Spherical Tumbler Robot with Position Sensing

ME433 Team L

Natalia Alvarez, Claire Melvin, Anvesh Tanuku

June 14, 2013

Contents

List of Figures

1 Project Description

The following report is a detailed description of the design and creation of a spherical tumbler robot to be used for research. This robot was conceptualized and prototyped for ME 433 Advanced Mechatronics, at Northwestern University.

1.1 Scope of Research

A common energy waste in industry is over-mixing fluids past this point of total mixture. It is a new source for research to determine the point at which a sample of particles will reach "total mixture." By determining an algorithm of motion to reach total mixture, money and energy will be saved. The spherical tumbler is designed specifically for this purpose. By using a spherical model, the sample can be tumbled in any direction, helping to find the best combination of rotations. The tumbler holds a clear acrylic sphere, partially filled with beads, and can roll it over itself around two predetermined axes.

1.2 Design Considerations

There are a number of specifications required for the practical use of the spherical tumbler. The user is able to enter the angle the tumbler will rotate in a first axis, and the angle to rotate in a second axis. The default second axis is orthogonal to the first axis, however the angle between axes can be changed if desired.

To model mixture patterns, the sample, sphere, and tumbler will be imaged together using an X-Ray machine. The machine is 14" wide x 18" long x 14" high, requiring that the tumbler remain within these measurements. In addition, the center of the sphere is required to be roughly 7.84" from the top of the image intensifier and that minimal parts are located directly below the sphere, as to get the best image of the sphere.

2 Theory of Operation

The spherical tumbler consists of two optical mouse sensors and three omniwheels driven by stepper motors. The sphere is supported on the wheels and the wheels allow slip in directions in which they do not drive. The two mouse sensors give velocity readings in two orthogonal directions on the surface of the sphere. When turned on, each motor induces an angular velocity vector, in the same direction as a vector orthogonal to it's point of contact with the motors. The combination of the three vectors produces an orthogonal basis. This allows us to generate any arbitrary angular velocity vector using a linear combination of motor speeds. We now go into details on how the different components work together.

Determining Angular Velocity from Sensor Readings

In order to track a specific trajectory, the user supplies us with a desired angular velocity vector and we use readings from the mouse sensors to determine the actual angular velocity and correct any deviations from the desired trajectory.

Our objective is to find the angular velocity components in the x,y and z directions $(\omega_x, \omega_y, \omega_z)$. Recall each mouse sensor gives us two readings: horizontal velocity and vertical velocity. Let v_{ix} be the horizontal velocity reading of sensor i and let v_{iy} be the same for the vertical velocity reading. Let $\vec{s}_{ix}$, $\vec{s_{iy}p_i}$ be the sensing direction of the i^{th} sensor in the x direction, the sensing direction of the i^{th} sensor in the y direction and the omni-wheel contact respectively. If the angular velocity of the sphere is given by $\vec{\omega} = (\omega_x, \omega_y, \omega_z)$, the circumferential speed is given by

$$\vec{v} = \vec{\omega} \times \vec{p_i} \tag{1}$$

.

The projection of the circumferential velocity in a particular sensing direction is given by:

$$\vec{v}_{i(x,y)} = \vec{s}_{i(x,y)} \cdot \vec{v} = \vec{s}_{i(x,y)} \cdot (\vec{\omega} \times \vec{p}_i) \tag{2}$$

$$= \vec{\omega} \cdot (\vec{p}_i \times \vec{s}_{i(x,y)}) \tag{3}$$

We use sensor1's horizontal and vertical readings as well as sensor 3's vertical velocity reading (v_{1x}, v_{1y} and v_{3y} respectively) to obtain three velocity readings. We now have three equations and three unknowns $(\omega_x, \omega_y, \omega_z)$. Written in matrix form:

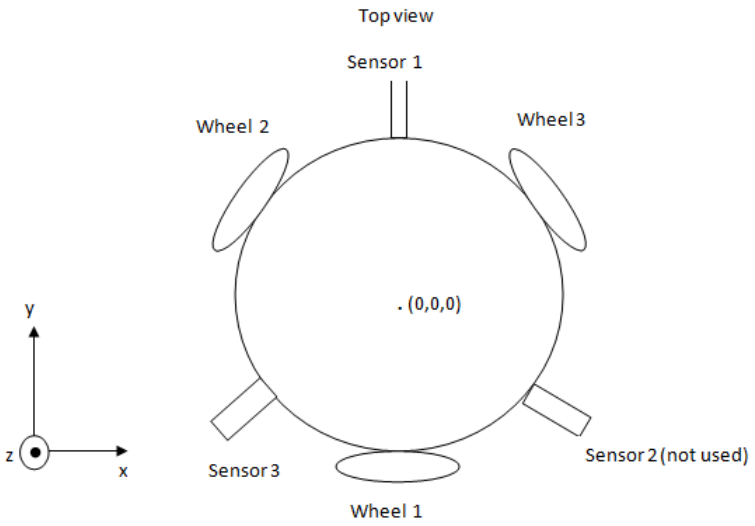
$$\begin{pmatrix} v_{1x} \\ v_{1y} \\ v_{3y} \end{pmatrix} = \begin{pmatrix} \vec{p}_1 \times \vec{s}_{1x} \\ \vec{p}_1 \times \vec{s}_{1y} \\ \vec{p}_3 \times \vec{s}_{3y} \end{pmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} = A\vec{\omega} \tag{4}$$

By inverting the A matrix and multiplying it's inverse by our sensor readings, we have the ability to read sense the angular velocity.

2.1 System Geometry

In calculating our A^{-1} matrix, a few details of our system's geometry need to be considered. The radius of our sphere is seven centimeters. We allow the origin of the system to be at the center of the sphere. Each omni-wheel's point of contact with the sphere is an angle is 45 degrees below the x-y plane as is the point of contact of each sensor. The wheels are placed 120 degrees apart and each sensor is directly across from each wheel. The number of each sensor corresponds to the number of the wheel it sits across from. A top view of this configuration is shown below:

Figure 1: Top View of tumbler



Using this geometry, the coordinates of contact between the motors and the sphere are easily found to be:

$$\text{wheel 1, } p_1\text{: } \left(0, \frac{7\sqrt{2}}{2}, -\frac{7\sqrt{2}}{2}\right)$$

$$\text{wheel 2, } p_2\text{: } \left(\frac{7\sqrt{6}}{4}, -\frac{7\sqrt{2}}{4}, -\frac{7\sqrt{2}}{2}\right)$$

$$\text{wheel 3, } p_3\text{: } \left(-\frac{7\sqrt{6}}{4}, -\frac{7\sqrt{2}}{4}, -\frac{7\sqrt{2}}{2}\right)$$

Sensors are numbered with respect to the motor they sit across from. Due to port constraints on the PIC32, we are limited to using two SPI ports for communication, therefore we could only use sensors one and three. Relative to this geometry the sensor coordinates are:

$$\text{sensor 1: } \left(0, \frac{7\sqrt{2}}{2}, -\frac{7\sqrt{2}}{2}\right)$$
$$\text{sensor 3: } \left(\frac{7\sqrt{6}}{4}, \frac{7\sqrt{2}}{4}, -\frac{7\sqrt{2}}{2}\right)$$

The sensing directions, are easily found to be:

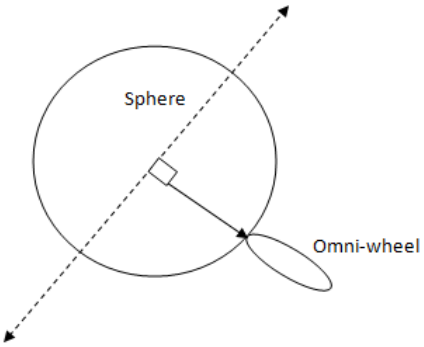
$$s_{1x} : (-1, 0, 0)$$
$$s_{1y} : \left(0, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$$
$$s_{3y} : \left(\frac{\sqrt{6}}{4}, \frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{2}\right)$$

Using this information, the A^{-1} matrix is easily calculated, and can be found in feedback.c, (the function update_w). A more detailed description of system geometry can be found in section C.

