COLLEGE OF ELECTRICAL ENGINEERING

AND COMPUTER SCIENCE

ELECTRONIC LEGO SORTER

Group 33

Nike Adeyemi

David Carey

Katrina Little

Nickolas Steinman

Senior Design 1 - Fall 2014

TABLE OF CONTENTS

1.0 Executive Summary

LEGO has become a household name – you can ask just about anyone if they have ever played with LEGO's as a child and the answer will most likely be yes. Not only are LEGO's a toy, they also help children develop important skills such as problem solving, organization, and planning through construction (among many other critical development skills). LEGO has evolved tremendously through the years. They have even developed LEGO kits for adults such as the "Mindstorms NXT" which includes a programmable robotics kit. There are really only two drawbacks when it comes to LEGO's. The first drawback is the price. The second drawback is for the true LEGO collectors. If you collect LEGO's for many years, you wind up with many pieces and it is very difficult to sort through 100 pounds of bricks to replicate one of the original LEGO "sets" that you purchased a long time ago. This project is being designed to tackle both of these problems.

2.0 Project Description

This section outlines the motivation, goals & objectives, and team member responsibilities of the project.

2.1 Project Motivation

More than 400 billion LEGO bricks have been produced since 1949. LEGO bricks are available in 53 different colors and 19 million LEGO elements are produced every year. There are more than 8 quadrillion (8,181,068,395,500,000) possible combinations of minifigures that can be made using all of the unique minifigure parts over the last 30 years. The motivation for an electronic LEGO sorter is extremely significant. There are many blogs online of LEGO fanatics with approaches to sort LEGO's using home storage solutions to tackle the organization problems associated with LEGO's. Unfortunately, most LEGO collectors do not have the technical knowledge to build a sorter to solve this problem. The motivation behind this project is to tackle the two main difficulties with LEGO's. The first shortcoming being the price and the second being the headache of sorting LEGO parts by hand.

Figures 2.1-A and 2.1-B illustrate a before and after depiction to the LEGO sorting process. Even after spending hours hand sorting an entire LEGO collection you can easily see there is still a lot of room for improvement. Hand sorting 20 pounds of LEGO's takes about 8 hours and by the end of it all the small parts get thrown into a bin unsorted due to exhaustion.

Figure 2.1-A : Huge Collection of Unsorted LEGO (permission pending)

Figure 2.1-B: Huge collection of LEGO’s that have been hand sorted

Currently, there is only 1 type of LEGO sorting element that you can purchase. These plastic sorters consist of four tiers, each tier having different sized slots. The figure below easily sums up the problem with this method to LEGO sorting.

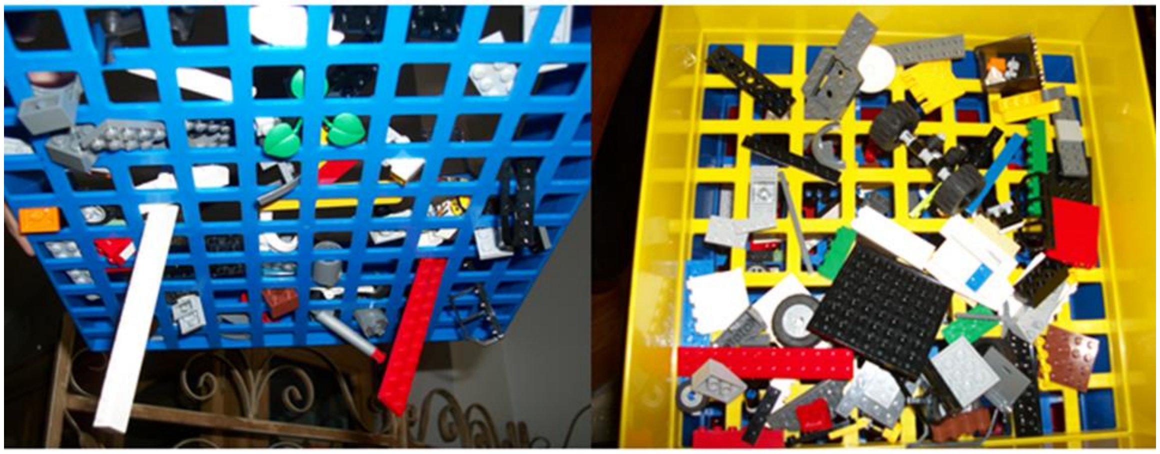


Figure 2.1-C: Multi-Tiered LEGO sorting element

The goal of this project is to sort through large quantities of LEGO’s autonomously (meaning you turn on the sorter and leave it alone). You may wonder how this tackles the first drawback to LEGO’s which is the price. Some LEGO “Star Wars” sets sell for thousands of dollars. You can buy a 20 pound lot of LEGO’s for around $200 (give or take). So, the idea is that if you buy large bulk lots of LEGO’s you can use this “Electronic LEGO Sorter” to slowly piece together these more expensive sets slowly over time as a more affordable option to buying a new set in stores. This project aims to tackle both problems of price and inconvenience.

2.2 Goals and Objectives

The goal of this project is to sort a set library of LEGO bricks as accurately as possible by color or by shape, which is decided by the user using a touch screen. This way if the user wants to further sort the bricks such as by color and shape they would just remove the bin they wish to sort and put it back in for further sorting. Because of the variety of LEGOs that are out in the market sorting through them to get the exact part needed becomes very time consuming. So for hobbyist who cannot find a certain part or sellers who need an efficient way to find and catalogue their LEGO parts this project could be the solution to relieve this problem. All the user needs is a Library of the different LEGO parts and colors.

The main objective of this project is accuracy therefore speed is not as important of a factor for this LEGO sorter. With the library loaded, the user will use the touch screen to specify how to sort the bricks whether it is by color or by shape. Then the user decide which of the nine containers the blocks will go in with one of the containers being used to collect any errors or bricks the image processor could not determine. The user can assign multiple buckets to hold the same types and there exists common colors like yellows, whites, blues, etc. The bricks to be sorted will be loaded into a container with a lift system built in. The lift will carry a hand full of LEGO parts up and dump them on a two-stage conveyor belt. The conveyor belts are used to space out the parts before they enter the image processor. Setting the first stage belt to a slow speed and the second stage belt to a higher speed does this. The bricks will slowly drop to the faster belt spacing the bricks apart. Each brick will enter the image processor one by one. Once a LEGO enters the processor the conveyor belt stops and the image processor system becomes active. Depending on what the user decided how to sort, the image processor will either check color and compare it to some color constants or using a mirror, edge detection algorithm, and other algorithms to compare the given part to the parts in the library in memory to decide what kind of part it is. Once the processor is done and decides which bucket to put it and rotates the arm to the correct position. The sorter will then push the part into a chute, which will guide it to the appropriate container. The process will continue until all parts have been dealt with. All these processes will be done using MCUs for motor and lift controls and a Beagle Bone Black to act as an image processor.

2.3 Team Member Responsibilities

The group involved in the project consists of two Electrical Engineers and two Computer Engineers. This helps to determine a natural way to divide the work within the project. The Electrical Engineers will be able to focus more on the hardware side of the project, dealing with the circuit boards and power, while the Computer Engineers will be devoting the majority of their time to the software of the project, coding everything up and making sure all of the parts are communicating properly. Beyond that, each person has an even more specific role that they play within the project.

Nike Adeyemi (Computer Engineering)
Nike’s focus in the project will be around the User Interface. Her role is to build a user interface that will be quick to learn, easy to understand, and very effective in aiding the user in setting up the organization of the project in exactly the way that the user desires.

David Carey (Computer Engineering)
David’s primary area of focus for the project will be on the image processing portion of the sorter. His role is to make sure that the Legos are properly analyzed and then sorted correctly based upon their color shape and size.

Katrina Little (Electrical Engineering)
Katrina's primary area will be that of the power supply, lift arm subsystem, and interfacing with the Tiva-C and BeagleBone. She will be designing the source of power for the entire project to ensure that the sorter receives enough power to be able to function properly, without receiving so much power that it becomes more harm to the system than good. Also she will be the one in charge of making sure that communications are working properly between the brains of the project and the mechanical parts.

Nick Steinmann (Electrical Engineering)
Nick's primary roles include setting up the Conveyor Belt and Rotating Arm systems as well as setting up the PCB for the Tiva-C. The Tiva-C chip will be utilized on a custom PCB board built by Nick for this project.

3.0 Related Research

This next section will be devoted to showing the research done in different areas that ultimately led to the decisions that were made regarding the Lego sorter.

3.1 Existing Projects and Products

The first place that research for the project was started was with similar projects and products. This includes other Lego sorters as well as other projects that used components similar to those being used in the Lego sorter.

3.1.1 Existing (Electronic) LEGO Sorters

When choosing this project it was already known that there were already a handful of sorters that organize blocks and other various LEGO parts. Many of those were created using the Mindstorm NXT LEGO smart brick as the main controller. The sorters that we focused on were those that could sort LEGO bricks and other parts and of those sorts they were organized into three categories: Sorters that sort only by Size and shape, Sorters that organize by color, and sorters that are organized by some combination of the previously stated types.

There were sorters we saw organized only specific parts by size or length for example is a sorter that only sorts lift arms, created by a user on YouTube known as Akiyuky, and uses a simple system to process them. First the sorter will orient the parts a certain way to enter the system for example the sorter in figure 3.1.1 – a used two lifts that alternate going up a down to change the lift arm orientation and to load the lift arm onto the conveyor belt. The sort then makes sure the that the parts only enter one at a time and back to back and any part that ends up onto of the other or does not fit in the entryway is pushed into a chute and back into the loading bucket. The lift arms that made it into the sorter then travel across the conveyor belt until it slides into the first hole that is big enough to fall in the hole getting bigger the longer it travels and into the correct containers.

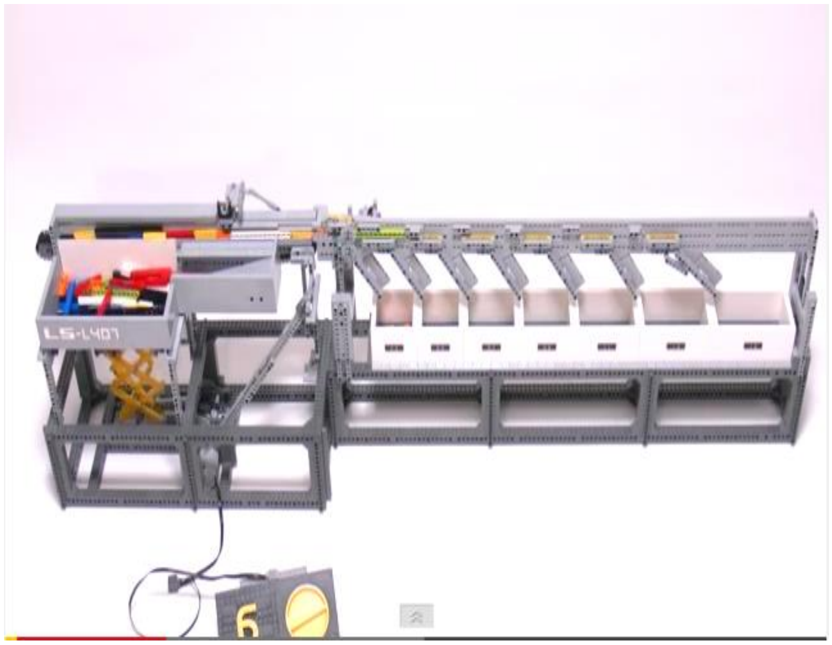


Figure 3.1.1-A LEGO Lift Arm Sorter

There is also another sorter that is able to separates a wider variety of bricks both using a similar set up. The same YouTube user Akiyuky created this sorter as well using image processing as a means to sort the parts. In his sorter, the parts entered the sorter with a lift arm by the hand full and carried though the two-stage conveyor belt. The two conveyor belts are used for spacing out the parts so they can enter the image processor one by one and sorted into the appropriate bucket. This being a good start but they only sort by shape and the sorter that we build must also sort by color.

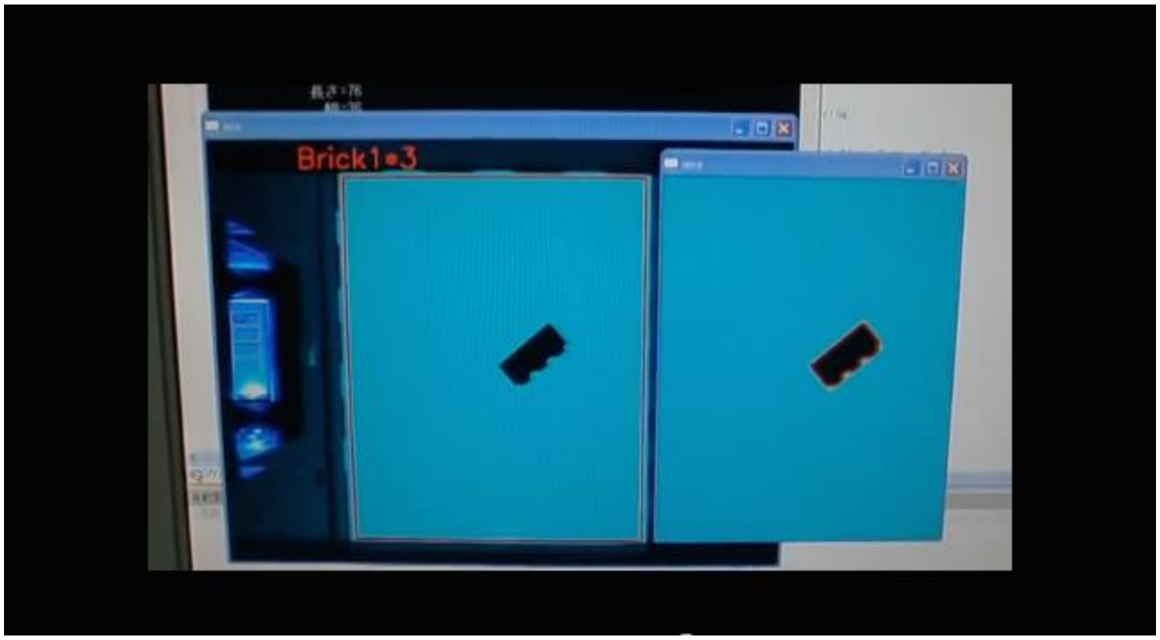


Figure 3.1.1-B Image recognition screen

There also LEGO sorters that organized by color. It goes through a similar process, as the part sorter only that a color sensor is needed to detect the brick color. However many of the sorts seen using this method was only using the same brick type, as color was the main focus of their machine.

Finally there were sorters that are able to sort both by shape and color and are what we want for our own sorter. Of the ones that were seen they were simple as that only used simple bricks like the 2 x 2 and 2 x 4 bricks and sort them. It makes sense though as sorting a variety of parts with a variety of colors will need a large amount of containers to use. While our sorter will sort through many parts there is only a limited amount of container to use.

So with everything considered Akiyuky's model is the closest to the model we had in mind. It was mainly because of its use of image processing. Because of this it can easily be adjusted in order to sort by shapes in one setting and sort by color in a different setting. This essentially turns our sorter into a hybrid sorter as in order to sort by color and shape the user will just have to go one pass to sort by color or shape and a second pass with the opposite settings.

3.1.2 Projects with Relevant Components

For our sorter the design will require use of image processing, a conveyor system, and a loading system. There are a wide amount of parts to choose from so to narrow down our choices we looked at other senior design project and what parts they used for their project that relates to our project by providing similar need to what we need.

T-100 Watchdog - The T-100 Watchdog was an earlier senior design project in spring 2014 in UCF. It was an all-terrain vehicle that uses computer vision and autonomy to track and pursue a target based on the heat signature. Using the web camera and thermal camera, the image processor it is able to track any movement that has occurred, acquire the target that was the source of that movement, and track that target all in real time.

The team was able to program these functionalities using OpenCV an open source computer vision library running in a Linux based operating system as OpenCV has the majority of the algorithms needed for their task. All the image processing was done through the BeagleBone Black as it acted as the main controller for all the subsystems that were running in this vehicle.

For the motors each motor that controlled each wheel individually were controlled by separate microcontrollers each of them connected to the main controller the BeagleBone black.

Solar Tracker – The solar tracker is another UCF Senior design project. The purpose of this project is to use a mobile application to control the solar tracker to convert the solar energy in order to provide power to other devices. Since they are dealing with solar tracking it is important that the solar panels are facing the sun to get maximum energy input. Therefore motors are needed to rotate the panels and are program to rotate to the right orientation depending on the suns location. Here they considered using motor controllers as opposed to microcontrollers believing that just only using microcontrollers might not have enough power to drive the motors. When used in conjunction with the microcontroller the motor controllers just receive information from the microcontroller on how to run with just a few inputs. They use DC motors to rotate the panels and the servomotor to determine how much the motor should rotate. The LCD screen they had was just a simple 2 x 16 screen that is linked the microcontroller used to monitor the status of the device.

3.2 Initial Project Considerations

This section will cover existing projects with similar components to be used in this project, as well as similar LEGO sorters, subsystem designs that were rejected, and finally subsystem specific research.

There is no “one size fits all” on how one sorts their LEGO collection. Some people sort by color, others by shape and size, and others try to sort their collection by sets. It all depends on what you plan TO DO with your LEGO’s.
Some parents buy new sets for their children because they want their children to develop engineering skills by following a specific detailed instruction manual to build a project. Also, many adults stick solely to “Star Wars” collections (among others). These groups of people would require a sorter to sort their pieces specifically by sets.
“Lego artists” are more inclined to sort their bricks by color and shape. Another thing to consider is that many people have such large collections of LEGO’s that sorting just the “bricks” and “plates” by color separates the collection enough to find the other smaller parts that are needed. This is because the “bricks” and “plates” make up the majority of most sets.

3.2.1 Lego Separation, Transportation, & Identification

There are countless ways to tackle the sorting issue. This section describes initial considerations for the transportation, separation, and identification components of the project design.

One of the ideas considered was dumping the parts into a large bucket that had a camera inside. The camera would photo the parts as they fell through the bucket. As the parts fell there would be fans to blow the parts into a specific section for sorting. This idea was too complicated mechanically and was therefore ruled out.

Another consideration was to pre-sort the parts using a sifting device such as the one shown in figure 2.1- c (above) before reaching a camera for image recognition. This was not ideal due to the huge variety of LEGO parts.

The use of a conveyor belt system using sensors and fans was also considered. The parts would pass a series of sensors and fans situated on the sides of the conveyor belt. If the sensor was tripped the data would be sent to the microcontroller. The microcontroller would turn on the fan and blow the part off of the belt into the appropriate sorted bin of similar parts. This would be done using color sensors. This would sort parts by color but not by size or shape. This idea was ultimately ruled out because LEGO's are made from plastic which limits the type of sensor that could be used to detect the size of parts that pass by. The only sensors that can detect plastic are capacitive sensors. Capacitive sensors are not as readily available as Inductive sensors (which are used to detect metal). Capacitive sensors are more so used to detect different types of plastic materials (not shape or size) which is needed for the application of this project. The other issue with this method is that if many parts pass by the sensors (as well as fans) at the same time they will all be blown into the bins and ultimately the sorting process is destroyed.

3.2.2 Lego Sorting

After deciding specifically which Legos will be sorted, the next issue to arise is exactly how to separate them into different receptacles.

One of the first ideas that was discovered in research was a sorter that was able to separate parts using different sized gates to catch the various Legos as they were transported down the conveyor belt. While it would allow for a bit more speed in the separation, it only accounts for a specific Lego type. With multiple different types of Legos being introduced to the sorter all at the same time, simple gates would end up putting Legos of multiple different types into the same receptacles.

The next idea was that of dropping the Legos down into waiting receptacles. So essentially we would have a set of buckets waiting below the end of the conveyor belt which would dispense the most recently identified Lego into the proper bucket. This became the idea that was ultimately settled upon. With the method of using image processing to determine the type of Lego, the difficult part was trying to find a way to cycle through the different available buckets in order to separate the Legos.

When it came to the orientation of the buckets, two different options were presented: a circular arrangement, and a linear arrangement. The linear arrangement would essentially line up the buckets underneath the conveyor belts. It would then extend or retract the set of buckets so that the bucket used for the current Lego would sit underneath the conveyor. This option has the advantage of being an option that the space the sorter would occupy would be minimalized a bit……. Or at least that would be the case when the sorter is not active. That’s where one of the major downsides occurs. The entire idea for the sorter is that the user does not have to be actively involved after initially starting it, therefore the user would not be totally aware of when the buckets would extend from underneath the conveyors. This could cause a potential safety hazard for the user. It also creates a requirement that extra space needs to be accounted for when setting up the sorter, otherwise the bucket system could end up running into a wall. Another issue to consider is the weight of the Legos. After a rather long session of sorting, the buckets may begin to contain rather large amounts of Legos that could cause more stress on the motors involved in the system. While this wouldn’t be an immediate issue, it could cause fairly extensive wear and tear to the system.

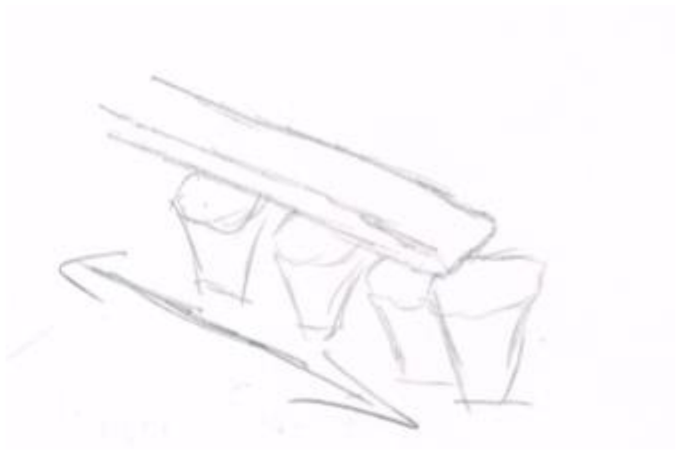


Figure 3.2.2-A Initial drawing of a linear bucket system

In the case of the circular configuration, the more major issue of the linear system is accounted for. In a circular configuration, the bucket setup, while still able to move, would never take up unexpected space. This eliminates the safety hazard of the unexpected extension of the linear system. However, the issue of extensive wear and tear of the motor would still occur if the buckets were to rotate. So the ideal situation for the sorter would be a circular configuration for the buckets in which the buckets would not have to be the subject of the rotation. That is when the idea for a rotating arm was proposed.

In the case of the rotating arm, the buckets would be set up in a static circular configuration still, but rather than the conveyor belt dropping the Legos off into the bucket below, it would drop them into a very simple slide that would rotate between the buckets as necessary. The arm would first rotate to the desired receptacle, and then the conveyor would drop off the next available Lego onto the slide. This would ensure that the amount of weight handled by the motors would only ever handle the weight of the arm itself and nothing more. This way the motors would be able to last for much longer.

Ultimately, this rotational arm system is what was settled upon. This will be the safest, most efficient way to sort out the Legos while causing as little wear and tear to the motors as possible.

3.2.3 User Input

One of the most important considerations that needed to be made concerning the Lego sorter was that of the user input. Without a proper system for user input there is no way to manage the Legos in a way that's truly useful. The Lego sorter needs to be able to accommodate specifications specific to what the user has in mind, and, more importantly, it needs to be simple to use and understand. The difficult part is deciding what the best way to implement a user input system would be.

One of the first ideas that was brought up was the use of a mobile app. Smartphones are everywhere and it would make a good deal of sense to simply make the user interface into a mobile app that the user could carry with them everywhere, allowing them to simply connect wirelessly to their sorter. While at first glance it seems to be a good idea, there are a couple of factors that set this idea back a bit. The first immediately recognizable downside to using a mobile app is the screen size for the interface. The goal of the user interface is to create something that a first time user would be able to approach for the first time and be able to use with relative ease. Restricting the screen size to that of mobile devices would be fairly restricting to the abilities of the user interface in that cramming an efficient and easy-to-use interface onto a small screen would be fairly difficult. Another downside would be that using a mobile app would require multiple versions. With both iOS and Android being the primary competitors in the mobile market, it would require at least two versions of the app. And even after those two have been covered, it still limits our user audience. The thought behind the Lego sorter is that ANYONE would be able to approach it and use it, so creating the user interface exclusively for mobile devices would NOT be ideal.

So if a mobile application isn't the answer, then the next logical step is a simple built-in interface. But now this raises the question as how to narrow this down even further. Would a keyboard and a screen be suitable, or would a touch of some sort make more sense? In the case of a buttons and a screen, there are so many factors that would have to be accounted for. It could be trickier to navigate from one screen to another or to assign different parts to different buckets. Overall it just doesn't seem to be practical. It would essentially have to be a full computer setup complete with a mouse to be a truly easy-to-use system. However, a touch screen is entirely reasonable. The Legos that are being sorted will already be programmed into the system, so little typing will be needed, and touch screens have become such a basic part of everyday life that it would be easily recognizable by virtually everyone, including children. This settles the goal of securing a system that would be usable by users of virtually every age. It creates a simple environment with very little learning curve for the user which makes the system very approachable.

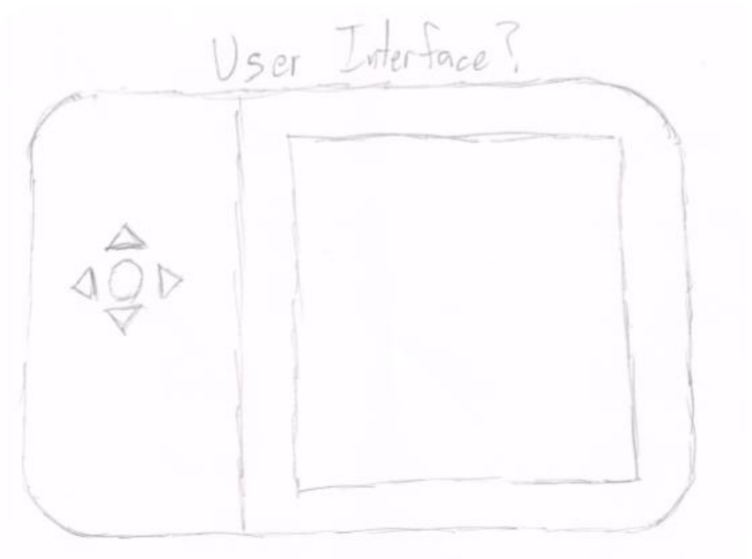


Figure 3.2.3-A Initial user interface considerations with buttons

While there are a number of different ways to approach the user interface, the initial goals of the project create limitations that ultimately lead to a built-in touch screen interface being the most ideal selection for the sorter. Users from ages 5 to 85 will be able to simply approach the system and learn how to use it with incredibly little difficulty.

3.2.4 Control

This project requires a central control unit to respond to and control each subsystem. Figure 3.2.4-A shows a rough data-flow diagram of the subsystems that will need to be connected to the main controller. The arrows indicate the direction of the data flow.

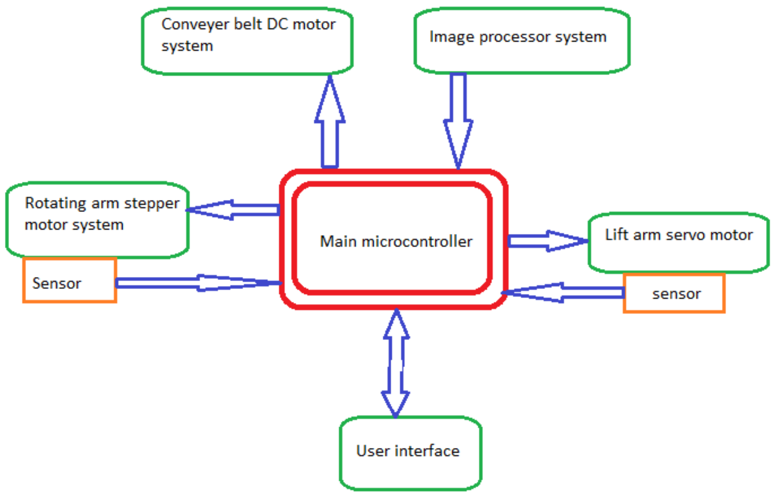


Figure 3.2.4-A: Block diagram of control

For the conveyer belt subsystem, the central control unit will need to be capable of turning off and on the DC conveyer belt motors as well as vary the DC motor speeds. The DC motor speeds do not need to be very fast as the belts will be turning slowly. Pulse width modulation (PWM) from the main controller can be used to vary the speeds. Each of the two conveyer belt DC motors will likely take one I/O port from the main controller.

The rotating arm subsystem will utilize a stepper motor and a feedback sensor to rotate towards the correct bucket for sorting. The main controller will need to control the rotation of the motor through a stepper motor driver circuit and receive feedback from a sensor to make sure the arm is in the correct position. Depending on the specific driver circuit and chips used, the stepper motor driver may take around 4 control lines from the main microcontroller.

The lift arm subsystem will utilize a servo motor to lift the LEGOs onto the belt and possibly a feedback sensor. The main controller will also need to control this motor and receive the sensor input. The sensor will tell the main controller to stop moving the lift arm at the maximum range of motion needed and to reverse the servo back to its start position. The servo motor and sensor should each only use one control line. The main controller will also need to transmit and receive data from the user control subsystem. This will likely be a touchscreen with an LCD and capacitive or resistive touch overlay. The main controller will need to be capable of rendering simple graphics for the user on the LCD screen as well as receiving the touch coordinates signal from the touch overlay. It may be the case that a separate controller will need to interface with the touchscreen to relieve the main microcontroller of I/O ports. The image processing subsystem will also need to communicate with the central controller. The image processor will identify the LEGO piece and send the data to the main controller. This data lets the main controller know which type LEGO piece was identified with the image processor. The main controller can then tell the rotating arm to move to the appropriate location. The main controller must have a serial communication port as that is the likely method for the controller and image processor to interface and communicate.

Controller boards have a limited number of I/O ports. It may be the case that this project requires more I/O lines than some microcontrollers have built in. There are a few ways this issue can be mitigated. Multiple microcontroller boards can be used to receive and transmit signals through their I/O ports. The controllers can communicate with each other through serial communication ports to operate like one larger control unit. Another way to handle a large number of I/O signals is through added circuitry and chips. There are logic chips built, like shift registers, that use 3 or 4 microcontroller pins to control many output devices. The serial communication port can also be utilized for additional I/O control.

3.2.5 Image Processing

One of the ways discussed to sort the LEGO pieces is through the use of image processing. Image processing is a way to take a picture or video frame input from a camera device and analyze it with algorithms.

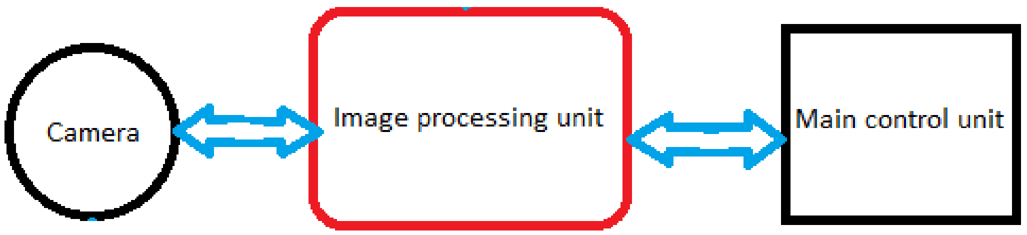


Figure 3.3.4-A

Figure 3.3.4-A shows the general data flow diagram for the image processing subsystem. The image processor operates on the input images received from the camera. The processor then runs algorithms on the image and relays information about it to the main controller.

For this project, LEGO pieces need to be separated based on their shape and color attributes. The color identification is fairly straightforward. The camera will input its image or video frame and the processing device will identify the color based on the pixel information from the camera sensor. Shape identification is a more complicated task. The image processor will need to perform complicated algorithms like edge detection to process the shape seen by the camera sensor. A major issue to consider is that the LEGO pieces will enter the camera's field of view at arbitrary angles and orientations on the conveyer belt. A possible way around this concern is to utilize a shape library of LEGO pieces for the image processor to compare against. The edge detected LEGO piece can be compared with the library of possible LEGO piece orientations to ultimately identify which piece it is. This information can then be sent to the main microcontroller to have the piece be transported into the correct bucket. One of the objectives of this project is for it to be a completely embedded solution. This means that it should not have to be hooked up to a laptop or outside computer to function. Image processing is a fairly computationally intensive procedure and careful consideration must be taken to pick a capable platform. Technologies being considered are; microcontroller, microcomputer, digital signal processor (DSP) development board, and field programmable gate array (FPGA) development board. Detailed research on the various technologies capable of image processing must be done. The image processor unit will need to interface with a camera sensor to identify the LEGO pieces. Depending on the image processing technology used, interfacing can be direct through I/O pins or through a communication port like USB. Many inexpensive webcams exist currently that use USB interfacing. It may be a good idea to take advantage of this fact by using an image processing device that has USB connectivity. The image processor must also be able to communicate with the main microcontroller. The processing platform should contain a serial communication port to transmit data to the controller.

For this project, the image processing can be done on either an image taken every few seconds or on a constant stream of video input. The method of receiving a snapshot every few seconds vastly decreases the processing demands from the image processing unit but would require additional sensors to tell the camera when to take a picture. Each time a LEGO piece enters the camera's field of view, it must be completely within the view without any of it sticking out. Because there are many different shapes and orientations the LEGO pieces can arrive in, the sensors may not be a great solution for consistent results. Alternatively, the image processing unit can take in a video stream, frame by frame. This is much more demanding of the processing unit, but has the advantage of not needing extra sensors to detect whether the LEGO pieces are completely within the camera's field of view. An edge-detect algorithm can run on the processor to determine that the piece's outline is entirely within the image. This method is much more robust than the snapshot method as it does not rely on outside sensors to determine what is in its field of view.

3.2.6 How to Gather the Image

For the image processing chamber that will be used within the project, the most immediate thing that had to be addressed was what method would be used to gather the image that would ultimately be analyzed. It was very clear from the beginning that there would somehow need to be more than one angle of view on the Lego in question in order to gather all of the necessary details for analysis. A couple of different approaches were arrived at for this issue.

The first proposed solution was the most obvious, and that was the use of two separate cameras. One camera would be mounted above the Lego, while the other would be mounted on the side. This would give the two angles that were needed in order to gather enough information to properly analyze the Lego and correctly determine how it needs to be sorted. This creates a couple of minor issues in the process though. With the two separate cameras, it creates a need for more space for the cameras. It also requires a good deal more processing power in order to handle the two separate images, and it would also increase the difficulty of the analysis, in that the coding would also have to handle two separate images. While these issues are certainly not massive, finding a simpler method would be ideal.

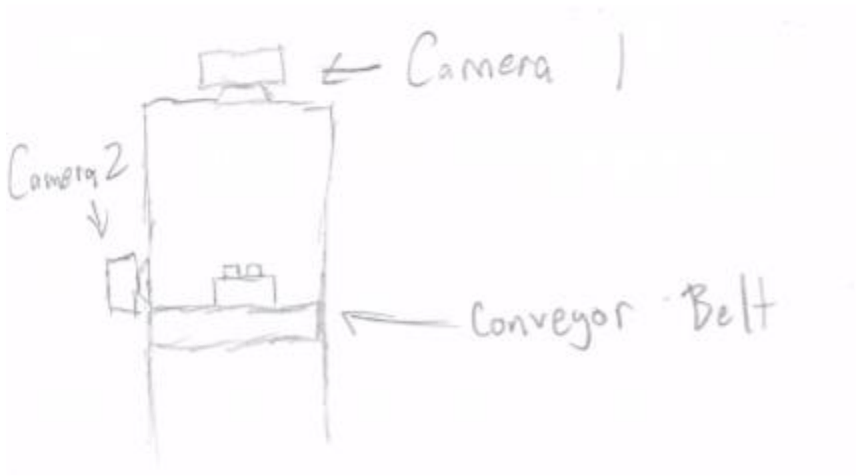


Figure 3.2.6-A Dual Camera option

After seeking advice from a reputable source, the idea of using a mirror was introduced. The thought was that rather than introducing an entirely new image source to get the second angle, the extra angle could be taken by the exact same camera, but by using a mirror at a

45 degree angle to the conveyor belt. This would essentially allow two separate images be processed at the exact same time. This would reduce the camera requirement down to one, and reduce the processing power needed for the image, however it would make setup slightly more difficult with the mirror. Perfectly placing the mirror will take a great deal of precision.

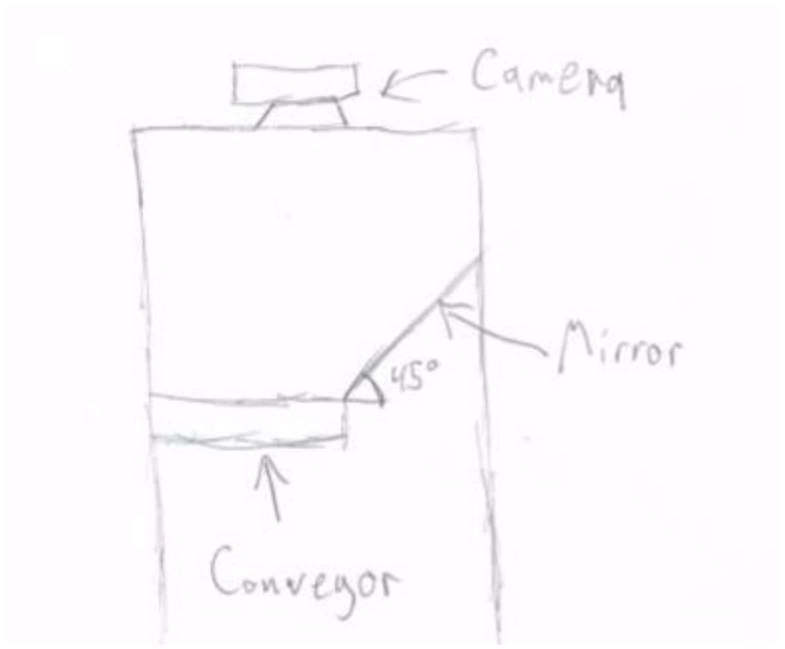


Figure 3.2.6-B: Single Camera w/mirror option

In the end, the ultimate decision was to go with the option that required only one camera and the mirror. While it will take more care to prepare the chamber with that setup, the belief is that the multiple advantages of using that system outweigh the single disadvantage.

3.3 Subsystem Specific Research

For organizational purposes, the project has been broken down into subsystems. Since this project requires a lot of mechanical parts, much of the subsystem specific research is devoted to the achievement of mechanical parts in an electrical/computer engineering project as well as research devoted to the electrical & computer engineering aspects of the project. The figure in 3.3-A below provides a rough sketch of the mechanical components of the project.

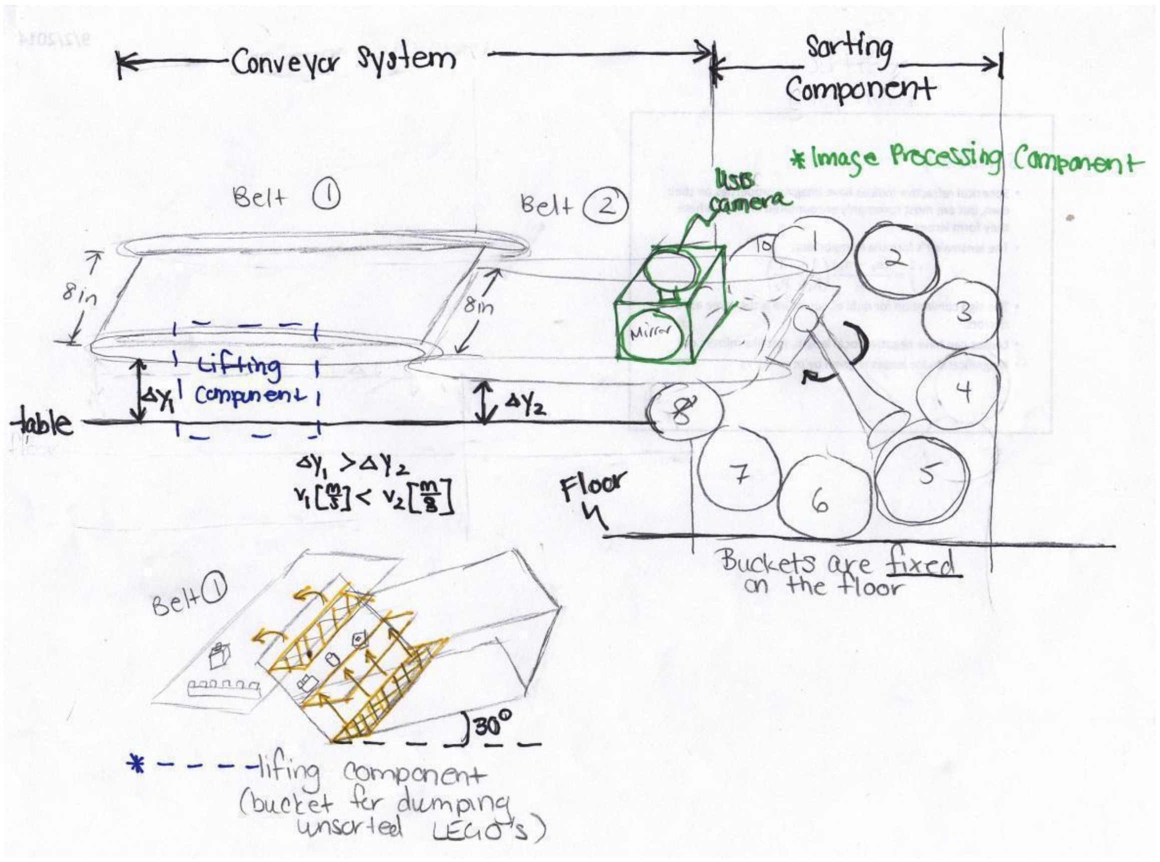

Conveyor System
Sorting Component
*Image Processing Component
USB camera
Mirror
Belt ①
Belt ②
8in
8in
Lifting Component
ΔY_1
ΔY_2
table
$\Delta Y_1 > \Delta Y_2$
$v_1[\frac{m}{s}] < v_2[\frac{m}{s}]$
Floor
10
1
2
3
4
5
6
7
8
Buckets are fixed on the floor
Belt①
30°
* - - - - lifing component (bucket for dumping unsorted LEGO's)

Figure 3.3-A: Mechanical subsystems to be built.

3.3.1 Conveyor Belts

Mechanical Component

Overview - The conveyor belt subsystem generated some issues of concern during research. It is understood that mechanical parts are allowed to be purchased. When browsing pre-fabricated conveyor belts online, it became evident that the conveyor belt system would have to be fabricated instead of purchased due to cost (the minimum price found was around $1300 upwards of $20,000). Since the project requires two belts, the cost was too hefty.

It was decided that there would be 2 belts utilized to initially separate the LEGO parts. The first belt will move as slow as the DC motor will allow for continuous operation. This configuration is needed so that if a huge pile of parts is clumped on the first belt, the slow motion of the belt will allow 1 part to drop onto the second belt at a time (with some error). The second belt will move much faster since the parts will be separated. The exact speed is unknown at this point but will be assumed to be (3x) the speed of the first belt. The second belt will feed into the image processing system.

Research done on conveyor belt construction was not as easy as originally anticipated. Since mechanical engineering is not a requirement of this project, use of pulley systems would be overly complicated for this project and would take too much time to design. During research a makeshift conveyor belt made from paint rollers was found and briefly considered. Figure 3.3.1-A illustrates the construction. Paint rollers were to be screwed to steel panels on the sides and an industrial sewing machine would be used to construct the actual belt out of leather. Lack of access to power tools and an industrial sewing machine quickly ruled out this method of construction.

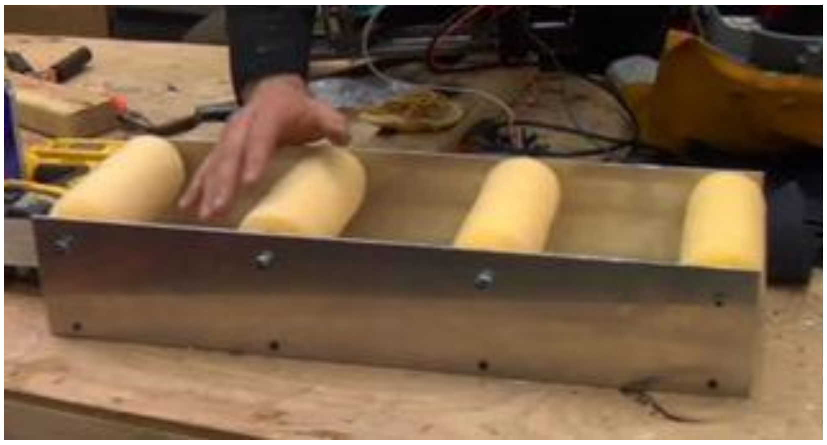


Figure 3.3.1-A Conveyor belt construction from steel and paint rollers(permission pending)

Summary - It was ultimately decided to build the structure of the conveyor belts out of LEGO parts. The beauty of building the structure out of LEGO's is that the size of the belt can be adjusted accordingly as needed. Furthermore, there are many different sizes, rims, and treads of tire parts that can be selected to turn the belt during trial and error testing. The material of the belt is the problematic part. Since the parts are going to be processed by a webcam situated over the second belt, the backdrop needs to be white, which means the material of the actual belt needs to be white. First, a white shiny material was selected from the fabric store. This became too complicated needing a sewing machine and the material did not move smoothly over the LEGO notched belt structure. The next thing utilized was thick artist paper. This worked much better and slid smoothly over the belt structure, but after a couple of weeks the paper started to crumple. A trip to the office supply store provided the desired material. The office supply store prints large posters on photo paper. The photo paper comes in huge reams and is over 3 ft wide, so there is essentially no limitation on width or length in respect to belt material and it costs only $5/ square foot. We learned that for image processing the best backdrop is 17% gray, so we will need to have this color printed onto the photo paper. It was found that the longest LEGO rod (axle) is 10" long. So it is figured that the belt will be around 8" wide. The length of each belt has yet to be determined.

Electrical Component

The conveyer belts serve a couple functions in this project. The first function is moving the LEGO pieces from the lift arm, through the image processing system, and finally into the proper sorting container. The second function the belts serve is as a method of separating the pieces from one another. The first belt, which is elevated above the second belt, will be moving slower. The second quicker moving belt will separate pieces that are dropped within close succession and proximity to each other. This is so the image processing system only looks at one piece at a time without overlaps. The belts need to be stopped and speed controlled depending on the code in the main controller. Basic DC motors seem to be the most logical first consideration to accomplish this task. There are two general approaches to using DC motors with the conveyer belts. The first approach is to use two DC motors, one per belt. The second approach is use one motor, but have a gear system spin the second belt faster than the first. The problem with the second approach is that if one belt stops, then the other must stop as well because of the physical geared connection between them. Independent control of the belts is not possible. The second approach also leaves no room for speed-up efficiency in sorting time when both belts have to stop for the same amount of time. The first approach of using two motors, while more power consuming, leaves much more room for control versatility and speed-up efficiencies.

A way to control the speed of DC motors is through pulse width modulation (PWM). The microcontroller controlling the motors provides pulsed signals to vary the speed. Longer pulses make the motor spin faster while shorter pulses make the motor spin slower. Figure 3.3.1-B below shows the PWM signal generated by the main microcontroller which controls the motor speeds.

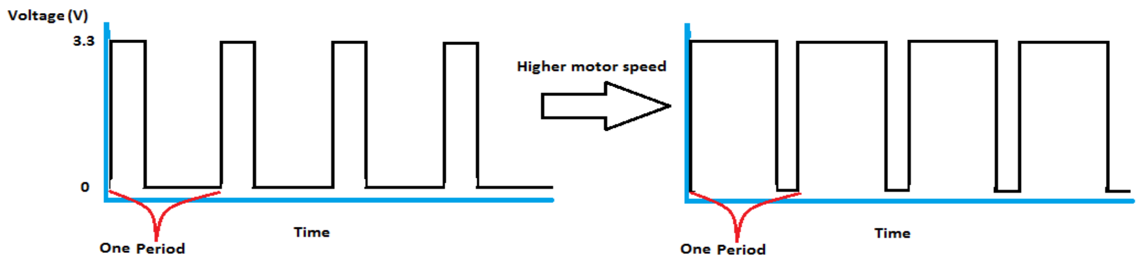


Figure 3.3.1-B PWM signals from the microcontroller vary the speed of the motors using pulse width modulation

As shown above in Figure 3.3.1-B, the longer the pulse is high during each period determines how fast the motors turn. The bottom conveyor belt will need to run at a higher speed than the top to separate LEGO pieces as they fall onto the bottom belt. The bottom belt's motor will be fed a higher duty cycle signal than the top, which will be implemented with code functions and on the microcontroller. Since the motors require higher voltage and current than microcontrollers can supply, extra circuitry is needed. A simple transistor circuit used as a switch should work. Figure 3.3.1-C below shows a simple transistor circuit that separates the PWM control signal from the high power needed to turn the motor.

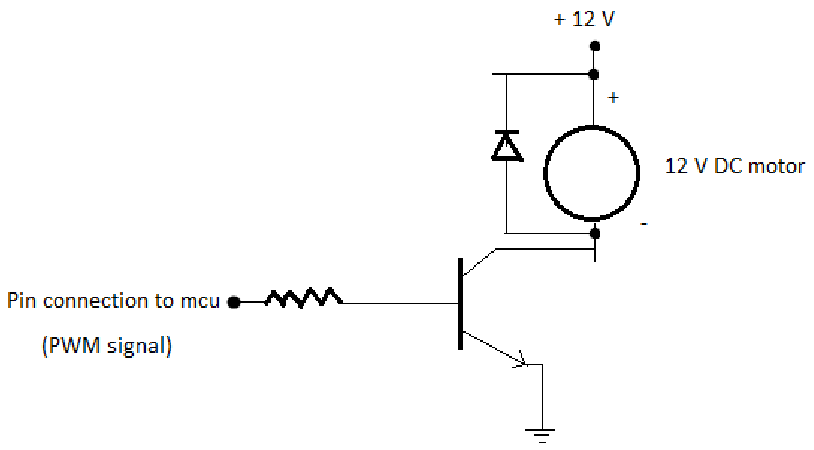


Figure 3.3.1-C: Transistor circuit with flyback diode for DC motor control

Figure 3.3.1-C above shows a transistor acting as a switch to control the 12 V DC motor. The resistor tunes the base current to make sure the transistor is in saturation mode. The diode connected across the DC motor terminals is there to remove the voltage kick-back from the motor coils when the motor is turned on and off. This circuit implementation can be repeated for each of the two DC conveyer belt motors.

3.3.2 Lift Arm

Overview - The lift arm can be considered to be the first electro mechanical stage of the sorting process. There will be a large bucket for the user to dump unsorted LEGO parts. The bucket will have a large LEGO "panel" on the bottom. This panel will have a small moving platform on the end of it situated towards the entrance of the conveyor belt (This is the black "arm" shown in figure 3.3.2A below). The bucket holding the unsorted parts will be situated next to the conveyor belt at a slight 30° angle. When the platform, driven by a servo motor drops out, the LEGOS will fall into the empty slot. The platform will proceed upwards until it interrupts a sensor, either a mechanical switch or IR sensor. At this point, the black "lift arm" will stop and the slight 30° pitch will allow the parts to fall over the side onto the belt.

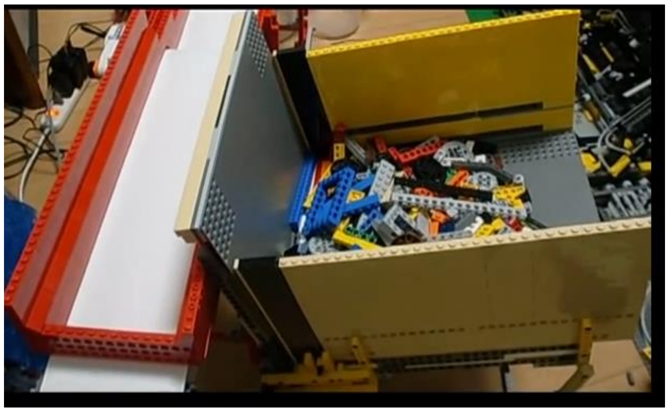


Figure 3.3.2 – A: Sub System Mechanical Lift Arm.

Lift Arm - Figure 3.3.2-A shows the basic concept of the lift arm. A servo motor will be utilized in conjunction with LEGO racks, pinions, rods, and gears to drive the lift arm up and down. A 10" LEGO rod will have gears attached to either side. The space in between the rods will form the platform utilizing "Technic" LEGO bricks that have holes in the middle for the rods to go through.

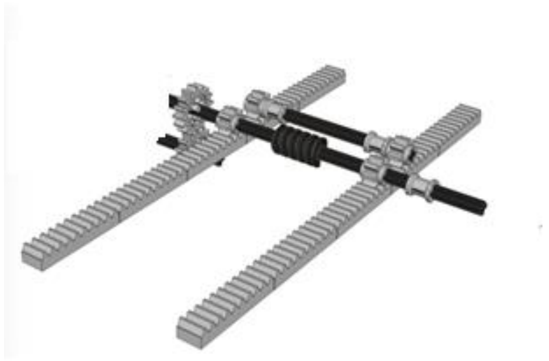


Figure 3.3.2 – B: Servo Motor in Conjunction with LEGO Technic gear racks and pinions

Servo Motor Basics - There are three main types of Servo motors to be considered for the project:

- Positional Rotation
- Continuous Rotation

- Linear Servo

Positional rotation servos rotate a half circle or 180°. These servos rotate between a neutral point and either left or right depending on the control signal applied. This would only be able to move the lift arm a couple of inches up and down which is not helpful for the application of this project. Continuous rotation servos operate on the same principle of positional rotation servos except they can turn in either direction indefinitely. The control signal of this configuration controls the speed of rotation as well as the direction of rotation (CW or CCW) which make the movements much more precise than positional rotation servos. Linear servos would be the best option for the application needs of this project because they have additional gears, which usually include a rack and pinion mechanism such as the one illustrated in figure 3.3.2-B above. This edits the standard circular motion of a servo motor to be converted to a linear up and down motion. Linear servo motors would be lowest in complexity as far as design goes but they are much more expensive than positional or continuous rotation servo motors. The nice thing about this project is the wide range if LEGO technic parts that mimic traditional mechanical gears, pinions, and racks. With some ingenuity the position and continuous rotation motors can be modified with a little elbow grease to behave like linear servo motors.

[http://www.sciencebuddies.org/science-fair-projects/project_ideas/Robotics_ServoMotors.shtml?from=Blog]

Servo Motor Attachment Frame - It was found in a similar project online that LEGO Technic part #64179 can be used in conjunction with an MG995 High Speed RC Servo motor to create an attachment for the servo for the lift arm. This arrangement is shown in figure 3.3.2 -C. [http://www.mocpages.com/moc.php/240340]

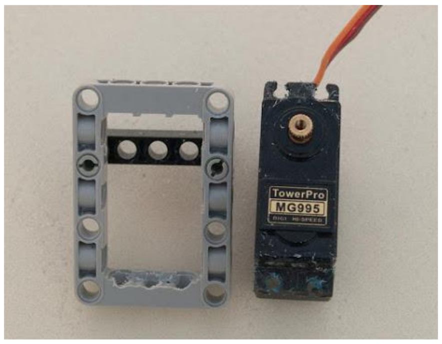


Figure 3.3.2-C Construction of Lift Arm Servo Motor Attachment

The website suggests using a dremel tool, minicraft, or proxxon (die grinder) to grind away some of the technic frame on one end which can be seen in figure 3.3.2-C. The next step is to remove 3-4mm from the servo’s mounting lugs so that it can fit inside the LEGO frame. Plumbers PVC pipe glue was suggested as the best option to weld the parts together, hot glue has also been used successfully. A large Servo Horn will be used to mount Technic LEGO gears to the motor. This will be done by drilling 3/16" holes into the horn and using Technic LEGO pins to join the gears to the servo horn as shown in figure 3.3.2-D. This gives more freedom in designing the motion of the lift arm. With this configuration the motor can move up and down with the attached LEGO frame or alternatively remain stationary and drive the rack up and down.

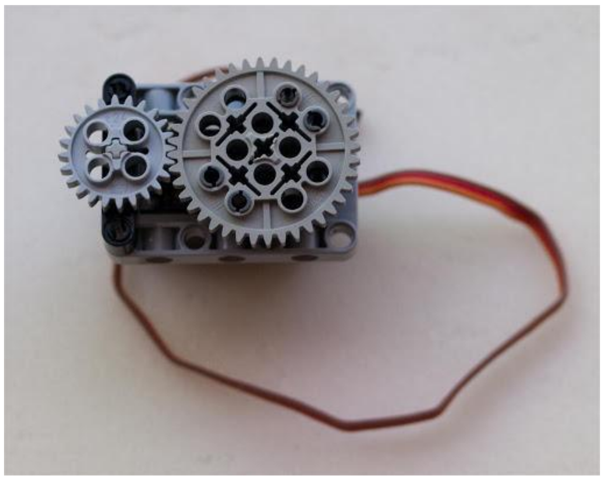


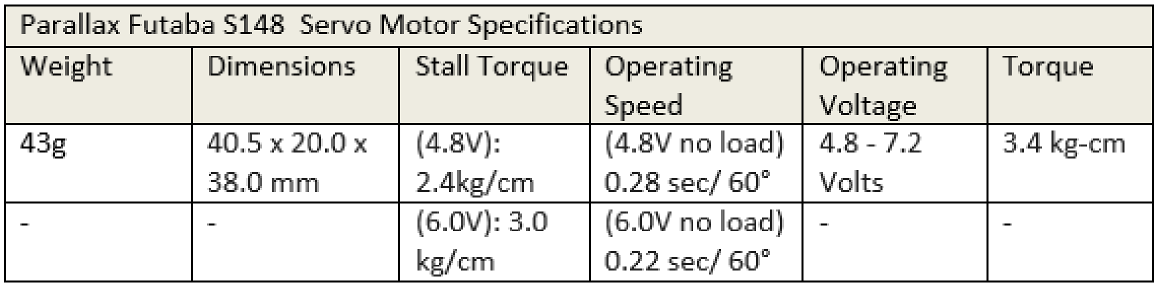
Figure 3.3.2-D: Servo Motor Encased in Technic LEGO Frame w/ Gears attached to Servo Horn

Servo Motor Specification - The Servo motor that was used mounted to the frame is model # MG995. This motor is a positional rotation motor. To be used for the application of the lift arm in this project it would need to be "hacked" to operate like a continuous rotation motor. This motor has been considered since it was found on a LEGO moc pages website that successfully utilized it in constructing a lift arm. The specifications of the motor are shown in Figure 3.3.2-E below.

MG995 RC Servo Motor Basic Specifications					
Weight	Dimensions	Stall Torque	Operating Speed	Operating Voltage	Deadband Width
55g	40.6 x 19.8 x 37.8 mm	(4.8V): 13kg/cm	(4.8V no load) 0.17sec/ 60°	4.8 - 7.2 Volts	5 µs
-	-	(6.0V): 15kg/cm	(6.0V no load) 0.13sec/ 60°	-	-

Figure 3.3.2-E: Positional rotation servo motor specifications

Since the project really requires continuous rotation another servo motor needs to be selected since it would be too much work to "hack" a positional servo. The Parallax (Futaba S148) comes with the appropriate star shaped horns that can be modified to mount a LEGO gear to it. The specifications of the Parallax Futaba S148 can be found in figure 3.3.2-F.



Parallax Futaba S148 Servo Motor Specifications					
Weight	Dimensions	Stall Torque	Operating Speed	Operating Voltage	Torque
43g	40.5 x 20.0 x 38.0 mm	(4.8V): 2.4kg/cm	(4.8V no load) 0.28 sec/ 60°	4.8 - 7.2 Volts	3.4 kg-cm
-	-	(6.0V): 3.0 kg/cm	(6.0V no load) 0.22 sec/ 60°	-	-

Figure 3.3.2-F Continuous Sero Motor Specifications

Sensor - A sensor will need to be utilized for this lift arm. The lift arm needs to go up and down repeatedly dropping LEGO parts onto the conveyor belt. There will be a zero point that is the bottom level. The servo motor will go up from there until it reaches the top part where it needs to reverse. A mechanical switch would be the easiest way to stop the upwards motion. After hitting the mechanical switch the servo motor will reverse directions and go back down to the zero point. The Amico Opto Photo Micro switch is typically utilized to automatically monitor and indicate whether the travel limits of a particular device have been exceeded. The specifications can be found in table 3.3.2 –C. [http://www.ia.omron.com/product/item/eesx1991c/index.html]

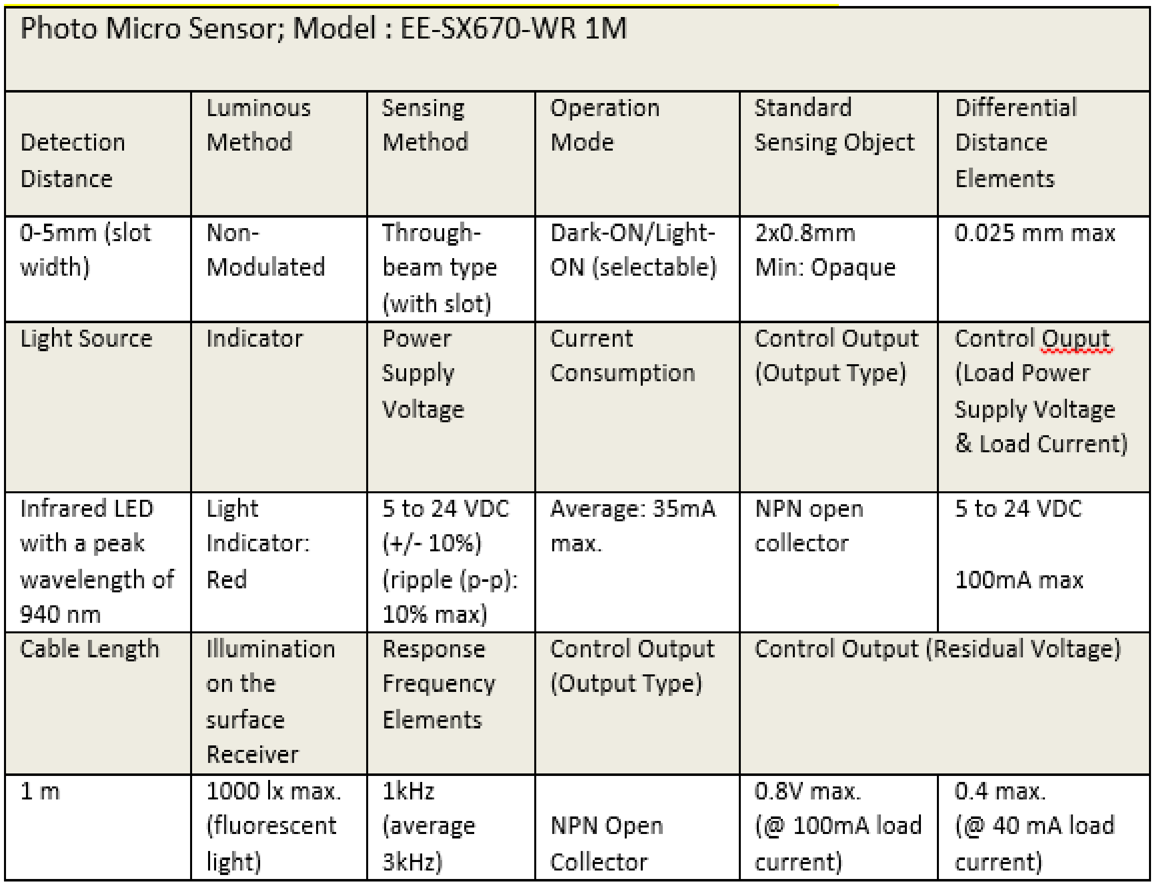


Photo Micro Sensor; Model : EE-SX670-WR 1M					
Detection Distance	Luminous Method	Sensing Method	Operation Mode	Standard Sensing Object	Differential Distance Elements
0-5mm (slot width)	Non-Modulated	Through-beam type (with slot)	Dark-ON/Light-ON (selectable)	2x0.8mm Min: Opaque	0.025 mm max
Light Source	Indicator	Power Supply Voltage	Current Consumption	Control Output (Output Type)	Control Ouput (Load Power Supply Voltage & Load Current)
Infrared LED with a peak wavelength of 940 nm	Light Indicator: Red	5 to 24 VDC (+/- 10%) (ripple (p-p): 10% max)	Average: 35mA max.	NPN open collector	5 to 24 VDC 100mA max
Cable Length	Illumination on the surface Receiver	Response Frequency Elements	Control Output (Output Type)	Control Output (Residual Voltage)	
1 m	1000 lx max. (fluorescent light)	1kHz (average 3kHz)	NPN Open Collector	0.8V max. (@ 100mA load current)	0.4 max. (@ 40 mA load current)

Figure 3.3.2-G: Lift Arm Feedback Sensor Specifications

Summary - The continuous rotation Parallax S148 servo motor was selected to run the lift arm. The star shaped horn attachment of the servo motor will be drilled if necessary with holes to fit a LEGO technic gear. The gears attached to the horn will act as a linear servo motor to drive the pinion up and down the gear rack. The pinion will have a rod attached to it and the rod will be covered with LEGO bricks to form the platform. To start the Platform will be level with the dump bucket. The motor will be configured so the platform drops out and the LEGOS positioned on the 30° pitch will fall down onto the platform. The servo will then be configured to move the platform upwards until it disturbs the light emitted from the sensor. Once the sensor has been tripped, it will send a message to the main microcontroller to reverse the direction of the servo motor to start the process again.

3.3.3 Rotating Arm System

Motor - The sorting bins will be placed in a circular pattern below and around the rotating arm system. The rotating arm will ultimately sort the LEGO pieces by rotating to the proper location and let the LEGO piece fall through it and into the proper bin. The range of motion of the sorting arm will need to be at least 360 degrees to accommodate the location of all the bins. There are two types of motors that were considered for this task: servo motor and stepper motor. Servo motors are great for high speed applications, at thousands of RPM. At high speeds, servo motors hold their torque rating up to 90%. The drawback of servos is their complicated circuitry and high price. Servos must have an encoder to provide feedback to the motor. Because servos have a small number of magnetic poles for the motor shaft to rest at, the encoder and motor will be using more current to keep adjusting itself to maintain the correct position. Stepper motors, on the other hand, have a large number of magnetic poles (between 50 and 100) and require a much less complicated driving circuit. Because of the greater number of poles, the stepper motor draws less current to keep the motor at a position under load. For this reason, stepper motors are ideal for lower speed applications. The rotating arm only needs to rotate at a low speed, possibly in the range of 60 RPM or less. The load will be very small on the motor, considering it will just rotate a light-weight shaft and LEGO slide. With this information in mind, the stepper motor would be the better choice for the task.

There are two main types of stepper motors to consider: unipolar and bipolar. Unipolar stepper motors require less circuitry than bipolar but are less efficient and provide less torque. Bipolar motors are simpler in design, but the driving circuitry is more complex. Both motor types can be implemented so that they use two or four pins from the main microcontroller. The bipolar motor will be used for this project because of its higher efficiency. The more complex circuitry to drive the bipolar motor is not an issue as it can be packaged in a tiny IC chip. A simplified diagram of a bipolar stepper motor is shown below in Figure 3.3.3-A.

