2020

DESIGN OF AN AUTOMATED SYSTEM FOR CORNER BRICK ASSEMBLY PROCESS

Zachary Whittaker
University of Rhode Island, zwhittaker@uri.edu

Follow this and additional works at: <https://digitalcommons.uri.edu/theses>

Recommended Citation

Whittaker, Zachary, "DESIGN OF AN AUTOMATED SYSTEM FOR CORNER BRICK ASSEMBLY PROCESS" (2020). *Open Access Master's Theses*. Paper 1826.
<https://digitalcommons.uri.edu/theses/1826>

This Thesis is brought to you for free and open access by DigitalCommons@URI. It has been accepted for inclusion in Open Access Master's Theses by an authorized administrator of DigitalCommons@URI. For more information, please contact digitalcommons-group@uri.edu.

DESIGN OF AN AUTOMATED SYSTEM FOR CORNER BRICK
ASSEMBLY PROCESS
BY
ZACHARY WHITTAKER

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND
2020

MASTER OF SCIENCE THESIS

OF

ZACHARY WHITTAKER

APPROVED:

Thesis Committee:

Major Professor Musa Jouaneh

Chengzhi Yuan

Bin Li

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2020

ABSTRACT

Insulated Faux Brick (IFB) is an improvement on traditional faux brick that aims to further reduce weight and installation time while providing insulation characteristics beyond traditional brick. In preparation for growing demand, the manufacturers of IFB are seeking to increase throughput and scalability through automation. One area of difficulty, for which the company has sponsored this project, is the assembly of corner bricks. Current production of corner bricks relies on human workers to apply adhesive and press together two end pieces until the adhesive has dried. This is a time-consuming step which is not easily scalable to higher production rates. Thus, this research explored potential automated solutions to meet the ever-increasing demand. Specifically, the concept discussed in this report uses a modular array of joining units that can be repeated to achieve the desired scale of production. These joining units use a combination of pneumatic and electric actuators to quickly align a pair of components and move them together after the glue is applied. Further, the low cost and small size of the joining units allows many to be grouped into a frame and serviced by a single operator. This concept was verified through the construction of a beta prototype containing two joining units and a low-cost glue dispensing system to service both units. The glue dispensing system was created by modifying a manually operated glue gun through replacement of the advancing mechanism with a linear actuator with feedback control. Production test runs conducted with the prototype showed very promising results. During these runs, the prototype was operated continuously with an average throughput greater than two bricks per minute. Further, the part quality of the bricks was similar to that of the manual process. Based on a critical path analysis of the prototype, a system containing eight joining units with the same modular architecture developed in this thesis has the potential for a three-fold increase in production rate per worker.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Musa Jouaneh for presenting me with this research opportunity and for being an excellent mentor along the way. He provided a voice of reason and important technical wisdom to keep me on the right path throughout the project. Further his confidence in me helped me stay motivated during the difficult portions of this project, when the many tasks became overwhelming. I would also like to thank my committee members Dr. Chengzhi, Yuan, Dr. Bin Li, and Dr. Paolo Stegagno for their willingness to lend their time and thoughtful critiques during my defense.

I would like to extend thanks to Joe Gomez for his continued assistance in the machine shop. This project required much more machining than anticipated, and Joe was always willing to lend his time and tools to the cause. I am truly grateful for the expertise he shared with me which gave me the skills and the confidence to finish the machining on my own.

Thank you to Jim Byrnes for his technical support with all of the electronics issues faced during this project, of which there were many. Jim was always willing to search high and low for components and datasheets to fix any problem we had. His knowledge was especially helpful in wiring the high voltage components inside the control box to avoid any costly mistakes.

Lastly, I would like to thank my girlfriend, Meg Morrissey, for her continued support which made this experience possible. Her words of encouragement helped me make the decision to leave my job to pursue my degree full time, even when research opportunities seemed out of reach.

TABLE OF CONTENTS

ABSTRACT.....	ii
ACKNOWLEDGEMENTS	iii
LIST OF TABLES	vi
LIST OF FIGURES	vii
Chapter 1: Introduction	1
1.1 Background	1
1.2 Statement of the Problem.....	4
1.3 Approach.....	6
1.4 Literature Review.....	8
Chapter 2: Concept Development.....	14
2.1 Production goals.....	14
2.2 Initial Observational Visit.....	14
2.3 Concept Generation Phase	17
2.4 Production Concept.....	18
2.5 Minimum Viable Product Prototype.....	22
Chapter 3: Prototype Design.....	24
3.1 Automated Joining Unit.....	24
3.1.1 AJU Mechanical Concept	24
3.1.2 AJU Construction.....	27
3.1.2.1 Actuator Selection.....	31
3.1.2.2 Pneumatic Actuator Control	35
3.1.2.3 Electric Actuator Description	38
3.1.2.4 AJU Electronics	41
3.1.3 Configuring the AJUs	44
3.2 Glue Subsystem	47
3.2.1 Existing Dispensing Systems	47
3.2.2.1 Actuator Selection.....	50
3.2.2.2 Glue Gun Modifications	55
3.2.2.3 Electronics Design	57
3.2.2.4 Microcontroller and Software Structure	59

3.2.3 Testing and Results of Glue Subsystem.....	62
3.3 Linear Slide and Frame	64
3.4 System Control.....	71
3.4.1 Control Computer	72
3.4.2 Control Interface	77
3.4.3 Control Box Hardware	80
3.4.4 Control Software	83
3.4.4.1 Task Descriptions.....	85
3.4.4.2 System Operational Sequence.....	92
Chapter 4: Optimization and Final Testing.....	97
4.1 Parameter Optimization	97
4.2 Pre-Production Testing	100
4.3 Testing at Company Facility	106
Chapter 5: Conclusion.....	107
5.1 Recommendations for Future Work.....	111
References	113
Appendix A: AJU Mechanical Design Documents	116
Appendix B: Electrical Schematics	135
Appendix C: Source Code	139
Bibliography	164

LIST OF TABLES

Table 1: AJU Interface Board Connections.....	42
Table 2: Summary of Glue Dispensing Options.....	48
Table 3: Force Sensing Resistor Calibration Data	52
Table 4: Voltage Outputs Relative to Glue Flow	53
Table 5: Actuonix Linear Actuator Signal Table	57
Table 6: Singal Table of Glue Subsystem	59
Table 7: Connection Diagram of the Linear Slide I/O	69
Table 8: Supply Voltage Requirements for Control Box Devices	82
Table 9: Signal Table of the 16-pin Raspberry Pi to Linear Slide Driver Cable.....	83

LIST OF FIGURES

Figure 1. IFB Corner Bricks Installed on a Commercial Building	3
Figure 2. Flat Components Which Make Up the Corner Brick	4
Figure 3: Left-Oriented Corner Brick Styles	5
Figure 4: Manual Process of Assembling Corner Bricks.....	15
Figure 5: Observed Glue Dispensing Technique	16
Figure 6: Estimated Timing of Events for Full-Scale Concept.....	20
Figure 7: Render and Constructed Prototype Delivered to The Company	23
Figure 8: Two AJUs Installed in the Final Prototype	24
Figure 9: Mechanism of AJU Motion for Joining the Components	25
Figure 10: Stopper Positions and Safe Zone for the Retracted Stopper	26
Figure 11: Deviation of the Front Edge of the Components.....	27
Figure 12: Adjustable Stopper Viewed From Top (Left) and Side (Right).....	28
Figure 13: Pneumatic Actuators on AJU viewed from Side (Left) and Front (Right)	29
Figure 14: Electric Flipper Motor with Tension Block, and Timing Belt	30
Figure 15: Pneumatic Control Hardware Mounted to Underside of AJU	37
Figure 16: Pneumatic Actuator Sensor Location and Activation	38
Figure 17: AJU Interface Board Top (Left) and Bottom (Right) Views	41
Figure 18: Approximate Location of Actuator Inside Glue Gun Shell.....	50
Figure 19: Voltage Divider Used to Measure FSR Resistance.....	51
Figure 20: Speed, Force, and Current Characteristics of Actuatorix L16 Series Actuators	54
Figure 21: Glue Gun Handle Removed and Actuator Mounting Bracket Installed.....	56
Figure 22: H-Bridge Wiring Diagram for the Glue Subsystem.....	57
Figure 23: State Transition Diagram of the Glue Subsystem Software.....	60
Figure 24: Glue Test Configuration.....	62
Figure 25: Glue Dispensed vs. Dispense Duration	63
Figure 26: Linear Slide Holding the Glue Subsystem	64
Figure 27: Potential Dispense Paths Using Two-Axis Positioning System.....	65
Figure 28: Dispense Pattern from Single-Axis Operation with Delayed Flipping	67
Figure 29: Horizontal View of Prototype For Determining Minimum Slide Length	68

Figure 30: Adjustable Coordinates of the Linear Slide	71
Figure 31: Compute Stack Mounted Inside the Control Box	72
Figure 32: General Signal Diagram of the Compute Stack	73
Figure 33: Raspberry Pi GPIO Pin Functions [Source: Element 14].....	74
Figure 34: IO Pi+ Board Mounted to Raspberry Pi GPIO Pins [Source: Adafruit]	75
Figure 35: RELAY Plate [Source: Pi-Plates].....	76
Figure 36: Front (Left) and Reverse (Right) Views of the Control Interface	77
Figure 37: Arrangement of Devices Inside the Control Box	80
Figure 38: Task Scanning Sequence Followed by the Control Software	84
Figure 39: Scheduler State Transition Diagram.....	86
Figure 40: Position Sequence Used by the Move Task	88
Figure 41: General State Transition Diagram of the Move Task.....	89
Figure 42: AJU Task General State Transition Diagrams	91
Figure 43: Test Samples to Determine Glue Start and Stop Points	98
Figure 44: Prototype During Pre-Production Test Runs.....	101
Figure 45: Estimated Timing of Events for The Two-Unit Prototype.....	103
Figure 46: Samples of Successfully Completed Bricks from Pre-Production Runs.....	103
Figure 47: Samples of Issues from Pre-Production Tests	105

Chapter 1: Introduction

1.1 Background

The increasing global consensus on climate change has led many to voluntarily set goals upon themselves to reduce long-term effects from greenhouse gas emissions. The United States for example, enacted the Climate Change Action Plan in 1993 with the goal of reducing greenhouse gas emissions to 1900 levels before the year 2000 ^[1]. In that same year, the non-profit United States Green Building Council (USGBC) was founded to increase sustainability in building design. To promote its goals, the USGBC began by providing certification through its Leadership in Energy and Environmental Design (LEED) program for new constructions and renovations which demonstrate high levels of sustainability. This program assigns points to each project pursuing LEED certification for implementing specific measures to reduce environmental impact ^[2]. As indicated by its name, LEED places a strong emphasis on reduction in energy consumption of buildings, with the Institute for Research in Construction finding that LEED-certified buildings consume 18-39% less energy per floor than traditional buildings of similar size and purpose ^[3].

The LEED program represents just one of many ongoing efforts to reduce building energy use, which was estimated by the US Department of Energy to account for 40% of the total domestic greenhouse gas emissions in 2011 ^[4]. The US federal government and many state governments encourage energy efficiency through tax incentives to business and homeowners for taking steps to reduce their energy consumption. For example, US citizens are eligible to receive tax credits amounting to 22-30% of the cost of installed renewable energy systems through the year 2021 ^[5]. In addition to government efforts, individuals are also driving the

increase in energy efficiency due to the lower energy bills resulting from these choices, or simply due to their own environmental concerns.

Given this climate, it is no surprise that investment in more efficient building insulation systems has increased exponentially in recent decades. One branch of the insulation market which has seen exceptional growth is that of External Insulation and Finishing Systems (EIFS). Traditional residential installations consist of insulation packed between wooden studs. The disadvantage of these traditional systems is that they do nothing to decrease heat flux through the studs themselves, regardless of the insulation quality. In contrast, EIFS make use of a continuous layer of insulation around the core structure to reduce the heat flux across the entire exterior ^[6]. This insulation is usually made of a low-density, high-rigidity material such as expanded or extruded polystyrene with a protective fiberglass mesh. The outer layer is typically composed of an acrylic coat to give the appearance of stucco. This is one of the few finishes which can be supported by the low-density cladding. This cladding is usually separated from the wall to provide a drainage plane and provides very little structural support on its own. Thus, finishes with significant weight, such as brick veneer, are incompatible with some EIFS.

Insulated Faux Brick (IFB) was developed to combine the benefits of an EIFS with the appearance of traditional brick. When installed (Figure 1), this product matches the look of traditional brick, requiring no compromises for architects interested in a brick aesthetic. Inside the IFB however, the bulk of the volume is made of extruded polystyrene (XPS) which reduces weight and increases thermal resistance greatly when compared to traditional brick.

Additionally, IFB is engineered with a groove on the upper edge to hold mortar and allow quick position of each row with respect to the row below it.

Figure 1. IFB Corner Bricks Installed on a Commercial Building

Due to the lack of existing brick systems compatible with EIFS, demand for IFB has been strong since its release. To meet this increasing demand, the company behind IFB is investing heavily in new processes for higher throughput manufacturing of IFB. Currently, much of this process has been automated; however, there remains a limited number of bottlenecks which must be removed for the process to be truly scalable. One such area is the assembly of corner bricks which are used for exterior corners of buildings and window trims. These corner bricks are made from two flat components, shown in figure 2, cut at 45° angles such that they form a right angle when assembled. After being cut, the components are transferred to a station where workers manually apply glue, align the parts, and apply force for up to 20 seconds until the glue is dried. This process is time-consuming for the workers and cannot be scaled to larger production quantities without training additional workers.

Figure 2. Flat Components Which Make Up the Corner Brick

1.2 Statement of the Problem

The end goal for the company is to procure a production-ready machine to produce its complete line of corner bricks with a throughput and operating cost similar to that of the rest of its production steps. As with machines procured by the company to accomplish other production tasks, such a machine is likely to require significant cost and time working with an established automation company. Prior to investing significant funds into this effort, the company graciously sponsored this project with the University of Rhode Island to serve as a risk mitigation step. The specific terms of the cooperate effort included a budget of \$50,000 with which to investigate a semi-automated or automated system for corner brick production.

To fit this budget and the eight months of allotted time, the scope of this project had to be reduced where possible while still addressing the main goal. Thus, the project was limited to the development of a process for producing corner bricks which could be carried out efficiently at large scale, as well as a prototype to investigate the feasibility of the large-scale concept. Further, of the four sizes of corner brick produced, shown in figure 2, this project considers only the modular variant shown in the upper left as this is the most popular style. The four sizes

are currently manufactured using the same method, so any process developed for the modular variant should be applicable to all sizes. Lastly, each style of brick is produced in both right and left orientations which are mirrored versions of each other. Because these must be produced in equal numbers to alternate direction with each brick layer, both are equally important for this study. Thus, the concept and prototype must be designed to operate on either orientation.

Figure 3: Left-Oriented Corner Brick Styles

Unfortunately, the use of hot-melt glue as an adhesive and the precise alignment required to make the corner bricks makes this a very difficult step to automate. Hot-melt glue requires a long time to harden, and the quantity applied must be carefully controlled. If too much glue is used, the glue will be pressed out of the sides and possibly fall into the machine which is unacceptable. If too little glue is applied or if the coverage is not complete, the bricks may not hold together during transport and installation. Additionally, the bricks must be indistinguishable from normal bricks when assembled, so a misalignment of more than 1mm is also unacceptable.

1.3 Approach

Developing and adhering to a project plan was vital in achieving the stated goals within the allotted timeframe. The first step of this plan was to gain a deeper understanding of the current process employed by the company. Thus, in March 2019, the team visited the facility to speak with subject matter experts and observe the corner brick assembly process. Following this visit, the next two months were spent conducting a literature review of existing systems that perform similar assembly tasks to the problem at hand. Although no single system was found which could conduct the full assembly process, this review helped drive the design of the alignment and glue dispensing systems. Further, the review of large assembly systems was crucial for the concept generation phase, which began shortly afterwards. The goal of this phase was to adapt the methods uncovered in the literature review to the challenge of corner brick assembly. From the many concepts considered for each subtask (e.g. transferring, aligning, and joining the components), the most promising concepts were combined into a few comprehensive designs, representing different levels of automation.

Of these, the design with the greatest benefit that could be validated in the project timeframe was selected for further study and documented in the concept generation report delivered in May 2019. The chosen design uses a modular arrangement of subsystems to support future expansion. At the core of this modular design is the Automated Joining Unit (AJU) which aligns and holds one set of components. The number of AJUs in the machine can be increased as needed to meet the desired scale, with many of the other subsystems being unaffected. An additional benefit of this approach is that the same mechanism is used to join the parts, regardless of the scale. This allows for any results from a small-scale prototype to be directly applied to the full-scale concept.

From the general concept, a detailed design was created which replaced structural components with off-the-shelf parts or those easily machined from raw materials. This design was modeled in Solidworks to verify the fitment, mass, and inertial moments of the design and specific actuators were selected based on the results from this model. Where possible, the concepts of the design were tested using simple fixtures and similar actuators from previous projects. In other cases, concepts were assessed using simulations in Excel and MATLAB.

Once the design was refined, work began on the building a prototype. Three parallel efforts were launched to develop and test the joining unit, glue positioning system, and glue dispensing system. To test the joining unit concept, a single unit was constructed based on the detailed design and actuated using a Raspberry Pi computer. At the same time, the linear slide and frame were procured from Gibson Engineering and the slide was tested using the same Raspberry Pi as an interface. Lastly, the initial development of the glue dispensing subsystem was performed by then undergraduate student, Joe Kearny.

Next, the three subsystems were integrated into a complete alpha prototype. This involved mounting the joining unit and glue subsystem to the linear slide and creating a basic controller to operate the three systems together. A single Raspberry Pi was used to control the joining unit and linear slide and an Arduino was used to control the glue dispensing system. At this time, a test script was written in Python containing minimum control structures needed to step through the sequence. Meanwhile, breadboards were used to test the interfacing circuits between the subsystems and the controller.

Once the concept was proven through a successful automated assembly, work began to upgrade the system to a beta prototype with more robust operation. First, a second joining unit was constructed using the lessons learned from the first unit. Next, the interfacing electronics were moved from breadboards to soldered circuit boards and the bench-top power supplies

were replaced with dedicated power supplies. The interfaces on the control computer were modularized using add-on boards to allow for future expansion to accommodate up to eight joining units and a control box was purchased and modified to house the power supplies, control computers, and user interface. Lastly, a new version of the software was developed with a more robust control architecture and a text-based user interface.

In the final phase, the beta prototype underwent extensive testing to validate the concept and determine the optimal parameters. To support this testing, the company provided the team with several hundred samples of corners with different finishes and orientations. Using these samples, the prototype was tested methodically to determine the parameters which produce the maximum quality and throughput. Once the parameters were set, the prototype was operated continuously to estimate the throughput of the full-scale system based on the concept. Finally, the prototype was delivered to the company and configured for use in their production line. Additionally, the design documents, cost information, and test results were provided to the company to assist with any further development of this system.

1.4 Literature Review

Since this is a new product with unique construction, there have not yet been any investigations into this specific process which could be found in the literature. Thus, the task of high-throughput automated corner brick assembly was broken into its basic elements for which previous studies could provide some insight.

The first basic element considered was that of joining polystyrene components to one another. For this, the handbook of plastics joining provided an excellent compilation of previous studies into methods of welding polystyrene. In this context, welding refers to the merging of the two components by adding heat or dissolving agents to melt the two components

together. Previous studies have considered melting polystyrene substrates via heated tools, ultrasonic vibrations, friction, lasers, and solvents such as toluene and methylene chloride. Of these methods, ultrasonic welding and solvent welding were quite effective at forming bonds with strengths close to the bulk properties of polystyrene ^[7]. Further these methods do not require any material to be added to the interface between the components (a negligible amount is added in the case of solvent welding). This may provide an advantage for automated assembly since there is no consumable material to be replaced; however, this is a problem for the IFB, which has a concrete shell that is flush with the polystyrene core at the mating surface. Melting polystyrene results in the loss of entrapped air which reduces the volume of the polystyrene in the melt pool ^[8]. Reduction of the polystyrene volume at the surface of the IFB would therefore result in a gap between the two substrates, and much poorer bond strength.

As demonstrated by the current corner brick manufacturing process, adhesives can also be used to bond polystyrene to other substrates. This method involves applying a volume of liquid between the two substrates which adheres strongly to both materials, then solidifies to form a semi-permanent bond. This method is used frequently in the aerospace and marine industries where foam cores are adhesively bonded to a high-strength film to create rigid, lightweight sandwich panels. Some adhesives used in the manufacturing of polystyrene sandwich panels include two-part epoxies, thermoset films, and sprayable hot-melt glue ^[9,10]. Two-part epoxies consist of epoxy resin and a hardener which are liquid at room temperature. Combining these two liquids starts an exothermic reaction, in which the epoxy resin undergoes significant cross-linking and forms a thermoset polymer. This mixture is then applied between the two components and held until the epoxy is cured, forming a very strong permanent bond. Thermoset films act in much the same way as two-part epoxies; however, the two parts are combined into a solid film and require heat to start the curing process. These films are applied

between the layers of a sandwich panel, then the entire panel is heated in an oven ^[11]. This type of fabrication is advantageous for automation since the film is applied as a solid, so there is no need for a dispenser. The disadvantage with both types of thermosets, is the curing time, which can be 15 minutes to an hour. Therefore, to achieve high throughput, many bricks would need to be prepared and cured in large batches. Meanwhile, the set time of hot-melt glue in sandwich panels is typically less than 1 minute, which allows for smaller batches to be produced, but the bonds created are not as strong as those of thermoset adhesives. It should be noted that based on current production, the bond created by hot melt glue is already stronger than the cohesive strength of extruded polystyrene.

Lastly, very little could be found on mechanical fastening methods of expanded polystyrene. The only references to this technique were found in fastening EPS sheets to the base layer within an EIFS. In this method, screws are passed through the EPS and threaded into the substrate below, which is usually made of wood or light metals. A large washer is used to catch the head of the screw and distribute the force over the EPS surface. This method may be used to reduce the strain from thermal expansion between the EPS and substrate ^[12]; however, Sto Corp specifically advises against the use of mechanical fastening due to its worse performance against negative wind pressures. Under high wind-loads, the smaller supported area of mechanical fasteners compared to adhesive methods results in EPS failure or fastener “pull-out” ^[13]. Aside from this issue, the geometry of the IFB does not easily support mechanical fastening. For a straight fastener to pass through both EPS cores, the fastener would need to be inserted through the concrete base of one or both components. Additionally, to reduce the risk of pull-out, this fastener would require a method of distributing the force across the brick which is difficult to achieve without altering the outside surfaces. Based on the issue of pull-out, and

the geometry of the corner bricks, mechanical fastening was not considered a viable option for joining.

Based on this review of joining methods, continued use of hot melt glue is the most reasonable approach; therefore, the next step was to study existing applications of hot melt glue in automation. The systems identified all used one of three application methods: spraying, rolling, or extruding. Spray type applicators use heated, pressurized glue tanks and wide nozzles to quickly cover large areas with molten adhesive. This allows for high throughput; however, the placement of the glue is not as accurate as for other methods. Examples of applications using spray type systems include sandwich panel and large furniture manufacturing ^[11,14]. Next, roll coaters distribute glue via long rollers which are partially submerged in a tank of molten glue. Rotating the roller produces an even coating of glue on the roller, which can then be transferred to the work piece. This method is often used to join planar substrates such as veneers to plywood in the assembly of tables and desks ^[15]. Lastly, extruder systems are built for high precision with small nozzles and methods to quickly start and stop the flow and include everything from manually operated glue guns to fully automated bead dispensers. Some manual glue guns such as the Power Adhesives TEC 7100 use actuators to apply a consistent force on the glue and can be combined with an electronic timer to control the dispense duration for low-cost automated systems. In systems for high-throughput manufacturing, such as cardboard box assembly, the glue is heated to a very high temperature inside a pressurized tank to lower the viscosity significantly. Solenoid valves are used to start and stop the flow of glue to dispense a consistent quantity onto each component as it passes by. Where greater precision is needed, extruder type applicators have been combined with vision systems to measure the size of the dispensed bead and modify the dispensing parameters in real time ^[16]. These advances have allowed the use of hot melt glue in electronics

manufacturing for mounting small components, electrical insulation of connectors, and waterproofing ^[17].

The second process element considered was the method of mechanically aligning and joining the two components. Industrial motion systems come in many forms for different scales of production. Production of cardboard cartons for example, uses a belt-driven assembly for high-throughput manufacturing. These systems use rollers and guides to fold cardboard templates into three-dimensional shapes. Such systems can produce multiple units per second and can account for the drying time of hot-melt glue by using a long section of the belt, located after the gluing station, for drying. Though this process could be adapted for corner brick manufacturing, the large footprint required and high upfront costs of these continuous systems makes them impractical for this situation ^[18].

Industrial manipulator robots provide flexibility in the geometry and precision that can be achieved during material handling. These robots consist of an effector device at the end of a manipulator, which can move and orient itself in multiple dimensions. This effector can be a gripper for grabbing components, or a tool such as a spot welder. Grippers can use suction or mechanical clamping to grab components temporarily for repositioning. The manipulator often includes redundant degrees of freedom and can be programmed to position and orient the effector as needed by the application. Once programmed, the robot can carry out its motion very rapidly due to closed-loop control, achieving more than 100 cycles per minute in some applications. Further, vision systems can be implemented to accommodate variations in the part position and part geometry which can affect assembly ^[19]. Due to their speed and flexibility, these robots are used heavily in the automotive industry for assembly, disassembly, and welding ^[20]. Interestingly, these robots have also been used in the assembly of sandwich panels to move large sheets of foam core with vacuum grippers ^[21]. Based on this, two robots

of this type could be used for corner brick assembly to grab the components, run one of the components across a roll coater, then join the two components together. For higher throughput, this would require a separate method of holding the components together which does not occupy the robots. Such a system would provide inherent flexibility for all styles of corner bricks but would increase cost significantly and require two manipulators for each assembly carried out simultaneously.

Another option is to use a custom arrangement of actuators to move and align the components. Electric actuators are used to enable rotational motion of parts and can also be combined with lead screws or pulleys to convert the rotational motion into linear motion. Manipulator arms are an example of many electric servo motors chained together to simulate the movement of multiple joints. Meanwhile, electric linear actuators have been used for similar tasks to corner brick alignment such as the positioning laptops for automated disassembly ^[22]. In this application, the laptops were loaded by hand, then pushed into position by electric actuators. The second class of actuators most often used in automation is that of pneumatic actuators. Due to their high power density and fast actuation time, pneumatic actuators are used extensively as grippers in assembly systems. Pneumatic actuators are also used to push components as in the case of high-speed sorting modules. These systems use a combination of computer and sensors to classify components moving across a belt, and one or more pneumatic actuators to push the components from the belt into corresponding bins ^[23,24].

Chapter 2: Concept Development

2.1 Production goals

Because the scope of the challenge was not fully known at the time of concept generation, no hard requirements were set for the production concept. Instead, the company set a ceiling for the production rate of 3000 bricks per eight-hour shift, to match the rate of the steps leading up to corner brick assembly. At that rate, the bottleneck from the corner brick assembly would be eliminated and further efforts would provide more benefit directed elsewhere. Additionally, while the company did not set a requirement for the number of workers needed for the new process, representatives expressed desire for the process to require as few operators as possible. In this way, workers currently assigned to corner brick production could be reassigned to other labor-intensive steps. Lastly, no requirement was set for the cost of the production concept, which will be assessed with respect to the level of throughput and quantity of operators that the concept is expected to achieve.

2.2 Initial Observational Visit

In March 2019, the team visited the production facility to observe the current corner brick assembly process and establish baseline performance metrics. Specific items of interest during this visit were as follows:

- Average time to produce a corner brick
- Method of alignment
- Amount and distribution of glue on the parts
- Time that part must be held before handling as a single unit
- Average gap size of the two components in a completed brick
- Approximate percentage of acceptable bricks
- Amount of floor space used by corner brick assembly

On the day of the visit, 4-6 workers were assigned to corner brick assembly. The space set aside for the assembly station was roughly 10' x 10' consisting of a long table and space for workers to sit on each side. The station was positioned just at the outlet of the brick cutting machine, where long strips are cut to shape using a water jet cutter. After cutting, the components were set aside on racks to dry overnight. Meanwhile, workers grabbed components cut the previous day in groups of roughly 10 long and 10 short components and moved them to their station for assembly. Each station was supplied with a TEC 3400-43 glue gun and wooden alignment jig as seen below in figure 4. Workers were observed applying glue to the small component, then placing both components in the jig as shown in figure 4B. The workers then aligned the components with the right angle of the jig and applied force to close the gap between the components. The force was held for 15-25 seconds for the glue to harden, then the completed brick, shown in figure 4C was set aside. The preceding steps were repeated for each set of components, then the completed bricks were moved to the finishing station to blend the seam with additional concrete.

Figure 4: Manual Process of Assembling Corner Bricks

This process was observed for about 10 minutes to gather qualitative data on the performance metrics listed previously. Using a stopwatch, the time for each worker to produce a brick was measured to be between 36 and 45 seconds depending on the worker. The stopwatch was started when the worker picked up one set of components from the stack and stopped when the worker picked up the next set of components in the stack. Thus, this time includes the full process of producing a brick but does not include the time to gather more components from the rack or the time to replace the glue. The amount of glue could not be measured directly but was instead based on company logs of the number of glue slugs used per day vs. the number of corner bricks produced. From this, it was estimated that the workers produce 35-40 bricks from each glue slug. The workers typically dispensed the glue using a back and forth pattern similar to that shown in figure 5, before pressing the components together. Some workers also spread the glue around further by sliding the two components together in a circular pattern before placing the components in the jig.

Figure 5: Observed Glue Dispensing Technique

In some cases, the bricks fell apart after being removed from the jig. Most often, this was a result of removing the bricks before the glue had hardened. In these cases, the bricks could sometimes be reassembled by placing them back in the jig for a short amount of time. In cases where the glue had hardened and the bricks could not be reassembled, the bricks were

discarded. This occurred only a few times during the observational period, so the exact proportion of successful bricks could not be identified but appeared to be around 95%. Lastly, data on the gap sizes of the completed bricks could not be measured directly but appeared to be less than 1mm. Figure 4C shows a representative part with a gap size of around 1mm at the front edge.

2.3 Concept Generation Phase

The initial concepts were grouped into three categories based on their level of automation. With this approach, the goal was to cover the full range of costs and further explore those concepts which provide the most benefit at each cost level. At the lowest level were concepts with no automation, aimed at increasing worker efficiency with low up-front cost. One such concept considered adding clamps to the wooden jigs used by the workers that could hold the bricks in place while the glue dries. To provide further benefit, the wooden jigs were to be arranged on an indexing table in front of the worker, such that the worker could load each jig as the table rotates, then unload the bricks after a full rotation. This provides an immediate benefit in that each worker would spend less time holding bricks; however, this solution is not scalable as each worker requires a much larger footprint.

In the middle tier, were concepts which use automation to assemble many bricks at once but require a worker to perform small tasks and manage the operation. The worker efficiency of these concepts is expected to be many times greater than the manual concepts but is still somewhat limited. For example, the concept which was ultimately selected for this study relies on the operator to load and unload each component but uses automated solutions to perform the rest of the assembly. In this case, the production reaches its maximum efficiency when the

worker's time is utilized 100% by loading and unloading. To increase the production beyond this rate would require additional workers or a faster method of loading and unloading.

Lastly, with the fully automated concepts, the goal was to completely remove the operators from the assembly process to allow for much greater efficiency. Many of these concepts were simply add-ons to the middle-tier concepts which automate the remaining steps. For example, the chosen concept could be coupled with an automated feeder to load the components, and another mechanism to move the finished bricks to the next station. With this system, the production rate would be limited only by the scale of the machine. Further, once the feeder is loaded, production could proceed continuously without dependence on the workers' schedules.

2.4 Production Concept

The full-scale concept was based on a modular architecture with various subsystems that can be repeated or upgraded to meet future needs. The final concept delivered in May of 2019 consists of six subsystems with the following functions:

1. Frame - Holds the other subsystems and maintain their alignment with respect to one another.
2. Glue Subsystem – Dispenses a consistent amount of hot-melt glue onto each brick and informs the rest of the system when the glue is too low and must be replaced
3. Linear Slide – Moves the glue subsystem over each brick during dispensing, then moves the glue system out of the way to allow for the AJU to complete the assembly. The linear slide allows a single glue subsystem to handle many units in quick succession.
4. Automated Joining Unit (AJU) – Aligns and holds one set of components such that the front component is directly under the axis of the glue subsystem. After the glue is applied, the AJU moves the two components together and applies force until the joint solidifies.

5. Control Box – Houses the power supplies, control computers, and stepper driver for the linear slide. Also contains the user interface consisting of a display screen and buttons to advance or pause the operation.
6. Control Software – Runs on the computer within the control box to synchronize the operation of all other subsystems based on feedback from the other systems.

To meet a production rate of 2400 bricks per shift, it was determined that the large-scale machine should house eight AJUs within reach of a single operator. Figure 6 shows the estimated timeline of the forward operation for a machine with eight joining units. In this figure, each row represents a different task in the sequence and the bars to the right represent the approximate start and end times of the tasks. The colors of the bars represent the subsystem that is responsible for the given task with blue, green, and yellow representing the AJU, linear slide, and glue subsystems respectively. Meanwhile, the orange bars represent tasks which must be completed by the operator.

Figure 6: Estimated Timing of Events for Full-Scale Concept

The sequence of events presented assumes that the operator is standing by to load and unload the components one at a time as they are released from the machine. Following this sequence of events, the operator would complete eight bricks every 95s for a throughput of 5.0 bricks per minute (12s per brick). Using this technique, the operator is occupied for 55 seconds during each 95s cycle. Conversely, this leaves 40 seconds per cycle for the operator to perform other work, such as operating a second assembly machine nearby. With a second machine, we

can assume that the operator will alternate between the two and spend 55 seconds occupied with each machine. This technique would result in the operator producing up to 16 bricks in 110s or 8.7 bricks per minute (6.9s per brick). With this sequence, the operator is now occupied 100%, so there would be no benefit to adding additional machines.

The second mode of operation involves the operator loading and unloading all eight bricks of the machine at one time. With this schedule, the total time for the cycle is 135s, with the operator being occupied for 44s and having 91s of idle time between operations. If we add a second machine, the operator can now spend 44 seconds of this downtime operating the second machine and produce 16 bricks in 135s or 7.1 bricks per minute at 65% utilization. Adding a third machine, with 44 seconds of operational time, a single operator could potentially produce 24 bricks in 135s (10.6 bricks per minute).

The above throughput estimates are the maximum that could be expected and there are many factors not included that would lead to lower production rates in practice. First, human operators cannot be expected to perform at 100% utilization for any length of time. During the short test runs, from which the task durations are based, the operators attempted to maximize throughput to validate the concept. During daily use however, the schedule should allow additional time for the operator tasks, and for breaks at regular intervals, to reduce stress and increase well-being.

These estimates also do not include the machine start-up sequence, time for the worker to gather more components, time for the worker to travel between machines, or time for glue replacement operations. The start-up time of the prototype is approximately 5 minutes and is determined by the time for the glue gun to reach its operating temperature. This time must be added at the beginning of each shift, and after any extended break in production such as a lunch

break. Additionally, each time the computers are powered on, the motors must complete their homing sequence and the slide must return to home, adding an additional 30s.

Depending on the number of AJUs, the rest of the subsystems must be designed to accommodate the number of AJUs and the desired throughput. For example, the frame and linear slide must be long enough to fit the width of the AJUs and the minimum spacing between them. With more AJUs, the glue system will exhaust its supply more quickly, so faster loading or a larger quantity of glue is necessary. The software also needs to handle the added inputs from more AJUs and be written to maintain consistent timings despite the additional overhead. Additionally, increasing the dispensing rate of the glue subsystem or increasing the travel speed of the linear slide are both upgrades to subsystems that could be made to achieve higher throughput using the same number of AJUs.

2.5 Minimum Viable Product Prototype

The prototype that was built and delivered was based on a minimum viable product to validate the large-scale concept. The prototype contains two AJUs and is shown in figure 7 below with the subsystems called out according to the list presented in section 2.4. Though this machine may not achieve the same benefits from scaling as the larger machine, a two AJU system requires the use of a linear slide and a modular software structure to handle the parallel operation of both AJUs. Further, as shown in the timing diagram, the amount of time added to the overall process with each AJU is fixed. Thus, by measuring the actual time of the two-unit system, an accurate estimate can be made for a system containing any number of units.

Figure 7: Render and Constructed Prototype Delivered to The Company

Chapter 3: Prototype Design

3.1 Automated Joining Unit

The objective of the Automated Joining Unit (AJU) development was to create a repeatable subsystem to align and hold each pair of components. Further, the AJU was to contain its own interfacing hardware (i.e. circuit boards, sensors, pneumatic hardware) to minimize rework when adding additional AJUs. With this modular approach, the joining concept could be proven with the first AJU, then the AJU design could be replicated to meet the desired scale of production.

Figure 8: Two AJUs Installed in the Final Prototype

3.1.1 AJU Mechanical Concept

The concept of the AJU mechanism is to align each component within its own channel, then rotate the right-side channel, to press the two components together after the glue is applied. By precisely setting the position of each channel and the locations of the components within them, the components can be rapidly joined before the glue hardens or drips down the component.

Figure 9 shows a side view of the geometry used in the final design to align the parts. The channels begin the cycle in the horizontal position where the components are loaded and aligned, and glue is dispensed onto the front component (left of figure). After the glue is applied, the side component (right) rotates 90° about the axis of rotation to mate the two faces and hold the components together. The axis of rotation is outside of the cross-section of the component which produces a translation of the side component along with the rotation. With this method, there is a physical gap between the components before the rotation, which is required for the horizontal positioning mechanism and also allows for easier maneuvering of the glue subsystem.

Figure 9: Mechanism of AJU Motion for Joining the Components

The horizontal positioning mechanism, henceforth referred to as the “stopper”, fits within the gap between the channels and is the main feature to control the position of the components with respect to the axis of rotation. The stopper has two vertical faces which restrict the horizontal position of each component during loading and alignment. After alignment, the stopper is pulled down to allow the right channel to rotate without obstruction. Figure 10 shows the stopper in the active and retracted positions with the dashed arc representing the smallest radius of the flipper channel that is most likely to contact the stopper.

Figure 10: Stopper Positions and Safe Zone for the Retracted Stopper

Adjustment of the stopper faces in the horizontal direction is required for optimal alignment of the parts on each AJU. The goal of the stopper is to set the 45° faces of both components to be coincident with the axis of rotation such that all points match up with their corresponding points on the opposite part. Ideally, the location of the stopper faces is set by the geometry; however, small variations of each AJU and of the front edges of the brick components can change the optimal location. The axis of rotation of each AJU is determined by its dimensions; therefore, each AJU will require slightly different stopper positions due to the tolerances of its assembly. Also, unlike the ideal component which has a perfect 45° angle and a front edge with zero radius of curvature, these components can have slightly larger or smaller angles, and rounded front edges. Depending on the radius of this edge, there is some deviation between the true front edge and that of the ideal case as shown in figure 11. Thus, when the front edge is pushed against the stopper, the component is closer to the axis of rotation than in the ideal case and the ideal alignment is lost. To fix this, the stopper face must be moved away from the axis of rotation in the same magnitude as this deviation.

Figure 11: Deviation of the Front Edge of the Components

3.1.2 AJU Construction

The realized concept uses two aluminum U-channels to hold the two components with the front edges machined at 45° angles to allow for the channels to meet at a right angle. The left channel is 10.25" long and fixed to a precision-ground aluminum base plate (6" x 24" x $\frac{1}{4}$ ") via three sections of the same 5"x2" aluminum channel that hold the brick. Two right-angle brackets are mounted near the center of the plate, with a $\frac{1}{4}$ " hole in the side of each to serve as the axis of rotation of the right-side channel. The right-side channel is 6.25" long and mounted to an inverted section of the U-channel with holes on either side to line up with the axis of rotation. This channel also rests on another small channel, fixed to the plate, when in the down position. To allow for rotation, two $\frac{1}{4}$ " shoulder screws are inserted through the angle brackets and the walls of the inverted channel with Roulon washers separating the aluminum components to reduce friction. The positions of the fixed channel and the right channel are set such that the aligned components extend $\frac{1}{8}$ " beyond the end of the channels. This prevents the two channels from contacting one another and ensures that the components receive the full torque applied at this point. Lastly, the inverted channel has a cut-out in the area where the stopper resides and a timing-belt pulley mounted in line with the axis of rotation on the back

side. This pulley is where the torque is applied, via a timing belt, to rotate the right-side channel.

The stopper shown below in figure 12 is made of slotted T-frame components from 80/20 Incorporated. The base is a 3” long section of the 2012 profile with a half inch thickness and two T-slots along its length. The vertical surfaces are made of two 3034 angle brackets (1” x 1”) mounted to the 2012 profile via the T-slots. Adjustment of the vertical faces is performed by loosening the two screws holding the angle bracket, then manually moving the stopper face forward or backward and retightening the screws. For fine adjustment, shims of known thickness can be inserted between the stopper face and the channel to measure the distance, then the new stopper location can be set by adding or removing shims with the desired thickness.

Figure 12: Adjustable Stopper Viewed From Top (Left) and Side (Right)

The 80/20 assembly sits on a guided linear actuator mounted to the base plate below. The height and travel distance of the actuator are vital in ensuring that the components contact the stopper when in the up position, and that the stopper does not obstruct the rotation when in the down position. The use of a guided actuator was necessary to reduce the amount of lateral

motion when the components are pressed against the stopper. More details on the selection process of the actuators can be found in the next section.

Each channel has two rotational actuators to align and hold the components as shown below in figure 13. The actuators mounted through the bottom of each channel are responsible for pushing the bricks against the stopper faces, while the actuators mounted above the channels hold the components in this position until the assembly is complete. Two strips of spring steel are mounted inside the channel to apply pressure and guide the bricks along the front wall. The angle of the spring steel is adjustable via a screw threaded into the holding brackets. By adjusting the angle, the lateral force on the bricks can be controlled to properly align the components and allow the completed brick to be easily removed. Though additional actuators could have been used to perform this function, this method saves significant cost and software complexity compared to the alternative.

Figure 13: Pneumatic Actuators on AJU viewed from Side (Left) and Front (Right)

To reduce the friction forces from sliding the components and reduce the wear on the aluminum channel, the base, inside wall, and spring steel surfaces of each channel are covered with a pre-cut adhesive melamine sheet. These sheets must be replaced about every three days

at the maximum expected production rate and are the only consumable items of the AJU design. Despite the short lifetime, these sheets provide a low-cost preventative measure against direct wear on the aluminum channels. Additionally, these sheets allow the pushers to move the components into alignment from a much wider range of starting positions. Further explanation for their use is given in the following section on actuator selection.

The last component of the AJU mechanical design is the flipping mechanism for the right channel. This mechanism uses a motor mounted off-axis from the axis of rotation with a timing belt to transfer the torque to the channel as shown in figure 14. This timing belt allows for a greater gear ratio, and significantly reduced width of the AJU, than would be possible with the motor mounted along the axis of rotation of the channel.

Figure 14: Electric Flipper Motor with Tension Block, and Timing Belt

One drawback of the belt, however, is that it must be kept under tension to reliably transfer torque. To add tension to the system, the motor bracket is mounted to slots in the baseplate which run parallel to the belt. Next to the motor bracket is a fixed block with two thru holes running horizontally and aligned with two threaded holes on the motor bracket. After the belt

is looped around the pulleys, screws are inserted through the block then threaded into the motor mounting bracket. Turning these screws clockwise pulls the mounting bracket toward the block and increases the tension on the belt. Once the desired tension is reached, the bracket is tightened via the four screws.

3.1.2.1 Actuator Selection

A large portion of the development time for this project was used to determine appropriate actuators to achieve the concept described earlier. Pneumatic actuators can produce high force in a small volume, provided the appropriate pneumatic lines and a powerful air compressor. These actuators are simple to control, typically requiring one or two solenoid valves to move the actuator to either limit. Further, the maximum force and speed of these actuators can be controlled by regulating the input pressure and installing flow-control valves in series to limit the volumetric flow rate. Lastly, the cost of these actuators is typically lower than for an electric motor of similar power.

Along with these advantages, pneumatic actuators have many disadvantages which make them not suitable for some applications. To start, pneumatic actuators require a compressor to create the pressure difference. These compressors are large compared to the actuators and can be quite loud when the compressor is running. Additionally, the compressors themselves can consume a high amount of power, limiting any potential savings from using pneumatics vs electric motors. Second, pneumatic actuators require air lines to the compressor whose diameter is determined by the pressure and flow rates required. These can become cumbersome to route to each actuator and special care must be taken to ensure that the hoses do not become kinked. Lastly, and most importantly for this application, closed-loop control of a pneumatic actuator is usually not possible due to the simple on/off nature of their actuation. When the

actuator is pressurized, the resulting force (or torque) is constant until the pressure is released. Thus, without external springs, dampers, or forces, the actuator will accelerate until a hard stop is reached.

The second class of actuators typically used is that of electric motors which offer unique capabilities and challenges compared to pneumatic actuators. Electric motors are produced in a wide range of sizes from micro-motors less than 1cm in diameter, to those used in utility-scale power generation stations with diameters of several meters. There are also multiple types of electric motors for different needs such as direct current (DC), alternating current (AC), stepper, and servo type motors. In general, the disadvantages of all types of electric motors are their lower power density and higher cost compared to pneumatic actuators in the range of sizes where the two classes overlap. Additionally, almost all electric motors produce mechanical power in the form of a rotating shaft, so creating linear motion with an electric motor requires extra hardware. To generate linear motion, the rotating shaft is often coupled with a lead screw, belt-drive, or a rack and pinion system. This is in contrast to pneumatic actuators which are readily available with straight pistons for linear motion, or vanes for rotational motion.

As mentioned, the nature of electric motors allows for feedback control that is not possible with pneumatic actuators. If desired, additional hardware can be added to the motors such as encoders and tachometers to measure the angular displacement and angular velocity respectively. Feedback from these devices can be used as control inputs to control the position, velocity, and torque of the motor and to generate complex motion paths. In the case of stepper motors, basic control is possible by tracking the inputs to the stepper motor driver or monitoring feedback signals from the driver depending on the model of driver. Lastly, servo motors

combine a motor and controller in a single package to provide closed-loop control using simple input signals.

Selection of the actuators was performed on a case-by-case basis for each application: stopper activation, pushing the components, holding the components, and flipping the right-side channel. For the stopper activation, the concept calls for linear motion, so only pneumatic actuators were considered. Key parameters of the selection process were the height of the actuator in the retracted and extended positions, the area of the footprint, mounting options, and the resistance of the actuator to side loading. The height of the extended actuator, plus the 80/20 components which make up the stopper assembly must be greater than the height of the channel (5 3/16"). When in the retracted position, the full stopper must be inside the arc shown in figure 10 which has a height of 4 13/16" at the stopper faces. The heights identified are absolute minimum and maximum to achieve the desired function, so the conservative limits were set which exceed these limits by 5mm in either direction.

The goal when selecting the pusher and holder actuators meanwhile, was to select the smallest actuators which could function reliably. Size was especially important for the actuators on the right-side channel as these actuators add weight and inertia to the moving flipper channel. Additionally, the two-AJU prototype has four pusher actuators and four holder actuators so the cost of each actuator has a large impact on the total budget. Therefore, pneumatic rotary actuators were selected to perform these functions due to their high power density and low cost when compared to electric actuators.

To select the actuators, a test channel was machined with mounting points for an actuator to either push or hold the component. An NCRB1BW-20 vane-style rotary actuator from SMC from a previous project was used as a baseline. Initial tests with a straight arm showed that the actuator could hold the component in place with the channel at all angles and during moderate

acceleration of the channel when the pressure was set to 60psi. When configured with the actuator as a pusher, the tests showed erratic acceleration of the component as a result of high friction between the component and the bare aluminum channel. These tests also showed that the straight arm contacts the back corner of the component first, resulting in a lateral force which pushes the component to the inside wall. When the component was loaded more than ½” behind the stopper position, the angle of the arm at contact was too small, resulting in a high side force and small forward force and jamming the mechanism. To reduce the risk of jamming, the straight pusher arm was swapped with a curved arm which is more likely to contact the back face of the component and at a greater angle than before. This increased the range of positions from which the components could be moved without jamming, but still resulted in erratic acceleration.

To create smoother motion of the component and reduce the wear on the channel, a number of low-friction coatings were considered such as glazed ceramic, Teflon spray, and adhesive melamine. Ceramic provides very high abrasion resistance but proved difficult to cut to shape and fix to the channel. Also, the friction on the ceramic test sheets was only marginally lower than on the bare aluminum. Teflon spray was considered due to its very low friction but is known to wear away from mild abrasion and could end up in the machine or on the assembled bricks. Lastly, adhesive melamine rolls are produced at low cost to be used as white boards in classrooms. These rolls are easy to cut and apply to the channels and provide a surprising reduction in friction. Additionally, the abrasion resistance of these sheets is adequate as determined from sliding the bricks across the sheets repeatedly to simulate a full day of production. Only after several simulated production days did the tests sheets show enough wear to require replacement. Removal and replacement of the sheets was also quick enough such

that performing this maintenance activity twice per week would not significantly impact the production rate.

As a result of the reduced friction coefficient of the melamine coating, the force of the holder actuator must be increased to produce the same amount of friction. When the testing was repeated with the actuator again in the holder configuration, the actuator required an input pressure of 80psi to achieve the same result as before. This is close to the upper limit of acceptable input pressures for this actuator; therefore, a larger actuator was considered which could produce the same torque at a lower input pressure. Within the same series as the test actuator, the NCRB2BW-30 has a 50% larger bore diameter, resulting in more than double the output torque of the test actuator at the same input pressure. After purchasing a single NCRB2BW-30 actuator, the same tests were successfully performed with the larger actuator and an input pressure of 40psi. To increase interchangeability, the AJU design was modified to use the larger actuator for both the pushers and holders.

The most important factor in selecting the flipper actuator was controllability. As described in section 3.1.1, the flipper actuator must lift the right-side channel from the rear support, rotate the channel 90°, then return the channel to the down position. To rotate the right-side channel, the actuator must overcome the torque due to gravity, which is highest when the channel is in the down position and decreases as the angle increases. Thus, the actuator must provide enough torque to lift the channel from the down position, but then reduce the torque to maintain control. For this reason, a large electric motor from Teknic was selected for the flipper actuator.

3.1.2.2 Pneumatic Actuator Control

To operate the pneumatic actuators, each AJU contains its own hardware to control the airflow to the actuators and therefore requires only a single hose from the machine to supply

the pressurized air. This hardware, consisting of a manifold block and five solenoid valves, is mounted to a plate on the bottom of the AJU as shown in figure 15. The purpose of the manifold block is to route air from the single input port to the five solenoid valves and route exhaust pressure from the AJU to the environment through a set of mufflers. The solenoid valves used in the AJU prototype are two-position valves with only a single digital input. When in the off state, the solenoid valve directs air from its pressure port to the 'A' outlet port. When a 24V signal is applied to the digital input, the solenoid valve moves to block airflow to the 'A' port and changes the airflow from the pressurized port to the 'B' port. Meanwhile, pressure which has built up on the 'A' port is exhausted through the manifold block.

The solenoid valve outputs are connected to the actuator input ports such that the actuator activates when the solenoid is powered, and releases when the power is removed. In either case, the actuator will accelerate until reaction forces from the environment, or from the internal mechanism balance the force from the pressure differential. Thus, by positioning the actuators such that the arms contact the bricks just before the actuator reaches an internal hard stop, the actuators can be used to provide a holding force on the components. It should be noted that due to the use of two-position valves, there is always a pressure differential on the actuator as long as the AJU is pressurized. Thus, the machine includes a primary safety valve at its input to quickly depressurize the system during a safety emergency.

Speed of the actuator arms is controlled by manual flow control valves in line with each hose. These valves restrict air flow through the hose to reduce the rate at which the pressure builds across the ports; thereby reducing the acceleration of the actuator. As shown in figure 15, these flow valves are mounted to a plate on the bottom of the AJU, opposite the manifold block. The valves are arranged in two columns for the 'on' speeds, and the 'off' speeds respectively.