2.2 Feedback Control

Recall, each motor induces an angular velocity vector on the sphere, and the angular velocity vectors, form an orthogonal basis. The angular velocity vector induced by a particular wheel, is orthogonal to the position vector of its point of contact with the sphere (see fig. 2).

Figure 2: The dotted line is the angular velocity vector induced by an omni-wheel. The solid line is the position vector of the point of contact between the omni-wheel and the sphere. Note, the angular velocity vector is orthogonal to the position vector point of contact with the omni-wheel.



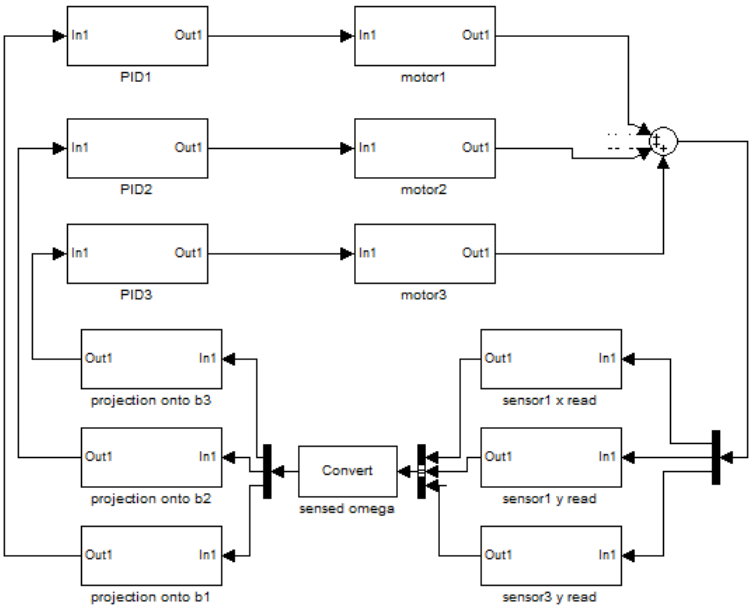
From fig. 2, we can easily define our angular velocity basis by negating the z coordinate of our wheel vectors ($\vec{p}_1, \vec{p}_2$ and $\vec{p}_3$) and normalizing.

Thus, our angular velocity basis is given by:

$$\vec{b}_1 : \left(0, \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}\right)$$
$$\vec{b}_2 : \left(\frac{\sqrt{6}}{4}, -\frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{2}\right)$$
$$\vec{b}_3 : \left(-\frac{\sqrt{6}}{4}, -\frac{\sqrt{2}}{4}, \frac{\sqrt{2}}{2}\right)$$

Our feedback consists of three PID controllers that run in parallel. After finding $\vec{\omega}$, we project it onto each of the basis vectors. If we have a desired angular velocity vector, there is a component of the vector that lies along each of our basis elements. At each iteration of our algorithm, we take the quantity $\vec{b_i} \cdot \vec{\omega}$ and see how much it differs from the desired projection along $\vec{b}_i$. If it is too high, we slow down the motor, if it is too low, we speed up the motor. The process is summarized in 3.

Figure 3: Block Diagram of feedback strategy.



3 Construction

The casing of the spherical tumbler consisted of three components: the sensor brackets, the motor brackets, and the base to which all of those components were connected with screws.

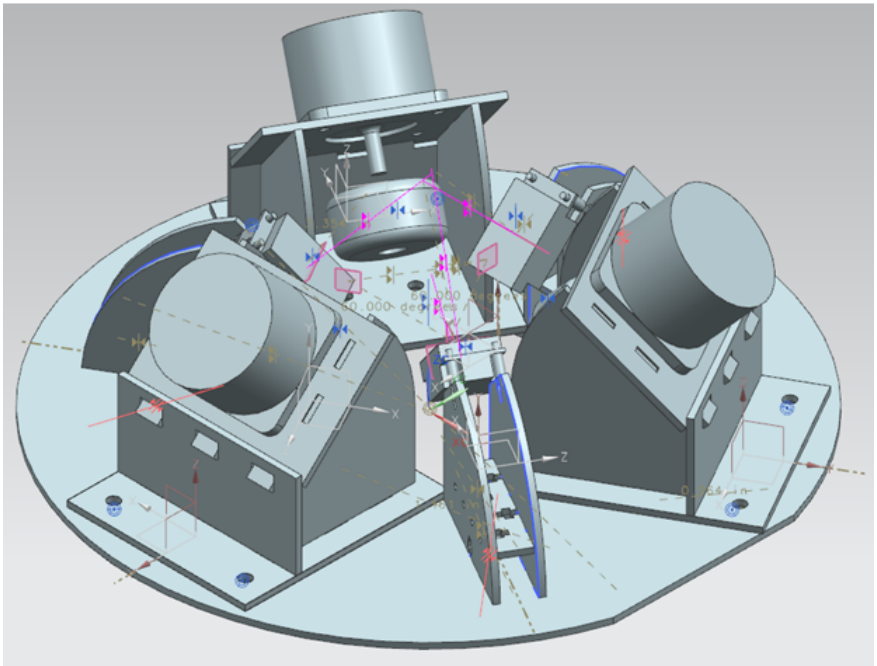
The sphere holding the granular material rested on three 48 mm Lego NXT omniwheels driven by three unipolar stepper motors that ran on 12V with 1.8 deg/step. Each motor rested on a bracket, which were placed 120 degrees from each other around the sphere and bolted to the base with screws, and the omniwheels were angled at 45 degree to the sphere. (Figure 1)

The wheels had an inner diameter of 0.2325 inches while the shaft of the motor was 0.5 inches, so an aluminum shaft had to be manufactured to create a press fit between the two components and keep the wheel 2 inches from the surface of the motor. The motor brackets were constructed from laser cut acrylic because they were more rigid and able to support the heavier motors. (Appendix A)

The sensors were angled at 45 degrees from the bottom of the sphere and mounted on brackets made out of 0.125 inch thick wood, which is not as stiff as acrylic but is less dense. The mouse sensors were attached to the vertical pieces of the bracket with 7/32 inch long nylon threaded standoff screws. A rectangular piece was screwed between the two sides to stabilize the bracket. (Appendix B)

The brackets were connected to a 12" diameter wooden base that is 0.125" thick. The holes for the bolts and slits were laser cut, and the entire structure was also mounted on a 1.74 inch thick wooden block to stabilize it when the motors are running. A flat end was included on the base so that the user could better orient the piece. A hole was cut out from the middle of the base for wires connecting the motors and sensors to the circuit board, hiding the electric components that the user would not typically interact with. (Appendix C)

Figure 4: Setup of tumbler with motors, sensors, and brackets. This does not include the thick piece of wood everything rested on.



4 Motors

The spherical tumbler is driven by three VEXTA PH264-01 stepper motors. These motors are all controlled by pins B4-15 on a single NU32 PIC and powered by 6V and 3A.

The stepper motors are controlled by three interrupts on timers 3, 4, and 5 of the NU32 PIC. Each interrupt "checks" which step the motor is on, and then proceeds to the next step as outlined in the table below, depending on which direction the motor is running.

Table 1: Forward Stepping Sequence

	Coil 1	Coil 2
Step 1	H	L
Step 2	H	H
Step 3	L	H
Step 4	L	L

The h-bridges are enabled by 3.3V and 5V from the NU32 board, and connected to 6V power supply for the motors. To allow for an excess of current for one L293d, two h-bridges are stacked on top of each other for each motor. The inputs of the h-bridge align to the digital outputs from the PIC, i.e. pins B4 through B7 on the PIC correspond to Inputs 1 through 4 on one h-bridge, and similarly with pins B8 – B11 and B12 – B15 for the other two. Outputs 1 through 4 on the h-bridge connect to each end of the two coils within the stepper motor. Outputs 1 and 2 control the first coil, and 3 and 4 control the second.

4.1 Power

Each motor requires about one amp to run smoothly under the weight of the sphere and sample, and so the motors are together powered by 6V and 3A. The power comes from a 12V, 3A wall power supply through an LM317-N variable voltage regulator. A voltage regulator must be used to divide the voltage because of the large current necessary to drive all three motors.

