Final Report
Pre-tinning Process

May 8th, 2015

Mechatronic Team G

Eric Newhall
Guillermo Manuel Cidre
Michael O'Connor
Christian Heaney-Secord

Executive Summary

As sensors, actuators, and integrated circuits become more precise and accessible many opportunities arise to either assist with or automate processes. One such process is the pre-tinning procedure currently performed by Kennametal. This is a meticulous process in which workers must carefully sort and handle small carbide saw tips. A mechatronic system has the potential to automate this task, freeing up workers to help in other areas and relieving them of any strain injuries they may be susceptible to from performing this repetitive motion. Our team designed a robust and durable system capable of taking in these carbide saw tips, placing them in neat rows on a tray, extruding flux on to each part, and placing either one or two pieces of wire on each part. We also created a method to feed wire into the system and cut it, removing the need for workers to have to handle these small pieces of wire. By utilizing stepper motors, DC motors, servos, and an array of sensors we were able to demonstrate the ability to handle multiple part sizes, orient the parts correctly, and place appropriate flux and wire on each part. While there are still refinements that can be made to our system we have successfully created a proof of concept prototype that lays the groundwork for automating the pre-tinning process.

Table of Contents

1. Project Description……4
2. Design Requirements……4
3. Functional Architecture……5
4. Design Concepts……6
    4.1. Part Separator and Part Orientator……6
    4.2 Computer Vision and Part Flipper……7
    4.3 Part Placer……7
    4.4 Flux Extruder……7
    4.5 Wire Feeder and Wire Cutter……8
    4.6 Wire Storer and Wire Placer……8
5. Cyberphysical architecture……9
6. System description and evaluation……10
    6.1 System/subsystem descriptions/depictions……10
        6.1.1 Part Separator……10
        6.1.2 Vision Processing……10
        6.1.3 Part Placer……10
        6.1.4 Tray Positioner……11
        6.1.5 Wire Cutter……11
        6.1.6 Flux and Wire Dispenser……12
        6.1.7 Software……13
    6.2 Modeling, analysis, and testing……14
    6.3 Performance Evaluation……15
    6.4 System Highlights……15
7. Project management……16
    7.1 Schedule……16
    7.2 Budget……16
    7.3 Risk management……18
8. Conclusions……18
    8.1 Lessons learned……18
    8.2 What would you do differently……19
    8.3 Future work……19
9. References……20
10. Appendix A - Final Arduino Code……21

1. Project Description

Kennametal is an American supplier of tooling and industrial materials that has a plant in Victoria, British Columbia, Canada that manufactures saw tips and prepares them for attachment to a handle through a process known as pre-tinning. In this process a dab of flux is applied to the surface of a carbide saw tip and then either one or two silver wires are carefully placed in the flux, oriented along the major diagonal of the saw tip. Once this process is complete, tinning can occur, in which the wires melt onto the saw tip in an oven.

The process of pre-tinning saw tips is laborious and meticulous because of the size of the saw tips and wires that are handled. This work also can have negative health effects as the repetitive motion can cause strain injuries. Our objective is to design and build a mechatronic system capable of pre-tinning saw tips, removing the need for this task to be accomplished manually. In doing so the potential for worker injury will be greatly reduced, the small wires can be placed with greater accuracy, and there is the potential that the machine may be able to perform the task significantly faster than a worker.

2. Design Requirements

a. Explicit Design Requirements:

- 20 parts must be processed in <=5 minutes
- Flux dab must be centered on the part-top centroid and cover ¼ to ⅓ of the part-top's surface area
- Must be able to handle at least two different part sizes
- Machine fits within 2'x2'x2'
- The two wires must be placed side by side in the flux dab such that: a) their orientation is along the major diagonal to w/in 5°; b) there is no space between them laterally; c) their longitudinal misalignment is <=10%
- The parts will be deposited on a flat output tray in 4 rows of 5 parts each, with the part-tops facing up. Part edges should be at least ½" away from the tray edge, and there should be at least ¼" spacing between the edges of adjacent parts.
- No more than 5% of wires detached or wrongly positioned

b. Implied Design Requirements:

- Machine is durable enough such that it can endure a harsh factory environment
- The machine will have a robust design to accommodate any changes or additional requirements of the pre-tinning process
- Design is simple enough such that the basics of the machine will be understood by an assembly worker

- Design will allow easy access for a worker to fix the machine in the event of failure
- Designed such that if a local failure occurs within the machine the remaining functions of the machine will not be compromised

c. Design Requirements for Coolness Factor:

- Must be able to feed lengths of wire into the system
- Should feed the same length of wire each time
- Must be able to fully and consistently cut wire

3. Functional Architecture

A part enters our system through a hopper attached to the side of the assembly. The part then leaves the hopper and arrives at the camera in one of four possible orientations. Vision processing determines both when the part has arrived and its orientation. When the part arrives the part hopper must be told to hold the remaining parts while the part is being handled. Also, the part placer must receive part orientation from vision processing in order to move the part correctly. The part placer must command the tray positioner into the proper position since the part placer only has freedom to move in the x axis and the tray can only move in the y axis. After all twenty parts are placed on the tray, the wire feeder system is continually feeds and cuts pieces of wire and loads them into the revolver. Once all twenty parts are in place the part placer notifies the flux and wire dispenser that the part is available. The flux and wire dispenser also can only move along the x axis so it too must command the tray positioner to reposition below it. Once all twenty pieces of wire are cut and loaded, the subsystem positions itself atop each part and dispenses flux along with either one or two wires, depending on the part size.

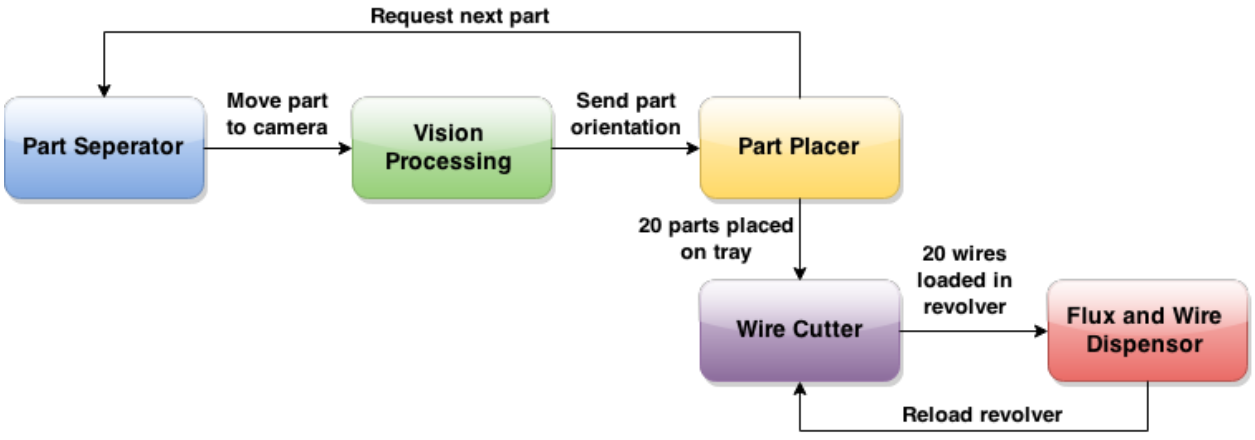

Figure 1: Visual of Functional Architecture

4. Design concepts

4.1 Part Separator and Part Orientor

Our initial design concept for our part separator consisted of hopper with a small hole at that bottom whose dimensions were such that parts would only be able to fall through it in one of four orientations. We did quite a bit of testing on various hole sizes so that we could make the hole as large as possible while only allowing parts to fall through in one of the four desired orientations as you can see in Figure 2 on the right. We planned on agitating the hopper using a vibration motor in order to constantly shift the parts in the hopper until one fell through the hole. However, after some preliminary testing with some 3D-printed hoppers we discovered that we could not efficiently sort the parts using this method and decided to abandon the design concept. You can see our prototype designs in Figure 3**.** We then did some testing with parts sliding down along the wall of an aluminum L-bracket. We had a block a set distance away from the wall so that parts would not hit the block if they were coming down in one of our desired four orientations but would hit the edge of the block if it was in any other orientation. The block would then rotate about 45 degrees, rotate back, and then the individual parts would go down the slide. After further testing, we found that a constantly spinning gear was more efficient and had a higher success rate of getting parts through than the rotating block did. The gear operated similarly to the block in that parts in one of the four desired orientations would not hit the gear, but parts in any other orientation would hit the gear. The gear would then knock the individual parts back up (rotating them a random amount) until the part fell through in one of the correct four orientations. The success of this subassembly was contingent on having a single part be processed at a time. In order to achieve this we chose to apply a similar design that we had implemented with the wire feeder. We used two rotating pulleys that were covered in a layer of foam to pull parts out of the hopper in a controlled manner. However we had some small issues with this design in that parts would jam if too many were loaded. So, we iterated on that design and replaced one of the gears with an electromagnet. The electromagnet would turn on and off very quickly to slowly move parts down the slide.