Figure 15: Pneumatic Control Hardware Mounted to Underside of AJU

Feedback from the pneumatic actuators is critical to the safe and reliable operation of the AJUs. For example, if the flipper is actuated before the stopper has retracted, the mechanism will jam, and the flipper motor may shutdown due to excessive torque. Likewise, if the components are loaded incorrectly, the pushers may not be able to move the components into position, resulting in the glue dispensing directly onto the machine. In either case, the solution is to provide the controller with feedback, and make each step dependent on completion of the pre-requisite steps.

To accomplish this, each pneumatic actuator contains an internal magnet which moves with the shaft (or piston), and two solid-state magnetic switches. Figure 16 below shows these sensors on holder 1, with the off sensor mounted in the lower left and the on sensor mounted just to the right of the top-most point on the rear cylinder. The magnet which activates the sensors is mounted inside this cylinder as indicated by the red arc shapes. In the off position, the magnet is just inside the active region of the off sensor. As the holder arm activates, the magnet rotates 90° clockwise and enters the active region of the on sensor.

Figure 16: Pneumatic Actuator Sensor Location and Activation

3.1.2.3 Electric Actuator Description

To move the flipper, the AJU uses a clearpath all-in-one servo motor from Teknic paired with a 10:1 gearbox from Anaheim Automation as shown in figure 13. By itself, this motor has a maximum torque of 3.5Nm which is much more than the torque required to lift the flipper from the down position. During testing however, it was determined that the motor controller is not designed for external torques that vary greatly with angle, such as the torque from the gravitational load of the flipper. Under this loading condition, the motor oscillates wildly while moving the flipper to the down position. Thus, the gearbox was added to reduce the effect of external torques on the motor behavior. This gearbox, combined with the gear ratio of the pulley system, reduces any torque on the flipper by a factor of 12.5 before the torque is applied to the motor shaft. Additionally, the 12.5:1 gear ratio means that the motor shaft must rotate through $3 \frac{1}{8}$ revolutions to move the flipper $\frac{1}{4}$ turn.

In addition to basic position control, this motor can be tuned to generate its own dynamic model of the driven system which may contain both inertia and damping coefficients. The controller can be further configured by the user, via a PC, to limit the maximum torque,

acceleration, and speed of the motor and set position limits to match those of the driven system. In this way, the motor can create a torque curve that will move the system to the commanded position with no overshoot, while staying within the user-specified limits. The absence of overshoot is especially important in this application where the flipper is restricted physically by the rear support in the down position and the fixed channel in the up position. With these physical limits in place, any overshoot in the position would cause the flipper channel to impact the rest of the system, causing potential damage.

An additional benefit of this motor is the simple input method made possible by the intelligent controller. After configuration, the motors require only three digital signals to operate: enable, position, and home. The enable input is used to allow or disallow motion of the motor and is used as a safety switch. For the position input, the motors of the prototype were set to the “move to absolute position (2-position)” mode, in which a single binary input (i.e. on or off) is used to set the commanded position. Depending on the state of this input, the commanded position will be set to one of two positions defined by the user via the configuration interface. These positions are set only once, and then stored in non-volatile memory inside the motor. Normally in this mode, the motor will attempt to reach the exact commanded position even if a physical limit exists before that position. In this case, the motor will increase the torque to reach the position and generate a tracking-error once the torque reaches 100% and the motor is still not in position. For the AJU, we would like the flipper to move approximately 90° , then maintain a small torque, to hold the components together. This was accomplished by configuring the motor to use “clamped” limits which limit the torque to a set value once the position is inside a set range around the commanded position. This mode also prevents a shutdown from tracking error, allowing the set torque to be held until the commanded position is changed.

Lastly, the home input is used to establish the home position of the system which is required after each power up before the motor can be used. Although the motor contains a built-in encoder, many applications allow for multiple rotations of the motor shaft or potentially continuous rotation. In these cases, absolute encoders can be used which track the angle and the number of full rotations; however, the cost, size, and complexity of these systems increases based on the number of rotations which are possible. Instead, the clearpath motors perform a homing operation to find the home position, then use an incremental encoder to track the position relative to the home position. In this way, a single-rotation encoder can be coupled with a software counter to track the position through any number of rotations.

Often, the home signal is generated from a physical sensor on the system which activates when the system is in its home position. For the AJU, this input comes from a proximity sensor which is mounted to the rear flipper support such that the sensor is active when the flipper is in the down position. Upon power up and enable, the motor detects this input then rotates a small amount in the negative direction (toward the up position) until the home sensor deactivates and the motor verifies its dynamic model of the system. The motor then moves to the position commanded by the position input signal. It is important to note that the user-set positions are defined in terms of encoder counts relative to the home position, which is set at 0 counts. Thus, if the home sensor is moved, the commanded positions will also change.

Though the motor has a suite of sensors internally to allow for precise control, the output from the motor to external devices is limited to a single digital output. This output, known as the High-Level Feedback (HLFB), can be configured through the user interface, but can only output one variable at a time such as the ready status, torque, or position. For continuous variables such as torque or position, the variable, as a percentage of some range, is represented as the duty cycle of a PWM wave on the digital output. For this application, the HLFB was set

to “ASG Position, Latched” in which the ‘ASG’ stands for “All Systems Go” and applies when the motor is enabled, has completed homing, and has not performed a safety shutdown. If this is true, the HLFB will output a high signal once the motor has reached its commanded position and maintain that output until the position input is changed.

3.1.2.4 AJU Electronics

The purpose of the AJU interface board, shown in figure 17, is to simplify the electrical connections to the AJU for further modularity. This board mounts to the underside of the AJU and contains connectors and additional hardware for the electronics on each AJU. By including this hardware with each unit, there is less need to modify the control box to fit more components as each unit is added.

Figure 17: AJU Interface Board Top (Left) and Bottom (Right) Views

Figure 17 shows the top and bottom views of the interface boards built for this prototype. Additionally, the functions of each of the connectors is presented in table 1 below.

Table 1: AJU Interface Board Connections

Connector	Function	Connects To
J1	Flipper Motor I/O Signals	Flipper Motor
J2	Flipper Motor Power	Flipper Motor
J3	Solenoid Valve Signals	Solenoid Valves
J4	Sensor Inputs	Actuator Proximity Sensors
J5	AJU Signal Connections	Raspberry Pi Interface Board
J6	AJU 24V Power	Control Box Power 24V Power Supply

The AJU requires 24V power which is delivered via J6 on the interface board. From here, the power is routed to J4, J3, and J2 to provide 24V to the sensors, solenoid valves, and the flipper motor respectively. All other signals between the AJU and the control box are passed through a 24-pin ribbon connector on the J5 connector. This includes the output signals from the proximity sensors and flipper motor, the input signals to activate the solenoid valves, and a 5V reference required for the flipper motor feedback.

The proximity sensors contain an internal open collector circuit which runs on 24V. The sensors have three pins – VCC, Output, and Ground – which are connected to the 11x3 pin header, J4, in the above image. When the sensor is not active, no current can flow from the middle lead to ground, so the voltage of this lead is equal to the source voltage of 24V. When the actuator is within the active region of the sensor, current flows from the middle lead to ground through the transistor, causing the voltage of the middle lead to drop significantly to around 0.2V. By measuring the voltage on the middle lead, the state of the sensor can be determined.

Before the signals can be read by the control computer, the output voltages (Range of 0.2V – 24V) must be converted. The interface board on the Raspberry Pi uses values of 5V and 0V on the input/output pins for high and low respectively and voltages higher than 5V will cause permanent damage if connected directly to the pins. To convert the voltage levels, each sensor

circuit is connected in parallel with a voltage divider. When the sensor is inactive, all current passes through the voltage divider, so the output voltage is determined by the voltage divider equation:

$$V_{out} = V_{cc} * \frac{R_2}{R_1 + R_2}$$

Substituting 24V, 5.1K, and 1K for the variables Vcc, R1, and R2 respectively yields an output voltage of 3.93V. When the sensor is active, the lower resistor is bypassed and current instead flows through the transistor, reducing the voltage to 0.2V as before.

The home sensor uses a slightly different circuit to allow for the output signal to be simultaneously read by the motor and the control computer. Instead of the 5.1K and 1K resistors, the home sensor is connected with a 2K resistor on the top leads, and a 1K resistor in parallel with a 5.1V Zener diode on the bottom leads. When inactive, the voltage divider would result in an output voltage of 8V; however, when the voltage exceeds 5.1V, the Zener diode allows additional current to flow, reducing the output voltage to the Zener voltage. When this 5.1V signal is connected to the motor input, current passes through the motor which is interpreted by the motor's CPU as a high input. This current also drops the voltage by a small amount, allowing the signal to be safely read by the control computer.

The flipper motor has three input signals and one output signal which are connected to J1 in the above figure. When configured for absolute position operation, the three input signals serve the functions of motor enable, motor position, and home input. To register a high input on any of the input pins, the motor requires a 5-24V potential across the positive and negative reference rails for that input pin. On the interface board, the three negative references are all connected to the common ground of the board. For the motor enable and motor position inputs,

the positive reference pins are connected directly to the pins of connector 5. The home input is connected to the output lead of the proximity sensor mounted to the rear flipper support.

The output signal of the motors requires a separate circuit to produce a usable voltage signal. As specified in the motor documentation, the motor output pins are connected to the top and bottom leads of a phototransistor. Instead of producing a voltage directly, the output signal is represented by the current carrying capability of the phototransistor (i.e. current is allowed to flow when the signal is high, but the current is blocked when the signal is low). To read the state of the output, the board supplies 5V to the positive pin of the motor output via a 5.1K pull-up resistor. When the output is high, the positive input is shorted to ground via the phototransistor, reducing the voltage below 2.4V. In this way, the voltage at the positive input pin can be read by the control computer uses standard TTL voltages.

Lastly, the interface board provides the connections to activate the five solenoid valves on each AJU via the J3 header. Each solenoid has a two-pin connector, which requires a 24V, 8mA signal to activate. The five negative reference pins are connected to common ground and the positive pins are connected directly to pins on the J5 connector. Since the solenoids of the two pusher actuators are activated together, a single pin on the J5 connector is branched to two pins on the J3 connector.

3.1.3 Configuring the AJUs

Within the AJU subsystem, there are many configurable parameters which affect the quality of the final product. First, the stopper positions affect the angle and gap size of the final part. If the stoppers are too far apart, the flipper must rotate by a larger angle to generate the same contact force between the bricks, resulting in bricks with angles less than 90° and a gap near the bottom edge. Conversely, if the stopper faces are too close together, the components will

contact sooner, resulting in angles greater than 90° and a gap near the top edge. Meanwhile, shifting both stoppers left or right results in one of the components being shifted along the plane of the cut faces. The difficulty of correctly positioning the stopper faces is further complicated by the slight differences in the parts. Those parts with rougher finishes often had rounder front edges with larger deviations than bricks with smooth finishes. Also, some bricks were cut at angles slightly larger or smaller than 45° or along planes slightly off from perpendicular with the planes of figures 9 and 10.

Rather than adjust the stopper positions for each new lot of components, the goal was to find stopper positions which produce acceptable bricks for all of the components. To perform the adjustment, the stopper faces were initially positioned to be 0.060" from the front edges of the channel using the shim method described in section 3.1.2. A pair of components with smooth finishes was loaded, then the actuators were activated individually to push and hold the components. The flipper was then lifted manually until the two components contacted one another, then the gap size and approximate angle were noted. Next, force was applied to the flipper to observe if the components would shift and what effect this has on the alignment. The process was repeated for two more sets of smooth-finished bricks, then three more times with medium-finished, and rough-finished bricks. In all cases, the final angle was less than 90° , indicating that the stopper faces were too far apart. The process was repeated with additional shims to move the stopper faces farther from the channels and decrease the overall stopper distance until the final angle was roughly 90° . In general, the final angle of the smooth bricks was less than that of the rough bricks due to the difference in the deviation at the front edges. Fortunately, when the components were too close together, applying force to the flipper shifted the components slightly away from the center, resulting in angles closer to 90° .

At this point, the components were still shifted slightly along the plane; therefore, the next step was to adjust the left/right positions of both stoppers while maintaining the distance between the two. Initially, the edge of the fixed component extended past the edge of the small component. To fix this, shims were moved from the left stopper face to the right stopper gap. This moved the left component closer to the axis and the right component farther from the axis, while keeping the distance the same. The total thickness of the shims transferred was increased until the bricks which met at angles closest to 90° showed very little shifting along the mating face. After acceptable positions were set, the test was repeated with glue to determine how the glue effects the final alignment. These tests showed that with the right amount dispensed, the alignment is maintained; however, dispensing too much glue or allowing the glue to reach the concrete would result in larger gaps and increased mating angles.

The next step in optimizing the AJU was to add in the effects of the flipper motor parameters. The flipper motor provides a configuration user interface to adjust many of the dynamic parameters including commanded positions and limits on maximum acceleration, maximum speed, maximum torque. First, position A was set at 0 counts from home, such that the default position has the flipper as close as possible to the home position. Unfortunately, the home sensor has a small range of activation so the homing operation would set the home position just above the rear support where the sensor first activates. This, combined with the backlash of the pulley system, resulted in the flipper moving just above the active region of the home sensor when this position was commanded. Further, the software does not allow positions to be defined which are past the home position in the direction of homing. A fix for this was later implemented in the control software, which disables the motor after the motor completes its movement to the A position. In this way, the weight of the flipper channel can move the channel to the desired position to activate the home sensor. For position B, the counts from

home were initially set around -2500 representing 3.125 rotations (at 800 counts/revolution) in the clockwise direction. Combined with the 12.5:1 gear ratio of the gearbox and pulley combination, this results in the desired 0.25 rotations for the flipper. In practice, the final angle of the flipper with this number of counts was not exactly 90° due to backlash in the system and the slight offset of the home position. To account for this, position B was moved further in the clockwise direction, until the flipper position was approximately 90° with components and glue applied (final position was -2600 counts). Next, the maximum speed and acceleration were increased such that the motion could be completed in about 1s. This is faster than the duration estimated for the flipping action in chapter 2, but still allows for smooth motion with no shifting of the components. Lastly, the torque was limited to 7% of the maximum to provide enough torque to distribute the glue along the mating surface, without pushing the components out of alignment.

3.2 Glue Subsystem

The primary purpose of the glue subsystem is to dispense a consistent quantity of glue onto each brick such that the components form a strong bond with minimal gap between the two components. Secondary goals of the glue subsystem are to limit the degree of human interaction required and reduce the amount of excess glue outside of the mating area from squeezing or dripping. These objectives are derived from the top-level system requirements to increase throughput, maintain the level of quality achieved by the human operators, and reduce the quantity of wasted materials.

3.2.1 Existing Dispensing Systems

Review of existing dispensing systems reveals a wide range of capabilities. At the low end of cost and functionality, are manual systems such as the HotMelt TEC 3400-43 glue guns

currently used for production. This glue gun is available for \$409 at the time of writing; however, one unit was provided by the sponsor for reverse-engineering. This unit uses cylindrical slugs of hot melt glue measuring 45mm tall and 43mm in diameter which must be replaced approximately every 35-45 bricks. The actuation mechanism is entirely mechanical with each trigger pull advancing the plunger by a fixed amount. For slightly higher cost, these handheld systems add pneumatic or electric actuators to apply a consistent dispensing pressure. Some pneumatic glue guns, such as the TEC-6100 from hotmelt, can also be combined with a controller to dispense automatically for a specific duration. Lastly, at the highest price point, are systems for heavy-automation applications such as those produced by Valco Melton and Glue Machinery Corp. These systems store large quantities of glue in heated, pressurized tanks and use solenoid valves to quickly start and stop dispensing. This allows for precise control of the dispense quantity and location. Further, the large tanks allow the system to operate for long durations before more glue is needed. Table 2 presents some examples of the options identified at each level of capability. Where additional capability is needed to meet the requirements, this table also lists the modifications needed and associated costs.

Table 2: Summary of Glue Dispensing Options

Type of System	Examples	Cost (If Available)	Benefits/Drawbacks
Purpose Built Systems:			
Pneumatic Bead Actuator	Valco Melton MS Series	No Quote	+Very large glue tank (10lbs) +Fast, Precise Application
	Glue Machinery Corp. Econogun System	\$3000-\$4000	+Simple control inputs -Cost Prohibitive - Requires 120 psi Air Supply - Requires very high glue temps (>200C)
Additional Costs:		● Air Regulator for separate glue/AJU air pressures	
Buy and Modify:			
Pneumatic Glue Gun	TEC-6100	\$806	+ Consistent Dispense Rate + Easy to Control (Electric On/Off Trigger) - No Indication of Glue Level - Requires 120psi Air supply

			<ul style="list-style-type: none">- Cannot easily adjust dispense rate- Built for manual operation, must be modified for automated use
Pneumatic Glue Gun w/ multiload	TEC-7100	\$1101	Same as TEC-6100 but with + Faster Loading (can hold 3 slugs)
Additional Costs:		<ul style="list-style-type: none">• Air Regulator for separate glue/AJU air pressures• Actuation method of trigger (mechanical, relay, solid-state switch)• Electronics to control time of actuation/send feedback	
Modify TEC 3400-43 (Provided by Sponsor):			
Pneumatic Actuator on Plunger	SMC CXSL15-50	\$123.20	<ul style="list-style-type: none">+ High Power Density- Limited Position Control/Feedback- Actuator must fully retract/extend each dispense cycle- Requires compressed air- Force directly related to air pressure- Must fit inside glue gun body
Pneumatic Actuator on Trigger	SMC CXSL15-50	\$123.20	<ul style="list-style-type: none">+ High Power Density+ Less force needed due to lever arm- Requires compressed air- Force directly related to air pressure- Limited Position Control/Feedback
Electric Actuator on Plunger	Actuonix L16P-50-150 12	\$80	<ul style="list-style-type: none">+ Easily Controllable Speed/Force+ Opportunity for Position/Force Feedback- Lower Power Density Than Pneumatics- Must fit inside glue gun body
Electric Actuator on Trigger			<ul style="list-style-type: none">+ Easily Controllable Speed/Force+ Opportunity for Force Feedback+ Less force needed due to lever arm- Lower Power Density Than Pneumatics- No position feedback
Additional Costs:		<ul style="list-style-type: none">• TEC 3400-43 (\$409 if purchased)• Electronics to control Speed/Force and send feedback• Physical modifications to glue gun and mounting hardware	

3.2.2 Glue Subsystem Approach

With cost playing the largest factor for prototype development, the chosen approach was to modify the TEC 3400-43 glue gun provided by the sponsor. Electric actuation on the plunger was judged to be most viable of the options considered, due to the ability to read position and force from the actuator, and the ability to control the speed, position, and force from within software.

3.2.2.1 Actuator Selection

The concept was first tested with an L16-50-35-12-S linear actuator from Actuonix to determine a baseline for the force on the glue during dispensing. This actuator was ideal for testing due to its small form factor (largest cross section measuring 12mm by 15mm) and its availability within the mechatronics lab; however, the maximum force is limited to 50N (11.2lbf). To set up the test, the plunger and advancing mechanism was removed from the glue gun, and the actuator head was inserted in the hole at the back of the glue gun previously occupied by the plunger as shown below in figure 18. The glue gun was then powered on and allowed to heat to its operating temperature of 165C as indicated by the indicator LED on the temperature control module. Next, 12V was applied to the actuator leads to extend the actuator at its maximum speed until the tip of the actuator contacted the glue slug. At this point, the actuator stalled and was not able to dispense the glue; therefore, a more powerful actuator would need to be procured.

Figure 18: Approximate Location of Actuator Inside Glue Gun Shell

To gain a better quantify the required force, a second experiment was conducted using a force-sensitive resistor (FSR) to measure the force during a regular dispense cycle. A typical FSR has an active area consisting of two conductive layers separated by a small gap. As pressure is applied to the active area, the contact between the two layers increases and the resistance between the two layers decreases. It is common to use the FSR in series with a pull-down resistor of know resistance to generate a voltage dividing circuit similar to that shown below in figure 19 ^[25]. Using this circuit, the change in resistance of the FSR can be used to drive a change in voltage at the output for easier measurement. The relationship between the FSR resistance and the output voltage is also presented below. It is important to note that the FSR resistance does not vary linearly with applied force, nor does the equation below describe a linear relationship between the FSR resistance and the output voltage. Further, the properties of two FSRs from the same manufacturer can vary by up to 10%, so each FSR must be calibrated before any useful information can be gathered from its use.

Figure 19: Voltage Divider Used to Measure FSR Resistance

For this experiment, an Interlink 402 FSR was selected with a circular active area of 0.125 in² and a force range of 0-20 lb. (0 to 100N). The resistance of this FSR varies from infinite

resistance (i.e. open circuit) when no force is applied, to 200Ω at the maximum rated force of 20lbs. Thus, when using a $10K\Omega$ pull-down resistor and a supply voltage of 5V, the output voltage should vary between 0V and 4.9V based on the equation above.

Once the FSR was selected, the required calibration was performed to generate the voltage/force curve for the specific model. To calibrate the FSR, the circuit in figure 19 was constructed including the $10K\Omega$ resistor and 5V supply voltage from a benchtop power supply. From available supplies, items with known weights of 2, 10, 12, and 20lbs were gathered to use as data points. A washer of negligible weight was placed on top of the FSR to ensure that the weight of each item was distributed within the active area of the sensor. The output voltage was measured first with no weight on the sensor, then once more for each item that was placed on the sensor. In this configuration with a $10K\Omega$ pull-down resistor, the voltage plateaued at 10lbs, with additional weight producing a negligible change in the voltage. Thus, the same experiment was conducted using a $1K\Omega$ pull-down resistor for which the data is shown below in table 3.

Table 3: Force Sensing Resistor Calibration Data

1K Resistor	
Weight (lbs)	Voltage
0	0.004
2	1.61
10	2.32
12	2.65
20	3.11

With data points covering the full measurement range of the sensor, the FSR could then be used to measure the actual force on the glue during dispensing. To accomplish this, the glue gun was reassembled and the FSR placed under the plunger. A metal nozzle was also inserted between the FSR and the glue gun to concentrate the force onto the active area of the FSR. The

glue gun was then heated to 160C as indicated by the indicator LED on the temperature control module. The plunger was advanced manually via the trigger to achieve different levels of glue flow rates while the voltage was measured using the same circuit as the calibration trials. Data and observations from the two trials are presented below in table 4.

Table 4: Voltage Outputs Relative to Glue Flow

Glue Flow w/ 1K Resistor					
Trial 1			Trial 2		
	Voltage	Force (lbs)		Voltage	Force (lbs)
No Glue Flow	0-2.7V	0-12	No Glue Flow	0-2.7V	0-12
Glue Flow Starts	2.8V	12	Glue Flow Starts	2.75V	12
Optimal Glue Flow	~3V	~18	Optimal Glue Flow	~3V	~18
Max Glue Flow	3.3V	>20	Max Glue Flow	3.3-3.5V	>20

In the data above, “Optimal Glue Flow” refers to the rate at which a steady stream of glue is dispensed with minimal discomfort to the operator. Fortunately, this flow rate occurs at a force which is within the measurement range of the sensor. Higher flow rates were achieved during this testing by gripping the lever very tightly, but the forces to achieve these flow rates were outside the range of the sensor. Further, the minimum force required to start dispensing glue was measured to be around 12lbs which explains the inability of the previously used Actuonix actuator to dispense glue. Given this data, the force capability of the selected actuator must be at least 12lbs, and ideally greater than 18lbs to match the optimal flow rate achieved by the human operator.

The Actuonix L16-50-150-12-P is within the same family as the L16-50-35-12-S actuator used previously, but with a higher gear ratio (150:1 vs 35:1) to achieve much greater forces at the cost of lower speed. The speed/force characteristics of the two actuators are represented by the red and blue curves in the left plot of figure 20 below.

Figure 20: Speed, Force, and Current Characteristics of Actuonix L16 Series Actuators

As shown above, the same actuator can achieve 220N (49.3 lbs) of applied force when the internal gearbox is configured for a 150:1 ratio. Although there is a speed reduction when using the higher gear ratio, the speed never drops below 3mm/s within the operating range of this actuator. This speed, which would correlate with full depletion of the glue slug in 15s, is well-above the observed rates during testing. Thus, the L16-50-150-12-P, with its 150:1 gear ratio, meets the speed and force requirements as determined from this testing.

The L16-50-150-12-P actuator also contains many features which make it ideal for use with a controller. The “P” suffix at the end of its name indicates that this actuator has an integrated linear potentiometer to give feedback on the current actuator position. This actuator can also be used with the Linear Actuator Control (LAC) board from Actuonix, which provides closed loop control of the desired position using the feedback from the potentiometer. As will be discussed in section 3.2.2.3 however, the function of this board was instead replicated using an Arduino and H-bridge driver to achieve greater control. Additionally, by forgoing the LAC board, the cost of the actuator was reduced to only \$80.

3.2.2.2 Glue Gun Modifications

Integrating the actuator into the glue gun required a number of small modifications to the glue gun body and custom components to be machined. First, the hole at the back of the glue gun was expanded using a Dremel and file to fit the largest cross section of the actuator. Next, the disk at the end of the original plunger was removed and a hole was drilled in the center of the disk to fit around the removable tip of the actuator. The disk was secured to the tip with a strong adhesive, then the two pieces were screwed back onto the actuator once the actuator was inside the glue gun body.

To restrict the actuator along its axis, a bracket (Figure 21) was created to hold the hole at the back of the actuator and thread into the plastic shell of the glue gun. First, a plate was machined from ¼” thick aluminum with a hole in the center to secure the included clevis for mounting to the back of the actuator. Two more holes were drilled at the ends of the plate, through which two M6 x 60mm bolts were secured. These bolts also pass through slots on angle brackets, of which one face of each bracket is pressed against the right and left sides of the glue gun body. On the left side of the glue gun is a tapped hole, through which the plunger tension adjustment screw, shown in figure 3 above, had been inserted in the original design. Since the plunger was removed in the previous step, this adjustment screw is no longer needed; thus, this hole was used as a mounting point for the angle bracket on the left side. On the right side, shown figure 21, a hole (1 1/32” Dia.) was drilled into the body of the glue gun at the same location as the existing hole on the opposite side and an M6 tapping insert was threaded into the hole to secure the angle bracket on that side.