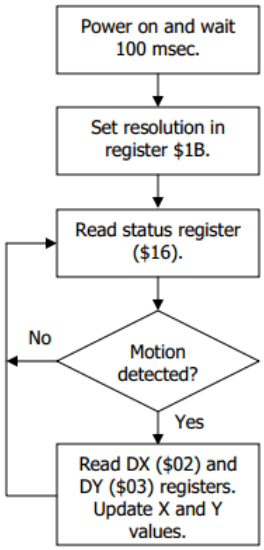
5 Sensing

5.1 The Basics

Our sensing consists of two optical mouse sensors. The sensors are essentially tiny cameras that take pictures of a surface and two different time instances, and compare the differences between them to sense the most likely displacement of movement. While they are very good at sensing relative position, they accumulate small errors in their readings and over time give rise to errors in absolute positioning.

Communication with the sensors uses an 8 bit SPI-like protocol. The master device (the PIC32), can either read or write to the sensor's registers. A write operation is usually done to configure settings (such as resolutions, power down, reset etc..). A read operation is done to receive data from the chip (such as x displacement, y displacement, image quality, movement occurrence etc..). If the master wants to write, it first sends over the address it wants to write to with a 1 as the most significant bit of the word. It then waits 100 us and sends over the data it wants to write. If the master wants to read, it sends over the address it wants to read from, with a 0 as it's most significant bit. It waits 100 us, in order to give the device enough time to prepare the requested data and then turns over control of the data line to the sensor and turns it's input to a hi-Z state. The clock is always controlled by the PIC32. A typical flow chart for operating the chip is shown below:

Figure 5: flow chart for operating sensors



Register information and timing diagrams can be found in this document: http://www.parallax.com/Porta MouseSensorKit-v1.0.pdf so they will be omitted from this document. Instead, we will focus on aspects of operation relevant to interfacing with the PIC32.

So the two major challenges with communication with the sensors were : 1.) clock

polarity, and 2.) configuring the PIC32 to work with the nonstandard SPI protocol of the sensor.

5.2 Clock Polarity

This was the simpler of the two problems to fix. Before talking about clock polarity, it is important to note that the max clock frequency we can use for operation with the sensor is 2 MHz, we used a frequency somewhere around 1 Mhz. The issue here is that by default, the SPI clock on the PIC32 is low when not operating, and starts on a rising edge. The rising edge is ok, but we want the clock to remain high by default. In order to accomplish this, we need to set the set the CKP bit in the SPIxCON register. CKP needs to be 1 and CKE needs to be 0. This configures the clock to operate in the correct mode. Pretty simple.

5.3 The Non-standard SPI Protocol

Traditionally, SPI is a two wire protocol: the master device transmits on one wire, and receives on the other–usually at the same time. However, on most mouse sensor chips, there is only one data line that sends and receives information. Since the PIC32 likes to send and receive data at the same time, there will always be a collision on the data line unless we do something to non-standard. The process for solving this problem involved disabling the output function and tri-stating the port on the SPI peripheral between send and receive operations. The full process is detailed below:

1) enable SDO by clearing DISSDO
2) write address byte to SPIxBUF
3) wait for SPI send to complete by polling SPIRBF
4) read SPIxBUF to empty receive buffer
5) tri-state SDO by setting DISSDO and TRISx bit
6) wait 100us for sensor to prepare data
7) write dummy data byte to SPIxBUF
8) wait for SPI read to complete by polling SPIRBF
9) read data byte from SPIxBUF

6 MATLAB interface

The MATLAB function ME433_v1 asks for the user to input the angle of rotation along the first axis, the angle of rotation along the second axis, and the angle of repose. In the current setup, it converts the angle to the numbers of steps the motors must take, searches for an available communication port, and sends the values to the PIC32. puTTY is required to run it.

7 Limitations and Future Considerations

The sensors used for this project are very fragile. They operate on strict power constraints and even a momentary misplacement of a wire can render them nonoperational. Currently they need to be replaced. They also give good readings of relative positioning, but in order to get absolute positioning, multiple sensors and redundancy is needed. The system response time is possibly limited by the speed of the stepper motors. Initial testing with the PID controllers lead to jerky motion, it is unclear whether this was due to the response time of the motors or whether this was due to a poorly tuned controller. Regardless, the sensors failed before we had the opportunity to determine the cause. Future designs might benefit from the use of dc motors to increase the range of operation speeds. The original goal of the stepper motors was to attain absolute positioning if we could minimize slip enough. However the slip seemed to be significant enough that there is minimal benefit from using stepper motors over dc motors with encoders. Currently, the max speed we can generate comes from an interrupt with the period register set to 1000 and the pre-scalar at 256 (approx 300 Hz). Manufacturing wise, the brackets may have to be redesigned to make them more compact, and placing the screw holes in a formation so that the entire system does not vibrate when operating. There should also be a cavity on the bottom to hold the circuit board within the construction.

