LEGO® Sorting Robot for the Classroom

A Major Qualifying Project Report

Submitted to the Faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

in Robotics Engineering

and in Mechanical Engineering

by

Maria Sharman

Phillip Lund

Date: May 5, 2021

Project Advisors:

Professor Craig Putnam, Professor Kenneth Stafford

Abstract

In this Major Qualifying Project, a small-scale robotic system that sorts a classroom set of LEGO Mindstorms Robotics Education pieces back into kit trays was developed. Using AI and a mechanical part manipulation system, the robot sorts and counts individual pieces to ensure each kit is organized with the correct number and layout of pieces. The final design is effective, portable, and affordable for organizations who could benefit from it, such as schools, summer camps, and extracurricular programs who incorporate LEGO Robotics education into their curricula.

Table of Contents

Table of Figures

Introduction

According to Forbes Magazine, "[LEGO Education's] Mindstorms [robotics kit] is one of the most prevalent, most powerful hands-on science and engineering resources in schools around the world [1]." Since its release in 1998 [1], the Mindstorms robotics product line has been integrated into K-12 classroom, after-school, and summer camp curricula as a powerful, hands-on STEM learning tool. Hands-on learning helps students become more engaged, demonstrate higher-level understanding of topics, and learn to work with their peers [2]. But there is one big downside: cleanup.

Education programs who use the LEGO Mindstorms or similar platforms constantly need to reorganize and inventory their LEGO kits. Even when involving students, this is a very labor-intensive task that cuts hours out of the limited time they have for learning robotics. This problem is exacerbated when allowing students to expand their creativity and the complexity of their systems by combining kits or using extra LEGO parts.

If kits are staying in one place, approximate sorting that can be done more quickly may be acceptable—a missing part can be found elsewhere in the room. However, a school district that owns many LEGO kits often uses them for many different programs at different locations throughout the year and usually wants to ensure each kit is exactly complete. In one of the team member's K-12 school district, a city-wide district serving over 47 schools, LEGO robotics kits are used for STEM summer camps at a few locations in the summer, and then for *FIRST* LEGO League at many elementary schools in the fall. In between these major events, all the district's LEGO kits need to be carefully inventoried – a necessary, but tedious, and costly process. After one month of summer camps, the school district spent over $2000 in labor costs inventorying these kits- a recurring, considerable cost for a public-school system's STEM program.

The goal of this project is to develop a small-scale robotic system that sorts LEGO Mindstorms pieces back into kit trays. The system will sort and count individual pieces to ensure each kit is organized with the correct number and layout of pieces. Ideally, the final consumer product will be small, portable, and affordable for organizations who could benefit from it. Our MQP team made progress towards this goal by designing an overall LEGO sorting system and then building and testing its most innovative components.

Background

The background information for this project comes from two sources: research on what potential consumers, who are educators who use LEGO Education to teach, would like to see from a LEGO Sorting Robot, and investigation of past attempts to create similar machines. Based on this background research, the team developed a set of design goals and criteria for a LEGO sorting classroom robot.

Consumer Research

At the beginning of the design process, the team held a small focus group with two potential consumers for this robot: Mike Barney, director of the Massachusetts Academy of Math and Science, and Colleen Shaver, director of WPI's Robotics Resource Center and former Education Resources Coordinator at *FIRST*. Their ideal version of this product would be a robot that sorts multiple kits simultaneously, prioritizes completing a few kits over partially completing many kits and notifies the user of missing parts. They also pointed out that the design should be easy for an educator to lift, set-up, use, and store. Speed was not a huge concern if an unsupervised robot could sort a class set of kits overnight. Finally, they said a reasonable cost would be between $500-$700 for this product.

The educators we interviewed suggested that a major consumer would be for-profit STEM education centers and summer camps that showcase completed robots, leaving staff to organize the kits. Instead of having to pay a team to sort out entire kits, a robot could do all of this with minimal human interaction, reducing costs. The costs would shift from reoccurring labor costs to a one-time purchase of the robot, as well as occasional repairs. Since the robot requires minimal interaction, it would also allow educators in any environment to spend more time on other activities.

In addition to saving time and money, an automatic sorter system used in a classroom would inspire young roboticists as an example of a "cool" robot, a step up from the ones they have been building. This ties into the purpose of STEM education- to show kids how they can apply these skills in the future.

While this LEGO-sorting system on its own would be invaluable to STEM education programs, its general hardware and software framework could allow this robot to be used for other consumer/small business sorting applications. A hobbyist, for example, could use this robot to sort their own LEGOs into their own trays. In contrast, a business might be able to use the kit building system to fulfill orders from an inventory of small parts. Our focus group even suggested using the robot to sort other companies' robotics education kits, such as VEX.

Previous Work

There have been many attempts to sort LEGO pieces using robots over the past several years. Two MQPs at WPI—in 2018 and 2019—have approached this problem in vastly different ways. In addition, many individuals have published YouTube videos showcasing their LEGO sorting machines. A recent one, the "Universal LEGO Sorting Machine" is particularly interesting because the creator used 3D models to teach a neural network to recognize every part LEGO has ever produced.

The AY2018-19 A.I LEGO Sorter MQP had 3 subsystems: a Serializer, a classifier, and a Distributor. It has a similar goal to our system: to sort LEGOs using a vision system. To accomplish its goal, it used ROS for communication between the 3 subsystems. The Serializer used a series of rotating drums to separate parts, the classifier used a convolutional neural network trained on real images, and the Distributor, which uses a polar robot to sort the pieces.

Some improvements for this system the team suggested are to make it cheaper and smaller, have more reliable lighting, and add a second camera for better depth perception [3].

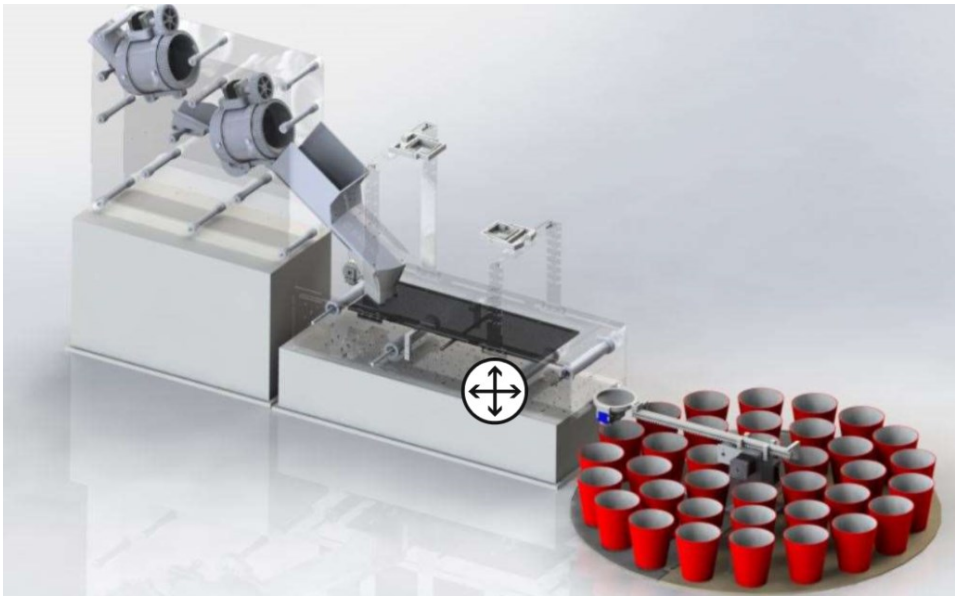


Figure 1: AI LEGO Sorter

The AY2019-20 Scalable Sort Automation MQP took the problem of sorting LEGO pieces in a different direction. Their goal was to make a general system that could sort any collection of parts. To do this, the team created a large system that performed sorting based on size, weight, and vision data. They also mathematically analyzed how changing different parameters of a sorting system affects the time it takes to sort [4].

In December 2019, software engineer Daniel West published a YouTube video introducing his "Universal LEGO Sorting Machine" (Figure 2). This machine, which is constructed from LEGO itself, sorts at a rate of 0.5 bricks/second. Using a series of conveyors and a V-shaped "vibration feeder", it pulls the parts out of a hopper and serializes them. Next, it takes a video of the part under an extremely bright light and uses a convolutional neural network to identify the part based on the video frames. Finally, it drops the part into one of 18 buckets by opening a gate along the conveyor [5]. The innovative feature of this machine is that its

universal—it can identify any LEGO part ever created, using a convolutional neural network trained on LEGO 3D models. To overcome the simulation-to-reality gap, West employs a technique called "domain randomization." Instead of trying to make the computer-generated data match the real world perfectly, domain randomization randomizes parameters like color, material, and lighting so the neural network is more robust [6]. An example of the randomized data is shown in Figure 3.

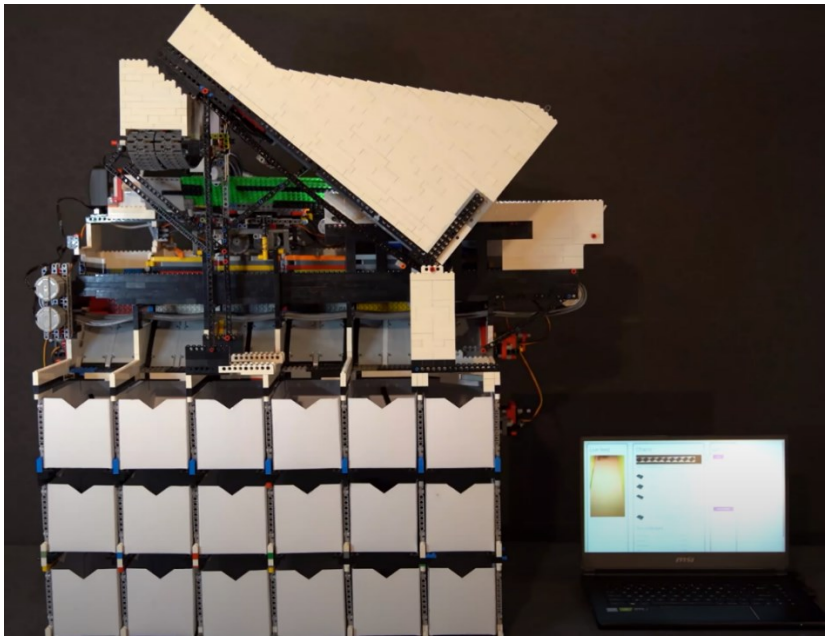


Figure 2: West's Universal LEGO Sorting Machine [5]

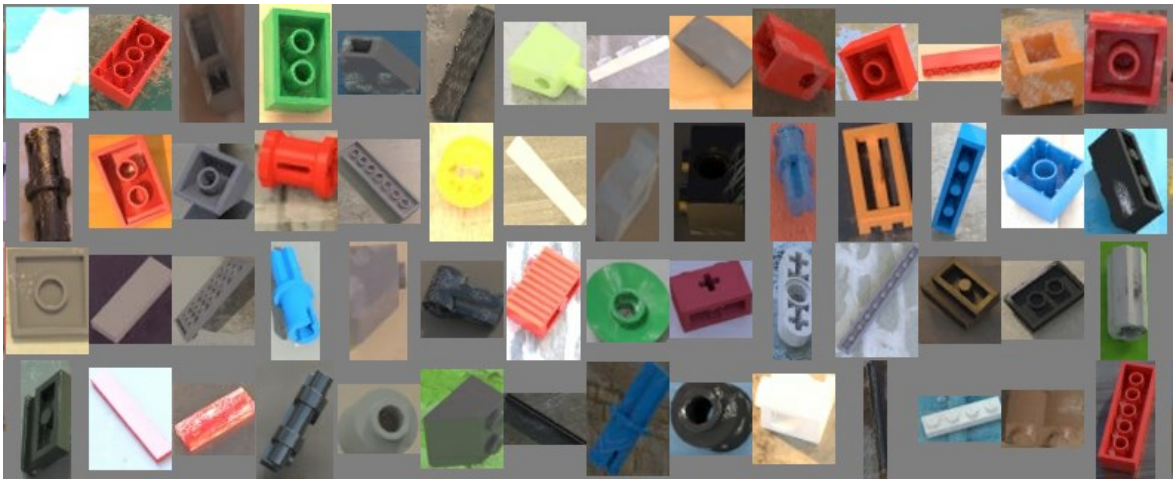


Figure 3: Domain Randomization of Lego Models [7]

Design Specifications

Balancing the consumer wish-list and a reasonable timeline for this project, the team devised a set of specifications for a LEGO Sorting Robot. A viable robot should sort the upper tray of one LEGO Mindstorms NXT Education kit (Figure 4) in 40 minutes. This equates to sorting 10 parts every minute. This set of LEGOs has 76 distinct pieces with 399 pieces total[1].

Figure 4: NXT Upper Sorting Tray [8]

Based on our research into past sorting attempts, we divided the sorting process into three subsystems: the Serializer, the Identifier, and the Distributor. As seen in Figure 5, the Serializer intakes parts from a hopper and separates them so only one enters the next system, the Identifier at a time. The Identifier assigns a part number and color to the part, then the Distributor uses that information to place the part in the designated compartment.

[1] The current generation of LEGO Mindstorms, the EV3, replaced the NXT in 2013. However, the robot should be able to switch between the two with only software training updates because the sorting tray is exactly the same and the pieces are similar. We chose the NXT because it was easier for us to find kits to test with.