Figure 21: Glue Gun Handle Removed and Actuator Mounting Bracket Installed

To hold the glue gun in a vertical position, a large angle bracket was machined with a hole in the base to hold the tapered nose of the glue gun body. To prevent lateral motion, the angle bracket contains a groove into which the hook at the top of the glue gun, behind the open glue chamber, is inserted. An adjustable holder bar was also added to hold the hook within the groove. This method allowed for slight misalignment of the glue gun tip less than 1cm, which was deemed sufficient at this stage.

Lastly, the electronics in the handle, consisting of a screw terminal block for live, neutral, and ground wires, as well as a 110V AC to 5V DC converter for the PCB, was moved into the lower section of the body which formerly contained the steel linkage to advance the plunger. The handle was then removed to reduce the overall size of the glue gun and the power cable was secured to the body using cable ties.

3.2.2.3 Electronics Design

The actuator connector has five pins for which the functions are given in table 5 below.

Table 5: Actuonix Linear Actuator Signal Table

Pin	Color	Function
1	Orange	Feedback Potentiometer Negative Reference
2	Purple	Feedback Potentiometer Wiper
3	Red	Motor V+ (12V)
4	Black	Motor V- (Ground)
5	Yellow	Feedback Potentiometer Negative Reference

The actuator extends when a positive 12V potential is applied across pins 3 and 4 and retracts when the polarity of the 12V signal on pins 3 and 4 is reversed. An L298 Dual H-Bridge Motor Driver from RobotShop was used to quickly reverse the polarity and allow for input signals at 5V rather than 12V. Each full H-Bridge circuit on this board has three inputs: I1, I2, and Enable as shown in the circuit diagram in figure 22 below.

Figure 22: H-Bridge Wiring Diagram for the Glue Subsystem

The I1 and I2 inputs control the direction of the current through the device attached to OUT1 and OUT2 and the enable input allows the current to flow when set high and disallows current when this input is set low. Further, this H-Bridge uses TTL logic levels for inputs and allows for pulse-width modulation (PWM) input signals. Lastly, this board has a jumper between the output (pin 1 on the bottom of bridge A) and ground, which can be removed and replaced with a connection to ground through an external sensing resistor. When a sensing resistor is used, all of the current through the bridge passes through this resistor, creating a voltage drop across the resistor proportional to the current through the bridge. The circuit created for this project makes use of a $1\Omega \pm 1\%$, 20W resistor which can safely measure currents up to 4.4A and produces a voltage approximately equal to the value of the current measured in Amps. This current value can then be converted to a force value by interpolating the red curve on the Force vs. Current graph in figure 5. As will be discussed, the force feedback was a necessary component of the control software to determine when the plunger was acting on the glue, and to protect the actuator from stalling.

The feedback potentiometer uses the voltage divider principle to produce an output voltage which varies linearly with the position of the actuator. Between pins 1 and 5 is a $12K\Omega$ resistor which is distributed along the length of the actuator. A wiper is fixed to the end of the actuator and makes electrical contact with the resistor at different locations depending on the position of the actuator. The resistance between the wiper and ground varies between 100% of the total resistance when fully extended, and 0% of the total resistance when fully retracted. Thus, the voltage of the wiper, as a percentage of the voltage between pins 1 and 5, will vary linearly with the position of the actuator. By measuring the wiper voltage on pin3, the actuator position can be determined with a linearity error of less than 2%.

3.2.2.4 Microcontroller and Software Structure

An Arduino Uno R3 was used to control the glue subsystem and monitor the feedback signals due to its low cost, built-in A/D converters, and 0-5V operating range. Three digital output pins were used to send the I1, I2, and Enable signals to the H-Bridge. Digital pin 10 was used for the Enable output since this pin supports PWM, allowing for speed control of the actuator. The feedback signals from the potentiometer wiper and current sensing resistor were connected to analog pins 0 and 5 respectively with a 10K resistor between pin 5 and the current sensing resistor voltage to add impedance and limit any current from the H-Bridge from going through the Arduino. Lastly, three digital pins (3,4,7) were reserved for control inputs to the software and two digital pins (2,11) were reserved for control outputs from the software to the main system.

Table 6: Singal Table of Glue Subsystem

Pin	Source	Destination	Name	Function
A0	Actuator	Arduino	PosChan	Position feedback from Linear Actuator
A5	H-Bridge	Arduino	Force Chan	Measurement of actuator current to approximate force
2	Arduino	Ext. Controller	Ready	Output signal to the external controller that goes high when the glue subsystem is ready to dispense
3	Ext. Controller	Arduino	CMD Ready	Input signal required to send the glue subsystem to the ready position
4	Ext. Controller	Arduino	CMD Enable	Input signal required to enable the glue subsystem. When signal is low, the system will retract
7	Ext. Controller	Arduino	CMD Dispense	Input signal required to begin dispensing. System will dispense until this signal is low or 6000ms elapsed
8	Arduino	H-Bridge	DIR1 / I1	Determine the direction of actuator motion DIR1↑ DIR2↓ → Extend DIR1↓ DIR2↑ → Retract DIR1 = DIR2 → No Motion
9	Arduino	H-Bridge	DIR2 / I2	
10~	Arduino	H-Bridge	PWMPin Enable	Enable signal for the H-bridge. Can be a PWM signal with duty cycle less than 100% to decrease actuator speed
11	Arduino	Ext. Controller	Glue Low	Output signal to the controller that goes high when glue is low and resets only upon re-enable

The software of the glue subsystem consists of a single script, written in the Arduino Integrated Development Environment (IDE), which runs on the Arduino Uno. The objective of this software is to control the dispense cycle of the actuator based on measured force and timing. This software also sends feedback signals when the glue subsystem is ready to dispense, and when the glue level is low as determined by the position of the actuator.

This software uses the state-transition structure as described in chapter 6 of Fundamentals of Mechatronics by Jouaneh ^[26]. The general state transition diagram that the software follows is shown in figure 23.

Figure 23: State Transition Diagram of the Glue Subsystem Software

Upon power-up, the system is in state 1 until the enable signal is received on the enable input pin at which time the system will transition to state 2. The system will then set the enable and direction outputs to the H-Bridge to start extending the actuator and transition to state 3.

Since the Arduino maintains the digital outputs and PWM signals until a new write command is received for that pin, the actuator will continue to extend as long as it is in state 3. Once the measurement of the force exceeds the threshold, the system will go to state 4 and count the number of consecutive force measurements above the threshold. If the count exceeds 5, the system has confirmed that the actuator has reached the glue and sets the direction signals to state retracting the actuator before switching to state 45. In state 45, the system will wait for 150ms (1.2mm of retraction at 8mm/s) then stop the retraction, send the ready output signal to high, and transition to state 5. This retraction relieves the pressure on the glue to reduce dripping and protects the actuator from stalling while waiting for the dispense signal.

Once in state 5, the Arduino will wait for the dispense signal from the external controller, then set the direction and enable PWM signals to extend the actuator, and transition to state 6. The actuator will extend to make contact with the glue again, then apply a constant force until the dispense signal is turned off or the glue timeout duration has elapsed. The dispense signal should be turned off by the controlling system after each dispense; however, if the controlling system is halted for any reason, this timeout protects the actuator from stalling and potentially damaging itself and stops excessing glue from being dispensed onto the target. After dispensing, the system will go to state 7 and retract for 150ms to relieve the pressure, then return to state 5.

The system will cycle through states 5-7 each time a new dispense signal is received until the actuator reaches the end of the travel limit or the enable input is turned off. If the travel limit is reached, the system will start retracting, turn off the ready output, and transition to state 1. Prior to full extension however, the system will send the “glue low” output signal to the external controller but will allow further dispensing until the absolute limit is reached. This informs the external controller that the glue will run out soon, allowing the external controller

to schedule the glue change operation at a convenient time in the overall process. When appropriate, the external controller can turn the enable signal to off, which will cause the glue subsystem to retract and revert to state 1.

3.2.3 Testing and Results of Glue Subsystem

Figure 24: Glue Test Configuration

Before being integrated into the system, the glue subsystem was tested on its own to verify the objectives had been met. This verification test involved dispensing several drops onto a strip of paper and assessing the drops visually for any size discrepancies. To perform this test, the system was mounted above a table, with roughly two inches between the nozzle and the table surface as shown in figure 24. The control software was modified to ignore all control inputs except for the enable input, which was wired to a physical toggle switch. In this way, the system dispensed only once each time the switch was moved from off to on. Further, the dispense duration was controlled in software using the `glueTimeOut` variable, which stops the dispense after a certain delay regardless of the enable input. Figure 25 shows the results from

this test, arranged horizontally by the dispense time. Within each vertical strip, the dispense duration was held constant and the drop sizes are very similar.

Figure 25: Glue Dispensed vs. Dispense Duration

The above results are less promising with respect to the second objective to reduce the glue which falls outside of the target surface. As seen on the two strips farthest to the right of figure 23, there is significant glue in between the drops, representing glue dripping in between dispense commands. It should be noted that this was not observed for the other strips, even when the same parameters were used as in the case of the 3rd strip from the right. From further testing, this behavior was observed more frequently near the end of the glue slug, just before a new slug was inserted. Also, because this was observed even when the actuator was retracted, causes related to the actuator could be discarded. Based on this, a reasonable explanation is that the steady-state temperature of the glue is dependent on the amount of glue present. At the end of the glue slug, there is less thermal mass to store heat and less surface area through which heat is lost, resulting in a higher temperature from the same heater output. This higher temperature reduces the viscosity of the glue and increases the tendency to drip.

Based on these results, the glue subsystem was mostly successful at meeting the stated objectives but requires further development. In terms of cost, this system was much cheaper than any of those identified in table 3, with a total cost of only \$540. Even at this low cost, the system provided excellent consistency and minimal dripping outside of the target area under most circumstances. Despite this, the dripping which occurs near glue depletion negatively affects both objectives, reducing the overall effectiveness. This problem can likely be solved by adding a temperature sensor near the glue nozzle and using the Arduino to control current to the heater, but these upgrades could not be completed in the project timeframe.

3.3 Linear Slide and Frame

Figure 26: Linear Slide Holding the Glue Subsystem

The linear slide, shown above in figure 26, transports the single glue subsystem to each AJU and slowly moves over the components as the glue is dispensed to create an even distribution of glue on the mating surface. Based on the observations from the March visit, the initial plan was to use a multi-axis positioning system to reproduce the pattern dispensed by the operators. The system was to allow motion in the x and y directions as shown in figure 27 below. This

figure shows a top-down view of two bricks within the assembly with the x axis aligned with the length of the machine and the y axis perpendicular to the x axis and parallel to the ground. In this configuration, the range of the y-axis motion required is much smaller than that of the x-axis motion; therefore, the primary slide was to be aligned with the x axis. Mounted to the moving table of the x-axis slide, a smaller slide would allow for motion in the y direction.

Based on a two-axis system, various patterns were considered to achieve the same result as the manual process with the two most promising being depicted in figure 27. The pattern on the left uses a back-and-forth pattern similar to that dispensed by the operators, but with the lines oriented along the y axis. In this pattern, the x-axis slide moves only in a single direction while the y-axis moves in and out to increase the glue coverage. This reduces the time that the glue system is over the brick by eliminating any backtracking along the main axis. In the second pattern on the right, coordination of the two axes is used to trace an outline of the gluing surface. This ensures that there are no gaps along the outside edge where stresses are strongest during bending. Unfortunately, any pattern which fully covers the outside edge will require some amount of backtracking and require precise coordination to produce the desired pattern.

Figure 27: Potential Dispense Paths Using Two-Axis Positioning System

In addition to requiring two-dimensional control, two-axis positioning systems bring with them other considerations. First, such systems cost more than a single axis system, as each axis

requires at least one motor and a driver to operate it. Second, the y-axis positioning system would increase the size and weight of the components mounted to the x-axis positioning system as both the slide and driver motor of the y-axis would need to be moved as well. Additionally, since the entire slide for the secondary axis moves with the primary slide, the tolerances of the primary slide must be carefully controlled to achieve accurate positioning. Lastly for this application, the height of the glue surface in the z-direction changes with the y value due to the angled surface. Thus, as the glue system is moved in the negative y direction, the distance between the nozzle and the surface increases. This effect could be mitigated by rotating the system about the x axis such that the x-y plane of the position system is parallel to the glue surface; however, this would add much engineering complexity.

To reduce the complexity, the prototype instead uses a simpler single-axis system and takes advantage of the angled glue surface to distribute the glue. During testing, it was discovered that large drops of glue will travel down the angled surface with a distance affected by the viscosity of the glue and the duration of motion. Thus, a large amount of glue dispensed near the top edge will be distributed across the entire surface using only gravity. The downside of this method is that the time to achieve optimal coverage is based on the amount of glue dispensed and the viscosity of the glue which is dependent on its temperature. Further, the delay must not be so long that the glue hardens before the two components are pressed together. Lastly, since it takes some time for the slide to travel from one edge of the component to the other, the glue dispensed at the leading edge will have more time to travel than the glue dispensed at the far edge. This will result in better coverage at the leading edge and a slight gap at the lower corner of the trailing edge as shown in figure 28. In this figure, the solid green line represents the path of the glue nozzle and the dashed lines define the approximate coverage of the glue after a short delay. Note that when the direction of travel is reversed, the right edges

of the components become the leading edge, so the trapezoidal region of coverage is flipped left to right.

Figure 28: Dispense Pattern from Single-Axis Operation with Delayed Flipping

Based on the planned single-axis operation, the prototype uses a linear slide from Oriental Motor Company with a travel range of 600mm (23.62"). The minimum travel range was determined from the widths of the subsystems shown in figure 29. At this time, the glue subsystem width was fixed at 4" and the AJU design was targeting a width of 6". Further, the planned operation requires space at each end, and between the AJUs, for the glue subsystem to park without obstructing the motion of the AJUs. Using a minimum clearance of ½" between the AJU and the glue subsystem in the three positions, the total travel distance of the nozzle is no less than 22" (558.8mm). Thus, the slide provides an additional 41.2mm (1.62") of travel beyond the requirement which was used to move the travel range away from the absolute limits of the slide.

Figure 29: Horizontal View of Prototype For Determining Minimum Slide Length

The linear slide consists of a stepper motor and a moving table threaded around a long lead screw inside the body. As the lead screw rotates, the table translates left or right depending on the direction of rotation. A stepper motor driver mounted inside the control box is used to interface with the motor. This driver performs closed-loop control of the position using feedback from an absolute encoder mounted to the shaft of the motor. The use of an absolute encoder enables the driver to know the slide position, even after the power has been cycled.

The stepper motor driver can be operated from an external controller via the RS-485 communication protocol, through USB, or directly from the 24pin digital I/O header on the driver. Use of the RS-485 protocol allows for modularity, in that multiple drivers can be connected to a single network switch or connected in series and addressed individually. This method also reduces the number of wires and allows for the most data in both directions for feedback and control signals. Unfortunately, the Raspberry Pi does not natively support the RS-485 communication standard, so an additional board is required to convert the data stream to a compatible serial format. This method also requires additional software structures to package the output packets and parse the input packets from the network.

Instead, the prototype uses digital signals to control the driver through the 24-pin I/O connector. The functions of these pins are presented in table 7 below, which is copied from the driver manual for reference. When using this interface, the driver operates the motor based on pulses received on pins 1 and 2 with pins 13 and 14 serving as the negative references for the two signals. As shown in the table, a pulse signal on pin 1 moves the motor in the clockwise direction and a pulse signal on pin 2 moves the motor in the counterclockwise direction. Each transition from low to high on one of the input pins moves the motor by a single step in that direction, provided that the other input signal is low at the time of the transition. The motor itself rotates once per 1000 pulses which, combined with the 10mm lead of the screw, results in a translation of the table of 0.1mm per pulse. As indicated in the operating manual, the driver supports pulse rates up to 250KHz, which translates to a maximum speed of 250mm/s.

Table 7: Connection Diagram of the Linear Slide I/O

Pin #	Signal Name	Description	Pin #	Signal Name	Description
1	CW+	CW pulse input +	13	CW-	CW pulse input -
2	CCW+	CCW pulse input +	14	CCW-	CCW pulse input -
3	IN4	ZHOME	15	IN5	FREE
4	IN6	STOP	16	IN7	ALM-RST
5	IN-COM [4-7]	IN4-IN7 common	17	IN-COM [8-9]	IN8, IN9 common
6	IN8	FW-JOG	18	IN9	RV-JOG
7	OUT0	HOME-END	19	OUT1	IN-POS
8	OUT2	PLS-RDY	20	OUT3	READY
9	OUT4	MOVE	21	OUT5	ALM-B
10	OUT-COM	Output common	22	GND	Ground
11	ASG+	A-phase pulse output +	23	ASG-	A-phase pulse output -
12	BSG+	B-phase pulse output +	24	BSG-	B-phase pulse output -

Of the 24 signals, those which have connections in the final prototype are represented by the shaded boxes with bold text in the above table. In addition to the pulse inputs, the control computer also sends the signals for the ZHOME And FREE inputs on the slide driver through pins 3 and 15 respectively. A high signal on the ZHOME input will cause the driver to start the home operation, which quickly moves the slide to the user-specified home location. This must

be performed once after power on to synchronize the control computer with the actual position of the slide. A high signal on the FREE input, meanwhile, will cut all current to the motor and allow the slide to move freely. This input is used as a safety precaution to release any force on an obstructing object or operator. Lastly, the ALM-RST input is wired directly to a button on the user interface and is used to reset the alarm and re-enable the slide after a safety shutdown.

To monitor the status of the linear slide, the control computer receives the following output signals from the slide driver: HOME-END, READY, MOVE, and ALM-B. The HOME-END output signal is normally low and switches to high once the home operation, initiated by the ZHOME input, is complete. This signal is used in the software to determine when the slide has reached home, at which point the position tracking variable is reset to 0. The READY output indicates when the driver is ready to perform an operation. When using the pulse input method, this signal turns high after the motor has completed its setup and switches low when the slide is in motion as a result of a ZHOME operation or from pulse inputs. After initial setup, the MOVE output operates counter to the READY output by switching high when the slide is in motion and low when the slide is stopped. Finally, the ALM-B output is high under normal conditions and switches to low when the driver recognizes that a fault has occurred such as an obstruction or a command to move the slide beyond the physical limits. When this occurs, the driver will also cut power to the motor until the ALM-RST input is set high. Thus, this signal is used in the software to pause execution while the alarm is active.

The y and z positions of the nozzle, as shown in figure 30, are controlled by the positions of the frame members which hold the slide. The slide is mounted to an aluminum member which runs along the top of the frame. The y-position of this member can be adjusted to set the y-position of the glue nozzle. From the primary cross member, two short members are mounted vertically to hold the ends of the slide. The slide mounts to channels in these vertical frame

members, allowing a small range of motion in the z direction. Ideally, these positions are set such that the glue falls a short distance and lands near the top edge of the mating face without spilling over the top edge.

Figure 30: Adjustable Coordinates of the Linear Slide

3.4 System Control

The controller developed for this project was designed to fulfil the needs of both the prototype and a full-scale machine containing eight AJUs. Expansion to eight-units is made possible through a modular arrangement of add-on boards which provide the interfaces for the AJUs, and a modular software structure. The add-on boards communicate with the main control computer through I2C and SPI communication protocols, which allow each AJU to be uniquely addressable in the control software. The software, contained in a single Python Script, uses a task scanning structure with tasks to monitor and control each subsystem. By specifying the unique addresses, the AJU task can be executed for each AJU installed in the system. This section details the specifications of the control computer, the user interface, and the control

software of the prototype. Additionally, this section describes in detail, how the prototype controller can be adapted to the full-scale machine.

3.4.1 Control Computer

Control of the prototype is managed by a Raspberry Pi 3B+ computer mounted inside the control box. Stacked on top of the Raspberry Pi are an IO Pi + expansion board, two RELAY plate expansion boards, and a custom interface board as shown in figures 31 and 32 below. These boards add extra capability to the base hardware to fulfil the needs of the prototype. This section will discuss the capabilities of each board and their role in controlling the prototype.

Figure 31: Compute Stack Mounted Inside the Control Box

Figure 32: General Signal Diagram of the Compute Stack

The Raspberry Pi provides multiple general-purpose input/output (GPIO) pins, as shown in figure 33, and libraries to read or write digital signals via software. This computer also benefits from a small form factor, and very low power draw, making it very user-friendly in embedded system applications. In terms of software, the Raspberry Pi comes standard with a Linux-based operating system, Raspbian, complete with a full user interface. Also included with the standard operating system are the development environments and compilers to create source code in C and Python.

Raspberry Pi 3 GPIO Header				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Figure 33: Raspberry Pi GPIO Pin Functions [Source: Element 14]

Additional boards can be stacked on top of the Raspberry Pi to add A/D conversion, and other capabilities not supported by the base hardware. To communicate with these boards, the Raspberry Pi supports I2C and SPI serial communication protocols directly through the GPIO header. For the prototype, additional boards were required to expand the number of digital input and output pins available. Each AJU has 11 proximity sensor outputs and 1 digital output from the motor to indicate when the motor is in motion. Additionally, four signals are needed to activate the solenoid valves, and two more are needed for the enable and position inputs to the motor. Thus, each AJU requires connections for 12 inputs to, and 6 outputs from the Raspberry Pi.

To expand the number of I/O pins available, an IO Pi+ board from AB Electronics was added to the control computer. This board contains two MCP23017 16-bit IO expander chips, each with its own header, for a total of 32 digital inputs and outputs. When stacked onto the GPIO header, the IO Pi+ communicates via the I2C pins on GPIO pins 2 and 3. Each chip on the board has its own I2C address which are selected on the board using solder jumpers. Using

this method, up to four boards can be stacked on the Raspberry Pi to gain an additional 128 uniquely addressable digital I/O pins. This is sufficient to control 7 AJUs at 18 I/O pins per AJU; however, this would require AJUs to share I2C busses to maximize efficiency. Thus, to maintain modularity, a single bus/address is dedicated to the 12 inputs from each AJU, with a separate approach to handle the 6 outputs.

Figure 34: IO Pi+ Board Mounted to Raspberry Pi GPIO Pins [Source: Adafruit]

To activate the solenoid valves and motor inputs, a RELAY Plate from Pi-Plates was included for each of the two AJUs in the prototype. This add-on board contains 7 relays with the ability to switch voltages up to 120V. Unlike the IO Pi+, this board communicates via the SPI communication protocol, which maintains distinct addresses from those used for I2C devices. Similar to the IO Pi+, multiple of these boards can be stacked on top of the Raspberry Pi and communicate simultaneously, provided each has a unique address as configured by the on-board jumpers. The benefit of this approach is that the board can be used to switch multiple 24V signals, as required by the solenoid valves, without any transistors. Also, since the addresses do not interfere with those of the IO Pi+, the approach could be expanded to a full 8-unit system with four IO Pi+ boards and eight RELAY Plates.

Figure 35: RELAY Plate [Source: Pi-Plates]

The last board which makes up the compute stack of the prototype is the Raspberry Pi Interface Board. This custom board was built from the Proto-Plate also available from Wally Ware. The primary purpose of this board is to route the connections from the add-on boards to a single connector for each of the other systems in the prototype. These include a 24-pin connector for each AJU, 16-pin connector for the linear slide driver, 5-pin connector for the Arduino, and a 10-pin connector for the user interface built into the control box. Since the slide driver uses 24V logic levels for the inputs and outputs, the interface board includes transistors and voltage dividers to step the voltage up or down to allow communication with the Raspberry Pi. By including the interfaces for the slide driver, control box, and Arduino on this interface board, the space available for AJU connections is limited to two 24-pin connectors. For a larger system, more interface boards would be required; however, the additional boards would not require hardware for the single-instance devices and could support more than two boards each.

3.4.2 Control Interface

The control box serves as the operator interface and houses the main electrical components of the machine. The operator interface (figure 36) is built into the lid of the control box and consists of a text display, basic input buttons, a power selector switch, and an emergency stop button. The text display is a 7" touch screen from the Pi Foundation with touch input disabled to simplify the operator interaction. During operation, the display prompts the user for input when required, and provides feedback as each step is completed. Additionally, the display informs the user when the glue must be replaced, when a pause request has been acknowledged, or when the linear slide alarm is active.

Figure 36: Front (Left) and Reverse (Right) Views of the Control Interface

The three buttons on the interface – START, PAUSE, and ALARM RESET – are the sole inputs available to the operator during normal operation. As much of the process is automated, input is only necessary after each manual operation and when an error occurs that requires

operator intervention. Applied to the full system, the time between inputs would be on the order of minutes, allowing the operator to perform other tasks during this downtime.

Further, the mechanism and software were designed such that the machine can produce both orientations of bricks without changes to the software parameters. Unless the operating conditions are changed, there is no need for additional input methods for adjusting the parameters. Should this be necessary, the software can be modified directly through the text display using a wireless keyboard. Although this option is available, this input method is recommended only for troubleshooting purposes.

The purpose of the START button is to indicate that the operator is clear of the machine and has completed the steps as requested by the display. In this beta prototype, the steps performed manually are the verification steps on powerup, loading and unloading of the components, and glue replacement. Prior to each of these, the various tasks will go to safe states where motion is disallowed, and the scheduler will prompt the operator to complete the specific task. Once the 'START' button is pressed, the scheduler will allow the other tasks to resume the operation. At this stage in the development, the machine does not have any sensors to verify that the operator is free of the machine (e.g. light curtain) and will immediately begin the next operation after the button is pressed.

The PAUSE and ALARM RESET buttons are present only for unexpected errors which may occur during operation. The function of the PAUSE button is to pause the execution after the current AJU and is to be used for non-critical errors which may cause problems on the remaining AJUs if operation continues. Such errors could be obstructions, loss of air pressure, or problems with the glue subsystem. Further, the pause feature could be used if there are not enough bricks in the current batch to fill the remaining units. As the pause feature does not stop operation on the current AJU, this is only applicable during the first AJU of each cycle for

the prototype but could be more applicable with a larger system. One additional feature of the PAUSE button is that a long press of more than 10 seconds will close the current instance of the Python script and load the newest version. This can be performed to deal with run-time errors or during troubleshooting to quickly load new software changes.

The last button, ALARM RESET, is only used to reset the linear slide driver after the alarm is activated and the problem has been resolved. The linear slide driver has protections in place to shut down during events of excessive current, or when the commanded movement is beyond the travel limits of the slide. The first of these will occur if the operator or a piece of the machine blocks the motion of the slide, thus the alarm reset button can be safely pressed once the obstruction has been removed. Meanwhile, violation of the travel limits indicates a run-time error in the software or an issue tracking the current position of the slide. In this case, the software must be reset by holding the pause button for 10 seconds before the alarm can be reset.

At the bottom of the interface are the power selector switch and emergency stop (E-Stop) button to control the power to the system. The power selector switch has three positions for OFF, ALL POWER, and AJU/CPU POWER. Inside the control box, the live conductor of the 110VAC input cable is split with one line to the glue gun, and another to the DC power supplies. When the power switch is set to the OFF position, both live circuits are opened, cutting all power to the system. In the ALL POWER position, both live circuits are closed, powering all systems. Lastly, from the ALL POWER position, the user can flip the switch to the AJU/CPU POWER position which cuts power to the glue gun while maintaining power to the DC power supplies. This is useful for short breaks in production to reduce dripping from the glue gun and allows the operator to bypass the initialization steps when production resumes.

Lastly, the E-Stop button provides a method for immediately cutting all electrical power to the system for critical errors and safety hazards. At the same time, cutting power to the 24V power supply closes the main air inlet valve and opens the exhaust valve to quickly depressurize the system. Unlike the pause button, this will stop motion at any point in the operation and release any force that the machine is applying on itself or the operator. Once the issue has been addressed, the operator can reset the E-Stop button and restore power to the machine.

3.4.3 Control Box Hardware

The control box body, shown in figure 37, houses the computers, power supplies, and slide driver to protect them from the factory environment. Additionally, the control box protects the workers from the 110VAC present on the components inside the red rectangle in figure 36 and the lower DC voltages on the central terminal strip.

Figure 37: Arrangement of Devices Inside the Control Box

The main power to the system is routed into the control box through a hole in the bottom wall. The three conductors for live, neutral, and ground are screwed into different positions of a safety terminal block to shroud the exposed ends of the conductors. From the terminal block, neutral and ground conductors are wired directly to the two power supplies and to the glue gun power cable. The live conductor is branched into two wires: one for the glue gun power and the second for both power supplies. Both live conductors are routed through a fuse box (black box in figure 37), then through the emergency stop button (E-Stop) which are both fixed to the inside of the control box lid. The fuse box ensures that if the current exceeds 5A, a likely indicator of a short, the current is stopped before the rest of the system is damaged. Meanwhile, the E-Stop button provides a manual method for cutting the power in the case of a failure or work hazard. After the E-Stop button, the conductors pass through the selector switch, which is configured to open both circuits, close both circuits, or close only the circuit to the power supplies. The live conductors from the selector switch are then routed to the glue gun power cable, and the two power supplies.

The power supplies inside the control box provide DC power for all components of the machine. The larger power supply has a maximum output of 24V 6A, while the smaller power supply outputs both 12V 3A, and 5V 3A. A screw terminal block in the center of the back wall is used to route the three levels of DC power to the appropriate devices as indicated in the table below. The ground references of the three voltage outputs are connected to the same point on the terminal block to provide the same reference for all devices.

Table 8: Supply Voltage Requirements for Control Box Devices

Device	Voltage			
	5V	12V	24V	Common Ground
Compute Stack	x	x	x	x
Arduino	x	x		x
Slide Driver			x	x
AJU 1			x	x
AJU 2			x	x

Along the top of the control box from left to right are the Raspberry Pi compute stack, the Arduino with interface shield, and the linear slide driver. The compute stack on the left communicates with the Arduino via a 6-pin cable which carries the three control inputs to the glue subsystem, two control outputs, and a redundant ground reference (see table 7 in section 3.2.2.4). Since these signals are connected directly to the Raspberry Pi GPIO pins, the Arduino interface board contains voltage dividing circuits to produce 3.3V signals from the 5V outputs of the Arduino. Meanwhile, the 3.3V outputs from the Raspberry Pi GPIO are connected directly to the Arduino inputs since these inputs only require an input greater than 2.7V to register a high input. Lastly, a 16-pin ribbon cable connects the Raspberry Pi Interface Board to the I/O header of the linear slide driver. The signal table of this connector is presented in the table below. It should be noted that the 24V output signals of the slide driver are not connected directly to the GPIO pins, but instead through voltage dividers to drop the voltage below 3.3V. Likewise, the 24V inputs are connected using a transistor switch circuit to allow for control from the Raspberry Pi 3.3V outputs.

Table 9: Signal Table of the 16-pin Raspberry Pi to Linear Slide Driver Cable

Signal #	Color	Source (Pin)	Destination	Type	Name
1	Green	Interface Board (24V)	Slide Driver (10)	24V Reference	OUT_COM
2	Blue	Slide Driver ()	Raspberry Pi (5)	Slide Output (24V)	MOVE
3	Purple	Interface Board (GND)	Slide Driver (22)	GND Reference	GND
4	Grey	Slide Driver (21)	Raspberry Pi (27)	Slide Output (24V)	ALM_B
5	White	Interface Board (GND)	Slide Driver (5)	GND Reference	IN_COM
6	Black	Interface Board (NC)	Slide Driver (4)	Slide Input (24V)	STOP
7	Brown	Interface Board (GND)	Slide Driver (14)	GND Reference	CCW-
8	Red	Raspberry Pi (6)	Slide Driver (15)	Slide Input (24V)	FREE
9	Orange	Interface Board (GND)	Slide Driver (13)	GND Reference	CW-
10	Yellow	Raspberry Pi (13)	Slide Driver (3)	Slide Input (24V)	ZHOME
11	Green	Reset Button	Slide Driver (16)	Slide Input (24V)	ARM_RST
12	Blue	Raspberry Pi (19)	Slide Driver (2)	Slide Input (5V)	CCW_PULSE
13	Purple	Slide Driver (19)	Interface Board (NC)	Slide Output (24V)	IN_POS
14	Grey	Raspberry Pi (26)	Slide Driver (1)	Slide Input (5V)	CW_PULSE
15	White	Slide Driver (20)	Raspberry Pi (14)	Slide Output (24V)	READY
16	Black	Slide Driver (7)	Raspberry Pi (17)	Slide Output (24V)	HOME_END

The compute stack must also communicate with the control interface built into the lid of the enclosure. The display uses a standard Raspberry Pi display cable which connects directly into the Raspberry Pi display output. The rest of the connections are routed from the GPIO pins on the Raspberry Pi interface board, to an intermediate board mounted to the left wall of the control box. From the intermediate board, the signals for the START button, PAUSE button, and display power are transmitted via another ribbon cable. The purpose of this board is to combine the connections into a single cable to allow the baseplate of the control box to be easily removed and later reconnected to the interface. This single ribbon cable is also oriented to prevent the lid from closing on the cable and breaking the connections.

3.4.4 Control Software

The software of the Raspberry Pi is responsible for synchronization of the subsystems and handles all processing except for that of the glue subsystem which is performed by the Arduino. Like the rest of the prototype, the software uses a modular structure to control the machine,

regardless of the number of AJUs present. This software was written in Python 2.7 due to its inclusion with the default Raspbian operating system and the availability of Python libraries for interfacing with both types of add-on boards. A functional description of the software is presented here, and the full printout of the Python script can be found in appendix C.

This script uses the same state-based structure as the Arduino code discussed previously for the glue subsystem but has multiple tasks which are scanned sequentially as shown in figure 38. Each task is contained within a function and are called sequentially inside an infinite loop after the initial setup and until the machine is powered off. In the case of the AJUTask, the same function is called once for each AJU in the system where the number inside the parentheses of the function call indicates the address of the AJU. As will be demonstrated in the discussion on this function, this allows the same code to act on the different inputs and variables associated with each unit.

Figure 38: Task Scanning Sequence Followed by the Control Software

The goal when using this structure is to minimize the processing time inside each task, to approximate parallel operation of the tasks using a single processor thread. The benefit of near-parallel processing is that key events which affect the transitions of a task can be responded to in a timely manner with less risk of missing such an event. To accomplish this, the tasks must

not contain any blocking code that requires an indefinite amount of time to complete such as while loops or recursion. If such code is present, the execution will become stuck in the problematic task, until the right conditions are met.

3.4.4.1 Task Descriptions

Inside the set-up block is all of the code needed only once to initialize the software. This includes import statements for the various libraries, the declaration of global variables used in the tasks, and certain operational parameters which can be modified to change the behavior of the machine. Additionally, some of the tasks include set-up states which are not accessible in the state-transition diagrams of those tasks once the set-up is complete.

The purpose of the scheduler task is to track the overall state of the machine and synchronize the different tasks. During their operation, the other tasks send feedback to the scheduler by setting flags to inform the scheduler that they have completed their current operation and reached the next stop point. Depending on the current system state, the scheduler will check that the appropriate completion flags have been set, then set enable flags to allow the tasks to proceed to the next stop point. The state transition diagram which the scheduler follows is shown below in figure 39.

60, the system is waiting to apply glue to the first AJU in the forward direction. To transition to the next state, the scheduler must receive feedback from the AJU task that the AJU 0 is in the down position. Once this is received, the scheduler task sets the commanded glue position flag to position 2 (just after the first AJU) and sets the flag “GlueStartCMD” to allow the move task to move to the commanded position and apply glue on AJU 0. In other cases, the scheduler will present information to the operator that the tasks have been completed and request input from the operator before proceeding. An example can be seen between states 18 and 25, where the scheduler task prompts the user that the AJUs are ready and waits for the user to press the START button to indicate that the loading is complete.

The move task is responsible for controlling the position of the linear slide and sending the dispense command to the glue subsystem. This task maintains a set of global variables which track the current position of the slide, the desired destination, and the desired speed. On the first call, this task sends the glue gun to the home position, then waits for the scheduler before moving to each subsequent position. Figure 40 below shows the sequence of positions of the move task from the home position through a full assembly cycle. In this figure, the red lines indicate points where the move task is sending a high dispense signal to the glue task running on the Arduino. Further, the circles represent positions which can be commanded by the scheduler task, and the diamond markers indicate positions at which the move task sets the flag to allow the previous AJU to flip. Note that after the slide has homed, subsequent cycles start at position 1 rather than position 0.

Figure 40: Position Sequence Used by the Move Task

The move task uses the state transition diagram presented in figure 41 below to follow the sequence of positions as commanded by the scheduler. The states in this diagram are arranged such that after state 120, the direction of the diagram indicates the direction of the slide with up arrows indicating the forward operation, and down arrows indicating the reverse operation. Additionally, the dashed boxes contain the states in which the task is operating on each AJU. It should be noted that some boxes contain multiple states for conciseness such as the box containing states 130-150. This collection of states represents the moving and dispensing operation on AJU 0 in the forward direction. Transitions between states within this box are based on slide position and are used to change the slide speed and turn the dispense signal on and off. The same is true for the other grouped states which represent the same process for the other combinations of AJU and direction. Further, the red arrows between states indicate transitions which require scheduler intervention. For example, in state 125, the glue system is positioned just before AJU 0 in the forward direction and must wait for the startGlue command from the scheduler to proceed. Meanwhile, bent red arrows from the states indicate points at which this task modifies the global gluePos variable. This variable stores the most recent

position of the slide using the position numbers as shown in figure 40. The variable is referenced by the scheduler to determine when the move task has finished moving to the commanded position.

Figure 41: General State Transition Diagram of the Move Task

To assist with its operation, the move task has access to two additional functions: setMotionParams() and updateSlide(). The first of these changes the desired speed and position variables based on the next operation. For example, after the glue finishes dispensing, this

function will be called with the maximum speed of the slide and a destination away from the AJU, to move the glue gun out of the way as fast as possible. The second function, `updateSlide()`, moves the slide by controlling the pulses that are sent to the slide driver. This function is called during the motion states of the move task until the position is within one count of the desired position. Inside the call to `updateSlide`, the function toggles the state of the pulse signal if one half pulse width has elapsed since the last state change. Each time the pulse switches from low to high, the pulse counter is incremented or decremented depending on the current direction of motion. The pulse width is determined by the `setMotionParams` function to generate a pulse rate corresponding with the desired speed.

The AJU task controls the movement of the AJU whose address is specified in the parameters of the function call by following the state transition diagram shown below in figure 43. To track multiple AJUs, the states and control flags used by this task are arranged in 1-dimensional arrays with lengths corresponding to the number of AJUs. Thus, when the task is called for a specific AJU, the function references the state transition diagram using only the state and sensor values for that AJU. As with the move task diagram, some states are grouped according to their shared objective such as raising the stopper, going to ready position, and joining the parts. At key points such as after enabling, raising the stopper, or reaching the ready position, the AJU task sets the appropriate flag and waits for a response from the scheduler. These points are indicated by the pairs of red arrows between the states. For most other transitions, the task requires that a certain amount time has elapsed since the last transition or that the correct combination of inputs is received from the AJU sensors. This is aided by the `readSensorState(AJUAddr)` function which reads the bits of the specified I2C bus, and produces a 16-bit integer whose first 12bits indicate the sensor values. Lastly, the AJU task commands the positions of the AJU actuators using the `setAJURelays(AJUAddr, reState)`

function. This function uses the 7-bit number 'reState' to update the commanded states of the 7 relays of the RELAY Plate at the address specified by AJUAddr. It should be noted that this structure is dependent on the fact that no I2C bus or relay plate address is shared by more than one AJU.

Figure 42: AJU Task General State Transition Diagrams

On the first call of the AJU task for each AJU, the function waits for the scheduler to declare that the slide is in the home position, then enables the motor. If the motor has not been enabled since it was last powered on, the motor will begin the homing sequence to determine its position and motion parameters. After homing, the AJU task lifts the stopper to allow for the bricks to be loaded, then waits for the ready flag from the scheduler. Once the ready command is received, the AJU task steps through the states to position and hold the bricks, then retracts the stopper. In this ready position, the AJU task must wait for the AJUfinish command from the

scheduler, which is sent only after the glue system is clear of the flipper. After receiving the finish command, the AJU task activates the flipper, holds for the given duration, then releases the completed brick. At this point, the AJU task is complete, and the task returns to the initial state to prepare for the next cycle.

Lastly, the watchdog task monitors for alarms from the linear slide and input from the pause button on the control panel. Since these events can occur at any time and require immediate action, assigning them to a separate task ensures that these events are checked for each time the main loop is executed. If the alarm is present, the watchdog task immediately sets the alarm present flag to true, preventing execution of the AJU task and move task until the cause of the alarm is removed and the operator presses the alarm reset button on the control panel. Pressing the pause button serves two functions which are both handled by the watchdog task. If the pause button is pressed for more than 10 seconds, the program will terminate and restart using the newest file at the specified location. If the pause button is released before 10 seconds has passed, the watchdog task will indicate to the scheduler that the user has requested a pause. The scheduler will then check this flag and pause after the current AJU.

3.4.4.2 System Operational Sequence

At the beginning of the shift, the operator moves the power switch from ‘off’ to ‘all power’. This sends AC power to the glue gun to begin heating, and powers on the DC power supplies inside the control box. The 24V power supply energizes the main air supply valve to pressurize the system and turns on the slide driver and AJUs. Meanwhile, the 5V and 12V power supply powers the compute stack, Arduino, and user interface display. First, the Arduino begins execution of the glue task which retracts the glue actuator. After its boot sequence, the Raspberry Pi immediately executes the control software to begin its own setup process. The

software first initializes the GPIO pins to their default states, then prompts the user to press the START button.

Once the START button is pressed, the various tasks begin their setup phases. The objectives of the setup phase (in the order they are completed) are to move the slide to the home position, allow the AJU flipper motors to perform homing, and verify that glue is loaded in the glue gun. First, the scheduler checks that both AJU flippers are down by reading the output of the two home sensors, then sets the glueHomeCMD flag high. This flag allows the move task to move to state 10, which sends the ZHOME signal to the linear slide and readies the glue subsystem. The slide then moves to the home position and sends the homeEnd signal back to the control computer. Once the homeEnd signal is received, the move task initializes the absolute pulse counter to 0, sets the motion parameters to move to position 1, then waits for the scheduler. The homeEnd signal is also required by the scheduler before the AJU motors can begin homing since the slide must not interfere with the homing operation. By waiting for this signal, the scheduler can know the position of the slide before sending the AJUEnableCMD flag to allow the AJU task to proceed with homing. Once the AJUEnableCMD is received by each instance of the AJUTask, the relays are switched to enable the motors. The AJU tasks then wait for the home sensors to deactivate, indicating that homing has begun. Once the home sensors are reactivated or 20 seconds has passed, the AJU task declares that homing is complete via the AJUEnableDone flag and disables the motors for safety. In response, the scheduler sends back the AJUStopperCMD which allows the AJUs to raise the stopper for loading. Meanwhile, the enable and ready signals sent to the glue subsystem command the glue task to go to the ready position. While the slide is homing, the glue actuator extends until it contacts the glue slug as determined by the force threshold. If the glue is detected, the glue task retracts the actuator a small amount, then sends the ready signal back to the control computer. If the

actuator reaches 80% extension without detecting the glue, the glue task instead sends back the glue low signal to the control computer. Based on the signal returned from the glue task, the scheduler either exits the setup phase or prompts the user that a glue replacement is needed. In the latter case, the scheduler turns off the enable signal to the glue system to retract the actuator. The operator must load a new glue slug then press the START button for the scheduler to reenable the glue subsystem.

After the setup is complete, the scheduler prompts the operator to load the components and press the START button when complete. Once the button is pressed, the scheduler allows both AJUs to proceed to the ready position by sending the AJUReadyCMD. The AJUs then step through states 20-70 to push the components, hold the components in position, then retract both the stoppers and pushers. At each state in this sequence, the AJU task changes the actuator states then waits for the sensor states to match the expected values. By using the sensor feedback, the reliability of the system is improved since the task will only proceed if the physical system is ready for the next state change. This can also increase the speed of operation, since the next action can be commanded immediately after the previous transition is complete.

Once AJU 0 is in the ready position (state 70), the scheduler commands the move task to move to position 1 via the gluePosCMD variable. The move task moves to position 1, just before AJU 0, then waits for the scheduler to confirm that the AJU 0 flipper is down. If the flipper is down, the scheduler sends the startGlueCMD signal to the move task to allow dispensing on AJU 0. The move task sets the motion parameters to move to position 1A at high speed, then turns on the dispense signal to the glue subsystem. After a short delay to allow the glue to reach the components, the move task sets the parameters to move to position 1B at the speed specified by the variable 'glueSpeed'. Once the slide reaches position 1B, the move task turns off the dispense signal to the glue gun, waits for the glue to stop dripping, then moves to

position 2 at high speed. At position 2, the slide updates the gluePos variable to let the scheduler know that the glue gun is clear of the AJU 0 flipper. The scheduler then sends the AJUFinishCMD signal to allow AJU 0 to complete the assembly, while the move task and scheduler repeat the above process on AJU 1. The finishing steps carried out by AJU 0 consist of mating the two parts, waiting for the glue to harden, then releasing the completed brick. To allow the glue time to cover the component, the AJU task specifies the delay between the initial glue dispense time, and flipper activation. After this delay, the AJU task reenables the flipper motor, moves the flipper to 90°, then activates the pushers to force the components from the back. The components are held in this position for about 20-25 seconds before the brick can be handled as a single unit. At this point, both pushers and the holder of the flipped channel are released then the flipper is moved to the down position. The AJU task for AJU 0 then sends the AJUIsDone flag, lifts the stopper, then goes back to state 5 to prepare for the reverse operation. Lastly, once the slide is in position 4 and both AJUs have completed the finishing steps, the scheduler prompts the operator to remove the components and load new components. If at any point during the above operation, the scheduler received the glue low warning from the glue subsystem, the scheduler sends the disable signal to the glue gun and prompts the operator to replace the glue.

Rather than travel back to home, the prototype saves time by operating on AJU 1 then AJU 0 while travelling in the reverse direction. As shown in figures 39 and 41, the scheduler and move task contain additional states for this reverse operation. The only differences between these states and those of the forward operation are the order of the transitions and the roles of the numbered positions shown in figure 40. For example, in the reverse operation, position 3 serves as the trigger position to begin flipping AJU 0, rather than position 4 for the forward

direction. Meanwhile, the AJU tasks are unaffected by the change in direction and will operate exactly the same as in the forward direction.

Lastly, the operator has limited capability to pause the operation by pressing the PAUSE button during the first AJU of each direction. This button press is recorded by the watchdog task, which raises the 'pauseRequest' flag. The scheduler will check for this flag in states 80 and 380, just before the final check that the next AJU is down. The scheduler will then go into the paused state and prompt the operator to press the button to continue. This state allows the current AJU to proceed but pauses the operation before the next AJU in the cycle. The purpose of this function is to allow the operator to stop the operation if there is a problem with the next AJU, or if there are not enough components to fill the remaining AJUs.

Chapter 4: Optimization and Final Testing

To deal with the uncertainties of the new process, the prototype was designed with many configurable parameters. Many of these were finalized during the testing of the individual subsystems, but some required integration with the rest of the system before their effects could be studied. The beta prototype provided the first opportunity to test the subsystems together; thus, there were a number of items which could not be tested until its completion in November 2019. The following sections detail the testing which was performed on the completed prototype during the final months of the project.

4.1 Parameter Optimization

In the first phase, the software parameters were adjusted to improve the throughput and reliability of the operation. The software provides many global variables to change the system behavior such as dispensing force, timing, and key slide positions. Because the methods used in the prototype are different from those of the manual process, it was difficult to predict how the parameters would affect the final product. For example, the prototype uses rotation to move the two components together while the worker moves the parts orthogonally during the manual process. Further, many processes adapted from the manual process are aided considerably by human perception such as glue dispensing. Unlike the workers who can change their pulling force based on the dispense rate, the glue subsystem can only measure the actuator travel distance, which may not correlate with the actual dispense rate.

First, the glue start and stop positions were adjusted to achieve the most coverage of the mating surface without any glue spreading outside of the XPS area. Figure 43 shows many of the parts resulting from this testing. The components are arranged in rows for each forward and reverse cycle, with the lower rows produced from earlier cycles. The red dashed regions along

the columns represent the approximate safe areas which were targeted during this testing. It should be noted that the prototype does not check which orientation of brick is loaded in each AJU. Thus, the dispense pattern must stay within the target area regardless of the brick orientation or direction of travel. For this reason, the target region was limited to be away from the alignment groove where glue has a shorter distance to travel before reaching the concrete. Additionally, this testing was conducted without any small components loaded such that the dispense pattern could be observed. Thus, the glue had more time to drip down the surface than it would during normal operation before the two components are joined, resulting in some glue appearing on the bottom edge of the components.

Figure 43: Test Samples to Determine Glue Start and Stop Points

These positions are defined in the software in terms of the offsets from the edge of the AJU. Specifically, the 1a and 3a positions, where the glue dispensing begins in the forward direction, are defined by the AJU positions plus the value glueStart. Likewise, the 1b and 3b positions are defined as the same AJU positions plus the variable glueStop. In this way, if either AJU is

moved, changing the position of the AJU in the software will update all of the necessary positions. Further, changing the glueStart and glueStop variables will affect the start and stop points on both AJUs. As shown in figure 43, the early runs had many instances of the glue starting and stopping outside of the target area. After each cycle, the parameters were adjusted to offset the errors observed from the previous tests until the dispense pattern was just inside the limits of the target area as can be seen along the top row of figure 43.

Next, the glue dispense quantity and hold times were adjusted to achieve greater reliability of the bond. To achieve dispense rates close to that from manual operation, the glue system is designed to apply the maximum force of the actuator during dispensing. Thus, the only method to adjust the amount of glue dispensed onto the bricks is to change the speed of the slide within the dispense region, defined by the variable 'glueSpeed'. Slower speeds will result in longer dispense durations and more glue per unit length. Slowing down the slide also affects the dispense pattern by allowing the glue at the leading edge more time to drip down the surface. Additionally, more glue per unit length results in faster movement down the surface. This problem is made worse at the end of the glue slug, when viscosity is lowest. Thus, the speed was adjusted such that the two parts are joined just as the glue reaches the bottom edge for the worst-case scenario. Unfortunately, this resulted in too little glue being applied at the opposite extreme of viscosity. The solution to this was to increase the hold time from 15 to 20 seconds, to create a stronger bond from the smaller quantity of glue. Additionally, the spring steel holding the right-side components was adjusted to decrease the holding force as the flipper is released.

4.2 Pre-Production Testing

With the prototype configured, the final two weeks prior to delivery were used to conduct a series of pre-production runs. To support these tests, the company provided the team with multiple glue slugs and components for left-oriented and right-oriented bricks. The objective of these runs was to produce twenty bricks with equal numbers of left and right oriented bricks to determine the approximate throughput, success rate, and quality of the prototype. Throughput was calculated in seconds per brick by dividing the total time by the number completed bricks including those which broke apart during unloading. Meanwhile, the success rate was determined by the number of successfully produced bricks as a percentage of the total number of bricks attempted. To measure quality, the successfully assembled bricks were packed into boxes to be sent back to the company for assessment by their quality control personnel. To achieve a passing mark in quality, the alignment of the bricks must be similar to, or better than, that of the manually produced bricks.

A secondary objective of this testing was to determine a simple and efficient operating procedure to be documented and delivered with the final prototype. In all prior tests, operation was paused after each brick (or pair of bricks) to assess the quality and change the system parameters. Thus, this testing represented the first attempt at continuous operation of the prototype from which additional challenges were revealed. Some considerations which resulted from this testing include the arrangement of components prior to assembly, the method of organizing the completed components, the order and method of loading the components, and the point in the sequence at which to unload the components. As determined during testing, these are all important factors in maximizing the throughput and reducing the failure rate from misalignment or jamming. Lastly, because the operating procedure evolved throughout this testing, the results from later tests were given significant weight relative to the earlier tests.

These tests were conducted in the URI mechatronics laboratory by personnel familiar with the prototype development. As shown in figure 44, the prototype was placed on a level table such that the height of the control interface was approximately at shoulder level. One issue with this location, as shown in the figure, is the limited access to the rear of the machine due to the shelving supports and glass divider. In the final procedure, the operator is instructed to travel around the machine and load the components from their respective sides to reduce the risk from reaching through the machine. Due to the barriers of the test location, the procedure was modified for all components to be loaded from the open side. This removes the time needed for the operator to travel to both sides of the machine but increases the difficulty of loading the right-side components. Unfortunately, the impacts of these changes could not be assessed until the prototype was moved to a more open location at the production facility.

Figure 44: Prototype During Pre-Production Test Runs

Although the technique evolved with each test, the general procedure throughout the tests was the same. Prior to testing, the bricks were arranged by orientation and finish, then 10 pairs

of components for each orientation were gathered and stacked on a table behind the operator. After recording the start time, the operator loaded the machine with the first set of components and pressed the START button to begin the forward operation. The completed bricks were removed from the machine immediately after their release and set to the side of the prototype. Bricks which fell apart at any point in the assembly were placed in a separate area for later analysis. To produce all twenty bricks, the process was repeated for five forward and reverse cycles, then the final time was recorded.

Results from the final pre-production tests showed positive results compared to the previous estimations of throughput. These estimates are based on the timeline shown in figure 45, in which the durations of events are the same as in figure 6 for the large-scale concept. Thus, if the operator can match this timeline, then it would follow that the same operator could meet the throughput estimates for the large-scale prototype. For the last two runs, the procedure was modified only slightly to stack the next set of components to be assembled on the AJU baseplates while the AJU was holding the current set of components. In this way, the completed bricks could be quickly removed and replaced with the new components as each AJU finishes its operation. With this technique, the measured durations of the final two runs were 9:55 and 9:43 (or 29.75s per brick and 29.15s per brick) respectively. These values represent an increase of about 18% compared to the rate observed during the initial visit (36 seconds per brick). More importantly, these rates are very close to matching the estimated rates (26.5s per brick). Based on this result, it seems likely that the same concept could achieve the expected throughput with some small improvements.

Figure 45: Estimated Timing of Events for The Two-Unit Prototype

Figure 46: Samples of Successfully Completed Bricks from Pre-Production Runs

The quality of the successfully produced bricks was about the same as that of the manual process, however the rate of successfully completed bricks was slightly lower. Figure 46 shows a sample of bricks produced by the prototype during the production test runs. The bricks shown

represent the best-case scenario of the prototype, with alignment and gap sizes that exceed the expectation of the manual process. About 75% of the produced bricks fell into this category.

The remaining bricks either had acceptable flaws or failed to hold together during the entire process. Figure 47 shows samples of flaws which were observed during the test runs. In the case of the top-left image, the final angle of the brick is less than 90° which could result from a slightly lower cutting angle. Unlike the manual process which holds the two parts at a 90° while the worker applies forces orthogonally, the flipper of the prototype applies a torque until the contact force of the brick reaches the threshold. Thus, if the two bricks are cut at a shallower angle, the flipper must rotate farther to produce the same force at the contact faces. This reduces the gap between the two parts but allows for variation in the final angle. Small variance in the angle is acceptable, but angles significantly displaced from 90° will lead to gaps between the brick and the building. Next, images B and C of figure 47, show instances of the glue seeping out of the interior and exterior edges of the completed brick respectively. The presence of these flaws is related to the dispense position and viscosity of the glue, with lower viscosity resulting in more glue dispensed and faster rates of movement across the angled surface. Ideally, the glue would be dispensed just below the top edge of the mating surface to prevent the glue from escaping over this edge and viscosity would be controlled to limit dripping down to the bottom edge. In practice, when the viscosity of the glue was low, the higher dispense rate would cause the glue to build up on the top edge and sometimes fall onto the top surface (figure 47B). In other cases, the low viscosity would cause the glue to drip over the bottom edge of the part (figure 47C). In both cases, the low viscosity increased the tendency of the glue to spread as the force was applied, leading to more glue being squeezed out of the target area. Lastly, figure 47D shows an example of a part with a small vertical misalignment. This was rarely observed and likely caused by bumps on one of the components which prevent the component from

being flush with the inside wall of the channel. It should be noted that the manual process assumes flatness of the bottom face and would result in the same flaw in the final alignment.

Figure 47: Samples of Issues from Pre-Production Tests

Unfortunately, the rate of parts produced successfully with the prototype was lower than that of the manual process. During the pre-production runs, the failures were clustered with some runs having no failed assemblies, and other runs showing failure rates of up to 25%. It should be noted that the observed failure rate was higher for right-oriented assemblies. In this case, the spring steel acts on the bottom surface of the brick without the alignment groove. As the flipper lowers, the entire flat surface must slide across the spring steel to release the part. Compared to the opposite orientation where the spring steel acts only on the small edge of the alignment groove, the work done by friction is greater due to the larger sliding area. Further, the glue viscosity had a significant effect on the success rate of the prototype. When the

viscosity was low, such as after a glue replacement, the amount of glue dispensed was decreased significantly. Further, the low viscosity results in minimal spreading from gravity and applied force, decreasing the contact area of the glue with the components. Lastly, the prototype does not allow for the joining to be reattempted as was possible with the manual process. By the time the parts are released, there is little time for the worker to remove the parts and manually join the parts accurately before the glue has hardened. Thus, any parts which fail the initial joining are much more likely to result in failed assemblies.

4.3 Testing at Company Facility

The prototype was then delivered to the facility for the final phase of testing. The goal of this phase was to gather feedback on the operation and determine what information would be required for the operators. During the following visit to the facility, the prototype was demonstrated to a small number of representatives using a similar test plan to that described in section 4.2. While conducting this test, the issue of inconsistent glue viscosity arose once more, leading to several failures from not applying enough glue. As a temporary solution, the glue dispense time was increased for all cases resulting in many fewer failed assemblies, but also more instances of excess glue spilling onto the edges of the brick. Based on discussion with representatives, the amount of excess glue that was observed has less impact on the production than the failed bricks, so this tradeoff was acceptable. Better long-term solutions were also discussed such as integrating a temperature sensor near the nozzle of the glue gun or a full upgrade of the glue subsystem but were understood to be outside the scope of the remaining efforts. After increasing the glue dispensed, no further changes were made to the prototype; however, the company was given instructions on how to modify the system parameters discussed in section 4.1.

Chapter 5: Conclusion

The main goals of this project were to develop a concept for large-scale automated corner brick assembly and to validate the concept through a working beta prototype. The conceived concept was based on a modular arrangement of subsystems to assemble multiple bricks simultaneously with minimal operator interaction. The core of this modular concept was a low-cost automated joining unit (AJU), designed to manipulate one pair of components during the assembly. The proposed machine would contain many identical AJUs and other subsystems for dispensing glue, moving the glue nozzle, and providing feedback control of the machine's operation. Where possible the subsystems will contain their own interfacing hardware to allow for future upgrades and expansion without affecting the other subsystems.

First, a detailed mechanical design of the AJU was created to operate on both orientations of the sponsor's most popular size of corner brick. Due to the components' unique shape and combination of materials, the final design required many novel features to facilitate the assembly. First, the XPS use of hot melt glue required the two components to be rapidly aligned and pressed together after the glue is dispensed. To solve this, the AJU moves the two components into known positions within two alignment channels before the glue is applied, and the two alignment channels are moved together. The design uses a strip of spring steel and pneumatic actuators to align the components within the channels. A novel stopper mechanism was designed to precisely limit the depth of insertion of the components within the channels. Friction from the concrete coating as the components are moved presented an additional challenge to the design. This friction hindered the movement of the components and caused wear of the AJU. To mitigate these effects, the surfaces of the alignment channels were covered with sheets made of a durable melamine compound. This protective layer significantly reduced the effects of friction, resulting in much smoother loading and wear protection lasting multiple

eight-hour shifts. Lastly, the AJU uses a rotational motion to join the parts in contrast to the linear translation of the manual process. The offset axis of rotation allows the two components to be loaded horizontally and provides space between the two components for the stopper and glue system to operate.

To more easily integrate with the rest of the system, the AJU was designed to contain a multitude of sensors and its own interfacing hardware. These sensors signal the ‘on’ and ‘off’ positions of each actuator and the status of the flipper motor to be used by the control computer. Meanwhile, the interfacing hardware included with each AJU consisted of the solenoid valves, pneumatic routing equipment, pneumatic speed control valves, and an electronics interface board. The interface board performed level shifting of the sensor outputs from 24V to 5V and combined all of the I/O of the AJU into a single cable. In this way, each AJU required only a single hose for pressurized air, a power connector, and a 24-pin ribbon cable.

After the AJU design was refined, a critical path analysis was performed to estimate the duration of operation of the machine for any number of joining units. This method, commonly used to estimate the durations of projects containing many tasks, requires knowledge of the tasks required to complete the project, their interdependencies, and their estimated durations. For this application, the “project” analyzed was a single direction of operation of the machine, in which each AJU performs one assembly and the glue system travels from one side to the other. The tasks of this project included operator actions, and actions performed by the machine’s subsystems which are required to assemble one brick for each AJU in the system. Dependencies were derived from the physical system and the operational sequence of the prototype. Through this method, it was determined that the parallel operation of the AJUs results in a fixed amount of time added for each additional AJU. Specifically, the first AJU requires 46 seconds to complete its operation, with each AJU adding 7 seconds thereafter.

Based on this relationship, the number of units in the concept was set at eight to meet the target throughput of 2400 bricks per eight-hour shift. Additionally, this method highlights the tasks whose durations directly affect the overall timeline. For the concept, these critical path steps consist of the operator loading, slide movement, dispensing, and the finishing steps of the final AJU. Thus, to increase the throughput with a fixed number of AJUs, further work should attempt to increase the speed of these steps.

A prototype was designed using a minimum viable product (MVP) strategy to prove the operation of the large-scale concept with the minimum amount of resources. The MVP was built with a linear slide, glue system, control hardware, and two AJUs representing the minimum number required to confirm the estimated timelines. The AJU and system controller developed for the MVP were feature complete versions matching the detailed design planned for the full-scale machine. Meanwhile, the auxiliary subsystems were designed with only the basic capabilities needed to perform the automated operation. The reason for this distinction was the lack of prior art related to the AJU operation compared to that for the other systems. For example, many commercial systems fit the requirements of the glue dispensing system and glue positioning systems. Thus, the methods used by these systems have been extensively tested by their manufacturers and present a low risk if inserted into the large-scale concept.

The system controller was designed to meet the requirements of the MVP by allowing seamless expansion to control eight joining units and other subsystems from a single processor. The controller hardware consists of Raspberry Pi 3B computer with two RELAY Plates and one IO Pi + expander board for each pair of AJUs in the system. These boards can be stacked to provide unique interfaces and addresses for up to eight AJUs using a small subset of the Raspberry Pi's GPIO pins. To provide additional flexibility, the control software developed for this project separated the processing for each subsystem and AJU into discrete tasks. Each task

maintained a set of global variables to track the state of its corresponding system and performs a small amount of processing depending on the current state, input signals, and control flags. For the AJU task, the control flags and global variables were stored in arrays with elements for each AJU in the system. In this way, the same task could be used to control each AJU by specifying a unique address for each function call. A separate scheduler task was added to manage communication between the tasks via control flags. Finally, an infinite loop rapidly executed each task in sequence to simulate parallel operation of the tasks.

In the process of building out the prototype, this project also successfully demonstrated a method for creating a low-cost glue gun for automated dispensing. The automated glue gun was built from the same TEC 3400-43 glue gun used by the company for its current production. The manual advancing mechanism was replaced with an electric linear actuator and full H-bridge driver to control the actuator's speed and direction. Feedback control of the actuator was achieved using an Arduino Uno microcontroller with inputs from a linear potentiometer inside the actuator and a current sensing resistor placed between the H-bridge and ground. Additionally, the Arduino used three digital control inputs to transition between different operational states and provided two digital outputs for monitoring the status of the system. These inputs provided a simple interface for an external controller to ready the system, start and stop dispensing, and receive information on the current glue level. With this architecture, the prototype was able to achieve precise control of the dispense quantity with little glue outside of the target area under most conditions. Additionally, the information on glue levels allowed the prototype to schedule the glue replacement operation to have a minimum impact on the ongoing production.

Production test runs conducted with the prototype showed very promising results with respect to the sponsor's end goal. During these runs, the prototype was operated continuously

with an average throughput over 2 bricks per minute. Further, the part quality of the bricks was similar to that of the manual process. With the completion of this testing, the prototype successfully demonstrated the concept of the AJU operation at a cost of less than \$2500 per unit. This testing also demonstrated the effectiveness of the modular software and interface structure. Based on this success, an eight-unit system is entirely plausible using the current AJU and controller design. Further, we believe that these results significantly reduce the risks associated with a larger-scale development effort.

5.1 Recommendations for Future Work

Before recommending any changes, it is important to clarify that the prototype was designed to serve as a proof of concept for the AJU mechanism and control software only. Thus, improvements related to the specific implementation of the other subsystems are not considered for this discussion. With that said, testing did reveal important features which should be added in future development efforts. The following items represent the most pressing matters based on results of the final testing:

1. Replace spring steel on AJU with small linear actuator:

From testing, it was found that the spring steel was not a reliable method of aligning the parts laterally within the channel. The analysis of this feature assumed that the end of the steel would lay tangent to the component and provide a distributed force to move the component parallel to the wall. In practice, the force acted more like a point force, allowing some yaw of the right-side component. Further, because the spring steel is a passive mechanism, there is no way to release the lateral force on the component. As mentioned during the final testing, the spring steel holding the right-side component was the cause of many failures. Thus, increasing the lateral force to improve alignment results in a trade-off with reliability. To fix both issues, it is recommended to replace the spring steel with a

small linear actuator in each channel. This actuator should have a flat plate which pushes on the side of the brick to align it with the front wall. The actuator would then release before the flipper is lowered to avoid pulling on the brick.

2. Use a commercial system for glue dispensing:

As mentioned, it is fully expected that a future development effort will start with a more capable glue dispensing system. The development of the prototype glue system was effective in highlighting three features which are recommended for the replacement system. First, the new glue system should provide accurate control of the temperature at the nozzle to reduce fluctuations in viscosity. Second, the system should contain a check valve close to the nozzle, or similar feature, to eliminate the problem of glue dripping in between dispenses. Third, the system should dispense glue on the entire area without relying on gravity to distribute the glue.

References

- [1] Deangelis, T. 1994. "Clinton's Climate Change Action Plan," *Environmental Health Perspectives* 102, 102 (5): 448–49.
- [2] Boeing, Geoff, Daniel Church, Haley Hubbard, Julie Mickens, and Lili Rudis. 2014. "LEED-ND and Livability Revisited," *Berkeley Planning Journal* 27, 27 (1). doi:10.5070/BP327124500.
- [3] Newsham, Guy R., Sandra Mancini, and Benjamin J. Birt. *Do LEED-Certified Buildings Save Energy? Yes, but....* vol. 41, 2009
- [4] The U.S. Energy Information Administration. *Annual energy outlook with projections to 2040*. Washington DC; 2016.
- [5] Jensen, Thor, and Hadi Dowlatabadi. "Federal Tax Credits and Residential Investment in Renewable Energy.", 2017.
- [6] The Hartford Loss Control Department. "Exterior Insulation and Finish Systems". *Technical Information Paper Series*. TIPS S 140.012. 1997. 1.
- [7] Troughton, Michael J, and International Institute of Welding. 2008. *Handbook of Plastics Joining*. PDL Handbook Series. US: William Andrew.
- [8] Maharana, T, Y. S Negi, and B Mohanty. 2007. "Review Article: Recycling of Polystyrene," *Polymer-Plastics Technology and Engineering* 46, 46 (7). Taylor & Francis Group: 729–36. doi:10.1080/03602550701273963.
- [9] Uddin, N., A. Vaidya, U. Vaidya, and S. Pillay. 2013. "12 - Thermoplastic Composite Structural Insulated Panels (CSIPs) for Modular Panelized Construction." Edited by

- Nasim Uddin. Developments in Fiber-Reinforced Polymer (FRP) Composites for Civil Engineering. Woodhead Publishing. doi:<https://doi.org/10.1533/9780857098955.2.302>
- [10] Karlsson, Kicki F., and B. TomasÅström. 1997. "Manufacturing and Applications of Structural Sandwich Components." *Composites Part A: Applied Science and Manufacturing* 28. doi:[https://doi-org.uri.idm.oclc.org/10.1016/S1359-835X\(96\)00098-X](https://doi-org.uri.idm.oclc.org/10.1016/S1359-835X(96)00098-X).
 - [11] Grünewald, Jonas, Patricia Parlevliet, and Volker Altstädt. "Manufacturing of thermoplastic composite sandwich structures: A review of literature." *Journal of Thermoplastic Composite Materials* 30, no. 4 (2017): 437-464.
 - [12] Jerome, G, and Rene M Dupuis. n.d. "The Response of Molded Expanded Polystyrene (EPS) Insulation Board to Various Attachment Methods."
 - [13] Corp, Sto. 2014. "Adhesive v. Mechanical Fastening of StoTherm®."
 - [14] ["Adhesives and Sealants 101: Hot Melts"](#). Adhesives & Sealants Industry. October 1, 2008. Retrieved 20 February 2020.
 - [15] Coyle, Dennis J. "Knife and roll coating." In *Liquid film coating*, pp. 539-571. Springer, Dordrecht, 1997.
 - [16] Razban, A., and B. L. Davies. 1991. "An Automated System for Dispensing Adhesives." *International Journal of Adhesion and Adhesives* 11. doi:[https://doi-org.uri.idm.oclc.org/10.1016/0143-7496\(91\)90022-A](https://doi-org.uri.idm.oclc.org/10.1016/0143-7496(91)90022-A)
 - [17] Sprovieri, John. 2018. "Hot-Melts for Electronics Assembly." *Assembly* 61. Troy: BNP Media. <https://search.proquest.com/docview/2049987849>.

- [18] Ribas-Xirgo, Lluís, José M. Moreno-Villafranca, and Ismael F. Chaile. "On using automated guided vehicles instead of conveyors." In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 14. IEEE, 2013.
- [19] Shi, Jane, Glenn Jimmerson, Tom Pearson, and Roland Menassa. "Levels of human and robot collaboration for automotive manufacturing." In *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, pp. 95-100. 2012.
- [20] Agin, Gerald J. "Computer vision systems for industrial inspection and assembly." *Computer* 5 (1980): 11-20.
- [21] A. Björnsson, Automated layup and forming of prepreg laminates, 2017.
- [22] DiFilippo, Nicholas M., "Framework for the Automated Disassembly of Electronic Waste Using the Soar Cognitive Architecture" (2016). Open Access Dissertations. Paper 514. https://digitalcommons.uri.edu/oa_diss/514
- [23] Sujatha, G., and V. Perasiriyana. "Conveyor control and sorting module controlled by programmable logic controller." *International Journal of Engineering and Technology* 4, no. 2 (2014): 12-18.
- [24] Wu, Wenqi, Xingang Wang, Guan Huang, and De Xu. "Automatic gear sorting system based on monocular vision." *Digital Communications and Networks* 1, no. 4 (2015): 284-291.
- [25] Schumacher, Paul, and Jouaneh, Musa. "A Force Sensing Tool for Disassembly Operations." *Robotics and Computer-Integrated Manufacturing* 30.2 (2014): 206-17. Web.
- [26] Jouaneh, Musa. "Chapter 6: Control Software." *Fundamentals of Mechatronics*. Cengage Learning, Stamford, CT, 2013.

Appendix A: AJU Mechanical Design Documents

AJU Front Side:

Right-Component Alignment Mechanism:

AJU Reverse Side:

Left-Component Alignment Mechanism:

AJU Machined Parts List:

Dwg No.	Name	Quantity	Stock Part	Vendor
1	Baseplate	1	3511T115	McMaster-Carr
2	Fixed Rear Support	1	9001K97	McMaster-Carr
3	Fixed Right Support	1	9001K97	McMaster-Carr
4	Flow Control Bracket	1	9001K89	McMaster-Carr
5	Flipper Axis Cutout	1	8982K139	McMaster-Carr
6	Fixed Channel	1	9001K97	McMaster-Carr
7	Flipper Channel	1	9001K97	McMaster-Carr
8	Flipper Rear Support	1	9001K97	McMaster-Carr
9	Bottom Flipper Bracket	1	9001K97	McMaster-Carr
10	Stopper Block	1	2012	80/20 Inc.
11	Flipper Pulley	1	A 6A25-030DF1508	SDP/SI
12	Spacer	2	8975K473	ZXHAO
13	Pusher2/Holder 1 Mount	2	8mm Rigid Shaft Mount	McMaster-Carr
14	Pusher 2 Arm	1	8975K596	McMaster-Carr
15	Pusher 1/Holder 2 Mount	2	8mm Rigid Shaft Mount	ZXHAO
16	Holder 2 Arm	1	8975K596	McMaster-Carr
17	Holder 2 Bracket	1	8982K57	McMaster-Carr
18	Short Spring Steel	1	N/A	N/A
19	Pusher 2 Guard	1	9001K58	McMaster-Carr
20	Spring Steel Outer Mount	2	8975K596	McMaster-Carr
21	Spring Steel Inner Mount	2	8975K596	McMaster-Carr
22	Belt Tensioner Block	1	8975K473	McMaster-Carr
23	Flipper Motor Bracket	1	8982K139	McMaster-Carr
24	Flipper Axis	1	8982K139	McMaster-Carr
25	Bottom Mounting Plate	1	9001K89	McMaster-Carr
26	Fixed Left Support	1	9001K97	McMaster-Carr
27	Holder 1 Bracket	1	8982K57	McMaster-Carr
28	Holder 1 Arm	1	8975K596	McMaster-Carr
29	Pusher 1 Guard	1	9001K58	McMaster-Carr
30	Long Spring Steel	1	N/A	N/A
31	Pusher 1 Arm	1	8975K596	McMaster-Carr

Off the Shelf Components:

Name	Qty.	Stock Part	Vendor
Pusher/Holder Actuators	4	CDRB2BW30-90SZ	SMC
Stopper Actuator	1	MXS8-30	SMC
Flipper Motor	1	CPM-MCVC-2321S	Teknic
Stopper Face	2	4134	80/20 Inc
Gearbox	1	GBPH 0601-CS-001	Anaheim Automation
Pneumatic Hose	100'	TU0425B-33	SMC
Timing Belt	1	A 6R25M052150	Stock Drive Parts / Sterling Instruments
Drive Pulley	1	15-5M15-0912.250	Custom Machine and Tool Co.
Flipper Pulley	1	A 6A25-030DF1508	Stock Drive Parts / Sterling Instruments
Bar Stock Manifold	1	SS5Y3-42-05-N3T	SMC
Solenoid Valve	5	SY3140-5LZ	SMC
Pneumatic Fitting	10	KQ2L04-M5A	SMC
Actuator Position Sensor	10	D-M9NL	SMC
Flow Control Valve	10	AS1002F-04	SMC
Proximity Sensor	1	N/A	N/A

Appendix B: Electrical Schematics

AJU Interface Board:

Control Computer Interface Board

Control Panel:

Glue Dispensing Subsystem:

IO Pi + [Source AB Electronics]:

Appendix C: Source Code

C.1:

Filename: newbrickLatest.py

Description: This Python script runs on the Raspberry Pi control computer and contains tasks command the glue dispensing subsystem, and control the AJUs, linear slide, and user interface.

```
#!/usr/bin/env/python
from IOPI import IOPI
import time
import RPi.GPIO as GPIO
import piplates.RELAYplate as RELAY

#Program Parameters
verbose = False
noGlue = False
loopsForPrint = 10000 #When printing enabled, print after this many loops
version = 110
## 105 Changed AJU2 glue speed to 11
## 106 R1 start pos +3mm
## 107 R2 start pos +3 R1 stop pos +3
## 108 Changed AJU1 glue speed to 11
## 109 Added Glue Dispense Counter
## 110 Added Performance Tracker

#-----System Level Setup-----#
watchDogState = 0
nextWatchDogState = 0
sysState = 0
nextSysState = 0
recoveryState = 0 #When pause button is pressed, restore the system to this state

#-----AJU Task Setup-----#

AJUisPresent = [True,True]
bus = [IOPI(0x20),IOPI(0x21)]
AJUsafeFlags = [True, True]
nextAJUState = [0,0]
AJUEnableCMD = [False,False]
AJUEnableDone =[False,False]
AJUisReady = [False,False]
AJUfinishCMD = [False,False]
stopperIsUp = [False,False]
glueStartTime = time.time()
glueMove = False
glueReady = True
glueStart = [True,True]
glueClear = [True,True]
AJUStopperCMD = [False,False]
AJUSwitchTime = [time.time(), time.time()]
AJUStateDelays = [1,0,.5,0,0,0,25,.5,0,0,0,0]

busRemap = [12,10,8,6,4,2,7,5,3,1,14,16]

for i in range(1,17): #Set all IOPI Plus pins as inputs
    bus[0].set_pin_direction(i,1)
    bus[1].set_pin_direction(i,1)
```

```

#-----Glue Task Setup-----#
GPIO.setmode(GPIO.BCM)

ready = 4 #GPIO pin number for slide ready input
moving = 5 # GPIO pin number for MOVE output
freeOut = 6
alarmIn = 27
homeEndIn = 17
pulseCW = 26 #GPIO pin number for CW pulses
pulseCCW = 8 #GPIO pin number for CCW pulses
dispense = 12 #GPIO pin to send dispense command to Arduino
glue2ready = 20 #GPIO pin to send ready command to Arduino
glueEnablePin = 16 #GPIO pin to send Enable to Arduino (was physical switch)
GPIOZHOME = 13 #GPIO pin to send ZHOME command to slide
goButton = 15 #GPIO pin number for 'Go' button
pauseButton = 14 #GPIO pin number for 'Stop' button
glueReady = 21 #GPIO pin to receive ready signal from Arduino
glueReplaceIn = 19 #GPIO pin to receive the low glue warning from the Arduino
thisTime = time.time()
lastTime = time.time()
pulseCheckTime = time.time()
cycleStartTime = time.time()
cycleStopTime = time.time()
sysStateChangeTime = time.time()
pausePressTime = time.time()
pulseDuration = 100
systemIsPaused = False
ignorePause = False
alarmActive = False

infiniteLoop = True

inputs = [ready, moving, alarmIn, homeEndIn, glueReady, glueReplaceIn] # List of all inputs for later
initialization
outputs = [pulseCCW, pulseCW, freeOut, GPIOZHOME, dispense, glue2ready, glueEnablePin]

relayOnTime = 0.1
waitForSlideTime = 0.2
glueDripStartDelay = 1
glueDripStopDelay = 0.5
pressureWarningCooldown = time.time()
initializeTime = time.time()
last10startTime = time.time()

pulseEdgeTime = time.time()
glueOnTime = time.time()
glueOffTime = time.time()
glueTimeout = 5 #Timer to shut off glue automatically after 4 seconds if not already turned off
pulseState = 0 #Current State of the pulse PWM signal
nextPulseState = 0 #Next State of the pulse PWM signal
moveState = 0
nextMoveState = 0
destination = 0
direction = 0
glueLevel = 0 #How much glue is left. 0 = normal, 1 = replace soon, 2 = replace now
dispRemaining = 100 #Approximately how many dispenses can be completed before the glue must be
replaced
glueRefPos = 0 #When passing over AJU, this stores the position for that the glue timing
shall be relative to
startGlue = 87 #Distance from glueRefPos (mm) to begin gluing
stopGlue = 117 #Distance from glueRefPos (mm) to stop gluing
allowFlip = 270 #Distance from glueRefPos (mm) to allow flipping
startGluePosOffset = 5
stopGluePosOffset = 5

```

```

allowFlipRev = 0
startGlueRev = allowFlip-stopGlue-10
stopGlueRev = allowFlip-startGlue-20

#Define the key slide positions based on the positions of each AJU (pos1 and pos3 should change if
the AJUs are moved)
pos1 = 10
pos1a = pos1+startGlue
pos1b = pos1 + stopGlue
pos2 = pos1+ allowFlip
pos3 = 310
pos3a = pos3+startGlue
pos3b = pos3+stopGlue
pos4 = pos3+allowFlip

testOffset = 0

inPOS = False
sendHomeFlag = False
inHome = False
motionParamsSet = False
pulseEntryFlag = False
AJUclear = True
blockAJU = [True,True] #When slide is moving or planning a move, use these flags to block motion
of the individual AJUs
dispenseSent = False
glueIsEnabled = False
glueSentReady = False
slideInHome = False
pauseRequest = False
reverseRequest = False
pauseForGlueReplace = False
airPressureWarning = False
absDispenseCounter = 0
dispenseCounter = 0

AJUreadyCMD = [False,False]
gluePosCMD = 0      #Glue Station commanded by scheduler
gluePos = 0         #Glue Station reported by moveTask
glueHomeCMD = False
startGlueCMD = False
moveTaskDone = False
AJUtaskDone = [False,False]
continueAfterGlueReady = False

absPulseCount = 0   #Slide Encoder Counts (Relative to Home)
loopCount = 1
#moveState=0
glueState = 0
AJUStates = [0,0]

pauseStateEntryFlag = False

for i in inputs:
    #Configure Inputs
    GPIO.setup(i,GPIO.IN)
    GPIO.setup(goButton,GPIO.IN, pull_up_down =GPIO.PUD_UP)
    GPIO.setup(pauseButton,GPIO.IN, pull_up_down =GPIO.PUD_UP)

for i in outputs:
    #Configure Outputs
    GPIO.setup(i,GPIO.OUT)
    GPIO.output(GPIOZHOME, GPIO.HIGH)
    GPIO.output(freeOut, GPIO.HIGH)
    GPIO.output(pulseCCW, GPIO.HIGH)
    GPIO.output(pulseCW, GPIO.HIGH)

```

```

#Returns a 16bit integer whose bits correspond with the sensor outputs
def readSensorState(AJUAddr):
    sensorState = 0
    for i in range(len(busRemap)):
        j = busRemap[i]
        sensorState = sensorState*2 + bus[AJUAddr].read_pin(j)
    return sensorState

#This function will print information to the screen only if the verbose flag is true
def printDebug(message):
    if verbose:
        print(message)

#Sets only the first 6 relays of the designated Relay Plate (leaves relay 7 as is)
def setAJURelays(AJUAddr, reState):
    reState = reState + (RELAY.relaySTATE(AJUAddr) & 0b1000000)*64
    RELAY.relayALL(AJUAddr, reState)

def timeCheck(AJUAddr, delay):
    return (time.time()-AJUSwitchTime[AJUAddr]>delay)

def AJUTask(AJUAddr):
    global nextAJUState
    global AJUEnableCMD
    global AJUisReady
    global AJUsafeFlags
    global glueStartTime
    global AJUSwitchTime
    global AJUtaskDone
    global AJUEnableDone
    global stopperIsUp
    global airPressureWarning

    state = nextAJUState[AJUAddr]

    if state ==0:#System is in default state (all relays inactive)
        setAJURelays(AJUAddr, 0b000000)
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 1

    elif state ==1:
        if readSensorState(AJUAddr) ==2729:
            nextAJUState[AJUAddr] = 2

    elif state ==2: #System is waiting for glue to home before starting enable sequence
        if AJUEnableCMD[AJUAddr]:
            AJUEnableCMD[AJUAddr] = False
            setAJURelays(AJUAddr, 0b100000)
            AJUSwitchTime[AJUAddr] = time.time()
            nextAJUState[AJUAddr] = 3

    elif state == 3: #Wait 2 second for homing operation to complete
        if timeCheck(AJUAddr, 1) and (bus[AJUAddr].read_pin(14)): #Flipper not in home
            print ("\n AJU "+ str(AJUAddr+1)+ " homing")
            AJUSwitchTime[AJUAddr] =time.time()
            nextAJUState[AJUAddr] = 4
        elif timeCheck(AJUAddr,3) and not(bus[AJUAddr].read_pin(14)): #Two seconds has
            passed and flipper has not moved. Motor must have homed previously
            AJUEnableDone[AJUAddr] = True
            setAJURelays(AJUAddr, 0b000000)
            nextAJUState[AJUAddr] = 6

    elif state ==4: #Wait for homing to complete
        if timeCheck(AJUAddr,20): #Home sensor triggered or moveDone signal is on
            AJUSwitchTime[AJUAddr] = time.time()
            setAJURelays(AJUAddr,0b000000)

```



```

        nextAJUState[AJUAddr] = 5
        # elif (readSensorState(AJUAddr)%2)==0: #Motor is telling us that move is AJUtaskDone
        #     setAJURelays(AJUAddr,0b000000)
        #     AJUSwitchTime[AJUAddr] = time.time()
        #     nextAJUState[AJUAddr] = 5

elif state ==5:
    if not(bus[AJUAddr].read_pin(14)):
        AJUEnableDone[AJUAddr]=True
        nextAJUState[AJUAddr] = 6

elif state ==6:
    if AJUStopperCMD[AJUAddr]:
        AJUStopperCMD[AJUAddr] = False
        nextAJUState[AJUAddr] = 7

elif state == 7: #Activate Stopper
    if readSensorState(AJUAddr) == 2729:
        setAJURelays(AJUAddr,0b001000) #Activate Stopper
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 8

elif state ==8:
    if readSensorState(AJUAddr) ==2725:    #Stopper is up
        AJUSwitchTime[AJUAddr] = time.time()
        stopperIsUp[AJUAddr] = True
        nextAJUState[AJUAddr] = 10
    elif timeCheck(AJUAddr,3):
        airPressureWarning = True
        nextAJUState[AJUAddr] = 9

elif state ==9:
    if readSensorState(AJUAddr) ==2725:    #Stopper is up
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 10

elif state ==10:
    if AJUreadyCMD[AJUAddr]:
        printDebug( "AJU ready command received on AJU" + str(AJUAddr))
        AJUSwitchTime[AJUAddr] = time.time()
        AJUreadyCMD[AJUAddr] = False
        nextAJUState[AJUAddr] =20

elif state == 20: #Activate Pushers
    if timeCheck(AJUAddr,.5) and (readSensorState(AJUAddr) == 2725):
        setAJURelays(AJUAddr,0b001001) #Activate Pushers
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 30

elif state == 30: #Activate Holders
    if timeCheck(AJUAddr,.5) and readSensorState(AJUAddr) == 1445:
        setAJURelays(AJUAddr,0b001111) #Activate Holders
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 40

elif state == 40: #Deactivate Pushers
    if timeCheck(AJUAddr,.5) and readSensorState(AJUAddr) == 1365:
        setAJURelays(AJUAddr,0b001110) #Deactivate Pushers
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 50

elif state == 50: #Deactivate Stopper
    if timeCheck(AJUAddr,.5) and readSensorState(AJUAddr) == 2645:
        setAJURelays(AJUAddr,0b010110) #Deactivate Stopper and command flipper to up position,
        but do not enable yet
        AJUSwitchTime[AJUAddr] = time.time()

```

```

        nextAJUState[AJUAddr] = 60

elif state == 60:
    if (readSensorState(AJUAddr) == 2649):
        AJUisReady[AJUAddr] = True        #Tell the scheduler that the AJU is ready
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 70

elif state == 70: #Wait for start glue signal
    if glueStart[AJUAddr]:
        glueStartTime = time.time()
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 80

elif state == 80: #Wait for AJU finish command
    if AJUfinishCMD[AJUAddr]:
        AJUfinishCMD[AJUAddr] = False
        setAJURelays(AJUAddr,0b110110) #Command flipper up position
        #AJUsafeFlags[AJUAddr] = False #Tell system that flipper 1 is up
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 100

# elif state == 90:
#     if timeCheck(AJUAddr, 2.5):          #Enable motor to allow flipper to move up
#         setAJURelays(AJUAddr, 0b110110)
#         AJUSwitchTime[AJUAddr] = time.time()
#         nextAJUState[AJUAddr]=100

elif state == 100:
    if readSensorState(AJUAddr) == 2651: # Flipper has begun operation
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 105

elif state == 105:
    if timeCheck(AJUAddr,.5) and readSensorState(AJUAddr) == 2650: # Flipper has completed
operation
        #AJUSwitchTime[AJUAddr] = time.time()
        setAJURelays(AJUAddr, 0b110111) #Activate Pushers
        nextAJUState[AJUAddr] = 110

elif state == 110: #Wait 20 seconds
    if timeCheck(AJUAddr,20): #Glue has been held for 20s
        setAJURelays(AJUAddr,0b110010)
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 115

elif state == 115: #Wait .5 seconds
    if timeCheck(AJUAddr,.5):
        setAJURelays(AJUAddr,0b110010) #Deactivate Pushers and Holder 2
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 120

elif state == 120:
    if readSensorState(AJUAddr) == 2666:
        setAJURelays(AJUAddr,0b100010) #Move Motor to "Flipper Down Position"
        AJUSwitchTime[AJUAddr] = time.time()
        nextAJUState[AJUAddr] = 130

elif state == 130:
    if timeCheck(AJUAddr, 0.5) and readSensorState(AJUAddr) >= 2664 and
readSensorState(AJUAddr)<=2666: #Flipper has completed its operation
        #AJUsafeFlags[AJUAddr] = True #Tell glue that flipper 1 is down
        setAJURelays(AJUAddr, 0b000000) #Deactivate Holder 1. Also disable motor
        AJUSwitchTime[AJUAddr] = time.time()
        AJUtaskDone[AJUAddr] = True
        nextAJUState[AJUAddr] = 5

```

```

def mm2counts(position):
    # This function converts a desired position in mm to the corresponding
    # number of pulse counts
    countsPerRev = 1000
    lead_mm = 10
    counts = 1.0*position/lead_mm*1000
    return counts

def glueReadyOn():
    global glueSentReady
    printDebug('GlueReadyOn')
    glueSentReady = True
    GPIO.output(glue2ready,GPIO.HIGH)

def glueReadyOff():
    global glueSentReady
    printDebug('GlueReadyOff')
    glueSentReady = False
    GPIO.output(glue2ready,GPIO.LOW)

def glueEnable():
    global glueIsEnabled
    printDebug('GlueEnable')
    GPIO.output(glueEnablePin,GPIO.HIGH)
    glueIsEnabled = True

def glueDisable():
    global glueIsEnabled
    printDebug('GlueDisable')
    GPIO.output(glueEnablePin, GPIO.LOW)
    glueIsEnabled = False

def DispenseOn():
    global dispenseSent
    global dispenseCounter
    # global absDispenseCounter

    printDebug('DispenseOn')
    dispenseSent = True
    if not(noGlue):
        GPIO.output(dispense, GPIO.HIGH)
        dispenseCounter = dispenseCounter + 1
        # absDispenseCounter = absDispenseCounter + 1

def DispenseOff():
    global dispenseSent
    printDebug('DispenseOff')
    dispenseSent = False
    GPIO.output(dispense, GPIO.LOW)

def setMotionParams(targetmm,speed):
    global pulseDuration
    global destination
    global direction
    pps = mm2counts(speed)
    pulseDuration = 1.0/pps
    printDebug(str(pulseDuration))
    destination = mm2counts(targetmm+testOffset)
    if destination>58000:
        destination = 58000
    if destination >absPulseCount:
        direction = 1
    elif destination<absPulseCount:
        direction = 0
    else:
        direction = 0
    inPOS = True

```

```

GPIO.output(pulseCCW, GPIO.HIGH) #Set both pins to high (sets pulse inputs low due to inverted
logic)
GPIO.output(pulseCW,GPIO.HIGH)

def moveTask():
    global moveState
    global nextMoveState
    global motionParamsSet
    global timeSentHome
    global resumeState
    global glueRefPos
    global glueOnTime
    global glueOffTime
    global absPulseCount
    global glueStart
    global blockAJU
    global bus
    global slideInHome
    global glueHomeCMD
    global gluePos
    global moveTaskDone
    global gluePosCMD
    global startGlueCMD
    global glueLevel
    global continueAfterGlueReady

    moveState = nextMoveState

    if moveState == 0:      #Send the Home signal to the slide when commanded by the scheduler
        if glueHomeCMD:
            blockAJU = [True,True]
            glueHomeCMD = False
            glueEnable() #Send the enable command to the arduino upon startup
            GPIO.output(GPIOZHOME, GPIO.HIGH)
            timeSentHome = time.time()
            nextMoveState = 10

        elif moveState == 10: #Wait for 'relayOnTime' then release Home Command
            if time.time()-timeSentHome >relayOnTime:
                GPIO.output(GPIOZHOME, GPIO.LOW)
                nextMoveState = 20

        elif moveState == 20: #Wait for 'waitForSlideTime' then go to state 3
            if time.time()-timeSentHome>waitForSlideTime:
                nextMoveState = 30

        elif moveState ==30:      #Wait for the driver to give the 'ready' signal (Home operation is
complete).
            if GPIO.input(homeEndIn):
                slideInHome = True
                absPulseCount=0
                nextMoveState =100

        ##### BEGIN FORWARD 1
        elif moveState ==100:      #Slide is in Home and ready. Set motion parameters to go to position
1
            setMotionParams(pos1,90)
            nextMoveState = 105
            printDebug(' State: ' + str(nextMoveState) + ' Waiting for Go Conditions')

        elif moveState ==105:      #Wait for the 'Ready' signal from the glue gun
            if continueAfterGlueReady:
                continueAfterGlueReady = False

```

```

        nextMoveState = 110
        glueLevel = 0
    # elif pauseForGlueReplace:
    #     glueDisable()
    #     nextMoveState = 107 #Placeholder
    #     timeSentHome = time.time()

elif moveState ==107:    #Wait for the actuator to go up
    if time.time()-timeSentHome>4:
        nextMoveState = 108

elif moveState == 108:
    if not(pauseForGlueReplace):
        glueEnable()
        nextMoveState =105

elif moveState ==110:
    if gluePosCMD ==1:
        nextMoveState = 115

elif moveState ==115:    #Move slide to position 1 (Just before AJU1)
    if not(abs(absPulseCount-destination)<1):
        updateSlide()
    else:
        gluePos =1
        nextMoveState = 120

elif moveState ==120: #Set motion parameters to glue and move to position 2 (Just before
AJU2)
    glueRefPos = absPulseCount
    setMotionParams(pos3,90)
    nextMoveState = 125
    printDebug( 'Current Position: ' + str(absPulseCount))
    printDebug( 'Destination: ' + str(destination))
    printDebug( 'GlueRefPos: ' +str(glueRefPos))
    printDebug( 'Glue Starting At: ' + str(glueRefPos + mm2counts(startGlue)))
    printDebug( 'Glue Stopping At: ' + str(glueRefPos + mm2counts(stopGlue)))
    printDebug( 'Waiting for Go Conditions')

## STOP. WAIT FOR SCHEDULER ##
elif moveState == 125: #Slide is waiting for 'Go' Conditions
    if startGlueCMD: #Wait for scheduler to check conditions and send the glue command
        startGlueCMD = False
        nextMoveState = 130

elif moveState == 130: #Slide is moving over AJU1, but not dispensing glue
    if (absPulseCount<mm2counts(pos1a-startGluePosOffset+4)):#abs(absPulseCount-
glueRefPos)<mm2counts(startGlue):
        updateSlide()
    else:
        DispenseOn()
        glueOnTime = time.time()
        glueStart[1] = True
        nextMoveState =135

elif moveState ==135:
    if time.time()-glueOnTime > glueDripStartDelay:
        setMotionParams(pos3,11)
        nextMoveState = 140

elif moveState == 140: #Slide is moving past AJU1 and dispensing glue
    if (absPulseCount<mm2counts(pos1b-stopGluePosOffset+3)) and time.time()-
glueOnTime<glueTimeout:#(abs(absPulseCount-glueRefPos)<mm2counts(stopGlue)) and time.time()-
glueOnTime <glueTimeout:

```

```

        updateSlide()
    else:
        DispenseOff()
        glueOffTime = time.time()
        nextMoveState = 145

elif moveState == 145:
    if time.time() - glueOffTime > glueDripStopDelay:
        setMotionParams(pos3, 90)
        nextMoveState = 150

elif moveState == 150: #Slide is getting out of the way of AJU 1
    if absPulseCount < mm2counts(pos2): #abs(absPulseCount - glueRefPos) < mm2counts(allowFlip):
        updateSlide()
    else:
        gluePos = 2
        blockAJU[0] = False
        nextMoveState = 155

elif moveState == 155: #Slide is continuing on to position 2 (Just before AJU2)
    if abs(absPulseCount - destination) > 1:
        updateSlide()
    else:
        gluePos = 3 #Let the scheduler know that the glue has reached position 3
        nextMoveState = 200

elif moveState == 200:
    if gluePosCMD == 5:
        nextMoveState = 205

##### BEGIN FORWARD 2
elif moveState == 205: #Set motion parameters to glue and move to position 5 (End of Slide)
    glueRefPos = absPulseCount
    setMotionParams(pos4, 90)
    nextMoveState = 210
    printDebug( 'Current Position: ' + str(absPulseCount))
    printDebug( 'Destination: ' + str(destination))
    printDebug( 'GlueRefPos: ' + str(glueRefPos))
    printDebug( 'Glue Starting At: ' + str(glueRefPos + mm2counts(startGlue)))
    printDebug( 'Glue Stopping At: ' + str(glueRefPos + mm2counts(stopGlue)))
    printDebug( 'Waiting for Go Conditions')

## STOP. WAIT FOR SCHEDULER ##
elif moveState == 210: #Slide is waiting for 'Go' Conditions
    if startGlueCMD: #Wait for scheduler to check conditions and send the glue command
        startGlueCMD = False
        nextMoveState = 215

elif moveState == 215: #Slide is moving over AJU2, but not dispensing glue
    if absPulseCount < mm2counts(pos3a - startGluePosOffset + 7): #abs(absPulseCount -
glueRefPos) < mm2counts(startGlue):
        updateSlide()
    else:
        DispenseOn()
        glueOnTime = time.time()
        glueStart[1] = True
        nextMoveState = 220

elif moveState == 220:
    if time.time() - glueOnTime > glueDripStartDelay:
        setMotionParams(pos4, 11)
        nextMoveState = 225

elif moveState == 225: #Slide is moving past AJU2 and dispensing glue

```

```

        if (absPulseCount<mm2counts(pos3b-stopGluePosOffset+7)) and time.time()-glueOnTime
<glueTimeout:#(abs(absPulseCount-glueRefPos)<mm2counts(stopGlue)) and time.time()-glueOnTime
<glueTimeout:
            updateSlide()
        else:
            DispenseOff()
            glueOffTime = time.time()
            nextMoveState =230

    elif moveState ==230:
        if time.time()-glueOffTime > glueDripStopDelay:
            setMotionParams(pos4,90)
            nextMoveState = 235

    elif moveState ==235: #Slide is getting out of the way of AJU 2
        if absPulseCount<mm2counts(pos4):#abs(absPulseCount-glueRefPos)<mm2counts(allowFlip):
            updateSlide()
        else:
            gluePos = 4
            blockAJU[1] = False
            moveTaskDone = True
            nextMoveState = 300

    elif moveState ==240: #Slide is continuing on to position 4 (Just after AJU 2)
        if abs(absPulseCount-destination)>1:
            updateSlide()
        else:
            gluePos = 4 #Let the scheduler know that the glue has reached position 4
            nextMoveState = 300
            moveTaskDone = True

##### BEGIN REVERSE 2
    elif moveState ==300: #Set motion parameters to glue and move to position 2 (Just before
AJU2)
        glueRefPos = absPulseCount
        setMotionParams(pos2,90)
        nextMoveState = 305
        printDebug( 'Current Position: ' + str(absPulseCount))
        printDebug( 'Destination: ' + str(destination))
        printDebug( 'GlueRefPos: ' +str(glueRefPos))
        printDebug( 'Glue Starting At: ' + str(glueRefPos - mm2counts(allowFlip - stopGlue)))
        printDebug( 'Glue Stopping At: ' + str(glueRefPos - mm2counts(allowFlip-startGlue)))
        printDebug( 'Waiting for Go Conditions')

## STOP. WAIT FOR SCHEDULER ##
    elif moveState == 305: #Slide is waiting for 'Go' Conditions
        if startGlueCMD: #Wait for scheduler to check conditions and send the glue command
            startGlueCMD = False
            nextMoveState = 310

    elif moveState ==310: #Slide is moving to the gluestart position on AJU 2 in reverse direction
        if absPulseCount>mm2counts(pos3b+startGluePosOffset+1):#abs(absPulseCount-
glueRefPos)<mm2counts(startGlueRev-10):
            updateSlide()
        else:
            DispenseOn()
            glueOnTime = time.time()
            glueStart[1] = True
            nextMoveState =315

    elif moveState ==315: #Slide is waiting in the gluestart position for glue to reach brick
        if time.time()-glueOnTime > glueDripStartDelay:
            setMotionParams(pos2,11)
            nextMoveState = 320

    elif moveState == 320: #Slide is moving past AJU2 in reverse direction and dispensing glue

```

```

        if (absPulseCount>mm2counts(pos3a+stopGluePosOffset-2)) and time.time()-glueOnTime
<glueTimeout:#{abs(absPulseCount-glueRefPos)<mm2counts(stopGlueRev-10)) and time.time()-glueOnTime
<glueTimeout:
            updateSlide()
        else:
            DispenseOff()
            glueOffTime = time.time()
            nextMoveState = 325

    elif moveState ==325: #Slide is waiting in the gluestop position for glue to stop dripping
        if time.time()-glueOffTime > glueDripStopDelay:
            setMotionParams(pos2,90)
            nextMoveState = 330

    elif moveState ==330: #Slide is getting out of the way of AJU 2
        if absPulseCount>mm2counts(pos3):#abs(absPulseCount-glueRefPos)<mm2counts(allowFlip):
            updateSlide()
        else:
            gluePos = 3
            blockAJU[1] = False
            nextMoveState = 335

    elif moveState ==335: #Slide is continuing on to position 2 (Just before AJU1 in reverse
direction)
        if abs(absPulseCount-destination)>1:
            updateSlide()
        else:
            gluePos = 2 #Let the scheduler know that the glue has reached position 2
            nextMoveState = 400

##### BEGIN REVERSE 1
    elif moveState ==400: #Set motion parameters to glue and move to position 1 (Just after AJU1
in reverse direction)
        glueRefPos = absPulseCount
        setMotionParams(pos1,90)
        nextMoveState = 405
        printDebug( 'Current Position: ' + str(absPulseCount))
        printDebug( 'Destination: ' + str(destination))
        printDebug( 'GlueRefPos: ' +str(glueRefPos))
        printDebug( 'Glue Starting At: ' + str(glueRefPos - mm2counts(allowFlip - stopGlue)))
        printDebug( 'Glue Stopping At: ' + str(glueRefPos - mm2counts(allowFlip-startGlue)))
        printDebug( 'Waiting for Go Conditions')

## STOP. WAIT FOR SCHEDULER ##
    elif moveState == 405: #Slide is waiting for 'Go' Conditions
        if startGlueCMD: #Wait for scheduler to check conditions and send the glue command
            startGlueCMD = False
            nextMoveState = 410

    elif moveState ==410: #Slide is moving to the gluestart position on AJU 1 in reverse direction
        if absPulseCount>mm2counts(pos1b+startGluePosOffset-5):#abs(absPulseCount-
glueRefPos)<mm2counts(startGlueRev):
            updateSlide()
        else:
            DispenseOn()
            glueOnTime = time.time()
            glueStart[0] = True
            nextMoveState =415

    elif moveState ==415: #Slide is waiting in the gluestart position for glue to reach brick
        if time.time()-glueOnTime > glueDripStartDelay:
            setMotionParams(pos1,11)
            nextMoveState = 420

    elif moveState == 420: #Slide is moving past AJU1 in reverse direction and dispensing glue

```



```

        if (absPulseCount>mm2counts(pos1a+stopGluePosOffset-2)) and time.time()-glueOnTime
<glueTimeout:#(abs(absPulseCount-glueRefPos)<mm2counts(allowFlip -startGlue)) and time.time()-
glueOnTime <glueTimeout:
            updateSlide()
        else:
            DispenseOff()
            glueOffTime = time.time()
            nextMoveState = 425

    elif moveState ==425: #Slide is waiting in the gluestop position for glue to stop dripping
        if time.time()-glueOffTime > glueDripStopDelay:
            setMotionParams(10,90)
            nextMoveState = 430

    elif moveState ==430: #Slide is getting out of the way of AJU 1
        if abs(absPulseCount-destination)>1:#abs(absPulseCount-glueRefPos)<mm2counts(allowFlip):
            updateSlide()
        else:
            gluePos = 1
            blockAJU[0] = False
            moveTaskDone = True
            nextMoveState = 100

def updateSlide(): #This subtask controls the pulse input to move the slide to its destination
    global absPulseCount
    global inPos
    global pulseEdgeTime
    global pulseEntryFlag
    global pulseState
    global nextPulseState
    pulsePins = [pulseCCW, pulseCW]
    pulseState = nextPulseState
    if time.time()-pulseEdgeTime >0.5*pulseDuration:
        pulseEdgeTime = time.time()
        if pulseState ==0:
            absPulseCount = absPulseCount + GPIO.input(moving)*(2*direction - 1)
            GPIO.output(pulsePins[direction],GPIO.LOW)
            nextPulseState = 1
        elif pulseState ==1:
            GPIO.output(pulsePins[direction],GPIO.HIGH)
            nextPulseState = 0

def scheduler():
    global nextSysState
    global sysStateChangeTime
    global blockAJU
    global AJUreadyCMD
    global AJUfinishCMD
    global AJUisReady
    global gluePosCMD
    global gluePos
    global slideInHome
    global startGlueCMD
    global glueHomeCMD
    global AJUEnableCMD
    global AJUEnableDone
    global AJUStopperCMD
    global continueAfterGlueReady
    global airPressureWarning
    global pauseRequest
    global systemIsPaused
    global ignorePause
    global stopperIsUp
    global pauseForGlueReplace
    global dispenseCounter
    global initializeTime

