 2.2 Component, Responsibility, Collaborator (CRC) cards

### 2.2.1 Vuzix M100 Smart Glasses

**Table 3. HLS Library CRC Card**

|  |  |
| --- | --- |
| **Component:** HLS <<library>> | |
| **Sub-Components:** n/a | **Parent-Components:**  Communication Subsystem |
| **Responsibilities** | **Collaborators** |
| Transmits encoded video data |  |

#### 

**Table 4. Data Processor Component CRC Card**

|  |  |
| --- | --- |
| **Component:** Data Processor <<component>> | |
| **Sub-Components:** n/a | **Parent-Components:**  Communication Subsystem |
| **Responsibilities** | **Collaborators** |
| **Read encoded video data**  **Description:**  Takes encoded input video data using HLS library | HLS Library |
| **Decode Video**  **Description:**  Decode video and outputs to DisplayData component | DisplayData Component |
| **Encode video**  **Description:**  Takes in raw video data from the CaptureData component and encode the data | CaptureData component |
| **Stream encoded data**  **Description:**  Outputs encoded data using HLS Library | HLS Library |
| **Send user Input**  **Description:**  Takes in user input data from CaptureData component and outputs using HLS Library | HLS Library |

### 

**Table 5. Capture Data Component CRC Card**

|  |  |
| --- | --- |
| **Component:** CaptureData <<component>> | |
| **Sub-Components:** n/a | **Parent-Components:** n/a |
| **Responsibilities** | **Collaborators** |
| **Record audio**  **Description:** Captures data from the microphone | Device Microphone |
| **Record video**  **Description:**  Captures data from the camera | Device Camera |
| **Send raw video**  **Description:**  Outputs raw video data to the Data Processor component | Data Processor Component |

### 

**Table 6. Display Data Component CRC Card**

|  |  |
| --- | --- |
| **Component:** DisplayData <<component>> | |
| **Sub-Components:** n/a | **Parent-Components:** n/a |
| **Responsibilities** | **Collaborators** |
| **Read Decoded Data**  **Description:**  Takes in decoded data from Data Processor component | Data Processor Component |
| **Display video**  **Description:**  Process decoded data and outputs video data to the device display | Device display |
| **Play Audio:**  **Description:**  Process decoded data and outputs audio data to the device speaker | Device speaker |

### 2.2.4 Server

**Table 7. HSL Library CRC Card**

|  |  |
| --- | --- |
| **Component:** HLS <<library>> | |
| **Sub-Components:** n/a | **Parent-Components:**  Communication Subsystem |
| **Responsibilities** | **Collaborators** |
| Transmits encoded video data | Data Processor Component |

**Table 8. Data Processor Component CRC Card**

|  |  |
| --- | --- |
| **Component:** Data Processor <<component>> | |
| **Sub-Components:** n/a | **Parent-Components:**  Communication Subsystem |
| **Responsibilities** | **Collaborators** |
| **Read encoded input data:**  **Description:**  Takes in encoded input data using HLS library | HLS Library |
| **Decode and output video data:**  **Description:**  Decode video data and outputs to VideoProcessing component | VideoProcessing Component |
| **Output decoded data:**  **Description:**  Decode user input data using HLS library and output to UserInputProcessing Component | HLS Library, UserInputProcessing component |
| **Retrieve raw data and transmit:**  **Description:**  Takes in and encode raw data from VideoProcessing component and output using HLS library | HLS Library, VideoProcessing component |

**Table 9. Video Processing Component CRC Card**

|  |  |
| --- | --- |
| **Component:** VideoProcessing <<component>> | |
| **Sub-Components:** n/a | **Parent-Components:**  Communication |
| **Responsibilities** | **Collaborators** |
| **process Video**  **Description:**  Takes unprocessed video and user input data from DataProcessor component and process video using OpenCV library | DataProcessor component, UserInputProcessing component, OpenCV library |
| **Send processed Video**  **Description:**  Outputs processed video to the DataProcessor component | DataProcessor component |

**Table 10. OpenCV Library CRC Card**

|  |  |
| --- | --- |
| **Component:**openCV<<library>> | |
| **Sub-Components:** n/a | **Parent-Components:**  Communication |
| **Responsibilities** | **Collaborators** |
| **process video**  **Description:**  Process video data base requirements of VideoProcessing component | VideoProcessing component |
| **Output processed video**  **Description:**  Outputs processed video data to VideoProcessing component | VideoProcessing component |

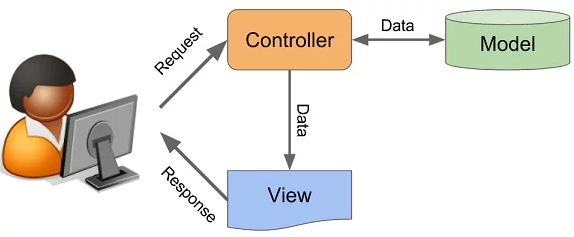
**Table 11. User Input Processing Component CRC Card**

|  |  |
| --- | --- |
| **Component:** UserInputProcessing<<component>> | |
| **Sub-Components:** n/a | **Parent-Components:**  Communication |
| **Responsibilities** | **Collaborators** |
| **Update user settings**  **Description:**  Process user input data from the DataProcessor component then output user input(requirments) to VideoProcessing component | Data Processor component, VideoProcessing component |

# 3 Quality

## 3.1 Design Patterns

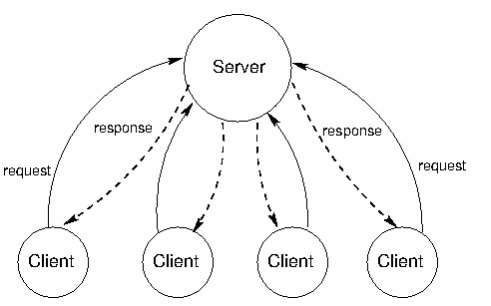
### 3.1.1 Model View Controller



**Figure 4 - MVC Model**

We are using the Model View Controller pattern so that loose coupling and high cohesion can be adhered to. The model is on the server as it will be where the data is processed. Conventionally the controller and the model is supposed to be on the glasses/clients, but in our situation, the overhead of the processing will slow down the clients, because they do not have that much processing power in comparison to the server. So instead the requests for the controller will be streamed from the glasses and the view will be updated on the server but it will be streamed back to the corresponding glasses.

### 3.1.2 Client - Server Architecture Pattern



**Figure 5 - Client Server Architecture**

The client - server architecture pattern is the basis of our architecture, in that the clients are the Vuzix glasses. This allows the centralisation of communication to the server and it allows the possibility for the remote communication of the glasses.

## 3.2 Extensions

As the vuzix glasses will be hard coded to have the role of the operator or the instructor, in future development, there will be added the functionality of switching roles. There will be added object recognition using machine learning algorithms, which will be used in conjunction with the hand gestures and the sketching. The object recognition algorithm model will be pretrained on a computer and then the model will be stored on the server, to reduce the overhead of the system. Meat quality processing, such as detecting the colour of the meat, the size of the meat and detecting its composition will be added in future iterations.

## 3.3 Conclusion

As this is only a high-level overview of the system architecture, the lower level architecture still needs to be developed, and that will be done in a software design document to be submitted for the next stage of the project. We have been able to show the environment where the system will be deployed and also showed the high-level component diagram of how the main modules will be communicating to each other.