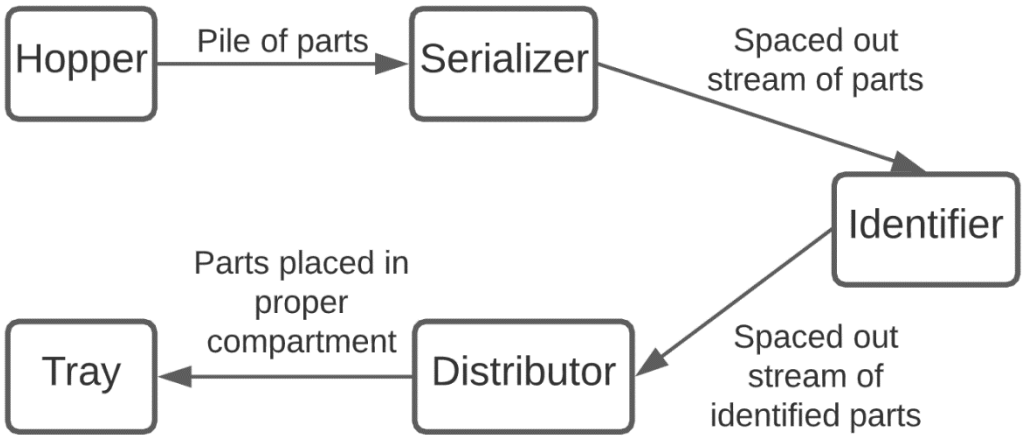


Figure 5: Sorting Process Diagram with Subsystems

The completed robot should be able to:

1. Run for at least 1 hour without human assistance
2. Serialize parts correctly at least 97% of the time and negatively impact the Identifier due to serializing errors, such as by having parts overlap, less than 1% of the time
3. Correctly identify the color and part number of properly serialized parts at least 97% of the time and unknowingly misidentify a part less than 1% of the time. Parts that are unknowingly misidentified will be parts in the set that end up in the unknown parts drawer.
4. Successfully transport parts from the vision zone to the correct section of the tray (based on results from Identifier) at least 98% of the time
5. Sort at a rate of .17 parts per second (6 seconds per part)
6. Keep track of the number and types of parts sorted and know when a kit is complete

Design Methodology

As dictated by our sorting process (Figure 5), our LEGO Sorting Robot design is divided into 3 subsystems: a Serializer, an Identifier, and a Distributor; and two part storage units: an unsorted part Hopper and a tray Storage Tower. These sections of the robot are labeled and defined in the graphic below.

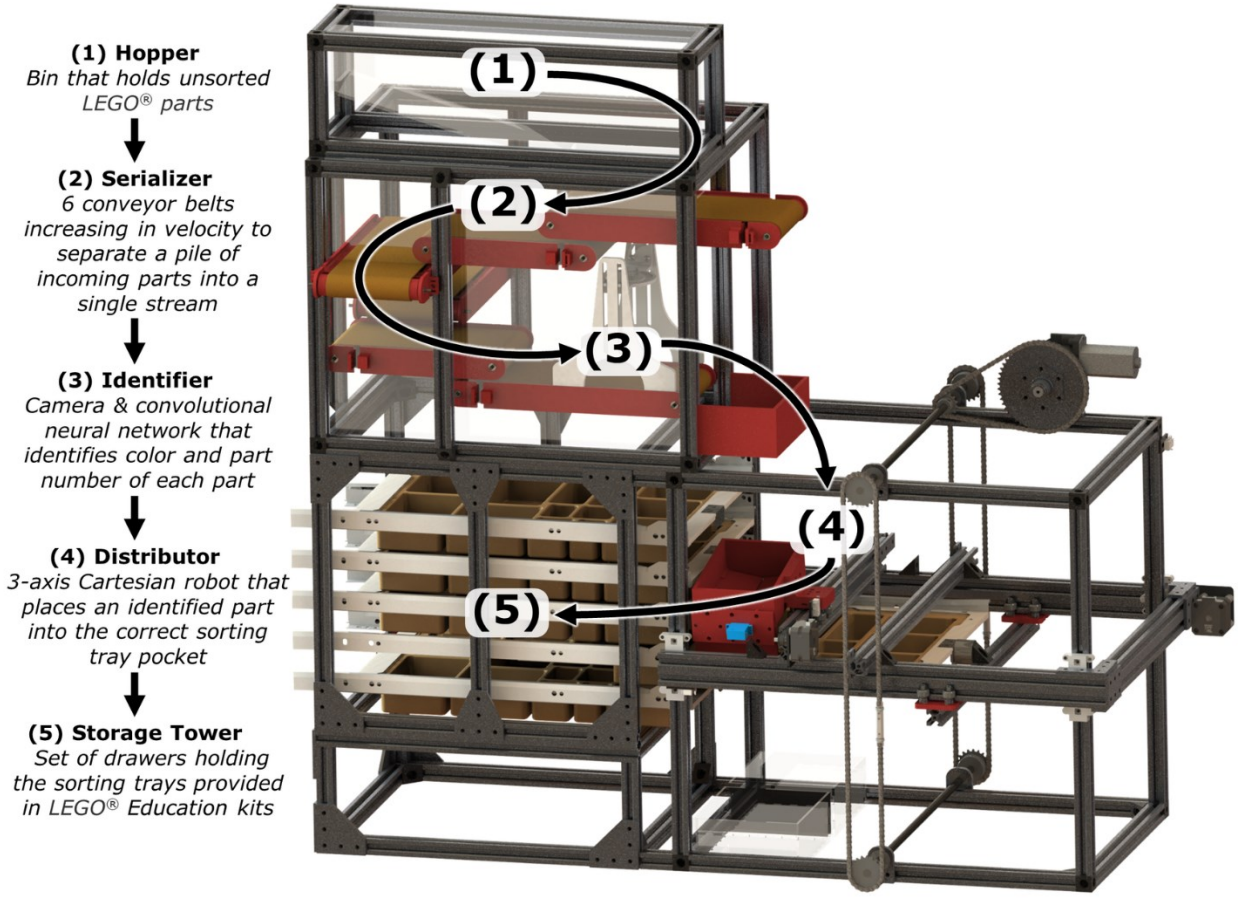


Figure 6: Overall Robot Design & Sorting Process

Parts are poured into a hopper, which in turn slowly releases them onto the main part of the Serializer. The Serializer consists of conveyer belts, each of which are each faster than the last. This results in parts getting further apart from each other. In addition, some belts are

oriented perpendicularly to each other so that parts next to each other on one belt will be separated on the subsequent one.

Above the Serializer's final belt is the Identifier camera. The Identifier's purpose is to determine which piece is on the conveyer belt so the Distributor can put the part in the correct location. This system was designed to interface with a convolutional neural network which identifies parts (similar to previous work in LEGO Sorting). From there, if the part is identified with enough certainty, the Distributor will move it to its designated location. Otherwise, the part will be put in a bin at the bottom of the robot to be reevaluated by the machine or human later. During this step, the identified parts are noted by the system so it can determine if any parts are missing, or if there are any extras.

The Distributor's role is to move parts to the correct location in a tray. The Distributor receives the location from the Identifier, and then moves there. This is achieved using a series of systems which move the end effector in the x, y, and z directions relative to the tray or trays. The x-direction is controlled by moving the end effector, which holds the part, being moved by a timing belt. The y-direction is controlled by a set of moving electromagnets which move the tray to the correct location. This makes it so the end effector does not have to slide between the system's trays; the trays are instead moved to it. Finally, the z-direction is controlled by a series of chains, which allow for the robot to switch between different trays.

Hopper & Serializer

As the first subsystem in our sorting process, the goal of the Serializer subsystem is to feed parts out of a hopper and process them so that they are "serialized," or leave the subsystem in a steady stream of individual parts. This will allow the subsequent subsystems, the Identifier and the Distributor, to process one part at a time. The Serializer should be able to serialize parts

correctly at least 97% of the time and negatively impact the Identifier due to serializing errors less than 1% of the time. An example of a negative impact would be an incorrect identification by the Identifier, rather than two non-serialized parts being placed in the unknown parts bin.

Physical Design

For the Serializer design, we considered two ideas used on past projects: the drums from the 2018 MQP and a vibrating table from the Universal Lego Sorter. However, the 2019 MQP suggested the idea of a series of conveyor belts, each subsequent one moving faster than the last, to separate out parts. Since there would need to be a conveyor belt to pass the parts under the Identifier anyway, we decided that this would be the simplest solution to try.

We improved on the conveyor belt idea, by adding a 90-degree turn, so that parts traveling next to each other would also be separated. To test if this idea would work, we created the prototype belts shown in Figure 7.

Figure 7: Serializer, First Prototype

We found that the concept worked well but would benefit from a second 90 degree turn and two conveyor belts. We also found that parts could get stuck on top of each other. To mitigate this problem, we added a passive door on the first conveyor belt (Figure 8). This door concept will be used to create the feed mechanism from our hopper, which will be mounted above the first conveyor.

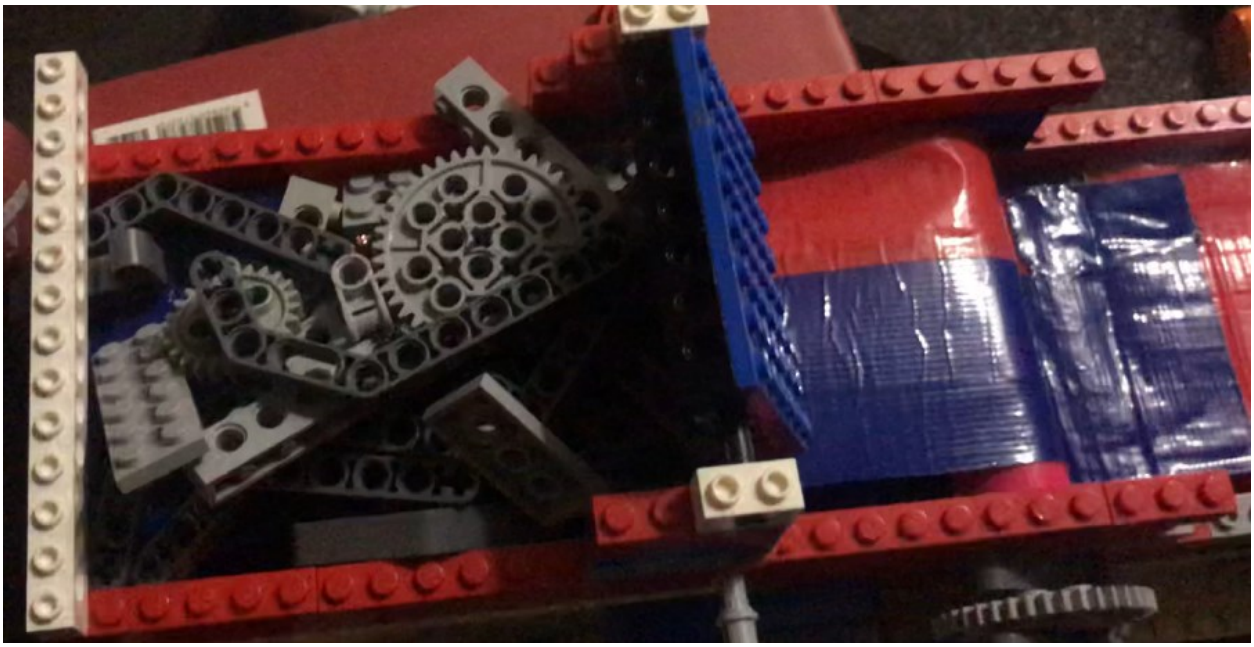


Figure 8: Serializer First Prototype, Passive Door

Belt Design

After validating that the idea worked, we designed a second prototype belt: this one with the components we plan to use on the final design. Because LEGOs are not a heavy load, we determined we could 3D print the frame. The belt is from McMaster-Carr and is made from brown neoprene rubber, which is perfect because none of the parts we are sorting are brown. To add friction to the 3D printed pulleys we coated them in rubber spray. This prototype is shown in Figure 9 below. It works well, but even though we designed to standard conveyor belt

specifications to avoid tracking issues [9], the belt still runs off to one side after a while. This is because the tensioner on the driven pulley is very hard to adjust.

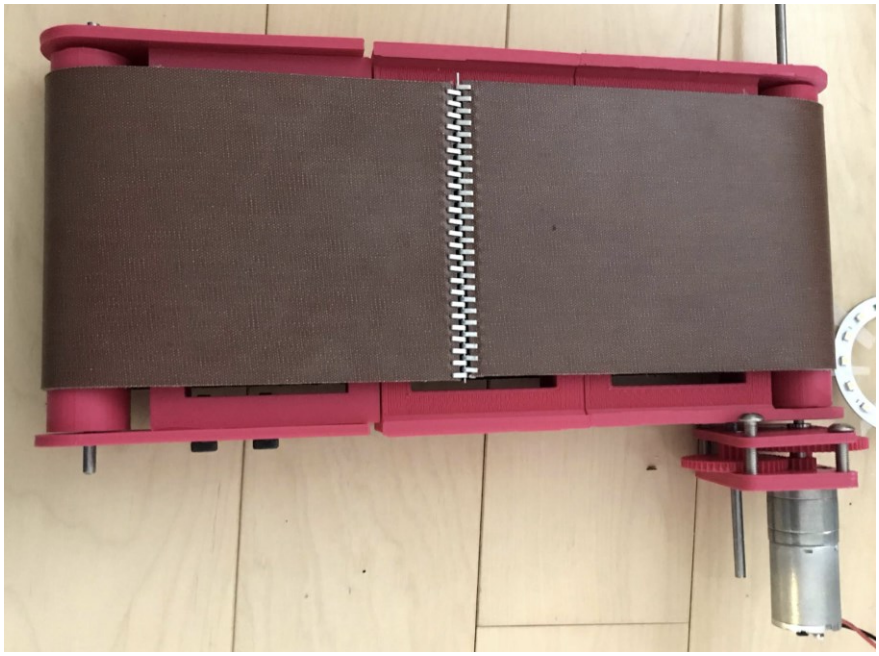


Figure 9: Serializer, Belt Construction Prototype

To improve the conveyor belt design, we changed to a jack-screw tensioning system (shown below). While this improved the belt tracking problem, it increased the profile of the belt chassis, which complicated construction of a full serializer system.

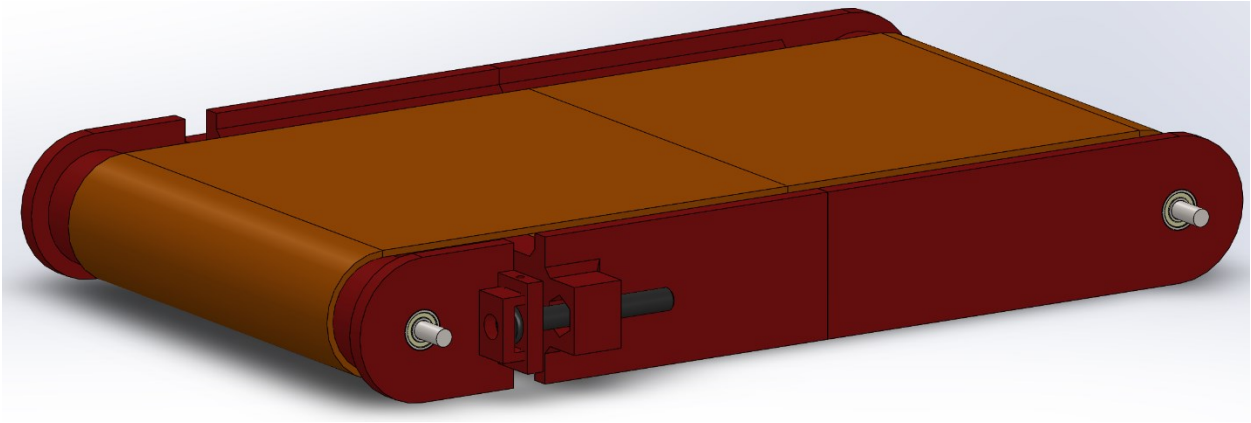


Figure 10: Serializer, Jack-Screw Tensioning Design

The intended layout of the completed Serializer system is shown in Figure 11. The parts move in the direction indicated by the arrows. This system sits above the Distributor system, and the hopper is located above the top belt. Rails should be included to keep the parts from falling off the side of the belt.

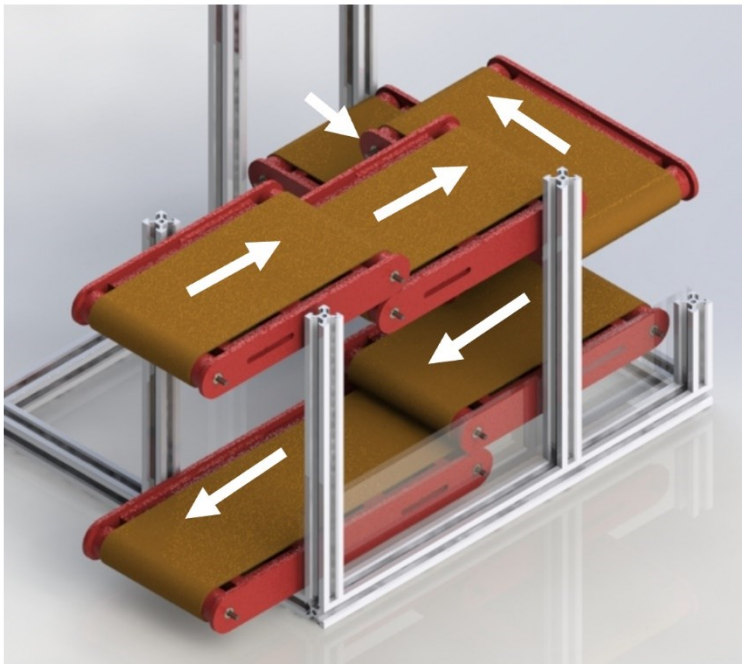


Figure 11: Serializer Belt Layout

To simplify controls and lower costs of this system, closed loop timing belts could be used to set the speed of each belt in relation to the next. The base rate of speed may be slowed at times deemed appropriate in order to allow for a process taking excess time to finish being performed. Since our target audience preferred reliability over speed, this is an acceptable proposition to them.

Ideally, the whole system could be controlled with one motor. Based on our target sorting speed, where the final belt would run at 1.6 cm/s, we calculated the power needed for this motor to be 0.6 W. These calculations can be found in Appendix A.

Finally, our proposed hopper design is shown below. This design is incomplete, as it would most likely need a non-passive latch at the bottom to control the rate parts entered the system, but this figure illustrates the geometry and construction of the hopper.

Figure 12: Hopper Design

Identifier

The goal of the Identifier subsystem is to identify the part type and color of each part in the NXT set correctly at least 97% of the time and unknowingly misidentify a part less than 1% of the time. These metrics assume the parts have entered the Identifier serialized correctly. The team reviewed various methods to identify a LEGO part, including measuring physical properties like size and weight, but, for simplicity, ultimately decided to use a vision system.

The physical build of the Identifier is a camera mounted about the end of the Serializer system. The software component uses a convolutional neural network to identify the part type and traditional computer vision techniques to identify the color. To create training data the neural network, we planned to use the Universal Lego Sorter's technique of rendering digital 3D models of Legos using domain randomization.

Physical Design

In a final design, the height and angle of the camera will be fixed, but, for testing, we designed a 3D printed mount where these parameters could be adjusted (Figure 13). For the camera, we are using an 8MP IMX219 camera with a 77° field of view. We are also using an LED ring light to keep the lighting consistent. The prototype setup with the Serializer conveyor belt below is shown in Figure 12.

Figure 13: Identifier Prototype CAD Model

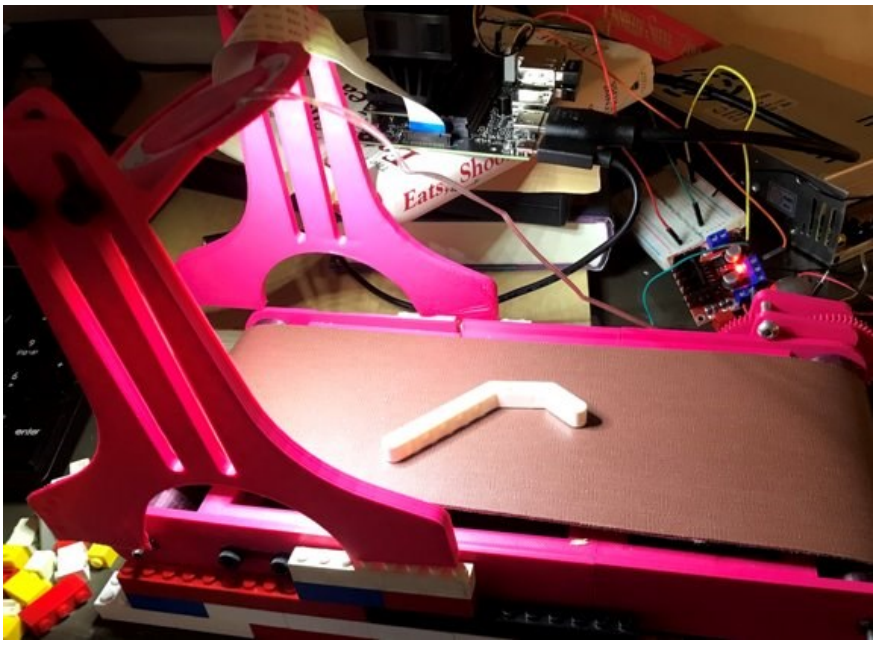


Figure 14: Identifier Prototype

Hardware Selection

We based the hardware selection for the whole system around the hardware requirements for the Identifier. We need a large GPU to train and run the neural network, so we chose to use the NVIDIA Jetson Nano 2GB. This single board computer has a 128-core NVIDIA GPU, is specifically designed for machine learning, and, at only $60, fits into the budget for this product. It also has 40 GPIO ports, which mirror the ones found on the Raspberry Pi, which we use for motor and sensor control, instead of purchasing another controller.

Software Design

We also based the software selection of the whole system around the requirements for the convolutional neural network. We chose to program in Python 3 because it is the most well supported language for machine learning. For the Identifier, the software had four main tasks: processing the camera feed, creating the training data, building the neural network, and

determining the part based on the part database. The Identifier will accomplish these by taking pictures of the parts going through it.

Camera Feed Processing

To feed our images into the neural network and identify them, we need to be able to process them from the camera feed. We will do this using a background subtractor mask. An example of this can be seen in Figure 15, but we will also add a blur to reduce noise. Once the part is separated from the background, we will identify the color by averaging the colors in the masked section crop the camera field to that image, turn it into grayscale, and feed it into the neural network to identify the part type.

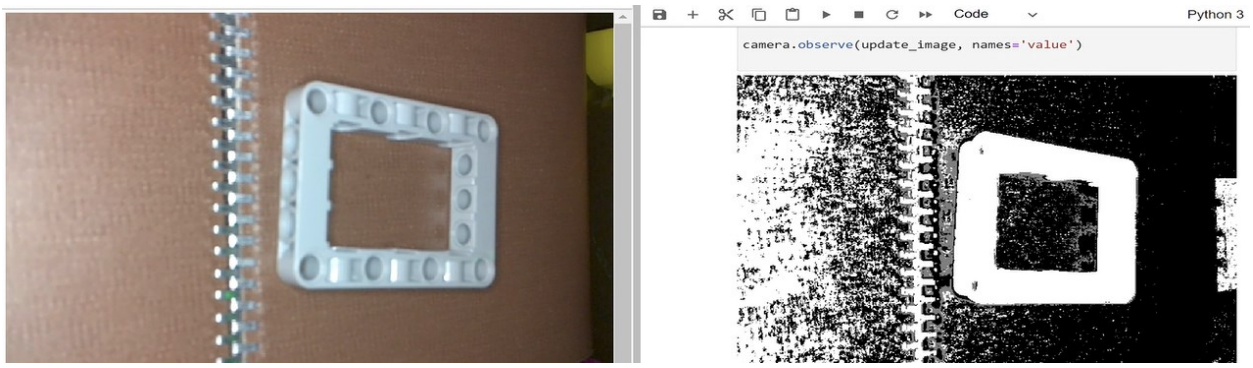


Figure 15: Background Subtractor

Training Data

To get appropriate outputs from the neural network, it must be trained, and to that, data must be collected for training. Traditionally, data collection for training would involve taking hundreds, if not thousands, of pictures, and then labelling them. To circumvent the time-sink of that process, we tried a technique introduced in 2017, and used by the Universal LEGO Sorter called domain randomization. This technique allows us to use computer generated data to train machine learning system. Traditionally, using computer generated data to train a neural network

have not worked, mostly because attempts were made to make the computer renders closely resemble real life. The computer renders cannot perfectly match real life though, so the AI struggles to properly identify the real-life images. However, by randomizing parameters like lighting, noise, background, and rotation, it makes the pool of information that the neural network is able to understand much bigger, so then it is better able to understand the real-life images [6].

To generate this data, the first step was to obtain the part numbers present in the subset of NXT kit parts that are being sorted, (this can be accomplished either by hand or using the BrinkLink website API). Next, the CAD files for those parts are obtained from Ldraw, a CAD program designed to be used with LEGO bricks. The program has a CAD library of every LEGO manufactured. From there, the CAD files are converted into Blender files. The reason for the conversion is that BlendTorch [10], a Python library, uses the Blender API to automate the image generation and domain randomization process for use with the machine learning library, TorchVision. The output from this process is a series of tensors and labels, which are then imported into the Neural Network to be used as its training data.

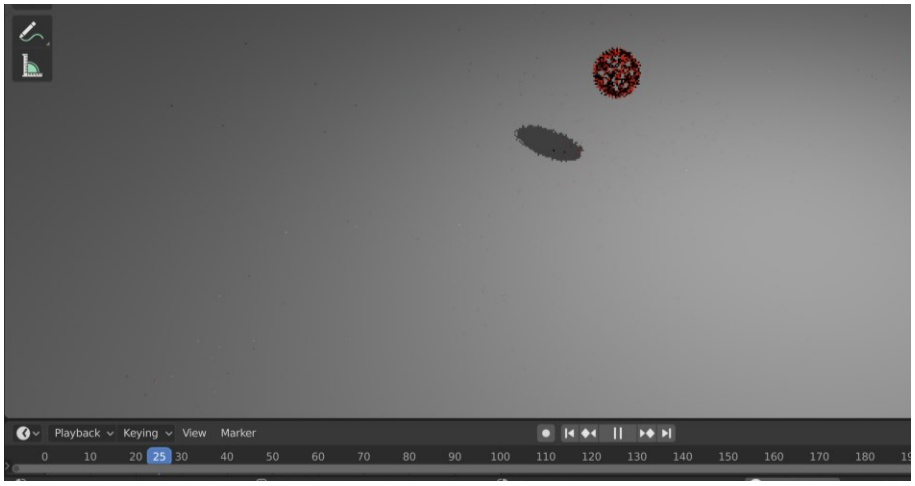


Figure 16: Screen Capture of Blender Animation for Domain Randomization

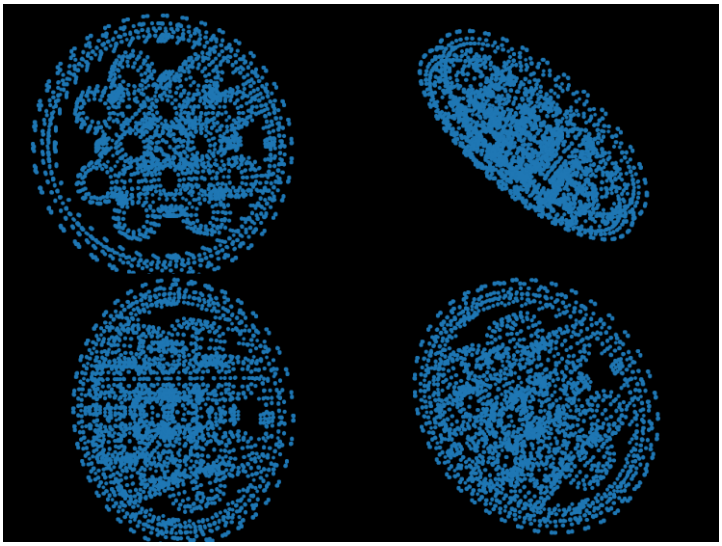


Figure 17: Visualization of the Tensor Outputs for 64T gear (part #3649)

Neural Network

The Neural Network is designed to take in images from the Identifier, which have been cropped by a bounding box, removing as much extraneous data as possible. In addition, the images are converted into grayscale to eliminate any challenges that coloration will pose since the color is determined separately. In addition, this will decrease the amount of data needed for each picture since each pixel can be encoded into a single value, instead of the combination of red, green, and blue needed for RGB-encoded photos. In order to train the Neural Network, 90% of the PyTorch tensors will be used, leaving 10 % for verification. This data will be split evenly by part, ensuring that the neural network receives an equal amount of training for each part, preventing a disparity between its ability to distinguish between certain parts wherever possible.

When being used by the robot to identify parts, the Neural Network will work as follows. It will start off by intaking a photo taken by the Identifier, which has been converted to a 2d array of 8-bit brightness values and cropped by a bounding box. Next, they will go through series of layers which will down-sample the arrays with 3x3 max-pooling. In this case, that would

result in the array's values in a 3x3 subsection of the matrix being replaced with a single value in the next layer, which will contain the highest number in that subsection. This technique allows for the data to be reduced, while preserving the shape and edges, which are the aspects that the later stages will need to know. After the down-sampling is complete, the resulting array will be flattened, turning the array from a 2d array to a single dimension. The dense layers, which are the heart and soul of the Neural Network, are designed to take in a one-dimensional array, so this step must be taken. The dense layers contain perceptrons, which will adjust their values based on the training data, to eventually return the probability that each part is the one that was just obtained by the Identifier.

Color Algorithm

To calculate the color, an image of the part, with its background RGB pixels replaced with [0,0,0] is made. From there, the R, G, and B channels of each pixel which are not [0,0,0] is averaged. From there, the resulting average RGB value is compared to the reference value for each color in the set, which can be found in in ColorDB.py. To facilitate testing, a GUI was created, as seen in Figure 18. which could calculate the color of a user selected image and convert it to Greyscale, which would be used by the Neural Network.

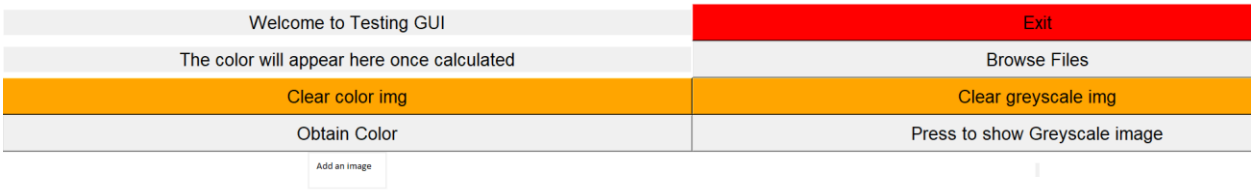


Figure 18: The Image Testing GUI, on startup

Part Database

The Part Database is responsible for determining what part has been spotted by the Identifier. This is done by inputting the color and part number/shape of the part. From there, those two parameters are appended to create a "key" which will be used to search for the part. However, before looking for the part, a preliminary search takes place to see if it is present in the database, which in this case is modeled as a dictionary. If there is no value with the key, the program will keep searching the database as long as the certainty of the part remains above 25%. If it falls below this threshold, the function will return unknown, which from there will be treated as the name of the part. If a part is found, another search will take place, but this time, the name of the part will be returned. Once the name of the part has been determined, it is passed to the Distributor software, and the tray database to determine where the part goes.

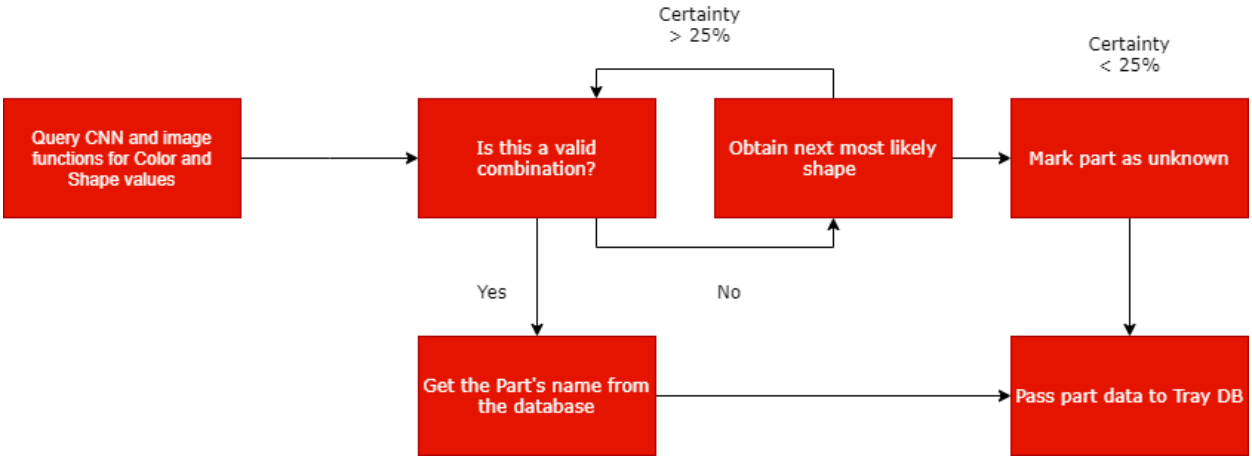


Figure 19: The flow of the Part Database

In short: there are two outcomes for a part. If there is no shape and color combination with a high enough certainty, it is unknown and it will be placed into the tray for unknown parts. If there is a valid combination, it is valid and the TrayDB will place it in the first available place.

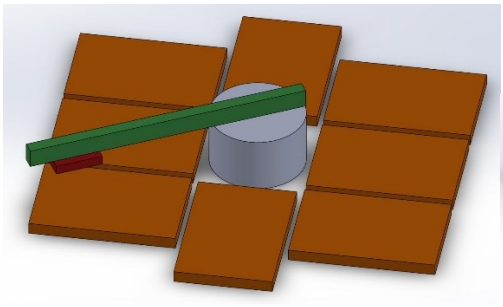
Distributor & Storage Tower

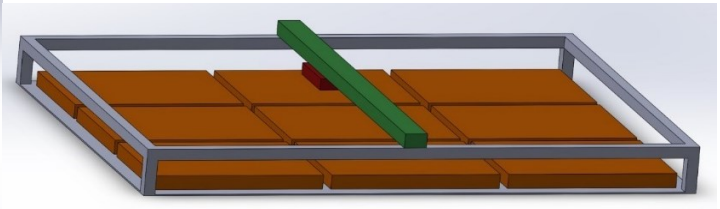
The Distributor's job is to sort the identified piece into the correct storage location. To be most useful for the consumer, LEGO Robotics educators, we determined that the Distributor should be able to sort multiple trays at once. Ideally, the Distributor should be "small," which means being able to fit well inside a classroom, both during use and for storage. It must also be easy to move and set up. as well as have variable or expandable tray capacity.

Physical Design

Design Ideas

We produced four different Distributor design ideas, based on the goal for the subsystem to sort multiple trays at once. These ideas, which can be seen in Figure 20, were a polar robot, a horizontal array, a drawer stack, and a drawer-less stack. The orangey-brown boxes represent trays, the green and red represent moving gantries, and the blue represents an elevator.





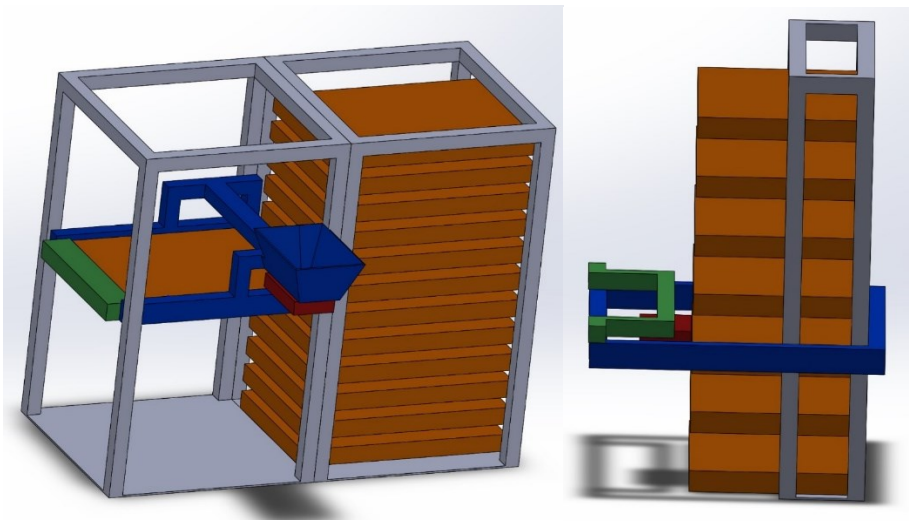


Figure 20: Distributor Design Ideas

We quickly eliminated the polar and drawer-less stack ideas. The polar was removed because the end of the arm would be moving too fast for comfort and safety, posing a hazard to children in its classroom environment, and the drawer-less stack was decided against because the space between trays would make it too tall to be practical. That left us with two options, the horizontal array, and the drawer stack. The horizontal array, having only two axes, was far simpler, but took up a lot of floor space. The drawer stack was more complex but took up far less space. In addition, the drawer stack would be easier to add capacity to. When asked about their preference, the consumers we interviewed preferred the drawer stack.

To determine which option would be best and most feasible, we ran a simulation in Python where all the parts in 12 NXT trays were mixed randomly, and then distributed one by one in the first tray that needed the part. The results of this analysis can be seen in Figure 21 below, where you can see which tray, out of the 12 possible, each part is sorted into.

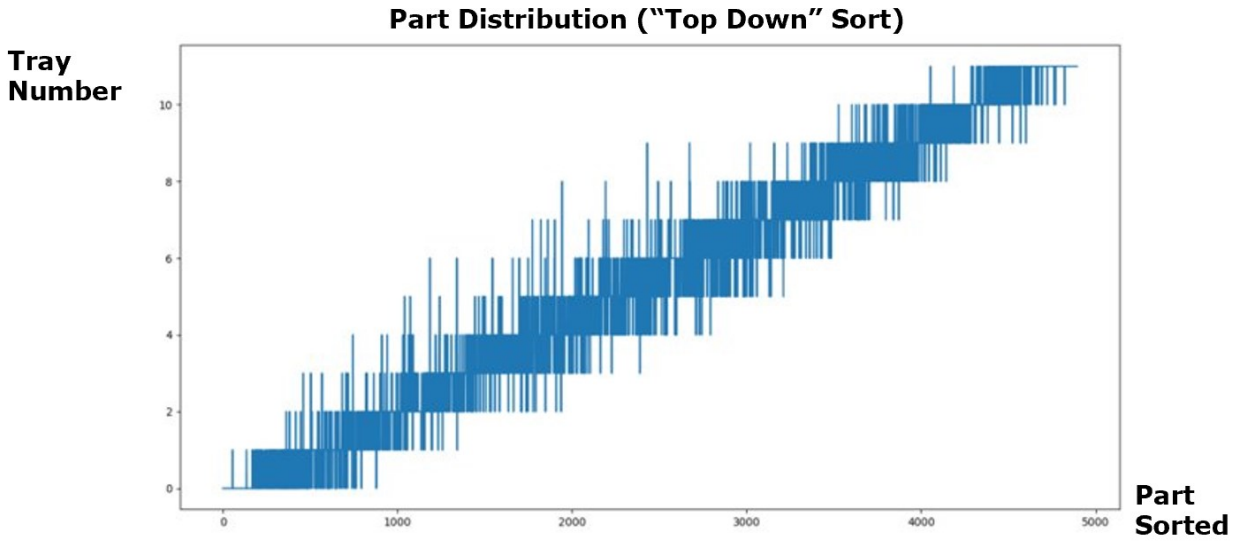


Figure 21: Results of Distributor Sort Analysis

This simulation gave us some insight into how our two Distributor options would move. For the drawer stack, the graph would mirror the movement of the vertical "z-axis", which would have to switch heights and drawers about 50% of the time. However, the z-axis does not have move extremely far each time it switches heights, as most of the time it simply switches between neighboring trays. For the horizontal array, however, this graph illustrates how the robot would first fill up the trays closest to the part pickup point. As the sort went on, the Distributor would become less and less efficient, as it would have to move a further distance between the part pickup and the tray to drop-off. This analysis, along with speed calculations to ensure the robot could meet the 6 second part processing time, lead us to choose the vertical drawer Distributor design.

Final Design

The vertical drawer system, which can be seen in Figure 22, has a series of trays, stored vertically in drawers. In front of the drawers is a 3-axis cartesian robot, which puts parts into the correct pocket. This is accomplished by first dropping the part into the end-effector, which

simultaneously moves to the height of the targeted tray. Once the part has arrived, the tray drawer is pulled open using an electromagnet, until the desired y position is reached. While the drawer is being opened the end-effector moves in the x-direction until it is above the desired pocket. Once the end effector and tray are in the goal location, the end effector opens and drops the part into the pocket. From there, the tray is returned to its initial position if needed and the cycle continues again for each part.

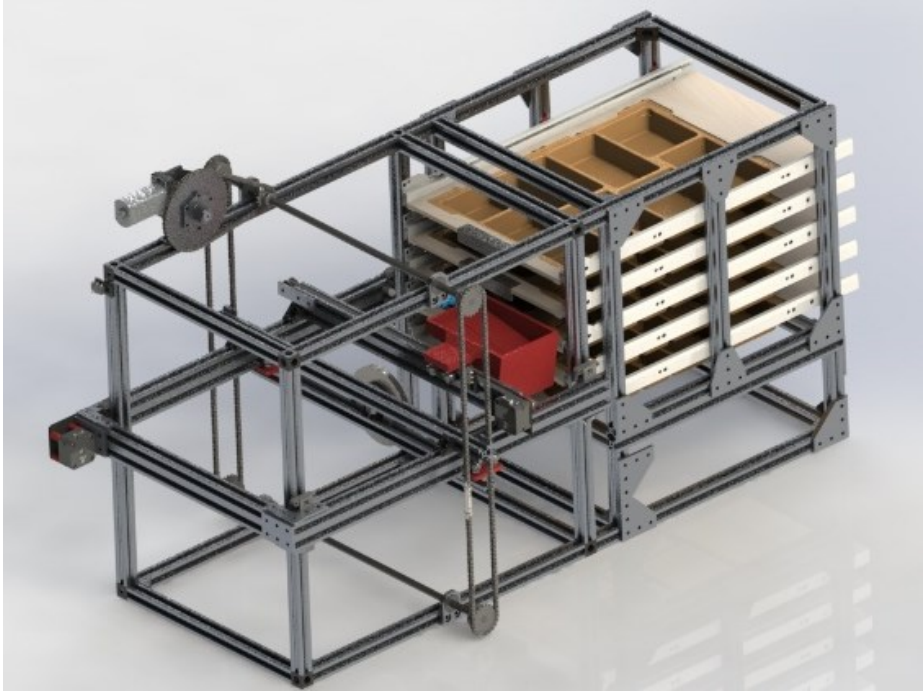


Figure 22: Distributor CAD

The first part of this design, the Storage Tower (Figure 23), is made of off-the-shelf drawer slides to reduce cost. Each drawer is built with laser cut wood that fits the trays inside, and a steel pull in front allows it to be pulled by an electromagnet attached to the Distributor. The empty space at the bottom is so that the Distributor mechanism can rest there at the end of the sort and the drawers can be pulled out easily. To increase the capacity of the storage tower, one must simply increase the height of the structure and add more drawers. This allows for a Distributor design customized to each classroom's needs.

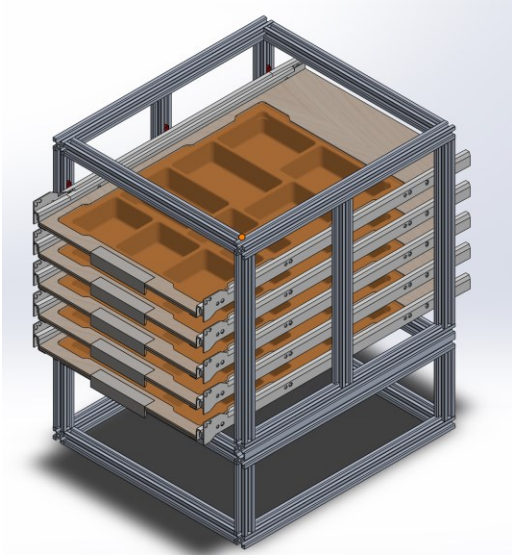


Figure 23: Distributor Tray Storage Tower

The x and y axes (Figure 24), are actuated by wheeled gantries that slide on the v-slot aluminum extrusions. These are powered by timing belts and stepper motors, as calculated in Appendix A. Each axis has a limit switch to zero the axis position. The x-axis holds the end effector and is responsible for moving side to side above the tray to drop the part in the correct place. The y-axis has an electromagnet that attaches to and opens the drawer. It can position the correct compartment of the drawer under the x-axis, to control where the part is placed in the y-direction. The whole frame that holds these axes slide up and down to manipulate the other trays.

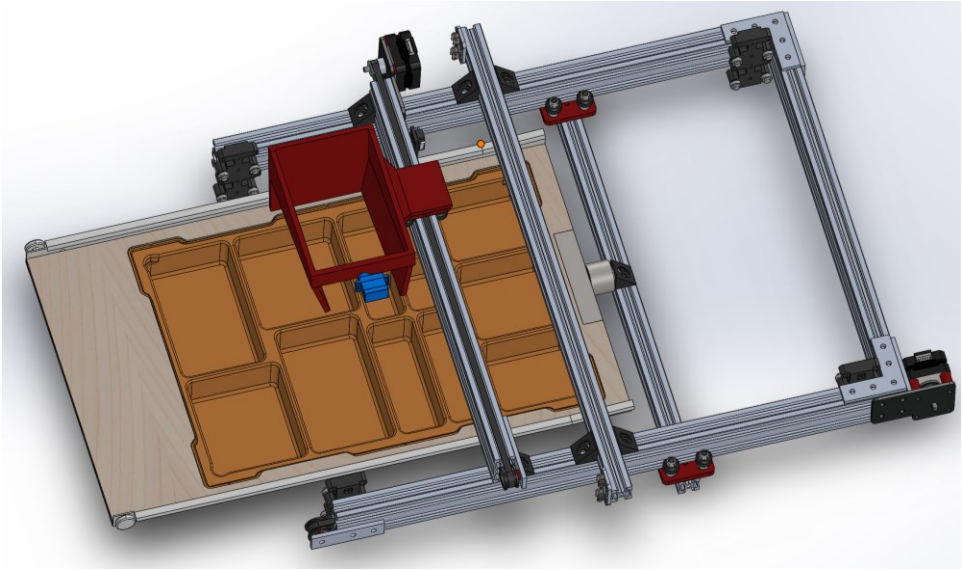


Figure 24: Distributor X and Y axes

Figure 25 shows the vertical movement, or “z-axis” mechanism. The structure that holds the x and y axis slides up and down on a chain-driven elevator system to manipulate other drawers. This system uses a snow blower motor, which has a worm-gearbox to prevent back drive, and an ultrasonic sensor to track the elevator’s position. Power calculations for the motor can be found in Appendix A. This system uses chain so that it can be easily modified to extend the height to add capacity for additional tray drawers. A user-adjustable inline chain tensioner on each chain ensures the chain is correctly tightened.

Figure 25: Distributor Z-Axis

The end-effector bucket (Figure 26) holds the part, which drops in from the Serializer and Identifier above. The bucket is tilted, so that parts fall to one end, then the doors on the bottom will open to match the width of the tray-compartment below it, so the part falls into the correct tray. The end-effector was 3d printed and powered with one servo.

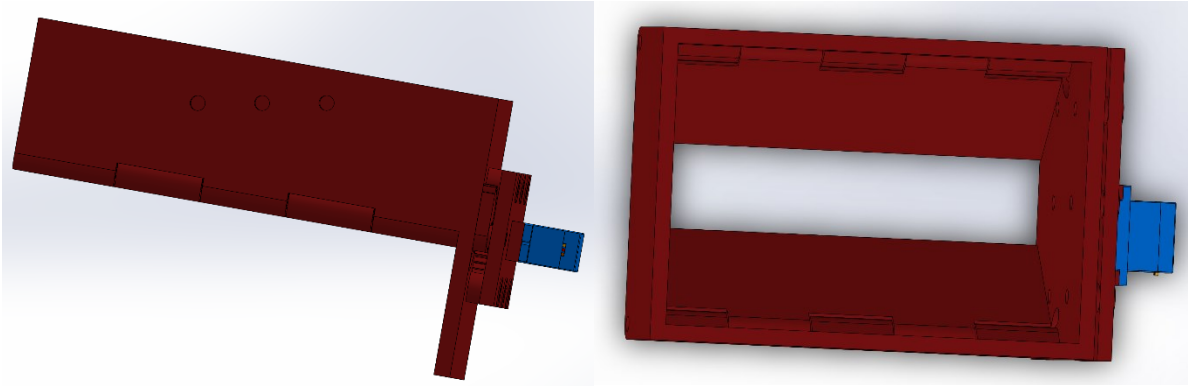


Figure 26: Distributor End Effector

To ensure that the physical design worked, we first constructed and tested the z-axis elevator. Once we conformed binding of the corner slides was not an issue, we constructed and tested the rest of the Distributor, along with the storage tower.

Software Design

The software for the Distributor is designed to be as simple as possible, and can also be found in Appendix C. The end effector is told where to move based on the destination location of the part. The destination location is found by using the tray database, which tracks what parts are in which trays, and how many more parts each pocket can hold.

The Distributor movement function, goTo, takes in a list of three numbers, which represent the destination x, y, and z values, and a pocket number, which it obtains the amount the claw should open to ensure that the part will end up inside the pocket. Opening the claw to far could cause it to fall into another, and not opening it enough could prevent some pieces from not ending up inside of it. goTo‘s job is to tell the individual motors and magnet to do. It performs the following sequence:

1. Ensures that the robot is in the correct state to start the sequence, which consists of closing the end-effector to prevent the part from prematurely vacating it
2. The Y-axis motor moves away from the trays, and into the position for the part to be placed into it from the Serializer
3. The X-axis motor moves into place to receive the part
4. The part is dropped into the end-effector by the Serializer
5. The robot moves to the z and x positions necessary to place the part
6. The y axis motor moves the magnet into position (self.CLOSETRAYY) to attract the tray, and the magnet is turned on .1 seconds after it is reached.
7. The y axis motor moves the magnet to the goal y position, resulting in the end-effector ending up on top of the pocket.
8. The magnet is turned off, and the claw is opened, placing the part into the pocket

9. After a .25 second break, the claw is closed, ensuring that it had enough time to open.
10. The X and Y motors are told to move to –1000, resulting in them hitting the bumper switches on the x and y axes, ensuring that their positions stay calibrated
11. The Y axis is moved to 100, since leaving it at 0, the reset position, lead to trays getting damaged when manually moving the Z-axis

Low-level Motor Code

The Distributor's lower-level motor code is primarily controlled by a function called MotorGoTo. Its parameters are the goal position for that motor, as well as the name of the motor, which is either "X", "Y", or "Z". From here, depending on the motor, operation starts to diverge, but the x and y axes work almost identically. For those two motors, to find the direction pin value to be sent to the stepper controller, the direction needed for it to reach the goal position is determined and set based on the calculation. From there, a while loop runs until the end effector's position on that axis equals the goal, as determined by an estimation of the distance traveled by each step, or the axis hits the reset switch while moving in the negative direction. As long as the while loop runs, a square wave with a period of .002 seconds low and .002 seconds high is sent. If the robot tries to send a robot too far in the positive direction, the command will be ignored since this would result in the motor trying to move to a position it cannot physically reach and could damage the motors.

The Z-axis motor functions differently. Instead of adding a set value to the calculated position each time the loop iterates, it uses an average of fifty ultrasonic readings to determine the end-effectors z-position. Also, since it is a conventional DC motor, it wants a duty cycle to determine the amount of effort exerted by the motor. To calculate this value, a PID controller is used. The error value for the P value is multiplied by the difference between the robots desired

position and its measured position, the error value for the d controller is determined by comparing the previous iteration's error to the current one, and the I controller's error value is determined by making cumulative sum of all the errors present in the current motor movement attempt. The kI ended up a fair amount greater than the kP or kI values due to how the robot had a high steady state error. This could be attributed to static friction in the system making the z axis movement require about 30% duty cycle to move, according to our calculations. calculate this value, a PID controller is used. The error value for the P value is multiplied by the difference between the robots desired position and its measured position, the error value for the d controller is determined by comparing the previous iteration's error to the current one, and the I controller's error value is determined by making cumulative sum of all the errors present in the current motor movement attempt. The kI ended up a fair amount greater than the kP or kI values due to how the robot had a high steady state error. This could be attributed to static friction in the system making the z axis movement require about 30% duty cycle to move, according to our calculations.

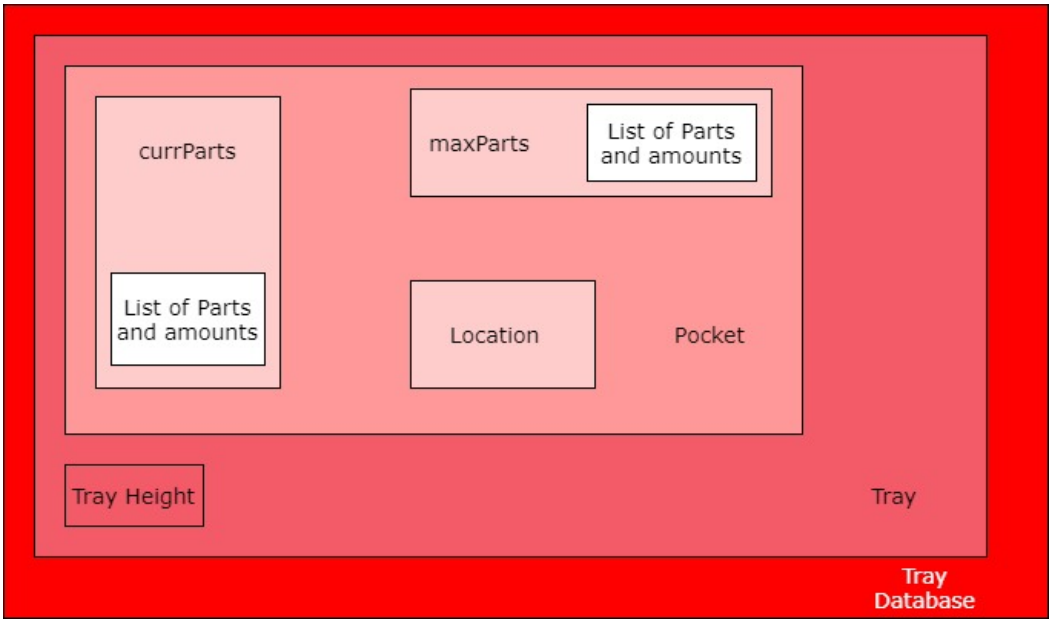


Figure 27: The Data Structure of the Tray Database

The goal of the Tray Database is to determine where a part should be placed. To fully understand the inner workings of the Tray Database, its data structures must be discussed. The outermost layer of it is the aptly named TrayDB, which contains the entirety of the data. Within it is a list of trays. Each tray consists of a list of pockets and a height. Each tray is assigned a height because all trays that the robot can accomidate are parallel to the ground, making all of the pockets in that tray have the same z-value. The z-xis is perpendicular to the z-bumber switch, which is also the case for the x and y axes relative to their respective limit switches. Each pocket consists of a location, a list of current parts (currParts) and a list of the maximum parts a pocket can hold (maxParts). The location is the corner oriented to the near left when with reference to the side opposite the storage tower. This location, paired with the height of the tray, is sent to the Distributor to determine where it should move. currParts and maxParts each contain a list of the type numberOfPart. numberOfPart contains the name and amount of a part, acting as the actual

database portion of the database as the other portions merely serve to help direct the program to the correct tray or pocket, or tell the Distributor where to go. In order to determine which pocket a part should go into, a dictionary named PartToPocket takes in a part name, and returns its pocket number. This is a shortcut over the previous method, which checked every pocket. The TrayDB places a prt as shown in Fig. 23. However, there is an extra level of safety added, which is a check to see if the number of current parts in the pocket and tray, allowing for the algorithm to know if the part was correctly placed.

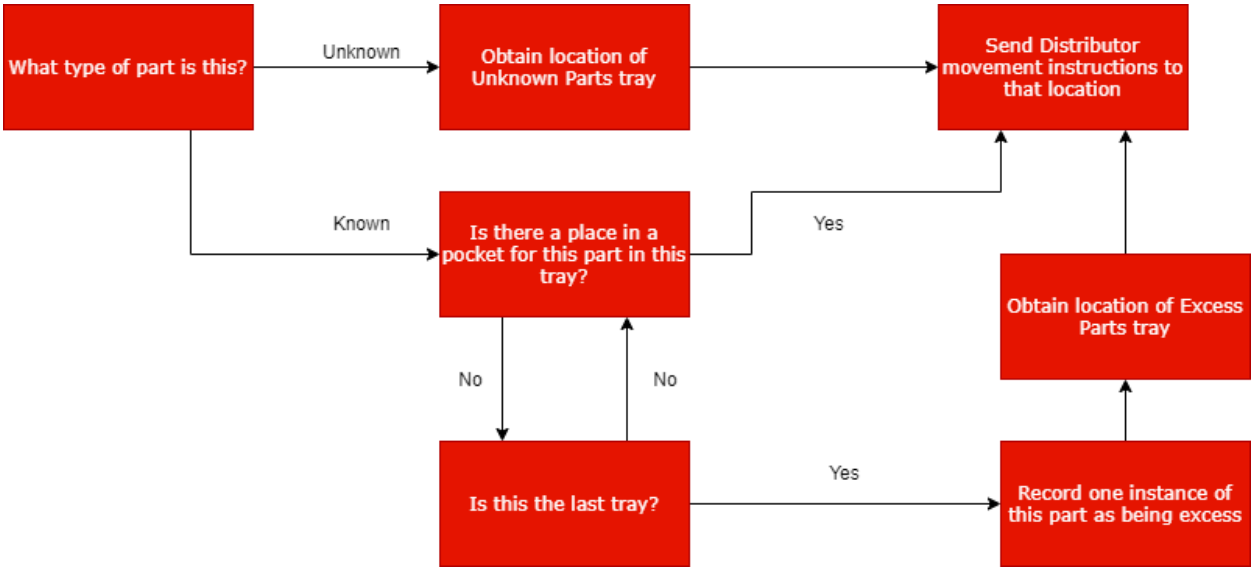


Figure 28: The Program Flow of the Tray Database

While the TrayDB represents the entire storage stack, each tray represents a single physical tray in the system. Each tray has the layout displayed in Fig. X. Those trays were than given the numbers shown in Fig. Y. In order to calculate their positions, the trays were centered manually on top of the pocket, and then the x and y position of it was recorded. Their z-positions were calculated by measuring the distance between the trays as the offset between them, and the z-position for the top tray, since z = 0 at the top, was experimentally determined. In order to move the end effector manually, the Controller GUI was used.

Figure 29: Tray Layout documenting where each part goes

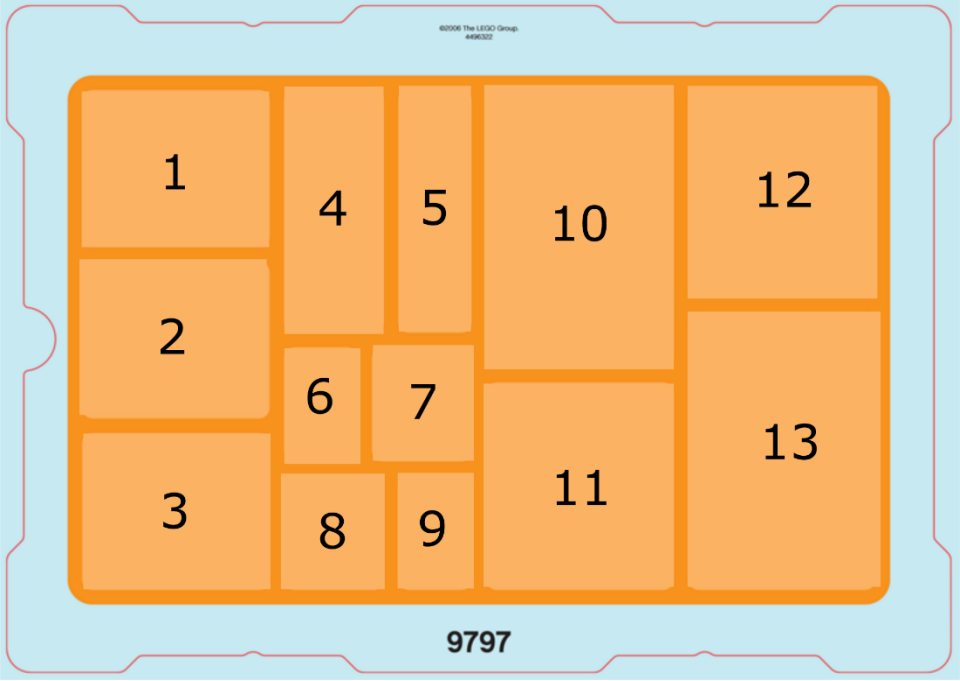


Figure 30: Pocket Numbering

The goal of Controller GUI was to be able to manually move the end effector and trays, and to be able to control other individual mechanisms, like the end effector claw, electromagnet, and Ultrasonic sensor readings, allowing for the code and mechanical systems to be tested without having to create any additional code.

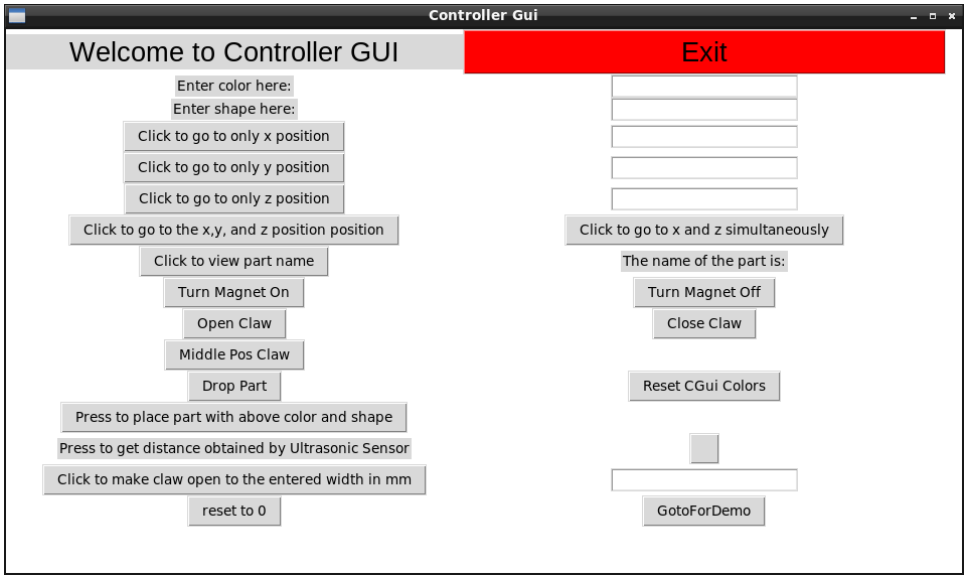


Figure 31: A View of Controller GUI in its default state

The first function that the GUI performs when launched is sending X and Y to –1000, ensuring that the end effector always starts at (0,0), which is what the program assumes it starts at. The *reset to 0* button does this as well, but only when the user clicks it. Going from top to bottom, here is what the rest of the buttons and fields do. Starting at the top, there is a color and a shape field. The color and shape, respectively, would be entered into these text boxes. From there, a user could click *Click to view part Name* to see what the part is internally referred to, as well as to see if the values were typed in properly, or if it is a valid part. Also, clicking *Press to place part with above color and shape* would have the robot perform the goTo function, using the entered shape and color. The next three fields are to enter x, y, and z values for the robot to

treat as goals. Clicking *Click to go to _ Position* would only move the robot to the position entered in the respective box. The button beneath those three labels, *Click to go to the x, y, and z position* allows the user to manually set the goal positions for goTo, making them not need to know parts to test out the robot. The six buttons in the rows beneath *Click to view part name* are toggles to manually control mechanisms, specifically the claw and magnet. *Drop Part* was intended to move the Serializer until it drops the part, but the Serializer was not built in time. Beneath them is a button to obtain the Ultrasonic Sensor's distance. This is done over I2C to an ADC, which converts the sensor's voltage into a digital value, which the Jetson Nano then converts to the distance it represents, using the equation below:

$$distance\ (mm) = \frac{ADC_{VALUE}}{8} * \frac{520}{3270} * 10$$

This relationship was created by scaling the possible voltages from the ultrasonic sensor to the values returned by the ADC, and then converting those values to the distance that the sensor detected. The *10 at the end was added to convert from cm to mm, the unit that the robot uses. Finally, there is a field to enter a value in mm to test if the claw would open to that value after hitting the appropriate button.

Integrated Electrical System

To reduce costs, the whole electrical system was designed to be controlled using the Jetson Nano, and to be compatible with the distributor motors, it ran on 12V DC power. To choose a power supply, we calculated how much power each subsystem required, and found that, at maximum reasonable loads, a 250W power supply would suffice. Because the robot is meant to be stationary, we used a 12V AC-DC converter. To run our system's 5V DC servo motors, we used a 12V-5V step-down DC power supply module. The two shortcomings of the Jetson Nano as an GPIO device are: the maximum current supplied on the output pins is 1 mA, and there are

no physical PWM signal drivers on board. To compensate for both these issues, we used an I2C to PWM breakout board (PCA9685), for both sending our motor controllers PWM signals, and sending output signals with enough current to trigger a relay.

Results & Discussion

The main goal of this MQP was to design, build, and test a robotic system to sort LEGO Mindstorms pieces for a classroom environment. We believe the first goal: to identify criteria this product would need to meet and design a useful sorting product was met (see Figure 32 below). However, due to project time and team size restraints, we had to cut back on the building and testing goals. Because of this, we decided to focus our energy on the parts of our robot that were distinct from previous projects. These were the Serializer, because we were using a different separation method from previous work, and the Distributor/Storage Tower combination, because that part of the robot added an innovative storage solution for our use case. Two previous projects, the A. I. LEGO Sorter, and the Universal LEGO Sorting Machine, made it clear that sorting LEGOs with a convolutional neural network is possible.

