Grand Valley State University
The Padnos School of Engineering

SEGWAY DESIGN PROJECT

EGR 345 Dynamic Systems Modeling and Control

Instructor : Dr. Jack

Team 8
Mitch Burkert
Taylor Groll
Thao Lai
Tyler McCoy
Dennis Smith

December 3, 2004

Executive Summary

A one-fifth scaled and self-contained prototype of a Segway, or two-wheeled self-balancing cart, was designed and built with line following and mass transportation capabilities. A 0.975 kg, 6 inch tall with 6 inch wheel diameter design utilizing polycarbonate materials was developed with the consideration of competition constraints. Final costs totaled to be $120.17. For the purpose of aesthetics and identifying the Segway, handlebars were produced but were not included in the final mass of the cart or attached during the actual competition. A control system diagram, a system architecture diagram, and an electrical schematic were used to organize and develop a control program in C language for the Segway. Three photoresistors were used to detect a line while infrared sensors were mounted to detect the body's angular position from the floor. The final competition involved a series of tests where the carts balanced with and without weight, followed straight and curves paths of standard black electrical tape while carrying a 200 g mass. Each of the tests during the competition was performed successfully.

Table of Contents

Executive Summary
Table of Contents 2
1. Design Description 4
1.1 Design Constraints 4
1.2 Mechanical Design 4
1.2.1 Calculations 6
1.3 Software Design 9
1.3.1 Control System 9
1.3.2 System Architecture 11
1.4 Electrical Design 12
2. Inventory 13
2.1 Budget and Bill of Materials 13
2.2 Mass Inventory 14
3. Test Results 15
3.1 Scilab Simulation 15
3.2. Score Estimate 17
4. Conclusion 17
5. References 19
6. Appendices 20
6.1 Appendix A: Prints 20
6.1.1 Exploded Assembly 21
6.1.2 Top Plate 22
6.1.3 Side Plate 23
6.1.4 Bottom Plate 24
6.1.5 Sensor Bracket 25
6.1.6 Photoresistor Mount 26
6.1.7 Wheel 27
6.1.8 Axle Part 28
6.2 Appendix B: Calculations 29
6.2.1 State Equations 29
6.2.2 Motor torque 32
6.2.3 Wheel Stress 33

6.2.4 Deformation and Strain 34
6.3 Appendix C: Controller Program in C 35
6.4 Appendix D: Scilab Simulation Program 47
6.5 Appendix E: Purchasing Evidence 50

1. Design Description

1.1 Design Constraints

The Segway design had to be self contained and turned on and off by a switch. Thus, the batteries, or power source, must be on the cart itself. The container that was used as the mass in the competition had a 6 inch diameter base, so the cart had to be designed with a top plate large enough to hold the container. The maximum distance of the top plate from the floor had to be less than 6 inches. This height does not include the vertical handlebar post which had to be removable and between the lengths of 7 to 9 inches. The cart also had to be rigid enough to hold and balance a mass of up to 2 kg. The two cart wheels had to have diameters between 0.5 and 6 inches. To test the balance of the cart, it was necessary to enable a maximum free range of motion to 60 degrees in both directions of the wheel axle.

The cost of the finished cart could not exceed the specified budget of $200.00 and the weight of the cart should not have exceeded 1 kg. The incentive to keep costs and weight low was included into the final score estimate with the score improving as those parameters were reduced. Another important component of calculating the final score was the time to execute a motion from the start of the motion to the end. The tape paths that the robot had to follow during the competition grew increasingly complex to test the cart's agility and success of design.

1.2 Mechanical Design

A 0.975 kg, 6 inch tall with 6 inch wheel diameter design was developed with consideration of the competition constraints. The polycarbonate-framed cart was designed with a two 12V DC direct drive motor system in which the shafts of the motors acted as the wheel axles. Through circuit calculations, a 9V battery was identified as suitable to power the 8-bit ATMega32 board used to control the functions of the cart. Nine AAA batteries were calculated to supply enough power to drive the motors.

Photoresistors were used to detect the black electrical tape path on the day of the competition and infrared sensors were incorporated into the design to act as balance sensors.

For the purpose of aesthetics and identifying the Segway, handlebars were produced but were not included in the final mass of the cart or attached during the actual competition. The final cart design is shown in Figure 1.

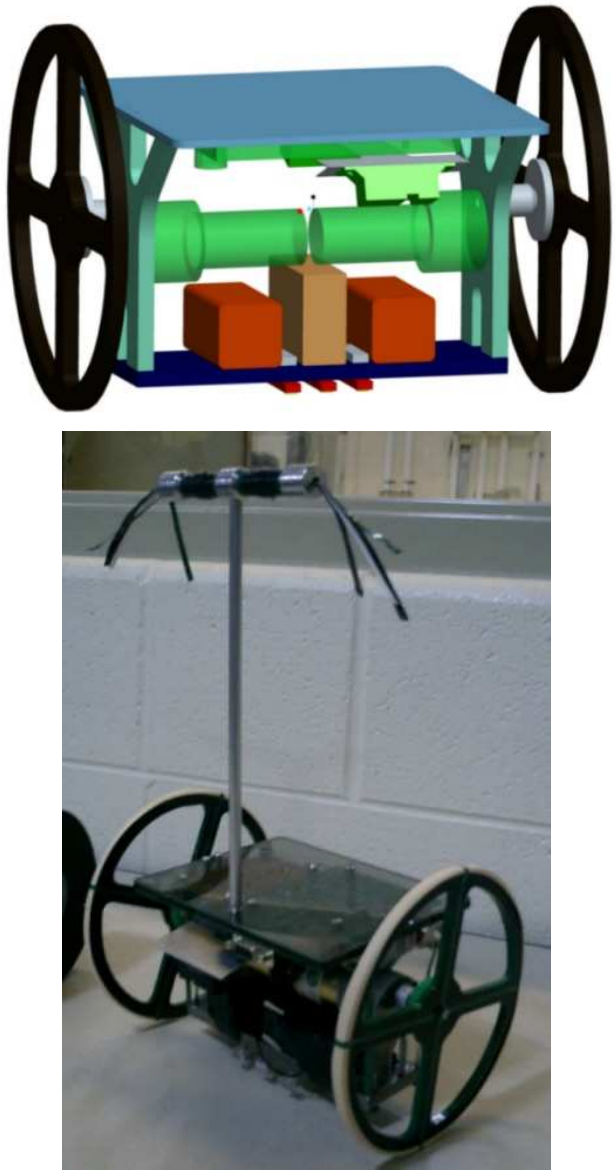
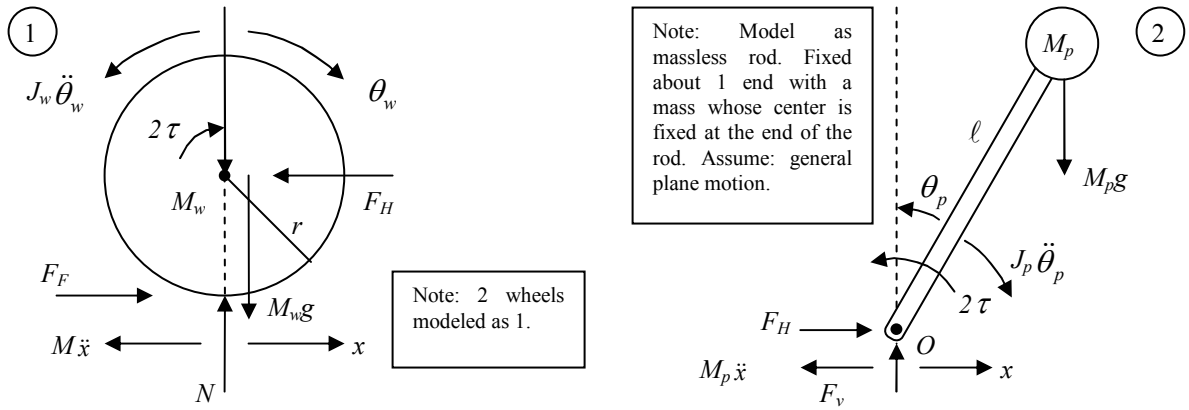


Figure 1: Final Segway design

1.2.1 Calculations

State equations were developed to model the system, using the wheel and cart free body diagram shown in Figure 2.



where,

J_w = wheel rotational inertia
M_w = mass of both wheels
θ_w = angle that wheel turns
r = wheel radius
x = horizontal distance cart moves
F_H = horizontal force
F_F = friction force
N = normal force

J_p = pendulum rotational inertia
M_p = mass of rod or pendulum
θ_p = angle that pendulum moves
ℓ = distance from wheel axle, O
τ = torque
F_v = vertical force
g = gravity

Figure 2: Wheel and cart free body diagrams

The final state equations were,

$$\dot{x} = v$$

$$\dot{v} = \left(\frac{= K^2}{R\left(r^2 M_p + J_m + r^2 M_w\right)}\right)v + \left(\frac{Kr}{R\left(r^2 M_p + J_m + r^2 M_w\right)}\right)V_s$$

$$\dot{\theta}_p = \omega_p$$

$$\dot{\omega}_p = \frac{-g\sin\theta_p}{\ell^2} + \left(\frac{M_w r}{2M_p\ell^2} - \frac{J_m}{rM_p\ell^2}\right)\dot{v} + \left(\frac{-K^2}{rRM_p\ell^2}\right)v + \left(\frac{K}{RM_p\ell^2}\right)V_s$$

The motor torque needed to maintain a mass position and the torque in the Beuhler motors were calculated to verify whether they were sufficient. The maximum motor torque needed to support an inverted pendulum, such as the Segway, can be calculated using the moments in the model in Figure 3.

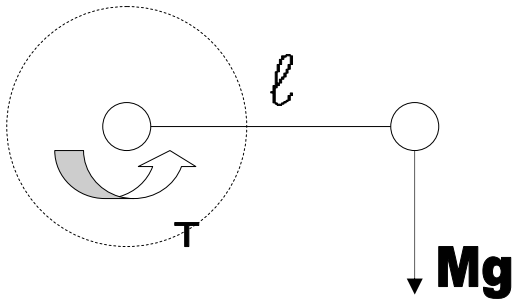


Figure 3: Free Body Diagram of Segway

With the free body diagram of the Segway, the calculated torque that was needed to keep the Segway balanced was 0.81 Nm. It was calculated that the Beuhler motors supplied 1.2 Nm of torque to the wheels. Thus, the Beuhler motors were adequate for the design. These calculations are found in Appendix B.

The material and design of the wheels had to withstand the mass of the cart chassis and the 2 Kg load that was applied to it during the competition. By calculating the amount of stress the wheels would experience, and comparing that stress to the yield stress in the chosen polycarbonate material, the material was verified sufficient for the design. A free body diagram of a wheel spoke, Figure 4, displays where the wheels experienced a load stress.

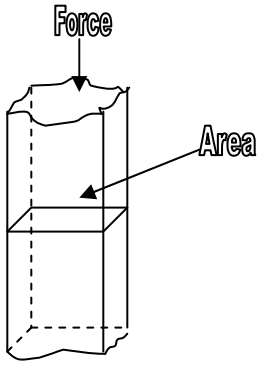


Figure 4: Wheel spoke model

The shear stress in the wheel was calculated to be 0.270 MPa. Appendix B demonstrates the systematic calculations for computing that value. Since the yield strength of polycarbonate material was 72 MPa, the polycarbonate material was calculated to be adequate for holding the weight of the cart and the testing load, with a safety factor of 266, also calculated in Appendix B [4].

The Buehler 12VDC gearhead motors that required at least 500 mA and 12 volts to operate at full potential. Nine AAA batteries were used in series to supply a maximum of 13.5 volts to both gearhead motors, shown in Figure 5. The AAA batteries were rated at 1250 mAh, which make them better suited to handle the two gearhead motors than the 9-volt batteries that were rated at 625 mAh. The weight of the cart was reduced by using AAA (11.5g) instead of AA (23g), which were rated at 2850 mAh. The risk of running excessive current through the motor driver chip was also reduced [2].

A 9-volt battery was used to power the ATMga32 microcontroller. This power source was separate from the source for the motors to allow the operator to load the program without risking an accidental fall caused by sudden power to the motors.

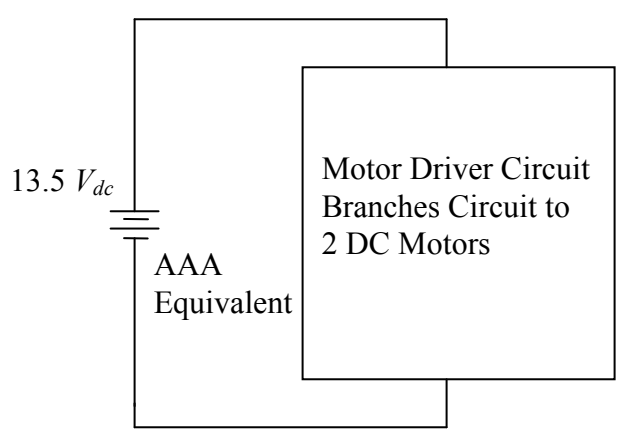


Figure 5: AAA equivalent power supply

1.3 Software Design

1.3.1 Control System

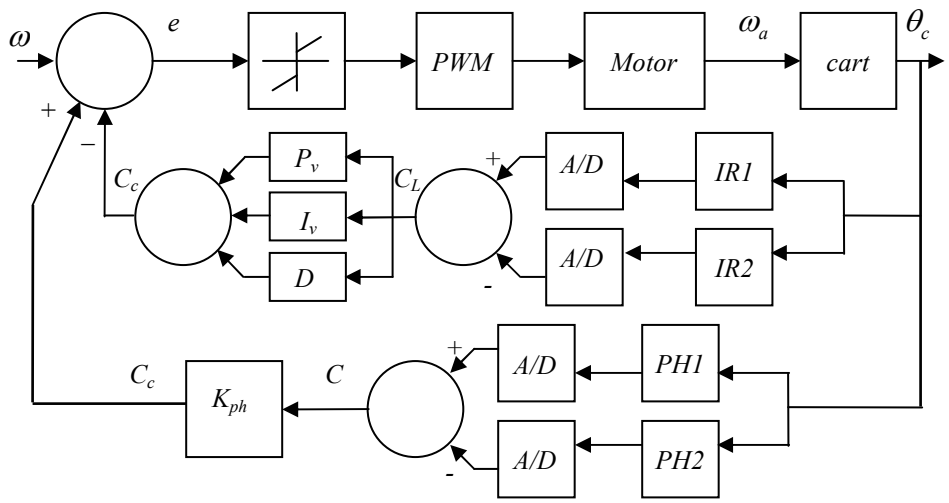
The system developed to control a two motor driven self-balancing cart is in Figure 6. The ATMega32 microcontroller was used to control the cart motion with the C program found in Appendix C. The program was designed to control the cart motion through evaluating differences between sensor and photoresistor voltage values. It was also written so that the function responsible for balancing the cart would take precedence over all other movement functions. Therefore, any tipping that occurred while the cart was in motion would have caused the balancing program to impede forward progress.

To eliminate deadband from friction, the deadband limits that were used for the motors were around 125 in all positive and negative static and kinetic. The PWM signal was effectively divided into three separate parts: balancing, turning, and forward motion. Each of the functions was allowed a percentage of the PWM, depending upon the importance and precedence of the function. The balancing function was allowed use of the entire PWM, if necessary, since the complete over-turning of the cart would have resulted in failure.

While the cart was in motion, the two photoresistors on its underside produced a voltage. The A/D converter read the difference in voltage value between the two sensors. This value changed as the photoresistors detected the dark tape, a high voltage, or comparatively light floor, a lower voltage.

The angular position of the cart was detected using two infrared sensors mounted at a 45 degree angle approximately 3.25 inches from the ground, one on the front of the cart and one on the rear. The A/D converter read the voltage difference between the two sensors and variable gain was implemented to adjust motor speed depending on the magnitude of voltage difference. By comparing the voltage values between each sensor type, it was found that encoders were unnecessary. Encoders would have compared the change in position between the two motors to adjust the cart for following a line. However, the photoresistors already provided a feedback system that allowed the cart to follow a line. The

voltage difference for both the IR sensors and photoresistors was continually checked every interrupt, or every 10 milliseconds.



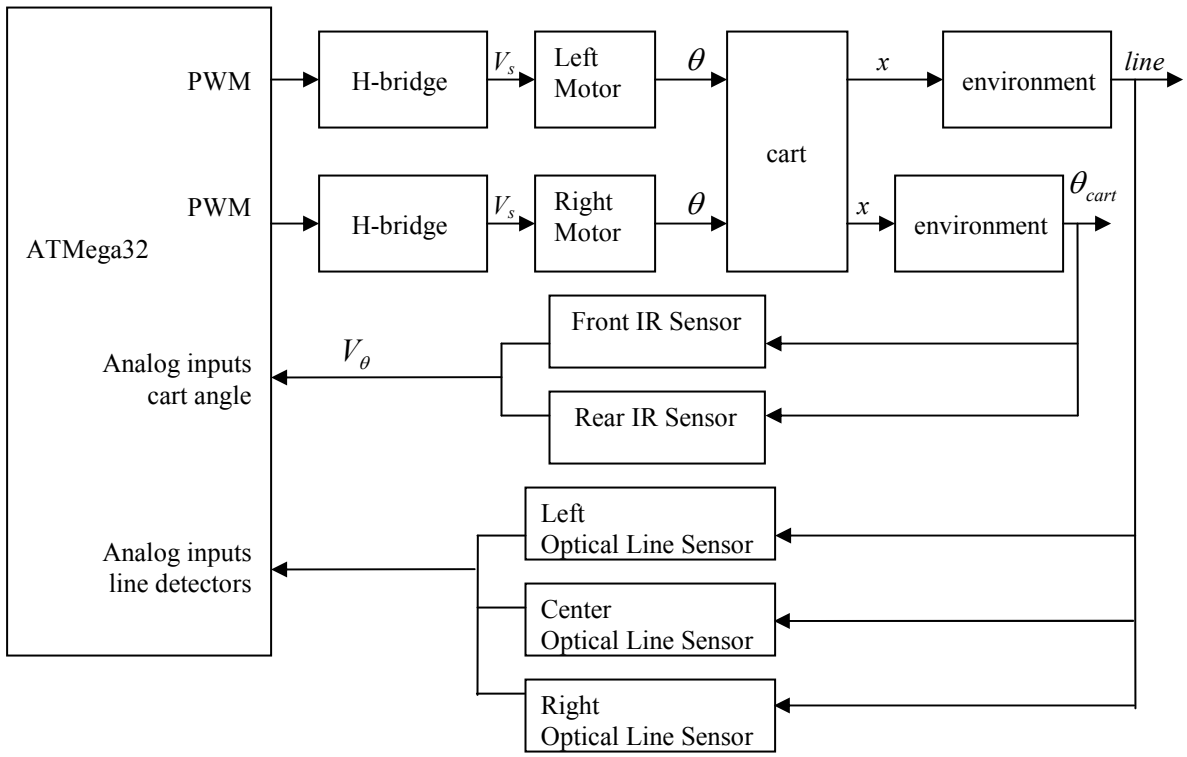
where,

e = error	V_s = voltage source
PWM = pulse width modulator	ω = set angular velocity
ω_a = angular motor speed	θ_c = cart angular position
$IR1$ = sensor 1 voltage	$IR2$ = sensor 2 voltage
$PH1$ = photoresistor 1 voltage	$PH2$ = photoresistor 2 voltage
A/D = analog to digital converter	V_L = voltage difference
P = variable proportional gain	I = variable integral gain
D = differential gain	C = integer value
P_{ph} = variable proportional gain for photoresistor	
C_L = integer value for cart angle	C_c = output command

Figure 6: Control system block diagram for line-following Segway [3]

1.3.2 System Architecture

The control software that modeled the control block system diagram in Figure 6 was designed with the system architecture in Figure 7 as the skeleton.



where,

V_s = voltage source

x = position cart moved

θ_{cart} = angular cart position

θ = angular motor position

$line$ = detected black tape line

V_θ = angular position voltage

Figure 7: System architecture for two identical motors [3]

1.4 Electrical Design

The electrical schematic of the Segway control system, shown in Figure 8, includes the ATMega32 microcontroller, an L293 H-bridge push-pull four channel driver chip, photoresistors, and IR sensors as inputs.

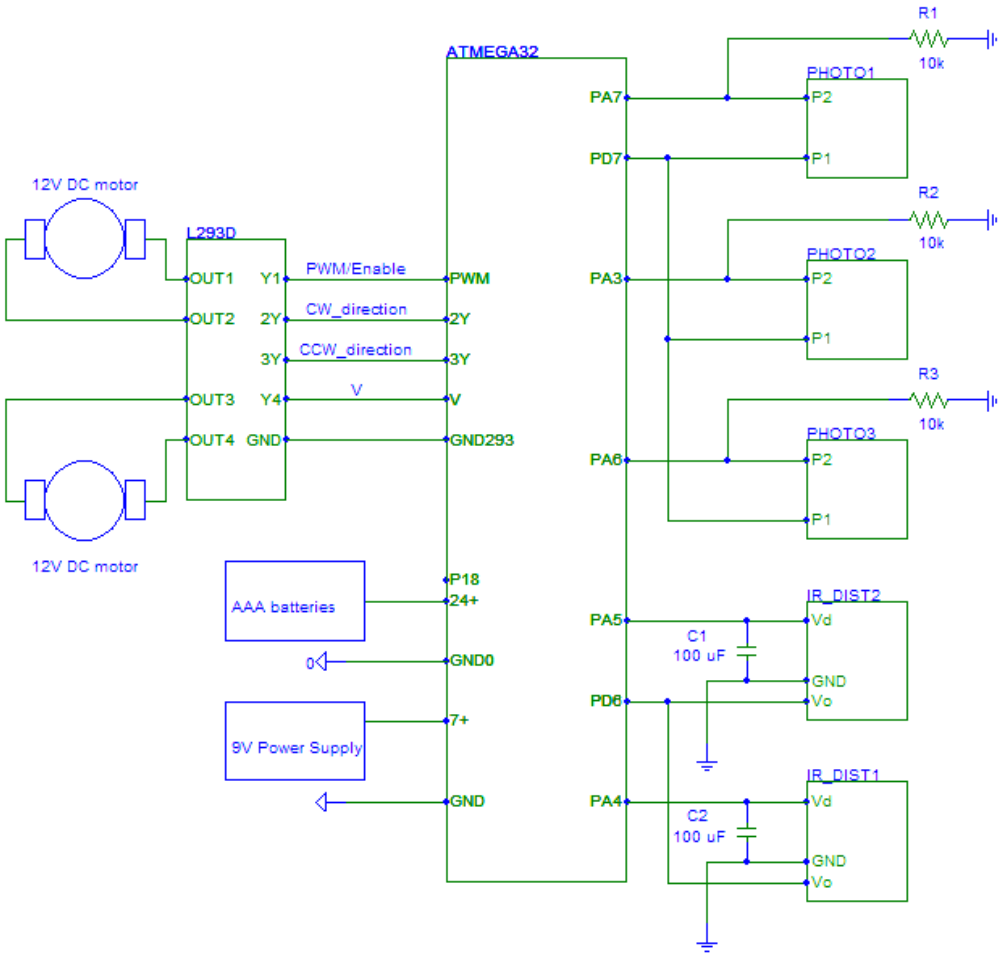


Figure 8: Segway electrical schematic

2. Inventory

2.1 Budget and Bill of Materials

The maximum allowed budget for each cart in the competition was $200.00. In order to ensure that the cost was an important design aspect, it was a factor in the final score equation. Table 1 is the bill of materials and Table 2 is showing the budget components and that was kept current throughout the production of the Segway. Sources of these materials can be found in References and on the receipts in Appendix E [5][6][7].

Table 1: Bill of Materials

Part Description	**Qty.**
Cart Parts	
Wheel	2
Frame (Bottom)	1
Frame (Side)	2
Frame (Top)	1
Sensor Support	1
9 Volt Battery	1
AAA Batteries	9
AA Battery Case	1
AAA Battery Case	4
Controller Parts	
Distance Measuring Sensor	2
Atmega32 Board	1
Photoconductive Cell	3
Gearhead Motors	2
Consumables	
Screws	28
Nuts	12
Washers	12
Resistors (10kΩ)	3
Capacitors (100μF)	2
Switches	2
Velcro	1

Table 2: Budget

Part Description	Qty.	Manufacturer/ Supplier	Part Number	Unit Cost ($)	Total Cost ($)
Cart Parts					
Polycarbonate (12x24) 0.25" thick	1	McMaster-Carr	N/A	$15.06	$15.06
22 gage aluminum sheet	1	N/A	N/A	$7.29	$7.29
9 Volt Battery	1	GVSU	N/A	$2.00	$2.00
AAA Batteries	9	Radioshack	2300820	$0.60	$5.40
AA Battery Case	1	Radioshack	2700401	$0.99	$0.99
AAA Battery Case	4	Radioshack	2700398	$0.99	$3.96
Controller Parts					
Distance Measuring Sensor	2	Sharp	GP2D120	$8.25	$16.50
3-pin JST Sensor Cable	2	Sharp	N/A	$1.10	$2.20
Atmega 32 Board	1	Atmel	N/A	$37.00	$37.00
Photoconductive Cell	2	Digikey	PDV-P8001-ND	$0.57	$1.14
Gearhead Motors	2	N/A	N/A	$10.00	$20.00
Switches	2	Radioshack	2750645	$2.99	$5.98
Consumables					
Screws	28	GVSU	N/A	$0.05	$1.40
Nuts	12	GVSU	N/A	$0.05	$0.60
Washers	12	GVSU	N/A	$0.05	$0.60
Resistors (10kΩ)	3	GVSU	N/A	$0.01	$0.03
Capacitors (1μF)	2	GVSU	N/A	$0.01	$0.02
Total Cost					$120.17

The total cost of the Segway was $120.17, below the budget by $79.83. The unit cost of the consumables used for joining the assembly was given by Bob Bero.

2.2 Mass Inventory

The mass of the cart was constrained to be less than 1 Kg. Should the cart weight increase beyond the constraint, the final score would increase exponentially by the mass value. The mass table was kept current on a weekly basis. Some of the components had masses that were negligible, since the values were almost insignificant compared to the mass of the entire cart. The final mass of the cart was measured to be 0.975 kg. The actual cart mass differed from the mass that is shown in

Table 3 since the calculated mass did not account for the exact mass of consumables such as screws, nuts, and washers, but was added to the mass of the actual cart. Individual consumables could not be weighed accurately since they were so small.

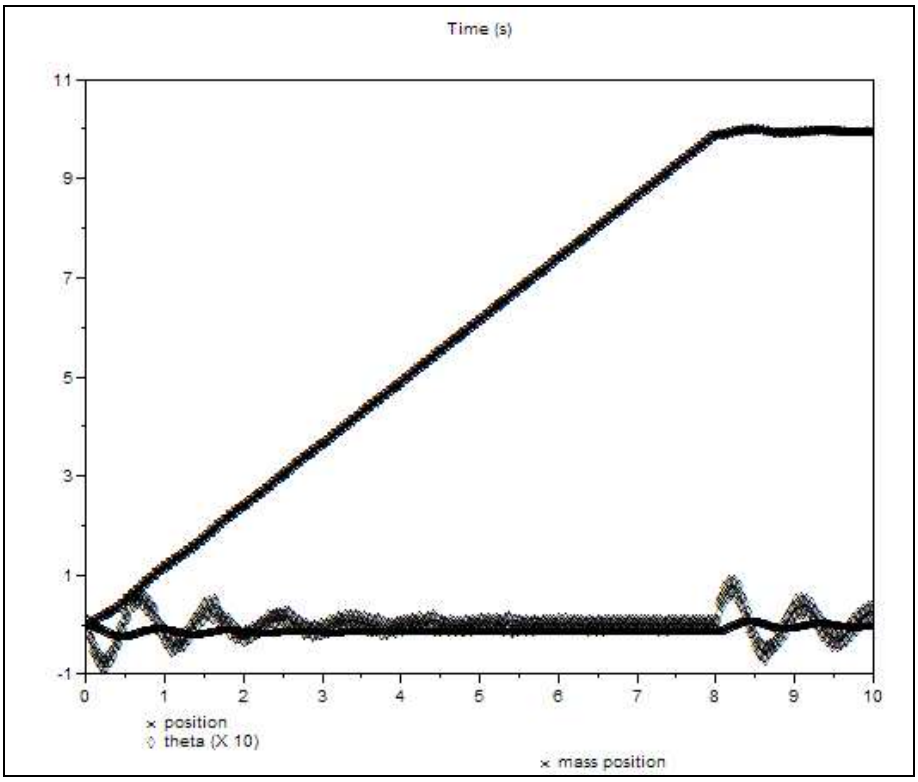
Table 3: Mass of Cart Components

Part Description	**Qty.**	**Mass per part (g)**	**Mass (g)**
Cart Parts			
Wheel	2	67.35	134.7
Frame (Bottom)	1	33.77	33.77
Frame (Side)	2	44.56	89.12
Frame (Top)	1	117.17	117.17
Sensor Support	1	4.47	4.47
9 Volt Battery	1	94.35	94.35
AA Battery Case	1	1.00	1.00
AAA Batteries	9	11.75	105.75
AAA Battery Case	4	1.00	4
Consumables	52	0.50	26
Controller Parts			
Distance Measuring Sensor	2	4.10	8.20
3-pin JST Sensor Cable	1	1.00	1.00
Atmega 32 Board	1	85.00	85.00
Photoconductive Cell	2	0.40	0.80
Gearhead Motors	2	163.00	326
Calculated Weight			1031.33
Measured Weight			**975**

3. Test Results

3.1 Scilab Simulation

Using the Scilab software, a program was written to produce the graphical simulation of the Segway's motion in Figure 9. The program can be found in Appendix D.



t (sec)	x (cm)
0.01	0
0.51	0.47862
1.01	1.20246
1.51	1.75032
2.01	2.44484
2.51	3.0616
3.01	3.64544
3.51	4.30117
4.01	4.91036
4.51	5.56845
5.01	6.18628
5.51	6.77454
6.01	7.39714
6.51	8.02246
7.01	8.64757
7.51	9.27226
8.01	9.8974
8.51	10.0454
9.01	9.98048
9.51	9.96899

Figure 9: Scilab simulation of the system

The Scilab simulation produced for this project modeled the motion of a cart whose body was fixed about its axle shaft. However, the body was designed to freely rotate about the axle shaft. The added motion compounded the tipping of the cart about a point between the ground and the wheels. To completely model the system, the program would have had to incorporate the rotation about the central axis of the wheels. It was also noted that the actual motion of the cart was not smooth.

To summarize, the Scilab simulation to accurately model the motion of the cart would not have been linear. The graph representing the displacement would have had continuous oscillations in it as it progressed upward; a feature representative of the surging nature of the carts forward movement. The graph of the angle about the axle would have also oscillated in a controlled manner since the continuous swaying would

have also been seen by the body of the cart, independent of the motion of the cart in its entirety.

3.2 Score Estimate

The Segways were scored at the final competition on November 23, 2004, with the following equation.

$$\text{score} = \left(\frac{t_s}{d}\right)^2 (4)^{\frac{C}{200}} (10)^B (10)^T (2)^M (S)^2$$

where,

t_s = the time to settle (s)

c = total cost of the part ($)

d = distance moved in test (m)

B = build quality score assigned by judges (0=best, 1=worst)

T = theory quality score assigned by judges (0=best, 1=poor)

M = mass of apparatus (Kg)

S = spillage (mL)

The final competition consisted of various activities for testing each Segway's agility. The first test determined the settling time and whether each Segway could self-balance without a mass. The cart was placed on the ground as a team member tapped on its top plate in a manner that could throw it off balance. Typically, the Segway would rock back and forth until motion stopped. The time from when the cart started rocking to when it settled was defined as the settling time. The Segway designed by Team 8 successfully balanced itself and had a quick settling time. The cart adjusted to the tap within a fraction of a second. Each cart was also tested for balancing a 200 g mass without having to follow a line. Team 8's Segway held the mass for approximately 5 seconds before the cart was tipped.

The next test was for each Segway to follow a straight standard black electrical tape path without a mass. The path was approximately 3 meters long. After successfully

following the line, the 200 g mass was placed on the top plate and the straight line following test was repeated. The cart traveled the entire length of the line. This distance was defined as the distance moved in the test, *d*.

Build quality was estimated to be a 0.3. Several theories were applied to the cart and system control design. In the initial design, the center of mass was high. Through constant testing and redesign, the center of mass of the Segway was lowered by lowering the top plate. Thus, a low center of mass was more successful at balancing a mass than a high center of mass. Nine AAA batteries were calculated to supply enough power to drive the motors. This was very evident during the days prior to the final competition. The groups using a 9 V battery constantly had to buy more of those batteries to keep their carts moving. Team 8's Segway never had to replace the batteries used to drive the motors. Since the theories proved true, the theory quality score was estimated to be 0.2. If water was used during the competition, then the amount of spillage could have been approximately 125 milliliters due to the constant rocking back and forth of the cart to balance. The following values were substituted into the score equation.

Table 4: Final Competition Results

Parameter	Value
Time, *t* (s)	0.5
Cost of cart, *c* ($)	120.17
Distance traveled in test, *d* (m)	3
Build quality, *b* (score)	0.3
Theory quality, *T* (score)	0.2
Mass of cart, *M* (Kg)	0.975
Spillage, *S* (mL)	125

Thus, the score for the Team 8 Segway was,

$$score = \left(\frac{0.5}{3}\right)^2 (4)^{\frac{120.17}{200}} (10)^{0.3} (10)^{0.2} (2)^{0.975} (125)^2$$

$$score = \underline{\mathbf{6205.40}}$$

4. Conclusion

The cart followed the entire length of the tape path, 3 meters, during the final competition while carrying the mass. After the competition, the Segway was also tested to see if it could follow non-linear tape paths. Without a mass, the cart performed with excellence. It followed any curved path where the lighting was not too dark or too bright.

It was theorized that a lower center of mass would help the Segway balance. This was evident in competition when the carts with lower center of masses were more successful in balancing the mass. The designed Segway did not use encoders, which did not hinder its performance. The provided photoresistors were sufficient feedback systems for allowing the carts to follow the lines.

The design of the Segway could have improved by adjusting for different light conditions. When the lighting was too bright or dark, it got confused and hesitated or did not react to any line. The center of mass for this design could have also been lowered by adding additional weight or lowering the top plate to hang below the wheel axles.

5. References

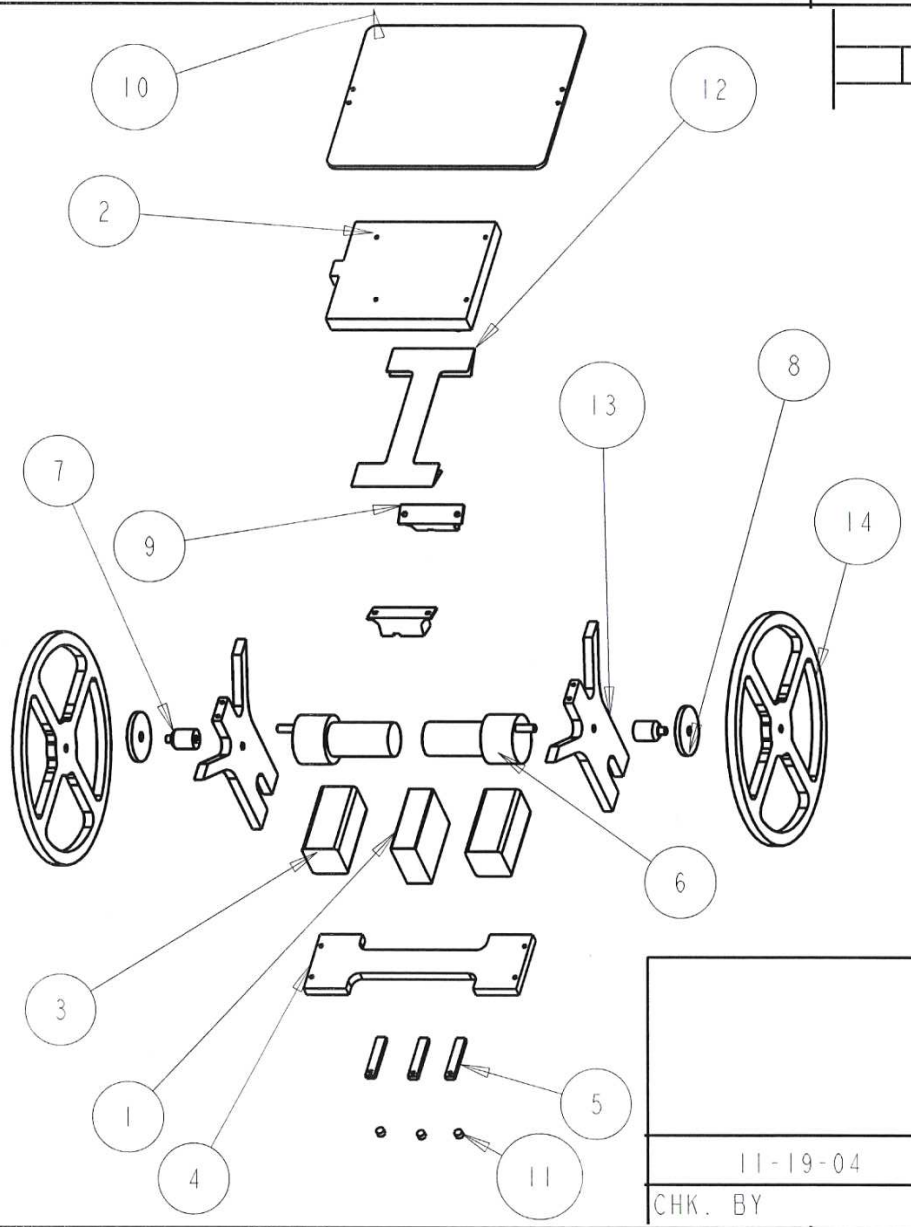
1. Aluminum. McMaster-Carr. http://www.mcmaster.com.
2. "DC PM Gearmotor." Buehler Motor Group. http://www.buehlermotor.com/cgi-bin/sr.exe/productpageus&productpage=68.
3. Jack, H. EGR 345 Project – Load Sway Compensation for Cranes (Fall 2004). http://claymore.engineer.gvsu.edu/~jackh/eod/courses/egr345/fall04/contest2004.pdf.
4. "Overview – Polycarbonate, PTFE Filled." Matweb. http://www.matweb.com/search/SpecificMaterial.asp?bassnum=O3107.
5. Plastics. McMaster-Carr. http://www.mcmaster.com.
6. Sharp GP2D120 Distance Measuring Sensor. Mark III Robot Store. http://www.junun.org/MarkIII/Info.jsp?item=37.
7. Silicon Detectors and Emitters. Digikey. http://dkc3.digikey.com/pdf/T041/1215.pdf.

6. Appendices

6.1 Appendix A: Prints

Each manufactured component developed for the Segway chassis was drawn and dimensioned using Pro E Wildfire Software. Purchased components were modeled in the assembly with the designed parts. The individual three view prints of each part are also attached after the exploded assembly.

6.1.1 Exploded Assembly with BOM
6.1.2 Top Plate
6.1.3 Side Plate
6.1.4 Bottom Plate
6.1.5 Sensor Bracket
6.1.6 Photoresistor Mount
6.1.7 Wheel
6.1.8 Axle Collar



PRT. #	QTY	INDEX	PART NAME
SGW0001	1	1	9V_BATT
SGW0002	1	2	ATMEGA32
SGW0003	2	3	BATTERY_PACK
SGW0004	1	4	BOTTOM
SGW0005	3	5	BOTTOM_SENSOR_ARM
SGW0006	2	6	BUEHLER_MOTOR
SGW0007	2	7	COLLAR
SGW0008	2	8	GEAR
SGW0009	2	9	GP2D120
SGW0010	1	10	LEVEL_1
SGW0011	3	11	PHOTORESISTOR
SGW0012	1	12	PLATE
SGW0013	2	13	SIDE_TRUSS
SGW0014	2	14	WHEEL2

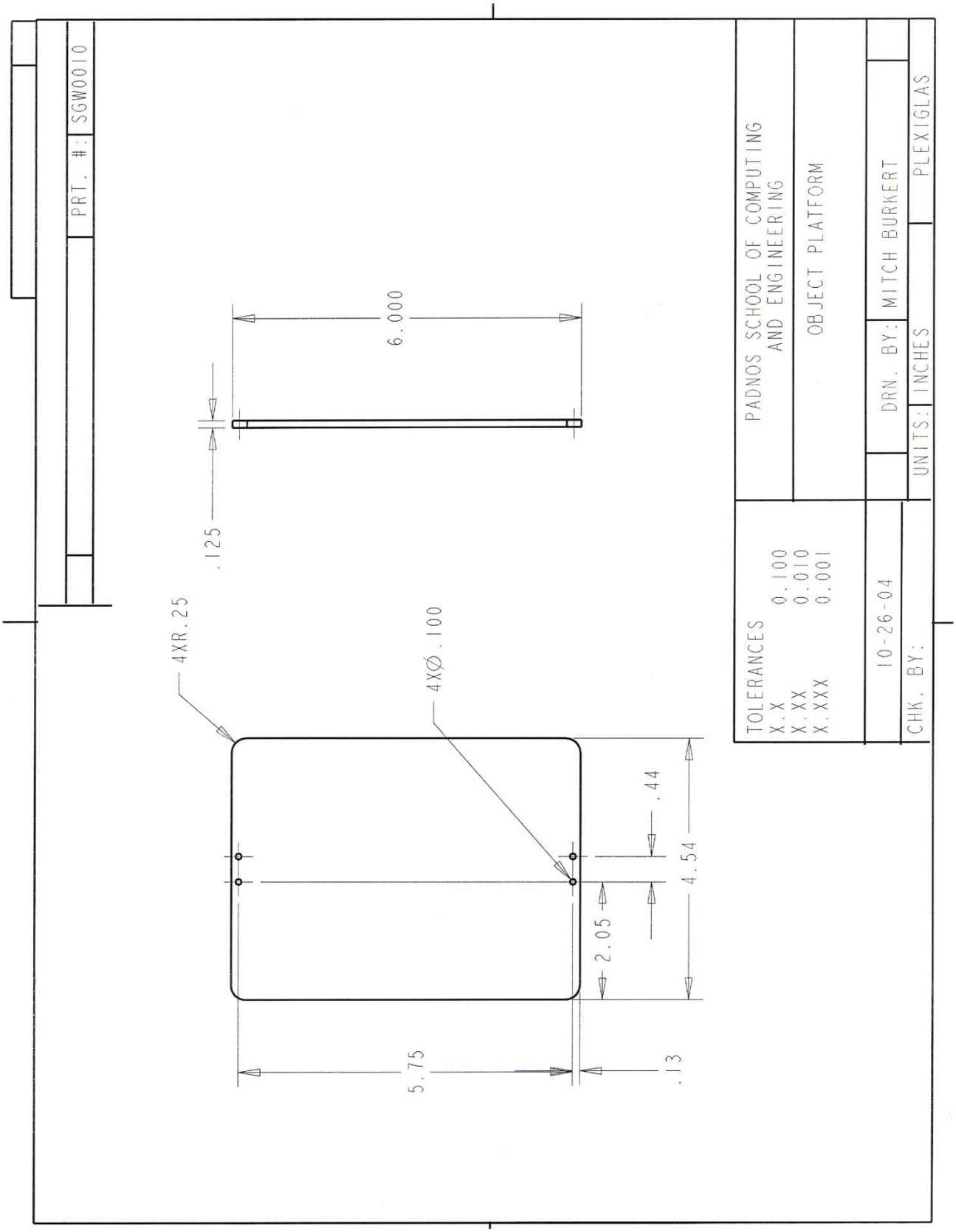
PADNOS SCHOOL OF
COMPUTING AND ENGINEERING

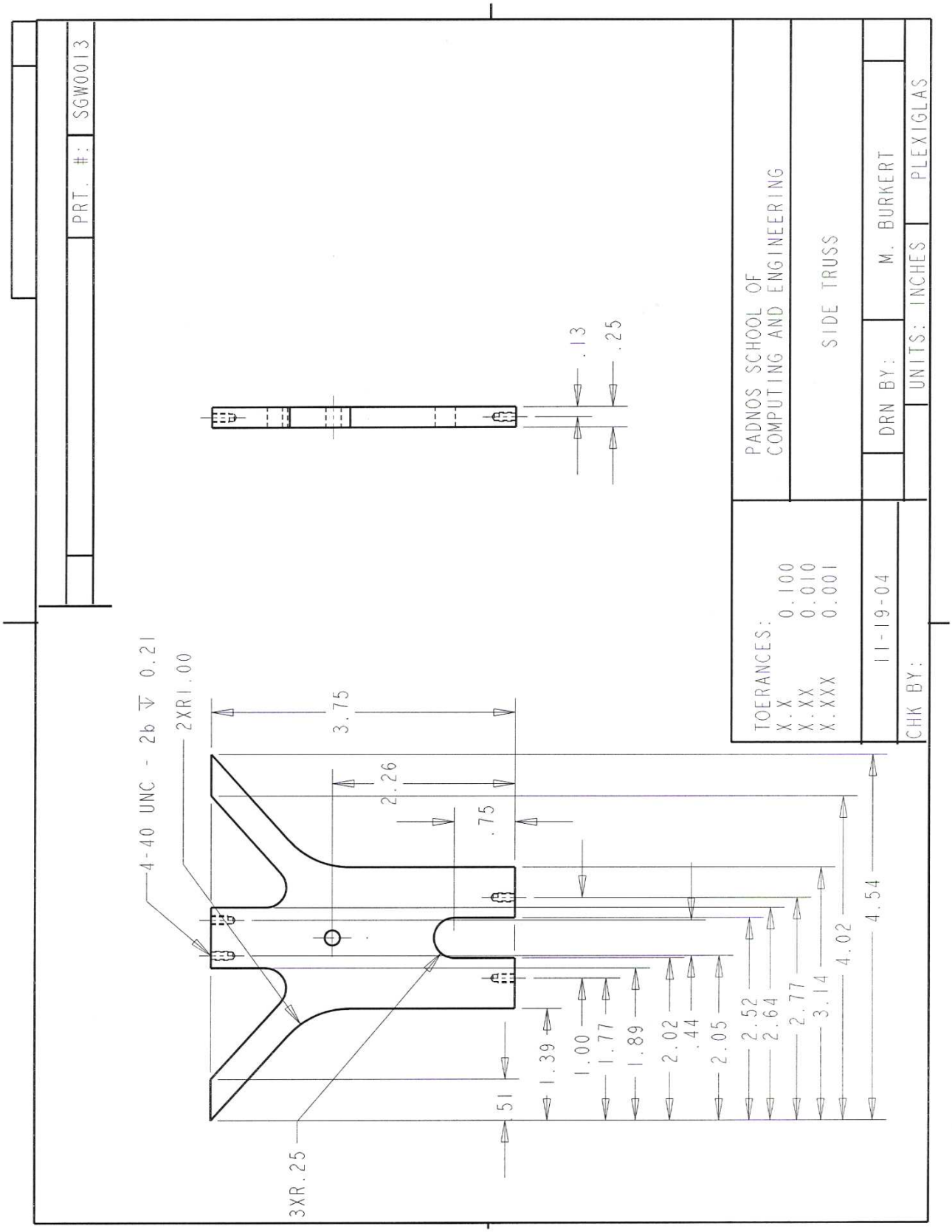
SEGWAY B.O.M.

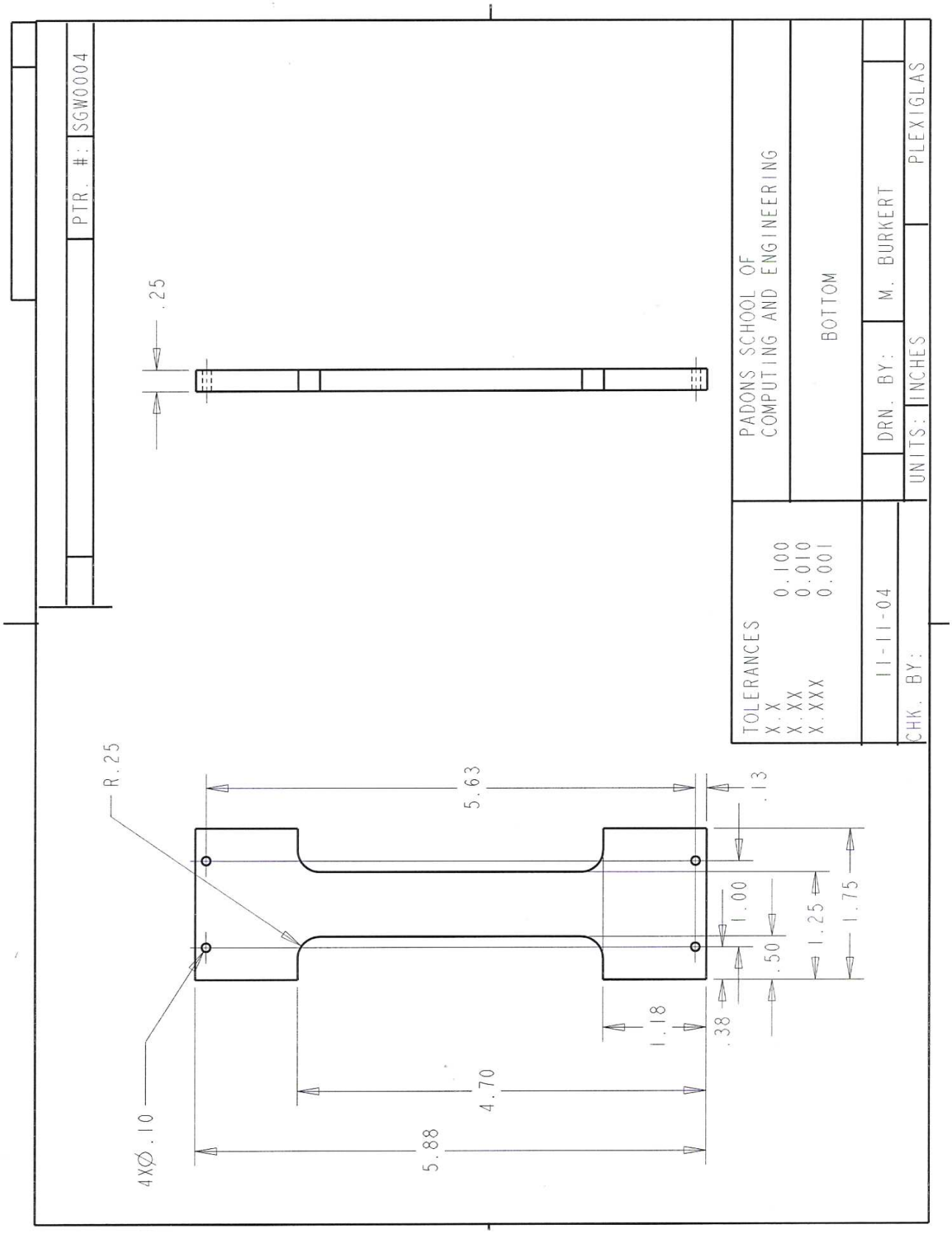
11-19-04

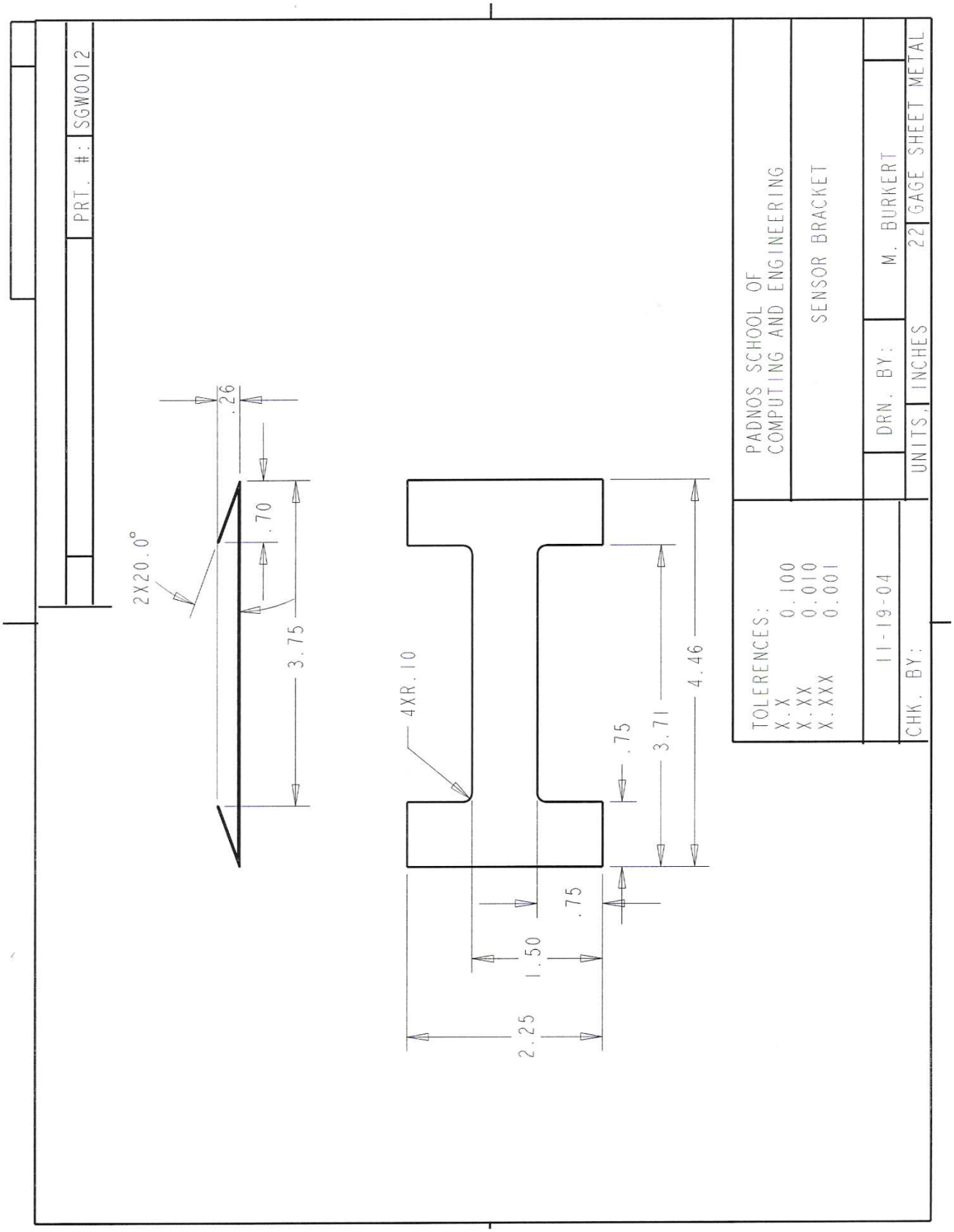
DRN. BY: M. BURKERT

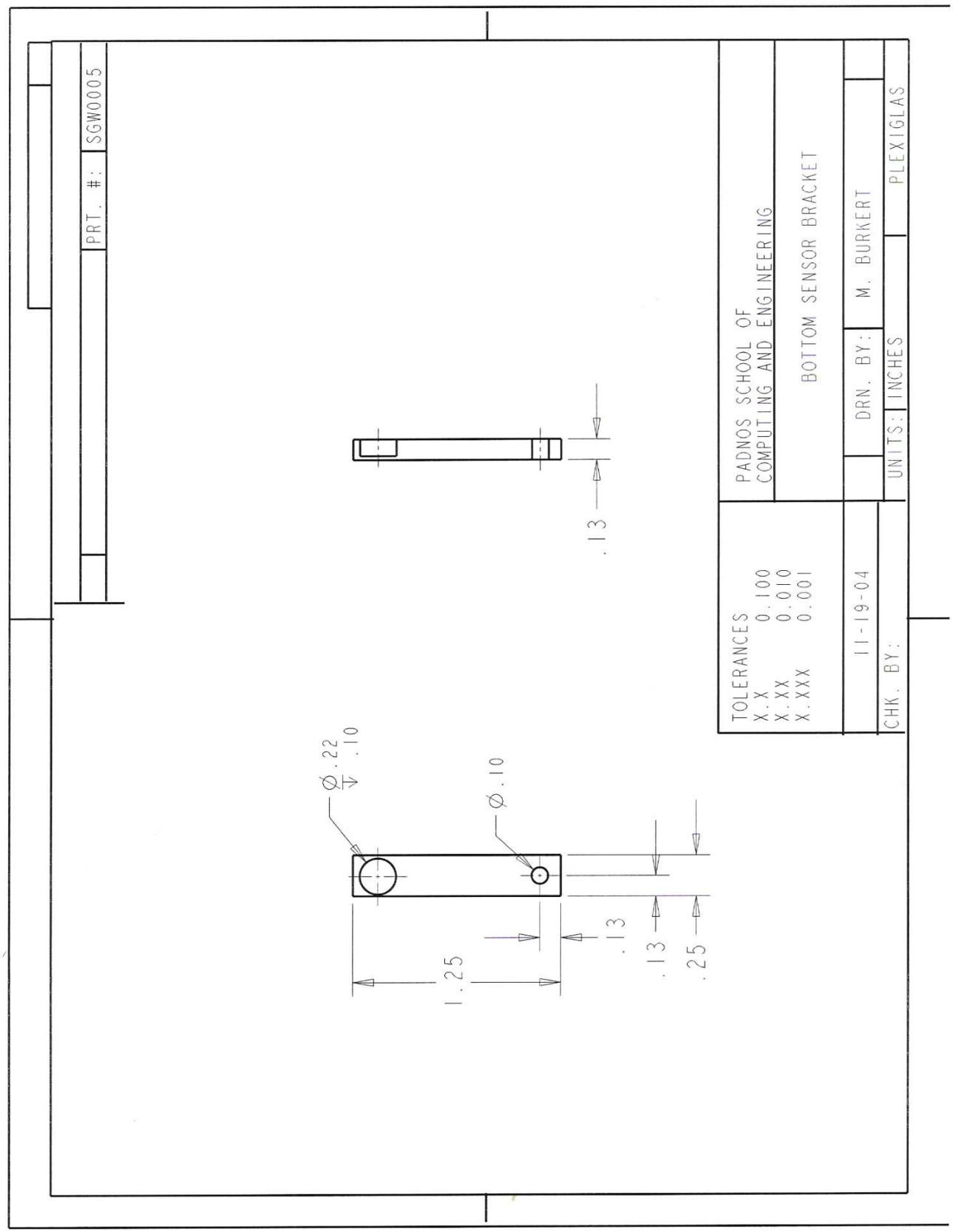
CHK. BY

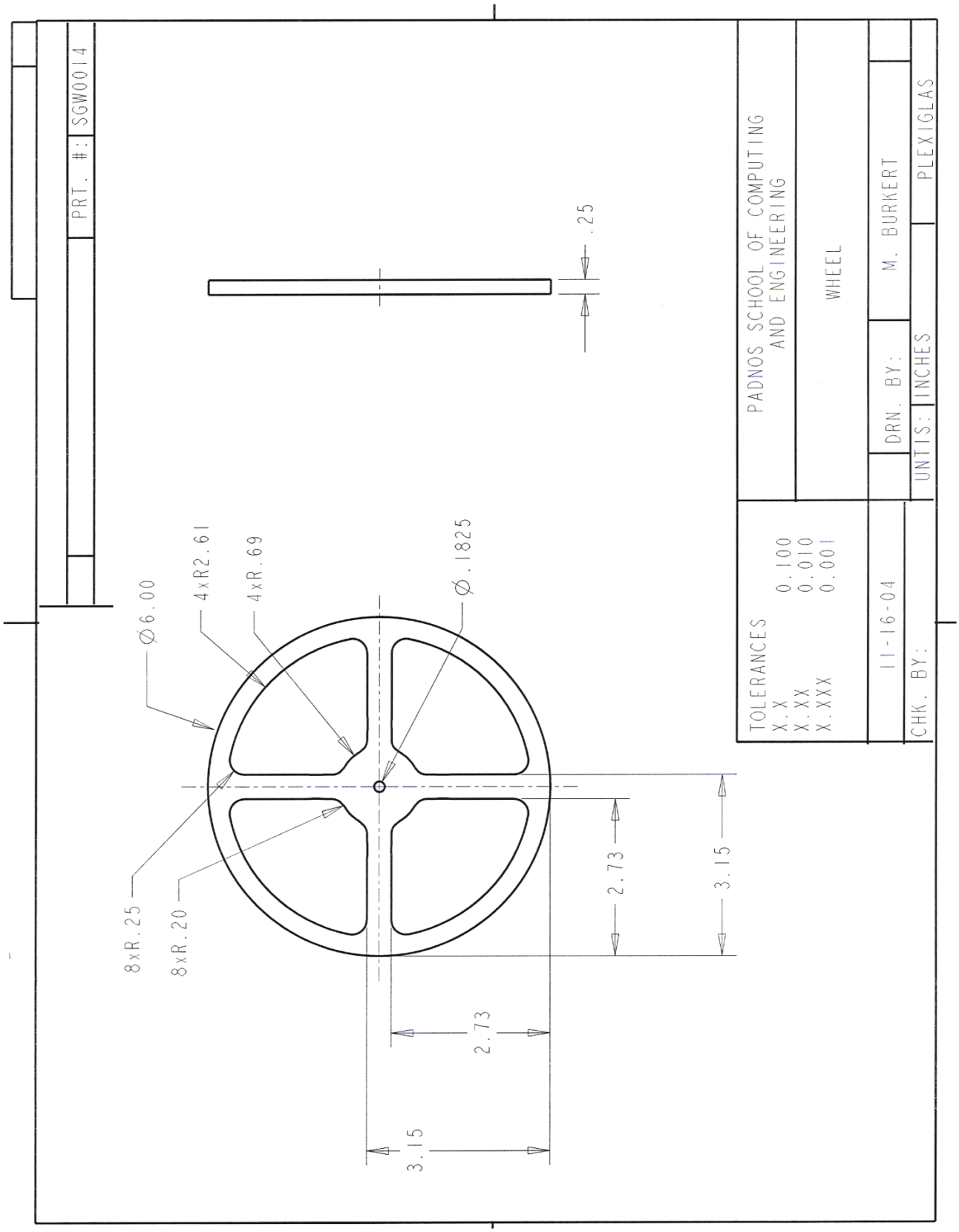

PRT. #: SGW0010
6.000
.125
4XR.25
4XØ.100
.44
4.54
2.05
5.75
.13
PADNOS SCHOOL OF COMPUTING
AND ENGINEERING
OBJECT PLATFORM
TOLERANCES
X.X 0.100
X.XX 0.010
X.XXX 0.001
10-26-04
DRN. BY: MITCH BURKERT
UNITS: INCHES
PLEXIGLAS
CHK. BY:

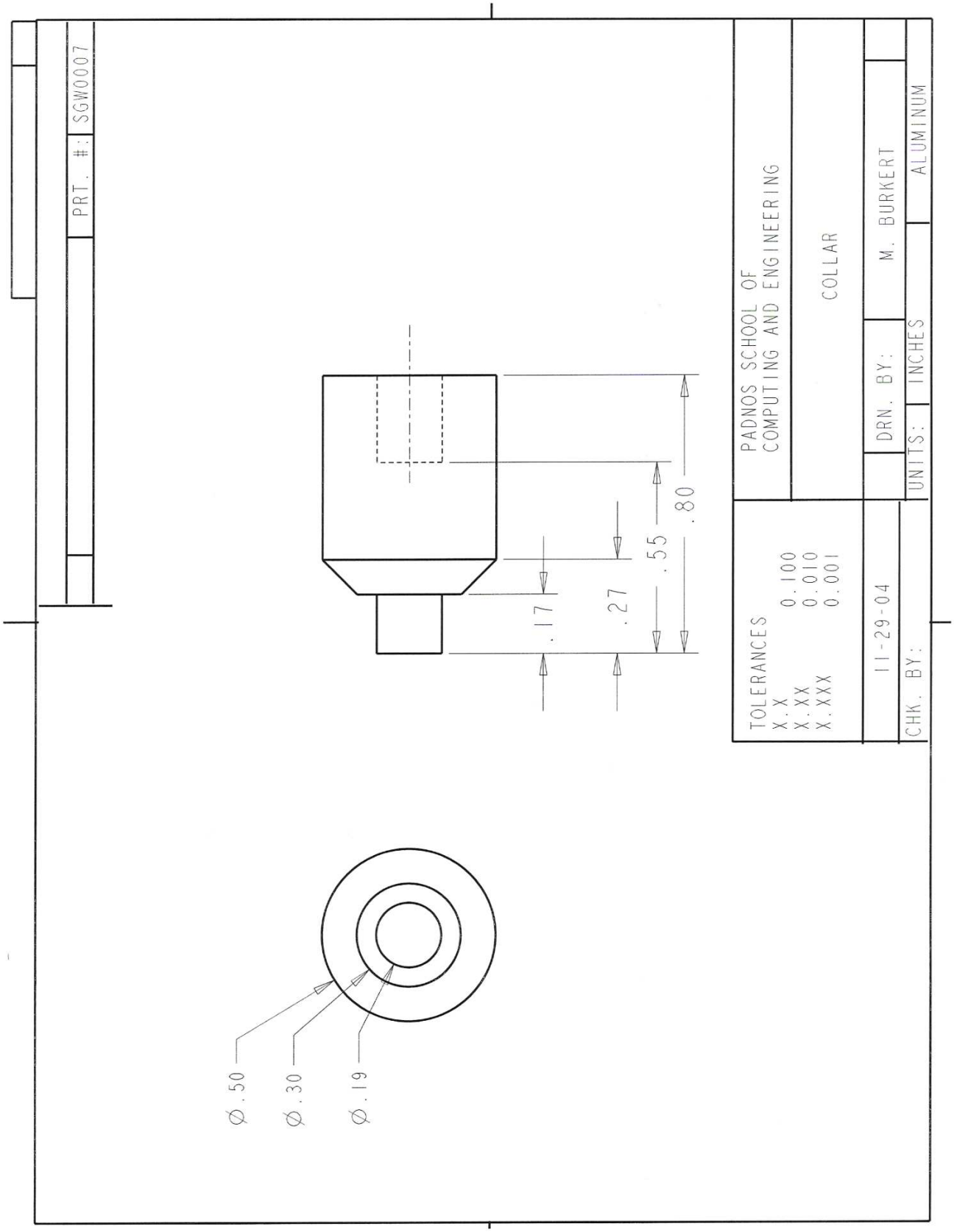

PRT. #: SGW0013
PADNOS SCHOOL OF
COMPUTING AND ENGINEERING
SIDE TRUSS
DRN BY: M. BURKERT
UNITS: INCHES
PLEXIGLAS
TOERANCES:
X.X 0.100
X.XX 0.010
X.XXX 0.001
11-19-04
CHK BY:
.13
.25
4-40 UNC - 2b ↧ 0.21
2XR1.00
3.75
2.26
.75
.51
1.39
1.00
1.77
1.89
2.02
.44
2.05
2.52
2.64
2.77
3.14
4.02
4.54
3XR.25


PTR. #: SGW0004
.25
PADONS SCHOOL OF
COMPUTING AND ENGINEERING
BOTTOM
DRN. BY: M. BURKERT
UNITS: INCHES
PLEXIGLAS
TOLERANCES
X.X 0.100
X.XX 0.010
X.XXX 0.001
11-11-04
CHK. BY:
R.25
5.63
.13
1.00
1.25
1.75
.50
.38
1.18
4.70
5.88
4XØ.10


PRT. #: SGW0012
.26
.70
2X20.0°
3.75
4XR.10
.75
3.71
4.46
.75
1.50
2.25
PADNOS SCHOOL OF
COMPUTING AND ENGINEERING
SENSOR BRACKET
TOLERENCES:
X.X 0.100
X.XX 0.010
X.XXX 0.001
DRN. BY: M. BURKERT
11-19-04
CHK. BY:
UNITS, INCHES
22 GAGE SHEET METAL

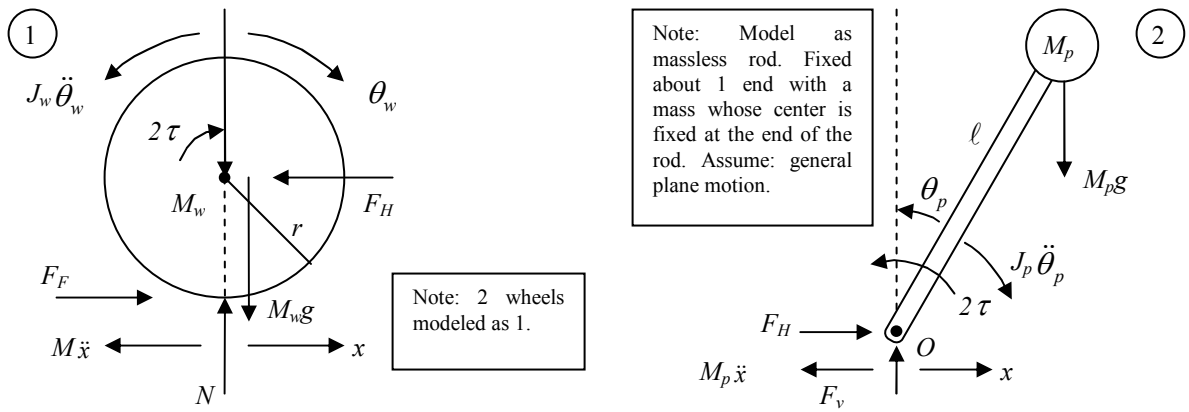

PRT. #: SGW0005
.13
Ø.22
.10
Ø.10
1.25
.13
.13
.25
TOLERANCES
X.X 0.100
X.XX 0.010
X.XXX 0.001
11-19-04
CHK. BY:
PADNOS SCHOOL OF
COMPUTING AND ENGINEERING
BOTTOM SENSOR BRACKET
DRN. BY: M. BURKERT
UNITS: INCHES
PLEXIGLAS


PRT. #: SGW0014
.25
PADNOS SCHOOL OF COMPUTING
AND ENGINEERING
WHEEL
M. BURKERT
PLEXIGLAS
DRN. BY:
UNTIS: INCHES
TOLERANCES
X.X 0.100
X.XX 0.010
X.XXX 0.001
11-16-04
CHK. BY:
Ø6.00
4xR2.61
4xR.69
Ø.1825
8xR.25
8xR.20
2.73
3.15
2.73
3.15


PRT. #: SGW0007
.17
.27
.55
.80
Ø.50
Ø.30
Ø.19
TOLERANCES
X.X 0.100
X.XX 0.010
X.XXX 0.001
11-29-04
CHK. BY:
PADNOS SCHOOL OF
COMPUTING AND ENGINEERING
COLLAR
DRN. BY: M. BURKERT
UNITS: INCHES
ALUMINUM

6.2 Appendix B: Calculations

6.2.1 State Equations



where,

J_w = wheel rotational inertia
M_w = mass of both wheels
θ_w = angle that wheel turns
r = wheel radius
x = horizontal distance cart moves
F_H = horizontal force
F_F = friction force
N = normal force

J_p = pendulum rotational inertia
M_p = mass of rod or pendulum
θ_p = angle that pendulum moves
ℓ = distance from wheel axle, O
τ = torque
F_v = vertical force
g = gravity

Figure 10: Wheel and cart free body diagrams

The state equations were solved by using the free body diagrams shown in Figure 10 and using the following steps. The general plane motion equations, Equations (1) to (3) were used to model the forces and accelerations from the free body diagrams to equation form.

$$\sum (F_{ext})_x = (ma_G)_x \tag{1}$$

$$\sum (F_{ext})_y = (ma_G)_y \tag{2}$$

$$\left\{\sum (M_{ext})_G = I_G\alpha\right\}$$

$$\therefore \sum M_o = I_o\alpha \tag{3}$$

The sum of the forces in each direction was developed from both free body diagrams. The equations from the wheel free body diagram were,

$$\overset{\rightarrow}{+}\sum F_x = -F_H + F_F - M_w\ddot{x} = 0 \tag{4}$$

$$+\uparrow\sum F_y = -F_v + N - M_w g = 0 \tag{5}$$

$$\sum M_o = -2\tau + J_w\ddot{\theta}_w + F_F r = 0 \quad \text{(Right hand rule)} \tag{6}$$

The summed forces from the free body diagram of the cart were,

$$\overset{\rightarrow}{+}\sum F_x = F_H - M_p\ddot{x} = 0 \tag{7}$$

$$+\uparrow\sum F_y = F_v - M_p g = 0 \tag{8}$$

$$\sum M_o = -M_p g\sin\theta_p - J_p\ddot{\theta}_p + 2\tau = 0 \quad \text{(Right hand rule)} \tag{9}$$

The basic torque equation (10) was substituted in the motor model in Equation (11) to obtain equation (12).

$$\tau_{applied} = F_F r \tag{10}$$

$$\dot{\omega} + \omega\left(\frac{K^2}{J_m R_m}\right) = V_s\left(\frac{K}{JR}\right) - \frac{\tau_{applied}}{J} \tag{11}$$

$$F_F = V_s\left(\frac{K}{rR}\right) - \dot{\omega}\left(\frac{J}{r}\right) - \omega\left(\frac{K^2}{rR}\right) \tag{12}$$

It was known that the horizontal distance that the wheel moved was proportional to its angular rotation by the radius.

$$x = \theta_w r_w \tag{13}$$

Thus, substituting Equation (13) into (12) yields Equation (14).

$$F_F = V_s\left(\frac{K}{rR}\right) - \ddot{x}\left(\frac{J_m}{r^2}\right) - \dot{x}\left(\frac{K^2}{r^2 R}\right) \tag{14}$$

The rotational inertia of the system is the rotational inertia of the motor, Equation (15); therefore, the rotational inertia of the cart, J_p, was solved in terms of the cart mass and distance away from the origin of rotation by the parallel axis theorem.

$$J = J_{motor} \tag{15}$$

$$J_p = J_o + MD^2 \qquad \text{// parallel axis theorem}$$

$$\therefore J_p = M_p\ell^2 \tag{16}$$

Equation (9) was substituted into Equation (6) to derive Equation (17).

$$2\tau = J_w\ddot{\theta}_w + F_F r \tag{17}$$

Using known equations, (18), the motor equation (11) was substituted into (17) to obtain the two of the state equations.

$$\ddot{\theta}_w = \frac{\ddot{x}}{r} \qquad J_p = M_p\ell^2 \qquad J_w = \frac{1}{2}M_w r^2 \tag{18}$$

$$-M_p g\sin\theta_p - J_p\ddot{\theta}_p + J_w\ddot{\theta}_w + F_p r = 0$$

$$-M_p g\sin\theta_p - J_p\ddot{\theta}_p + J_w\ddot{\theta}_w + r\left(V_s\left(\frac{K}{rR}\right) - \ddot{x}\left(\frac{J_m}{r^1}\right) - \dot{x}\left(\frac{K^2}{r^2R}\right)\right) = 0$$

$$-M_p g\sin\theta_p - M_p\ell^2\ddot{\theta}_p + \frac{\frac{1}{2}M_w r^2\ddot{x}}{r} + V_s\left(\frac{K}{R}\right) - \ddot{x}\left(\frac{J_m}{r^2}\right) - \dot{x}\left(\frac{K^2}{rR}\right) = 0$$

$$-M_p\ell^2\ddot{\theta}_p = M_p g\sin\theta_p + \frac{1}{2}M_w r\ddot{x} - \frac{KV_s}{R} + \ddot{x}\frac{J_m}{r} + \dot{x}\left(\frac{K^2}{rR}\right)$$

$$\ddot{\theta}_p = \frac{-g\sin\theta_p}{\ell^2} + \left(\frac{M_w r}{2M_p\ell^2} - \frac{J_m}{rM_p\ell^2}\right)\ddot{x} + \frac{KV_x}{RM_p\ell^2} - \left(\frac{K^2}{rRM_p\ell^2}\right)\dot{x}$$

$$\ddot{\theta}_p = \frac{-g\sin\theta_p}{\ell^2} + \left(\frac{M_w r}{2M_p\ell^2} - \frac{J_m}{rM_p\ell^2}\right)\ddot{x} + \left(\frac{-K^2}{rRM_p\ell^2}\right)\dot{x} + \left(\frac{K}{RM_p\ell^2}\right)V_s \tag{19}$$

Equation (4) was substituted into Equation (7) to derive Equation (20).

$$F_H = M_p\ddot{x} \tag{20}$$

The motor equation (11) was substituted into (20) to obtain the last two state equations.

$$-M_p\ddot{x} + F_F - M_w\ddot{x} = 0$$

$$-M_p\ddot{x} + \left[V_s\left(\frac{K}{rR}\right) - \ddot{x}\left(\frac{J_m}{r^2}\right) - \dot{x}\left(\frac{K^2}{r^2R}\right)\right] - M_w\ddot{x} = 0$$

$$M_p\ddot{x} + \ddot{x}\left(\frac{J_m}{r^2}\right) + M_w\ddot{x} = V_s\left(\frac{K}{rR}\right) - \dot{x}\left(\frac{K^2}{r^2R}\right)$$

$$\ddot{x}\left(M_p + \frac{J_m}{r^2} + M_w\right) = V_s\left(\frac{K}{rR}\right) - \dot{x}\left(\frac{K^2}{r^2R}\right)$$

$$\ddot{x}\left(\frac{r^2M_p + J_m + r^2M_w}{r^2}\right) = V_s\left(\frac{K}{rR}\right) - \dot{x}\left(\frac{K^2}{r^2R}\right)$$

$$\ddot{x} = V_s\left(\frac{K}{rR}\right)\left(\frac{r^2}{r^2M_p + J_m + r^2M_w}\right) - \dot{x}\left(\frac{K^2}{r^2R}\right)\left(\frac{r^2}{r^2M_p + J_m + r^2M_w}\right)$$

$$\ddot{x} = -\dot{x}\left(\frac{K^2}{R\left(r^2M_p + J_m + r^2M_w\right)}\right) + V_s\left(\frac{Kr}{R\left(r^2M_p + J_m + r^2M_w\right)}\right) \tag{21}$$

The state equations (22) and (23) were derived from Equation (21) and state equations (24) and (25) were derived from Equation (19).

$$\dot{x} = v \tag{22}$$

$$\dot{v} = \left(\frac{= K^2}{R\left(r^2M_p + J_m + r^2M_w\right)}\right)v + \left(\frac{Kr}{R\left(r^2M_p + J_m + r^2M_w\right)}\right)V_s \tag{23}$$

$$\dot{\theta}_p = \omega_p \tag{24}$$

$$\dot{\omega}_p = \frac{-g\sin\theta_p}{\ell^2} + \left(\frac{M_w r}{2M_p\ell^2} - \frac{J_m}{rM_p\ell^2}\right)\dot{v} + \left(\frac{-K^2}{rRM_p\ell^2}\right)v + \left(\frac{K}{RM_p\ell^2}\right)V_s \tag{25}$$

6.2.2 Motor Torque Calculation

The motor torque needed to maintain a mass position was calculated with the sum of the moments about the motor shaft axle shown below in Equation (26), derived from Figure 3.

$$\sum M_{axle} = \tau_{motor} - Mg\ell = 0 \tag{26}$$

Where,

ℓ = Length from axle to center of mass
Mg = Force due to gravity
τ = Motor torque needed to maintain mass position

$$\therefore \tau_{motor} = Mg\ell$$

$$\tau_{motor} = \left(\frac{1}{3}kg\right)\left(9.81\frac{m}{s^2}\right)\left(0.127m\right) = 1.61963Nm$$

Necessary motor torque can be divided between two motors.

$$\frac{\tau_{motor}}{2} = 0.80916 Nm$$

Calculate Torque from Buehler motors at 12 Volts.

$$\tau_{motor} = KI$$
$$\tau_{motor} = K\left(\frac{V_s}{R_\Omega}\right)$$
$$\tau_{motor} = 3\left(\frac{12}{30}\right) = 1.2 Nm$$

The Beuhler motors can supply 1.2 Nm of torque when only 0.81 Nm is needed.

6.2.3 Wheel Stress Calculation

The maximum shear stress in the wheel spokes was calculated using Figure 4. Stress was calculated with Equation (27).

$$\sigma_{shear} = K\left(\frac{P}{A}\right) \tag{27}$$

Where,

P = Force

A = Cross sectional area around stress riser

K = Stress concentration factor (2.5)

The mass of the cart and load was divided in half, as the stress is shared by two wheels; therefore,

$$\sigma_{shear} = 2.5\left(\frac{1.5kg \times 9.81\frac{m}{s^2}}{0.00015m^2}\right) = 0.270 \text{ MPa}$$

Since the maximum strength of polycarbonate material was 72 MPa, the weight of the cart and load was able to be supported [4]. The calculated factor of safety , n, was computed with Equation (28).

$$n = \frac{\sigma_{\max}}{\sigma_{shear}} \tag{28}$$

$$n = \frac{72MPa}{0.270MPa} = 266$$

6.2.4 Deformation and Strain Calculations

The equations used to calculate the deformation and elongation in the wheel spoke are given as Equation (29) and (30), respectively.

$$\delta = \frac{PL}{AE} \tag{29}$$

$$\varepsilon = \frac{\delta}{L} \tag{30}$$

Where,

δ = Deformation

ε = Strain

Force, P = 1.5 Kg

Length, L = 0.045 m

Cross sectional area of wheel spoke, A = 5 x 10^{-5} m^2

Modulus of Elasticity, E = 2 x 10^9 GPa

Therefore from Equation (29) and (30),

$$\delta = \frac{1.5kg * 9.81\frac{m}{s^2} 0.045m}{(2E9)GPa * 0.00005m^2} = 7\mu m$$

$$\varepsilon = \frac{7\mu m}{0.045m} = 1.47 \times 10^{-4}$$

Deformation was negligible since it was too small to impinge on the motion of the cart.

6.3 Appendix C: Controller Program in C

```
/*****************************************************************************
 * Author(s): Tyler McCoy, Taylor Groll, Thao Lai, Dennis Smith              *
 * File Name: Segway.c                                                       *
 * Description: This programs uses the Atmega32 processor to run a           *
 * continous feedback control system for a two wheeled, self balancing,      *
 * line following robot. The system utilizes distance measuring IR sensors   *
 * for balance control, and photoresistors for line following feedback.      *
 *                                                                           *
 *****************************************************************************/
#include <avr/io.h>          /* ATMEGA32 I/O register defines */
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <serial.h>


#define T 4 /* define as 4 ms, must divide by 1000 later*/
#define Kp 3          //Proportional Gain
#define Kt 1          // turning gain

#define c_kinetic_pos2 125 /* static friction motor 2*/
#define c_kinetic_neg2 124 /* make the value positive motor 2*/
#define c_static_pos2 125  /* kinetic friction motor 2*/
#define c_static_neg2 124  /* make the value positive motor 2*/
#define c_kinetic_pos1 100 /* static friction motor 1*/
#define c_kinetic_neg1 92  /* make the value positive motor 1*/
#define c_static_pos1 100  /* kinetic friction motor 1*/
#define c_static_neg1 92   /* make the value positive motor 1*/

#define ADC_VREF_TYPE 0xC0

#define c_max 255
#define c_min 255 /* make the value positive */
/* --------------Global Variables---------------------------*/
int moving = 0;
int e_sum = 0;
int count1=0; //variables used in manual control
int count2=0;

int bal_error; // the balancing error
int ebal=0; //the balancing error gain


int current =0;
int Kd=0; //derivative gain

int photo1=0;  //Initialization of line following variables
int photo2=0;
int photo3=0;
int lines1=0;
int lines2=0;

int IR1=0;  //Initialization of balance variables
int IR2=0;
```

Figure 11: C system controller program

```
int setpoint=20; //voltage sent to motors if Segway was perfectly balanced and walking the line


/*---------------Function Prototypes---------------------------*/
void delay(int ticks);
int v_output(int v_adjusted); /* call from the interrupt loop */
int integrate(int e);
void CLK_setup();
void PWM_init(); /* call this routine once when the program starts */
void AD_setup();
void PWM_update1();
int v_output1();
int deadband1();
int controller1();

void PWM_update2();
int v_output2();
int deadband2();
int controller2();
void lights();
int derivative();


/*-------------The meat and bones of the whole deal------------------*/

void IO_update()// This routine will run once per interrupt for updates
{
    while (photo3 > 500 )
    {
            setpoint=0;
    }
    PWM_update1(v_output1(deadband1(controller1())));
    PWM_update2(v_output2(deadband2(controller2())));

}


//----------------------Setup Stuff-------------------------------
void Init_IO()
{
    AD_setup();
    PWM_init();

    DDRD = 0xFF; //Sets all of Port D to outputs
    PORTD = 0xFF; //Turns all of pins on Port D high (+5 V)

    CLK_setup();
}

void AD_setup()
{
    DDRA = 0x00; //set Port A to inputs
}
```

Figure 11 (continued): C system controller program

```
void PWM_init() /* call this routine once when the program starts */
{
    DDRD |= (1 << PD5); /* set PWM outputs */
    DDRC |= (1 << PC0) | (1 << PC1);
    DDRC |= (1 << PC2); /* set PWM outputs */
    DDRC |= (1 << PC3); /* set PWM outputs */
    /* set motor direction outputs on port C*/
    //using OCR1
    TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM10);
    // turn on both PWM outputs on counter 1
    TCCR1B = _BV(CS11) ; // set the internal clock to /8
}

unsigned int read_adc(unsigned char adc_input)
{
    ADMUX = adc_input | ADC_VREF_TYPE;
    ADCSRA = 0xC0;
while ((ADCSRA & _BV(ADSC))!=0);
    ADCSRA = 0xC0;
    while ((ADCSRA & _BV(ADSC))!=0);
    return ADCW;
}
void delay(int ticks)
{ // ticks are approximately 1ms
    volatile int i, m;
    for(i = 0; i < ticks; i++)
    {
            for(m = 0; m < 1000; m++){}
    }
}
int derivative()
{
    int prev;
    int der_gain;
    prev=current;
    current= IR2-IR1;
    der_gain= current - prev;
    if (der_gain >30)
    {
            der_gain=30;  //limits the value of the derivative gain to help with

smoothness
    }
    return der_gain;
}
//-----------------PWM Stuff - Motor 1---------------------------

int controller1()
{
    int Kt1;
  // -----------Update All AD Values------------
    photo2= read_adc(3);
    photo1= read_adc(6);
    photo3= read_adc(7);
    IR1= read_adc(5);
    IR2= read_adc(4);
```

Figure 11 (continued): C system controller program

```
// -----------Calculate Errors for Motor 1---
  int line_error1= photo1-photo2; //Calculate Turning Error

  bal_error= (IR2 - IR1);                //Calculate Difference in IR sensors

  int deriv= derivative();

 // -----------"Software Transmission" for Balance----------

  if (bal_error > 0)
  {
            ebal=1;
  }
  if (bal_error ==0)
  {
            ebal=0;
  }
  if (bal_error > 60)
  {
            ebal=2;
  }
  if (bal_error > 120)
  {
            ebal=4;
  }

  if (bal_error < 0)
  {
            ebal=1;
  {
            ebal=2;
  }
  if (bal_error < -120)
  {
            ebal=4;
  }

 // -----------"Software Transmission" for Line Following----------

  if (line_error1 >= 60)
  }
  if (bal_error < -60)
  {
            Kt1=3;

  }

  else if (line_error1 >= 40)
  {
            Kt1 = 2;

  }
  else if(line_error1 > 20)
  {
```

Figure 11 (continued): C system controller program

```
                Kt1 =1;

        }
        else if(line_error1 > 10)
        {
                Kt1 = 1;

        }
        else
        {
                Kt1 =0;

        }


        lines1= line_error1*Kt1;
        if (lines1 >= setpoint) /*Keeps value of Line Following less than the setpoint

    for guaranteed forward motion*/
        {
                lines1 = setpoint;
        }

        if (bal_error*ebal < 0) //Corrects Turing in the robot is moving backwards
        {
                lines1= -lines1;
        }
        if (photo3 > 300)  //To sit and balance when on white paper
        {
                return (bal_error*ebal + deriv*Kd);
        }
        else  //To balance and line follow
        {
                return (bal_error*ebal + deriv*Kd + setpoint - lines1);  //for balance
        }
    }

    int v_output1(int v_adjusted) /* call from the interrupt loop */
    {
        int RefSignal; // the value to be returned
        if(v_adjusted >= 0)
        {
                /* set the direction bits to CW on, CCW off */
                PORTC = (PINC & 0xFC) | 0x02; /* bit 1 on, 0 off */
                if(v_adjusted > 255)/* clip output over maximum */
                {
                        RefSignal = 255;
                }
                else
                {
                        RefSignal = v_adjusted;
                }
        }
        else    /* need to reverse output sign */
        {
```

Figure 11 (continued): C system controller program

```
                        /* set the direction bits to CW off, CCW on */

    PORTC = (PINC & 0xFC) | 0x01;           /* bit 0 on, 1 off */
            if(v_adjusted < -255) /* clip output below minimum */
            {
                    RefSignal = 255;
            }
            else
            {
                    RefSignal = -v_adjusted; /* flip sign */
            }
    }
    return RefSignal;
}

void PWM_update1(int value)
{ /* to update the PWM output */

    if(value > 255) value = 255;
    if(value < 0) value = 0;
    OCR1A = value; // duty cycle
}

int deadband1(int c_wanted)
{ /* call this routine when updating */
    int c_pos;
    int c_neg;
    int c_adjusted;

    if(moving == 1)
    {
            c_pos = c_kinetic_pos1;
            c_neg = c_kinetic_neg1;
    }

    else
    {
            c_pos = c_static_pos1;
            c_neg = c_static_neg1;
    }

    if(c_wanted == 0)/* turn off the output */
    {
            c_adjusted = 0;
    }

    else if(c_wanted > 0) /* a positive output */
    {
            c_adjusted = c_pos +(unsigned)(c_max - c_pos) * c_wanted / c_max;
            if(c_adjusted > c_max) c_adjusted = c_max;
    }

    else
```

Figure 11 (continued): C system controller program

```
        { /* the output must be negative */
                c_adjusted = -c_neg -(unsigned)(c_min - c_neg) * -c_wanted / c_min;
                if(c_adjusted < -c_min) c_adjusted = -c_min;
        }

        return c_adjusted;
    }

    //---------------PWM Stuff - Motor 2-----------------------------

    int controller2() //needs to be updated to include IR, PH, and investigate benefits of

    integrate function
    {
        int Kt2;
        int deriv= derivative();
        int line_error2 = photo2-photo1;



        // -----------"Software Transmission 2" for Line Following----------

        if (line_error2 >= 60)
        {
                Kt2=3;
        }

        else if (line_error2 >= 40)
        {
                Kt2 = 2;
        }

        else if(line_error2 > 20)
        {
                Kt2 =1;
        }
        else if (line_error2 > 10)
        {
                Kt2 = 1;
        }
        else
        {
                Kt2=1;
        }

        int lines2 = line_error2*Kt2;

        if (lines2 >= setpoint)
        {
                lines2=setpoint;
        }

        if (lines2 < 0) lines2=0; //Prevents robot from turning on axis too hard(only 1

    wheel turning)
```

Figure 11 (continued): C system controller program

```
    if (bal_error*ebal < 0)  //Corrects turning if robot is moving backwards
    {
            lines2= -lines2;
    }

    if (photo3 > 300)  //Sits and balances if robot is sitting on white paper
    {
            return (bal_error*ebal + deriv*Kd);
    }
    else  //For balancing and line following
    {
            return (bal_error*ebal + deriv*Kd + setpoint - lines2 );

    }
}

void PWM_update2(int value)
{ /* to update the PWM output */
    if(value > 255) value = 255;
    if(value < 0) value = 0;
    OCR1B = value; // duty cycle
}

int v_output2(int v_adjusted) /* call from the interrupt loop */
{
    int RefSignal; // the value to be returned
    if(v_adjusted >= 0)
    {
            /* set the direction bits to CW on, CCW off */
            PORTC = (PINC & 0xF3) | 0x08; /* bit 1 on, 0 off */

            if(v_adjusted > 255)/* clip output over maximum */
            {
                    RefSignal = 255;
            }

            else
            {
                    RefSignal = v_adjusted;
            }
    }

    else    /* need to reverse output sign */
    {
            /* set the direction bits to CW off, CCW on */
            PORTC = (PINC & 0xF3) | 0x04; /* bit 0 on, 1 off */
            if(v_adjusted < -255) /* clip output below minimum */
            {
                    RefSignal = 255;
            }
            else
            {
            RefSignal = -v_adjusted; /* flip sign */

    }
```

Figure 11 (continued): C system controller program

```
    }
return RefSignal;
}

int deadband2(int c_wanted)
{ /* call this routine when updating */
    int c_pos;
    int c_neg;
    int c_adjusted;

    if(moving == 1)
    {
            c_pos = c_kinetic_pos2;
            c_neg = c_kinetic_neg2;
    }

    else
    {
            c_pos = c_static_pos2;
            c_neg = c_static_neg2;
    }

    if(c_wanted == 0)/* turn off the output */
    {
            c_adjusted = 0;
    }

    else if(c_wanted > 0) /* a positive output */
    {
            c_adjusted = c_pos +(unsigned)(c_max - c_pos) * c_wanted / c_max;
            if(c_adjusted > c_max) c_adjusted = c_max;
    }

    else
    { /* the output must be negative */
            c_adjusted = -c_neg -(unsigned)(c_min - c_neg) * -c_wanted / c_min;
            if(c_adjusted < -c_min) c_adjusted = -c_min;
    }

    return c_adjusted;
}


//--------------PWM Stuff - Both Motors--------------------------

int integrate(int e)
{
    e_sum += e * T;
    return e_sum;
}

//------------------Clock/Interupt Setup-----------------------------------

#define CLK_ms      10 //set the updates for every 10ms
```

Figure 11 (continued): C system controller program

```
unsigned int CNT_timer; // the delay time
volatile unsigned int CLK_ticks = 0; // the current number of ms
volatile unsigned int CLK_seconds = 0; // the current number of seconds

SIGNAL(SIG_OVERFLOW0)
{ // The interrupt calls this function
    CLK_ticks += CLK_ms;
    if (CLK_ticks >= 100)
    { // The number of interrupts between output changes
            CLK_ticks = CLK_ticks - 100;
            CLK_seconds++;

    }
    IO_update();
    TCNT0 = CNT_timer;
}

void CLK_setup()
{ // Start the interrupt service routine
    TCCR0 = (0<<FOC0) | (0<<WGM01)| (0<<WGM00) | (0<<COM00)
    | (0<<COM01) | (1<<CS02) | (0<<CS01) | (1<<CS00);
    // use CLK/1024 prescale value
    // disable PWM and Compare Output Modes
    CNT_timer = 0xFFFF - CLK_ms * 8; // 8 = 1ms, 80 = 10ms
    TCNT0 = CNT_timer; // start at the right point
    TIFR |= (1<<TOV0);
    TIFR &= ~(0<<OCF0);
    // TIFR = (0<<OCF2) | (0<<TOV2) | (0<<ICF1) | (0<<OCF1A) | (0<<OCF1B)
    // | (0<<TOV1) | (0<<OCF0) | (1<<TOV0); // set to use overflow interrupts
    TIMSK |= (1<<TOIE0);
    TIMSK &= ~(0<<OCIE0);
    // TIMSK = (0<<OCIE2) | (0<<TOIE2) | (0<<TICIE1) | (0<<OCIE1A) | (0<<OCIE1B)
    // | (0<< TOIE1 ) | (0<<OCIE0) | (1<<TOIE0);// enable TCNT1 overflow
    sei(); // enable interrupts flag
}

//----------------Main----------------------------------

int main()
{
    Init_IO();
    SerialSetup();
    puts("Segway Test Program");
    {
            while((c = input()) == -1){}  //wait for keypress

            if (c == 'a')  //Prints values of all analog inputs
            {
    int c;


    for(;;)
            putint(IR1); puts(" = IR1\n");

            putint(IR2); puts(" = IR2\n");
            putint(photo1); puts(" = photo1\n");
```

Figure 11 (continued): C system controller program

```
            putint(photo2); puts(" = photo2\n");
            putint(photo3); puts(" = photo3\n\n");
        }

        else if (c == 'i') //Prints values of IR sensors
        {
            putint(IR1); puts(" = IR1\n ");

            putint(IR2); puts(" = IR2\n\n ");
        }

        else if (c == 'p')  //Prints values Photoresistors
        {
            putint(photo1); puts(" = photo1\n ");
            putint(photo2); puts(" = photo2\n ");
            putint(photo3); puts(" = photo3\n\n ");
        }

        else if (c == 'h')  //Prints help menu (list of commands)
        {
            puts(" a: print all values\n");
            puts(" i: print IR values\n");
            puts(" p: print Photoresistor values\n");
            puts(" l: print line following error\n");
            puts(" b: print balance error\n");
            puts(" 9: Increases derivative gain\n");
            puts(" 0: Decreases derivative gain\n");
            puts(" +: Increases balance gain\n");
            puts(" -: Decreases balance gain\n");
            puts(" q: quit\n");
        }

        else if (c == 'q') //Quits program
        {
            puts("\nYou Quitter...\n");
            break;
        }
        else if (c == 'b') //Prints the balance error
        {
            putint(bal_error);puts(" = balance error\n");
        }
        else if (c == '+') //Increases the balance gain
        {
            ebal++;
            putint(ebal);puts(" = balance gain +\n");
        }
        else if (c == '-') //Decreases the balance gain
        {
            ebal--;
            putint(ebal);puts(" = balance gain -\n");
        }
        else if (c == 'l')  //Prints the line following error
        {
            int diff;
```

Figure 11 (continued): C system controller program

```
                        diff=(photo1 - photo2);
                }
                else if (c == '9')
                {
                        Kd++;
                        putint(Kd);puts(" = derivative gain ++\n");

                }
                else if (c == '0')
                {
                        Kd--;
                        putint(Kd);puts(" = derivative gain --\n");
                }

        }
        return 0;
}
```

Figure 11 (continued): C system controller program

6.4 Appendix D: Scilab Simulation Program

```
// Grand Valley State University - School of Engineering
// EGR 345 - Dynamic Systems - Fall 2004
// AUTHORS:    Taylor Groll, Thao Lai, Tyler McCoy, Dennis Smith
// DATE:       October 27, 2004
// FILENAME:   project_simulation1.sce
// DESCRIPTION - This program simulates the motion of the Segway

     // System component values
     l = 0.2032; // 40cm
     Mp = 1.65; // 3.3kg
     Mc = 0; // 200g
     g = 9.81; // good old gravity
     rw = 0.0762; // 6cm dia tires
     K = 2.5; // motor speed constant
     R = 30; // motor resistance
     J = 0.003; // rotor inertia


     // System state
     x0 = 0;           // initial conditions for position
     v0 = 0;
     theta0 = 0.0;     // the initial position for the load
     omega0 = 0.0;
     X=[x0, v0, theta0, omega0];


     // The controller definition
     PI = 3.14159;
     ppr = 16;  // encoder pulses per revolution;

     Kpot = 1.9791;    // the angle voltage ratio
     Vzero = 1.8229;   // the voltage when the pendulum is vertical
     Vadmax = 5;       // the A/D voltage range
     Vadmin = 0;
     Cadmax = 255;     // the A/D converter output limits
     Cadmin = 0;
     Cacomp = 24.2;
     tolerance = 0.5;  // the tolerance for the system to settle
     Kpp = 20;  // position feedback gain
     Ksp = 2;   // sway feedback gain
     Vpwmmax = 12; // PWM output limitations in V
     Cpwmmax = 255; // PWM input range
     Cdeadpos = 100;   // deadband limits
     Cdeadneg = 95;

     function foo=control(Cdesired)
        Cp = ppr * X($, 1)/(2*PI*rw);
        Cpe = Cdesired - Cp;
        Cpc = Kpp * Cpe;
        VL = Kpot * X($,3) + Vzero; // assume the zero angle is 2.5V
        CL = ((VL - Vadmin) / (Vadmax - Vadmin)) * (Cadmax - Cadmin) + Cacomp;
        if CL > Cadmax then CL = Cadmax; end  // check for voltages over limits
        if CL < Cadmin then CL = Cadmin; end
        CLc = Ksp * (CL - (Cadmax + Cadmin) / 2);
        Cc = Cpc + CLc;
        Cpwm = 0;
        if Cc > 0.5 then       // deadband compensation
                Cpwm = Cdeadpos + (Cc/Cpwmmax)*(Cpwmmax - Cdeadpos);
        end
        if Cc <= -0.5 then
                Cpwm = -Cdeadneg + (Cc/Cpwmmax)*(Cpwmmax - Cdeadneg);
        end

        foo = Vpwmmax * (Cpwm / Cpwmmax) ; // PWM output
        if foo > Vpwmmax then foo = Vpwmmax; end // clip voltage if too large
        if foo < -Vpwmmax then foo = -Vpwmmax; end
    endfunction
```

Figure 12: Scilab system simulation program

```
// The motion profile generator
function foo=motion(t_start, t_end, t_now, C_start, C_end)
   if t_now < t_start then
           foo = C_start;
   elseif t_now > t_end then
           foo = C_end;
   else
           foo = C_start + (C_end - C_start) * (t_now - t_start ) / (t_end - t_start);
   end
endfunction


// define the state matrix function
term1 = Mc*rw^2 + J;      // Precalculate these terms to save time
term2 = R*term1;
Xd = 10;   // the setpoint 10 turns == 160 pulses
Cd = ppr * Xd / (2 * PI * rw) ;
function foo=derivative(state,t, h)
   Vs=control(motion(0, 8, t($, 1), 0, Cd));
// Vs=control(Cd);
   term3 = cos(state($,3));      // precalculate some repeated terms to save time
   term4 = sin(state($,3));
   term5 = term1 + Mp * (term4 * rw)^2;

//printf("%f   %f \n", Cd, Vs);
   v_dot = -state($,2)*(K^2) / (term5 * R) ...          // d/dt v
           - term3*term4*Mp*g*rw^2 / term5 ...
           + state($,4)^2 * Mp*l*term4*rw^2 / term5 ...
           + Vs*K*rw / term5;

   foo = [ state($,2), ...               // d/dt x = v
           v_dot, ...
           state($, 4), ...      // d/dt theta
           - g * term4 / l - state($, 2) * term3 / l ...       // d/dt omega
           ];
endfunction

//FIRST_ORDER PROGRAM
// Integration Set the time length and step size for the integration
//steps = 5000;   // The number of steps to use
//t_start = 0;    // the start time - normally use zero
//t_end = 10;     // the end time
//h = (t_end - t_start) / steps; // the step size
//t = [t_start];  // an array to store time values
//for i=1:steps,
// t = [t ; t($,:) + h];
// X = [X ; X($,:) + h * derivative(X($,:), t($,:), h)];       // first order
//end

//RUNGE_KUTTA INTEGRATION
// Set the time length and step size for the integration
steps = 1000;
t_start = 0;
t_end = 10;
h = (t_end - t_start) / steps;
t = [t_start];
// Loop for integration
for i=1:steps,
t = [t ; t($,:) + h];
F1 = h * derivative(X($,:), t($,:));
F2 = h * derivative(X($,:) + F1/2, t($,:) + h/2);
F3 = h * derivative(X($,:) + F2/2, t($,:) + h/2);
F4 = h * derivative(X($,:) + F3, t($,:) + h);
X = [X ; X($,:) + (F1 + 2.0*F2 + 2.0*F3 + F4)/6.0];
end

// Graph the values for part e)
plot2d(t, [X(:,1) + l*sin(X(:,3))], [-2], leg="mass position");
plot2d(t, [X(:, 3), X(:,4)], [-2, -5], leg="position@theta (X 10)");
//plot2d(t, [10*X(:, 3)], [-2, -5], leg="position@theta (X 10)");
```

Figure 12 (continued): Scilab system simulation program

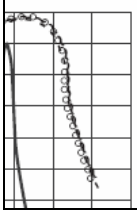
```
     xtitle('Time (s)');


     // printf the values over time
     intervals = 20;
     for time_count=1:intervals,
        i = int((time_count - 1)/intervals * steps + 1);
        xmass = X(i,1) + l*sin(X(i,3));
        printf("Mass location at t=%f  x=%f \n", i * h, xmass);
        // printf("Point at t=%f  x=%f, v=%f, theta=%f, omega=%f \n", i * h, X(i, 1), X(i, 2),
X(i, 3), X(i, 4));
     end


     // find the settling time in the results array
     ts = 0;
     for i = 1:steps,
        xmass = X(i,1) + l*sin(X(i,3));
        if xmass > (Cd + tolerance) then ts = i*h + t_start;
        end
        if xmass < (Cd - tolerance) then ts = i*h + t_start;
        end
     end
     printf("\nTheoretical settling time %f \n", ts);
```

Figure 12 (continued): Scilab system simulation program

6.5 Appendix E: Purchasing Evidence

The receipt for the infrared sensor by Sharp, batteries and switches from Radioshack, are attached.

Fig. 18



Photoconductive Cells

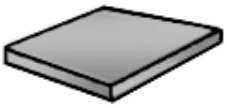
Dim. D (mm)	Cell Resistance* 10 Lux @ 2856K Min. (K ohm)	Max. (K ohm)	Dark Min. (M ohm)	SEC	Peak Wavelength (nm) Typ.	Rise/Fall Time (ms) Typ.	Digi-Key Part No.	Price Each 1	25	100	250
Fig. 18, Curve Line G											
11.00	8	16	0.3				PDV-P5001-ND	.71	.61	.59	.56
	12	30	0.5	10	520	55/25	PDV-P5002-ND	.71	.61	.59	.56
	12	58	1				PDV-P5003-ND	.71	.61	.59	.56
5.10	3	11	0.2			50/20	PDV-P8001-ND	.57	.49	.48	.45
	8	24	0.5			50/20	PDV-P8002-ND	.57	.49	.48	.45
	16	33	0.5	10	520	55/20	PDV-P8003-ND	.57	.49	.48	.45
	20	60	0.5			55/20	PDV-P8004-ND	.57	.49	.48	.45
	40	120	1			60/25	PDV-P8005-ND	.57	.49	.48	.45
	80	240	5			60/25	PDV-P8006-ND	.57	.49	.48	.45
	4	11	0.15			55/20	PDV-P8101-ND	.57	.49	.48	.45
	9	20	0.3	10	520	60/25	PDV-P8102-ND	.57	.49	.48	.45
	16	33	0.5			60/25	PDV-P8103-ND	.57	.49	.48	.45
	27	60	2			60/25	PDV-P8104-ND	.57	.49	.48	.45
	50	94	2.5				PDV-P8105-ND	.57	.49	.48	.45
	50	140	20	10	520	60/25	PDV-P8106-ND	.57	.49	.48	.45
	80	240	20				PDV-P8107-ND	.57	.49	.48	.45

Figure 13: Photoresistor cost table [7]

Form > Material > Length x Width > Thickness > Clear > Polycarbonate Material

Plastics

This item matches all of your specifications.



Part Number 8574K43 $15.06 Each

Form	Sheets
UL Rating	UL 94HB and UL 972
Material	Polycarbonate
Polycarbonate Material	Polycarbonate
Length x Width	12" x 24"
Thickness	.25"
Thickness Tolerance	+0 to -.026"
Length Tolerance	+.625"
Width Tolerance	+.625"
Clear	Clear with No Tint
Lowest Operating Temperature	Below +45° F
Highest Operating Temperature	Above +100° F and Above +212° F
Operating Temperature Range	-40° to +240° F
Performance Characteristic	High Impact Strength and Flame Retardant and Weather Resistant
Tensile Strength	9000 psi
Impact Strength	12 ft.-lbs./in.
Specifications Met	Underwriters Laboratories (UL) Ratings
Hardness	Rockwell R: 118

Figure 14: Polycarbonate material sheet price [5]

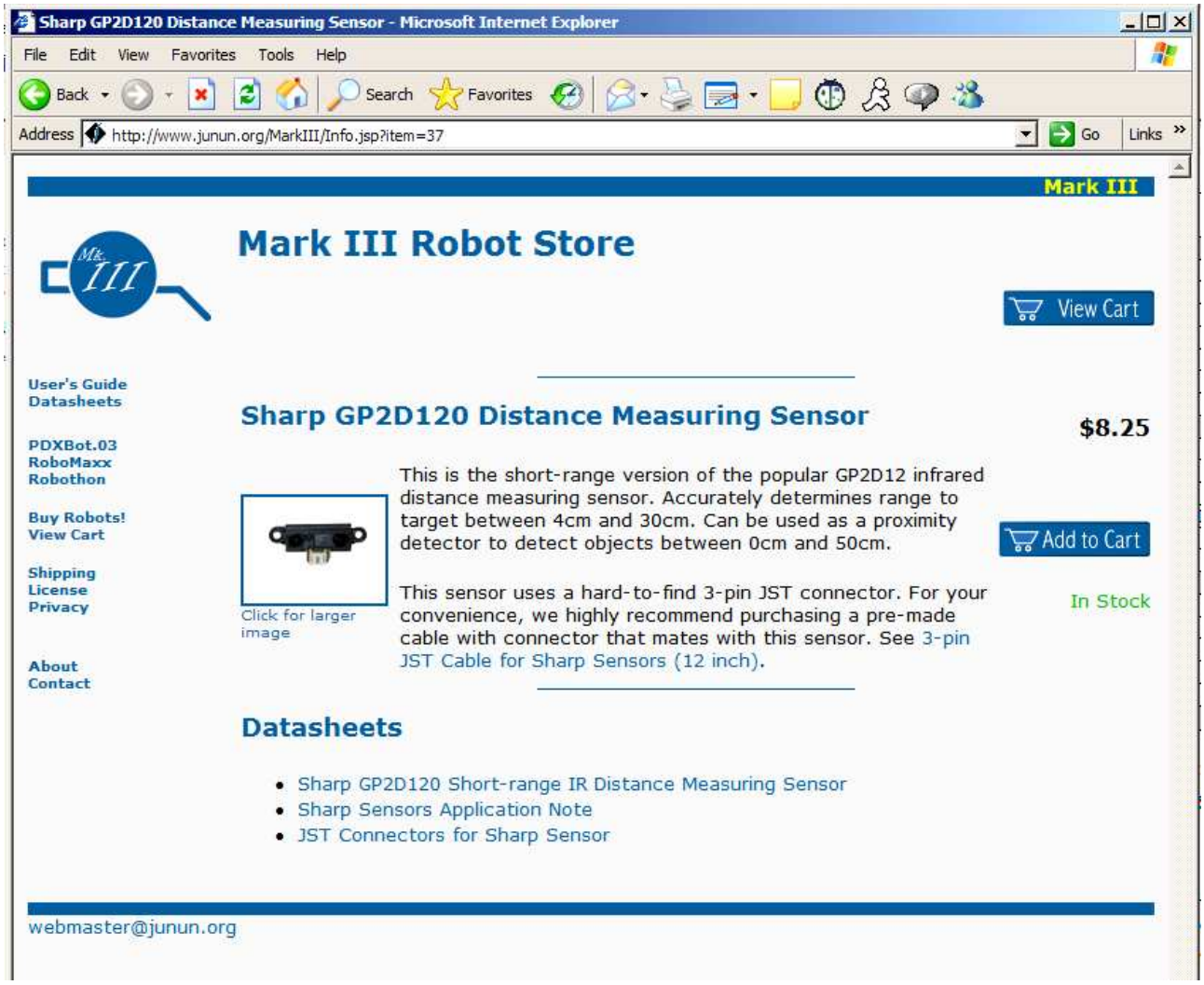


Figure 15: Sharp GP2D120 distance measuring cost sheet

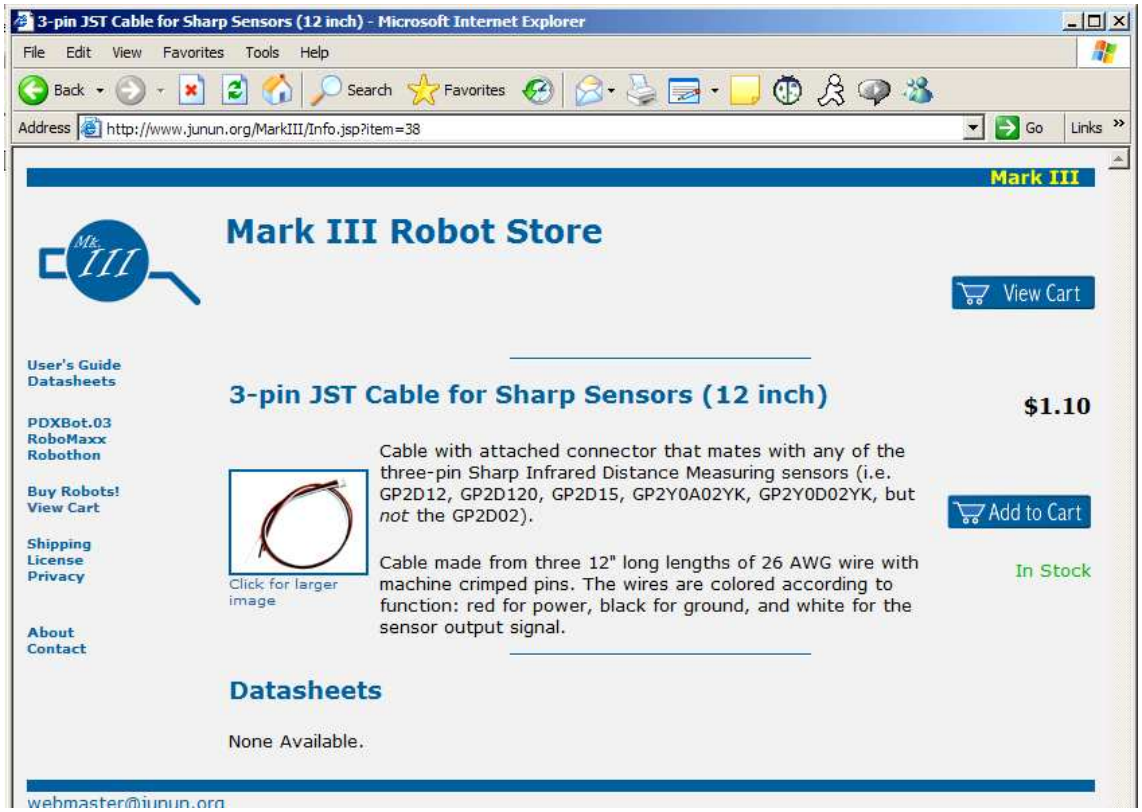
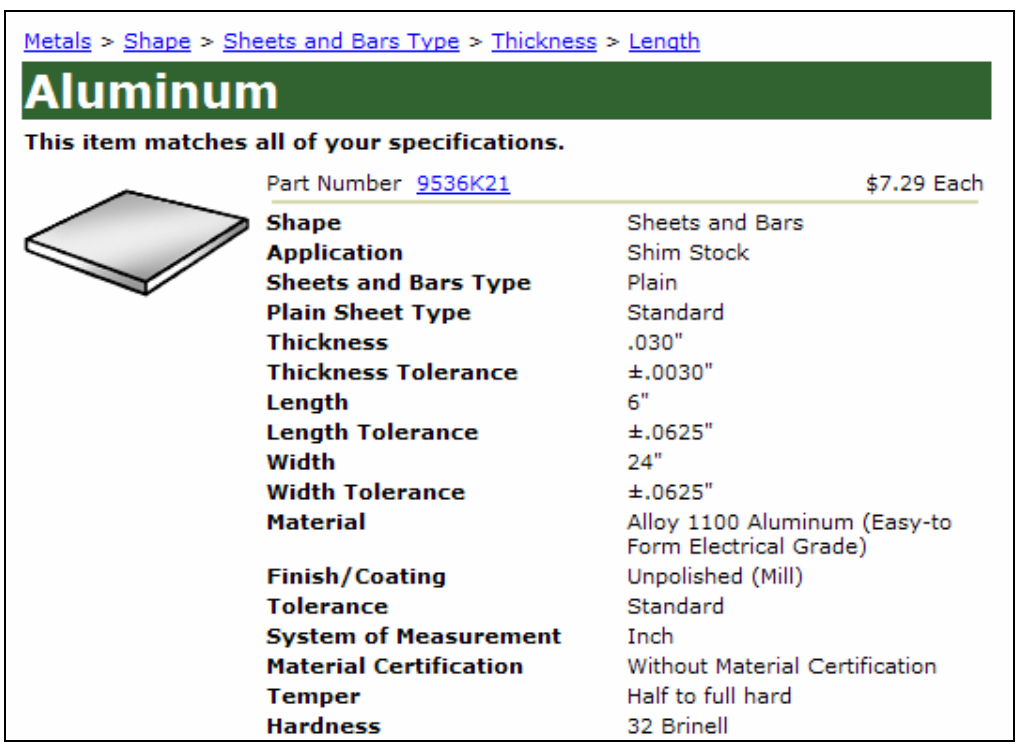


Figure 16: 3-pin JST cable cost sheet



Metals > Shape > Sheets and Bars Type > Thickness > Length

Aluminum

This item matches all of your specifications.

Part Number 9536K21 $7.29 Each

Shape	Sheets and Bars
Application	Shim Stock
Sheets and Bars Type	Plain
Plain Sheet Type	Standard
Thickness	.030"
Thickness Tolerance	±.0030"
Length	6"
Length Tolerance	±.0625"
Width	24"
Width Tolerance	±.0625"
Material	Alloy 1100 Aluminum (Easy-to Form Electrical Grade)
Finish/Coating	Unpolished (Mill)
Tolerance	Standard
System of Measurement	Inch
Material Certification	Without Material Certification
Temper	Half to full hard
Hardness	32 Brinell

Figure 17: 22 gage aluminum sheet pricing [1]