Software Formalization

Year: 2022 Semester: 2 Team: 2 Project: VRm

Creation Date: 2/15/2022 Last Modified: 2/16/2022

Author: Matthew Wen Email: wen101@purdue.edu

Assignment Evaluation:

| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| --- | --- | --- | --- | --- |
| **Assignment-Specific Items** | | | | |
| **Third Party Software** |  | x2 |  |  |
| **Description of Components** |  | X3 |  |  |
| **Testing Plan** |  | x3 |  |  |
| **Software Component Diagram** |  | x4 |  |  |
| **Writing-Specific Items** | | | | |
| **Spelling and Grammar** |  | x2 |  |  |
| **Formatting and Citations** |  | x1 |  |  |
| **Figures and Graphs** |  | x2 |  |  |
| **Technical Writing Style** |  | x3 |  |  |
| **Total Score** |  | | |  |

5: Excellent 4: Good 3: Acceptable 2: Poor 1: Very Poor 0: Not attempted

General Comments:

1.0 Utilization of Third Party Software

In this section, we will be talking about some of the third party libraries that we used for our project. We use third party libraries to facilitate tasks that will be hard to be done if built from scratch. In addition, some of these libraries help prevent errors like Docker Containers [6], or some are just required to use a certain network protocol, like dgram [10] for udp connections. Here is a breakdown of the third party libraries we used.

**1.1 Third Party Software on Unity**

Unity is a game engine software that is meant for Software Developers to develop Apps and Games for multiple Virtual Reality Headsets. We intend to use additional third party software that makes the experience more engaging, however, in terms of third party software that is meant to get the fundamental functionality working, here are the unity third party packages we are planning on using.

| **Name** | **Functionality** |
| --- | --- |
| IK Constructor | This third party software [1] is used in the Unity simulation. We used this package because, while 2D inverse kinematics is simple enough, kinematics with multiple targets in a 3D space is much harder. In order to save time, we decided to use this software. IK Constructor allows us to easily track multiple targets (Hand and Elbow targets) in a 3D space with a simulated robotic arm. |
| FMETP Stream | This third party software [2] is also used in the unity simulation. This software package will help with converting the transmitted video data from the server to the Unity simulation to usable video data the user can view. This is important because we are trying to replicate 3D video with two cameras, so software to help with the more complicated parts of working with video is appreciated. |

**1.3 Third Party Software on the Server**

The server utilizes Google Cloud Virtual Machine Instances. The reason why we utilized a cloud service like Google Cloud is so that Google Cloud can provision a stable internet connection, and seamless upgrades if we need (like additional RAM or faster connection) than setting up a server at our apartment. For Google Cloud cost, since we want to do most of the setup process rather than use one of Google Cloud third party services like Cloud Functions etc, it costs around $2 dollars a month, which is a lot cheaper than setting up our own server. Here is the software we used on Google Server. Note that all software listed before is open source.

| **Name** | **Functionality** |
| --- | --- |
| Debian GNU/Linux 10 (buster) [3] | This is the default Operating System Running on the Google VM Instance. |
| NodeJs [4] | NodeJs is a Just in Time (JIT) Compiler that transcribes Javascript Code into machine code that can run on a VM Instance. We are using this programming language because it is simple to use and learn compared to other options like C++, and its JIT runtime speed is faster than some interpreter languages like Python. Lastly, it has more third party support than programming languages like GoLang. |
| Docker [5] | Docker is an Image Builder of our application. We are using Docker Image of NodeJs [6] to serve our node application. This third party is useful because it separates the application from the VM specific resources, so we don't have to worry about the program breaking our VM instance. In addition, it makes it easier to deploy our application with all the required libraries to run it. |
| Nginx [7] | We are using nginx because it port forwards our application to different url links. So for our server, its url address is <https://ecess-api.matthewwen.com>, but with nginx, we can map localhost:2001 to <https://ecess-api.com.matthewwen.com/vrms/>. We also use certbot [20] to ensure encryption with https instead of http, and the domain name of matthewwen.com comes from google domains [21]. |
| ExpressJs [8] , and ExpressJsWs [9] | ExpressJs is a third party library that creates a layout on serving an http web server. ExpressJsWs [9] is a wrapper on top of ExpressJs [8] application where it adds websocket support. We are using websockets to ensure a strong connection between the unity headset to the server. |
| dgram [10] | This is a third party library supported by NodeJs themselves. It allows for UDP connections between the server and any sort of endpoints. The current plan is send data packets of video streaming data from the pi to the server, then use Real Time Streaming Protocol (RTSP) from the server to the unity headset. |
| zeromq [12] | This is a NodeJS library that utilizes the protocol from ZeroMq [12]. The purpose of this library is to send robot arm joints from the server to the Pi. |
| firebase-admin [13] | This is a third party library we will temporarily use depending on if unity needs a database. We used it to show that the server can receive arm data. |
| node-cron [14] | This third party library is currently not being used, but it is an application where it can trigger a script to run at a certain frequency or time. For example, one can trigger to run Nodejs code every 12 hours, or at 12 am every day. |