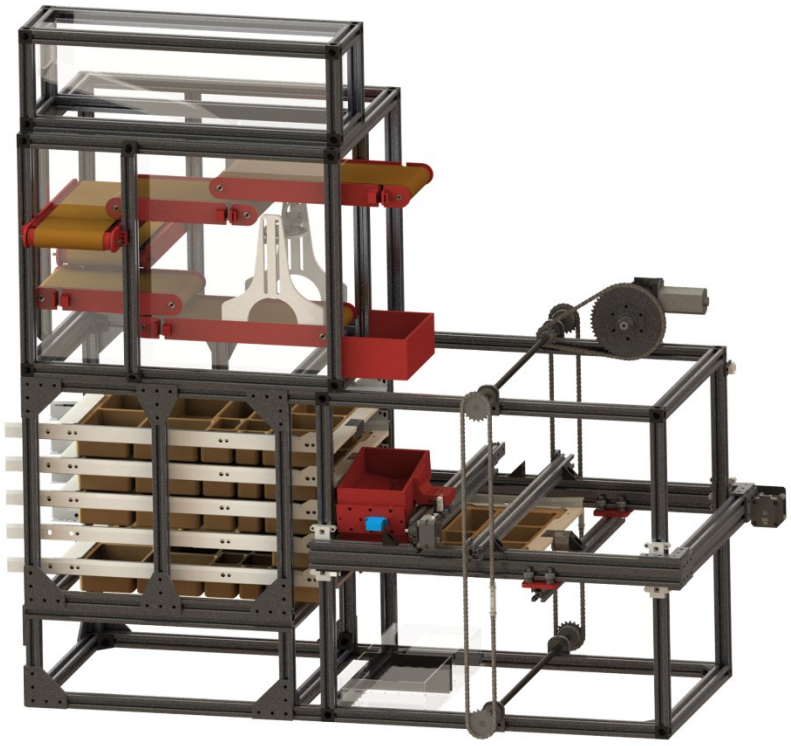


Figure 32: Final System Design

Serializer

With the Serializer design, we managed to design, build, and test an easily tension-able conveyor belt with a 3D printed frame. This design will be useful in the future because reliable conveyor belts were needed both for our Serializer design and the Identifier. However, we found that although the concept of serializing using conveyor belts works, they are not actually the cheapest or most convenient form of serialization since we found that the tensioning needs to be very precise. Expecting our consumer, a teacher, to adjust 6 conveyor belts using 12 tensioning screws, is unreasonable from an ease-of-use perspective. Also, a frame that held the six conveyor belts and allowed easy access to the tensioning mechanism was difficult to design and used a lot of extra structural material. Additionally, the conveyor belts took upwards of 40 hours each to 3D print, making them an unreasonable manufacturing burden. Additionally, the conveyor belts took upwards of 40 hours each to 3D print, making them an unreasonable manufacturing burden.

Distributor

As the main innovation of our project, we spent the most time designing and building the Distributor, corresponding storage tower, and testing code. The final Distributor can be seen in the figure below.

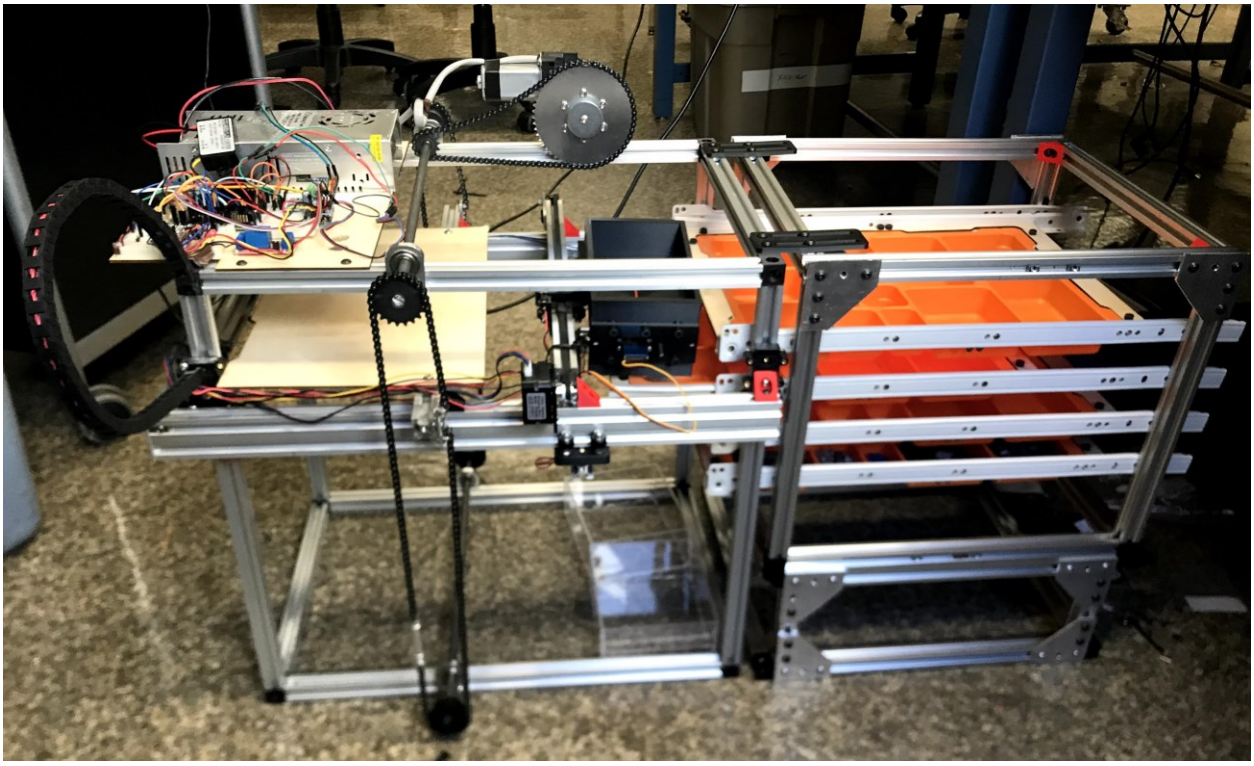


Figure 33: Distributor Final Build

Although we did not have enough of the entire system built to test the first four of our robot's six main goals, we were able to test goals number 5 and 6 on the Distributor: sort at a rate of 6 seconds per part and keep track of the number and types of parts sorted and know when a kit is complete. In our Distributor tests, the parts were deposited by a human into the end-effector when it returned to the place it would deposit the part. We found that because of our chosen position control methods, the sorting speed was much too slow. Keeping track of the parts, however, was successfully accomplished by our code.

On average, the Distributor took about 25 seconds between the entry of the color and shape into program and the system being ready to obtain another part. This timeline of events of course includes the placement of the part into the correct pocket. This is over four times slower than goal 6's baseline of placing a part every six seconds. There were two main causes of this, both of which are easily adjustable for a future project. Firstly, most of the 25 seconds were used

by the z-axis attempting to reach its goal position. The chosen position control method, an ultrasonic sensor, turned out to be much too noisy, so we had to average 50 readings per movement. This slowed the z-axis control loop down considerably. Therefore, the cost savings associated with the ultrasonic sensor over a motor encoder were not worth it in terms of controllability. Secondly, the Distributor code was not built to multi-task: that is move the z and x axes at the same time. This would also reduce distribution time per part. Unlike the z-axis, the x and y axes motors were very accurate and precise, consistently placing parts into the correct pockets.

The Distributor's software was able to determine if a pocket is full or not by comparing the maximum amount of each part that can be placed in a pocket, maxParts, with the amount that it has attempted to place in the pocket, currParts. This was done by the function, canAddPartToPocket, which is located in Pocket.py. However, this only works on a part-to-part basis, since TrayDB only needs to know the status of one part at a time. It returns true if the current amount of a part is at least one less than its maximum amount. Returning false would indicate that the pocket is full. Adding "not" before an instance of this function is used would return true if the pocket is full, accomplishing this part of the goal. However, there is no current code that can explicitly calculate if a tray is complete or not. However, this functionality could be added by iterating through the list of pockets in a tray, and then checking if each pocket's parts are full by calling canAddPartToPocket on each pocket.

In addition to the performance goals, we were also able to evaluate the Distributor on the consumer's subjective preferences: cost, size, and ease of use. The first of these was for the robot to cost between $500 and $700. The physical Distributor build and associated electronics, including the Jetson Nano, cost $831.82 total (Appendix B). However, this cost could be reduced

in future iterations through several methods. A design that moves from the prototyping to the consumer build stage could switch to a different aluminum for the robot frame. The v-slot aluminum on this robot cost $140 total. Future versions could also reduce the amount of aluminum on the robot in total, especially on the storage tower. The base underneath where the drawers start was designed so the z-axis elevator could move out of the way to allow a user to access the drawers. However, this same functionality can be achieved without the extra metal and brackets, simply by spacing the drawers to start up higher. Another option to reduce cost in a final consumer product is to manufacture custom PCBs. This would lower the cost of the electrical system by reducing it to just the necessary chips and components. Finally, design optimizations could be made to decrease the amount of material on the robot all together. For example, if the Distributor could be made physically smaller, that would reduce the size of the corresponding motor, the number of linear slides needed, and the amount of material needed. One option to do this would be to have the y and z axis on the other side of the drawers, pushing them out instead of pulling them. This would also allow the drawers to be more easily accessible to the teacher.

Another consumer goal was for the robot to fit easily in the classroom. This goal was successful since the robot was constructed in such a way that it takes up as little floorspace as possible. This is accomplished by having the trays stacked on top of each other, and putting the Identifier, Serializer, and various electronics on top of the Distributor. The distribution and storage tower can be lifted and moved individually by one person. When the storage tower is decoupled from the Distributor the drawers are easily accessible. A more consumer ready product, however, would improve the ease at which these two components can be coupled and decoupled.

Future Work & Conclusion

Although this MQP did not accomplish its goal of building a fully functional sorting system, we were able to focus on the most innovate aspects of our LEGO Sorting Robot for the Classroom, thereby introducing a new part distribution and storage system for any sorting machine, especially ones that separate multiple identical sets. We think this innovation marks progress in the field of sorting robots, especially LEGO sorting robots, because it demonstrates how a LEGO sorting system can be practically applied to a very real-world need. The concept of vertical storage and distribution also has the potential to be used in a wide array of sorting applications where floor space is at a premium and distribution efficiency is key.

A wide variety of work can still be performed to improve our overall system's functionality. Although working examples are found in past projects, an Identifier and Serializer that meet the specific consumer-oriented goals of this project still need to be built and tested. Our team did find that the multiple conveyor belt Serializer, while effective, requires too much maintenance to be practical for this purpose. Future work should look back at past Serializer designs and modify one to fit this system. However, the conveyor belt design used for our Serializer, is still a useful model as the base of the Identifier portion of the robot. As previously discussed, improvements could also be made to the Distributor to make it cheaper, more efficient, and more modular for the consumer. Finally, a user-facing GUI would be a useful addition to this project, allowing users to customize their robot by changing data such as the parts which go in each pocket, the layout of the tray, and the parts the neural network is trained for. Settings for a tray could be saved, and shared with other users, who could then import the data into the software and have the robot be configured to support that set.

Overall, the team believes we have laid out useful design criteria, designed a reasonable sorting system, and tested the most innovative components of a LEGO Sorting Robot for the Classroom, providing a solid framework for future work on this problem.

References

[1] D. Newton, "LEGO Is Probably The Biggest Education Company On Earth," Forbes, 8 Feburary 2020. [Online]. Available: https://www.forbes.com/sites/dereknewton/2020/02/08/lego-is-probably-the-biggest-education-company-on-earth/#53324e9a478c. [Accessed 12 October 2020].

[2] LEGO Education, "In Philadelphia, LEGO® Education Lets Every Type of Learner Shine," Lego Education, [Online]. Available: https://education.lego.com/en-us/customer-stories/paula-don. [Accessed 12 October 2020].

[3] A. Sucheler, D. Jin and P. Donaldson, "A. I. Lego Sorter," Worcester Polytechnic Institute, Worcester, 2019.

[4] D. Zhidro and J. Gallagher, "Scalable Sort Automation," Worcester Polytechnic Institute, Worcester, 2020.

[5] D. West, "The WORLD'S FIRST Universal LEGO Sorting Machine," 3 December 2019. [Online]. Available: https://www.youtube.com/watch?v=04JkdHEX3Yk. [Accessed 31 August 2020].

[6] D. West, "LEGO Sorter AI: How Does It Work?," 3 December 2019. [Online]. Available: https://www.youtube.com/watch?v=-UGl0ZOCgwQ. [Accessed 31 August 2020].

[7] D. West, ""Here are some examples of the 3D rendered images used to train the LEGO sorting AI..."," 9 December 2019. [Online]. Available: https://twitter.com/JustASquid/status/1204154896313798657. [Accessed 31 August 2020].

[8] WorthPoint, "LEGO MINDSTORMS NXT EDUCATION BASE SET (9797)," WorthPoint Corperation, [Online]. Available: https://www.worthpoint.com/worthopedia/lego-mindstorms-nxt-education-base-885456360. [Accessed 14 October 2020].

[9] Habasit, "Fabric Conveyor Belts Engineering Guide," July 2018. [Online]. Available: http://blog.habasit.com/wp-content/uploads/2018/07/Engineering_Guide_Fabric_Conveyor_Belts_Habasit_6039BRO.CVB-en01212HQR.pdf.

[10] C. Heindl, B. L., S. Zambal and J. Scharinger, "BlendTorch: A Real-Time, Adaptive Domain Randomization Library," in *1st Workshop on Industrial Machine Learning at International Conference on Pattern Recognition (ICPR2020)*, 2020.

Appendix A: Mechanical Calculations

Serializer Motor

last conveyor belt speed = $V_6 = \frac{76\ mm}{1\ piece}\left(\frac{1\ m}{10^3\ mm}\right)\left(\frac{1\ piece}{4.5\ s}\right) = .0169\ m/s = 1.69\ cm/s$

$V_{n-1} = \frac{V_n}{2}$ (each belt goes twice the speed of the previous one)

$r = \frac{1\ in}{2}\left(\frac{.0254\ m}{1\ in}\right) = .0127\ m$

$\omega_6 = \frac{V_6}{r} = \frac{.0169\ m}{s}\left(\frac{1}{.0127\ m}\right) = 1.33\ rad/s = 12.7\ rpm$

$\tau = F_{app} * r * \mu_{pulley}$

$\mu_{pulley} \approx .6$ (rubber on steel = .8, rubber on uhmw plastic = .6, rubber on urethane = .6)

$\tau_6 = \left(.2\ kg * \frac{9.8\ m}{s^2}\right) * (.0127\ m) * .6 = .015\ N \cdot m$

$P_6 = \tau_{motor} * \omega_6 = .015\ N \cdot m * 1.33\ rad/s = .020\ W = 20\ mW$

For one motor to power all 6 belts: $P \approx 6 * P_6 = .12\ W$

There is lots of uncertainty b/c of conveyor tension, exact design, etc.

So, using a safety factor of 5

$P = .6\ W$

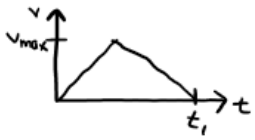
Distributor Y-Axis Motor

Y-axis stepper:

Mass of y-bridge + drawer = 1.5 kg

Max travel distance = 400 mm = .40 m = d

Time to travel = .65 s = t_1



$\Delta x = \frac{1}{2}at^2 = \frac{1}{2}d \Rightarrow a = \frac{d}{t^2} = \frac{.4m}{(.65s)^2} = .95\ m/s^2$

$v_{max}^2 = 2a\Delta x = a \cdot d \Rightarrow v_{max} = \sqrt{ad} = \sqrt{(.95\ m/s^2)(.4m)} = .62\ m/s$

For a belt driven system:

Torque for constant velocity = $T_c = \frac{F_{axial} \cdot r_{pulley}}{\eta_{belt\_sys}}$

$\eta \leq 98\%$ for timing belts

$F_{axial} = m \cdot g \cdot \mu_{guide}$

μ for rolling bearings is very small. We'll say .1

$F_{axial} = 1.5\ kg \times 9.8\ m/s^2 \times .1 = 1.5\ N$

For 22 mm OD pulley ($r = .011\ m$) & assumming $\eta = 95\%$:

$T_c = 1.5\ N \times \frac{22\ mm}{2}\left(\frac{1\ m}{10^3\ mm}\right)\left(\frac{1}{.95}\right) = .017\ N \cdot m$

Torque for acceleration = $T_a = T_c + T_{acc}$

$T_{acc} = J_{sys} \cdot \alpha$

$J_{sys} = J_{motor} + J_{drive\ pulley} + J_{idolar\ pulley} + J_{belt} + J_{load}$

$J_{load} = I_{xz} = 8.1 \times 10^{4}\ g \cdot mm^2 = 8.1 \times 10^{-5}\ kg \cdot m^2$ (From solidworks)

2 mm timing belt $m = .095\ kg/m$

$J_{belt} = m_{belt} \cdot r_{pulley}^2 = (.095\ kg/m)(.8\ m)(.011\ m)^2 = 9.2 \times 10^{-6}\ kg/m^2$

$J_{drive\_pulley} = I_{xx} = 4.9 \times 10^{-7}\ kg \cdot m^2$

$J_{ideler\_pulley} = I_{xx} = 5.9 \times 10^{-7}\ kg \cdot m^2$

$\Rightarrow J_{sys} = 9.1 \times 10^{-5}\ kg \cdot m^2 + J_{motor}$

$\alpha = \frac{a}{r} = \frac{.95\ m/s^2}{.011\ m} = 86\ rad/s^2$

$T_a = (.017\ N \cdot m) + (9.1 \times 10^{-5}\ kg \cdot m^2 + J_{motor})(86\ rad/s^2)$

$T_a = .025\ N \cdot m + (J_{motor})(86\ rad/s^2)$

safety factor = 2

$T_{a\_safe} = .050\ N \cdot m + (J_{motor})(172\ rad/s^2)$

17HS4023

$J_{motor} = 38\ g\cdot cm^2 = 3.8\times10^{-6}\ kg\cdot m^2$

$T_{det} = 12\ mN\cdot m = .012\ N\cdot m$
$T_{rot} = 13\ N\cdot cm = .13\ N\cdot m$

$T_{running} \approx T_{rot} - 2T_{det} = .1\ N\cdot m$

$T_{a\_safe} = .051\ N\cdot m < T_{running}$

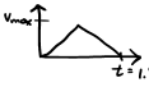
1.8 deg step = .032 rad = .35 mm

Distributor X-Axis Motor

x-axis stepper

time for part to fall: $\Delta x = \frac{1}{2}at^2 = (50\ mm)(N) \Rightarrow t = \sqrt{\frac{2(50\ mm)N}{a}} = \sqrt{\frac{2(.05\ m)(10)}{9.8\ m/s^2}} = .3\ s$ (N=10)

So, x-axis time is 1.9 s go to part pickup (d_{max} = 170 mm)
.3 s wait
1.9 s go to dropoff (same d_{max})
.4 s release part (.36 s to rotate servo 180°)



$a = \frac{d_{max}}{t^2} = \frac{170\ mm}{(1.9\ s)^2} = .047\ m/s^2$

$V_{max} = \sqrt{a\cdot d_{max}} = .089\ m/s$

$m = .28\ kg$

$I_{xy} = 3.8\times10^{4}\ g\cdot mm^2 = 3.8\times10^{-5}\ kg\cdot m^2$

So, similarly to the y-axis calcs:

$F_{axial} = .27\ N$

$T_C = .0032\ N\cdot m$

$J_{sys} = 4.8\times10^{-5}\ kg\cdot m^2$

$\alpha = 4.3\ rad/s^2$

$T_{acc} = (4.8\times10^{-5}\ kg\cdot m^2)(4.3\ rad/s^2)$

$T_a = .0032\ N\cdot m + (4.8\times10^{-5}\ kg\cdot m^2 + J_{motor})(4.3\ rad/s^2)$

$T_a = .0034\ N\cdot m + (J_{motor})(4.3\ rad/s^2)$

$T_{a\_safe} = .0034\ N\cdot m + J_{motor}(4.3\ rad/s^2)$

3.4 N·mm
.34 N·cm

⇒ same motor as y-axis will be way more than enough

Distributor Z-Axis Motor

Z-motor:

$m = 3.2$ kg

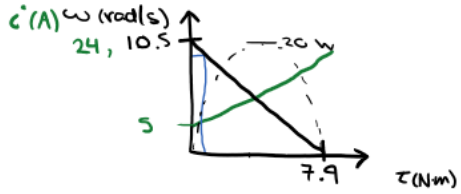
$d_{max} = (.05\ m)N \Rightarrow$ for $N=10$ $d_{max} = .5$ m

$t = 4.5\ s - \underbrace{2(.65\ s)}_{\text{y-movement}} - \underbrace{.4\ s}_{\text{part dropoff}} = 2.8\ s$

$$\text{Power} = \frac{F \cdot d}{t} = \frac{(3.2\ kg)(9.8\ m/s^2)(.5\ m)}{s} = 5.6\ W$$

$$\text{speed} = \frac{.5\ m}{2.8\ s} = .18\ m/s$$

Snow-blower motor



No transmission, $r_{pulley} = .019$ m (20T sprocket)

$\tau = F_g \times r_{pulley} = 3.2\ kg \times 9.81\ N \cdot m \times .019\ m = .60\ N \cdot m$

$\omega = 10.5 - \frac{10.5}{7.9}\tau \Rightarrow \omega = 10.5 - \frac{10.5}{7.9}(.75\ N \cdot m) = 9.7\ rad/s$

$v = \omega r = (9.5\ rad/s)(.019\ m) = .18\ m/s \Rightarrow$ technically ok, but no safety factor

$i = \frac{24-5}{7.9}\tau + 5 = 6.8\ A$

Max efficiency, $r_{pulley} = .019$ m:

$\tau_{in} = 2.5\ N \cdot m$

$\tau_{out} = .60\ N \cdot m$

$e = \frac{\tau_{in}}{\tau_{out}} \times \eta = \frac{2.5\ N \cdot m}{.60\ N \cdot m} \times .95 = 4.0$

$\omega_{in} = 10.5 - \frac{10.5}{7.9}(2.5\ N \cdot m) = 7.2\ rad/s$

$\frac{\omega_{out}}{\omega_{in}} = e \Rightarrow \omega_{out} = e \cdot \omega_{in} = 28.5\ rad/s$

$v = (23\ rad/s)(.019\ m) = .54\ m/s \Rightarrow$ definitely fast enough

$i = \frac{24-5}{7.9}(2.5) + 5 = 11\ A$

Max power, $r_{pulley} = .024\ m$.

$\tau_{in} = \frac{7.9}{2} = 3.95\ N \cdot m$

$\tau_{out} = .6\ N \cdot m$

$e = 6.6$

$\omega_{in} = \frac{10.5}{2} = 5.25$

$\omega_{out} = 34.6\ rad/s$

$v = .66\ m/s$

$i = \frac{24-5}{2} + 5 = 14.5\ A$

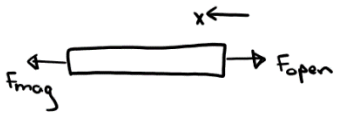
Distributor Electromagnet

Electromagnet:

mass of drawer = 1 kg

desired acceleration = 1 m/s^2

Force to open drawer = 3-7 N (from paper)



$\sum \vec{F} = m\vec{a} \Rightarrow F_{mag} - F_{open} = ma \Rightarrow F_{mag} = (1\ kg)(1\ m/s^2) + 7N = 8\ N$

Electromagnet rated force is about 5-10x what it can actually hold, so $F_{mag\_rated} \approx 80\ N$

Appendix B: Cost of Built Prototype

Item	Cost for Each	Number	Total
Drawer Slides (set of 2)	$4.18	5	$20.90
Wood for laser cutting	$28.68	1	$28.68
16/20 snowblower motor	$36.00	1	$36.00
20mm x 20mm exoslide	$6.99	8	$55.92
12v to 5 v controller k240505 cocarieyklin	$11.89	1	$11.89
12v power supply	$20.00	1	$20.00
Cytron MD20A Motor Controller	$19.80	1	$19.80
L298N	$6.89	1	$6.89
Relay	$2.00	1	$2.00
Electromagnet	$10.99	1	$10.99
ADS1115	$14.95	1	$14.95
"Gravity" Ultrasonic sensor	$10.00	1	$10.00
PCA9685	$14.95	1	$14.95
pololu purple stepper controller	$8.95	2	$17.90
two trees 17hsa203	$9.89	1	$9.89
jetson nano 2gb	$59.00	1	$59.00
17hs4401 Stepper Motor	$9.75	1	$9.75
Chain	$12.00	1	$12.00
Timing Belt	$11.85	3	$35.55
Snowblower motor Accessories:	$-	0	$-
Big Sprocket	$14.00	1	$14.00
hex hub converter	$8.00	1	$8.00
8cm washer	$0.66	1	$0.66
Motor bracket	$8.00	1	$8.00
4 medium sprockets	$13.06	4	$52.24
1 small sprocket	$11.29	1	$11.29
Mini v wheel kit	$3.20	12	$38.40
Bearings: set of 4	$9.49	1	$9.49
Bearing blocks: material for 4	$4.03	1	$4.03
Material to make all Brackets	$4.04	6	$24.24
Bumper switch kits	$3.78	3	$11.34
Bracket for electromagnet	$2.79	1	$2.79
Jx pdi-6221mg 20 Kg Servo (servo for claw)	$15.00	1	$15.00
stepper motor brackets	$4.95	2	$9.90
timing belt pulleys (on motor)	$10.90	2	$21.80
timing belt pulleys (off motor)	$4.80	2	$9.60

Misc hardware	$55.00	1	$55.00
chain - metal connector (retail)	$2.50	4	$10.00
chain master link	$6.00	4	$24.00
chain tensioner	$15.00	2	$30.00
3d printed parts (3d printed, negligeable)	$20.00	1	$20.00
drawer slides, set of two	$4.18	5	$20.90
wires	$10.00	1	$10.00
3ft x 3/8” aluminum rod	$5.18	2	$10.36
Wire Loom	$9.99	1	$9.99
Steel angle (enough for two)	$7.46	0.50	$3.73
2020 and 2040 aluminum	$140.42	1	$140.42
		Total:	$831.82

Appendix C: Code

If you are interested in viewing the code for this project, it can be found on GitHub at the following link:

https://github.com/mariasharman137/LEGOSorterMQP