Trinity University
Digital Commons @ Trinity

Engineering Senior Design Reports | Engineering Science Department

4-17-2008

Semiconductor Robot

Jude Ankrah
Trinity University

Marton Gergley
Trinity University

Mike Gonzalez
Trinity University

Charles Jarvis
Trinity University

Sean C. Kienle
Trinity University

Follow this and additional works at: https://digitalcommons.trinity.edu/engine_designreports

Repository Citation
Ankrah, Jude; Gergley, Marton; Gonzalez, Mike; Jarvis, Charles; and Kienle, Sean C., "Semiconductor Robot" (2008). *Engineering Senior Design Reports*. 13.
https://digitalcommons.trinity.edu/engine_designreports/13

This Restricted Campus Only is brought to you for free and open access by the Engineering Science Department at Digital Commons @ Trinity. It has been accepted for inclusion in Engineering Senior Design Reports by an authorized administrator of Digital Commons @ Trinity. For more information, please contact jcostanz@trinity.edu.

TRINITY UNIVERSITY

Final Design Report

Semiconductor Robot

Jude Ankrah
Marton Gergley
Mike Gonzalez
Charles Jarvis
Sean Kienle

Advisor: Michael Yockey

4/17/2008

A large line-following robot was built to transport semiconductor wafers in semiconductor factories. The criteria for a successful robot were decided upon prior to designing. The robot was designed using both mechanical and electrical engineering techniques to ensure that the final design met the outlined criteria. Each electrical subsystem was tested successfully prior to being installed on the robot. However once all the components were installed the robot was not able to move autonomously because the DC motors selected could not provide adequate torque. For future robot designs it is imperative that the motors be tested thoroughly because there is a substantial difference between specifications and actual performance.

1 Executive Summary

The task was to design and construct a robot to work in a semiconductor fabrication plant that could transport semiconductor wafer cassettes between stages in the fabrication process. The motivation was to provide a way for older fabrication plants to be competitive with new fabrication plants that feature overhead systems for transporting wafer cassettes.

In order for the robot to improve efficiency it needed to perform better than the current system in which the plant operators carry the cassettes to the next stage in the process. This meant that the robot had to move and navigate faster than an average person walking. The robot must not emit particles since it would be operating in a clean-room environment and it must be tall enough that the average worker could unload and load the cassette with minimal bending. The robot must also be able to protect the semiconductor wafers both during transport and in the event of a collision and it must be able to operate within the factory without hindering the flow of traffic in the factory which means it should not take up more room than is necessary.

A large line-following robot was designed to meet these constraints. A line following robot was selected because it is simple and nondestructive to install paths within the factory. The robot uses omni-directional wheels to allow the robot to translate instead of turning and an array of infrared sensors mounted on the bottom ensure that the robot stays on the line. The robot can operate autonomously by using an onboard computer that tells it to follow preprogrammed paths and only requires the operator to load or unload the cassette and then enter the next destination.

The mechanical components of the design (the frame, cassette holder, electronic housing, and wheel assemblies) were constructed without testing. All of the electrical components were tested prior to installation to ensure that they would work once installed and to attempt to decrease the amount of time spent debugging. Once the electronics were working properly, they were mounted onto the frame along with the battery. The test of the robot as a whole was not successful since the motors that were purchased were not capable of providing enough torques to move the robot. This problem arose because the torque specifications listed on the website were nearly double the measured torque.

Given this knowledge regarding the difference between motor specifications and their actual performance, it is recommended that a replacement motor is used that can provide adequate torque by means of either a higher gear ratio or a more powerful motor.

The robot successfully met the height requirement, the clean-room compatibility requirement, the size requirement and the wafer protection requirement. It did not meet the requirements that governed the robot in motion.

2 Table of Contents

3 Table of Figures

4 Table of Tables

5. Problem Statement

Semiconductor fabrication plants (also known as fabs) take semiconductor wafers and produce chips and integrated circuits that are used in everyday electronics. The production of these chips takes place in clean room compatible environment where the wafer cassettes are transported from one hallway (known as bays) in the factory to another. The construction of fabrication plants in on the rise as the demand for integrated circuits increases. Most of these newer fabrication plants use state of the art transport systems to move their wafers to one location in the factory to another. The wafers sometimes need to be moved to the opposite side of the plant in order to be processed and 20% of the factories use the newer overhead system to transport them. The other 80% are left to rely on manual labor to transport the wafer cassettes. A line following robot will allow the transportation of the wafer cassettes to not rely on manual labor which will decrease costs and increase productivity. This will also give the older fabrication plants a better ability to compete with the newer fabrication plants.

5.1 Constraints

With an overall design concept in mind, the constraints and working criteria for the project were established. The constraints and criteria were weighed according to importance and applied to proposed design ideas. The weighed constraints and criteria were applied to the various sub-part designs of the robot and were instrumental in the final design decision for the project. They were also instrumental in the decision making process as the design, construction and testing stages of the project progressed.

The first two constraints were imposed on the project as a result of its working environment. The semiconductor fabrication plant requires that several standards be met and therefore, these constraints were given the highest priority. The first constraint required that the robot be clean room compatible which means the robot cannot emit particles. Semiconductor fabrication plants required that everything and everyone working inside them be clean room conscious. This is due to the fact that the treatment of semiconductor wafers is very sensitive to outside particles. Therefore, the robot must follow fabrication plant regulations and must also be clean room compatible. Due to the critical nature of this constraint for anything that operates in this plant's environment, clean room compatibility was weighed the highest in the decision making process for the best design.

Another constraint that received the highest weighing factor was the loading and unloading height of the robot. Factories must make sure that when workers load and unload material they do so at a minimum height. This standardized height is implemented to prevent any injuries that could occur from the repetitive bending and lifting motions as well as lifting heavy objects. Since this robot must interact with the workers inside the fabrication plant, it must adhere to the rules of the factory. The height recommended for repetitive lifting by the Occupational Safety and Health Administration is half way between the mid-thigh and mid-chest, which is approximately 900 mm (1).

Aside from following fabrication plant regulations, the robot must also be able to perform its job without disrupting the inner operations of the fabrication or jeopardizing the semiconductor material that it must transport. The ability to protect the wafers during transportation was the last working constraint that received the highest weighing. Workers are instructed to carry the cassettes at an angle to prevent them from getting scratched.

Semiconductor wafers are very delicate and a small scratch can lead to a loss of valuable semiconductor material. The protection of the wafers is of the utmost importance within the factory and thus, it has the same importance in the design.

The last two constraints were weighed slightly less than the first three but were still weighted high relatively high when compared to the working criteria. The first of these two involves the size of the robot. Since the robot should not impair operations within the factory, importance was placed on building a robot that would not take up a lot of space. The robot must be able to operate in bays that are as narrow as 6 feet. While many of the bays are wider, most have racks lining the wall where raw materials are stored. The robot is also being designed to substitute manual labor and therefore its presence in the factory must not be more noticeable or disruptive than that of a worker. With this size constraint, it is believed that the width of the robot was more important than its length because the production bays within the factory where the robot is operating could be 50 feet long, but are only 6 feet wide and the robot should not obstruct movement through the bays.

The last constraint was cost. With a budget of only $1000 to work with for the entire project it is import for the design to be cost effective. This constraint was applied to the different design concepts by analyzing how easy it would be to build the proposed design within the budget constraint. While this constraint is important, it was not weighed as highly as the others because of the nature of this project. Estimates showed that the materials, mechanical and electrical components needed to build a robot of this scope would exceed the $1000 limit. Even the designs that were proposed because of their low cost prediction required several expensive components and bringing them about while staying within the budget would require clever

budgeting and good use of resources. Along with the constraints, six different working criteria were applied to the designs.

5.2 Working Criteria

The working criteria, like the constraints, were weighed according to importance. The first of these criteria determined the projected speed of the robot. A very important social consideration of this robot is that it might help older fabrication plants compete with the newer ones that use overhead transport systems. Also, the robot is being designed to substitute manual labor and therefore must be at least as efficient as a worker in the factory. Therefore, a speed criterion of at least a person walking (which is about 1.34 m/s) was implemented and the proposed designs were weighed according to their projected ability due to their mechanical and electrical designs, to meet this criterion.

The first of these criteria analyzed how easily the robot could potentially control its own speed. Different designs pose different mechanical challenges; the robot has to be able to change speed during operation to maneuver around the factory without damaging the wafers inside the cassette. This criterion analyzed the potential difficulty in controlling the speed of the robot taking into account the layout and driving mechanism. A mechanical analysis was performed to determine the ease of control of a proposed design. Since this was purely a mechanical criterion, it received a moderate weight because even if controlling the speed proves difficult, it is a problem that could be worked around and should not affect the overall score for that concept.

It is also important that the robot be able to recharge its battery between shifts. Therefore a criterion was created to determine how easy it would be for the robot to successfully maneuver to a docking station to recharge. This is dependent on the mechanical analysis and layout of the component within the robot, most importantly, the battery placement with respect to the driving components. While this is important, it is not crucial to the design of the project and therefore received the lowest weight out of all the criteria and constraints.

Turning ability was one of the most important criteria. This involves the ability of the robot to make turns from the main hallway in the factory to the individual bays where the wafers are kept. Turning ability is determined by the drive mechanism within the robot and the wheel layout. It also takes into account the line track needed for the robot to make successful turns into the bays. As previously stated, the robot must not be more of a disturbance to the workings of the factory than a worker. A very simple line track would allow the robot to stay out of the way while a more complex track might get in the way of workers. The more complex the line layout needed, the lower the ranking for this criterion.

Maneuverability received an average weight and determined how easy it would be to keep the robot on the line and adjust its position when and if it leaves the path. This also includes its different capabilities during motion such as changing directions and rotating without moving. This would allow the robot to perform a wide variety of tasks that might be needed during operation.

The criteria that ranked second highest was programming. Resources are limited by only having one electrical engineer with background in programming line following robots. When analyzing design concepts, it was important to keep this in mind that a very ambitious programming requirement could put a potentially solid mechanical design in jeopardy.

Finally, the last criterion was innovation. It was debated whether or not to include innovation as a criterion since often an innovative solution is not the easiest to implement nor is it always successful. In the end, it was decided that an innovative design would push the group to create a robot that is different from what has been seen in previous senior design presentations at Trinity University. Taking into account the educational aspect of this project, the group felt that an innovative design might also inspire other robotics students in the future. The group wanted to show that it is possible to take a different route and still build an efficient line following robot.

The set of working constraints and criteria were used to evaluate the viability of each proposed design. An analysis was performed to determine the mechanical and electrical components of each design and how these components measure up to the constraints and criteria set forth for the project. Finally, the analysis led to the decision of the final design: a translating line following robot using omni-directional wheels. A chart of the weighting criteria can be seen in Fig 1.

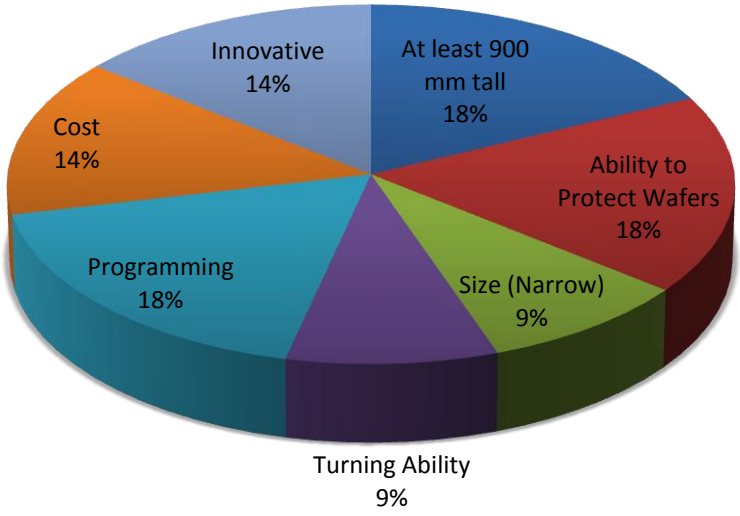


Figure 1: Pie-chart of weighting constraints

6. Final Design

The final design for the semiconductor carrying robot takes into account some societal constraints in addition to the design constraints and working criteria mentioned previously. Deliberate efforts were made to design for manufacturability, understand the political and economic implications, and the potential impact this robot could have on the environment.

6.1 Social Design Constraints

To make the robot easier for mass production, only readily available materials were used. Parts and accessories were mounted on the central frame which was a simple cubic-like structure. The robot is made primarily of square steel tubes and Plexiglas, both of which are available at most local hardware stores. Additionally, almost all of the screws used are 10-32. Using a standard screw size throughout the design removes the need to keep track of which screw goes where since the screws are all the same size and length and therefore interchangeable. Standard wheel type was also used. Interestingly, the wheel used had its gear box pre-mounted and incorporated into its structure. This dismissed the need for an elaborate system of gears which would make automation more difficult.

Using robots to transport semiconductor cassettes around a semiconductor fabrication plant will likely result in increased efficiency. An increase in efficiency means a greater output of the final product in the same amount of time. If the semiconductor plants in the United States can increase their efficiency it will result in a larger stake in the world market. Additionally, more consumer goods that use semiconductor wafers could be made in the United States without having to use imported semiconductors. This robot has the potential to have an impact on the global economy.

In this day and age where studies have indicated that the actions of human beings are having permanent impacts on the planet, it is especially important to take into consideration the possible environmental impact when designing. The large golf cart battery and the electronic components on the semiconductor robot have the potential to harm the environment if not disposed of properly. In addition, the improved efficiency in semiconductor fabrication plants could lead to more consumer products that also contain batteries and electronic components that may or may not be disposed of properly. While the future impact from consumer goods may not be measureable it is something to take into account.

6.2 Mechanical Design

The main purpose of the mechanical design of the semiconductor robot is to support the semiconductor cassette with minimal vibrations at the desired height while the robot navigates the factory. The mechanical design achieves all of the criteria while maintaining simplicity and functionality.

The main mechanical design from the ground up consists of four omni-directional wheels oriented so that in all four primary directions, two wheels are capable of driving and two wheels translate. Attached to the wheels are four DC motors oriented in a similar fashion so that two DC motors will drive the robot whenever it is moving. The motors are attached to the frame which is made of welded steel tube in a rectangular prism that stands one meter from bottom to top. Screwed onto the steel frame are Plexiglas sheets of various thicknesses, one on the bottom, one on the top, and one on each side. Sitting on top of the top Plexiglas sheet is the cassette holder, which is made of Plexiglas and foam insulation, and holds the cassette at the necessary 30 degree angle to minimize the wafers rattling inside the cassette. The initial design and the completed construction can be seen in Fig 2.

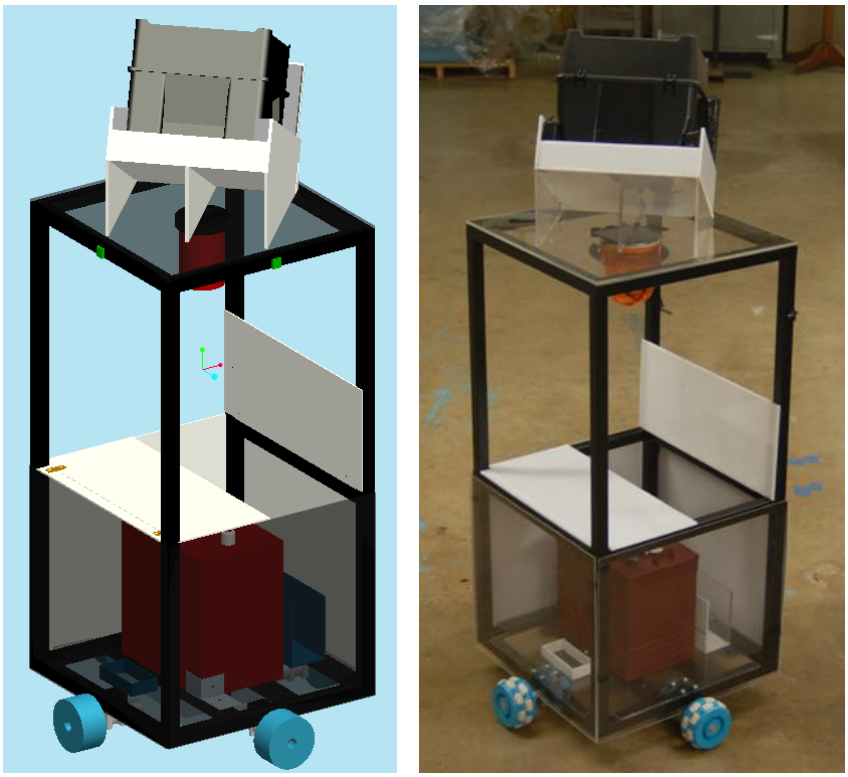


Figure 2: Side by side comparison of the original plans and the actual robot

6.2.1 Materials Used

The final mechanical design is composed of three core materials: square steel tubes, Plexiglas sheets, and insulating foam. Steel tube was used as the frame for the robot because it provides the necessary structural support to hold the battery and electronics and to keep the robot from collapsing under the weight of the cassette. The steel tube was selected because of its high strength to weight ratio. The steel tube has yield strength of 29,500 psi which is well over the demand that the robot will require, and the total mass of the steel used in the robot is only 21 lbs. which is light enough that the motors are able to move the robot at the desired velocity. Tubular steel does not emit particles under normal conditions, so it meets the criteria for clean room compatibility. Tubular steel was also selected because it is readily available in industry and if at

some point the robot should go into mass production the materials will not be difficult to find. The final reason that tubular steel was selected over other possible materials was because the tubular steel was available at no cost.

Plexiglas sheets were used to hold the electronics, provide housing for the battery compartment, and support the cassette on the top of the robot. Two thicknesses of Plexiglas were used in the robot; ½ inch thick Plexiglas was used for the bottom base, top base, and cassette holder, while ¼ inch Plexiglas was used as the housing for the electronics. Plexiglas was used because it is capable of providing the necessary support to hold the electronics and the wafer cassette and, like tubular steel, it does not emit particles. Moreover, Plexiglas is easy to work with and inexpensive to buy in local hardware stores and plastic suppliers.

Insulation was used to protect the cassette from inherent vibrations. The insulation was installed inside the cassette holder in double thickness to cradle the cassette and provide damping to the vibrations transferred from the motors. Insulation was used instead of a clean room alternative such as injection molded foam because of the low cost. In an effort to minimize the amount of particles that the insulation sheds, black electrical tape was wrapped around the cut edges.

6.2.2 Motors

The robot uses four Banebots RS-385 DC motors. The motors are arranged so that only two motors are driving the robot at a time, and which two depends on which direction the robot is traveling. An example of this arrangement can be seen in Fig. 3 where the red arrows correspond to the red motors and the blue arrows correspond to the blue motors.

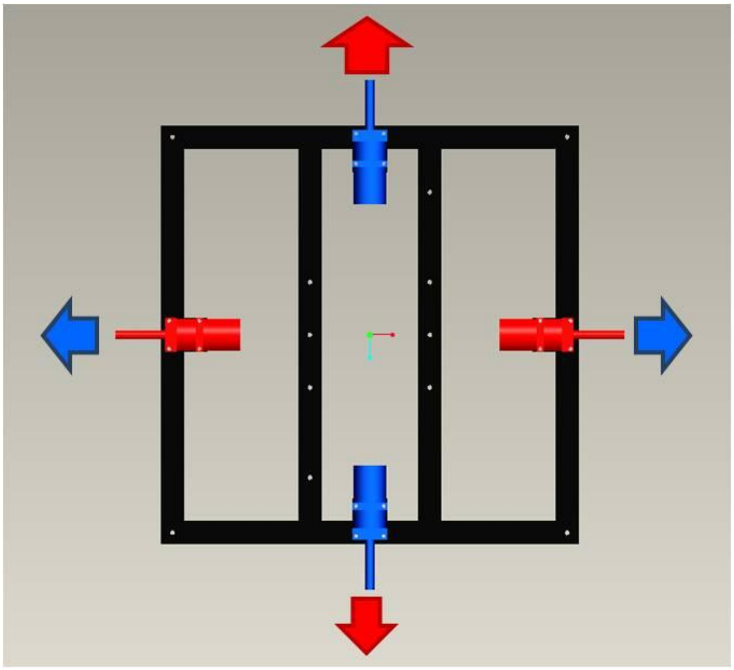


Figure 3: Direction of Driving Motors.

The Banebots RS-385 DC motors were selected after because they are capable of providing enough torque to allow the robot to accelerate to the required 3mph fairly quickly while their current draw from the battery was less than comparable alternatives. In addition, after careful calculation of bending moments, it was determined that the RS-385 motors with the thicker shaft diameter were a better choice. A thicker shaft diameter allows the motors to support the entire weight of the robot without the need for additional reinforcement.

6.2.3 Wheels

The defining feature of the semiconductor carrying robot is the wheels. The omni-directional wheels allow the robot to translate instead of turning. This is achieved through the use of several small rollers positioned along the outside of the wheel which make it possible for the wheel to roll freely in a path collinear to the central axis but maintains the ability to be driven when traveling normal to the central axis. A picture of the wheel can be seen in Fig 4.

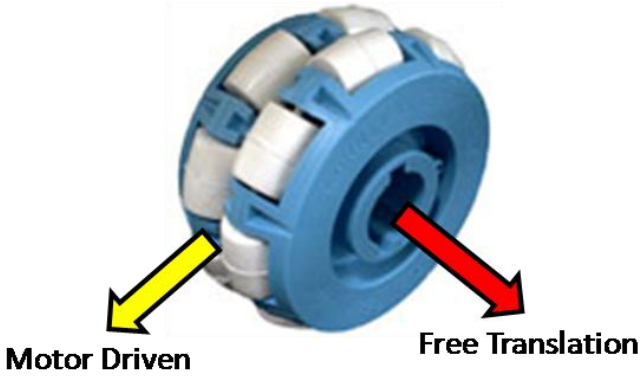


Figure 4: Diagram showing the modes of travel for the omni-directional wheel (3)

The wheel that was selected was the Transwheel 4202X from Kornylak because it is rated to carry the necessary load. Polypropylene rollers were selected instead of hard plastic or solid urethane because they have better grip on the floor than hard plastic rollers and will deform less under a heavy load than the solid urethane rollers (3).