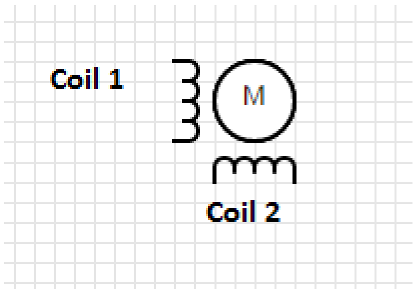


Figure 3.3.3-A: Simplified bipolar stepper motor diagram

In the Figure 3.3.3-A above, the two coils would be polarized in an alternating fashion to step the motor through its rotation. In an actual stepper motor, there are many poles to increase the step resolution and total possible number of angles the motor can hold at.

Motor Circuitry - The bipolar stepper motor was chosen for the task of moving the rotation arm. For the motor to be controlled by the main microcontroller in the project, a stepper motor driver circuit must be integrated. A common way to control stepper motors is through the use of an H-bridge circuit. The H-bridge circuit allows for current to flow through a device in either direction. Stepper motors usually are connected to an H-bridge to allow its coils to be energized in either direction. An H-bridge concept diagram is shown below in Figure 3.3.3-B.

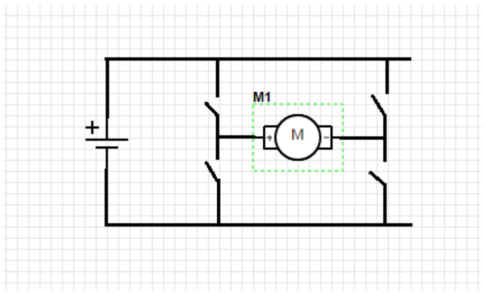


Figure 3.3.3-B: H-bridge conceptual circuit

The switches in Figure 3.3.3-B above are transistors in the actual H-bridge circuit. Biasing resistors and flyback diodes are common in H-bridge circuits as well. To save project complication and space on the PCB board, the bipolar stepper motor will be driven by an IC chip. Shown below in Figure 3.3.3-C is a L2394D duel H-bridge IC.

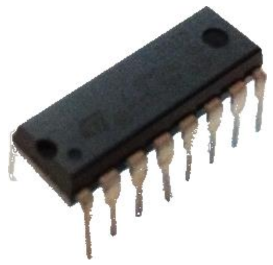
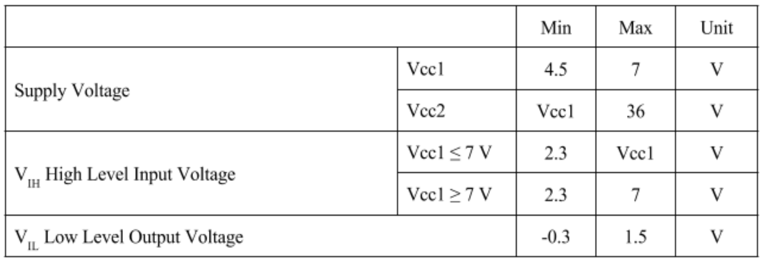


Figure 3.3.3-C L2394D duel H-bridge IC

Since stepper motors have two coils to control, the use of two H-bridge circuits is needed. The L2394D has two integrated H-bridges and is ideal for this project. Consideration must be taken to find an appropriate bipolar stepper motor that will not overload the H-bridge chip in any way. The recommended operating conditions of the H-bridge IC are shown below in Figure 3.3.3-D.



		Min	Max	Unit
Supply Voltage	Vcc1	4.5	7	V
	Vcc2	Vcc1	36	V
V_{IH} High Level Input Voltage	Vcc1 ≤ 7 V	2.3	Vcc1	V
	Vcc1 ≥ 7 V	2.3	7	V
V_{IL} Low Level Output Voltage		-0.3	1.5	V

Figure 3.3.3-D: Recommended voltage levels of the L2394D duel H-bridge IC

The motor supply voltage on pin “V2” in Figure 3.3.5-1 can be in the range of 4.5V – 36V. This gives a fair amount of flexibility for the choice of motor. The pin “V1” shown in the same figure can be in the range of 4.5V to 7V. Connected to that pin will be a logic output pin from the main microcontroller in the 5 V range.

Sensor - The rotating arm system should use a sensor as feedback to ensure the correct positioning of the arm during each rotation. At the very least, a “home” position for the rotating arm stepper motor needs to be established each time the system is started up. This can be done many ways with various types of sensors. The initial consideration is through use of a color sensor that can identify which sorting container it is facing at all times. Each container can be a different color with the sensor attached to the rotating arm. The advantage of this method is that it provides a continuous stream of feedback to the microcontroller to reduce poor arm positioning. A sketch of the possible physical layout of the sensor incorporated into the rotating arm system is shown below in Figure 3.3.3-E

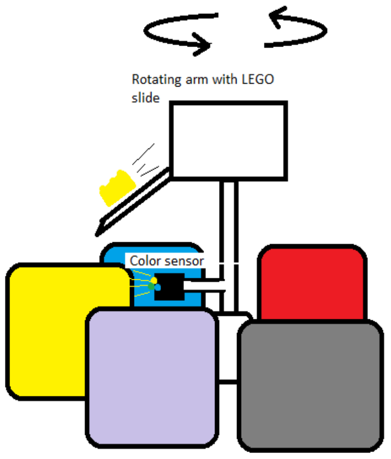


Figure 3.3.3-E: Sketch of rotating arm with color sensor and LEGO bins

As can be seen in Figure 3.3.3-E above, the color sensor is attached to the shaft of the rotating container that the LEGO falls through. The sensor is extended outwards to face the colored sorting containers as the shaft and LEGO slide rotate around. The sensor makes sure that the initial rotation by the stepper motor is adequate by relaying the color information of the containers to the microcontroller.

A possible drawback to this configuration is outside lighting. If the sensor is not completely contained in a system with controlled lighting, ranges of color values may be different from location to location. This issue can be mitigated with a completely contained sensor system, or perhaps through some clever programming.

Possible sensors considered are the Color Sensor Breakout HDJD S822, Geeetech TCS230 and TCS34725. Specifications and features of each sensor are shown below in Figure 3.3.3-F.

	Supply voltage	Unique Features	Interfacing	Price
HDJD S822	5 V	Analog output	GPIO	\$24.95
TCS230	2.7 – 5.5 V	Light intensity current to frequency conversion	I2C	\$14.00
TCS34725	3.3 – 5 V		I2C	\$7.95

Figure 3.3.3-F

The HDJD S822 has a more intuitive design with the RGB sensors each outputting an analog value based on the intensity of the respective colors detected. The TCS320 is cheaper and has a more powerful integrated feature that converts the current of each color channel into a frequency that can be read by the microcontroller. The TCS34725 requires the least amount of pins and is the most cost effective solution. The TCS34725 is shown below in Figure 3.3.3-G.

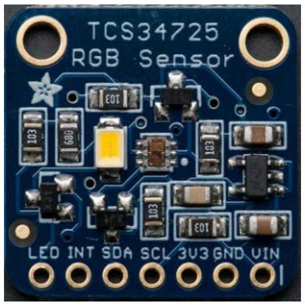


Figure 3.3.3-G (Permission Pending)

3.3.4 Image Processing (technologies research)

There are a few technological barriers to consider when deciding on the image processing device. One potential barrier is the data size of the image or video frame, which is determined by a number of factors. For this project, LEGO pieces need to be identified by their color and shape. With this in mind, the color depth as well as the resolution of the image must be sufficiently high to identify the pieces successfully.

There are a total of 58 different LEGO colors[1] that exist that can potentially be ran through the LEGO sorter. The image processing device must operate on a color depth that can distinguish each color. A couple issues to note is that some LEGO pieces are very similar in color and there may be situations where shadows or reflections on the LEGO pieces are registered as the wrong color by the image processing device. The way the pieces enter the camera's field of view cannot easily be controlled. The color depth will have to be able to accommodate for these variations as well. Some typical color depths that are used in various camera sensor and display applications are 8-bit color, 16-bit High color, and 24-bit True color. The numbers of colors these depths can "resolve" are 256, 65k, and 16.7M respectively. Because of the issues mentioned above, 256 identifiable colors may not be enough to resolve the pieces reliably. The 16-bit High color may be a good depth to use with over 65,000 colors. Resolution of the image is also important for edge detection of the LEGO pieces. The resolution does not need to be high definition, but needs to be high enough for detecting the edge of the LEGO against a solid color background. VGA format (640x480) should be a sufficient resolution for this project. With the color depth of 16 bits and a resolution of 640x480, this gives an image size of 614,400 bytes, or about 614 Kb. The image processing device must have enough onboard memory to hold an image of this size. It may be possible to have a smaller image, but for reliability purposes, the device used in the project should have hardware that is able to exceed this value. Knowing the above, there are various technologies that can accomplish the image processing needed for this project. Among the platforms to consider are: microcontrollers, microcomputers, digital signal processing (DSP) development boards, and field programmable gate array FPGA development boards.

Microcontrollers, while generally not suitable for higher end image processing, are inexpensive and consume little power relative to the other technologies listed above. A table of some popular microcontroller development boards, their prices, and relevant specifications is shown below in Figure 3.3.4-A.

Microcontroller name	CPU	SRAM (bytes)	Code flash memory (bytes)	Clock (Hz)	Development board Price	CPU Price
MSP430G2553	16-bit RISC CPU	0.5K	16K	16M	$9.99	$2.80
MSP430FG4618	16-bit RISC CPU	8K	116K	8M	$117.00	$17.54
Arduino Due	Atmel SA M3X8E A RM Cortex-M3	96K	512K	84M	$44.93	$12.28
Tiva C Series TM4C1294	ARM Cortex-M4	256K	1M	120M	$19.99	$18.64

Table 3.3.4-A Microcontroller development boards being considered

The MSP430G2553 is a great, very low power and cost effective device, but the on-board SRAM of only 0.5KB is not anywhere near the 614KB figure calculated for the size of the image. Not only that, but there is no USB interface for USB webcam. It seems there would be no simple way for this microcontroller to handle the image processing for the project. The MSP430FG4618 is a development and experimenter board used for learning and developing embedded projects. This board also comes up short in the memory department for this project at only 8K in on-board SRAM. There is also no USB interface for the camera. It will not be used. The Arduino Due is a higher end microcontroller that is popular among hobbyists. The Arduino was initially considered for this project because of the large online community support around the board. However, looking at the technical specifications, the SRAM is again, not high enough to hold a relatively low resolution image to be processed. The Tiva C series TM4C1294 was also initially considered because of its higher overall specs relative to other popular microcontrollers. It's 256KB SRAM memory may be high enough to hold lower resolution images, but it might be pushing the limits for use in this project. The hardware specs should exceed the rough estimates of memory usage for the project, which this device does not do.

Because none of the microcontrollers initially considered for image processing in the project met the specification requirements, the technology will not be used. Another technology that was initially considered is microcomputers. Microcomputers, like microcontrollers, have a general purpose central processor. Unlike microcontrollers, microcomputers are capable of running an operating system and control a large variety of peripherals. The use of an operating system, while more power consuming, has the benefit of allowing for easier use of code libraries such as OpenCV. Microcomputers also have the added benefit of more onboard memory compared with microcontrollers. Table 3.3.4-B below shows some relevant specifications of two popular single-board microcomputers.

	CPU	RAM(bytes)	Clock(Hz)	Relevant peripherals	Storage	Price
Raspberry Pi B	ARM1176JZFS	512M	700M	(2) USB host, (17) GPIO pins	External SD (up to 2GB)	$39.95
Beaglebone Black Rev C	AM3359 Sitara ARM Cortex-A8	512M (DDR)	1G	(1) USB host, (65) GPIO	Onboard 4GB eMMC	$55.00

Table 3.3.4-B

As shown in the Table 3.3.4-B above, The Raspberry Pi Model B is plenty capable of the image processing requirements with 512MB of onboard RAM and a 700 MHz CPU clock speed. The Raspberry Pi also has a dedicated floating point unit (FPU). The Beaglebone Black Rev C is also very capable of meeting and exceeding the image processing demands for the project. The Beaglebone is $15 more than the Raspberry Pi, however, it beats the Raspberry Pi in computing power and memory. Both boards are capable of running OpenCV and have large support communities, albeit, Raspberry Pi's community is a bit larger. Both boards have a USB host port(s) that can be used for webcam integration. The Beaglebone is a fair amount faster, which will aid in the speed of the LEGO piece identification.

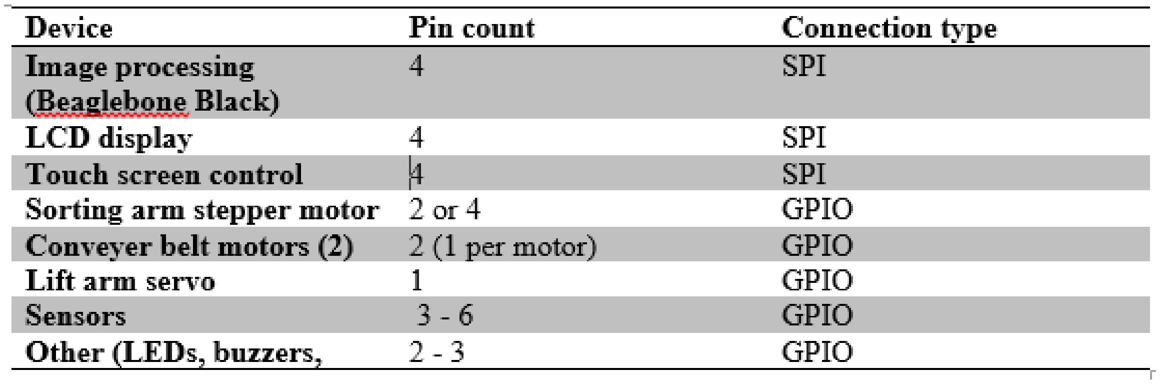
Another technology initially considered is DSP development boards. DSP boards are similar to microcontrollers in their functionality and lack of an OS, however they have specialized built-in hardware for dealing with real-time digital signal processing. One DSP board considered was the TMS320C6748 DSP Development Kit by Texas Instruments. It features a dedicated DSP CPU at 456MHz, 128 MB DDR3 SDRAM, and (1) USB host. It costs $195.00. The DSP kit is faster in computing than both microcontrollers and microcomputers, but they are pricier and may take longer to develop due to the lack of community support.

FPGA boards were also initially considered. FPGAs differ from the previous technologies in that there is no already-built processor. Instead of a general CPU, the programmer designs the circuitry to directly handle the processing of a signal or image. This technology is much faster than the previously listed ones. The speed that FPGAs process at is not required for this project. The price range of FPGA kits is generally higher than the other tech with the Altera Cyclone II FPGA Starter Dev Kit at $199.00. When all is considered, FPGA technology is overkill in terms of processing speed for this project. Due to the limited amount time to design and build the project, FPGA is not a feasible solution.

Conclusion - Most microcontrollers are too slow and do not have enough onboard memory to process images. DSP development kits have more than enough speed and memory to handle the task, but are more expensive and have less resources and community help to prevent and resolve design roadblocks. FPGA kits are expensive and way overkill in terms of processing speeds for the comparatively low level image processing demands of the project. While microcomputers use more power to run an OS on the CPU, they contain enough processing power and memory to get the job done. The two microcomputers considered, the Raspberry Pi and Beaglebone Black Dev C, both have good online support and can run useful libraries like OpenCV. In the end, the Beaglebone Black Dev C seems like the best choice for the image processing of the project because of its quick processing speeds, high amount of memory, and it's relatively low price.

3.3.5 Main Microcontroller

The main microcontroller will be the central processing device that connects all the sub systems together. It will receive data from the image processing system, touchscreen controller, and the various sensors throughout the project. The microcontroller will also be in charge of controlling the lift arm motor, sorting arm motor, conveyer belt motors, as well as displaying the user interface on the LCD. The main controller needs enough I/O pins to communicate with and control each sub system. Figure 3.3.5-A below shows the estimated pin usage of the various devices that will be connected to the main microcontroller.

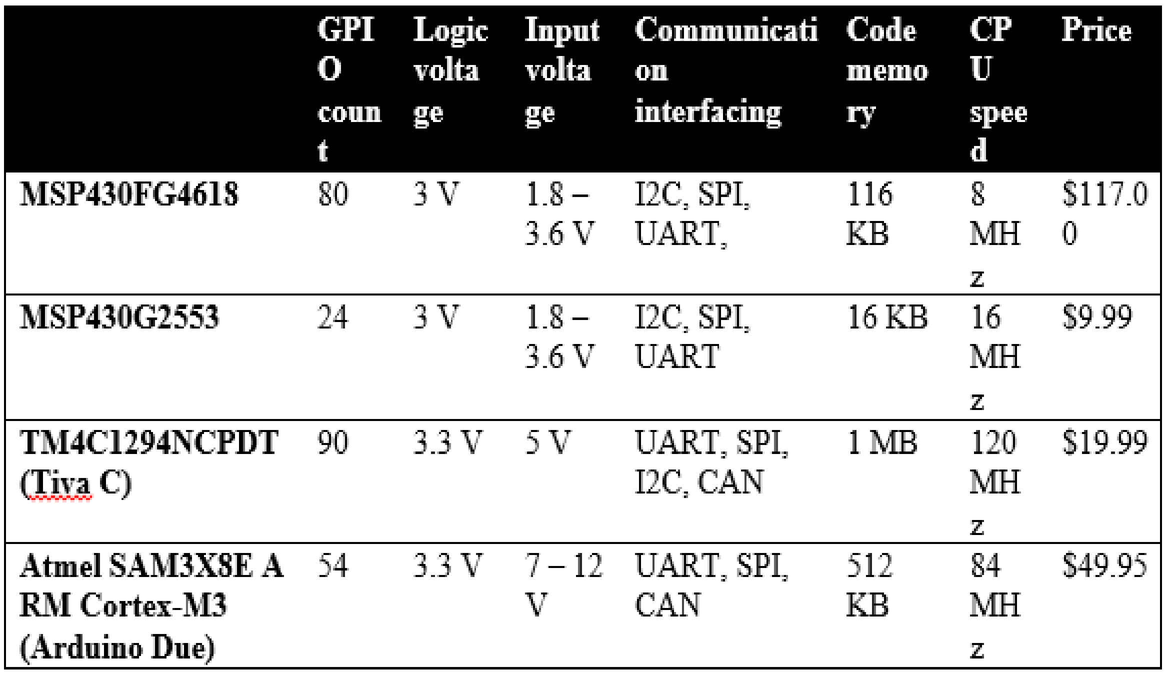


Device	Pin count	Connection type
Image processing (Beaglebone Black)	4	SPI
LCD display	4	SPI
Touch screen control	4	SPI
Sorting arm stepper motor	2 or 4	GPIO
Conveyer belt motors (2)	2 (1 per motor)	GPIO
Lift arm servo	1	GPIO
Sensors	3 - 6	GPIO
Other (LEDs, buzzers,	2 - 3	GPIO

Figure 3.3.5-A

From the table, the high estimate is 29 pins in total could be used by devices connected to the controller. Twelve of those pins must be involved with SPI master-slave interfacing. While the GPIO pin count may be reduced by adding logic devices like shift registers, it may be best to err on the side of caution to select a controller with more I/O pins.

Other considerations during microcontroller selection include: logic voltage of the GPIO pins, supply voltage, current rating, serial/parallel interfacing, CPU speed, and price. Figure 3.3.5-B below shows various microcontroller platforms initially considered for the project with relevant specifications:



	GPIO count	Logic voltage	Input voltage	Communication interfacing	Code memory	CPU speed	Price
MSP430FG4618	80	3 V	1.8 – 3.6 V	I2C, SPI, UART,	116 KB	8 MHz	$117.00
MSP430G2553	24	3 V	1.8 – 3.6 V	I2C, SPI, UART	16 KB	16 MHz	$9.99
TM4C1294NCPDT (Tiva C)	90	3.3 V	5 V	UART, SPI, I2C, CAN	1 MB	120 MHz	$19.99
Atmel SAM3X8E ARM Cortex-M3 (Arduino Due)	54	3.3 V	7 – 12 V	UART, SPI, CAN	512 KB	84 MHz	$49.95

Figure 3.3.5-B Specifications on the considered platforms for image processing. The MSP board prices reflect the cost of the development board.

From figure 3.3.5-B it can be seen that all but one microcontroller provide enough I/O pins for the project. The MSP430G2553 was initially considered because of the familiarity of its usage as well as its low power reputation. The controller, however, falls short of the 29 pin estimate for this project and will therefore not be used.

The rest of the controllers meet the project requirements so selection comes down to the most power for the price. The TM4C1294 is the cheapest of the remaining choices and it contains the most I/O pins as well as the fastest clock rate. It also has a recommended 5 V input, which can be shared with the other devices connected to the 5 V power supply output. An extra voltage output does not need to be designed on the power supply to accommodate the controller.

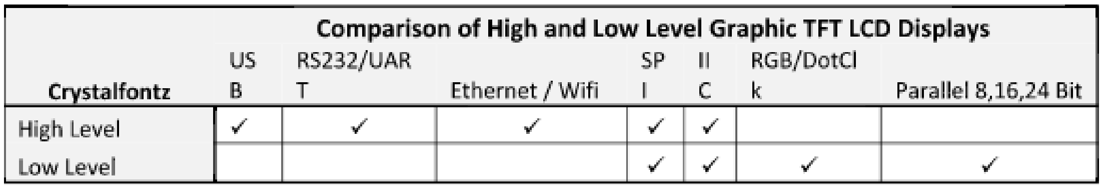
3.3.6 LCD Screen

Size - The menu for the user interface will need to be moderate in size. A 3” display was considered but it will probably be too small to accommodate all of the menu options. A 7” display may be able to be utilized but seemed to be a little too large for the scope of the project. A 5” display seemed to be the appropriate size.

Resolution and Response Time - Resolution and Response time were a couple more considerations. Since the project requires a simple user interface it was ultimately decided to use an LCD with a lower resolution display. Response time is also not a huge issue since the user will only need to input a couple of selections.

Memory - Memory may pose unforeseen challenges later on in the project. The user interface will need a top menu to select method of sorting, as well as three lower level menus for user “sub” - selections. Combine that with 19 million different LEGO parts per year and you will need a display with a pretty good amount of memory. Granted, this project will be limited to a finite number of LEGO parts to be sorted. This is another reason that a low resolution display was decided upon since a higher resolution display will take up more memory.

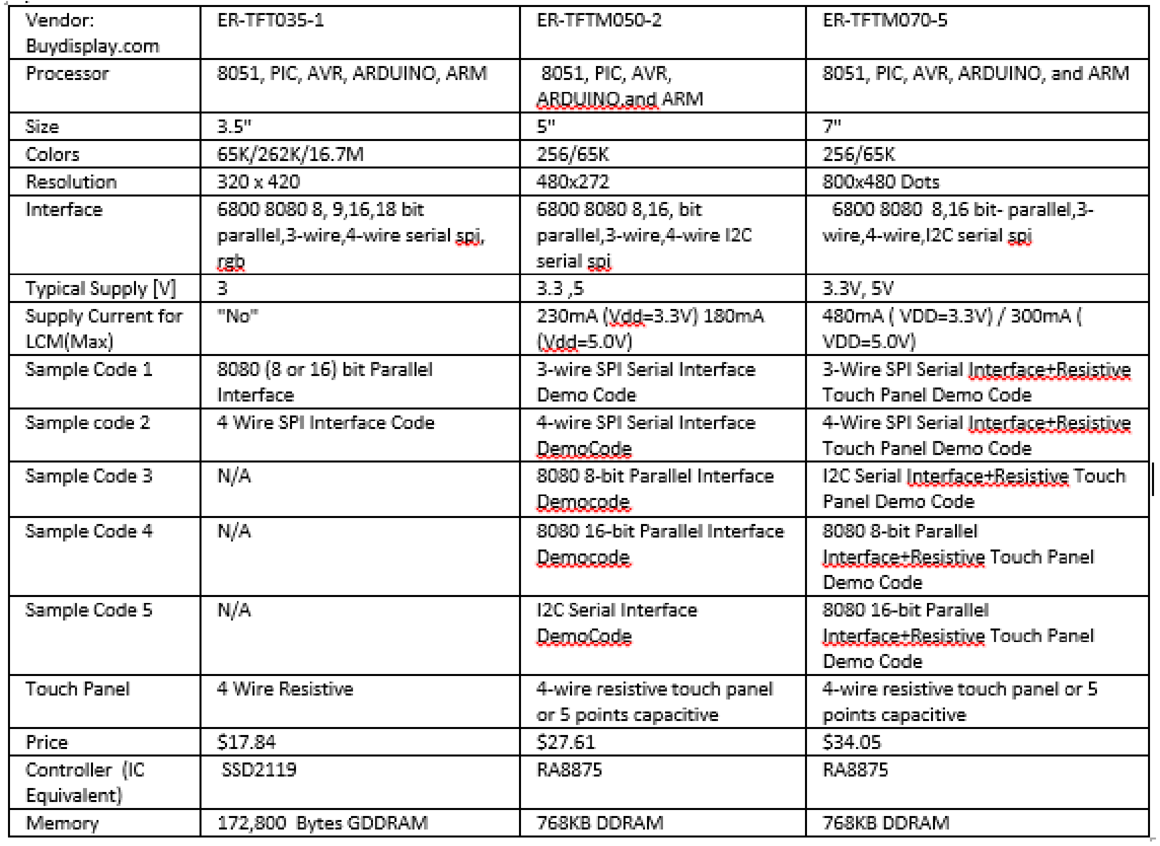
Graphic vs. Character Display - It was decided that a “graphic” LCD display will be utilized to create a menu for the user to select the method of sorting. A “character” display is too simple for the user interface desired. A TFT LCD is more than enough to accomplish the job at an affordable price. The only drawback is the TFT LCD consumes more power to drive the backlight (in comparison to a top of the line OLED display). There are two options for a TFT LCD display: High or Low level. The high level version interfaces to an on-board microcontroller whereas the low level version interfaces to a display controller. [2 - Crystalfontz.com]. The high level LCD displays were a lot pricier and not necessary for the project but they do have different interfacing options such as UART/RS232 connections , USB, and Ethernet/wifi interfaces.



	Comparison of High and Low Level Graphic TFT LCD Displays						
Crystalfontz	USB	RS232/UART	Ethernet / Wifi	SPI	IIC	RGB/DotClk	Parallel 8,16,24 Bit
High Level	✓	✓	✓	✓	✓		
Low Level				✓	✓	✓	✓

Figure 3.3.6 – A

Interfacing the LCD display Controller to the Main Microcontroller- Figure 3.3.6-B offers a clear representation of some specifications that need to be considered. The TFT LCD displays were all found on buydisplay.com. Interfacing the LCD display will propose some of the biggest challenges from the hardware point of view. For example, if an LCD is purchased that communicates with the Motorola 6800 series through 8 bit parallel interface and you try to connect it with an arbitrary microcontroller that only offers 4 wire SPI interfacing, you are going to run into major problems and a lot of valuable time will be wasted because the two devices do not have a compatible interface. It is imperative that the LCD is paired with an appropriate microcontroller that has a compatible interface. The nice thing about the TFT displays found on buydisplay.com is that they all have sample code to help initialize interfacing.Buydisplay.com also offers “accessories” for interface, meaning that each TFT LCD from their website offers a selection of options for interfacing instead of selling their displays with a fixed interface for each device as crystalfontz.com does. This makes it much easier to select an LCD since you can add more than 1 option to interface between the LCD controller and the microcontroller. Buydisplay.com also offers PDF datasheets for the touch panel controllers, TFT (Thin Film Transistor) LCD (Liquid Crystal Display) module, and for the IC equivalent controllers.



Vendor: Buydisplay.com	ER-TFT035-1	ER-TFTM050-2	ER-TFTM070-5
Processor	8051, PIC, AVR, ARDUINO, ARM	8051, PIC, AVR, ARDUINO and ARM	8051, PIC, AVR, ARDUINO, and ARM
Size	3.5"	5"	7"
Colors	65K/262K/16.7M	256/65K	256/65K
Resolution	320 x 420	480x272	800x480 Dots
Interface	6800 8080 8, 9,16,18 bit parallel,3-wire,4-wire serial spi, rgb	6800 8080 8,16, bit parallel,3-wire,4-wire I2C serial spi	6800 8080 8,16 bit- parallel,3-wire,4-wire,I2C serial spi
Typical Supply [V]	3	3.3 ,5	3.3V, 5V
Supply Current for LCM(Max)	"No"	230mA (Vdd=3.3V) 180mA (Vdd=5.0V)	480mA (VDD=3.3V) / 300mA (VDD=5.0V)
Sample Code 1	8080 (8 or 16) bit Parallel Interface	3-wire SPI Serial Interface Demo Code	3-Wire SPI Serial Interface+Resistive Touch Panel Demo Code
Sample code 2	4 Wire SPI Interface Code	4-wire SPI Serial Interface DemoCode	4-Wire SPI Serial Interface+Resistive Touch Panel Demo Code
Sample Code 3	N/A	8080 8-bit Parallel Interface Democode	I2C Serial Interface+Resistive Touch Panel Demo Code
Sample Code 4	N/A	8080 16-bit Parallel Interface Democode	8080 8-bit Parallel Interface+Resistive Touch Panel Demo Code
Sample Code 5	N/A	I2C Serial Interface DemoCode	8080 16-bit Parallel Interface+Resistive Touch Panel Demo Code
Touch Panel	4 Wire Resistive	4-wire resistive touch panel or 5 points capacitive	4-wire resistive touch panel or 5 points capacitive
Price	$17.84	$27.61	$34.05
Controller (IC Equivalent)	SSD2119	RA8875	RA8875
Memory	172,800 Bytes GDDRAM	768KB DDRAM	768KB DDRAM

Figure 3.3.6-B LCD Specification Comparison

3.3.7 Power Supply

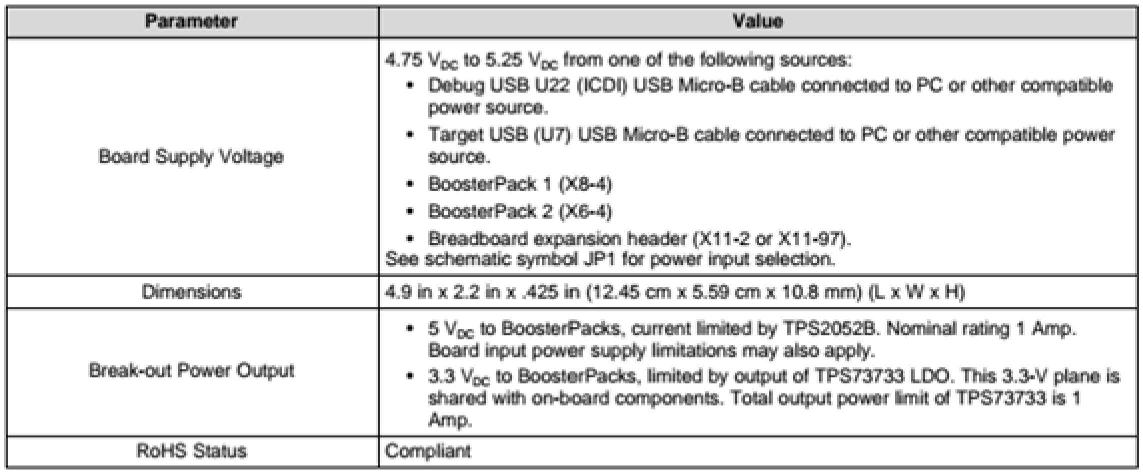
Overview - A power supply is nothing more than a device that is used to "step down" or "step up" a voltage. There are many factors that need to be explored when designing a power supply. Since the objective of this project is for a user to turn on the sorter and leave it on overnight, it is imperative that the supply is stable enough to run continuously for long periods of time. Considering that the sorter is to be used in home at all times, the input to be converted will be from a typical 120V AC wall socket (US) to constant DC voltage(s). Therefore, this project will require "stepping down" an AC voltage to a DC voltage.

Subsystem Power Consumption - The first step in designing a power supply is finding out what voltage (or current) of which each device operates. The project has been broken down into subsystems for simplicity. The subsystems to be dissected: conveyor system, lift arm system, image processing system, rotating arm, & User Interface. Every component used in each subsystem that consumes power will need to be taken into account by finding the voltages and currents of which these devices operate. It is also important to note that some devices may operate at the same voltage but different current. Once the summation of voltages and currents drawn by each device from each and every subsystem is determined, the power supply can be rated and designed accordingly.

Item	Rated Voltage	Rated Current	Power Consumed
Rotating arm stepper motor	12 V	330 mA	3.96 W
Conveyor Belt DC motor (1)	12 V	300 mA	3.6 W
Conveyor Belt DC motor (2)	12 V	300 mA	3.6 W
Lift arm (servo)	5 V	300 mA	1.5 W
Tiva C	(4.75 – 5.25) Note: Use 5V	300 mA (estimated)	1.5 W
LCD screen (User Interface)	[3.3 or 5] V	[180 or 230] mA	[0.594 or 1.15] W
Beaglebone Black	5 V	(210 - 460) mA	(1.05 – 2.3) W
Sensors	5 V	50 mA (estimated)	0.25 W
Rated System Total		2.27 [A]	17.86 [W]

Figure 3.3.7-A: Estimated power consumption.

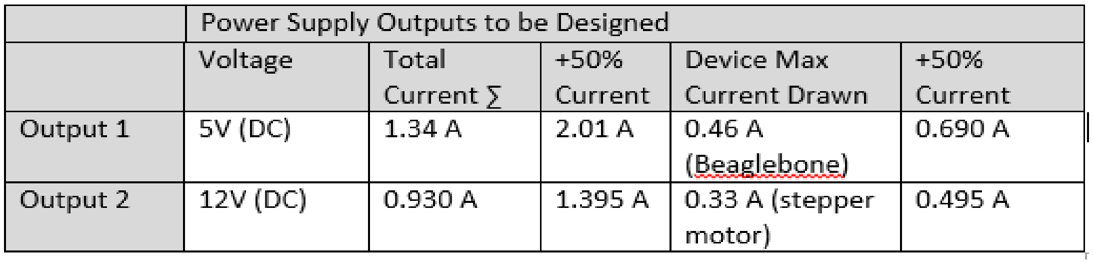
Figure 3.3.7-A indicates the operating voltages and currents of the devices to be used in the project. The LCD screen has a typical output voltage of 3.3 or 5V. Since the Tiva C, Beaglebone Black, Lift arm servo, and sensors all operate at 5V, the LCD supply voltage was chosen to be 5V as well for simplicity. It was found in the User Guide that the Tiva C (EK – TM4C129XL) has an operating voltage of (4.75 – 5.25) Vdc, but there is no current listed (see Figure 3.3.7-B). Since the Beaglebone Black is much more powerful than the Tiva C, the current was estimated to be around 300mA and a 5V input will be used to simplify design and match the other devices.



Parameter	Value
Board Supply Voltage	$4.75\ V_{DC}$ to $5.25\ V_{DC}$ from one of the following sources: • Debug USB U22 (ICDI) USB Micro-B cable connected to PC or other compatible power source. • Target USB (U7) USB Micro-B cable connected to PC or other compatible power source. • BoosterPack 1 (X8-4) • BoosterPack 2 (X6-4) • Breadboard expansion header (X11-2 or X11-97). See schematic symbol JP1 for power input selection.
Dimensions	4.9 in x 2.2 in x .425 in (12.45 cm x 5.59 cm x 10.8 mm) (L x W x H)
Break-out Power Output	• $5\ V_{DC}$ to BoosterPacks, current limited by TPS2052B. Nominal rating 1 Amp. Board input power supply limitations may also apply. • $3.3\ V_{DC}$ to BoosterPacks, limited by output of TPS73733 LDO. This 3.3-V plane is shared with on-board components. Total output power limit of TPS73733 is 1 Amp.
RoHS Status	Compliant

Figure 3.3.7-B: Tiva C power supply specifications

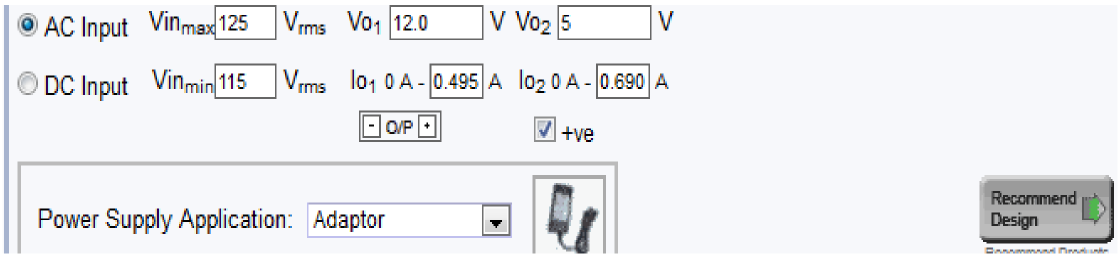
Power Rating - The devices selected for the project all operate around 5V or 12V. Therefore, there will be two output voltages that need to be designed. For the devices that operate at 5V, the Beaglebone Black draws the most current @460mA. For that reason, the 5V output (output 1) will be designed around the Image Processing System - Beaglebone. For the devices that operate at 12V, the rotating arm stepper motor draws the most current @ 330mA. For that reason, the 12V output (output 2) will be designed around the Rotating Arm – Stepper Motor. Please refer to Table 3.3.8 – c.



	Power Supply Outputs to be Designed				
	Voltage	Total Current ∑	+50% Current	Device Max Current Drawn	+50% Current
Output 1	5V (DC)	1.34 A	2.01 A	0.46 A (Beaglebone)	0.690 A
Output 2	12V (DC)	0.930 A	1.395 A	0.33 A (stepper motor)	0.495 A

Figure 3.3.7-C: Power supply outputs to be designed

Design Flaws and Non-Idealistic Characteristics - In real life, the voltages and current ratings given in data sheets are only very good estimates at best. There are statistical deviations in all electronic devices no matter how well they are designed. Electrical circuit components are very sensitive devices. For example, transformers store energy in the form of electromagnetic fields. These EMF fields can create noise in nearby circuit components if they are not isolated appropriately. The wall supply has an internal resistance which can alter design calculations as well. Diodes and transistors are heavily temperature dependent devices which can be damaged easily if there is too much current flowing through them. There are many other limiting factors involving circuit components that can alter an expected output voltage. To have a reliable power source, these non-idealities need to be considered and designed around accordingly. To accommodate possible design flaw & non-ideal characteristics of circuit components to be used, as well as internal resistance supplied by the AC wall voltage source, and to allow for continuous operation, 50% current will be added to the maximum current that has been rated for the power supply. This can be seen in figure 3.3.7-D.


[✓ AC Input] DC Input

Vin$_{max}$ 125 V$_{rms}$ Vo$_1$ 12.0 V Vo$_2$ 5 V

Vin$_{min}$ 115 V$_{rms}$ Io$_1$ 0 A - 0.495 A Io$_2$ 0 A - 0.690 A

[✓] +ve

Power Supply Application: Adaptor

Figaure 3.3.7-D: Powersim Input Specifications

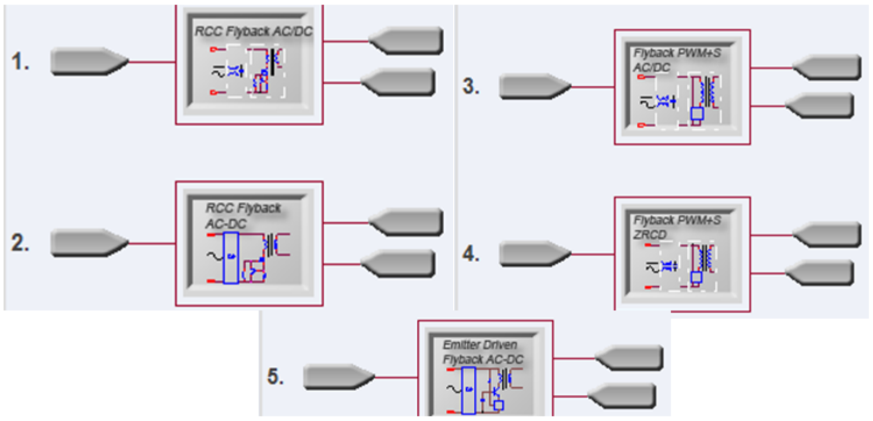
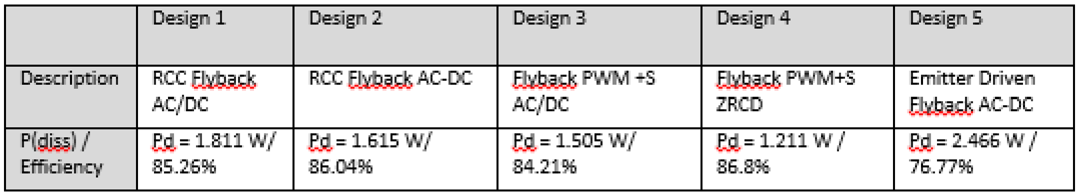


Figure 3.3.7-E: Suggested power supply topologies for comparison



	Design 1	Design 2	Design 3	Design 4	Design 5
Description	RCC Flyback AC/DC	RCC Flyback AC-DC	Flyback PWM +S AC/DC	Flyback PWM+S ZRCD	Emitter Driven Flyback AC-DC
P(diss) / Efficiency	Pd = 1.811 W/ 85.26%	Pd = 1.615 W/ 86.04%	Pd = 1.505 W/ 84.21%	Pd = 1.211 W / 86.8%	Pd = 2.466 W / 76.77%

Figure 3.3.7-F: Efficiency of designs

Figure 3.3.7-F shows the efficiency of the designs. Normally, when designing a power supply, consideration needs to be taken into account for the amount of space on the PCB board (footprint), cost, complexity, and efficiency. Since this is a rather bulky project, with many mechanical parts the issue of footprint does not really need to be taken into consideration. Typically, the footprint goes up with efficiency. Since the footprint is really not a cause for concern, power supply design will be aimed towards efficiency. Another consideration is complexity of Power supply design. Typically, the higher the efficiency the more complex the design will be. As demonstrated in table 3.3.8 – d, there is not much discrepancy in efficiency. A moderate efficiency will be selected to accommodate moderate complexity in design.

Additional Devices - Unforeseen devices may need to be added to the project at a later date. These devices may draw more current than the maximum current of the devices that are currently accounted for in the project. Table 3.3.8 – d will most likely need to be edited at a later time as things come up that were overlooked in the in the initial project research phase.

Devices Requiring Voltage Dividers - Any device that operates at a different voltage than the two outputs provided (5 and 12) Vdc will need to be accommodated using a simple voltage divider. Figure 3.3.7-G shows the concept of the sensors which will most likely need to be supplied by a different voltage than the two output voltages given.

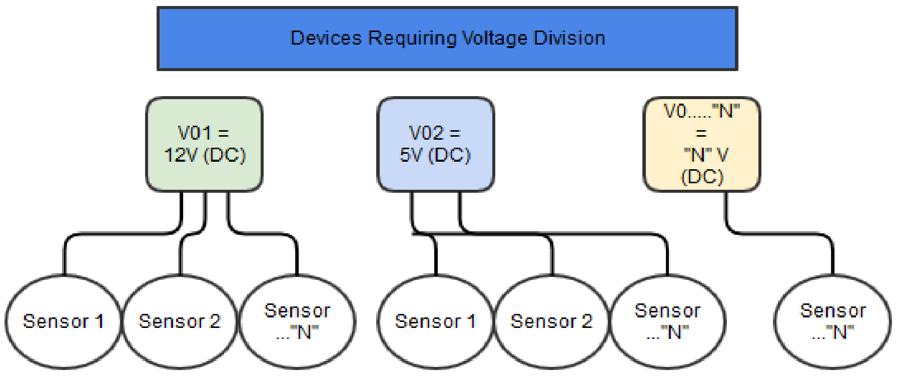


Figure 3.3.7-G: Theoretical chart of Voltage Dividers from the two output voltages

DC to DC Voltage Conversion (Non- Switching Mode power Supply) - A voltage divider is the simplest way to step down a voltage. The issue with voltage dividers is that they cannot be used to power devices that need a constant output voltage. This is because voltage dividers make use of components that consume power. Resistor values are never constant. Also, the internal resistance of the voltage source is often neglected. These two effects result in a discrepancy of output voltage that differs from the one calculated for design. The motors used in the project need to have a constant output voltage to drive the conveyor belts, lift arm, and rotating arm without interruption. The motors use very sensitive driver circuitry which requires a steady output voltage which cannot be provided using a linear DC to DC voltage conversion. Discrete Linear Regulators are another option for DC to DC (non – SMPS) voltage conversion. Linear regulators use intricate circuitry to combat the effects of internal resistance supplied by the load. The principle behind the linear regulator is that they can be used as a variable voltage divider (as opposed to a fixed voltage divider designed from basic circuit components). Efficiency is the main drawback of the Linear Regulator. Since linear regulators employ circuit components that create thermal energy in the form of heat dissipation, they have very low efficiency.

Regulators, Voltage References, Switched Mode Power Supplies) - Use of heat sinks may be helpful to counteract the effect of power dissipation from use of linear regulators (or other devices that dissipate a lot of power). Again, choice of heat sink is constrained to size which probably will not affect this project allowing more freedom when choosing the right heat sink to cool the circuitry. Generally speaking, the surface area of the heat sink corresponds to how well it will perform, but this is not necessarily always true as there are many configurations of heat sink design.

Summary - Upon inspection of the basic circuit diagrams for the Flyback, Forward, 2-Switch Forward, Active Clamp Forward, 2- Switch Forward, Active Clamp Forward, Half-Bridge, Push Pull, Full Bridge, and Phase Shift ZVT, it is apparent that the Flyback converter is the lowest in complexity to supply power needed for the project. A Flyback converter was chosen because it is isolated using a transformer. The turns ratio of the can be adjusted to vary the output voltage. Multiple output loads are attainable with the flyback and it is one of the simplest topologies to design. It uses less inductors than other topologies so the footprint is smaller and less complex. It is also cost effective in comparison.

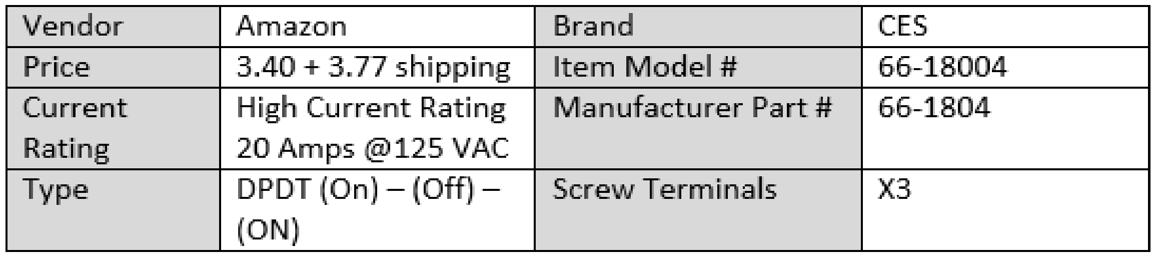
It was decided to use topology #5 Emitter Driven Flyback AC-DC. Upon inspection of the circuits, the Flyback AC-DC is the lowest in complexity to design at the cost of the lowest efficiency of the other topologies considered at 76.77%.

3.3.8 Kill Switch

Overview - A Kill switch will need to be included in the project for the user to stop the sorter when no longer in use. A kill switch is nothing more than a selection between an open circuit and a short circuit. The switch will need to be a maintained switch. The difference between a momentary switch and a maintained switch is that a momentary switch only works while it is being pressed down while a maintained switch maintains the state that it is switched to until it is switched to another state.

Poles/Throws - The number of *poles* determines how many circuits the kill switch can control. The number of *throws* determines the number of terminals each circuit can connect to (this can be helpful if wires cannot reach one of the terminals). Single Pole Double Throw (SPDT) switches can easily be turned into SPST switches by leaving a switch throw terminal unconnected. Since things may come up in this heavily mechanical project that are unplanned, it is best to choose a kill switch with the maximum number of throws and poles. This of course would be the ideal decision but power consumption ratings also need to be considered. This project will need to be able to turn off a total of 4 electromechanical motors: Lift arm Servo motor, DC belt motor #1, DC Belt motor #2, and the Rotating Arm Stepper motor. A double pole double throw (DPDT) switch has the ability to turn off two circuits at the same time. It was decided to use DPDT switch to stop the sorter.

AC or DC Connection Specifications - The kill switch can either be connected to the circuitry of the power supply itself with an AC voltage and kill the power to everything or it can be connected to the DC voltage supplied circuits. The basic specifications for a DPDT kill switch are shown in Figure 3.3.8-A.



Vendor	Amazon	Brand	CES
Price	3.40 + 3.77 shipping	Item Model #	66-18004
Current Rating	High Current Rating 20 Amps @125 VAC	Manufacturer Part #	66-1804
Type	DPDT (On) – (Off) – (ON)	Screw Terminals	X3

Figure 3.3.8-A Kill Switch AC DPDT

Vendor	DigiKey	Brand	CTS Electro components
Price	$1.36	DigiKey Part #	CT204213ST - ND
Current Rating	50mA @ 24V DC	Manufacturer Part #	204-213ST
Type	DIP DPST 3 Position	Mount	Surface

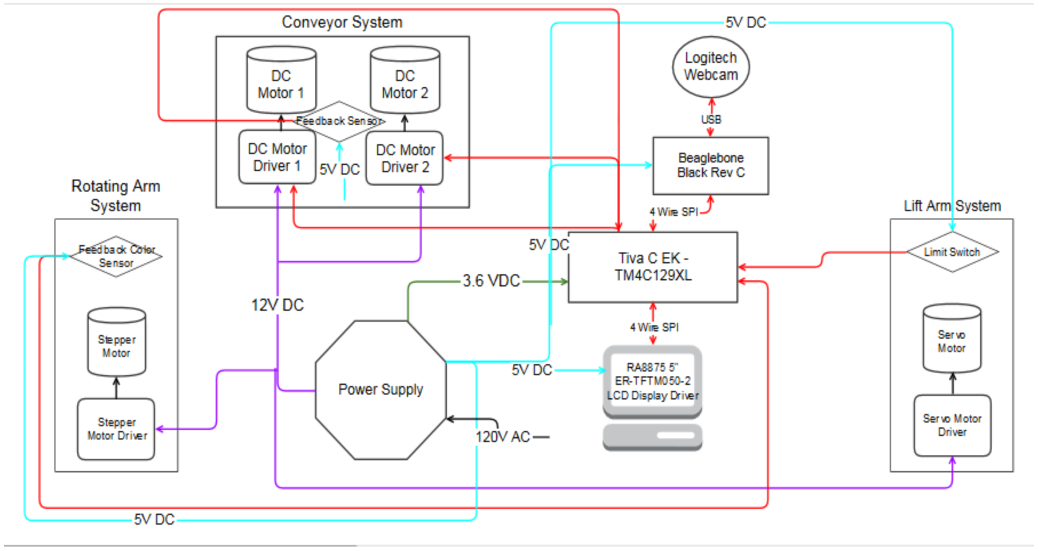
Figure 3.3.8-B Kill Switch DC DIP DPST

Summary - The kill switch will have to be designed from the AC Mains power. A DPDT switch was found on digikey that operates at 0.1A @ 5DC, which would be appropriate for this project but it was found on another website that it is now obsolete because it is not RoHS compliant. Otherwise the 24V DC killswitch would have to be configured somehow for the project or designed completely from scratch.

4.0 Project Design Details

Section 4 will be dealing with the overall design plan of the Lego sorter. These are the plans that will be used to build the foundation of the project and will help to be the guide for the entire construction process.

4.1 Block Diagram

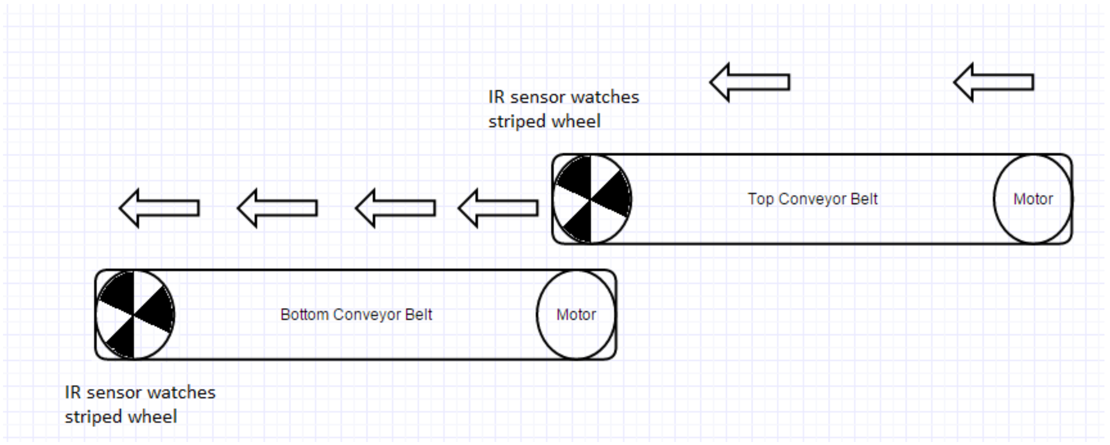


4.2 Hardware Design

Section 4.2 will outline the hardware design details for each of the separate subsystems. By breaking down the project into various subsystems, this makes it simple to test everything individually and then when each part is working on its own, then they will be brought together at the end of the construction process.

4.2.1 Conveyer Belt System Design

The conveyer system is comprised of two belts driven at different speeds. The bottom belt moves at a quicker speed than the top belt to allow for separation of clumped LEGO pieces. Each belt is animated by its own 12 V motor. Infrared (IR) reflectance sensors are pointed at striped wheels on the conveyor belt shafts. If a shaft stops moving when it should be moving, there is likely a motor stall. The main microcontroller then stops the program and lets the user know there is a jam or some other issue with the belt motors. Figure 4.2.1-A below shows a physical layout of the conveyor belt system.


IR sensor watches striped wheel
Top Conveyor Belt
Motor
Bottom Conveyor Belt
Motor
IR sensor watches striped wheel
Figure 4.2.1-A: Conveyor belt system with belts, belt motors, and IR reflectance sensors.

The IR reflectance sensors will pulse high when a black stripe is in view and pulse low when a white stripe is in view. The signal generated by this pulsing will let the main microcontroller know that the belts are indeed moving and have not jammed.

The motors connect to driver circuitry, which is controlled by the main microcontroller. The IR reflectance sensors connect directly to the main microcontroller. A general block diagram of the conveyor belt system electronics is shown below in Figure 4.2.1-B.

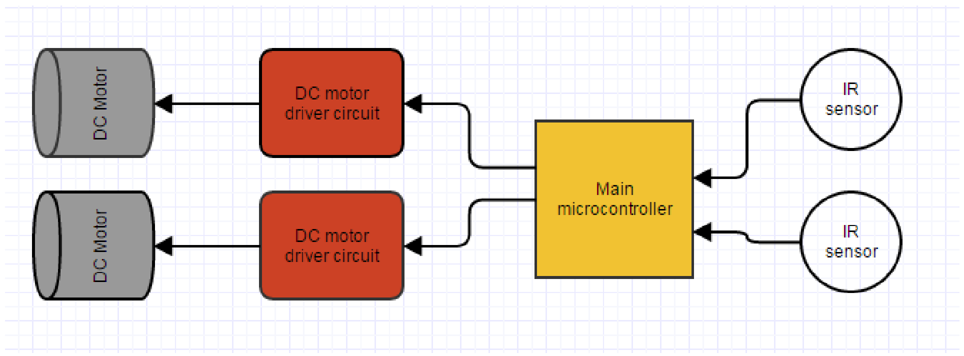


Figure 4.2.1-B Conveyer belt system circuitry block diagram

Motor Driver Circuitry Design - Motor circuitry design will be identical for both of the conveyor belt motors. The conveyer belts need only move in one direction, so advanced circuitry like an H-bridge for bi-directional movement is not needed. A simple transistor switch circuit will allow for a small signal from the main microcontroller to control the motors. A schematic of the DC motor circuitry is shown below in Figure 4.2.1-c.

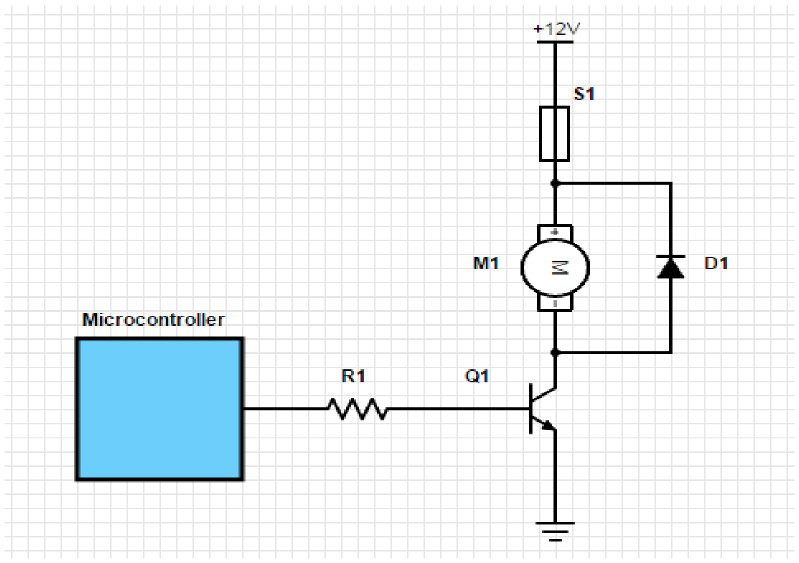


Figure 4.2.1-C DC motor driver circuit

Besides a transistor, Figure 4.2.1-C above shows resistor R1, diode D1, transistor Q1, motor M1, and fuse S1. The transistor acts as a switch when given a signal from the main microcontroller. The signal will be pulse width modulated to control the motor speeds.

When the transistor base receives a 3.3V high pulse from the microcontroller signal, current flows through the motor to turn the conveyor belts. Diode D1 acts as a flyback diode to suppress voltage spikes from the motor when it is turned on and off. Although the IR sensors will allow for software control to turn off the motors if a stall is detected, it will not be quick enough to prevent damage to the circuit components. Fuse S1 is in place to prevent overdraw of current due to a stall. The fuse will be automatically resettable so that the user does not have to intervene. Resistor R1 biases the signal from the microcontroller so that the transistor is in complete saturation when the signal is pulsed high. Assuming the forward voltage of the transistor is ~ 0.7V, the resistor at the transistor base should have a value of 2.6KΩ. To insure complete saturation at all times, a 1K value will be used in place of the 2.6K. A simulation of the DC driver circuit is shown below in Figure 4.2.1-D.

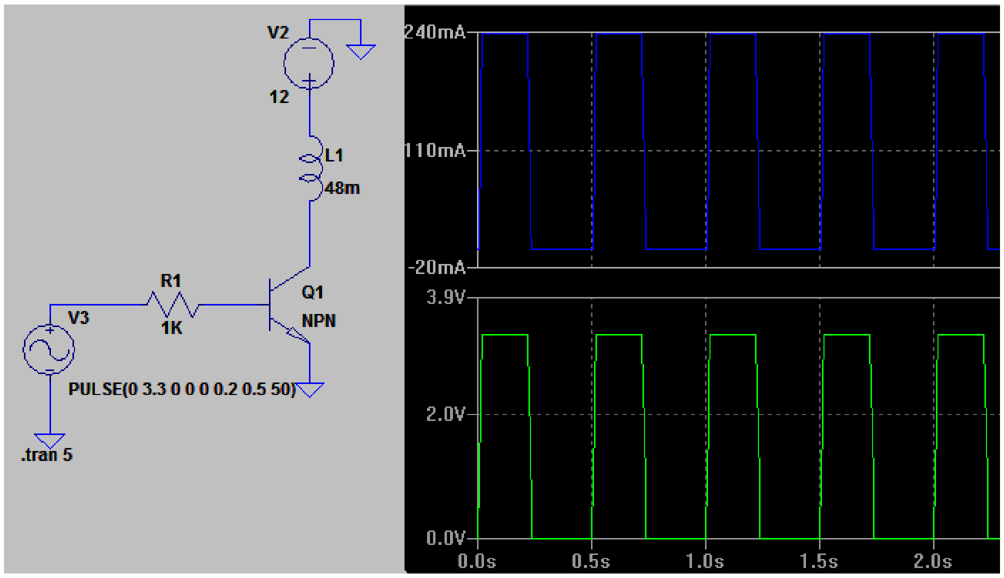
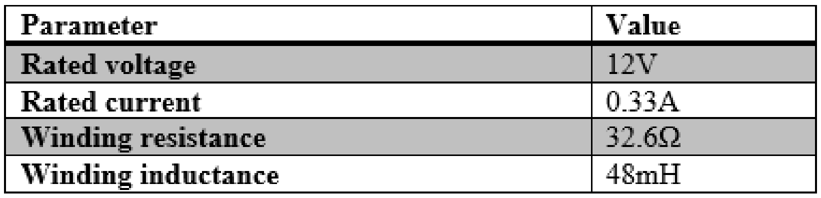


Figure 4.2.1-D Simulation of DC driver circuit

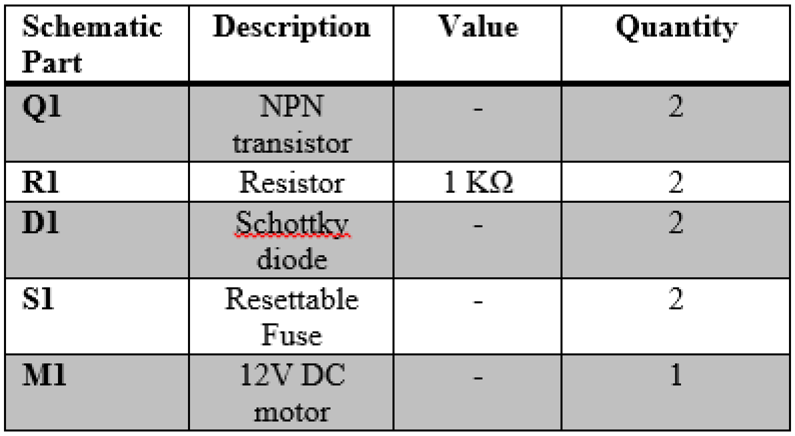
A pulse period of 0.5 seconds with 40% duty cycle was used in the simulation to mimic a PWM signal from the microcontroller. The simulation shows a current of 240mA through the motor when a high pulse of 3.3V is sent to the transistor base. The inductor model parameters were changed to match the selected motor specifications, which are shown below in Figure 4.2.1-E below.



Parameter	Value
Rated voltage	12V
Rated current	0.33A
Winding resistance	32.6Ω
Winding inductance	48mH

Figure 4.2.1-E Electrical characteristics of the motor

Looking again at Figure 4.2.1-D, the 12V source comes from the power supply. A parts list of the schematic in Figure 6367268 is shown below in Figure 4.2.1-F.



Schematic Part	Description	Value	Quantity
Q1	NPN transistor	-	2
R1	Resistor	1 KΩ	2
D1	Schottky diode	-	2
S1	Resettable Fuse	-	2
M1	12V DC motor	-	1

Figure 4.2.1-F: Conveyor belt component descriptions and values

The input signals that control the DC motor driver come from the main microcontroller. The microcontroller will have a pulse width modulation (PWM) peripheral on-board that can send a modulated signal to the motor driver. The higher the duty cycle of the modulated signal, the faster the motors will turn.

Infrared reflectance sensor - As shown in the physical layout diagram in Figure 4.2.1-A, the conveyor belts will each have an infrared reflectance sensor and striped wheel. Depending on the conveyor belt speed, a signal is generated by the IR reflectance sensor because of the differing colored stripes (black and white). The IR reflectance sensor chosen is the Line Sensor Breakout QRE1113 shown below in Figure 4.2.1-G.

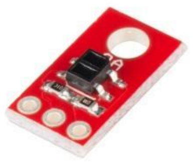


Figure 4.2.1-G Line Sensor Breakout - QRE1113 Infrared reflectance sensor (permission pending)

The sensor has an analog output and can be operated at 5V or 3.3V. The IR diode in the sensor will be powered by the main microcontroller as it operates at 3.3 V as well. The sensor is already mounted along with a resistor to limit diode current and output resistor. The connector holes from left to right are ground, output, and VCC. The output contact will be connected to a pin on the microcontroller that has analog to digital conversion

(ADC) capabilities. The output signal from the sensor will look similar to the waveform in Figure 4.2.1-H below.

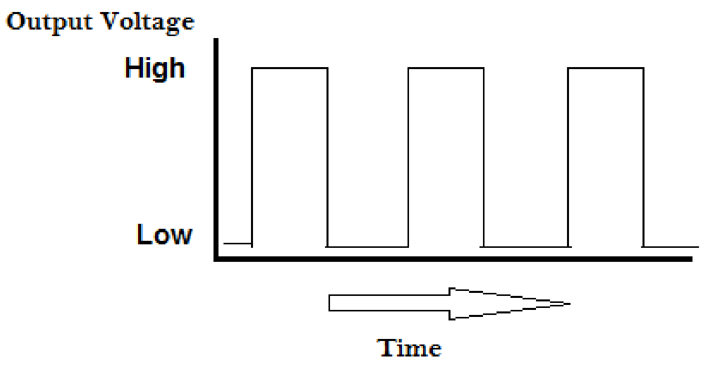


Figure 4.2.1-H Idealized output of IR sensor when conveyor belts are turning

Figure 54 above shows the predicted ideal output signal of the IR reflectance sensor when the conveyor belts are moving. This sensor will output "high" voltage (3.3 V) when the light reflected back is minimal. Conversely, the sensor output will be "low" (0 V) when light reflected back is high. The pulses are created from the black and white striped wheels on the conveyor belt shafts turning in front of the sensor. High voltage output corresponds to low light reflection from the black sections of the striped wheel. Low voltage output corresponds to high light reflection from the white sections of the striped wheel. The program code will be adjusted so the two levels are within two separate ranges of ADC values from 0 to 1024. The final electrical schematic of the sensor is shown below in Figure 4.2.1-I.

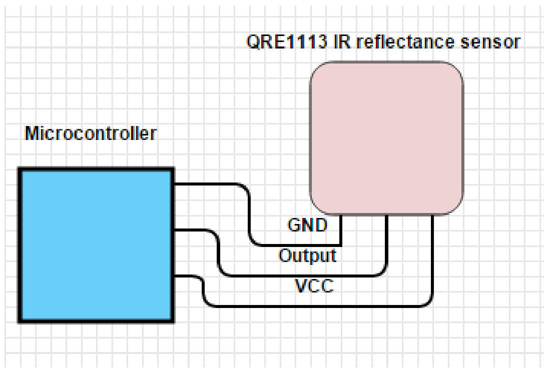
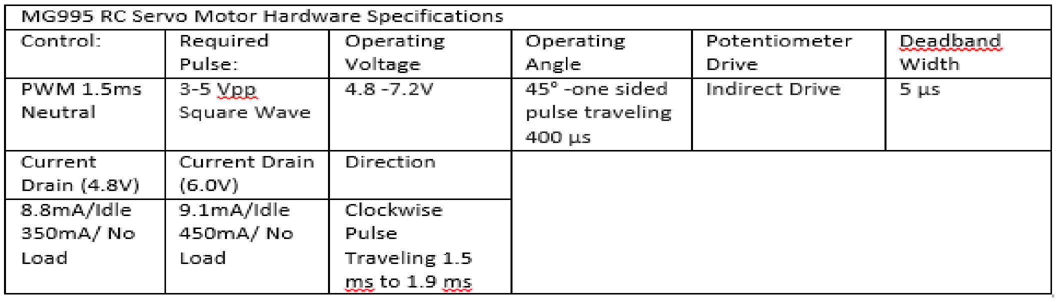


Figure 4.2.1-I: Electrical connections of QRE1113 IR reflectance sensor

4.2.2 Lift Arm System

Positional Rotation Servo – Servo motors consist of three wires: +Vcc, ground, and a control wire that utilizes pulse width modulation (PWM).The pulse varies from low (0V) to high which is the battery voltage (4.75) for the Tiva C. The period of the pulse will need to be set to be 20ms. The angle of the pulse is determined by the minimum pulse signal and the maximum pulse signal. The motion of the lift arm is unidirectional moving up and down along the Y-axis. The motor will need to drive the lift arm to move upwards until it triggers the limit switch at the appropriate set point and then the lift arm will need to be designed to proceed in the reverse direction.