**1.4 Third Party Software on the Pi**

The Raspberry Pi is a development computer that has accessible pins where it can communicate to other different devices like our microcontroller. We utilize some libraries that can help process data from the server, as well as libraries that help parse the data from the server and transcribe it and send it to other devices connected to the pi locally, whether that is a microcontroller, buzzer, etc. Here is the software we used in the table below, Note that all software listed before is open source.

| **Name** | **Functionality** |
| --- | --- |
| Raspbian [15] | The official operating system from Raspbian themselves. We used the lite version because we don't need a desktop. |
| python3.9 and socket [16] | The programming language and interpreter we used to run python code. In python, we are using sockets themselves to send packets via UDP. In terms of python as our language, it has multiple third party support, and it is simple to use. |
| pyserial [17] | A serial connection library to send serial data to other hardware devices, like our arm. |
| pyzmq [18] | It utilizes the protocols from ZeroMq [11]. It basically listens to data from the server by analyzing the small packets. The small packets in this case contain arm data from the server. |
| ArduCamera Drivers [19] | These are the drivers needed to be able to read the scene data from the camera. |
| OpenCv [22] | Read Camera Data from ArduCamera |

2.0 Description of Software Components

**2.1 Unity Simulation**

| **Functionality** | **Description** |
| --- | --- |
| Calculating Physical Arm Positions | The Unity simulation is where the arm joints rotational data is calculated, and then sent to the server. We are importing software (1.1) to handle the inverse kinematics required to calculate the rotational data needed for each joint to reach its target. We are importing software (1.2) to handle the video streaming coming from the server and converting it to 3D video. Everything else, like menus, controls, and functions will be written by us, using the default assets Unity provides. |
| Sending Joint Data to the Server via TCP and WebSockets | This will be a WebSocket class that will connect to the server, and then send Joint data from the VR headset to the server. |
| Receiving Video Streaming Data from the Server via RTSP and UDP | This will use some sort of wrapper where it can retrieve video streaming data from the server using the RTSP protocol. |

**2.2 Server**

| **Functionality** | **Description** |
| --- | --- |
| Receive Joint Data from Unity to the Pi via Websockets | Code: <https://github.com/477-vrms/vrms-api/blob/main/src/ws/joint.ts>  Function:async (ws: ws, req: Request, res: Response)   * *ws* is the Websocket. *req* is the incoming traffic, and *res* is where we write bytes back to the device asking for the request. * In this case, *req* is coming from the unity headset.   Logical Flow   * Parse the Authentication to ensure that the request is coming from the unity headset. Close the connection if the request is invalid. * If the request is valid, create a UDP socket for Real Time Streaming Protocol. It will be used on the VR headset. * If the request is valid, create a websocket connection with the devices sending the request (VR headset). On the Unity side, it will acknowledge that the websocket was created successfully, and it will start sending JSON packets which represent the arm data.   + For each packet it receives, it will do the following     - Upload the changes to Firebase Real Time Database [13] for Debugging     - Publish the packet to vrms\_pi topic on ZeroMQ Node JS [12] so then the Pi can receive the packet. |
| Send Joint Data from Unity to the Pi via ZeroMq | Code: <https://github.com/477-vrms/vrms-api/blob/main/src/zmq/index.ts>  Class: ZmqHandler  Class Variables:   * sender: Sends Messages to the Raspberry Pi via ZeroMq [11] * receiver: Listens to Messages from the Pi via ZeroMq [11]   Class Methods:   * send(topic, obj)   + if the topic equals "vrms\_pi", it will send the object to the raspberry pi via ZeroMq [11]. * list(port, callback)   + It will listen for any ZeroMq [11] response from port number being passed through. |
| Receive Video Data from the Pi via UDP, then Sending Video Streaming Data from the Server to Unity via RTSP and UDP | Class: WsUdp (uses dgram [10])  Class Variables:   * piSocket: Port listening for UDP connections from the raspberry pi * vrSocket: Port sending UDP connections via RTSP to the Vr Headset   Class Methods:   * listen()   + Listen to packets via UDP from the Raspberry PI. When it retrieves a packet, it will trigger send() method * send()   + Checks if there is a socket connection to the unity headset. If there is, it will send the packet from the PI to the VR headset using RTSP. |