To eliminate any potential slip between the motor and the wheel a keyway was cut into the wheel and the shaft collar and a thin piece of aluminum was inserted between the two. The keyway is shown in the above picture as the square section protruding from the inner radius. By adding a key and keyway none of the driving power from the motor to the wheel is lost to slippage.

6.2.4 Holder

The cassette holder is made of ¼ inch thick Plexiglas glued together using epoxy. Inside the holder where the cassette rests is covered with a dual layer of ½ inch thick insulation that absorbs the vibrations from the robot. The cassette rests in the holder and is tilted 30 degrees with respect to horizontal to minimize the wafers rattling within the cassette while it is being moved. The holder itself is oriented on top of the robot at a 45 degree angle relative to the direction of motion so that robot can translate in any of the four primary directions and the holder will always be ±45 degrees relative to the direction of motion. The holder and orientation can be seen in Fig 5.

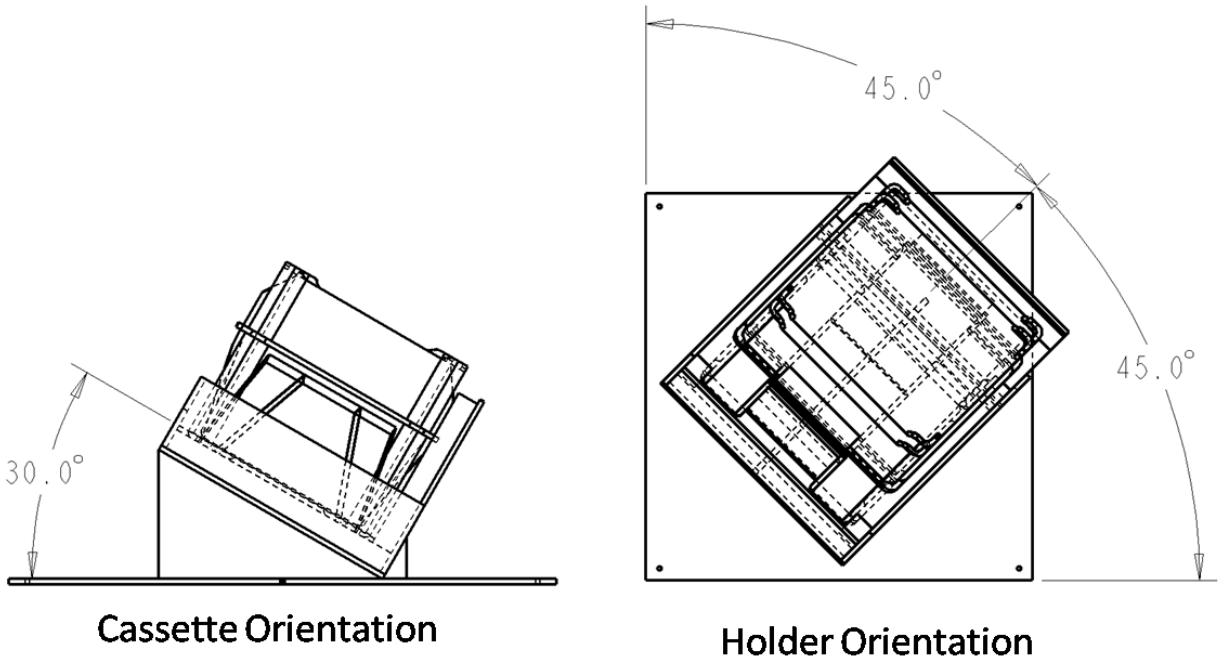


Figure 5: The orientation of the wafer cassette and the cassette holder on the robot.

6.3 Electrical Design

The purpose of the electrical design of the robot, in contrast to the mechanical design, is to make the robot move and navigate the factory floor safely. To do this, it will need several things:

- Sensors to let it see the outside world
- A controller to interpret the input, perform logic, and control the actuators
- A power source
- A warning system to make its presence known

6.3.1 Input

It was determined that the following inputs would serve the robot best: Infrared photopairs for line following, sonar rangefinders for gauging distance between objects and the robot, and a keypad to input the next destination.

6.3.1.1 Infrared Photopairs

The infrared photopairs used are Fairchild QRB1114s. The pair consists of one infrared LED and one infrared phototransistor. The photopair works by forward biasing the LED so it emits infrared light, which may or may not bounce off the surface it's pointed at. Any light that is reflected will saturate the phototransistor and current will begin to flow. A circuit diagram for the photopairs can be seen in Appendix B-1. The value of the resistor in series with the collector will determine the sensitivity of the photopair. In this case, a 4.7kΩ resistor was used. This allows for a range of nearly 0.25", compared to the range of about 0.125" with a 330Ω resistor. However, there is some risk in using such a large resistor. Using a larger resistor is akin to using

a larger gain in a control system—it will give a better response, but will introduce more noise into the system. However, since the area the robot is being designed for does not have sudden changes in floor color and elevation, the risk of using a large resistor is acceptable.

It should be noted that the photopairs cannot be used as is, since the voltage source being used (around 6VDC) is not friendly to TTL-type logic. The right half of the circuit shown in Appendix B-1 is used to condition the sensor output so that it is within the acceptable voltage range for TTL-type logic. The left half of the circuit in Appendix B-1 is the circuit representation of what is inside the photopairs. The conditioning circuit works by comparing the voltage from the photopair to a reference voltage set by a potentiometer. If the photopair voltage is lower than the reference voltage, the output will be low, and vice versa. The arrangement of the photopairs on the underside of the robot can be seen in Appendix B-1.

6.3.1.2 Sonar Rangefinders

Since the market for sonar rangefinders is usually in the military, the choices for sonar rangefinders for a robot of this scale and size were rather limited. The Devantech SRF-02 sonar rangefinders were ideal for this project; all distance computation and tuning was done in-circuit, and it was all put into a small, easily handled package.

Each SRF-02 is equipped with a set of registers in which data can be stored and written to. For instance, to ping the rangefinders, a command is simply written to the command register telling it in what units the distance should be returned. The SRF-02 allows for inches, meters, and usec. The SRF-02 can also operate using the I2C protocol, which helps simply communication issues.

6.3.1.3 Keypad Input

Since the robot is to be used in a factory and travel to and from given stations, it needs some form of user input so that the robot will know its final destination. It was decided that the robot would use a simple USB numerical keypad for input. Alternatives such as a phone keypad connected using resistors on each line were considered, but the availability of a USB port on the controller made a USB keypad the simplest alternative.

Since the only characters available would be numbers and arithmetic operators, a sequence needed to be devised so that the robot software could know everything it needed to know to get from location to location. For instance, to devise a path, the robot would need to know its current position as well as its intended destination. Therefore, a sequence such "*01+08*" could be used to tell the robot to go from station 1 to station 8. The controller could interpret the directions using '*' and '+' as delimiters and extracting the relevant numbers from the string.

6.3.2 Controller

As the controller market is rather large and extensive (ranging from PICs to PCs), the choices needed to greatly narrowed down. One requirement was that the robot needed either memory to store preconfigured paths through the factory or enough processing power to compute paths on the fly. This requirement ruled out many of the simpler controllers, but left the option of using a microcomputer or PC open.

It was decided that a PC would be used, albeit a slow and semi-powerful PC. The easiest choice and most easily available option was the Efika made by Genesi. This decision was made for several reasons:

- **USB and Ethernet capabilities:** The USB Keypad and the (later mentioned) U2C-12 required USB ports. Other smaller, embedded controllers offer USB capabilities, but generally, USB is not a feature needed in situations that need embedded solutions (e.g. automotive control, non-hobbyist robotics, etc.). The Efika also had an Ethernet port which allowed users to access the system remotely via SSH. This eliminated the need for a large display and gave the robot the ability to be controlled from another room (or another country if need be).
- **Memory:** The Efika offered 128 MB of RAM, as well as the capability of adding an IDE harddrive to the system.This allowed large (or small) operating systems such as Linux to be installed on the system. A USB flash drive can also be used to increase the amount of memory on the system, but this isn't recommended due to the limited write life of flash drives.
- **Computational power**: The Efika offered a 400 MHz PowerPC processor. While slow by PC standards, it was more than enough power to run Linux and interface with the peripherals.
- **Power Consumption:** The Efika was made with the intention of being "power-phobic." Without a harddrive or graphics card, it consumes 5W of power, while *with* a harddrive and graphics cards, it consumes 12W of power. Since the robot is required to run on a battery, the Efika is ideal since it would not be a big tax on the battery's life.

6.3.2.1 Peripheral Interface

The Diolan U2C-12 was selected to be the interface between the Efika and its peripherals. The U2C-12 allowed for interfacing between General Purpose Input/Output (GPIO),

I^2C, and USB. It was the ideal choice since the IR sensors could be configured to work with GPIO, and the motor drivers and sonar sensors were I^2C capable. The manufacturer provided an extremely useful API that made communication between the three different protocols very simple. Code that makes use of the API can be seen in the appendices.

6.3.2.2 Logic

One benefit to using the Efika (as mentioned before) is the ability to run Linux, which adds a nice layer of abstraction between the user and the hardware. Therefore, a high-level language like C can be used to perform all needed logic and control functions. This makes satisfying the working criteria much easier. For instance, a while loop can be used to check each sensor (i.e. IR, sonar) periodically. Every time the loop iterates, the program can check to see if the correct IR sensors are still on the line or if there is an object within a certain distance using the sonar rangefinders. Motor speed and direction can be adjusted accordingly depending on thresholds set by the user.

For example, say that a line of 3 IR sensors is used to track the line. If the two leftmost sensors are off the line, the robot would need to perform a hard right turn to get back on track, and vice versa. Softer turns would be used if only one sensor was off the line. There would of course be sanity checks in case the robot encountered a fork or something of that nature (which would never happen in a grid-like factory). With the sonar rangefinders, a threshold distance could be set. If an object passed within this distance, the robot would slow down. This distance could vary with speed, since more time would be needed to slow down as the robot went faster.

6.3.3 Output

After all the inputs are processed, motor drivers will be needed to get the motors moving. Since the motors that were chosen have a stall current of 17 Amps, an equally powerful motor driver will be needed. The motor drivers used were Devantech MD03 motor drivers. They are capable of pushing 20 Amps continuously to a single motor, and are capable of using the I^2C protocol to communicate.

Due to budget and time constraints, it was decided that only two motor drivers would be used to control 4 motors. To solve this issue, a switching system would be needed in order to switch between adjacent motors (as seen in Appendix B-2). An American Zettler AZ8-1CT-6DE SPDT switching relay would be used to switch between motors. However, in order to energize the relay's coil, the U2C-12 needs to supply 75 mA of current. This is beyond the operating limits of the U2C-12 so a "buffer" circuit needs to be implemented. This circuit can be seen in Appendix B-3.

The user can directly control the speed, acceleration, and direction of the motors using the MD03. The only issue with the MD03 is that it has only one motor output per driver, so the user has to keep track of wheel and robot directions while programming the robot. Due to the fragility of the robot's cargo, jerky movement needs to be avoided. Therefore, wheel speeds will have to be varied during turns. For instance, in a smaller, simpler line following robot, one wheel might be stopped or thrown into reverse for soft and hard turns, respectively. However, doing that causes jerky motion so instead, for a soft turn, one wheel will remain at the same speed while the other slows down slightly. For a hard turn, one wheel would speed up from the normal forward speed and the other would slow down. This way, no part of the robot would ever stop moving, thus avoiding jerkiness.

6.3.4 Power Source

Since the motors draw such a large amount of current, a source capable of producing a large deal of current over a long period of time would be needed. It was decided that 6V deep cycle battery should be used. This was chosen over a 12V car battery for several reasons:

- **Motors:** The motors were rated to work between 3VDC and 9VDC, and the given specifications were measured using a 7.2VDC source.
- **Cranking vs. Sustained Draw:** Car batteries are made to source hundreds of amps in a very short amount of time and react adversely to being drained to a very low amount of charge. Deep cycle batteries on the other hand are made to source "small" amounts of current for long periods of time. Draining the battery to a very low point will not harm a deep cycle battery like it would a car battery.

Therefore, all the circuitry used had to be made to use 6V as a source instead of the normal 5V or 12V. However, the Efika requires 12V power source. It uses a miniature power source called the PicoPSU which can operate on a source anywhere between 6V and 26V. However, the downside to using the PicoPSU is that under 12V, the Efika's USB ports stop working. Thus, a smaller 12V battery with a shorter lifespan was found and used to power the Efika.

6.3.5 Warning System

While the sonar system actively helps to avoid objects, it would be best if people were warned before ever placing themselves in the path of the robot. In order to do this, a rotating amber light was selected to be mounted directly under the cassette holder. Although the light is rated for 12V, it is able to run at 6V, but as a result rotates more slowly and less brightly.

7 Testing

As the robot was comprised mostly of electrical systems and the mechanical aspects were designed to handle stresses in excess of what will be encountered during operation, the testing done on the robot was mostly for the electrical systems.

7.1 Battery

A 6V deep cycle battery was salvaged from a Physical Plant golf cart. It was charged using the machine available at Physical Plant then brought to the lab for testing. Since the battery's main purpose was to drive the motors, the battery's ability to source a decent amount of current needed to be verified. This was done by connecting two of the motors in parallel directly to the battery through two DMMs also in parallel. This was done since the DMM's ammeter is fused at 10 Amps. When the circuit was closed, it was observed that the combined no load current draw of the motors was on average 2.3 Amps. Since the greatest load on the battery will occur when the motors are stalled, the next thing to test was the stall current. This was done by letting the motors run at no load, then suddenly grabbing hold of both wheels, thus stalling the motors. The battery was able to source 17 Amps at maximum. The manufacturer, Trojan Batteries, lists the battery's capacity at given loads—up to 75 Amps—in the following table:

Table 1: Battery Specifications (3)

Type	Capacity Minutes			Cranking Performance		5 Hr Rate AH	20 Hr Rate AH	Voltage	Terminal	Dimensions inches (mm)			Weight lbs. (kg)
	@25 Amps	@56 Amps	@75 Amps	CCA @0°F	CA @32°F					L	W	H	
T-105 Plus	447	-	115	-	-	185	225	6	AP, ELPT, EHPT	10 3/8 (264)	7 1/8 (181)	10 11/16 (272)	62 (28)

While the maximum current that the battery could source was not physically verified, it can be assumed that given the correct gauge wire and proper fusing, the battery would act as an acceptable source.

7.2 Input/Output

The rest of the testing dealt with all of the input and output functions of the robot, such as the IR and Sonar sensors, as well as the motor drivers. All I/O was taken care of using the U2C-12 which allowed the use of General Purpose Input/Output (GPIO) and Inter-Integrated Circuit (I^2C) protocols.

7.2.1 GPIO

The GPIO ports were capable of being used as both inputs and outputs. However, for this project, they were used only as inputs, since the relay switching system was never implemented. This is not to say that the switching system would have failed to work, but that given more time and dedication, it could be easily be implemented. As a result, testing was done to determine the speed at which the GPIO could collect data, and the IR Sensor circuit was tested to make sure that its output fell within an acceptable voltage range.

7.2.1.1 Speed and Functionality

The testing of the GPIO protocol on the U2C-12 was done by a C program, also with the help of the U2C-12 API. The GPIO protocol uses 40 pins, 24 of which are bi-directional ports that can be set individually as either input or output using the API. The purpose of testing the GPIO was to see if 1) the output pins could drive a relay which would switch power between two motors that share a single motor driver, and 2) if the software could accurately measure the frequency of a square wave, much like a set of binary rotary encoders would produce.

The first test was simply a test of the API, in which an LED circuit was used to see if the GPIO could drive it within safe current levels—that is, the test was done to see if an LED could be powered without burning up the U2C-12. The first thing that had to be done was to set the direction of a given pin to output. The pin chosen was number 3, which corresponds to PA0 (Port A0). The API gives the user a large amount of control when dealing with setting pin direction and reading from and writing to pins. Pin direction can be set individually or independently. Reading or writing data to pins can be done one-by-one or in a batch-type style. To get a feel for both styles of dealing with GPIO, the pin direction was set with the batch-style method while pin writing was done on a pin-by-pin basis. The resulting program could also be used to turn a relay on via some kind of power transistor.

The second test was a more functional test that would determine the “breaking point” of the EFIKA kernel in terms of measuring the frequency of a square wave. This test was performed by setting pin 5, which corresponds to PA2, to be an input. After that, the program entered a while loop to measure the frequency. The loop began by recording the previous value of the pin and then reading the new value of the pin. It would then compare the two values to see

if they had changed, and upon success, it would check to see if it had encountered a rising edge or a falling edge. In this case, the frequency would be measured based on the time from one rising edge to another. The frequency was computed by comparing the timestamps of consecutive rising edges, determining the wave's period, and then easily calculating its frequency.

However, as a result of the second test, it was decided that using encoders would be difficult to implement given the current hardware and OS. It was then decided that a hash-mark type system would be used instead. With the hash-mark system, the robot would count hash marks as it went along the line, and a map in memory would let the logic program know when to slow down, speed up, and when to turn.

7.2.1.2 Infrared Sensors

Since the voltage source being used for the IR sensors (~6.2V) was higher than the maximum input rating on the U2C-12 (5.5V), the output needed to be conditioned. This was done using the circuit in Appendix B-1. The circuit was tested by moving one of the photopairs from a black surface to a white surface, and then back again. A voltmeter was hooked up to the comparator's output to measure the output. It was observed that on a dark surface, the output would be around 200mV. On a white surface, the output would be around 4.3V, which was higher than the "high" threshold (~3.3V) on the U2C-12. Therefore, it was verified that the circuit worked properly, and given the right surface and line color, the sensors would be able to see the line.

7.2.2 I^2C

The I^2C bus greatly simplified communication with and control of the sonar sensors and motor drivers. Each device connected to the bus is uniquely identified by a "slave" address, so each one can be communicated with independently. Diolan provided an API which allowed easy communication between the Efika and the I^2C devices.

7.2.2.1 Sonar Sensors

The sonar sensors needed to be tested for accuracy. Therefore, a test was devised where the sonar sensor would measure the distance to a large, stationary object (e.g. a chalkboard).

Testing for the sonars was done using a C program using the Diolan API. The C program tells each sensor to "ping," and then reads the calculated output from a register on the chip. A C structure will then be used to send the necessary data to the sonar controller which starts the pinging sequence. The structure would then be followed by a write command which would send the data to the U2C-12 and then perform the proper operations on the I2C bus. The "write" structure consists of five different parts: the Slave Device Address, which for this specific sonar is 0x70 due to the U2C using 8-bit addressing; the Memory Address and its length, which tells the U2C where to write the data contained in the Buffer. In this case, the data 0x50 is being sent to the command register at 0x00, which tells the sonar to report the distance back in inches. After sending the write command to the sonar, the user would wait 70 microseconds before reading the distance register so the data could be processed. In the test code used, enough is going on between the read and write calls that the process does not have to pause to allow for the 70

microsecond delay. If that were not the case, a call to usleep(70) would allow for sufficient processing time.

To read the distance register, a structure similar to the write structure would be used. This structure is followed by a read command which reads the data contained at the memory address 0x03 to the buffer. The memory address 0x03 in this case corresponds to the low bit of the distance calculated by the sonar. In the test code, this was done to each of the four sonars at the same time, which reside on the addresses 0x70, 0x71, 0x72, and 0x73. The decision to only use 4 sonars was based on the range and width of the sonar signal, the speed of the robot, and the cost of each. After confirming that each sonar worked, calibration of the sensors had to be performed. Even though the sonars are supposed to be self-tuning and self-calibrating, they may still have built-in error and that needed to be checked. In this case, the sonar was set in the middle of the Electronics lab, while the chalkboard was moved away a certain distance which was measured with a tape measure. The actual value was then compared with the value the sonar was returning to determine error.

It was determined that the measurement error increased as the object moved farther away. However, this was not viewed as a bad thing since the distance is being underestimated. Since the computed distance was closer than the real distance, the robot would begin to slow down earlier than it should. Therefore, the chance of the robot running into something while running at a “high” speed would be very low.

7.2.2.2 Motor Drivers

The motor drivers came with an easy-to-use interface. In the I^2C mode, there were registers that controlled speed, acceleration, and direction. A test program was written which allowed the user to control a single motor by inputting speed, acceleration, and direction. It used the same "write" data structure as the sonars, and did not require any "read" processes since the motors were being controlled via open-loop control. The program successfully controlled the motors and by using an ammeter, it was observed that each motor drew approximately 1.3 Amps under no load.

After the initial program was written, a second program which allowed control of the motors via emulation of the IR senor inputs was written. Switches connected to 5V were input to the U2C-12 to act as the IR sensors. As the switches were flipped in the correct combinations, it could be seen that the logic in the program was working, and that given powerful enough motors and working sensor array, the robot would move.

7.3 Full Scale Testing

After construction was completed, the robot was ready for full scale testing. However, testing was cut short after the first test was performed. The first full scale test was to get the robot to follow a line of tape on the floor. At first, the robot failed to move, so it was assumed that there might be something wrong with the logic or with the sensor array, or that something might be wrong with the motors and motor drivers. In order to save debugging time, the initial motor driver program was modified to use 2 motor drivers and two motors. This program was

unable to move the robot. The first guess was that the robot was too heavy to move. The battery was removed and the program was run again. At first the robot did not move, but then the robot was given a push by one of the group members and the robot began to move for a little bit. It was determined that the motors were not providing enough torque to move the robot, and that bigger, more powerful motors would be needed to move the robot. This was the end of full scale testing as the project time had run out and the system could not be tested without the robot moving on its own.

8 Conclusions

The final design met most of the objectives that were proposed at the onset of this project. These constraint and working criteria became the driving force that pushed the project along and became crucial components of the decision making process. Throughout the project, changes were made in order to stay within the constraints of the project and still be able to meet the working criteria specified. However, not all the criteria was met and the analysis of the reasons why this was the case is as important as highlighting the successful aspects of the project.