Figure 2: Iterations of hopper outlet holes

Figure 3: 3D-printed hopper prototypes

4.2 Computer Vision and Part Flipper

Despite quite a bit of brainstorming we could not think of an easy way to identify which of the four orientations the parts came down the slide in. So, despite the difficulty of its implementation we resorted to computer vision in order to tell us which of the four orientations the parts were in. In order to re-orientate the parts if they were upside down we planned to grab the part and insert it into a horizontal U shaped conveyor belt so that the part would come out right side up. However, after thinking about the problem more we refined this idea into having a servo flip the part 180 degrees if it was upside down. The servo has an acrylic plate attached to it with an electromagnet on the bottom of the plate that holds the part in place as it flips 180 degrees. When it is flipped the part is dropped onto a sheet metal slide. If the part does not need to be flipped, the part flipper slants downwards and the part slides down onto the sheet metal slide. Our original plan for getting parts at a precise location in front of the camera involved having parts come down a conveyor belt and having the conveyor belt drop parts at the appropriate location. Unfortunately, this design concept was dependent on the sorting method we had originally planned on implementing in order to be functional. So instead we had parts slide onto the part flipper. The part flipper has a stopper attached to a servo that stops the part at the appropriate location in front of the camera. The part flipper is also slanted slightly to ensure that parts stay against the back wall of the part flipper as they slide down.

4.3 Part Placer

The part placer is one of the few subassemblies whose final design looks very similar to our initial design. In our design we actuate the part placer along the vertical axis using a rack and pinion. The part placer is attached to a rail system that moves the part placer along the x-axis. In our initial iteration we used a rather large electromagnet which caused our motor to backdrive after being actuated. You can see the part placer with the oversized electromagnet in Figure 4. This was problematic because it made it hard to control the exact positon of the part. In order to remedy this issue we simply replaced the large electromagnet with a smaller electromagnet and replaced the DC motor with a servo.

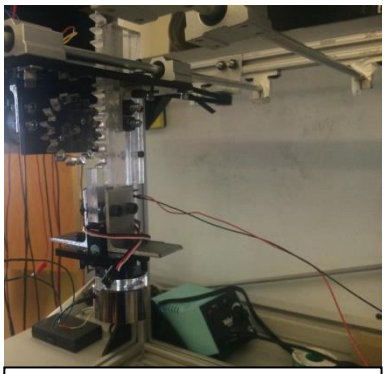

Figure 4: Original part placer assembly

4.4 Flux Extruder

Similarly to the part placer, we attached our flux extruder to a rail system that moves the subassembly along the x-axis. We planned to extrude from our flux extruder by using a stepper motor with a lead screw attached to its shaft. The plan was that it would allow for complete control of how far the actuator moves while also providing a significant amount of force to

overcome the high viscosity of the flux. However, after using the rack and pinion for the part placer and seeing the success of that design we chose to implement that design for our flux extruder as well. We felt that this was a good decision because there were minimal issues in its implementation.

4.5 Wire Feeder and Wire Cutter

We were one of the few teams that chose to cut their own wire. This decision created a significant amount of extra work for us. In our initial design we planned on unspooling the wire using a motor and feeding the wire into a channel where it would be cut. The wire would then be cut by a blade attached to a solenoid. We soon discovered that the wire was too stiff to feed itself through a channel if we unwound the spool and too hard to be cut by a blade attached to a solenoid, leading us to reconsider our design. We ultimately decided on having the wire be pulled through a series of rotating pulleys that were covered in a high friction material. After being fed into the appropriate location we had one handle of the bolt cutters bolted to a sheet metal plate that was attached to the side of our 80/20 frame. See Figure 5. A big issue that we did not foresee in this design is that the bolt cutters' head displaces fairly significantly while being actuated when one side is fixed. This was problematic because when the wire was fed to the bolt cutters, the bolt cutters' head displaced and shortened the initial amount of wire that was fed into the bolt cutters before cutting the wire. So, we had to modify our design so that both handles of the bolt cutters were actuated at the same time. We accomplished this by mounting a motor to an L-bracket at the bottom of the 80/20 frame and running a string from both handles of the bolt cutters to the motor.

Figure 5: Initial wire cutter configuration

4.6 Wire Storer and Wire Placer

In our initial design we planned to simply push our wire out of the channel that they were put in after being cut so that the cut wire was positioned atop of the dab of flux on the saw tip. Our second design iteration consisted of us having the cut wires being fed down a tube with a servo attached at the end. After the wire is cut, the rail system would move over to the appropriate location for the wire to be dropped and the servo would remove a cover at the end of the tube. However, we felt that this design concept would consume too much into our five minute time constraint. So we decided that we wanted to cut all of our wire at once and store each piece of wire until it was ready to be used. This way we could operate both the part orientation/part placement subassemblies and the wire cutting subassembly in parallel. This change would have helped save a lot of time except we were not able to run both systems in

parallel because it was too difficult from a software standpoint. After we came up with the initial concept of using a revolver to store the wire and a tube to place the wire we did not see many changes in those two subassembly designs. However, there were multiple iterations in order to get the geometries lined up because they needed to be very precise in order for the wire to enter and exit each checkpoint smoothly. The one small change that we decided to implement was having a funnel at the exit of the revolver to smooth out the transition from the revolver to the tube used to place the wire.

5. Cyberphysical architecture

The primary controller for the pretinning machine is the Arduino Microprocessor which responsible for interpreting all feedback from sensors and sending all commands to motors and other physical devices. The Raspberry Pi Microcomputer is dedicated entirely to image processing for the camera. The Raspberry Pi sends information about part location and orientation which is derived from image processing back to the Arduino so that correct movements can be made.

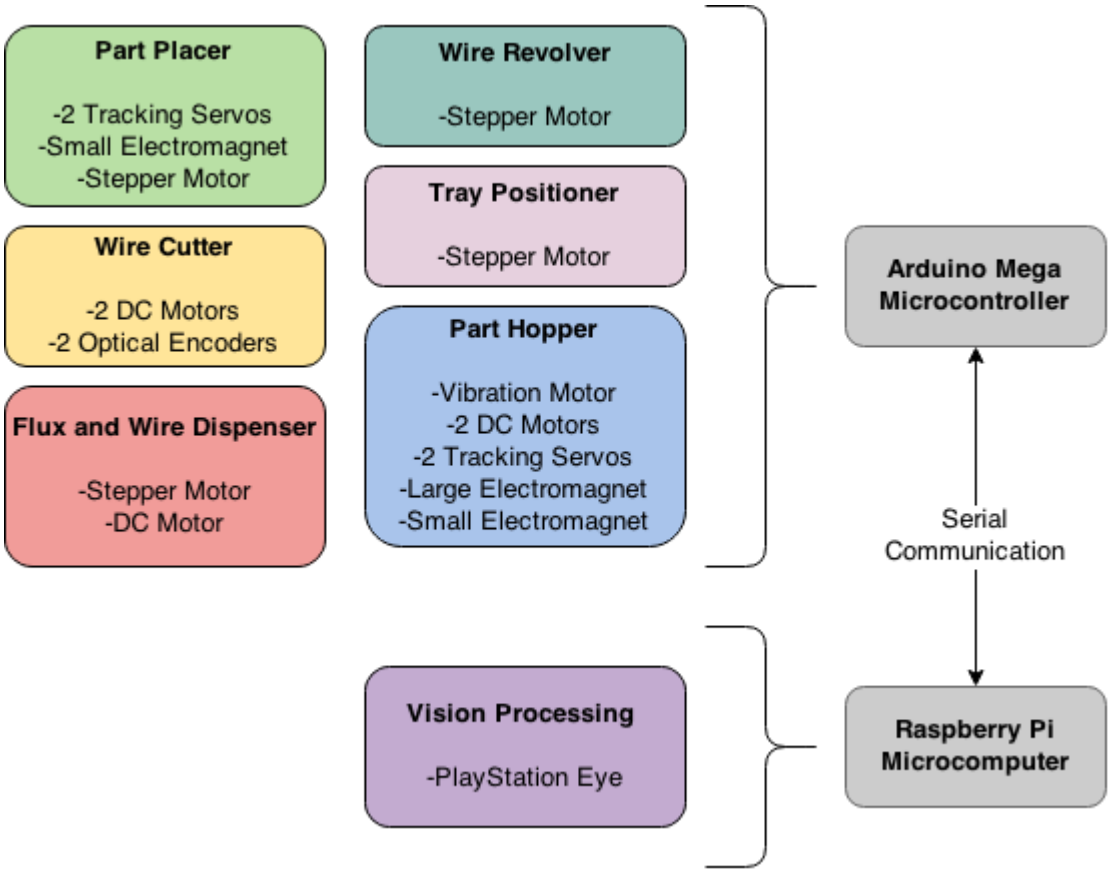

Figure 6: Overview of Cyberphysical Architecture

6. System description and evaluation

6.1 System/subsystem descriptions/depictions

6.1.1 Part Separator

Our part separator consists of two main components; the hopper and the part reorientation. The part hopper system consists of a bin make of laser cut acrylic. At the bottom of this bin is a spinning disk and a large electromagnet. The electromagnet pulses between on and off to allow for parts to leave the hopper one at a time. Then each part slides down a channel at the end of which is our part flipper and camera. The width of the channel can be adjusted to allow for both small and large parts to be processed. The channel has another spinning gear in it to ensure the parts arrive in front of the camera in only four different possible orientations.

6.1.2 Vision Processing

All of the system's vision processing is performed by a Raspberry Pi connected to a PlayStation Eye. The camera takes a side profile of the part and compares it to one of four different template images. Each template returns a rating based on how good of a match the part is to the template. The camera returns information on what match is the best about once a second to the Arduino as a serial message. If no match is good enough then the camera will return a zero. If a match is found then the camera will tell the servos on the part flipper to go to viewing position so a more accurate picture can be taken. The more accurate message is sent to the Arduino to tell it whether or not to flip and or rotate the part. If the part needs to be flipped then the electromagnet under the part flipper will be turned on and the part flipper servo will flip the part and then release the magnet. Otherwise the part flipper servo will elevate slowly and allow the part to slide off without flipping it. Rotating the part is handled by the part placer.

Figure 7: Template images for side profiles of part

6.1.3 Part Placer

The Part Placer system is a platform that is attached to two of the four rails on the top of the frame. The platform mount consists of mostly laser cut acrylic with machined aluminum that is made to hold on to the rails. On top of the stand, there are two aluminum brackets facing the end of the rails. On one end of the frame, there is a stepper motor hooked up with a timing pulley on an aluminum bracket. On the other end, there is only idler pulley attached to an aluminum bracket. A timing belt is hooked up from one aluminum bracket in the stand to the stepper motor

on the end of the frame, to the idler pulley in the other end of the frame, and to the other aluminum bracket in the stand in order to achieve movement along the rails. In the center of the platform, one of the two servo motors is attached to a spur gear. This gear meshes with a vertical rack that is attached to the arm of the system. This way, the servo motor is able to move the arm up and down as shown in Figure 8. The rack and pinion is made of laser-cut acrylic. On the end of the arm we have another servo motor attached to the stand. Other than that, the servo motor has an electromagnet, which will pick up the parts, attached to it. This way we can achieve rotation on the subsystem if it is in the wrong orientation.

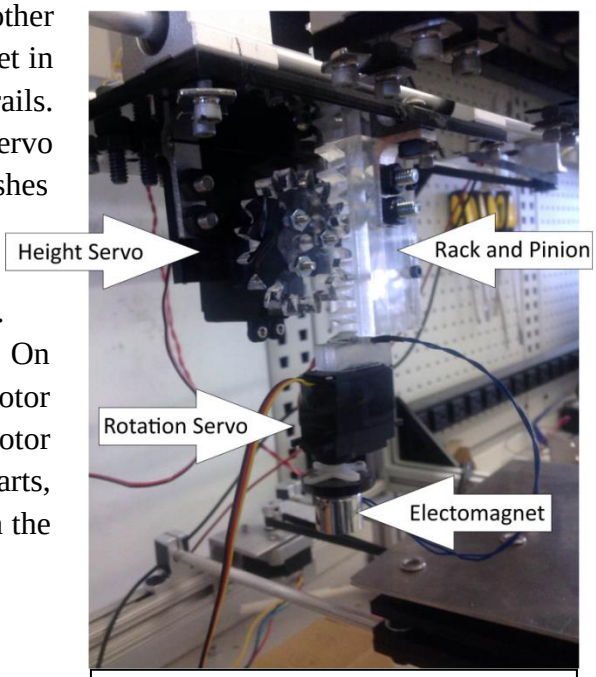

Figure 8: Part Placer assembly

6.1.4 Tray Positioner

The tray system works with the part placer and flux and wire systems to allow for movement in two axes. The tray system consists of only a single stepper motor which drives a timing belt to move the tray. The aluminum tray which the parts rest on lies above an acrylic tray which is connected to the timing belt on each end. The stepper motor is controlled by a stepper motor driver and the Arduino Stepper class which allows for precise control in the movements of the tray. Currently the tray is operating at full steps, but if greater precision becomes need the step size can be decreased.

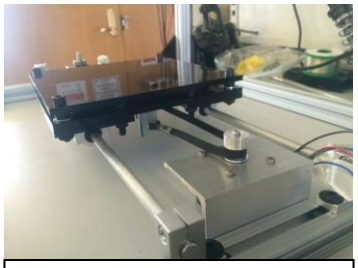

Figure 9: Tray Positioner

6.1.5 Wire Cutter

Our team chose to cut wires instead of using pre-cut wires. The wire was fed into a feeder that could turn four wheels in order to push the wire forwards into the cutter blades. The feeder used optical encoders in order to cut the same length of wire for each part. The wires were cut by a pair of modified bolt cutter. The arms of the bolt cutter were connected to a winch that would pull the two arms together until the wire was cut then release the arms to allow for the next length of wire to be moved in front of the cutter's blades. The cut pieces of wire would fall into a revolver that had twenty individual slots to hold cut wire pieces. After each cut the revolver would turn to the next slot until all twenty slots had been filled.

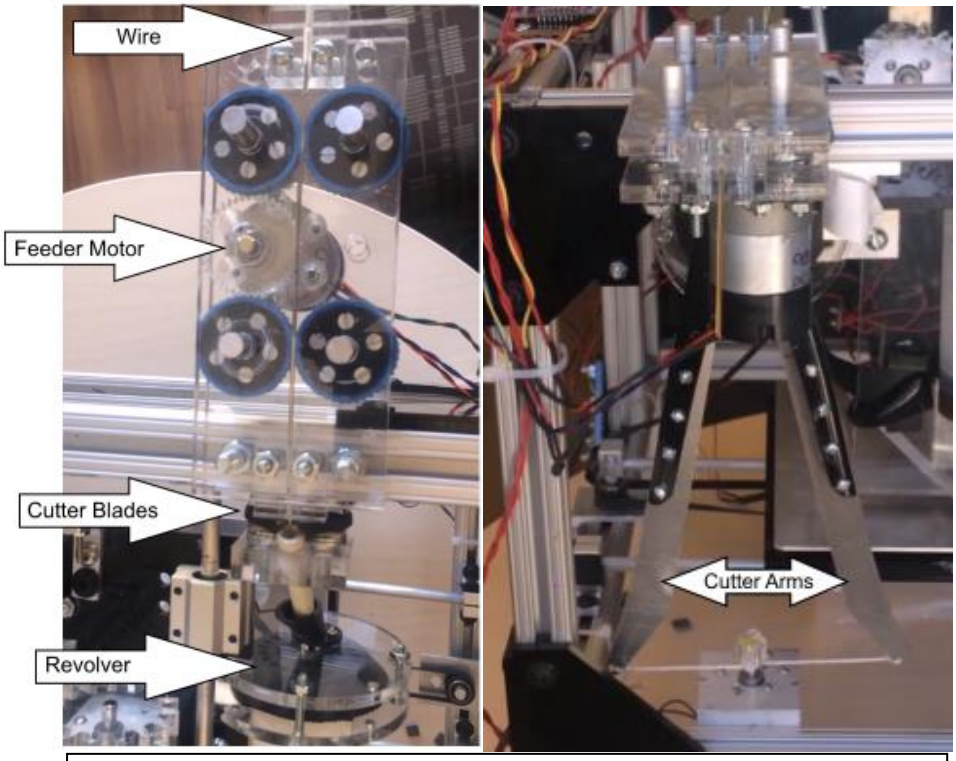

Figure 10: Wire feeder and wire cutter systems

6.1.6 Flux and Wire Dispenser

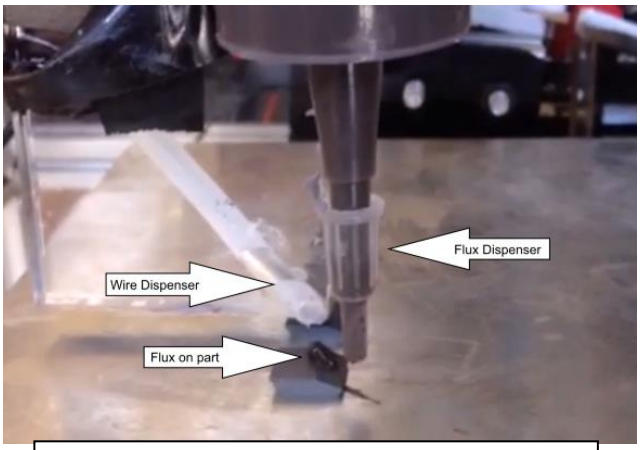

Figure 11: Depiction of flux extrusion and wire placement

The flux and wire dispensers are both located on the same rail system so that they can both be applied to each part at the same time. The flux is dispensed first using a rack and pinion connected to a plastic syringe. A DC motor pushes the syringe then waits a brief time for the flux to leave. Then the tray system is repositioned so that the wire revolver is aligned with the part. The revolver rotates to allow for a cut wire to leave and fall onto the part. The flux on that part holds the wire in place. This is repeated for all twenty parts. In the case of the large parts two wires will be needed for each part. The revolver only holds twenty parts so more will have to be cut after the first twenty are placed. The dispenser returns to the wire cutter system to obtain the remaining twenty then moves back to where it left off to complete the task.

6.1.7 Software

The software for our design was implemented across the Arduino Mega and the Raspberry Pi. For the Arduino Mega, we implemented a round-robin multitasking schedule that would run each of the state diagrams for each subsystem. Some are given below. In addition, we used libraries that were optimized for motor control for the Arduino Mega. Hence, there was almost no coding for the motor control themselves. But due to restrictions imposed by the libraries used, some of states specified actually take more than one state to do. But nevertheless, it does exactly what each of those states do. In order to make the code readable, to be able to better determine if an error was caused by the software or hardware, and to better organize the code, structures and functions were implemented that would transform motor related code into readable one-liners. Also, each state diagram code was separated into their own respective files and their own functions for better navigation and simplicity. The Raspberry Pi implemented vision processing through the use of a package called OpenCV 3.0 beta. In addition, the code was implemented to multitask between fetching the images and processing the image through the use of threads. By doing this, the fetching process of the code wouldn't be dragged behind by the processing of the image. In other words, we would always process the latest image.

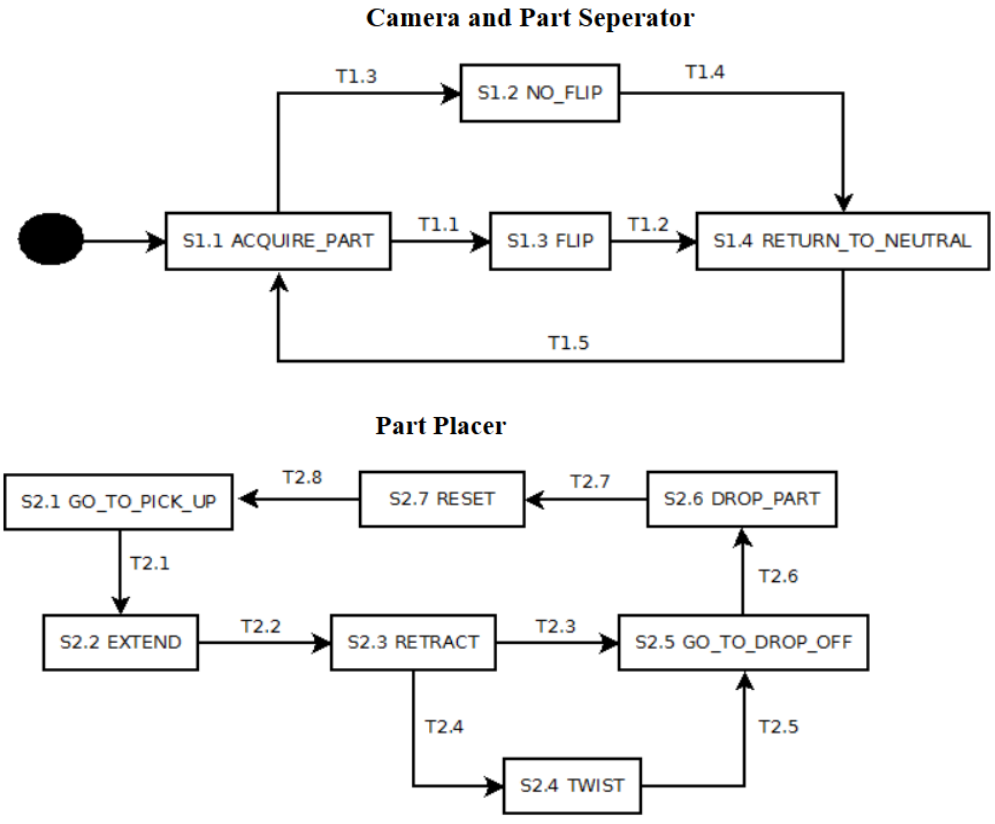

Figure 12: Examples of round-robin multitasking schedule used

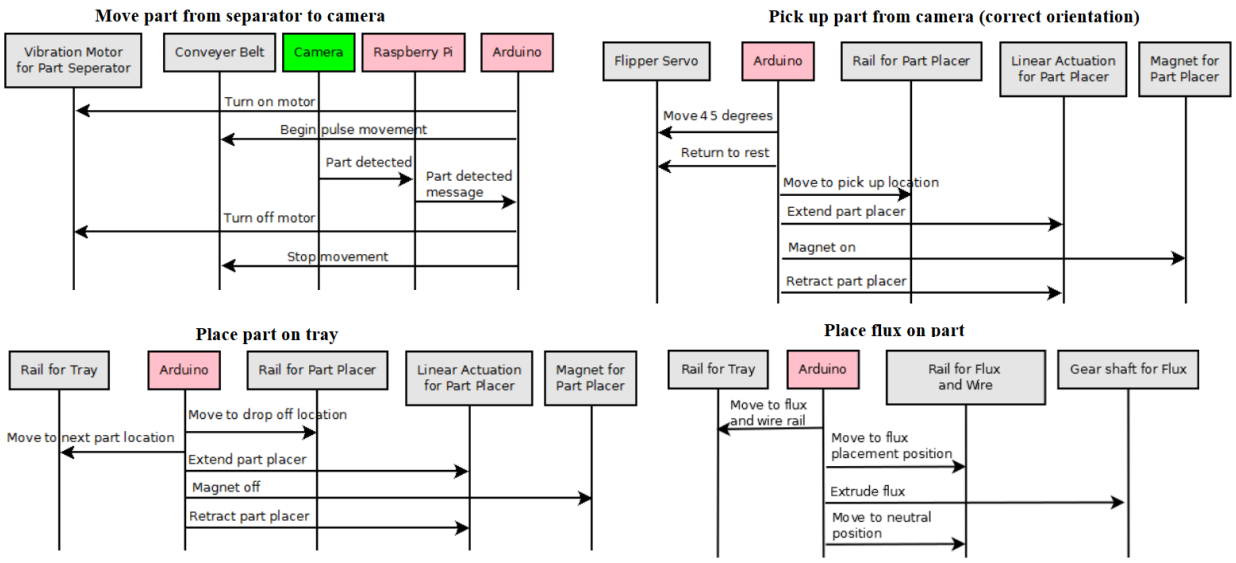

Figure 13: State diagrams used to define operations in each state

6.2 Modeling, analysis, and testing

Power consumption was a recurring issue during development. The initial lab provided power supply could only output up to 0.5 amps. Each stepper or DC motor could take up to two amps and the electromagnets we used could also take up to two amps. It was clear that we had to move to a supply that could output more current. The 12 volt supply we switched to provided up to five amps. When we were running the wire cutter and part placer concurrently it was possible for two stepper motors, 2 DC motor, and an electromagnet to be running at the same time. That was enough current to black out the system. Without the wire cutter at most one stepper, 2 DC motors, and an electromagnet could be running at once. The stepper was limited to 0.5 amps, the DC were running at reduced voltage and drew around one amp each and the magnet was drawing 2 amps of power. Total this was 4.5 amps of power which was just below the max amount of power that we could output.

Timing analysis was also taken into consideration to see whether or not the 5 minute deadline could be met. The vision processing system was at first a large delay in the system but we were able to have the image capturing software run in parallel with the image processing software. After this optimization the system would output the orientation at around 1 Hz. The time to move the parts from the camera to the tray and move the part placer back was about 8.5 seconds. In total moving 20 parts would take about 3 minutes. If the parts could always be ready for the part placer when it returned and if the wire could be cut in parallel then there would be 2 minutes left to put the flux and wire on the parts. We believe that 2 minutes would be enough time however the assumption we made were not true. The wire could not be cut in

parallel due to power requirements and the parts were not always ready for the part placer due to inconsistencies in the rate at which parts could reliably be sent one at a time.

6.3 Performance Evaluation

With regards to the explicit and implicit requirements of the system, our machine fulfills the majority of the requirements but falls short in a few sections. While we originally believed that we would be able to fully process all 20 parts in 5 minutes, we were unable to accomplish this goal due to issues with running several systems in parallel, however, that functionality still had the potential to be implemented. We were able to fulfill the requirement of placing a dab of flux on the center of each part and were able to extrude a consistent amount of flux on each part. While we had trouble ensuring that the pieces of wire landed on each part, we did make sure that when they did, they rested on the part at a 45 degree angle. Our system proved capable of laying out all 20 pieces in 4 rows of 5 with all pieces face up and in the same orientation thanks to our vision processing and part re-orienting systems. We designed our system to meet the implicit requirements of being robust and durable. Constructed from an aluminum 80/20 frame, our device is sturdy and durable. If a part needed to be manipulated in a new way, the flux and wire sub-system could easily be replaced or the frame could be expanded to allow for a third upper rail system which could hold a new sub-system. When evaluating the success of our coolness factor, we proved that we were able to feed and cut a consistent amount of wire to be placed on the parts.

6.4 System Highlights

While our system has all the components and subsystems needed to fully complete the task, certain processes are able to function smoother than others. Some examples of processes that function well are our part re-orientor and part placer systems. Although we had a few issues with our camera being able to recognize different orientation when it wasn't carefully calibrated, once correctly calibrated we were able to get all 20 parts on the tray in the exact same orientation. Our actuators that controlled flipping and rotating the part as well as placing it on the tray all functioned to the desired degree of precision. Our vision processing was one weak point of our system as its ability to correctly detect parts greatly decreased with changing lighting conditions. Another weak point of our system was our wire placing system as we had jamming issues on several occasions and were never able to properly calibrate our system to ensure the wire pieces landed on the parts. We believe that increasing the fidelity of our part orientation detection system could be achieved by creating a housing around our camera and the station where the part is viewed. We also think that a sturdier cutting mechanism might alleviate our jamming problems and that further calibration could ensure that the wire ends up on the parts.

7. Project management

7.1 Schedule

Below is a chart for the projected schedule our team planned to pursue as of mid-semester break. While we felt that this was originally a reasonable schedule, because of issues with our wire cutting and placing system as well as issues with our part hopper, we fell slightly behind schedule. Thankfully we had two weeks scheduled for subsystem cleanup and fine tuning. While we could not follow our original schedule we still completed our system on time thanks to those two buffer weeks.

Table 1: Mid-semester projected schedule

March 11	Flux extrusion and wire cutting	
March 18	Mount assembly for flux and wire placement	System Demo #3
March 25	Develop hopper configuration	System Demo #4
April 1	Integrate vision processing and conveyor belt into system	System Demo #5
April 8	Implement part reorientor	System Demo #6
April 15	Subsystem cleanup	System Demo #7
April 22	Calibrate system	Final System Demo
April 29	Fine tune / speed up assembly	System Demo encore
May 4	Prepare for public presentation	
May 8	Finish final report	
May 11	Add material to website and clean up lab station	

7.2 Budget

Overall we feel that we did a fair job of budgeting over the course of the semester. While we did come in $100 over the original $700 spending limit, this can be largely attributed to the $142 unexpectedly spend on shipping costs. We also spent an extra $127 on a backup Arduino Mega and stepper motor drivers in order to mitigate risk. The table on the next page shows the parts that we ordered over the course of the semester and their respective prices.

Table 2: Items purchased

Description	Quantity	Unit Price	Shipping	Price
Frame and Rail System				
Linear Rail Shaft Guide (8mm)	12	$3.95	$7.04	$54.44
Linear Bearing Platform (Small 8mm)	12	$6.95	$0.00	$83.40
Aluminum 1" 80-20 (4ft)	6	$14.20	$21.45	$106.65
Steel End-Feed Fastener (4 pack)	10	$2.30	$0.00	$23.00
8mm Aluminum Rod (6ft)	3	$4.88	$7.04	$21.68
Timing Belt GT2 Profile	3	$9.95	$7.04	$36.89
Aluminum GT2 Timing Pulley	3	$7.95	$0.00	$23.85
Belt Clamp Crimp Style	8	$0.60	$16.77	$21.57
Idler pulley kit	2	$5.75	$7.42	$18.92
Motors and Electronics				
Stepper Motor: Bipolar, 200 Steps/Rev	4	$16.95	$1.99	$69.79
Sucked Electric Lifting Magnet Electromagnet	1	$12.31	$0.00	$12.31
Arduino Mega	2	$45.95	$4.89	$96.79
A4988 Stepper Motor Driver Carrier, Black Edition	10	$7.49	$6.48	$81.38
Miscellaneous Hardware				
Colored Acrylic 12"x24" (black)	2	$18.80	$21.45	$59.05
Large Syringe	1	$6.00	$0.00	$6.00
Steel Socket Head Cap Screw M4 Thread	1	$4.50	$5.33	$9.83
Radial Ball Bearing 608ZZ - Set of 4	1	$6.95	$0.00	$6.95
Set Screw Shaft Collar, 8mm	1	$1.95	$0.00	$1.95
Black-Oxide Head Cap Screw M5 14mm length	1	$11.81	$0.94	$12.75
T-nut (25 pack)	2	$4.95	$0.00	$9.90
Steel Hex Nut M5 (100 pack)	1	$1.73	$0.00	$1.73
1/4"-20 Thread, 3/4" Length Button-head screw	1	$7.63	$1.92	$9.55
1/4"-20 Thread, 4" Long, Threaded Rod	1	$1.75	$6.57	$8.32
1/4"-20 Thread, 8" Long, Threaded Rod	1	$2.24	$5.26	$7.50
6x6.35mm CNC Motor Shaft Coupler	2	$7.93	$0.00	$15.86
			Total	**$800.06**

The Table 3 is a list of materials that were either borrowed from the lab or were privately owned by a team member and used for the project. The cost of each item was estimated in order estimate the total value of extra components that were not purchased through the class.

Table 3: Estimated cost of borrowed parts

Description	Quantity	Estimated Cost	Estimated Price
DC Gearmotor with Encoder	4	$20	$80
Stepper Motor	1	$20	$20
Standard Servo	3	$10	$30
Small Servo	1	$5	$5
12V power supply	1	$10	$10
5V power supply	1	$5	$5
Playstation Eye	1	$10	$10
Raspberry Pi	1	$35	$35
DC Motor Driver	3	$20	$60
Stepper Motor Driver	4	$6	$24
		Estimated Total	**$279**

7.3 Risk management

Our team took several steps in order to mitigate risks inherent to our project. In order to ensure that we complete the task in the allotted amount of time we attempted to complete several tasks in parallel. Specifically, we had planned to feed and cut pieces of wire while placing all 20 parts on the tray. Unfortunately, we were unable to work these functions in parallel. One problem that we witnessed other teams experiencing and that we experienced ourselves was accidentally burning out electronics. To prevent an accident like this from setting us back we ordered an extra Arduino Mega and extra stepper motor drivers, so of which we ended up needing. We also mitigated risk within our schedule. We originally planned to finish our prototyping two weeks before the final system demo in order to have time to refine our system. These two weeks acted as a cushion for us when prototyping certain subsystems took longer than expected.

8. Conclusions

8.1 Lessons learned

There were several important lessons learned that would help everyone in the group if we were to construct another mechatronic system. First of all, make sure that the mechanical

engineers and software engineers are in constant communication with each other. Often both would be waiting on the other to complete a task before they could continue. Second, using built in libraries for the Arduino can be very powerful. When we were completing motor lab we did not use libraries for the stepper motors, servos, or encoders. Later in the course we were using built in libraries for all of these including a custom stepper library that allowed steppers to run a set distance while other code ran in parallel. Finally, we learned that vision processing is not very reliable when searching for multiple similar objects. Other teams had success with different methods of detecting parts without using a camera where as our team spent a large amount of time on vision software that in the end was not very reliable.

8.2 What would you do differently

The part hopper that we used was trying to solve a problem that was harder than it needed to be. We wanted a hopper that would be able to accept parts that were poured into it however we did not realize that you had up to ten seconds to load the parts into the hopper. With this is mind we believe that we could have designed a much simpler hopper which would lead to an overall more reliable system since such a large amount of time was spent prototyping and redesigning the hopper. Additionally, the ability to only process a single part at a time was a concern. In order to meet the time requirement of 5 minutes the system would have to be able to run different subsystems in parallel. This was something that the system could not do due because of both power requirements and the manner in which the rail systems were designed.

8.3 Future work

Our team was very happy with the way that many of our subsystems functioned. The systems that we would want to improve in the future would be the part hopper, vision processing, and wire cutter. The part hopper would be changed so that the parts are placed into the hopper in a single file line. We believe that this could be done within the ten second time limit to load parts and would reduce the amount of jamming that occurred in the system. The camera that we were using to detect part orientation suffered greatly from variation in lighting conditions. By mounting a housing around the camera and setting up our own light source the vision processing could be improved enough to be satisfactory. Otherwise, a new system for detecting orientation would have to be implemented. Lastly, the wire cutter suffered from two problems. The cutters would dull easily after extended use and the string that we used would fray and break other time. The wire cutter would need to be sharpened and replaced and a substitute for the string that was flexible but still strong would need to be found. Overall, we were happy with the software but there was room for adding additional sensors to the system so that the small errors that built up over time in the stepper motors would be eliminated.

9. References

[1] "Making a Powerful Linear Actuator." *Instructables*. N.p., n.d. Web. <http://www.instructables.com/id/Making-a-Powerful-Linear-Actuator/>.

[2] "A4988 Stepper Motor Driver Carrier" *Pololu* N.p., n.d. Web <https://www.pololu.com/product/1182/>

[3] "Stepper Motor Data Sheet" *SparkFun* N.p., n.d. Web <https://www.sparkfun.com/datasheets/Robotics/SM-42BYG011-25.pdf>

[4] "Solarbotics L298 Compact Motor Driver Datasheet" *Solarbotics* N.p., n.d. Web <https://solarbotics.com/download.php?file=40>

[5] "OpenCV" *OpenCV* N.p., n.d. Web <http://opencv.org/>

[6] "Encoder Library" *PJRC* N.p., n.d. Web <http://www.pjrc.com/teensy/td_libs_Encoder.html>

[7] "CustomStepper" *Arduino Playground* N.p., n.d. Web <http://playground.arduino.cc/Main/CustomStepper>

10. Appendix A - Final Arduino Code

PreTinnerSoftware.ino:

```
#include <Servo.h>
#include <Encoder.h>
#include <CustomStepper.h>
#include "SupportFunctionsStructs.h"
#include "GlobalVariables.h"
#define ENCODER_OPTIMIZE_INTERRUPTS

void setup()
{
  placerSetup();
  revSetup();
  fluxSetup();
  LEDSetup();
  cameraSetup();
  //code useful for modification or debug

    //revRelState = 0;
  //revRelCount = 0;
  //fluxDispState = 0;
  //partPlacerDone = true;
  //partState = 100;
  //partPos = ;
  //partPlacerDone = ;
}

void loop()
{
  cameraLoop();
  LEDLoop();
  placerLoop();
  revLoop();
  fluxLoop();
}
```

CameraState.ino:

```
Servo flipperServo;
Servo cameraServo;

const char flipper_mag_pin = 4;
const char flipperServoPin = 5;
const char cameraServoPin = 11;

DCStruct hopperDC = DCINIT(50, 52);

// Constants for flipper servo positions
const int slidingPos = 25;
const int restingPos = 14;
const int flipPos = 180;
// Constant for camera Servo
const int holdingPos = 20;
const int viewingPos = 90;
//STATE VARIABLES
unsigned long cameraTime = 0;
unsigned char serialValue = '5';

byte partPosTemp = 0;
```

```
void cameraSetup() { //Also sets up serial communications

  DCSetup(&hopperDC);

  pinMode(flipper_mag_pin,OUTPUT);
  pinMode(flipperServoPin, OUTPUT);
  pinMode(cameraServoPin,OUTPUT);

  flipperServo.attach(flipperServoPin);
  cameraServo.attach(cameraServoPin);

  flipperServo.write(restingPos);
  cameraServo.write(holdingPos);
  cameraTime = millis();
  rotateUp(&hopperDC);
  Serial.begin(9600);
  while(!Serial.available());
  serialValue = Serial.read();
}

void cameraLoop() {
  if(Serial.available()) {
    serialValue = Serial.read();
    LEDValue = serialValue;
  }

  if(!partPlacerDone){
    switch(cameraState){
        case 1:
          //HOLD by rotating up
          rotateUp(&hopperDC);

          flipperServo.write(restingPos);
          cameraServo.write(holdingPos);
          digitalWrite(flipper_mag_pin,LOW);



          if(serialValue != '0' && serialValue != '5' && partPos == 0 && millis() - cameraTime > 1000){
            cameraState = 5;
            cameraTime = millis();
          }else if(partPos == 0 && millis() - cameraTime > 5000){
            cameraState = 2;
            cameraTime = millis();
          }
          break;
        case 2:
          //SPIN FOWARD by depowering the magnet
          DCStop(&hopperDC);
          flipperServo.write(restingPos);
          cameraServo.write(holdingPos);
          digitalWrite(flipper_mag_pin,LOW);

          if(millis() - cameraTime > 80){
            cameraState = 1;
            cameraTime = millis();
          }
          break;
        case 3:
          //FLIP PART
          rotateUp(&hopperDC);
          flipperServo.write(flipPos);
          cameraServo.write(holdingPos);
```

```
        digitalWrite(flipper_mag_pin,HIGH);

        if(millis() - cameraTime > 1000){
          cameraState = 2;
          partPos = partPosTemp;
          cameraTime = millis();
        }
        break;
      case 4:
        //SLIDE PART
        rotateUp(&hopperDC);
        flipperServo.write(slidingPos);
        cameraServo.write(viewingPos);
        digitalWrite(flipper_mag_pin,LOW);

        if(millis() - cameraTime > 1000){
          cameraState = 2;
          partPos = partPosTemp;
          cameraTime = millis();
        }
        break;
      case 5:
        //VIEWING PART
        rotateUp(&hopperDC);
        flipperServo.write(restingPos);
        cameraServo.write(viewingPos);
        digitalWrite(flipper_mag_pin,HIGH);

        if(millis() - cameraTime > 2500){
          if(serialValue == '1' ){
            cameraState = 4;
            partPosTemp = 1;
            cameraTime = millis();
          }else if(serialValue == '2'){
            cameraState = 4;
            partPosTemp = 2;
            cameraTime = millis();
          }else if(serialValue == '3'){
            cameraState = 3;
            partPosTemp = 2;
            cameraTime = millis();
          }else if(serialValue == '4'){
            cameraState = 3;
            partPosTemp = 1;
            cameraTime = millis();
          }

        }
        break;
    }
  }else{
    DCStop(&hopperDC);
  }

 serialValue = '0'; //only accept latest
}
```

LEDState.ino:

```
const char LED0 = 34; //34
const char LED1 = 40;
const char LED2 = 38;
const char LED3 = 36;
const char LED4 = 42;

void LEDSetup() {
```

```
  pinMode(LED0,OUTPUT);
  pinMode(LED1,OUTPUT);
  pinMode(LED2,OUTPUT);
  pinMode(LED3,OUTPUT);
  pinMode(LED4,OUTPUT);

  digitalWrite(LED0,LOW);
  digitalWrite(LED1,LOW);
  digitalWrite(LED2,LOW);
  digitalWrite(LED3,LOW);
  digitalWrite(LED4,LOW);
}

void LEDLoop() {

  switch(LEDValue){
    case '0':
      digitalWrite(LED0,LOW);
      digitalWrite(LED1,HIGH);
      digitalWrite(LED2,HIGH);
      digitalWrite(LED3,HIGH);
      digitalWrite(LED4,HIGH);
      break;
    case '1':
      digitalWrite(LED0,HIGH);
      digitalWrite(LED1,LOW);
      digitalWrite(LED2,HIGH);
      digitalWrite(LED3,HIGH);
      digitalWrite(LED4,HIGH);
      break;
    case '2':
      digitalWrite(LED0,HIGH);
      digitalWrite(LED1,HIGH);
      digitalWrite(LED2,LOW);
      digitalWrite(LED3,HIGH);
      digitalWrite(LED4,HIGH);
      break;
    case '3':
      digitalWrite(LED0,HIGH);
      digitalWrite(LED1,HIGH);
      digitalWrite(LED2,HIGH);
      digitalWrite(LED3,LOW);
      digitalWrite(LED4,HIGH);
      break;
    case '4':
      digitalWrite(LED0,HIGH);
      digitalWrite(LED1,HIGH);
      digitalWrite(LED2,HIGH);
      digitalWrite(LED3,HIGH);
      digitalWrite(LED4,LOW);
      break;
    case '5':
      digitalWrite(LED0,LOW);
      digitalWrite(LED1,LOW);
      digitalWrite(LED2,LOW);
      digitalWrite(LED3,LOW);
      digitalWrite(LED4,LOW);
      break;
    case '6': //Done
      digitalWrite(LED0,HIGH);
      digitalWrite(LED1,HIGH);
      digitalWrite(LED2,HIGH);
      digitalWrite(LED3,HIGH);
      digitalWrite(LED4,HIGH);
      break;
```

```
 }

}
```