```

```

global last10startTime

sysState = nextSysState

if sysState ==0:#System has just turned on
    glueDisable()
    glueReadyOff()
    DispenseOff()
    ignorePause = True
    nextSysState = 1
    print "Version " + str(version)
    print "**IMPORTANT* VERIFY THE FOLLOWING BEFORE OPERATING:"
    print "\n    Air Pressure Set to 40 psi"
    print "\n    Power Switch Set to All Power"
    print "\n    Glue Gun LED is Solid Green"
    print "\n    Glue Gun Is Oriented Vertically"
    print "\n    AJUs are Down and Clear of All Obstructions"
    print "\nPress START button to initialize"

if sysState ==1:
    if not(GPIO.input(goButton)):
        nextSysState = 5

if sysState ==5: #Sends the glue to home
    if not((AJUisPresent[0] and bus[0].read_pin(14)) or (AJUisPresent[1] and
bus[1].read_pin(14))): #
        printDebug( "glueHomeCMD sent")
        glueReadyOn()
        printDebug("Readying glue gun.")
        glueHomeCMD=True    #Needed for move task
        nextSysState = 10

elif sysState ==10: #Wait for home complete
    if slideInHome:
        slideInHome = False
        #print "AJU Motors Homing. Interference May Impact Position Tracking"
        AJUEnableCMD = [True,True]#Allow AJU motors to home

        sysStateChangeTime = time.time()
        nextSysState =12 #

if sysState ==12: #Wait 2 seconds to allow homing to begin
    if ((not(AJUisPresent[0]) or AJUEnableDone[0]) and (not(AJUisPresent[1]) or
AJUEnableDone[1]))):
        AJUEnableDone = [False,False]
        initializeTime = time.time()
        nextSysState = 13

##System will return to this state after the cycle has completed
if sysState ==13: #Check if homing has completed (flippers are down, then activate the
stoppers)
    if not((AJUisPresent[0] and bus[0].read_pin(14)) or (AJUisPresent[1] and
bus[1].read_pin(14))):
        AJUStopperCMD = [True,True]
        nextSysState = 15

elif sysState == 15: #Wait for the glue to be ready
    if GPIO.input(glueReplaceIn): #Glue is low
        glueDisable()
        glueReadyOff()
        print "\n*Glue Replacement Needed*"
        print "Wait for glue to finish retracting  (GD: " + str(dispenseCounter)+ ")"
        dispenseCounter = 0
        nextSysState=17

elif GPIO.input(glueReady) and all(stopperIsUp):
    #print "Glue is ready"

```

```

        stopperIsUp = [False,False]
        printDebug( '...glueReady Signal Received')
        print '\nLoad bricks on both AJUs, then press START button to begin forward operation
(GD: ' + str(dispenseCounter)+ ' )'
        continueAfterGlueReady = True
        nextSysState = 20

    elif sysState ==17:
        if not(GPIO.input(glueReplaceIn)): #Wait for glue replace input to go low. This indicates
the glue is fully retracted
            print "\nLoad glue, then remove hands from machine and \npress START button "
            nextSysState =18

    elif sysState == 18:    #Glue is Low
        if not(GPIO.input(goButton)):
            print "\nVerifying glue replacement. Please Wait."
            glueEnable()
            glueReadyOn()
            nextSysState =15

    elif sysState ==20: #Wait for user to release the 'Go Button'
        if GPIO.input(goButton):
            nextSysState =25

    elif sysState ==25: #Wait for user to press the 'Go Button'
        if not(GPIO.input(goButton)):
            print "\nStarting Operation. Do not touch machine while in operation."
            AJUreadyCMD = [True,True]
            printDebug( "Sending AJUs to ready position...")
            nextSysState = 40

    elif sysState ==40: #Wait for acknowledgements of ready command
        if (not(AJUisPresent[0]) or AJUisReady[0]) and (not(AJUisPresent[1]) or
AJUisReady[1]):#all(AJUisReady):
            AJUisReady[0] = False
            AJUisReady[1] = False
            printDebug( "...AJUs are ready")
            printDebug( "Sending glue to ready position 1")
            gluePosCMD = 1
            nextSysState =50

    elif sysState ==50:
        if gluePos ==1:
            printDebug( "Glue is in ready position 1")
            printDebug( "Verifying AJU1 is in down position")
            nextSysState = 60

    elif sysState ==55:
        if not(GPIO.input(goButton)):
            systemIsPaused = False
            print "\nSystem resuming after PAUSE operation"
            nextSysState = 60

    elif sysState ==60:
        if not(bus[0].read_pin(14)):
            gluePosCMD = 2
            startGlueCMD =True
            nextSysState = 70
            printDebug( "Dispensing glue on AJU 1")

    elif sysState ==70:
        if gluePos ==2:
            printDebug( "Glue in Position 2. AJU 1 proceeding with assembly")
            AJUfinishCMD[0] =True

```

```

        ignorePause = False
        printDebug( 'Waiting for AJU 1 to finish')
        nextSysState = 80

elif sysState ==80:
    if gluePos ==3:
        printDebug( "Glue in ready position 3")
        printDebug( "Verifying AJU 2 is in down position")
        if pauseRequest:
            systemIsPaused = True
            print "System paused (80). Press START button to resume"
            pauseRequest = False
            nextSysState =85
        else:
            nextSysState =90

elif sysState ==85:
    if not(GPIO.input(goButton)):
        systemIsPaused = False
        print "\nSystem resuming after PAUSE operation"
        nextSysState = 90

elif sysState ==90:
    if not(bus[1].read_pin(14)):
        ignorePause = True
        gluePosCMD = 5
        startGlueCMD= True
        nextSysState =100
        printDebug( "Dispensing glue on AJU 2")

elif sysState ==100:
    if gluePos ==4:
        printDebug( "Glue in Position 4. AJU 2 proceeding with assembly")
        AJUfinishCMD[1] = True
        printDebug( "Waiting for AJU 2 to finish")
        nextSysState = 110

elif sysState ==110:
    if AJUtaskDone[0]:
        print "\nAJU 1 assembly complete"
        AJUtaskDone[0] = False
    if AJUtaskDone[1]:
        print "\nAJU 2 assembly complete"
        AJUtaskDone[1] = False
        AJUStopperCMD = [True,True]
        print '\nForward operation complete'
        print "-----"
        print ""
        nextSysState = 315

elif sysState == 315: #Wait for the glue to be ready
    # if glueLevel ==1:
    #     print "\nReplace Glue Now. When finished, clear hands from glue gun and press START
button"
    #     pauseForGlueReplace = True
    #     sysStateChangeTime = time.time()
    #     nextSysState = 17
    if GPIO.input(glueReplaceIn): #Glue is low
        glueDisable()
        glueReadyOff()
        print "\n*Glue Replacement Needed*"
        print "Wait for glue to finish retracting  (GD: " + str(dispenseCounter)+ ")"
        dispenseCounter = 0
        nextSysState=317

elif GPIO.input(glueReady) and all(stopperIsUp):

```

```

        #print "Glue is ready"
        stopperIsUp = [False,False]
        printDebug( '...glueReady Signal Received')
        print '\nLoad bricks on both AJUs, then press START button to begin reverse operation
(GD: ' + str(dispenseCounter)+ ')\n'
        # if absDispenseCounter%10 == 0:
        #     print "Rate Since Power On: " + str((time.time()-initializeTime)/absDispenseCounter)
+ " seconds/brick"
        #     print "(Last 10: " + str((time.time()-last10startTime)/10)+ "seconds/brick)"
        #     last10startTime = time.time()
        continueAfterGlueReady = True
        nextSysState = 325

    elif sysState ==317:
        if not(GPIO.input(glueReplaceIn)):
            print "\nLoad glue, then remove hands from machine and \npress START button "
            nextSysState =318

    elif sysState == 318: #Glue is Low
        if not(GPIO.input(goButton)):
            print "\nVerifying glue replacement. Please Wait."
            glueEnable()
            glueReadyOn()
            nextSysState =315

    elif sysState ==325: #Wait for user to press the 'Go Button'
        if not(GPIO.input(goButton)):
            print "\nStarting reverse operation. Do not touch machine while in operation."
            nextSysState =330

    elif sysState ==330: # Send all AJUs to glue ready position
        AJUreadyCMD = [True,True]
        nextSysState = 340
        printDebug( "Sending AJUs to ready position...")

    elif sysState ==340: #Wait for acknowledgements of ready command
        if (not(AJUisPresent[0]) or AJUisReady[0]) and (not(AJUisPresent[1]) or
AJUisReady[1]):#all(AJUisReady):
            AJUisReady[0] = False
            AJUisReady[1] = False
            printDebug( "...AJUs are ready")
            nextSysState = 360
            # if pauseRequest:
            #     systemIsPaused = True
            #     print "System paused (340). Press START button to resume"
            #     pauseRequest = False
            #     nextSysState =355
            # else:
            #     nextSysState =360

    elif sysState ==355:
        if not(GPIO.input(goButton)):
            systemIsPaused = False
            print "\nSystem Resuming"
            nextSysState = 90

    elif sysState ==360:
        if not(bus[1].read_pin(14)):
            ignorePause = False
            gluePosCMD = 2
            startGlueCMD =True
            nextSysState = 370
            printDebug( "Dispensing glue on AJU 2")

    elif sysState ==370:
        if gluePos ==3:
            printDebug( "Glue in Position 2. AJU 2 proceeding with assembly")

```

```

        AJUfinishCMD[1] = True
        printDebug( 'Waiting for AJU 2 to finish')
        nextSysState = 380

elif sysState == 380:
    if gluePos == 2:
        printDebug( "Glue in ready position 2")
        printDebug( "Verifying AJU 1 is in down position")
        if pauseRequest:
            systemIsPaused = True
            print "\nSystem paused. Press START button to resume"
            pauseRequest = False
            nextSysState = 385
        else:
            nextSysState = 390

elif sysState == 385:
    if not(GPIO.input(goButton)):
        systemIsPaused = False
        print "System Resuming"
        nextSysState = 390

elif sysState == 390:
    if not(bus[0].read_pin(14)):
        ignorePause = True
        gluePosCMD = 1
        startGlueCMD = True
        nextSysState = 400
        printDebug( "Dispensing glue on AJU 1")

elif sysState == 400:
    if gluePos == 1:
        printDebug( "Glue in Position 1. AJU 2 proceeding with assembly")
        AJUfinishCMD[0] = True
        printDebug( "Waiting for AJU 1 to finish")
        nextSysState = 410

elif sysState == 410:
    if AJUtaskDone[1]:
        print "\nAJU 2 assembly complete"
        AJUtaskDone[1] = False
    if AJUtaskDone[0]:
        print "\nAJU 1 assembly complete"
        AJUtaskDone[0] = False
        AJUStopperCMD = [True, True]
        print '\nReverse operation complete'
        print "-----"
        print ""
        nextSysState = 13

def watchDogTask(): #This task monitors for warnings, alarms, and pause button presses
    global watchDogState
    global nextWatchDogState
    global pauseStateEntryFlag
    global pausePressTime
    global pauseRequest
    global reverseRequest
    global alarmActive
    global dispRemaining
    global glueLevel
    global airPressureWarning
    global pressureWarningCooldown
    watchDogState = nextWatchDogState

if airPressureWarning and (time.time()-pressureWarningCooldown>5):
    pressureWarningCooldown = time.time()
    print("\n Verify pressurized air supply.")

```


C.2:

Filename: automatedGluing.ino

Description: This code runs the glueTask to control the glue dispensing subsystem. This task uses input signals from an external controller to enable the actuator, go to the ready position, and dispense from the ready position. This task also sends two output signals for glueReady and glueLow to the external controller. Actuator control is performed through three output signals to an H-Bridge for direction 1, direction 2, and Enable. This task obtains position and force feedback from the actuator through a linear potentiometer, and current sensing resistor.

```
//M. Jouaneh Added operation by timer on 5/28/2019
// Changed to use H-bridge and force sensor 6/10/2019
//Version 10 Low Glue Value Changed to 550

#define DIR1 9
#define DIR2 8
#define PWMpin 10
void SetPosition(int);
void DoControl(void);
void GlueTask(void);

int GlueState;
int GlueNextState = 1; //Initialized to start at this state
int Go2ReadyPosition = 0;
int DispenseOn = 0;
int glueReady = 0; //Signal to send to the Raspberry Pi when ready
long LastTime;
long LastTime2;
int Enabled; // signal to send the actuator up or down
float sumerr = 0.0;
int Tsamp = 5;
int advpos = 11;
int retpos = 1;
int prev = 0;
int actpos2;
int lastForceReading = 0;
long GlueOnTime = 6000; //in ms
long RetractDelay = 4000;
float despos;
int despos2;
int ForceLevel = 10; //10
int Count;
int lowGlueValue = 550;
int glueLowVar = 0;
bool actuatorUp = false;
int posSum = 0;

#define UpSwitch 4
#define DispSwitch 7
#define ReadySwitch 3
#define glueLow 11
#define PosChan 0
#define ForceChan 5
#define ready2Pi 2

void setup()
{
    //ForceLevel = 205;
```



```

//set pins
pinMode(UpSwitch, INPUT);
pinMode(DispSwitch, INPUT);
pinMode(ReadySwitch, INPUT);
pinMode(ready2Pi, OUTPUT);
pinMode(glueLow, OUTPUT);
pinMode(DIR1, OUTPUT);
pinMode(DIR2, OUTPUT);
pinMode(PWMPin, OUTPUT);
//open serial window for log
Serial.begin(38400);

LastTime2 = millis();
Enabled = digitalRead(4);
Go2ReadyPosition = 0;
DispenseOn = 0;

digitalWrite(glueLow, LOW);
glueLowVar = 0;
}

void loop()
{
  Serial.print("Enabled ");
  Serial.print(Enabled);
  Serial.print(" State ");
  Serial.print(GlueState);
  Serial.print(" Count ");
  Serial.print(Count);
  Serial.print(" DispenseOn ");
  Serial.print(DispenseOn);
  Serial.print(" Go2ReadyPos ");
  Serial.print(Go2ReadyPosition);
  Serial.print(" glueReady ");
  Serial.print(glueReady);
  Serial.print(" Glue Low ");
  Serial.print(glueLowVar);
  Serial.print(" Act pos ");
  Serial.print(actpos2);
  Serial.print(" Force ");
  Serial.println(lastForceReading);
  //Go2ReadyPosition = 1; //moved into setup

  GlueTask();
}

void GlueTask(void)
{
  Enabled = digitalRead(4);
  DispenseOn = digitalRead(7);
  Go2ReadyPosition = digitalRead(3);

  GlueState = GlueNextState;

  if ((Enabled == 0) && (actuatorUp == false))
  {
    {
      digitalWrite(ready2Pi, LOW);

      SetPosition(1); //Move actuator to top position
      GlueNextState = 1;
      sumerr = 0; // reset error sum variable
      DoControl();
      actpos2 = analogRead(PosChan);
      if (actpos2 < 100)
      {
        analogWrite(PWMPin, 0);
        actuatorUp = true;
        GlueNextState = 1;
        digitalWrite(glueLow, LOW);
      }
    }
  }
}

```

```

        glueLowVar = 0;
    }
}

else if (GlueState == 1) // Start state - actuator at 1% extension
{
    if (Go2ReadyPosition == 1)
    {
        Count = 0;
        prev = 0;
        actuatorUp = false;
        GlueNextState = 2;
    }
}

else if (GlueState == 2) //Move actuator down
{
    actpos2 = analogRead(PosChan); //read current position of actuator
    if (actpos2 > lowGlueValue) {
        digitalWrite(glueLow, HIGH);
        glueLowVar = 1;
    }
    if (actpos2 <= 950)
    {
        /////ADVANCE/////
        digitalWrite(DIR1, LOW);
        digitalWrite(DIR2, HIGH);
        analogWrite(PWMPin, 200); //set the duty cycle or speed
        GlueNextState = 3;
        Count = 0;
    }
}

else if (GlueState == 3) //Check force sensor
{
    actpos2 = analogRead(PosChan); //read position sensor
    lastForceReading = analogRead(ForceChan);
    if (actpos2 > lowGlueValue) {
        digitalWrite(glueLow, HIGH);
        glueLowVar = 1;
    }
    if (actpos2 >= 1010) // if actuator reached travel limit
    {
        //Enabled = 0;
        //Go2ReadyPosition = 0;
        digitalWrite(ready2Pi, LOW);
        GlueNextState = 1;
    }
}

else if (lastForceReading >= ForceLevel) //if force reading above the threshold level
{
    Count = Count + 1;
    prev = 1;
    GlueNextState = 4;
}

else if (GlueState == 4) //wait for the proper # of counts
{
    lastForceReading = analogRead(ForceChan);

    posSum = 0;
    for (int i = 1; i < 10; i++) {
        posSum = posSum + analogRead(PosChan);
    }
    actpos2 = posSum / 10; //analogRead(PosChan);

    if (actpos2 > lowGlueValue)
    {
        digitalWrite(glueLow, HIGH);
        Serial.print("GLUE LOW");
    }
}

```

```

        glueLowVar = 1;
    }
    if (actpos2 >= 1010) // if actuator reached travel limit
    {
        digitalWrite(ready2Pi, LOW);
        GlueNextState = 1;
    }

    else if (lastForceReading >= ForceLevel)
    {
        if (prev >= 1)
        {
            Count = Count + 1;
        }
        else
        {
            prev = 0;
            Count = 0;
            GlueNextState = 3;
        }
        if (Count >= 5) //play with count #
        { //Actuator is ready
            digitalWrite(ready2Pi, HIGH);
            //Go2ReadyPosition =0; //Reset the ready position command
            GlueNextState = 45;
            LastTime = millis();
            if (actpos2 < (lowGlueValue-100)) {
                digitalWrite(glueLow, LOW); //After setup, this is the only way to return glueLow
to LOW
                glueLowVar = 0;
            }
            digitalWrite(DIR1, HIGH); //reverse motion direction
            digitalWrite(DIR2, LOW);
        }
    }

    else if (GlueState == 45) // Reverse motion
    {
        if (( millis() - LastTime) >= 150)
        {
            analogWrite(PWMPin, 0); //stop motion

            GlueNextState = 5;
        }
    }

    else if (GlueState == 5) //Waiting for new dispense command
    {
        if (DispenseOn == 1)
        {
            LastTime = millis();
            digitalWrite(DIR1, LOW);
            digitalWrite(DIR2, HIGH); //set direction down
            analogWrite(PWMPin, 220);
            GlueNextState = 6;
        }
    }

    else if (GlueState == 6) // Reverse motion
    {
        if (( millis() - LastTime) >= GlueOnTime || (DispenseOn == 0))
        {

            //Reverse motion direction
            // actpos2 = analogRead(PosChan);

            posSum = 0;
            for (int i = 1; i < 10; i++) {
                posSum = posSum + analogRead(PosChan);
            }

```

```

    actpos2 = posSum / 10; //analogRead(PosChan);

    if (actpos2 > lowGlueValue) {
        digitalWrite(glueLow, HIGH);
        glueLowVar = 1;
    }
    despos2 = actpos2 - 15;
    if (despos2 <= 10)
    {
        despos2 = 10;
    }
    digitalWrite(DIR1, HIGH);
    digitalWrite(DIR2, LOW);
    LastTime = millis();
    GlueNextState = 7;
}

else if (GlueState == 7) // Actuator Reached Retracted position
{
    if ((millis() - LastTime) >= 150) //(actpos2 <= despos2)
    {
        //DispenseOn = 0; //Reset dispense command if not reset by user
        GlueNextState = 5; //Go to next ready position and wait for the next dispense command
        analogWrite(PWMPin, 0);
    }
}

} // end of GlueTask

void SetPosition(int val)
{
    // Val is a number between 0 - 100
    despos = (val / 100.) * 1000.;
}

void DoControl(void)
{
    int error;
    float kp = 10;
    float ki = 0.0;
    float output;
    int pwm;
    int current;
    int actpos;

    if ((millis() - LastTime2) >= Tsamp)
    {
        LastTime2 = millis();
        actpos = analogRead(PosChan);
        current = analogRead(ForceChan);
        error = despos - actpos;

        output = kp * error + ki * sumerr;
        sumerr = error + sumerr;
        if (output > 0)
        {
            digitalWrite(DIR1, LOW);
            digitalWrite(DIR2, HIGH);
        }
        else
        {
            digitalWrite(DIR1, HIGH);
            digitalWrite(DIR2, LOW);
        }

        if (output >= 1023)
            output = 1023;
        if (output <= -1023)
            output = -1023;
        pwm = (abs(output) / 1023.0) * 255;
    }
}

```

```
    analogWrite(PWMPin, pwm);  
  }  
}
```

Bibliography

"Adhesives and Sealants 101: Hot Melts". Adhesives & Sealants Industry. October 1, 2008.

Retrieved 20 February 2020.

Agin, Gerald J. "Computer vision systems for industrial inspection and assembly." *Computer* 5 (1980): 11-20.

A. Björnsson, Automated layup and forming of prepreg laminates, 2017.

Boeing, Geoff, Daniel Church, Haley Hubbard, Julie Mickens, and Lili Rudis. 2014. "LEED-ND and Livability Revisited," *Berkeley Planning Journal* 27, 27 (1). doi:10.5070/BP327124500.

Coyle, Dennis J. "Knife and roll coating." In *Liquid film coating*, pp. 539-571. Springer, Dordrecht, 1997.

Deangelis, T. 1994. "Clinton's Climate Change Action Plan," *Environmental Health Perspectives* 102, 102 (5): 448-49.

DiFilippo, Nicholas M., "Framework for the Automated Disassembly of Electronic Waste Using the Soar Cognitive Architecture" (2016). Open Access Dissertations. Paper 514. https://digitalcommons.uri.edu/oa_diss/514

Grünewald, Jonas, Patricia Parlevliet, and Volker Altstädt. "Manufacturing of thermoplastic composite sandwich structures: A review of literature." *Journal of Thermoplastic Composite Materials* 30, no. 4 (2017): 437-464.

The Hartford Loss Control Department. "Exterior Insulation and Finish Systems". *Technical Information Paper Series*. TIPS S 140.012. 1997. 1.

Jensen, Thor, and Hadi Dowlatabadi. "Federal Tax Credits and Residential Investment in Renewable Energy.", 2017.

- Jerome, G, and Rene M Dupuis. n.d. "The Response of Molded Expanded Polystyrene (EPS) Insulation Board to Various Attachment Methods."
- Jouaneh, Musa. "Chapter 6: Control Software." *Fundamentals of Mechatronics*. Cengage Learning, Stamford, CT, 2013.
- Karlsson, Kicki F., and B. TomasÅström. 1997. "Manufacturing and Applications of Structural Sandwich Components." *Composites Part A: Applied Science and Manufacturing* 28. doi:[https://doi-org.uri.idm.oclc.org/10.1016/S1359-835X\(96\)00098-X](https://doi-org.uri.idm.oclc.org/10.1016/S1359-835X(96)00098-X).
- Maharana, T, Y. S Negi, and B Mohanty. 2007. "Review Article: Recycling of Polystyrene," *Polymer-Plastics Technology and Engineering* 46, 46 (7). Taylor & Francis Group: 729–36. doi:10.1080/03602550701273963.
- Newsham, Guy R., Sandra Mancini, and Benjamin J. Birt. *Do LEED-Certified Buildings Save Energy? Yes, but....* vol. 41, 2009
- Razban, A., and B. L. Davies. 1991. "An Automated System for Dispensing Adhesives." *International Journal of Adhesion and Adhesives* 11. doi:[https://doi-org.uri.idm.oclc.org/10.1016/0143-7496\(91\)90022-A](https://doi-org.uri.idm.oclc.org/10.1016/0143-7496(91)90022-A)
- Ribas-Xirgo, Lluís, José M. Moreno-Villafranca, and Ismael F. Chaile. "On using automated guided vehicles instead of conveyors." In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pp. 14. IEEE, 2013.
- Schumacher, Paul, and Jouaneh, Musa. "A Force Sensing Tool for Disassembly Operations." *Robotics and Computer-Integrated Manufacturing* 30.2 (2014): 206-17. Web.
- Shi, Jane, Glenn Jimmerson, Tom Pearson, and Roland Menassa. "Levels of human and robot collaboration for automotive manufacturing." In *Proceedings of the Workshop on Performance Metrics for Intelligent Systems*, pp. 95-100. 2012.
- Sprovieri, John. 2018. "Hot-Melts for Electronics Assembly." *Assembly* 61. Troy: BNP Media. <https://search.proquest.com/docview/2049987849>.

- Sto Corp. 2014. "Adhesive v. Mechanical Fastening of StoTherm®."
- Sujatha, G., and V. Perasiriyana. "Conveyor control and sorting module controlled by programmable logic controller." *International Journal of Engineering and Technology* 4, no. 2 (2014): 12-18.
- Troughton, Michael J, and International Institute of Welding. 2008. *Handbook of Plastics Joining*. PDL Handbook Series. US: William Andrew.
- Uddin, N., A. Vaidya, U. Vaidya, and S. Pillay. 2013. "12 - Thermoplastic Composite Structural Insulated Panels (CSIPs) for Modular Panelized Construction." Edited by Nasim Uddin. *Developments in Fiber-Reinforced Polymer (FRP) Composites for Civil Engineering*. Woodhead Publishing. doi:<https://doi.org/10.1533/9780857098955.2.302>
- The U.S. Energy Information Administration. Annual energy outlook with projections to 2040. Washington DC; 2016.
- Wu, Wenqi, Xingang Wang, Guan Huang, and De Xu. "Automatic gear sorting system based on monocular vision." *Digital Communications and Networks* 1, no. 4 (2015): 284-291.