The first constraint that was considered was clean room compatibility since the fabrication plant requires that all the components of the robot be clean room compatible. This affected two primary component decisions in the design: the frame and the Plexiglas. A carbon steel frame was used because it is relatively easy to obtain, it was donated to the project and it does not emit particles. The use of Plexiglas is very common in semiconductor fabrication plants and it was incorporated throughout the design. Like the carbon steel frame, Plexiglas is easy to

find and it does not emit particles. The downside is that is not cheap and thus increased the budget significantly. Plexiglas was used in the construction of a lot of the structural components including the infrared sensor holder, electronics holder, 12 V battery slot, wafer cassette holder, side paneling, hatches and top and bottom bases. The clean room compatibility of the wheels was another important consideration, polypropylene was selected as the primary polymer used in the rollers since it is a harder plastic and would emit fewer particles than the polyurethane alternative. Even though these wheels are clean room compatible for the scope of this project, there is still a possibility that small particles could be emitted from the constant use. Incorporating wheels that are specifically designed for clean room compatibility would take up the whole budget and was therefore beyond this project.

Another constraint that is determined by the working environment was the loading height. OSHA suggests that all the equipment that require something to be loaded onto it or unloaded from it be a minimum of 900mm. This constraint is implemented to prevent injuries to the workers that must continuously load or unload materials from equipment. In order to meet that constraint, final height of 1.2m was chosen.

It was imperative that the wafers be protected during their transportation. This constraint was incorporated into the design in two ways. The orientation of the cassette holder is such that it allows the robot to translate in all four directions while reducing the rattling of the wafers inside the cassette. The robot was also designed to have a very low center of gravity with the battery, which is roughly half of the weight of the robot, located at the lowest point. Even though a system is incorporated to identify and avoid obstacles, measurements were taken in order to assure that the robot is robust enough to withstand a collision in the plant and ultimately protect the wafers. The Omni-directional wheels prevent the robot from tipping or tilting if it

does experience a collision. Since the wheels are omni-directional, the friction acting on the side of the wheel results in free translation as oppose to creating a moment that could cause it to topple. These measures were taken to make sure that the wafers are protected during transportation and in case of a collision with an obstacle.

Older fabrication plants are very limited in terms of the width of the bays where the semiconductor materials must be transported. Taking this into consideration, it was important to make the robot as narrow as possible to prevent it from impeding traffic within the bays. The width of the frame is 18 inches and when the wheels are taken into account, the total width of the robot becomes 2 feet. These parameters allow the robot to maneuver around the narrow bays preventing inconvenience with the operations taking place in the factory. It is designed to take up less room than the workers it is being built to substitute.

Since there is a $1,000 budget for this project, cost is another constraint. The total cost of this project was $963 (not considering donated items). Even though this constraint was met, it had to be balanced against other constraints that prevented the project from meeting all the working criteria set out to accomplish. Programming, coding and circuit design was another constraint that had to be taken into consideration since there was only one electrical engineer on the project. The programming was not completely finished due to a lack of time and personnel. The difficulty in programming a translating, line-following robot was underestimated at the beginning of the project. Second, the budget constraints prevented extra motor drivers from being purchased.

Finally, two criteria that were also considered at the onset of the project and eventually met were turning ability and using a reusable source of energy. The turning ability criterion

requires that the robot have good maneuverability. This includes making a wide variety of turns, change in speed and change in direction. Good maneuverability will simplify the line tracking which will allow the robot to stay out of the way of workers and be less of a nuisance in the factory. This criterion was satisfied through the translating design and the dual motor drive system which allows for minor adjustments to keep the robot on the line. The fact that the robot can translate and is able to change directions on the line requires that a single line be run through the main hallways of the fabrication plant with single lines stretching out into the bays. The robot's maneuverability allows it to operate with a simple track seen in Fig 6 below.

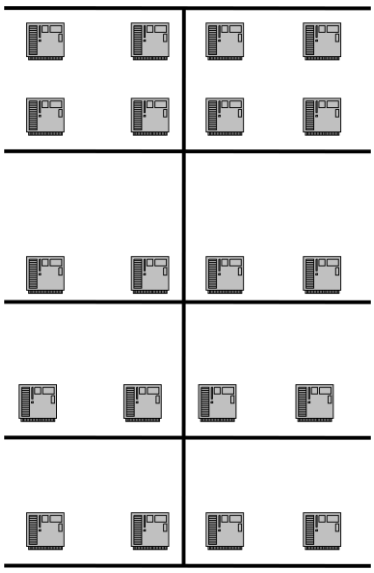


Figure 6: Line Track Required

The second of these criteria requires that the robot be powered by a reusable source of energy. This was met by incorporating a deep cycle 6V rechargeable battery into the design. Not only is this battery rechargeable, but it is also easily accessed. Hatches were implemented

into the design to allow access to the battery terminals to be recharged while still protecting the electrical components during transportation.

This project met most of the design constraints shown in Fig 1. The one constraint that was not fully met was the programming. The initial idea was to program the robot to follow a ling and turn, but the translating aspect of the design did not come to fruition. This had to do with the limited personnel working on the electrical and software part of the design and construction. After analyzing the constraints and their corresponding weights as seen in Fig 1, this project accomplished about 91% of the constraints.

Even though most of the constraints were met, the moving criterion was not accomplished and this was due to a couple of reasons. Throughout the project, three tight limitations had to be balanced continuously. These restrictions were time, budget and personnel. In the early stages of the project, proposed design ideas were weighed according to the constraints, working criteria and other considerations. One of these considerations was the fact that the project was limited to only one electrical engineer to do all of the programming. However, innovation was another consideration and an innovative design would deliver a product not common with line following robots in a university setting. The analysis concluded that a translating robot design met the constraints, criteria and other considerations best. Once the design was selected, another important decision had to be made. From an electronics standpoint, the design could be incorporated using either two or four motor drivers. Using four motor drivers would facilitate the implementation of the software; however, it was impossible to incorporate four motor drivers into the budget allotted for this project. A decision was made to use two motor drivers and incorporate a relay system. This way, the project could still be finished within the budget constraint even though the programming aspect would be more

difficult. This was a situation where the time, budget and personnel limitations had to be balanced and a decision made. The decision was also taken with the reassurance that if the translating aspect of the programming could not be incorporated, the design of the robot could facilitate turning if needed. This aspect of the design turned out to be beneficial since this was what ultimately occurred. The programming aspect of this project was finished to allow the robot to turn, while the translation portion of the programming could not be incorporated due to time. In retrospect, the difficulty in programming a translating robot might have been underestimated taking into consideration the budget, time and personnel constraints.

Regardless, the programming would have still allowed the robot to meet the working criteria if it was not for a problem that was encountered with the motors. The mechanical analysis of the design concluded that each motor needed to deliver about 1.1 Nm of torque (incorporating a 1.4 safety factor) in order for the robot to meet the working criteria. The specifications of the motors purchased stated that they could deliver a maximum torque of 1.68 Nm. The stall current for these motors was specified to be 17 A, and the relationship between current supplied and torque deliver was given to be 0.1 Nm/A, which supports the maximum torque and stall current value. However, in the final stage of the project, the motors were put on the robot and the realization was made that they were not delivering enough power. Further investigation concluded that the torque being supplied was half of the maximum torque in the specification of the motors. As explained in the testing section, the actual stall current was about 8 A instead of the 17 A specified and the actual torque being deliver was about 0.8 Nm instead of the 1.68 Nm specified. As was also concluded by tests, the battery was able to deliver more than 8 A to the motor and therefore was not a limiting factor. Hence, the working criteria were not

met because the actual performance torque of the motors was 50% less than the torque in the specifications.

9 Recommendations

While this project met most of the constraints, it also resulted in several valuable lessons stemming from working with tight limitations. A robot project of this scale requires would be much easier to complete with two more than one electrical engineer on staff. The vast amount of programming and debugging that is encountered is a daunting task for one person to undertake. The mechanical aspect of this project was ready for the final test in late February; however, the personnel constraint delayed the final testing for a month and a half and ultimately led to not meeting the moving criteria.

Another lesson learned was that all purchased items should be tested in a manner consistent with their application as soon as they arrive from the manufacturer. If testing in this manner had been performed, the problem with the motors would have been caught and action could have been taken to get the correct motors needed for the project. It was expected that the motor would not perform to specifications, and that was taken into account in the design of the robot. The torque that the design required was 1.1 Nm which is considerably less than the 1.68Nm that the motor's specifications claimed to deliver. More than 50% decreased performance was not foreseen.

If more funding were provided to the project, two more motor drivers would have been purchased and the idea of using relays would have been removed. This would have simplified the programming process and allowed easier implementation of the translating aspect of the design. To make the robot function properly, the correct motors must be purchased. The motors

needed must have either a gear ratio higher than 25:1 or the motor itself must be capable of providing the right amount of torque. These two courses of action would deliver a translating semiconductor factory robot that meets all the constraints and working criteria.

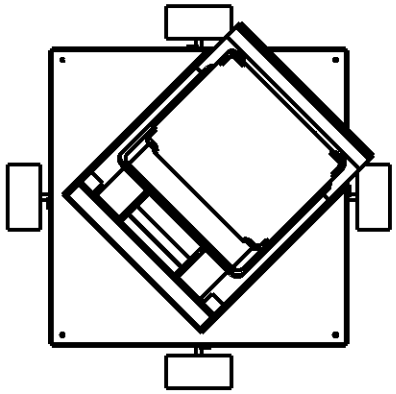
10 Bibliography

[1] U.S. Department of Labor. *Ergonomic eTool: Solutions for Electrical Contractors.* October 10, 2007. http://www.osha.gov (accessed April 28, 2008)

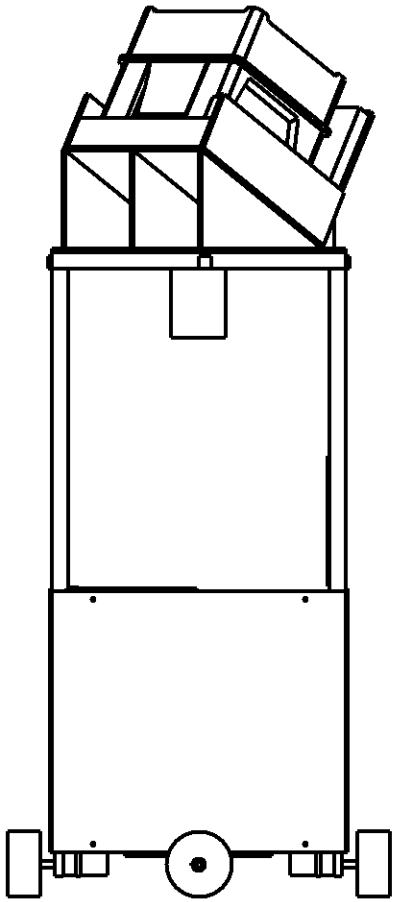
[2] Gere, James M. *Mechanics of Materials.* Thomson-Engineering, 2003.

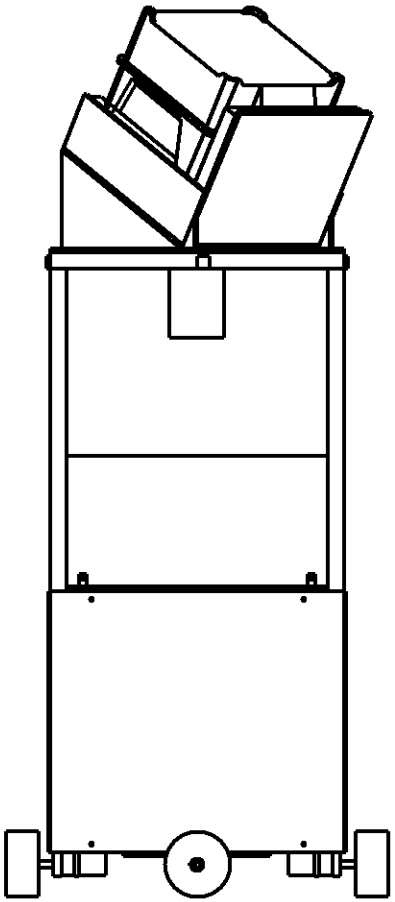
[3] Kornylak Corporation. *Transwheel Multidirectional, Powered Conveyor Wheel*. 2008. http://www.kornylak.com (accessed April 12, 2008)

[4] Trojan Battery Company. *Trojan Battery Company.* 2007. http://www.trojan-battery.com

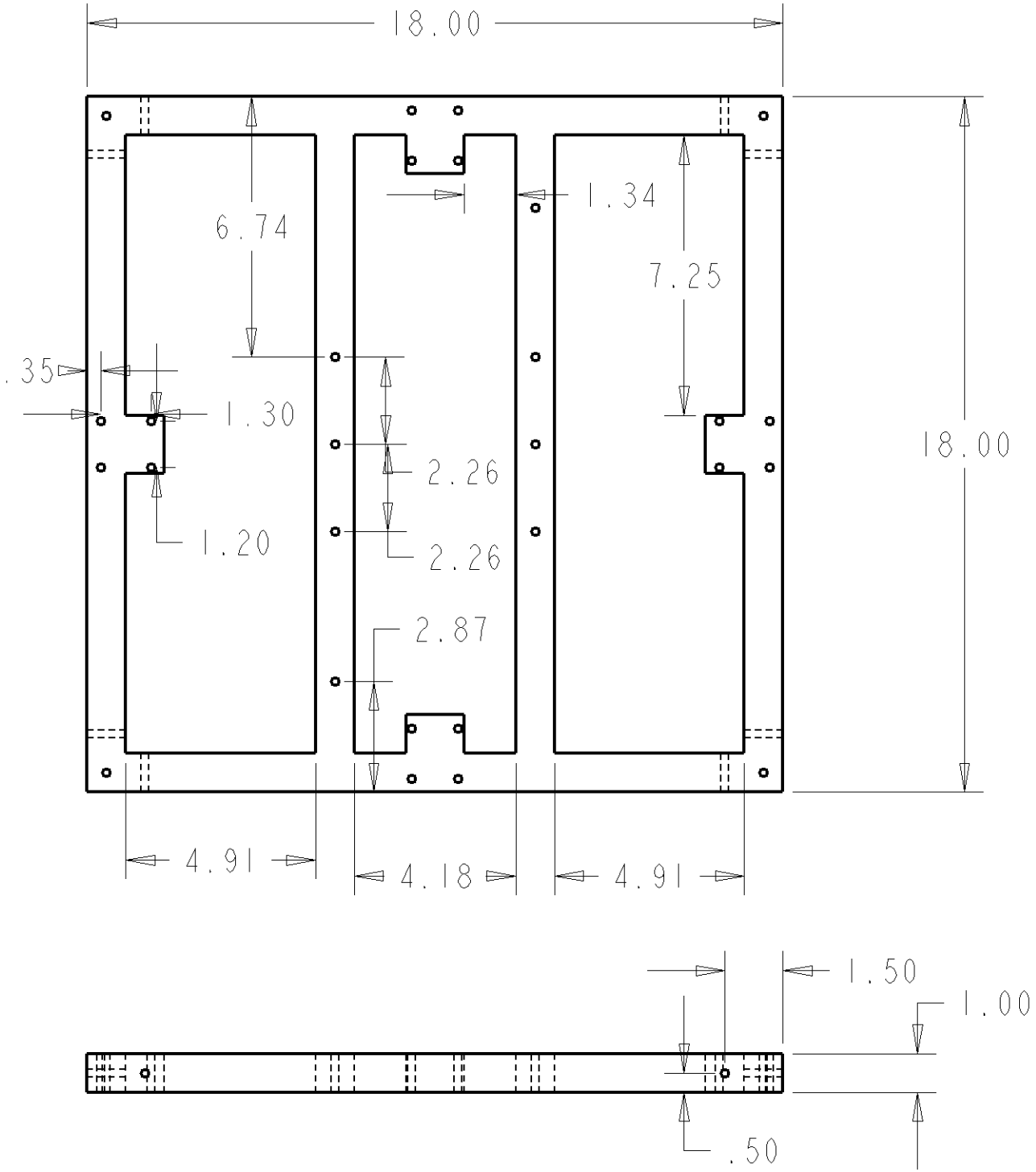


Mechanical Design Drawings
Factory Robot Group





A-1

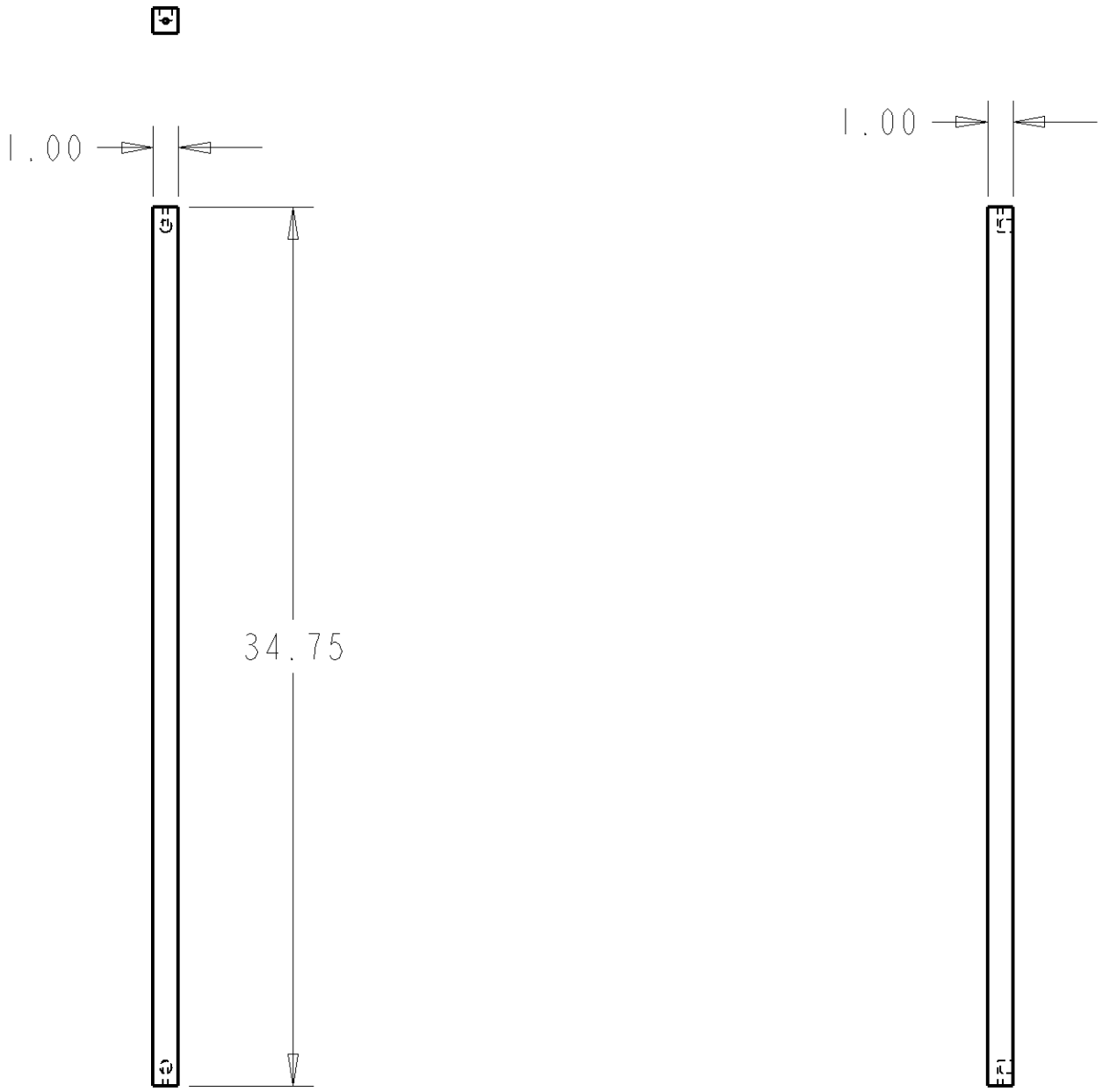


Base Chassis
Factory Robot Group

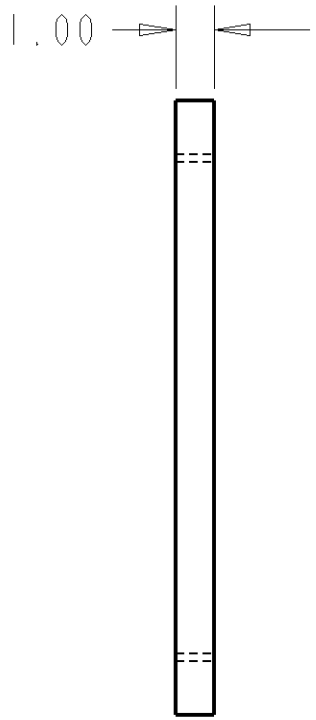
All Holes 10-32

A-2

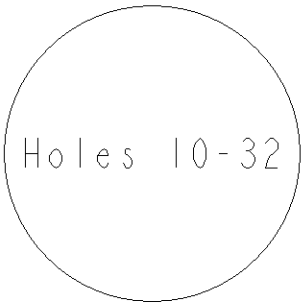
Frame Verticle Support
Factory Robot Group

