

Mechanical / Electrical / Software
Team Design and Prototyping
for a
Sun Tracking Solar Energy Collector
System

For Mechanical Engineering 4B03

Authors:

Tyler Kenyon

Yajie Jiang

Allan D. Spence

August 2010

Table of Contents

1. Introduction	1
2. Fixed vs. Tracking Solar Panels.....	1
3. Orbital Mechanics Review	2
4. System Requirements Summary.....	3
5. Components and Equipment	4
6. Conceptual Mechanical Design.....	7
6.1 Azimuth-Tracking Base	8
6.2 Zenith-Tracking Upper Stage.....	9
7. CAD Detail Design and Assembly	14
7.1 Azimuth-Tracking Base	14
7.2 Zenith-Tracking Top.....	14
9. Physical Model Realization from CAD.....	18
10. Summary	18
Acknowledgements.....	18
11. Appendix	19
Appendix A1 – Electrical Implementation.....	19
Appendix A2 – Arduino Software Tutorial	32
Appendix A3 – Software Implementation.....	37
Appendix A4 – CAD Drawings.....	38
Appendix A5 – Rapid Prototyped Component Costs	39
References	40

1. Introduction

Harnessing solar photovoltaic energy is a promising green energy technology. To ensure highest energy collection efficiency, it is proposed that an automatic sun tracking system be developed to continuously align the collector surface normal with the instantaneous “solar vector” (the direction from the collector towards the sun. An example of a full scale system is shown in Fig. 1. The focus of this project is twofold:

1. Mechanical design and prototyping
2. Electronic and software design and implementation

For consistency with established industry and scientific standards, the system will track using Azimuth and Zenith angles (Fig. 2) [1]. The project deliverables are design documents and an actual scale model prototype that faithfully represents the design required for a full size system. Coordinated team effort of students in both Mechanical and Mechatronics Engineering is mandatory.

The CAD design for the solar tracking system was created using Autodesk Inventor 2010 [2]. Prototype plastic components were produced using a Dimension BST [3] rapid prototyping machine. The software and electronics are based on an Arduino Duemilanove [4] microcontroller.

2. Fixed vs. Tracking Solar Panels

Small solar panels are frequently installed at a fixed orientation, whereas larger installations typically track the solar vector using 2-axis Azimuth/Zenith axes. The return on investment for the added expense of tracking can be estimated by referring to insolation data available from [7]. For Canada, the preferred fixed orientation is a due South facing Azimuth, with Zenith angle corresponding to the latitude. Using Hamilton as an example, selected mean daily insolation values [8] are reproduced in Table 1. During December days, when the days are short and the sun is predominantly towards the South, there is a limited gain from 2-axis tracking. During the long days of June and July, the 2-axis advantage is $8.9 \text{ kWh/m}^2 / 5.5 \text{ kWh/m}^2 = 62$ percent. The reason for the 2-axis advantage is the greater Azimuth range in the summer. Recognizing that the extreme weather summer air conditioning load causes a greater electricity load than winter heating [9], harnessing the extra energy from solar tracking on hot summer days is attractive. It is a balance between initial capital cost, extra maintenance, and power consumed for tracking vs. extra energy collected – essentially an economy of scale. This optimization is an essential engineering design issue for full scale implementation. For the scale model project in this course, it is sufficient to recognize the significant potential gain for

tracking, but limit activity to demonstrate application of the course product design and development concepts.

3. Orbital Mechanics Review

The sun changes position in the sky because of two effects (Fig. 3):

1. Daily rotation of the Earth
2. Seasonal variation due to 23.5 degree rotation axis tilt

The sun position (or solar vector) has been observed for thousands of years, and is known to high accuracy. Using almanac tables or online software [6], sunrise and sunset information for specific latitudes and longitudes can be obtained. For example (Fig. 4), on June 21, 2010 at Toronto, Ontario, the sun rises at 5:36am EDT at an Azimuth of 56 degrees East of North. It sets at 9:03pm at an Azimuth of 304 degrees. At noon the Altitude angle is 69.8 degrees. The Zenith angle is $90 - 69.8 = 20.2$ degrees. Because the latitude of Toronto is 43.7 degrees North, and the axis tilt is 23.5 degrees, a Zenith angle of $43.7 - 23.5 = 20.2$ degrees is expected.

The fact that, during the Northern summer, the sun rises and sets North of East/West, is also due to the axis tilt. See Fig. 5 for an illustration. At 43.7 degrees North latitude, the morning is longer by 24.49 degrees. Since 15 degrees is one hour, this corresponds to a longer morning by 1.63 hours or 1 hour and 38 minutes. The evening is similarly lengthened for a total day length of $12 + 3.26$ hours = 15.26 hours = 15 hours 16 minutes. Actual length of day is slightly longer than this due to atmospheric refraction (about 7 minutes in total at the equator).

At times other than sunrise, noon, or sunset, the solar vector can be calculated using published algorithms [1]. This particular algorithm accepts as input the location latitude and longitude, and the time of day expressed in Coordinated Universal Time (UTC), the mean solar time at the prime longitude meridian passing through the Royal Observatory at Greenwich, England. This information is available using a Global Positioning System (GPS) receiver. For a fixed position, the latitude and longitude are constant, and a real-time clock is sufficient. The algorithm output is the solar vector Azimuth angle and Zenith angle, as defined in Fig. 2.



Figure 1. Example Full Scale Sun Tracking Solar Photovoltaic Energy Collection System (photo by A. Spence)

Table 1. Mean Daily Insolation (in kWh/m²) for Hamilton, Ontario (summarized from [8])

	South-facing vertical (tilt=90°)	South-facing, tilt=latitude	South-facing, tilt=latitude+ 15°	South-facing, tilt=latitude- 15°	Two-axis sun- tracking	Horizontal (tilt=0°)
January	2.8	2.8	2.9	2.5	3.3	1.6
June	2.6	5.4	4.7	6.0	8.8	6.2
July	2.7	5.5	4.8	6.1	8.9	6.3
December	2.2	2.2	2.3	2.0	2.6	1.2
Annual	2.9	4.2	4.0	4.3	5.9	3.7

4. System Requirements Summary

For this report, the prototype solar tracker is to be a scale model of a proposed full size design with the following requirements:

- It is intended for use in Canada at latitudes between 40 and 60 degrees North.
- So far as is possible, it is to utilize the components and equipment described herein.
- It is to calculate azimuth/zenith angles using the current time obtained from the real time clock.
- It is assumed that the real time clock remains powered up from its backup battery and is always reliable, even if power is lost to the Arduino and stepper motors. Upon power up (including after a power failure), the system must recover in a way that avoids overtravel or wind-up of either the azimuth or zenith stage.
- Power will be provided using a wall transformer supplying 9 VDC 1000 mA total.
- During daylight hours, the system must track to within ± 1 degree. After dark the system is to park with azimuth South and Zenith angle zero (upwards). A pinhole or other means to verify tracking fidelity must be included.
- The prototype must be reasonably safe to operate with low risk of injury to operators, or damage to equipment.

5. Components and Equipment

Components

The project involves mechanical/electrical/software (mechatronic) product development. Emphasis is on the mechanical aspects, with judicious choice of electrical/software interface. The following list describes the supplied components available to complete the project.

- **Arduino Duemilanove microcontroller:**
This microcontroller provides a very simple programmable interface to the electrical switches, motor, clock, etc. For communication and customization, it is connected to a Windows / Mac OS computer using a standard USB cable. The associated programming software is available at no charge. Implementation of the current time to azimuth/zenith angle conversion program [1] is provided for you to use. Refer to Appendix A1 for details.
- **Arduino Duemilanove custom expansion shield:**
To provide a standard interface for the project, an expansion shield is provided. This custom made shield includes (see below) a DS 1307 real time clock, two MC 3479 stepper motor drivers, and support components for the mechanical limit and optical homing switches. Refer to Appendix A2 for details.
- **ElectroMechanical Components:**
Autodesk Inventor CAD drawings are provided in Appendix A3, and computer readable model files are available on the course web site.

- **Japan Servo KP35FM2-035 Stepper Motors**
These motors have a full step resolution of 1.8 degrees (200 steps per revolution). Maximum torque is 700 g-cm of torque at 24 VDC / 500mA. Installed conditions are 9VDC, and hence a maximum torque of 250 g-cm is realistic.
- **Mechanical Limit Switch**
A mechanical limit switch is used. On the provided prototype, the switch will be triggered when the azimuth stage approaches North, and can be sensed by the microcontroller. Together with the optical homing switches, checking for unintended azimuth wind-up after power loss can be programmatically detected and recovered from.
- **Optical Homing Switches**
Optical switches are used to sense, with high repeatability, both the azimuth and zenith home positions. By interrupting the LED light beam, the provided prototype homes the azimuth at 90 degrees (East), and the Zenith at 90 degrees (horizon).
- **Significant Mechanical Hardware:**
 - **Azimuth Stage Lazy Susan Bearing:**
The chosen 3 inch Diameter Lazy Susan bearing (Lee Valley Tools SKU 12K01.01) is used to take the gravity weight thrust load for the Azimuth rotational stage on the prototype. This bearing is quite loose, and a higher quality choice would be required for a full size implementation.
 - **Mounting Hardware:**
Unless stated otherwise, the threaded fasteners are #6-32 thread installed in 4 mm diameter clearance holes. For snugness, pivots use metric M4-0.7 thread. A length of #10-32 threaded brass rod, with corresponding nuts, was used for the prototype Zenith stage lead screw.
 - **Sheet Plastic:**
Small quantities of 0.125 inch and 0.250 inch thick clear sheet plastic are available from the Mechanical Engineering Technical Services staff located at JHE 205.

Equipment

- **Digital Multimeter Circuit Test Kit:**
The electrical components are prewired for connection to the Arduino custom expansion shield. For convenience, minor circuit testing can be accomplished using the Canadian Tire kit #52-0064-4 digital multimeter, wire strippers, and pliers.

- **Small Wrenches, Electric Drills, etc.:**

Small wrenches for securing fasteners, etc. are available with the multimeter kits. Electric drills for drilling sheet plastic are available from the Mechanical Engineering Technical Services staff located at JHE 205. Successful completion of a shop safety training course at the beginning of the term is required before using powered equipment.

- **Rapid Prototyping Printer:**

Rapid Prototyping services are provided by Mechanical Engineering Technical Services for a fee of \$12 per cubic inch of part material. The Dimension BST 768 maximum build size is 8 inches (~200 mm) × 8 inches (~200 mm) × 12 inches (~300 mm). The printing resolution is 0.010 inches (~0.25 mm). Support material will automatically be added by the Dimension Catalyst software. There is no charge for support material, but you should consider the need to manually remove the support material in your design.

The printer uses ABS plastic, a material with approximate material properties:

Young's modulus=2.89 GPa

Poisson's Ratio=0.38

Yield Strength=40 MPa

Density=1.06 g/cm³

Because the material is not truly solid, it is recommend that your design assume stiffness and strength values that are 50% of solid ABS. Using a wall thickness no thinner than 0.060 inches (1.5 mm) to avoid rapid prototyping layer delamination. To most faithfully emulate plastic injection moulding, constant wall thicknesses should be used.

- **Computer Aided Design Software:**

The faculty has now standardized on Autodesk Inventor for general CAD use. All work submitted for grading must be created using Inventor (no exceptions). It is expected that a kinematically correct mechanism solution will be completed before physical prototyping. Finite element analysis is not essential, but reasonable judgment of part stiffness is expected.

When saving parts in STL format for rapid prototyping, select the option to use highest resolution. Designs will be reviewed by the instructor/TAs/Technical Services staff before printing. Advice to revise designs are provided to avoid wasted printing, and the final decision when and if to print any part rests solely with the instructional team.

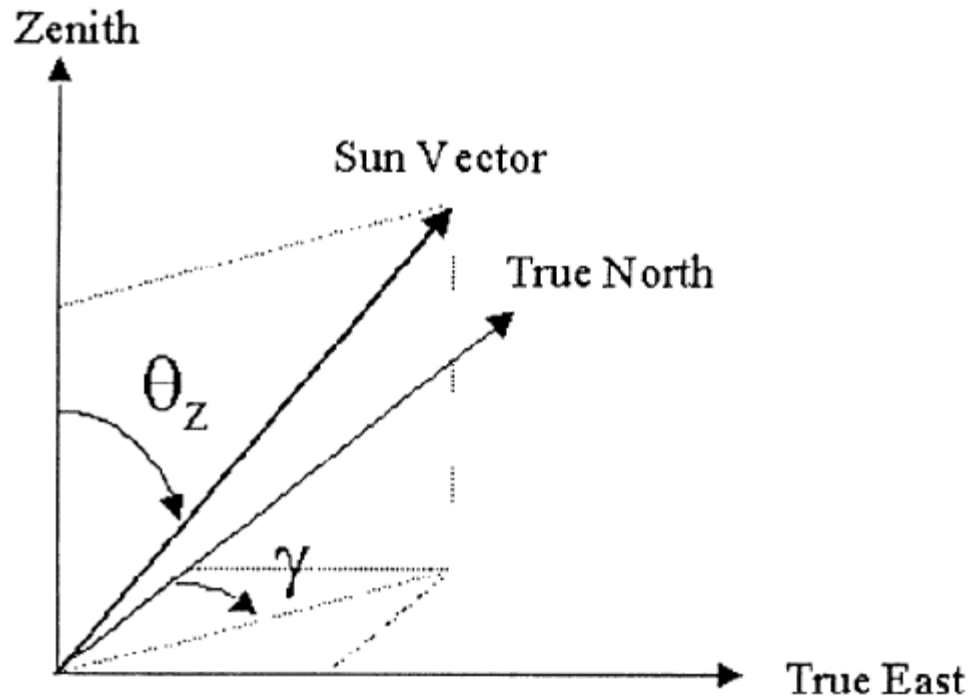


Figure 2. Solar Vector Angle Definitions (from [1]). The Azimuth angle (γ) is measured from True North towards True East. Therefore True North corresponds to $\gamma = 0$, True East corresponds to $\gamma = 90$ degrees, True South corresponds to $\gamma = 180$ degrees, and True West corresponds to $\gamma = 270$ degrees. The Zenith angle (θ_z) is measured from directly overhead. Hence $\theta_z = 0$ means that the sun is directly overhead, casting no shadow. This occurs, for example, at local solar noon on the equator on an Equinox (approximately March 20 or September 22).

6. Conceptual Mechanical Design

In order to fully tracking the sun from sunrise to sunset using the supplied real time clock and Azimuth/Zenith calculation algorithm, the preferred mechanical design should employ matching rotary axes. For a maximum latitude of 60 degrees North (such as near Yellowknife, NWT), at the summer solstice the sun rises at 27 degrees (measured East from North), and sets at 333 degrees. At 40 degrees North (such as near Windsor, ON), the Zenith stage, on which the solar panel is mounted, must cover a range from 90 degrees (horizon) to 16.5 degrees from overhead.

To avoid dependence on electric motor torque to maintain position against wind and gravity forces, good balancing and self-locking gear reduction should be used. For rotary motion, a worm gear can be used. For linear motion, a lead screw can be used. For learning purposes, the prototype described in this report uses a worm gear for the Azimuth stage and a lead screw for the Zenith stage. This is not necessarily optimal, and you are expected to critically evaluate choices for your team design based on specific requirements announced for the current year course offering.

6.1 Azimuth-Tracking Base

For learning purposes, and to permit customization, the Azimuth worm gear drive was designed using the Design Accelerator within Autodesk Inventor. Sizes were approximated to fit the supplied stepper motors, switches, and Lazy Susan bearing. For a demonstration prototype, this is somewhat a best judgment approach. Full size designs must include optimization for technical capability, attractiveness, environmental compatibility, cost, etc. A worm (the screw like gear that attaches to the stepper motor) pitch diameter of 20 mm was chosen. The center distance (from the center of the worm/motor axis to the center of the large worm gear axis (aligned with the Azimuth axis) was set to 82 mm (Fig. 6).

A coarse approximation for the ABS dry coefficient of friction is $c_f = 0.5$. For the chosen lead/helix angle $\gamma = 11.3$ degrees, $\tan(\gamma = 11.3^\circ) = 0.2 \ll 0.5 = c_f$ and hence self locking is assured.

With the chosen ratio of 36, the motor/worm will revolve 36 times for each revolution of the larger (Lazy Susan axis) worm gear. Each motor revolution equals 10 worm gear degrees, and hence with 200 full steps per motor revolution, the resolution is 10 degrees / 200 steps = 0.05 degrees per step.

At power up, the microcontroller has no information on the current Azimuth angle, and hence an initialization procedure is needed. After initial installation, a properly operating system should never be outside of the 27 to 333 degree Azimuth range, even if power is lost. Hence a power up motion that turns the Azimuth stage Eastward to an initialization position of 22.5 degrees should always succeed. In the prototype, this is accomplished using a small screw that protrudes downward from the Azimuth worm gear and interrupts the optical homing switch LED light beam. The microcontroller senses the change in switch status, stops the stepper motor, and initializes its internal Azimuth angle counter to correspond to an angle of 22.5 degrees. If, for any reason, the optical switch sensing procedure fails, motion will continue until an angle of zero degrees (due North) is reached. At this point a second, upward protruding screw will contact the mechanical roller-lever limit safety switch, disconnecting stepper motor power independent of the microcontroller. This is considered to be a fault condition that can

only be cleared by depressing the “override limit” switch and commanding the stepper motor to move the Azimuth stage Westward, or by manually moving the Azimuth worm gear Westward beyond 22.5 degrees while the stepper motor is unpowered.

6.2 Zenith-Tracking Upper Stage

The Zenith tracking stage similarly must be self-locking to avoid uncontrolled movement in the event of electrical power loss, high winds, and the gravity force due to the solar panel mass. A worm gear drive similar to that used for the Azimuth stage may be the best choice for practical implementation. For academic illustration, the prototype described herein uses a self-locking lead screw implementation. The kinematic concept is shown as a CAD sketch in Fig. 7. All units are in millimeters and degrees. The lead screw nut pivot is located at point $A = (X, 0)$, where $X = 0$ when the pivot is located at point D . As the stepper motor turns the lead screw, the nut pivot is constrained to travel along the horizontal lead screw bounded by points $M = (24, 0)$ and $N = (88, 0)$. The solar panel pivots are located at points B and C .

Using trigonometry, the non-linear kinematic relationship between the linear motion of the nut pivot A and the Zenith angle θ (theta) can be expressed using the system of equations

$$\begin{aligned} X &= |BC|\cos\theta + |AB|\cos(\theta + 180^\circ + \beta) \\ &= |BC|\cos\theta - |AB|\cos(\theta + \beta) \\ 0 &= 15 + |BC|\sin\theta + |AB|\sin(\theta + 180^\circ + \beta) \\ &= 15 + |BC|\sin\theta - |AB|\sin(\theta + \beta) \end{aligned}$$

For a desired Zenith angle θ , iterative Newton-Raphson zero finding techniques can be used to solve for the corresponding nut pivot position $A = (X, 0)$. Details are presented in the software algorithm implementation (Appendix A1).

The lead screw used has a #10-32 thread (32 threads per inch), and hence the nut pivot advances by $1/32 * 25.4 = 0.79375$ mm per revolution or $0.79375 / 200 \approx 0.004$ mm per step.

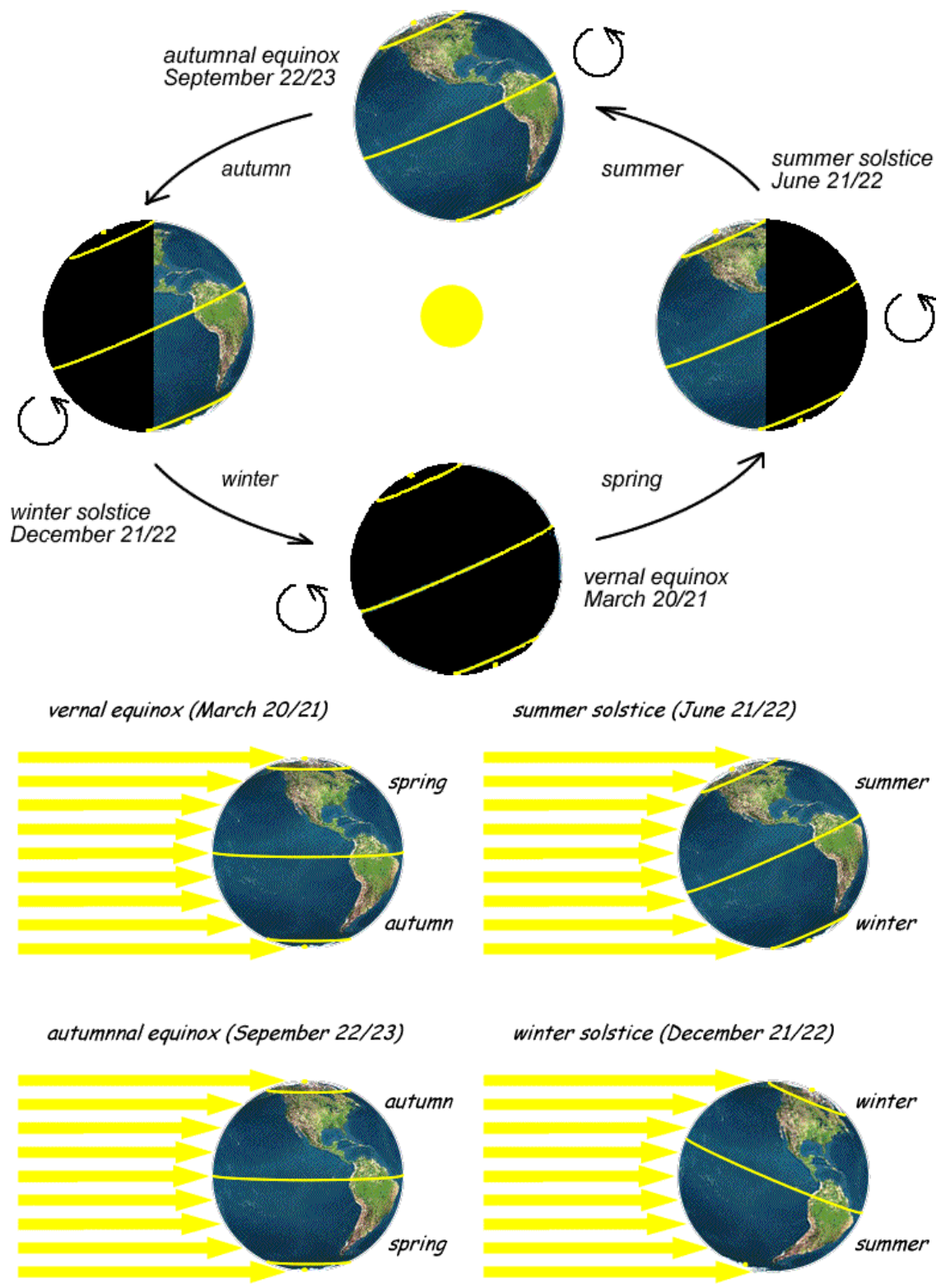


Figure 3. Earth Orbit and Axis Tilt Effect on Seasons (from [5])

Rising and setting times for the Sun

Date	Sunrise	Sunset	Azimuth		Length of day		Solar noon		
			Sunrise	Sunset	This day	Difference	Time	Altitude	Distance (10^6 km)
Jun 18, 2010	5:35 AM	9:02 PM	56° ↗	304° ↖	15h 26m 29s	+ 13s	1:19 PM	69.8°	151.993
Jun 19, 2010	5:36 AM	9:02 PM	56° ↗	304° ↖	15h 26m 38s	+ 09s	1:19 PM	69.8°	152.003
Jun 20, 2010	5:36 AM	9:02 PM	56° ↗	304° ↖	15h 26m 44s	+ 05s	1:19 PM	69.8°	152.013
Jun 21, 2010	5:36 AM	9:03 PM	56° ↗	304° ↖	15h 26m 45s	+ 01s	1:19 PM	69.8°	152.022
Jun 22, 2010	5:36 AM	9:03 PM	56° ↗	304° ↖	15h 26m 42s	- 03s	1:20 PM	69.8°	152.030
Jun 23, 2010	5:36 AM	9:03 PM	56° ↗	304° ↖	15h 26m 34s	- 07s	1:20 PM	69.8°	152.038

Figure 4. Sunrise and Sunset Data for June 21, 2010 at Toronto, Ontario (from [6])

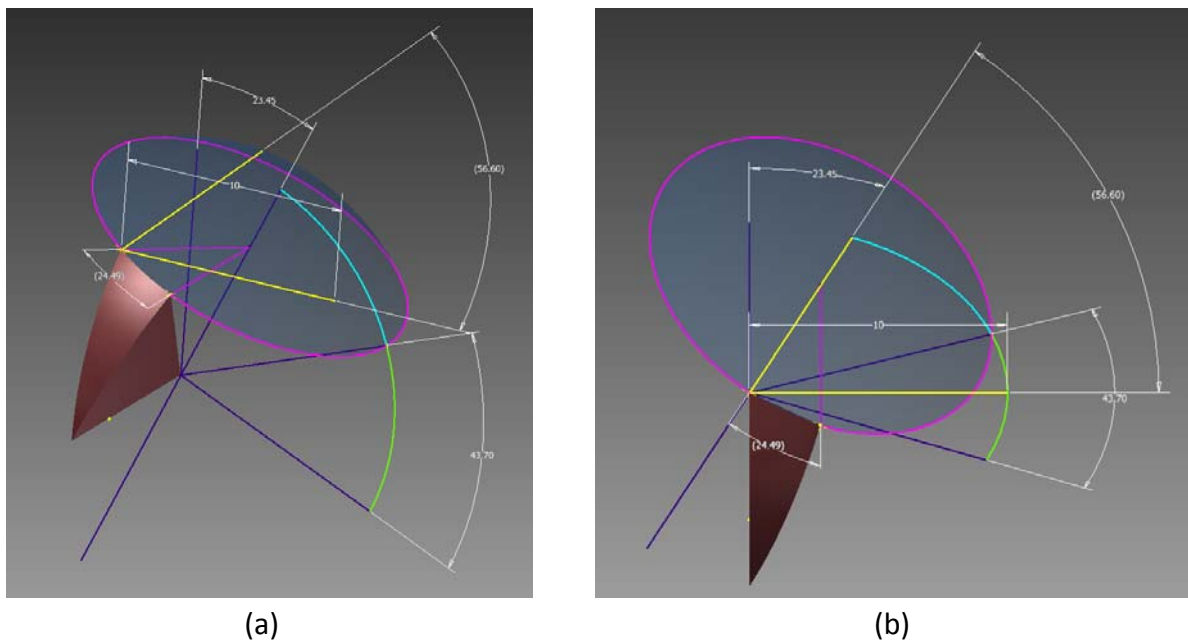


Figure 5. Summer Solstice Geometry at Toronto, Ontario. The earth axis tilt is 23.45 degrees. The latitude is 43.7 degrees north. The slice of earth (in the chestnut brown colour) is from 90 degrees east back to where the day/night boundary is on June 21. (a) At a latitude of 43.7 degrees, this lengthens the morning by 24.49 degrees of longitude, or 1 hour and 38 minutes. The evening length increases by an equal amount, for a total of 3 hours and 16 minutes, making the geometric day length 15 hours and 16 minutes. The extra time shown in Figure 4 is due to refraction near the horizon (up to 7 minutes near the equator). (b) To an observer on the surface of the earth at Toronto, the sun appears at the dawn horizon 56.6 degrees north of east.

Worm Gears Component Generator

Design **Calculation**

Common

Desired Gear Ratio: 36.0000 ul Tan. Module: 4.000 mm Tan. Pressure Angle: 20.0000 deg Helix Angle: 11.3099 deg

Preview... Center Distance

Worm

Component: Cylindrical Face

Number of Threads: 1.00000000 ul Start plane

Worm Length: 50.000 mm

Pitch Diameter: 20.000 mm Diameter Factor: 5 ul

Worm gear

Component: Cylindrical Face

Number of Teeth: 36.00000000 ul Start plane

Facewidth: 10.00000000 mm

Unit Correction: 0.0000 ul

Calculate OK Cancel <<

Input Type: ☐ Gear Ratio ☒ Number of Teeth

Size Type: ☒ Module ☐ Circular Pitch

Type of Gearing: ☐ Common ☒ Spiral

Unit Tooth Sizes

Worm ☐ Worm Gear

Addendum a^* : 1.0000 ul 1.0000 ul

Clearance c^* : 0.2000 ul 0.2000 ul

Root Fillet r_f^* : 0.3000 ul 0.3000 ul

Worm Size: ☒ Diameter Factor ☐ Helix Angle ☐ Pitch Diameter

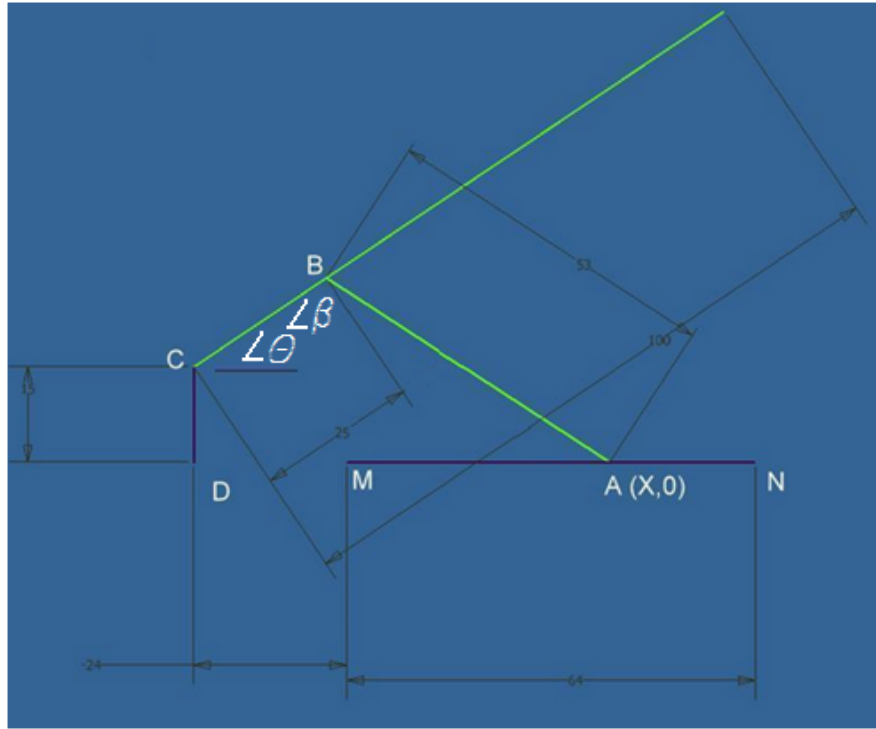
Gearing Choice

Center Distance: 82.000 mm Refresh

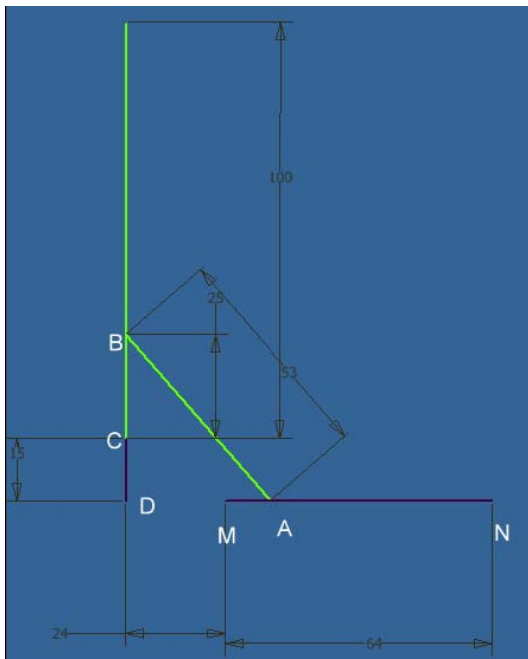
z	m_x	d	q	x
35	4.0000	25.20	6.30	-0.1500
35	3.5500	35.50	10.00	0.5986
35	2.8000	62.72	22.40	0.5857
36	3.5500	31.95	9.00	0.5986
36	3.5500	35.50	10.00	0.0986
37	3.5500	28.40	8.00	0.5986
37	3.1500	44.10	14.00	0.5317

OK Cancel

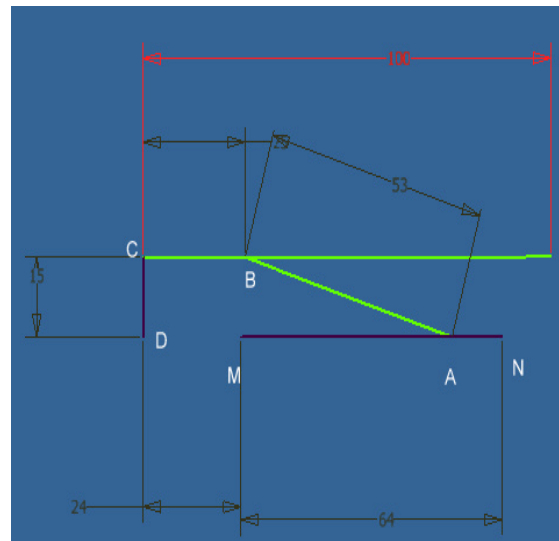
Figure 6. Azimuth Stage Autodesk Inventor Design Accelerator Worm Gear Parameters



(a)



(b)



(c)

Figure 7. Lead screw Zenith-Tracking Stage Kinematic Concept Sketch: (a) point and angle labels; (b) geometry at Zenith angle of 90 degrees; (c) geometry at Zenith angle of 0 degrees.

7. CAD Detail Design and Assembly

Computer Aided Design of all major system components was performed using Autodesk Inventor, with assemblies and kinematic mechanism constraints used to verify correct motion. As before, the CAD design is separated into the Azimuth and Zenith tracking stages. A brief summary is presented here. Detailed drawings are contained in Appendix A3.

7.1 Azimuth-Tracking Base

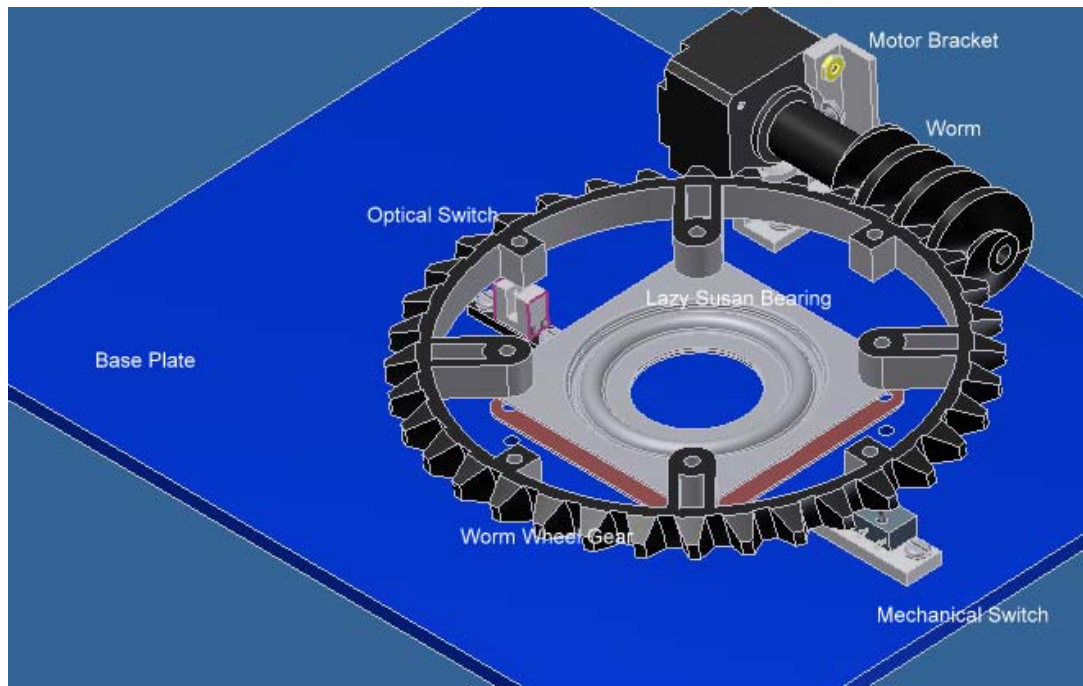
Fig. 8(a) shows the general CAD design of the azimuth-tracking base and its assembly situation. Other than purchased hardware, the detail designed components include an acrylic base plate, a rapid prototyped worm wheel gear ring, the worm, and a motor bracket. The base plate #6-32 clearance mounting holes were made using a workshop drill press / milling machine. The rapid prototyped stepper motor bracket is mounted to the base plate and adjusted so that correct gear engagement is achieved. The Lazy Susan bearing is positioned 10 mm above the plate using spacers and longer #6-32 screws, washers, and nuts. The worm gear is designed as a ring shape to reduce rapid prototyping material consumption. A corresponding hole pattern mates to the top plate of the Lazy Susan bearing. The optical switch is used to home the system at power up. A limit switch must be added to your design to avoid Azimuth wind-up.

Fig. 8(b) shows the design of worm and motor shaft coupling. The worm is designed with a central shaft-diameter hole. Two holes are drilled and tapped 90° apart to accept #6-32 plastic screws. The plastic screws protect the motor shaft from damage.

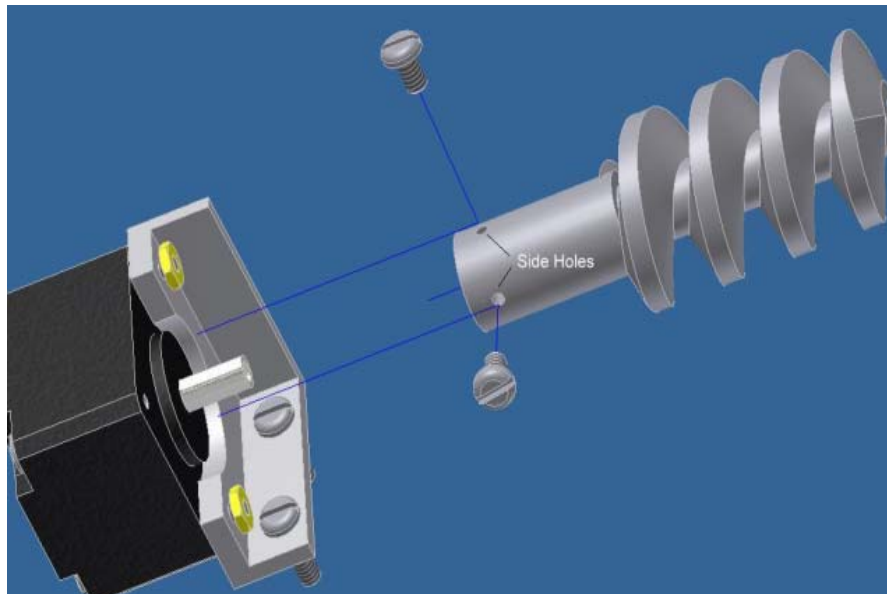
7.2 Zenith-Tracking Top

In addition to hardware components, the Zenith tracking top assembly (Fig. 9(a)) consists of an acrylic support plate, motor hub, motor bracket, lead screw slide bridge, sliding nut and pivot, crank, front pivot, pinhole sun locator, solar panel plate, homing optical switch mount, and homing tab. The crank frame is designed in a 'C' shape with strengthening corner ribs. The pinhole sun locator is installed above the panel using long #6-32 screws and nuts, and is used to verify the system performance. If the tracking system performs well the pinhole will aim sunlight onto the center mark on the solar panel. The two through holes in two upper corners of the panel can be used to install analog Light Dependent Resistors for optimal solar panel operation on cloudy days, and to take advantage of ground snow reflection. This optimization is beyond the scope of the course.

The homing optical switch is mounted onto the front side of the pivot (Fig. 9(b)). An extension tab, mounted on the solar panel plate, interrupts the optical switch at the 90° degree Zenith home position. Clearance slots are included to accommodate electrical wire routing.

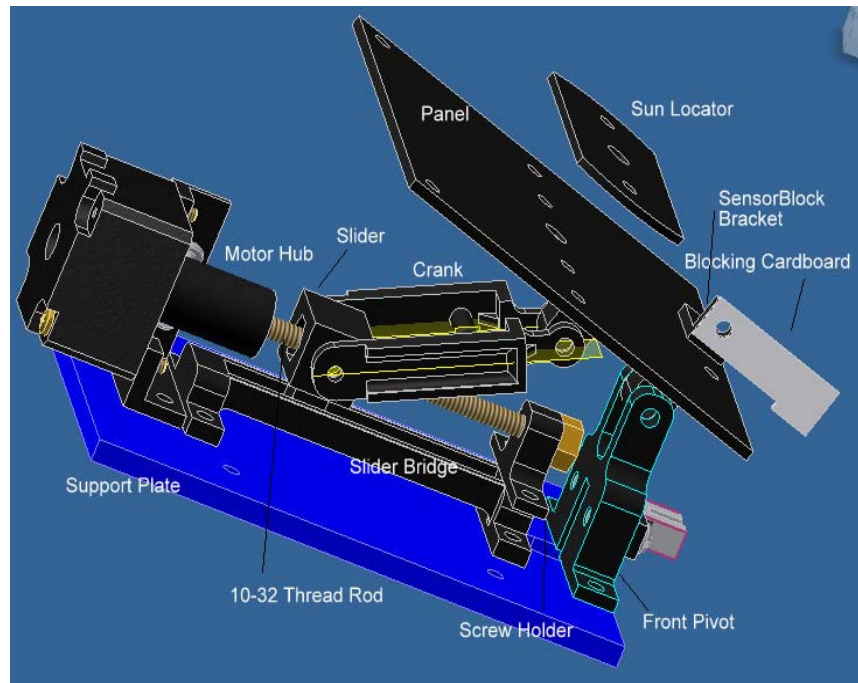


(a)

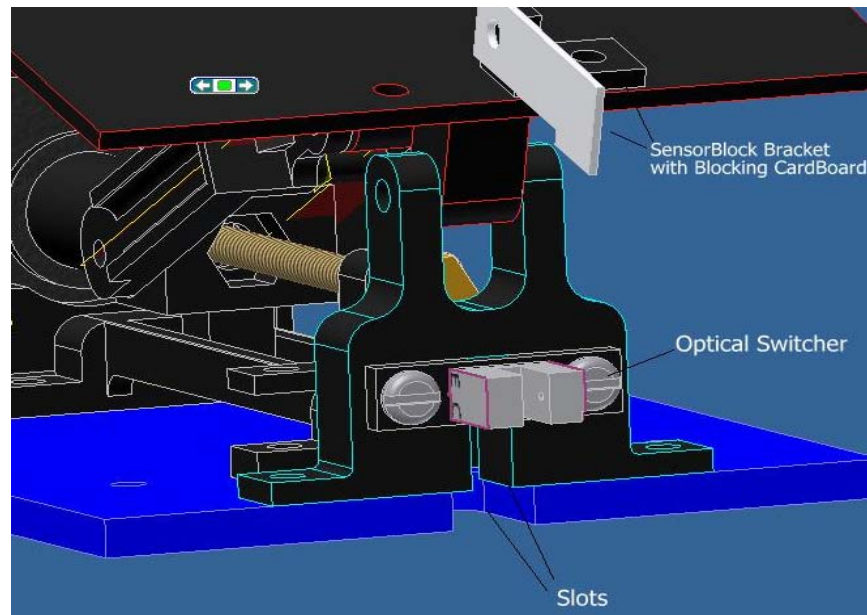


(b)

Figure 8. Azimuth Tracking Stage: (a) Base Design and Assembly; (b) Worm Gear Coupler



(a)



(b)

Figure 9. (a) Zenith Tracking Top Assembly; (b) Optical Switch Details

The rapid prototyped nut housing has axial #10-32 hexagonal depressions to accept the lead screw nut. The pivot axis has depressions to accept #6-32 nuts for the crank joint. The bridge has a slot to allow the sliding motion along the rail (Fig. 10). 8. Kinematic Simulation

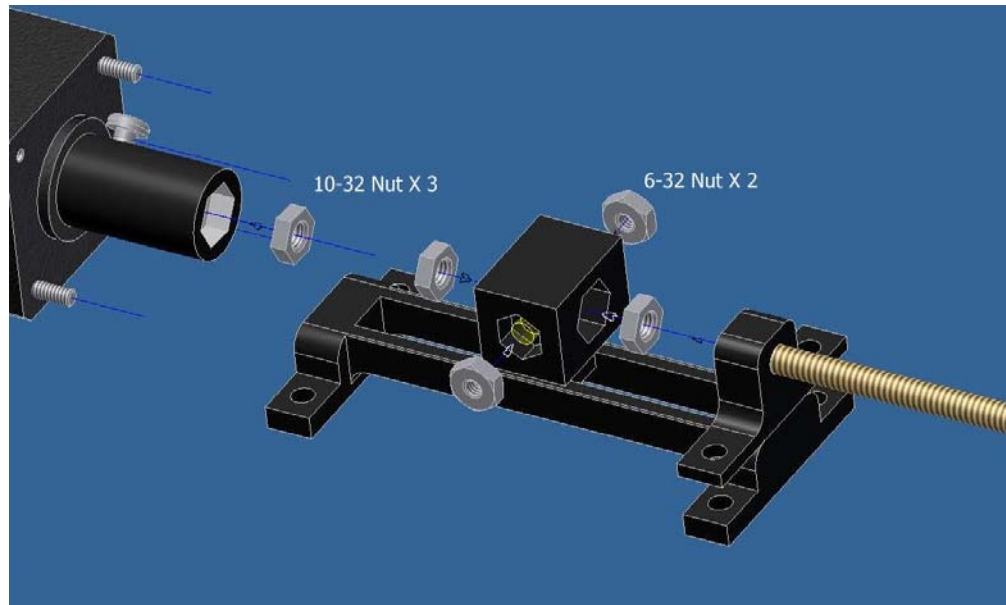
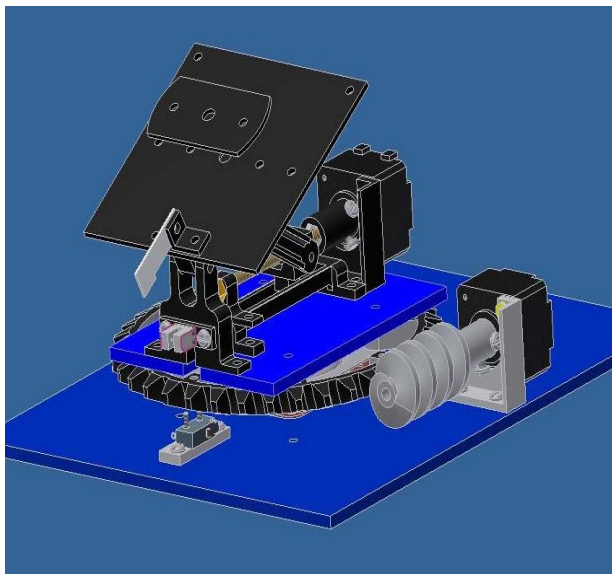
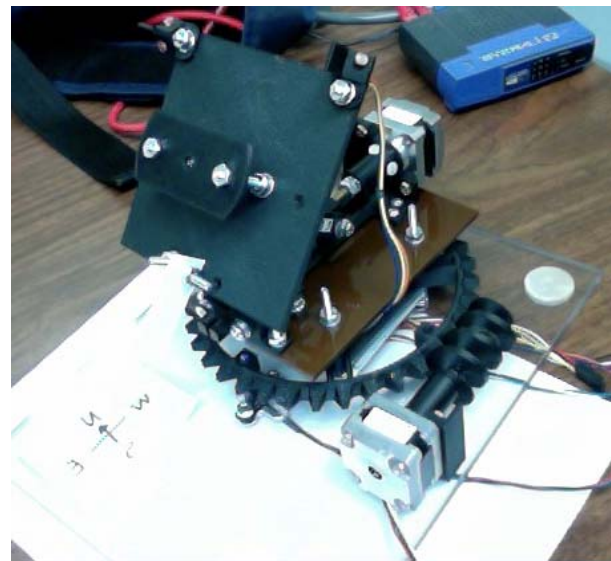


Figure 10. Lead Screw Details



(a)



(b)

Figure 11. Final (a) CAD; and (b) Physical Models

To simultaneously simulate the Azimuth and Zenith tracking motion, the Autodesk Inventor Assembly Studio application was used. First, the joint constraints were defined, including the positional constraint for the slider, the gear ratio constraint between worm and worm gear ring, and the rotational angular constraint for the lead screw rod. For each of the constraints, the corresponding motion speeds are set up according to the system kinematic properties.

9. Physical Model Realization from CAD

The rapid prototyped parts were created by saving from Inventor to .stl format, and then using the prototyping machine specific software to print the plastic parts. These parts were finally assembled with the electronics, fastener, and other hardware. A list of components and costs can be found in Appendix A4. Final CAD and physical models are shown in Fig. 11.

10. Summary

The established prototype of solar tracking system was designed with two degrees of freedom so that it is able to track the sun through Azimuth and Zenith. It contains two major assemblies: Zenith-tracking top and Azimuth-tracking base. The parts for each assembly were rapid prototyped. The Azimuth assembly consists of a worm gear drive and the Zenith assembly contains a lead screw mechanism.

An Arduino-compatible printed circuit board (PCB), called a shield, was designed and manufactured to mount and connect the supporting electronic components. The shield was modified to be fixed above the Arduino board and the layout was arranged to match the pin locations of the Arduino Duemilanove. The solar collector's PCB design was created using CadSoft EAGLE Schematic Capture and Layout Editor.

Acknowledgements

The authors gratefully acknowledge receipt of financial support from an Imperial Oil Charitable Foundation Departmental Grant, administered through the McMaster University Centre for Leadership in Learning. The Department of Mechanical Engineering, Faculty of Engineering, and McMaster Engineering Undergraduate Student Laboratory Endowment Fund (MacLAB) provided funding to purchase the rapid prototyping machine and supplies, and the electronic components. The Mechanical Engineering Technical Services staff provided valuable advice and shop services.

11. Appendix

Appendix A1 – Electrical Implementation

Arduino Duemilanove

To improve the efficiency of a solar energy collector, an active tracker that continuously accommodates the varying position of the sun is necessary. To meet these requirements, the Arduino Duemilanove [4] microcontroller board is responsible for calculating the solar vector and controlling the stepper motors. The Arduino Duemilanove is based on the AVR ATmega 328 and includes 14 digital input/output pins and 6 analog inputs (Fig A1-1).

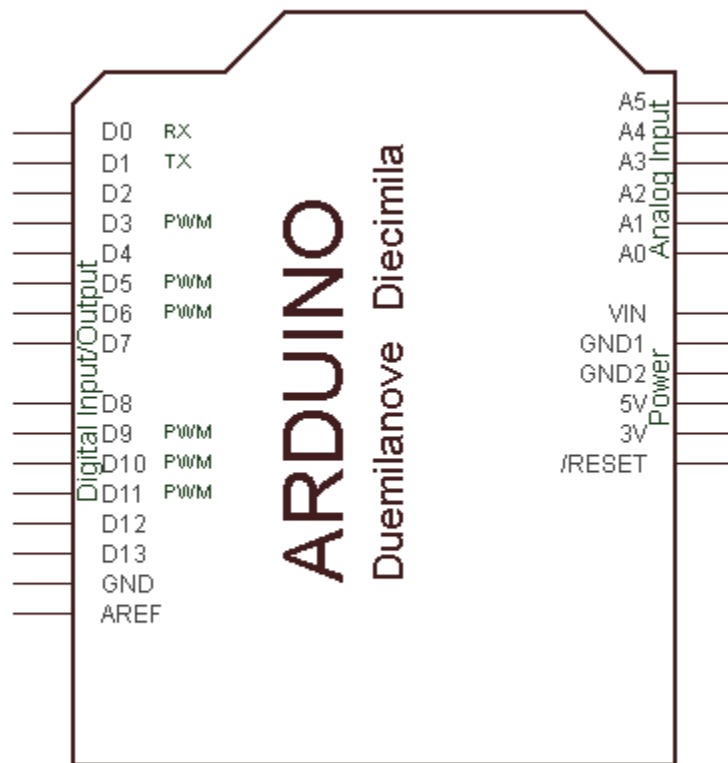


Figure A1-1. A circuit schematic symbol for the Arduino Duemilanove demonstrating the device pin-out.

Real Time Clock

For the purpose of Zenith and Azimuth angle calculations, the solar tracker requires an accurate estimation of the current Coordinate Universal Time (UTC). The DS1307 Real Time Clock integrated circuit is used to provide precise timekeeping (Fig. A1-2). The DS1307 requires a 5 volt supply wired to pin 8, and a 32.768 kHz crystal oscillator connected between pins 1 and 2. To utilize the I²C serial interface supported by the Arduino Duemilanove, the Serial Data Line and Serial Clock Line from the DS1307 must be wired to analog input pins 4 and 5 of the Arduino, respectively. The DS1307 chip also uses an external battery that can provide over 10 years of backup power in the event of power failure.

External components such as resistors or capacitors are not required for the supporting circuitry of the DS1307. However, a 32.768kHz crystal oscillator is required to generate a clock signal.

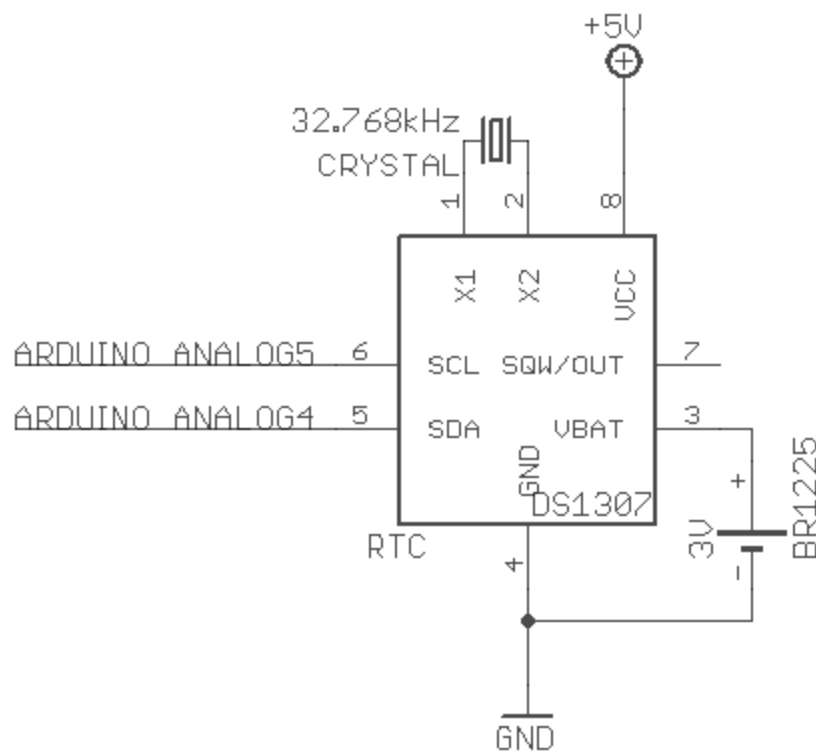


Figure A1-2. The circuit schematic illustrates the Arduino Duemilanove interfaced with the DS1307 Real Time Clock

Stepper Motor Driver

To control the Azimuth and Zenith stepper motors, an MC3479 stepper motor driver (Fig. A1-3) was used.

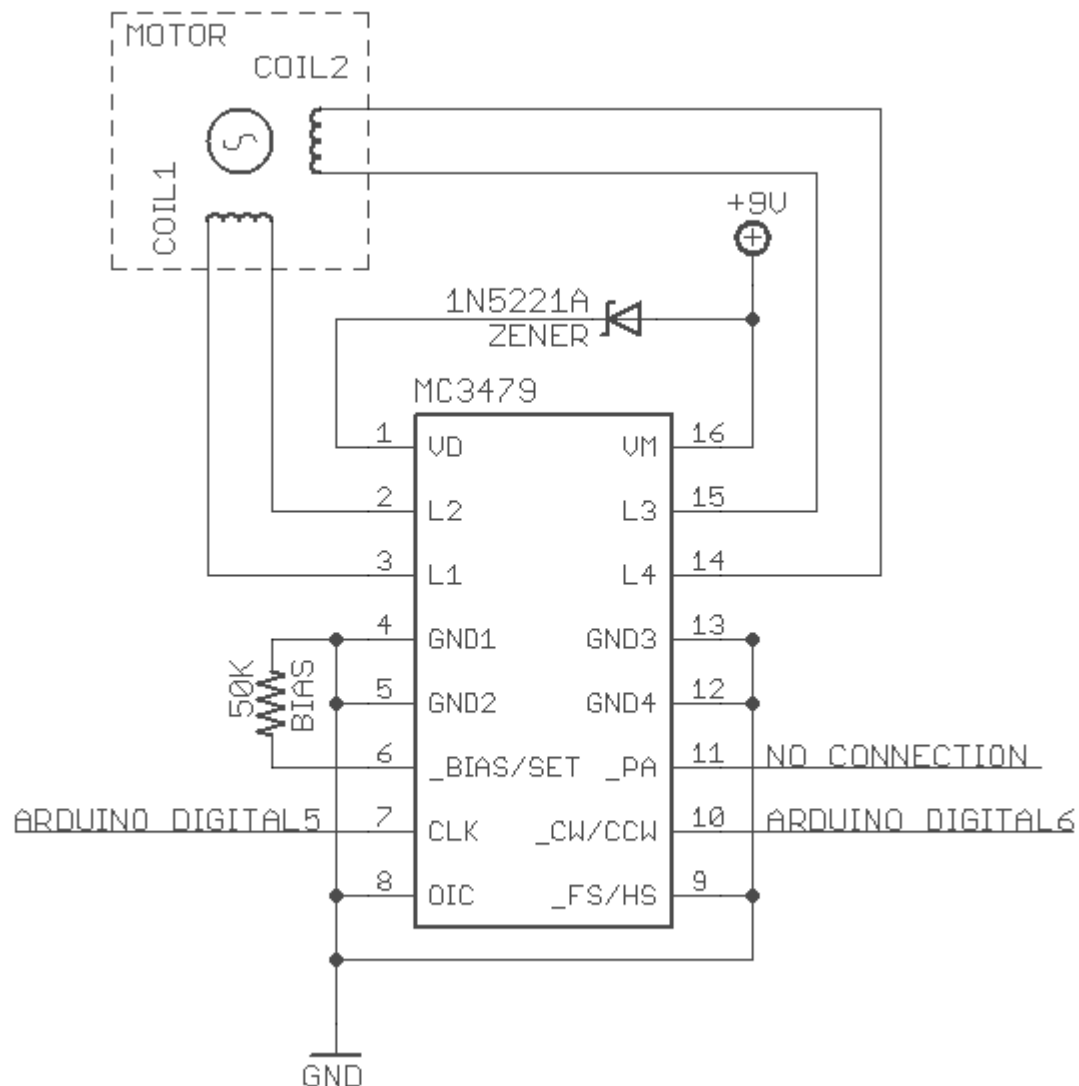


Figure A1-3. The MC3479 Stepper Motor Driver configured for use with the solar tracker design.

The stepper motor driver uses four input pins to configure the operation of a stepper motor. For the purpose of this solar collector design, only two pins from the Arduino controller are required. Digital output pin 6 of the Arduino Duemilanove is wired to CW/CCW of the driver to control the clockwise/counter-clockwise direction of the motor. Arduino digital pin 5 is connected to the CLK pin of the MC3479 to trigger each motor step. MC3479 pins 8 and 9 are

tied to ground to restrict operation to high-impedance and full-stepping mode, respectively. A second driver is used to control the solar Zenith-tracking stage. Arduino digital output pin 7 is connected to CW/CCW and digital output pin 8 is connected to CLK. Control and configuration of the MC3479 driver is summarized in Table A1-1.

Pin	Name	Input	Description
7	CLK	Positive Edge Triggered	The motor takes a step for each rising edge of the input signal.
8	OIC	GND	Output Impedance Control only pertains to half-stepping operation.
9	$\overline{\text{FULL/HALF}}$	GND	A logic low (ground) input signal enables full-stepping.
10	$\overline{\text{CW/CCW}}$	Digital Signal	The logic level of the input signal determines the direction of rotation. The rotor steps clockwise if the input signal is logic low (ground) or counter-clockwise if the input is logic high (+5V).

Table A1-1. Four pins of the MC3479 driver are used to configure the operation of the stepper motor.

The supporting circuitry for each stepper motor driver includes a 1N5221A Zener diode connected between V_m and V_d to prevent damage from voltage spikes during current switching in the motor coils. There also exists 4 ground pins on each MC3479 driver to improve heat dissipation within the integrated circuit from the motor coils. The bias/set resistor, R_b , is wired between Bias (pin 6) and ground. The resistor value was selected to encourage low power operation. By limiting the bias current, I_{BS} , the bias resistor reduces the output (motor) current and thereby reduces power consumption. From the formula for R_{bias} , the value is chosen to be 51.1k Ω .

$$I_{BS} = \frac{I_{OD}}{1000} \times 0.86$$

$$R_b = \frac{V_m - 0.7V}{I_{BS}}$$

Limit Switches

To ensure robust operation, the solar energy collector is fitted with limit switches to prevent a stepper motor from exceeding the angular bounds of the design. A micro roller switch mounted on the base of the solar energy collector prevents multiple revolution windup of the Azimuth-tracking stage. The solar collector also includes two more limit switches on the Zenith-tracking stage to prevent over travel damage to the lead screw mechanism. Each switch is rated up to 2A at 30VDC

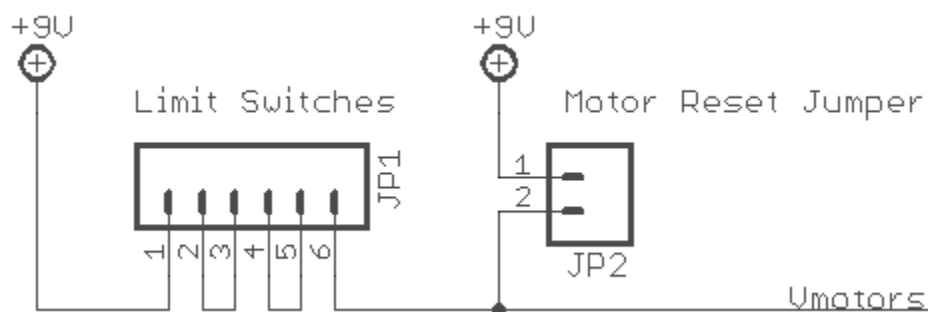


Figure A1-4. Limit Switches contribute to fail-safe operation of the solar collector.

The limit switches are connected in series with the power supply line (Fig. A1-4) and operate completely independent of the Arduino Duemilanove. The switches are organized in this way to immediately cut power to the motor drivers in the event of failure. As soon as each switch is triggered, the circuit is opened and unable to supply power.

After the source of the fault has been corrected, one can continue operation of the solar energy collector. To resume supplying power to the motors, a jumper shunt can be applied momentarily to a pair of header pins marked RESET_MTR on the printed circuit board. A full scale implementation would require relays and additional software to automatically correct the position upon fault correction.

Optical Homing Sensor

Two optical slot sensors are used to detect when the solar energy collector has been oriented in the home position. A homing sensor mounted on the base of the solar tracker is triggered by a #2-56 screw. The screw is designed to pass through the slot sensor as the Azimuth-tracking stage of the collector positions itself for sunrise at 22.5° degrees East of North. A second slot sensor detects when the Zenith-tracking stage is oriented directly towards the horizon (90° from Zenith).

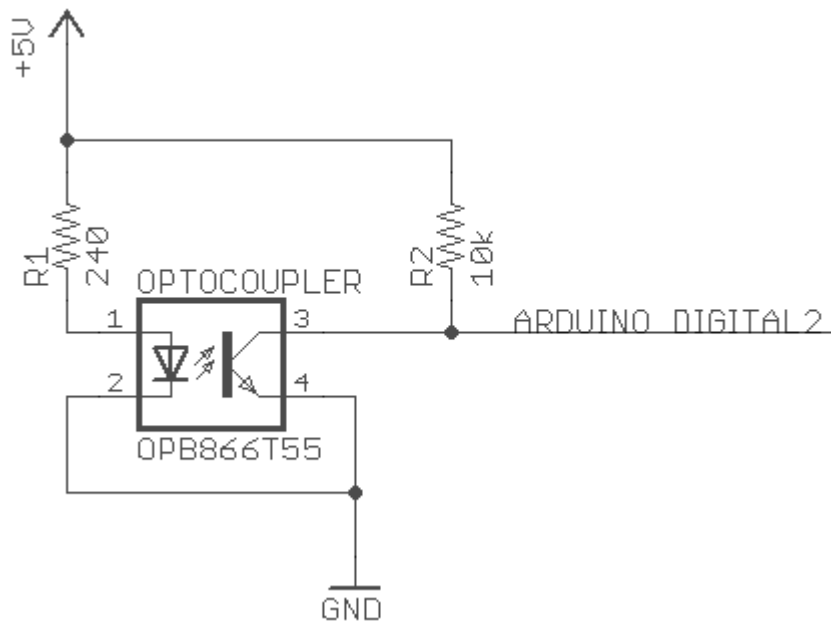


Figure A1-5. An optical slot sensor is used to home the device.

The optical slot sensors are connected to the printed circuit board through a polarized connector. Each homing sensor requires a 5V and ground connection. The sensor outputs are wired to digital input pins 2 and 3 on the Arduino Duemilanove (Fig. A1-5). A 10kΩ pull-up resistor wired to each digital input pin ensures that the sensor output avoids an indeterminate digital logic state. Upon activation, each individual optical slot sensor produces a logical low signal (0V). Otherwise, the sensor outputs a logical high signal (+5V).

Printed Circuit Board

The PCB consists of several core logical components (Fig. A1-6). The Arduino Duemilanove controls the operation of the stepper motor drivers and requires sensor input from the slotted homing sensors and the real time clock.

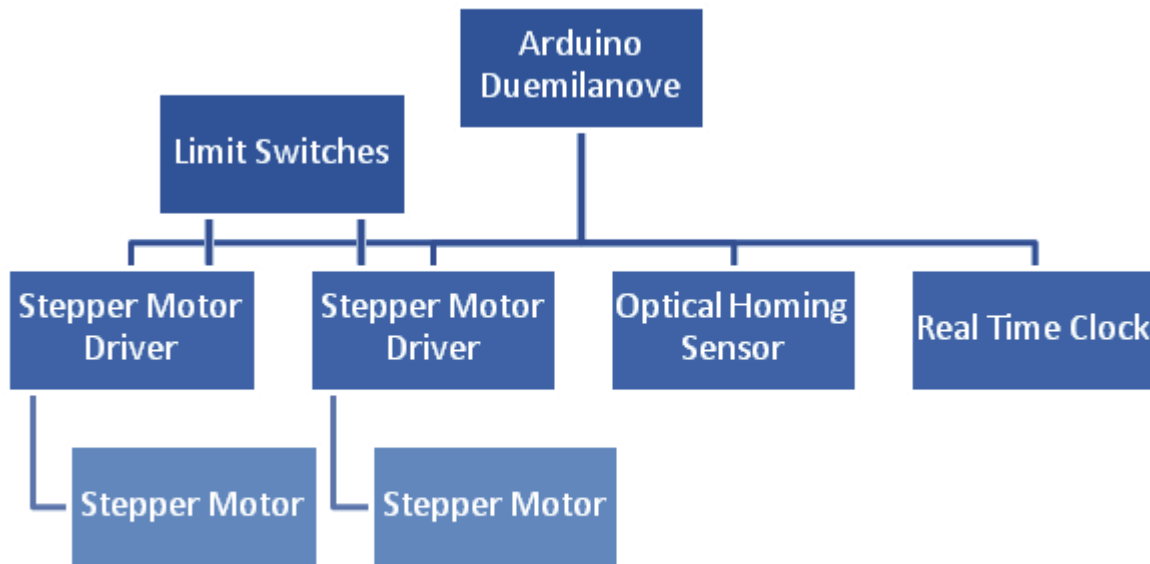


Fig. A1-6. Hierarchy of core solar energy collector components.

Circuit Schematic

First, the circuit schematic diagram was designed and drawn with the CadSoft EAGLE Schematic Editor (Fig. A1-7) [10].

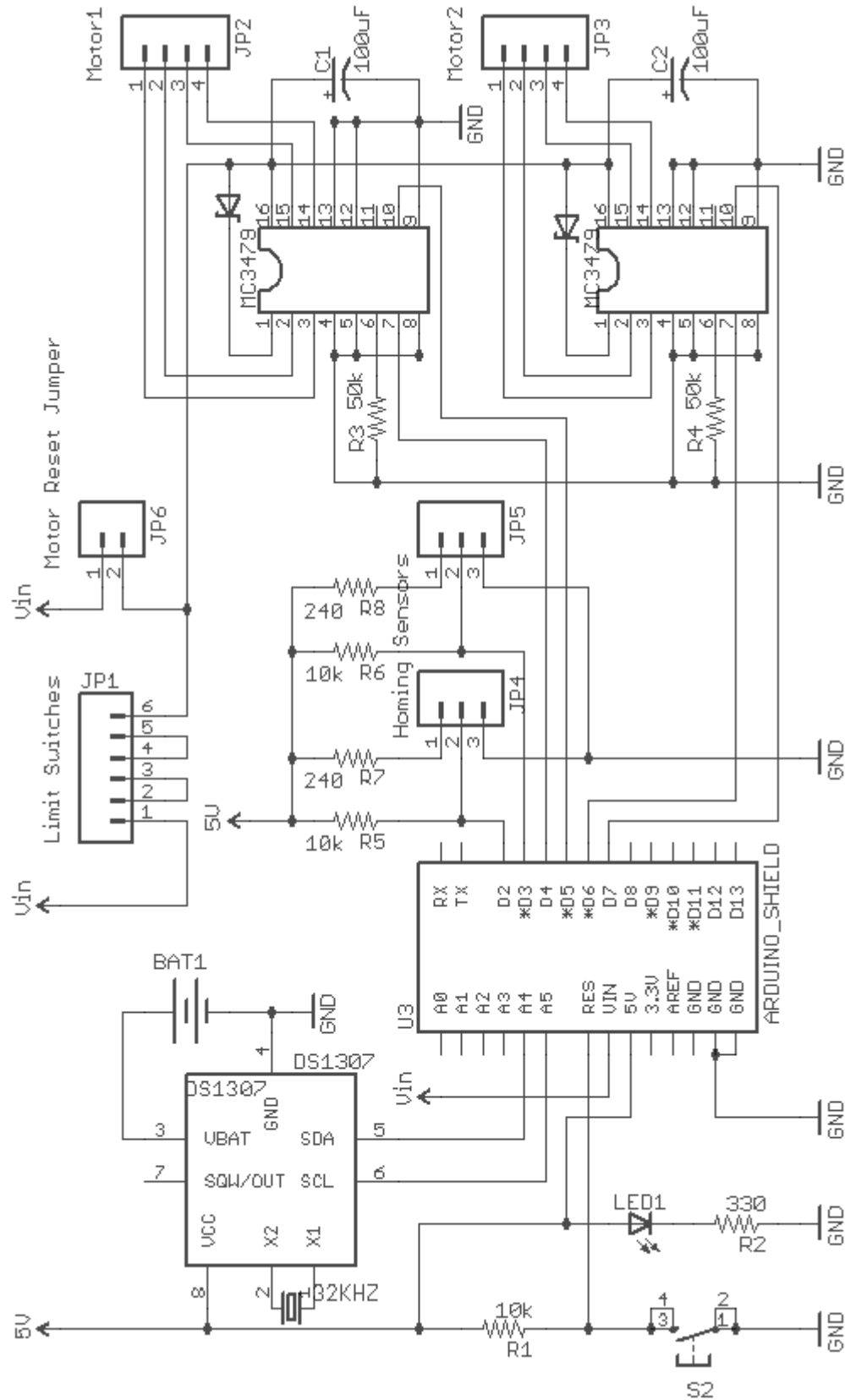


Fig. A1-7. SolarTracker-v4.4 complete circuit schematic.

PCB Prototype

Using a solderless breadboard, a prototype of the final design was constructed and tested using a digital multimeter, and by observing sensors and motor responses (Fig. A1-8).

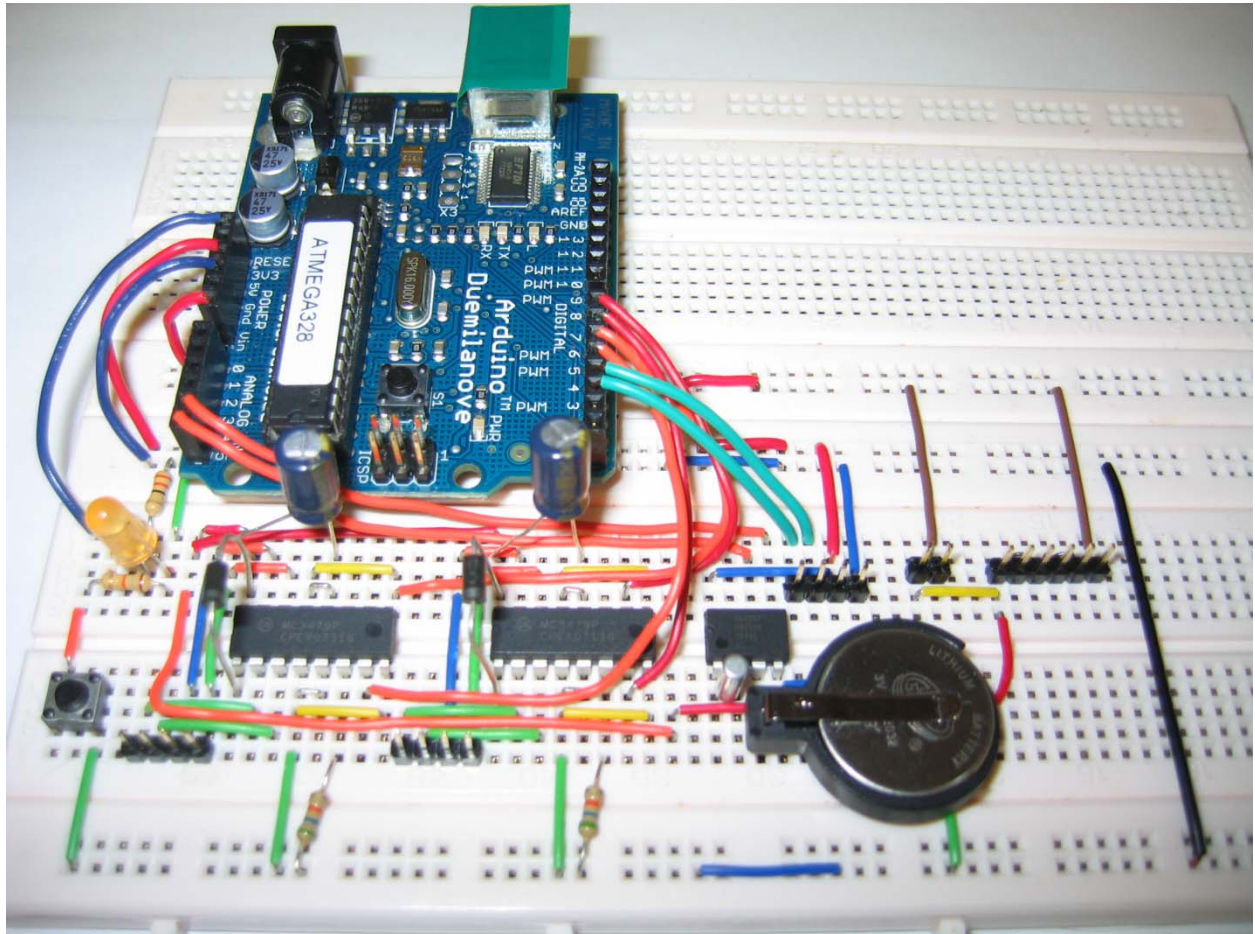


Figure A1-8. Each aspect of the design was tested with the use of a solderless breadboard.

PCB Fabrication

The printed circuit board layout was designed using CadSoft EAGLE Layout Editor. The dimensions of the printed circuit board (PCB) were designed to correspond with the standard Arduino dimensions so that it could be easily mounted above the Arduino Duemilanove (Fig. A1-9). To interface with the Arduino, male header pins from the printed circuit board are matched to mate with female headers on the Arduino Duemilanove. Three 0.125" diameter holes are drilled for the purpose of mounting the PCB.

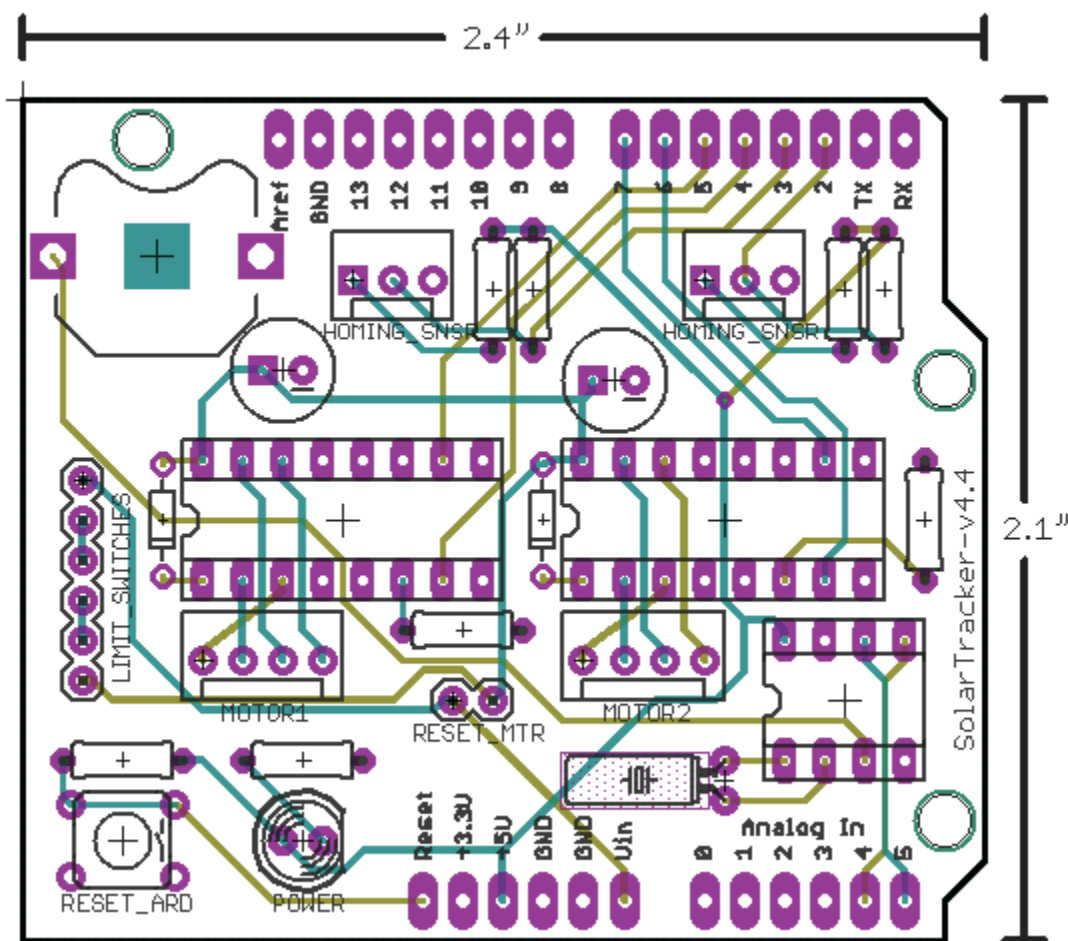


Figure A1-8. The PCB measures 2.4" x 2.1".

The board design consists of a top and bottom copper layer. Extra copper areas are used to reduce noise and ensure all components share a common ground potential. Components are use a through-hole mounting scheme. For the purpose of soldering components to the board, each through-hole on the printed circuit board is drilled 0.019" larger than the diameter of the component lead and plated with copper to provide an electrically conductive surface. Polarized connectors and sockets are used extensively throughout the design to prevent incorrect wiring during installation. As well, connectors, jumpers and other relevant components are labeled on the surface of the printed circuit board by means of silkscreen-printing (Fig. A1-10).

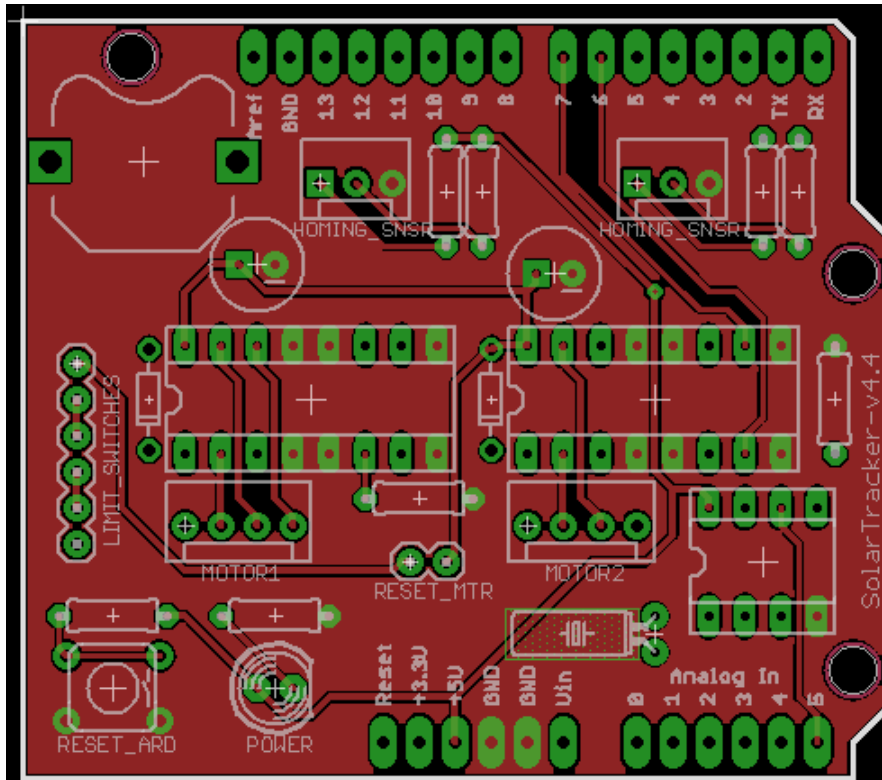


Figure A1-10a. The top layer of the SolarTracker-v4.4.

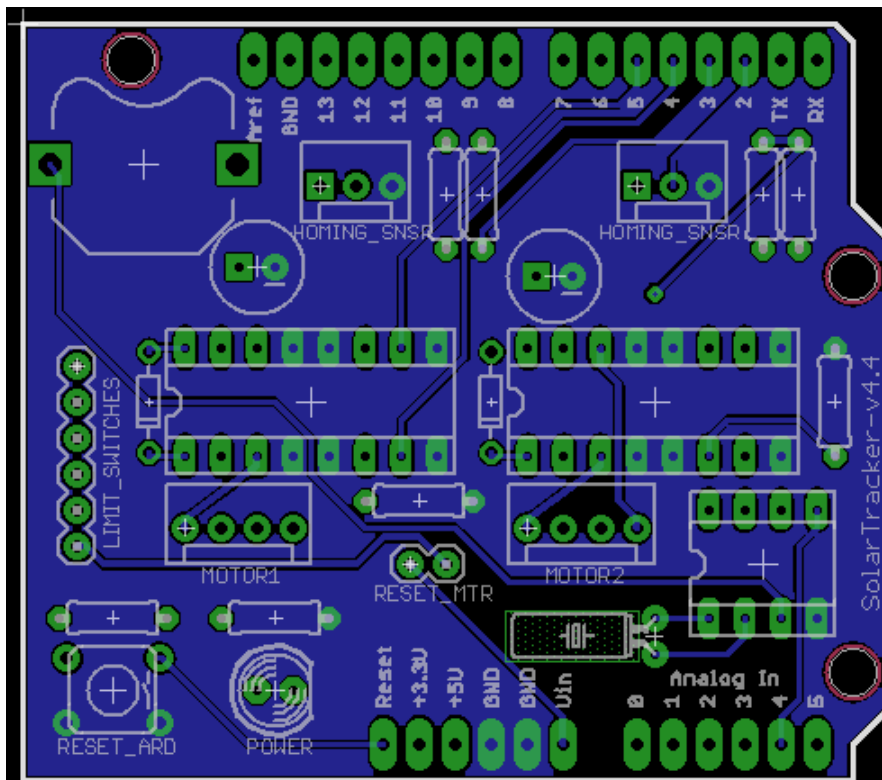


Figure A1-10b. The bottom layer of SolarTracker-v4.4.

The final layout of the printed circuit board was branded SolarTracker-v4.4 (Fig. A1-12). A CAM processor integrated with EAGLE was used to produce an industry standard file format for PCB manufacturing machines, Gerber RS-274X.

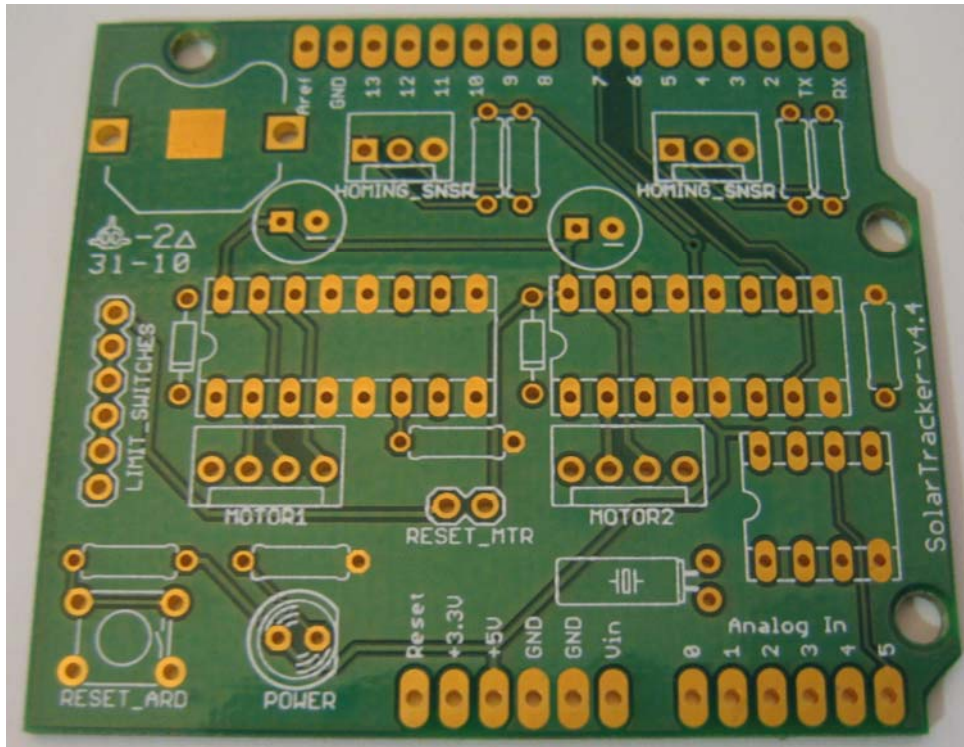


Figure A1-12. SolarTracker-v4.4 fabricated.

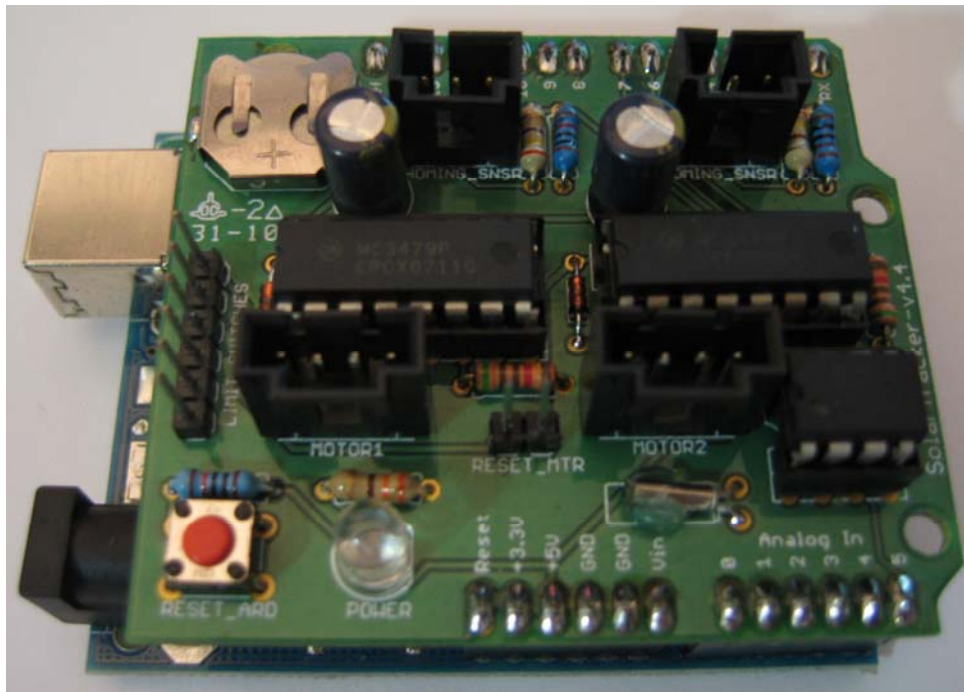


Figure A1-13. The final product of the printed circuit board after assembly.

Bill of Materials

All materials are through-hole mounted with 2.54mm spacing.

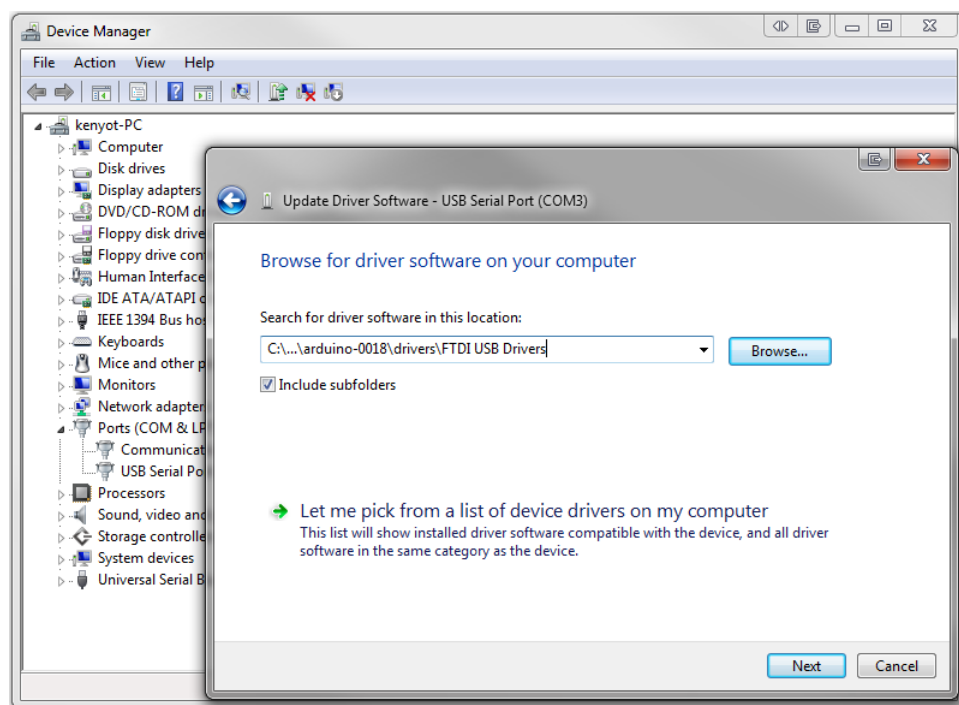
- 1 x SolarTracker-v4.4 Printed Circuit Board
- 2 x MC3479 Stepper Motor Driver
- 2 x DIP Socket Solder Tail - 16 Pin
- 3 x 4 Pin Molex SL Series Header Socket
- 2 x 0.1 μ F Electrolytic Capacitors
- 2 x 1N5221A Zener Diode
- 1 x 330 Ω Resistor
- 2 x 51.1k Ω Resistor
- 1 x 10k Ω Resistor
- 2 x KP35FM2 2-Phase Bipolar Stepper Motor
- 2 x Optical Slot Homing Sensors
- 3 x Normally Closed Limit (roller) Switches
- 4 x 40 Pin Male Header Pins
- 1 x 6 Pin Female Wire to Board connector
- 1 x 2 Pin Jumper Shunt
- 1 x DIP Socket Solder Tail - 8 Pin
- 1 x DS1307 Real Time Clock IC
- 1 x 32.768kHz Crystal Oscillator - 12.5pF Internal Capacitance
- 1 x 3V 48mAh 12mm BR1225 Lithium Coin Cell Battery
- 1 x 12mm Coin Cell Battery Holder

Appendix A2 – Arduino Software Tutorial

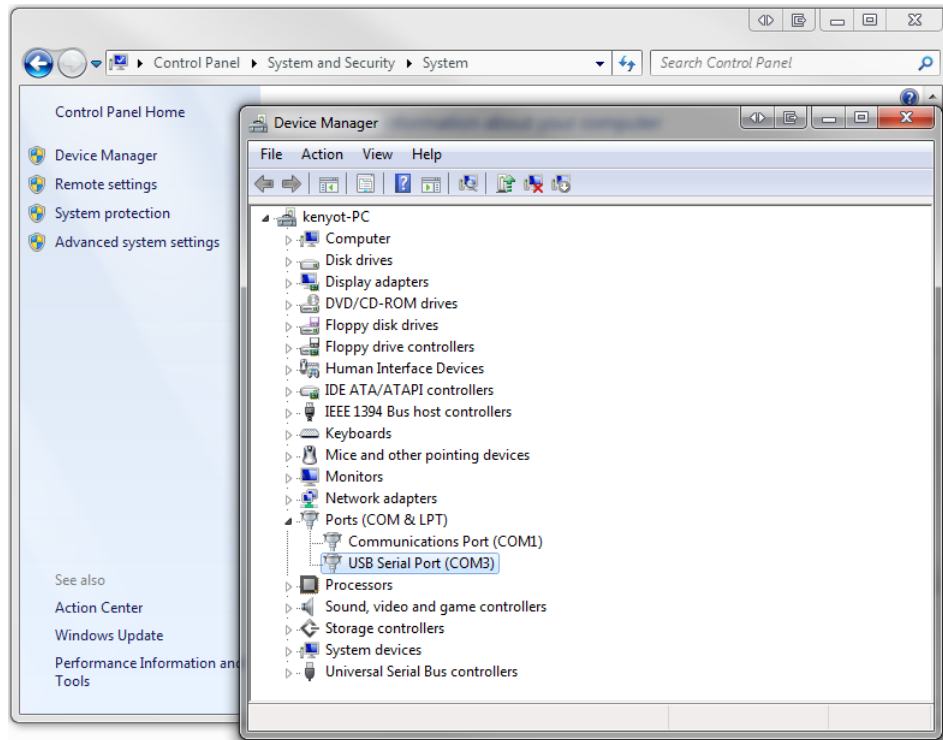
Arduino Installation and Test Procedure



To setup and familiarize yourself with the Arduino Duemilanove, you can try blinking the on-board LED (connected to pin 13) on and off. The steps below outline the procedure:

1. Download the Arduino integrated development environment (IDE) from the main Arduino website <http://arduino.cc/en/Main/Software>. Arduino provides distributions for Windows, Mac OS X, and Linux. For this course, the installation on the JHE219A computers (Windows XP) is officially supported. Instructions below can be used if you wish to test with your personal computer.
2. Connect the Arduino Duemilanove to your computer with a standard USB cable.
3. Install the necessary drivers for USB serial conversion. If the drivers are not automatically installed, follow the directions of the wizard to search for USB serial converter drivers. FTDI USB Drivers can be found in the “drivers” folder of the Arduino distribution.



4. Confirm that a USB Serial Port has been successfully installed. In Windows, open the *Device Manager* (Control Panel > System > Hardware > Device Manager) and note the name of the serial port used (e.g. COM3).



5. Launch the Arduino application.
6. Change the settings to work with your configuration. Select the serial port to be programmed (Tools > Serial Port > *Your Port*). Next, ensure that the Arduino environment is configured to program the Arduino Duemilanove (Tools > Board > Arduino Duemilanove or Nano w/ ATmega 328). Refer to Appendix A1 for more detail regarding the Arduino Duemilanove.
7. Open the Blink example (File > Examples > Digital > Blink).
8. Click  to compile the code.
9. Click  to upload the software to the Arduino board. Execution will automatically start a few seconds after the software has finished uploading.

Arduino Programming Tutorial

Software is written using the C/C++ based Arduino programming language. Each software program written using the Arduino platform is called a sketch. Sketches do not include a `main()` function but instead use functions `setup()` and `loop()` working in conjunction to perform similar functionality. The `setup()` function will run only once at the beginning of execution. It may be used to initialize serial port settings or configure input/output ports, for example. Any local variables defined within `setup()` cannot be used in `loop()`. The `loop()` function will execute repeatedly until the Arduino is reset.

The Arduino platform provides its own Integrated Development Environment (IDE) to develop software, upload programs, and communicate with Arduino hardware (Fig. A2-1). The development environment contains a text editor for writing code, a console for error messages, and a toolbar (Table A2-1). As well, there are a number of project templates and other features that can be accessed from the drop down menus.

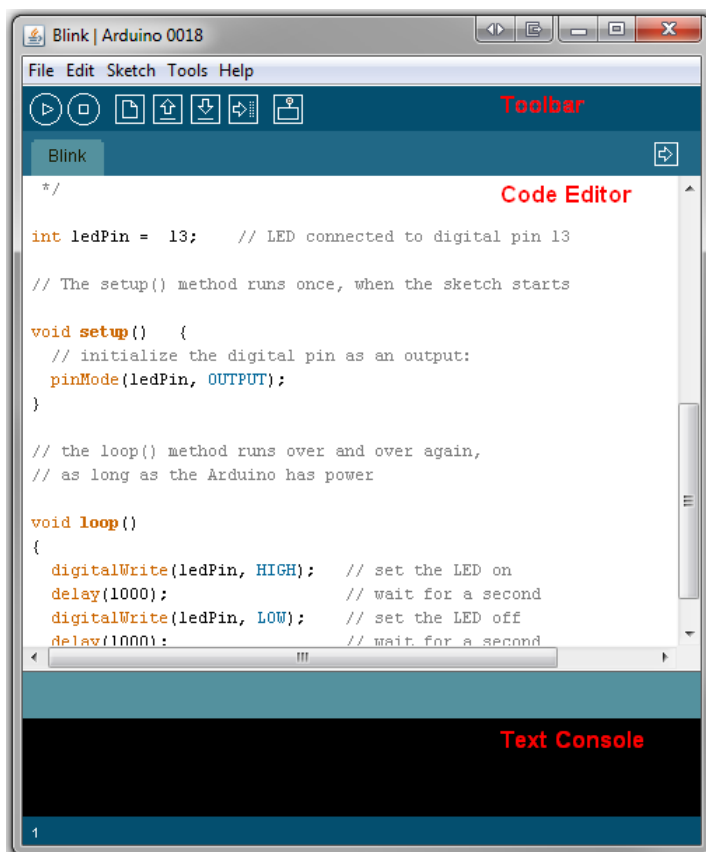


Figure A2-1.








	<i>Verify/Compile</i> - Checks software for errors and compiles code.
	<i>Stop</i> - Stops the serial monitor.
	<i>New</i> - Creates a new sketch.
	<i>Open</i> - Presents a menu of all the sketches in your sketchbook. Clicking one will open it within the current window.
	<i>Save</i> - Saves your sketch.
	<i>Upload to I/O Board</i> - Compiles your code and uploads it to the Arduino I/O board.
	<i>Serial Monitor</i> - Opens the serial monitor.

Table A2-1.

The *Serial Monitor* records all activity over the serial port and displays it onscreen. This tool is particularly useful for debugging your software because it can verify all input/output. After opening the *Serial Monitor*, adjust the baud rate so that it corresponds with the baud rate that was selected during initialization of the connection.

A tutorial on the Arduino language and development environment follows below. The tutorial will introduce serial communication with the DS1307 Real Time Clock.

Real Time Clock Communication

The DS1307 is a Real Time Clock with calendar functionality which tracks the second, minute, hour, date, and year in an independent timekeeping register. Setting the individual parameters of the timekeeping registers or retrieving the current time from the Real Time Clock requires communication using Inter-Integrated Circuit (I²C) serial communication protocol.

The I²C communication protocol operates in a master/slave configuration and requires only the SDA (Serial Data) and SCL (Serial Clock) lines for bidirectional data transmission. The Arduino Duemilanove operates as the master device, which supplies the clock line (SCL) and initiates data transfers. The slave DS1307 responds to the master Arduino requests. A 7-bit addressing system is used to differentiate between each individual slave device on the I²C bus. The DS1307 address (assigned by industry standard) is hexadecimal 0x68.

The Arduino Duemilanove hardware natively supports the integrated I²C interface found on the ATmega328 microcontroller. The Arduino architecture assigns SDA and SCL to analog input pins

4 and 5 respectively. Associated 20kΩ pull-up resistors are built into the Arduino hardware, as required for correct I²C operation. The Arduino Wire Library (Wire.h) is used to simplify serial communication with I²C devices.

Setting Parameters

For the DS1307, the contents of each clock and calendar register are encoded with Binary Coded Decimal (BCD). BCD represents each decimal digit with a half-byte (4 bits) binary sequence. Since only two decimal digits are required to represent each unit of time, 1 byte (8 bits) can store each timekeeper register. This facilitates data transmission since the Wire.h library can send and receive individual bytes sequentially. Because a hexadecimal digit is also encoded using a half-byte, we can simply write hexadecimal values to the timekeeper registers instead of worrying about converting to BCD.

The first byte of an I²C transmission includes the slave device address and data direction (7-bits are used for the device address followed by 1-bit to designate the read or write direction). If the direction bit is a “zero,” the master will write to the DS1307. If the direction bit is a “one” the master will read from the DS1307.

To set the date and time for Saturday, March 14, 2022, at 1:59AM UTC, for example:

```
Wire.begin(); // Join I2C Bus
Wire.beginTransmission(0x68); // Slave address byte for DS1307
  Wire.send(0); // Set register pointer
  Wire.send(0x26); // Second 00-59
  Wire.send(0x59); // Minute 00-59
  Wire.send(0x01); // Hour 00-23
  Wire.send(0x07); // Day of week 01-07
  Wire.send(0x14); // Date 01-31
  Wire.send(0x03); // Month 01-12
  Wire.send(0x22); // Year 00-99
  Wire.send(0x10); /* Control register defines square wave
                    operation on pin 7 */
Wire.endTransmission();
```

Appendix A3 – Software Implementation

```

// The Solar Vector tracking code is from the journal paper
// "Computing the Solar Vector"
// Solar Energy 70(5), 431-441, 2001
// authors M. Blanco-Muriel, D.C. Alarcon-Padilla, T. Lopez-Moratalla, and M. Lara-Coira

// Implementation of the Newton algorithm in C
// http://deadline.3x.ro/newtoncode.html

// instructor demonstration  Azimuth / Zenith

#include <Wire.h>
#include
#include
#include

#define ASCIIZERO      48          // '0' = 48
#define pi             3.1415926535897932384
#define twoPI          (2.0*pi)
#define rad            (pi/180.0)
#define dEarthRadius   6371.01    // in km
#define dAstroUnit     149597890   // in km

#define OPTOPINAZ      2
#define OPTOPINZE      3
#define MOTZECWCCW     5
#define MOTZECLK       4
#define MOTAZCWCCW     7
#define MOTAZCLK       6

#define MAXAZHOMESTEP  6250
#define MAXZEHOMESTEP 10000       // ZE home max subject to revision
#define STEPSAZHOME    1800       // AZ home position = East = +90 degrees = 1800 steps    ***Update for final version => Home = 22.5 degrees***
#define ANGLEZEHOME    90.0       // ZE home position = Horizon = +90 degrees
#define STEPDLY        5

#define ZENITHHOMENUTPOSITION 24

#define STEPSPERDEGAZ  20.0        // 1.8 deg per step and 36:1 reduction worm gears
#define STEPSPERMMZE   100         // temporary demo

//Latitude and Longitude for McMaster (JHE) = 43.260181 (N), 79.920892 (W). Latitude is considered positive to the North and longitude to the East.
//Use decimal format (Latitude = 43 + 26.0181/60 = 43.434; Longitude = -1 * (79 degrees + 92.0892/60) = -80.535;)
const double MCMMASTERLATITUDE = 43.434;
const double MCMMASTERLONGITUDE = -80.535;

struct cTime {
    int iYear;
    int iMonth;
    int iDay;
    double dHours;
    double dMinutes;
    double dSeconds;
};

struct cLocation {
    double dLongitude;
    double dLatitude;
};

```



```

struct cSunCoordinates {
    double dZenithAngle;
    double dAzimuth;
};

struct cTime utcTime;
struct cLocation utcLocation;
struct cSunCoordinates utcSunCoordinates;

int iErrorAZFlag;           // error flag homing AZ
int iErrorZEFlag;           // error flag homing ZE

int iStepsAZ = STEPSAZHOME;
double dAngleZE = ANGLEZEHOME;
double dZenithNutPosition = ZENITHHOMENUTPOSITION;
int iCurrentZEsteps = 8692;

//*****

void setup()
{
    // setup serial communication
    Serial.begin(9600);
    Wire.begin();
    Serial.println("SolarTracker v4.4");
    Serial.println("Serial Connection initialized");
    Serial.println("");

    pinMode(MOTAZCWCCW,OUTPUT); // AZ motor
    pinMode(MOTAZCLK, OUTPUT);
    pinMode(OPTOPINAZ, INPUT); // opto slot sensor AZ
    digitalWrite(MOTAZCWCCW,HIGH); // always go home CCW = HIGH
    pinMode(MOTZECWCCW,OUTPUT); // EL motor
    pinMode(MOTZECLK, OUTPUT);
    pinMode(OPTOPINZE, INPUT); // opto slot sensor ZE
    digitalWrite(MOTZECWCCW,HIGH); // always go home CCW = HIGH

    utcLocation.dLatitude = MCMasterLATITUDE;
    utcLocation.dLongitude = MCMasterLONGITUDE;

    Serial.println("Location: McMaster University, Hamilton, ON");
    Serial.print("Latitude (Decimal Format): "); Serial.println(utcLocation.dLatitude);
    Serial.print("Longitude (Decimal Format): "); Serial.println(utcLocation.dLongitude);
    Serial.println("");

    // home the AZ stepper by looking for blocked opto slot, when home = East = 90 degrees = 1800 steps
    homeAzimuth();

    // home the Zenith stepper
    homeZenith();
} // end setup()
//*****

void loop() {

```

```

    getCurrentTime();
    beginTracking();

    Serial.println("");
    delay(2000);
} // end loop()
//*****

void getCurrentTime() {

    Wire.beginTransmission(0x68);
    Wire.send(0);          // point to address of the timekeeping registers
    Wire.endTransmission();

    Wire.requestFrom(0x68, 7); // request 7 bytes from DS1307
    utcTime.dSeconds = convertHEX(Wire.receive());
    utcTime.dMinutes = convertHEX(Wire.receive());
    utcTime.dHours = convertHEX(Wire.receive());
    Wire.receive();          // disregard the day of the week
    utcTime.iDay = convertHEX(Wire.receive());
    utcTime.iMonth = convertHEX(Wire.receive());
    utcTime.iYear = 2000 + convertHEX(Wire.receive());

    Serial.println("");
    Serial.println("Universal Coordinate Time");
    Serial.print("Time (Hh:Mm:Ss): ");
    if ((int)utcTime.dHours < 10) Serial.print("0");
    Serial.print((int)utcTime.dHours, DEC);
    Serial.print(":");
    if ((int)utcTime.dMinutes < 10) Serial.print("0");
    Serial.print((int)utcTime.dMinutes, DEC);
    Serial.print(":");
    if (utcTime.dSeconds < 10) Serial.print("0");
    Serial.println((int)utcTime.dSeconds, DEC);

    Serial.print("Date (Dd/Mm/YYYY) ");
    if (utcTime.iDay < 10) Serial.print("0");
    Serial.print(utcTime.iDay, DEC);
    Serial.print("/");
    if (utcTime.iMonth < 10) Serial.print("0");
    Serial.print(utcTime.iMonth, DEC);
    Serial.print("/");
    if (utcTime.iYear < 10) Serial.print("0");
    Serial.println(utcTime.iYear, DEC);
    Serial.println("");

} // end getCurrentTime()
//*****

void homeAzimuth() {

    Serial.println("");
    Serial.println("Homimg the Azimuth-tracking stage to 90 degrees East of North");

    // Home the AZ stepper by looking for blocked opto slot. Home = East = 90 degrees = 1800 steps

```

```

iErrorAZFlag = 0;
int iCount;
int optoStateAZ;

for (iCount=0; iCount// should be home in 180 deg worth of steps
    optoStateAZ = digitalRead(OPTOPINAZ);
    if (optoStateAZ == HIGH) { // HIGH is blocked (home)
        break; // now home
    } // end if

    digitalWrite(MOTAZCLK, HIGH); // STEP 1.8 DEG (with 36 reduction = 0.05 deg)
    delay(STEPDLY);

    digitalWrite(MOTAZCLK, LOW);
    delay(STEPDLY);
} // end for

if (iCount < MAXAZHOMESTEP) {
    // safely home
    iErrorAZFlag = 0;
    iStepsAZ = STEPSAZHOME;
}
else {
    // didn't get home in 270 deg
    iErrorAZFlag = 1;
} // end if
} // end homeAzimuth()
/*****

void homeZenith() {

    Serial.println("Homing the Zenith-tracking stage to +90 degrees (Horizon)");
    Serial.println("");

    // home the Zenith stepper
    iErrorZEFlag = 0;
    int iCount;
    int optoStateZE;

    for (iCount=0; iCount// should be home in 180 deg worth of steps
        optoStateZE = digitalRead(OPTOPINZE);
        if (optoStateZE == HIGH) { // HIGH is blocked (home)
            break; // now home
        } // end if

        digitalWrite(MOTZECLK, HIGH); // STEP 1.8 DEG (document amount ratio)
        delay(STEPDLY);

        digitalWrite(MOTZECLK, LOW);
        delay(STEPDLY);
    } // end for

    if (iCount < MAXZEHOMESTEP) {
        // safely home
        iErrorZEFlag = 0;
        dAngleZE = ANGLEZEHOME;
    }
}

```

```

else {
    // didn't get home
    iErrorZEFlag = 1;
} // end if

} // end homeZenith()
// *****

void beginTracking() {

    Serial.println("Solar Tracking Initalized.");
    Serial.println("-----");

    int iDeltaStepsAZ,iDeltaStepsZE;

    GetSunPos(utcTime,utcLocation,&utcSunCoordinates); // get the current solar vector
    Serial.print("Azimuth = "); Serial.println(utcSunCoordinates.dAzimuth);
    Serial.print("Zenith = "); Serial.println(utcSunCoordinates.dZenithAngle);

    Serial.print("Motor AZ= "); Serial.print((double)iStepsAZ/(double)STEPSPERDEGAZ);
    Serial.print(" Current iStepsAZ= "); Serial.print(iStepsAZ);
    iDeltaStepsAZ = (int)(utcSunCoordinates.dAzimuth*STEPSPERDEGAZ) - iStepsAZ;
    Serial.print(" iDeltaStepsAZ= "); Serial.println(iDeltaStepsAZ);
    MoveMotorAZ(iDeltaStepsAZ);
    iStepsAZ = (int)(utcSunCoordinates.dAzimuth*STEPSPERDEGAZ);

    Serial.print("Motor ZE= "); Serial.print(dAngleZE);
    Serial.print(" Current dAngleZE= "); Serial.println(dAngleZE);

    Serial.print("utcSunCoordinates.dZenithAngle = ");
    Serial.println(utcSunCoordinates.dZenithAngle, DEC);

    if (utcSunCoordinates.dZenithAngle > 00.1 && utcSunCoordinates.dZenithAngle < 89.9) {

        int iFutureZEsteps = getZenithSteps(utcSunCoordinates.dZenithAngle);
        int deltaZenithSteps = iCurrentZEsteps - iFutureZEsteps; //store in getZenith result in variable

        MoveMotorZE(deltaZenithSteps);

        dAngleZE = utcSunCoordinates.dZenithAngle;
        iCurrentZEsteps = iFutureZEsteps;

    }
    else {
        Serial.println(" The sun has set - no update");
        homeAzimuth();
        homeZenith();
    } // end if

    Serial.println("-----");
} // end beginTracking()
// *****

void MoveMotorAZ(int iDeltaStepsAZ) {
    int iCount;

    Serial.print("Moving Azimuth motor this many steps: ");
    Serial.println(iDeltaStepsAZ);

```

```

if (iDeltaStepsAZ == 0) {
    return;
} // end if
if (iDeltaStepsAZ > 0) {
    digitalWrite(MOTAZCWCCW,LOW); // positive CW = LOW
}
else {
    iDeltaStepsAZ = -iDeltaStepsAZ;
    digitalWrite(MOTAZCWCCW,HIGH); // negative CCW = HIGH
} // end if
delay(10);

if (iErrorAZFlag == 0) {
    for (iCount=0; iCount<digitalWrite(MOTAZCLK, HIGH); // STEP 1.8 DEG (with 36 reduction = 0.05 deg)
        delay(STEPDLY);
        digitalWrite(MOTAZCLK, LOW);
        delay(STEPDLY);
    } // end for
} // end if
} // end MoveMotorAZ()
//*****
void MoveMotorZE(int iDeltaStepsZE) {
    int iCount;

    Serial.print("Moving Zenith motor this many steps: ");
    Serial.println(iDeltaStepsZE);

    if (iDeltaStepsZE == 0) {
        return;
    } // end if
    if (iDeltaStepsZE > 0) {
        digitalWrite(MOTZECWCCW,HIGH); // positive CW = LOW
    }
    else {
        iDeltaStepsZE = -iDeltaStepsZE;
        digitalWrite(MOTZECWCCW,LOW); // negative CCW = HIGH
    } // end if
    delay(10);

    if (iErrorZEFlag == 0) {
        for (iCount=0; iCount<digitalWrite(MOTZECLK, HIGH); // STEP 1.8 DEG (with 36 reduction = 0.05 deg)
            delay(STEPDLY);
            digitalWrite(MOTZECLK, LOW);
            delay(STEPDLY);
        } // end for
    } // end if
} // end MoveMotorZE()
//*****
void GetSunPos(struct cTime utcTime, struct cLocation utcLocation, struct cSunCoordinates *utcSunCoordinates)
{
    // Main variables
    double dElapsedJulianDays;
    double dDecimalHours;
    double dEclipticLongitude;
    double dEclipticObliquity;
    double dRightAscension;
    double dDeclination;

```

```

// Auxiliary variables
double dY;
double dX;

// Calculate difference in days between the current Julian Day
// and JD 2451545.0, which is noon 1 January 2000 Universal Time
{
    double dJulianDate;
    long int liAux1;
    long int liAux2;
    // Calculate time of the day in UT decimal hours
    dDecimalHours = utcTime.dHours + (utcTime.dMinutes
        + utcTime.dSeconds / 60.0 ) / 60.0;
    // Calculate current Julian Day
    liAux1 =(utcTime.iMonth-14)/12;
    liAux2=(1461*(utcTime.iYear + 4800 + liAux1))/4 + (367*(utcTime.iMonth
        - 2-12*liAux1))/12- (3*((utcTime.iYear + 4900
        + liAux1)/100))/4+utcTime.iDay-32075;
    dJulianDate=(double)(liAux2)-0.5+dDecimalHours/24.0;
    // Calculate difference between current Julian Day and JD 2451545.0
    dElapsedJulianDays = dJulianDate-2451545.0;
}

// Calculate ecliptic coordinates (ecliptic longitude and obliquity of the
// ecliptic in radians but without limiting the angle to be less than 2*Pi
// (i.e., the result may be greater than 2*Pi)
{
    double dMeanLongitude;
    double dMeanAnomaly;
    double dOmega;
    dOmega=2.1429-0.0010394594*dElapsedJulianDays;
    dMeanLongitude = 4.8950630+ 0.017202791698*dElapsedJulianDays; // Radians
    dMeanAnomaly = 6.2400600+ 0.0172019699*dElapsedJulianDays;
    dEclipticLongitude = dMeanLongitude + 0.03341607*sin( dMeanAnomaly )
        + 0.00034894*sin( 2*dMeanAnomaly )-0.0001134
        -0.0000203*sin(dOmega);
    dEclipticObliquity = 0.4090928 - 6.2140e-9*dElapsedJulianDays
        +0.0000396*cos(dOmega);
}

// Calculate celestial coordinates ( right ascension and declination ) in radians
// but without limiting the angle to be less than 2*Pi (i.e., the result may be
// greater than 2*Pi)
{
    double dSin_EclipticLongitude;
    dSin_EclipticLongitude= sin( dEclipticLongitude );
    dY = cos( dEclipticObliquity ) * dSin_EclipticLongitude;
    dX = cos( dEclipticLongitude );
    dRightAscension = atan2( dY,dX );
    if( dRightAscension < 0.0 ) dRightAscension = dRightAscension + twoPI;
    dDeclination = asin( sin( dEclipticObliquity )*dSin_EclipticLongitude );
}

// Calculate local coordinates ( azimuth and zenith angle ) in degrees
{
    double dGreenwichMeanSiderealTime;
    double dLocalMeanSiderealTime;

```

```

    double dLatitudeInRadians;
    double dHourAngle;
    double dCos_Latitude;
    double dSin_Latitude;
    double dCos_HourAngle;
    double dParallax;
    dGreenwichMeanSiderealTime = 6.6974243242 +
        0.0657098283*dElapsedJulianDays
        + dDecimalHours;
    dLocalMeanSiderealTime = (dGreenwichMeanSiderealTime*15
        + utcLocation.dLongitude)*rad;
    dHourAngle = dLocalMeanSiderealTime - dRightAscension;
    dLatitudeInRadians = utcLocation.dLatitude*rad;
    dCos_Latitude = cos( dLatitudeInRadians );
    dSin_Latitude = sin( dLatitudeInRadians );
    dCos_HourAngle= cos( dHourAngle );
    utcSunCoordinates->dZenithAngle = (acos( dCos_Latitude*dCos_HourAngle
        *cos(dDeclination) + sin( dDeclination )*dSin_Latitude));
    dY = -sin( dHourAngle );
    dX = tan( dDeclination )*dCos_Latitude - dSin_Latitude*dCos_HourAngle;
    utcSunCoordinates->dAzimuth = atan2( dY, dX );
    if ( utcSunCoordinates->dAzimuth < 0.0 )
        utcSunCoordinates->dAzimuth = utcSunCoordinates->dAzimuth + twoPI;
    utcSunCoordinates->dAzimuth = utcSunCoordinates->dAzimuth/rad;
    // Parallax Correction
    dParallax=(dEarthRadius/dAstroUnit)
        *sin(utcSunCoordinates->dZenithAngle);
    utcSunCoordinates->dZenithAngle= (utcSunCoordinates->dZenithAngle
        + dParallax)/rad;
}
} // end GetSunPos()

byte convertHEX(byte value) {
    //This works for decimal 0-99
    return ((value/16*10) + (value%16));
} // end convertHEX
//*****

int getZenithSteps(double theta) {
    double beta0;           // Initial guess
    double beta;            // Approximate solution
    double epsilon;         // Maximum error
    int    maxIterations;    // Maximum number of iterations
    int    iterations;      // Actual number of iterations
    int    converged;        // Whether iteration converged

    double thetaRAD = theta * rad;

    beta0 = 0;
    epsilon = 0.00001;
    maxIterations = 10000;

    double nutPosition =-1;

    // Find the first positive solution.
    while (nutPosition <= 0) {

```

```

    beta = newtonsMethod(beta0, epsilon, maxIterations, &iterations, &converged, thetaRAD);
    nutPosition = 25*cos(thetaRAD) + 53*cos(thetaRAD+PI+beta);
    beta0++;
}

if (converged) {
    Serial.print("Newton algorithm converged after this many steps: ");
    Serial.println(iterations, DEC);
    Serial.print("f( ");
    Serial.print(beta, DEC);
    Serial.print(") = ");
    Serial.println(f(thetaRAD, beta), DEC);
}
else {
    Serial.println("Newton's method algorithm didn't converge");
    Serial.print("The final estimate was: ");
    Serial.println(iterations, DEC);
}

// Serial.println("");
// Serial.print("|AD| length (mm): ");
// Serial.println(nutPosition, DEC);
// Serial.println("");

//calculate and return number of steps
// 0.004mm per step
return ((int)(nutPosition / 0.004));
} // end getZenithSteps
/*****

double newtonsMethod(double beta0, double epsilon, int maxIterations,
    int* iterations_p, int* converged_p, double thetaRAD ) {
    double beta = beta0;
    double beta_prev;
    int iter = 0;

    do {
        iter++;
        beta_prev = beta;
        beta = beta_prev - f(thetaRAD, beta_prev)/f_prime(thetaRAD, beta_prev);
    } while (fabs(beta - beta_prev) > epsilon && iter < maxIterations);

    if (fabs(beta - beta_prev) <= epsilon)
        *converged_p = 1;
    else
        *converged_p = 0;
    *iterations_p = iter;

    return beta;
} // end newtonsMethod
/*****

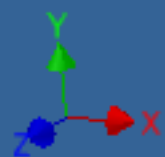
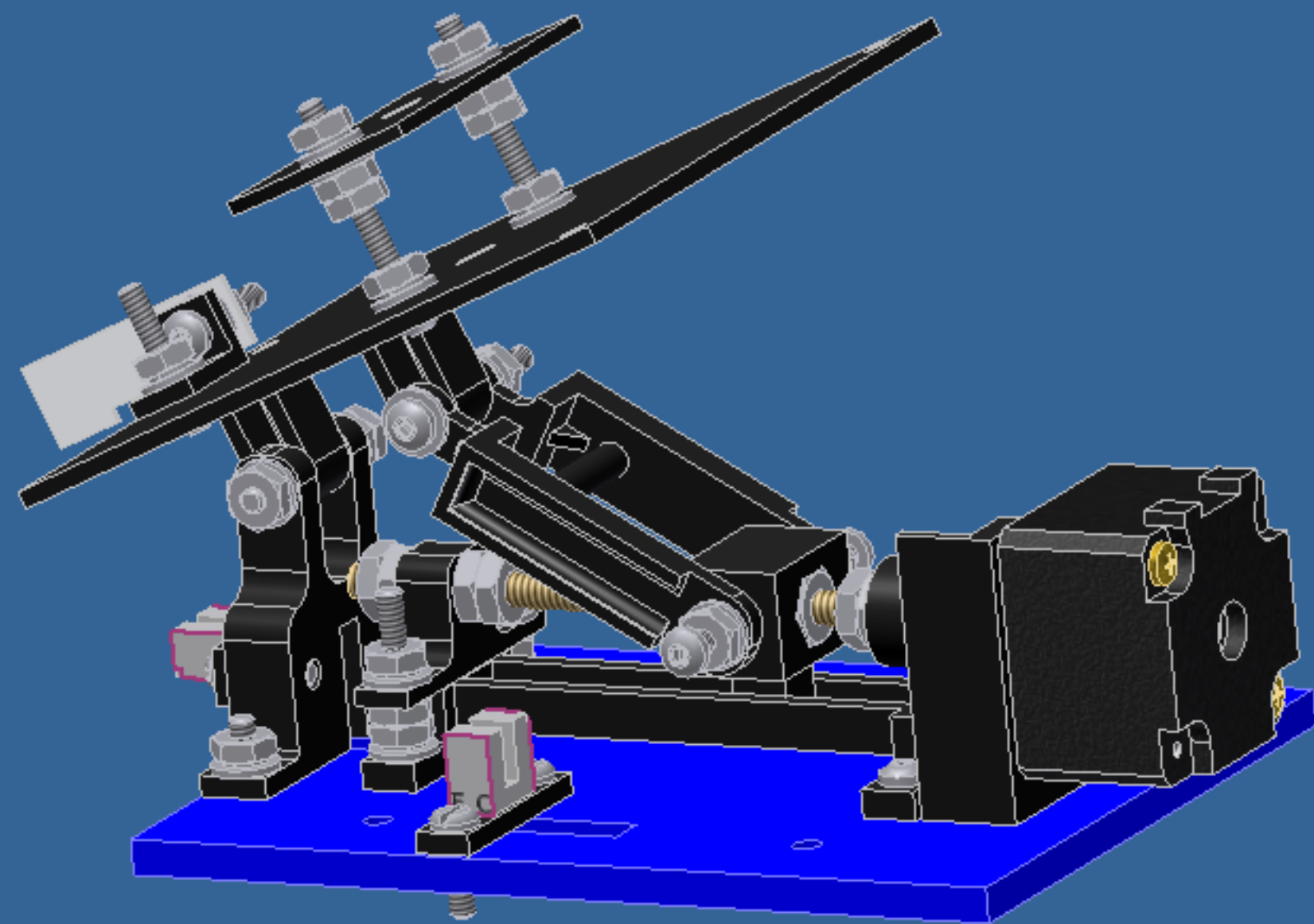
double f(double thetaRAD, double beta) {
    return 15 + 25*sin(thetaRAD) - 53*sin(thetaRAD + beta);
} // end f
/*****

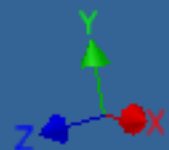
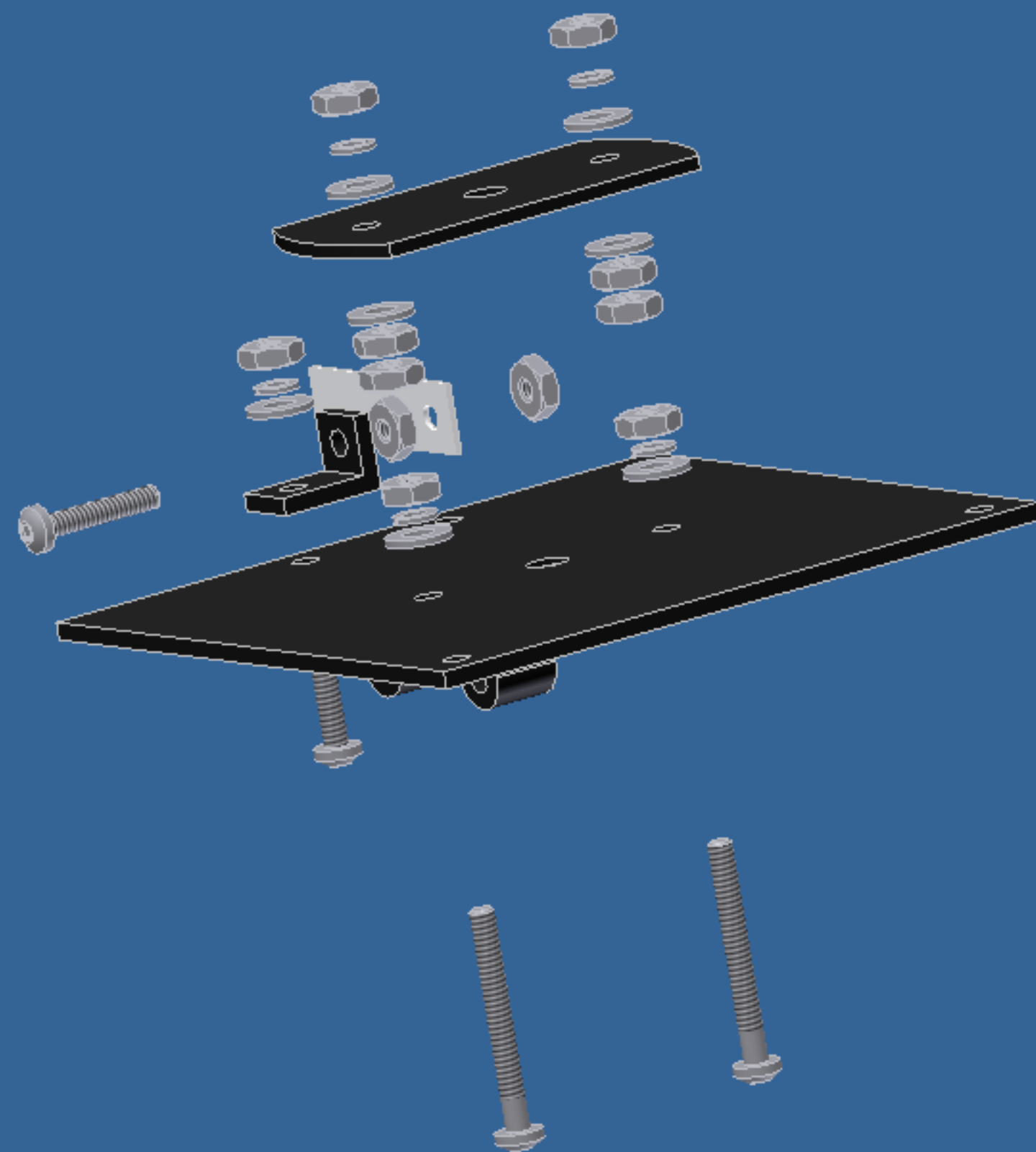
```

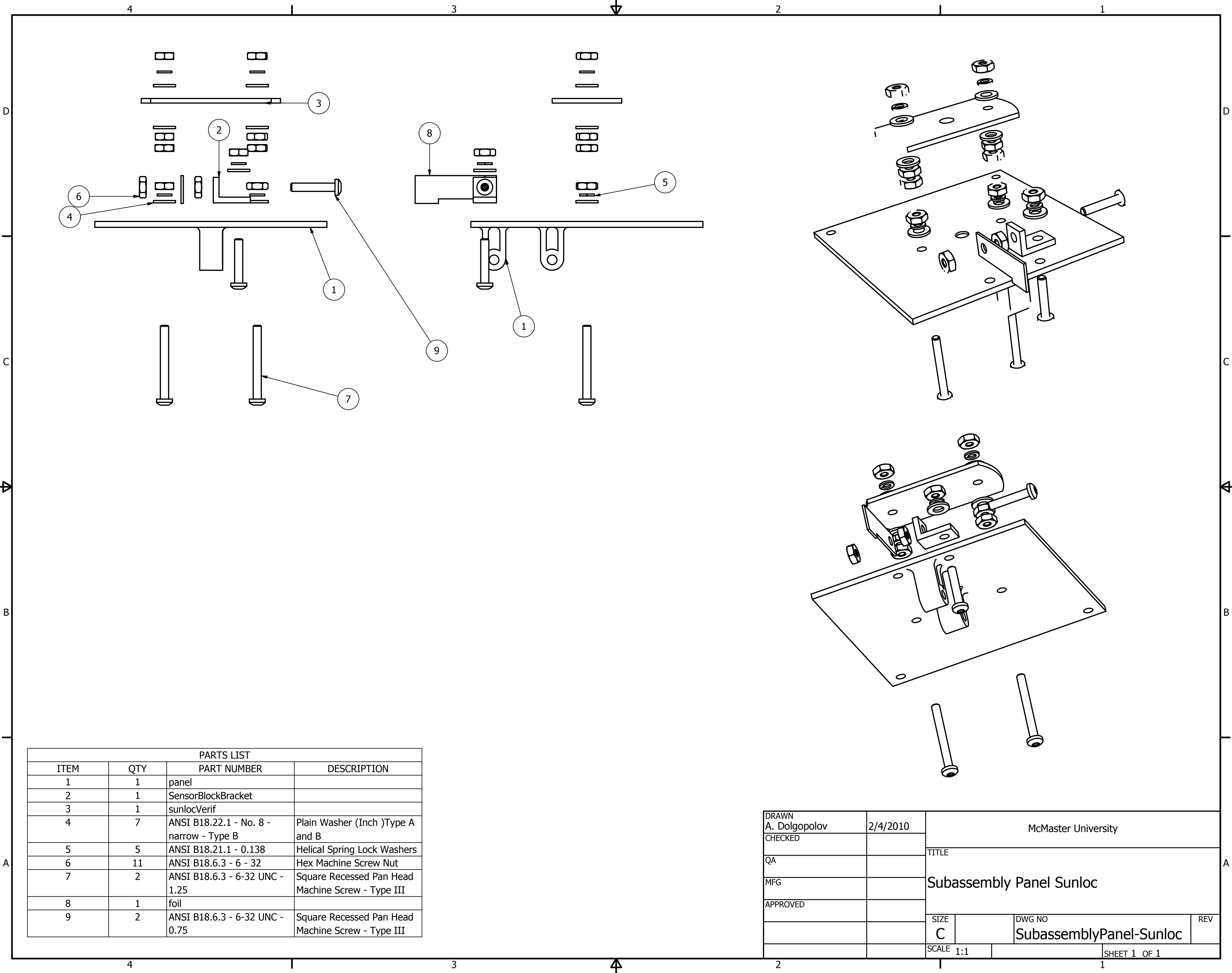


```
double f_prime(double thetaRAD, double beta) {  
    return -53*cos(thetaRAD + beta); //the derivative  
} // end f_prime  
//*****
```

Appendix A4 – CAD Drawings

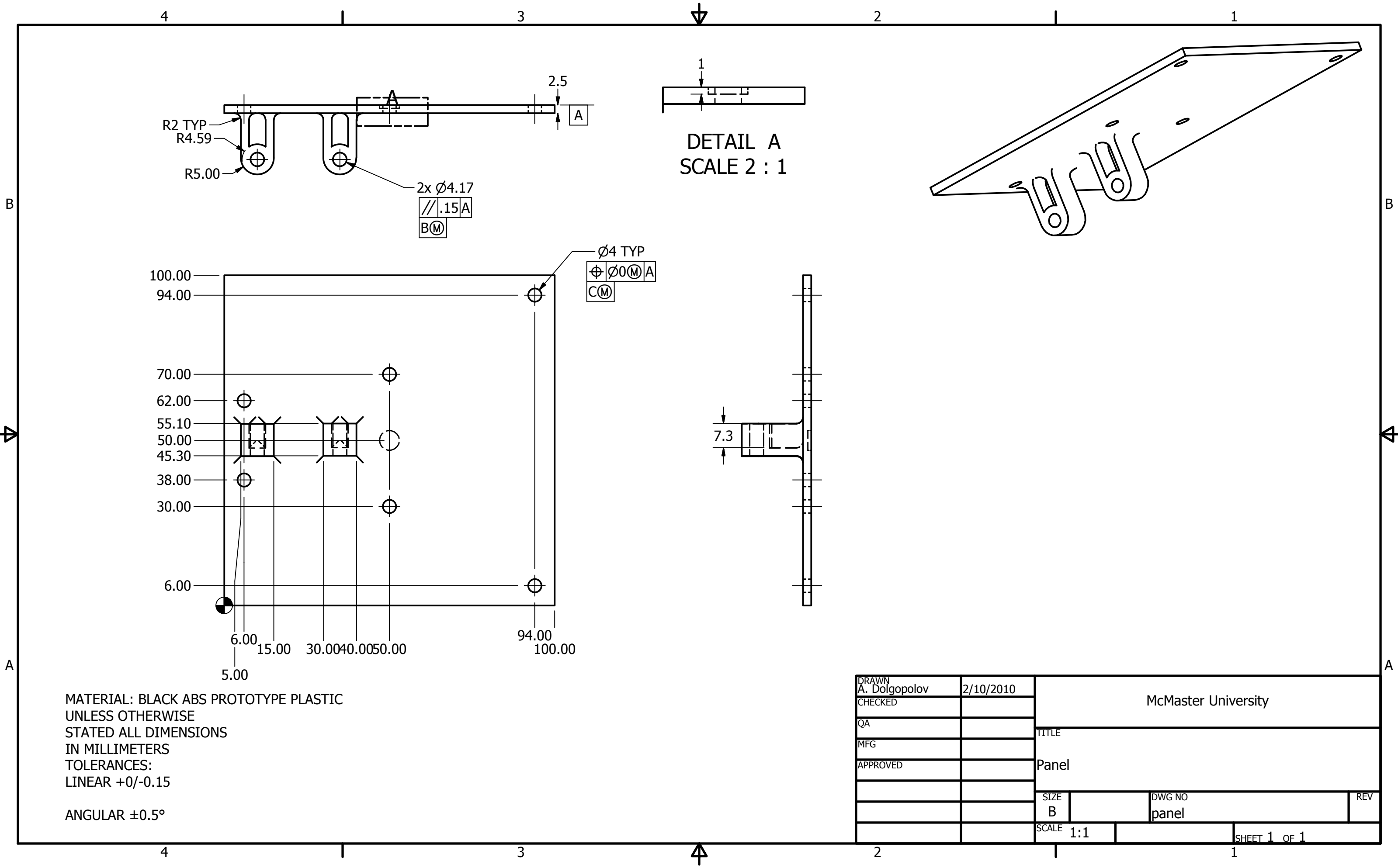


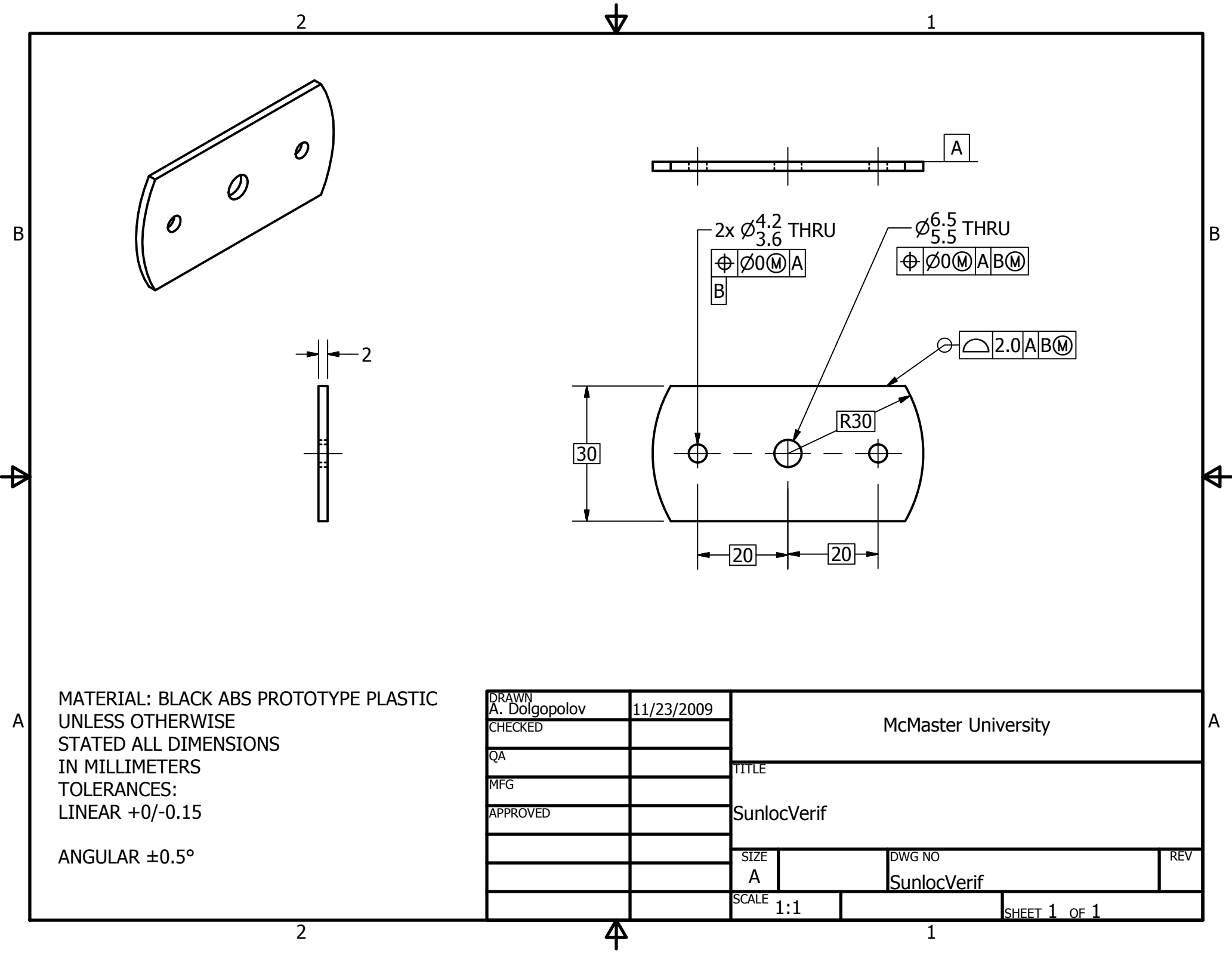




PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	panel	
2	1	SensorBlockBracket	
3	1	sunlocVerif	
4	7	ANSI B18.22.1 - No. 8 - narrow - Type B	Plain Washer (Inch)Type A and B
5	5	ANSI B18.21.1 - 0.138	Helical Spring Lock Washers
6	11	ANSI B18.6.3 - 6 - 32	Hex Machine Screw Nut
7	2	ANSI B18.6.3 - 6-32 UNC - 1.25	Square Recessed Pan Head Machine Screw - Type III
8	1	foil	
9	2	ANSI B18.6.3 - 6-32 UNC - 0.75	Square Recessed Pan Head Machine Screw - Type III

DRAWN A. Dolgoplov	2/4/2010	McMaster University		
CHECKED		TITLE		
QA		Subassembly Panel Sunloc		
MFG				
APPROVED				
		SIZE C	DWG NO SubassemblyPanel-Sunloc	REV
		SCALE 1:1	SHEET 1 OF 1	

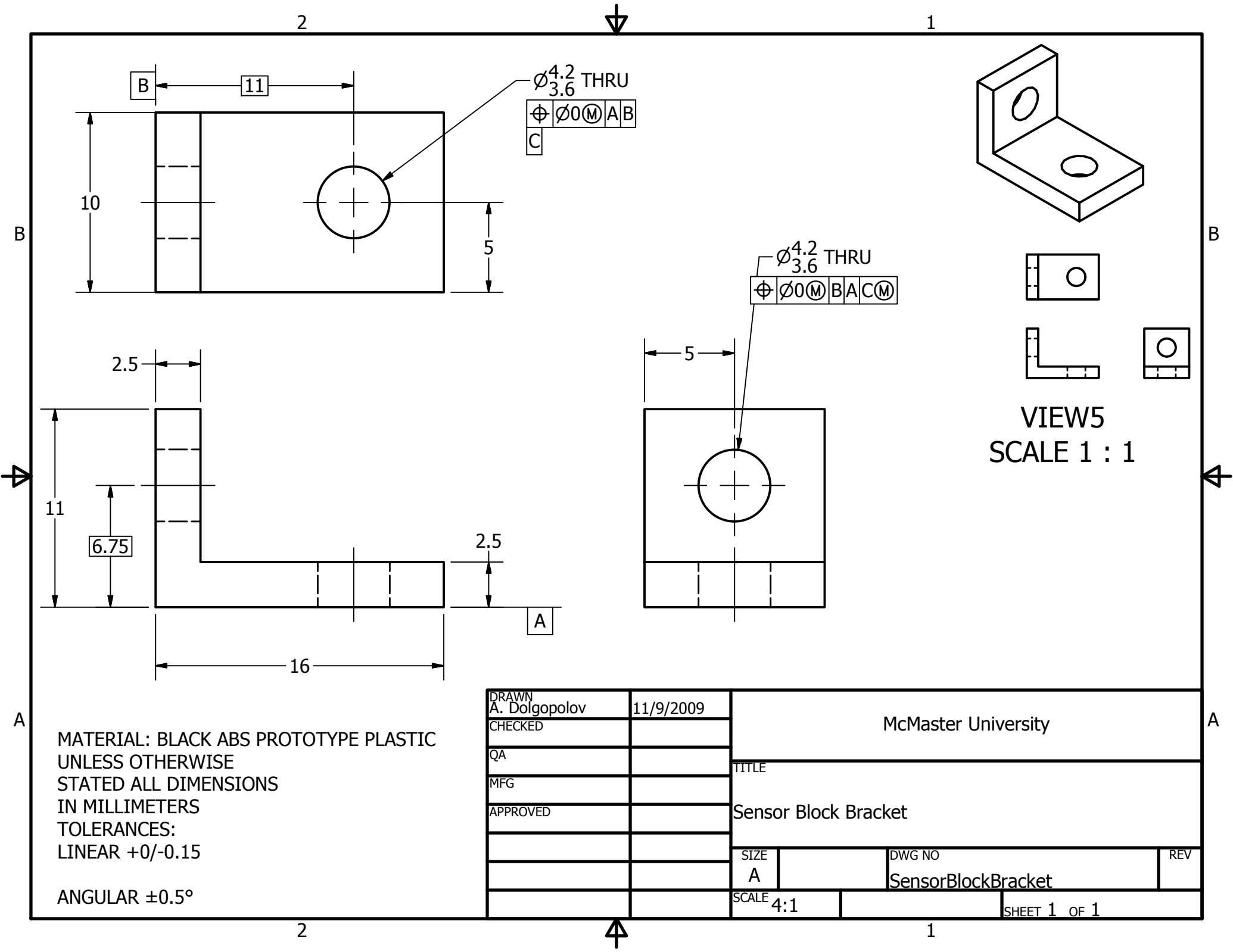




MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR +0/-0.15

ANGULAR $\pm 0.5^\circ$

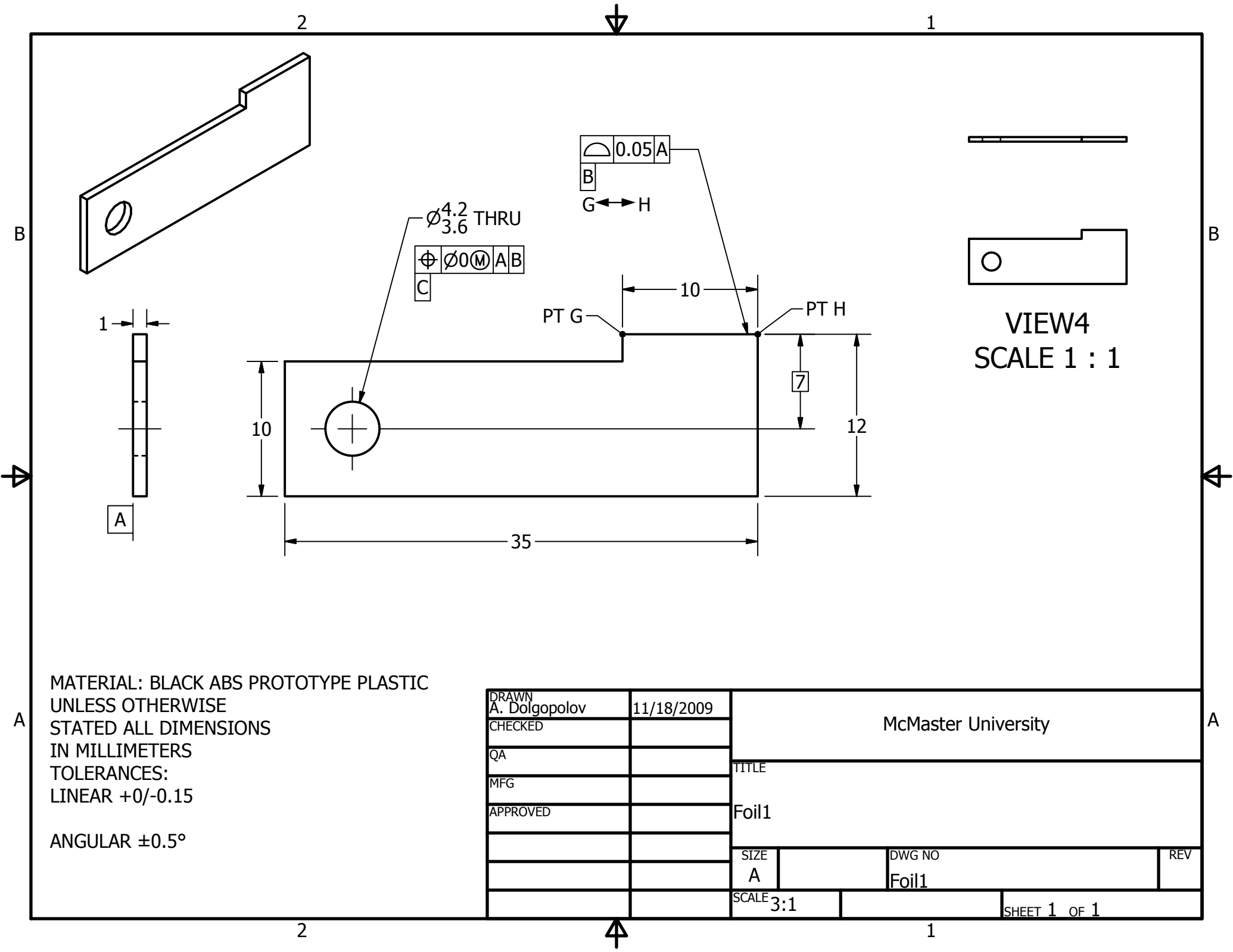
DRAWN A. Dolgoplov	11/23/2009	McMaster University		
CHECKED				
QA		TITLE SunlocVerif		
MFG				
APPROVED		SIZE A		
		SCALE 1:1	DWG NO SunlocVerif	REV



MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR +0/-0.15
ANGULAR ±0.5°

DRAWN	A. Dolgoplov	11/9/2009
CHECKED		
QA		
MFG		
APPROVED		

McMaster University		
TITLE		
Sensor Block Bracket		
SIZE	DWG NO	REV
A	SensorBlockBracket	
SCALE	SHEET 1 OF 1	
4:1		

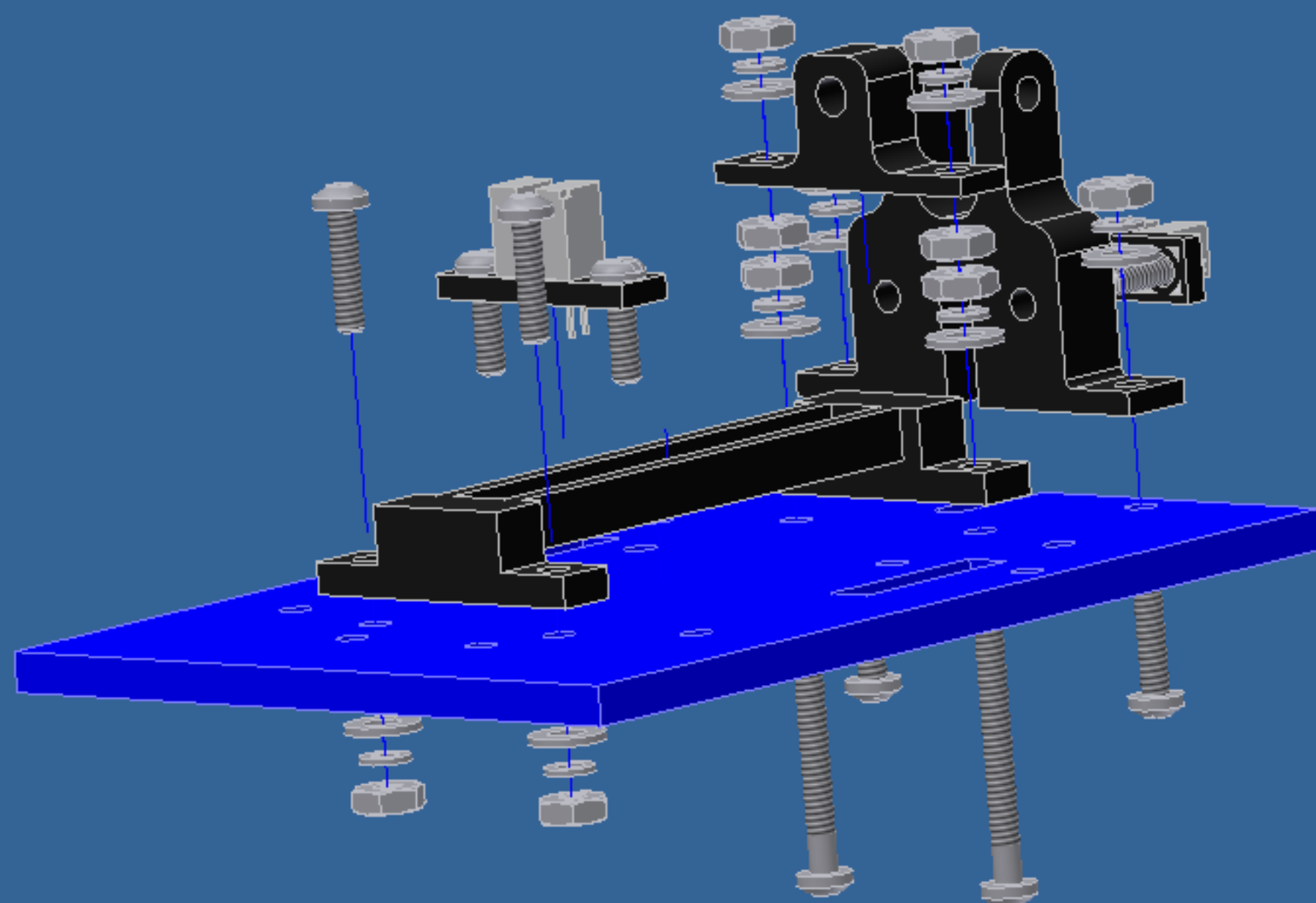


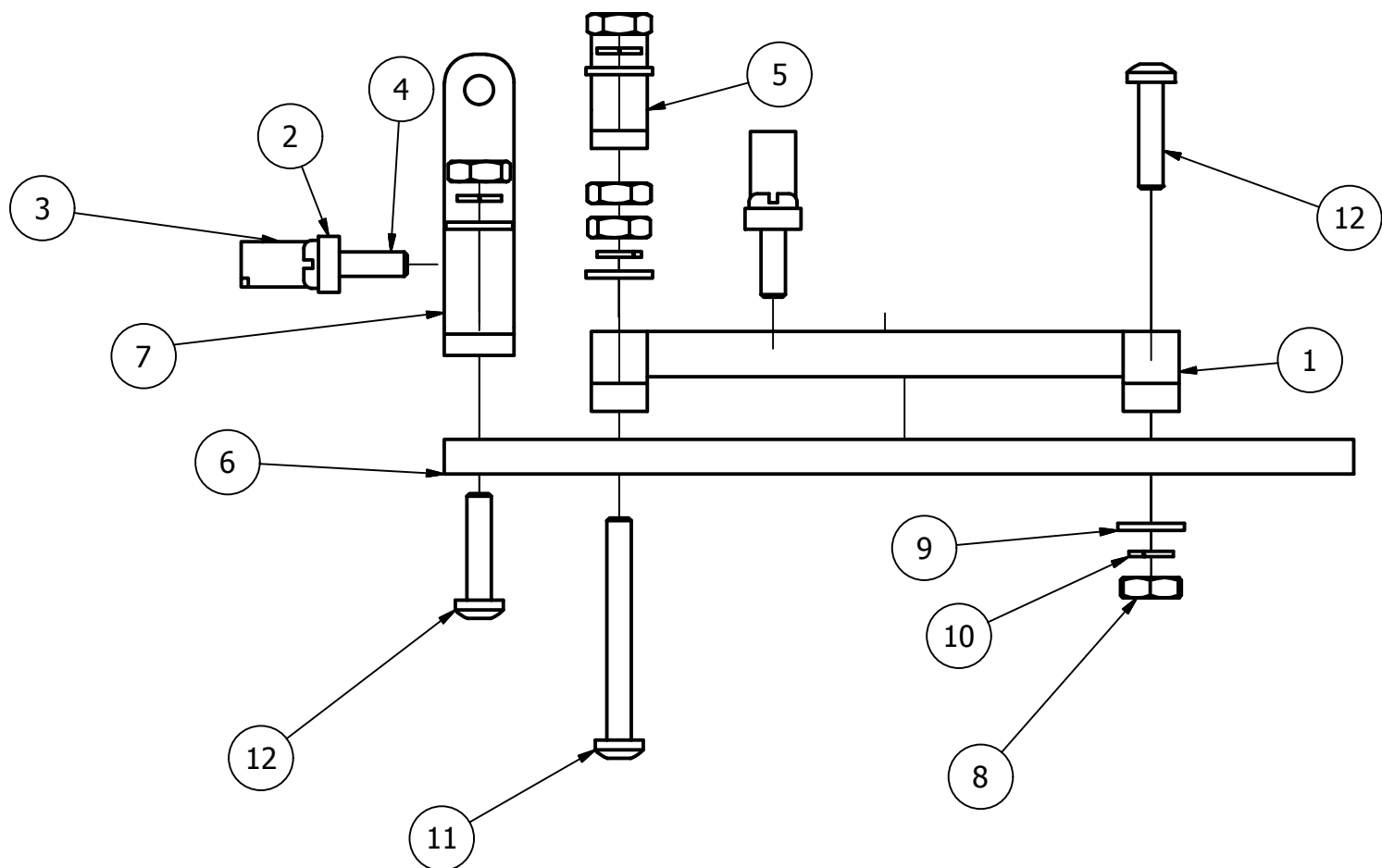
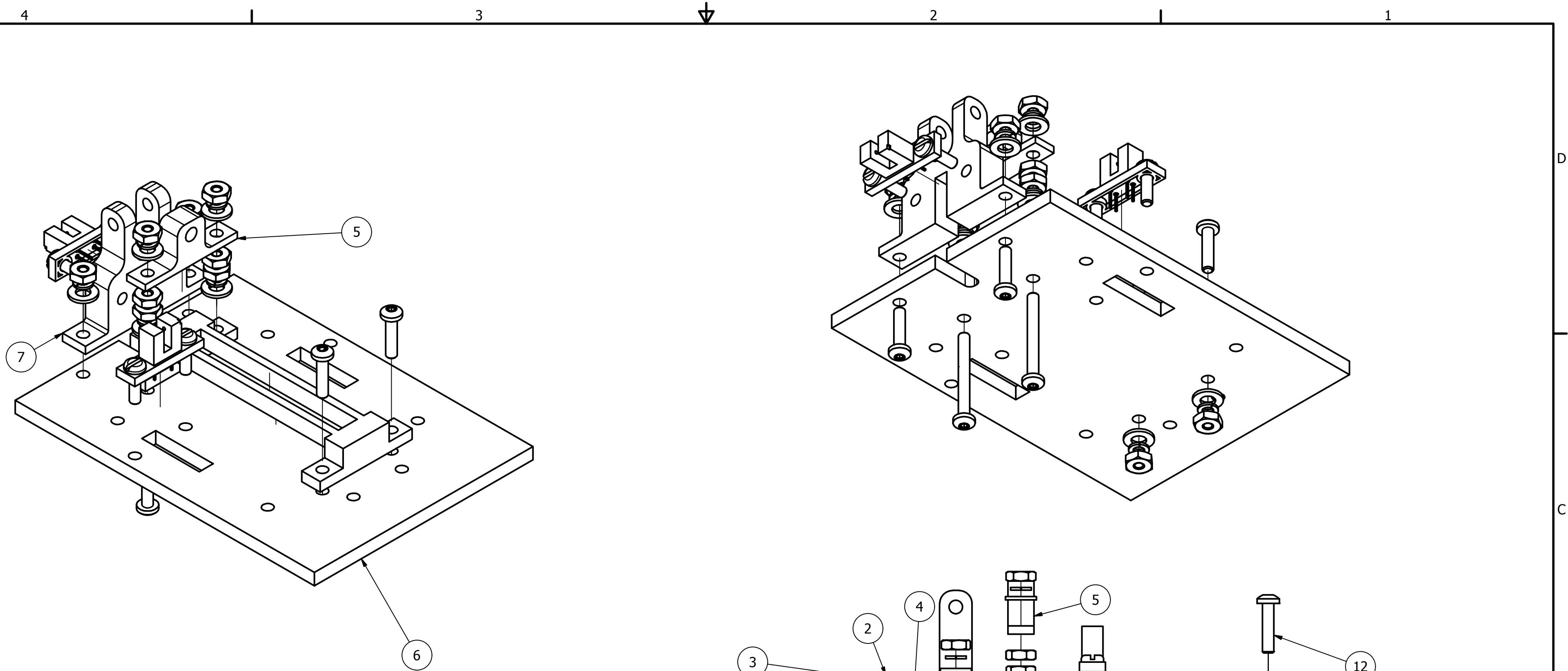
VIEW4
SCALE 1 : 1

MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR +0/-0.15

ANGULAR ±0.5°

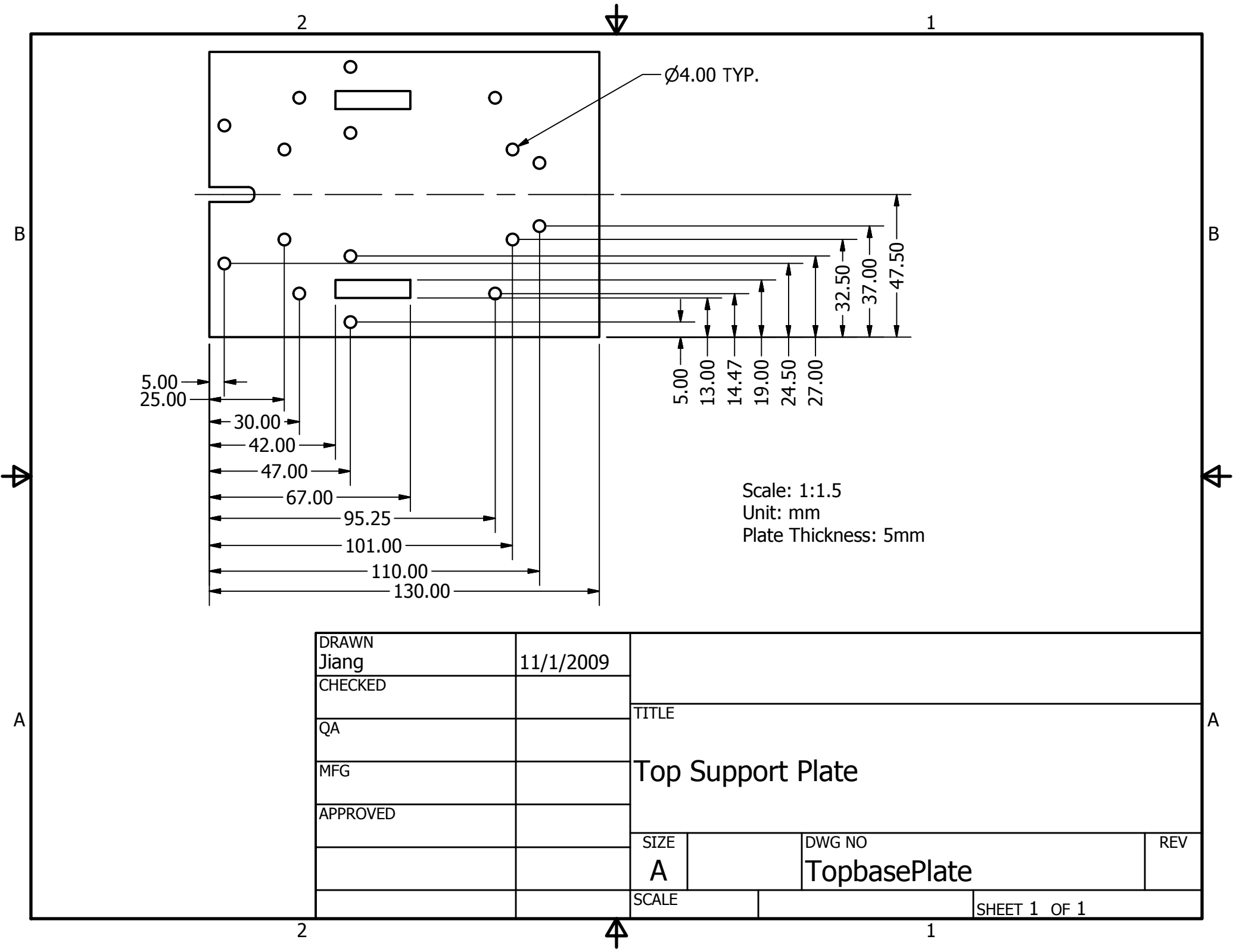
DRAWN A. Dolgoplov	11/18/2009	McMaster University		
CHECKED				
QA		TITLE Foil1		
MFG				
APPROVED		SIZE A	DWG NO Foil1	REV
		SCALE 3:1	SHEET 1 OF 1	



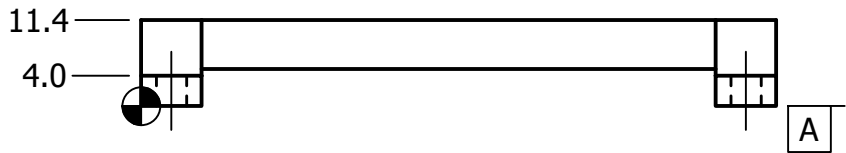
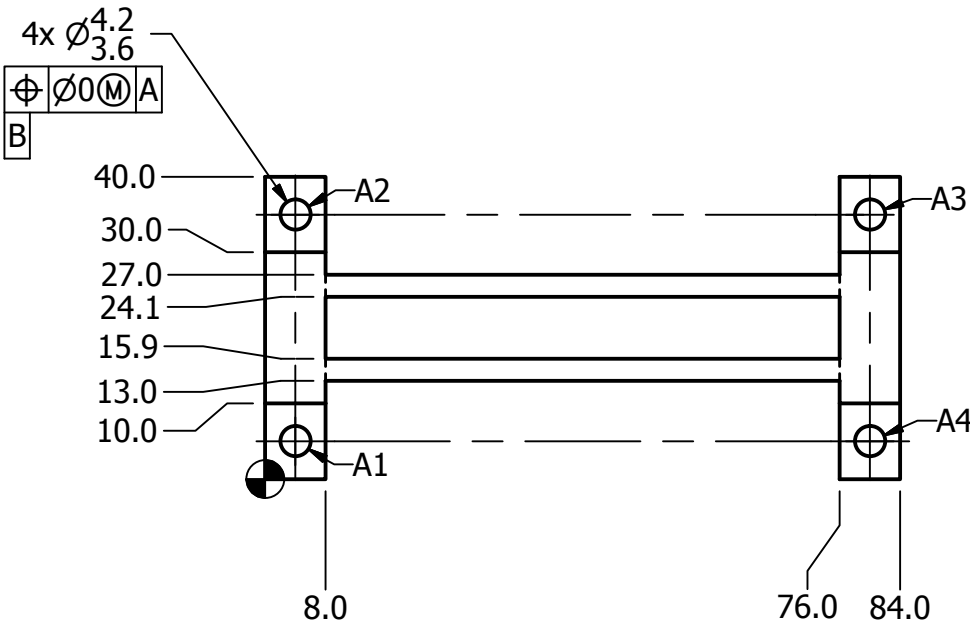
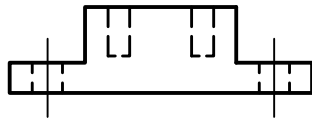
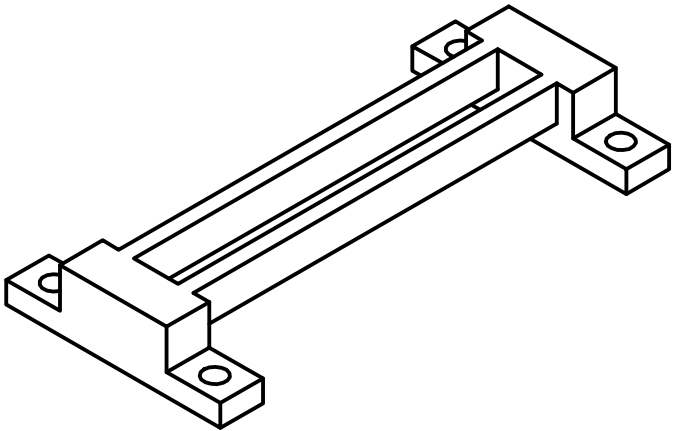


PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	Bridge	
2	2	TIL159Mount	
3	2	TIL159	
4	4	ANSI B18.6.3 - 6 - 32 x 1 1/4 SP HMS	Slotted Pan Head Machine Screw
5	1	screwholder	
6	1	TopbasePlate	
7	1	Pivotversion2	
8	10	ANSI B18.6.3 - 6 - 32	Hex Machine Screw Nut
9	8	ANSI B18.22.1 - No. 8 - narrow - Type B	Plain Washer (Inch)Type A and B
10	8	ANSI B18.21.1 - 0.138	Helical Spring Lock Washers
11	4	ANSI B18.6.3 - 6-32 UNC - 0.6	Square Recessed Pan Head Machine Screw - Type III
12	3	ANSI B18.6.3 - 6-32 UNC - 1.25	Square Recessed Pan Head Machine Screw - Type III

DRAWN A. Dolgoplov	2/4/2010	McMaster University		
CHECKED		TITLE Subassembly Top Plate Bridge Pivot		
QA				
MFG				
APPROVED				
		SIZE C	DWG NO TopPlateBridgePivot	REV
		SCALE 1:1	SHEET 1 OF 1	



DRAWN Jiang	11/1/2009	TITLE Top Support Plate		
CHECKED				
QA		SIZE A		
MFG				
APPROVED		DWG NO TopbasePlate		
		SCALE		
		SHEET 1 OF 1		

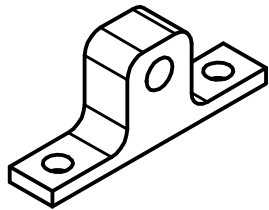


HOLE TABLE		
HOLE	XDIM	YDIM
A1	4.00	5.00
A2	4.00	35.00
A3	80.00	35.00
A4	80.00	5.00

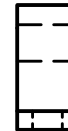
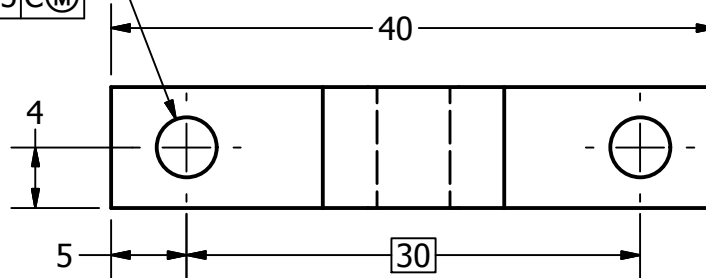
MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR +0/-0.15

ANGULAR $\pm 0.5^\circ$

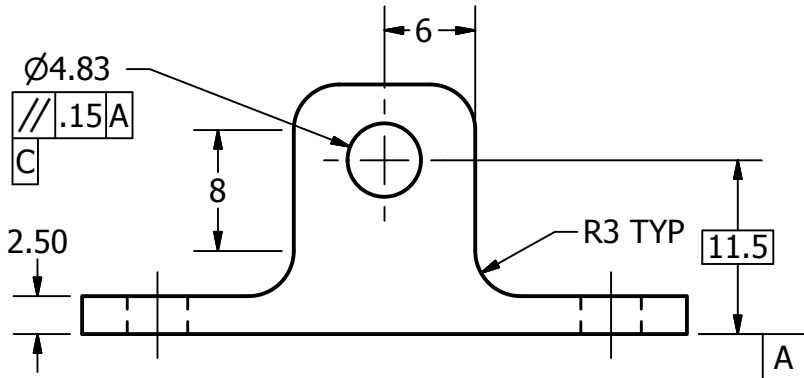
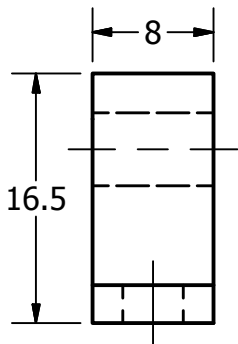
DRAWN A. Dolgoplov	11/23/2009	McMaster University		
CHECKED		TITLE Bridge		
QA				
MFG				
APPROVED				
		SIZE B	DWG NO Bridge	REV
		SCALE 1:1		SHEET 1 OF 1



\varnothing	$\varnothing 0$	\textcircled{M}	A
\perp	.15	C	\textcircled{M}
B			

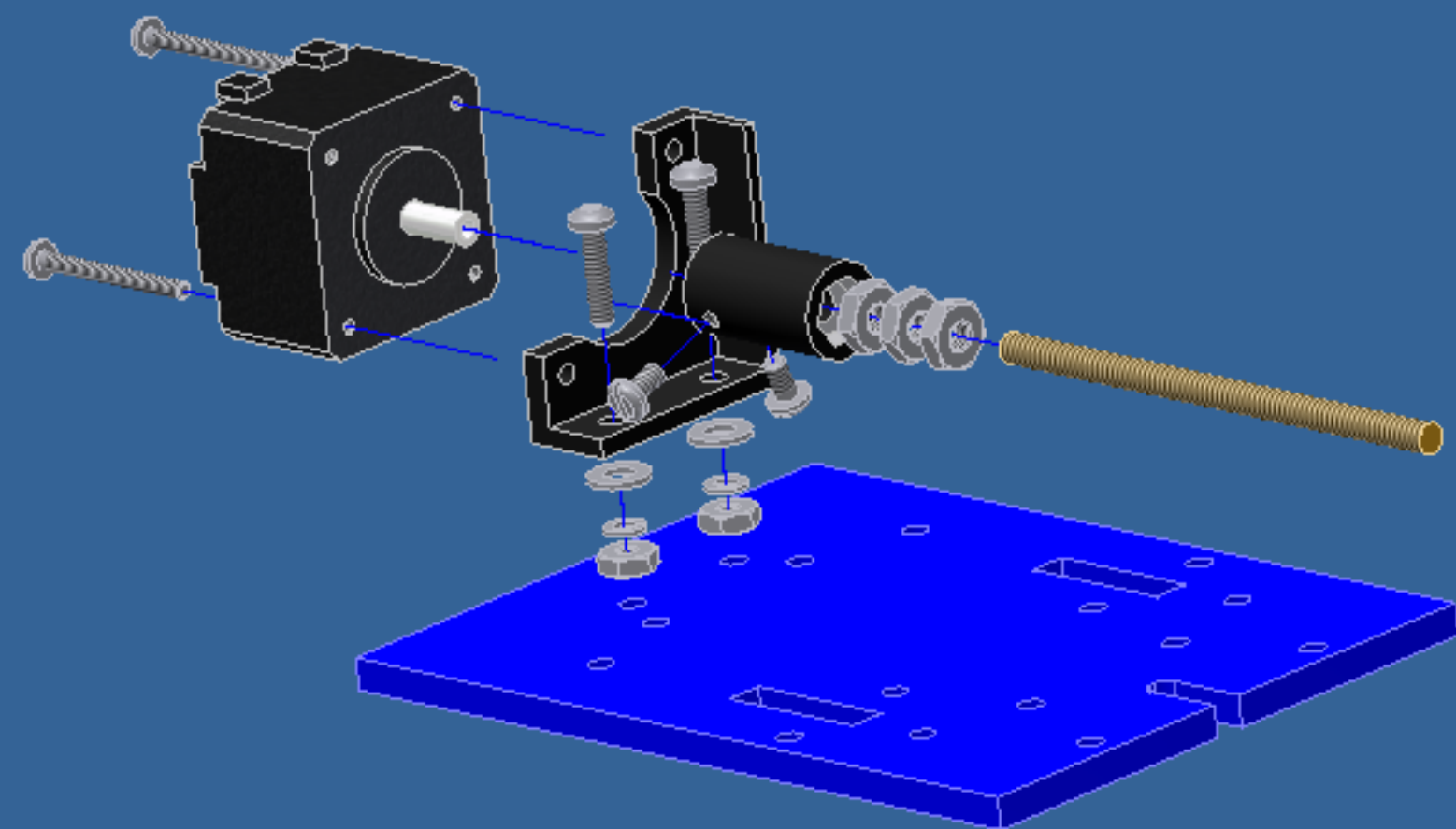


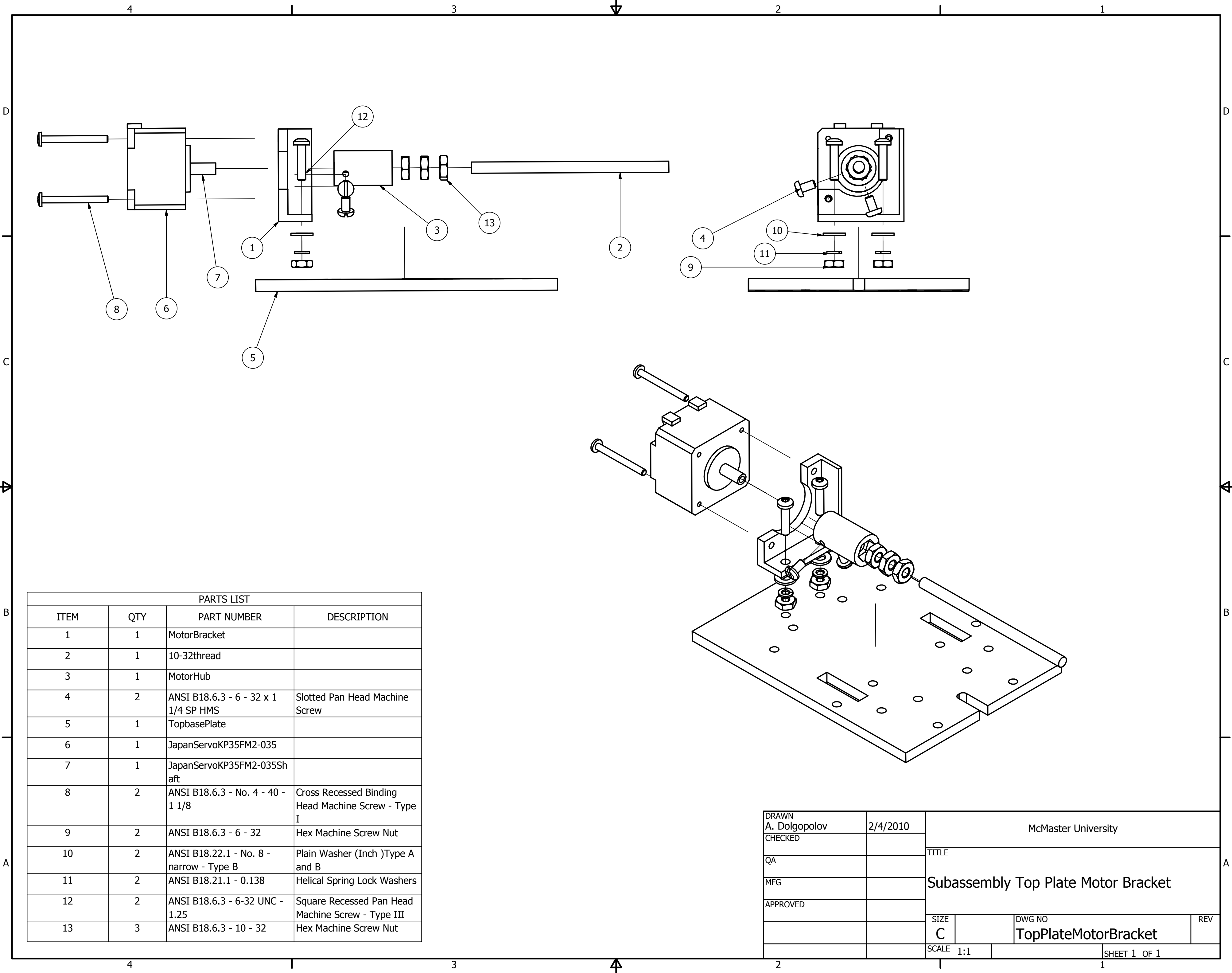
VIEW2
SCALE 1 : 1



MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR +0/-0.15
ANGULAR $\pm 0.5^\circ$

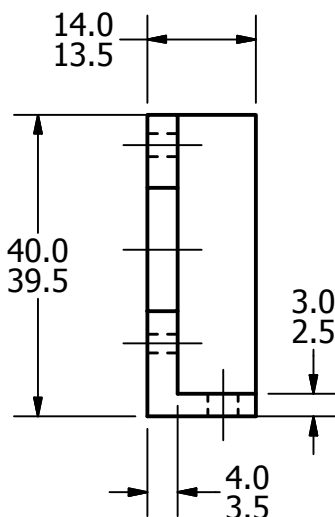
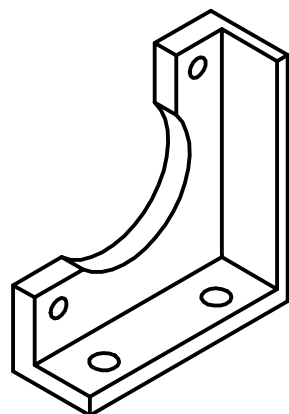
DRAWN A. Dolgoplov	12/16/2009	McMaster University		
CHECKED				
QA		TITLE Screw Holder		
MFG				
APPROVED		SIZE A	DWG NO screwholder	REV
		SCALE 2:1	SHEET 1 OF 1	





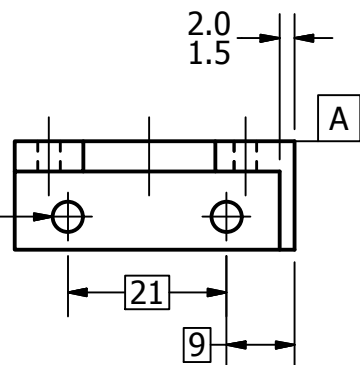
PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	MotorBracket	
2	1	10-32thread	
3	1	MotorHub	
4	2	ANSI B18.6.3 - 6 - 32 x 1 1/4 SP HMS	Slotted Pan Head Machine Screw
5	1	TopbasePlate	
6	1	JapanServoKP35FM2-035	
7	1	JapanServoKP35FM2-035Shaft	
8	2	ANSI B18.6.3 - No. 4 - 40 - 1 1/8	Cross Recessed Binding Head Machine Screw - Type I
9	2	ANSI B18.6.3 - 6 - 32	Hex Machine Screw Nut
10	2	ANSI B18.22.1 - No. 8 - narrow - Type B	Plain Washer (Inch)Type A and B
11	2	ANSI B18.21.1 - 0.138	Helical Spring Lock Washers
12	2	ANSI B18.6.3 - 6-32 UNC - 1.25	Square Recessed Pan Head Machine Screw - Type III
13	3	ANSI B18.6.3 - 10 - 32	Hex Machine Screw Nut

DRAWN A. Dolgoplov	2/4/2010	McMaster University		
CHECKED		TITLE		
QA		Subassembly Top Plate Motor Bracket		
MFG				
APPROVED		SIZE C	DWG NO TopPlateMotorBracket	REV
		SCALE 1:1	SHEET 1 OF 1	

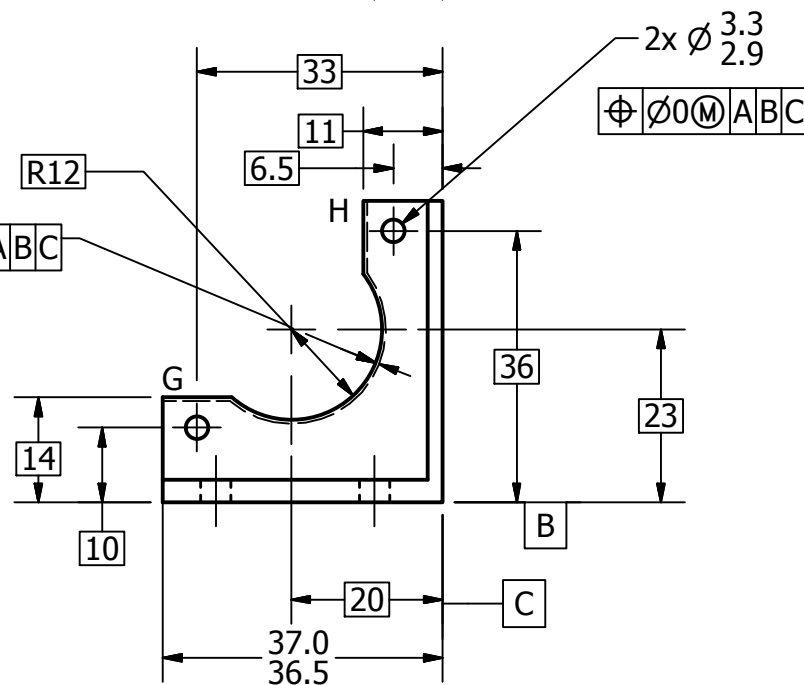


2x $\varnothing 4.3$
3.5

$\varnothing \varnothing 0 \textcircled{M} B A C$



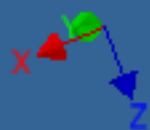
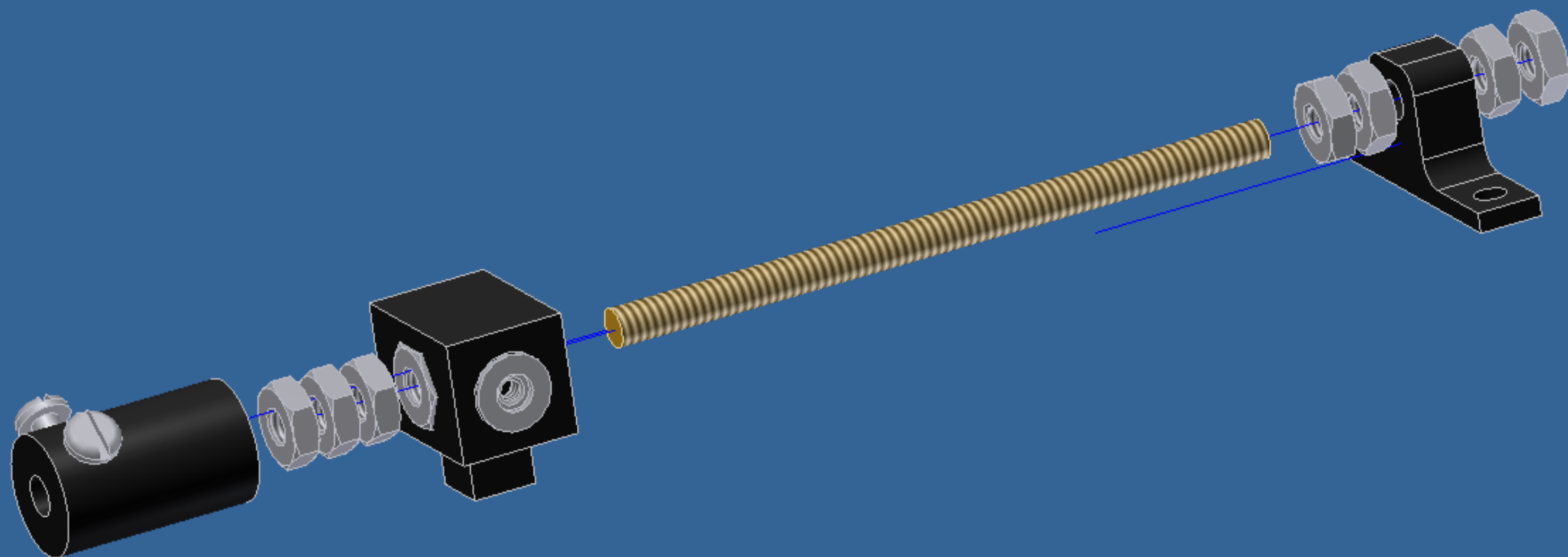
$R12$
 $\frac{1}{2} 0.5 A B C$
G ↔ H

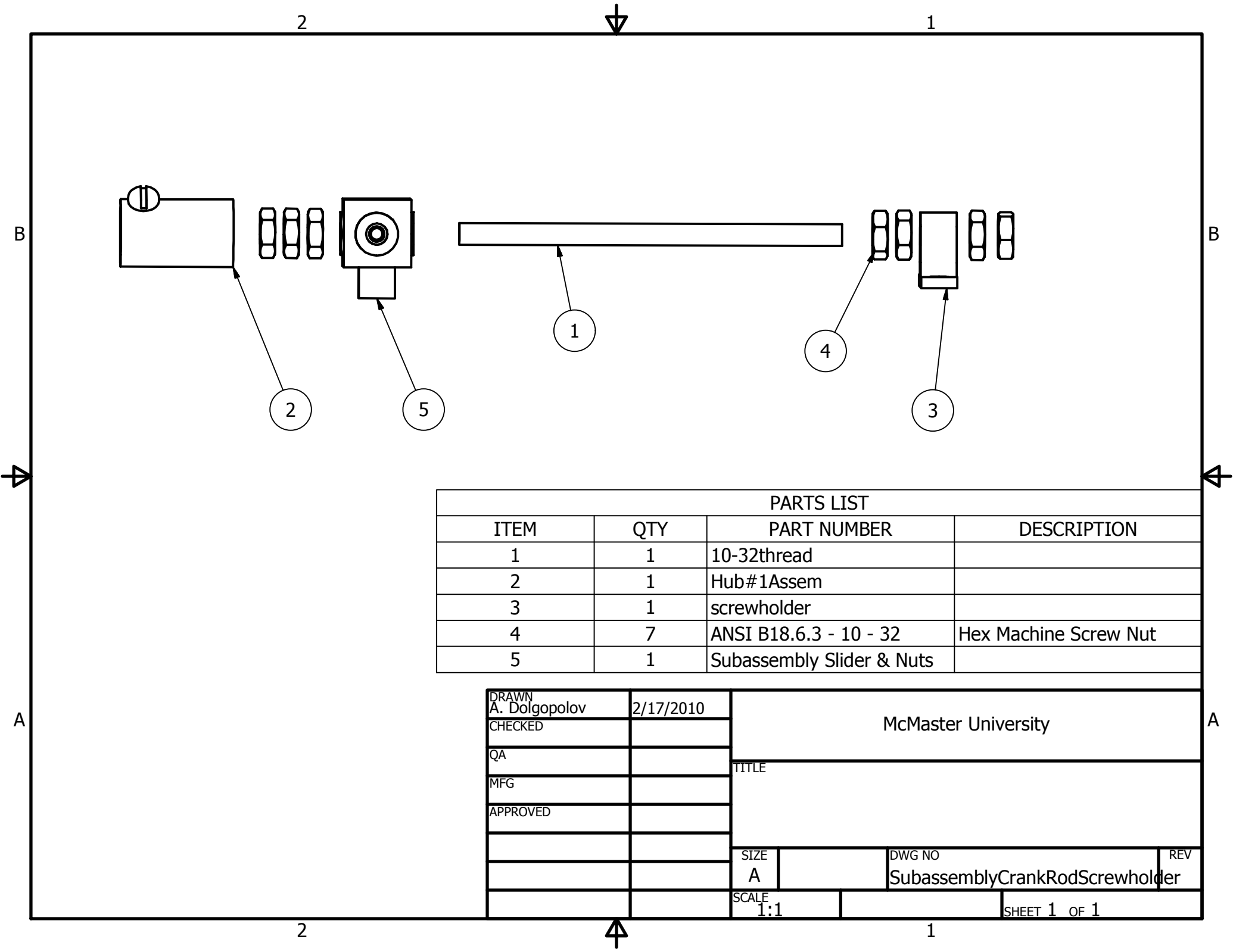


MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR $+0/-0.15$
ANGULAR $\pm 0.5^\circ$

DRAWN A. Dolgoplov	11/6/2009
CHECKED	
QA	
MFG	
APPROVED A. Spence	11/11/2009

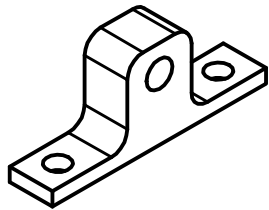
McMaster University		
TITLE		
Solar Tracker Motor Bracket		
SIZE A	DWG NO MotorBracket	REV
SCALE 1:1	SHEET 1 OF 1	



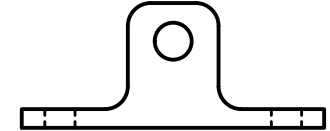
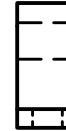
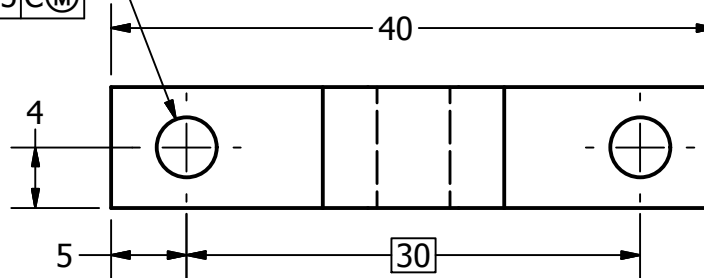


PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	10-32thread	
2	1	Hub#1Assem	
3	1	screwholder	
4	7	ANSI B18.6.3 - 10 - 32	Hex Machine Screw Nut
5	1	Subassembly Slider & Nuts	

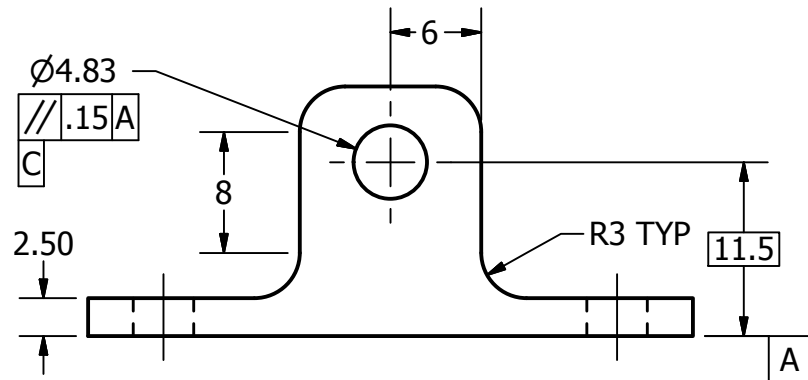
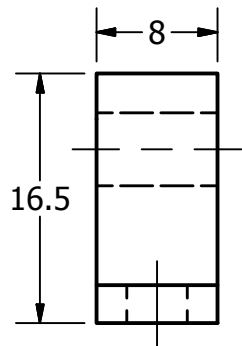
DRAWN A. Dolgoplov	2/17/2010	McMaster University		
CHECKED				
QA				
MFG				
APPROVED		TITLE		
		SIZE A	DWG NO SubassemblyCrankRodScrewholder	REV
		SCALE 1:1		
				SHEET 1 OF 1



\varnothing	$\varnothing 0$	\textcircled{M}	A
\perp	.15	C	\textcircled{M}
B			

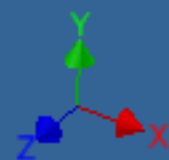
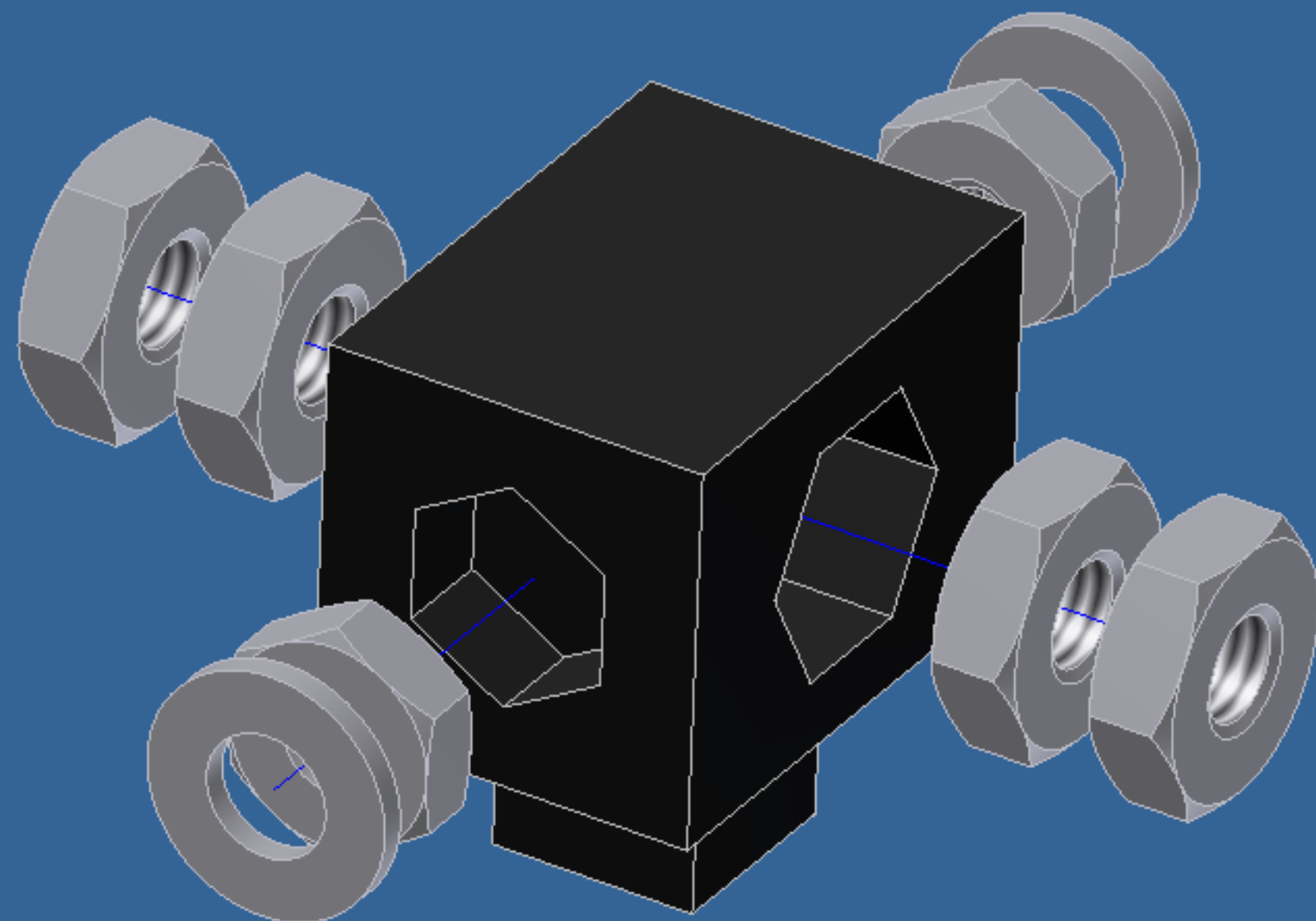


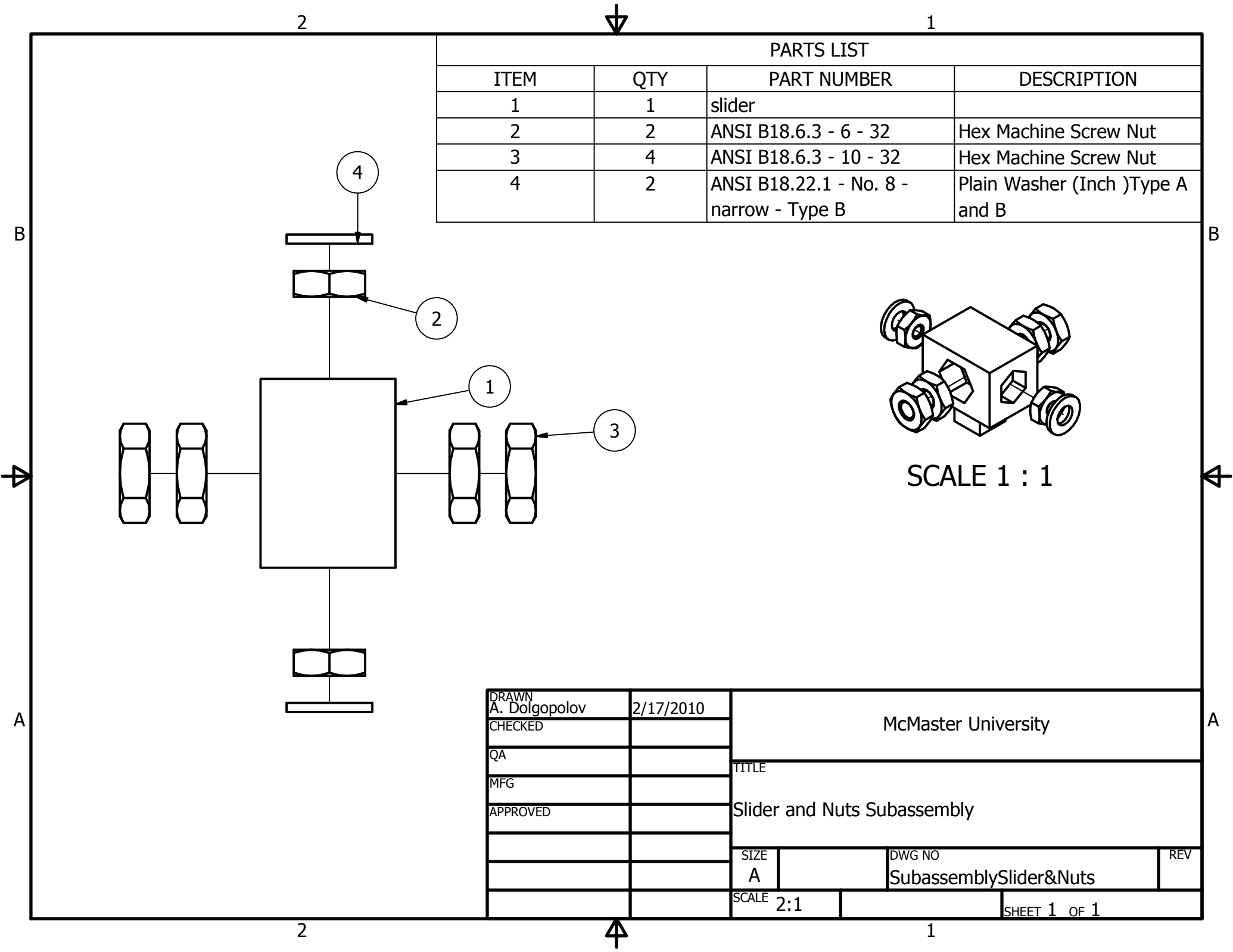
VIEW2
SCALE 1 : 1



MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR $+0/-0.15$
ANGULAR $\pm 0.5^\circ$

DRAWN A. Dolgoplov	12/16/2009	McMaster University		
CHECKED		TITLE Screw Holder		
QA				
MFG				
APPROVED				
		SIZE A	DWG NO screwholder	REV
		SCALE 2:1	SHEET 1 OF 1	

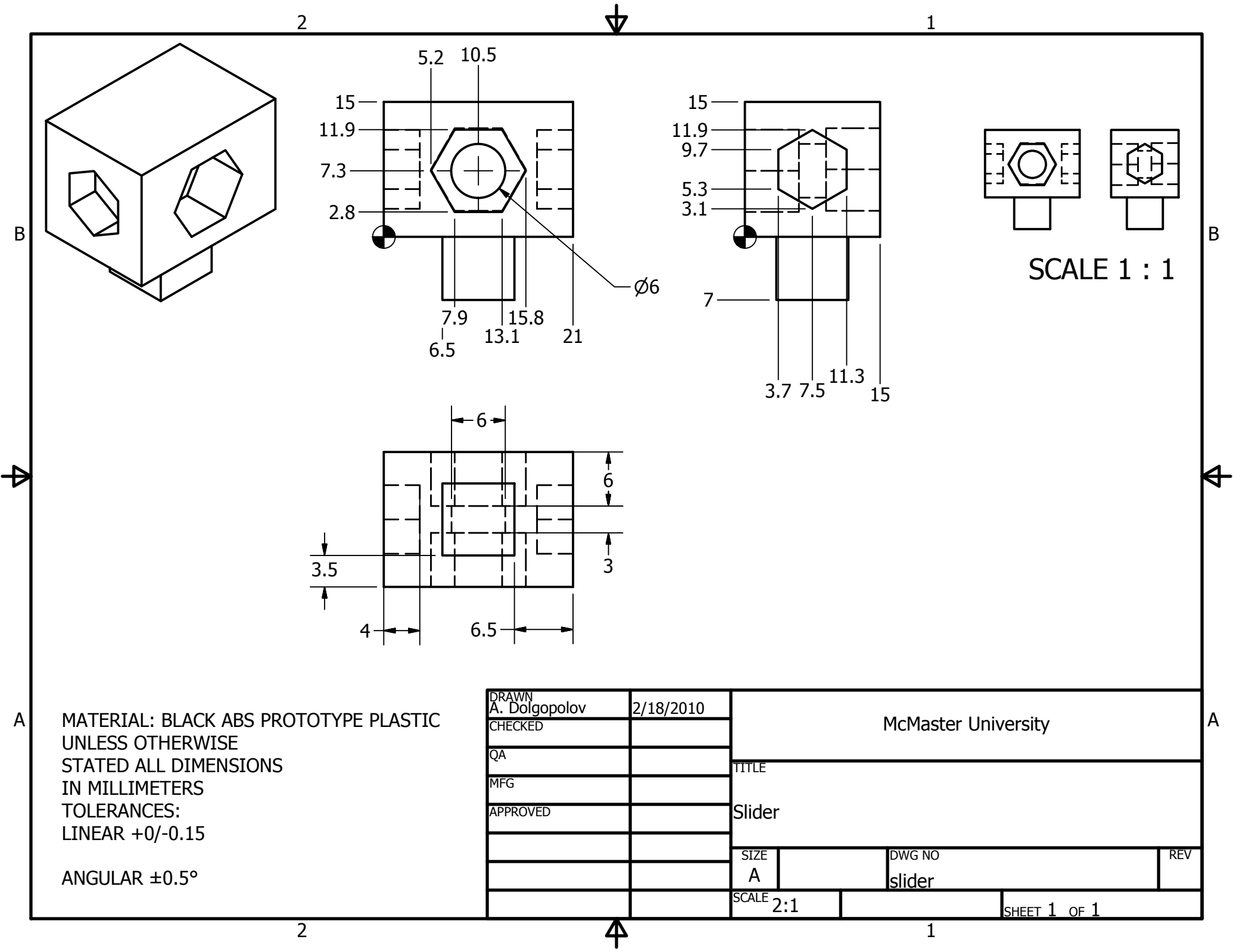




PARTS LIST			
ITEM	QTY	PART NUMBER	DESCRIPTION
1	1	slider	
2	2	ANSI B18.6.3 - 6 - 32	Hex Machine Screw Nut
3	4	ANSI B18.6.3 - 10 - 32	Hex Machine Screw Nut
4	2	ANSI B18.22.1 - No. 8 - narrow - Type B	Plain Washer (Inch)Type A and B

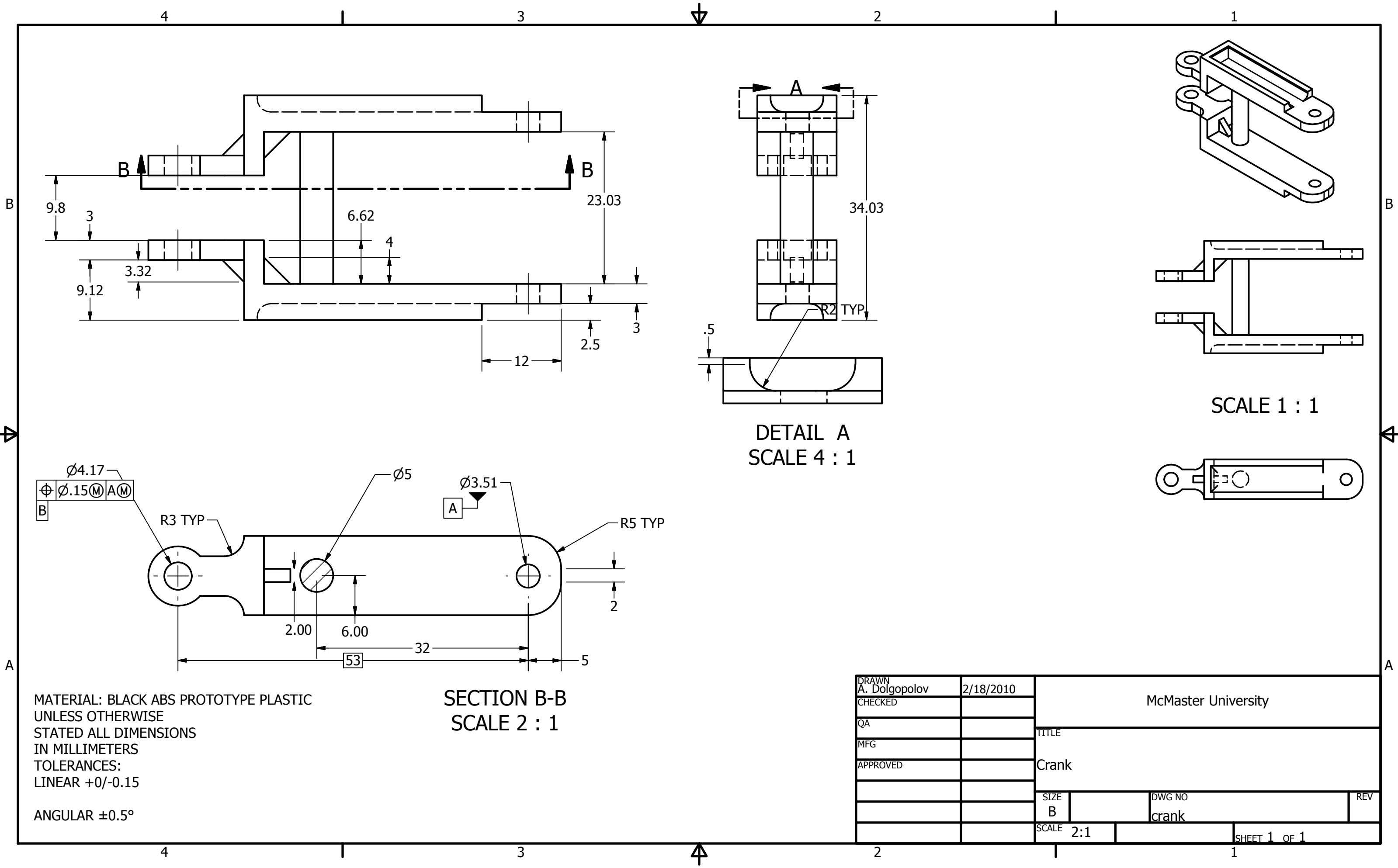
DRAWN A. Dolgoplov	2/17/2010
CHECKED	
QA	
MFG	
APPROVED	

McMaster University			
TITLE			
Slider and Nuts Subassembly			
SIZE A	DWG NO SubassemblySlider&Nuts	REV	
SCALE 2:1	SHEET 1 OF 1		



MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR $+0/-0.15$
ANGULAR $\pm 0.5^\circ$

DRAWN A. Dolgoplov	2/18/2010	McMaster University		
CHECKED				
QA		TITLE Slider		
MFG				
APPROVED		SIZE A	DWG NO slider	REV
		SCALE 2:1	SHEET 1 OF 1	



MATERIAL: BLACK ABS PROTOTYPE PLASTIC
UNLESS OTHERWISE
STATED ALL DIMENSIONS
IN MILLIMETERS
TOLERANCES:
LINEAR +0/-0.15
ANGULAR ±0.5°

SECTION B-B
SCALE 2 : 1

DETAIL A
SCALE 4 : 1

SCALE 1 : 1

DRAWN A. Dolgoplov	2/18/2010	McMaster University		
CHECKED				
QA		TITLE		
MFG		Crank		
APPROVED		SIZE B	DWG NO crank	REV
		SCALE 2:1	SHEET 1 OF 1	

Appendix A5 – Rapid Prototyped Component Costs

Cost of Rapid Prototype Parts						
Part #	RP Part Name	# of Pieces	ABS Material Used (inch^3)	Support Material Used (inch^3)	Cost (CAD)	Total Cost (CAD)
1	Gear Ring	4	8.68	3.64	104.16	232.32
2	Worm Gear	4	4.88	1.36	58.56	
3	Bridge Rail	1	0.45	0.89	5.4	
4	Crank Arm	1	0.38	0.24	4.56	
5	Motor Shaft Hub	1	0.23	0.11	2.76	
6	Screw Holder	1	0.13	0.04	1.56	
7	Pivot Stand	1	0.58	0.11	6.96	
8	Slider	1	0.25	0.09	3	
9	Panel	1	1.75	0.53	21	
10	Motor Bracket	5	1.5	0.45	18	
11	Switch Block Bracket	2	0.08	0.04	0.96	
12	Sun Beam Locator	1	0.22	0.09	2.64	
13	LDR Mount	2	0.12	0.08	1.44	
14	TIL159 Mount	4	0.11	0.07	1.32	

References

- [1] Blanco-Muriel, M., Alarcón-Padilla, D.C., López-Moratalla, T., and Lara-Coira, M., “Computing the Solar Vector”, *Solar Energy*, 70(5), 431-441, 2001.
- [2] Autodesk Inc., San Rafael, CA, www.autodesk.com.
- [3] Dimension, Inc., Eden Prairie, MN, www.dimensionprinting.com
- [4] Banzi, M., Cuartielles, D., Igoe, T., Martino, G., and Mellis, D., Arduino Alpha v17, www.arduino.cc
- [5] Vogt, N., Astronomy 110G-03 Lecture Notes – Fall 2007, New Mexico State University, <http://astronomy.nmsu.edu/nicole/teaching/astr110>.
- [6] www.timeanddate.com
- [7] CANMET Solar Paper, http://canmetenergy-canmetenergie.nrcan-rncan.gc.ca/eng/renewables/standalone_pv/publications/2006046.html
- [8] Photovoltaic Potential and Solar Resource Maps of Canada, Natural Resources Canada, https://glfc.cfsnet.nfis.org/mapserver/pv/index_e.php
- [9] Ontario Independent Electricity System Operator, Ontario Demand Forecast: 18-Month Outlook from April 2007 to September 2008, http://www.ieso.ca/imoweb/pubs/marketReports/18Month_ODF_2007mar.pdf
- [10] Eagle Layout Editor, <http://www.cadsoftusa.com/>