8 Appendices

Figure 6: Bracket

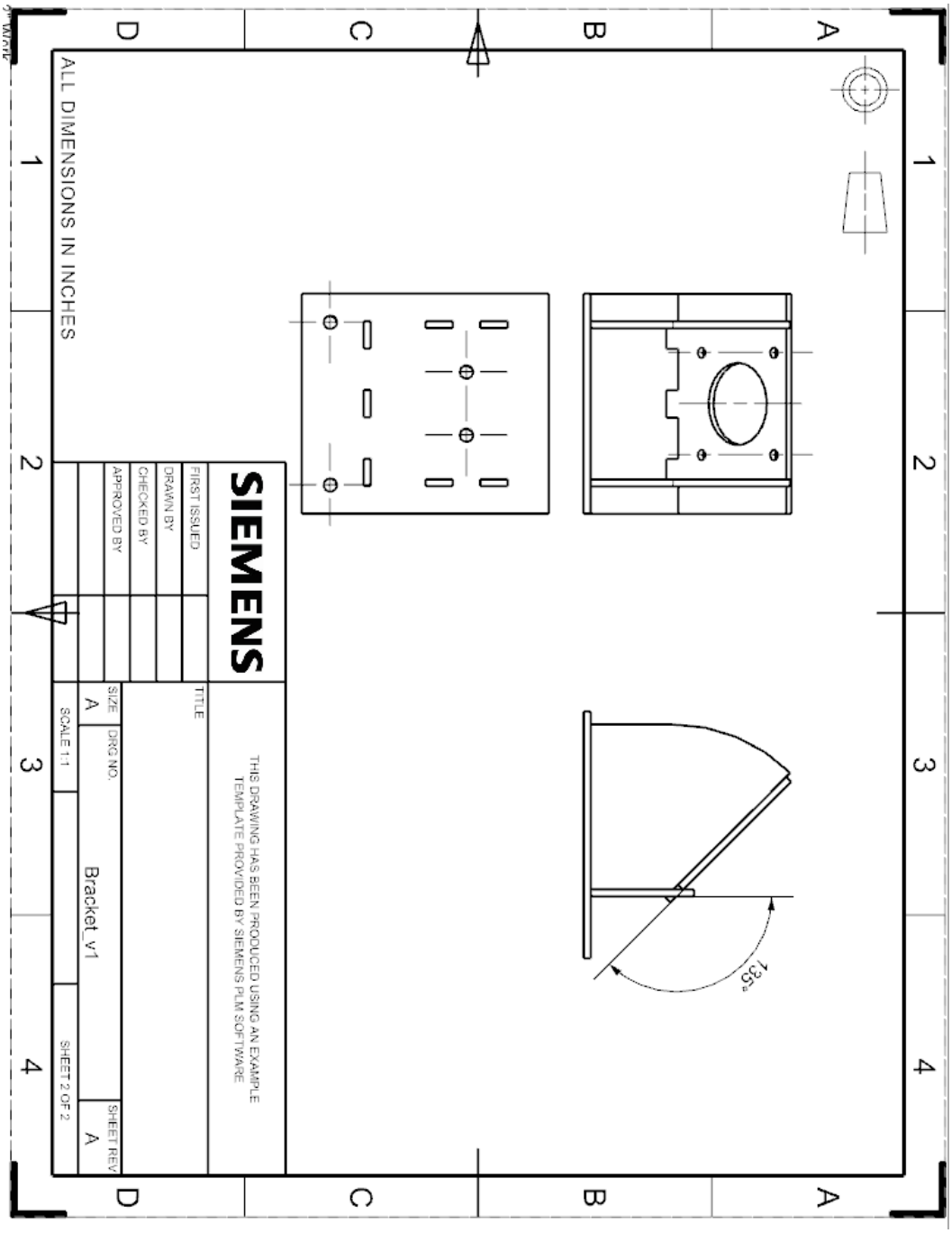


Figure 7: Bracket side panel

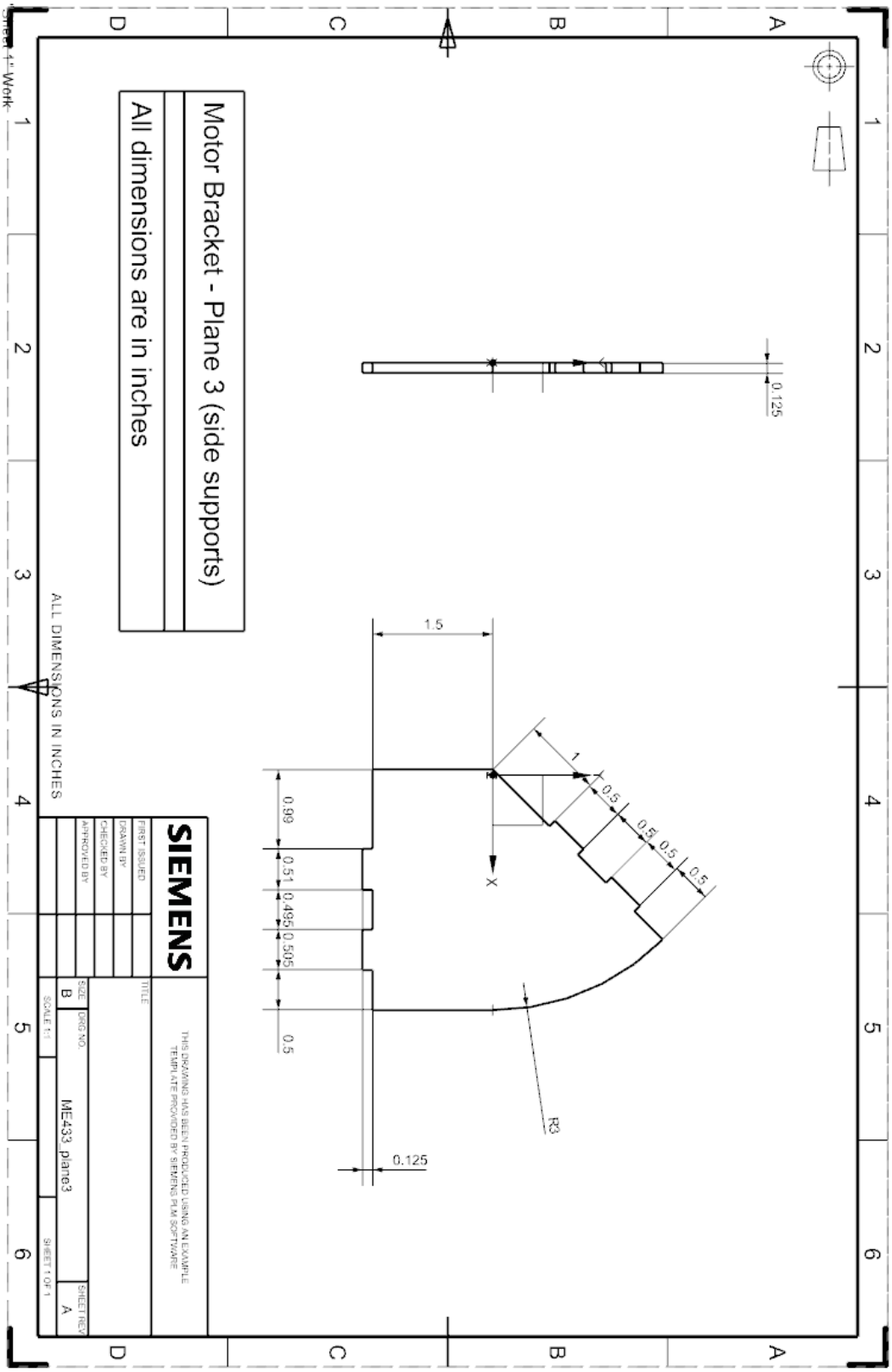


Figure 8: Bracket motor mount

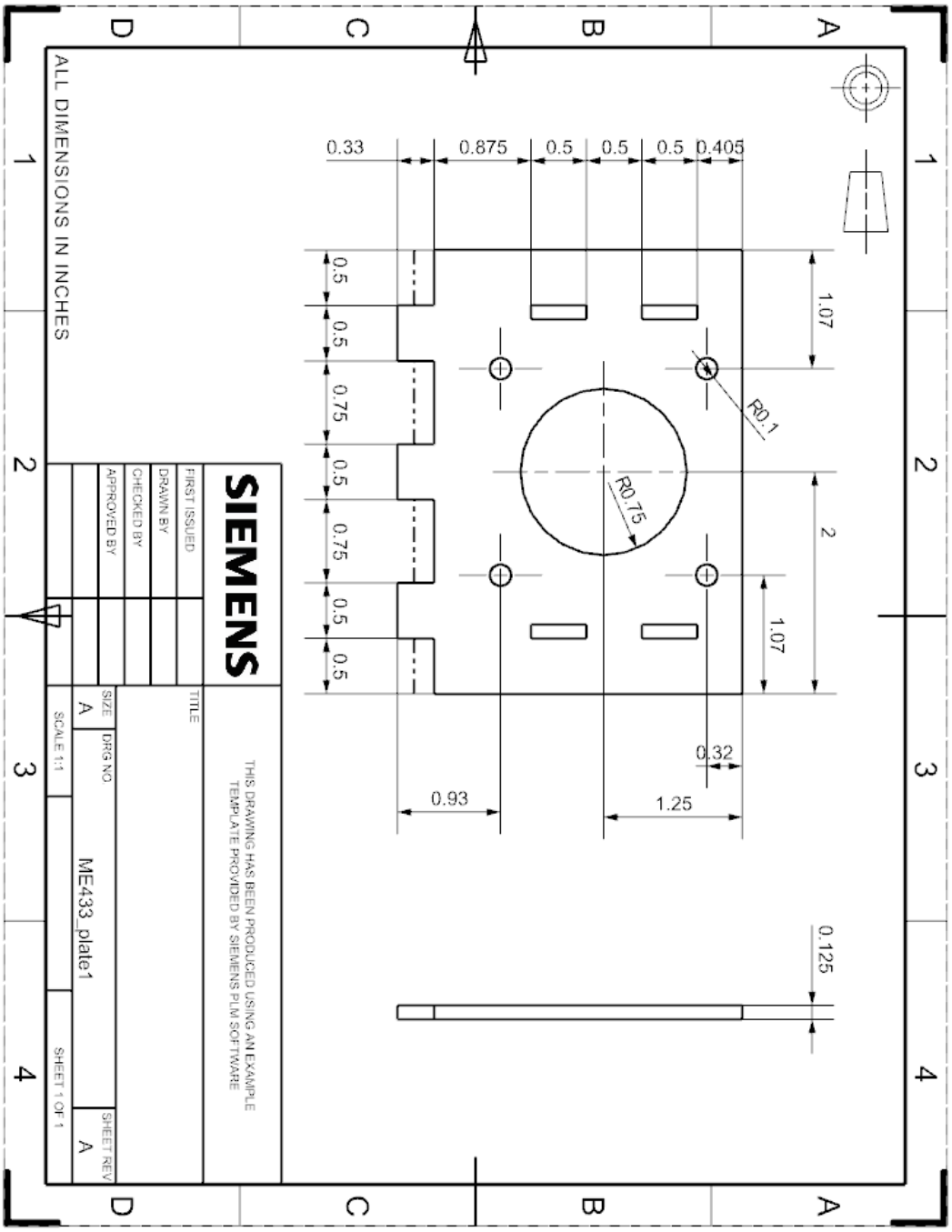


Figure 9: Bracket back panel

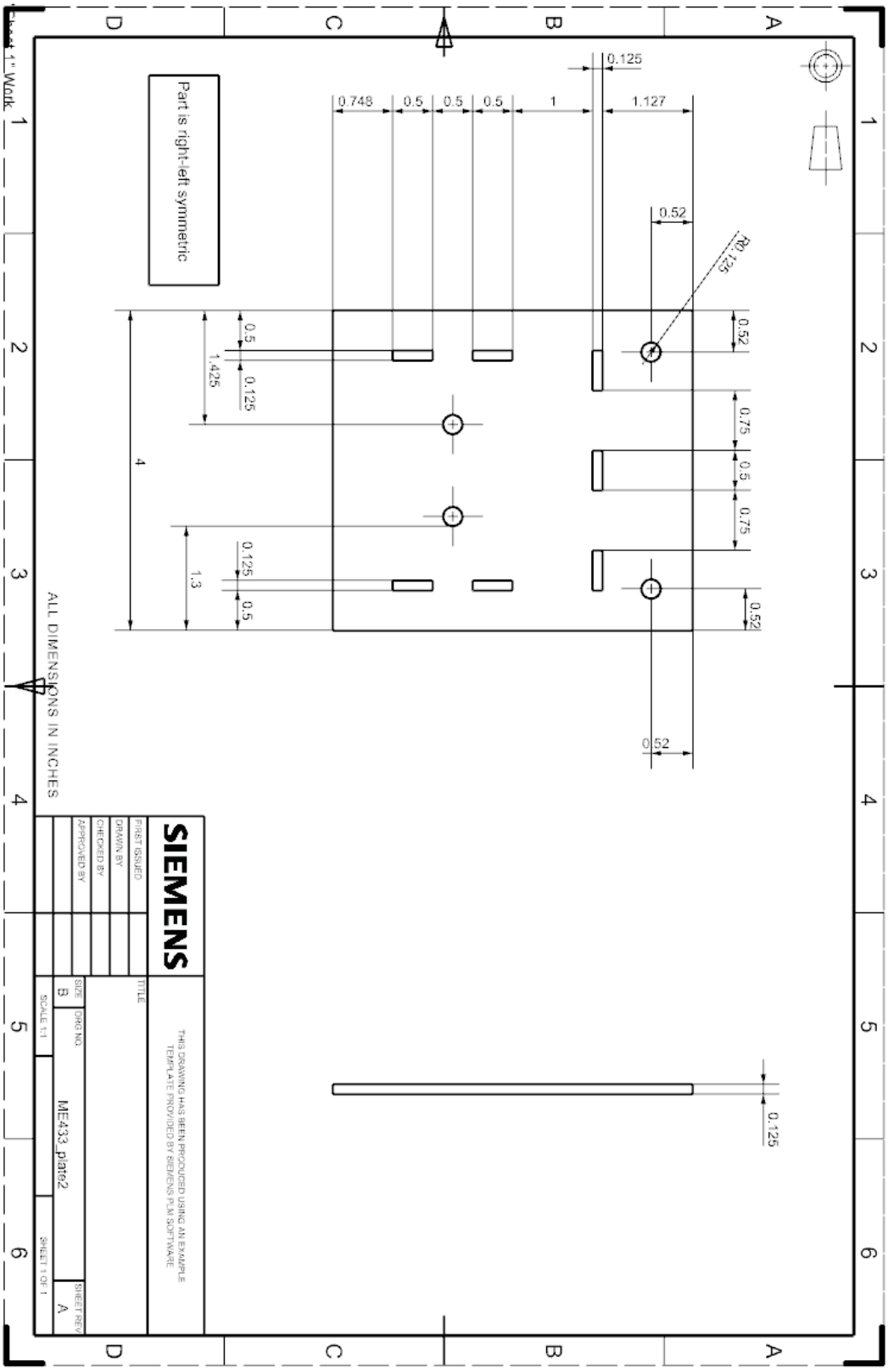


Figure 10: Bracket front panel

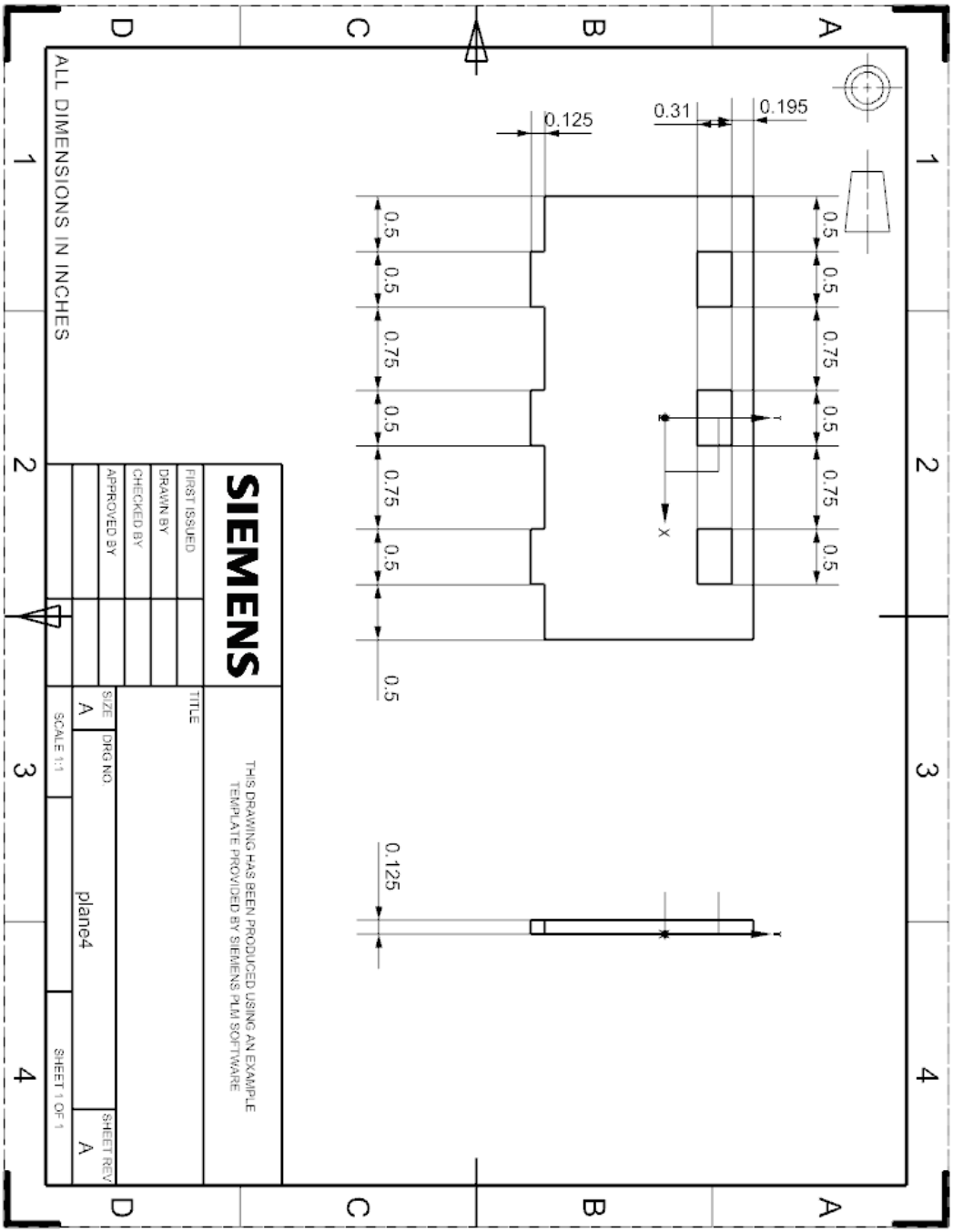


Figure 11: Sensor mount

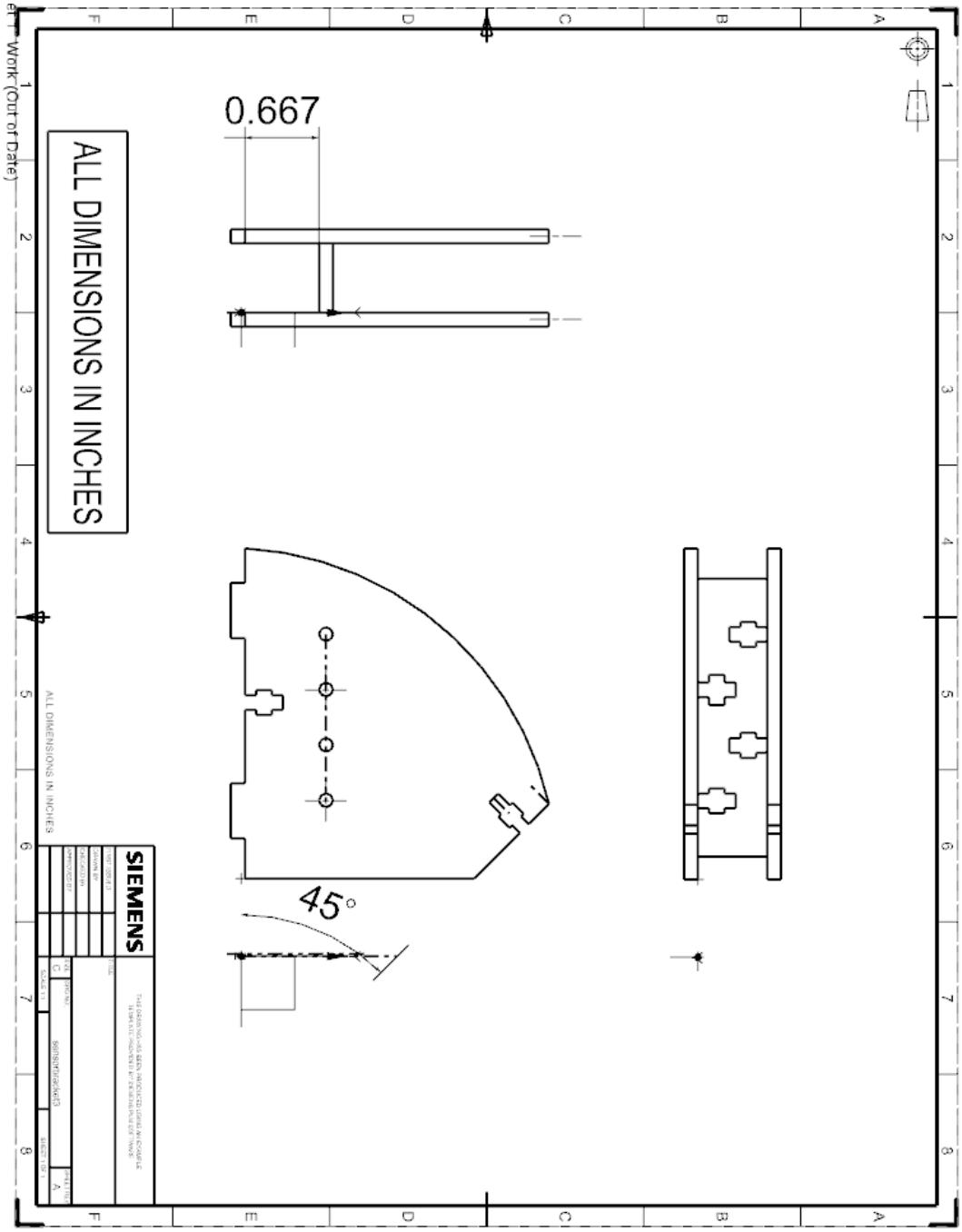


Figure 12: Sensor mount, side panel

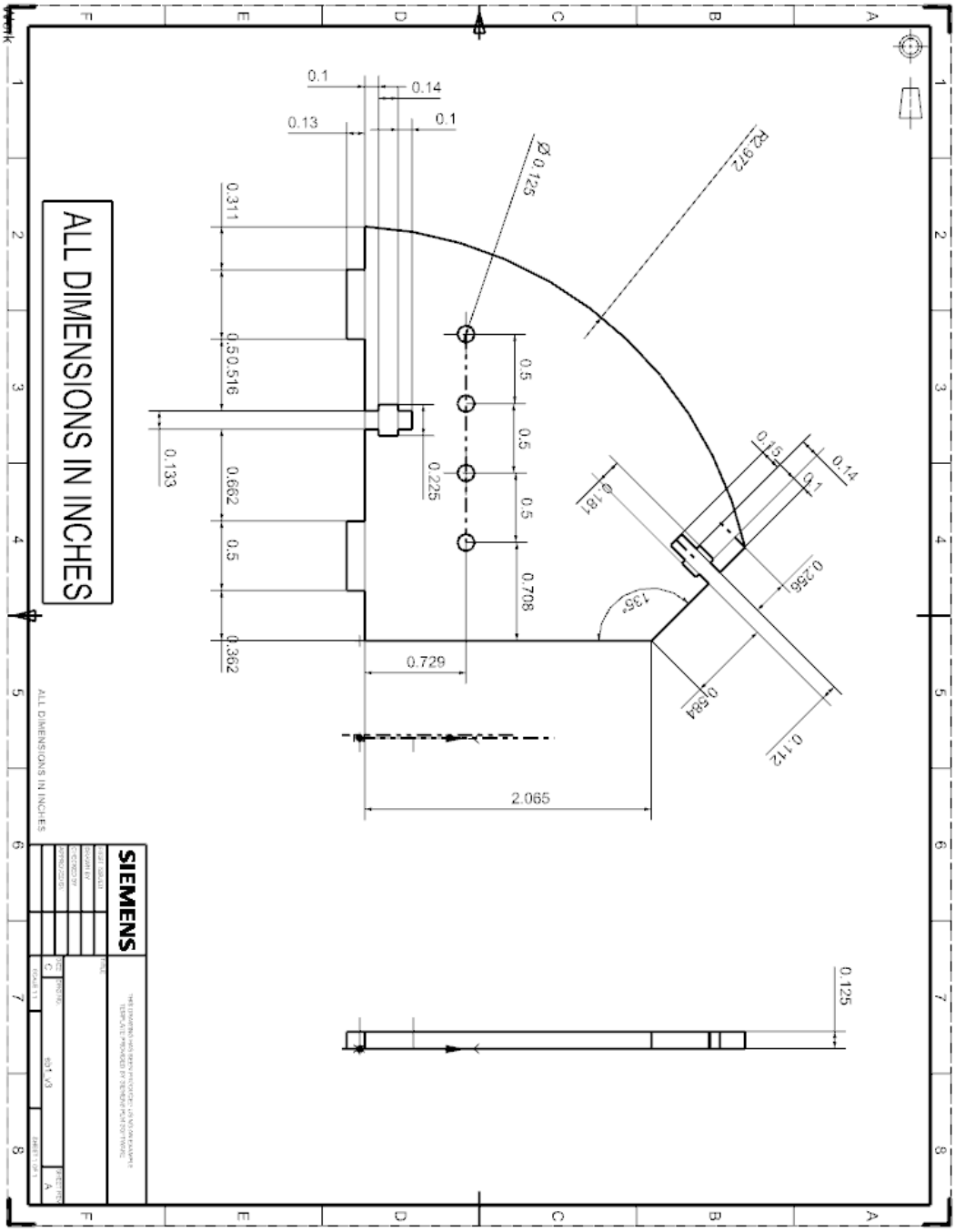


Figure 13: sensor mounts, spacer

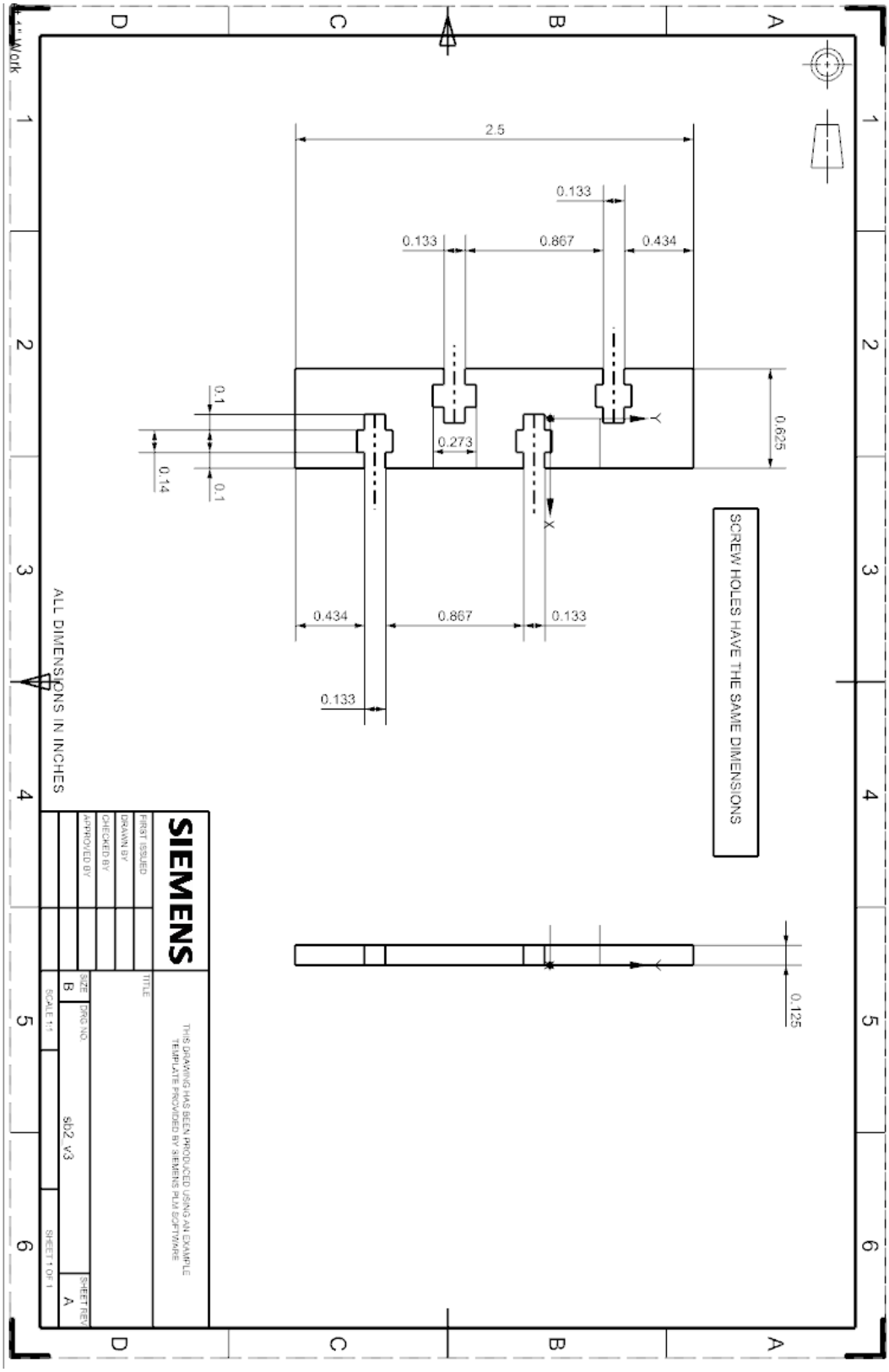


Figure 14: Base Top View

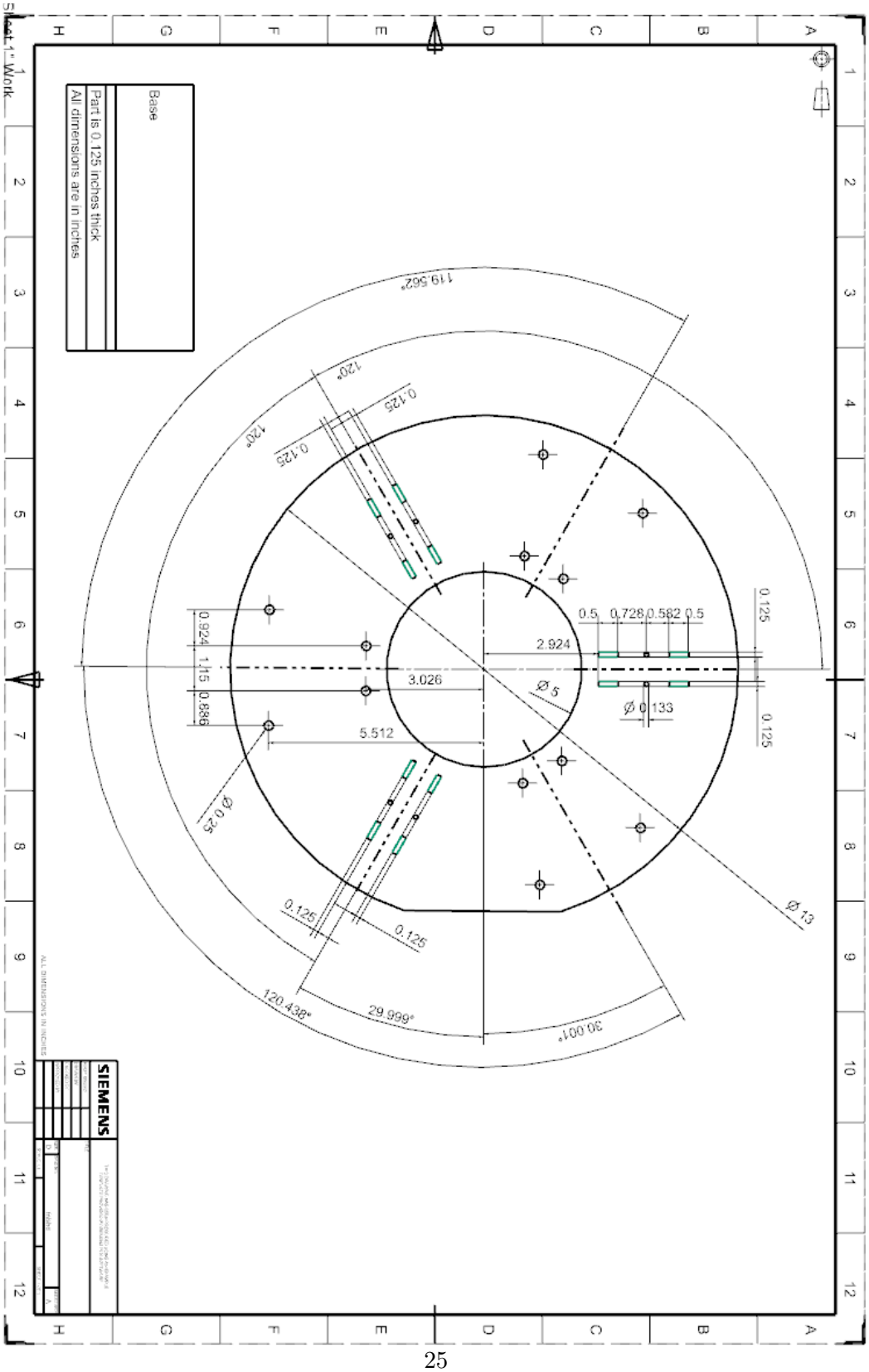


Figure 15: Circuitry (A note, we cut the lines between SDI3/SDO3 since they interfere with the UART

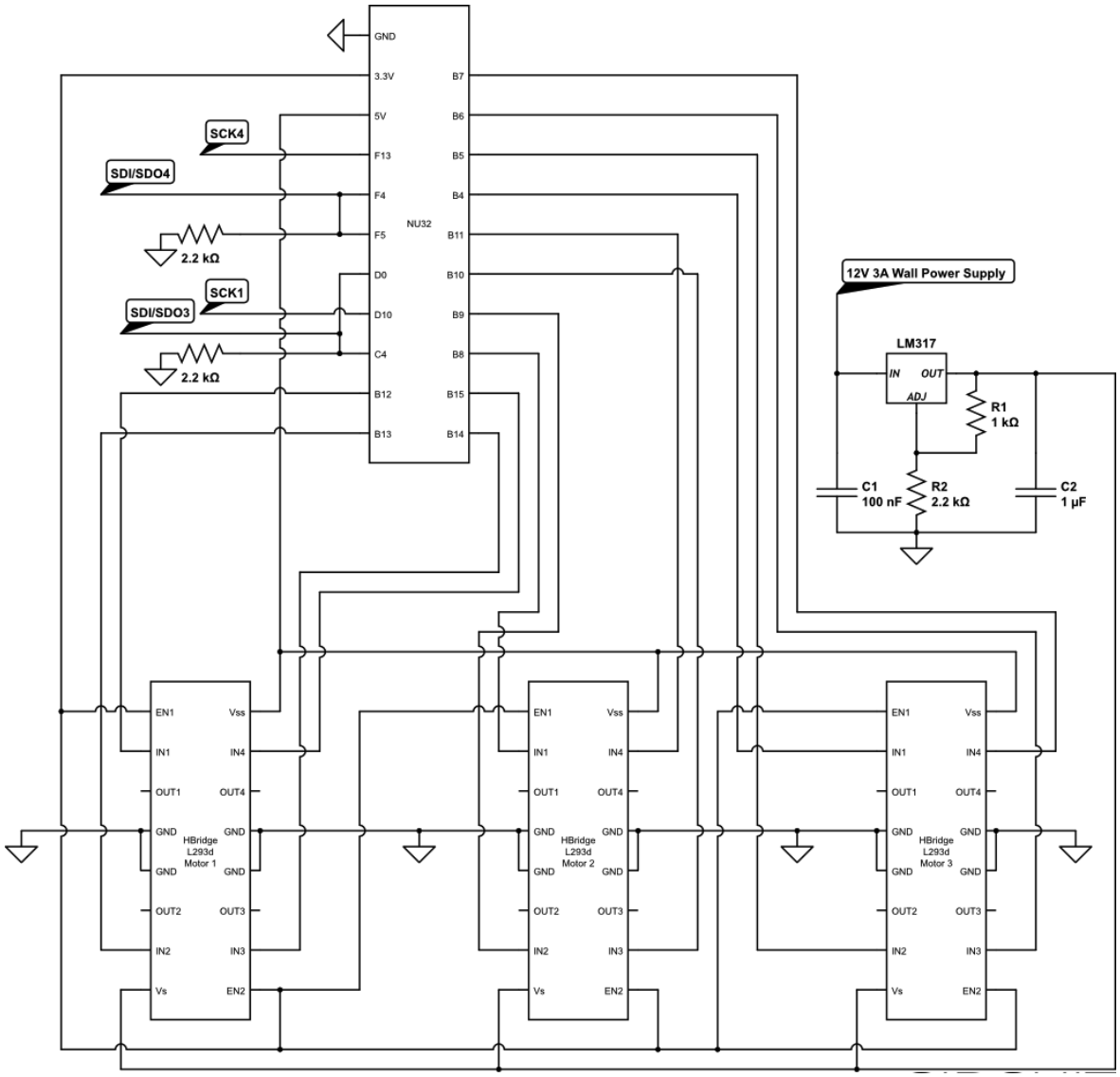


Figure 16: Sensor Circuitry

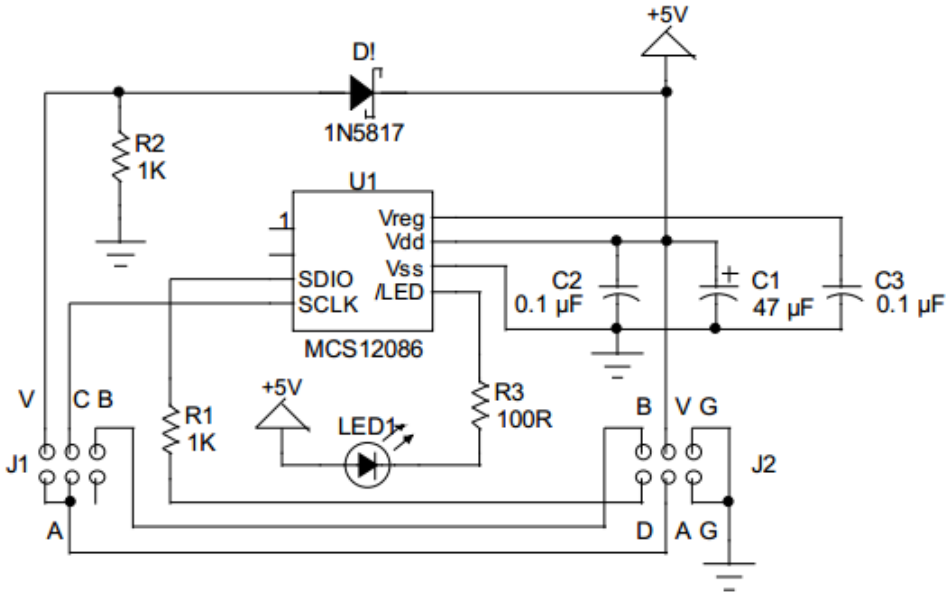


Figure 17: PCB screenshot

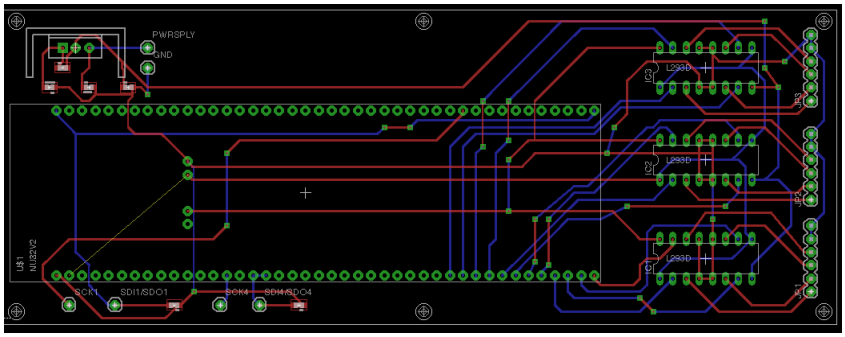


Figure 18: PCB picture

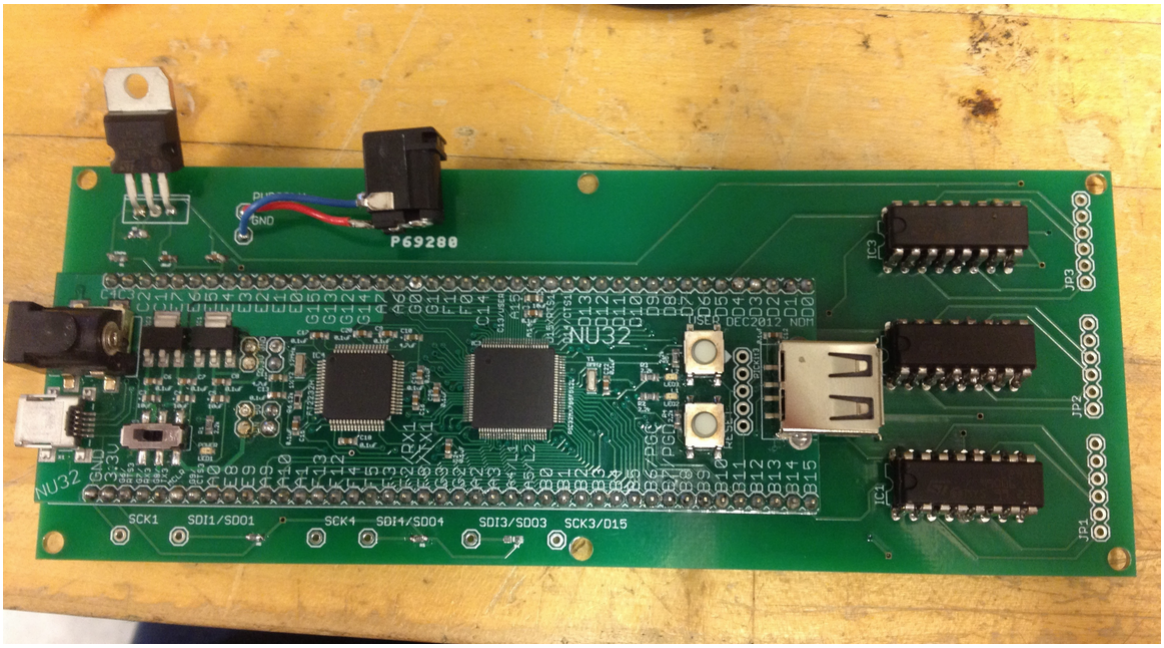


Figure 19: Full Setup