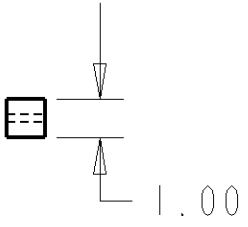


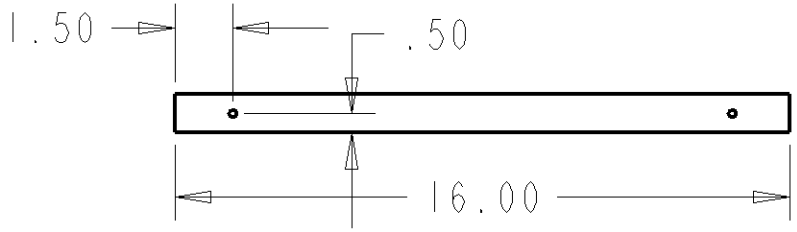
A-3



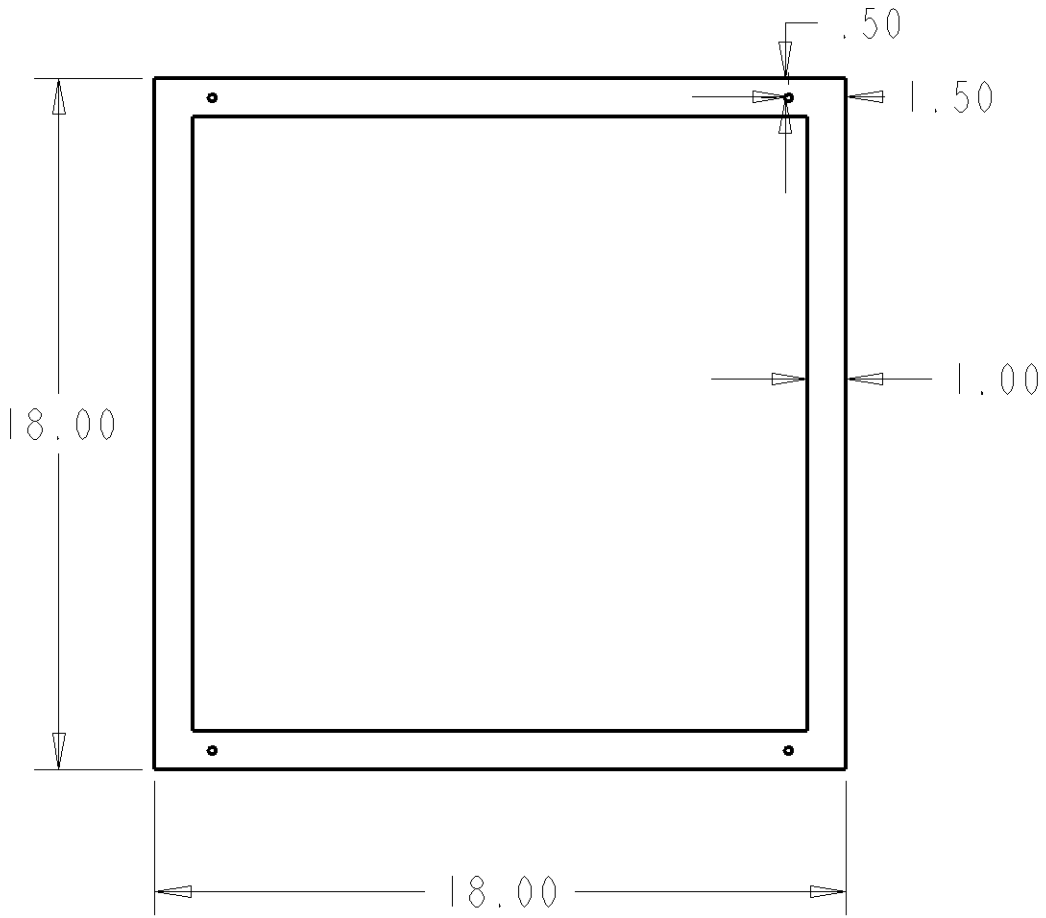
Side Support Beam
Factory Robot Group



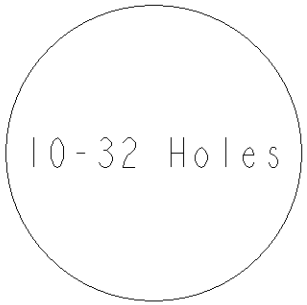


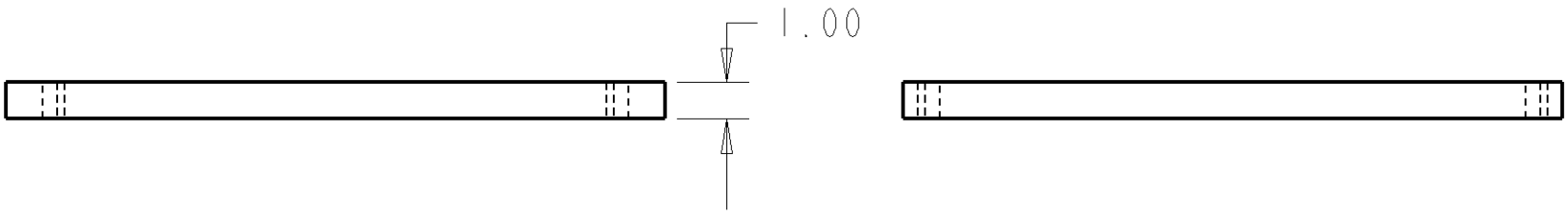


A-4

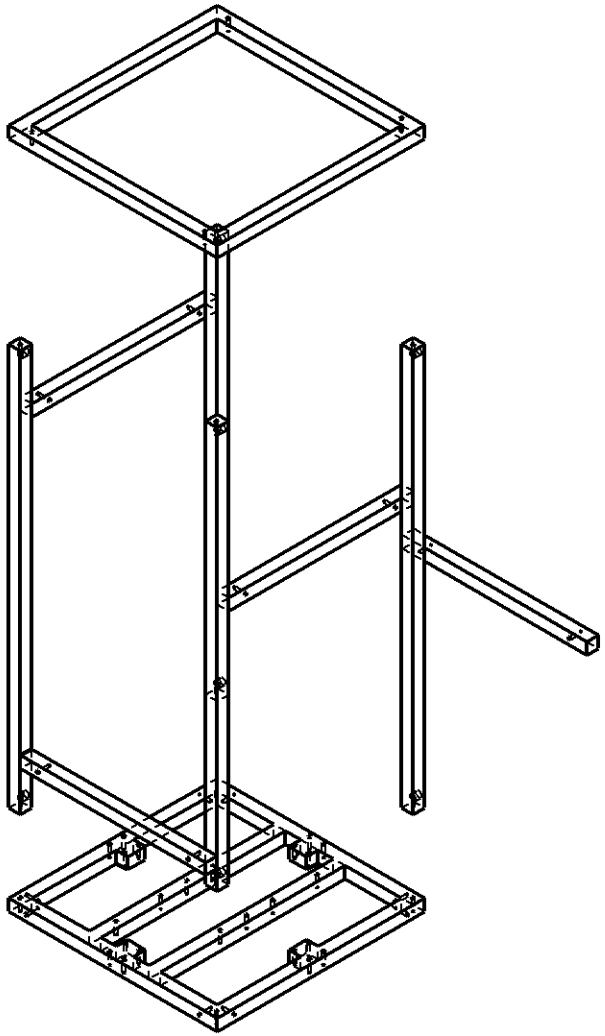


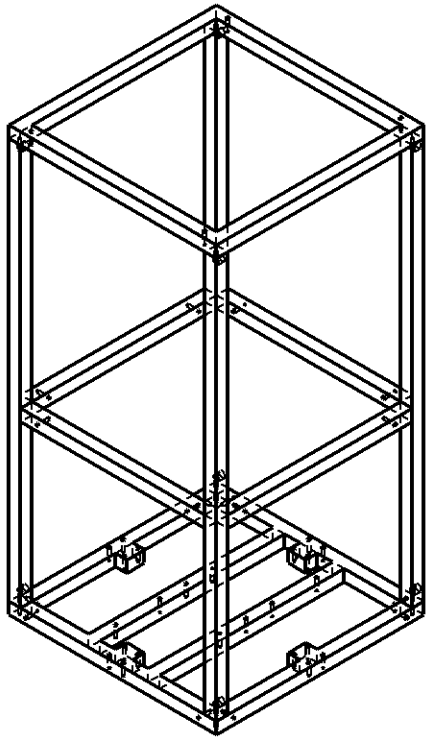
Frame Top
Factory Robot Group

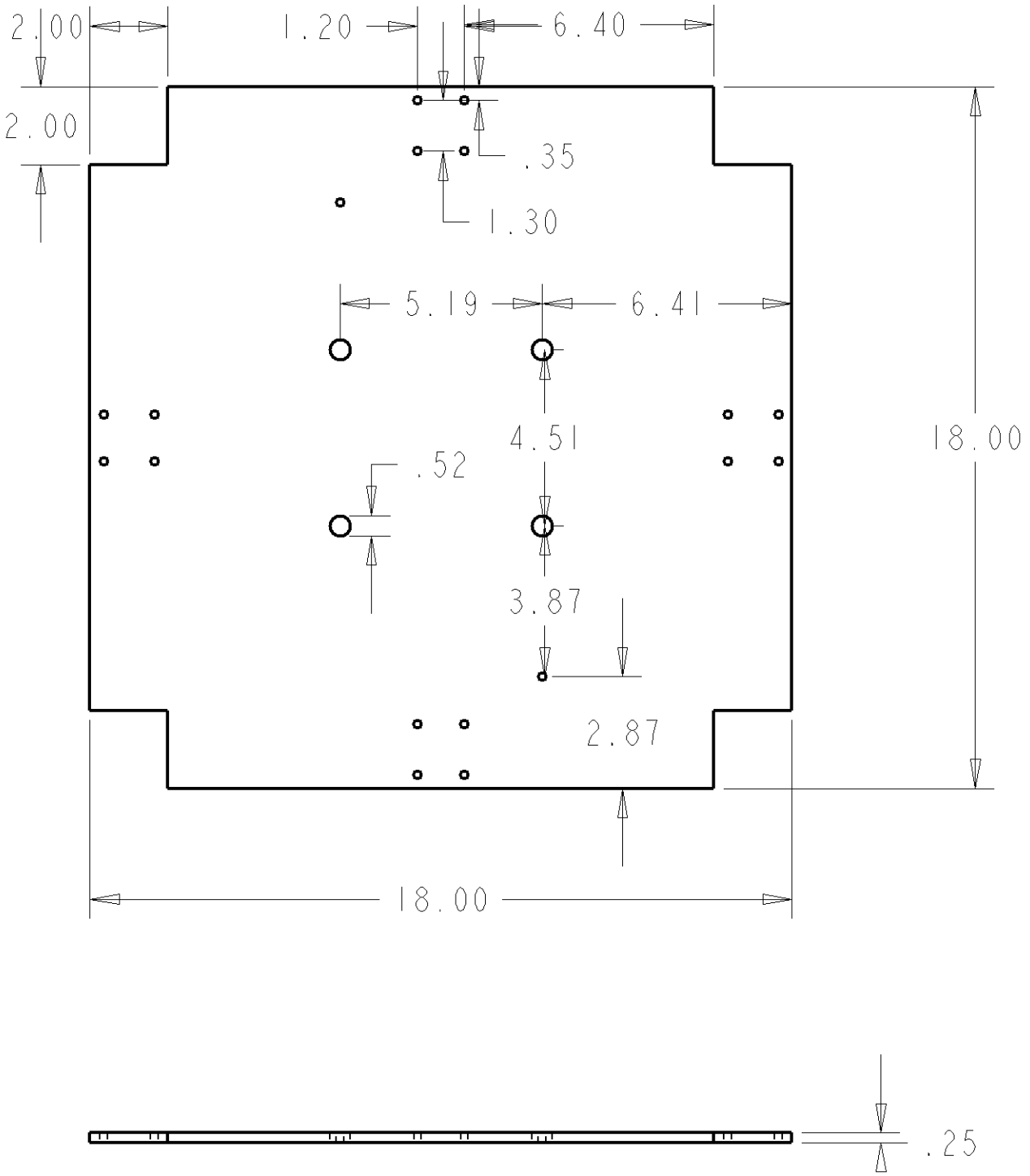




Complete Frame Assem
Factory Robot Group





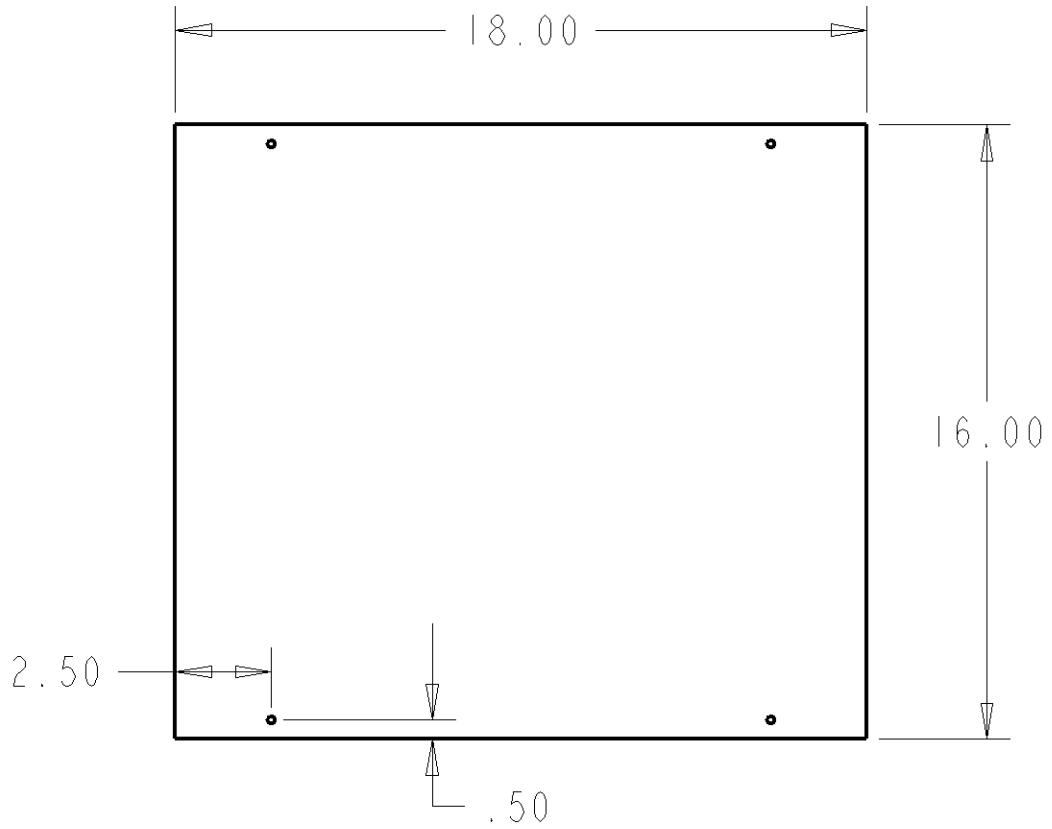


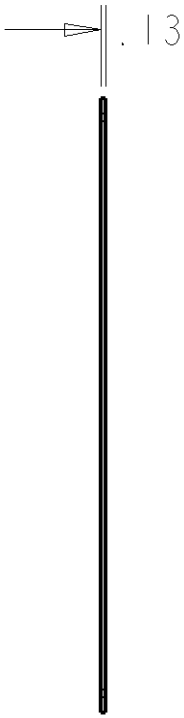
Plexi-Glass Base
Factory Robot Group

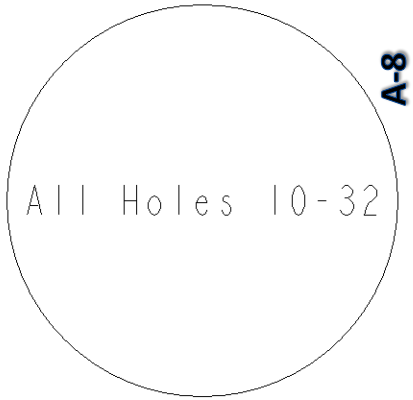
A-7

Small Holes 10-32

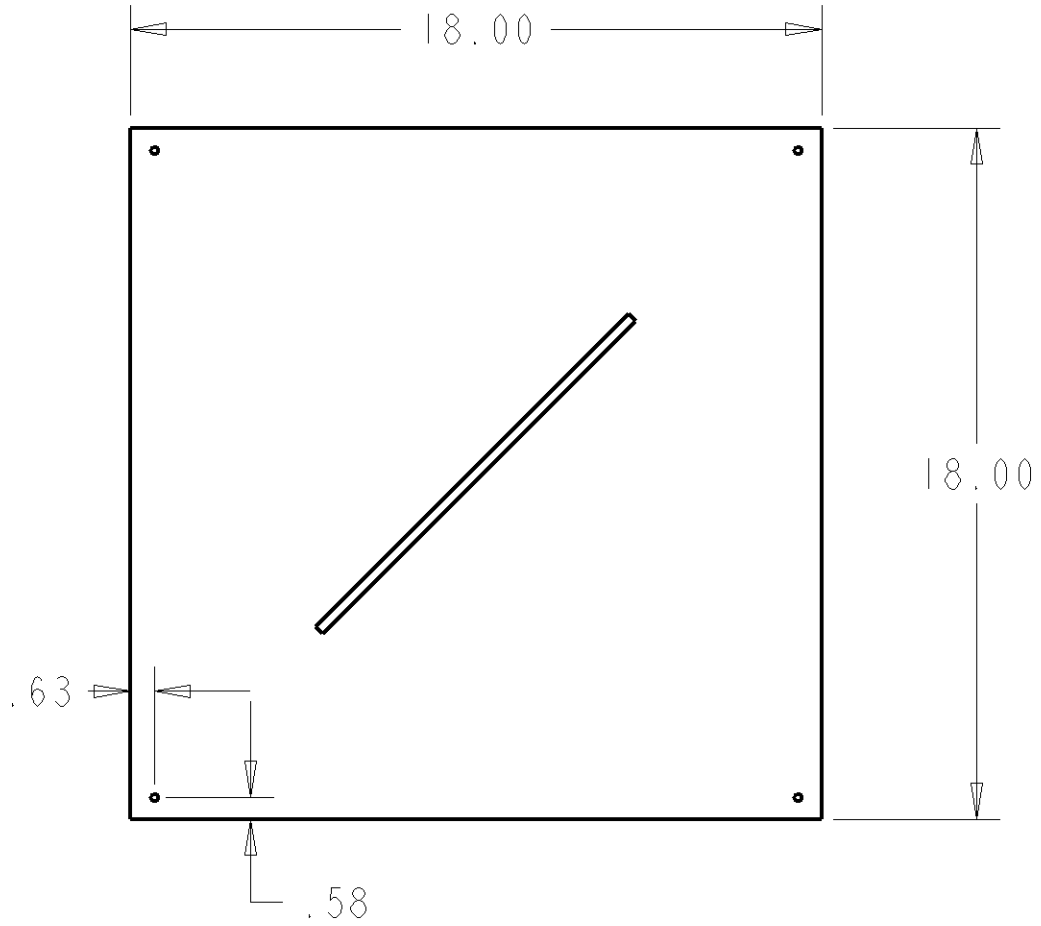
Side Plexi Panel
Factory Robot Group



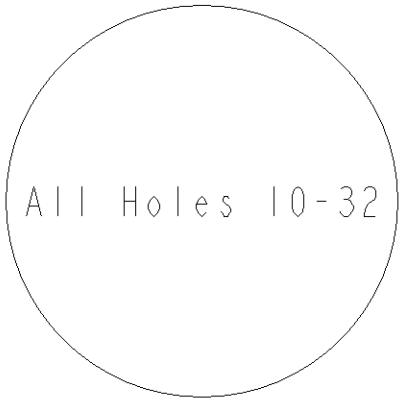


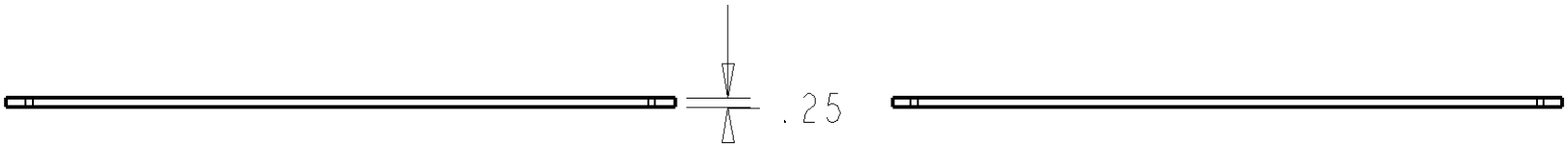


A-8

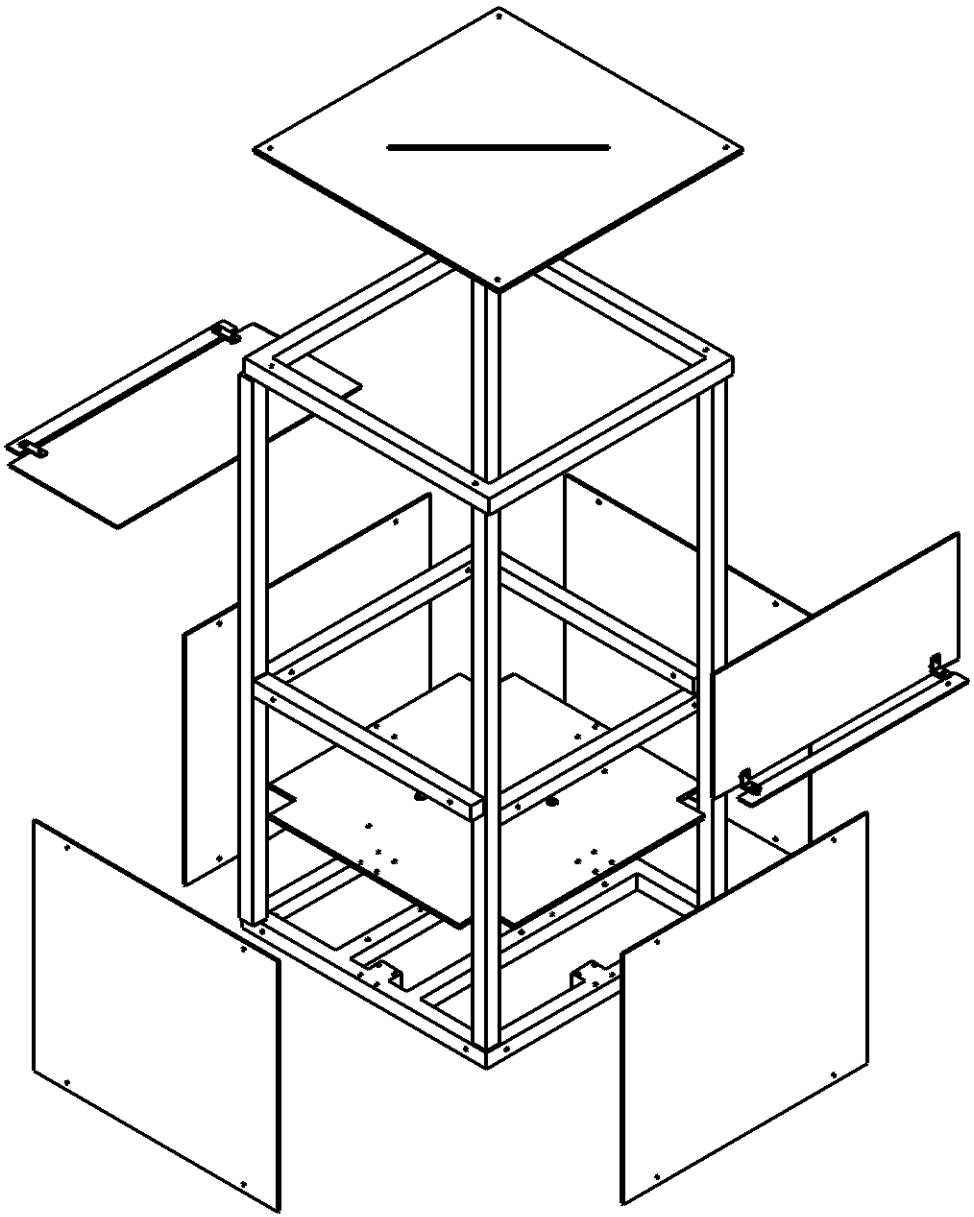


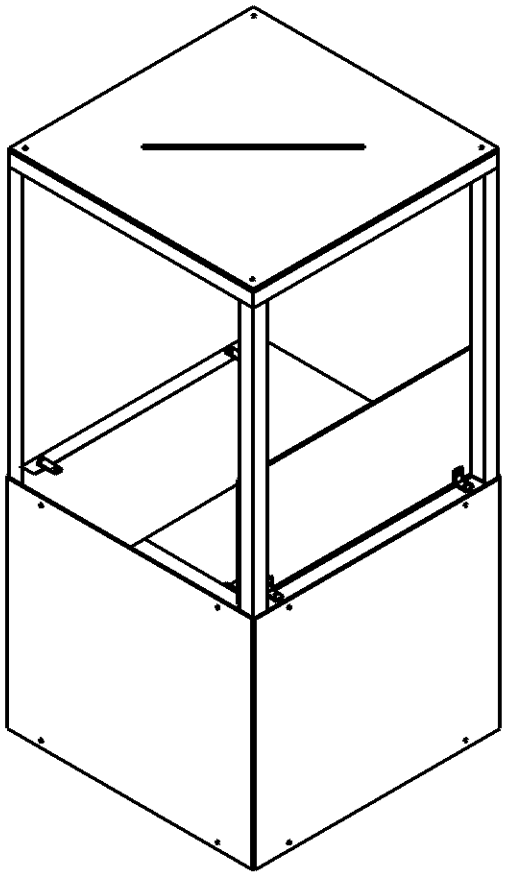
Top Plexi Base
Factory Robot Group



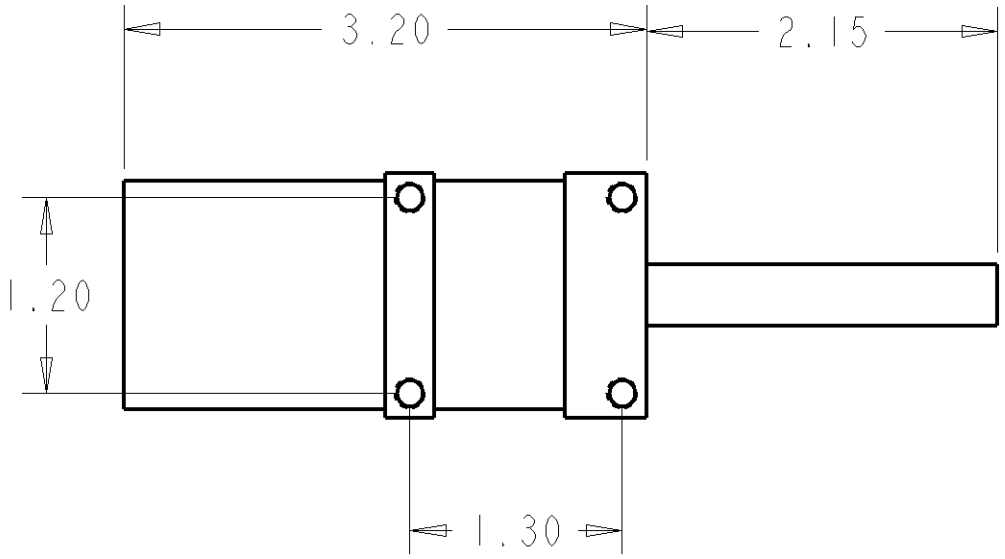


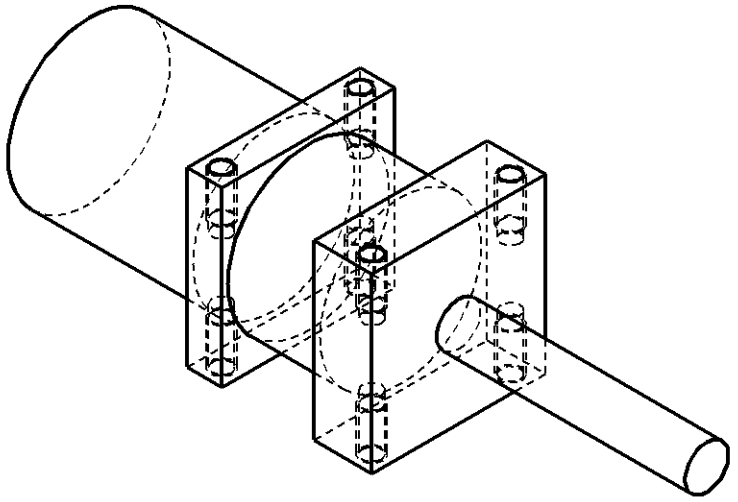
