Software Overview

Year: 2022 Semester: 2 Team: 2 Project: VRms

Creation Date: January 28, 2022 Last Modified: January 28, 2022

Author: Brian Latimer Email: blatimer@purdue.edu

Assignment Evaluation:

| **Item** | **Score (0-5)** | **Weight** | **Points** | **Notes** |
| --- | --- | --- | --- | --- |
| **Assignment-Specific Items** | | | | |
| **Software Overview** |  | x2 |  |  |
| **Description of Algorithms** |  | x2 |  |  |
| **Description of Data Structures** |  | x2 |  |  |
| **Program Flowcharts** |  | x3 |  |  |
| **State Machine Diagrams** |  | x3 |  |  |
| **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 Software Overview

*1.1 VR Simulated Robot Arm*

There are three main components of the VR simulation. The headset itself, the robot arm simulation, and the server the simulation connects to . The VR headset comes with their own controllers, allowing us to track head and hand position in the 3D space. The VR controllers also allow for rotation of the hands as well as analog triggers for precise input from the user. With the headset and controllers, the user will easily be able to manipulate objects inside the VR simulation.

The robot arm is made up of 8 joints, all being able to rotate on one axis like a servo motor, and the target, which is where the robot should be pointing at. These joints and target are connected together through an inverse kinematics algorithm, which calculates how far each joint needs to rotate on its axis for the final joint to end up at the target's location. The target is able to be picked up and moved around via the VR controllers, and the robot arm will follow the target as best it can while still working within each joint's rotation limits.

On the server side of the VR simulation, we need to send joint data, receive the real arm’s joint data, and receive the video feed from the two cameras. We will have a script that connects to a server via websockets (UDP) that will send the rotational data for each joint to the server, to tell the arm what rotation each joint should be at. There will be another script that receives data from the server to describe what the arm’s servo motors are currently at, which will be used to determine the accuracy of the simulation. A third script will receive video data from the server, which will then be used to show a 3D view of the physical arm inside the simulation. The 3D view of the physical arm will be created by taking the video feeds of the two cameras and overlaying that on each eye of the headset.

*1.2 Server*

The server we will be using is a Google Cloud VM Instance, and will be used for three purposes. The first purpose of the server is to receive the simulated arms rotational data from Unity, and store it so that the Raspberry pi can retrieve that data. The second purpose is to receive the servo motor’s angular positional data from the Raspberry Pi, which Unity will pull from. The third purpose is to receive video data from the Raspberry Pi, which Unity will also pull from. The server’s purpose is to act as a way to transfer data to and from the Pi and Unity.

The Google Cloud VM Instance will support both UDP and TCP. UDP will be the major component for most of our utility as in terms of keeping an active session, like video streaming or sending and receiving arm data. As for creating and destroying a session, we will still reserve that for TCP to ensure that when a user requests to start or end video streaming, that the server and the Pi should get the confirmation that the connection was successful.

*1.3 Raspberry Pi*

The Raspberry Pi will be used as a way to get data to the microcontroller from the server via UART communication, as well as from the microcontroller to the server quickly and easily. It will also be used to film video using the built in camera port, and send that data to the server with UDP protocol.

*1.4 Microcontroller*

The microcontroller is responsible for communicating with the Raspberry Pi to send and receive the servos motor’s angular data. The microcontroller will be connected to the servos via a motor driver, and will run an algorithm to convert the angular data (degrees) taken from the server to usable data (500-2500) to make the servos move to the correct positions. The microcontroller will also be equipped with a buzzer, which will sound whenever the arm has lost connection with the server. Along with the buzzer, the servos will move to a specified position if the microcontroller/Raspberry Pi cannot reach the server.

2.0 Description of Algorithms

*2.1 Inverse Kinematics*

Inverse Kinematics [1] is an algorithm used for computing the required joint angles for the end effector (final joint) to be able to reach a certain target in a 3D space. The algorithm takes into account the base position, the lengths of each joint, the axis of rotation for each joint, and the target location. Unlike forward kinematics, which calculates the position of the end effector based on the previous joint rotations, inverse kinematics iteratively calculates the rotations for all the joints based on the position of the end effector, which should be at the target. This allows for the joints to move on their own, insead of us having to move each joint rotation at a time. To save time working on this algorithm, we have decided to use a package [2] that handles this algorithm for our simulation.

*2.2 Joint Translation*

We will be writing an algorithm for the microcontroller that will allow us to take the angular data from the server, and convert that to usable data for the servos. This is needed because the data coming in is in degrees, while the servos operate with a range of 500-2500 to determine the angular position. This is important because the servos might be slightly off in angular position (50° from the simulation causing the servo to move to 60° for example) from the simulated arm, so this algorithm will determine the proper angles the servos will move at to best align with the simulations, through the means of calibration and offsets.

3.0 Description of Data Structures

*3.1 UDP/TCP*

So UDP is a protocol that we are going to stick with in terms of live streaming video from the Raspberry Pi and sending and receiving joint data. UDP is better than the other protocol in our network layer stack because we want to display recent data without the process of confirming that the data packet has been sent, which is a feature inside TCP. In addition, UDP is a lot faster in terms of sending data between our three endpoints. TCP will still be used in terms of starting and ending a session with the VR headset. The reason being is that before we start sending video and joint data to each other, all components need a confirmation with each other that they are ready to send data before actually start sending data.

*3.2 SPI*

Serial Peripheral Interface, SPI, is a synchronous serial data protocol used by microcontrollers to communicate with other devices quickly. In our project, we will be using SPI to communicate between the Raspberry Pi and the Microcontroller.

*3.3 UART*

UART is a common device-to-device asynchronous communication protocol. It allows for microcontrollers to work with many different types of serial protocols. In our project, we will be using UART to communicate between the microcontroller and the servo driver, since SPI is not supported by the servo driver.

*3.4 JSON*

JSON is a standard text-based format for representing data structures. It allows for data to be easily formatted, and is usually used to send data to servers. In our project, JSON packets will be used to transmit the robot arm’s angular data to the server, from both the simulated and physical arm.

*3.5 Base64*

Base64 is an encoding scheme to convert binary data to ASCII. Base64 is used to carry binary data through channels that only support text data. We are using this data type for the communication of our video taken from the Pi to the server, and then from the server to Unity to display in the headset. Base64 is used for this since it is fast, and low latency is necessary for a good live video viewing experience.

*3.6 Queue/FIFO*

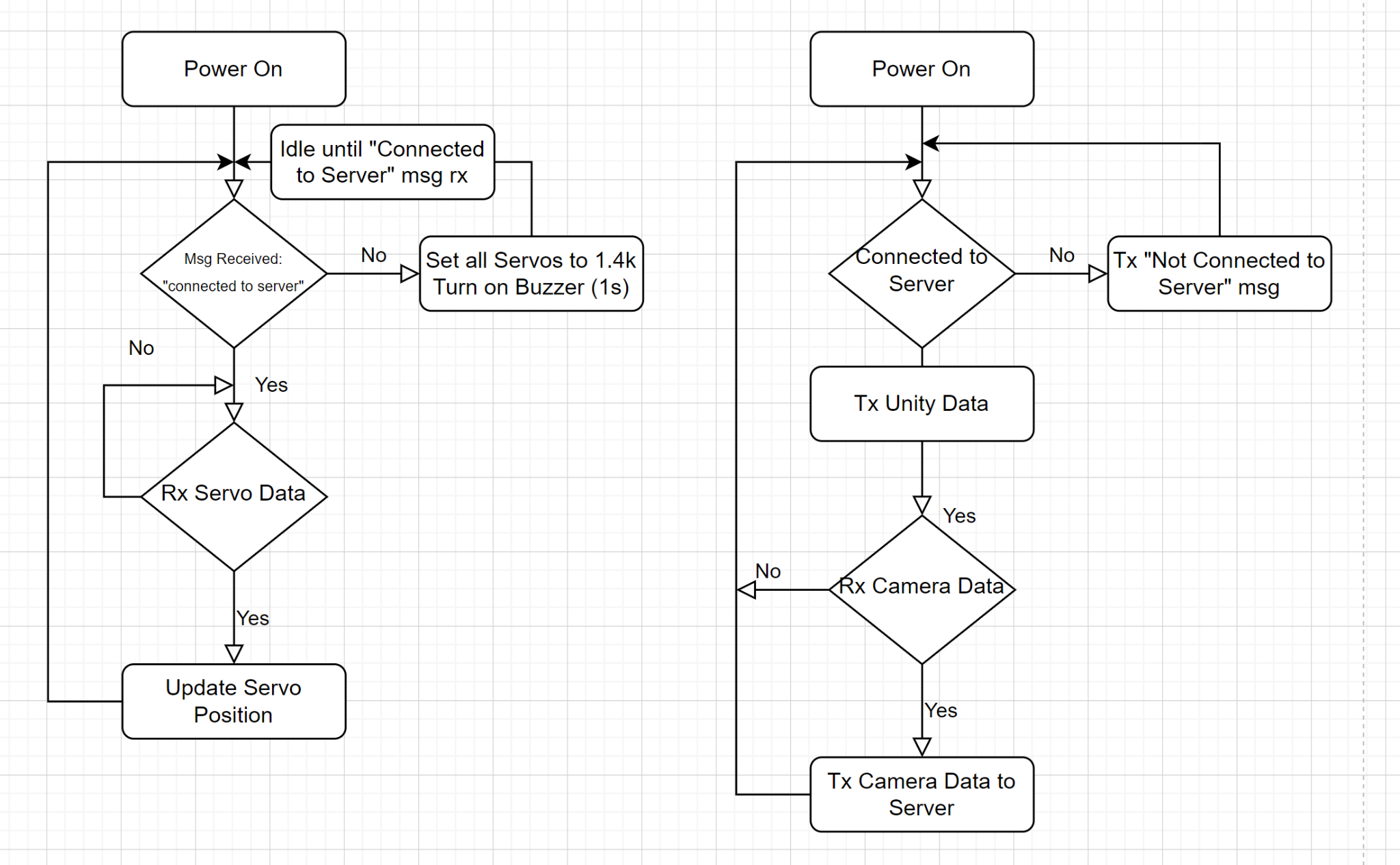
A queue is a linear first in first out data structure. That means that the first inputted into the queue is the first acted upon. Queues will be used to send angular data from the microcontroller to the servo drivers in order to control the servos.

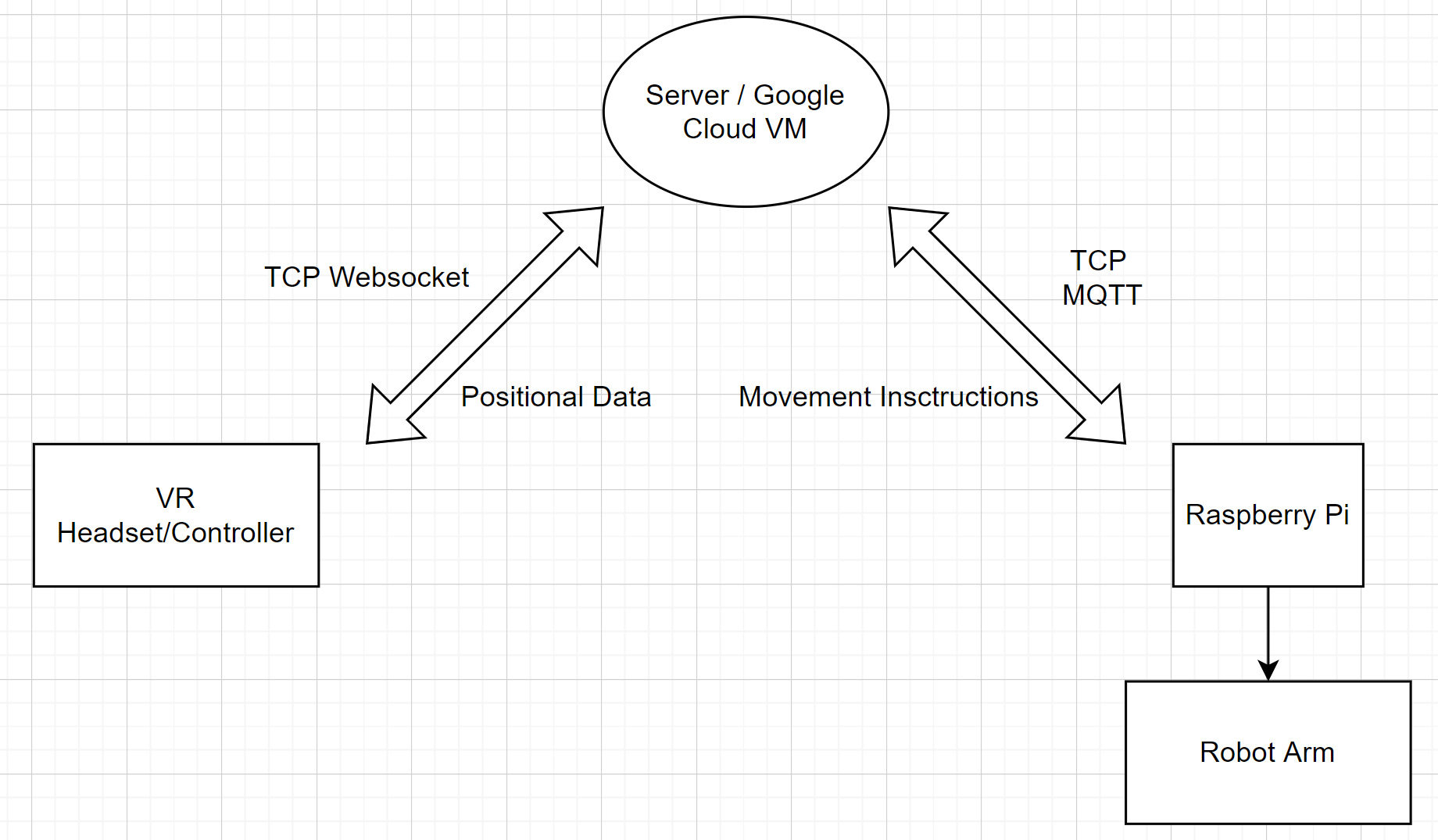
4.0 Sources Cited:

[1] automaticaddison, Author. “The Ultimate Guide to Inverse Kinematics for 6dof Robot Arms.” *Automatic Addison*, 6 Dec. 2020, https://automaticaddison.com/the-ultimate-guide-to-inverse-kinematics-for-6dof-robot-arms/.

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

Appendix 1: Program Flowcharts





Appendix 2: State Machine Diagrams