FluxDispensorState.ino:

```
//const char fluxDCPinUp = 30;
//const char fluxDCPinDown = 28;
DCStruct fluxDC = DCINIT(30, 28);

unsigned int fluxCor = 70;
unsigned int trayCor = 110;
bool reloaded = false;

//const char enPin4 = 39;
//const char stepPin4 = 41;
//const char dirPin4 = 37;
//CustomStepper fluxStep(stepPin4, 0, 0, 0, (byte[]){8, B1000, B1100, B0100, B0110, B0010, B0011, B0001, B1001}, 400, 100, CW);
//fluxStep.setRPM(800);
StepperStruct fluxStep = STEPPERINIT(41,39,37,400,400);

const int fluxStartingPos = 455;

unsigned char fluxDispXCounter = 0;
unsigned char fluxDispYCounter = 0;

void fluxSetup() {
  DCSetup(&fluxDC);
  StepperSetup(&fluxStep);
}

void fluxLoop() {
  switch(fluxDispState)
  {
    case 0:
      //get an indication that the partPlacer finished
      if(partPlacerDone)
      {
        fluxDispState = 1;
      }
      break;

    case 1: //move flux to initial position
      rotateDegrees(&fluxStep, (fluxStartingPos)*4, HIGH);
      fluxDispState = 2;
      break;

    case 2:
      if(Done(&fluxStep))
      {
        fluxDispState = 30; //testing if part is in correct position
        //Serial.println("Should push down");
      }
      break;

    case 30:
      fluxDispState = 3;
      rotateDown(&fluxDC);

    case 3: //pushing down time
      if(timerElapsed(&fluxDC, 110))
      {
        //Serial.println("finish pushing");
```

```
    fluxDispState = 4;
    revFluxTimer = millis();
  }
  break;

case 4: //hold time
  if(millis() - revFluxTimer > 55)
  {
    fluxDispState = 5;
    rotateUp(&fluxDC);
  }
  break;

case 5: //pushing up timem
  if(timerElapsed(&fluxDC, 65))
  {
    fluxDispState = 6;
  }
  break;

case 6: // do tray correction
  rotateDegrees(&trayStep, trayCor*4, LOW);
  fluxDispState = 7;
  break;

case 7:
  if(Done(&trayStep))
  {
    fluxDispState = 8;
  }
  break;

case 8: // do flux correction
  rotateDegrees(&fluxStep, fluxCor*4, LOW);
  fluxDispState = 9;
  break;

case 9:
  if(Done(&fluxStep))
  {
    fluxDispState = 10;
  }
  break;

case 10: //rotate the revolver
  rotateDegrees(&revStep, 360/21*4, LOW);
  fluxDispState = 11;
  break;

case 11:
  if(Done(&revStep))
  {
    revRelCount--;
    fluxDispState = 12;
    revFluxTimer = millis();
  }
  break;

case 12: //wait for tube
  if(millis() - revFluxTimer > 800)
  {
    //
    fluxDispState = 40;//* //skip tray correction
    if( (Large == true) && (reloaded == false))
```

```
    {
      fluxDispState = 10;
      reloaded = true;
    }
    //this algorithm will not result in inserting one less wire by parity
  }
  break;


case 40:
  fluxDispState = 13;
  rotateDegrees(&trayStep, (trayCor)*4, HIGH); //move back
  //Fight back the offense
  break;


case 13: //undo tray correction
  if(Done(&trayStep))
  {
    fluxDispState = 14;
  }
  break;


case 14: // undo flux correction
  rotateDegrees(&fluxStep, (fluxCor)*4, HIGH);
  fluxDispState = 15;
  break;


case 15:
  if(Done(&fluxStep))
  {
    //fluxDispState = 100;
    //break;


    reloaded = false; //refresh reload


    fluxDispState = 16; //move in the X direction
    //increase counter and ensure we are done
    fluxDispXCounter++;
    if(fluxDispXCounter == 5) //we did all five of them
    {
      fluxDispXCounter = 0;
      fluxDispYCounter++;
      fluxDispState = 17; //move in the Y direction
      if(fluxDispYCounter == 3) //all 20 pieces have been fluxed and wired
      {
        fluxDispState = 100;
      }
    }
  }


  break;


case 16: //do flux movement in X direction and refresh to state 2
  rotateDegrees(&fluxStep, (190)*4, HIGH);
  fluxDispState = 2; //restart the process
  break;


case 17: //move the tray a little bit up.
  rotateDegrees(&trayStep, (400)*4, HIGH);
  fluxDispState = 18;
  break;


case 18:
  if(Done(&trayStep))
  {
```

```
      //Move the flux back to original position
      rotateDegrees(&fluxStep, (850 + fluxStartingPos)*4, LOW);
      fluxDispState = 19;
    }
    break;


  case 19:
    if(Done(&fluxStep))
    {
      //start up state 0
      fluxDispState = 0;


      //By parity, all wires should run out on this state
      if(revRelCount == 0) //we ran out
      {
        revRelState = 0; //Restart revRelState
        fluxDispState = 100; //disable the fluxDispState
        //The flux will continue when 20 new wires are loaded
      }
    }
    break;
 }

 if(fluxDispState == 2 || fluxDispState == 9 || fluxDispState == 15 || fluxDispState == 19)
   fluxStep.Step.run();


 if(fluxDispState == 11)
   revStep.Step.run();


 if(fluxDispState == 7 || fluxDispState == 13 || fluxDispState == 18)
   trayStep.Step.run();



}
```

GlobalVariables.h:

```
//include variables that are shared in multiple files
byte partPos = 0;
bool partPlacerDone = false;
unsigned long revFluxTimer = 0; //used for both flux and revRel
char LEDValue = '5';


bool Large = true; //tells if the piece is large or not.
unsigned char revRelCount = 0;


//States
int cameraState = 1;
int partState = 0;
int revRelState = 100;
int fluxDispState = 100; //starts when revRelState finishes



//const char dirPin2 = 49;
//const char enPin2 = 51;
//const char stepPin2 = 53;
StepperStruct trayStep = STEPPERINIT(49,51,53,200,400);


//const char dirPin3 = 22;
//const char enPin3 = 24;
//const char stepPin3 = 26;
//CustomStepper revStep(stepPin3, 0, 0, 0, (byte[]){8, B1000, B1100, B0100, B0110, B0010, B0011, B0001, B1001}, 6400, 5, CW);
```

```
//revStep.setRPM(10);
StepperStruct revStep = STEPPERINIT(22,24,26,6400, 10);
```

