ELE 302 Car Lab

Ballerina: Ball Balancing Plate

May 15, 2015

Jenny Sun | Allie Wu

Contents

1 Introduction

The goal of this project was to design a plate system capable of balancing a ball in the presence of environmental disturbances of a reasonable magnitude. We accomplished this goal, as shown in our demo, and were further able to trace simple pre-programmed shapes using the plate (further fine tuning of feedback parameters would have provided more accurate tracing, though our parameters during demo demonstrated tracing enough to confirm that the concept was capable of working).

Upon being disturbed from rest, our ball typically converged back to the center of the board after three or four oscillations. While the time for convergence was not as fast as seen in some of the online videos which inspired us, it seems that our number of oscillations was actually smaller. Our slower convergence time stems from our bottlenecks of having slow PWM pulse changes, which are limited by the speed of software, and slow servos, but our PID control was robust enough to make up for these deficiencies, which will be expanded on in Section 4.

2 Hardware Design

Initially, we were hoping that the hardware for the act of balancing was readily sold online. Of course, they are available, but in the form of hexapods, for the purpose of fiber alignment, have slow feedback and are absolutely outside of our project budget. Thus, it was necessary for us to built an entire hardware system on our own. We used Autodkes Inventor 2015 for all of our CADs and went through many system designs and versions within each design. At one point, we ended up with a three-servo-driven delta robot that offers precision positioning in x, y and z but no row and pitch. Eventually, we settled on a simple 2 servo design that serves the purpose of balancing.

2.1 Autodesk Inventor 2015

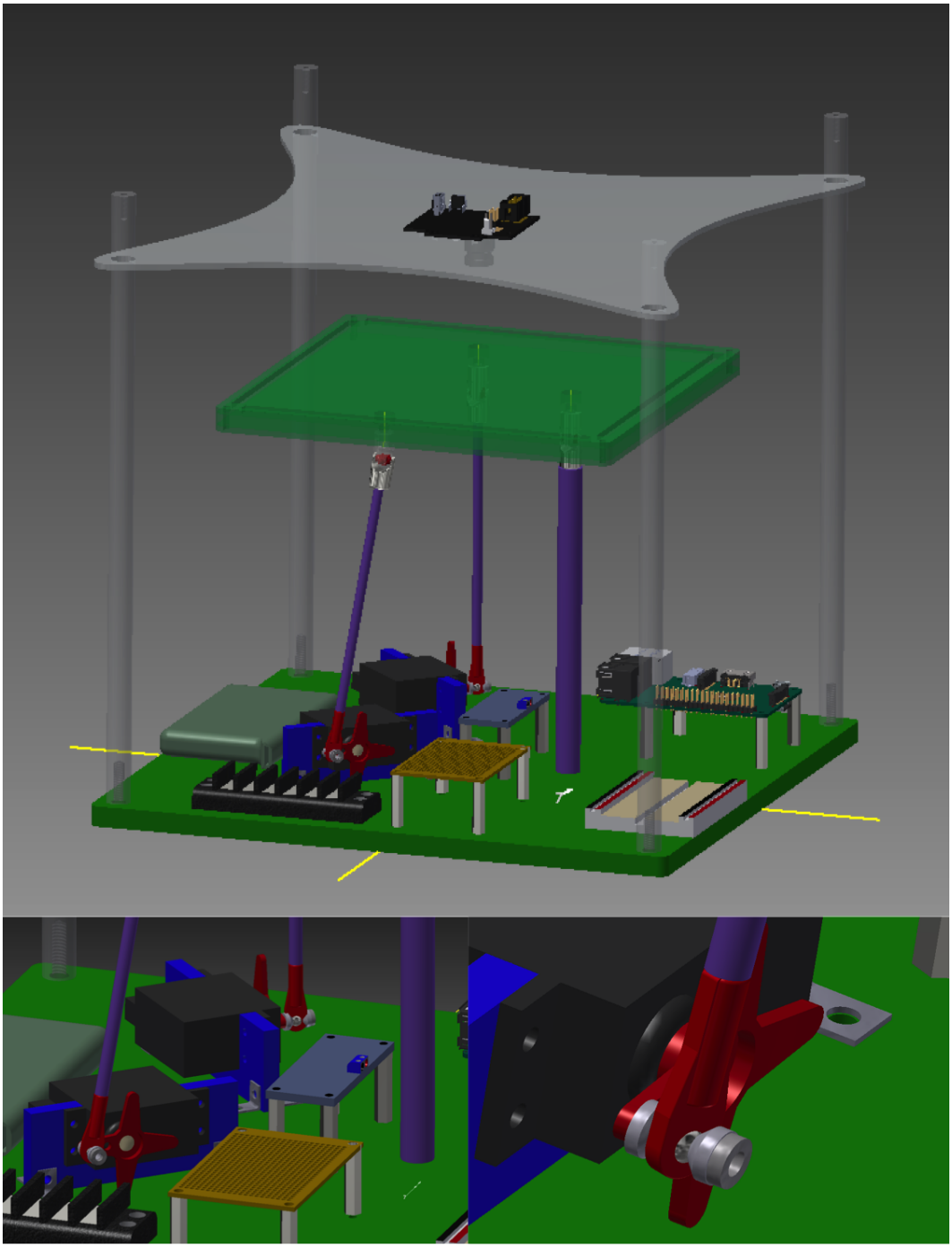


Figure 1: CAD simulation photo of our final design.

The previous figures shows the final CAD of Project Ballerina. After many days and many versions of system design, this is the final design we cut, machined and assembled for the independent portion of our Car Lab project, which would balance a small ball bearing. All of the CADs for Project Ballerina were produced by Autodesk Inventor 2015, which we're eternally in love with. However, using Autodesk Inventor 2015 posed varies challenges along the way.

First of all, the program demanded Win 7 operating system and a large amount of RAM, which meant the computer crashed a lot, especially in the late stages of CAD assembly. Secondly, CADing demanded highly accurate measurements, which we managed to acquire using a digital caliber that we borrowed from the TA. We believe it would be a worthy investment to make for future takers of Car Lab. Thirdly, we struggled a little bit with reconciling the metric units (mm) we used for our CADs and the non-SI units that governed all hardware bits we could get our hands on (such as screws, washers etc.). Thankfully, the digital caliber could switch between mm and inches. Again, we believe it's a worthy investment.

In the next couple of sections, we will detail all the steps from CADing parts, to CADing assemblies to finally cutting/machining/assembly in order to establish our hardware system for the independent portion of our Car Lab project.

2.2 CAD: Parts

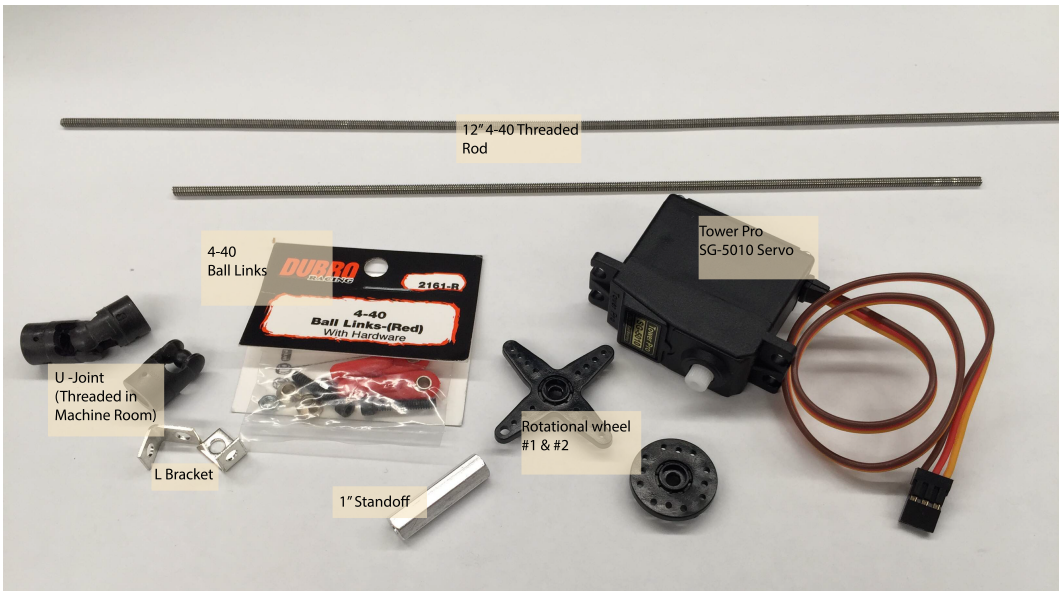


Figure 2: Photo of real hardware components.

All CADs must start with parts, which are accurately modeled after what they are in real life. The parts could be categorized into three groups. The first group (A-R) being those that

are modeled after existing hardware parts, some of which are pictured above. Eventually, they will serve as contact sets for realistic simulations in CAD assemblies or serve as place holders for allotting space. (Pixy Cam and Raspberry PI CADs are readily available online, thankfully. In addition, McMaster-Carr provided CADs A-C, however, without useful assembly).

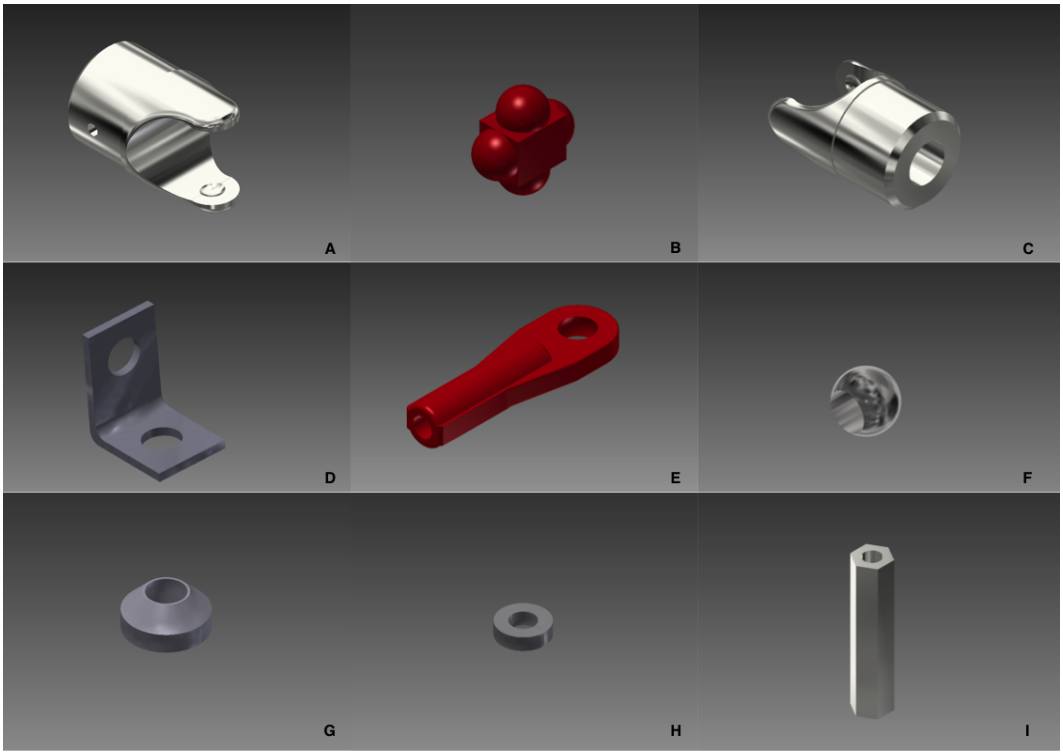


Figure 3: CAD simulation photo of parts A-I.

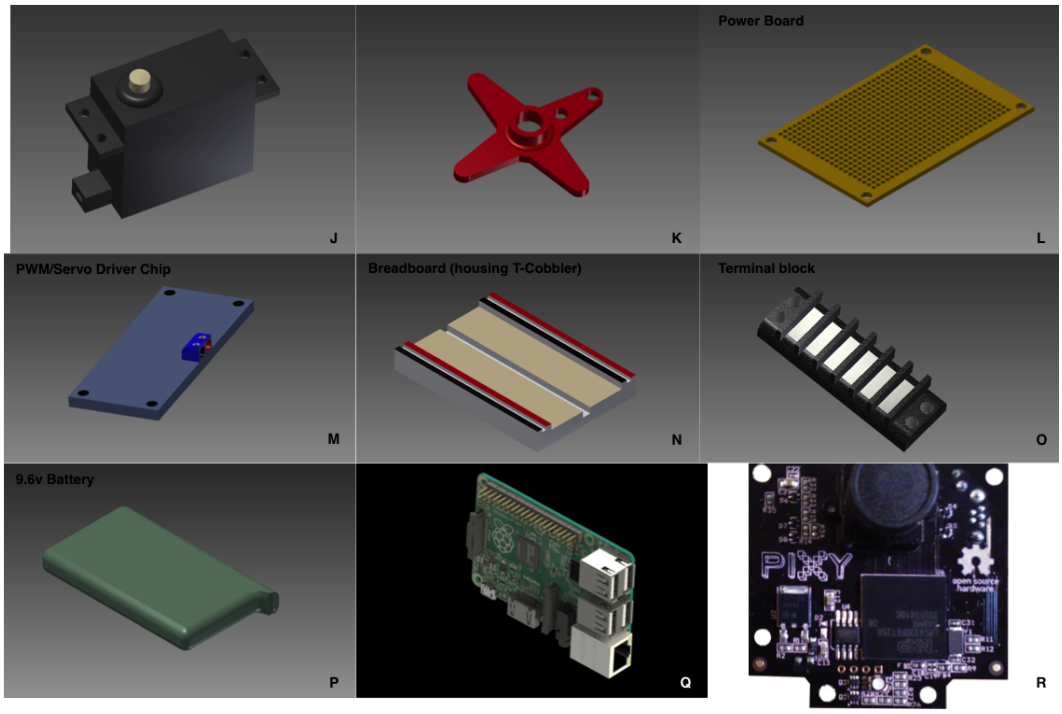


Figure 4: CAD simulation photo of parts J-R.

In addition, accuracy on their dimensions is critical to the making of the second group (S-X) of CADs, which are platform CADs. They are eventually exported as dxf files and cut out of acrylic plates of various thickness. The third group (Y & Z) is the mechanical group, which includes important bits that will contribute to the overall movement and attachment of various parts of the system belonging to both group 1 and 2. The images in this section are only Autodesk Inventor renderings. Please check out the Appendix B for dimensional drawings of all our CAD parts, which will come together as assemblies in the next section and be listed in parts tables.

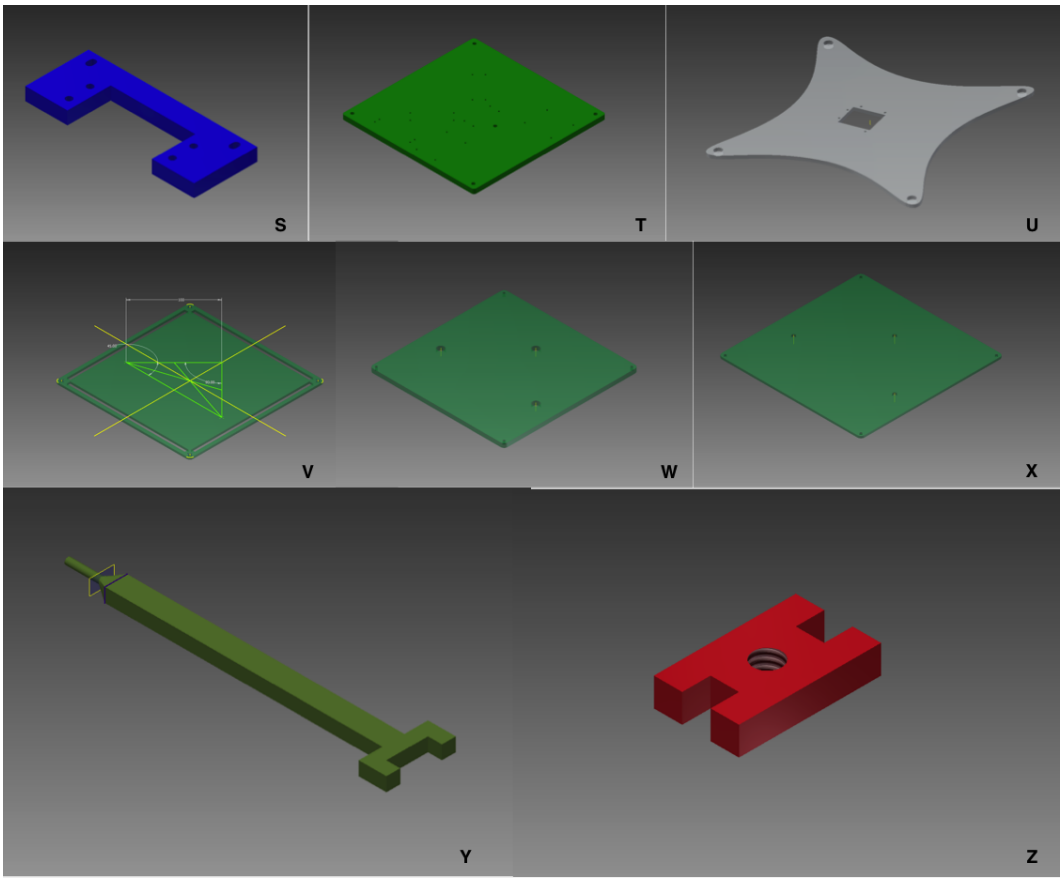
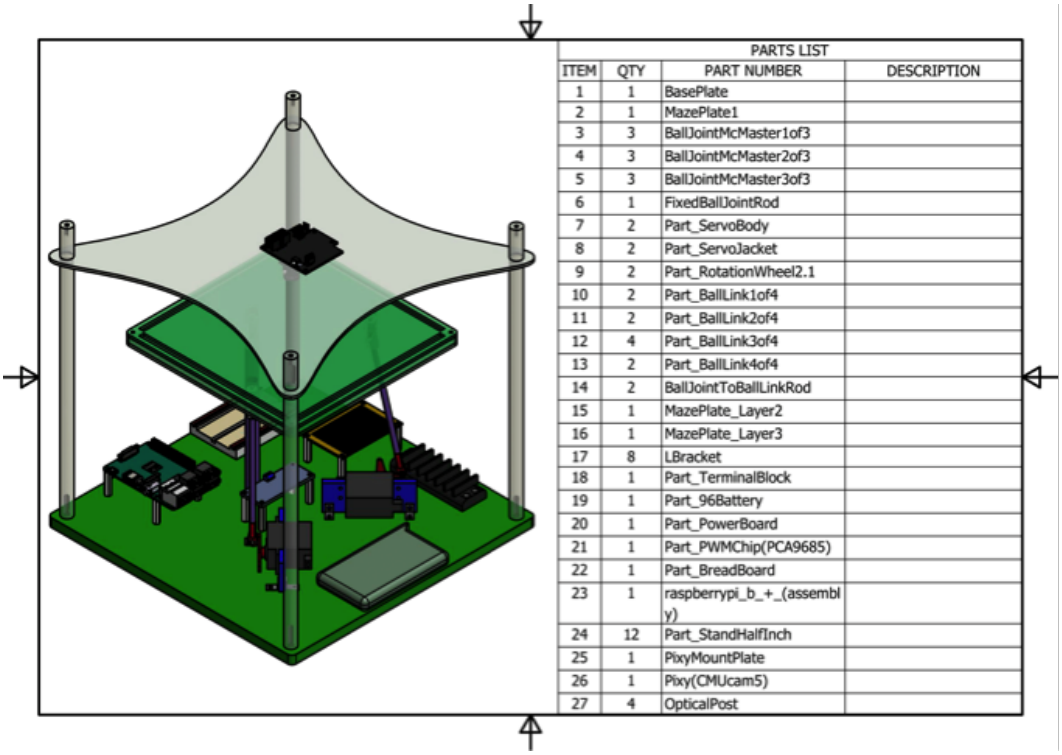


Figure 5: CAD simulation photo of parts S-Z.

2.3 CAD: Assemblies



PARTS LIST

ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	BasePlate	
2	1	MazePlate1	
3	3	BallJointMcMaster1of3	
4	3	BallJointMcMaster2of3	
5	3	BallJointMcMaster3of3	
6	1	FixedBallJointRod	
7	2	Part_ServoBody	
8	2	Part_ServoJacket	
9	2	Part_RotationWheel2.1	
10	2	Part_BallLink1of4	
11	2	Part_BallLink2of4	
12	4	Part_BallLink3of4	
13	2	Part_BallLink4of4	
14	2	BallJointToBallLinkRod	
15	1	MazePlate_Layer2	
16	1	MazePlate_Layer3	
17	8	LBracket	
18	1	Part_TerminalBlock	
19	1	Part_96Battery	
20	1	Part_PowerBoard	
21	1	Part_PWMChip(PCA9685)	
22	1	Part_BreadBoard	
23	1	raspberrypi_b_+_(assembly)	
24	12	Part_StandHalfInch	
25	1	PixyMountPlate	
26	1	Pixy(CMUcam5)	
27	4	OpticalPost	

Figure 6: Full CAD and parts list of our final design.

After all of the CAD parts are completed to their most accurate dimensions, assembly began and we began to encounter some design decisions. We knew that the Pixy Cam would need to be directly overlooking the ball and its balancing plate, so we matched the z-axis of the Pixy's lens with the z-axis of the balancing plates. Then we created the Pixy mounting board with holes in positions that matched the actual Pixy. In addition, we really wanted the Pixy Cam to be adjustable, so we opted for the use of optical posts that the mounting plate could slide up or down. For smaller adjustments, we used standoffs to mount the Pixy to its plate.

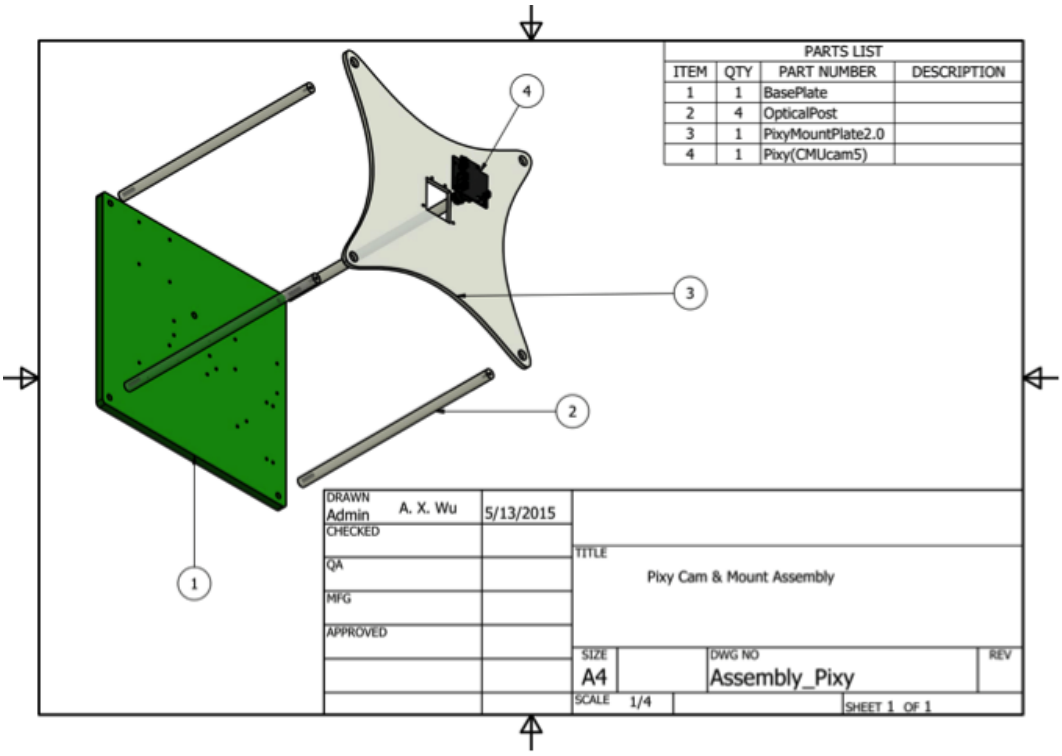


Figure 7: Assembly and parts list of Pixy camera mount.

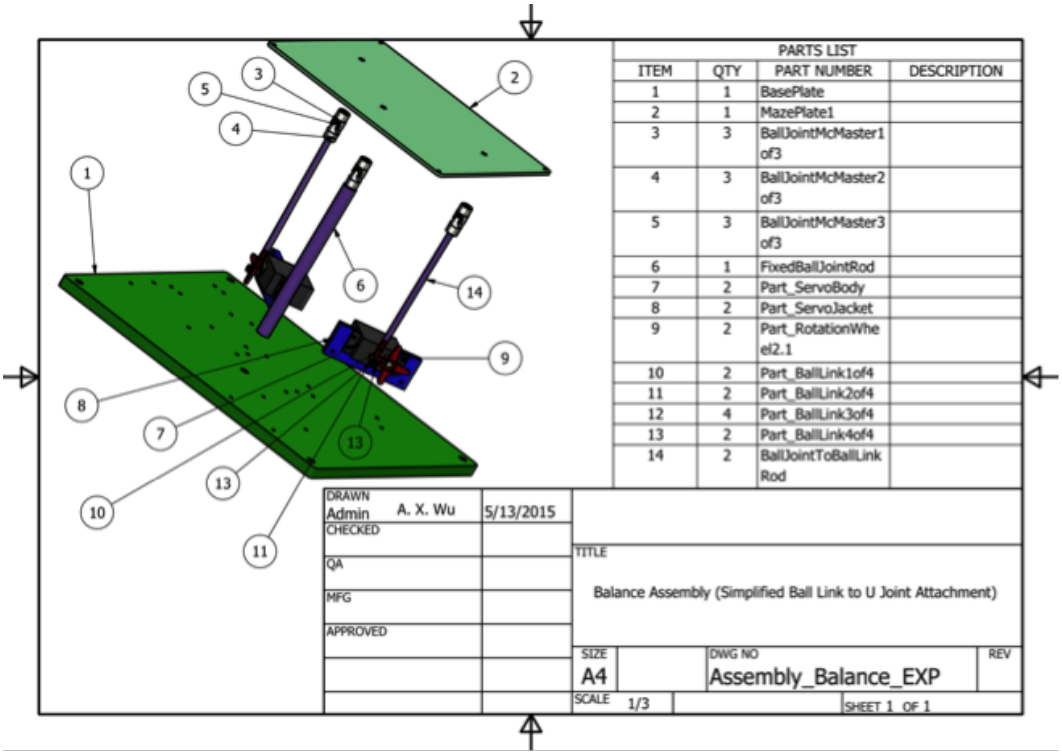


Figure 8: Assembly and parts list of balancing geometry.