Plexiglas and Frame
Factory Robot Group

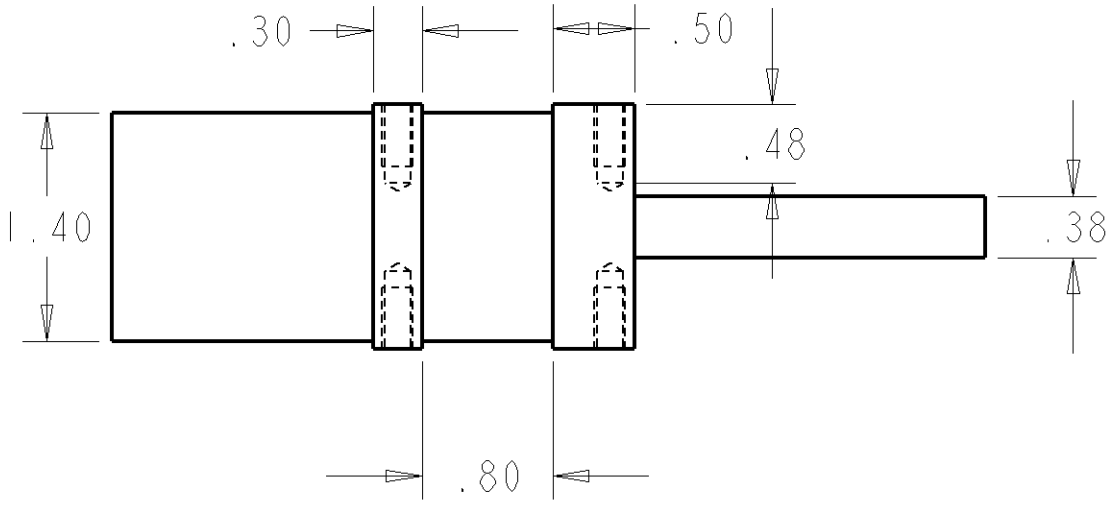


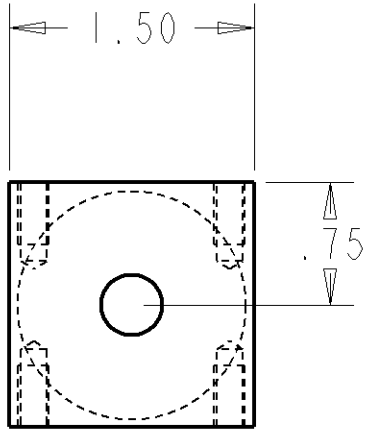


DC Motor
Factory Robot Group



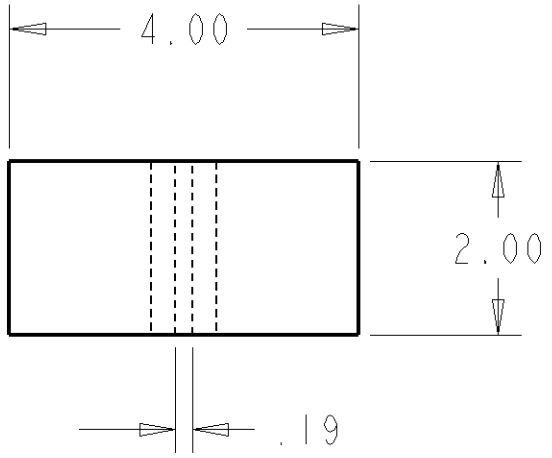


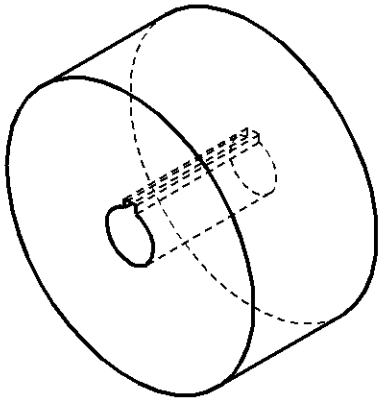


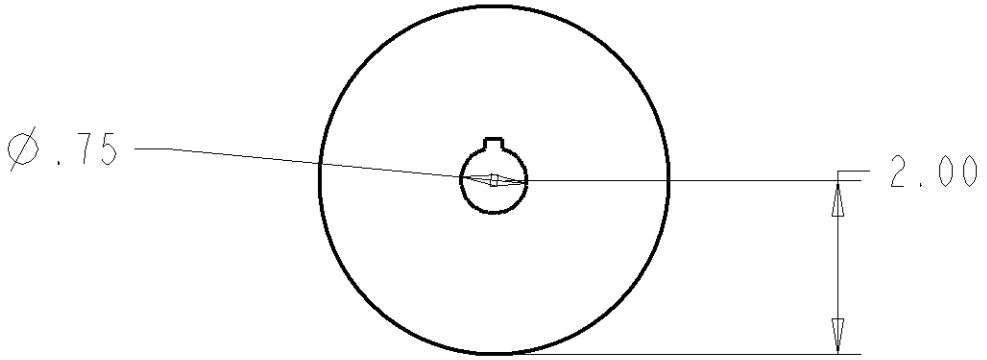


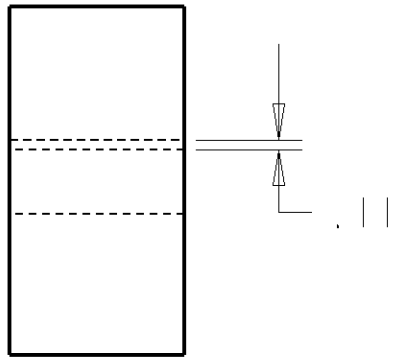
A-11

Wheel
Factory Robot Group



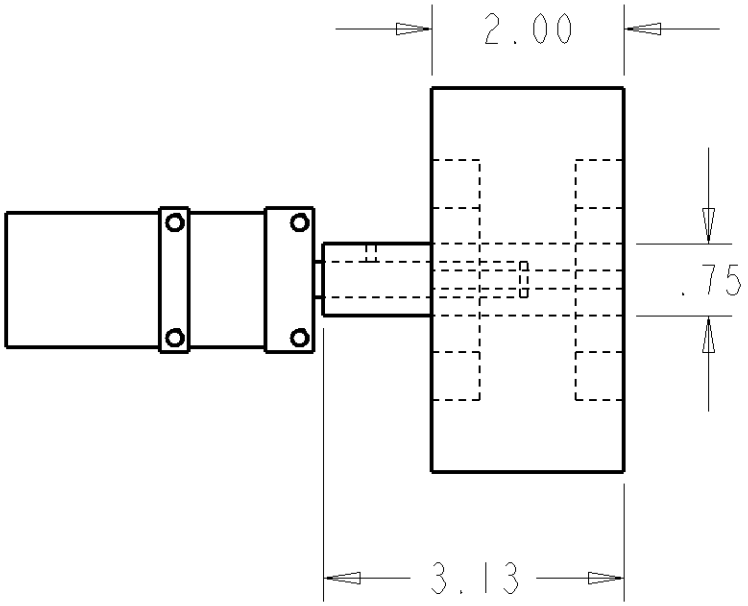


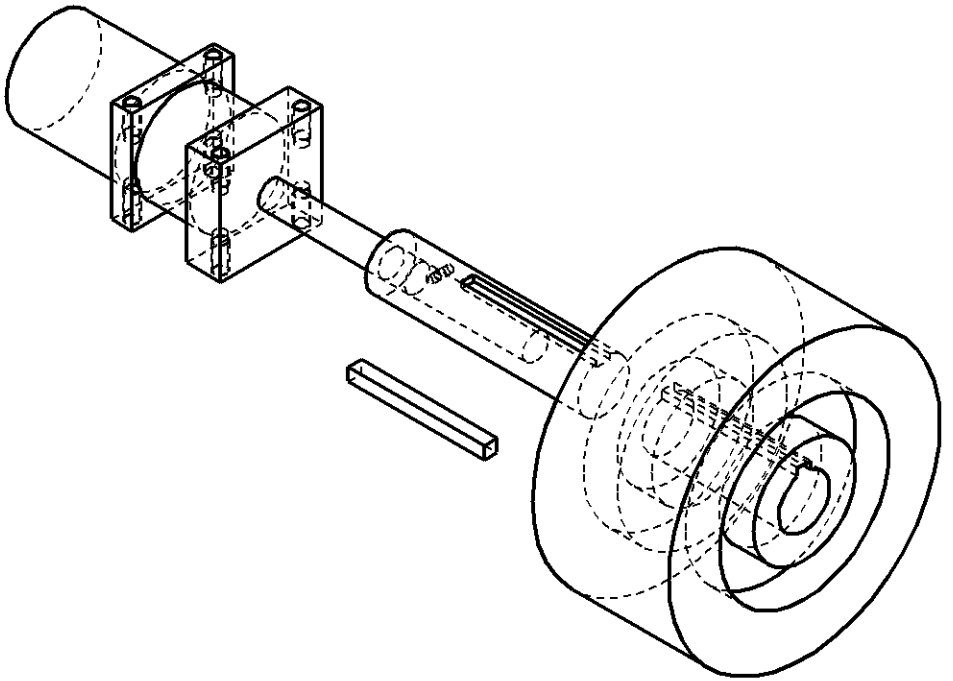


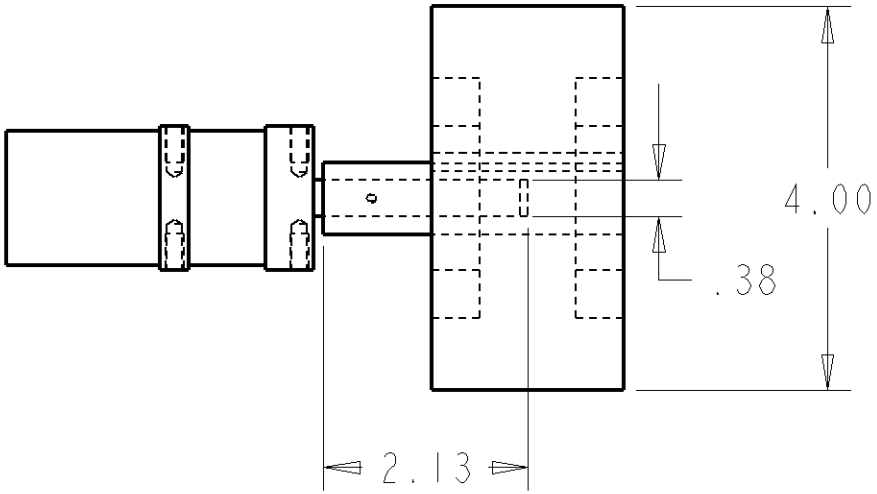


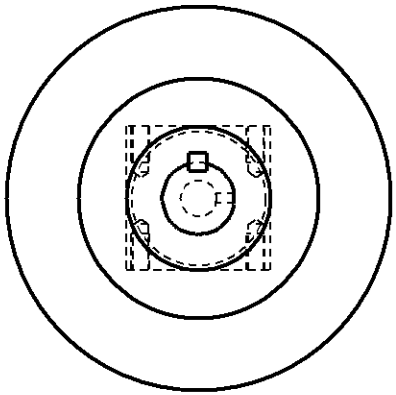
A-12

Wheel Assem
Factory Robot Group





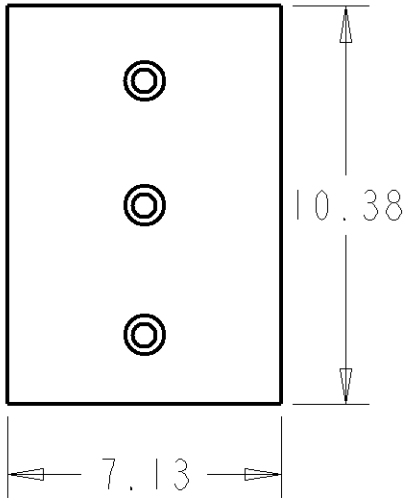




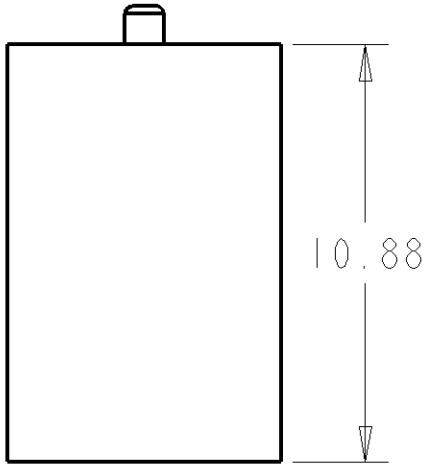
A-13

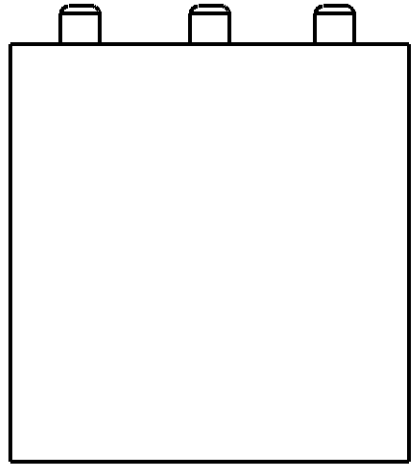
Key is 3/16 inch square

Aluminum sleeve is attached using a small screw

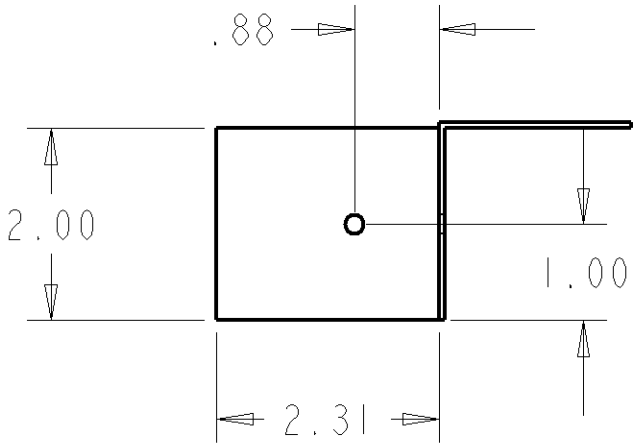


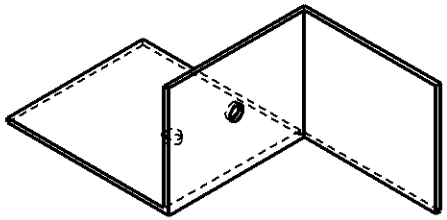
6 Volt Battery
Factory Robot Group



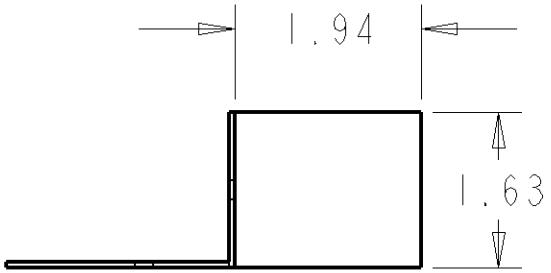


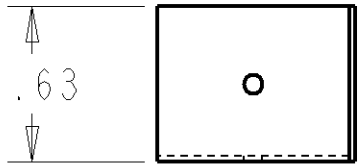
Large Battery Holder
Factory Robot Group