**2.3 Raspberry Pi**

| **Functionality** | **Description** |
| --- | --- |
| Receive Joint Data via ZeroMQ TCP protocol | Code: <https://github.com/477-vrms/vrms-pi/blob/main/vrms/network/mqtt.py>  Class: Mqtt (uses ZeroMq Python [18])  Class Methods:   * def \_\_init\_\_(self, arm\_handler, wifi\_state):   + Constructor Method for Raspberry Pi. It creates a connection to the server via its Google Cloud IP Address and Port used for MQTT. In addition, subscribe to "vrms\_pi" to listen for any packets from the server.   + arm\_handler is a class to process any requests it receives from the server.   + wifi\_state is an instance of the WifiState class that detects any sort of issues such as MQTT error. If there is a MQTT error, it will trigger code inside this class * def client(self):   + A Polling Class that runs into an infinite loop. It checks for any messages from the server, then processes the data from the server.   + Will stop / reload if wifi\_state realizes there is an error. |
| Send Video Data via UDP to the server | Class: Udp  Class Methods:   * def \_\_init\_\_(self, wifi\_state):   + Constructor Method for Raspberry Pi. It creates a connection to the server via its Google Cloud IP Address and Port used for UDP.   + It creates a connection to the Camera. An instance of the class is created when the VR headset starts getting joint packets from the server.   + wifi\_state is an instance of the WifiState class that detects any sort of issues such as UDP error. If there is a UDP error, it will trigger code inside this class * def send(self):   + Connects to Camera, creates a packet for Pi to Process, and sends the data via UDP to Server.   + Will stop / reload if wifi\_state realizes there is an error. |
| Serial Connection from the Pi to the Microcontroller | Class: ArmHandler  Class Methods:   * def \_\_init\_\_(self)   + Queue of All Incoming Traffic so it can run some of them concurrently * def add\_json(self, item):   + Add joint movement to the arm to later process * def client(self):   + A polling method where if there is an arm data to process, run the arm task concurrently in a new thread. * def send(self, packet):   + Send the content in a variable packet to UART Device. It will check the current positions if the changes seem sustainable enough where the arm needs to move or not. * def uart\_listen(self):   + Listen for packets from the UART device. The Uart Device will retrieve the current motor positions. |
| Detecting a Failure to Connect to Server, and the ability to reconnect when failed. | Class: WifiState  Class Method:   * def \_\_init\_\_(self):   + Empty Constructor Method * def on\_failure(self, type):   + Will trigger a buzzer via PWM to start. It will keep trying to connect via polling until all the components can reconnect. |
| Putting it All Together:  Main Code:  b = Background()  b.listen() | Class:Background  Class Method:   * def \_\_init\_\_(self):   + Create All Sub Processes for Uart, Mqtt, and UDP to run at the same time. * def listen(self):   + Create All New Sub Processes with each infinite loop running. * def close(self):   + Close All Sub Processes because of any sort of interrupt. |

**2.4 Microcontroller**

| **Functionality** | **Description** |
| --- | --- |
| Setup Timer 1 | Initialize Tim1 to output at PWM signal on PA8 |
| Send Data to Servo Driver | Send data over the UART connection to the servo driver |
| Receive Data from Servo Driver | Receive data over the UART connection from the microcontroller |
| Send Data to Raspberry-Pi | Send data over the UART connection to the Raspberry-Pi |
| Receive Data from Raspberry-Pi | Receive data over the UART connection from the Raspberry-Pi |
| Setup UART 1 | Setup a UART connection with a baud rate of 9600 bps to communicate with the servo driver |
| Setup UART 2 | Setup a UART connection with a baud rate of 96000 bps to communicate with the servo driver. |
| Fill Servo Buffer | Fill the data structure that will be sent to the servo driver. |
| Fill Buffer for Raspberry-Pi | Fill the data structure that will be sent to the servo driver. |
| Insert Value into FIFO | Append another data value to the end of the FIFO. If the FIFO is already full, drop the character. Keep in mind, FIFO refers to a queue, not a stack. |
| Remove Value from FIFO | Return the head of the FIFO and delete the value from the FIFO. If it is, return -1. |
| Main Function | Return data from pi, translate to servo data, fill buffer, and send off to servo-driver. |

3.0 Testing Plan

**3.1 Unity Simulation + Server to Unity + Unity to Server**

| **Functionality** | **How to Test** |
| --- | --- |
| Calculating Physical Arm Positions | Testing the unity simulation requires looking into the server and the robot arm. To test the simulation itself, we just have to look at the simulation to see if it is working as intended. (by using the VR headset) To check if the simulation works with the server, we will check the server to make sure that the simulation is properly communicating the correct values shown in the simulation. The robot arm would need to be compared to the simulation to see if the joint rotational data is properly being sent to the robot arm. If not, the arm would be out of sync with the simulation causing precision problems and would require tuning of either the simulation or the microcontroller controlling the robot arm. |
| Sending Joint Data to the Server via TCP and WebSockets | 1. We first create a simple websocket from the server side. It didn't do anything but create a connection if it recognized a connection. We ensure that there are no errors on the unity side. 2. We then upgrade the server code to see the message from Unity. If we can see it though print statements on the server, we are in business. 3. Added Authentication through the initial packet header (Packet Required to create a Websocket). We checked with the correct, and incorrect authentication. Authentication is simple; we just set the header to include "Authorization: Bearer (some password)", and we just check on the server if it matches 4. Connect it to Google Realtime Database for easy update. 5. Have the VR headset move, and see how fast the changes update to Firebase. |
| Receiving Video Streaming Data from the Server via RTSP and UDP | 1. Learn about RTSP, and see if we can get it connected to any RTSP examples. 2. Have the server send some similar RTSP packets to unity to create a connection between the VR headset and the server 3. Change it up where RTSP can process data from the camera. |

**3.2 Server to Raspberry Pi + Raspberry Pi to Server**

| **Functionality** | **How to Test** |
| --- | --- |
| Send Joint Data from Unity to the Pi via ZeroMq | 1. Have the local computer act as a server and pi, and test it locally where python code runs on the pi, and node js code on the server. 2. Deploy the server code onto the server. See if Macbook can instead connect to the server than local connection from one port to another port. 3. Transfer the code over to the Raspberry Pi. 4. Now trigger the server to send data to the Raspberry Pi when it gets a websocket packet from the VR headset (the same location where it upload the arm changes to firebase) 5. Use a phone camera to show when a particular packet was sent from unity and when it arrived to the pi by using print statements from the Pi when it retrieves a packet. |
| Receive Video Data from the Pi via UDP | 1. Do the same thing where we have the Raspberry Pi code running on one port, and the server code running on another port locally on a personal computer. 2. Deploy some server code to accept connections via UDP to a certain port. 3. Have the local computer send packets via UDP to the server. 4. Put the code onto the Raspberry Pi. Check if it is getting the packets in the correct format. 5. Have the Raspberry Pi now put Camera Data and additional timestamp for each UDP request. The server will save it locally in its file manager. |

**3.4 Microcontroller + Microcontroller to Raspberry Pi + Raspberry Pi to Micocontroller**

| **Functionality** | **How to Test** |
| --- | --- |
| Setup Timer 1 | Check that all the bits are set correctly in the debug menu, check that the PWM output on PA8 has the correct period, duty cycle, and Vpp. |
| Send Data to Servo Driver | Check that the value received by the robot arm is the correct value by evaluating the angle to which the motor moves. |
| Receive Data from Servo Driver | Check the data received from the Servo Driver is in the correct format and has the correct values. |
| Send Data to Raspberry-Pi | Check the data received by the Raspberry-Pi is in the correct format and has the correct values. |
| Receive Data from Raspberry-Pi | Check the data received from the Raspberry-Pi is in the correct format and has the correct values. |
| Setup UART 2 | Check that all of the bits are set correctly in debug mode and whether the baud rate is 9600 bps. Also check that data is being received in the correct format and content using PA2 and PA3. |
| Setup UART 3 | Check that all of the bits are set correctly in debug mode and whether the baud rate is 9600 bps. Also check that data is being received in the correct format and content using PC10 and PC11. |
| Fill Servo Buffer | Check that the buffer to be sent to the servo driver is being filled in the correct format and with the correct values. |
| Fill Buffer for Raspberry-Pi | Check that the buffer to be sent to the servo driver is being filled in the correct format and with the correct values. |
| Insert Value into FIFO | Check that when a value is added it is added to the tail. Check to make sure if the FIFO is already full, the character is dropped. |
| Remove Value from FIFO | Check that when a value is removed from the head it is removed and returned to the calling function. Check to make sure if the FIFO is empty that -1 is returned. |
| Main Function | Check that the data flow from the Raspberry-Pi to the FIFO in the micro to the servo driver is intact. This can be checked by measuring the angle of the robot arm and comparing that to the angular data sent to the micro by the Pi. |

4.0 Sources Cited:

[1] “IK Constructor: Animation Tools.” *Unity Asset Store*, <https://assetstore.unity.com/packages/tools/animation/ik-constructor-106482>.

[2]“FMETP Stream v2: Packs.” *Unity Asset Store*, <https://assetstore.unity.com/packages/templates/packs/fmetp-stream-v2-202537>.

[3] *Debian*, <https://www.debian.org/>.

[4] Node.js. *Node.js*, <https://nodejs.org/en/>.

[5] “Empowering App Development for Developers.” *Docker*, <https://www.docker.com/>.

[6] *Docker Hub*, <https://hub.docker.com/_/node>.

[7] “Advanced Load Balancer, Web Server, & Reverse Proxy.” *NGINX*, 15 Feb. 2022, <https://www.nginx.com/>.

[8]“Node.js Web Application Framework.” *Express*, <https://expressjs.com/>.

[9] HenningM. “HENNINGM/Express-WS: Websocket Endpoints for Express Applications.” *GitHub*, <https://github.com/HenningM/express-ws>.

[10] “Node.js V17.5.0 Documentation.” *UDP/Datagram Sockets | Node.js v17.5.0 Documentation*, <https://nodejs.org/api/dgram.html>.

[11] *ZeroMQ*, <https://zeromq.org/>.

[12] Zeromq. “Zeromq/Zeromq.js: Node.js Bindings to the ØMQ Library.” *GitHub*, <https://github.com/zeromq/zeromq.js/>.

[13] “Firebase Documentation.” *Google*, Google, <https://firebase.google.com/docs/admin>.

[14] “Node-Cron.” *Npm*, <https://www.npmjs.com/package/node-cron>.

[15] Raspberry Pi. “Operating System Images.” *Raspberry Pi*, <https://www.raspberrypi.com/software/operating-systems/>.

[16] “Welcome to Python.org.” *Python.org*, <https://www.python.org/>.

[17] “Pyserial.” *PyPI*, <https://pypi.org/project/pyserial/>.

[18] Zeromq. “Zeromq/Pyzmq: Pyzmq: Python Bindings for Zeromq.” *GitHub*, <https://github.com/zeromq/pyzmq>.

[19]“Camarray – ARDUCAM 2MP\*2 Stereo Camera Mipi Module.” *Arducam*, 10 Sept. 2021, <https://www.arducam.com/docs/cameras-for-raspberry-pi/multi-camera-adapter-board/2mp-stereo-camera-global-shutter-mipi-arducam/>.

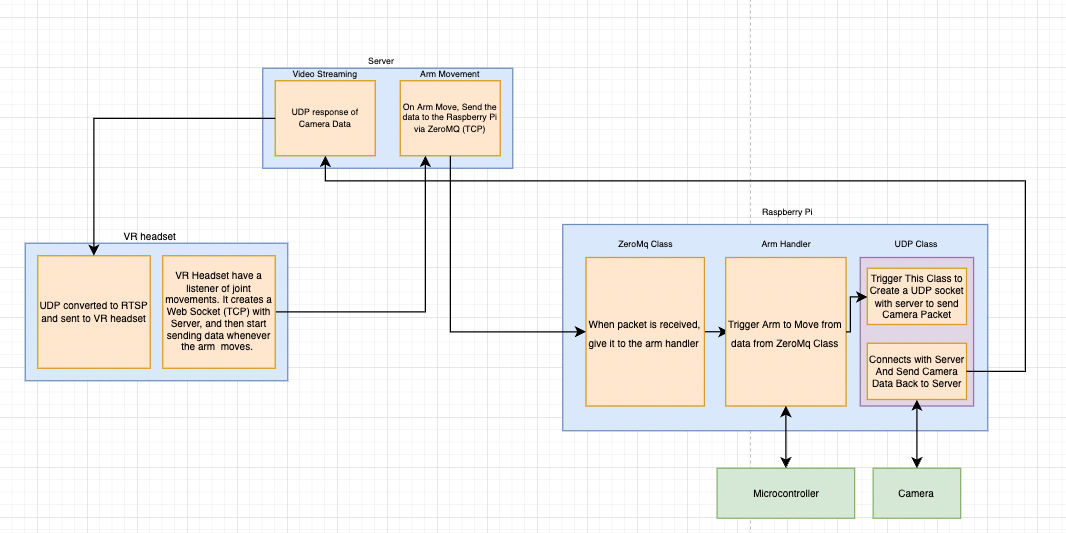
[20] *Certbot*, <https://certbot.eff.org/>.

[21] *Google Domains – Register Your Domain Name – Google Domains*, Google, <https://domains.google.com/>.

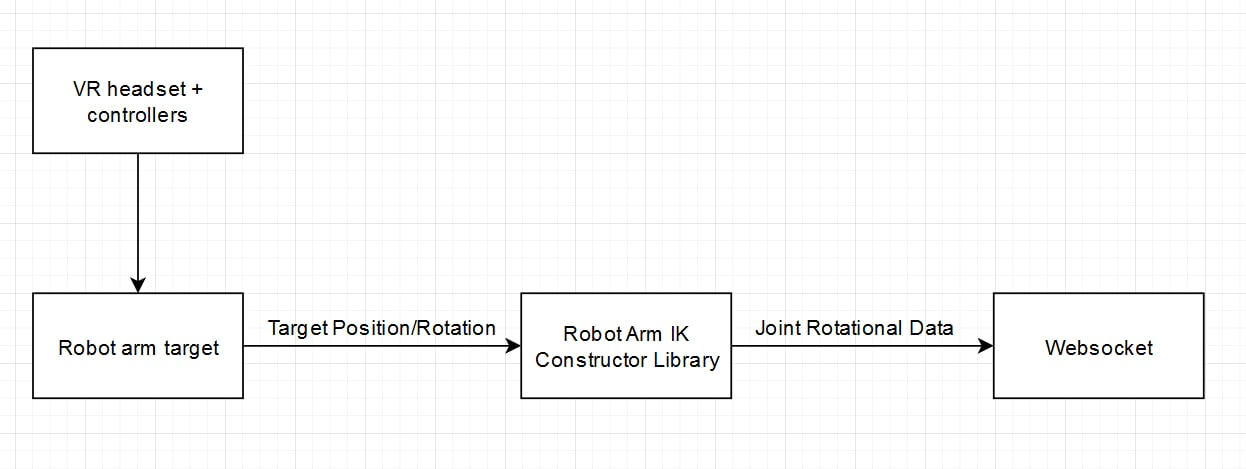
[22] *“Home.” OpenCV, 3 Feb. 2022, https://opencv.org/.*

