SYSC 3010A: 3rd Year Computer Systems Engineering Project

Line-Following Assistive Device [LAD]

LAD System Final Project Report

Github: https://github.com/xXBakonatorXx/SYSC3010-Group-W2.git

Denise Mayo

Zachary Porter

Erdem Yanikomeroglu

Brannon Chan

Table of Contents

RESERVED FOR ToC to be completed after report is finished (for page numbers and formatting purposes)

1.0 Introduction

Many people who are faced with pain or muscle weakness are unable to perform basic tasks due to age, previous injuries, disabilities and a variety of other factors. In many cases, they may not have access to around the clock care.

The LAD System is designed to alleviate this financial burden using assistive technology. It could retrieve items (such as medication containers or TV remotes) on demand or at a set interval, preventing the need to someone to navigate their home and risk a fall or other accident, thereby preventing personal injury and damage to property.

1.1 Design Solution

The LAD System consists of 3 modules: the LAD Mobile Unit, a Central Server, and an Android App.

The home would be outfitted with a ‘roadway’ consisting of black lines installed on the floor and QR codes installed at intersections or locations of importance. These QR codes provide the LAD unit self-awareness to its location, and allow it to make decisions on where to go when presented with an instruction.

These instructions are sent to the LAD Mobile unit from the Central Server. This module contains a list of items and their associated locations, as well as a relational map of locations in the home stored as nodes and connections between them.

All interactions with the LAD Unit will take place through the LAD’s user interface in the android app making it very intuitive for the user to use. The App will allow for updates to the item or location database, manual control of the LAD Unit, and requests for the performance of tasks on demand or on a schedule.

2.0 Design

2.1 System Architecture

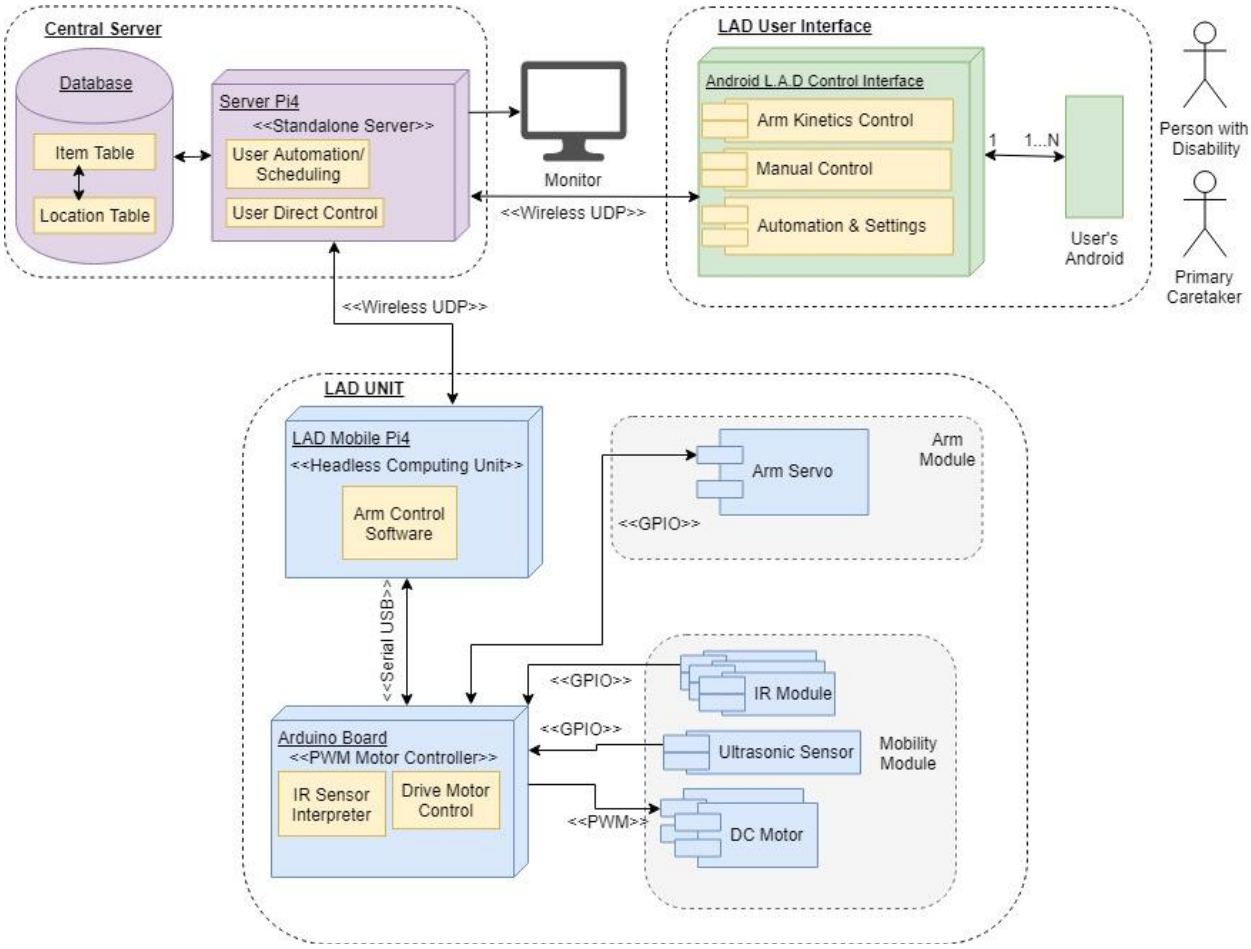


Figure 1:Overall System Architecture diagram for the LAD System, including the central server, User interface, and LAD Mobile Unit

As mentioned above, the LAD system is broken down into 3 main modules, each with their own sub-systems.

1. **LAD User Interface**, implemented via an Android smartphone app. Multiple functions will be available, including:
    a. Item lists - pre-program a list of commonly fetched items and their normal locations

    b. Manual Control - manual control of the LAD Unit
2. **Central Server** hosting 2 main tables in its database.
    a. Item Database - a database of stored and saved items and their designated locations
    b. Location Database - every location data point is saved in the form of a QR code, including a series of instructions on how to reach that QR code from home base
3. **LAD Mobile Unit**
    a. Mobility module - Includes two DC motors for mobility and three Infra-Red sensors for navigation, controlled by an Ardino
    b. Arm Module - Two servo motors controlled by the Onboard Processing module to adapt in height and pick up objects
    c. Onboard Processing module - Inputs and outputs for navigation, mobility, arm control, and server requests and polling will be routed through the main Raspberry Pi 4 and processed before being sent to appropriate destinations.

2.2 Hardware Layouts

2.2.1 Mobility

The mobility system is used is used for driving the LAD Mobile Unit. This system consists of 3 IR Sensors to detect the line that the LAD Mobile Unit follows, an Ultrasonic sensor to detect obstacles that are in the way, and the two DC motors used to drive the LAD Mobile Unit. The mobility system was controlled directly by the Arduino mounted to the LAD Mobility Unit.

2.2.2 Arm Control

The Arm control is run by the Arduino as well and contained one servo motor to raise and lower the arm. The Arm Control allows the LAD Unit to pick up small, light objects and carry them around.

2.3 Database Architecture

The Central Server acts as a data storage hub and a router for messages between the android app and LAD mobile unit. The Server contains 2 database tables, pictured below.

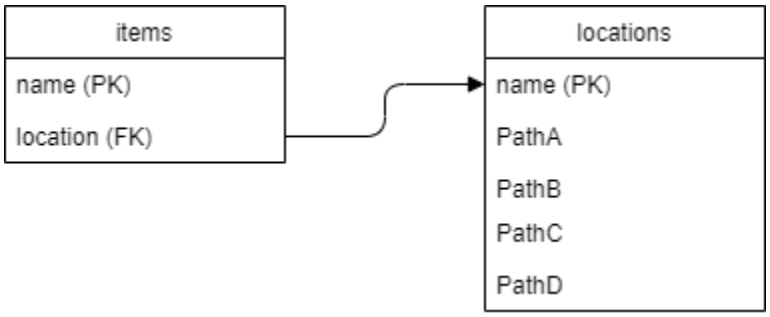


Figure #: Database Schema for the LAD Server

The database serves as a reference for all stored information in the system. The database consists of 2 tables:

- Items - a table of items registered in the system and their corresponding locations
- Locations - a table of each registered location and the connected nodes
    - The locations become a sort of map, pictured below with an example of a possible map and tables associated with it.

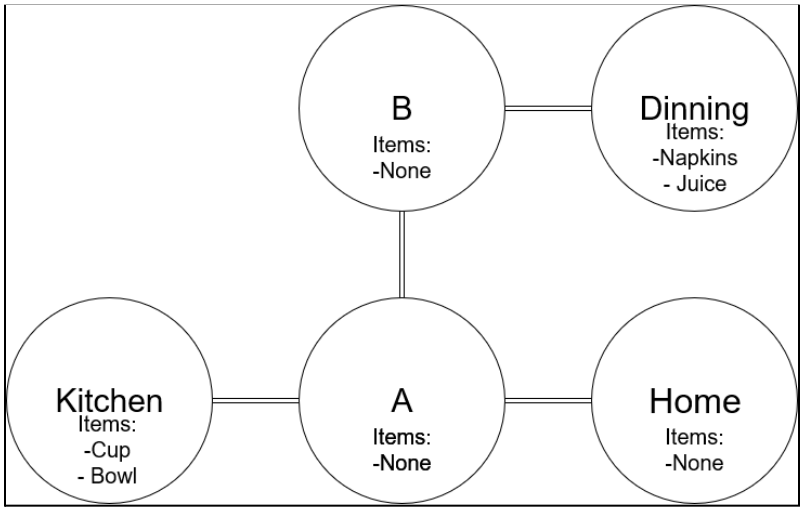


Figure #: A sample map of a home with items

Name	PathA	PathB	PathC	PathD		Name	Location
Home	A	---	---	---		Cup	Kitchen
A	Kitchen	B	Home	---		Bowl	Kitchen

B	---	---	Dinning	A		Napkin	Dinning
Kitchen	---	---	A	---		Juice	Dinning
Dinning	B	---	---	---			

Figure #: The database contents (locations in blue and items in green) for the above sample map

2.4 Android Interfacing

The Android Application used in this project was developed from the ground up in Android Studio; an IDE most commonly used to develop applications for mobile Android OS devices. This allowed us to create a rudimentary wireless User Interface for the Primary Caregiver to interact with (refer to FIgure # - LADapp main screen).

The Final version [LADappPhase4] is the latest version of the App despite the dated project name in the “android” branch of the team repository. This version of the App has all defunct code and UI elements removed, using only what data would have been provided to it from the For a better idea of the previous iterations of the LADapp, refer to Figure _ in Appendix C. This streamlined version allows the User to utilize the LAD Unit’s Manual Control Features to a far further extent by utilizing the ‘Manual Control’ Opcodes on the LAD Unit. In addition, the app provides the User the ability to ‘Create a new Scheduled PIckup” task, or pre-empt the scheduled task(s) and run an existing Item Pickup immediately.

To Utilize the Manual Control Options on the man View (the screen upon opening the app), the User must first “Enable Manual Controls” by clicking the Switch Object in the Top-Right of the screen. This will set all Manual Control Options for the LAD Unit to “setEnabled(true)”, making the buttons operable for the User. This provides the User with Drive, Arm, Wrist, and Claw Button Controls, as well as an Emergency Stop Button in case of a possible emergency during Manual Operation. All commands, once selected via a Button Press Event, are sent to the Central Server RPi via a UDP IPv4 Wireless Connection in an AsyncTask subclass thread instance that runs parallel to the Main UI Thread. The User will be notified of the state their sent command is in using the 4 stages provided in AsyncTasks’ ‘pipelining’ methods to update the Command Status TextView Object just under the Button objects. As an additional safety feature, the Emergency Button is linked to an

OnClickEventListener() that will, upon being pressed, disabled all manual controls, effectively acting as a redundant safety on the User's end.

The User also has the ability to immediately run any pre-existing Item Pickup task. This is done by selecting the specific item the User wants to retrieve using the Drop-Down style Spinner object, and then clicking the "Run Task Now" Button object. This will send a fetch command for the selected item to the Central Server via UDP IPv4 Wireless Connection, which then dispatches the correct command to the LAD Unit.

Lastly, the User may create a new Item to be fetched from a pre-existing location by clicking on the "New Scheduled Item" (or red '+') Button in the Bottom Right. This brings the User to a new "screen" (createTaskActivity), where they can enter the Name, Time, the day(s) in a week they wish their Item to be retrieved, and at which pre-existing location the item will be retrieved from. After the User is satisfied with their changes, they may choose to 'Create' the new Item Task, or 'Cancel' and go back to the main screen (MainActivity).

2.5 Communication Protocols

TThe messages sent within the system are nested JSON dictionaries and consist of the following structure:

- Commands - the overarching commands used to direct the system what to do with this message
    - The 4 commands used are **insert**, **delete**, **display** and **relay**.
        - **Insert** indicates that an item or location will be added to the corresponding table, including all required fields
        - **Delete** indicates that an item or location will be deleted from the corresponding table, including all corresponding foreign keys
        - **Display** indicates that the entire contents of a particular table can be displayed on the Android device
        - **Relay** indicates that the command being received needs to be forwarded to the LAD Mobile Unit. These messages usually contain manual control instructions
- Table Names - when applicable, a command will affect a table

- General data - Data to be affected or transported

For example, if we wanted to delete the item “cup” from the table, then the message would look like this:

```
{“delete”:
    {“items”:
        {“name”: “cup”}
    }
}
```

Because everything needs to be autonomous, the messages can be decoded and logic within the server unit can decide what to do with said data depending on the command and where it was sent from.

The series of calls made to make the system perform a task is as follows:

1. The Android App sends a message to the server including an instruction and some data.
2. The server will interpret the command and identify it as one of the above (listed above figure #). Each command has its own specific sequence. If the command doesn’t match one of the above, the system does nothing.
3. After the command has been performed, a message will be returned to the Android app informing the user the task has been completed.

3.0 Testing

3.1 Database Unit Testing

Mock - What did we build and what did we learn from it

The database functions and communication was tested using unit tests primarily. Commands were sent to the database as read in from a JSON file. A testing script then tested the results by reading from the database with the SELECT SQL functionality. Below is a table of unit tests for common and edge cases:

No.	Name	Description	Inputs	Expected Outputs

1	createMultiple	Create multiple rows in the same table	name, location (read in JSON file)	Multiple table rows created in input order (unsorted)
2	createItem	Create new row with no null values in **items** table (no null values allowed)	name, location (read in JSON file)	New database row containing the values
3	createItemNull	Throw error when null values are passed to table **items** (no null values allowed)	name, location (read in JSON file)	Throw error
4	createLocationNoNull	Create new row with no null values in table **map** (null values allowed in 4 path fields)	name, location (read in JSON file)	New row with no null values
5	createLocationNull	Create new row with null values in table **map** (null values allowed in 4 path fields)	name (not null), pathA-PathD = null	New row with null values
6	nullNameField	Throw error when name field is null (name column is not allowed to be null)	name = None, location = anything	Throw error
7	deleteItem	Delete a row in table **items**	name	Row is deleted
8	selectNotExist	Throws error if user tries to access a row that does not exist	name	Throw error, prompt to create a row containing that information
9	displayLocationItems	Display all items in table **items** registered to a specified location	location	Display all item names in that location
10	displayMap	Display all locations in table **maps** connected to path A	name = PathA	Display all location names connected to that point
11	displayItemPath	Input single item name from table **items** and return location and path to get there from home	name of item	return full path including all nodes to traverse to get there
12	denyDuplicate	Send warning for duplicate entries with the same name	name, location (read in JSON file)	Give option to overwrite existing entry of same name

Figure #: Table of Distributed systems tests performed on the Server

-We were testing the connections to other machines

Database testing also included testing the server functionality itself, that being the ability for the server to send and receive messages from other devices. For these tests, the localhost address was used to simplify the process and eliminate issues with connectivity in order to test the functions independently from the rest of the system.

3.2 Hardware Testing

Testing hardware involves running a test program on the arduino to print out the readings from the ultrasonic sensor, and IR sensors to ensure they are reading correctly given different inputs they read. This test program would also actuate the servos and drive the dc motors to physically test that they operate as intended.

Ultrasonic Testing

- Move object towards and sensors and read the distance measurement from the sensor.
- Compare the sensors measurement to a measuring tape/ruler's measurement to confirm the accuracy.

IR Sensor Testing

- Move an object in front of each sensor to ensure that the sensor detects the object.
- If the green LED lights up on the sensor then the sensor works.
- And if the sensor reading that is printed changes then the sensor can be read correctly.

Servo Test

- Sweep the servo from 0 degrees to 180 degrees and back to ensure the servo moves and does not stop at any point

DC Motor Test

- Drive motors forward, backward, and stop them.
- Drive motors at different speeds to ensure proper operation.

3.3 Acceptance Testing

3.3.1 Testing Scenarios

Test Case	Description	Test
Detect Line Position	Testing the algorithm for detecting the lines position	The tests are executed by running a test program that will read and output the line position. It the line was in the middle of the output should be 0. If the lines left or right the lines position was -1 or l respectfully
Detecting Obstacles	Tests to make sure that any obstacle that comes within 10 cm of the front of the LAD mobile unit is seen	This test could be run at any time by placing something in front of the LAD and ensuring the DC motors stopped once an obstacle was detected
Serial Communication	Tests that data sent from both LAD Unit's R pi and Arduino can be sent and received correctly	This can be tested by executed by running a simple python script to send serial data and an arduino program to send back any data received. Test is successful if all data sent is sent back properly..
Decode of serial Commands	Tests that the arduino can decode any command sent to the Arduino	This test can be run anytime by connecting the arduino to your computer and send the arduino commands through the Arduino serial monitor and check that the behaviour of the LAd Mobile Unit is expected.

4.0 Discussion

Our project changed significantly from when it was first proposed. In terms of scope, the finished LAD system could move around a room by tracking a line on the floor and had a functioning android app that sent commands to a server for the LAD unit to drive around, move its arm, or fetch an item. When demonstrated, the server could not reliably communicate with the LAD unit, which prevented it from being controlled directly from the app. Some of the parts of the project from the first proposal that could not be completed were:

- The arm with multiple joints and an articulated claw at the end
- The omnidirectional wheels
- The image recognition system that would detect junctions in the path as well as items

Some aspects could not be properly implemented due to hardware restrictions. The main aspect being the arm. The code could successfully drive the servos and should in theory been able to move the arm but due to the weight of the arm and power of the servos we discovered that the arm was too heavy. This lead to the replacement of the claw with a forklift like design, and the removal of the second joint in the arm. The QR code reading was also removed in the end as we discovered that we could not reliable read QR codes on the floor and that due to the weight limitations on the arm we could not mount a camera on the arm to read QR codes on items.

One challenge we faced once all the sensors and motors were connected to the arduino was powering all the hardware. This was an easy fix and involved adding a battery pack to power the arduino specifically.

Another risk we faced when transporting the LAD mobile unit was damaging the IR sensors as they are mounted very close to the ground. Often times the IR LEDs would be bent after transporting the unit. One IR sensor quit working a week before final demos and resulted in the reduction from 5 sensors to 3 sensors.

5.0 Technical Recommendations

Overall, given the scale of the project we were able to accomplish a lot but ran out of time to properly implement everything into one system.

Some things that could have been done better.

- Meeting every 2 weeks to test functionality of components that need to communicate such as : App to Server, Server to LAD Mobile Unit.
    - More accountability for members who did not perform to group standards or meet deadlines. We had very little in the way of punishment when things did not go according to plan.
- Distributing work more evenly: assign two people to hand the LAD Mobile Unit development. LAD Mobile Unit involved building the hardware, programming the arduino, and programming the Raspberry Pi. if someone was taking care of the LAD Units raspberry pi we could have built the Mobile Unit more efficiently and been able to test much soon.
- Assigning a team lead would have been an improvement instead of each of us defining when we need to be done, if we had one person helping to oversee all and keep things on track would have been beneficial.
- Simplification of the system as a whole. Many parts of the system were too complicated on their own, let alone putting them together.
    - The database could have been simplified. There were many foreign keys, including foreign keys within a table to the same table. This became conceptually very confusing, thus implementation was difficult.
- The arm may not have been a realistic goal. It was mechanically too complex for us to complete on a timely basis.
    - The materials used were also not ideal, as they were too heavy for the servo motors to lift. The original idea was to use LEGO or 3D printed parts to make it lighter and easier to maneuver, but proved too expensive. However, in the future, and with a larger budget, this may have been a more realistic way to implement it.

Appendices

Appendix A - Contributions

Team GitHub Repository Contributions

Git Branch	**Author**
mock	Denise
database-testing	Denise
android-antics	Brannon + Erdem
android	Brannon
camera-testing	Zach + Erdem
Integration	**Author**
Android to Database	Brannon + Erdem + Zach
Database to Mobile Unit	Zach + Denise
Hardware	**Author**
Arm	Zach + Erdem + Brannon
Mobility	Zach + Erdem + Brannon

Final Report Contributions

Section	**Author**
Project Description	Denise
System Architecture	Denise
Hardware Layouts	Zach
Database Architecture	Denise
Android Interfacing	Brannon
Communication Protocols	Denise

Database Unit Testing	Denise
Hardware Testing	Zach
Acceptance Testing	Zach
Discussion	Erdem + Denise
Technical Recommendations	Erdem + Denise
References	Erdem
Appendix A	Denise
Appendix B	Brannon
Appendix C	Brannon

Appendix B - ReadMe

Appendix C - Figures

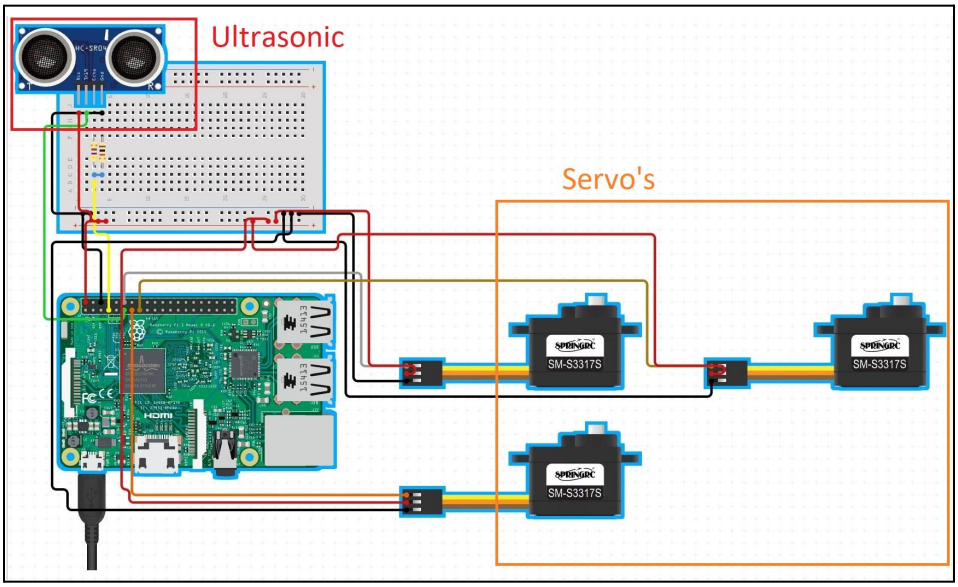


Figure #: A circuit diagram depicting the setup for the Arduino and the hardware necessary for detecting and following the line, along with driving the drive motors

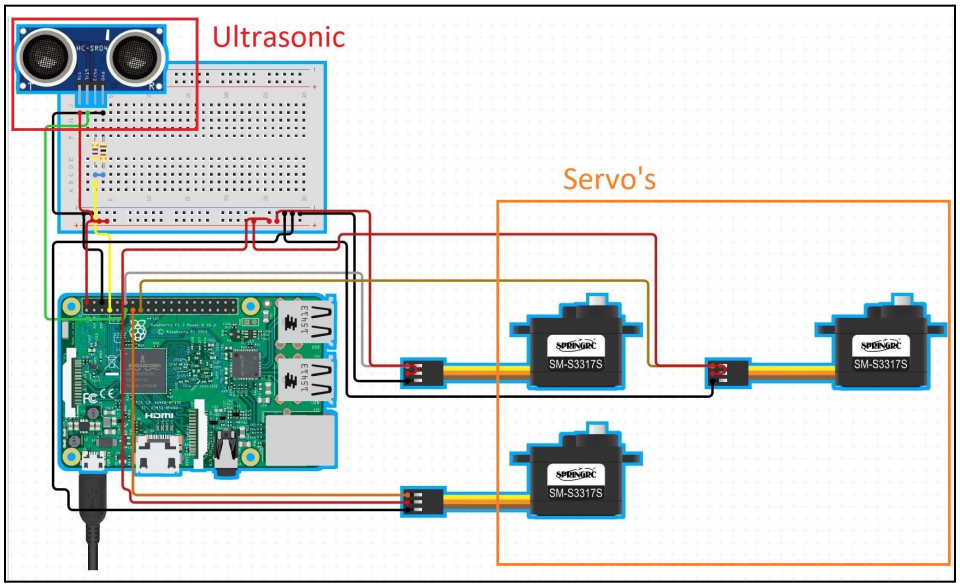
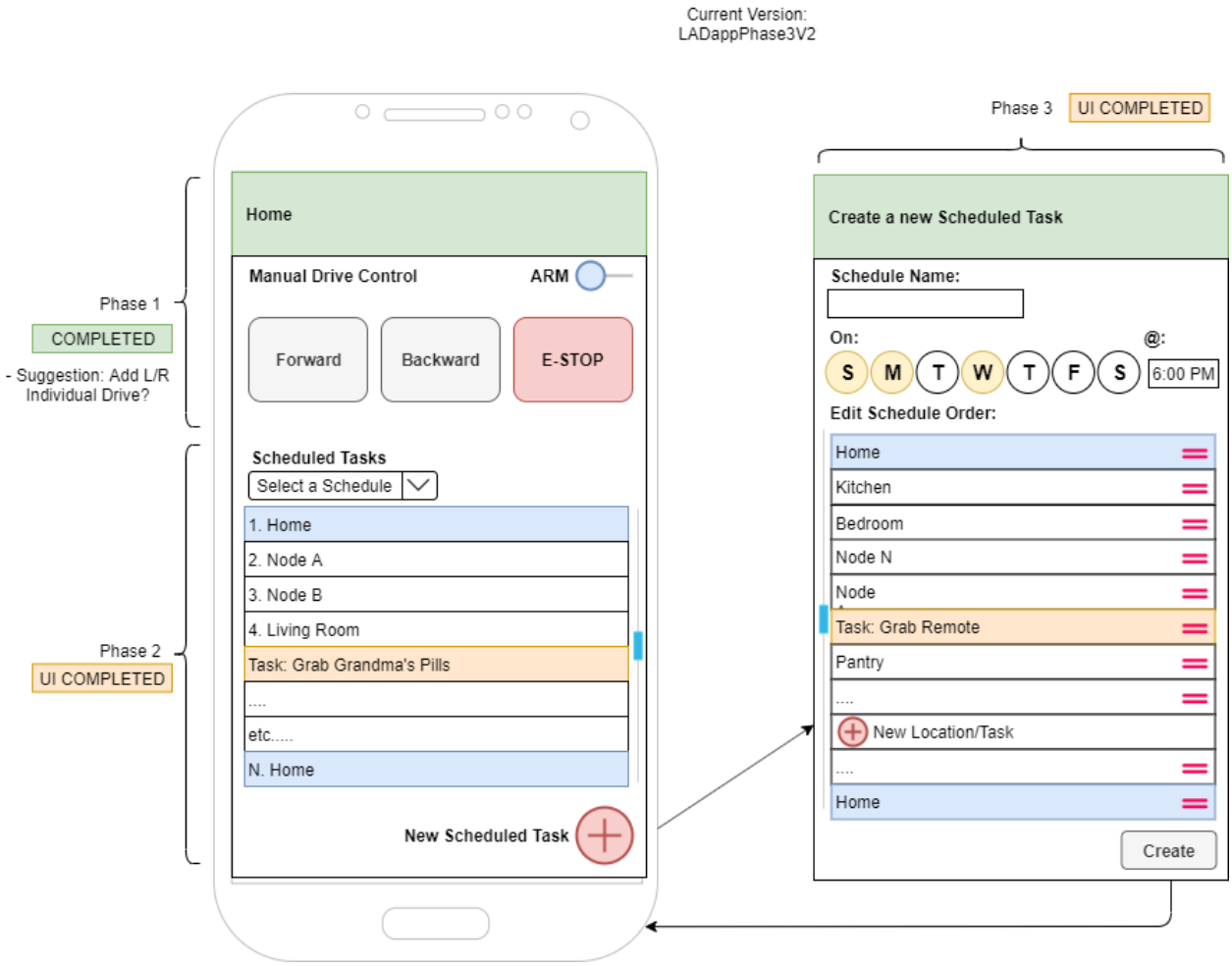


Figure #: A circuit diagram depicting the setup for the Raspberry Pi on the LAD and the hardware necessary for controlling the arm.

Appendix D - Arduino Op Codes

	OP CODES	Hex Value		OP CODES	Hex Value
	OP_MANUAL	0x400			
	OP_AUTOMATIC	0x200			
Manual OP codes	DRV_FORWARD	0x100	Automatic OP codes	LAD_X	0x100
	DRV_TURN_LEFT	0x080		LAD_XX	0x080
	DRV_TURN_RIGHT	0x040		LAD_XXX	0x040
	MOV_ARM_UP	0x020		LAD_SCAN_QR	0x020
	MOV_ARM_DOWN	0x010		LAD_DRIVE	0x010
	MOV_WRIST_UP	0x008		LAD_TURN	0x008
	MOV_WRIST_DOWN	0x004		LAD_DIRECTION	0x004
	MOV_HAND_OPEN	0x002		LAD_MOV_ARM	0x002
	MOV_HAND_CLOSE	0x001		LAD_NO_LINE	0x001

Appendix E - App design



Appendix F - Pictures