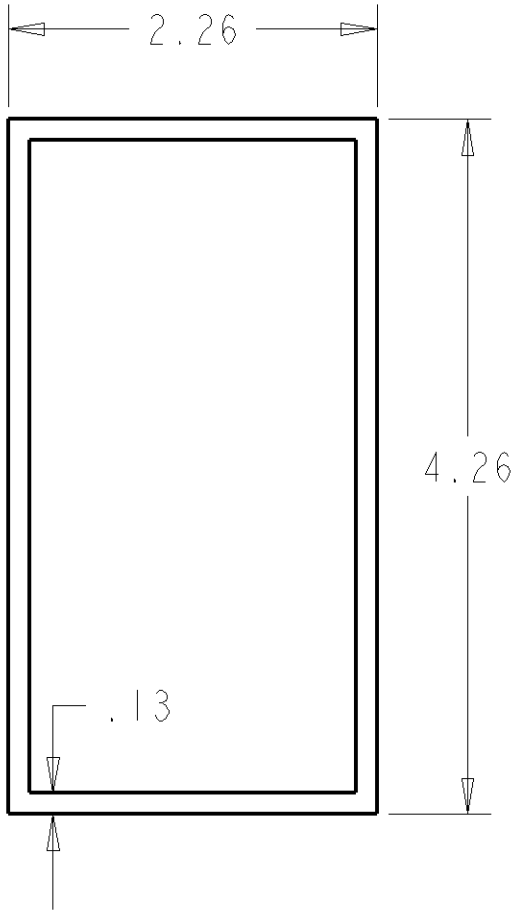
All Holes 10-32

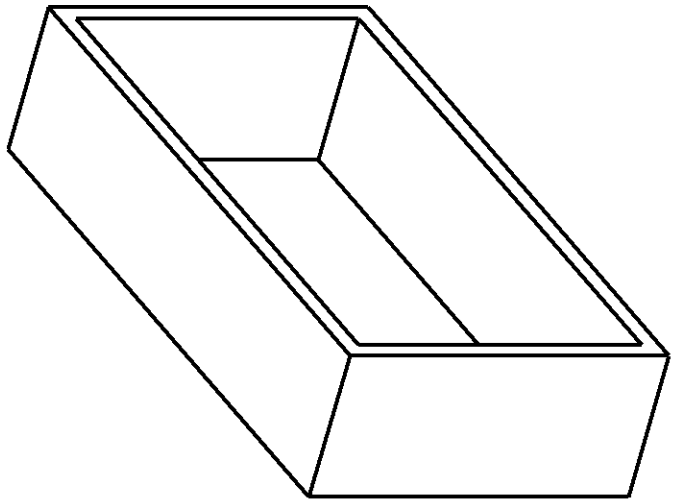




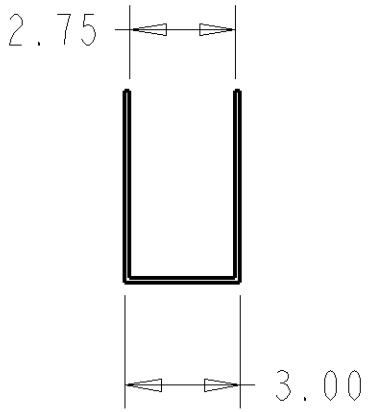
A-15

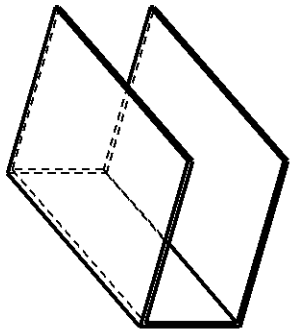
Electronics Battery Holder
Factory Robot Group

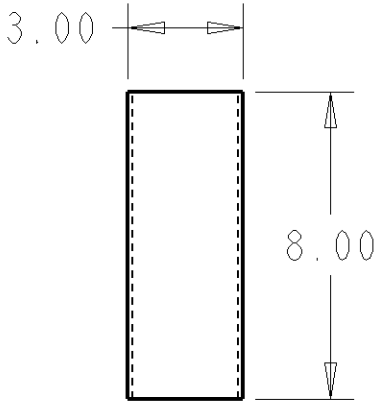


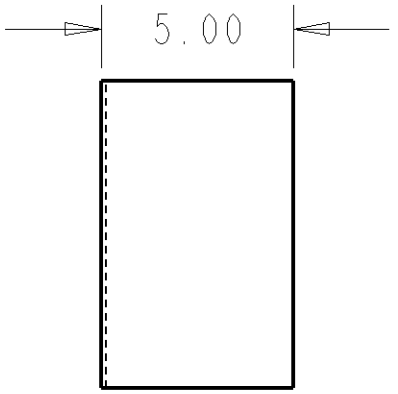


Electronics Holder
Factory Robot Group

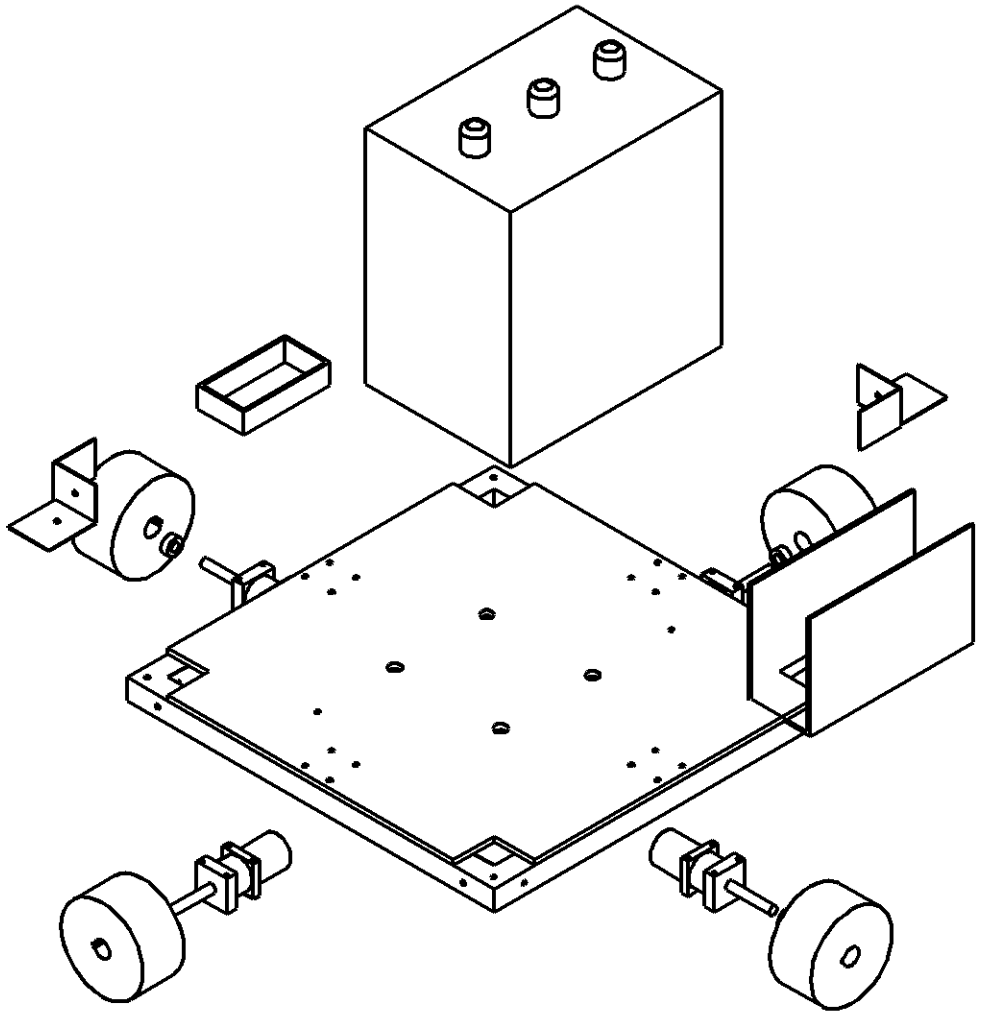


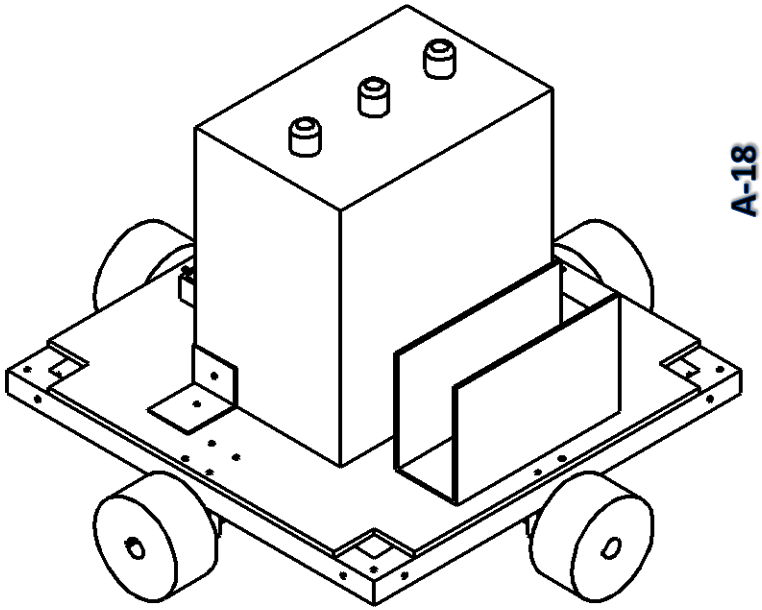


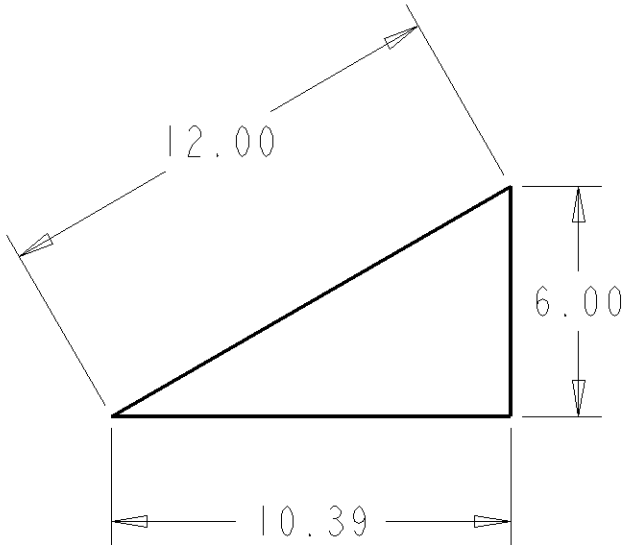




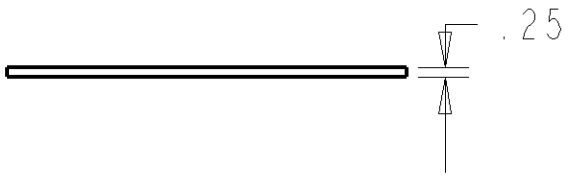
Mounted Electronics
Factory Robot Group

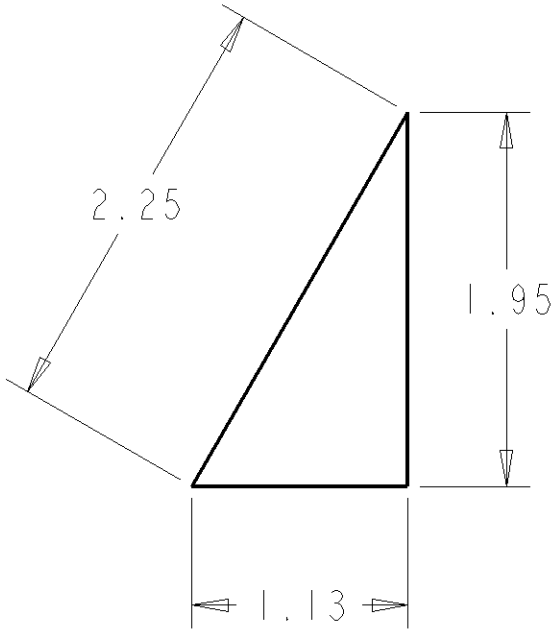




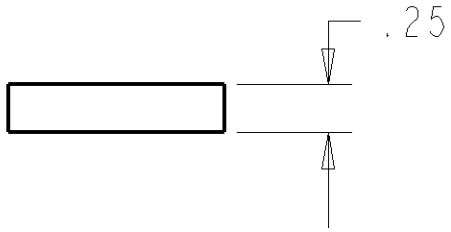


Large Angular Support
Factory Robot Group

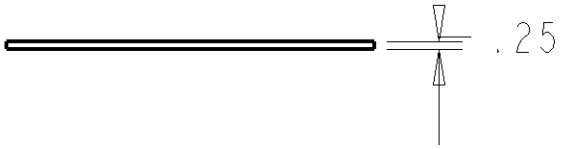




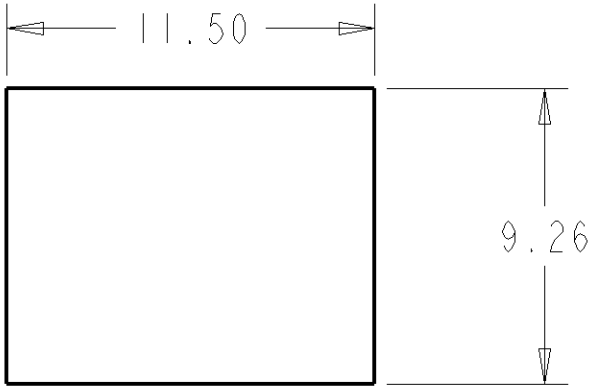
Small Angular Support
Factory Robot Group



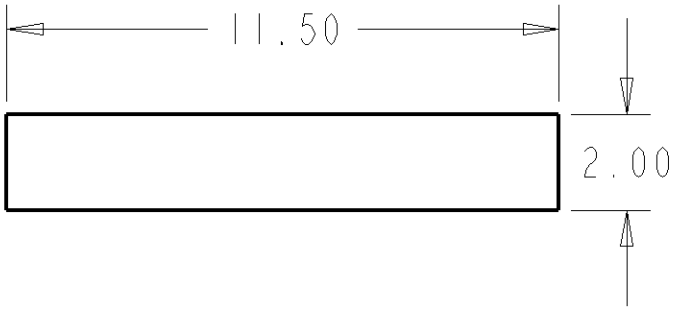
A-20



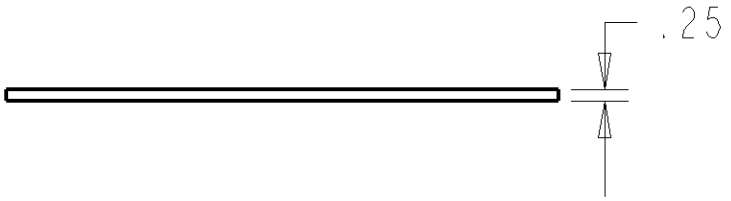
Holder Back
Factory Robot Group



A-21

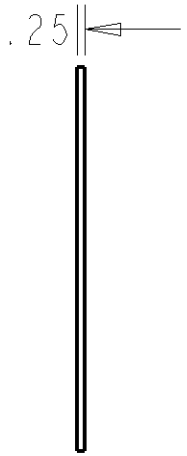


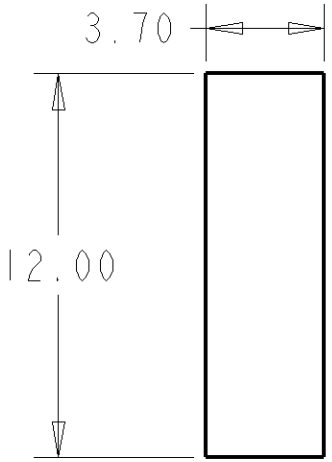
Holder Front
Factory Robot Group

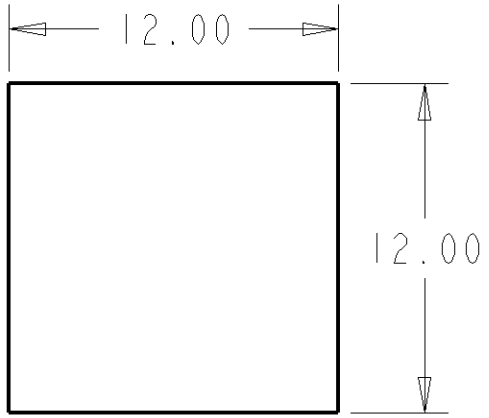


A-22

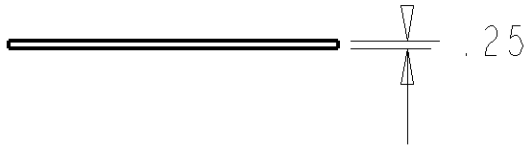
Holder Side
Factory Robot Group







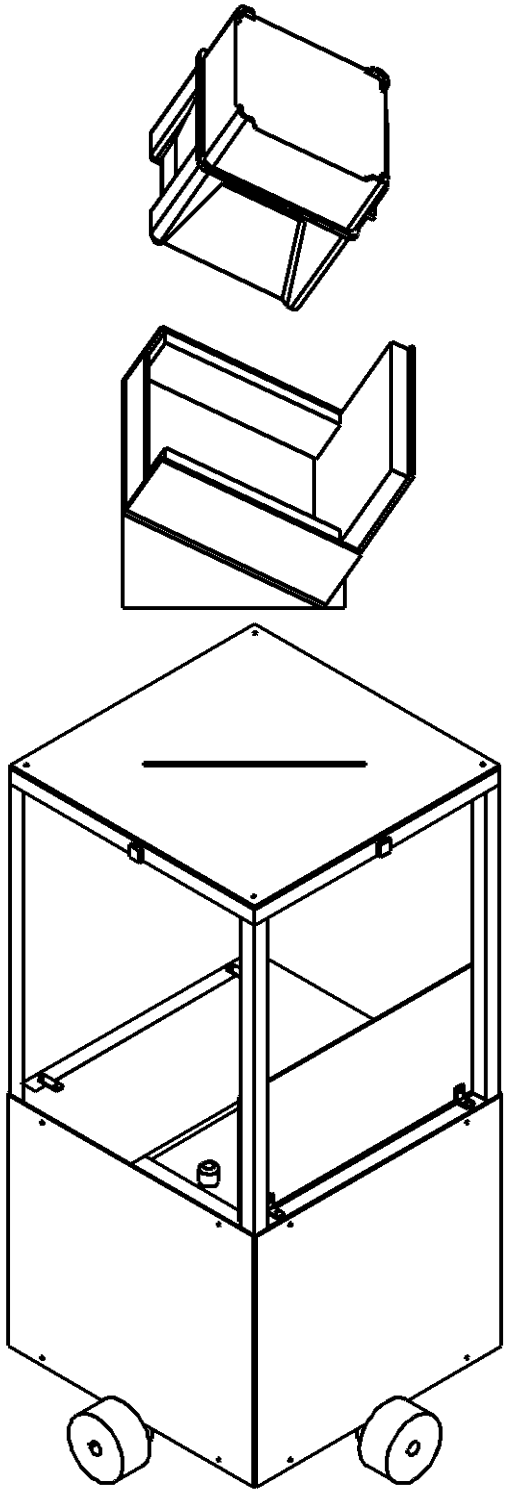
Holder Base
Factory Robot Group



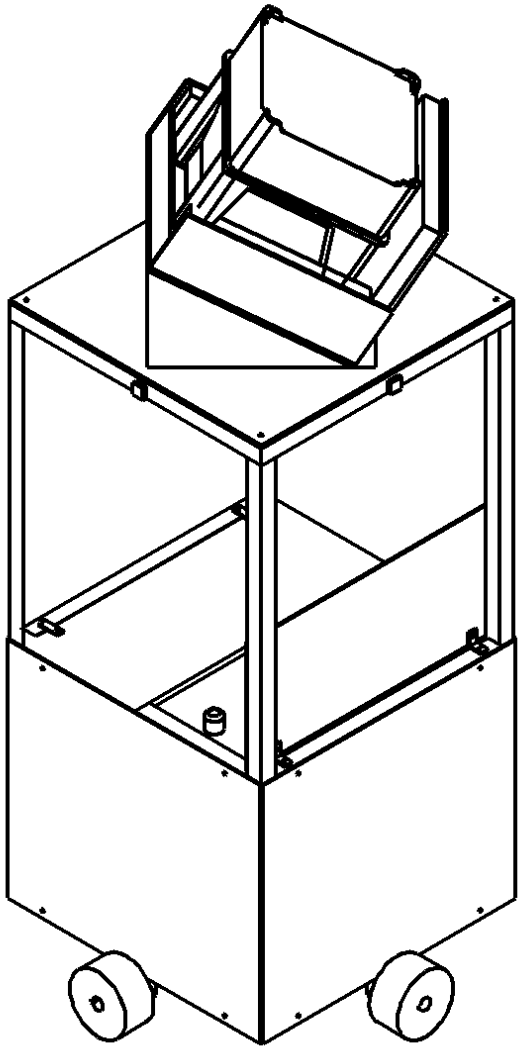
A-24

Holder Assem
Factory Robot Group

A-25



Complete Robot
Factory Robot Group



Appendix B: Circuits and Code Samples

Circuits and Diagrams:

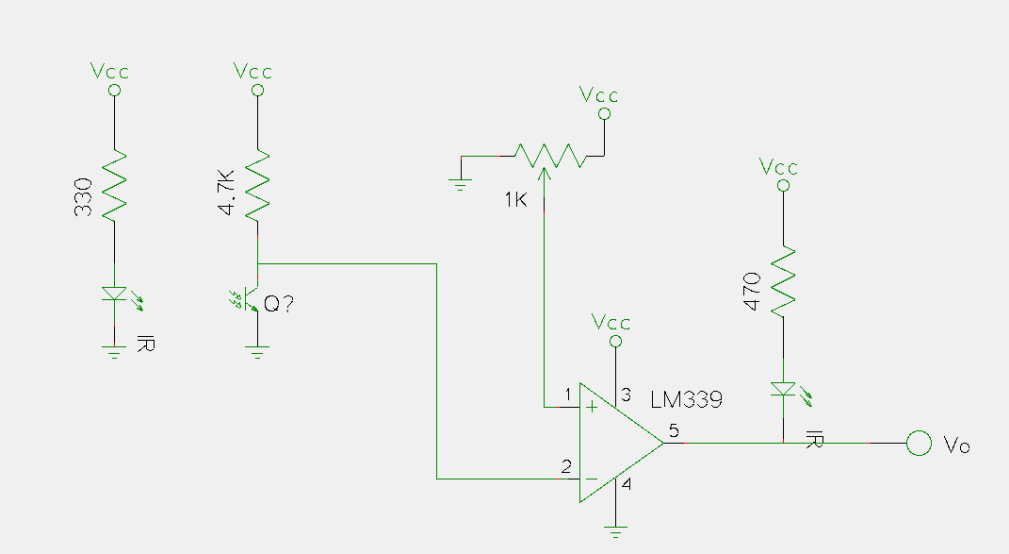


Figure 1: QRB1114 conditioner circuit

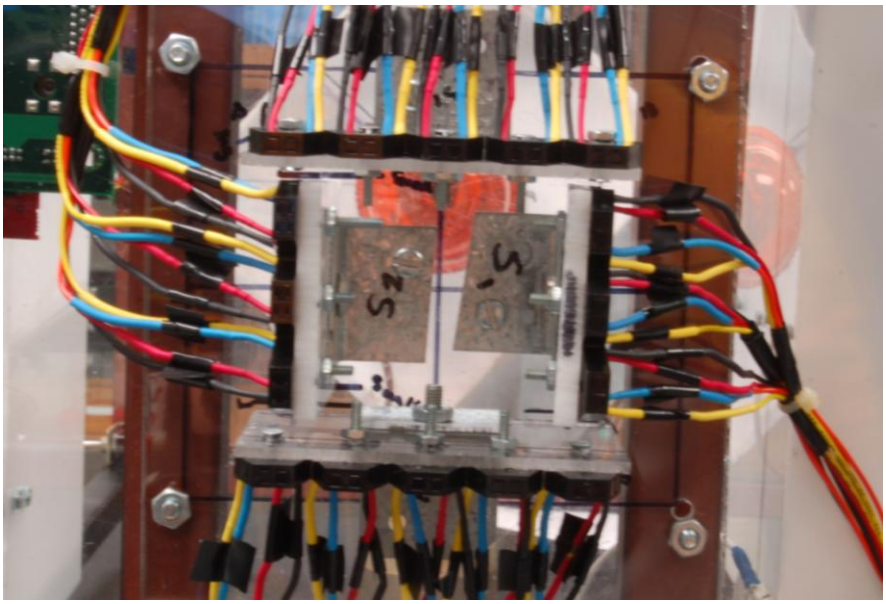


Figure 2: QRB1114 sensor array on bottom of robot

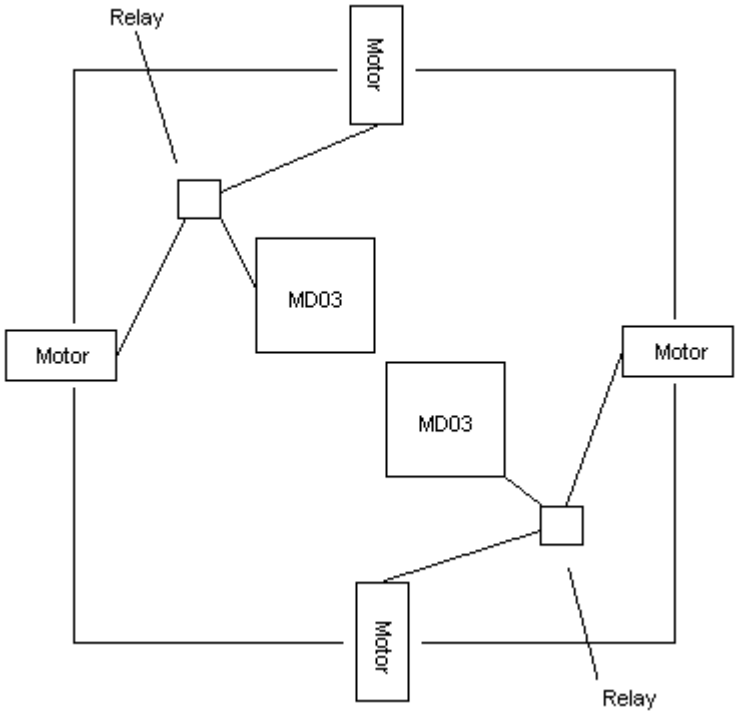


Figure 3: Switching diagram between motors and motor drivers

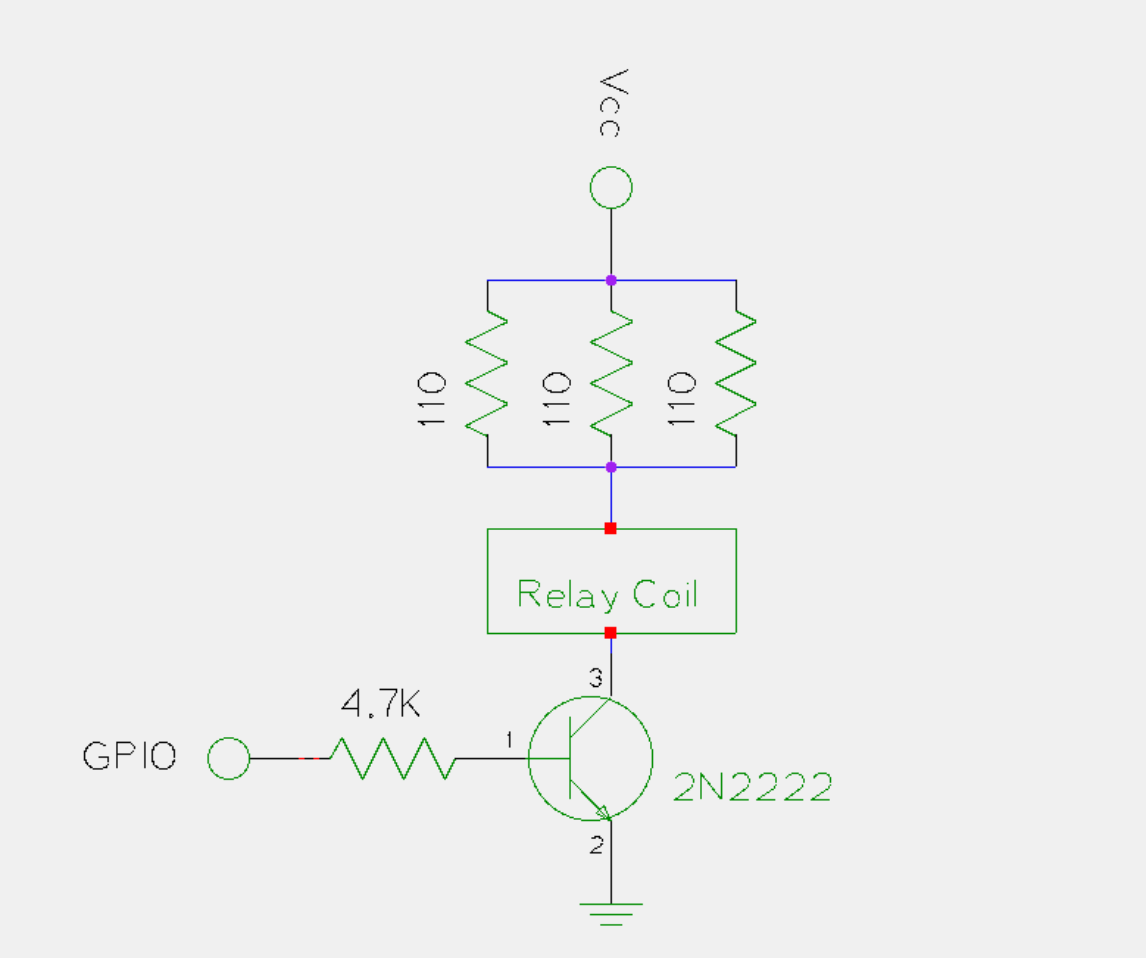


Figure 4: Relay buffer circuit

Code Samples:

```
#include <stdio.h>
#include <stdlib.h>
#include <i2cbridge.h>

// Program to interface between Devantech SRF-02 Sonar Rangers
// and the Diolan U2C-12 using the I2C protocol.

//HANDLE U2C_OpenDevice(BYTE);
//U2C_RESULT U2C_Read(HANDLE,PU2C_TRANSACTION);
//U2C_RESULT U2C_Write(HANDLE,PU2C_TRANSACTION);
//U2C_RESULT U2C_RW_Pack(HANDLE,PU2C_TRANSACTION_PACK,INT pack_count);

int main(void){

      int i;
      int rc=0; // will be set to a number upon success or failure of a
transaction
            // 0 means that the transaction was a success, and 1-16 mean
something bad happened
      HANDLE hDevice; // special data type used only to hold the device
number (in hex) of the u2c-12
      U2C_TRANSACTION
sonar1w,sonar1r,sonar2w,sonar2r,sonar3w,sonar3r,sonar4w,sonar4r; // struct
that contains all the data needed to perform a transaction
      PU2C_TRANSACTION_PACK transp[7];                                  //
fields are: nSlaveDeviceAddress, nMemoryAddressLength,
                                                                  //
nMemoryAddress, nBufferLength, Buffer
                                                                    // during a
write session, Buffer contains the data to be written.
                                                                  // during a read
session, Buffer contains the data read, where nBufferLength
                                                                  // is the number
of bytes to read.

      // some code taken from the U2C API Example Code
      // found at ~/i2c_bridge-0.1.0/u2c/

      // Open Device and grab device address (hDevice)
      hDevice = U2C_OpenDevice(0);
      if(hDevice == INVALID_HANDLE_VALUE)
            return -1; // exit if going crazy
      printf("hDevice = %p\n", hDevice);

      while(1){

            // the process for taking a reading in inches is to write 0x50 to
the command register (0x00),
            // wait 70 microseconds, then read 0x02 and 0x03 (0x02 is the
high byte of the range, 0x03 the low)


            // fill transaction struct for writing
            sonar1w.nSlaveDeviceAddress = 0x70;
```

```
sonar1w.nMemoryAddressLength = 1;
sonar1w.nMemoryAddress = 0x00;
sonar1w.nBufferLength = 1;
sonar1w.Buffer[0] = 0x50;

// fill transaction struct for writing
sonar2w.nSlaveDeviceAddress = 0x71;
sonar2w.nMemoryAddressLength = 1;
sonar2w.nMemoryAddress = 0x00;
sonar2w.nBufferLength = 1;
sonar2w.Buffer[0] = 0x50;

// fill transaction struct for writing
sonar3w.nSlaveDeviceAddress = 0x72;
sonar3w.nMemoryAddressLength = 1;
sonar3w.nMemoryAddress = 0x00;
sonar3w.nBufferLength = 1;
sonar3w.Buffer[0] = 0x50;

// fill transaction struct for writing
sonar4w.nSlaveDeviceAddress = 0x73;
sonar4w.nMemoryAddressLength = 1;
sonar4w.nMemoryAddress = 0x00;
sonar4w.nBufferLength = 1;
sonar4w.Buffer[0] = 0x50;

// Pinggggg!
rc = U2C_Write(hDevice,&sonar1w);
rc = U2C_Write(hDevice,&sonar2w);
rc = U2C_Write(hDevice,&sonar3w);
rc = U2C_Write(hDevice,&sonar4w);

sleep(1);

sonar1r.nSlaveDeviceAddress = 0x70;
sonar1r.nMemoryAddressLength = 1;
sonar1r.nMemoryAddress = 0x03;
sonar1r.nBufferLength = 1;

sonar2r.nSlaveDeviceAddress = 0x71;
sonar2r.nMemoryAddressLength = 1;
sonar2r.nMemoryAddress = 0x03;
sonar2r.nBufferLength = 1;

sonar3r.nSlaveDeviceAddress = 0x72;
sonar3r.nMemoryAddressLength = 1;
sonar3r.nMemoryAddress = 0x03;
sonar3r.nBufferLength = 1;

sonar4r.nSlaveDeviceAddress = 0x73;
sonar4r.nMemoryAddressLength = 1;
sonar4r.nMemoryAddress = 0x03;
sonar4r.nBufferLength = 1;

// Read back distance (in inches of course)
rc = U2C_Read(hDevice,&sonar1r);
rc = U2C_Read(hDevice,&sonar2r);
```

```
            rc = U2C_Read(hDevice,&sonar3r);
            rc = U2C_Read(hDevice,&sonar4r);


      printf("%2X\t%2X\t%2X\t%2X\n",sonar1r.Buffer[0],sonar2r.Buffer[0],sonar
3r.Buffer[0],sonar4r.Buffer[0]);

            // and do it all again forever until the end of time
      }

      return 0;

}
```

Figure 5: Code for Sonar Tester

```
#include <stdio.h>
#include <stdlib.h>
#include <i2cbridge.h>

// Program to interface between Devantech SRF-02 Sonar Rangers
// and the Diolan U2C-12 using the I2C protocol.

//HANDLE U2C_OpenDevice(BYTE);
//U2C_RESULT U2C_Read(HANDLE,PU2C_TRANSACTION);
//U2C_RESULT U2C_Write(HANDLE,PU2C_TRANSACTION);
//U2C_RESULT U2C_RW_Pack(HANDLE,PU2C_TRANSACTION_PACK,INT pack_count);

int main(void){

      int i;
      int rc=0; // will be set to a number upon success or failure of a
transaction
              // 0 means that the transaction was a success, and 1-16 mean
something bad happened
      HANDLE hDevice; // special data type used only to hold the device
number (in hex) of the u2c-12
      U2C_TRANSACTION
sonar1w,sonar1r,sonar2w,sonar2r,sonar3w,sonar3r,sonar4w,sonar4r; // struct
that contains all the data needed to perform a transaction
      PU2C_TRANSACTION_PACK transp[7];                                    //
fields are: nSlaveDeviceAddress, nMemoryAddressLength,
                                                                  //
nMemoryAddress, nBufferLength, Buffer
                                                                    // during a
write session, Buffer contains the data to be written.
                                                                  // during a read
session, Buffer contains the data read, where nBufferLength
                                                                  // is the number
of bytes to read.

      // some code taken from the U2C API Example Code
      // found at ~/i2c_bridge-0.1.0/u2c/

      // Open Device and grab device address (hDevice)
      hDevice = U2C_OpenDevice(0);
      if(hDevice == INVALID_HANDLE_VALUE)
            return -1; // exit if going crazy
      printf("hDevice = %p\n", hDevice);

      while(1){

            // the process for taking a reading in inches is to write 0x50 to
the command register (0x00),
            // wait 70 microseconds, then read 0x02 and 0x03 (0x02 is the
high byte of the range, 0x03 the low)


            // fill transaction struct for writing
            sonar1w.nSlaveDeviceAddress = 0x70;
            sonar1w.nMemoryAddressLength = 1;
            sonar1w.nMemoryAddress = 0x00;
            sonar1w.nBufferLength = 1;
```

```
sonar1w.Buffer[0] = 0x50;

// fill transaction struct for writing
sonar2w.nSlaveDeviceAddress = 0x71;
sonar2w.nMemoryAddressLength = 1;
sonar2w.nMemoryAddress = 0x00;
sonar2w.nBufferLength = 1;
sonar2w.Buffer[0] = 0x50;

// fill transaction struct for writing
sonar3w.nSlaveDeviceAddress = 0x72;
sonar3w.nMemoryAddressLength = 1;
sonar3w.nMemoryAddress = 0x00;
sonar3w.nBufferLength = 1;
sonar3w.Buffer[0] = 0x50;

// fill transaction struct for writing
sonar4w.nSlaveDeviceAddress = 0x73;
sonar4w.nMemoryAddressLength = 1;
sonar4w.nMemoryAddress = 0x00;
sonar4w.nBufferLength = 1;
sonar4w.Buffer[0] = 0x50;

// Pingggggg!
rc = U2C_Write(hDevice,&sonar1w);
rc = U2C_Write(hDevice,&sonar2w);
rc = U2C_Write(hDevice,&sonar3w);
rc = U2C_Write(hDevice,&sonar4w);

sleep(1);

sonar1r.nSlaveDeviceAddress = 0x70;
sonar1r.nMemoryAddressLength = 1;
sonar1r.nMemoryAddress = 0x03;
sonar1r.nBufferLength = 1;

sonar2r.nSlaveDeviceAddress = 0x71;
sonar2r.nMemoryAddressLength = 1;
sonar2r.nMemoryAddress = 0x03;
sonar2r.nBufferLength = 1;

sonar3r.nSlaveDeviceAddress = 0x72;
sonar3r.nMemoryAddressLength = 1;
sonar3r.nMemoryAddress = 0x03;
sonar3r.nBufferLength = 1;

sonar4r.nSlaveDeviceAddress = 0x73;
sonar4r.nMemoryAddressLength = 1;
sonar4r.nMemoryAddress = 0x03;
sonar4r.nBufferLength = 1;

// Read back distance (in inches of course)
rc = U2C_Read(hDevice,&sonar1r);
rc = U2C_Read(hDevice,&sonar2r);
rc = U2C_Read(hDevice,&sonar3r);
rc = U2C_Read(hDevice,&sonar4r);
```

```
      printf("%2X\t%2X\t%2X\t%2X\n",sonar1r.Buffer[0],sonar2r.Buffer[0],sonar
3r.Buffer[0],sonar4r.Buffer[0]);

            // and do it all again forever until the end of time
      }

      return 0;

}
```

Figure 6: Code for GPIO Speed and Functionality Test

```
#include <stdio.h>
#include <i2cbridge.h>

int ChangeSpeed(HANDLE);
int ChangeDir(HANDLE);
int Stop(HANDLE);

int main(void){

      HANDLE hDevice;
      int n,action;

      // first things first -- open U2C-12

      hDevice = U2C_OpenDevice( 0 );
      if(hDevice == INVALID_HANDLE_VALUE)
            return -1; // going crazy

      while(1){

            printf("Please choose an action:\n");
            printf("1. Change speed and acceleration\n");
            printf("2. Change direction\n");
            printf("3. Stop motors instantly\n");
            printf("4. Exit\n");
            printf("Choose the number of your choice: ");
            scanf("%d",&action);

            switch(action){
                  case 1:
                        n=ChangeSpeed(hDevice);
                        break;
                  case 2:
                        n=ChangeDir(hDevice);
                        break;
                  case 3:
                        n=Stop(hDevice);
                        break;
                  case 4:
                        printf("Leaving motors running at current
speed...\n");
                        printf("Goodbye!\n");
                        return 0;
                  default:
                        printf("Not a valid option!\n");
            }
      }
}

int ChangeSpeed(HANDLE hDevice){

      int int_accel,rc,speed;
      float time,usec_time;
      U2C_TRANSACTION maccel,mspeed,move;

      printf("Enter new \"speed\" (0-243): ");
      scanf("%d",&speed);
```

```
	printf("Acceleration time to new speed? ");
	scanf("%d",&int_accel);

	if(speed > 243)
		speed = 243; // MD03 clips it anyways, but just for robustness's
sake

	// compute acceleration value
//	usec_time = time * 1000000;
//	int_accel = (int)((usec_time/speed)-768)/125;

//	if(int_accel<0){
//		int_accel = 0;
//		printf("Requested acceleration too fast--can't be instant\n");
//	}
//	if(int_accel>255){
//		int_accel = 255;
//		printf("Requested acceleration too slow--we're not pitch\n");
//	}

//	printf("Accelerating to %d in %f
seconds!\n",speed,(((int_accel*125)+768)*speed)/1000000);

	printf("Accelerating to %d in %d seconds!\n",speed,int_accel);

	// build and write i2c structures
	// speed
	mspeed.nSlaveDeviceAddress = 0x58;
	mspeed.nMemoryAddressLength = 1;
	mspeed.nMemoryAddress = 0x02;
	mspeed.nBufferLength = 1;
	mspeed.Buffer[0] = speed;

	rc = U2C_Write(hDevice,&mspeed);
	if (rc){
		printf("Uhoh. Error No.: %d\n",rc);
		return rc;
	}

	// acceleration
	maccel.nSlaveDeviceAddress = 0x58;
	maccel.nMemoryAddressLength = 1;
	maccel.nMemoryAddress = 0x03;
	maccel.nBufferLength = 1;
	maccel.Buffer[0] = int_accel;

	rc = U2C_Write(hDevice,&maccel);

	if (rc){
		printf("Uhoh. Error No.: %d\n",rc);
		return rc;
	}

	// MOVE!
	move.nSlaveDeviceAddress = 0x58;
	move.nMemoryAddressLength = 1;
	move.nMemoryAddress = 0x00;
```

```
	move.nBufferLength = 1;
	move.Buffer[0] = 0x01; // assume we're moving forward

	rc = U2C_Write(hDevice,&move);

	if (rc){
		printf("Uhoh. Error No.: %d\n",rc);
		return rc;
	}

	return 0; // if everything's going well

}

int ChangeDir(HANDLE hDevice){

	char dir;
	int rc;
	U2C_TRANSACTION mdir;

	printf("Which direction? ");
	scanf("%s",&dir);

	if(dir == 'f'){
		// change direction struct
		mdir.nSlaveDeviceAddress = 0x58;
		mdir.nMemoryAddressLength = 1;
		mdir.nMemoryAddress = 0x00;
		mdir.nBufferLength = 1;
		mdir.Buffer[0] = 0x01;

		rc = U2C_Write(hDevice,&mdir);

		if (rc){
			printf("Uhoh. Error No.: %d\n",rc);
			return rc;
		}
	} else if(dir == 'r'){
		// change direction struct
		mdir.nSlaveDeviceAddress = 0x58;
		mdir.nMemoryAddressLength = 1;
		mdir.nMemoryAddress = 0x00;
		mdir.nBufferLength = 1;
		mdir.Buffer[0] = 0x02;

		rc = U2C_Write(hDevice,&mdir);

		if (rc){
			printf("Uhoh. Error No.: %d\n",rc);
			return rc;
		}
	} else {
		printf("Invalid direction. Please try again.\n");
		return 0;
	}

	return 0;
```

```
}

int Stop(HANDLE hDevice){

      int rc;
      U2C_TRANSACTION stop;

      printf("Stopping now!\n");

      stop.nSlaveDeviceAddress = 0x58;
      stop.nMemoryAddressLength = 1;
      stop.nMemoryAddress = 0x00;
      stop.nBufferLength = 1;
      stop.Buffer[0] = 0x00;

      rc = U2C_Write(hDevice,&stop);

      if (rc){
            printf("Uhoh. Error No.: %d\n",rc);
            return rc;
      }

      return 0;

}
```

