DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
THE UNIVERSITY OF TEXAS AT ARLINGTON

DETAILED DESIGN SPECIFICATION
CSE 4317: SENIOR DESIGN II
SPRING 2021

ARM SOLUTIONS
PACK MAN

GREGORY FERGUSON
KRISHNA PATEL
DONNY PHAM
INARA RUPANI
MATHEW ZINKE

Revision History

Revision	Date	Author(s)	Description
0.1	2.05.2021	MZ	Adds Team information
1.1	3.11.2021	GF	Completed Hand-Eye Calibration subsystem
1.2	3.11.2021	MZ	Completed Tool Effector subsystem
1.3	3.11.2021	DP	Added Vision layer
2.1	3.12.2021	IR	Revised Vision layer and added pictures
2.2	3.12.2021	KP	Revised Arm Control layer
3.0	3.12.2021	MZ, GF, DP, IR, KP	Document Completed
4.0	5.10.2021	MZ, GF, DP, IR, KP	Revised System

CONTENTS

List of Figures

List of Tables

1 Introduction

The overlying idea of Arm Solutions is to reduce contact between people and packages. This will be achieved through an automated robotic arm that picks up packages and moves them to an area within the work cell.

1.1 Purpose and Use

The ongoing global pandemic is an issue that affects many aspects of society, including supply chains. Companies are adapting to new conditions to allow for social distancing and more hygienic work environments. The goal of Arm Solutions is to reduce contact between people and mediums of transmission. By automating parts the movement & sorting of packages, exposure is reduced.

1.2 Intended Audience

The intended audience includes all major shipping and acquisitions companies. This includes not only postal companies, but companies that have high volumes of incoming and/or outgoing materials. Additionally, this can apply to companies it large amounts of internal mail. This reasoning is, because of Arm Solutions' goal to reduce personnel required to handle packages.

Figure 1: Conceptual drawing

2 System Overview

The system of UR5 parcel sorter machine is divided into two high level layers: Graphical User Interface (GUI) and Parcel Sorter robot as seen in Figure 2. This will allow users to understand what the Parcel Sorter is currently doing and to control it. The communication between these two layers can easily been shown in the figure below.

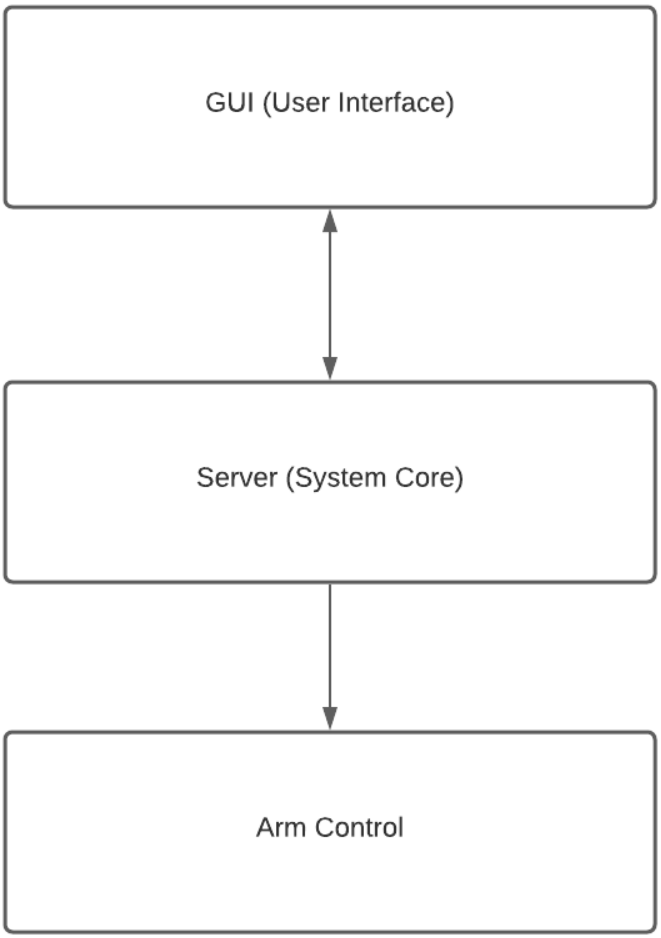

Figure 2: UR5 Parcel Sorter System Overview

3 Server (System Core) Layer

The Server (System Core) layer is responsible for maintaining and controlling the rest of the system. Moreover, the Server (System Core) layer will also have the core part of the communications subsystem, which controls the communication between all the layers (the implementation details for it are also below). Lastly, the Server will contain the Vision subsystem, responsible for identify the parcel and its location.

3.1 Layer Hardware

This layer will run on the Raspberry Pi.

3.2 Layer Operating System

Same as the rest of the layers, it will be running on Linux (Ubuntu).

3.3 Layer Software Dependencies

Python 3.x.x Python socket and multi-threading libraries (built-in).

3.4 Networking/System Control Subsystem

Each layer in the system needs a way to communicate with each other (each layer is isolated), so the communications subsystem deals with communication between different layers (code for the communication subsystem is in every layer, but its core part will be in the control layer). To facilitate communication, we are using sockets (endpoints of bidirectional communications channel) using the TCP/IP standard. The Server layer will act as the the main hub and the other layers will be clients. As well, since there will be multiple clients, the server will need to be multi-threaded to allow for full-duplex communication. While we tested other methods of communication (HTTP), we found the simple server and client modules work best for us. While the data structures and data processing used in the subsystem are explained in detail below, we will be using JSON (JavaScript Object Notation) as the format to exchange data. In each layer, the subsystem will be coded as a class or function (whichever makes more sense in implementation). Furthermore, the System Control part of this subsystem is the backbone for the rest of the system, controlling and monitoring the rest of the system.

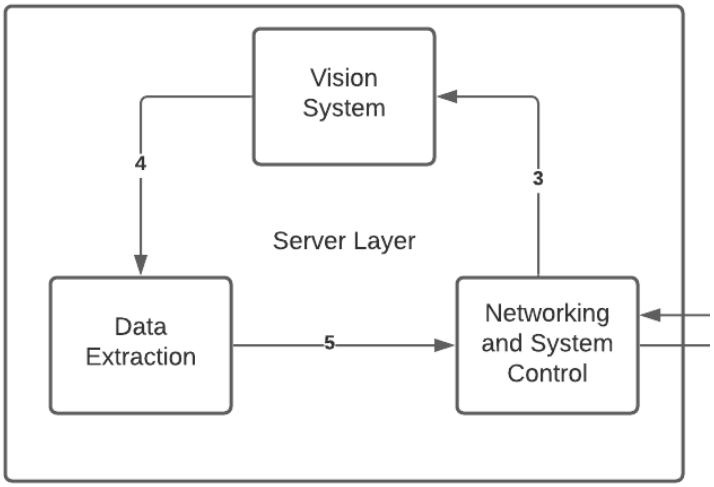

Figure 3: Networking/System Control subsystem diagram

3.4.1 Subsystem Hardware

This subsystem will run on the main PC (whatever we call it).

3.4.2 Subsystem Operating System

Same as the rest of the system, it will be running on Linux (Ubuntu).

3.4.3 Subsystem Software Dependencies

Python has built in libraries for network communication (socket). Multi-threading will be enabled by pythons built in libraries (threading and ThreadingMixIn).

3.4.4 Subsystem Programming Languages

Python 3.x.x

3.4.5 Subsystem Data Structures

As mentioned above, the format we will be using to exchange data will be JSON (the data structure containing the data). JSON is built on two data structures: A collection of name/value pairs and an order listed of values. For the first structure, this is often realized as an object and for the second, this is often an array (or vector). This is ideal since nearly all programming languages support these data structures. This allows us to treat the data as objects (e.g. a position in 3D space would have the form x:1, y:0, z:0).

3.4.6 Subsystem Data Processing

For network communication, sockets are used as endpoints for programs to connect to. Since we are using the TCP/IP standard, we will be using IPV4 How sockets work will not be explained here, but each program will have its socket object (the server socket has more parameters and setup than client sockets). The server socket will need to be started first for the clients to connect. Once the connection is established a new thread will be spawned to handle each client and data will start being transferred between each layer.

Since we are using JSON as the data format, the only processing between each message is the encoding of the JSON object and decoding the JSON object on the receiving end. A header for each message will be used so the server and clients can identify what the JSON message destination is. It is important to note any error checking is done in the transport layer (TCP takes care of this), not in the application layer.

3.5 Vision Subsystem

Convolution Neural Network is a deep learning model that is used to analyze visual imagery. For this project, the data-set will be sample of pictures taken by camera then we will normalize the data-set by creating bounding boxes around. We will use Intel Real Sense camera in real-time to catch these images of parcels on the tool-bench. Once the images are captured, the process of data extraction begins to process the images.

3.5.1 Subsystem Software Dependencies

YOLOv5 is an open source convolution network for object detection done in real time. The model is trained using a data set constrained to brown boxes and yellow Amazon envelopes for speedy development.

3.5.2 Subsystem Programming Languages

Python 3.x.x

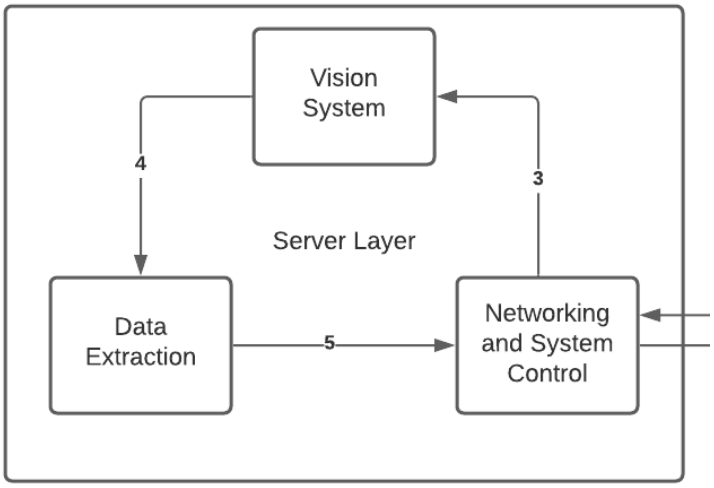

Figure 4: Vision Subsystem Diagram

3.5.3 Subsystem Data Structures

The parcels will be represented as the BoundingBox data structure contains the upper left corner x and y and the bottom right corner x and y in the array form [x1, y1, x2, y2]. The subsystem will output a single BoundingBox data structure that represents an object that the system has the highest confidence.

3.5.4 Subsystem Data Processing

The subsystem is using the yolov5 bounding box object detection model which is mainly used for its speed. It is so fast that it can process 45 frames per second. YOLO provides optimal speed and accuracy by the training model. The model will predict where the parcels from the image data and will output the detected parcel's bounding box.

3.6 Data Extraction Subsystem

The data extraction subsystem will apply algorithms to determine the center of the parcel's position coordinates and its depth from the object detection subsystem's bounding boxes. As the data extraction ends it sends the commands to UR5 arm to move and activate hand-eye calibration under Arm Control subsystem.

3.6.1 Subsystem Software Dependencies

Intel Realsense library, librealsense.

3.6.2 Subsystem Programming Languages

Python 3.x.x

3.6.3 Subsystem Data Structures

The subsystem will output a single position array called Position in the form [x,y].

3.6.4 Subsystem Data Processing

The midpoint of the bounding box is calculated. Then the depth value will be extracted by using the midpoint from the Intel Realsense's depth frame. Then each midpoint will transform the image position to position relating to the camera as [x, y, z] with the camera is the origin point. Then the camera

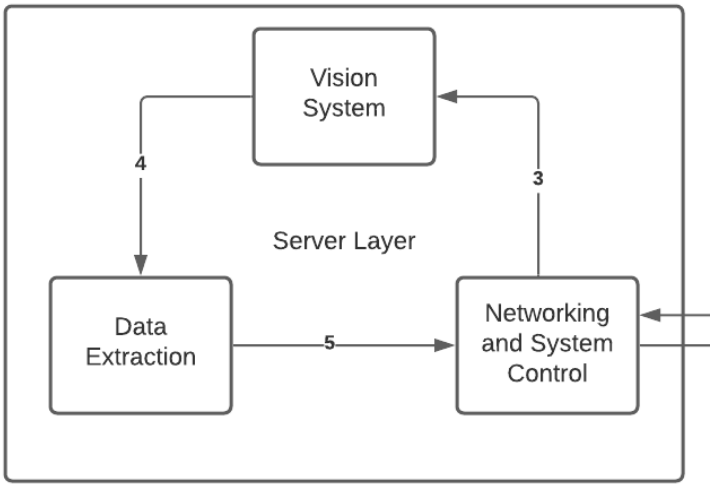

Figure 5: Data Extraction Diagram

based position will transform to the arm based position as [x, y, z] with the arm as the origin point. The subsystem will output an array called ParcelPoint in the form as [[x,y,z], [x,y,z] ...]

4 Arm Layer Subsystems

This system encompasses the use of a Universal Robotic Arm version U5

4.1 Layer Hardware

Data will be processed on a PC then transmitted to UR5 using an Ethernet connection. Image data will be collected using Intel Real Sense Camera.

4.2 Layer Operating System

UR5 Controller runs on a Linux based OS and the PC that will be used to process data uses Ubuntu 20.4

4.3 Layer Software Dependencies

Data such as raw images containing packages will be processed using OpenCV library.

4.4 Networking/User Commands Subsystem

Each layer in the system needs a way to communicate with each other (each layer is isolated), so the communications subsystem deals with communication between different layers (code for the communication subsystem is in every layer, but its core part will be in the control layer). To facilitate communication, we are using sockets (endpoints of bidirectional communications channel) using the TCP/IP standard. The Server layer will act as the the main hub and the other layers will be clients. As well, since there will be multiple clients, the server will need to be multi-threaded to allow for full-duplex communication. While we tested other methods of communication (HTTP), we found the simple server and client modules work best for us. While the data structures and data processing used in the subsystem are explained in detail below, we will be using JSON (JavaScript Object Notation) as the format to exchange data. In each layer, the subsystem will be coded as a class or function (whichever makes more sense in implementation). Furthermore, the User commands part of this subsystem sends commands (controlled by the operator) to the server, to modify the system parameters (e.g. drop location, arm velocity, etc.)

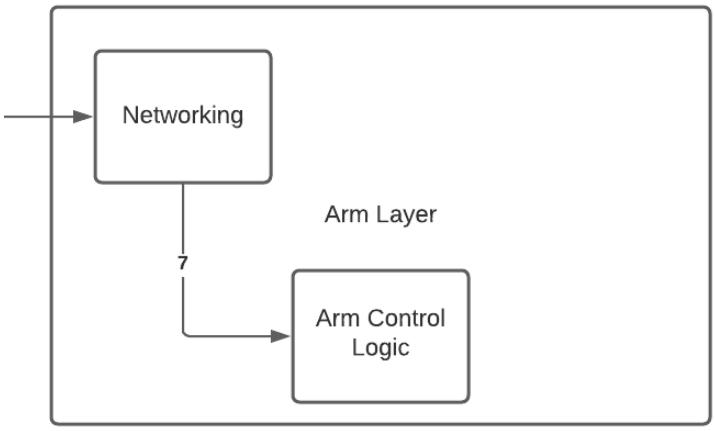

Figure 6: Networking subsystem diagram

4.4.1 Subsystem Hardware

This subsystem will run on the main PC (whatever we call it).

4.4.2 Subsystem Operating System

Same as the rest of the system, it will be running on Linux (Ubuntu).

4.4.3 Subsystem Software Dependencies

Python has built in libraries for network communication (socket). Multi-threading will be enabled by pythons built in libraries (threading and ThreadingMixIn).

4.4.4 Subsystem Programming Languages

Python 3.x.x

4.4.5 Subsystem Data Structures

As mentioned above, the format we will be using to exchange data will be JSON (the data structure containing the data). JSON is built on two data structures: A collection of name/value pairs and an order listed of values. For the first structure, this is often realized as an object and for the second, this is often an array (or vector). This is ideal since nearly all programming languages support these data structures. This allows us to treat the data as objects (e.g. a position in 3D space would have the form x:1, y:0, z:0).

4.4.6 Subsystem Data Processing

For network communication, sockets are used as endpoints for programs to connect to. Since we are using the TCP/IP standard, we will be using IPV4 How sockets work will not be explained here, but each program will have its socket object (the server socket has more parameters and setup than client sockets). The server socket will need to be started first for the clients to connect. Once the connection is established a new thread will be spawned to handle each client and data will start being transferred between each layer.

Since we are using JSON as the data format, the only processing between each message is the encoding of the JSON object and decoding the JSON object on the receiving end. A header for each message will be used so the server and clients can identify what the JSON message destination is. It is important to note any error checking is done in the transport layer (TCP takes care of this), not in the application layer.

4.5 Arm Control Logic Subsystem

This Arm Control Logic subsystem has several functions, including a pick and place (includes tool effector) and the path planning. Furthermore, the tool effector is the part of the robot that enables the arm to pickup a parcel. It contains the suction gripper and its mount. The gripper's mount is a 3D printed part, while the gripper is simple mechanical device designed to pick up the parcel. It works using a pneumatic tube system and compressor to create the vacuum. The vacuum is enabled (allowing for gripping) when the Hand-Eye Calibration subsystem has placed the tool effector onto the parcel.

4.5.1 Subsystem Hardware

The specialty hardware includes the suction gripper, a mount, and the pneumatic tubing to make it function. Beyond this is the main PC on which the program runs.

4.5.2 Subsystem Operating System

This system will use a PC running Ubuntu 20.04 OS

4.5.3 Subsystem Software Dependencies

Coordinates mapping the motion of the UR5 robotic arm will be sent using python scripts. Coordinates are determined relative to where the package is in the work cell. Image data will be processed using

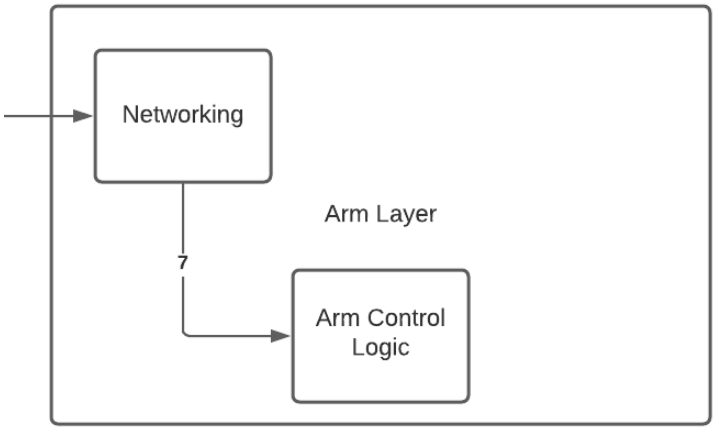

Figure 7: Arm Control Logic subsystem diagram

OpenCV.

4.5.4 Subsystem Programming Languages

All the code will be written in Python

4.5.5 Subsystem Data Structures

Data being transmitted from a PC to the UR5 via Ethernet can be sent as byte size commands that include the function command that needs to be executed along with the required parameters. Data can also be sent as strings with the same setup but by Real Time Data Exchange module built into the robot. Data such as joint data can be sent back to the PC via Ethernet in the format of vectors.

4.5.6 Subsystem Data Processing

There is no special algorithm being implemented except for the ones mentioned in the Hand-Eye Calibration.

5 GUI Layer Subsystems

This system encompasses the User Interface controlled by the operator (user).

5.1 Layer Hardware

Capable of running on any PC with a OS (hardware requirements are low).

5.2 Layer Operating System

Since socket are handled the same regardless of the OS, any OS should be compatible (tested on Windows and Linux).

5.3 Layer Software Dependencies

python 3.x.x

5.4 Interface Subsystem

This is the GUI part of the system that starts, stops, and troubleshoots the main functions for the other subsystems. The User Interface (GUI) allows the operator to control the system from anywhere if itâs connected to the network. The GUI itself is built using Tkinter (pythonâs default GUI library). A video streaming functionality is also built into the server and GUI to allow the operator a visual image of the work-space. Additionally, it will include install abilities for all python libraries and dependencies.

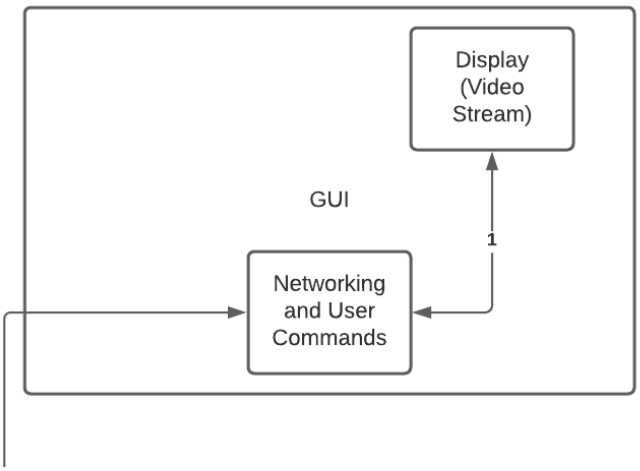

Figure 8: Interface subsystem diagram

5.4.1 Subsystem Hardware

This subsystem will run on the main PC.

5.4.2 Subsystem Operating System

Since socket are handled the same regardless of the OS, any OS should be compatible (tested on Windows and Linux).

5.4.3 Subsystem Software Dependencies

The main dependencies include python 3.x.x

5.4.4 Subsystem Programming Languages

Python

5.4.5 Subsystem Data Structures

For the convenience of creating and executing this interface, JSON was used as the data structure for packet communication. In python, JSON is treated as a dictionary. This allows us to standardize the communication process.

5.4.6 Subsystem Data Processing

Since we are using JSON as the data format, the only processing between each message is the encoding of the JSON object and decoding the JSON object on the receiving end. A header for each message will be used so the server and clients can identify what the JSON message destination is. It is important to note any error checking is done in the transport layer (TCP takes care of this), not in the application layer.

5.5 Networking/User Commands Subsystem

Each layer in the system needs a way to communicate with each other (each layer is isolated), so the communications subsystem deals with communication between different layers (code for the communication subsystem is in every layer, but its core part will be in the control layer). To facilitate communication, we are using sockets (endpoints of bidirectional communications channel) using the TCP/IP standard. The Server layer will act as the the main hub and the other layers will be clients. As well, since there will be multiple clients, the server will need to be multi-threaded to allow for full-duplex communication. While we tested other methods of communication (HTTP), we found the simple server and client modules work best for us. While the data structures and data processing used in the subsystem are explained in detail below, we will be using JSON (JavaScript Object Notation) as the format to exchange data. In each layer, the subsystem will be coded as a class or function (whichever makes more sense in implementation). Furthermore, the User commands part of this subsystem sends commands (controlled by the operator) to the server, to modify the system parameters (e.g. drop location, arm velocity, etc.)

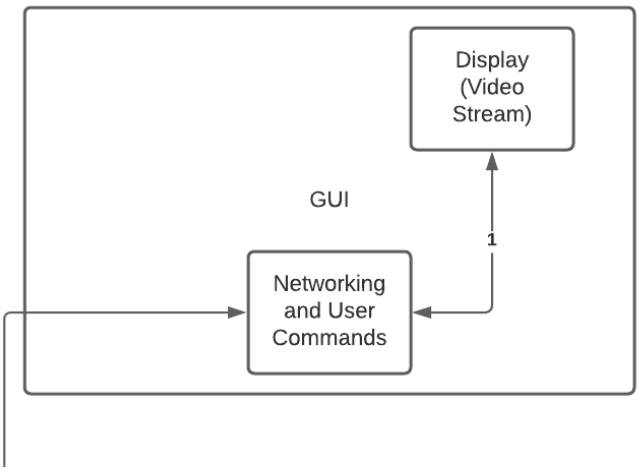

Figure 9: Networking/User Commands subsystem diagram

5.5.1 Subsystem Hardware

This subsystem will run on the main PC (whatever we call it).

5.5.2 Subsystem Operating System

Same as the rest of the system, it will be running on Linux (Ubuntu).

5.5.3 Subsystem Software Dependencies

Python has built in libraries for network communication (socket). Multi-threading will be enabled by pythons built in libraries (threading and ThreadingMixIn).

5.5.4 Subsystem Programming Languages

Python 3.x.x

5.5.5 Subsystem Data Structures

As mentioned above, the format we will be using to exchange data will be JSON (the data structure containing the data). JSON is built on two data structures: A collection of name/value pairs and an order listed of values. For the first structure, this is often realized as an object and for the second, this is often an array (or vector). This is ideal since nearly all programming languages support these data structures. This allows us to treat the data as objects (e.g. a position in 3D space would have the form x:1, y:0, z:0).

5.5.6 Subsystem Data Processing

For network communication, sockets are used as endpoints for programs to connect to. Since we are using the TCP/IP standard, we will be using IPV4 How sockets work will not be explained here, but each program will have its socket object (the server socket has more parameters and setup than client sockets). The server socket will need to be started first for the clients to connect. Once the connection is established a new thread will be spawned to handle each client and data will start being transferred between each layer.

Since we are using JSON as the data format, the only processing between each message is the encoding of the JSON object and decoding the JSON object on the receiving end. A header for each message will be used so the server and clients can identify what the JSON message destination is. It is important to note any error checking is done in the transport layer (TCP takes care of this), not in the application layer.

6 APPENDIX A

Include any additional documents (CAD design, circuit schematics, etc) as an appendix as necessary.

REFERENCES