PartPlacerState.ino:

```
//Servo and magnets
const char partPlacerServoPin =  6;
const char placer_mag_pin = 7;
const char partReorientatorServoPin = 10;

// Constants for part servo servo positions
const int cameraHeight = 125;
const int trayHeight = 59;
const int restingHeight = 0;

Servo partPlacerServo;
Servo partServo; //reorientator

//const char dirPin = 43;
//const char enPin = 45;
//const char stepPin = 47;

StepperStruct partStep = STEPPERINIT(43,45,47,400,800);
unsigned long partPlacerTimer = 0;

byte currentPartPos;
unsigned char partStepperXCounter = 0;
unsigned char partStepperYCounter = 0;

void placerSetup() {
  pinMode(placer_mag_pin,OUTPUT);
  pinMode(partPlacerServoPin, OUTPUT);
  StepperSetup(&partStep);
  StepperSetup(&trayStep);
  partPlacerServo.attach(partPlacerServoPin);
  partServo.attach(partReorientatorServoPin);
  //initial values
  digitalWrite(partStep.enPin, HIGH);
  digitalWrite(trayStep.enPin, HIGH);
  partServo.write(0);
  partPlacerServo.write(restingHeight);

 }

void placerLoop() {

  switch(partState){

    case 0:
      //wait until signal received
      if(partPos)
      {
        currentPartPos = partPos;
        if(partStepperYCounter != 0) //ignore first
          partState = 1;
        else
          partState = 11;
        partState = 1;

        partPlacerTimer = millis();

      }
```

```
  break;

case 1: //wait a while for part to settle
  if(millis() - partPlacerTimer > 1000)
  {
    partState = 2;
    partPlacerTimer = millis();


  }
  break;

case 2:
  digitalWrite(placer_mag_pin,HIGH);
  partPlacerServo.write(cameraHeight);
  if(millis() - partPlacerTimer > 800)
  {
    partState = 3;
    partPlacerTimer = millis();
  }
  break;

case 3:
  partPlacerServo.write(restingHeight);
  if(millis() - partPlacerTimer > 800)
  {
    partPos = 0;
    partState = 4;
  }
  break;

case 4:
  //move to tray
  rotateDegrees(&partStep, (1500 + 200*partStepperXCounter)*4, LOW);
  if(currentPartPos == 2)//ideally rotate as it moves
    partServo.write(170);
  partState = 5;
  break;

case 5:
  if(Done(&partStep))
  {
    partState = 7;
    partPlacerTimer = millis();
  }
  break;

case 7:
  //put piece down
  partPlacerServo.write(trayHeight);
  if(millis() - partPlacerTimer > 800)
  {
    partState = 8;
    digitalWrite(placer_mag_pin,LOW);
    partPlacerTimer = millis();
  }
  break;

case 8:
  partPlacerServo.write(restingHeight);
  if(millis() - partPlacerTimer > 800)
  {
    partState = 9;
    partPlacerTimer = millis();
  }
  break;
```

```
case 9: //return back
  rotateDegrees(&partStep, (1550 + 200*partStepperXCounter)*4, HIGH);
  partState = 10;
  break;


case 10:
  partServo.write(0); //ensure the servo is unrotated
  if(Done(&partStep))
  {
    //ensure we create a 5 by 4 layout of the parts
    //We will be using the X and Y counter to count
    //how much of the layout has been completed
    partState = 11;
  }
  break;


case 11: //update counter
  partState = 0;
  partStepperXCounter++;
  if(partStepperXCounter == 5) //did the fifth one already
  {
    partStepperYCounter++;
    partStepperXCounter = 0;
    if(partStepperYCounter == 4) //did all four of them
    {
      //prepare the flux dispenser state
      partState = 14;
    }
    else
    {
      partState = 12;


    }
  }
  break;


case 12:
  //Move the tray a little bit down
  //move to tray
  rotateDegrees(&trayStep, (400)*4, LOW);
  partState = 13;
  break;


case 13:
  if(Done(&trayStep))
  {
    //Start placing the pieces again
    partState = 0; //Debug: originally the code
  }
  break;


case 14:
  //move the tray to the proper part in the code
  rotateDegrees(&trayStep, (2275)*4, LOW); //2*(2300 - 600)
  partState = 15;
  break;


case 15: //flux is now ready to write upon
  if(Done(&trayStep))
  {
    partPlacerDone = true;
    revRelState = 0;
    partState = 100;
  }
```

```
    break;
  }

  //Ensure the code runs when needed
  if(partState == 5 || partState == 10)
  {
    partStep.Step.run();
  }
  if(partState == 13 || partState == 15)
    trayStep.Step.run();
    }
```

SupportFunctionsStructs.h:

```
//Introduce structures and Functions that simplifies code
//For the States

#define STEPPERINIT(dirPin, enPin, stepPin, SPR, RPM) {CustomStepper(stepPin, 0, 0, 0, (byte[]){8, B1000, B1100, B0100, B0110, B0010, B0011,
B0001, B1001}, SPR, RPM, CW), dirPin, enPin, stepPin}
#define DCINIT(pinUp, pinDown) {0, pinUp, pinDown}
#define DCENCINIT(pinUp, pinDown, pinA, pinB) {Encoder(pinA, pinB), pinA, pinB, pinUp, pinDown}

struct StepperStruct {
  CustomStepper Step;
  byte dirPin;
  byte enPin;
  byte stepPin;
};

struct DCStruct {
  unsigned long time_t;
  byte pinUp;
  byte pinDown;
};

struct DCEncStruct {
  Encoder enc;
  byte pinA;
  byte pinB;
  byte pinUp;
  byte pinDown;
};

//Stepper structure functions

void rotateDegrees(StepperStruct * stepper, float deg, byte dirSig )
{
  digitalWrite(stepper->dirPin, dirSig);
  digitalWrite(stepper->enPin, LOW);
  stepper->Step.rotateDegrees(deg);
}

bool Done(StepperStruct * stepper)
{
  if(stepper->Step.isDone())
  {
    digitalWrite(stepper->enPin, HIGH);
    return true;
  }
  return false;
}
```

```
void StepperSetup(StepperStruct * stepper)
{
  pinMode(stepper->dirPin,OUTPUT);
  pinMode(stepper->enPin, OUTPUT);
  pinMode(stepper->stepPin, OUTPUT);
}

//DCStructure
void rotateUp(DCStruct * DC)
{
  digitalWrite(DC->pinUp, HIGH);
  digitalWrite(DC->pinDown, LOW);
  //reset time
  DC->time_t = millis();
}

void rotateDown(DCStruct * DC)
{
  digitalWrite(DC->pinUp, LOW);
  digitalWrite(DC->pinDown, HIGH);
  //reset time
  DC->time_t = millis();
}

void rotateUp(DCEncStruct * DC)
{
  digitalWrite(DC->pinUp, HIGH);
  digitalWrite(DC->pinDown, LOW);
}

void rotateDown(DCEncStruct * DC)
{
  digitalWrite(DC->pinUp, LOW);
  digitalWrite(DC->pinDown, HIGH);
}

bool encoderGreater(DCEncStruct * DC, int steps)
{
  if(DC->enc.read() > steps)
  {
    digitalWrite(DC->pinUp, LOW);
    digitalWrite(DC->pinDown, LOW);
    return true;
  }
  return false;
}

bool encoderLess(DCEncStruct * DC, int steps)
{
  if(DC->enc.read() < steps)
  {
    digitalWrite(DC->pinUp, LOW);
    digitalWrite(DC->pinDown, LOW);
    return true;
  }
  return false;
}

bool timerElapsed(DCStruct * DC, unsigned long elapse)
{
  if(millis() - DC->time_t > elapse)
  {
    digitalWrite(DC->pinUp, LOW);
    digitalWrite(DC->pinDown, LOW);
    return true;
  }
```

```
  return false;

}

void DCStop(DCStruct * DC)
{
  digitalWrite(DC->pinUp, LOW);
  digitalWrite(DC->pinDown, LOW);
}

void DCSetup(DCStruct * DC)
{
  pinMode(DC->pinUp,OUTPUT);
  pinMode(DC->pinDown,OUTPUT);
}

void DCSetup(DCEncStruct * DC)
{
  pinMode(DC->pinUp,OUTPUT);
  pinMode(DC->pinDown,OUTPUT);
  pinMode(DC->pinA, INPUT);
  pinMode(DC->pinB, INPUT);
}

//General Timer function
bool timerElapsed(unsigned long time_t, unsigned long elapse)
{
  return (millis() - time_t) > elapse;

}
```

---

revRelState.ino:

```
//const char wirefeederPin = 12;
//Encoder wireEnc = Encoder(21, 20);
DCEncStruct wireEnc = DCENCINIT(12,0,21,20);
//Encoder myEnc = Encoder(2, 3);
//const char DCCutterPinCut = 46;
//const char DCCutterPinRelease = 48;
DCEncStruct cutEnc = DCENCINIT(46,48,2,3);

void revSetup() {
  DCSetup(&wireEnc);
  DCSetup(&cutEnc);
  StepperSetup(&revStep);
}

void revLoop() {

  switch(revRelState){
     case 0:
      rotateUp(&cutEnc);
      if(encoderGreater(&cutEnc, (1820 + 6*revRelCount) ))
      {
         revRelState = 1;
      }
      break;

    case 1:
      rotateDown(&cutEnc);
      if(encoderLess(&cutEnc, 29))
      {
         revRelState = 2;
```

```
    }
    break;


  case 2:
    rotateUp(&wireEnc);
    revRelState = 3;
    break;


  case 3:
    if(encoderGreater(&wireEnc, 40))
    {
      wireEnc.enc.write(0);
      revRelState = 4;
    }
    break;


  case 4: //rotate the revolver
    rotateDegrees(&revStep, 360/21*4, LOW);
    revRelState = 5;
    break;


  case 5:
    if(Done(&revStep)) //keep repeating until 20 wires are put in
    {
      revRelState = 0;//Debug remember to undo
      revRelCount++;
      if(revRelCount == 21) //inserted in all the 20 pieces
      {
        revRelCount = 20; //have accurate count of revRel wires
        revRelState = 100;
        fluxDispState = 0;
        //Give indication that the machine is ready
      }
      //debug
      //delay(1000);
    }
    break;
  }


  if(revRelState == 5)
  revStep.Step.run();


}
```