Appendix 1: Software Component Diagram

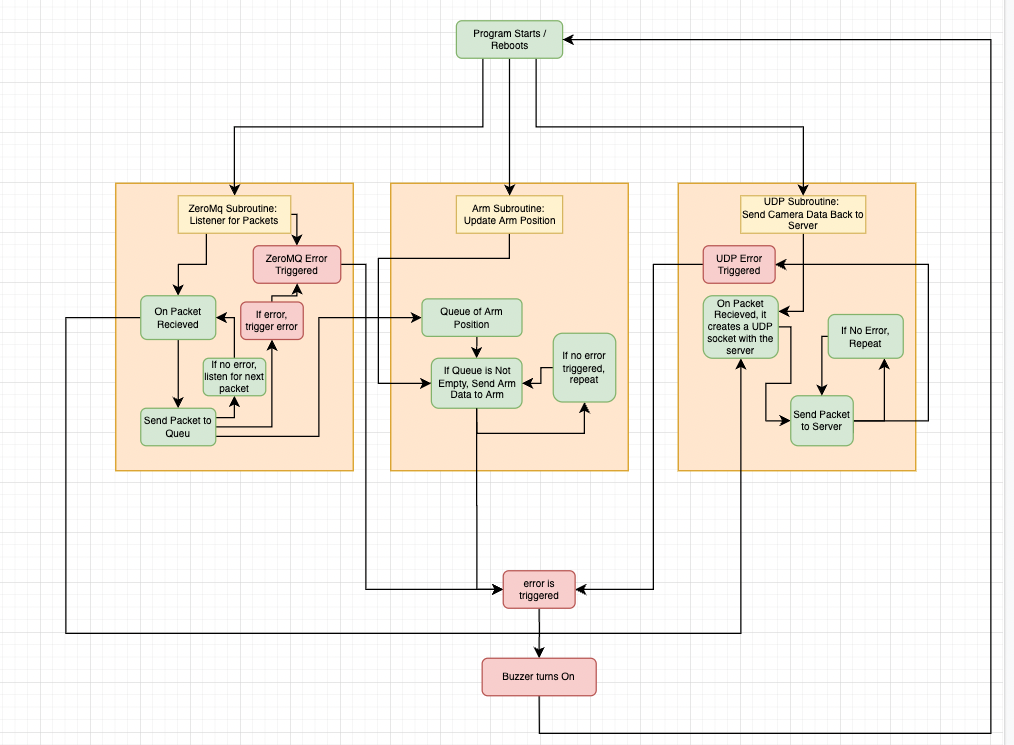
Full Level WorkFlow of Components and its Classes

**

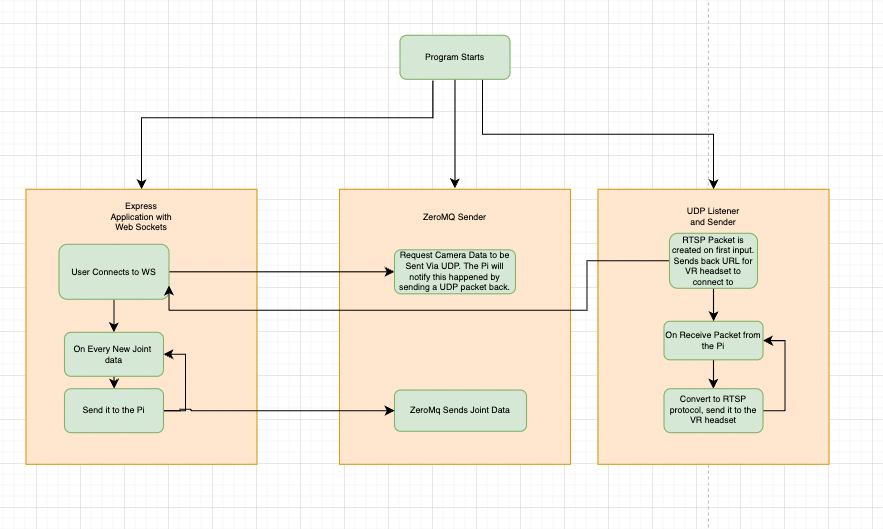
Unity Workflow on how Arm Data is Calculated

**

Raspberry Pi Code and its Subprocesses

**

Server Side Code and its Subprocesses

**