MG995 RC Servo Motor Hardware Specifications					
Control:	Required Pulse:	Operating Voltage	Operating Angle	Potentiometer Drive	Deadband Width
PWM 1.5ms Neutral	3-5 Vpp Square Wave	4.8 -7.2V	45° -one sided pulse traveling 400 µs	Indirect Drive	5 µs
Current Drain (4.8V)	Current Drain (6.0V)	Direction			
8.8mA/Idle 350mA/ No Load	9.1mA/Idle 450mA/ No Load	Clockwise Pulse Traveling 1.5 ms to 1.9 ms			

Figure 4.2.2-A MG995 Servo motor Specifications

For most servo motors the accepted convention is a minimum pulse width at 1ms a neutral position at 1.5ms and a maximum pulse width at 2ms as shown in figure 4.2.2-a. The actual specifications of the MG995 (shown in table 4.2.20-a) are the same for the minimum pulse width operating at 1.5ms, the neutral position at 1.5ms. The maximum pulse however, operates at 1.9ms as opposed to 2ms.

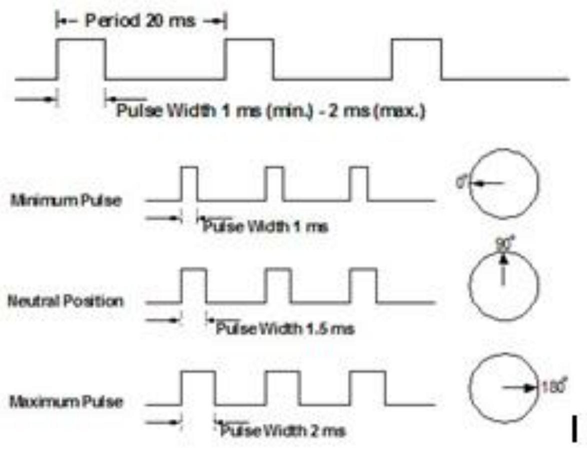


Figure 4.2.2-B Servo motor position control using PWM (Permission Pending)

Continuous Rotation Servo - The continuous rotation servo operates much like the positional rotation servo except there is no limit on its' range of motion. It either rotates CW, CCW, or has no rotation. Instead of PWM correlating to fixed positions at pulse width integers of 1ms, 1.5ms, or 2ms as shown in figure 4.2.2.-a (above) the continuous rotation servo uses the same three pulse width integers but includes an interval which correlates to the speed of the CW/CCW direction traveled. Table 4.2.2-(b) demonstrates the way a continuous rotation servo works.

Input (ms)	Rotation Speed (%)	Direction of Rotation
1.0	100	Clockwise
1.1	80	Clockwise
1.2	60	Clockwise
1.3	40	Clockwise
1.4	20	Clockwise
1.5	0	N/A
1.6	20	Counter-Clockwise
1.7	40	Counter-Clockwise
1.8	60	Counter-Clockwise
1.9	80	Counter-Clockwise
2.0	100	Counter-Clockwise

Figure 4.2.2-C Motion and Speed Control of Continuous Rotation Servos

The Parallax motor will need to be wired as shown in figure 4.2.2 –B connecting the red to +Vcc the black to ground and the white wire to the GPIO PWM pin of the Tiva C. To setup the Parallax servo motor the Tiva C PWM pin will need to be programmed to 1.5ms, which is considered the neutral position from the positional control servo motor standpoint at 20ms intervals. Once the pulses have been generated the set point can be set by adjusting the potentiometer. The set point is the neutral position. This position is set when the motor is receiving a 1.5ms pulse from the microcontroller. The Parallax servo comes with a screwdriver to do this. The potentiometer will gently need to be adjusted to the left and right very slowly until the servo stops turning. If it is done too quickly and the set point is passed the servo will change direction.

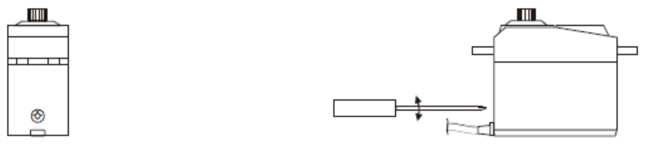


Figure 4.2.2-D Setting the set point of the potentiometer of the Parallax Servo (Permission Pending)

The Parallax servo motor has three wires as shown in figure 4.2.2-E below. The I/O pin will be connected to the Tiva C which will provide the PWM signal to control the position of the motor.

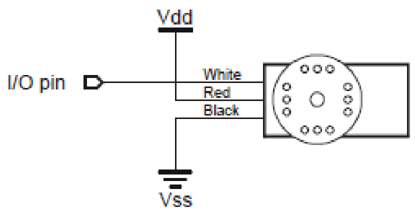
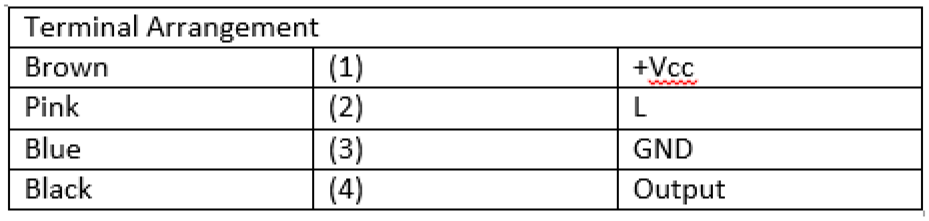


Figure 4.2.2-E Connections of the Parallax servo motor (Permission Pending)

Limit Sensor - The EE-SX670- WR sensor has 4 wires that need to be connected as shown in figure 4.2.2-F below.



Terminal Arrangement		
Brown	(1)	+Vcc
Pink	(2)	L
Blue	(3)	GND
Black	(4)	Output

Figure 4.2.2-F Connections for EE-670-WR Sensor

A 5 to 24V supply will need to be connected to the sensor as shown in the diagram in figure 4.2.2-G with the load connected between the output and +Vcc.

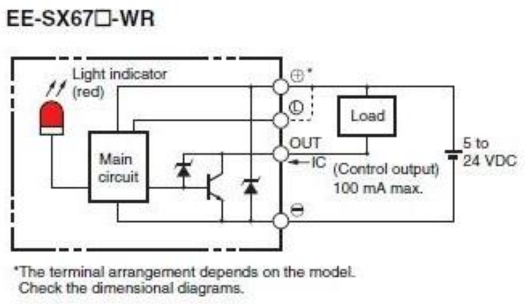
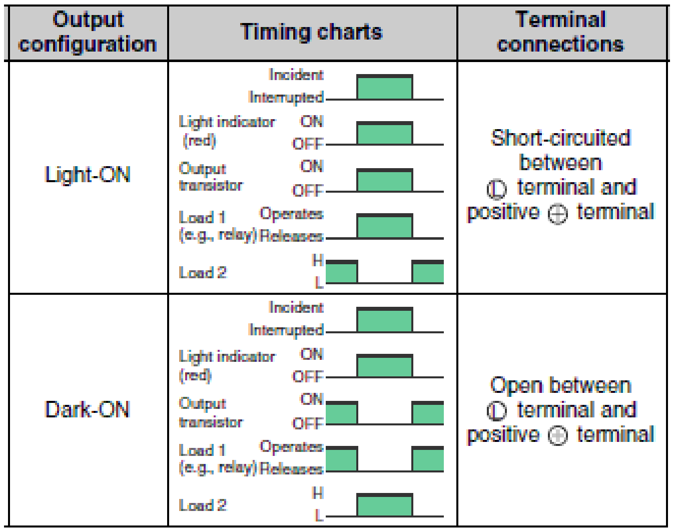

EE-SX67□-WR
Light indicator (red)
Main circuit
Load
OUT
IC
(Control output) 100 mA max.
5 to 24 VDC
*The terminal arrangement depends on the model. Check the dimensional diagrams.

Figure 4.2.2-G shows the physical connections of the EE-SX670-WR sensor (Permission pending)

The timing diagrams are governed by the pink wire (2) “L”. A short circuit between L and +Vcc will configure the output to have the light on and open circuit between L and +Vcc produces dark –on. This can be seen in figure 4.2.2-H. The sensor will be configured such that the light indicator is on. When a LEGO gear rolls in between the slot of the sensor the

IR LED will be blocked. The light that reflects from the object will cause the logic voltage level to go low. This message will be picked up by the microcontroller. The sensor acts as a limit switch for the LEGO gear that rolls in between the slot. The microcontroller will need to be programmed to switch the direction of the motor by adjusting the duty cycle of the pulse supplied to it.



Output configuration	Timing charts	Terminal connections
Light-ON	Incident / Interrupted; Light indicator (red) ON / OFF; Output transistor ON / OFF; Load 1 (e.g., relay) Operates / Releases; Load 2 H / L	Short-circuited between Ⓛ terminal and positive ⊕ terminal
Dark-ON	Incident / Interrupted; Light indicator (red) ON / OFF; Output transistor ON / OFF; Load 1 (e.g., relay) Operates / Releases; Load 2 H / L	Open between Ⓛ terminal and positive ⊕ terminal

Figure 4.2.2-H EE-SX670 timing diagram (Permission Pending)

Summary - In summary, the lift arm will have two hardware components that require design, the Parallax S148 and the EX –SX67- WR. The continuous rotation servo will move the motor up and down to transport LEGO parts from the dump bucket to the belt and the slot sensor will act as a limit switch feedback sensor for the lift arm.

4.2.3 Rotational Arm Design

The rotational arm system will use a stepper motor and color sensor to “point” towards the correct LEGO bin. With each bin a different color, the sensor knows exactly where it is pointing at all times and will have little chance of dropping the LEGO piece in a wrong bin. The rotating arm will utilize a bipolar stepper motor driven by a duel H-bridge chip. The detailed block circuitry diagram of the rotational arm system is shown below in Figure 4.2.3-A.

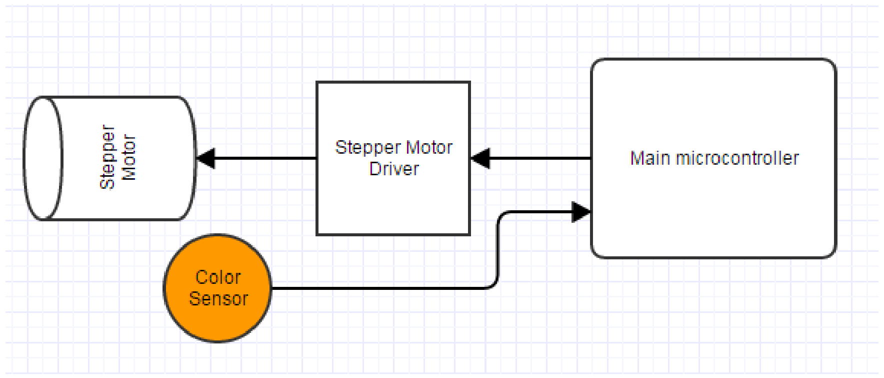


Figure 4.2.3-A Rotational arm system block diagram

The block diagram shows the data path of the various components. When the LEGO piece has been identified by the image processing system, the main controller feeds signals to the stepper motor driver which turns the arm to the correct position. The color sensor provides feedback to the microcontroller to ensure correct positioning. Stepper motors are more complex than the standard DC motor as there multiple poles to energize to step the motor into each position. The motor to be used is a 12V rated bipolar stepper motor with 1.8° step angles.

Stepper motor driver circuitry - A common method to drive stepper motors is using an H-bridge circuit. For this project, two H-bridge circuits are needed to control the motor windings. The L293D is a duel packaged H-bridge IC. This IC will be used to save space on the PCB as well as minimize development time. The L293D duel H-bridge IC pin-out and logic diagram are shown below in Figure 4.2.3-B.

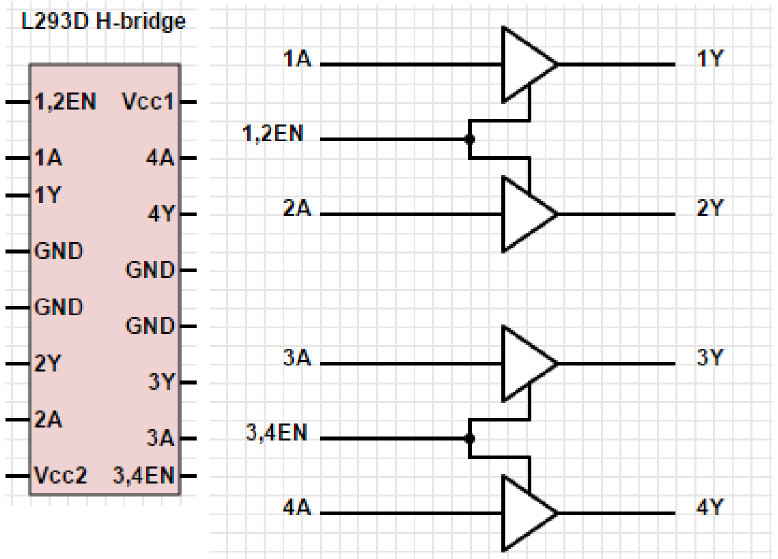
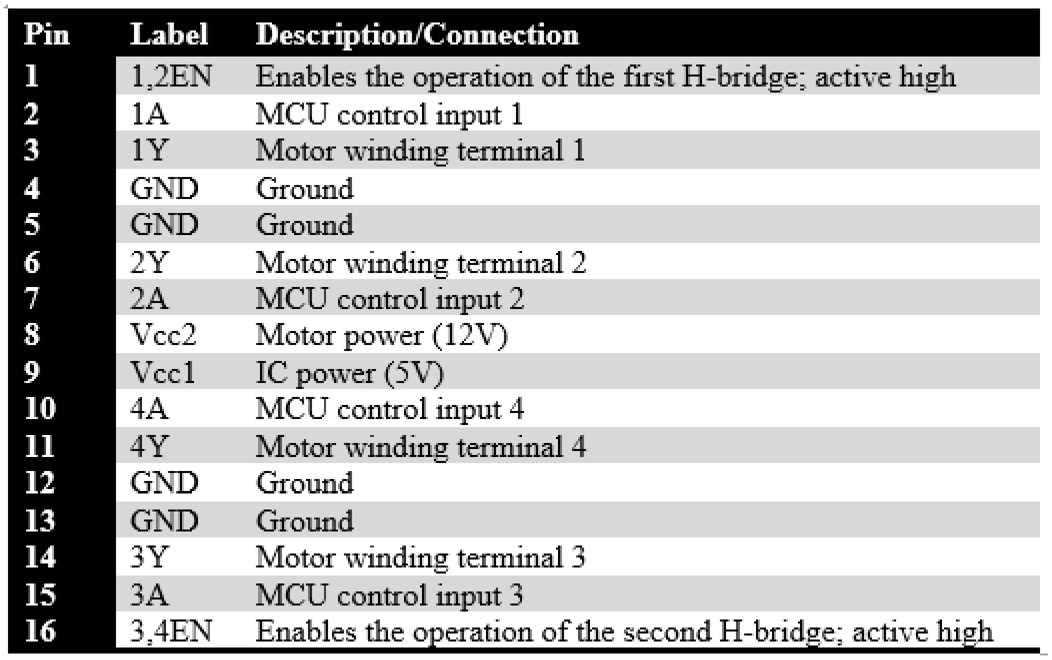


Figure 4.2.3-B L293D H-bridge IC pin-out and logic diagram

The duel H-bridge IC acts as a buffer in between the low voltage logic signals from a microcontroller and high voltage high, high current power supply for the motor. Figure 4.2.3-C below describes each pin on the IC and its function.



Pin	Label	Description/Connection
1	1,2EN	Enables the operation of the first H-bridge; active high
2	1A	MCU control input 1
3	1Y	Motor winding terminal 1
4	GND	Ground
5	GND	Ground
6	2Y	Motor winding terminal 2
7	2A	MCU control input 2
8	Vcc2	Motor power (12V)
9	Vcc1	IC power (5V)
10	4A	MCU control input 4
11	4Y	Motor winding terminal 4
12	GND	Ground
13	GND	Ground
14	3Y	Motor winding terminal 3
15	3A	MCU control input 3
16	3,4EN	Enables the operation of the second H-bridge; active high

Figure 4.2.3-C L293D pin descriptions

Bipolar stepper motors operate by energizing coils in a specific pattern to allow the motor shaft to turn. Each coil in the motor requires a connection on either end to allow for bi-directional current flow. For example, one coil will connect to the 1Y on one of its ends and 2Y on the other end. When positive voltage is applied to one end, the other end must have negative voltage applied, or else it would be a short. To ensure no shorts accidentally occur from the MCU code, inverter circuits will be applied to each pair of MCU inputs (1A, 2A and 3A, 4A). Figure 4.2.3-D below shows an inverter circuit.

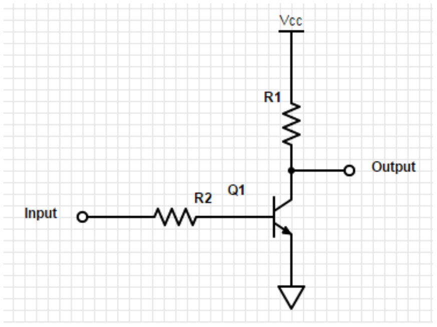


Figure 4.2.3-D Inverter circuit

The inverter takes the logic signal from the microcontroller and produces the opposite signal value at the output. Not only do the inverters ensure opposing voltage inputs for the motor winding terminals, but they also relieve the MCU of two additional pins. The complete schematic of the inverters attached to the L2934D IC is shown below in Figure 4.2.3-E.

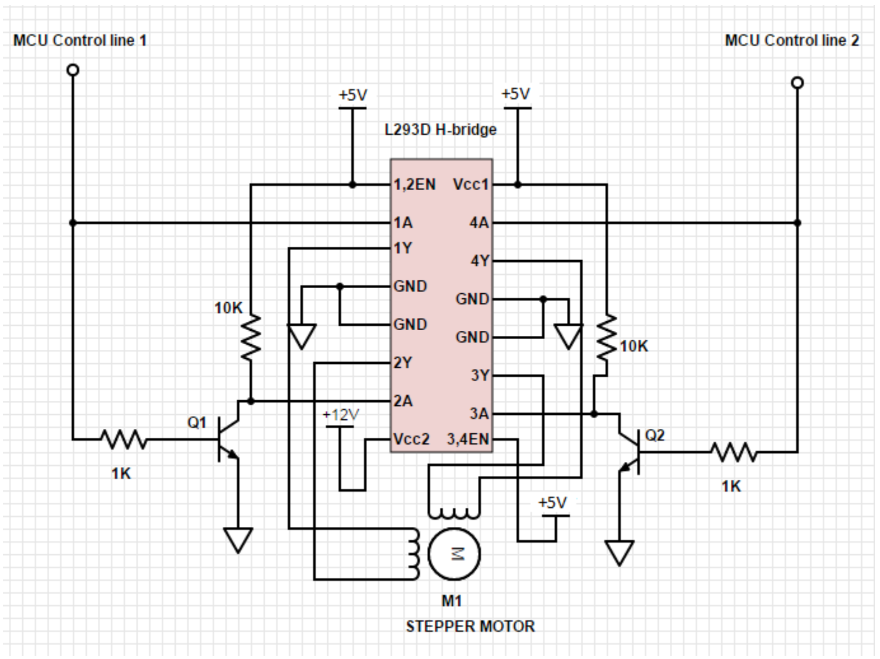


Figure 4.2.3-E Circuitry for rotational arm stepper motor

Color Sensor - The sorter arm will utilize a color sensor to ensure the arm is pointing towards the proper LEGO bin. Each bin will have a color scheme on it for the sensor to identify. The sensor chosen for the project is the Adafruit TCS34725. A pin out diagram and table of pin connections is shown below in Figure 4.2.3-F and Figure 4.2.3-G respectively.

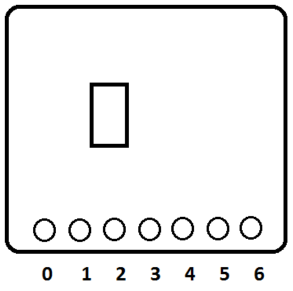


Figure 4.2.3-F Pin diagram of TCS34725 color sensor

Pin #	Pin label	Pin description
0	LED	LED control pin – active high
1	INT	Interrupt – active low (output)
2	SDA	I2C data terminal
3	SCL	I2C clock terminal
4	3V3	3.3 V Power
5	GND	Ground
6	VIN	5 V Power

Figure 4.2.3-G

The 3.3V power pin will be used instead of the 5V pin since the microcontroller being used operates at 3.3V. The sensor wiring is relatively straight forward and will be connected directly to the microcontroller through the various pins. A diagram of the connections is shown below in Figure 4.2.3-H.

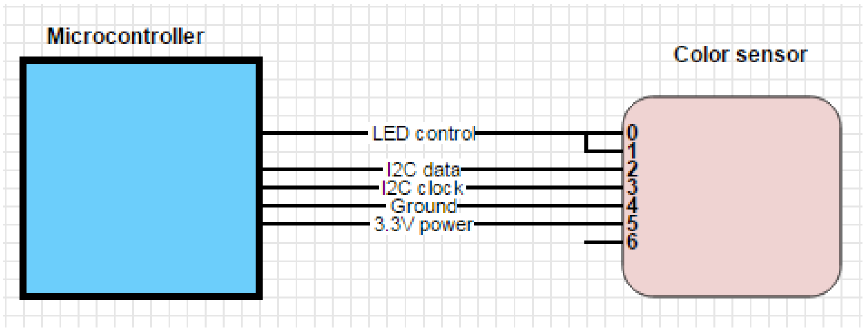


Figure 4.2.3-H

4.2.4 Image Processing Design

The Beaglebone Black Rev C was decided upon to complete the image processing task of the LEGO pieces. An image of the Beaglebone Black is shown below in Figure 4.2.4-A.

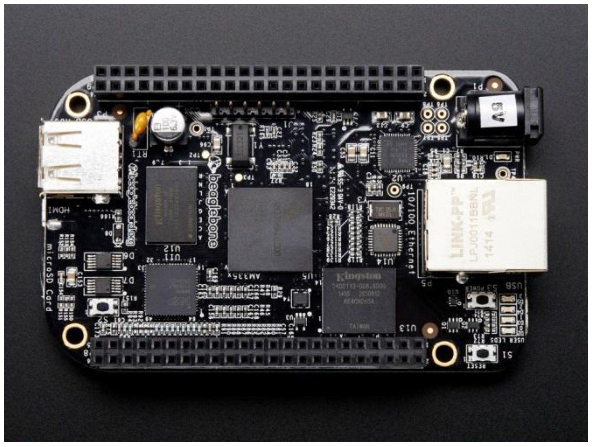


Figure 4.2.4-A Beaglebone Black Rev C

The electrical connections and design are minimal for the Beaglebone. It will connect to the power supply’s 5V output, interface with the main microcontroller through SPI, and have a USB based camera connected to a USB port. The Beaglebone has two USB ports already built on the board as well as available pins for SPI interfacing. A wire diagram of all the connections is shown below in Figure 4.2.4-B.

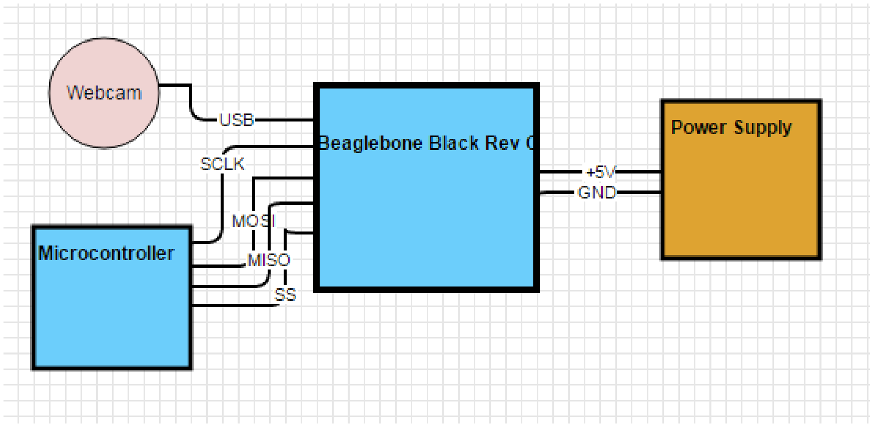


Figure 4.2.4-B Wiring diagram for the Beaglebone Black

The camera selected to feed images into the Beaglebone is the Logitech C110. The camera was selected because of the good online support for Logitech products. The C110 is a low resolution VGA camera as high definition is not needed for this project’s purposes. The camera is shown below in Figure 4.2.4-C.

Figure 4.2.4-C Logitech C110 webcam

PCB Software - The PCB vendor chosen is 4PCB. 4PBC offers free and intuitive PCB design software called PCB Artist. The software has extensive libraries for relatively quick designing. The software allows the designer to first make a schematic layout of the circuit with traditional circuit symbols and wiring. The software can automatically create a rough PCB layout from the schematic, which then requires the designer to place components and reroute wiring where needed. The software is free to download and use and directly communicates with the vendor, 4PCB. PCB Artist will be used for the PCB design layout of the project.

Rotating Arm Testing - The stepper motor requires a specific sequence of signals to operate as intended. Testing will begin once it is completely hooked up as designed. After hooking up the stepper motor and driver circuit to the Tiva C microcontroller and power supply as shown in Figure 4.2.4-B, the code will be written. Testing will make sure the timing in the code allows for smooth operation of the stepper motor. Care will be taken to make sure delays are put in the code to accommodate for the slower electro-mechanical processes that occur in the motor. Being that the motor has 1.8° steps, it should not be a problem to get the rotating arm slide to line up with each LEGO bin. Testing will make sure it does any way. The color sensor on the rotating arm will also be tested. It will be connected via I2C with the microcontroller before testing. The sensor has a register and some integrated features that will be read into to properly test its functionality.

4.2.5 Main microcontroller Design

The main microcontroller is the central command of this project. It is in charge of taking in sensor and image processing data, and controlling the conveyer belt, lift arm, and rotational arm accordingly. It was decided the microcontroller chip to use is the TM4C1294NCPDT. To quick-start the development and prototyping of the LEGO sorter, the use of a development board was decided to be advantageous. The Tiva C Series (Stellaris) TM4C1294 LaunchPad from Texas Instruments contains the TM4C1294NCPDT at its core as well as many relevant peripherals that will get the project started quickly. The Tiva C LaunchPad and the TM4C1294NCPDT's functional block diagram are shown below in Figure 4.2.5-A and Figure 4.2.5-B respectively.

Figure 4.2.5-A Tiva C TM4C1294 LaunchPad from TI (permission pending)

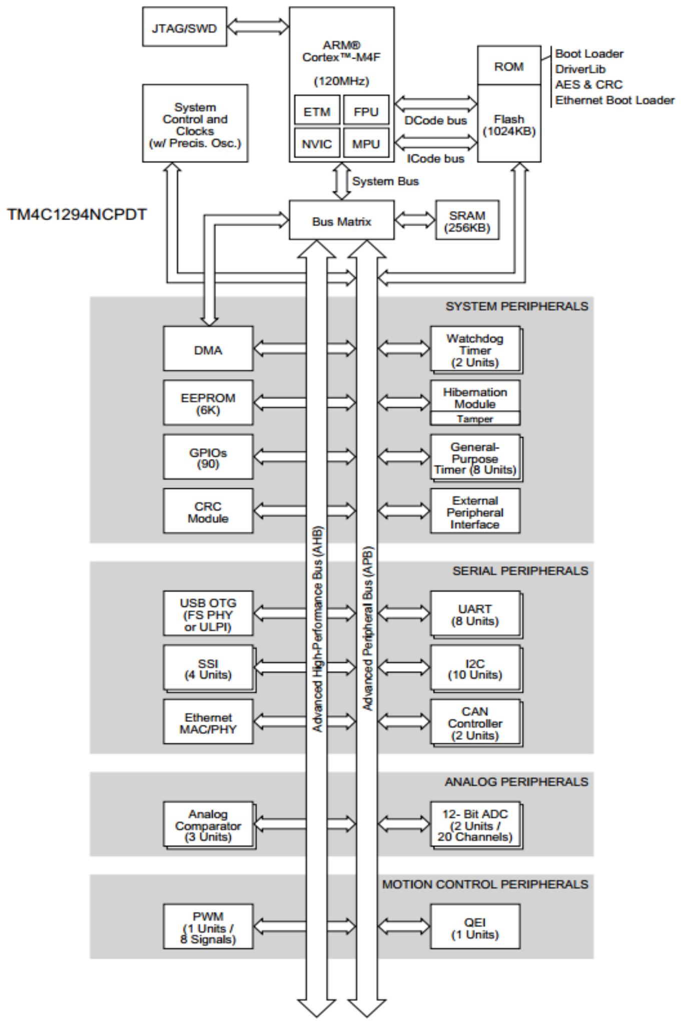


Figure 4.2.5-B Block diagram of the TM4C1294's feature (permission pending)

The block diagram of the TM4C1294 board shown above in Figure 4.2.5-B shows all the peripherals and technologies built onto the development board. The project will demand use of the USB, I2C, and UART serial peripherals. The analog and PWM systems will also be used for sensor inputs and motor control respectively.

An important design aspect of integrating a microcontroller into the project is pin selection. Although there are 90 pins available for GPIO requirements, the project demands use of some of the integrated peripherals and special functionality attached to certain pins. Namely, the LEGO sorter will need to utilize the microcontroller's interrupts, pulse width modulation (PWM), and SPI interfacing pins to complete the various tasks successfully. Figure 4.2.5-C below shows the TM4C1294 board's physical pin layout as well as each pin's available functionalities.

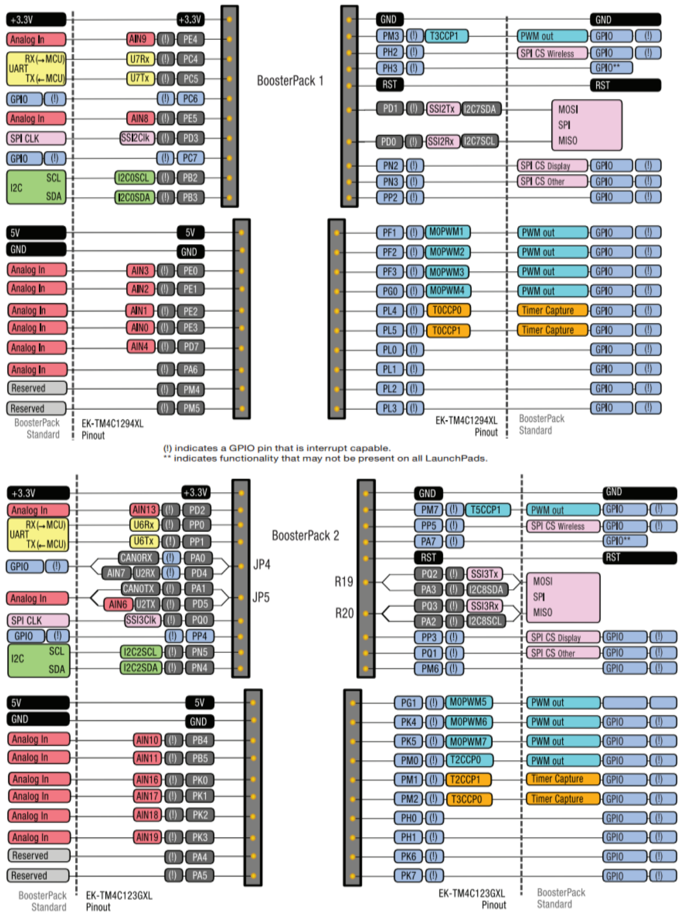
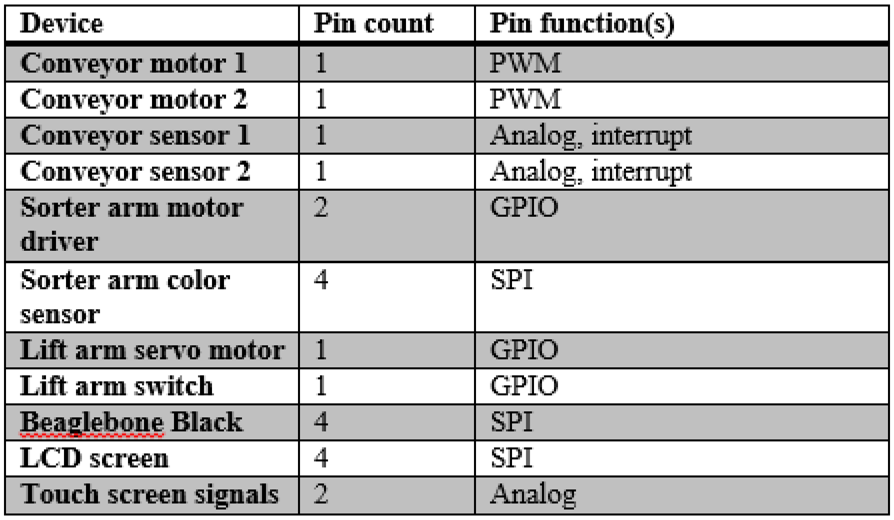

BoosterPack 1
BoosterPack 2
EK-TM4C1294XL Pinout
EK-TM4C123GXL Pinout
BoosterPack Standard
(!) indicates a GPIO pin that is interrupt capable.
** indicates functionality that may not be present on all LaunchPads.
JP4
JP5
R19
R20

Figure 4.2.5-C TM4C1294 pin layout and functionality (permission pending)

The pin layout in Figure 4.2.5-C above is for the LaunchPad's Booster pins, which will be used during development and prototyping. The LEGO sorter devices and their pin requirements of the TM4C1294 MCU are shown below in Figure 4.2.5-D.



Device	Pin count	Pin function(s)
Conveyor motor 1	1	PWM
Conveyor motor 2	1	PWM
Conveyor sensor 1	1	Analog, interrupt
Conveyor sensor 2	1	Analog, interrupt
Sorter arm motor driver	2	GPIO
Sorter arm color sensor	4	SPI
Lift arm servo motor	1	GPIO
Lift arm switch	1	GPIO
Beaglebone Black	4	SPI
LCD screen	4	SPI
Touch screen signals	2	Analog

Figure 4.2.5-D Devices and their pin requirements of the TM4C1294 MCU

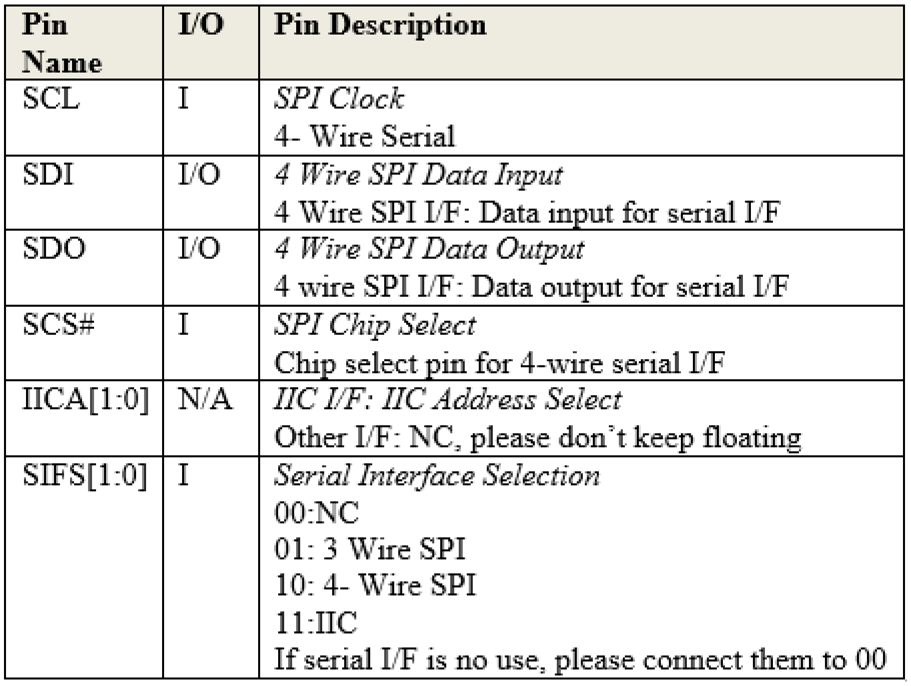
In total, 22 data pins are required to control all of the LEGO sorter devices. Interrupt pins will be used for the conveyor belt sensors to stop the belt motors in the event of a jam or motor stall. This way the code does not have to sequentially check the sensors every few lines of code to make sure the belts are still moving. The program will simply enter the interrupt routine and stop the belt motors. The PWM pins used on the conveyor motors allow for speed control. The analog pins convert an analog input into a digital signal value that can be used in the MCU code. The conveyor sensors will need to use analog pins on the MCU to get a reading. The color sensor, BeagleBone Black, and LCD screen will use SPI to interface with the MCU.

4.2.6 LCD Display

The pinouts in this section come from the RA8875 controller for the TFT LCD display specifications PDF. Some sections of specifications have been passed over. These sections specify a default setting for all pinouts and therefore will not be discussed and will be left to their default settings.

SPI MCU Interface to Tiva C - Since there are three devices that are to be interfaced to each other using SPI, a 4 wire connection is required. The 4th connection is for the clock line. A 3 wire connection is only compatible if two devices are to be interfaced using SPI.

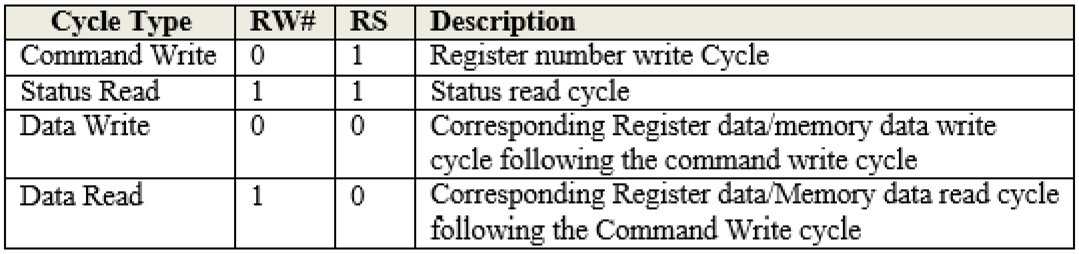
Figure 4.2.6-A shows the pins for SPI interface of the RA8875 controller to be configured.



Pin Name	I/O	Pin Description
SCL	I	*SPI Clock* 4- Wire Serial
SDI	I/O	*4 Wire SPI Data Input* 4 Wire SPI I/F: Data input for serial I/F
SDO	I/O	*4 Wire SPI Data Output* 4 wire SPI I/F: Data output for serial I/F
SCS#	I	*SPI Chip Select* Chip select pin for 4-wire serial I/F
IICA[1:0]	N/A	*IIC I/F: IIC Address Select* Other I/F: NC, please don't keep floating
SIFS[1:0]	I	*Serial Interface Selection* 00:NC 01: 3 Wire SPI 10: 4- Wire SPI 11:IIC If serial I/F is no use, please connect them to 00

Table 4.2.6 – A RAA8875 LCD Tiva C MCU Serial Interface

The chart includes some IIC connections which will not be used so NC will be needed for these pins. For the SIFS pin the logic will be set to "10" or binary "2" for 4-WIRE SPI interface. The RA8875 operates as a slave controller to the Tiva C. The SPI can be configured in command/data write mode or status/data read mode by setting the MSB two bits of first byte of protocol as seen in Figure 4.2.6-B.



Cycle Type	RW#	RS	Description
Command Write	0	1	Register number write Cycle
Status Read	1	1	Status read cycle
Data Write	0	0	Corresponding Register data/memory data write cycle following the command write cycle
Data Read	1	0	Corresponding Register data/Memory data read cycle following the Command Write cycle

Figure 4.2.6-B Read/Write RA8875 Cycle Commands

Since the RA8875 acts as a slave controller it has 3 input lines coming from the Tiva C SCS (chip select), SCL (SPI clock), and SDI (data input). The RA8875 has 1 output line to the Tiva C SDO (data output). This is shown in figure 4.2.6-C below. The serial interface selection line (SISF1) and the power supply are connected to Vdd and the serial interface connection line (SIFSO) must be connected to ground according to Figure 4.2.6-C.

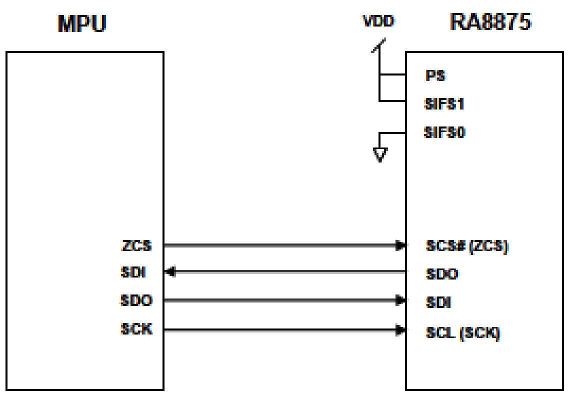


Figure 4.2.6-C Tiva C Interface Diagram 4-Wire SPI (Permission Pending)

RA8875 Pin Configurations - Serial interfaces on the RA8875 are disabled by default. The following jump point connections need to be made to enable the device for 4-wire SPI interfacing:

Short Circuit	J10, J12, J14, J16
Open Circuit	J9, J12, J14, J15
Resistor Values	R1= 10k, R2= 10k, R3=10K

Figure 4.2.6-D Jump point connections to enable 4-wire SPI on the RA8875 Controller

The Jump Point 1 / Connection 1 (JP1/CON1) pins of the RA8875 controller need to be configured for 4-Wire SPI interface according to Figure 4.2.8-D.

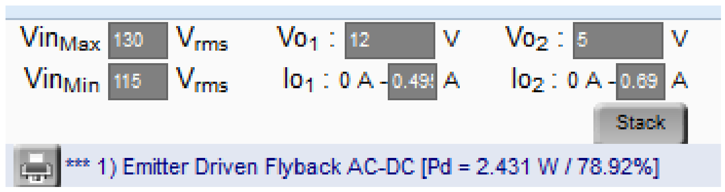
Pin #	Symbol	Description
1,2	Vss	Ground
3,4	Vdd	Power Supply
5	/SCS	SPI Chip Select Chip select pin for 4 wire SPI
6	SDO	4 –wire SPI I/F: Data Output for serial I/F
7	SDI	4-wire SPI I/F: Data input for serial I/F
8	SCLK	SPI Clock 4-wire Serial I/F Clock

Table 4.2.8-D Pin configuration – JP1/Con1 4 Wire SPI I/

Summary - The datasheet for the RA8875 controller provided by buydisplay.com will need to be followed carefully to initialize the appropriate pin configurations for a 4-wire SPI interface with the Tiva C. There are 3 PDF files as well as some example code to get the LCD display up and running. This section includes an overview of interface protocol for the RA8875 controller but may not include all the initializations that need to be made. Since I have never initialized registers for a control driver other than embedded systems, it is unsure (at this point) when reading through the LCD specification sheets how much is just background information of internal driver protocol or things that need to be configured by the user.

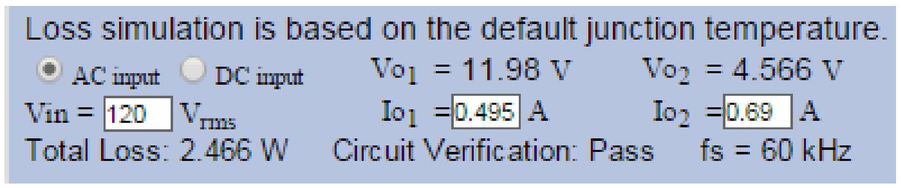
4.2.7 Power Supply Hardware Design

Efficiency Considerations - Upon running the "waveform tool" in the Powersim app, it was found that the output voltage $V_{o2} = 5V$ varied considerably as can be shown in figure 4.2.7-A that $V_{o1} = 11.98$ and $V_{o2} = 4.566V$. V_{o1} is acceptable but V_{o2} is operating at 0.434V than it should be.



Vin_{Max} 130 V_{rms} Vo_1 : 12 V Vo_2 : 5 V
Vin_{Min} 115 V_{rms} Io_1 : 0 A - 0.49! A Io_2 : 0 A - 0.69 A
*** 1) Emitter Driven Flyback AC-DC [Pd = 2.431 W / 78.92%]

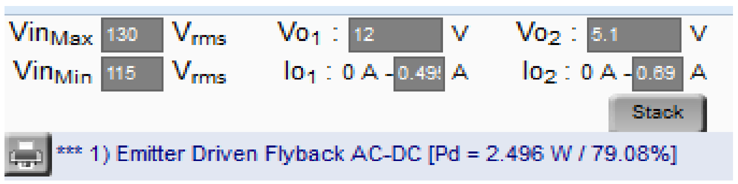
Figure 4.2.7-A Original Input Specifications for Power Supply



Loss simulation is based on the default junction temperature.
● AC input ○ DC input Vo_1 = 11.98 V Vo_2 = 4.566 V
Vin = 120 V_{rms} Io_1 = 0.495 A Io_2 = 0.69 A
Total Loss: 2.466 W Circuit Verification: Pass fs = 60 kHz

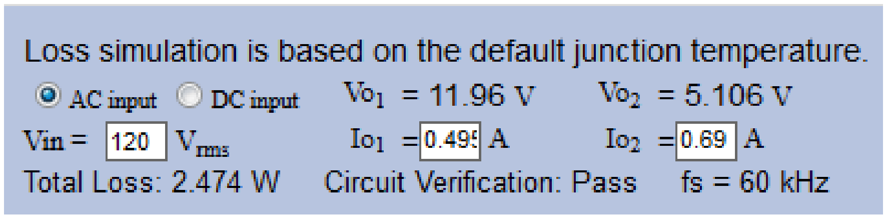
Figure 4.2.7-B V_{o2} Large Output Deviation from Power Supply Design

It was found that if Vo2 was set to a value slightly higher value than 5V DC that is specified for the design that the power losses can be accounted for and an output voltage can be attained that is closer to the desired output of 5V DC.



Vin_{Max} 130 V_{rms} Vo_1 : 12 V Vo_2 : 5.1 V
Vin_{Min} 115 V_{rms} Io_1 : 0 A - 0.49! A Io_2 : 0 A - 0.69 A
*** 1) Emitter Driven Flyback AC-DC [Pd = 2.496 W / 79.08%]

Figure 4.2.7-C Modified Input Specification for Power Supply


Loss simulation is based on the default junction temperature.

[✓ AC input] DC input V_{o1} = 11.96 V V_{o2} = 5.106 V

Vin = 120 V_{rms} I_{o1} = 0.49 A I_{o2} = 0.69 A

Total Loss: 2.474 W Circuit Verification: Pass fs = 60 kHz

Figure 4.2.7-D Slight Output Deviation from Power Supply Design

Circuit Schematic- The Emitter Driven Flyback AC- DC circuit to be implemented for the power supply to be designed is shown in figure 4.4.7-E.

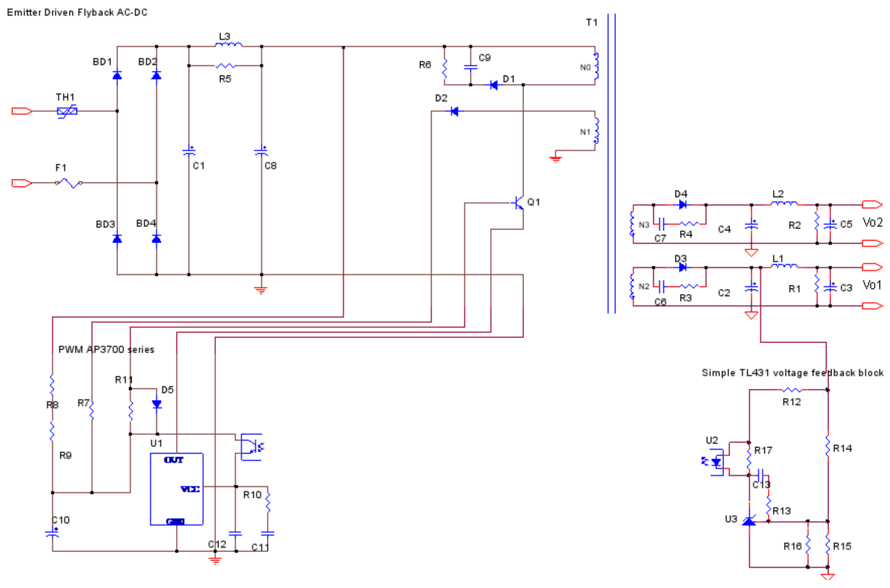


Figure 4.2.7-E AC-DC conversion circuit

Summary - Nothing is ideal in real life. 80% efficiency is considered to be very good in practice. It can be noted in figures Figure 4.2.8-A & Figure 4.4.8-B that the efficiency does increase from 78.92% to 79.08% with the slight modification of V_{o2}. Changing the output to be designed from Vo2 = 5V to Vo2 = 5.1V modified the simulated value from 4.566V to 5.106V which is much closer to the anticipated value. The currents were rated +50% to the actual value needed by components for the project. This of course could also be modified by +/- 5% to get the values closer to the expected ranges. It is better to have the voltage be a little bit higher than the voltage designed than a little bit lower since stepping up a voltage requires much more advanced circuitry than a simple voltage divider to step down a voltage. Problems arose with the BOM. It was thought that the app would generate a bill of materials and you could order all the circuit components in one place._Also, the circuit components had unusual part numbers that could not be found in Multisim for simulation. The total BOM required 61 parts. This issue will need to be sorted out. Research for the power supply will need to be extended over the break to find a solution to this mistake.

4.3 Software Design of the System

While the hardware will provide the parts that are needed to perform the task at hand, the instructions for how to do that are all provided by the software. The software will be what tells the hardware what to do, when to do it, and how to do it. Most importantly though, software is what will be used to identify the Legos in order to separate them.

4.3.1 User Interface

The only means the user has to communicate with the LEGO® sorter is though the LCD touch screen connected to the main controller and image processor. Therefore simplicity is kept in mind when designing the interface so that the user will not be confused one how to navigate through the choices. The user interface of the touch screen is designed using Tiva-C's graphics library. There will be at most 4 different screens to navigate through are the following: the start-up screen, the sorting type and bucket assignment screen, confirmation screen, and status screen.

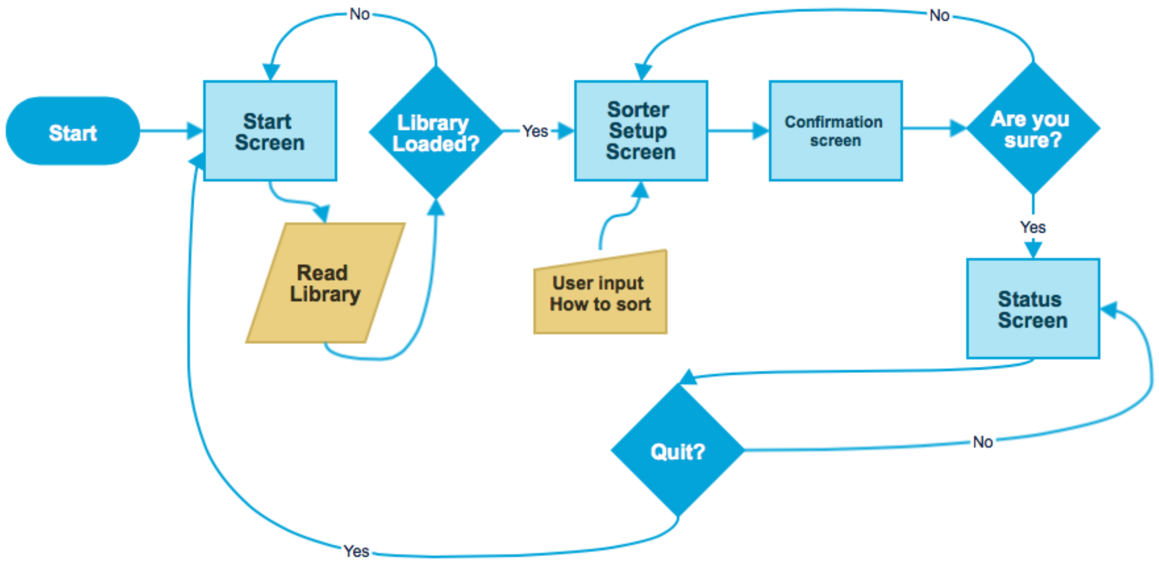


Figure 4.3.1-A Basic GUI Screen flowchart

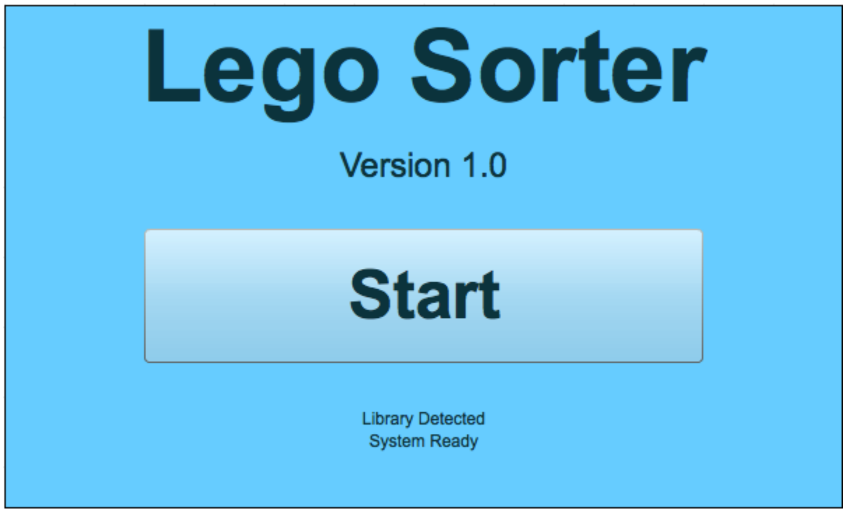


Figure 4.3.1-B Example Start screen

This will be the simplest screen to display it will first display the software name and version of the software. A message of whether the Library has been installed and formatted correctly for use and if the system is ready. Once all those cases are true the start button will appear on screen else it will not appear on screen. Pressing the start button will take the user to the next screen.

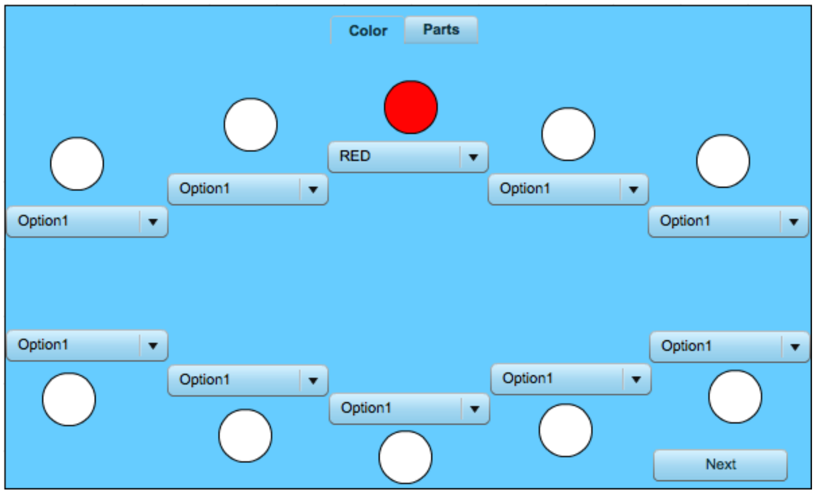
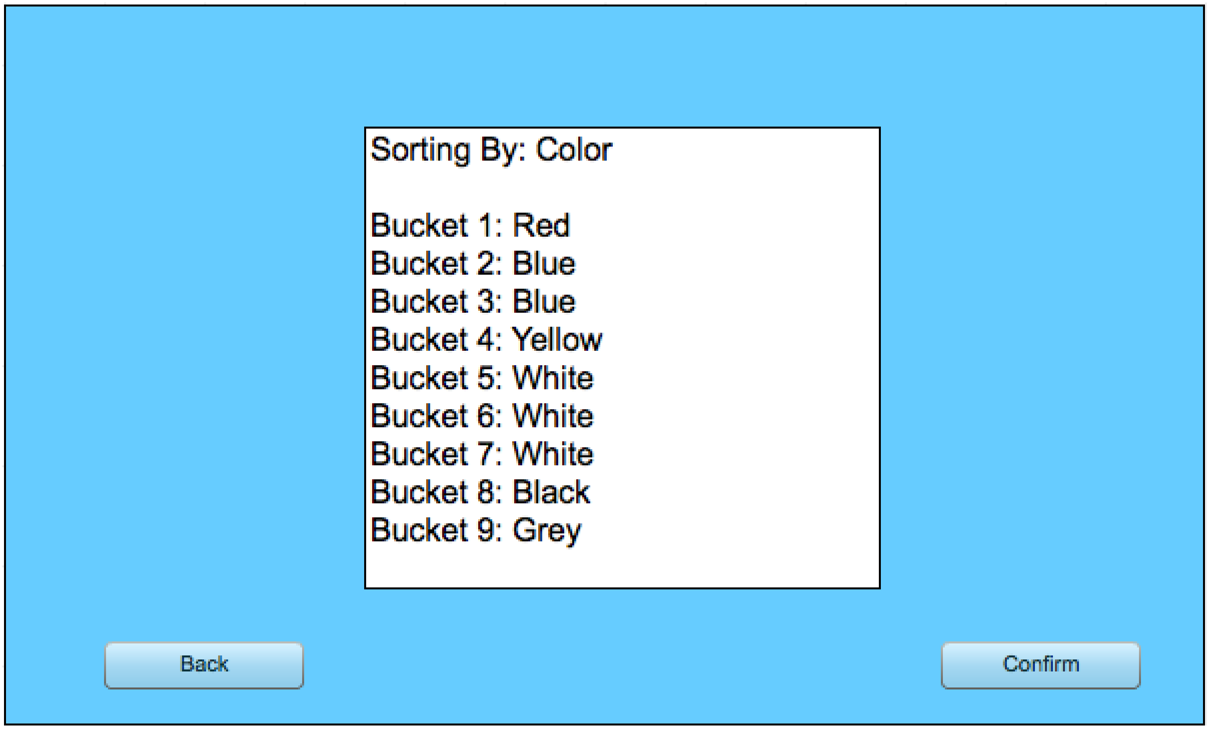


Figure 4.3.1-C Example Setup screen

Here the user will choose how to sort the LEGO® parts by either color or shape. If the user wants to sort color and shape the user will have to run the sorted bricks through the sorter again choosing the setting they did not use on the first run through. Depending on whether they choose to sort by color or by shape will change the bucket assigning choices. Because the interface should be simple a picture of the bucket should be displayed and labels similarly to the actual buckets being used. Near each bucket picture will be a drop menu of choices of what will go into the bucket. The choice information will come from the Library when sorting by part shape and from Color constants present in the software and additional colors added from the library. When chosen the picture will update either by coloring the picture to represent what color it will hold or show a picture of the part inside the bucket showing what kind of part will be put in the bucket. Once the user sets up nine buckets then tenth bucket will automatically be assigned to be the error bucket. The user is allowed to assign more than one bucket to hold the same items. For example the user is allowed to assign two buckets to hold red pieces since red is one of the most common colors it is expected to fill the bucket quickly. Once set the user must press the next button to move to the next screen.


Sorting By: Color

Bucket 1: Red
Bucket 2: Blue
Bucket 3: Blue
Bucket 4: Yellow
Bucket 5: White
Bucket 6: White
Bucket 7: White
Bucket 8: Black
Bucket 9: Grey

Figure 4.3.1-D Example Confirmation screen

Here the user will see a review page of how everything is going to be sorted. It will display which method the user decided to use to sort the LEGO® and a list of where they assigned the bricks. The user will review their set up. If they are okay then sorting will begin once they select the confirm button else the user will hit the back button and reconfigure their sorting set up. While this screen may appear unneeded it will serve as a checkpoint in case the user made some sort of mistake in set up.

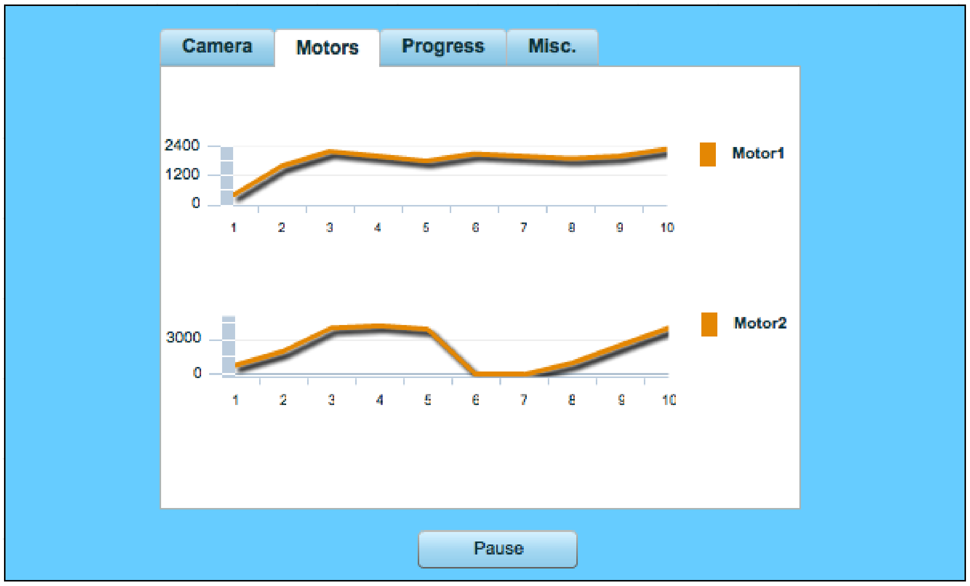


Figure 4.3.1-E Example Status screen

Here the user can see the miscellaneous processes that are going through the sorter like what the camera is seeing the speed of the conveyor belts, and other things. It serves as a way for the user to see if any problems appear visually. It also gives the user power to either pause or resume the sorting process. When paused, the user can decide to quit the current sorting process or resume at another time. If the user selects quit then the user will be sent back to the start screen and the sorter will stop completely.

4.3.2 Image Processing

This section of software programming is debatably the most important in the entire project. This is the part of the system that actually determines what the Legos in the system are, and determines the receptacle that they belong in.

The first portion of the design for this process is that of determining when an image on the camera is ready to be pulled for use in sorting. This will be done by a simple edge detection algorithm. In software, an image is simply represented by a two dimensional array of integers that all correspond to a different color. Ideally, an image of the background used in the image processing chamber will be represented by an array of all of the same number.

A simple pixel analysis algorithm will look for changes in this number as the webcam in the system continues to process the images that it sees. If an unusual spike occurs within the array, this almost certainly means that a Lego has entered the area and the algorithm will be able to process that. The algorithm will have a specific threshold however that determines when a spike is significant enough to deserve attention. This will eliminate errors that could be caused by smaller jumps in the array. For example, two cells in the array may only differ in value by one. This is almost certainly due simply to changes in the lighting, and thus can be ignored by the edge detection algorithm.

So the edge detection algorithm will be able to find where the Lego starts with ease, but the difficulty here lies in determining when the Lego has become fully introduced into the chamber. In order to do analysis on the Lego to determine shape and size it will take more than just half the Lego to gather details of course. The solution that is proposed for this particular issue is to simply run the edge detection algorithm along one column of the array, for the portion of the image that represents the top view of the Lego. The beginning of the Lego will be determined at the first spike in the values, and the algorithm should be continually experiencing this spike until the Lego has passed all the way through the particular column. So after the initial spike, the algorithm will simply wait until it experiences a segment with NO spikes in it. This will be the signal for the software to begin the full analysis of the Lego.

An important note about this process, is that it will be necessary for EVERY Lego, EVERY time, despite the simplicity of the analysis that needs to be done. An uneducated observer may argue that it isn't necessary to wait for the entire Lego to be on screen for a simple color analysis, but if multiple Legos are in the chamber at once it could cause an error.

In the case of determining the color of the Lego, the process will be incredibly simple. By searching through the two dimensional array of the image, it will determine what color the Lego is simply based off of the number that represents that color. Through testing, ranges will be determined that represent the multiple different colors that the project will be handling. It is important that these colors are represented by a small range of numbers instead of just a singular number. This will account for any possible discrepancies caused by the lighting. The most difficult color to determine will of course be white, seeing as this is going to be the same color as the background for the image processing chamber.

In the case of sorting by shape and size, processing the images will of course be slightly more involved. The two images that will be gathered from the camera will be able to gather all of the dimensions of the more basic Legos. The basic rectangular bricks will be, by far, the simplest bricks to distinguish. Using the images gathered and some very basic math applied to that information, it will be incredibly easy to gather the dimensions of the brick. So the approach that will be used will be to create a library of the parts that will be sorted by the algorithm. This will be a simpler algorithm and will most likely take less time than a sort of image comparison algorithm, but that will translate into more time consumed by simple data entry. This will require inputting details for every single Lego, and in order to gather those details it will be extensive testing with the image processing chamber.

The math that the algorithm will have to use will be to determine the length and width of the Lego in question, and then the side view of the Lego gathered from the camera will be able to provide the width. This being said, however, not all of these basic building Legos are going to be simple rectangles, so it is not safe to assume that the length and width are viable dimensions. But calculating the surface area can actually be incredibly simple. By measuring the surface area in array cells, that information can be used throughout the algorithm. With more complex figures to determine as well, such as the Lego people, determining shape may actually be more easily determined by analyzing the color first. So the color analysis will always be performed first, even if shape is the factor that is supposed to be determined. Only in extremely rare cases are basic bricks multi-colored, so this will be incredibly useful to determine the more complex pieces.

After the analysis has been completed in full on the Lego, it is also extremely important to analyze when the Lego has left the image processing chamber. To eliminate any possible error, it is important that an image only gets fully processed when there is one and only one Lego in the camera's field of view. To account for this, the same edge detection algorithm that was used to determine when a Lego ARRIVES can be used to determine when it DEPARTS by using it in the final column of the array. When the algorithm notices the spike for the first time it keeps track of this until it gets to a point where there are no longer any spikes in the array. At that point, the program will know that if a Lego is currently in the image processing chamber that it is now ok to process it.

Another important part of the image processing algorithm is that it will be able to determine whether or not the system needs to shut down if no more Legos are passing through. A predetermined value will decide how long the system will need to wait until it is ok to shut the system down completely. This value will likely be determined in testing.

So in summary, the software design of the image processing portion of the Lego sorter will run in a series of steps that are repeated until no more Legos are being presented into the image processing chamber. The steps are as follows:

1. Determine the initial entry of the Lego.
2. Determine that the back end of the Lego has also entered to begin full analysis.
3. Perform any color or shape analysis necessary by calculating surface area and height of the Lego.
4. Compare the results to the Lego library that has been created.
5. Output results to rotating arm unit to dispense Lego.

6. Determine exit of the Lego from the chamber.
7. Repeat process until Legos have stopped entering the chamber.

4.3.3 Conveyor System

In the case of the conveyor system, the software is going to handle a couple of different details. First and foremost is the speed of the two conveyors. The first conveyor will move at a much slower pace than the second. This is to ensure that Legos get spilled onto the second conveyor belt only one at a time. This will allow them to be spaced out as much as possible on the second conveyor belt. The second conveyor belt will then be able to move the Legos one at a time into the image processing chamber. The space between the Legos will help to ensure as few errors as possible in the image processing.

The other detail that has to be accounted for is simply whether or not the conveyors are running. While a Lego is being processed for sorting, this may cause a delay in the system requiring the conveyor belts to stop for small periods of time. The software needs to be able to communicate this with the conveyor system so that it can start and stop the belts freely.

Simply put, the software will communicate to the conveyor belts a static speed value that will be determined in testing, and the on/off state of the conveyor belt will be handled as a Boolean variable.

4.3.4 Lift Arm System

In the case of the lift arm system, one detail that the software needs to handle is the intervals in which the lift arm activates. In the case of the activity variable, if the lift arm activates too often, it would end up overloading the conveyor belt with too many Legos causing errors in the system. A value needs to be set so that there are intervals between lift arm activations.

The second variable in the lift arm system is its speed. If the lift arm moves too quickly, it could throw Legos past the conveyor belts and off of the system, but if it moves too slowly, the Legos may not move off of the lift arm.

The speed of the lift arm will be a static variable that will be determined in hardware testing.

4.3.5 Errors

With main moving parts that make this whole system function there are many things that can go wrong if not corrected. Even though there is a kill switch to be installed in case of emergency there should be methods that can be used to monitor and treat the problem as it appears, report the problem to the user or just maintenance. Here are some problems that could surface and solutions to those problems.

Conveyor Belt Jam - With many small pieces constantly spilling into the conveyor belt there is a chance that some like an axle can fall in the spaces or somewhere it shouldn't be like where the motors are. It could end wedge in with the motor and could stop the system or break a vital part or perhaps the belt itself could get caught in the motors seizing it up or some other outside force that managed to get wedged in the conveyor belt motors. There has to be a method implemented that will notify the main MCU and force a pause or shutdown of the entire system. What can be done is that the MCU that are controlling the motors report the speed that the motors are rotating. For normal performance they should be rotating at a constant speed. If there is any disturbance to the motors that will cause a noticeable variation a flag will be sent notifying the main MCU to stop the system and notify the user of the problem by displaying a prompt on the LCD display.

Motor Speed - The speeds of the two conveyor belts are important to the system, as they are the ones the space out the LEGO® parts to be processed. Because of this it is important that the maintained speed should be at a specified range. If the motor begins rotating at a higher or lower speed than it should it should self-correct itself and continue running otherwise stop the system and notify the user.

Jammed Chute - This is similar to the motors in the conveyor belt some outside force could get wedged into the motor. A sensor needs to pick up on those abnormal changes and report it to the main controller, display the message, and halt the entire system.

Lift Jam - Similar to the problem with the motor jam it is possible for the smaller LEGO® parts to become wedge into the lift components or some other form of outside force. So there needs to be some kind of pressure sensor to detect that something is wedged in as it would detect a change in the amount of effort the lift needs to exert. The flag will be sent to the Main controller and halt all processes and notify the user.

Multiple Pieces - The system that we have designed cannot handle processing two objects at the same time. So there needs to be a protocol that handles what to do if more than one part has entered the image processor. The image processor should pick up on the number of objects on screen. (There should be two. One for the actual part and the other is the reflected image of the part) If there is more parts then the sorter should be able to check is they happen to be the same color or same part and sort normally else all pieces goes into the error bucket.

Power Surge - The microcontrollers the LEGO® sorter are using are all low power so it is very important that the amount of voltage going into these devices are monitored carefully or else damage to the device will occur or a fire. There should be some kind of sensor to monitor the power input and output and send a flag is there is a spike in power. In case of smoke, or electrical fire the emergency kill switch must be used.

Memory - Because image processing is being used to determine what each LEGO® part is according to what is stored in the library it is important to take into account the amount of memory needed to perform this process. Therefore there should be some kind of safeguard in case there is no more space in memory to perform image processing. The Beaglebone must have a way to keep track of how much space in memory is available and communicate to the machine to stop and have enough memory to display the message that memory is low.

Camera - The camera must be stationary as it must be at the correct angle to view the incoming LEGO® parts and the mirror. A way to check if the camera is in place is to use some kind of mark to look at before the sorter starts up. If the camera sees the markers at a certain part of its view then we know the camera is in the right spot and the machine can start otherwise the main controller is notified and displays the message to the user that the camera is out of place. Also the LCD screen should have an option to display what the camera is see so they can see themselves if the camera is in the right positioning see if the parts entering and leaving.

Weight of Buckets - The user will have a choice to assign more than one bucket to hold the same type of parts for example two buckets both red parts. However the problem comes is that once a bucket is full the sorter need to know not to use that bucket anymore and look for another bucket that will hold the part otherwise the system will stop and display to the user on the touch screen which buckets are full. So now the user will have to unload the full buckets and replace them with new ones. The system will then recalibrate the weight for that bucket. So each of the buckets must have a weight under the buckets and be calibrated to ignore the weight of the buckets. Each of the weights needs to report the current weight to the main controller as it will decide if the bucket is full or not. Here we just need to hook up each of the weights to the main controller.

LCD Touch Screen - With touchscreens it’s important that the touchscreen portion remains accurate at all times. To do this the screen has to be calibrated every so often to function at top performance. There should be an option to run a calibration test if the user feels that the touchscreen coordinates are out of place. The system should ask the user to run the calibration test for each set amount of time. The test will just consist of the user pressing designated points of the screen.

4.3.6 Code Integration

The entirety of the Lego sorter is planned around building smaller subsystems separately, and then integrating all of them seamlessly together in the final stages of preparation. The software for the sorter is no exception. Each subsystem will be coded and tested on its own to ensure that each part works properly, and then all of the parts will ultimately be combined in the final stages.

A major advantage of taking this particular approach to the coding of the Lego sorter is that it makes things much easier to debug in the final stages. If all of the parts were put together from the beginning and then immediately tested, it creates a scenario where the bug could be in any number of locations, whether that be the subsystem, the main portion of code, or simply in the communications between the systems. By testing the subsystems before integrating everything together, this eliminates the possibility of the subsystem itself being at fault, and makes it easier to locate the problem.

A small downside to designing the software in this fashion will be the need to keep track of the variables across different sections of code before they are integrated with each other. When the subsystems begin to be integrated with one another, it will become imperative that all of the variables are accounted for so that the communications between each of the systems works properly. It’s a minor detail that can be tedious, but if not properly handled it could cause major frustration in building the project in the long run.

Once each of the individual subsystems are working on their own, the only concern will be the communications between them. The method that is being used to code up the software will allow major focus in specific areas of the software one part at a time, rather than dealing with an overwhelming amount content areas simultaneously. The plan promotes good organization in the software, and it also makes it easier to plan out time to ensure the best outcome for the project.

5.0 Interfacing Beaglebone Black & Tiva C

Slave/Master Overview - The interfacing will be accomplished by using a 4 wire SPI Master/Slave system of serial synchronous data transfers between the three digital devices of this project. Figure 5.3– (a) depicts a basic block diagram of the physical connections of the devices to be interfaced via SPI. The devices communicate via 4 wires: The SPI Clock, Master Output Slave Input (MOSI), Master Input Slave Output (MISO), and a Chip Select Line for the Master to choose which device to communicate with.

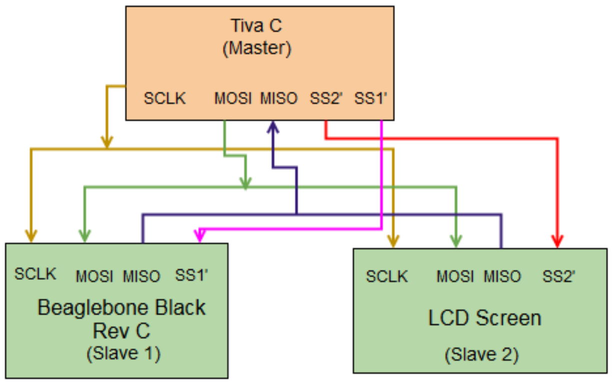


Figure 5.3-A: Beaglebone, Tiva C, & RA8875 LCD Controller Slave Master Block

As you can see, the Tiva C is the master with two slaves being the RA8875 LCD display controller and the Beaglebone Black REV C. The LCD interface is discussed in section 4.2.7.

Beaglebone Black Rev C - A website was forum was found specifically on SPI interfacing the Beaglebone Black to aid in the set up process. [http://beagleboard.org/Community/Forums]. It was also found that the Beaglebone Black will need a 3.3V serial adapter. [http://beagleboard.org/support/faq#Serial] The Beaglebone Sitara AM3358BZCZ100 Processor has two SPI serial connection buses. These buses are accessed through pre-initialized objects named "SPI0" and "SPI1." It appears as though "SPI0" is to configure the BBB as a slave and "SPI1" configures the BBB as a master due to the extra chip select (CS) line as shown in Figure 5.3-B

#SPI0	**#SPI1**
CS0 P9_17	CS0 P9_28
MISO P9_21	CS1 P9_42
MOSI P9_18	MISO P9_29
SCLK P9_22	MOSI P9_30
	SCLK P9_31

Figure 5.3-B BBB Pre-Initialized SPI Objects

Tiva C- The “TivaWare” for Tiva C series offers a lot of example programs that come with the software. In order to get familiar with how the MCU works these programs will be compiled using the experimenter board to get started. Some of the programs that will need to be tested to understand the SPI interface are “softSSI” which configures the drivers for the Tiva C and the corresponding “softSSI.h” which includes macros and defines.

Summary- The interface will be a challenging task. The BBB and the Tiva C will each need to be configured appropriately to communicate with one another before the physical connections can be made. Before the devices can be interfaced to each other via SPI they will have to be configured separately according to the given specifications and protocols of each device.

6.0 Prototype Construction

Section six will outline the process that will go into assembling the final prototype. This will cover both the hardware portion of the project as well as the software portion.

6.1 Materials List and Parts Acquisition

As of now our team has no sponsors so much of the parts we have we from previous projects or given to us via workshop or donated. For parts we do need we have all agree to spend what is needed to get the materials needed.

Parts	Price per Unit	Qty.	Total Cost	
Beagle Bone Black	$45.00	1	$45.00	
Microcontrollers	$-	4	$0.00	
Logitech WebCam	$17.00	1	$17.00	
RA8875 5" ER TFT-M050-2 LCD	$27.61	1	$27.61	
Legos	$-		$0.00	

Containers	$10.00	10	$100.00	
Ream Paper		2	$0.00	
Mirror	$5.00	1	$5.00	
Load Sensors	$10.00	11	$110.00	
DC Motors	$-	2	$0.00	
Lift Arm	$-	1	$0.00	
Stepper Motor	$17.00	1	$17.00	
Tiva-C TM4C 1294	$-	1	$0.00	
Servo Motor	$30.00	1	$30.00	
FeedBack Sensors	$20.00	2	$40.00	
LED Light	$1.00	1	$1.00	
			$392.61	Total

Figure 6.1-A Projected Budget

6.2 PCB Vendor

As suggested there are two vendors to be considered for ordering the PCB for the project, OSH park and 4PCB.

Osh Park – OSH park has a lot of resources for PCB design. The website offers a lot of resources about common file formatting issues and plenty of links on how to solve these issues including 13 files on general “FAQ” and 7 files on CAD Package specific help. The price is more than fair as can be seen in Figure 6.3-A. OSH Park supports Eagle Cad “Eagle BRD” files is only compatible with version 6.6 up to 7.1. The only real issue is the processing time and fabrication time. Design will probably require two layers. Time will need to be managed efficiently to allow 1 business day process time, 12 day fabrication time, and shipping time. It will take a minimum of 2 weeks for the PCB to arrive.

OSHPark	Price	# of Copies	Order Process Time	Fabrication Time
Standard 2 Layer Board	$5/in^2	3	1 Business Day	12 Days
Standard 4 Layer Board	$10/in^2	3	Weekly	14 Days

Figure 6.3-A PCB vendor OSH Park

4PCB – 4PCB offers a free design tool "PCB Artist" that can be used to develop the design needed. 4PCB offers a free file review which is reviewed by engineers so if there are any errors in design they will be found immediately with no surprises later. Figure 6.3-B shows that the turnaround time is very quick. It should take no longer than 1.5 weeks for the PCB to arrive.

4PCB	Price	# of Copies	Order Process Time	Fabrication Time
Standard 2 Layer Board	$33	4	Same Day	3 Days
Standard 4 Layer Board	$66	4	1 Business Day	5 Days

Figure 6.3-B PCB Vendor 4PCB

Summary- Depending on the size of the layers the price could be swayed in either direction since 4PCB has a flat rate for their PCB layers and OSH Park does not. Since our team has no prior experience to PCB design or EagleCAD software, 4PCB would be the winner since they have a professional engineer to check the design and send an email notification of the design flaw. In summary, 4PCB seems like the appropriate vender for this time sensitive project.

6.3 PCB Software

The PCB vendor chosen is 4PCB. 4PBC offers free and intuitive PCB design software called PCB Artist. The software has extensive libraries for relatively quick designing. The software allows the designer to first make a schematic layout of the circuit with traditional circuit symbols and wiring. The software can automatically create a rough PCB layout from the schematic, which then requires the designer to place components and reroute wiring where needed. The software is free to download and use and directly communicates with the vendor, 4PCB. PCB Artist will be used for the PCB design layout of the project.

6.4 Final Coding Plan

This section will outline the current plan on coding the system. The major parts to this include the image processing portion, the user interface, the moving portions of the system, and error handling. In order to do proper testing for the final prototype, all of these portions need to be able to work individually, then they have to be able to accurately communicate between each other.

6.4.1 User Interface

Since the Tiva – C is going to be used to control the LCD screen then the Tiva – C graphics library is going to be used for coding the systems GUI as it is already built in. All the code and function provided are already written in C and is designed to easy to implement. There are also free resources provided by the Texas Instruments that can support the preset library.

Initialization – Before any drawing takes place all data needs to be cleared out and the clock rate needs to be set. In the main function the clock is set to the appropriate frequency, then the LCD screens drivers are initialized by the function provided by the LCD screens files. The drawing context is then initialized and sets all drawings that will occur to be displayed onto the LCD, and finally clear the entire screen. Because a clear screen function does not existed it must be created.

Start screen – The start screen will just contain text displaying the name of our system and a push button that will only be enable when the system is ready. In order to implement this screen first the menu color is set, then the Name and version of software, and status of the system is displayed. The button appears when both the Library and camera are set. Because there is no function that handles event like in Java a function must be made to let the user move on to the next page.

Setup Screen – This screen is used to assign sorting type and where each Lego will go. On screen there will be two buttons that will decide sorting type. 10 circles that will represent the buckets and ten dropdown menus that contains the list of colors or parts that will go to each bucket and a button that leads to the next screen.

Confirmation screen – This screen is the simplest of the screens as it is just a text box showing what the user choices from the setup screen and two buttons with one leading back to the set up screen and the other leading to the status screen.

Status screen – The final screen is the most complicated screen as it has the most functionality. It shows the status of the camera, motor, and other sensors in system. A group of buttons will act as tabs that will bring the corresponding widgets on screen. For example pressing the camera button will bring up the camera screen and pressing the motor button will bring the motor status. The will also be a pause button that will stop the entire system and a new window will appear asking to quit or resume. Pressing resume will remove the window and the system resumes. Pressing quit will open a confirmation window with a yes or no button. Pressing yes will bring back the start screen. Pressing no will go back to the pause window.

Error windows – As the system run is possible that problems can occur. There the LCD should report any problem that can occur when the system is running. It will take in an error value based on the sensor and bring up an error window on screen.

6.4.2 Error Processing

Error processing is designed to handle any problems that occur in the hardware and software. With the amount of moving parts in the sorter it is safe to assume most of the errors will come from some sort of outside force there for sensors are needed for the vital parts of the sorter in or to catch any problem that appears. As for the software the vital areas are the image processor and the library as without them there will be nothing to compare the Lego parts with. Now once these problems arise depending on the severity of the problem the system needs to either pause the sorting and state the error on the LCD screen or dynamical correct the problem as it occurs and continue running. The coding language used to run this operation will be C language as that is the language the microcontrollers use.

Motor Speed – Because of how the conveyor belt is built and how it function the speed of the motors are critical as the conveyor belt is what helps separate the Lego parts before they enter the image processor. Therefore the speed of the motors of the first and second conveyor belt must be run at specific speeds, the first belt running at a slower speed and the second conveyer belt running at a faster speed. The motor drivers will constantly report the speed of the motor as it is running. If for some reason the motor run at a faster or slower speed than it is allowed the drivers should correct themselves otherwise the Conveyor Belt Jam error process is performed.

Conveyor Belt Jam – This occurs when some outside force is forcing the motors driving the conveyor belt. A way to monitor this is to keep track of the speed as it is running. If there is some sort of interruption of speed for a certain amount of time that failed to correct itself, then the system throws an error and stops the sorter. A message then displays on the LCD that an error occurred in one of the motors.

Jammed Chute – This error is similar to the Conveyor Belt Jam problem and solved through a similar fashion as figure 6.5.4-B.

Power Surge – This is a very important error to catch as many of the components running this machine are low power and a sudden change in power will damage the sorter or lead to a fire. To avoid that the power supply should shut itself off.

Memory – Because this sorter is dealing with images it is possible to quickly run out of memory though it is unlikely. The system just needs to monitor its memory usage. If for some reason there is not enough memory to perform then the system will stop.

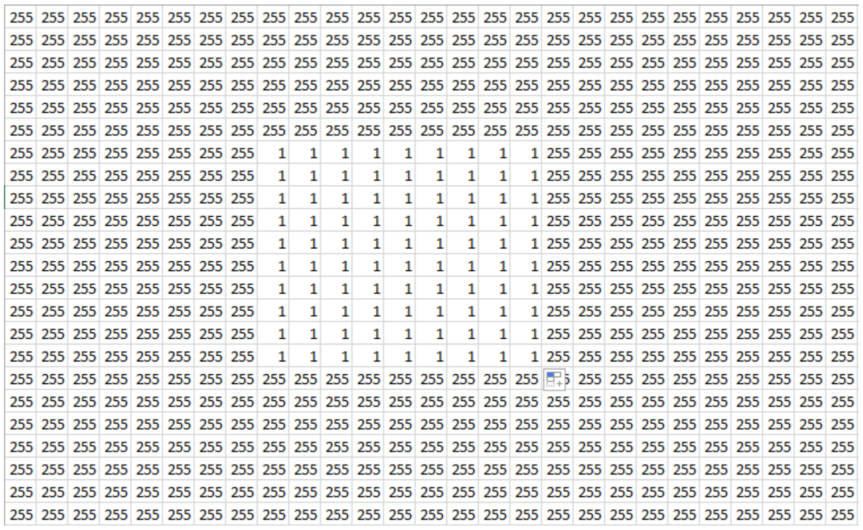
Camera – Before the system can even run the camera must be set in place. The camera is set when it sees certain marks in the area were the image processing takes place. If at any time the camera is moved out of place the system will pause and state message on the LCD screen that the camera is off center. Once back in place the image processor will rerun if there is a part that needs processing and continue operation from there.

Weight – When the containers become full that cannot be used anymore because the parts will just over flow. With the load sensors under the containers the system can keep track on the amount of parts in the container. Once the load has reached a certain threshold the rotating arm will skip that container and instead rotate to the next assigned container otherwise to the error bucket. If the Error bucket is full the system will pause and alert the user that there are full containers.

6.4.3 Image Processing

The process of coding up the image processing portion of the system is one that will have to be methodical and fairly meticulous. With it being the main brain behind the sorting of the Legos, it is absolutely imperative that the image processing code works EXACTLY how it's supposed to in every aspect. If it's even slightly incorrect, it could result in more errors in the final project than successful sorts, and that is unacceptable for the project.

The important thing to understand when it comes to the image processing is the two dimensional array that represents the image. Figure 6.4.3-A shows a rough visual representation of what a two dimensional array would look like with an image from the image processing chamber.



255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	1	1	1	1	1	1	1	1	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Figure 6.4.3-A

As shown in the figure, 255 is the number that is representing the background of the conveyor belt, while in this hypothetical case, 1 is representing the color of the Lego currently in the chamber. This is going to be the representation of all of the images that are processed throughout this entire program. To reach this result the code will have to move through a number of steps.

Step 1

The first portion of coding that will be done will simply be coding to read in images from the camera. This is a very basic step that will set up the foundation for the rest of the coding done in the image processing portion. It isn't an incredibly difficult task, but it's where everything needs to start, and this will allow initial testing with the image processing chamber to begin.

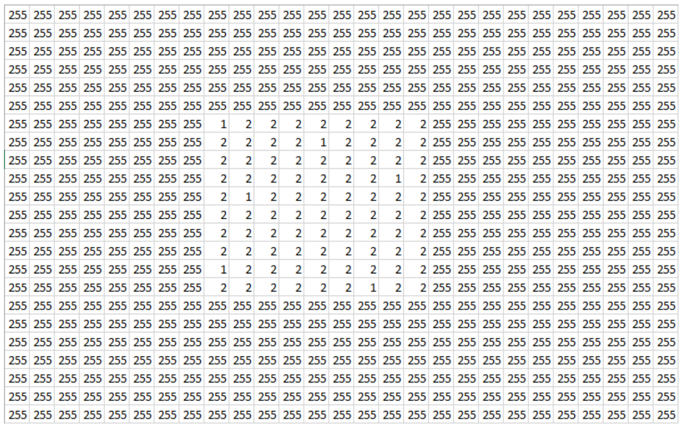
Step 2

The second step is to write the simple algorithm that detects when a Lego has FULLY entered the image processing chamber. This algorithm simply goes through the first column of the two dimensional array that represents the image given by the camera. The algorithm goes through the column of the two dimensional array that represents the image given by the camera and if it finds a color significantly different than white (the background) it will know that a Lego has indeed entered the area. It continues this process until it reaches a point where it reads in nothing but white again, indicating the Lego in question has gone completely into the chamber.

In code, two Boolean variables will be used (FRONT and BACK) to indicate that the front or the back of the Lego has entered the chamber. A loop will continue to check the same column of code until both FRONT and BACK are true, at which point the code will break the loop and move on to the next portion of code.

The code for this portion is fairly simple and very active. The only part that is a bit more time consuming is looking for the back end of the Lego. With the front end of the Lego, once it has been detected, the “for” loop can be broken and the program can proceed. In the case of the back end, however, the algorithm must always check the entire column.

Step 3
The next part of coding will be to test color detection. After the edge detection has done its work, it will be important for the colors to be detected correctly. This portion of coding won’t be so much a contribution to the algorithm as it will be testing to determine thresholds for the different Lego colors. With all of the lighting in the image processing chamber, the chances of a single Lego showing the same color code for all of its pixels is EXTREMELY low. There will definitely be a small range of numbers that each color will represent and testing will help to determine these ranges. These will then be set in the code so that when the program goes to detect colors, it will return every color represented within the image. Ideally, there will be two colors returned each time: white (the background color) and the color of the Lego. In some cases it may only return white if the Lego is white as well. As shown in figure 6.5.2-B, this Lego is representing most of its pixels as a value of 2, with some discrepancies where the value is one. These are very likely caused just by the lighting in the image processing chamber, and this could very well be the exact same Lego that was shown in figure (…………).



255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	1	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	1	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	1	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	1	2	2	2	2	2	2	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	2	2	2	2	2	2	1	2	2	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Figure 6.4.3-B

The coding for this portion will be fairly simple. Through testing, predetermined ranges of values will be set for the different colors. So to determine what colors are in the image currently being processed, the program will simply go through the image and determine all of the different colors represented. A simple brute force algorithm used a double "for" loop will suffice. The algorithm could possibly be further simplified by only processing the area that the Lego is definitely in, but the process in which that would be done will be unclear until testing has been performed.

The code will create an array that will contain the colors detected within the image so far, but will not contain duplicates. Ideally this array should only have data in two cells at the end of the process and then this can be used to ultimately determine which bucket the Lego will go in when sorting by color.

Step 4
This will by far be the most tedious and time consuming part of the code. It will require extensive testing, a good deal of data entry, and a good deal of plain effort.

From what research and preliminary observation has been done, the easiest and most efficient way to perform the shape recognition will be with some basic measurements. The part of the image that captures the top view of the Lego has the advantage that no matter what orientation the Lego is in, it will always be showing the same amount of surface area. The other part of the image (the side view) may will not have that advantage. But it still is able to gather the height of the Lego accurately. Theoretically, these two pieces of data SHOULD be able to narrow down the Lego selection to the particular Lego in question. These details can be measured simply in array cells. This is where the extensive testing and data entry will come in. A small library of the Legos the sorter is accounting for will have to be built, containing the details of the measurements. This will also have to be measured for the different possible orientations of the Lego (IE if the Lego is on its side) and have those measurements entered into the library as well.

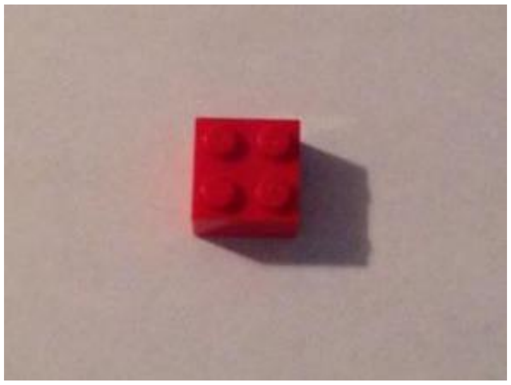


Figure 6.4.3-C

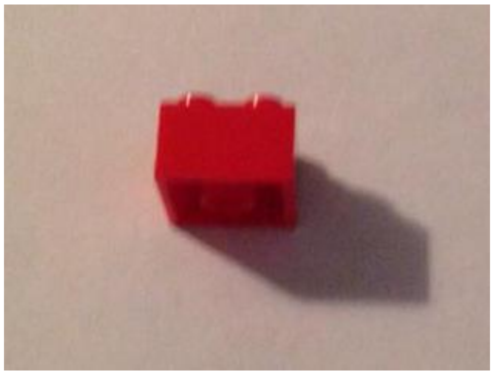


Figure 6.4.3-D

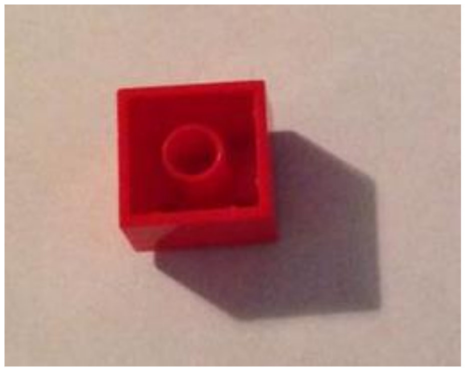


Figure 6.5.2-E

This library will be incorporated into the project so the image will take its measurements, compare the results to the library, and then compare the value returned from the library to the values associated with the receptacles in order to sort it in the proper position.

So the coding to read in the image is going to be extremely simple. It will be two separate double for loops (one for each portion of the image). The first set will be for the topside view. This view will simple count the cells that are not representing the background, but instead are representative of the Lego itself. This will give a measurement of the Lego's surface area in pixels. For the second portion of the image, the algorithm will run through each row searching for whether or not that ROW contains the color it is looking for. The code then simply keeps track of how many rows that were counted, and this will be the value that is used for the height of the Lego.

These steps combined together will ultimately make up the whole of the image processing algorithms. These are all of course pending change that may have to occur as a result of unforeseen circumstances, but these plans provide a firm foundation on which to start.

6.4.4 Other Subsystems

The coding plan for the subsystems is fairly straightforward. There doesn't need to be excessive coding, as the other subsystems don't have a massive amount of variables to handle. The only exception to this, may be the rotating arm system, which has to deal with each of the individual buckets.

Lift Arm System – The lift arm system will be treated as its own object, containing a few very basic variables. The variables will account for the speed that the lift arm moves, the interval at which the lift arm moves, and then a variable to denote whether or not the lift arm is active.

Variables:

- int Lift_Speed – this variable will account for the speed at which the lift moves up and down
- int Lift_Interval – this variable will account for the time interval in between lift activations, to make sure there is proper space between loads of Legos
- boolean Lift_Status – this variable is simply to denote whether the lift is active or inactive, with the default being inactive

Conveyor Belt System - The conveyor belt system will be only slightly more complex to represent in the code than the lift arm system, only because there are two conveyor belts to account for in the code. The necessary variables for the conveyor will have to simply account for whether or not each belt is active, and the speed at which each belt is moving.

Variables:

- int BeltOne_Speed – this variable will account for the speed of the first belt in the conveyor system
- boolean BeltOne_Status – this variable will determine the current status of the first belt in the conveyor system
- int BeltTwo_Speed – this variable will account for the speed of the second conveyor belt, which will definitely be greater than that of BeltOne_Speed
- boolean BeltTwo_Status – this variable will determine the current status of the second belt in the conveyor system

Rotating Arm System - Of the mechanical subsystems in the Lego sorter, this will be the most complex with the software. It will require variables to account for each of the buckets, as well as the rotating arm itself. Also there will need to be variables for each of the sensors that the rotating arm will be reading to know it has reached its destination. The buckets will be treated as objects that will each have an enum variable called LegoType that will track what type of Lego it contains. The rotating arm will also be its own object, containing variables for speed, whether it's active, and location.

Variables:

- Buckets[8] – the Lego library will contain an enum called LegoType in order to identify each of the Legos, and these Lego types can be assigned to each of the individual buckets, which will determine which bucket a Lego will be sorted into
- int Arm_Location – this variable will simply keep track of the current location of the rotating arm as it continues to travel around the buckets to find its final destination
- int Arm_Speed – this variable will account for the speed that the rotating arm will be moving
- boolean Arm_Active – this variable simply accounts for whether or not the arm is currently moving

After these various subsystems have been taken care of, the next step will be to integrate all of these parts into the main portion of code with the image processing chamber. The integration should be fairly simple at this point, given that each subsystem is properly working on its own. Each of the variables listed will be assigned by the main portion of code and then changed based on the status of the image processing chamber.

Status Changes:

- Conveyor Belt System – Ideally the conveyor belt system will be moving the entire time that the sorter is working, nonstop, in order to optimize the efficiency of the project. This, however, may not be how everything works out. If a Lego begins to reach the end of the image processing chamber before the image processing algorithms have determined the ID of the Lego, the conveyor system will need to stop moving to allow time to process the Lego and move the rotating arm before moving forward.
- Lift Arm System – The lift arm system will ideally be able to recognize if it needs to change its intervals by a small amount using a sensor on the conveyor belt. It will also need to stop just like the conveyor belts in the case of an image taking too long to process.
- Rotating Arm System – The rotating arm system should only be active as soon as the analysis on a current image is complete and the result has

determined the final location of the Lego being analyzed. So the status will change to active when it receives the signal to move, and inactive as soon as it reaches its location.

All of these processes have been taken care of in the code, the prototype will be ready to be pieced together and prepared for testing.

7.0 Prototype Testing

This section outlines the prototype testing from the hardware and software perspective.

7.1 Hardware Test Environment

Since this project is intended for indoor usage the hardware can easily be tested in the labs provided on campus, including the Senior design lab and the innovation lab. Circuit simulations will be done using Multisim for the initial testing. The final circuit simulation schematics will need to be done using the appropriate software that is compatible with the PCB vendor selected for the project.

7.2 Hardware Specific Testing

This section will outline the hardware testing of the three main mechanical subsytems: conveyor belts, rotating arm, and the lift arm

7.2.1 Conveyor Testing

Present Testing - Much of the conveyor belt testing has already been accomplished. The first conveyor belt has successfully been built and the desired speed has been attained. The DC motor used to test Belt #1 was purchased from Skycraft. It is a 6-24V DC motor. The testing was done in home using an old homeless AC adapter. The AC adapter's rated output was 12VDC @ 1200mA. The 12V made belt #1 slow enough to drop the LEGO's one at a time onto the (second) belt. The charger adapter was cut off and the wires were stripped for testing. The +Vcc and ground wires were connected to the corresponding contacts on the DC motor as shown in figure 7.2.1-A. Please note the photos were taken long after testing and the wires were not connected. The figure also shows the "coupler" which is a LEGO technic connecter that was hot glued to the shaft of the motor which connects to the axle of the conveyor belt wheels.

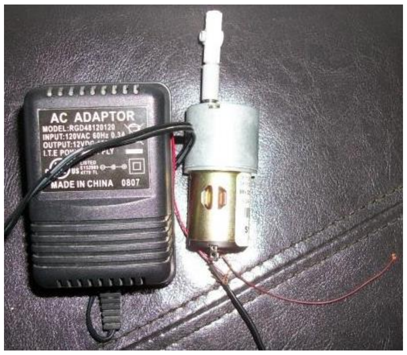


Figure 7.2.1-A DC motor with LEGO "Coupler" for conveyor belt testing.

The belt runs pretty smoothly for a first test run. It runs continuously without stopping, although it does work harder to turn the belt when the seam wraps around the wheels. Another arrangement will need to be sought other than taping the seam. The motor was held in hand during testing since it rotates with the belt so that the wires do not tangle. A vice or similar mechanism will be needed to hold the DC motors in place. Figure 7.3.2-B shows the actual prototype of the conveyor system made out of LEGOS.

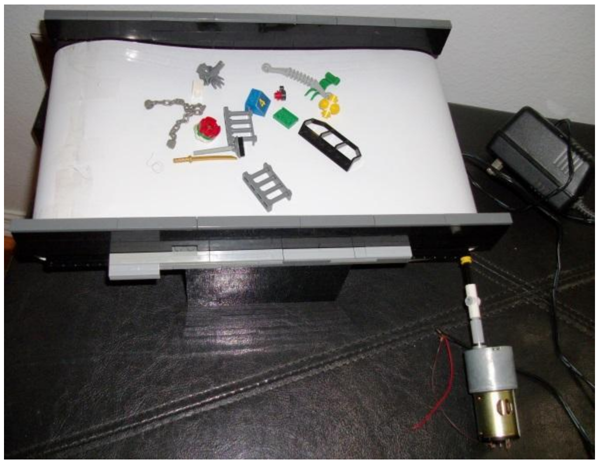


Figure 7.2.1-B Prototype conveyor belt constructed from LEGOS with DC Motor

Testing to be accomplished - Since the project has two power supplies to stem from (12 and 5V DC), we obviously can't triple the voltage of belt #1 which is 12V to 36V to make it 3x faster than belt #1. The two motors will be controlled by a pulse width modulation (PWM) signal to vary the speed of the belts. The PWM will come from the GPIO pins of the TIVA C which can be seen in figure 7.2.1-C.