Figure 7: Code to control a single motor's speed, acceleration, and direction

```
#include <stdio.h>
#include <i2cbridge.h>
#include <stdlib.h>
#include <string.h>

// motor driver situated at front of robot
#define     MD_F        0x58
// motor driver situated at back of robot
#define     MD_B        0x59
// extremes of speed and acceleration
#define     SLOW_ACCEL  255
#define           MED_ACCEL   100
#define     FAST_ACCEL  0
#define     STOP_SPEED  0
#define           MVMT_THRESH 100
#define     TOP_SPEED   243
// interim speeds
#define           QUARTER_SPEED     120
#define           HALF_SPEED  160
#define           THREEQUARTERSPD   210
// directions
#define           FORWARD           0x01
#define           BACKWARD    0x02
#define           INST_STOP   0x00

int InitSensors(HANDLE hDevice);
void MotorLogic(BOOL s2, BOOL s3, BOOL s4, char* dir);
int Drive(HANDLE hDevice, char dir);

int main(void){

      HANDLE hDevice;
      BOOL s2,s3,s4;
      int health;
      char dir;

      // first things first
      hDevice = U2C_OpenDevice( 0 );
      if(hDevice == INVALID_HANDLE_VALUE){
            printf("No Device. Plug in maybe?\n");
            return -1;
      }

      health=InitSensors(hDevice);

      if(health)
            printf("Something's gone wrong. Exiting: %d\n",health);

      while(1){

            // Program Flow:
            // Check sensors, then adjust direction
            // i.e.
            //    CheckSensors();
            //    Drive();

            // Check the sensors (s1=leftmost,s5=rightmost)
```

```
            // For this incarnation, we're only going to use the middle 3
//          U2C_SingleIoRead(hDevice,0,&s1);
            U2C_SingleIoRead(hDevice,1,&s2);
            U2C_SingleIoRead(hDevice,2,&s3);
            U2C_SingleIoRead(hDevice,3,&s4);
//          U2C_SingleIoRead(hDevice,4,&s5);

            // Do some logic to tell the motors what to do
            MotorLogic(s2,s3,s4,&dir);

            // Tell the motors what to do
            Drive(hDevice,dir);

      }

}

int InitSensors(HANDLE hDevice){

      ULONG IoValue,IoMask;
      int rc;

      // We're going to use PA0-PA4 for the IR sensors so set them as inputs
      IoValue = 0x00;
      IoMask = 0x1F;

      rc = U2C_SetIoDirection(hDevice,IoValue,IoMask);

      if(rc){
            printf("Something's wrong with GPIO. Exiting: %d\n",rc);
            return 1;
      }

      return 0;

}

void MotorLogic(BOOL s2, BOOL s3, BOOL s4,char* dir){

      // inputs: sensor values (s2,s3,s4): either a 1 or a 0
      // outputs: direction (dir):
hardleft,softleft,straight,softright,hardright,reverse

      // check left sensor first
      if (s2) {
            // if it's on line, see if middle sensor is on line
            if (s3) {
                  // if it's on line, see if right sensor is on line
                  if (s4) {
                        *dir = 's'; // straight
                  } else {
                        *dir = 'l'; // soft left turn
                  }
            } else {
                  if (s4) { // something's wierd here
                        *dir = 'f'; // full stop
                  } else {
```

```
                        *dir = 'h'; // hard left turn
                  }
            }
      } else {
            // so left sensor isn't on line, check middle
            if (s3) { // Aha! Found you!
                  // just to check for crazy times
                  if (s4) {
                        *dir = 'r'; // soft right turn
                  } else {
                        *dir = 'f'; // full stop
                  }
            } else { // Damn! Lost you!
                  if (s4) {
                        *dir = 't'; // hard right turn
                  } else { // not seeing anything
                        *dir = 'f'; // full stop
                  }
            }
      }

      // for the easily confused (and for myself)
      // i changed the strings that were there previously to single
characters
      // since strings are a bitch to pass back via pointers (</3), so i've
      // provided the following key
      // s: straight
      // l: soft left turn
      // f: full stop
      // h: hard left turn
      // r: soft right turn
      // t: hard right turn (because it's to the right of r and s was already
being used :D )

}

int Drive(HANDLE hDevice, char dir){

      int mot1spd,mot2spd,mot1accl,mot2accl,mot1dir,mot2dir,rc;
      U2C_TRANSACTION
motorFspd,motorBspd,motorFaccl,motorBaccl,motorFmove,motorBmove;

      // vary the speed parameters for each wheel based on the direction
calculated in MotorLogic()
      if (dir == 's'){
            mot1spd = HALF_SPEED;
            mot2spd = HALF_SPEED;
            mot1accl = FAST_ACCEL;
            mot2accl = FAST_ACCEL;
            mot1dir = FORWARD;
            mot2dir = BACKWARD;
      }
      if (dir == 'l'){
            mot1spd = HALF_SPEED;
            mot2spd = QUARTER_SPEED;
            mot1accl = FAST_ACCEL;
            mot2accl = FAST_ACCEL;
```

```
      mot1dir = FORWARD;
      mot2dir = BACKWARD;
}
if (dir == 'h'){
      mot1spd = THREEQUARTERSPD;
      mot2spd = QUARTER_SPEED;
      mot1accl = FAST_ACCEL;
      mot2accl = FAST_ACCEL;
      mot1dir = FORWARD;
      mot2dir = BACKWARD;
}
if (dir == 'r'){
      mot1spd = QUARTER_SPEED;
      mot2spd = HALF_SPEED;
      mot1accl = FAST_ACCEL;
      mot2accl = FAST_ACCEL;
      mot1dir = FORWARD;
      mot2dir = BACKWARD;
}
if (dir == 't'){
      mot1spd = QUARTER_SPEED;
      mot2spd = THREEQUARTERSPD;
      mot1accl = FAST_ACCEL;
      mot2accl = FAST_ACCEL;
      mot1dir = FORWARD;
      mot2dir = BACKWARD;
}
if (dir == 'f'){
      mot1spd = STOP_SPEED;
      mot2spd = STOP_SPEED;
      mot1accl = MED_ACCEL;
      mot2accl = MED_ACCEL;
      mot1dir = FORWARD;
      mot2dir = BACKWARD;
}

// write necessary values to motor drivers
// motor driver F
// write speed
motorFspd.nSlaveDeviceAddress = MD_F;
motorFspd.nMemoryAddressLength = 1;
motorFspd.nMemoryAddress = 0x02;
motorFspd.nBufferLength = 1;
motorFspd.Buffer[0] = mot1spd;

rc = U2C_Write(hDevice,&motorFspd);
if (rc){
      printf("Uhoh. Error No.: %d\n",rc);
      return rc;
}

// write accel
motorFaccl.nSlaveDeviceAddress = MD_F;
motorFaccl.nMemoryAddressLength = 1;
motorFaccl.nMemoryAddress = 0x03;
motorFaccl.nBufferLength = 1;
motorFaccl.Buffer[0] = mot1accl;
```

```
	rc = U2C_Write(hDevice,&motorFaccl);
	if (rc){
		printf("Uhoh. Error No.: %d\n",rc);
		return rc;
	}

	// motor driver B
	// write speed
	motorBspd.nSlaveDeviceAddress = MD_B;
	motorBspd.nMemoryAddressLength = 1;
	motorBspd.nMemoryAddress = 0x02;
	motorBspd.nBufferLength = 1;
	motorBspd.Buffer[0] = mot2spd;

	rc = U2C_Write(hDevice,&motorBspd);
	if (rc){
		printf("Uhoh. Error No.: %d\n",rc);
		return rc;
	}

	// write accel
	motorBaccl.nSlaveDeviceAddress = MD_B;
	motorBaccl.nMemoryAddressLength = 1;
	motorBaccl.nMemoryAddress = 0x02;
	motorBaccl.nBufferLength = 1;
	motorBaccl.Buffer[0] = mot2accl;

	rc = U2C_Write(hDevice,&motorBaccl);
	if (rc){
		printf("Uhoh. Error No.: %d\n",rc);
		return rc;
	}

	// Write direction and GO!
	// Remember 0x01 means forward, 0x02 means reverse, so with that in
mind
	// motor driver F
	motorFmove.nSlaveDeviceAddress = MD_F;
	motorFmove.nMemoryAddressLength = 1;
	motorFmove.nMemoryAddress = 0x00;
	motorFmove.nBufferLength = 1;
	motorFmove.Buffer[0] = mot1dir;

	// motor driver B
	motorBmove.nSlaveDeviceAddress = MD_B;
	motorBmove.nMemoryAddressLength = 1;
	motorBmove.nMemoryAddress = 0x00;
	motorBmove.nBufferLength = 1;
	motorBmove.Buffer[0] = mot2dir;

	rc = U2C_Write(hDevice,&motorFmove);
	if (rc){
		printf("Uhoh. Error No.: %d\n",rc);
		return rc;
	}
```

```
        rc = U2C_Write(hDevice,&motorBmove);
        if (rc){
              printf("Uhoh. Error No.: %d\n",rc);
              return rc;
        }
}
```

Figure 8: Code to let robot follow a line

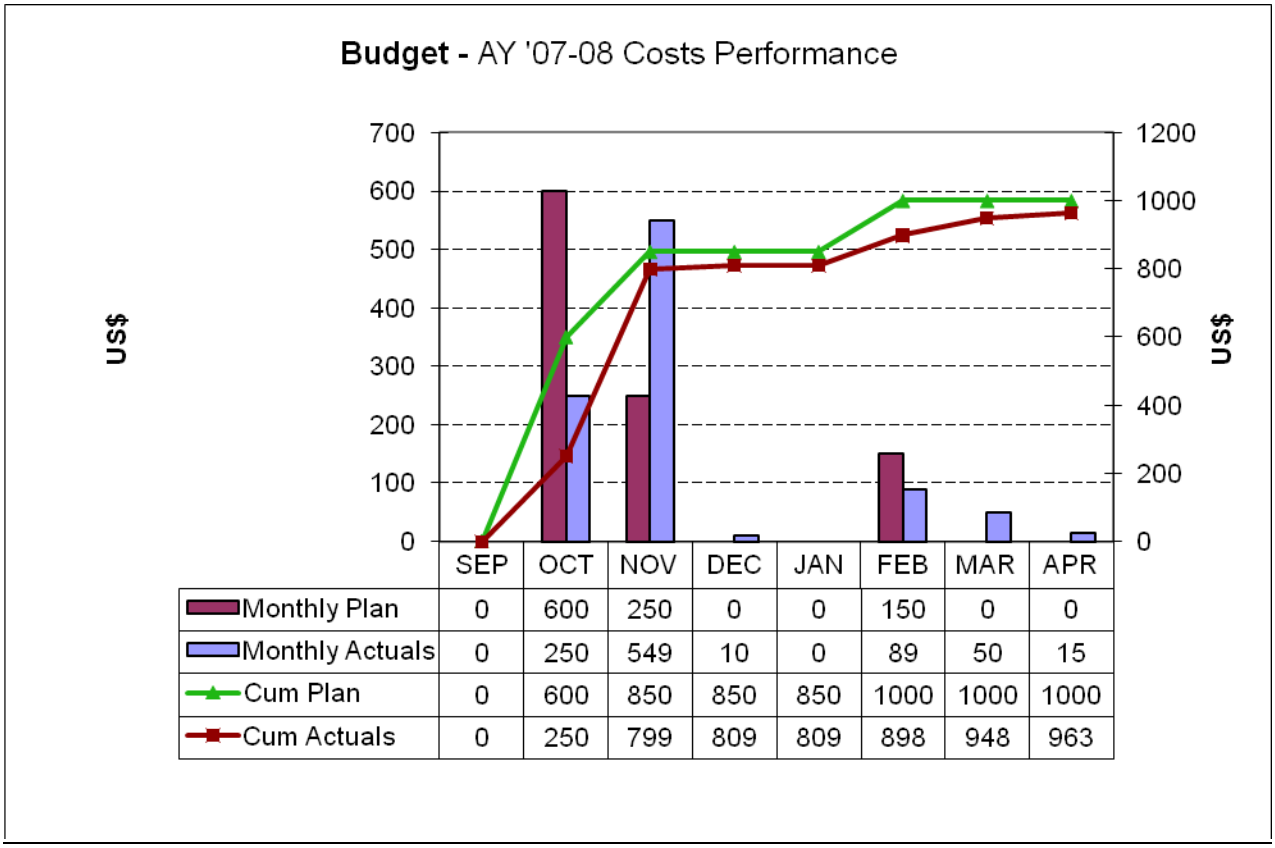
Appendix C: Bill of Materials

Section	Component	Description	Quantity
Body	Clear Side Paneling	1/8" thick Clear Plexi 36"x36"	1
Body	White Side Paneling	1/8" thick White Plexi 36"x36"	1
Body	Top and Bottom Base	1/4" thick Clear Plexi 18"x18"	2
Body	Metal Tubing	8' long 1"x1" A34 Carbon Steel Square Tubing	4
Body	Screws	1 1/2" long 10-36	100
Body	Nuts	10-36 nuts	100
Body	Washer	10-36 washer	100
Body	Wheels	Kornylak Omni-Directional Wheels	4
Body	Hinges	2" Small fence hinges	4
Body	Hinge Holders	10-36 Butterfly Nuts	2
Body	Magnetic Holder for Light	6"x6" Steel Plate	1
Body	Battery Holder	1 1/2" long 1"x1" Angle-Iron	4
Body	IR holder	1 1/2" long 1"x1" Angle-Iron	4
Body			
Body			
Cassette Holder	Case	1/4" thick White Plexi 24"x24"	1
Cassette Holder	Padding	1/2" Black Foam Insulation 8'x6'	1
Electronics	Battery	Trojan T-105 plus 6V battery	1
Electronics	Motors	Banebots RS-385	4
Electronics	Safety Light	Amber Rotating Light	1
Electronics	Computer	Genesi Efika	1
Electronics	Hard Drive	IBM Travelstar DJSA-220	1
Electronics	I/O Controller	Diolan U2C-12	1
Electronics	Motor Drivers	Devantech MD03	2
Electronics	Sonar Sensors	Devantech SRF02	4
Electronics	IR Sensors	Fairchild QRB1114	20
Electronics	Switch	SPDT Switch	1
Electronics	14 Amp Fuses		2
Electronics	Distribution Bar		1
Electronics	Barrier Jumpers		11
Electronics	Battery	12V Universal Battery	1
Electronics	Power Supply	Pico PSU	1
Electronics	Power Supply	Logisys PS480	1
Electronics	24 gauge wire		
Electronics	12 gauge wire		
Electronics	Various connectors		

Appendix D: Total Budget

Total Spent per Month:

	SEP	OCT	NOV	DEC	JAN	FEB	MAR	APR
Monthly Plan	0	600	250	0	0	150	0	0
Monthly Actuals	0	250	549	10	0	89	50	15
Cum Plan	0	600	850	850	850	1000	1000	1000
Cum Actuals	0	250	799	809	809	898	948	963
Cum Variance	0	350	51	41	41	102	52	37



Appendix D: Total Budget

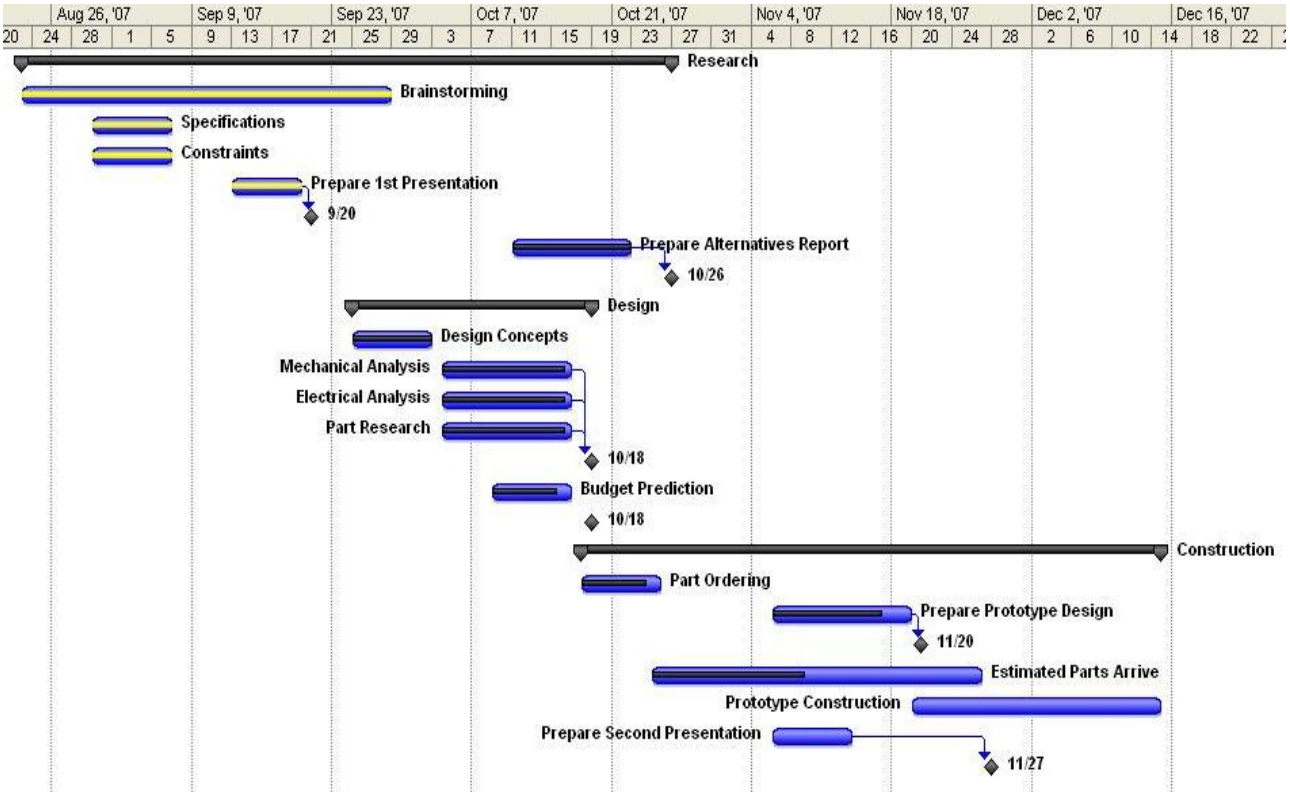
Detailed Budget Summary:

Parts	**Cost**	**Notes**
Motors (x4)	$150	
Battery	($130)	Obtained free from Physical Plant
Wheels (x4)	$100	Kornylak Omni-directional wheels
Base	($40)	Obtained free from Manuel
Flashing light	$19.99	
Electronic Accessories	$16.67 ($50)	Barrier jumpers and crimpers Wires, switches, circuit boards, etc. obtained from Manuel and Ernest
Motor drivers (x4)	$258.84	
Sonar sensors	$110.95	
Hard-drive	($223)	New Item Price; obtained from Dr. Nickels
Efika	($200)	Joined Powerdeveloping program
U2C-12	$96.55	
Battery power supply	$61.83	To power Efika
Wall power supply	$20.02	To power the Efika while testing
Mechanical Accessories	$14.83 ($20)	Plexy for base, screws, nuts, etc. Bolts, hinges, tape, etc. obtained from Manuel
Cassette holder	$73	Includes Plexy, glue and foam padding
QRB-114 IR sensors	$40.80	
Total spent	$963.48	
Total cost of robot	$1626.48	Includes items obtained for no cost

Appendix E: Schedule/ Man Hours

Fall Semester:

Gantt Chart (Fall 2007):



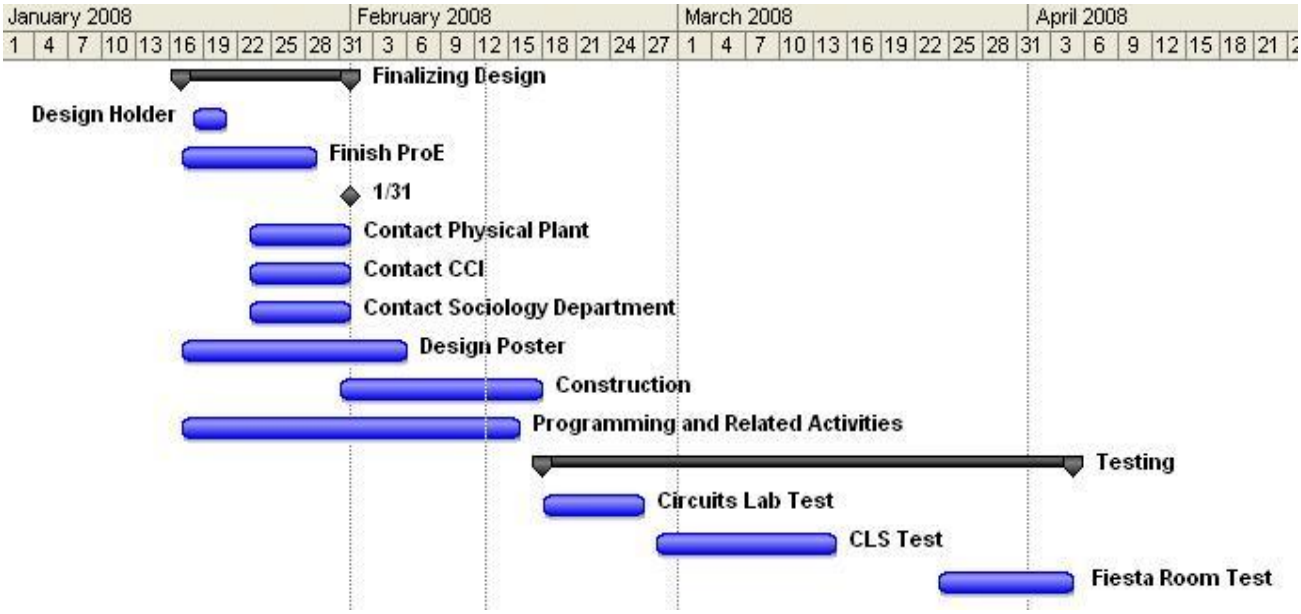
Student Work Hours per Month:

	SEP		OCT		NOV		DEC	
Team Plan		180.00		180.00		180.00		100.00
Mike Actual	35.00		33.00		34.00		8.00	
Sean Actual	35.00		38.00		39.00		4.00	
Marton Actual	33.00		36.00		32.00		2.00	
Jude Actual	28.00		20.00		27.00		2.00	
Charles Actual	34.00		33.00		40.00		10.00	
Team Actuals	165.00		160.00		172.00		26.00	
Cum Plan		180		360		540		640
Cum Actuals	165		325		497		523	
Cum Variance		-15.00		-20.00		-8.00		-74.00

Appendix E: Schedule/ Man Hours

Spring Semester:

Gantt Chart (Spring 2008):



Student Work Hours per Month:

	JAN		FEB		MAR		APR	
Team Plan		200.00		200.00		200.00		200.00
Mike Actual	26.00		38.00		37.00		45.00	
Sean Actual	22.00		32.00		30.00		40.00	
Marton Actual	18.00		28.00		34.00		42.00	
Jude Actual	15.00		23.00		26.00		38.00	
Charles Actual	28.00		50.00		40.00		46.00	
Team Actuals	109.00		171.00		167.00		211.00	
Cum Plan		840		1040		1240		1440
Cum Actuals	632		803		970		1181	
Cum Variance		-91.00		-29.00		-33.00		11.00

Appendix E: Schedule/ Man Hours

Budget Hours Summary: