Smart Trash Sorting

Tristen Miller, Nadia Shabbar, Aadhav Sivakumar

June 2024

Contents

1 Introduction

1.1 Acknowledgements

Author of this section: Nadia Shabbar

This project was made possible thanks to the teaching staff, BELS, and Professor Tae Myung Huh who allowed us the use of his Franka Emika Research 3 Robot arm that was used within this project. All writers and members of this team contributed equally to the paper and project, and were listed in alphabetical order.

1.2 Problem Statement

Author of this section: Nadia Shabbar

Compost occupies landfills at an unprecedented rate due to contamination entering the compost stream and making it unfit for composting on a large scale level. The purpose of this project is to address the concerns of removing contaminants from the compost stream to allow compost-able material to be compost-able.

1.3 Background

Author of this section: Nadia Shabbar

According to a study conducted at Indiana University, it is estimated that in the United States, 268 million tons of waste has been produced in 2017[4]. While some of these materials are recyclable and avenues are being taken to reduce the number of recyclables in landfills, another neglected component of trash that can be removed from the landfill stream is compost. The United States Environmental Protection Agency estimates that roughly 24.1 % of trash is actually composed of food; and "when yard trimmings, wood, and paper/paperboard" are added, this number can reach as high as 51.4 % [2].

1.4 Client Summary and Information

Author of this section: Nadia Shabbar

At the University of California, Santa Cruz, Chris Leverenz, our stakeholder for this project and the head of waste management department for the campus, notified the team of how the green bins spread throughout the campus suffer

from too much contamination leading his team having to resort to throwing compost-able material into the landfills. Due to his desire to see this dilemma rectified, he wanted to see a device constructed that can sort contaminants out of the compost stream utilizing a conveyor belt and a robotic arm.

1.5 Team Labor Division

Author of this section: Nadia Shabbar

To properly divide the labor of work, the team broke the system into four major subsystems: Computer Vision, Robotic Arm, Power, and Micro-controller with Sensor suite. The labor was then divided amongst the three team members according to skill and experience.

With Tristen Miller being the most software savvy member, it became a natural choice to have him work on the computer vision. This was also compounded with his general interest to learn how a computer vision system worked. For the robotic arm, Aadhav was assigned to this section of the project as he was taking the graduate level robotic arm manipulation class and would have the knowledge to make it work. In addition to his knowledge, he was able to secure the robotic arm via connections with his research lab in which the robot arm was used. For the remaining power and micro-controller systems, Nadia was assigned to this task as her general interests were with electrical engineering and sensor interfacing. In addition, she has knowledge in working with micro-controllers.

Other labor division was broken up to managerial roles such as Team Facilitator, Devil's Advocate, Financial Lead, Communications Lead, and Documentations Lead. For both the Team Facilitator and Devil's Advocate, the roll rotated every sprint so as to allow each team member the chance to learn how to lead and interface with the management software, or to exercise their ability to question the work being done in a meaningful manner.

Communications lead was assigned to Aadhav as he expressed interest in seeking out and conversing with the stakeholders and other relevant parties. Tristen was made the financial lead as he is great with organizing the ongoings of the team and making sure the team was not over budget. Nadia was assigned as documentations lead as she had relevant experience in industry with documentation format and standards.

1.6 Project Management Tools

Author of this section: Nadia Shabbar

In order for the team to conduct business in an orderly manner, management software tools such as Jira were used. The purpose of Jira was to regulate sprint planning and sprint tasking, with sprints beginning on every Monday and ending on every Friday in one week intervals. Monday daily stand ups were reserved for planning, in which tasks would be delegated to team members; Wednesday stand ups were reserved for reviews, in which tasks would be looked over and

updated with progress; and Friday stand ups were reserved for retrospectives, in which the team could discuss issues that came up and resolve them or to praise a team member on work done.

Within Jira, the team was able to take all tasks within a sprint and in the backlog to then organize within the Gantt Chart under "Epics" which were then linked together and displayed the team's "Critical Path"; the "Critical Path" being the overarching timeline of tasks that blocked other tasks, but also measuring major milestones for the team. Figure 1 shows an example of the teams Gantt Chart with the Critical Path highlighted in red.

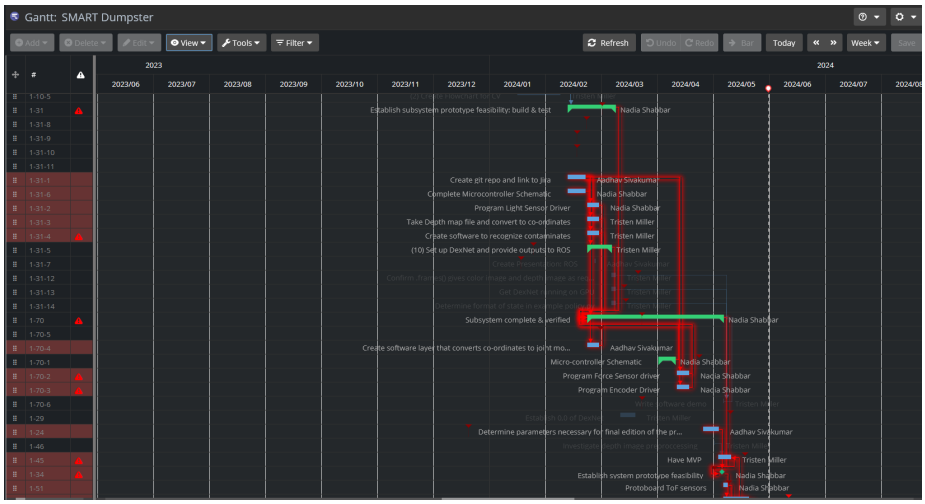


Figure 1: Gantt Chart in Jira

2 Design Overview

Author of this section: Nadia Shabbar

The overall design of this project came down to initially speaking with the client and other relevant stakeholders to gather a repository of requirements our design would need to follow. Such requirements were: Must accurately detect contaminants in the compost stream up to 99% of the time, must be able to work with a conveyor belt, must notify the user when the bins are full and bins do not exceed 50 lb weight limit, must have safety features in case a person is near machinery.

2.1 Concept of Operations

Author of this section: Nadia Shabbar

With the requirements in hand, the team was able to flesh out a concept of

operations their system could follow. Figure 2 shows the general flow of how the system would interface with a user, and how the system operates. The system begins with a boot up which checks if all initialization returns true. In the event that a component fails to initialize, an error is thrown in which the user follows steps to reinitialize the system. From there, the system begins with the user loading the compost and pressing start where it will then check if the active work area is free of any FOD (such as a person's limb) before activating the conveyor belt and moving into the "Active Sorting" stage. All actions in the grey box of Figure 2 are actions the system takes without user input. At any time, the user can press the "E-stop" button, or type "Stop" into the GUI to shut the system down.

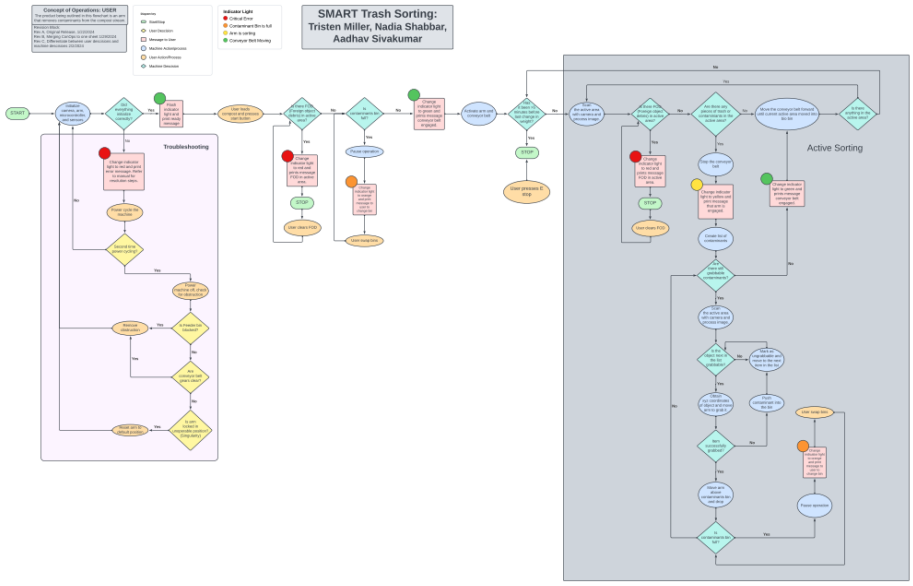


Figure 2: Concept of Operations

2.2 Block diagrams

Author of this section: Tristen Miller, Nadia Shabbar, Aadhav Sivakumar

2.2.1 Top Level

To better help the team capture the overall system and how it interfaces, a top level block diagram was created with only one or two levels of abstraction that are than broken up and explored more deeply in separate block diagrams as "Subsystems." The major subsystems include: Power, Micro-Controller with

Sensor Suite, Arm, and Computer Vision. Figure 3 shows the top level block diagram and how these subsystems interface with one another. Each subsystem will be discussed in their respective sections, and a brief synopsis will be given here.

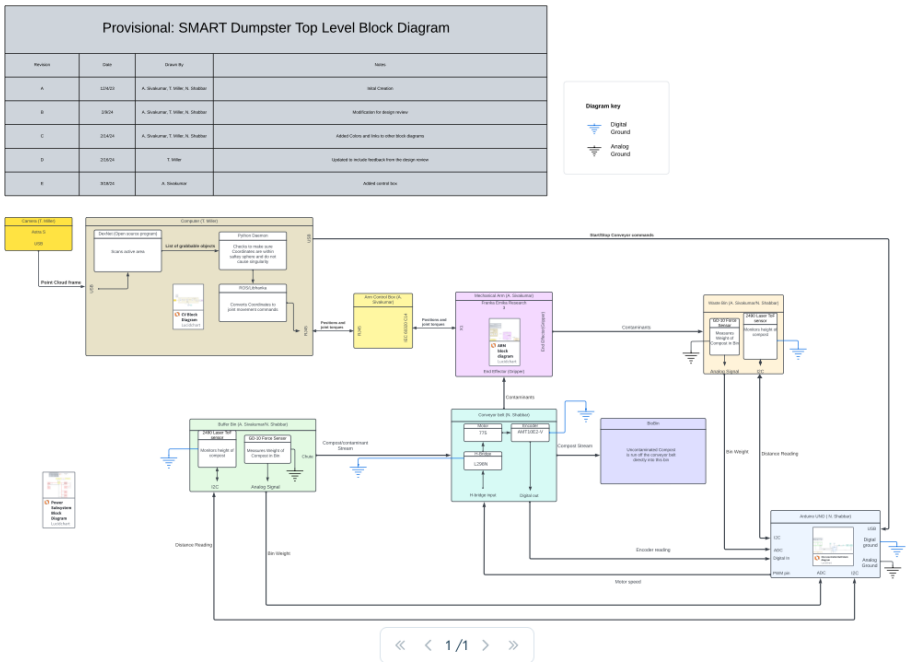


Figure 3: System Block Diagram

2.2.2 Power

The power subsystem encapsulates the entirety of the projects power budget. Its purpose was to help the team design around power constraints, and to determine an adequate power supply as well as an adequate micro-controller. Figure 4 has been included for reference.

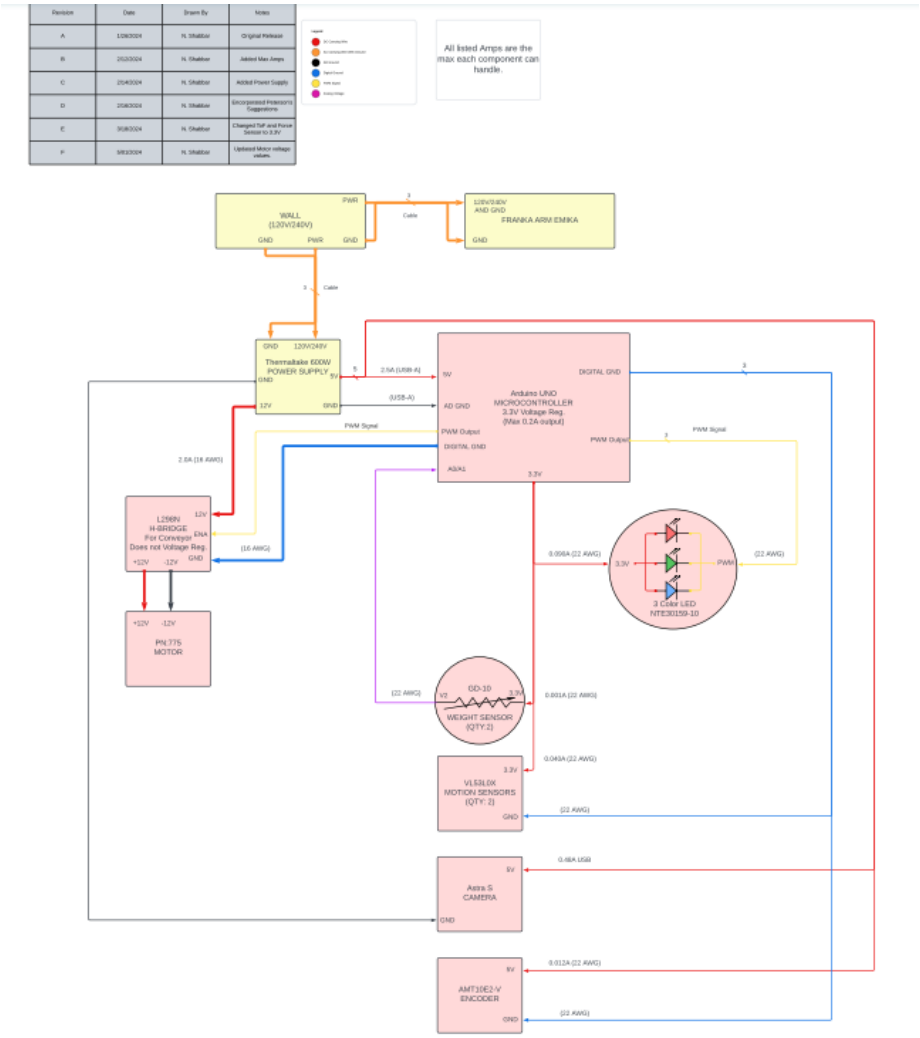


Figure 4: Power Block Diagram

2.2.3 Micro-controller

The micro-controller subsystem encompasses all sensors that drive results from the bins and the conveyor belt, the motor, and the camera. The micro-controller runs a state machine that digests the information from the sensors and camera to enter particular states to perform desired actions. Figure 5 has been included for reference.

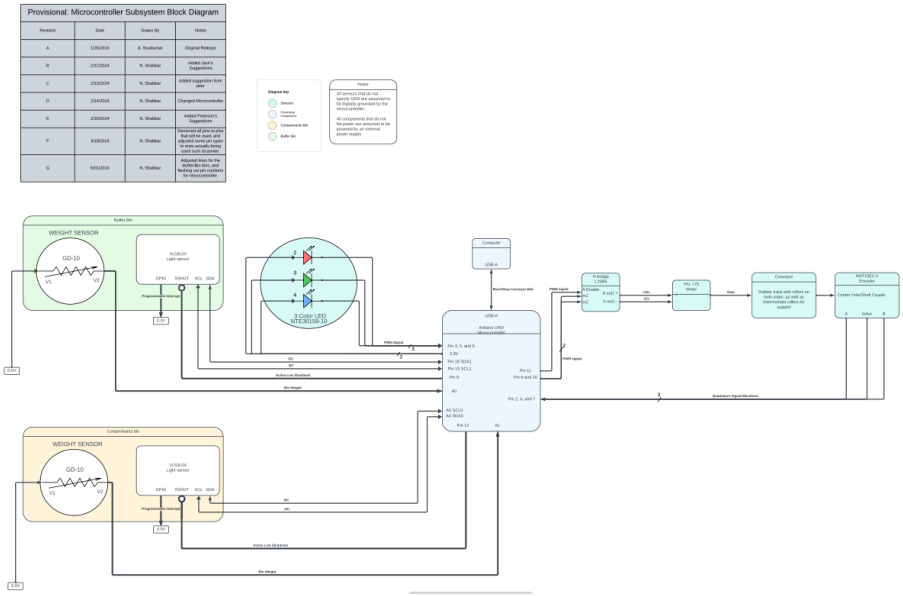


Figure 5: Micro-Controller Block Diagram

2.2.4 Arm

The robot arm inherently does not have any information about it's current environment, but it creates a 3D space around itself and assigns coordinates relative to its base (the base being (0,0,0), or reference origin). This block diagram shows how the information from the camera and the computer are processed and then passed as torque values for the arm to move each joint, eventually moving smoothly to the proper location. Figure 6 has been included for reference.

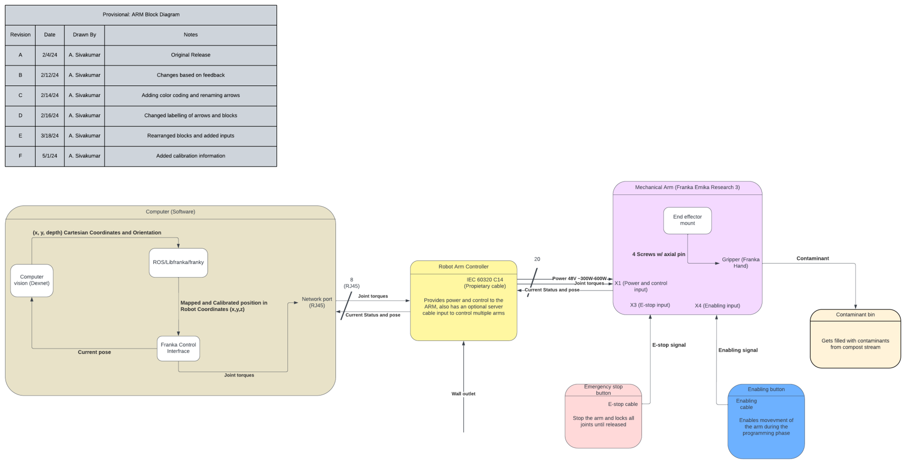


Figure 6: Arm Block Diagram

2.2.5 Computer Vision

The computer vision subsystem will be talked about in depth in section 3.4, however Figure 7 shows the general flow of data from the camera on the left, through several stages set up to assist in detecting objects, grasp planning, and user interface.

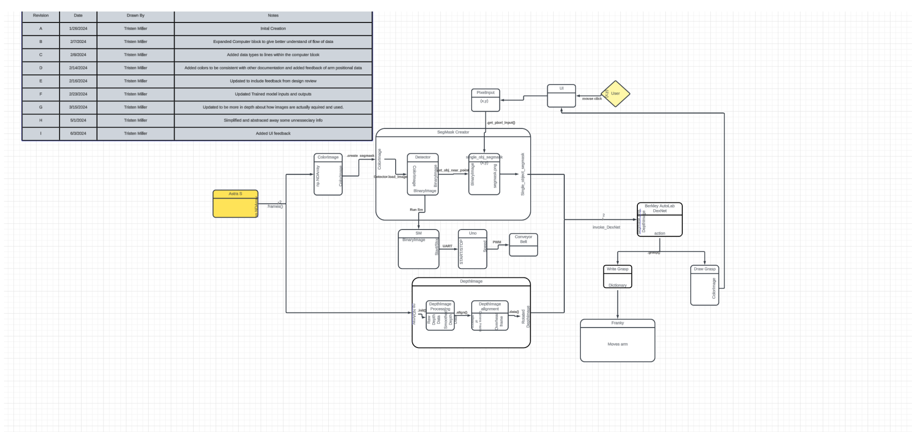


Figure 7: Computer Vision Subsystem Block Diagram

3 Project Components

3.1 Power Subsystem

Author of this section: Nadia Shabbar

Due to the nature of this project, little work needed to be done in terms of allocating power throughout the project. As the larger components of this project were the arm and computer vision systems, nothing else proved to be taxing to the power demand. The arm was able to plug directly into the wall, and the camera for the computer vision plugged directly into the computer being used for this project meaning they only needed documentation on the block diagram. From there, the rest of the power budget came into determining how to power the motor, encoder, Time of Flight Sensors, Force Sensitive Resistors, the LED, and the Micro-Controller. Therefore, the power subsystem kept track of max current outputs and max voltage inputs, as well as digital and ADC signals. The power subsystem was also responsible for keeping track of digital ground, analog ground, and wire gauge thickness (AWG).

3.2 Micro-controller and Sensor Suite Subsystem

Author of this section: Nadia Shabbar

3.2.1 Overview and Requirements

For the subsystem of Micro-controller and Sensor Suite, the concept was to take a conveyor belt and bins, attach sensors to them that feed into a microcontroller, and create a system that could interface with the robot arm and computer vision systems. In order to accomplish this, a DC motor, IR Time of Flight (ToF) sensors, Force Sensitive Resistor (FSR) sensors, an Encoder, a LED, and a Motor Driver were used with an Arduino UNO.

For each bin, a ToF and FSR would be attached with the purpose of gauging whether the bin was full or not. A FSR would be used to make sure the bin would not surpass the weight limit requirement of 50 lbs which was set by our stakeholder. In the event that the material filling the bins was incredibly light, the ToF sensor would measure if the contents reached the top of the bin which would also signify the bin was full.

An encoder would be attached to the conveyor belt to allow backwards motion of the belt on the off chance the belt overshot its destination for the camera, allowing the camera the chance to assess potentially missed contaminants. The motor driver used was to ensure direction and speed of the DC motor which was used to drive the conveyor belt.

Lastly, a LED was used to notify the system which stage in the state machine it was in, or if there was a critical error that needed to be addressed. The microcontroller took the information from all sensors and fed it into a state machine which would then take actions based on the sensor input. An example would

be that if the FSR or ToF triggered, an orange or yellow light would appear in the LED signifying one of the two bins needed to be swapped.

3.2.2 Sensor Setup

The overall subsystem was designed to connect to the micro-controller, allowing all sensors and the computer which the camera was attached to, to interface. This can be seen in Figure 8.

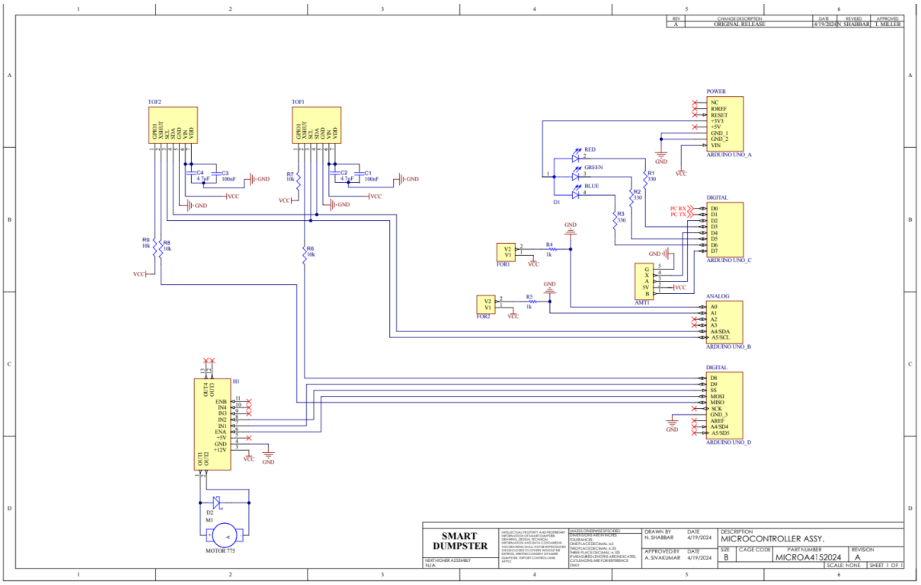


Figure 8: Micro-Controller Schematic

The concept of this project takes the FSRs and ToF sensors, and attaches them to the bins. The FSRs sit on the bottom of the bin to detect the weight of items being dropped into the bin while the ToF sensor sits on the lip of the bin to detect objects reaching the top of the bin. The LED does not have a specific place it needs to sit so it was determined that on a raised pole would be best to allow the user to see the LED color from various places. For testing in our lab, the LED was set near the testing equipment. The motor was placed at the top of the conveyor in order to drive the belt as can be seen in Figure 9. For the encoder, the plan was to attach it to the shaft of the motor, but as the motor part was changed and that was no longer an option, the plan became to create a gear for the conveyor belt that would allow us to attach the encoder.

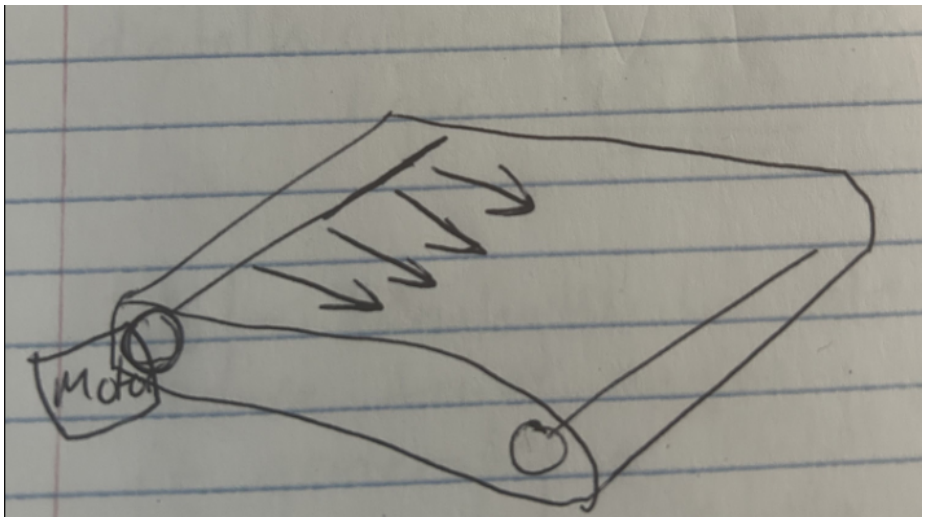


Figure 9: Motor Placement and Direction

3.2.3 Standards Followed

The project adheres to the ANSI C[1] standard, specifically focusing on modularity, type safety, preprocessor directives, and function usage conventions as outlined in the ANSI X3.159-1989 standard. The implementation of ANSI C standards in the code files is demonstrated through several key practices:

The Config.h file utilizes header guards (#ifndef CONFIG_H, #define CONFIG_H, #endif) to prevent multiple inclusions of the same file. This is crucial for avoiding redefinition errors and maintaining code modularity, as specified in section 2.1.1 of the ANSI C standard. The use of #define to define constants for pin assignments in Config.h enhances code readability and maintainability. This aligns with section 4.1.2 of the ANSI C standard, which emphasizes the importance of using preprocessor directives for constant definitions. The separation of function prototypes and definitions into different files, with Config.h for definitions and various .ino files for implementations, adheres to the modular programming principles promoted by ANSI C. This practice ensures that code is organized, manageable, and scalable, as highlighted in section 3.1.4 of the ANSI C standard. Each .ino file maintains type safety by explicitly declaring variable types. Examples include volatile int encoderValue in Encoder.ino and int forceValue in Force.ino. This adherence to type safety is outlined in section 3.1.2 of the ANSI C standard, ensuring data integrity and preventing type-related errors. The consistent use of standard C functions such as digitalWrite, analogRead, Serial.print, and delay across the files follows ANSI C's function usage conventions. This practice is crucial for ensuring code portability and reliability, as detailed in section 4.1.3 of the ANSI C standard. The structure of the ISR in Encoder.ino follows ANSI C guidelines for defining functions and using volatile variables to handle data shared between the ISR and main code. This

is in compliance with section 2.2.1 of the ANSI C standard, which addresses the handling of volatile data in interrupt routines.

The constraints articulated in the ANSI C standard that are integrated into the project's requirements include: Ensuring that code is divided into manageable modules to enhance readability and maintainability. Enforcing strict type declarations to prevent data integrity issues and type-related errors. Utilizing preprocessor directives for constant definitions and macro functions to maintain code consistency and readability. Adhering to the use of standard C library functions to ensure code portability and reliability.

3.2.4 Software Architecture

For the Micro-Controller subsystem, software was an important factor. The first task to tackle was familiarizing oneself with an Arduino, as Nadia never worked with one. This did not take long and was rather straight forward. Next, it came to getting each individual driver for each sensor to work. She initially ran into issues with the ToF sensor as nothing worked to make it accurate in software until she rewrote it in C on a different micro-controller. This led her to find that the library created by the initial manufacturer was not accurate. In addition, she discovered that the ToF was only accurate in reading when it was fully soldered to a board rather than on a breadboard. The FSR also proved to be uncooperative, but this was later discovered to be due to noisy sensors that were bad.

From there, a "main.ino" was created in a main project folder that contained all the sensor drivers. As discovered by Nadia, the arduino files work slightly different in how files get stitched together, and so a "config.h" file was created. This "config.h" file contained all pin assignments, all function headers, all state machine structs, and all OpCodes that would be used throughout the project. With the "config.h" file in place, the first algorithm written for the "main.ino" was a checkOperationStatus. The purpose of this algorithm, which can be seen in Appendices 2, is to take a command via UART either by the computer vision/arm system, or by the user themselves and drive the conveyor belt state machine.

The state machine would then take a command via UART, and parse the information to determine which state it would end up beginning in with regards to the start state or error on the off chance that issues arose in initialization, or an object was detected in the active working area that should not be present such as a person's limb. The initial conception of the conveyor belt state machine can be seen in Figure 10. As seen in Figure 10, the state machine only shows the cases of a one bin, and one set of sensors system. For testing purposes, this was ideal to ensure that the code and subsystem worked with the other subsystems of the overall project. However, for the full scope of this project, that was not ideal and so the conveyor belt state machine had to be reworked to account for a second bin system. The revised state machine can be seen in Figure 11 where now a bio bin and contaminant bin can be seen present in the state machine.

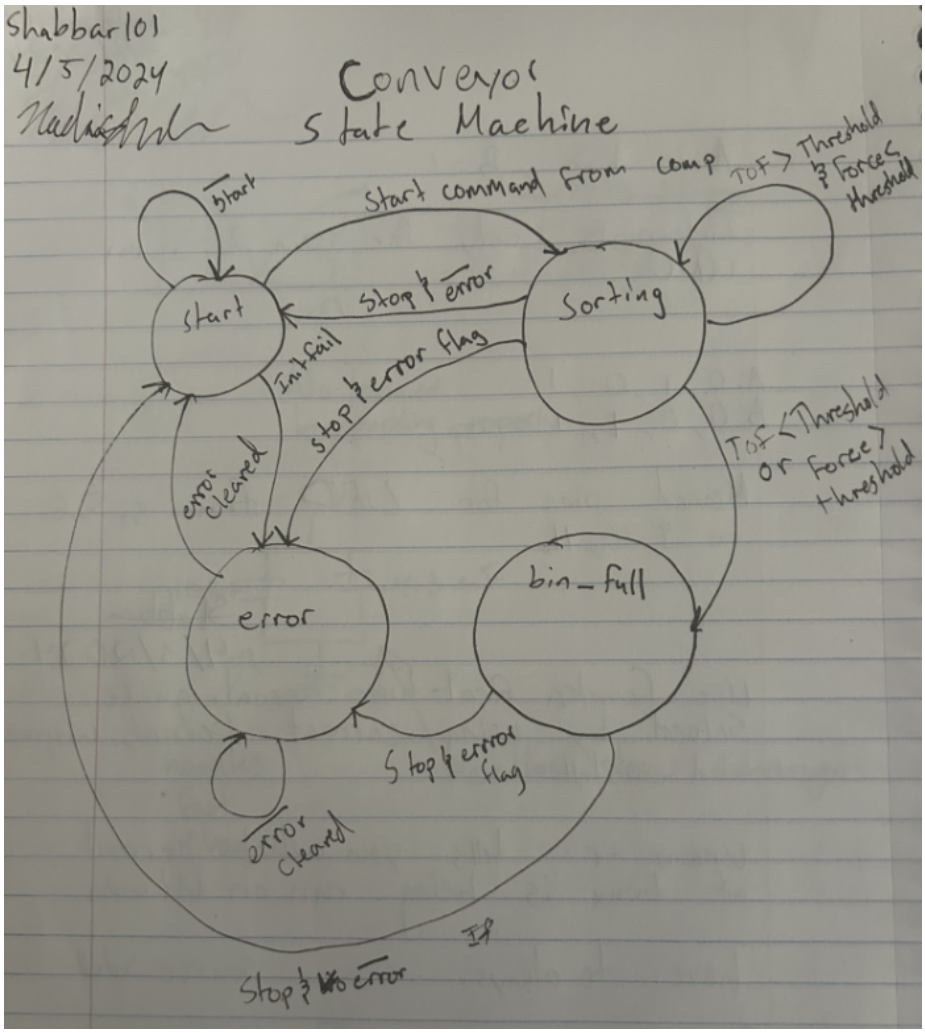


Figure 10: Conveyor Belt Operations State Machine

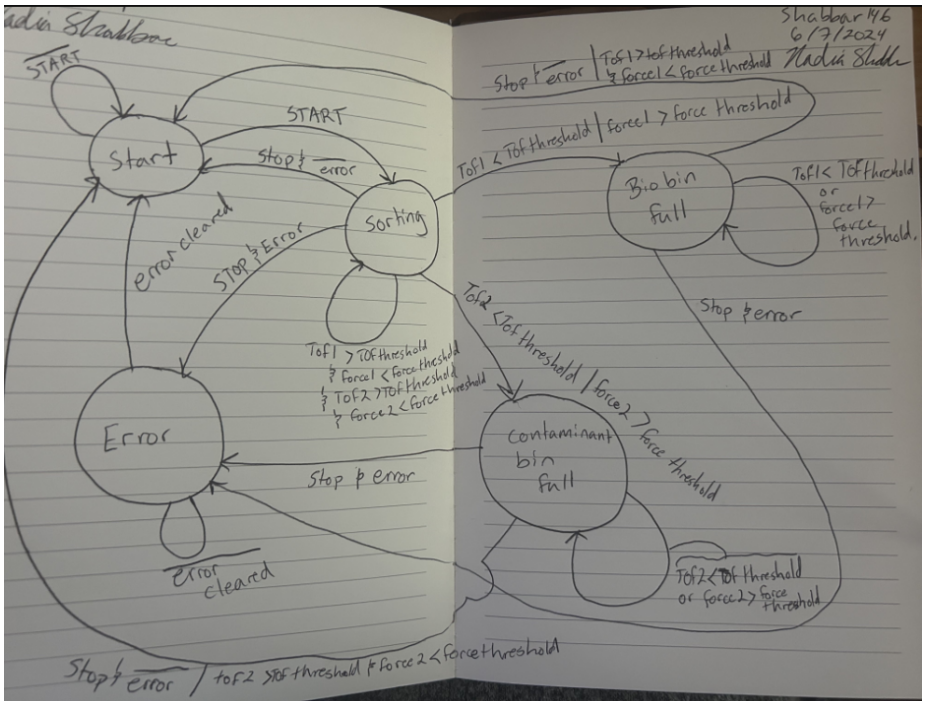


Figure 11: Conveyor Belt Operations State Machine Revised

Should an error ever arise, the conveyor belt state machine would enter the error state where a separate error state machine would be called to address any potential errors. This error state machine concept can be seen in Figure 12.

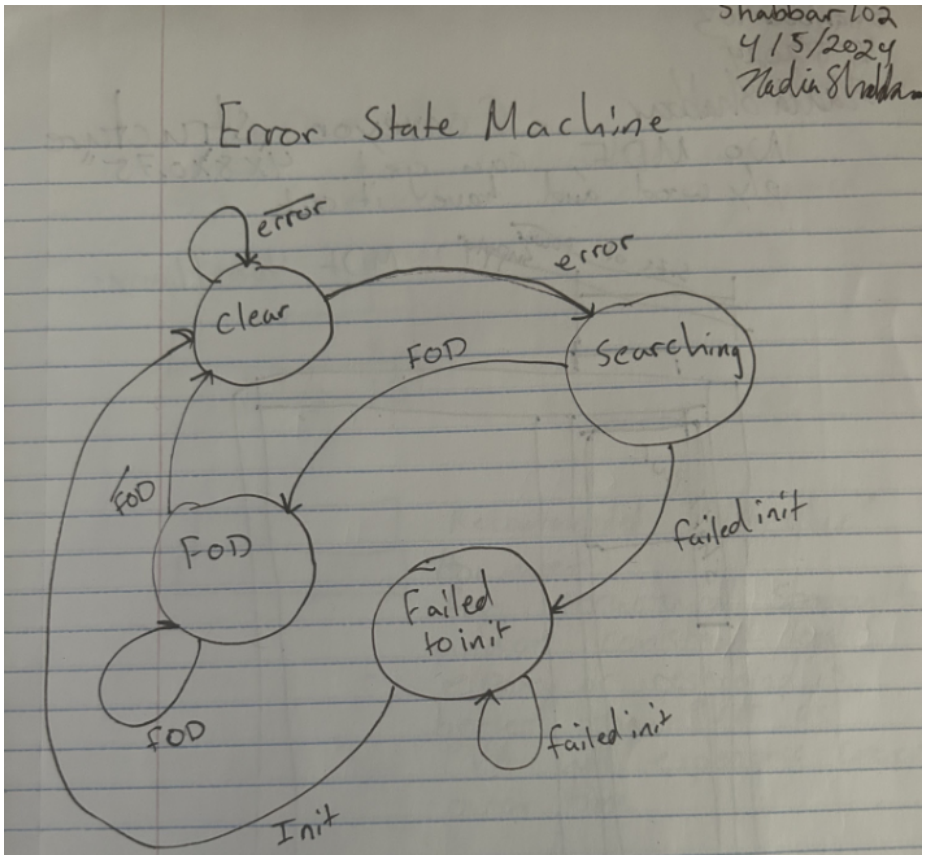


Figure 12: Error Handling State Machine

3.2.5 Problems With Setup

With our software created and thoroughly tested, the last thing to do was connect all subsystems together, and to be sure that all components worked as intended. In doing so, the team needed a physical conveyor belt. At first, the team worked on designing their own conveyor belt based off of the sketches from Nadia's notebook as seen in Figure 13, but unfortunately, the team found that the cost of doing so would drive too much of the budget. Therefore, the team had to pivot their design. After some time searching on Facebook Market, the team located a free treadmill that could then be repurposed as a conveyor belt. Issues quickly arose with this solution as the chosen motor and motor drivers were no longer spec-ed to provide enough tension and power to drive the conveyor. As such, the team had to return to the drawing board.

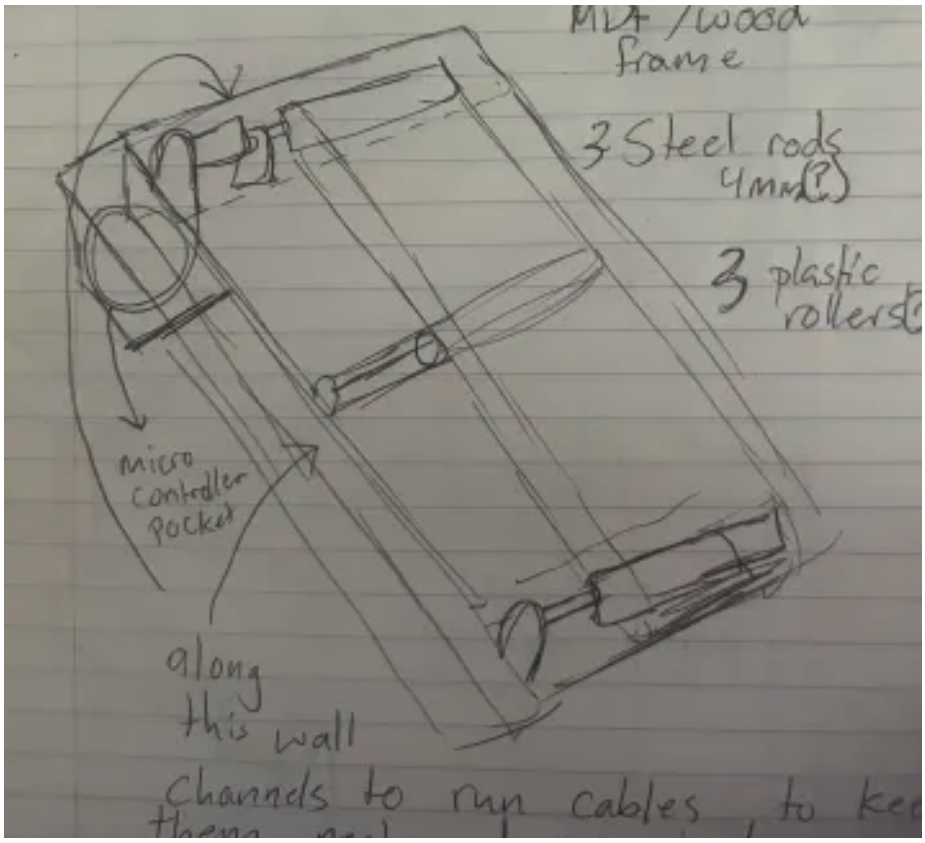


Figure 13: Conveyor Belt Concept

With luck, the original motor of the treadmill still worked, though upon inspection, Nadia and Aadhav realized the controller was broken beyond repair. While Aadhav worked on a temporary solution to drive the motor with no speed control via relays, Nadia worked on finding a replacement motor driver, settling for Toshiba TB67H303HG as it was the only through hole part that fit the criterion of the project and was not obsolete as seen in Figure 14.

REV:	Change Log:			Created By:		Date:		Title:	
A	Original Release			N. Shabbar		5/12/2024		H-Bridge Pugh Chart	
Product Name:	**>12V Input:**	**Full Bridge:**	**5V PWM Signal Input:**	**<100mA Signal Current:**	**>5A Output:**	**Switch time:**	**Total:**	**Datasheet:**	**Notes:**
Weighting:	2x	1x	1x	1x	2x	1x		N/A	
Existing Solution	0	0	0	0	0	0	0		
Toshiba TB67H303HG	5	5	5	5	3	0	31	https://toshiba.se	Maxes at 8A output, deadband is 1 microsecond

Figure 14: Motor Controller Pugh Chart

In addition to choosing a motor driver, a proper heat sink was needed to be chosen. As such, heat dissipation calculations were performed using the following functions seen in Figure 15.

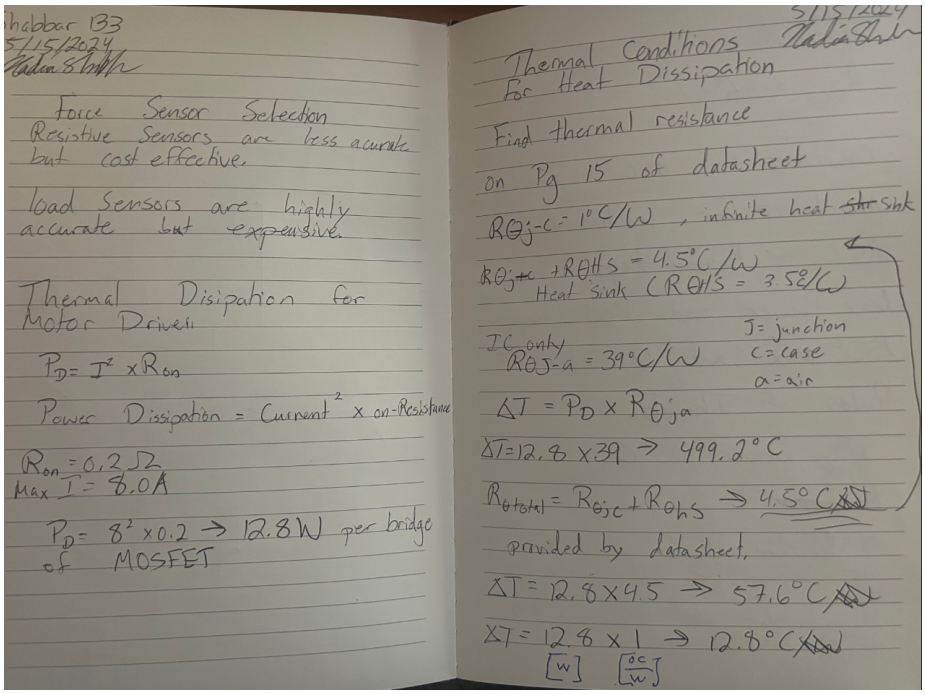


Figure 15: Heat Dissipation Calculations

An additional problem that arose with this subsystem comes back to the FSR. While the state machine worked phenomenally, the states would trip incorrectly due to noisy FSRs. In order to trouble shoot this, the FSR and power were input into an Oscilloscope. The power stayed consistent, but the FSR would constantly fluctuate which lead us to believe there was something wrong with the sensors themselves. Sure enough, when attached to a good multi-meter, the variable resistance would bounce around too much. As a result, the team opted to order new parts to test for a replacement. However, due to the strikes, the team never received their parts.

3.2.6 Results

The overall result was that the state machine worked, but some parts needed replacement. A new motor, motor driver, heat sink for the motor driver, and FSRs were necessary and picked out. However, due to the strikes, the team were never able to implement the changes the team desired or to thoroughly test the new parts as the team were unable to receive any purchased parts as delivery drivers could not pass the picket line.

3.2.7 Future Work

Due to the strikes, the team was never able to successfully receive their parts to further their project. As such, if the students had more time to work on their project, some of the things they would have liked to do would be to continue working with the larger motor and rework the schematics to include the motor in the design. Another issue the students would have liked to tackle would have been ADC averaging for a single AD pin with the FSRs. What the student responsible for this section came up with, is that four FSRs could be arranged in parallel for an average reading that could then be used to determine weight more accurately than if a single sensor was used on the bin as the FSR required the weight to be directly on top of the sensor to read accurately. Given the size of the FSR and the bin, that would not work, hence why the averaging was necessary. The suggested setup can be seen in Figure 16.

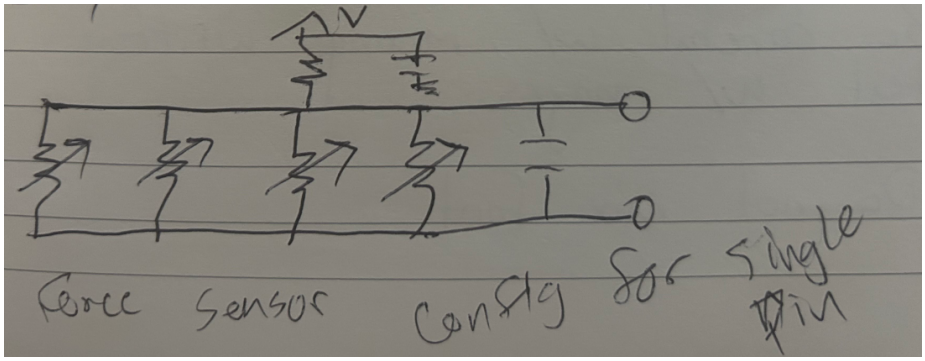


Figure 16: Concept of ADC Averaging for FSR

3.3 Robotic Arm (Franka Emika Research 3) Subsystem

Author of this section: Aadhav Sivakumar

The arm the team decided to go with is the Franka Emika Research 3. The decision to use this arm was due to the requirements of one of two stakeholders. Tae Myung Huh, owner of the tactile manipulation lab, offered the capstone team use of his lab and the robot arm, in exchange for setting up the arm to interface with the computer.

3.3.1 Movement

When considering robotic movements, there are two primary ways to go about doing it. The first way is a joint movement, which calculates all of the necessary angle changes then executes them all simultaneously, arriving at the desired end pose in the least amount of time possible. The problem with this approach is

that it frequently ends up with joints that are coincident with each other, as well as end-effector positions that go through the table or bounding area on the way toward its final destination. The calculations for the angular changes, the angular velocities, and the torques for each joint can be derived from the Denenvit Hartenburg parameters, which can be found on the Franka Emika website [3].

Another control method that avoids these problems is the Cartesian movement, which moves the end-effector in a straight line from one specified X Y Z coordinate to another, along with a quaternion representing a rotation at the last point. This way, the team can ensure that the arm doesn't go through the table since the team can manually input z values greater than 0. This method does have a drawback of decreased range because the calculations happen in real-time, so even if a coordinate is slightly out of bound, it will attempt to reach that point and immediately seize up once close to a singularity, which is the point of no return where no movement command can recover the robot's position and an operator needs to come in and reset the mechanical locks.

3.3.2 Space Mapping

The camera is able to output an X, Y, and depth value of any particular object through the dexnet algorithm. In order to map camera X Y depth space to robot X Y Z positions, the team needed some reference objects. It was decided to use rectangular prisms of various sizes, using the algorithm to grab their X, Y, and depths. Using 8 different points, it was possible to do some linear algebra to map each X Y Z coordinate individually. For the next x coordinate, this equation was used:

$$Xnext = \begin{pmatrix} a\_X\_Bot * Xcam * Ycam * Zdep + b\_X\_Bot * Xcam * Xcam + \\ c\_X\_Bot * Ycam * Ycam + d\_X\_Bot * Zdep * Zdep + \\ e\_X\_Bot * Xcam + f\_X\_Bot * Ycam + g\_X\_Bot * Zdep + h\_X\_Bot \end{pmatrix}$$

. There is a corresponding equation for the Y and Z values, using different coefficients. Since there are 8 unknown coefficients, they can be solved for, effectively creating a new plane that has the inputs of the camera X, Y, and depth values. Different equations were used throughout the course of the project and this was the most effective one.

3.4 Computer Vision Subsystem

Author of this section: Tristen Miller

3.4.1 Overview and Requirements

The smart trash sorting system utilizes computer vision to identify contaminants in the client's compost stream. The user interface allows the user to select an

object by clicking on it, initiating a process that automatically translates the object's coordinates from the camera's frame of reference to the robot's frame of reference, thereby enabling the robot to pick up the object. To plan the grasp on an object, the team opted to use UC Berkeley's DexNet [5], a neural network trained to handle novel objects. To effectively employ DexNet required a camera capable of capturing both depth and color images.

The team selected the Orbecc Astra RGBD camera as a balanced choice between affordability and quality. Further details regarding the Astra camera are discussed in Section 3.4.3. Besides setting up the camera, significant effort was dedicated to object detection (detailed in Section 3.4.4) and relaying the information to the robot. This involved ensuring precise communication between the camera, the detection algorithms, and the robotic system to achieve reliable object manipulation and removal.

3.4.2 Standard followed

The PEP8[7] Python style guidelines were meticulously followed to ensure the code quality and maintainability of this project's design. PEP8, widely recognized in the Python community, provides conventions for writing readable and consistent Python code. Key sections of the standard that were particularly relevant to our project include naming conventions, code layout, and import order. By adhering to these guidelines, the team ensured that the codebase remains understandable and manageable not only for the current team but also for future contributors. The guidelines facilitated a collaborative environment where code reviews and debugging processes were more efficient, significantly reducing the potential for errors and inconsistencies.

Another aspect of PEP8 that was strictly followed was the guidelines on documentation and comments. Clear and concise documentation was incorporated throughout the code, following PEP8's recommendations for docstrings. Each function and class was accompanied by detailed docstrings explaining its purpose, parameters, and return values. This practice greatly enhanced the usability and readability of the code, making it easier for new team members to quickly understand and contribute to the project. Additionally, inline comments were used judiciously to explain complex logic and important decisions, providing context and insights that are invaluable during maintenance and further development. By implementing thorough documentation and comments as prescribed by PEP8, the team ensured that the project remains transparent and accessible, promoting long-term sustainability and ease of use.

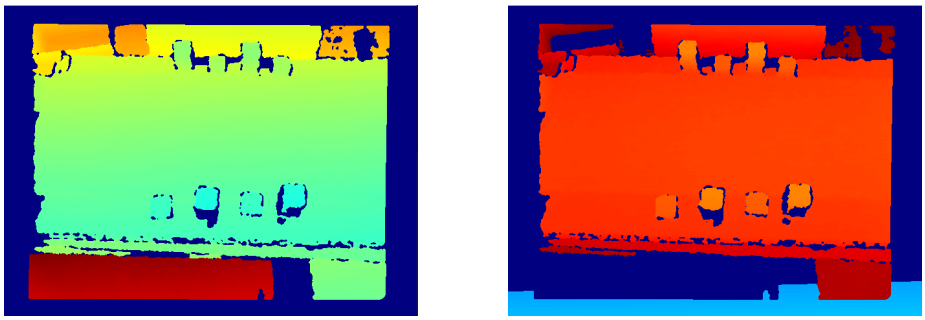
Consistent naming of classes, variables, and functions was another key element derived from PEP8 that contributed to the clarity and professionalism of our codebase. PEP8 advises using descriptive names that convey the purpose of the entity being named. For example, class names follow the CapWords convention, such as TrashSorter. Variables and function names use lowercase words separated by underscores to improve readability, such as trash_items for a variable and sort_trash() for a function. This consistent naming convention made our code easier to read and understand, as each name clearly indicated

its role and functionality within the code. Adhering to these naming conventions helped prevent confusion and errors, as well as making our codebase more intuitive for both current and future developers.

3.4.3 Camera Setup

To utilize the camera, which maps a two-dimensional plane to three-dimensional coordinates, the initial step involved calibrating the camera. The team elected to employ Zhang's Method [8], which uses a chessboard pattern of known dimensions to correct distortion at the periphery of the camera's field of view. This method involves capturing multiple images of the chessboard at different orientations and positions, allowing the algorithm to compute the intrinsic and extrinsic parameters of the camera. By applying this technique, the tea ensured that the RGB data from the camera was accurately rectified and free from lens distortions.

Following the calibration of the RGB camera, the team proceeded to calibrate the IR sensor to ensure precise depth measurements. This procedure was akin to Zhang's Method; however, the team discovered that, in addition to the planar movement of the chessboard, it was necessary to adjust the distance between the chessboard and the camera. This additional step was essential for achieving accurate depth calibration, as it accounted for the varying focal lengths and depth perception of the IR sensor. To facilitate this process, a larger chessboard pattern was printed, which allowed the team to capture calibration images at different distances and orientations, thereby enhancing the accuracy of the depth readings. Once the team had accurate depth readings from the camera, the team realized that due to physical limitations it was impossible to mount the camera directly overhead as DexNet was trained on. This resulted in a slanted depth map. Figure 17a shows the depth readings mapped to a color. The change in color from bottom to top causes the DexNet algorithm to interpret objects as being longer and more askew than they actually are. In order to correct this slant, the team developed an algorithm (Appendix 6.1:1) to compute the best rotation matrix by choosing four points known to be of equal height, and minimizing the standard deviation of the set. This allowed us to virtually shift the camera such that it was measuring from a flat plane over the work area.



(a) Raw Depth Map

(b) Depth Map after rotation algorithm

Figure 17: Depth map before and after rotation

3.4.4 Object Detection Method

In order to use the readings from the camera with DexNet, the team had to create a binary image, that was black everywhere except what the team were interested in. the team used two different methods to highlight objects of interest for our project. The first method the team used is background subtraction[6]. Background subtraction is a technique used in computer vision to identify objects by separating the foreground from the background. This method typically involves creating a background model, which is a statistical representation of the scene's background. In our case, the team created the background model using the mean of histograms of the color channels. By comparing each frame to this background model, the team can identify pixels that deviate significantly, indicating the presence of an object.

The team found that this worked well for most cases, however if an object was close in color to the background of the active working area, it would often get masked out. In addition to background subtraction, the team created a binary image based on the depth map. Objects that were outside the bounds, either too far or too close, to the camera were removed.

These binary images were combined using a logical AND operation to remove any noise that was introduced in the processing and creation of these images.

3.4.5 Software Architecture

The software architecture of the system is designed to facilitate efficient data capture, processing, and analysis to achieve the objectives of the project. Below, the team outline the main components and their interactions within the system.

Camera Class: The camera serves as the primary data acquisition tool, capturing both RGB and Depth images. It is implemented as its own class, encapsulating all the necessary pre-processing steps required for the subsequent stages of the system. This design choice allows for modularity and ease of maintenance, enabling future enhancements or modifications to be made inde-

pendently of other system components. Because the Camera class is independant of the rest of the codebase, the actual camera can be swapped out for another camera that provides RGB and Depth pictures.

RGB and Depth Classes: The captured RGB and Depth images are represented as objects of their respective classes (ColorImage and DepthImage). These classes provide various methods for image manipulation, such as filtering, resizing, and transforming images. This abstraction ensures that image processing tasks are streamlined and can be easily managed within the system.

Detector Class: The core functionality of object detection is handled by the Detector class. This class leverages the DexNet algorithm, a robust computer vision model, to identify objects within the processed images. The use of DexNet enables high accuracy in detecting and classifying a wide range of objects, making it a critical component of our system.

DetectedObject Class: Once objects are detected, they are stored as instances of the DetectedObject class. Each DetectedObject contains detailed information about the detected entity, including its contour, color, and positional data. This class structure facilitates easy access and manipulation of object-related information, enabling subsequent processing tasks such as sorting and classification.

Data Flow and Processing: The data flow within the system begins with the Camera class capturing RGB and Depth images. These images are then pre-processed and fed into the Detector class, where the DexNet algorithm performs object detection. Detected objects are instantiated as DetectedObject class objects, storing all relevant information. This structured approach ensures that data is efficiently processed and available for further analysis or action.

Integration with Hardware: The software components are tightly integrated with the hardware, specifically the camera. The Camera class interfaces directly with the camera hardware, managing the capture of images and any necessary pre-processing steps. This integration ensures seamless communication between the hardware and software, enabling real-time data acquisition and processing.

Modular Design: The modular design of the software architecture enhances the system's maintainability and scalability. By encapsulating functionalities within distinct classes, the system can be easily extended or modified. For instance, improvements to the object detection algorithm can be made within the Detector class without impacting other components. Similarly, enhancements to image processing techniques can be localized to the ColorImage and DepthImage classes.

3.4.6 Trouble with set-up

In order to get DexNet running, the team faced several challenges with versioning. The version of DexNet in use was last updated in 2016 and was designed to work with TensorFlow that requires Python 3.7. However, the computer that was available to the team was equipped with an NVIDIA 4090 graphics card, which requires a CUDA library compatible with Python 3.10. This incompati-

bility forced the team to update TensorFlow to its newest version. During this process, several deprecated functions needed to be replaced and some compatibility issues with other libraries needed to be resolved.

Another problem arose with height inconsistencies when using DexNet. Despite correctly a rotation matrix to the depth data, DexNet provided height readings that varied by as much as 10 cm from the expected values. To address this, the conducted a validation process, comparing DexNet's height readings with the consistent and accurate measurements from the depth camera. Given these discrepancies, the team decided to use DexNet solely for identifying the optimal grasping locations. For determining the precise height, the team relied on the depth camera readings, which consistently provided accurate surface measurements.

Additionally, the poor lighting conditions in the lab made it challenging to consistently detect RGB colors. This issue was particularly problematic for identifying objects and performing color-based segmentation. To overcome this, the team converted the color space from RGB to HSV. The HSV color space separates brightness (value) from color information (hue and saturation), making it more robust to lighting variations. This conversion significantly improved the ability to detect and differentiate colors, leading to more reliable object recognition and segmentation in varying lighting conditions.

These solutions required detailed problem-solving and adaptation, ensuring our system could effectively integrate and utilize DexNet while maintaining accuracy and reliability in our grasping and detection tasks.

4 Future Research

Author of this section: Tristen Miller

Future research areas for this project are abundant and hold significant potential for enhancing its practical applications. One primary focus should be the further refinement of the ability to consistently grasp objects. Improving the precision and adaptability of the grasping mechanism is essential for ensuring that a wide variety of objects can be reliably and efficiently handled. This could involve exploring more advanced sensor technologies and machine learning algorithms to better understand and predict the physical properties and orientations of objects. Additionally, integrating tactile feedback mechanisms could provide real-time adjustments to the grasping process, thereby reducing the likelihood of errors and increasing the overall efficiency of the system.

Another area of future research is the implementation of automatic identification of trash and compost. Currently, the system relies on user input to decide which objects to pick up, which is not only time-consuming but also prone to human error. Developing an automatic identification system would involve extensive research into computer vision techniques and artificial intelligence algorithms. These methods could enable the system to distinguish between different types of waste with high accuracy. By leveraging large datasets and training so-

phisticated models, the system could learn to recognize and categorize various objects autonomously, thus streamlining the sorting process and significantly improving the efficiency and effectiveness of waste management.

Additionally, research into methods for detecting when the bin is full will be essential for creating a fully autonomous waste management system. Current challenges in this area include developing reliable sensors that can accurately measure the bin's capacity and determining the optimal placement and integration of these sensors within the system. Addressing these challenges would ensure that the system can alert users or initiate appropriate actions when the bin is nearing its capacity. Unfortunately, the project faced setbacks due to difficulties in acquiring necessary parts, a situation exacerbated by the Strike. This delay impacted progress, particularly in areas requiring physical components for construction and testing.

5 Results and Conclusion

Author of this section: Aadhav Sivakumar

The overall grab success rate was a little over 80 percent for the test runs on the conveyor belt. Given any cube that fits within the dimensions of the gripper, the object has a higher percentage of being grabbed if its side length is closer to half the gripper width, and a lower chance of being grabbed if its toward the extremes (very large cube or very small cube).

The final product is not completely aligned with what the team's original idea for the project was due to limiting time, supply chain issues, and unpredictable events. As a result, the team was only able to implement a system that met one stakeholder's set of requirements, but not the other's. The team's system isn't able to differentiate non-compost-able items and compost-able items though feel enough work has been established that if a future group would be interested in continuing the efforts, time can be saved in the setup while enough work would allow a team to continue with interesting challenges. The work achieved can be summarized as creating a system that can interface with a micro-controller and various sensors to operate a conveyor belt, a computer vision system that can detect objects manually, and a robot arm system that can move and pick up objects at a specified position. For future teams, they would need to train the model and improve efficiency as well automate the system to work for compost-ables.

References

[1] American National Standards Institute. Programming language c, 1989. ANSI X3.159-1989.

[2] EPA. Impacts of sending food and other organic materials to landfills. *Environmental Protection Agency*, 2019.

[3] Franka Emika. Franka emika research 3 control parameters, 2020.

[4] Environmental Resilience Institute. Composting at home: How to reduce your waste and make your own fertizier. *Indiana Uiniversity*.

[5] Jeffrey Mahler, Matthew Matl, Vishal Satish, Michael Danielczuk, Bill DeRose, Stephen McKinley, and Ken Goldberg. Learning ambidextrous robot grasping policies. *Science Robotics*, 4(26):eaau4984, 2019.

[6] Shahrizat Shaik Mohamed, Nooritawati Md Tahir, and Ramli Adnan. Background modelling and background subtraction performance for object detection. In *2010 6th International Colloquium on Signal Processing its Applications*, pages 1–6, 2010.

[7] Python Software Foundation. PEP8: Style Guide for Python Code. https://peps.python.org/pep-0008/, 2001. Accessed: 2024-05-29.

[8] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

6 Appendices

6.1 Algorithms

Algorithm 1 Compute Best Rotation Matrix

function ComputeBestRotationMatrix(depth, points)
  Initialize $min\_dif \leftarrow \infty$, $best\_\theta \leftarrow 0$, $best\_\phi \leftarrow 0$
  Initialize $\theta\_min \leftarrow -1$, $\theta\_max \leftarrow 0$, $\theta\_step \leftarrow 0.1$
  Initialize $\phi\_min \leftarrow -1$, $\phi\_max \leftarrow 0$, $\phi\_step \leftarrow 0.1$
  while $\theta\_step > 10^{-4}$ **or** $\phi\_step > 10^{-4}$ **do**
    for $\theta\_deg \in$ range$(\theta_min, \theta_max, \theta_step)$ **do**
      $\theta \leftarrow \theta\_deg \times \frac{\pi}{180}$
      for $\phi\_deg \in$ range$(\phi_min, \phi_max, \phi_step)$ **do**
        $\phi \leftarrow \phi\_deg \times \frac{\pi}{180}$
        $T \leftarrow$ CreateTransformationMatrix(θ, ϕ)
        $transformed\_depth \leftarrow$ TransformImage(depth, T)
        $vals \leftarrow$ ExtractValues(transformed_depth, points)
        if $0 \in vals$ **then**
          continue
        end if
        $dif \leftarrow$ AverageDifference(vals)
        if $dif < min\_dif$ **then**
          $best\_\theta \leftarrow \theta\_deg$
          $best\_\phi \leftarrow \phi\_deg$
          $min\_dif \leftarrow dif$
        end if
      end for
    end for
    $\theta\_min, \theta\_max, \theta\_step \leftarrow$ AdjustRangeAndStep(best_θ, θ_min, θ_max, θ_step) $\phi\_min, \phi\_max, \phi\_step \leftarrow$ AdjustRangeAndStep(best_ϕ, ϕ_min, ϕ_max, ϕ_step)
    $min\_dif < 0.001$
    break

  end while
  $\theta \leftarrow best\_\theta \times \frac{\pi}{180}$
  $\phi \leftarrow best\_\phi \times \frac{\pi}{180}$
  $R \leftarrow$ CreateTransformationMatrix(θ, ϕ)
  return R
end function

function CreateTransformationMatrix(θ, ϕ)
  return $\begin{bmatrix} \cos(\phi) & \sin(\theta)\cdot\sin(\phi) & \cos(\theta)\cdot\sin(\phi) & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ -\sin(\phi) & \sin(\theta)\cdot\cos(\phi) & \cos(\theta)\cdot\cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
end function

function AdjustRangeAndStep(value, range_min, range_max, step_size)
  if $(value - range\_min < step\_size)$ **or** $(range\_max - value < step\_size)$ **then**
    $step\_size \leftarrow step\_size/10$
  end if
  $range\_min \leftarrow value - 5 \cdot step\_size$
  $range\_max \leftarrow value + 5 \cdot step\_size$
  return $range\_min, range\_max, step\_size$
end function

Algorithm 2 Check Operation Status

```
1: function checkOperationStatus(void)
2:     if Serial is available then
3:         Read string until new line.
4:         Remove any white space and new lines.
5:         Make string all lowercase.
6:         if command == "start" then
7:             Initialize operation flag to START.
8:             Echo Command to Serial Port.
9:         end if
10:        if command == "stop" then
11:            Initialize operation flag to STOP.
12:            Echo Command to Serial Port.
13:        end if
14:        if command == "fod" then
15:            Initialize operation flag to STOP.
16:            Echo Command to Serial Port.
17:            Initialize error flag HIGH.
18:            Initialize error type error_FOD.
19:        end if
20:        if command == "failedtoinitialize" then
21:            Initialize operation flag to STOP.
22:            Echo Command to Serial Port.
23:            Initialize error flag HIGH.
24:            Initialize error type error_FailedToInit.
25:        end if
26:        if command == "cfi" then
27:            Initialize operation flag to STOP.
28:            Echo Command to Serial Port.
29:            Initialize error flag HIGH.
30:            Initialize error type error_FailedToInit.
31:        end if
32:        if command == "clear" then
33:            Initialize operation flag to STOP.
34:            Initialize error flag to LOW.
35:            Initialize error type to error_clear.
36:            Echo Command to Serial Port.
37:        end if
38:        if then
39:            Error: Invalid Response!
40:        end if
41:    end if
42: end function
```