Breadboard Adapter Odd- Numbered Pad GPIO and Signal Muxing														
Pin	Port	MCU Pin	Analog	Digital Function(GPIOPCTL Bit Encoding)										
				1	2	3	5	6	7	8	11	13	14	15
29	PG1	50	-	-	I2C1SDA	-	-	M0PWM5	-	-	-	-	-	EPI0S11
31	PG0	49	-	-	I2C1SCL	-	-	M0PWM4	-	-	-	-	-	EPI0S11
75	PK5	62	-	-	I2C3SDA	-	EN0LED2	M0PWM7	-	-	-	-	-	EPI0S31
77	PK4	63	-	-	I2C3SCL	-	EN0LED0	M0PWM6	-	-	-	-	-	EPI0S32
Breadboard Adapter Even – Numbered Pad GPIO and Signal Muxing														
66	PF0	42	-	-	-	-	EN0LED0	M0PWM0	-	-	-	-	SSI3XSAT1	TRD2
68	PF1	43	-	-	-	-	EN0LED2	M0PWM1	-	-	-	-	SSI3XDAT1	TRD1
70	PF2	44	-	-	-	-	-	M0PWM2	-	-	-	-	SSI3Fss	TRD0
72	PF3	46	-	-	-	-	-	M0PWM3	-	-	-	-	SSI3Clk	TRCLK

Figure 7.2.1-C PWM Pinouts of the TIVA C for DC Motor Control

There are 8 general purpose input/output pins available with pulse width modulation. To test the belt the two DC motor driver circuits will be connected to two of the PWM general purpose input/output pins. The Tiva C will generate a square wave pulse to switch between high & low (0 & 3V) respectively. The duration of the pulse will be modified with some programming until the desired speed of the two belts is achieved. The testing can be done in home by hooking up the driver circuit to a 12V power supply or the testing can be done in the Senior Design Lab by using the triple power supply provided. The AC adapter described above should be sufficient for in home testing. This will allow the conveyor belt subsystem to be tested over the holiday break.

7.2.2 Rotating Arm Testing

The stepper motor requires a specific sequence of signals to operate as intended. Testing will begin once it is completely hooked up as designed. After hooking up the stepper motor and driver circuit to the Tiva C microcontroller and power supply as shown in Figure 4.2.4-B, the code will be written. Testing will make sure the timing in the code allows for smooth operation of the stepper motor. Care will be taken to make sure delays are put in the code to accommodate for the slower electro-mechanical processes that occur in the motor. Being that the motor has 1.8° steps, it should not be a problem to get the rotating arm slide to line up with each LEGO bin. Testing will make sure it does anyway. The color sensor on the rotating arm will also be tested. It will be connected via I2C with the microcontroller before testing. The sensor has a register and some integrated features that will be read into to properly test its functionality.

7.2.3 Rotating Arm Testing

Servo Testing (Overview) - The lift arm will have the most mechanical aspects to this project. For that reason, the exact motion of the lift arm will need to be determined before connecting it to the main microcontroller. Either the motor will move with the lift arm or the motor will remain stationary and the lift arm will move upwards. Many mechanical parts will need to be utilized to accomplish this motion such as racks, pinions, axles, and gears (all LEGO parts.) Since we are Electrical Engineers, we have been having some trouble visualizing how this lift arm will operate. The initial testing of the lift arm will be done using circuitry components instead of the Tiva C. There are two things that need to be tested: speed and direction. The reference designs for testing were found online. The motor used for testing was a very similar Parallax continuous rotation motor so the circuitry provided should work for initial testing of the motor.

Servo Rotation Testing - The circuit in figure 7.2.3-A will be implemented to control the servo motor direction.

Parts List:

- TLC 555 Timer
- 68K, 47K, 33K, 10K, 1K Resistors
- BC547 Transistor
- 2 Push Switches
- Parallax Futaba S148 Continuous Rotation Servo
- 5V Power Supply

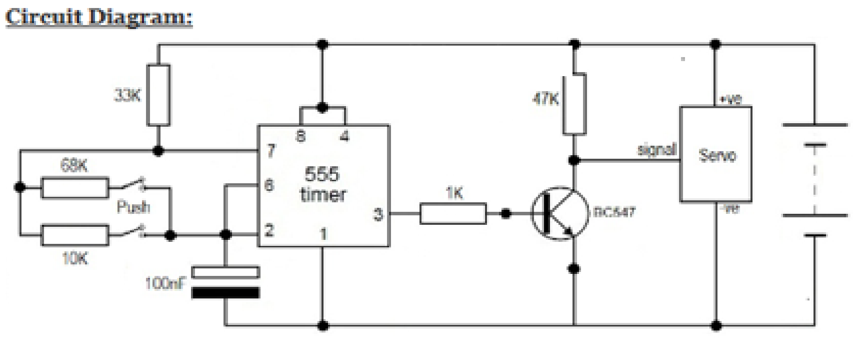


Figure 7.2.3-A Servo Motor Direction Driver Circuitry (Permission Pending)

The 555 timer acts as a stable oscillator to provide PWM to the control line. The push buttons will be used to start the motor. When the push button connected to the 10kohm resistor is pressed, the servo motor will rotate in the CW direction and when the push button connected to the 68K resistor is pressed, the motor will rotate in the CCW direction. When no buttons are pressed, no rotation occurs which makes it easier to test the lift arm if the motor is not continuously rotating without stopping. [19] [http://rookieelectronics.com/servo-tester/]

Servo Motor Speed Testing - The speed of the servo motor depends on the interval of pulse width. As shown in chapter 4, 2ms correlates to full speed rotation in the CCW direction, 1.5ms correlates to the neutral (no rotation) setting, and 1ms correlates to full speed rotation in the CW direction. A potentiometer can be used to adjust the time constant of the circuit which will change the speed of the motor. The circuit in figure 7.2.3-B will be implemented to test the speed of the motor. The potentiometer will be manually adjusted carefully to vary the speed of the motor. [20] [http://rookieelectronics.com/servo-speed-tester/]

Parts List:

- TLC 555 Timer
- 56K, 10K Resistors
- 100K Potentiometer
- N4148 Diode
- 0.1 nF Capacitor
- Parallax S148 Continuous Rotation Motor

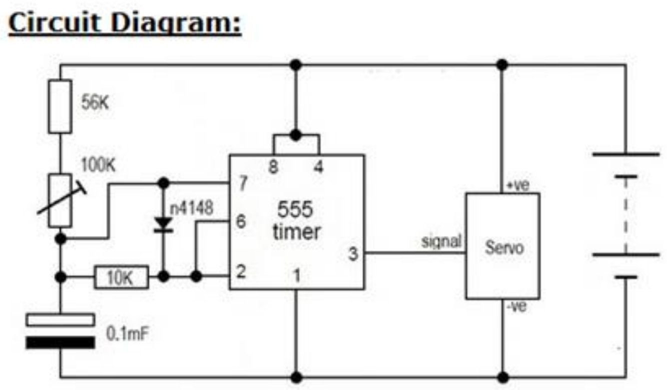


Figure 7.2.3-B Servo Motor Speed Testing (Permission Pending)

Sensor Testing- Some specifications for the EE-SX670-WR will need to be selected from Table 3.3.2 –(c) for design purposes. The power supply is specified to operate between [5 to 24V DC] +/- 10%. Since there are two outputs for the power supply, 12 and 5V, the supply voltage will be designed to operate at 10.8V (12V – 10%). As shown in figure 7.2.3-(c), the typical impedance, Z, between the output and ground is 4.7kΩ. The Tiva C supplies a logic level voltage of 3.3V. Therefore Vhigh will be designed to operate at 3.3V by selecting an appropriate resistor value, R. [21]

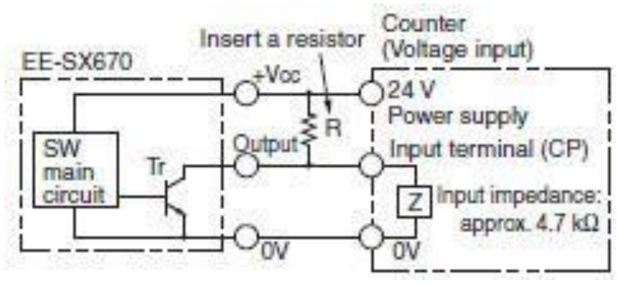


Figure: 7.2.3-C V_{high} and V_{low} Design

Given:

- $V_{cc} = 10.8$ V DC
- $V_{high} = 3.3$ V DC
- $Z = 4.7$ kΩ
- $R = ?$

Input voltage at high level:

$$V_{\ high} = \frac{Z}{(Z+R)} * (V_{cc})$$

$$3.3 = \frac{4.7k}{(4.7k+r)} * (10.8\,V)$$

R = 10.682kΩ

$$V_{\ high} = 3.29996\text{ V} \approx 3.3V$$

The input voltage and load current at the low level:

Input voltage $V_{low} \leq 0.4V$ *(Residual voltage for 40mA load current)*

Load Current (Collector) $I_C = \frac{Vcc}{R} = \frac{10.8}{10.682} = 1.011\ mA$

$1.011mA \leq 49\ mA$
[http://www.ia.omron.com/product/cautions/24/precautions_for_correct_use.html]

Conclusion - After the initial testing of the servo motor and the sensor for the Lift Arm subsystem is done and it is confirmed that the components are operating appropriately, the servo motor and sensor will be programmed using the Tiva C main microcontroller to the desired speed and rotation and connected to the GPIO pins for the final prototype.

7.3 Software Test Environment

An extremely important portion of testing is creating an accurate environment in which to test the software. In order to make sure that the testing done will accurately reflect how the project will respond in the final prototype, it is of utmost importance that the testing be done in the same kind of environment. Therefore, since all of the software will be coming from the Arduino and the Tiva-C, those will be the testing environments for all of the software. The software will be written onto the microcontrollers via a Windows 8 computer with standard specifications, and this computer will also be used to help debug the code during active testing.

7.4 Software Specific Testing

This section covers the user interface testing, image processing testing, error handling tests, hardware communication testing.

7.4.1 User Interface Testing

It's important the user interface functions behaves properly without many issues as it the only way the user can communicate with the LEGO® sorter. That why it important to test the responsiveness of the LCD screen and GUI such that the screens flow in the correct order, the bucket screen is assigned to the correct bucket, test that the interface communicated to the controller the sorting method, reading the data from the sensors, and calibrating the screen in order to interact with the GUI properly. Before any of these tests can be done first make sure the LCD screen is properly calibrated for use.

Screen flow – Testing the screen progression is simple enough. It's mainly to see if all the screens are displaying in the right resolution and if the buttons go to the correct pages. During this stage there is no other functionality like reading sensor data or sending the bucket assignments.

Supplies:

- LCD Screen
- Tiva C

- Computer
- Camera
- Beagle Bone Black

Preparation:
Attach the LCD screen to the Tiva C. Run the User interface program from the computer to the Tiva-C to begin Debugging.

Procedure:

1. Run the first screen on the display. The entire screen must fit on the screen and must not leave any space or get cut off screen. If it is not fitted properly then go back into the code and readjust the GUI window size.
2. Once the first screen is displaying properly press the start button to go to the next page. The page showing should be the set up screen where the user picks how to sort and what each bucket is going to hold. If it is not got back to the code and correct the where the button should be pointing to.
3. Then check to see if the page is the right resolution.
4. Set the sorting type and assign any value to the buckets.
5. Once everything is set up properly press the next button. The next screen should be the confirmation screen. There will be an un-editable text box that displays how the blocks will be sorted and what each bucket is going to hold.
6. Again check if the resolution of the window and adjust accordingly in the code if needed. Make sure that the text box is un-editable.
7. Next press the back button. This should lead back to set up screen.
8. Make any changes in the setup screen and press next.
9. The confirmation screen should change according to the new settings. Press the next button. The next screen should be the status screen and the final screen. Here the window should display an area for the camera screen and tabs that show the status of the other sensors, motors, and other statuses.
10. Check the resolution. As of now the camera screen should not be working so leave it be.
11. Next click on the tabs to see if the appropriate display is shown. For example pressing the Motor screen should show a graph of the motors rotation speed over time, pressing power tab shows the amount of power going into the system, etc. All graphs should be displaying the value zero, as the screen is not attached to any part of the system at this point. If not then adjust the code appropriately. If the appropriate screens are not displayed under the appropriate tabs then go back into

the code and set the tabs properly.

12. Press the pause button. A new window will display and the status screen should pause with choices to resume or quit as buttons.
13. Press resume. The window should close and return to the status screen and continue running.
14. Press pause again.
15. Press the quit button. The window should close and a new window should pop up asking for a confirmation.
16. Press no and the window should disappear returning to the status screen again.
17. Return to the confirmation window and press yes. The screen should return to the start window concluding the test.

With all the screens progressing in the right order it is ready to be integrated with system.

System Readiness – Before the user should even start interacting with the sort the system need to be ready. Therefore the start screen should disable the start button before the system is ready. The system is considered ready when the library is found and formatted correctly, and the camera is set in place. These can easily be set as flags. First we test all of the possible flag combination to make sure the button is enable when it should then we test the button by have it communicate with the camera and library. First round of testing is just going through all the combinations of the states of the library and the camera.

Procedure:

1. In the code set Camera_State and Library_Set flags according to Figure 7.4.1-A.
2. Compare results with Figure 7.4.1-A.

Camera_State	Library_Set	Enable? / Disabled?
False	False	Disabled
False	True	Disabled
True	False	Disabled
True	True	Enable

Figure 7.4.1-A

Once the behavior of the start button is set it is time for the LCD and Tiva C to be connected to the main microcontroller that will be communicating with the camera and the library. Once everything is hooked up and the Library can camera can signal the LCD perform the following procedure.

Procedure:

1. Check values of Camera_State and Library set. Both values should be false.
2. Run the code and observe debugger to confirm connection.
3. Check the LCD. The button should now be enabled.
4. Plug out camera.
5. Observe the Debugger. The Camera_State flag should now be false.
6. Check the LCD. The button should now be disabled.
7. Stop the debugger and hard wire the Library_set flag to false.
8. Redo Procedure 1 – 6. The Button should be disabled the entire time.
9. Set the Library_Set flag back to its original setting.

Setting Up – Here is where is where we assign what will go into each bucket. Here we just need to test if the information is loaded into the array properly.

Procedure:

1. Go the set up screen
2. Select the setting for the nine buckets and compare it to the information in the debugger.
3. When everything is set correctly go to the confirmation screen. All the information in the array should be displayed on the confirmation screen's text box. If it's not then that means the Library is not organized properly and some data does not have all its information.

Once it is know that all the data selected is going into the array next is to test communication of the LCD screen to the image processor.

Procedure:

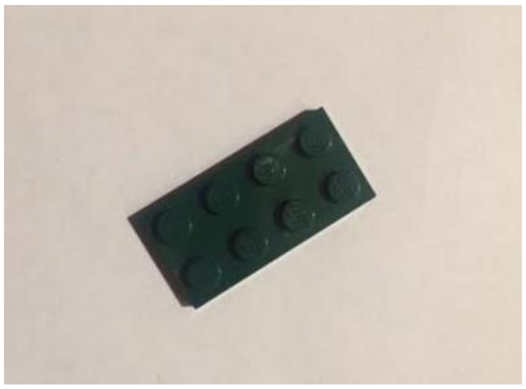
1. First set all the data for all the buckets on the set up screen.
2. Then hit next to go to the confirmation screen. Make sure to check the data in the array with a debugger.
3. Press the next button on the confirmation screen. This will send the array to the image processor.
4. Check the image processor with a debugger to see if the array data has been received. If that is not the case then there is something wrong in communication. Check the code to see if any port name has been misspelled or if the wiring of the microcontrollers is wrong.

7.4.2 Image Processing Testing

What will very likely be the most important part of testing will be testing the success of the image processing component of the system. This will consist of a number of different tests that will determine how the system will handle different scenarios that may occur within the system, as well as testing the very basic situations that should be occurring regularly.

The simplest version of this test is by simply running through Legos one at a time in the normal fashion to ensure that they are sorted into the correct buckets. These tests will have to be run for every available color that is going to be used in the project. The supplies and preparation for these tests will all be the same.

Color Test Number 1: Basic Situation Test



Supplies:
- Image Processing Chamber
- BeagleBone Black
- Computer

Preparation:
Connect and power up the BeagleBone via the computer, and then connect the BeagleBoard to the Image Processing Chamber.

Procedure:
1. Place singular Lego within the parameters of the Image Processing Chamber.
2. Run the algorithm on the BeagleBone for gathering the details of the Lego to attempt to identify the color of the Lego.

3. Check result of algorithm to confirm that it matches the color of the Lego.
4. Repeat 1-3 for other singular Legos to test all of the different colors and a wide variety of shapes.

There are multiple other situations, however, that may arise that should be accounted for as well when it comes to color testing. One such situation involves having multiple Legos on frame. In this case, the test will be run to see if all of the Legos on screen coincidentally are the same color. In this case, the multiple Legos can be dispensed into the correct bucket as normal, however if the two are different colors then they should be put into the error bucket instead.

Color Test Number 2: Multiple Legos

Procedure:

1. Place multiple Legos within the Image Processing Chamber either all of the same color, or with different colors.
2. Run the algorithm on the BeagleBone for gathering the details of the Lego to attempt to identify the color of the Lego.
3. Check the result of the algorithm. In the case where the Legos are all the same color, confirm that this is the result that was gathered by the algorithm. In the case where the Legos used were of various colors, confirm that the algorithm resulted in an error.
4. Repeat steps 1-3 for various different combinations of Legos. This includes overlapping Legos, Legos in different positions, and Legos that represent each of the colors used in the algorithm.

Another situation that could occur would be multi-colored Legos. The most obvious case of this would be the Lego people. With different outfits they could came in a myriad of different sizes. In this case, the assumption is that Lego people are in a completely different category as far as color and shape goes, and therefore do not fit in with a specific color. So the best option for this scenario would be to put the piece into the error bucket.

Color Test Number 3: Multi-Colored Legos

Procedure:

1. Place Lego(s) with multiple colors in the Image Processing Chamber.
2. Run the algorithm on the BeagleBone for gathering the details of the Lego to attempt to identify the color of the Lego.
3. Check the result of the algorithm to ensure that it is an error every time.

4. Repeat steps 1-3 for a wide variety of Legos to ensure a consistent result.

These three tests will account for virtually all of the situations that the system may encounter. With multiple tests in a wide variety of areas involving color it will help to ensure that the sorter is able to successfully separate the Legos properly based off of their color. Testing the system for its ability to recognize shape will be a different set of tests however.

In the case of shape, there will be a set library of shapes that the system will be able to recognize. In the case of a shape that is not recognized, it will be sent to the error bucket.

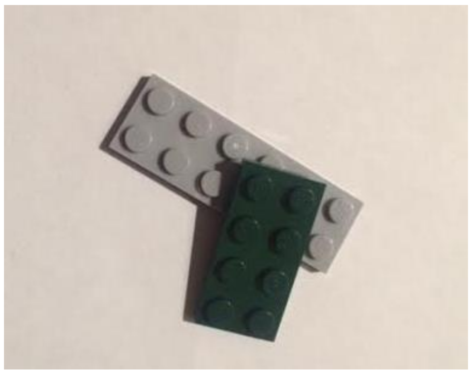
Shape Test Number 1: Basic Shape Recognition

Procedure:

1. Place a single Lego into the Image Processing Chamber.
2. Run the algorithm on the BeagleBone for shape recognition on the Lego.
3. Check the result of the algorithm to ensure that it recognizes the shape if it is in the library, or if it processes it as an error if it is not.
4. Repeat steps 1-3 for all the Legos in the library as well as a variety of those that are not in the library.

One of the additional situations that could arise is that of overlapping Legos in the image processing chamber. If two Legos overlap within the chamber, the system will likely recognize this as a single Lego. While this is certainly not an ideal situation, the algorithm should be able to simply register this situation and drop the two Legos into the error bucket.

Shape Test Number 2: Overlapping Legos

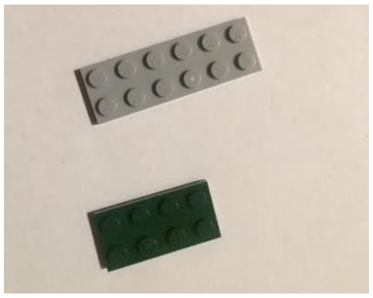


Procedure:

1. Place multiple Legos within the Image Processing Chamber, making sure that they overlap each other.
2. Run the algorithm on the BeagleBone for shape recognition on the Lego.
3. Check the result of the algorithm to ensure that it process the Legos as an error, rather than mistaking it for an actual Lego.
4. Repeat steps 1-3 multiple times to ensure consistency of results.

That covers the basic situations that could complicate the simple color and shape recognition abilities of the system, but there is also the possibility that multiple shapes could appear on the image WITHOUT overlapping. This isn't so much of a test of the shape or color recognition algorithm, but the ability of the algorithm to know WHEN to process what is on the image, or possibly how much of the image to process. In this case, the first Lego passed through will have already been processed, but the second Lego will be the one that creates the trouble. The algorithm should be able to recognize this and be able to handle it. The difficulty with this test though, is that it relies on a lot of different parts to be finished and working to properly test it. This simply means that this will be one of the last bits of testing that the project will be subjected to.

Multiple Legos (Separated) Test



Supplies:

- Image Processing Chamber
- BeagleBone Black
- Conveyor Belt
- Rotating Arm System
- Computer

Preparation:

Set up the conveyor belt, Image Processing Chamber and Rotating Arm System as it will be in the final prototype. Power up the BeagleBone and display it via the computer.

Procedure:

1. Place singular Legos along the conveyor belt. Ensure that the Legos are close enough that there will be more than one in the Image Processing Chamber at a time.
2. Begin running the system.
3. When the first Lego has reached the Image Processing Chamber, ensure that it is processed properly and is ready to be dispensed into the rotating arm system.
4. Continue to run the system as the second Lego begins to enter the chamber. Ensure that the algorithm DOES NOT begin to process the second Lego until it has received a signal from the sensor on the rotating arm system that the first Lego has left.
5. If another Lego has gotten into the chamber before the second Lego has been processed, ensure that the algorithm works correctly in ONLY processing the second Lego.
6. Repeat this process multiple times. Space the Legos out at different intervals and use multiple different types of Legos to ensure that the test occurs under multiple situations.

This series of tests should be able to effective test every aspect of the image processing algorithm that will be used in the project. This will ensure the best possible results when the project is finished to make the Lego sorter the best it can be.

7.4.3 Error Handling Testing

Majority of test plans involve similar procedure, as all that is needed are the sensors and the LCD. The following tests can only be done when all testing on the LCD screen pass. Attach motors, load sensors, and power sensors to the main controller in order to communicate to the LCD screen. Have the LCD be set to the status screen. Here were the sensor error window will appear.

Supplies:

- LCD Screen
- Tiva-c
- Beagle Bone Black
- Camera

- DC Motors
- Servo Motors
- Stepper Motors
- Motor Sensors
- Motor Drivers
- Power Sensor
- Power Supply

Motor Sensors – It is important that all the motors run at the appropriate speed. Especially the motors driving the conveyor belt at they serve as a mean to separate the LEGO® parts. This applies to all

Procedures:

This part of the procedure applies only to DC motors.

1. Run the motors. Check the read the data of the motor that is running on the LCD is showing the correct reading.
2. Increment speed of motor while monitoring it vial the LCD screen.
3. Continue until speed suddenly drops to initial speed. The monitor should reflect this change in speed.
4. Repeat this procedure except decrease the speed instead.

This part of the procedure applies to all motors

5. Now carefully hinder the motor spinning to simulate an obstruction.
6. The motor should come to a halt. Check the LCD screen. A window should appear displaying a message that there is a jam in a motor.
7. Remove the obstruction.
8. Press resume on the LCD screen. The motor should start running again.

Load Sensors – If more LEGO®s are being loaded into the sorter as it is still running then is possible that containers to soon become full. Therefore the system need to determine when a container is full and notify the user through the LCD screen that the Error bucket is full meaning that there are other buckets that are full. In this test we check if the system can recognize a full error bucket.

Procedure:

1. Hard code senor to represent an error bucket.
2. Slowly increment the amount of weight put on the sensor.
3. Monitor the LCD screen to keep track of the amount of weight put on the sensor.
4. Continue incrementing weight until window appears on screen indicating a bucket is full.

Camera – The camera must be on center in order to properly view the LEGO® part and the mirror in order to see the details of the part otherwise mistakes in comparison are more likely to happen.

Procedure:

1. Set camera to view the marker.
2. Check if LCD screen is seeing the same marker by pressing the camera tab.
3. Slowly move the camera off marker while monitoring the LCD screen.
4. Continue until an error window appears stating that the camera is off its mark.
5. Move the camera back in place. The pop up should then disappear.

Power Sensors – Because of the amount of low power devices it is important to monitor the power coming out of the power supply. If the output is too high the power supply must shut off to prevent any damage to the rest of the system.

Procedure:

1. Set a safe maximum level to test with.
2. Increase power slowly.
3. Monitor activity on LCD screen.
4. Continue until entire system shuts off.

7.4.4 Hardware Communication Testing

One of the most important things to test within the project is how well the algorithm on the BeagleBone communicates with the various hardware that will be used. Multiple test need to be done to make sure that the BeagleBone is sending signals to the hardware properly, but also that those are the RIGHT signals.

The first set of tests that will be run will be that of the conveyor belts. The conveyor belts will be fairly necessary in other parts of the testing, including portions of testing the image processing algorithm (see 7.4.2, Multiple Legos (Separated) Test). The initial tests of the conveyor belt will start off in an extremely basic manner. These first tests are simply to confirm that the signals are being sent correctly to the conveyor belt, so for these tests an extremely simple test program can be run that sets the speed of the conveyor belt and can alternate it turning on and off.

Conveyor Belt Test

Supplies:
- BeagleBone Black
- MSP430 Microcontroller
- Conveyor Belt System
- Computer

Preparation:
Connect the BeagleBone Black to the MSP430 Microcontroller in the same manner as the final prototype, and connect the MSP430 to the conveyor belt system. Power up and display the BeagleBone via the computer. Prepare a simple test program for the conveyor system and run it with the BeagleBone.

Procedure:
1. Run the sample program on the BeagleBone that starts the conveyor belts.
2. Confirm that the conveyors move at proper speeds and turn off and on correctly when instructed to.
3. Repeat this process until the conveyor belts are all behaving properly and until proper settings for the final prototype have been determined.

Another simple part that will need to be tested will be that of the lift arm. The lift arm will be running at intervals throughout the process of the Lego sorter and it needs to be tested as such. This will require another very simple test program that simply sends the required signals to the lift arm.

Lift Arm Testing

Supplies:
- BeagleBone Black
- MSP430 Microcontroller

- Lift Arm System
- Computer

Preparation:
Connect the BeagleBone Black to the MSP430 Microcontroller in the same manner as the final prototype, and connect the MSP430 to the lift arm system. Power up and display the BeagleBone via the computer. Prepare a simple test program for the conveyor system and run it with the BeagleBone.

Procedure:

1. Run the sample program on the BeagleBone that starts the conveyor belts.
2. Confirm that the lift arm is moving at an appropriate speed and at the correct intervals.
3. Repeat this process until the lift arm is behaving accurately and until a desired setting for the final prototype is arrived at.

The final hardware communications testing that needs to be done involves the rotating arm system. This will likely be the most difficult of the testing that needs to be done. It will involve communicating with the rotating arm system and keeping track of where the rotating arm is at and where it needs to go. This will require more extensive testing than that of the conveyor belts and the lift arm.

Rotating Arm System

Supplies:

- Rotating arm system
- BeagleBone Black
- MSP430 Microcontroller
- Computer

Preparation:
Connect the BeagleBone Black to both the computer and the microcontroller, and then the microcontroller to the rotating arm system as it will be in the final prototype. Prepare the BeagleBone with a sample code that will test the various necessities of the rotating arm system.

Procedure:

1. Begin running the sample code to test the rotating arm system.
2. Ensure that the rotating arm system stops at the appropriate positions, moves at an appropriate speed and starts when instructed.

3. Repeat this process for a multiple of different scenarios to test the various situations that the system may encounter in the final prototype.

These tests cover the three major hardware components in the system. Before any of these test are done the hardware tests will have been performed to ensure that the parts are already working, these tests are to confirm that the software portion of the process is communicating in the proper manner and that the signals are being communicated correctly between the BeagleBone, the microcontroller, and the hardware itself. After all of these tests have been performed on the system, all that's left will be to test everything together in the final prototype.

Final Prototype Testing

Supplies:

- All subsystems, including the Lift Arm System, the Conveyor Belt System, the Rotating Arm System, the Image Processing Chamber and the User Interface.
- Tiva-C and BeagleBone Black

Preparation:
Prepare the final prototype by piecing everything together in the same manner that the final project will be prepared. Feed conveyor belt 1 into conveyor belt 2, feed conveyor belt 2 through the image processing chamber and over the rotating arm system. Connect the BeagleBone Black and the Tiva-C to each of the subsystems and power on the machine.

Procedure:

1. Test each aspect of the User Interfaces options to ensure that they are all working at first glance. Make sure displays are working properly, that kill switch is functional, and that the interface is assigning the buckets the proper Legos.
2. Begin running the Lego sorter.
3. Pore over the sorters process in search of any sign of errors or needed changes including:
    a. Lift Arm speeds and intervals
    b. Conveyor Belt speeds and stops
    c. Rotating Arm timing
    d. Image Processing error/success ratio
4. Repeat this process THOROUGHLY to check that every aspect of the sorter is working properly.

The goal of the Lego sorter is to create a project that is highly accurate, and as efficient as possible given the constraints of its accuracy. In order to ensure this outcome, testing is of utmost importance with the project. The sorter needs to be tested thoroughly and repetitively so that no stone is unturned in making sure that the project is the best it can be. As long as these tests are adhered to, it should help to produce a superior project.

8.0 Milestone

Figure 8.0 –a shows the milestone chart for the project including deadlines for subsystem specific research, testing, and final prototype plans.

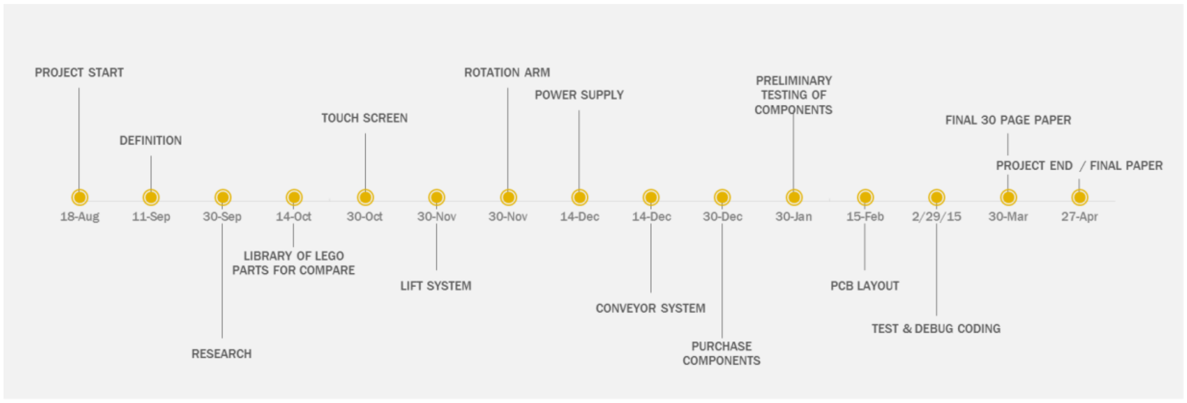


Figure 8.0 – Milestone timeline for Senior Design 1 & 2

Appendix

[1]http://www.sciencebuddies.org/science-fair-projects/project_ideas/Robotics_ServoMotors.shtml?from=Blog

[2] http://www.mocpages.com/moc.php/240340

[3] http://www.ia.omron.com/product/item/eesx1991c/index.html

[4] http://poweresim.com/index.jsp

[5] http://www.ti.com/lsds/ti/analog/webench/power-architect.page

[6] .https://www.youtube.com/watch?v=BjTD9EVs9v8

[7] .http://www.ti.com/lit/sg/sluw001e/sluw001e.pdf

[8]http://www.we-online.com/web/en/passive_components_custom_magnetics/blog_pbcm/blog_detail_electronics_in_action_45887.php

[9] https://learn.sparkfun.com/tutorials/switch-basics

[10] http://www.digikey.com/product-detail/en/204-213ST/CT204213ST-ND/267312

[11] http://www.digikey.com/product-detail/en/2-435469-1/2-435469-1-ND/1202087

[12] http://www.sager.com/2-435469-1-657566.html

[13] https://www.servocity.com/html/how_do_servos_work_.html#.VHVGAclWV_w

[14] http://www.nskelectronics.com/slot_sensor.html

[15] https://www.youtube.com/watch?v=iDS3P1WsWq0

[16] http://beagleboard.org/Community/Forums

[17] http://beagleboard.org/support/faq#Serial

[18] https://github.com/alexanderhiam/PyBBIO/wiki/SPI

[19] https://github.com/alexanderhiam/PyBBIO/wiki/SPI

[20] http://rookieelectronics.com/servo-speed-tester/

[21] http://www.ia.omron.com/product/cautions/24/precautions_for_correct_use.html