ELECTRICAL & COMPUTER ENGINEERING

Project Harvest: Final Report

03-17-2020

Dept. of Electrical & Computer Engineering,
Box 352500, University of Washington Seattle,
WA 98195-2500 U.S.A.

COURSE

ECE 475: Embedded Capstone

LECTURER

Rania Hussein

TEAM MEMBERS

Doruk Arisoy, Nick Huber, J. Brandon Iams, Khang Lee

1. Team, Roles & Responsibilities

Team 7 consists of 4 team members from various technical backgrounds. The description of our expertise and role within the team is specified below. The roles described simply represent the work that we had spent the most time on individually; we were overall involved in each other's work of the process as a team.

Doruk Arisoy

Expertise — Full Stack Software Development
Role — Raspberry Pi Communication, Computer Vision, Inverse Kinematics

Nick Huber

Expertise — Linux OS, Back-End Software Development, Robotics, System Administration
Role — Raspberry Pi Communication, Stepper Controller Development

J. Brandon Iams

Expertise — PCB Circuit Design, Additive Manufacturing, I2C Protocol, Logic Level Shifting, Peripheral Devices.
Role — PCB Development, Robotic Arm Development, Stepper Controller Development

Khang Lee

Expertise — Front-End Development, Technical Communication
Role —Stepper Controller Development, Front-End Development, Documentation

2. Product Requirements

The motivation behind this capstone project is to tackle the regression in supply of agricultural goods against the increasing demand for food. According to the World Economic Forum, we will be facing a 214 trillion calorie deficit by 2027 [1]. One of the main problems of agriculture is that many people are leaving the industry for better quality of life, causing labor shortages that are aggravating every year [2]. This led to a growth in the development of agricultural technologies (AgriTech) to automate the farming process, which is an industry that is growing tremendously — AgriTech startups saw $16.9 Billion investments in 2018 alone which is a whopping 43% increase year-over-year [3].

With our motivations in mind, we decided to tackle a smaller scope of autonomous farming, specifically on harvesting fruits on trees. We crafted our project statement to align every team member with a single north star vision:

> “To create a robotic arm that can autonomously detect, orient, and harvest fruits through machine learning.”

In other words, the goal of this project is to create a robotic arm that can be controlled through an interface to automatically harvest fruits using computer vision. For the scope of this class, we plan to focus on strawberry specifically but we intend to include more fruits if given more time.

3. Impact & Consequences

On top of addressing the aggravating issue of food shortage and the dire demand of farm labor, we aim to challenge the current methods in which farming technologies have been built. There are actually many autonomous machines already created for agricultural purposes. The Robocrop for example is a $700,000 work-in-progress that can currently pick 25,000 raspberries a day [4]. The Agrobot and the Harvest CROO are gigantic strawberry-picking robots that require specific farming layouts to operate efficiently [5] [6]. Many of these implementations are highly specialized and are tied to specific farming structures and hence can be very expensive.

For this project, we aim to utilize a commonly used, versatile component — a 6-axis robot arm which may ultimately cut down costs. We will be attaching a 3d-capable camera to the robot arm and implement a machine learning framework to help detect fruits. In a specialized autonomous machine, farmers need to follow strict farming layouts in order for the machine to accurately find and harvest fruits. By using a camera instead, we can easily detect fruits in 3-dimensional space with the flexibility of the robotic arm. With a focus of cheaper components, this project aims to make such precision tools more accessible for farmers.

Additionally, the objective of this project is not to be the first team to create this technology, but to understand the problems and constraints that can be faced in the novel industry of AgriTech so that we can apply our newfound experiences in our future (potentially commercial) attempts in building this machine.

4. Product Constraints & Engineering Standards

The project vision is definitely ambitious given that we only have 10 weeks to design and develop the entire system. To ensure that we achieve realistic success, we have decided to create a prototypical harvester that is not built to the scale in which it may be on an actual farm. This will eliminate the issues of mechanical implementation that many of us are not experienced enough to deal with. Moreover, The pruning mechanics will not be within the scope of the project; given the time constraint, we will at most incorporate a simplistic grabber onto the system. Other than that, this prototype will simply be in charge of detecting the target object and orienting towards it.

In terms of meeting engineering standards, there are several International Electrotechnical Commission (IEC) standards that pertain to our project, namely:

1. IEC 61140 Protection Against Electric Shock
2. IEC 60529 Protection by Enclosures
3. IEC 60664 Insulation Coordination for Equipment within Low Voltage Systems
4. IEC 61000 Electromagnetic Compatibility

There is also the TIA-485 Serial Communication Protocol which is the main communication protocol of our system. This facilitates communication between the Pi and the arm.

5. System Requirements

As a clearer description of the system, we will create a 6-axis robotic arm using stepper motors and servos. This section provides a description on the technical details of the system by breaking it down into the Robotic Arm and the Raspberry Pi. Figure 1 shows a simple sequencing chart of the basic functions that will be supported on our system including the initialization process, automatic harvesting process, and the manual arm control process.

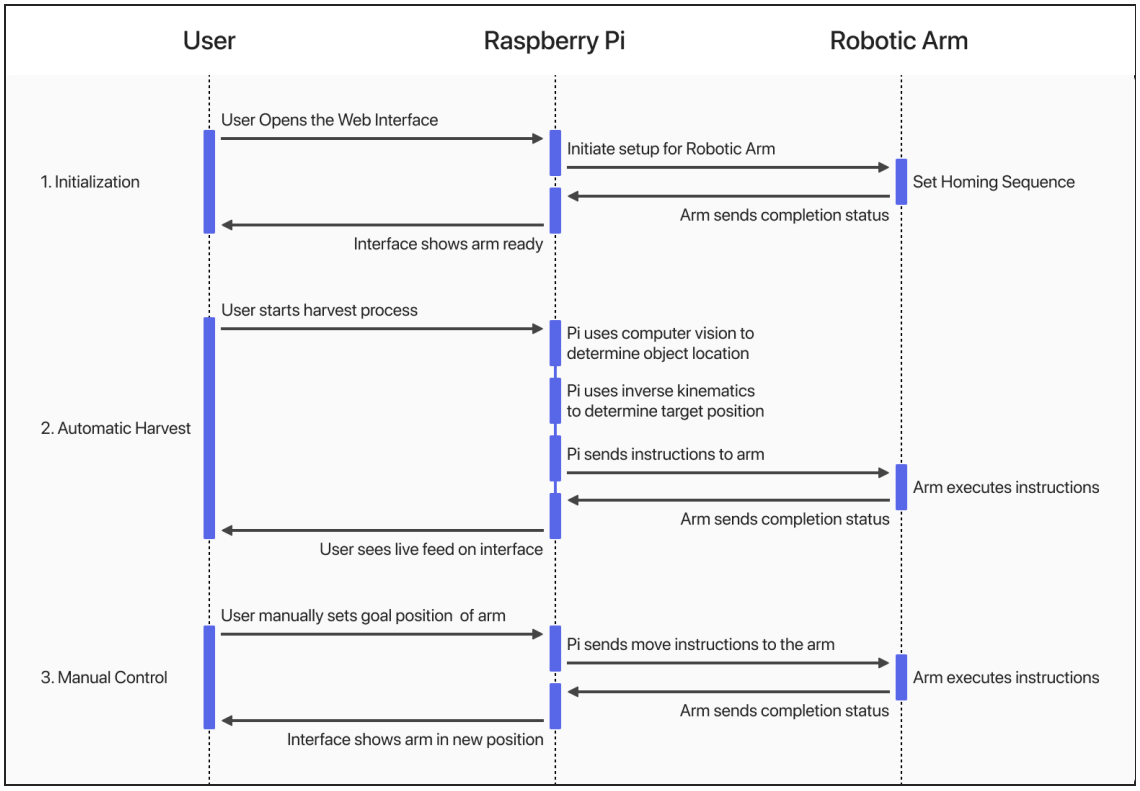

Figure 1: Sequence Chart

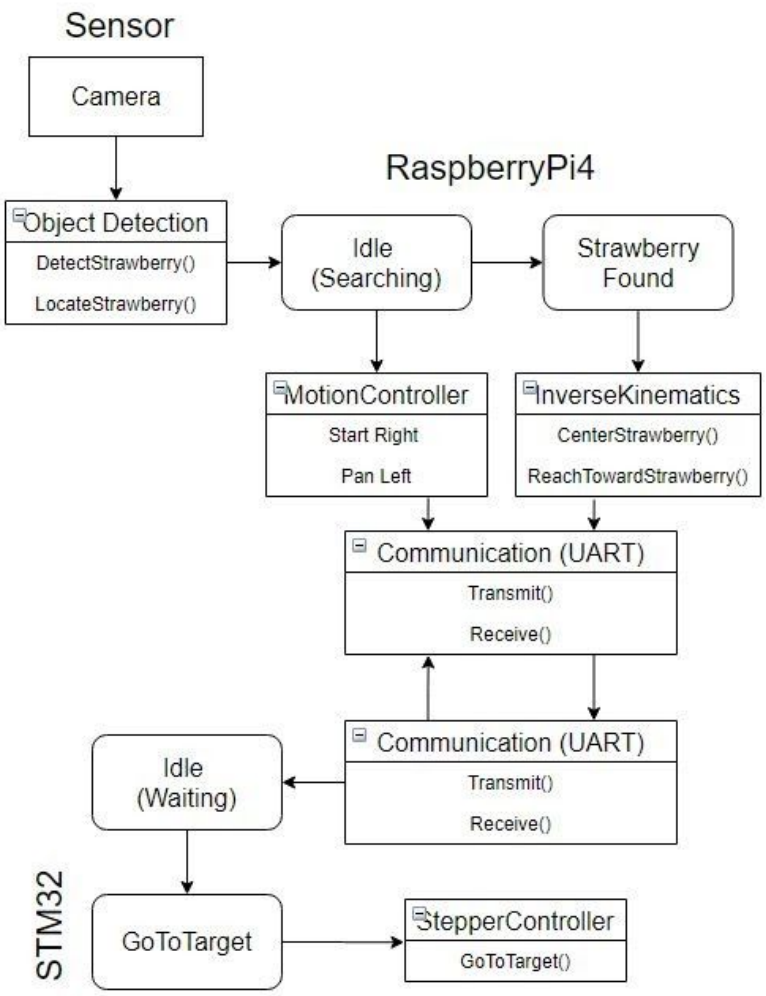

Figure 2: System Level UML

HARDWARE

This section provides a detailed breakdown on the hardware requirements of the system. We first discuss the components used in the robotic arm then the peripherals for the Raspberry Pi.

Robotic Arm

The 6-axis robotic arm was built by using the guidelines provided by the open source Niryo One Robotic Arm [last]. The guideline provided the necessary 3D-models for us to fabricate the robot frame using the material PLA on a 3D Printer from scratch and buy all necessary components including stepper motors, servos, gears, etc. The mass of the robotic arm sums up to about 3.3 kg with a maximum reach of 44 cm. In terms of the components

within the robotic arm, Figure 3 shows a detailed breakdown on all the hardware integrated within the arm. Note that all components share the same ground plane.

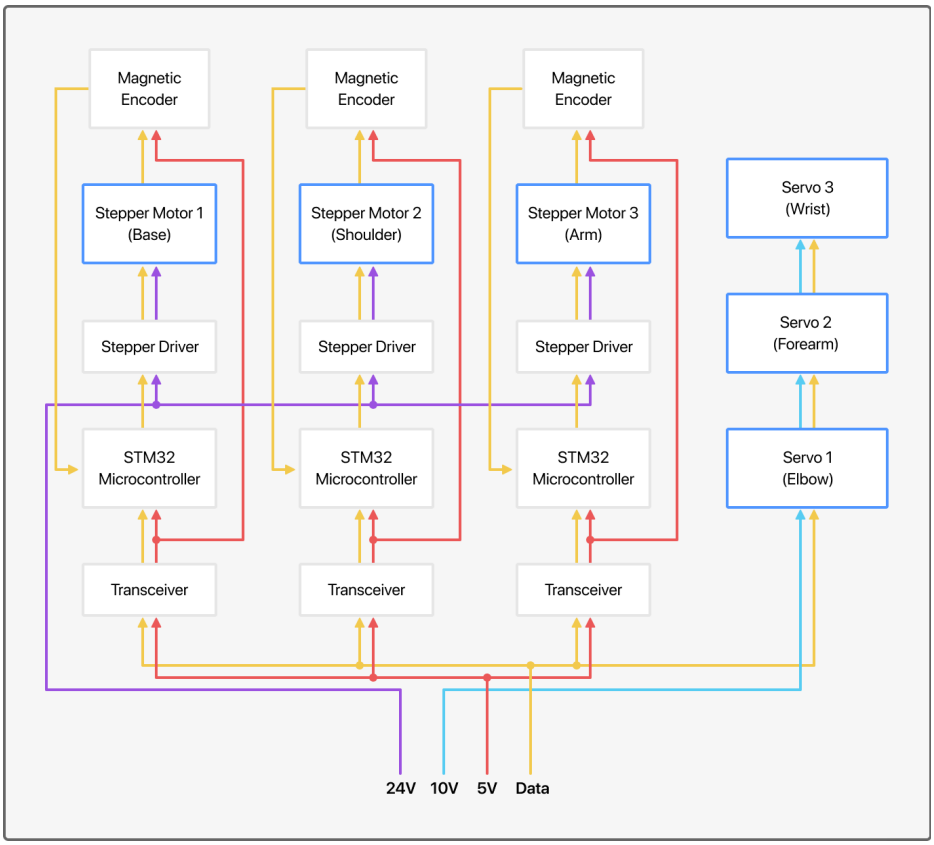

Figure 3: Robotic Arm Hardware Diagram

The main turning components that give the arm its six degrees of freedom are the seros and motors as highlighted in blue in Figure 1. The servos are used near the ends of the robotic arm to provide high precision control on the more delicate parts of the arm. We decided to use the Dynamixel servos that was provided by one of our team members. These servos require a 10V DC supply, have a built in microcontroller to receive incoming instructions, a PID controller to set different kinematic preferences, and a magnetic encoder to provide a feedback loop to the system. The Dynamixels are addressable and hence can be daisy-chained to ease the wiring process.

The stepper motors are used as the bases of the robotic arm to provide control on the heavier parts of the arm. The steppers used for this project require a 24V DC supply to take on a heavier load. To ensure a fluent and cohesive communication between the Pi and the robotic arm, we have built controller boards on each of the stepper motor. Each controller board contains a STM32F103C8T6 Microcontroller (also called the Blue Pill) to interpret the

incoming instructions, a DRV8825 High Current Driver to drive the stepper motors, a AS5047D High Speed Position Sensor (also called a Magnetic Encoder) for precision sensing, and a THVD1410DR Transceiver to moderate communication. The Blue Pill will be programmed to be addressable and to conform with the communication protocol of the Dynamixels so that we can simply daisy chain all of the steppers and servos into a single cluster of data and power lines.

Raspberry Pi

The Raspberry Pi will act as the central control for the entire system. This system will be interfaced using a desktop monitor and a conventional keyboard and mouse. We aim to create a circuit board that will attach directly above the Pi. This hat will have a bus transceiver to mediate the TIA-485 half-duplex communication protocol and two buck converters (a 24V to 10V and a 24V to 5V) to supply the right voltages to various components within the robotic arm. This will lead to a single bundle of wires going between the Pi and the arm. Additionally, a stereographic camera, specifically the Intel RealSense Depth Camera D435i, will be wired to the Pi has the vision of the system to help detect strawberries. The camera will be able to provide the relative position of the strawberries in three-dimension space which will help the robotic arm orient itself towards a target object. The camera will be mounted at the end of the robotic arm, also called the wrist of the arm. Figure 4 shows a detailed breakdown on all the hardware integrated on the Pi. Note that all components share the same ground plane.

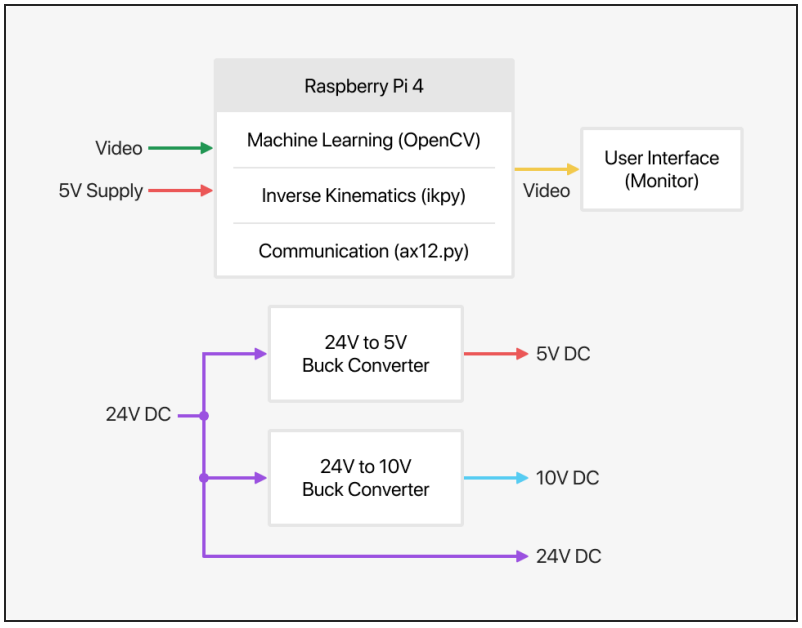

Figure 4: Raspberry Pi Hardware Diagram

SOFTWARE

In terms of the software on the Pi, the operating system (OS) used on our Pi is the default OS provide by the Raspberry Pi Organization — the Raspbian Buster [7]. For the backend aspect of our system, we are using OpenCV [8] and the RealSense [9] Python Library to integrate a machine learning framework to interpret the input coming from the camera. To convert a target coordinate from the camera to actual angles on the robotic arm, we utilize an inverse kinematics library ikpy [10] in our system to calculate the fastest route to move from one position to the next on the 6 different axes of the arm. The angles of configuration along with the statuses of the system are outputted into JSON files saved within in the Frontend directory for the web interface to access. To communicate the desired position to the robotic arm, we have complied with the Robotis Protocol 2.0 [11] which is the proprietary protocol used by the Dynamixels Servos on the arm. This will be our main way of communicating with the arm with the stepper controllers programmed to receive the same kind of information.

For the frontend of the system, we build a simple web platform that is run locally on simple Node.js server. The frontend is built using conventional HTML, CSS, and Javascript. The server, built using the Express.js [12] framework, simply receives the GET/POST HTTP request from the frontend and outputs any necessary instructions in JSON format so the Python backend to parse.

6. Project Schedule

Figure 5 shows our overall timeline of the project schedule in a Gantt Chart format. The grey areas show tasks that we've completed and the purple area are tasks that are incomplete (in this case, the failed implementation of ROS). In terms of our progress throughout the project, we were able to meet our deadlines throughout each phase and even get a head start on the next phase.

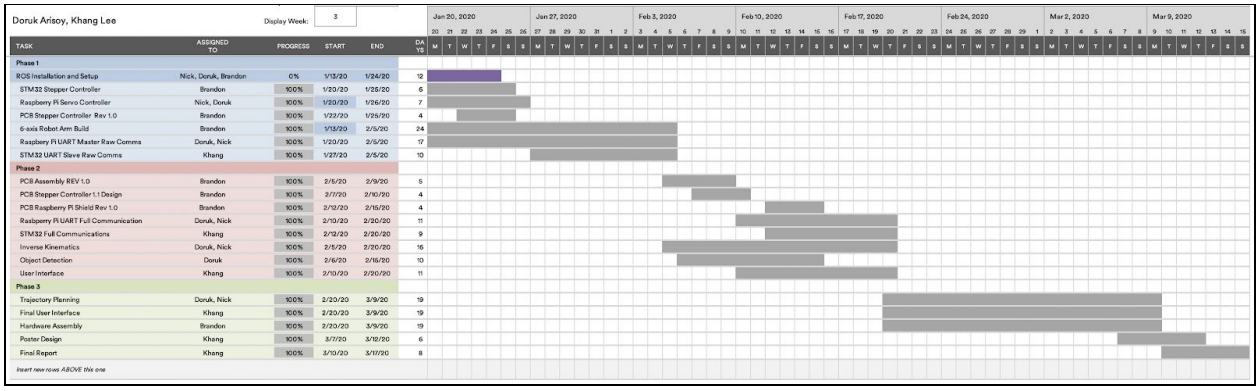

Figure 5: Project Gantt Chart

7. Project Resources

The physical components used in this project are internally funded by one of our team members with the agreement that the components will remain under his ownership after the project is completed. The main working space for the team is in the designated class laboratory CSE 003E. All working components are left within the lab so that it is accessible to the team members; if someone intends to move something, they would need to inform the entire team. The software and programming frameworks used in this project are all either free for students or open source.

The following is a comprehensive list of resources:
Tools:

Soldering iron/reflow station	Circuit board holder	Wire Crimpers
Wire Strippers	Directional Cutters	Allen Wrench Set
Tweezers	½ inch wrench	Screw Driver Set
3D Printer	Drill	Digital Microscope

Consumables:

Flux	Solder	Heat Shrink
Cable Harness Wrap	Zip Ties	Allen Screws
Cable Lacing	20 AWG Wire	22 AWG Wire
PLA Fillament	Suction Cups	Bolts/Nuts

Electronic Components for Stepper Controller PCB (BOM)

Manufacturer Part Number	Qty	Unit Price	Extended Price	Description
DRV8825PWPR	3	4.24	$12.72	IC MTR DRVR BIPLR 8.2-45V 28SSOP
AS5047D-ATST	2	8.43	$16.86	ROTARY ENCODER MAGNETIC PROG
STM32F103C8T6	3	4.58	$13.74	IC MCU 32BIT 64KB FLASH 48LQFP
MCP1802T-3302I/OT	3	0.53	$1.59	IC REG LINEAR 3.3V 300MA SOT23-5
TC33X-2-103E	10	0.277	$2.77	TRIMMER 10K OHM 0.1W J LEAD TOP
THVD1410DR	4	2.57	$10.28	TRANSCEIVER
T2N7002AK,LM	10	0.131	$1.31	MOSFET N-CH 60V 0.2A
S4B-EH(LF)(SN)	10	0.177	$1.77	CONN HEADER R/A 4POS 2.5MM
S10B-PHDSS(LF)(SN)	3	0.47	$1.41	CONN HEADER R/A 10POS 2MM
NX3225GD-8MHZ-STD-CRA-3	3	0.94	$2.82	CRYSTAL 8.0000MHZ 8PF SMD
ERJ-2BWFR100X	20	0.343	$6.86	RES 0.1 OHM 1% 1/4W 0402
ERJ-2GEJ101X	100	0.0128	$1.28	RES SMD 100 OHM 5% 1/10W 0402
RK73B1ETTP200J	100	0.0065	$0.65	RES 20 OHM 5% 1/10W 0402
RC0402FR-07100KL	100	0.0064	$0.64	RES SMD 100K OHM 1% 1/16W 0402
CC0402JRNPO9BN200	100	0.0162	$1.62	CAP CER 20PF 50V C0G/NPO 0402
GRM155R62A104KE14D	100	0.0143	$1.43	CAP CER 0.1UF 100V X5R 0402
GRT155R61E105KE01D	20	0.123	$2.46	CAP CER 1UF 25V X5R 0402
5040500491	4	1.3	$5.20	CONN HEADER SMD R/A 4POS 1.5MM
CL10B474KO8NNNC	100	0.0287	$2.87	CAP CER 0.47UF 16V X7R 0603
C0603C104Z3VACTU	100	0.0143	$1.43	CAP CER 0.1UF 25V Y5V 0603
CL10A105KB8NNNC	100	0.0568	$5.68	CAP CER 1UF 50V X5R 0603
8.85012E+11	100	0.0169	$1.69	CAP CER 10000PF 50V X7R 0603
RC0402FR-0710KL	100	0.0064	$0.64	RES SMD 10K OHM 1% 1/16W 0402

Devices

Intel RealSense D435i Camera	Raspberry Pi 4	STM32 Microcontroller
NEMA 17 Stepper Motor 3x	Dynamixel XL430 2x	Dynamixel XL320 2x

8. Outline of Experiments

The experimental development process for our project is simply incorporating various pre-existing frameworks into the design of our system. This helps rapidly develop the design considering most of the design has already been proven. This concept is not foreign to embedded system design. The pieces of the design that already exist do so within industry standards. For communications, universal asynchronous receiver-transmitter (UART) is used, specifically, the TIA-485 communication protocol. This broadly accepted form of communication is effective as it can have an adjustable speed and asynchronous communication. Another element that is based on standards is SPI communication. The magnetic encoder uses Serial Peripheral Interface (SPI) to communicate with the microcontroller. While this is still being developed, it should help aid the development process as it is a well documented communication standard.

COMMUNICATION

While many aspects of our design focus on standards, other components are purely experimental. While the communication protocol is of standard form, the packets of data in which it transmits are unique to the Dynamixel servo. Since we are using these servos in our design, it seemed best to put the three stepper controllers on the same communication path. This keeps the cabling to a bare minimum and offers better abstraction for the system level interface. It also means the stepper motors are limited to the same packet structure as the servos. This presents the challenge of having to create a communication path that is robust to limit failure modes but still adheres to a minimally documented packet structure.

PCB DESIGN

Another experimental design component is the PCB design. The game with PCB design is getting to the final design quickly which means reiteration. It is almost impossible to create a perfect design on the first try. This means manufacturing PCB's early to find failures and reiterate. In rapid development, it is best to manufacture sooner than later to make room for revisions.

9. Trial Designs

The largest setback for the design was Robot Operating System (ROS). This software framework is very standard amongst robotic design. As such, it seemed beneficial to incorporate its built-in inverse kinematic library to create route planning and motor control. The problem was that ROS does not yet support the Raspberry Pi 4. Despite this, our commitment to its use meant finding a way of implementing it anyways. After two weeks of trials, the team agreed to cut the loss and make progress in a different direction. This is an example of the dynamic nature of design.

The communication pathway had to be built in slow steps with continuous testing throughout. Since the Dynamixels are already established, the communication from the Pi as a master device was built first. Afterwards, the packet handler on the STM32 microcontroller was implemented. Since the STM32 is a slave device, it was important to implement the UART communication with interrupts. In order to keep the interrupt service routine short, the UART receives one character and pushes it into a circular queue data structure. Every character is put into this data structure and when the ISR is not executing, the packet handler deQueues until the packet signature is found (0xFFFFFD). It then follows the packet structure to determine the command being sent over UART. While this works, there is a significant amount of lag that causes commands to be delayed. This bug will be investigated further through Phase 2.

In order to facilitate performance and expand capabilities, the stepper controller with STM32 microcontroller is assembled on a custom PCB. The small PCB design is mounted on the back of the motor with only voltages and data coming into and leaving the stepper controller. The PCB contains the STM32 microcontroller, stepper driver IC, tri-state buffer, and magnetic encoder. The PCB also allows the magnetic encoder to be used as it must be local to the stepper motor. The magnetic encoder uses hall effect sensors to detect the rotation of the stepper driver and serves as a feedback loop. When the stepper controller is given the command to move 30 degrees, the magnetic encoder can monitor missed steps and make them up. This is important to the accuracy of the 6-axis arms as they must move with precision.

The following schematic (Figures 6 and 7) is the custom PCB with only necessary components to meet the design's purpose. In creating a custom PCB, a lean design with small form and focused function is achieved. If this design were to scale to market, it would create a major cost savings.

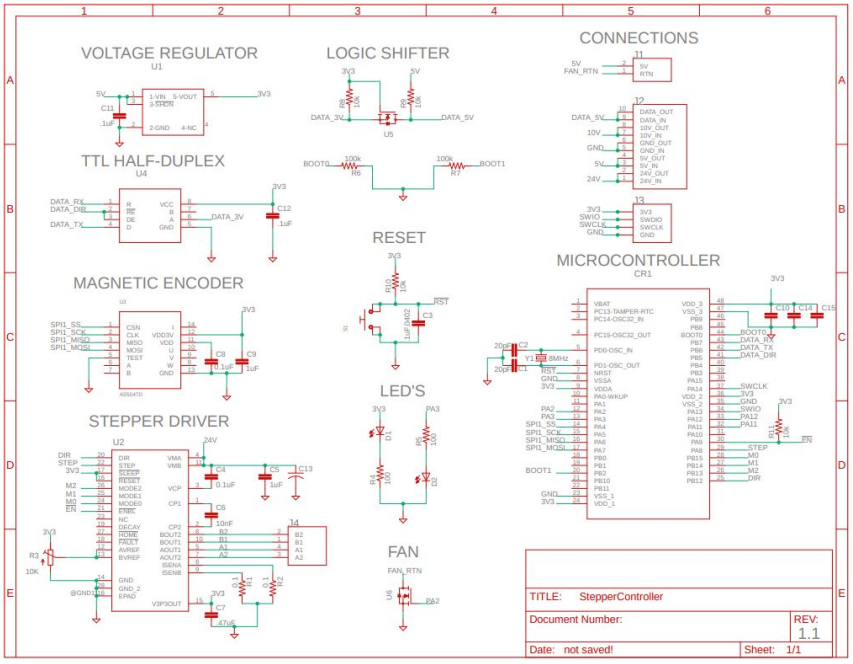

Figure 6: Stepper Controller Schematic

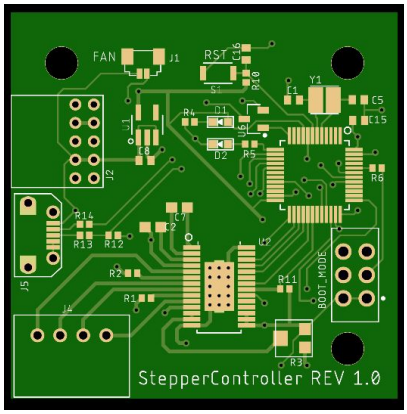

Figure 7: Stepper Controller PCB Front/Back Rev 1.0

After receiving the custom PCB's, it was discovered that the connection to form uplink was too small of a connector. At this point, it became necessary to redesign the PCB with a different connector. The opportunity to redesign also allowed for other improvements. These improvements include increasing current carrying capacity for the high current outputs to the stepper motor. It also meant removing the boot mode rail and USB connection. Since these two components were unnecessary and were only implemented for feature enhancements, their removal meant REV 1.1 could have a much needed electrolytic capacitor used to help drive the stepper motors.

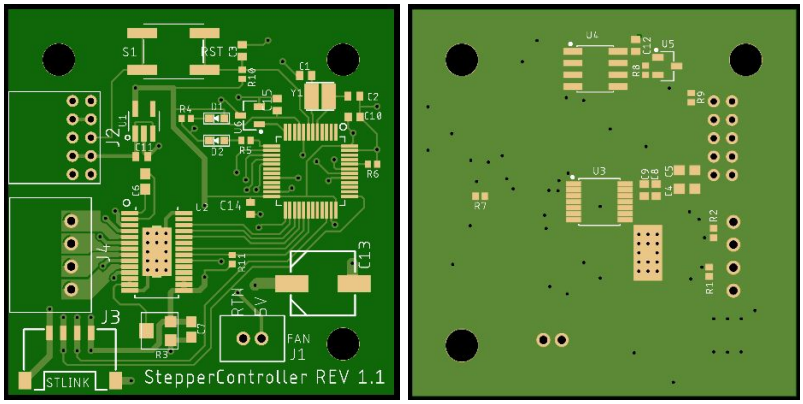

Figure 8: Stepper Controller PCB Rev 1.1

While the changes seem small, they were enough to merit a second design. It can be seen that J3 is a significantly larger connector with the pads stitched with via's. This PCB design technique offers increased mechanical strength.

Another design under trial is the mounting of the camera. While most of the 6-axis arm is open sourced and 3D printed from the original CAD design, it does not account for the stereoscopic camera used in our design. As a result, the gripper that will eventually become the pruner had to be redesigned to accommodate the camera. The initial design must be determined through testing. This design will be 3D printed and tested during phase 2 to ensure the positioning will suffice.

Figure 9: Intel Realsense camera mount design

10. Experimental Outcomes

After many trials and errors, we were able to develop a fully functioning prototype. Figure 10 shows the fully developed prototype with all the components, including the stepper controller boards and the camera, installed. We will go through the results of each of the stages.

Figure 10: Final Prototype

CAMERA VISION

The computer vision system on the robot utilizes two main methods of detecting objects. The first way is using a machine learning based object classifier called OpenVino. This allows us to detect complex objects such as people. The Pi is not powerful enough to run this model at an acceptable framerate, so we offload the processing to a Movidius Compute Stick, which accelerates the neural network processing. Our second method is much more generic and allows us to detect simpler objects such as strawberries using only computer vision and image signal processing techniques. This method is much more reliable and faster than using a neural network in practice. The technique we utilize for detecting strawberries consists of a series of filters. The first step is to take the image provided by the Intel RealSense camera and apply a HSV (Hue-Saturation-Value) filter to it such that pixels in the image which match the color of a strawberry are highlighted. Once we have this binary image mask, we can turn that into a series of contours. These contours are the

edges of the mask, where the image transitions from a pixel that is the correct color to a pixel that is not. This is known as a Sobel operator, or edge detection. Next, we smooth out these contours by making them all convex, this closes all contours to make them more comparable. Finally, we filter these contours such that only contours which represent a shape that is large enough and is of the correct aspect ratio is accepted. This filters out small noise and objects which are the wrong shape. This leaves us with contours which are only strawberries. This style of object detection requires hand tuning, but is, in our experience, faster and more accurate than any prebuilt neural network based models that exist.

PI-ARM COMMUNICATION

We managed to create a Python program with sufficient abstraction between an instruction and the resulting data packets that are sent to the arm. Signal integrity is guaranteed through the use of the oscilloscope. By monitoring the UART half-duplex communication signal, it was discovered that LC noise was being absorbed by the data path and small spikes compromised the UART signal. This is most likely attributed to multiple voltage sources and determination of this hypothesis will be further tested through phase 2. Ultimately, the implementation of the PCB's for single source DC power and robust data transmission mitigated this error.

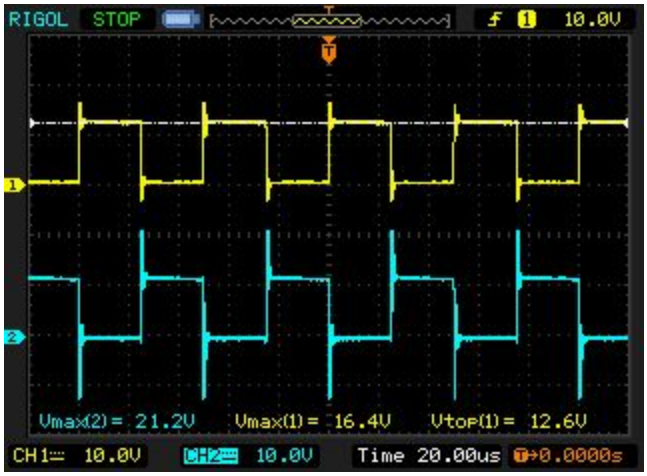

Figure 11: Signal noise on UART communication

INTERFACE

Figure 12 shows the resulting interface. The left top window was initially intended to show a 3D model of the robotic arm. However due to time constraints, we were unable to do so. The top right window shows the live feed of the camera which is done by continuously

loading the snapshot from the camera saved from the backend. The bottom left section shows the live angle configurations of each joint; this is updated constantly by constantly loading a local JSON file. The buttons on the right side of the interface are the main controls of the system and it works by sending POST HTTP Requests to the backend. Lastly, the messages at the bottom right show the current status on each section of the system by also constantly loading a local JSON file.

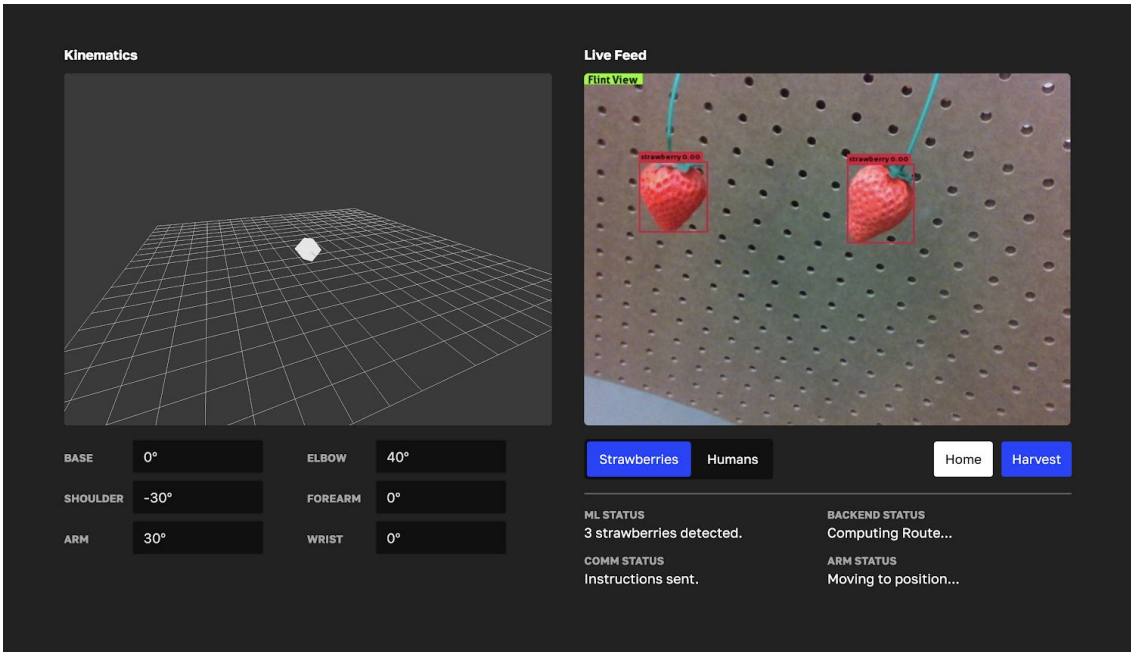

Figure 12: Frontend Interface

11. Conclusion & Recommendations

Through this project, we were able to identify what steps we needed to take to deliver the project we initially proposed. When we first proposed the autonomous fruit picking robot arm project, we were planning on using the Robot Operating System (ROS) to handle the inverse kinematics and communications of the robot arm. However, after coming across multiple roadblocks with ROS and time constraints, we have decided to implement our own communications channel along with a different inverse kinematics library that does not require the use of ROS framework. These changes got us back on track and by the end of this capstone project, we were able to deliver a robotic arm that functions on the Raspberry Pi and can successfully pick strawberries autonomously.

During this project, we were able to learn the valuable skill of reverse-engineering complex communication protocols and simulate it in a different programming language. We learned how to use open source software and libraries to build our own product and also contribute to the open source community. We also learnt how to intercept various signals going to and from the Pi using the oscilloscope to diagnose our issues. Finally, we got a better understanding of robotics and motion planning works along with machine learning integrated computer vision.

In terms of functionality of our robot arm, we have reached the goal we have set since the initial project proposal. Like every project, there is room for improvement. While we improved the communications between the Raspberry PI and the motors on the arm significantly and implemented inverse kinematics to help guide the robot to a strawberry, we assumed that there would not be any objects blocking the path of the robot. Implementing collision avoidance to the route planning of the robot arm is a difficult task that can be implemented if this robot arm were to become a consumer product. At the end of our project, our robot arm was able to detect strawberries and it was able to grab them.

Along the course of this project we not only demonstrated everything we have learned so far in school but we also learned a lot as we came across various different challenges. We believe that with robots like the one we have designed and built, we can revolutionize the agriculture industry.

12. Appendix

REFERENCES

1. "In 10 years, the world may not be able to feed itself," *World Economic Forum*. [Online]. Available:https://www.weforum.org/agenda/2017/09/in-10-years-the-world-may-not-be-able-to-feed-itself/. [Accessed: 14-Jan-2020].
2. "Agriculture: How immigration plays a critical role," *New American Economy*. [Online]. Available: https://www.newamericaneconomy.org/issues/agriculture/. [Accessed: 14-Jan-2020].
3. "AgFunder AgriFood Tech Investing Report - 2018 | AgFunder." [Online]. Available: https://agfunder.com/research/agrifood-tech-investing-report-2018/. [Accessed: 14-Jan-2020].
4. J. Kollewe and R. Davies, "Robocrop: world's first raspberry-picking robot set to work," *The Guardian*, 26-May-2019.
5. "About – Harvest Croo." [Online]. Available: https://harvestcroo.com/about/. [Accessed: 14-Jan-2020].
6. "HOME," *agrobot*. [Online]. Available: https://www.agrobot.com. [Accessed: 14-Jan-2020].

OPEN SOURCE FRAMEWORKS

7. Raspbian OS. Available: https://www.raspberrypi.org/downloads/raspbian/ [Accessed: 4-Feb-2020].
8. OpenCV Repository. Available: https://github.com/opencv/opencv [Accessed: 01-Mar-2020].
9. Intel Realsense Repository. Available: https://github.com/IntelRealSense/librealsense [Accessed: 01-Mar-2020].
10. Ikpy Repository. Available: https://github.com/Phylliade/ikpy [Accessed: 01-Mar-2020].
11. Robotis Protocol 2.0. http://emanual.robotis.com/docs/en/dxl/protocol2/ [Accessed: 01-Mar-2020].
12. Express.js Repository. https://github.com/expressjs/express [Accessed: 01-Mar-2020].

INTELLECTUAL PROPERTY

13. Niryo One Repository. Available: https://github.com/NiryoRobotics/niryo_one [Accessed: 10-Feb-2020].