We then had to decide how to position the 2 servo posts and the fixed (on one end) post. We eventually opted for a 90-45-45 triangle, whose center point matched the center point of the square balancing plate. With this setup, we get equal amount of pitch and roll of the balancing plate in all directions (as opposed to if we put the fixed post at the center of the plate and make the 2 servos parallel with 2 adjacent sides of the square). Then, we positioned the servo so that the rotational wheel of each servo is flush against the two 45° planes of the triangle. The fixed post then occupied the corner of the 90° angle. With 2 servos, we could not get independent control of the x- and y-axis, so the coupled angle of movement would have to be compensated in software; however, stability and equal change in pitch and roll is achieved with this setup.

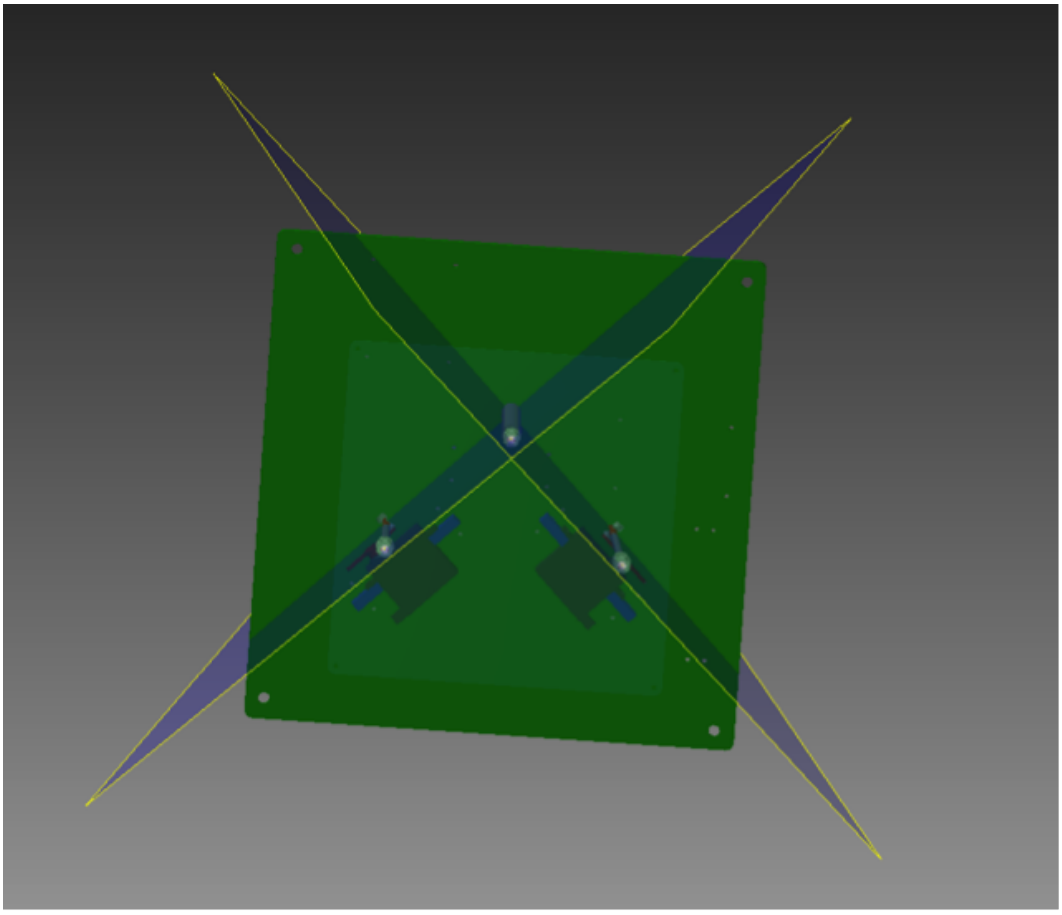


Figure 9: Bird's-eye-view of balance plate, bottom plate and 45° angle axis in relation to the position of the 3 posts.

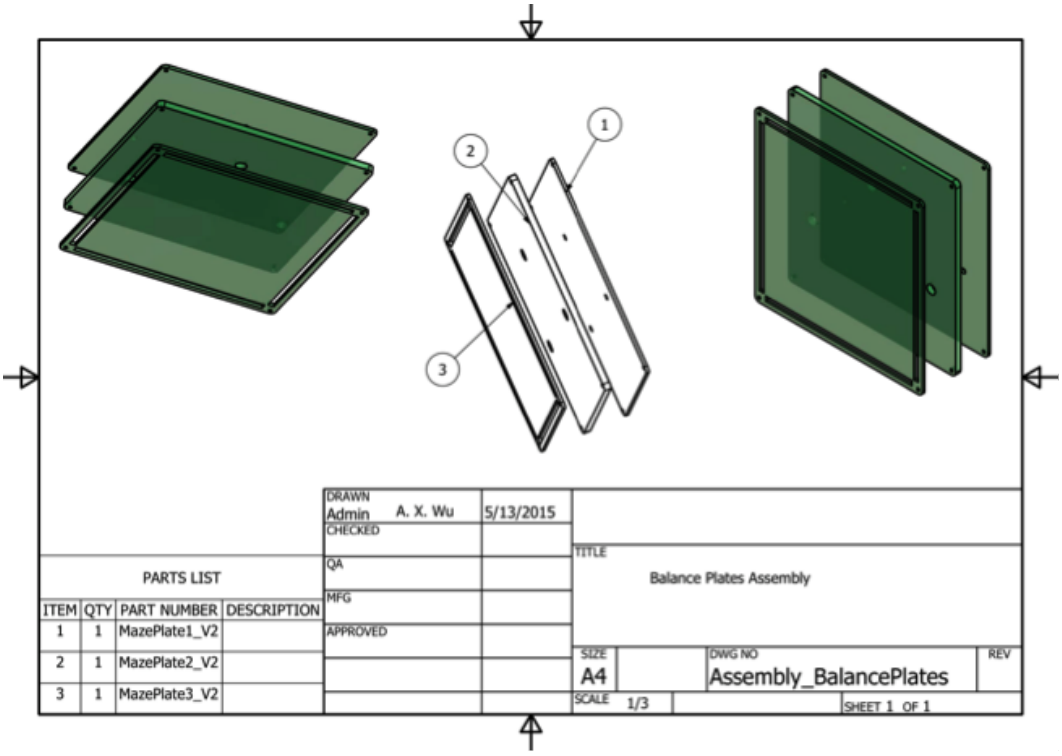


Figure 10: Assembly and parts list of layered balancing plates design.

The next design challenge we faced was how to make sure that the balancing plate connects to the u-joints while remaining a flat surface, which is quite critical to balancing. We came up with a layered design of 3 layers. The bottom layer would be thin and connects to the 3 u-joints that form the 90-45-45 triangle with the use of screws. The middle layer would be very thick and features 3 wide holes (10 mm) that would accommodate and house the screw tops that protrude on the bottom plate. Then, the thin top layer would then be able to sit flush again the middle plate. Screws through holes at the four corners of each plate hold the 3 plates together. Lastly, the top plate also contains grooves on all 4 sides to catch the ball when it reaches the edge, saving us the trouble of running after balls all the time.

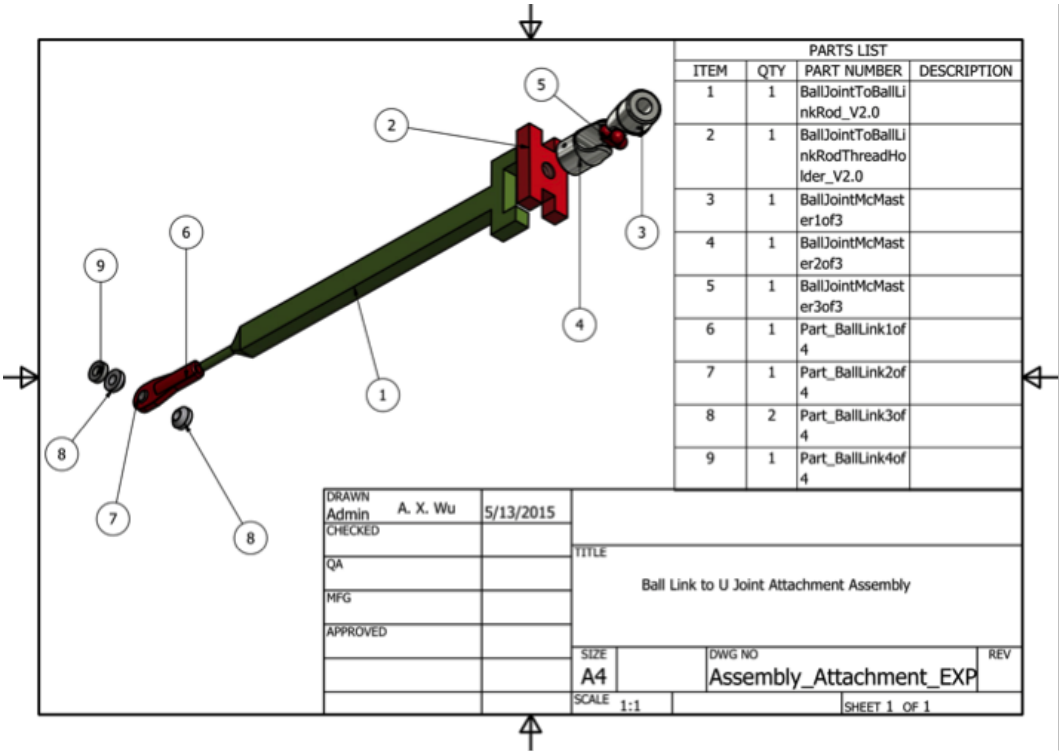


Figure 11: Assembly and parts list of attachment mechanics.

Our next challenge is how to actually connect the ball link joints to the u joint, since we have no time to order or make rods that would meet both ends of dimensional needs. We then came up with 2 mechanical parts (see Section 2.2) that would fit into the 4-40 hole of the ball link joint at one end and also be able to hold onto u-joint with a setscrew of a much larger dimension at the other end. These two parts would then fit together snugly and be hot glued together in the end. The next image shows the full connection from servo, to rotational wheel, to ball link parts, to attachment rod, to H thread holder and finally to u joint.

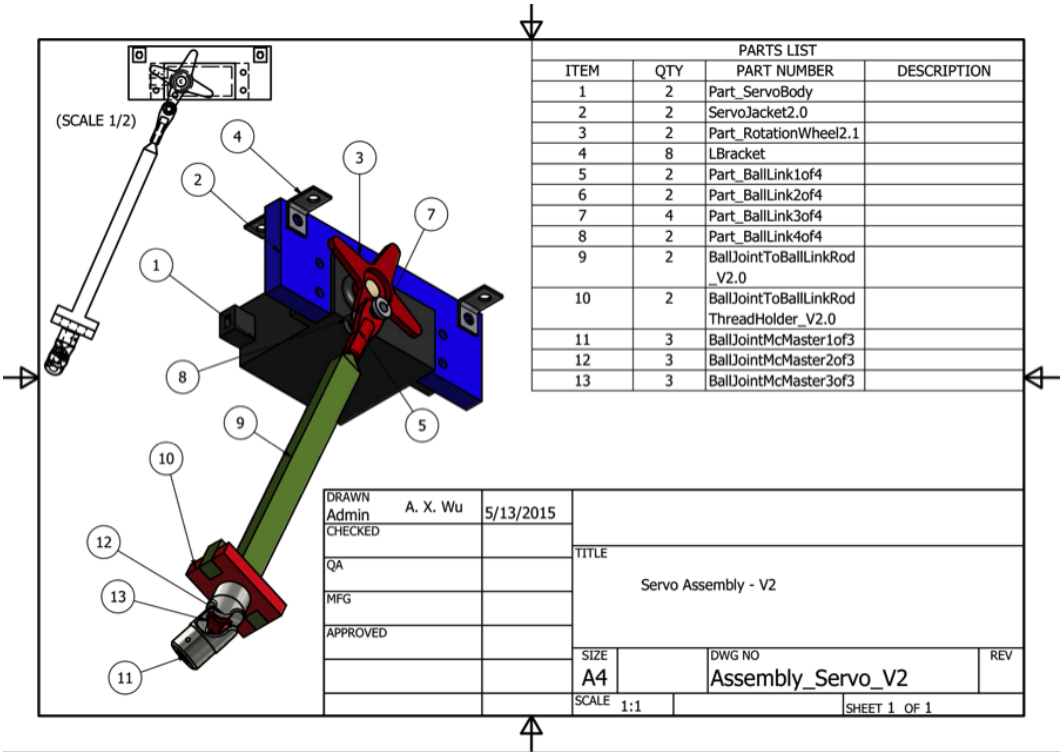


Figure 12: Full attachment assembly and parts list.

Please check out Appendix B for bigger drawings of the assemblies.

2.4 Laser Cutting, Machining, Assembly

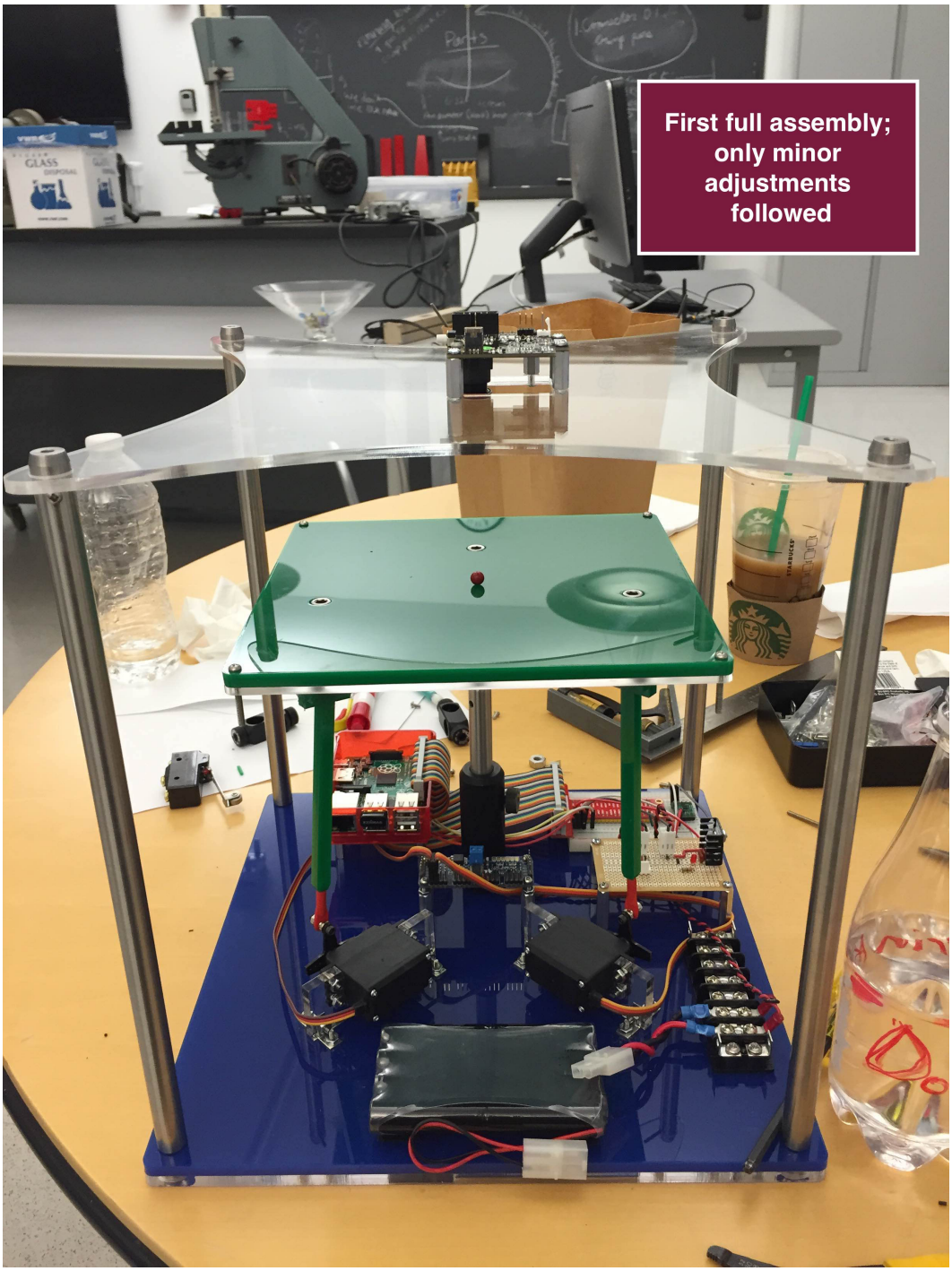


Figure 13: First full assembly.

We laser cut parts S-Z (see Section 2.2) from acrylics of various thicknesses and began built the system. The plastic u-joints from McMaster-Carr had to be threaded in the machine room with the help of TA. Rotational wheel attachment to the servo had to hand threaded. 2 layers of the bottom plate was printed, one on top (blue) with holes the size that would accommodate all hardware components snugly and another thick one on bottom (clear) with slightly bigger hole diameters to house the screw heads and 5 countersink holes to attach the 5 optical posts. With this setup, we did not have to spend time countersinking over 20 4-40 holes in the machine room. Overall, very minor adjustments had to be made after the first assembly.

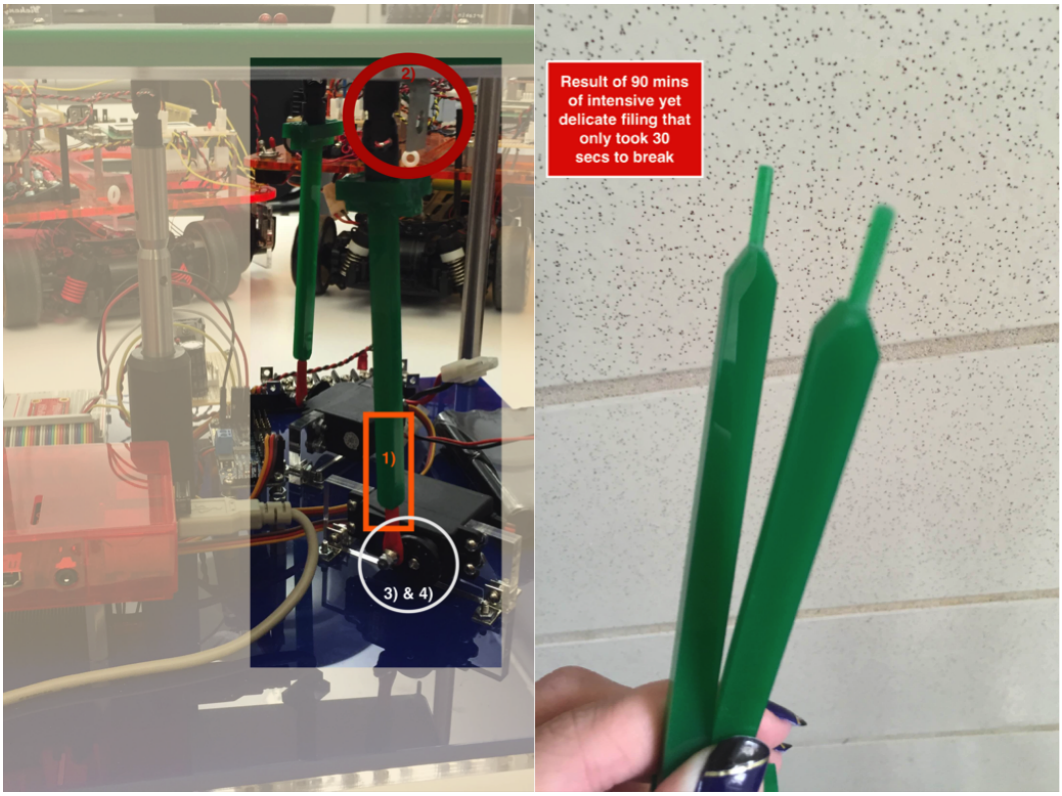


Figure 14: Adjustments since first assembly.

Adjustment 1): after spending a long time filing acrylic rods that we laser cut to accommodate the 4-40 opening of the ball link joint, these little delicate insertions broke without any kind of real force. The solution we came up with is the drill about an inch into the end of the rod with 4-40 threading and then chop off pieces from the 12" fully threaded 4-40 rods we ordered and use them as setscrews. They worked beautifully.

Adjustment 2): the price we had to pay, however, was that due to the drilling the two rods were not of equal length. The solution we came up with was putting an extra large L bracket to compensate for the difference in height. It doesn't look elegant, but it works well.

Adjustment 3): the ball link joints we use actually afford the rods substantial linear movement. They were used with the consideration that the linear freedom would lessen the chance of snapping the delicate end insertions of the rods. However, they were out of the pictures quite quickly and we ended with too much yaw for the plates. The solution we came up with was taking out the little ball (see Appendix B for details on these parts) in the ball link joint. The yaw was significantly reduced.

Adjustment 4): half way through testing, we realized that there's still a little bit of yaw to the balancing plates. The linear freedom actually came from the cross-shaped rotational wheel, since the balancing plates are quite heavy (even though we made them as light as possible) and the wheel is made of flexible plastic. To solve this issue, we opted to circular rotational wheels instead.

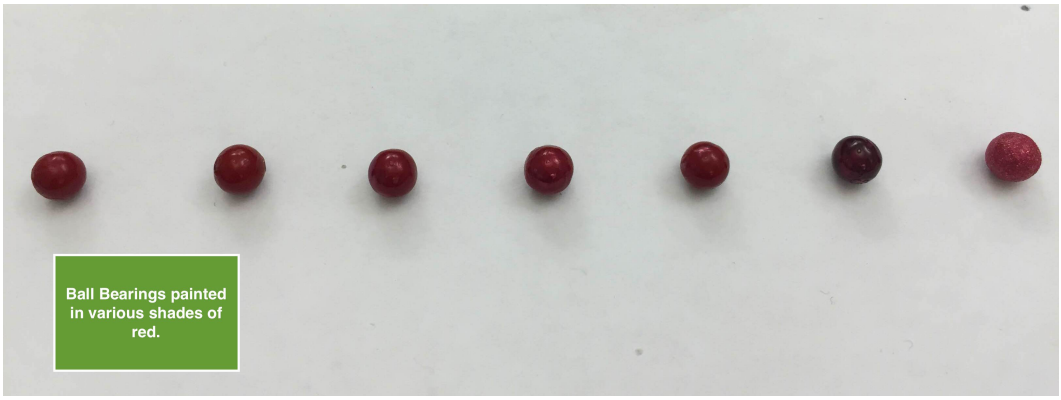


Figure 15: Ball bearings in various shades of red nail polish.

Lastly, we had to paint our ball bearings red for Pixy Cam recognition. Initially, we used nail polish in various shades of red. However, they made the balls slightly bumpy and uneven. Eventually, we realized our best option was a ball bearing colored red by a red sharpie.

PROJECT BALLERINA					
Part	**Acrylic No.**	**Thickness (mm)**		**Dimensions (mm)**	**Quantity**
Base Plate	B2	5.82		292 x 292	1
Maze Plate top	G1	2.83		200 x 200	1
Maze Plate middle	G3	6.06		200 x 200	1
Maze Plate bottom	C9	3.07		200 x 200	1
Servo Jacket	C8	6.15		80 x 30	2
Pixy Plate	C6	5.98		410 (diameter)	1
Ball Joint to Ball Link Rod	G3	6.06		15 x 25	2
Ball Joint to Ball Link Thread Holder	G3	6.06		127.2 x 25	2

Figure 16: Specifications measured for laser cutting.

In terms of laser cutting, we had to make sure that the H thread holder's thickness matched the depth of the rod, so when they're hot glued together, they will be flush. In addition, the top and bottom balance plate had to be the thinning option we had in green to reduce the weight that the servo had to support. On the other hand, the clear bottom plate (not listed) and the middle balance plate had to be the thickest plates we had to accommodate and house screw heads.

2.5 Hardware & Connection

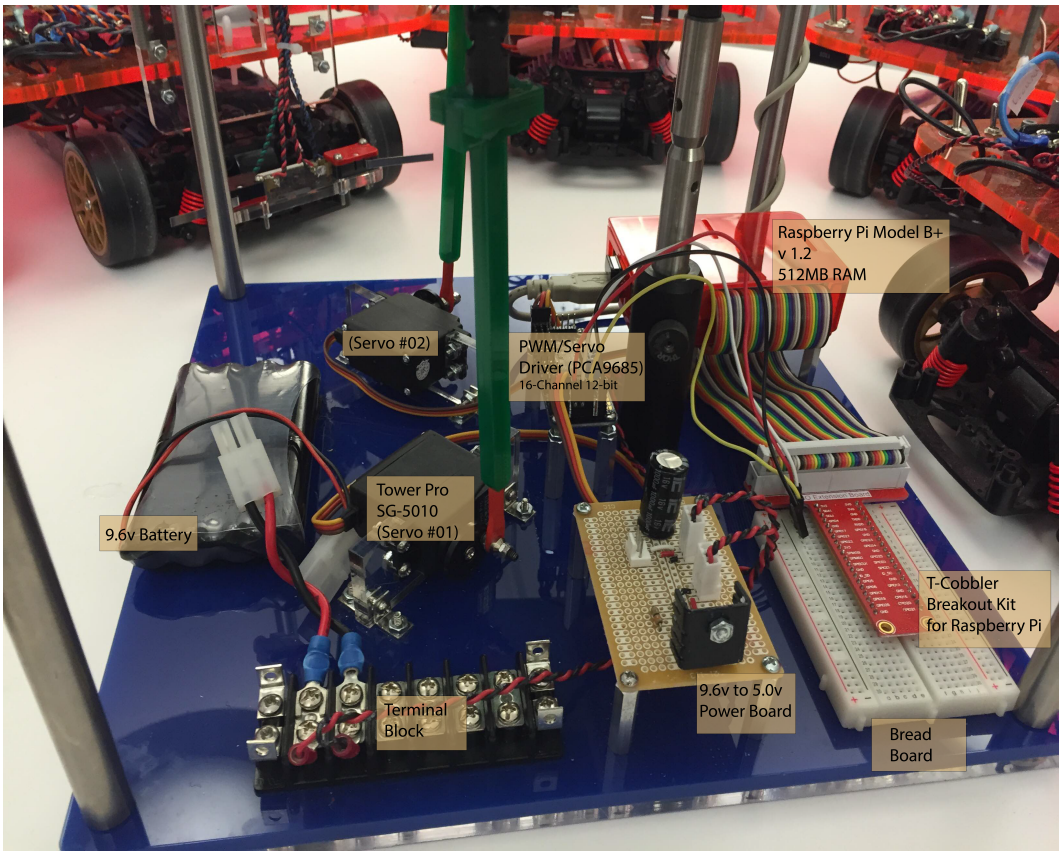


Figure 17: Photo of all control components involved.

After the whole system was assembled, we could clearly see all the hardware components and their connections in place. In terms of power supply, we have a 9.6 V battery, which is attached to a terminal block and connected to the power board. On the power board (see schematic), we have a LM3405 power regulator whose input is connected to the terminal block. A 0.1 µF capacitor is connected to the input to eliminate raw power noise.

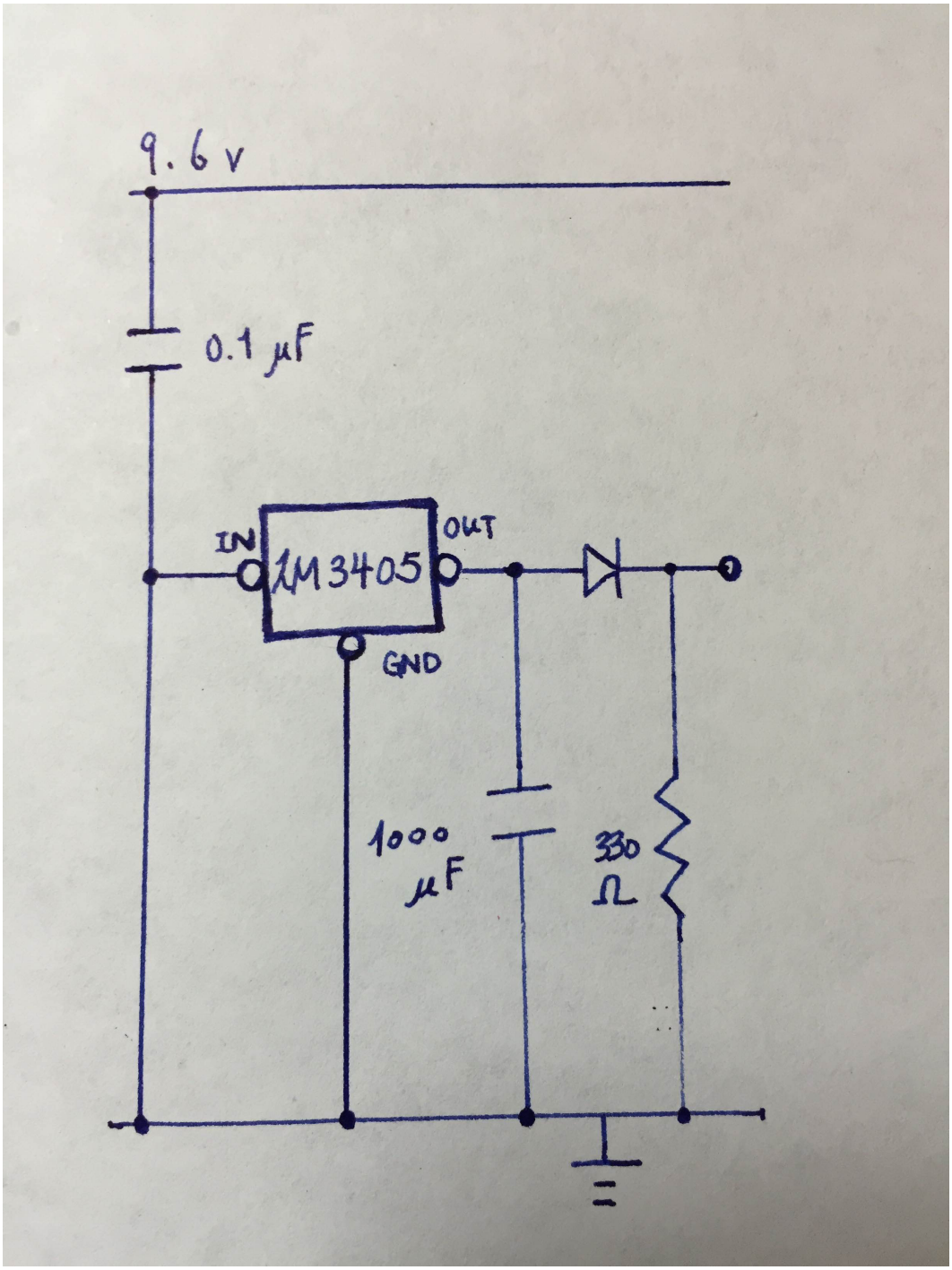


Figure 18: Artist rendition of noise-reducing power board.

As always, a diode along with a 330 Ω resistor is connected to the 5.0 V output of the power regulator. What sets this power board apart from our previous ones is the fact that we used a 1000 µF capacitor at the output of the power regulator. It might be overkill, but we experienced significant noise on the PWM/Servo driver board servo power outputs resulting from the current drawn during each pulse of the PWM outputs.

The PI is also connected to a monitor display (not shown), powered by a wall-plug micro-USB, and takes USB input from a keyboard and mouse which we need to execute our python codes. Two other USB ports are used for a WIFI dongo and Pixy cam input.

Lastly, the two Tower Pro SG-5010 servos are connected directly to the PWM/Servo driver board, which also powers them. In terms of control, the Raspberry PI is connected to a T-Cobbler breakout board, which is then connected to the PWM chip to drive the servos. We needed the PWM chip to drive the servos since the Pi does not contain a clock of its own and generating accurate PWM waves would be somewhat of a challenge. Since the PWM waves come from the servo driver chip which also powered the servos, the large capacitor was necessary so that the two outputs did not interfere with each other.

3 Controls Software

We used a combination of Raspberry PI B+, Pixy Cam, PWM Driver Board, and two servos to control our balancing board. The general idea is to have a software program running on the Raspberry PI which takes input coordinates of a colored ball from the Pixy Cam, uses those coordinates to calculate PID feedback control, and sends the appropriate pulse widths to a PWM Driver which controls two servos adjusting the tilt of the board along the x- and y- axis in the z direction, where z is the axis perpendicular to a the top and bottom plates.

All of our code and revision history for this project is publicly available on `https://github.com/jenny-s/Maze`.

3.1 Raspberry PI

Our Raspberry PI was the controller for the entire system and ran Python code that we wrote called `balance.py`. Several steps went into setting up our Raspberry PI. Specifically, we needed to partition the SD card, install Raspbian Wheezy OS, configure wireless Internet, configure I2C settings, set up the appropriate libraries for interfacing with the PWM driver board and Pixy cam. Along the way, many configuration details were changed back and forth in the OS in order to make everything work together in the best way. In retrospect, we definitely learned much more about Raspberry PI than was necessarily needed solely for this project.

Additionally, we decided to write our code in Python, since the Python libraries and example code we found looked perfectly suited for our purposes and were neat, uncluttered scripts. However, this also meant that we had to learn how to write Python and use Python libraries for the first time. In line with good coding practice, we learned to use GitHub for version control and figured out how to set up a repository on Raspberry PI.

`balance.py` relied mainly on the Adafruit Python library for General Purpose Input Output (GPIO) and the Charmed Labs Pixy library for getting the x- and y- coordinates of our ball. We did run into some trouble getting these libraries to be properly imported, since it was our very first time coding in Python and using Raspberry PI, but eventually it did work out nicely.

3.2 PWM Driver Board

Partway into the testing process, we discovered that the Raspberry PI only has one GPIO pin for PWM ouptut, meaning there seems to only be one pin which outputs signals controlled by a proper clock. We saw this in testing because originally, we had used Raspberry PI's GPIO library and sent PWM waves through an arbitrarily chosen channel, but the waves seen on the oscilloscope were clearly not even. The effects of the uneven PWM waves made the servos extremely unstable, which would definitely not work well to help us balance a ball.

Therefore, we found a specific PWM driver board from Adafruit which is controlled by Raspberry Pi via a prewritten I2C library. The PWM driver board is capable of sending 16 independent PWM waves, and it is is further capable of regulating a power for 16 servos. Testing showed the PWM waves output by the board to be stable with negligible error seen by an oscilloscope.

However, the PWM pulse widths can only be changed one output pin at a time in software, so the speed of the servo adjustments is dependent on how fast the each line of code get processed in software. Unfortunately, that speed is not very fast and turned out to be the limiting factor for our balancing control.

3.3 Pixy Cam

We used a Pixy Cam connected to the Raspberry PI via micr-USB to track the x- and y-coordinates of the ball on a plane perpendicular to the overall system structure z-axis. This was done automatically by the pre-written image processing software in the Pixy cam and the Pixy Cam Python library we imported.

The first task in using Pixy Cam was teaching it to recognize a red ball bearing. While this is easy to do once (simply following the instructions they Charmed Labs has on their website), what we found was that Pixy forgets the object sometimes when it gets power-cycled. Eventually, we stopped ever turning off the Raspberry PI so that the Pixy would remain continuously powered.

Further, the relatively small size of the ball bearing we used made it difficult for Pixy to recognize the ball without very finely tuned calibration. Exposure settings, block size, and signature signature settings are all important for making sure the Pixy recognizes our ball consistently. This was done using a program from Charmed Labs called PixyMon, which does not run on Raspberry PI. This meant that we had to calibrate all of our settings on a Macbook and then connect the Pixy back to the PI and hope that the object we just taught the Pixy didn't disappear in the meantime.

3.4 Vector Transformation

Before we could do this though, we needed to do a series of vector transformation on our x- and y- coordinates so that the tilt of each axis controlled by the corresponding servo would correspond exactly to the coordinate system of the Pixy cam. This is because our balancing plate was designed to be controlled on axis shifted and rotated from the Pixy cam axis, as seen in Figure 9. Further, the image is mirrored across the y-axis, caused by the Pixy Cam lens.

More specifically, let us from now on refer to the the Pixy cam coordinate system as the reference coordinate system with vectors represented as (x, y) and the balancing plate coordinate system as the target coordinate system with vectors represented as (x', y'). The origin of the target coordinate system represented in the reference coordinate system will be called (x_o, y_o). Then,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left[\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_o \\ y_o \end{bmatrix} \right] \begin{bmatrix} -\cos\theta & \sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{1}$$

Note that the rotation matrix also includes a flip across the y-axis in order to remedy the mirror effect of the Pixy Cam mentioned earlier.

Although the design of the plate system had an target coordinate system rotated 45° relative to the reference coordinate system, we used $\theta = 40°$ as discussed in Section 4. More complicated matrix transformations could have been done, also as discussed in Section 4, but we felt those transformations were unnecessary given our purposes and time constraints.

3.5 Feedback

The transformations described in the previous section allowed us to do feedback control for each of our two servos independently, with each servo corresponding to the tilt of a separate axis in the target coordinate system. Without those transformations, movement of the ball in the x and y directions would have been coupled together, making PID feedback control difficult. Now, the x- and y- axis tilt controlled by the respective servos corresponds directly to the x- and y- movements of the ball on the plane perpendicular to the z-axis and parallel to the top and bottom plates of the whole system.

Independent PID feedback calculation means that each servo/axis combination has its own error calculation, as well as $\Delta\ x$ ring buffer. Without loss of generality, we will only look at feedback control in the x direction.

Unlike the PID control we wrote for speed and navigation control, the version we used for our ball balancing plate used not only proportional (P) and derivative (D) control (without integration (I) control), but also included second derivative (2D) control. In terms of the physics of the movement of the ball, this makes sense. P control corresponds to displacement control and attempts to minimize error between the goal coordinates and the actual coordinates. D control aims to keep the velocity of the ball (which is the first derivative of displacement) at a minimum, and likewise, 2D aims to keep the acceleration of the ball at a minimum.

Both D and 2D control keep the ball from flying off the board due to oscillations; from observation, a tiny bit of 2D especially helps the ball come to a stop. More importantly, these components of feedback control made it possible for our incredibly slow servos to keep up with the ball. Without good control of velocity and a little control of acceleration, the gravity and momentum cause severe oscillations to occur even with K_p values too small to achieve noticeable error minimization. However, too much of these feedback variables, especially acceleration, actually resulted in increased instability.

PID calculations followed from Equation 2 implemented in code, where K_p, K_i, K_d and K_{2d} are the proportional, integral, first derivative and second derivative constants, respectively, and e_{ss} = (`goal coordinate`) − (`actual coordinate`) is the error in the feedback loop. Together, they

$$\texttt{cmp} = K_p e_{ss} + K_i \int_0^t e_{ss} dt + K_d \frac{\Delta x}{dt} + K_{2d} \frac{\Delta^2 x}{dt} \tag{2}$$

The cmp value calculated is taken as a parameter by the `setPWM` function of our `pwm` object, as in `pwm.setPWM(servo1, 0, int(cmp_x))`. I control is not used, so K_i is set to 0.

We also have an drawing option in our code, which makes the ball trace between two hard-coded goal points. The ball switches between the two goals based on whether it has reached its current goal (within a buffer zone) for a certain amount of time in ticks. Drawing utilizes different feedback constants. This is because having the goal coordinates be at the edge of the board means there is less room for oscillations. Thus the values for K_p and K_{2d} need to be smaller.

3.6 Calibration and Testing

The actual process of testing and debugging involved a lot of calibration to make sure the board was level, the servos had a good minimum/maximum movement range, and that they were moving in the proper directions. To help with this process, several helper Python programs were written. Please see Appendix A for this code.

First, `location.py` printed the coordinates of the ball seen by the Pixy Cam. This helped a lot with setting up the transformation matrices, since we first needed to know exactly what the Pixy saw and how its coordinate system was set up. We tested the coordinate system by spot-checking key locations on the board using a red Q-tip taped to a long acryllic arm.

Next, `testServo.py` was used to determine the full range of the servos such that each was able to tilt its respective axis by the maximum angle. Once the full range of tilt was determined, this script was used to adjust the pivot post to the optimal height such that a 45° angle on each servo arm corresponded to a level plate. This helped ensure not only the maximum amount of freedom in plate tilt, but also ensured that both axis had the same amount of room for adjustment in feedback control.

Lastly, `level.py` was used to set the plate to a stable level configuration. This was useful for calibrating the open loop control constant, as well as initializing a reasonable position for testing with the real ball (as opposed to Q-tip arm). Along the way, many minor adjustments and calibrations were done to get a level open loop controlled plate with the full range of motion. Only after this set was complete were the servo arms were finally screwed into place and range constants set in software so that we could test PID constants.

4 Error Analysis

Much of our error, though not extremely noticeable, came from the our Pixy cam. First, the coordinates we obtained from the Pixy cam were relative to a fixed coordinate plane perpendicular to the top and bottom of our platform; they were not fixed relative to the ball balancing plate. This means that the tilt of the plate affects the camera's perspective of where the ball is to a non-negligible amount. The position of our ball is always given relative to the overall larger stable structure and not relative to the balancing plate.

It would have been possible to write a variable skew transformation matrix dependent on the servo pulse widths (ie. angle of the tilt of the board) to make positioning relative to the balancing plate, but the benefits of that task would have been far outweighed by the time costs of the many calibrations needed.

Additionally, the Pixy Cam has a noticeable fish-eye from its camera lens. Both of these effects were ignored as well. The fish-eye could have been fixed by another very specifically tuned transformation matrix. However, this would have been more difficult than it sounds because the Pixy lense sees a rectangular rather than square image such that the fish-eye is not consistent within the square of the balancing plate, which itself has changing coordinates as mentioned earlier.

In all, the distortions contributed by the Pixy, though not insignificant, were not bad enough to severely affect our balancing controls. However, these distortions were definitely carried through into our rotation and translation vector transformations–our 45° rotation transformation matrix turned out to do non-linear rotations because of the fish-eye on the Pixy. To make up for this, we decided to use a 40° rotation matrix instead to artificially correct for this error.

Separate from distortions, our strongest bottleneck was servo speed. The servos seemed to be limited to a PWM frequency of 100 Hz. The servos we were using not only were slow to begin with, but also could not be adjusted concurrently. Essentially, the second servo could not adjust its angle until the first servo was done being adjusted, since the pulse width changes were being executed sequentially in software. As mentioned before in Section 3.5, this made derivative and second derivative control even more important. If the velocity and acceleration of the ball were not controlled, the servos would simply not be able to keep up with the effects of gravity and momentum.

We strongly suspect that given better servos, our convergence time would have been significantly improved, and our controls would have worked even without such finely tuned parameters. However, we were still able to get past these severe limitations and achieve a

beautifully working ball balancing platform.

A Code

`balance.py` is the actual control code for our balancing plate. The other python files were used in the process of testing, calibrating, and finding appropriate feedback constants.

balance.py

```
#!/usr/bin/python
from AdafruitLibrary.Adafruit_PWM_Servo_Driver.Adafruit_PWM_Servo_Driver import PWM
from collections import deque
import pixy
import ctypes
import math
import time

# ============================================================================================
# PID control to balance ball on platform with 2 degrees of freedom
# ============================================================================================

# CONTROL ON
TEST = False
PID = True
DRAW = False

################################ __GLOBAL VARIABLES__ ################################
# Set PID constants
Kp = 1.5
Ki = 0.0
Kd = 50.0
K2d = 2.0

# Transform params
ORIGIN = [119, 96]
CENTER = [33, 37]
angle = 40

# Draw params
V1 = [60, 10]
V2 = [9, 51]
V3 = []
V4 = []
GOAL = CENTER
if DRAW:
  GOAL = V1

BUFFER = 20
HOLD = 30

XSHAPE = deque()
YSHAPE = deque()

XSHAPE.append(V1[0])
XSHAPE.append(CENTER[0])
XSHAPE.append(V2[0])

YSHAPE.append(V1[1])
YSHAPE.append(CENTER[1])
YSHAPE.append(V2[1])
```

```
# Draw count variable
holdCount = 0

# Set servo channels
servo1 = 0
servo2 = 15

# Initialize Pixy Interpreter thread
pixy.pixy_init()
blocks = pixy.Block()

# Initialize the PWM device using the default address
pwm = PWM(0x40)

# Uncomment for debugging mode
# pwm = PWM(0x40, debug = True)

# Define suitable pulse ranges
servo1Min = 10.0
servo1Max = 600.0
servo2Max = 600.0
servo2Min = 10.0

# Set open loop pulses
openLoop1 = 320
openLoop2 = 400

# vars for integral control
xIntegralSum = 0
yIntegralSum = 0

# vars for derivative control
xQue = deque()
yQue = deque()
count1 = 0
prevTime = 0

# 2nd derivative stuff
dxQue = deque()
dyQue = deque()
count2 = 0

################################## __FUNCTIONS__ ###################################
# Get system time to seconds
def Time():
  return time.clock() / 1000000

# Transform vector into new plane rotated by 45 degrees
def transform(vector):
  coordinates = [0, 0]
  translation = [0, 0]
  translation[0] = -(vector[0] - ORIGIN[0])
  translation[1] = vector[1] - ORIGIN[1]
  coordinates[0] = math.cos(angle) * translation[0] + math.sin(angle) * translation[1]
  coordinates[1] = -math.sin(angle) * translation[0] + math.cos(angle) *
    translation[1]

  return coordinates
```

```
# Calculate error in x direction
def Error_x(x):
  return GOAL[0] - x

# Calculate error in y direction
def Error_y(y):
  return GOAL[1] - y

# Integral Control
def IControl(x, y):
  global xIntegralSum, yIntegralSum

  xIntegralSum = xIntegralSum + Error_x(x)
  yIntegralSum = yIntegralSum + Error_y(y)

# Derivative control
def DControl(x, y):
  global xQue, yQue, dxQue, dyQue, count1, count2

  dx = 0
  dy = 0

  d2x = 0
  d2y = 0

  if count1 < 10:
    xQue.append(x)
    yQue.append(y)
    count1 = count1 + 1
  elif count1 == 10:
    dx = xQue.popleft() - xQue.pop()
    dy = yQue.popleft() - yQue.pop()
    count1 = count1 - 2
    if count2 < 10:
      dxQue.append(dx)
      dyQue.append(dy)
      count2 = count2 + 1
    elif  count2 == 10:
      d2x = dxQue.popleft() - dxQue.pop()
      d2y = dyQue.popleft() - dyQue.pop()
      count2 = count2 - 2
  dx = dx / 10
  dy = dy / 10

  return [[dx, dy],[d2x, d2y]]

# Draw something
def Draw(xError, yError):
  global GOAL, HOLD, holdCount

  if abs(xError) < BUFFER and abs(yError) < BUFFER:
    holdCount = holdCount + 1

    if holdCount == HOLD:
      GOAL[0] = XSHAPE.popleft()
      GOAL[1] = YSHAPE.popleft()

      XSHAPE.append(GOAL[0])
      YSHAPE.append(GOAL[1])
```

```
      holdCount = 0  # reset timing for next point

# Feedback control based on independent PID in x, y
def PID_Control(x, y):   # Currently only does P control
  xError = Error_x(x)
  yError = Error_y(y)
  derivative = DControl(x, y)

  # Decide on proper goal coordinate in shape
  if DRAW:
    Draw(xError, yError)

  # Calculate feedback for x and y, respectively
  cmp_x = openLoop1 + Kp * xError + Ki * xIntegralSum + Kd * derivative[0][0] + K2d *
    derivative[1][0]
  cmp_y = openLoop2 - Kp * yError - Ki * yIntegralSum - Kd * derivative[0][1] - K2d *
    derivative[1][1]

  if cmp_x < servo1Min: cmp_x = servo1Min
  elif cmp_x > servo1Max: cmp_x = servo1Max
  if cmp_y > servo2Max: cmp_y = servo2Max
  elif cmp_y < servo2Min: cmp_y = servo2Min

  pwm.setPWM(servo1, 0, int(cmp_x))
  pwm.setPWM(servo2, 0, int(cmp_y))

##################################### __MAIN__ #######################################
def main():
  global prevTime

  # Vector in x-y plane
  vector = [0,0]

  # Transform matrix
  coordinates = transform(vector)

  # Initialize servos
  pwm.setPWMFreq(60)   # Set PWM frequencies to 60Hz
  pwm.setPWM(servo1, 0, openLoop1)
  pwm.setPWM(servo2, 0, openLoop2)

  while True:
    #t = Time()
    #print "time: ", t - prevTime

    count = pixy.pixy_get_blocks(1, blocks)

    if count > 0:

      # Save coordinates of ball
      vector = [blocks.x, blocks.y]

      # Use coordinates if valid
      if vector[0] > 70 and vector[0] < 265 and vector[1] < 190:
        # Transform coordinates
        coordinates = transform(vector)

        # Print statements for testing
```

```
        if TEST:
          print "transformed: ", coordinates
          print "goal: ", GOAL
          print "error: ", Error_x(coordinates[0]), Error_y(coordinates[1])

        # Execute PID control
        if PID:
          PID_Control(coordinates[0], coordinates[1])

    #prevTime = t

################################### __Run MAIN__ ####################################
main()
```

locate.py

```
#!/usr/bin/python
import sys
sys.path[0:0] = '~/'
import pixy
from ctypes import *

# ============================================================================
# Prints coordinates of object
# ============================================================================

# Initialize Pixy Interpreter thread #
pixy.pixy_init()

# NOT USED #
class Blocks (Structure):
  _fields_ = [ ("type", c_uint),
               ("signature", c_uint),
               ("x", c_uint),
               ("y", c_uint),
               ("width", c_uint),
               ("height", c_uint),
               ("angle", c_uint) ]

# Create block object #
blocks = pixy.Block()

# Wait for blocks #
while True:

  count = pixy.pixy_get_blocks(1, blocks)

  if count > 0:
    # Blocks found #
    print '[X=%3d Y=%3d]' % (blocks.x, blocks.y)
```

level.py

```
#!/usr/bin/python

from AdafruitLibrary.Adafruit_PWM_Servo_Driver.Adafruit_PWM_Servo_Driver import PWM
import time

# ============================================================================
```

```
# Sets platform to be level
# ==========================================================================

# Initialise the PWM device using the default address
pwm = PWM(0x40)
# Note if you'd like more debug output you can instead run:
#pwm = PWM(0x40, debug=True)

servo1 = 320  # servo1 pulse length out of 4096 for level axis
servo2 = 400  # servo2 pulse length out of 4096 for level axis

def setServoPulse(channel, pulse):
  pulseLength = 1000000                   # 1,000,000 us per second
  pulseLength /= 60                       # 60 Hz
  print "%d us per period" % pulseLength
  pulseLength /= 4096                     # 12 bits of resolution
  print "%d us per bit" % pulseLength
  pulse *= 1000
  pulse /= pulseLength
  pwm.setPWM(channel, 0, pulse)

pwm.setPWMFreq(60)                        # Set frequency to 60 Hz
while (True):
  # Change speed of continuous servo on channel O
  pwm.setPWM(0, 0, servo1)
  pwm.setPWM(15, 0, servo2)
```

testServo.py

```
#!/usr/bin/python
from AdafruitLibrary.Adafruit_PWM_Servo_Driver.Adafruit_PWM_Servo_Driver import PWM
import time

# ==========================================================================
# Test range of two servos
# ==========================================================================

# Initialise the PWM device using the default address
pwm = PWM(0x40)
# Note if you'd like more debug output you can instead run:
#pwm = PWM(0x40, debug=True)

servo1Min = 150  # Min pulse length out of 4096
servo1Max = 600  # Max pulse length out of 4096
servo2Min = 600
servo2Max = 125

def setServoPulse(channel, pulse):
  pulseLength = 1000000                   # 1,000,000 us per second
  pulseLength /= 60                       # 60 Hz
  print "%d us per period" % pulseLength
  pulseLength /= 4096                     # 12 bits of resolution
  print "%d us per bit" % pulseLength
  pulse *= 1000
  pulse /= pulseLength
  pwm.setPWM(channel, 0, pulse)

pwm.setPWMFreq(60)                        # Set frequency to 60 Hz
while (True):
```

```
    # Change speed of continuous servo on channel O
32  pwm.setPWM(0, 0, servo1Max)
    pwm.setPWM(15, 0, servo2Max)
34  time.sleep(1)
    pwm.setPWM(0, 0, servo1Min)
36  pwm.setPWM(15, 0, servo2Min)
    time.sleep(1)
```

B CAD Datasheets

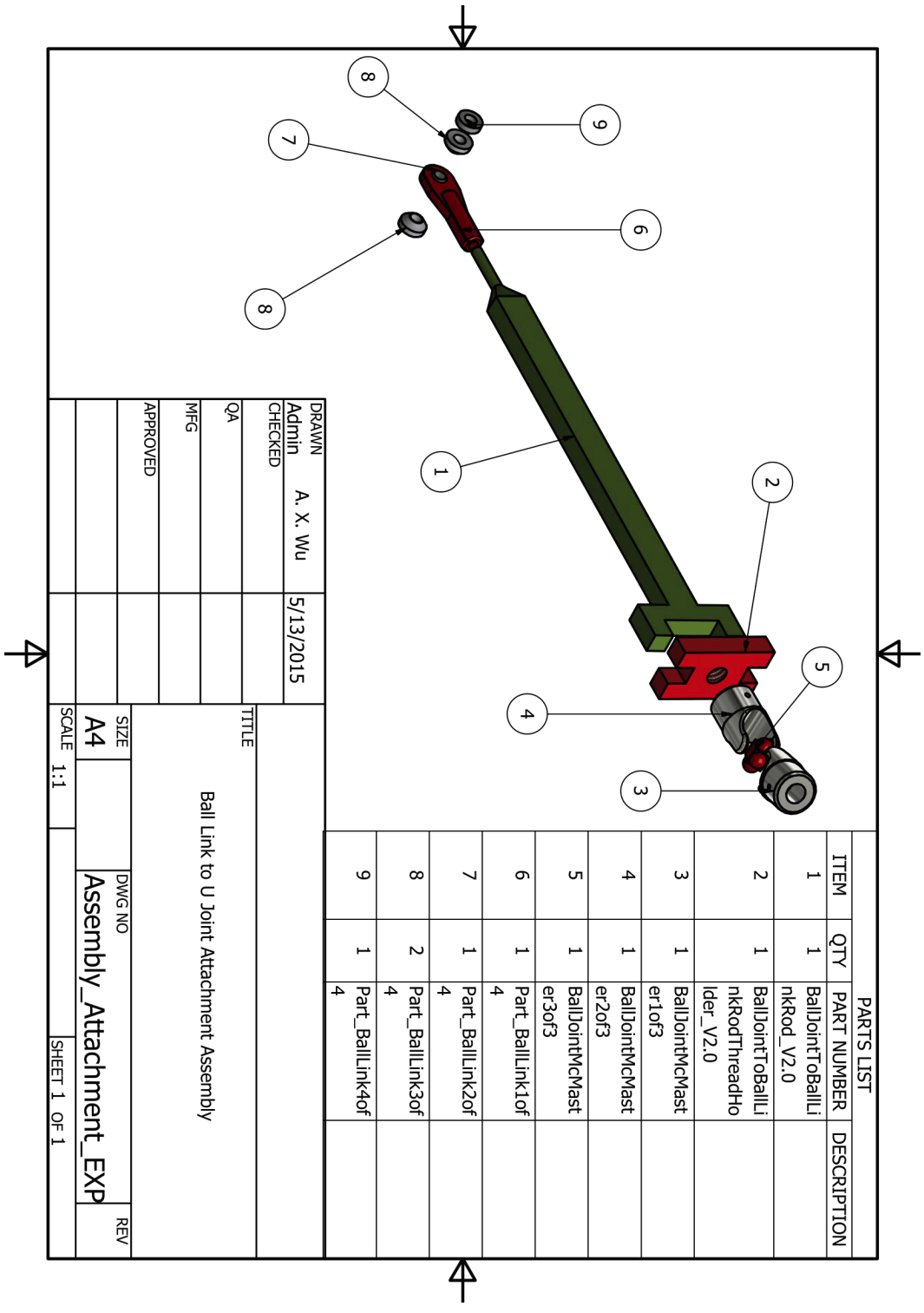
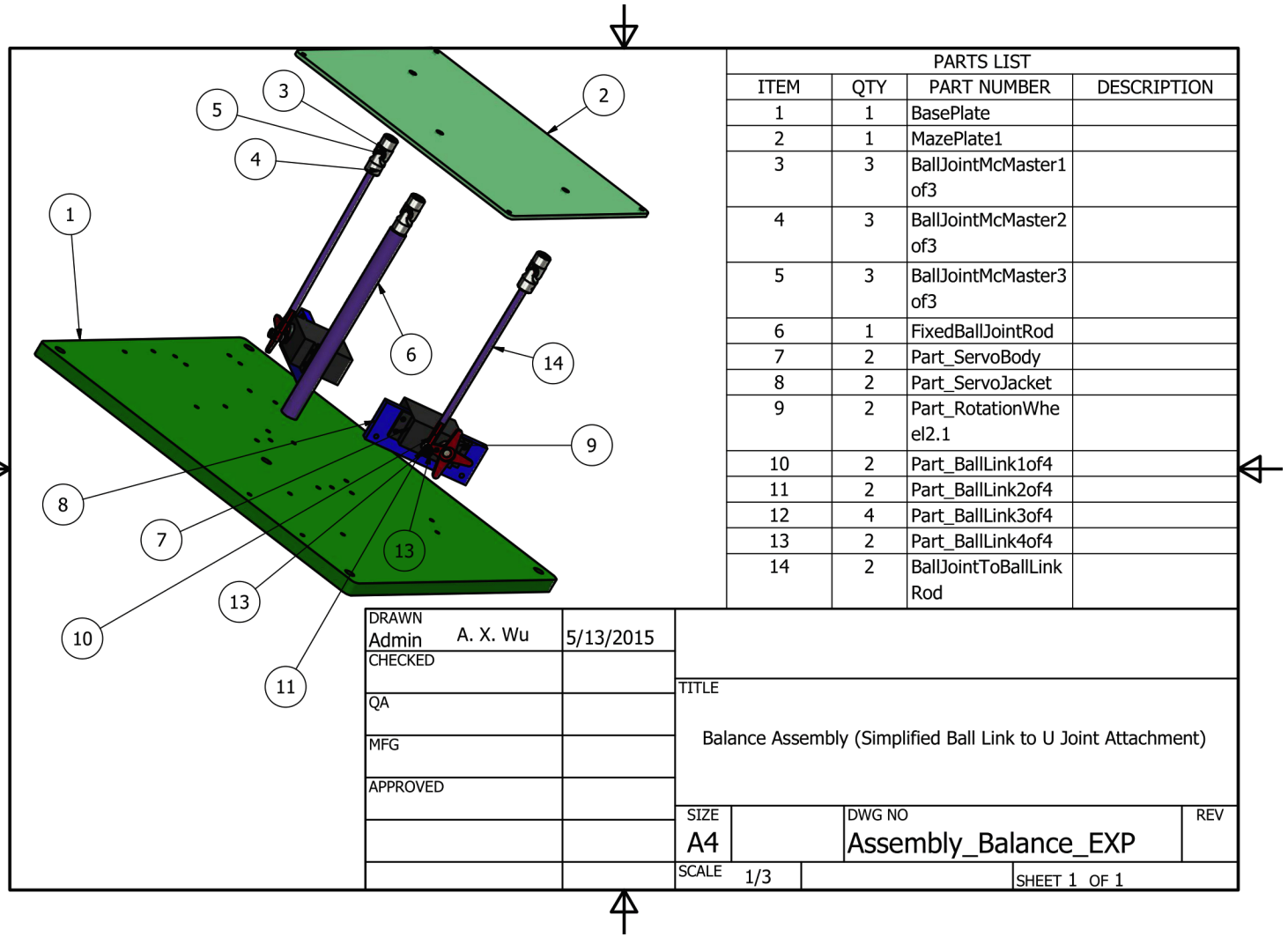


Figure 19:



PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	BasePlate	
2	1	MazePlate1	
3	3	BallJointMcMaster1 of3	
4	3	BallJointMcMaster2 of3	
5	3	BallJointMcMaster3 of3	
6	1	FixedBallJointRod	
7	2	Part_ServoBody	
8	2	Part_ServoJacket	
9	2	Part_RotationWheel2.1	
10	2	Part_BallLink1of4	
11	2	Part_BallLink2of4	
12	4	Part_BallLink3of4	
13	2	Part_BallLink4of4	
14	2	BallJointToBallLink Rod	

DRAWN Admin A. X. Wu	5/13/2015	
CHECKED		
QA		
MFG		
APPROVED		

TITLE: Balance Assembly (Simplified Ball Link to U Joint Attachment)

SIZE		DWG NO	REV
A4		Assembly_Balance_EXP	

SCALE 1/3 SHEET 1 OF 1

Figure 20:

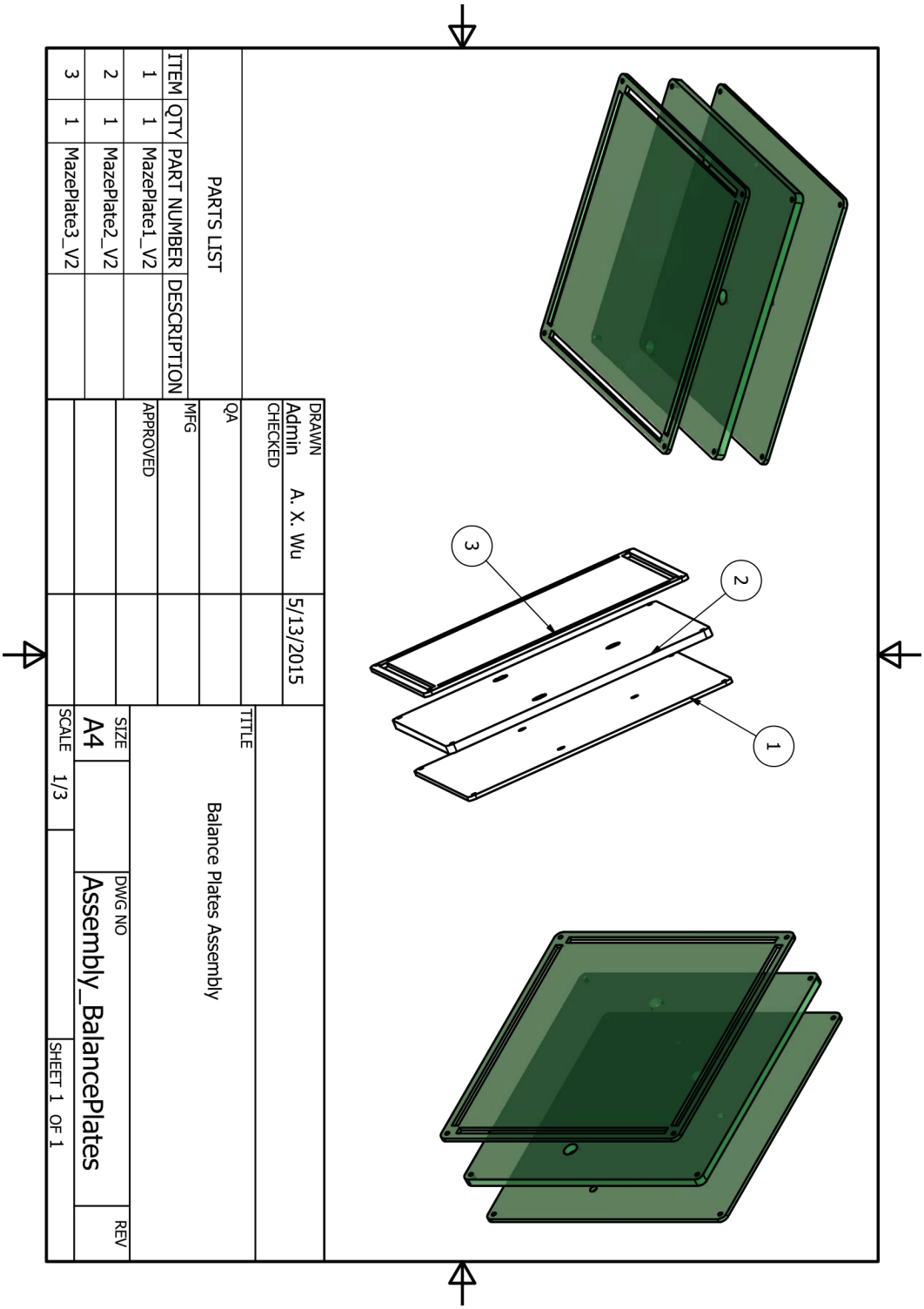

PARTS LIST
ITEM QTY PART NUMBER DESCRIPTION
1 1 MazePlate1_V2
2 1 MazePlate2_V2
3 1 MazePlate3_V2
DRAWN Admin A. X. Wu 5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Balance Plates Assembly
SIZE A4
DWG NO Assembly_BalancePlates
REV
SCALE 1/3
SHEET 1 OF 1
1
2
3

Figure 21:

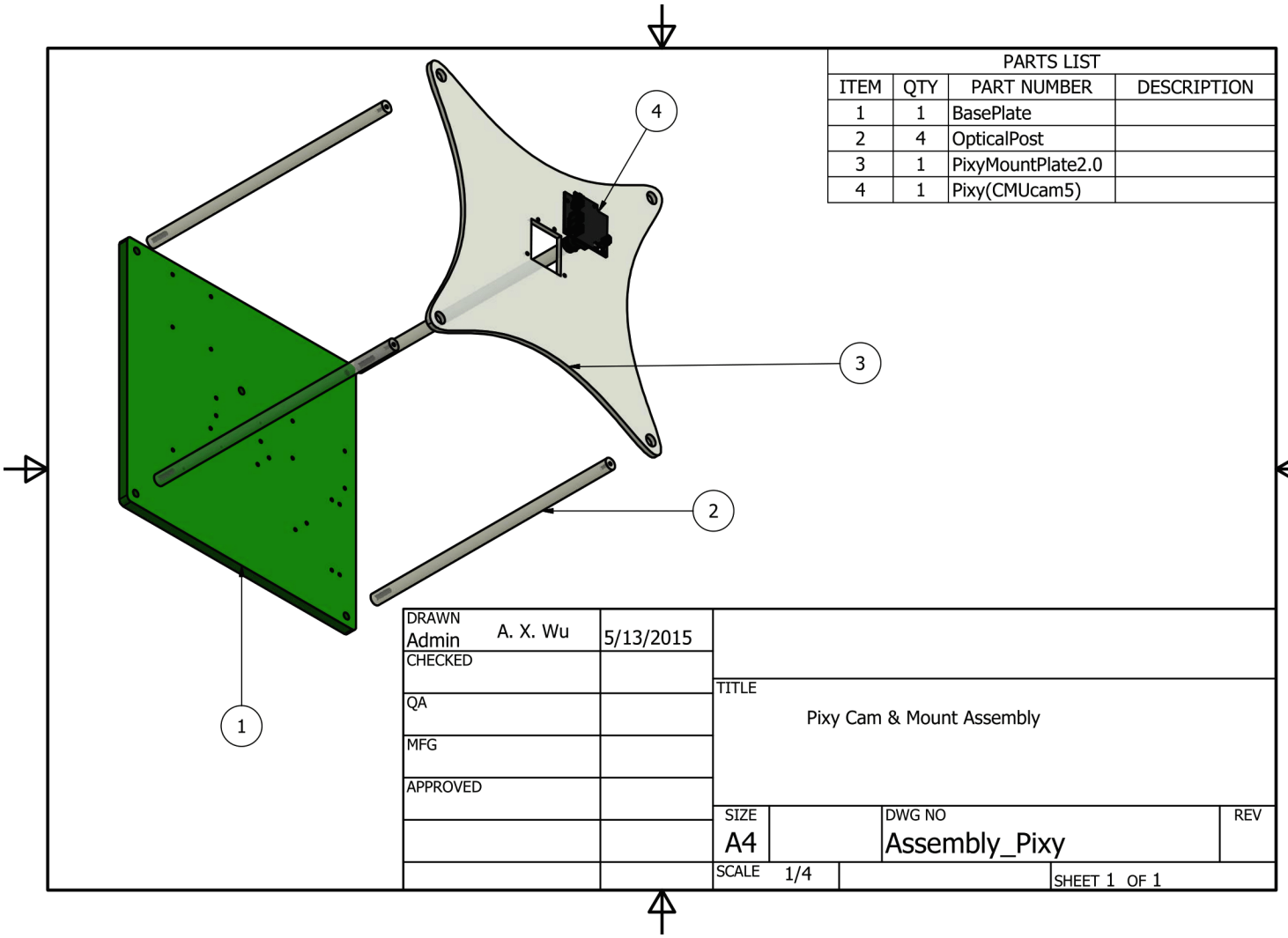

PARTS LIST
ITEM QTY PART NUMBER DESCRIPTION
1 1 BasePlate
2 4 OpticalPost
3 1 PixyMountPlate2.0
4 1 Pixy(CMUcam5)
4
3
2
1
DRAWN Admin A. X. Wu 5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Pixy Cam & Mount Assembly
SIZE A4
DWG NO Assembly_Pixy
REV
SCALE 1/4
SHEET 1 OF 1

Figure 22:

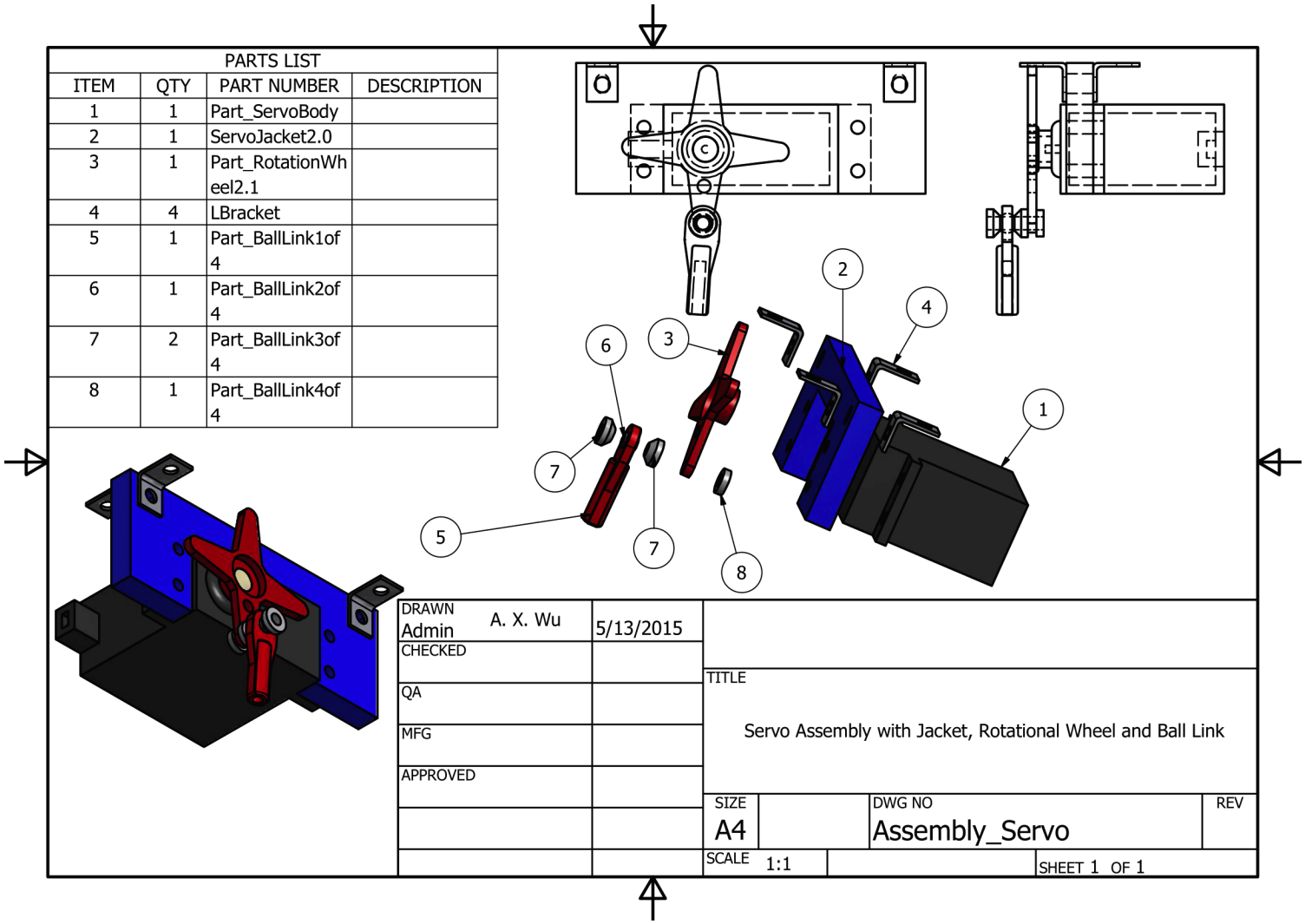
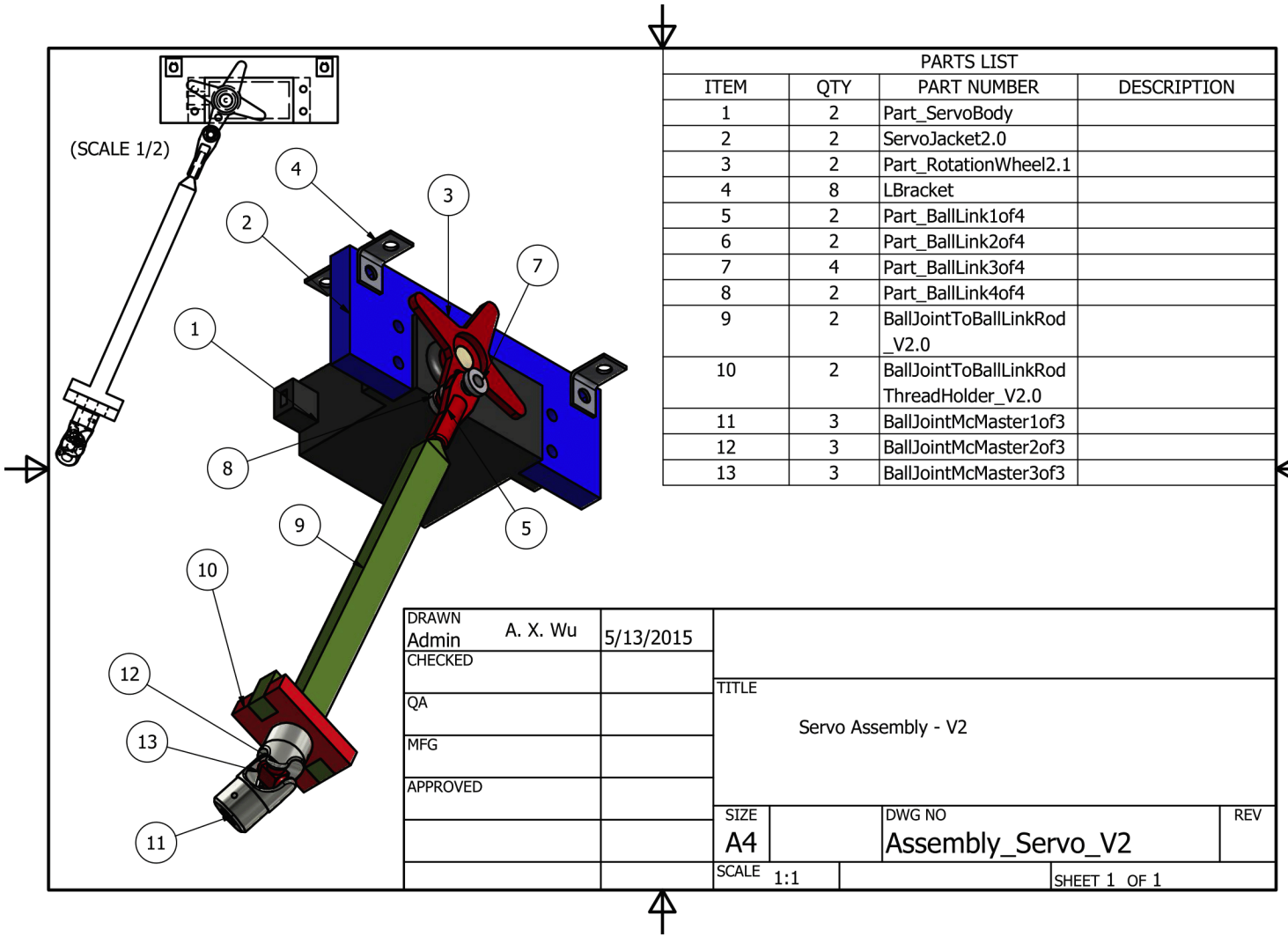

PARTS LIST
ITEM | QTY | PART NUMBER | DESCRIPTION
1 | 1 | Part_ServoBody |
2 | 1 | ServoJacket2.0 |
3 | 1 | Part_RotationWheel2.1 |
4 | 4 | LBracket |
5 | 1 | Part_BallLink1of4 |
6 | 1 | Part_BallLink2of4 |
7 | 2 | Part_BallLink3of4 |
8 | 1 | Part_BallLink4of4
1
2
3
4
5
6
7
7
8
DRAWN Admin A. X. Wu 5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Servo Assembly with Jacket, Rotational Wheel and Ball Link
SIZE A4
DWG NO Assembly_Servo
REV
SCALE 1:1
SHEET 1 OF 1

Figure 23:



PARTS LIST

ITEM	QTY	PART NUMBER	DESCRIPTION
1	2	Part_ServoBody	
2	2	ServoJacket2.0	
3	2	Part_RotationWheel2.1	
4	8	LBracket	
5	2	Part_BallLink1of4	
6	2	Part_BallLink2of4	
7	4	Part_BallLink3of4	
8	2	Part_BallLink4of4	
9	2	BallJointToBallLinkRod _V2.0	
10	2	BallJointToBallLinkRod ThreadHolder_V2.0	
11	3	BallJointMcMaster1of3	
12	3	BallJointMcMaster2of3	
13	3	BallJointMcMaster3of3	

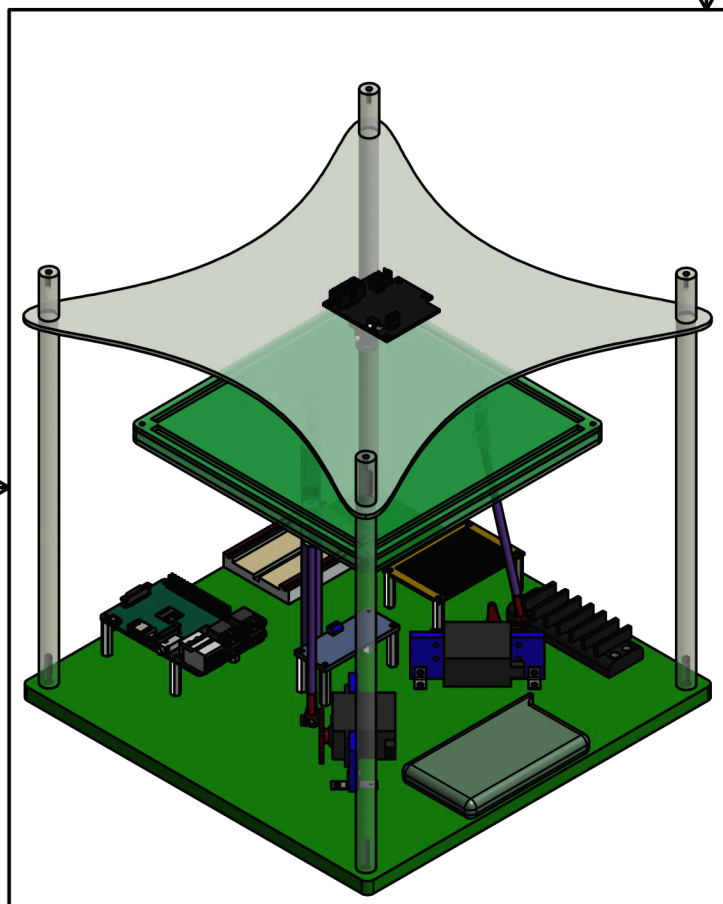
DRAWN Admin	A. X. Wu	5/13/2015
CHECKED		
QA		
MFG		
APPROVED		

TITLE: Servo Assembly - V2

SIZE A4 | DWG NO Assembly_Servo_V2 | REV

SCALE 1:1 | SHEET 1 OF 1

Figure 24:



PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	BasePlate	
2	1	MazePlate1	
3	3	BallJointMcMaster1of3	
4	3	BallJointMcMaster2of3	
5	3	BallJointMcMaster3of3	
6	1	FixedBallJointRod	
7	2	Part_ServoBody	
8	2	Part_ServoJacket	
9	2	Part_RotationWheel2.1	
10	2	Part_BallLink1of4	
11	2	Part_BallLink2of4	
12	4	Part_BallLink3of4	
13	2	Part_BallLink4of4	
14	2	BallJointToBallLinkRod	
15	1	MazePlate_Layer2	
16	1	MazePlate_Layer3	
17	8	LBracket	
18	1	Part_TerminalBlock	
19	1	Part_96Battery	
20	1	Part_PowerBoard	
21	1	Part_PWMChip(PCA9685)	
22	1	Part_BreadBoard	
23	1	raspberrypi_b_+_(assembly)	
24	12	Part_StandHalfInch	
25	1	PixyMountPlate	
26	1	Pixy(CMUcam5)	
27	4	OpticalPost	

Figure 25:

PARTS LIST

ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	BasePlate	
2	1	MazePlate1	
3	3	BallJointMcMaster1of3	
4	3	BallJointMcMaster2of3	
5	3	BallJointMcMaster3of3	
6	1	FixedBallJointRod	
7	2	Part_ServoBody	
8	2	Part_ServoJacket	
9	2	Part_RotationWheel2.1	
10	2	Part_BallLink1of4	
11	2	Part_BallLink2of4	
12	4	Part_BallLink3of4	
13	2	Part_BallLink4of4	
14	2	BallJointToBallLinkRod	
15	1	MazePlate_Layer2	
16	1	MazePlate_Layer3	
17	8	LBracket	
18	1	Part_TerminalBlock	
19	1	Part_96Battery	
20	1	Part_PowerBoard	
21	1	Part_PWMChip(PCA9685)	
22	1	Part_BreadBoard	
23	1	raspberrypi_b_+_(assembly)	
24	12	Part_StandHalfInch	
25	1	PixyMountPlate	
26	1	Pixy(CMUcam5)	
27	4	OpticalPost	

Figure 26:

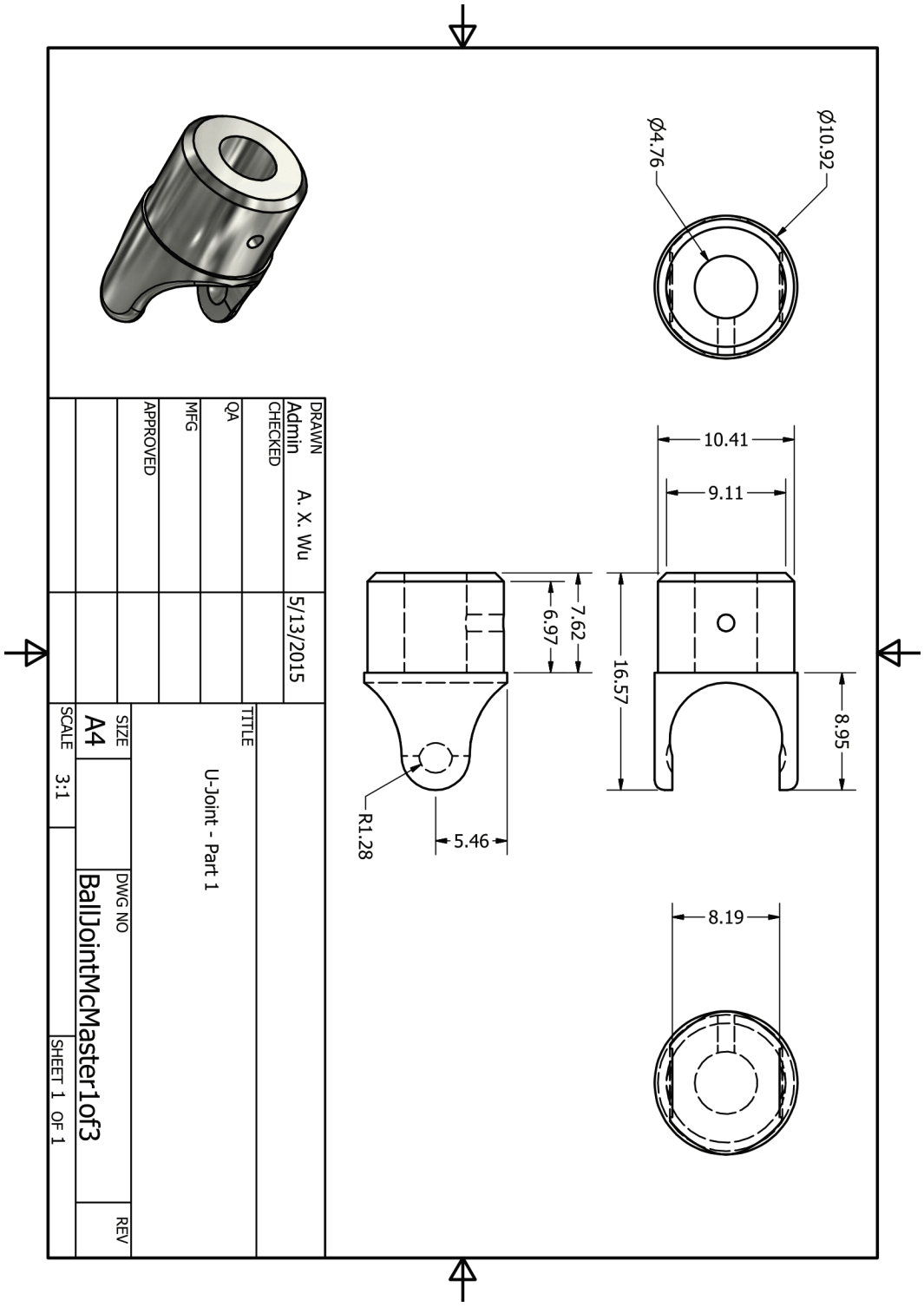

Ø10.92
Ø4.76
10.41
9.11
7.62
6.97
16.57
8.95
R1.28
5.46
8.19
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
U-Joint - Part 1
SIZE
A4
DWG NO
BallJointMcMaster1of3
REV
SCALE
3:1
SHEET 1 OF 1

Figure 27:

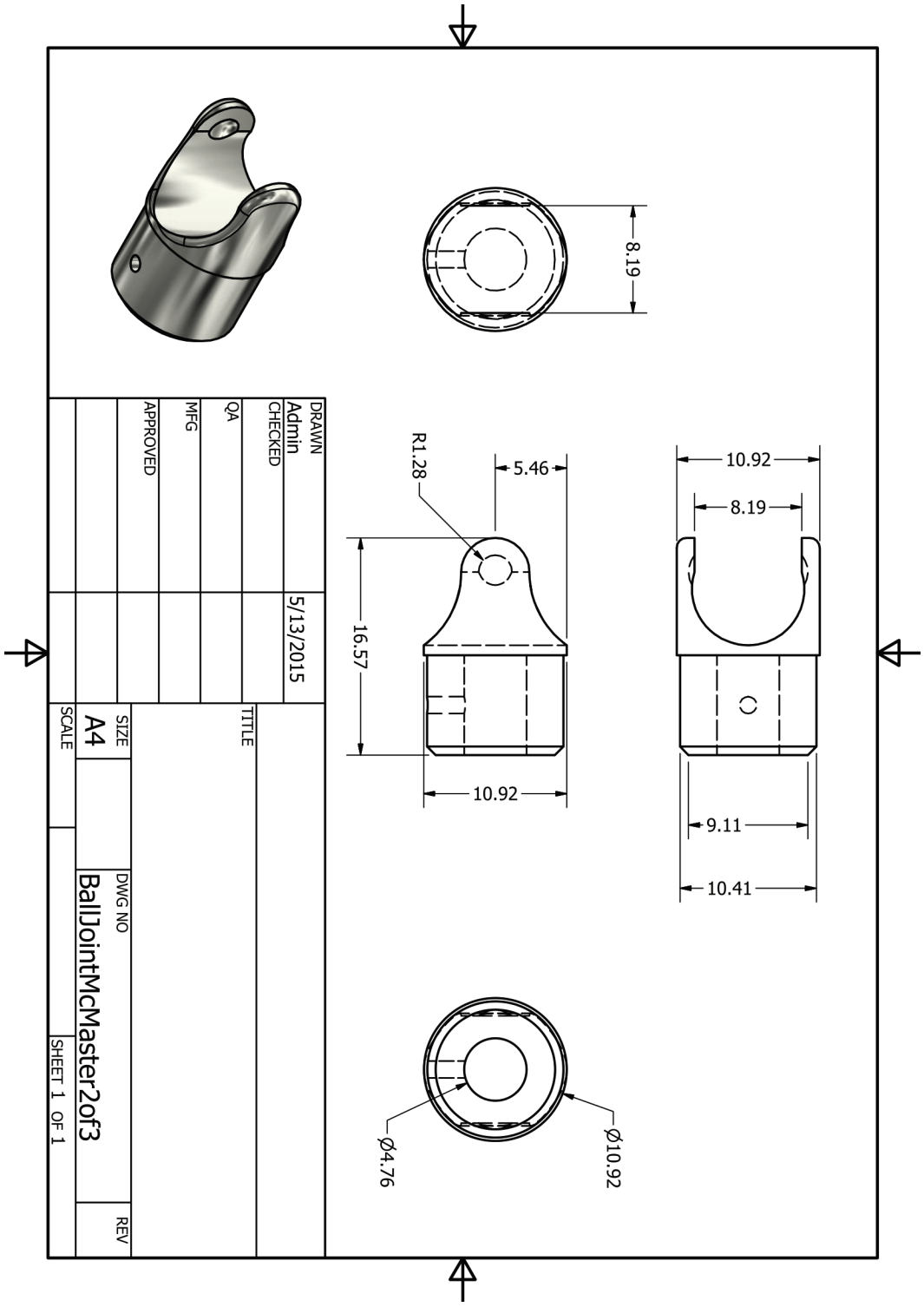

8.19
R1.28
5.46
16.57
10.92
10.92
8.19
9.11
10.41
Ø4.76
Ø10.92
DRAWN
Admin
CHECKED
QA
MFG
APPROVED
5/13/2015
TITLE
SIZE
A4
SCALE
DWG NO
BallJointMcMaster2of3
REV
SHEET 1 OF 1

Figure 28:

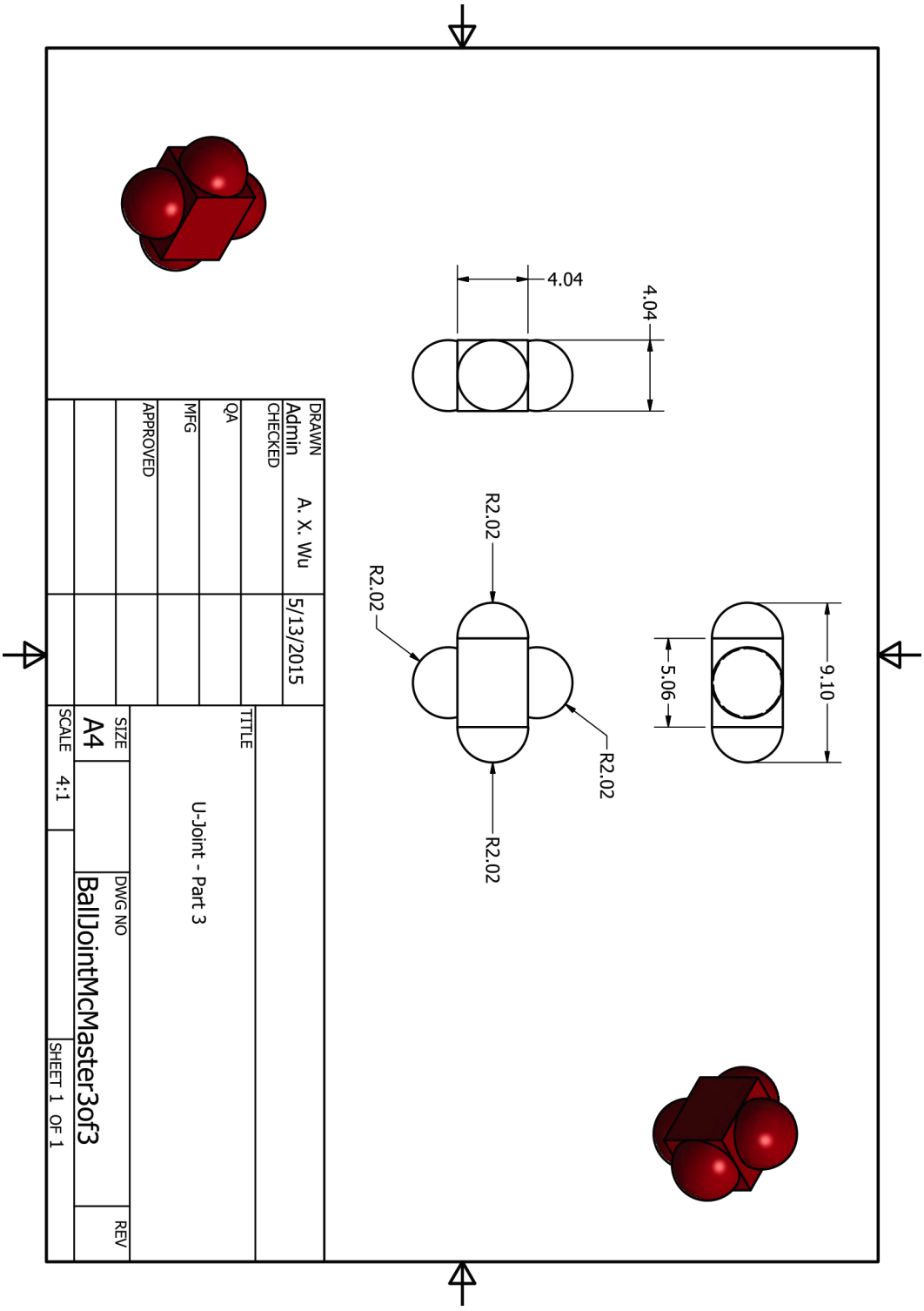

4.04
4.04
R2.02
R2.02
R2.02
R2.02
5.06
9.10
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
U-Joint - Part 3
SIZE
A4
DWG NO
BallJointMcMaster3of3
REV
SCALE
4:1
SHEET 1 OF 1

Figure 29:

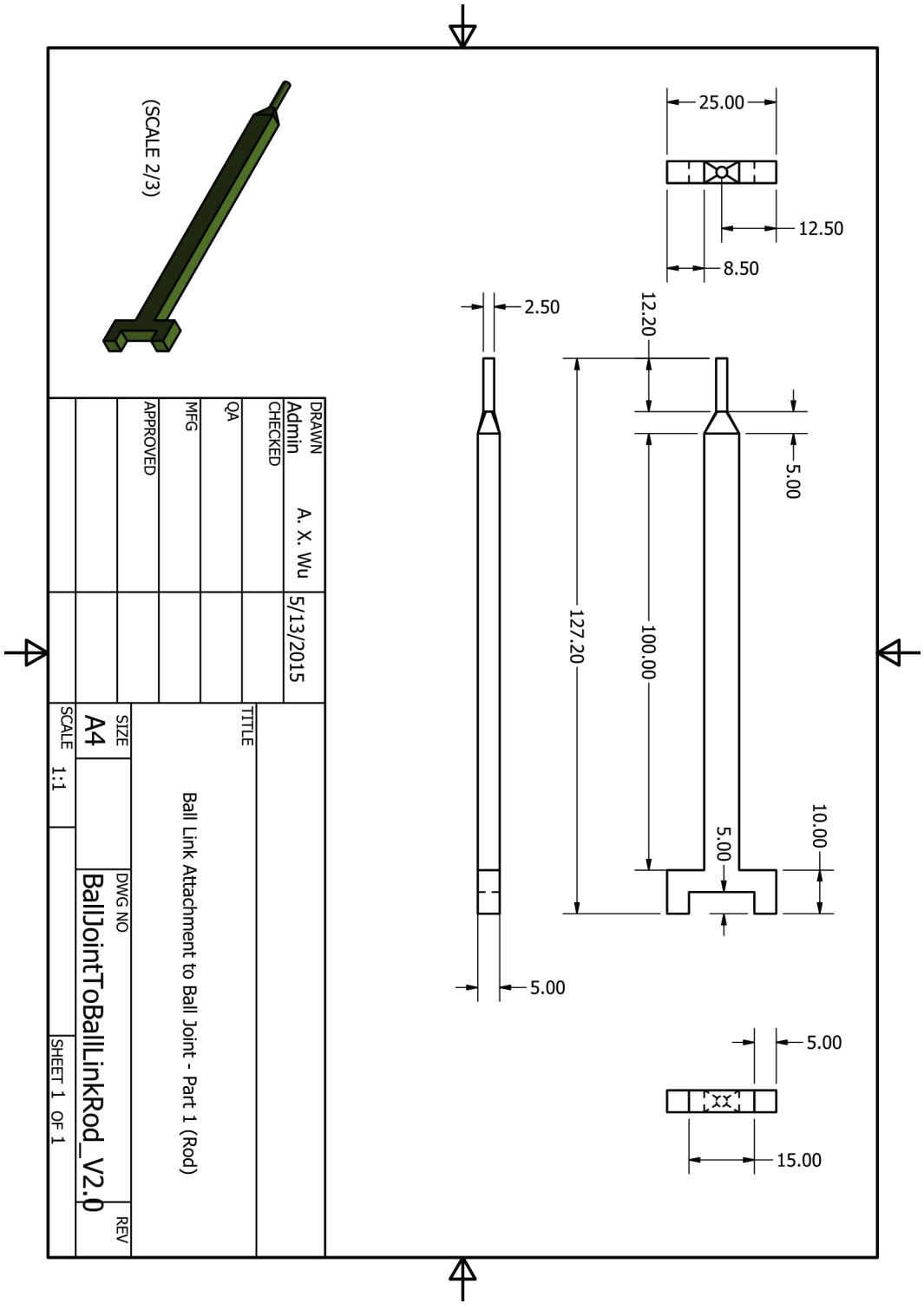

(SCALE 2/3)
25.00
12.50
8.50
2.50
12.20
5.00
127.20
100.00
5.00
10.00
5.00
5.00
15.00
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Ball Link Attachment to Ball Joint - Part 1 (Rod)
SIZE
A4
DWG NO
BallJointToBallLinkRod_V2.0
REV
SCALE
1:1
SHEET 1 OF 1

Figure 30:

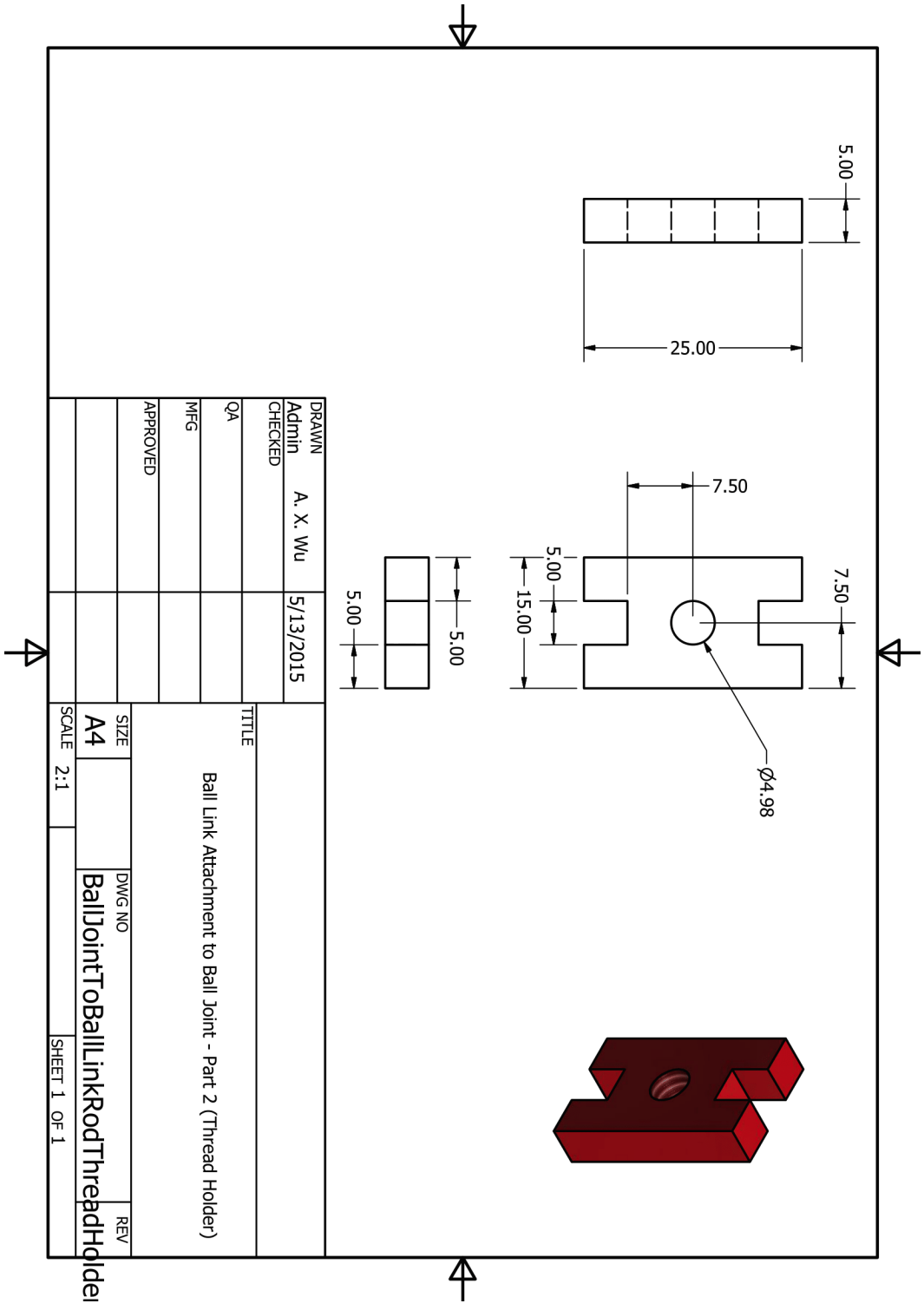

5.00
25.00
7.50
5.00
15.00
7.50
5.00
5.00
Ø4.98
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Ball Link Attachment to Ball Joint - Part 2 (Thread Holder)
SIZE
A4
DWG NO
BallJointToBallLinkRodThreadHolder
REV
SCALE 2:1
SHEET 1 OF 1

Figure 31:

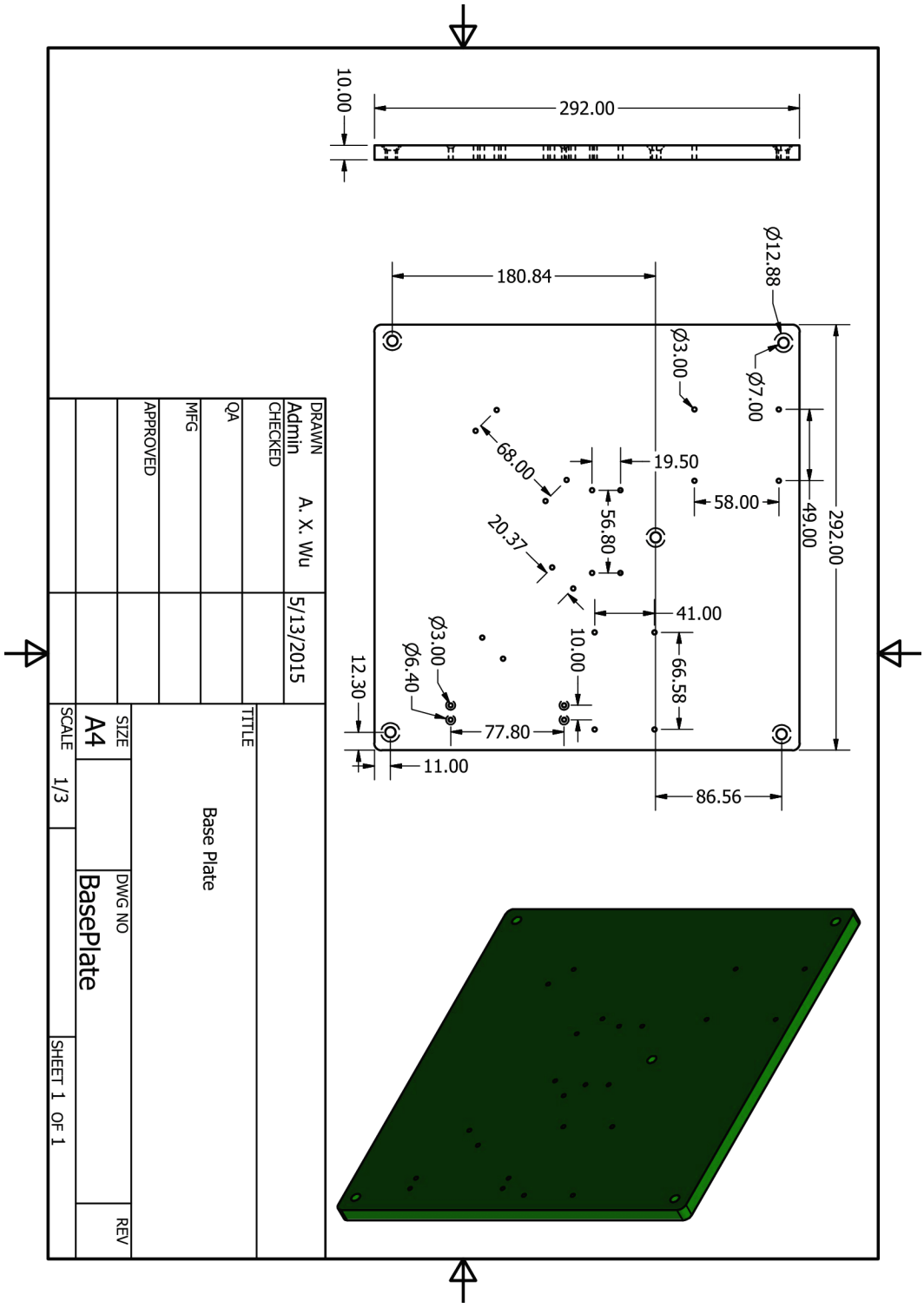

10.00
292.00
Ø12.88
180.84
Ø3.00
Ø7.00
68.00
19.50
58.00
49.00
292.00
56.80
20.37
41.00
Ø3.00
Ø6.40
10.00
66.58
12.30
77.80
11.00
86.56
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
SIZE
A4
SCALE
1/3
TITLE
Base Plate
DWG NO
BasePlate
REV
SHEET 1 OF 1

Figure 32:

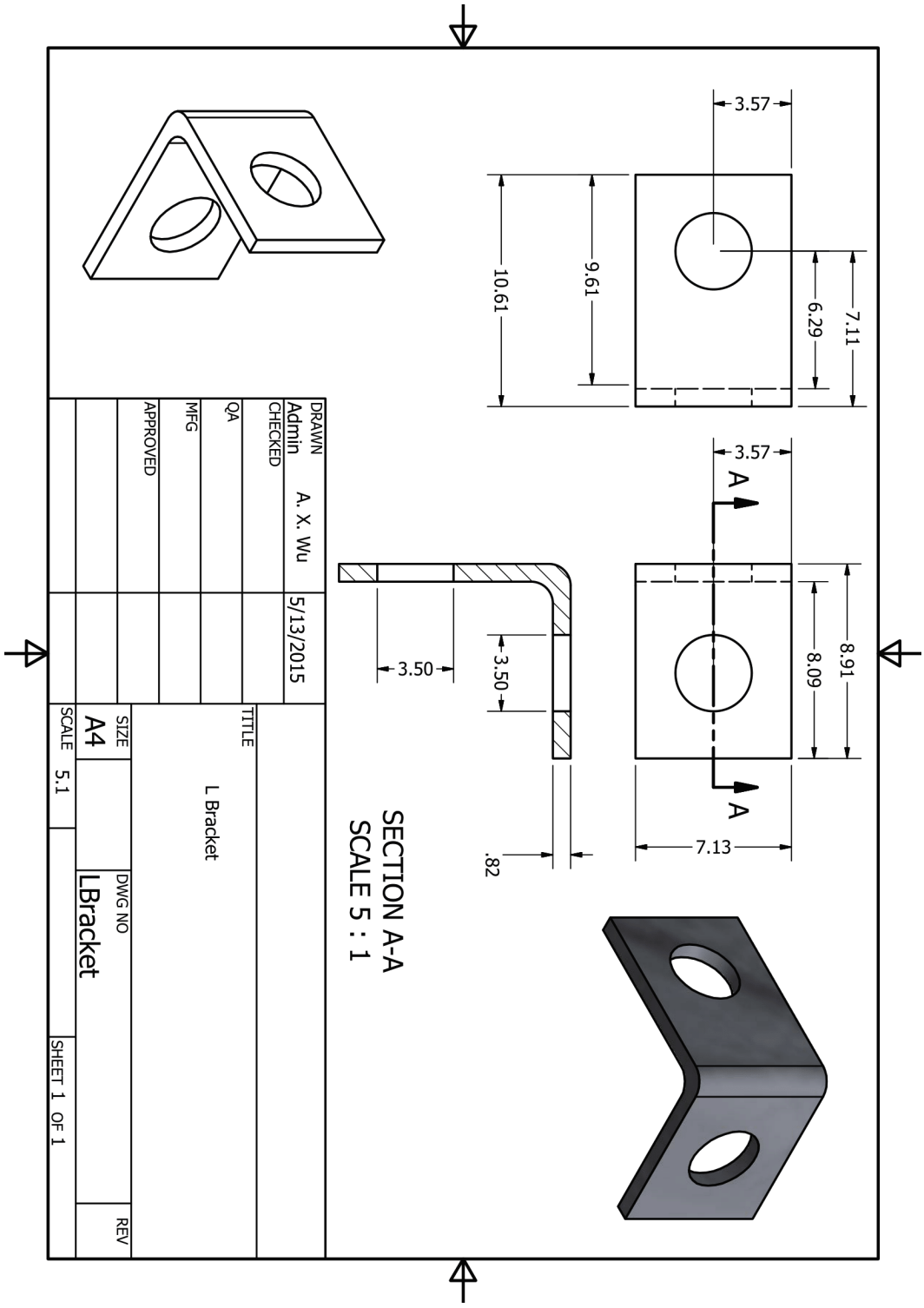

3.57
10.61
9.61
6.29
7.11
3.57
A
A
8.91
8.09
7.13
3.50
3.50
.82
SECTION A-A
SCALE 5 : 1
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
L Bracket
SIZE
A4
DWG NO
LBracket
REV
SCALE
5.1
SHEET 1 OF 1

Figure 33:

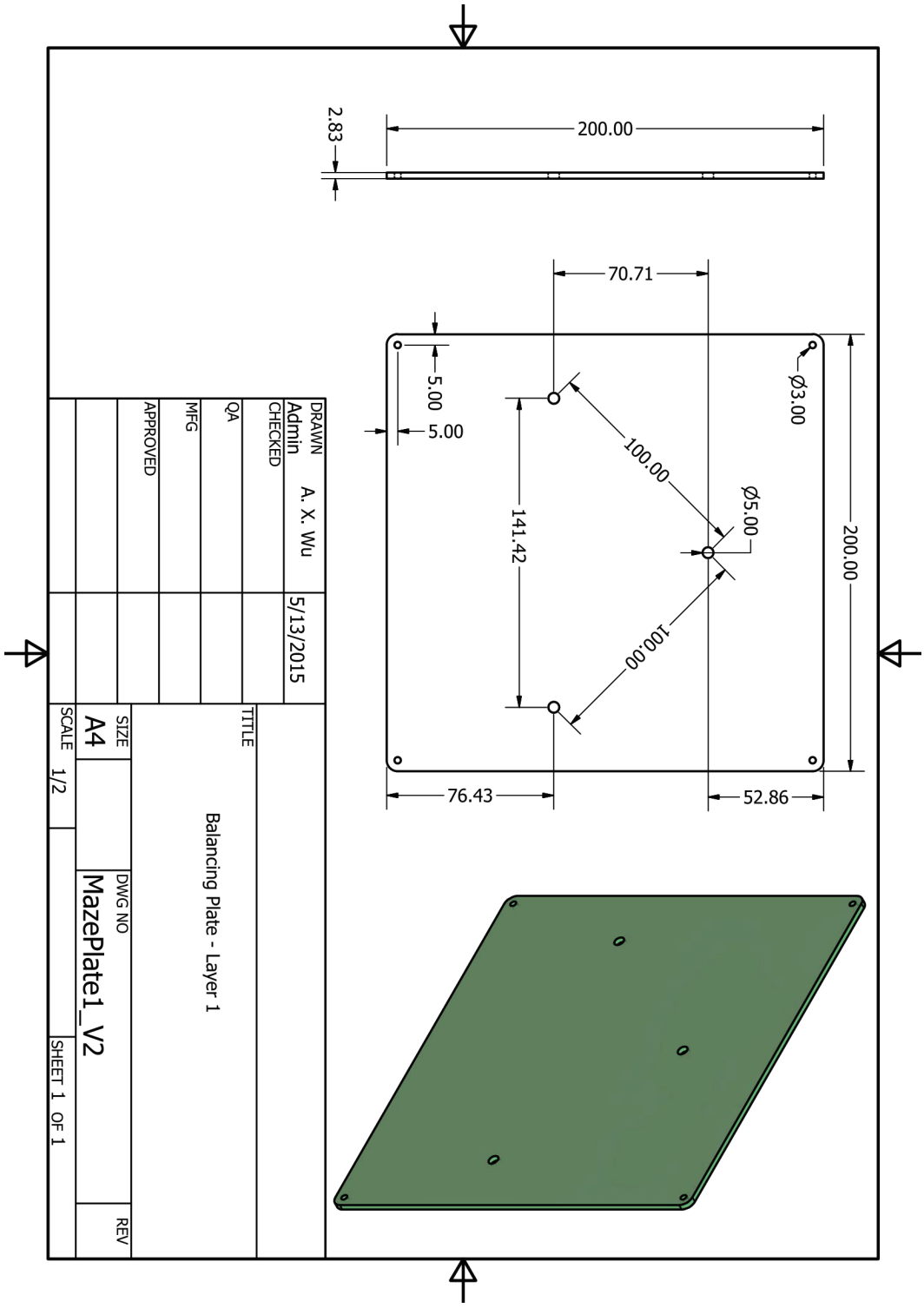

2.83
200.00
70.71
5.00
5.00
Ø3.00
100.00
Ø5.00
141.42
100.00
200.00
76.43
52.86
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Balancing Plate - Layer 1
SIZE
A4
DWG NO
MazePlate1_V2
REV
SCALE
1/2
SHEET 1 OF 1

Figure 34:

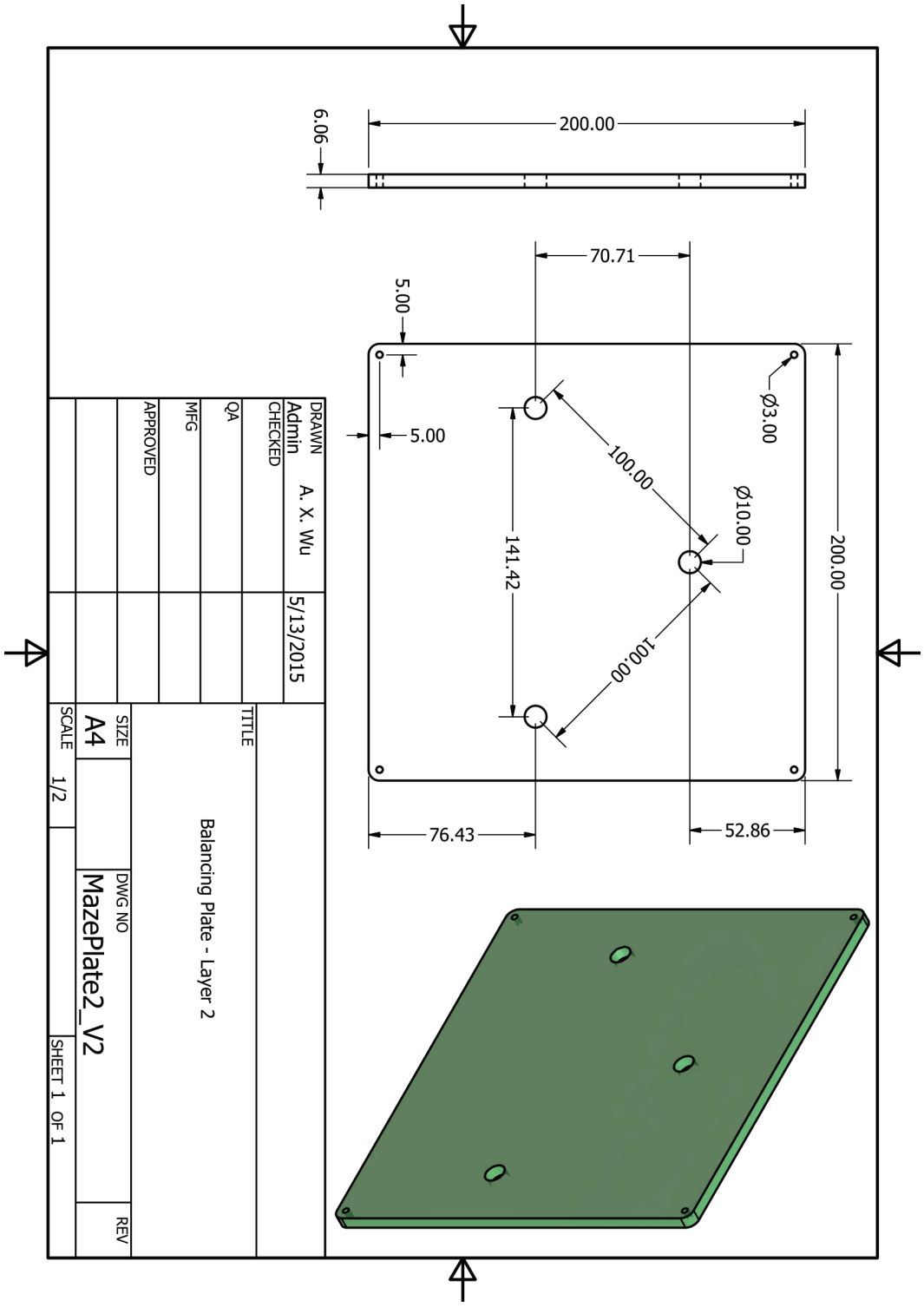

6.06
200.00
70.71
5.00
5.00
Ø3.00
100.00
Ø10.00
141.42
200.00
100.00
76.43
52.86
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Balancing Plate - Layer 2
SIZE
A4
DWG NO
MazePlate2_V2
REV
SCALE
1/2
SHEET 1 OF 1

Figure 35:

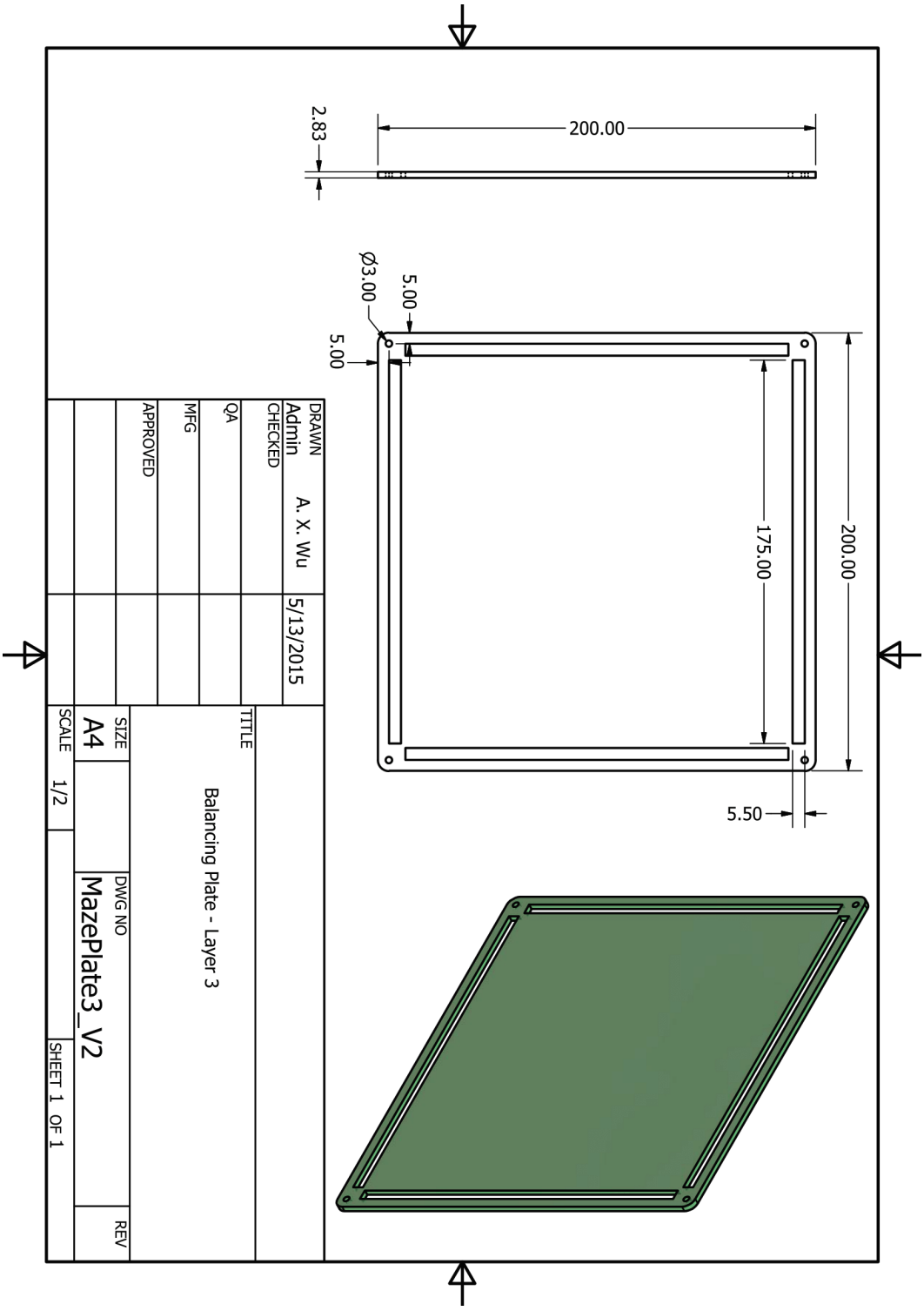

2.83
200.00
Ø3.00
5.00
5.00
175.00
200.00
5.50
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Balancing Plate - Layer 3
SIZE
A4
DWG NO
MazePlate3_V2
REV
SCALE
1/2
SHEET 1 OF 1

Figure 36:

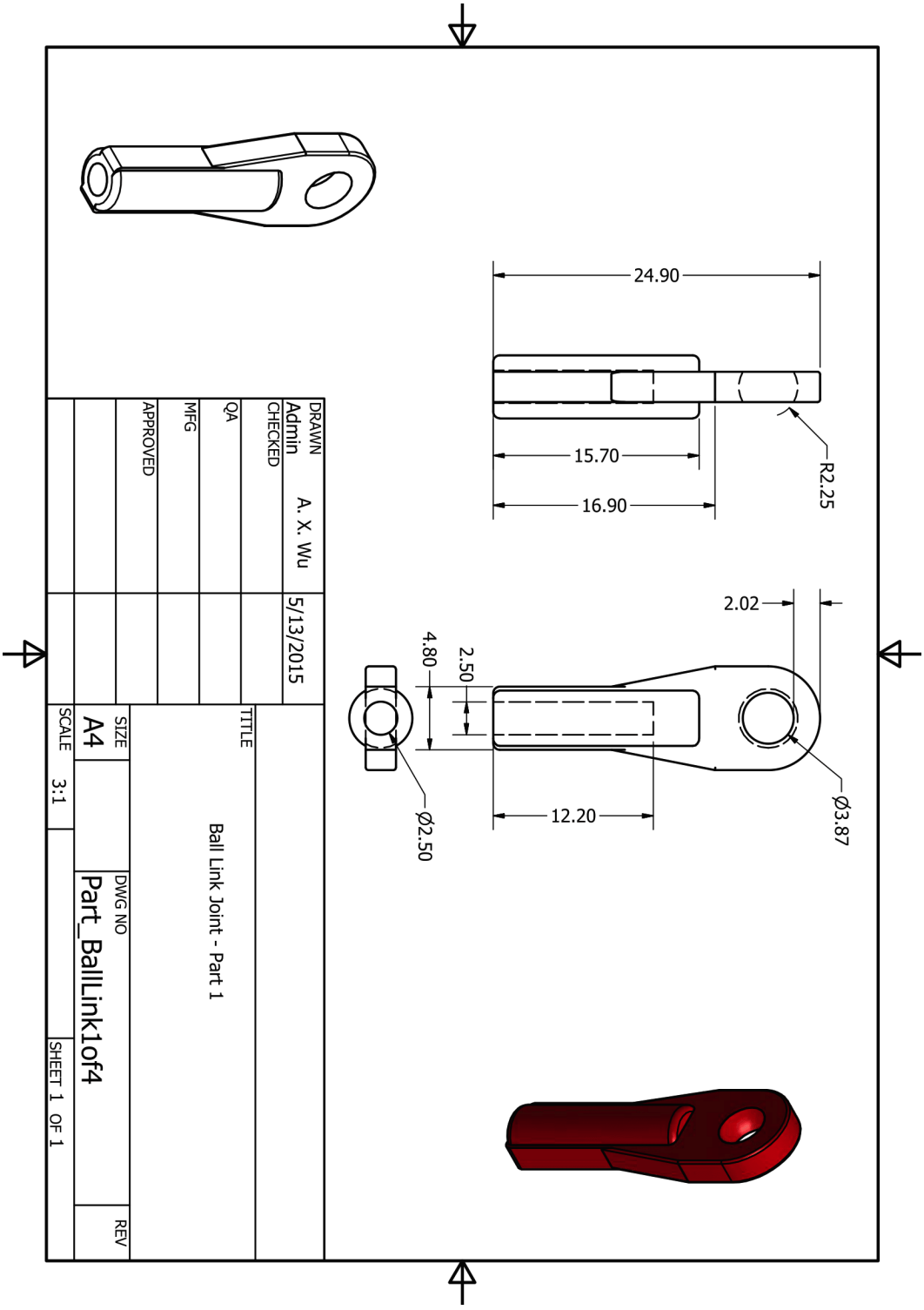

24.90
15.70
16.90
R2.25
2.02
4.80
2.50
Ø2.50
12.20
Ø3.87
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Ball Link Joint - Part 1
SIZE
A4
DWG NO
Part_BallLink1of4
REV
SCALE
3:1
SHEET 1 OF 1

Figure 37:

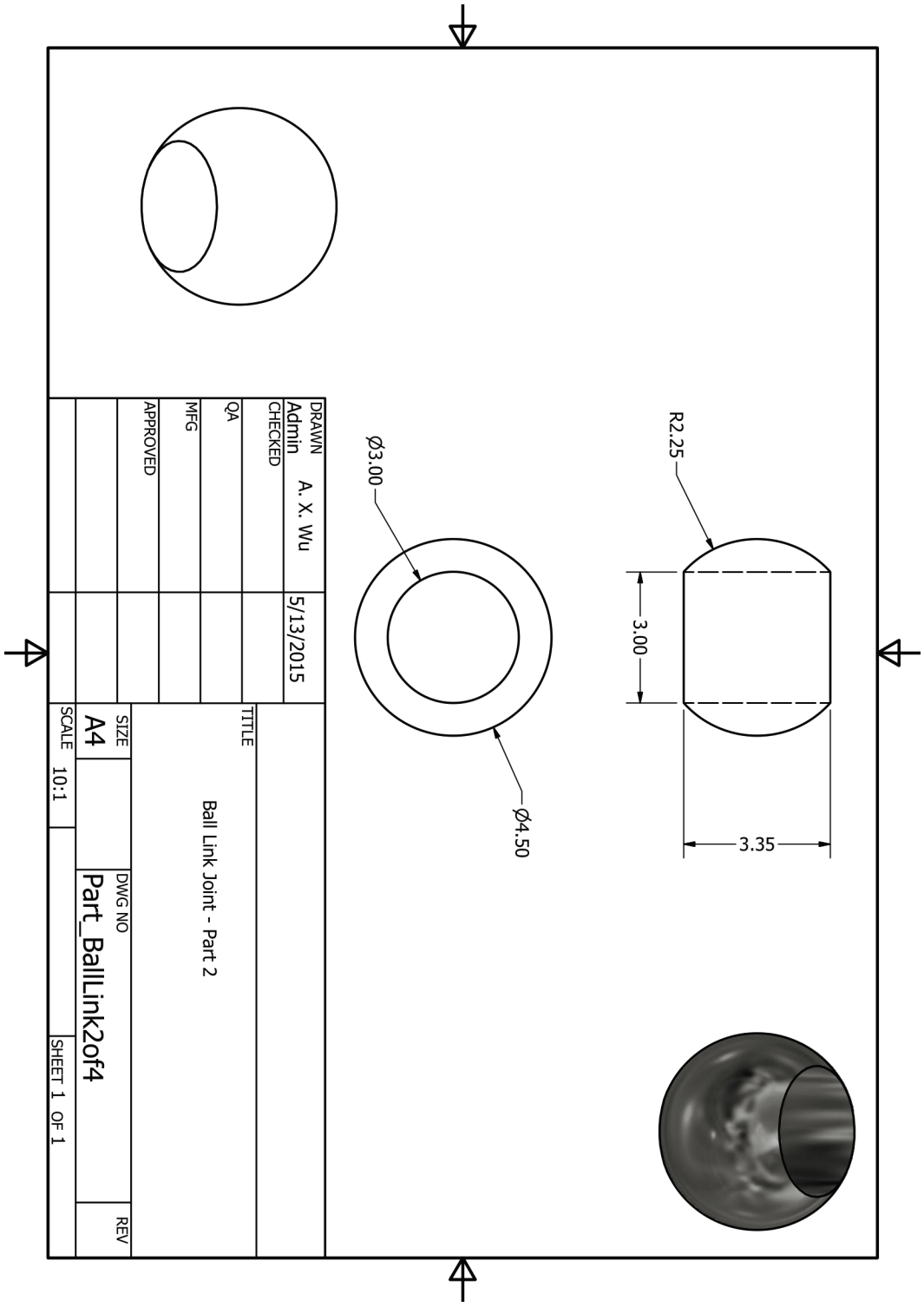

R2.25
3.00
3.35
Ø3.00
Ø4.50
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Ball Link Joint - Part 2
SIZE
A4
DWG NO
Part_BallLink2of4
REV
SCALE
10:1
SHEET 1 OF 1

Figure 38:

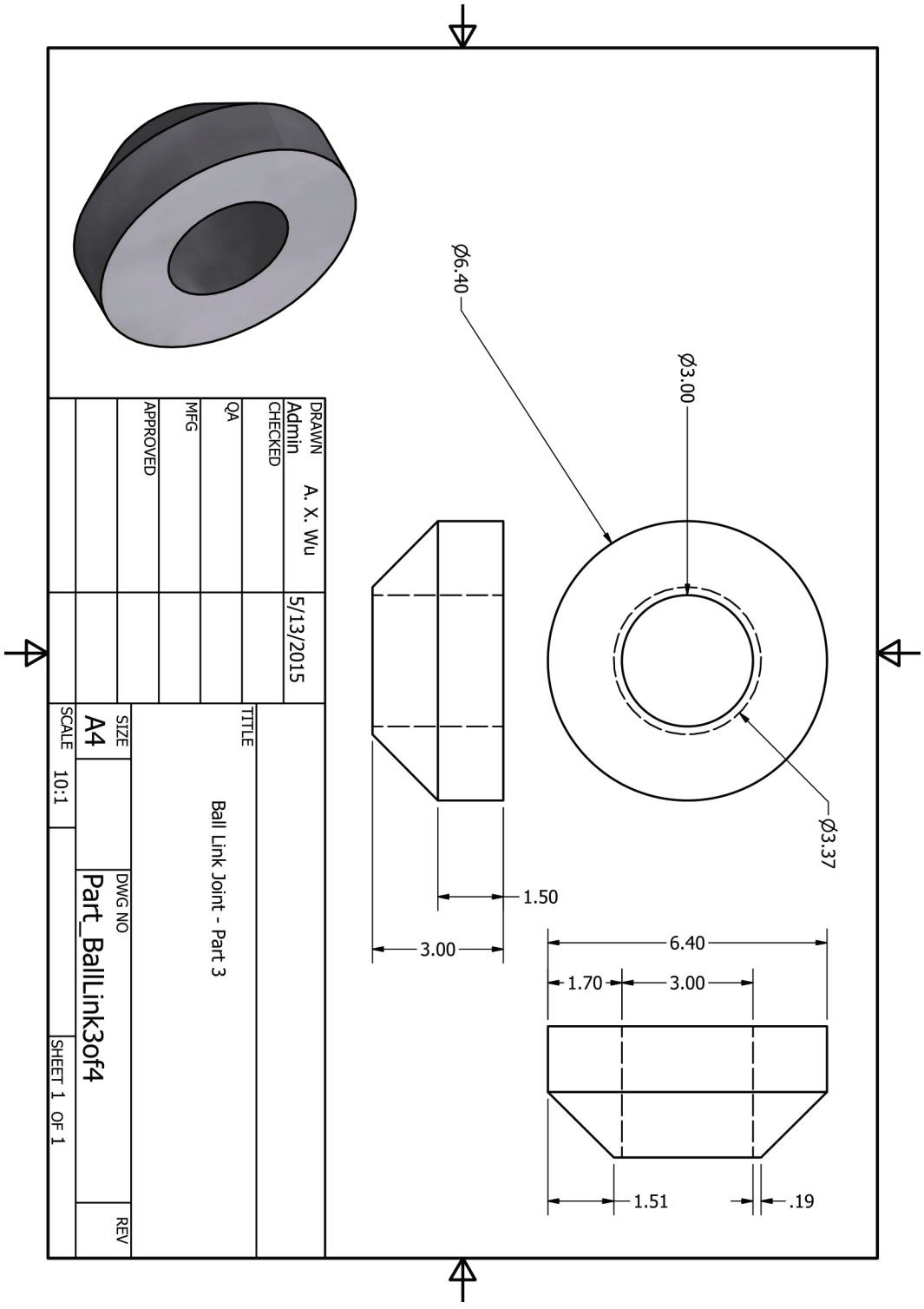

Ø6.40
Ø3.00
Ø3.37
1.50
3.00
6.40
1.70
3.00
1.51
.19
DRAWN
A. X. Wu
5/13/2015
Admin
CHECKED
QA
MFG
APPROVED
TITLE
Ball Link Joint - Part 3
SIZE
A4
DWG NO
Part_BallLink3of4
REV
SCALE
10:1
SHEET 1 OF 1

Figure 39:

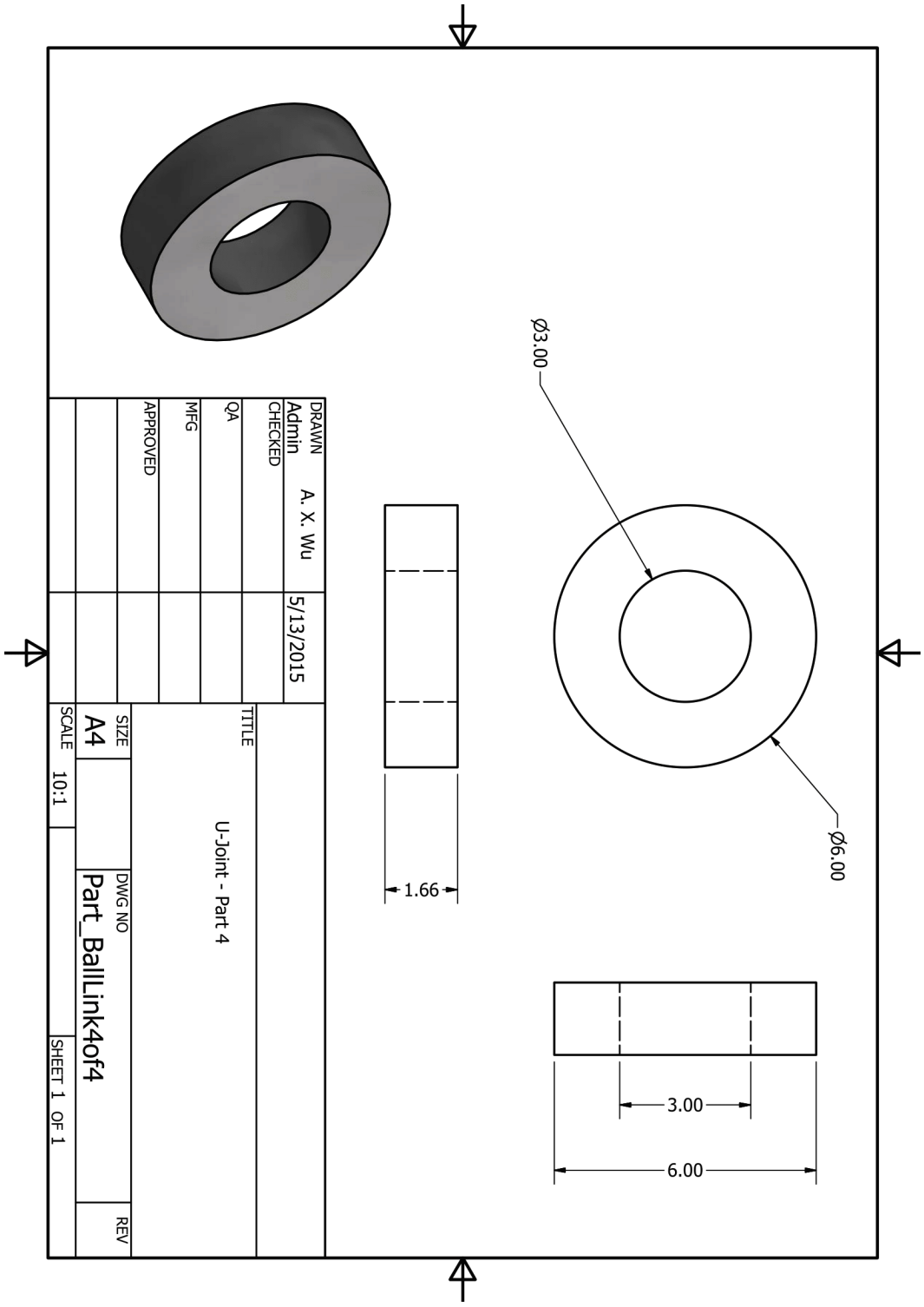

Ø3.00
Ø6.00
1.66
3.00
6.00
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
U-Joint - Part 4
SIZE
A4
DWG NO
Part_BallLink4of4
REV
SCALE
10:1
SHEET 1 OF 1

Figure 40:

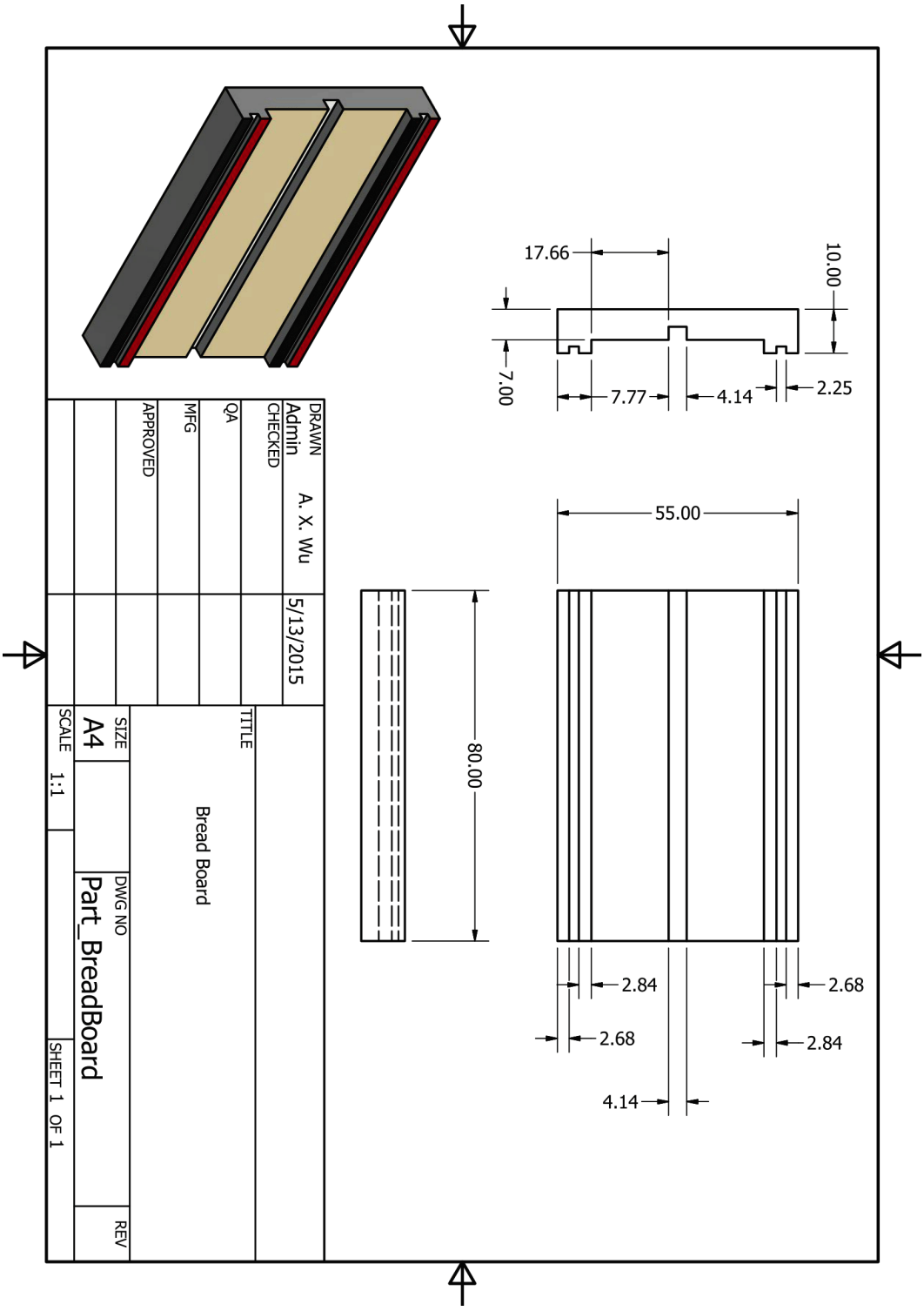

17.66
10.00
7.00
7.77
4.14
2.25
55.00
80.00
2.84
2.68
2.68
2.84
4.14
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
SIZE
A4
SCALE
1:1
TITLE
Bread Board
DWG NO
Part_BreadBoard
REV
SHEET 1 OF 1

Figure 41:

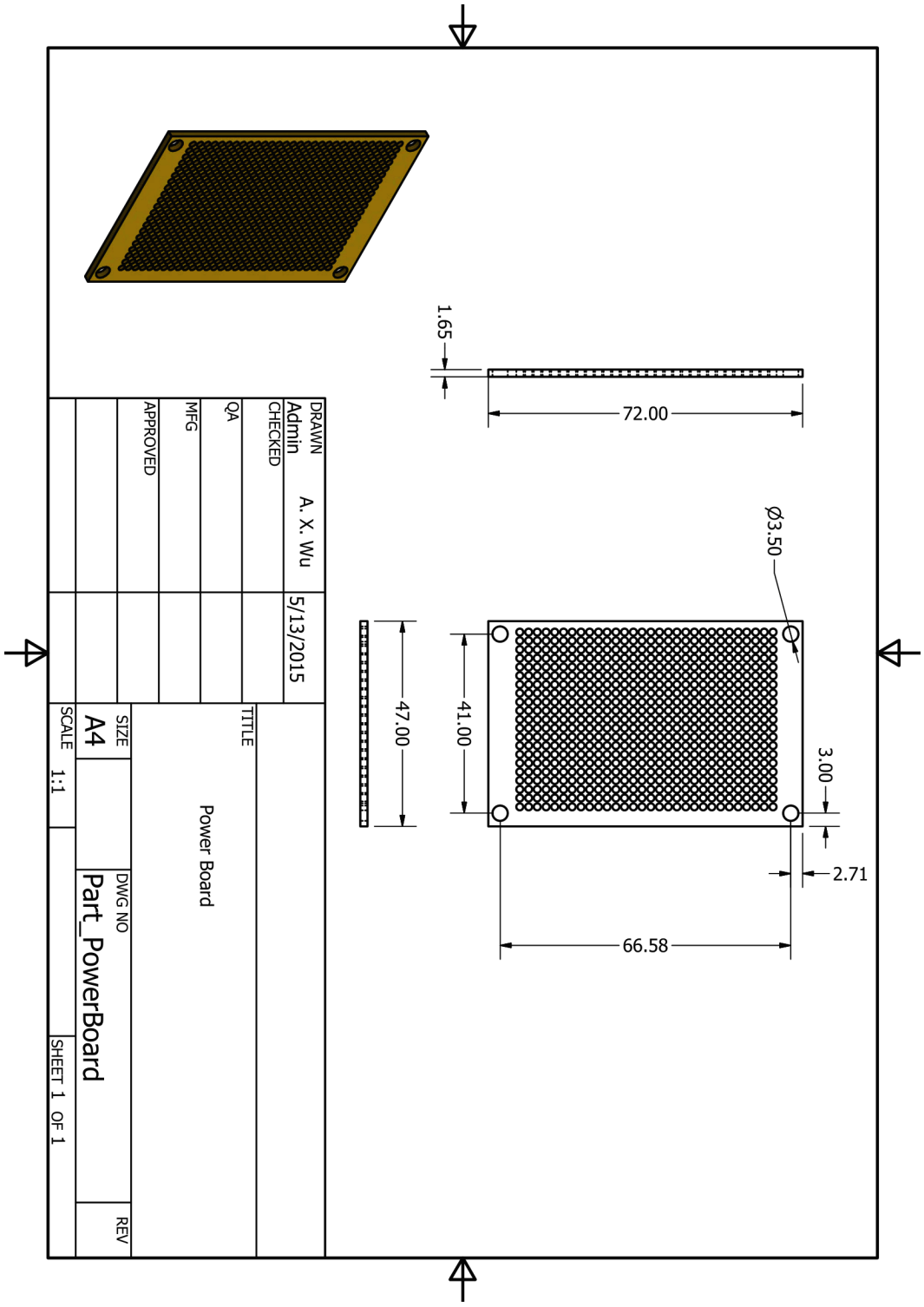

1.65
72.00
Ø3.50
47.00
41.00
3.00
2.71
66.58
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Power Board
SIZE
A4
DWG NO
Part_PowerBoard
REV
SCALE
1:1
SHEET 1 OF 1

Figure 42:

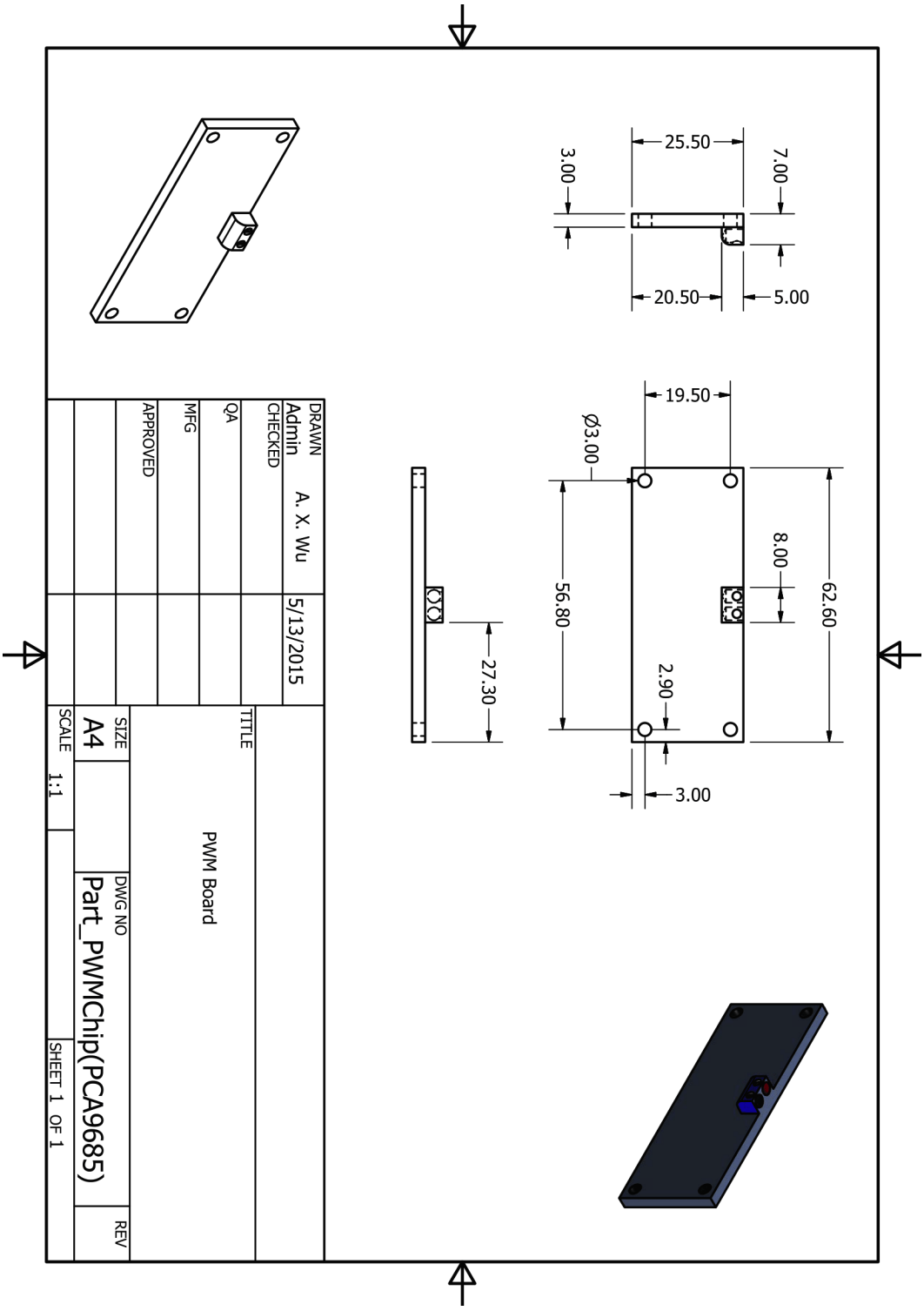

25.50
7.00
3.00
20.50
5.00
19.50
Ø3.00
8.00
62.60
56.80
2.90
3.00
27.30
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
PWM Board
SIZE
A4
DWG NO
Part_PWMChip(PCA9685)
REV
SCALE
1:1
SHEET 1 OF 1

Figure 43:

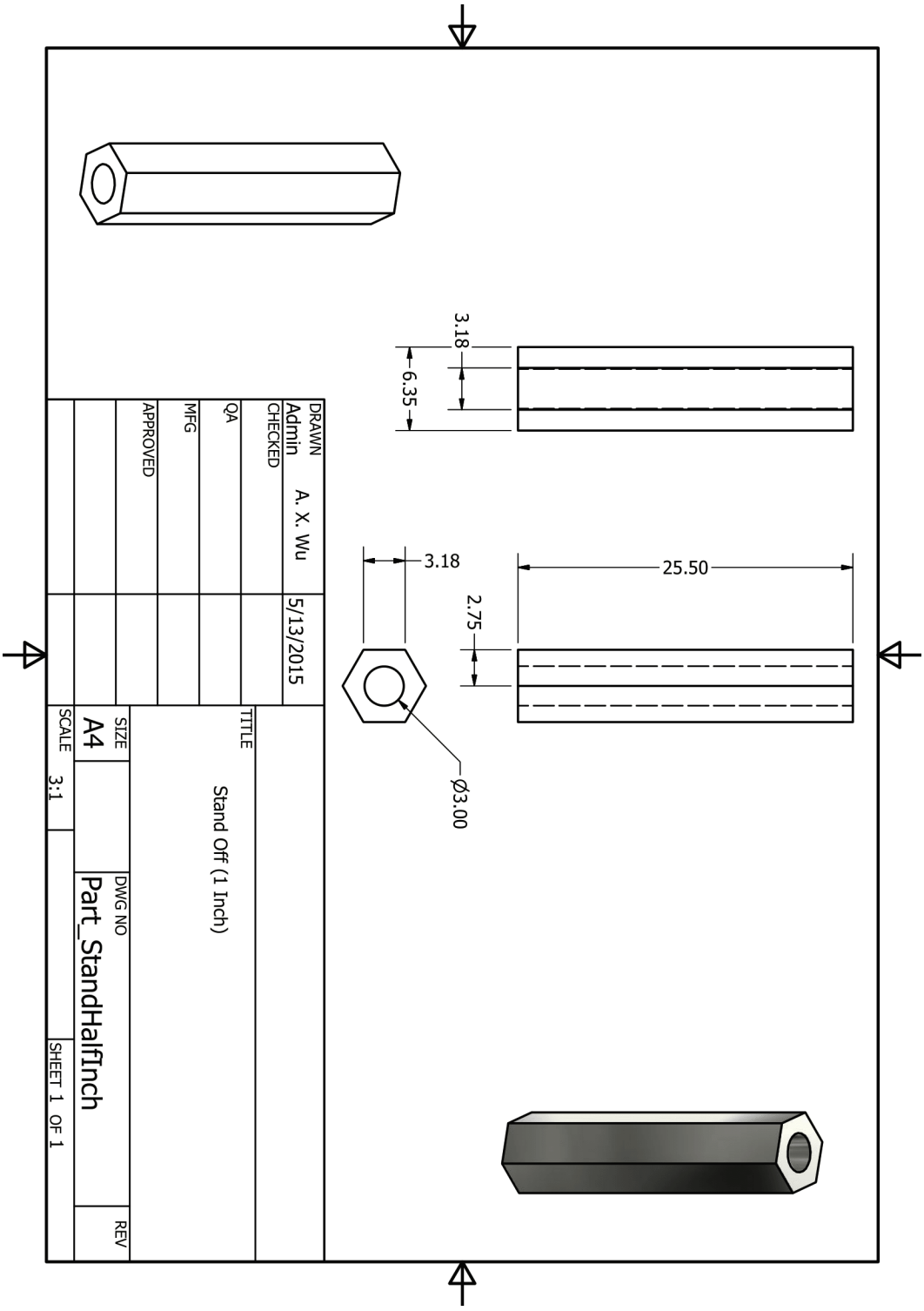

3.18
6.35
3.18
25.50
2.75
Ø3.00
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Stand Off (1 Inch)
SIZE
A4
DWG NO
Part_StandHalfInch
REV
SCALE
3:1
SHEET 1 OF 1

Figure 44:

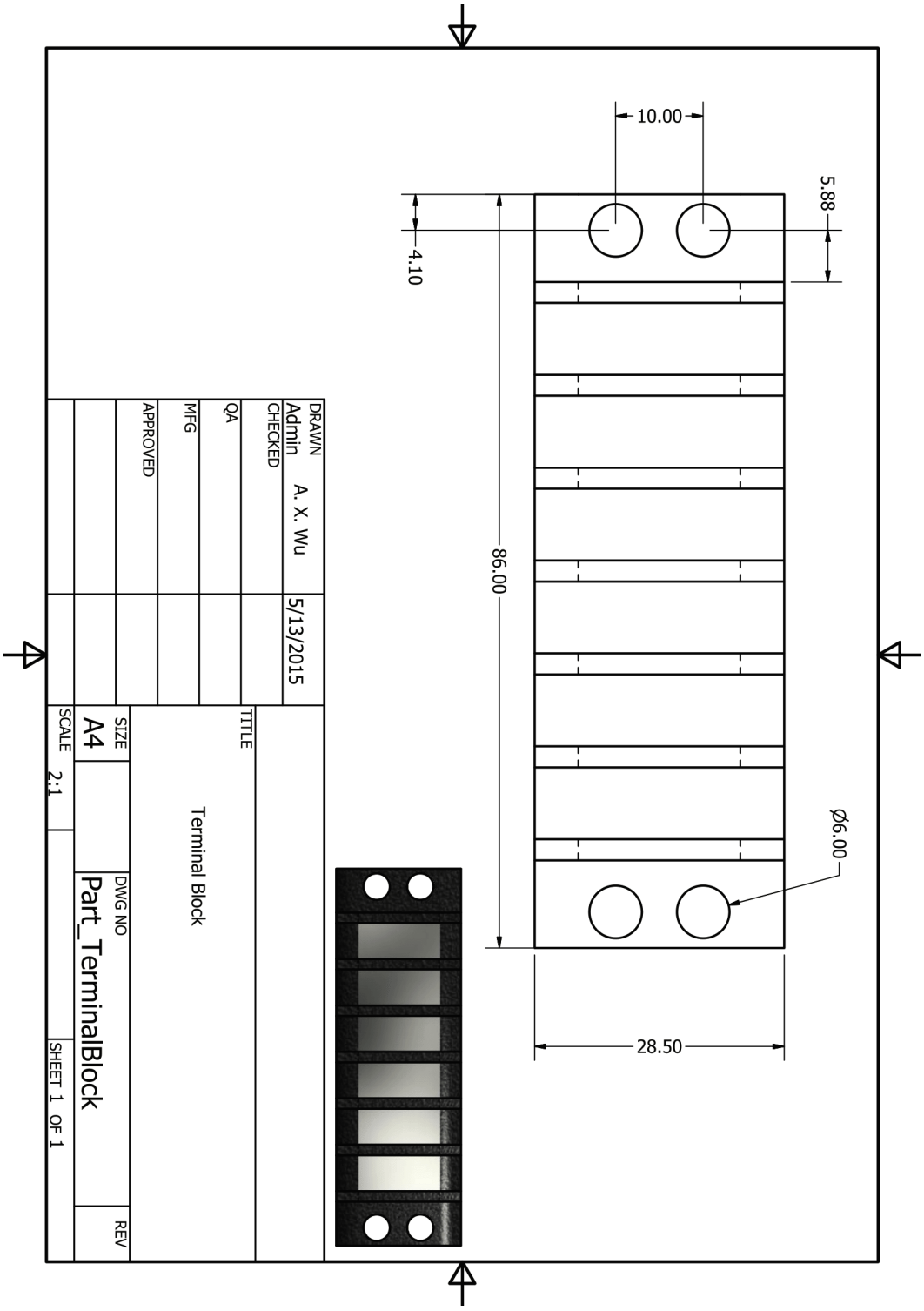

10.00
5.88
4.10
86.00
Ø6.00
28.50
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Terminal Block
SIZE
A4
DWG NO
Part_TerminalBlock
REV
SCALE
2:1
SHEET 1 OF 1

Figure 45:

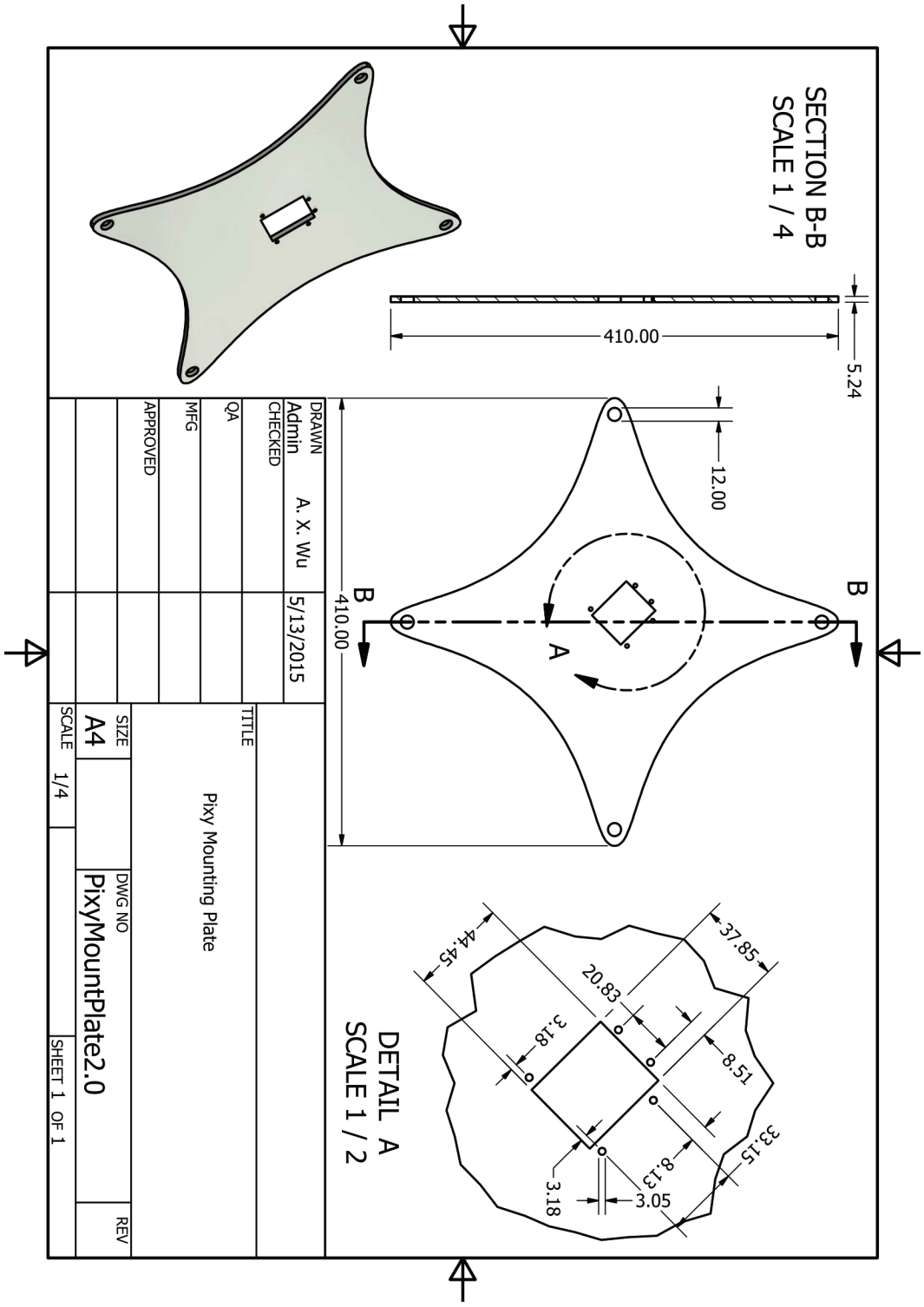

SECTION B-B
SCALE 1 / 4
410.00
5.24
12.00
410.00
B
B
A
DETAIL A
SCALE 1 / 2
44.45
37.85
20.83
3.18
8.51
33.15
8.13
3.18
3.05
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Pixy Mounting Plate
SIZE
A4
DWG NO
PixyMountPlate2.0
REV
SCALE
1/4
SHEET 1 OF 1

Figure 46:

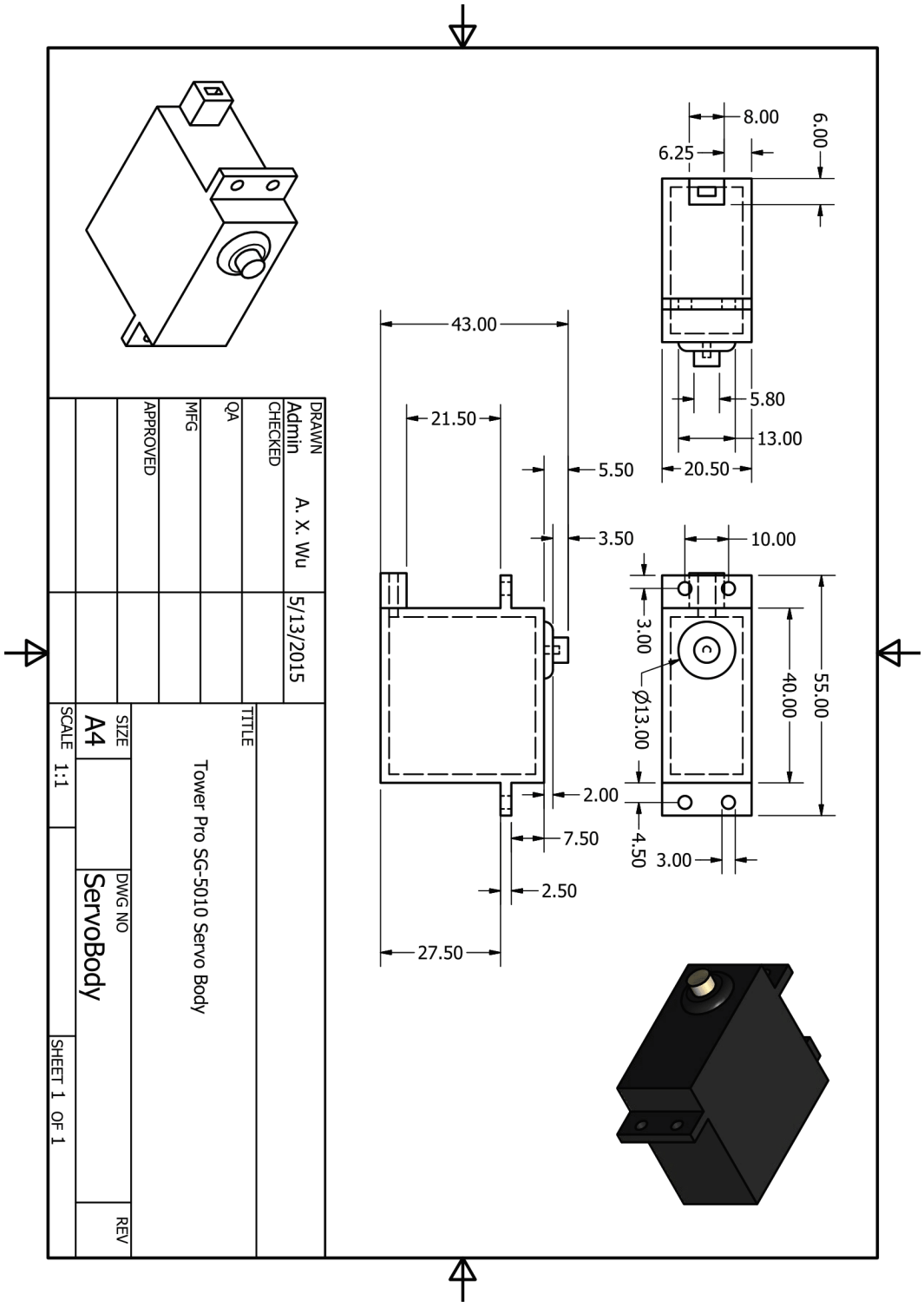

8.00
6.00
6.25
43.00
5.80
21.50
13.00
5.50
20.50
3.50
10.00
3.00
40.00
55.00
Ø13.00
2.00
7.50
4.50
3.00
2.50
27.50
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
SIZE
A4
SCALE
1:1
TITLE
Tower Pro SG-5010 Servo Body
DWG NO
ServoBody
REV
SHEET 1 OF 1

Figure 47:

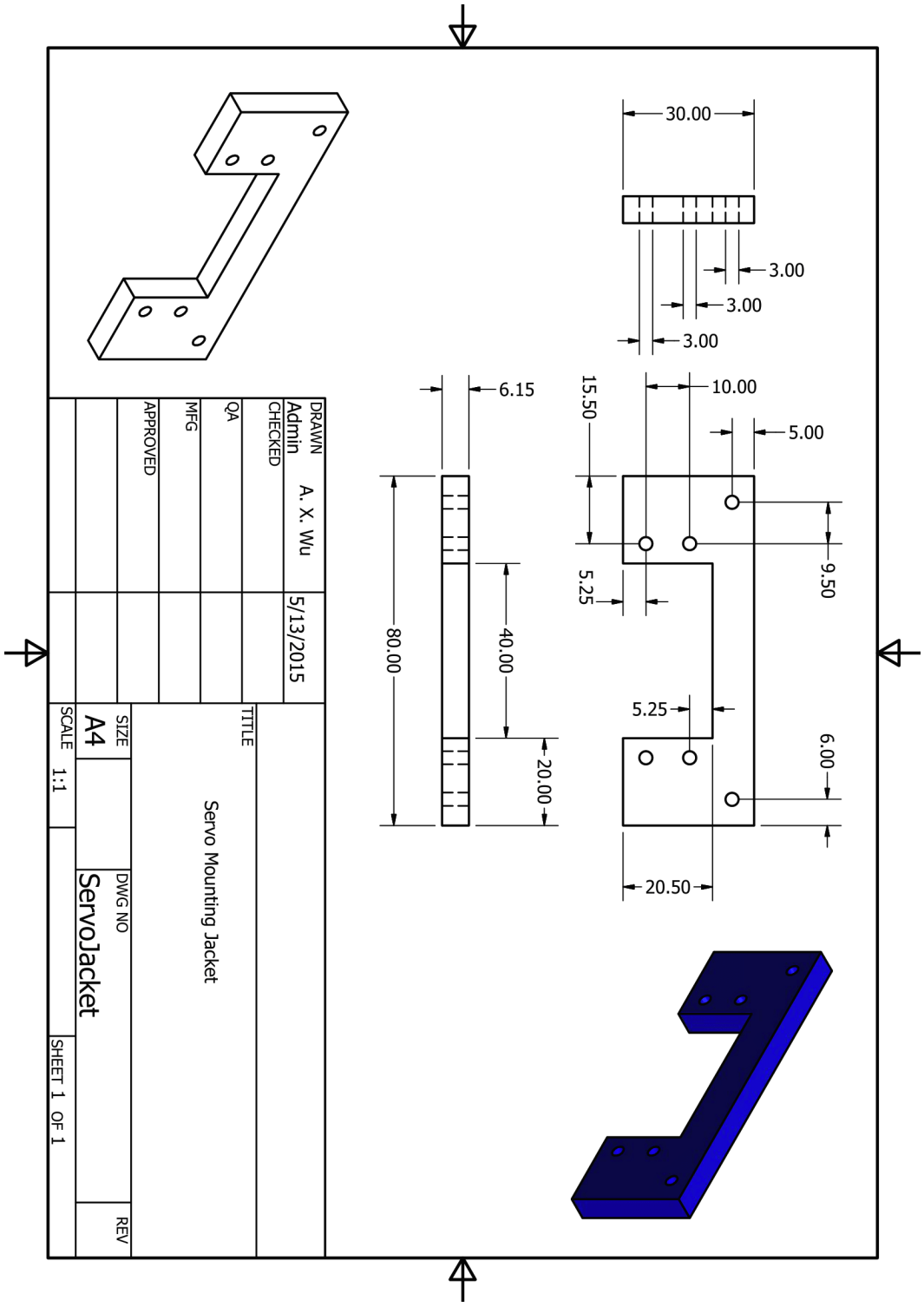

30.00
3.00
3.00
3.00
6.15
15.50
10.00
5.00
9.50
5.25
80.00
40.00
20.00
5.25
6.00
20.50
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Servo Mounting Jacket
SIZE
A4
DWG NO
ServoJacket
REV
SCALE
1:1
SHEET 1 OF 1

Figure 48:

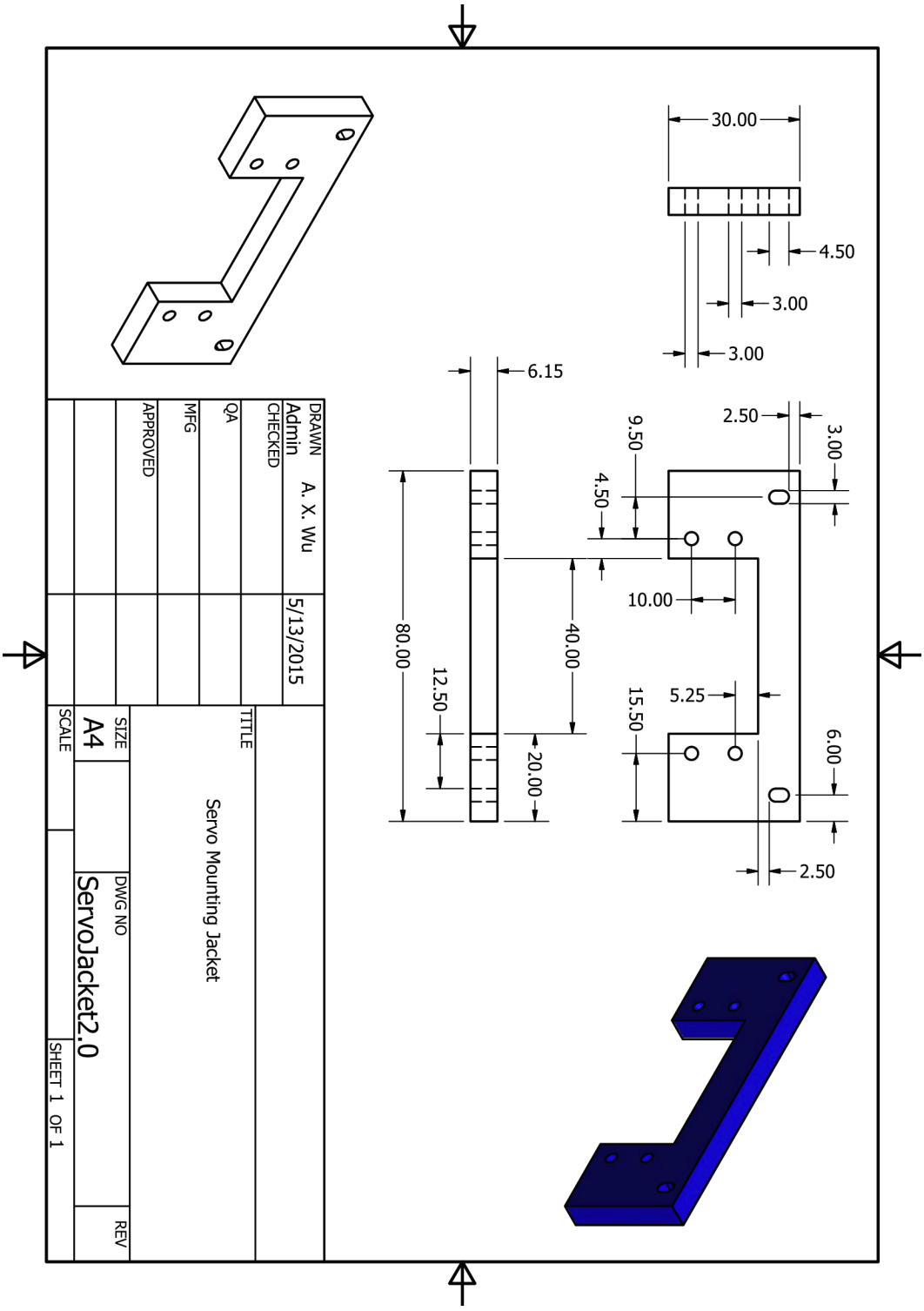

30.00
4.50
3.00
3.00
6.15
2.50
3.00
9.50
4.50
10.00
80.00
40.00
12.50
20.00
5.25
15.50
6.00
2.50
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Servo Mounting Jacket
SIZE
A4
DWG NO
ServoJacket2.0
REV
SCALE
SHEET 1 OF 1

Figure 49:

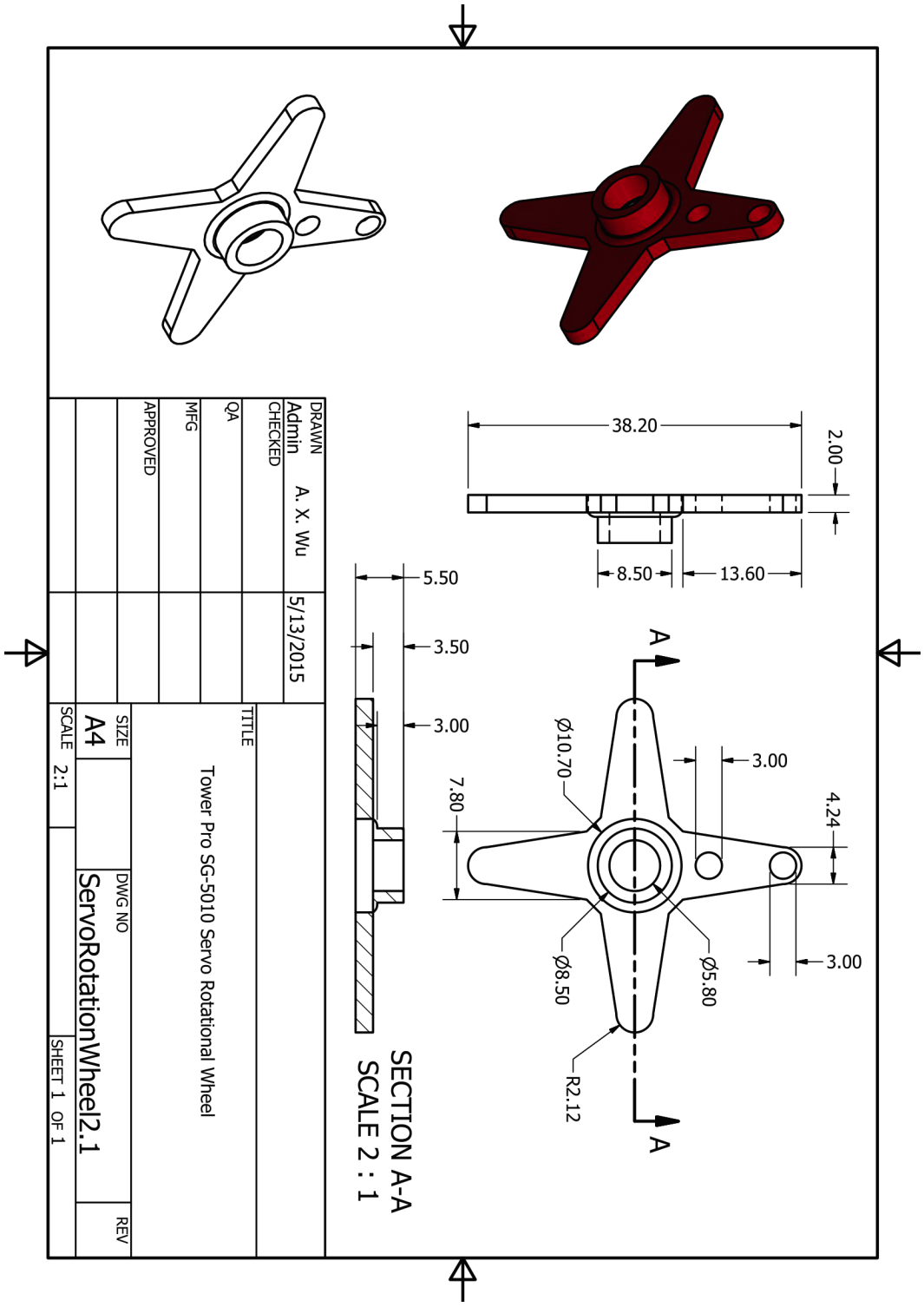

38.20
2.00
8.50
13.60
5.50
3.50
3.00
A
A
Ø10.70
7.80
3.00
4.24
3.00
Ø8.50
Ø5.80
R2.12
SECTION A-A
SCALE 2 : 1
DRAWN
Admin
A. X. Wu
5/13/2015
CHECKED
QA
MFG
APPROVED
TITLE
Tower Pro SG-5010 Servo Rotational Wheel
SIZE
A4
DWG NO
ServoRotationWheel2.1
REV
SCALE 2:1
SHEET 1 OF 1

Figure 50: