Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft

Part IV: Optical Encoder & Elevation Drive

James A Crawford

Synopsis

Part I provided a simple introduction to the big-picture objectives for this multi-phase project.

Part II first looked at telescope mounts, ultimately focusing on the azimuth-elevation type mount for the project. The basic mathematics for dealing with 3-phase DC motors (e.g., Clarke and Park transformations) were introduced, along with the first ingredients for modeling and controlling the DC motors in a precision manner. The *Launchpad* hardware platform from Texas Instruments was selected to host the motor control algorithms.

In Part III, most of the attention was focused on the mechanical design, fabrication and assembly of the telescope mount. The detailed design of the hardware changed appreciably from the first concept as better approaches were recognized during the detailed design. A first-look at low rotational speed cogging torque was also made.

In this part (Part IV), attention is directed to (i) the mechanical drive details for the elevation axis utilizing finally settling on a 25:1 belt-drive step-down approach and (ii) interfacing the optical encoders [5] *on the azimuth and elevation axes to an Arduino Mega2560 as an interim step to the TMS320F28379D digital signal processor.*

This project update has been a long time in coming primarily due to detours involving development of the elevation drive elements. Even so, the journey has been enjoyable and full of learning experiences pertaining to harmonic drives, involute gears, and associated topics. The chosen optical encoder (Avago AEAT-9000) at 17-bits has proven to be a very adjustment-delicate device to calibrate and that factor also slowed progress. For example, the associated datasheet led me to initially believe the output 17-bit value was a linear binary word, but in the end, I figured out it was a Gray-encoded value! Had there been more information on the Internet from others using this encoder, my journey would have been streamlined considerably.

1 Project Context

Most of the azimuth-elevation telescope mount was completed in the previous portion (Part III) of this project as shown in Figure 1. Several important mechanical ingredients were still not completed, however. These included:

- Elevation axis counter-weight assembly
- Azimuth axis counter-weight assembly
- Elevation motor concept, design, fabrication and assembly

Of these mechanical items, the elevation axis motor approach has required the most development effort by far.

Figure 1 Az-El telescope mount design and fabrication progress as reported in the Part III report [3]

The electronic elements of the project are in their infancy at this point. Although the 3-phase driver electronics (DRV-8301) mate with the TMS320F28379D digital signal processor nicely, this cannot be said of the optical encoder electronics. As discussed later in §4.1, I decided to take a detour to first develop the encoder interface using an Arduino MEGA2560 for simplicity.

2 Counter-Weight Assemblies

The net moment arm about elevation axis must be zero in order to demand minimum torque from the elevation drive motor. Since the telescope is offset from the elevation axis slightly as shown in Figure 2, the balancing act is a bit more involved and requires attention to two axes.

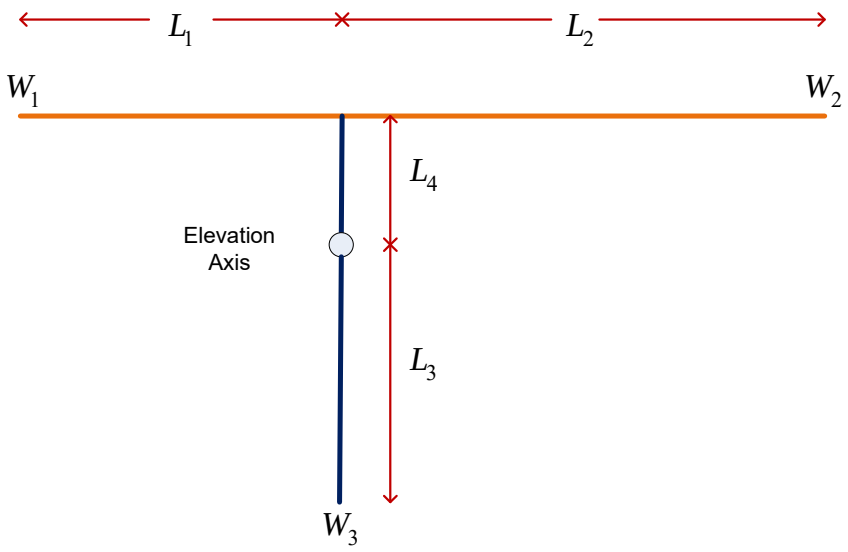


Figure 2 Moment arms about elevation axis[1]

Balance about the elevation axis in Figure 2 requires the net moment about the axis to always be zero. The net moment condition must also be maintained when the L_1-L_2 segment is tilted at an angle θ as shown in Figure 3. An application of engineering statics shows this condition (and consequently balance) is achieved when

$$-W_1 R_1 \cos\left[\theta - \tan^{-1}\left(\frac{L_4}{R_1}\right)\right] + W_2 R_2 \cos\left[\theta + \tan^{-1}\left(\frac{L_4}{R_2}\right)\right] + L_3 W_3 \sin(\theta) = 0 \tag{1}$$

where W_k represent point-weights with L_k and R_k being the distances shown in Figure 2. In the simple case where $\theta = 0$ corresponding to the L_1-L_2 segment being horizontal, this reduces to

$$W_1 L_1 = W_2 L_2 \tag{2}$$

which is a necessary balancing condition. This point also illustrates the steps to be taken in balancing such an assembly: (i) with the L_1-L_2 arm in the horizontal position, adjust W_1 or W_2 to achieve (2), and then (ii) adjust weight W_3 to achieve (1) for nonzero values of θ.

In order to solve (1) for the more general case, the starting point is dictated by (2) as just stated. Equation (1) can be slightly re-written for the net moment error as

$$M_e(\theta, W_3) = -W_1 R_1 \cos\left[\theta - \tan^{-1}\left(\frac{L_4}{R_1}\right)\right] + W_2 R_2 \cos\left[\theta + \tan^{-1}\left(\frac{L_4}{R_2}\right)\right] + L_3 W_3 \sin(\theta) \tag{3}$$

[1] From U26802 Diagrams.vsd.

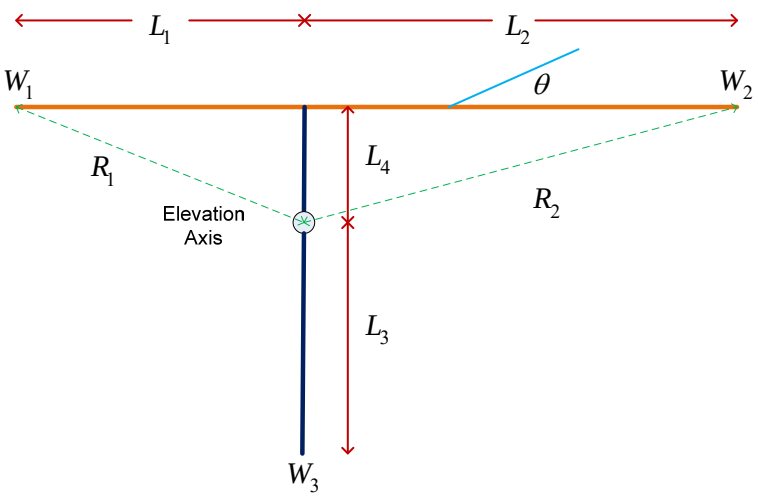


Figure 3 Repeat of Figure 2 with additional values shown

This error function is shown in the case of W_1 = 2, L_1 = 1, L_2 = 2, and L_3 = 0.50 for several values of W_3 in Figure 4. Since all of the curves have a positive slope, the exact solution for W_3 can be found by setting the derivative of (3) to zero thereby producing

$$W_3 = \frac{W_1 R_1 \sin(\theta_a) + W_2 R_2 \sin(\theta_b)}{L_3} \tag{4}$$

where $\theta_a = \tan^{-1}\left(\frac{L_4}{R_1}\right)$ and $\theta_b = \tan^{-1}\left(\frac{L_4}{R_2}\right)$.

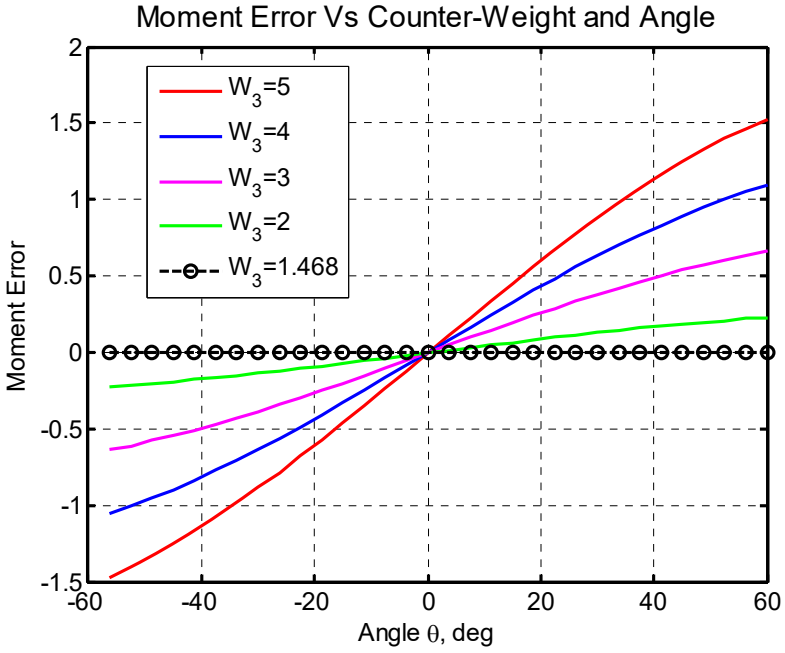


Figure 4 Net moment error example versus different counter-weight values[2]

[2] Using u26803_counterbalacing.m.

2.1 Telescope Tube Counter Weight Assembly

I fabricated a counter weight holder from 1” thick 6061 aluminum as shown in Figure 5. I had decided early on not to use a large-diameter rod to support the counter weights, but instead to use two 3/8” diameter stainless steel rods as shown in the figure. This approach led to a better counter weight torque to physical weight ratio.

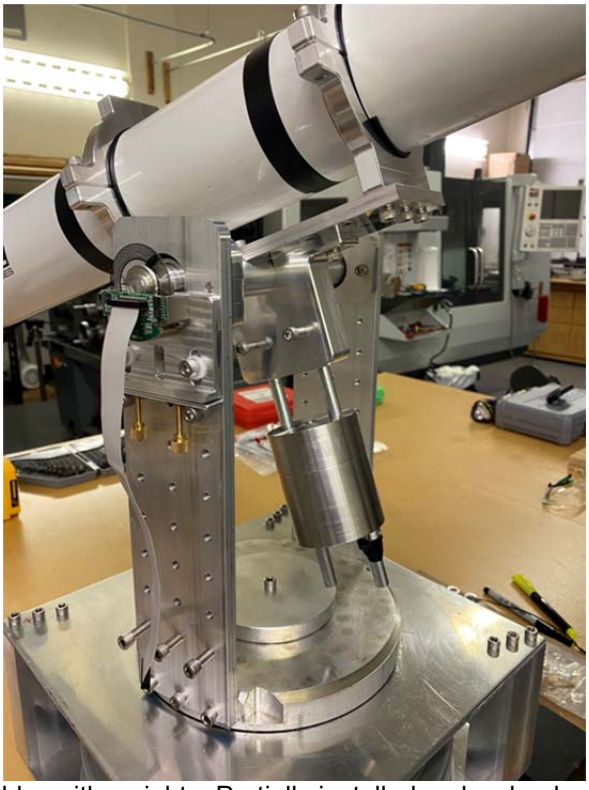


Figure 5 Counter weight holder with weights. Partially installed code wheel and reader shown on the left-hand side of the elevation axis.

2.2 Counter Weights

I needed a more dense material than aluminum for the elevation counter-weight. I also wanted the selected material to be durable. I happened onto several lengths of 2.75” diameter 15-5 stainless steel material which I was told were remnants for Boeing 727 landing gear axles.

I turned the material on my manual lathe primarily just to clean up the lateral surface somewhat. This also entailed drilling a center hold in each cylinder of material first.

Drilling the two additional holes for the 3/8” stainless steel rod holders was a little more challenging than first anticipated. The slightest amount of vibration led to substantial drill-bit walking (and dulling) on the surface even though I had pre-drilled locations with a center-stub drill. I ultimately opted to use one of the chucks from my lath to firmly secure the 15-5 stock material with the entire assembly secured in a Kurt vice as shown in Figure 6. Even the slightest amount of vibration proved to be problematic and this point cannot be over emphasized.

Figure 6 15-5 stock material secured in lathe chuck further secured in a Kurt vice

Figure 7 Telescope mount with counterweights temporarily installed

Figure 8 One counter weight with finished drilling (3/8” holes). The small piece of rectangular aluminum bar stock underneath the 15-5 workpiece made it possible to drill completely through the workpiece without damaging the chuck.

3 Elevation Drive

Unlike the azimuth drive for the project, the elevation drive needs to use a much lower torque motor to keep its size down. As such, the motor torque needs to be increased by some mechanical means. Generally speaking, motor torque is proportional to motor RPM, but the RPM is essentially zero for a positioning-only application like this.

3.1 Introduction to Harmonic Drives

The motivation for looking at the *harmonic drive* concept in the first place is that it offers (i) potentially zero back-lash, (ii) high reduction ratios, (iii) good torque efficiency, and (iv) potentially small size. Commercial harmonic drives offer excellent performance but are also quite expensive ranging from $280 to more commonly $1,000 or more each. Purchasing ready-made harmonic drives would more or less defeated the purpose of having my own precision mill too. A break-out picture of a commercially available harmonic drive is shown in Figure 9.

3.2 Baseline Trial

The most difficult component to obtain for a harmonic drive is unquestionably the flexspline. The flexspline shown in Figure 9 does not lend itself to easy machining. Rather than attempt to use a fully geared design as shown in this figure, I initially opted to pursue a friction-based design and subsequently found someone else who had done this earlier and patented it [10]. A conceptual approach from this patent is shown in Figure 10.

Figure 9 Traditional harmonic drive illustrating the fixed outer gear ring, the flexspline, and wave generator

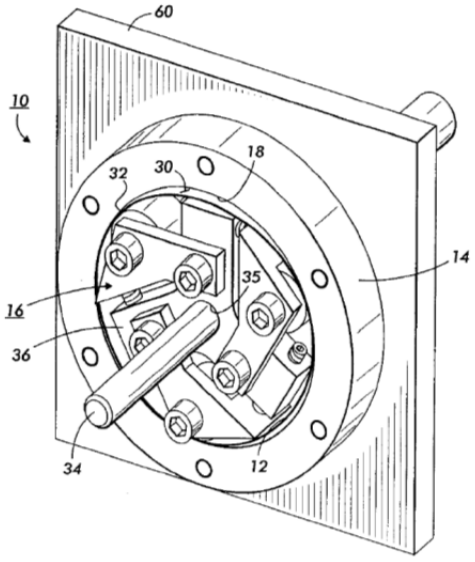


Figure 10 Baseline harmonic friction drive concept from [10]

Before completing the mechanical work, major portions shown in Figure 11 through Figure 18, I realized, however, that the coefficient of friction between the flexspline and aluminum outer frame was going to be insufficient to deal with the torque forces I expected for the telescope mount. Had I known the coefficients of friction involved in advance, I could have saved myself the fabrication effort, but I did not. I

Figure 11 Plain tinned-steel soup can

Figure 12 Center-punched cut-off soup can from Figure 11

Figure 13 My flespline shown with fabricated axle mount assembly

Figure 14 Back-side of Figure 13

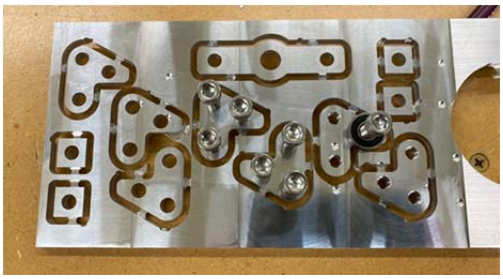


Figure 15 Milled piece-parts for center arm portion

Figure 16 Assembled center section including ball bearings which apply pressure to the inside of the flexspline

considered roughening the lateral surface of my flexspline to increase the coefficient of friction, but in the end I deemed this more ad-hoc than I really wanted to pursue. I was also concerned about sinking more time into the friction-based design only to find out again that it could not deliver the holding torque my application required.

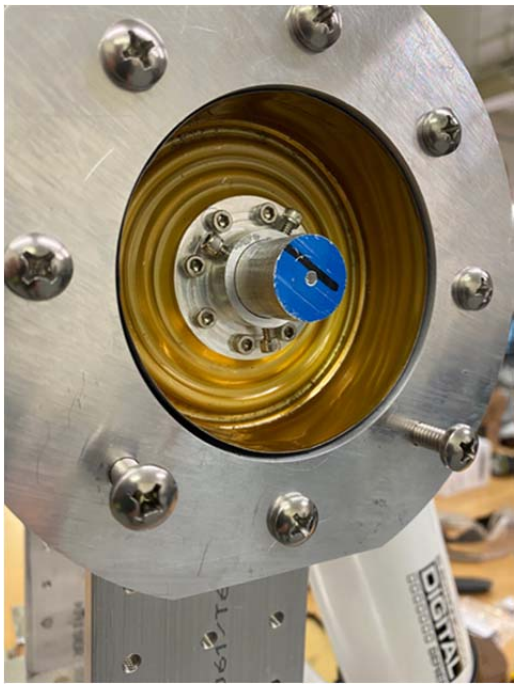


Figure 17 Flexspline and outer frame attached to the elevation axis

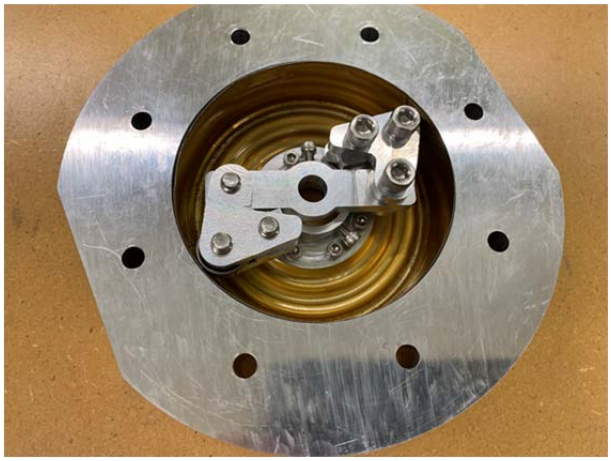


Figure 18 Mock-up arrangement of outer frame, flexspline, axle mount, and center section

3.3 Other Elevation Drive Options

The mechanical drive must ideally exhibit:

- torque amplification on the order of 10X or more
- provide good torque efficiency
- ease of fabrication
- zero backlash

The last attribute is vital because the elevation motor plus mechanical drive will be embedded within a closed-loop control system. Any backlash will be manifested as very undesirable elevation jitter if not outright instability. The mechanical drive options I considered are:

- Cycloidal
- Harmonic Drive Using Metallic Flexspline
- Harmonic Drive Using Plastic Flexspline
- Harmonic Drive Using Wide Timing Belt as Flexspline
- Step-Down Assembly Using Metal Disk + Rubber Wheel
- Step-Down Assembly Using Belt-Drive & Pulleys

3.3.1 Cycloidal Drive

Cycloidal drives utilize a combination of flat disks and pins as shown in Figure 19 and Figure 20. For the example shown there are *N* = 9 internal pins and *N+1* outer pins which serve to deliver a 9:1 reduction ratio. Pros and cons for the cycloidal drive as it applies to this project are Table 1.

Table 1 Cycloidal Drive Pros and Cons

Pros:	Cons:
• Easily fabricated on my Haas mill	• Good tolerances required for minimum back-lash
• Ideally zero or very low back-lash	• Periodic error associated with eccentric shaft portion, but not an issue when embedded in a control loop with precision rotary encoder
• Low profile	• Some view it as obsolete
• Aluminum / hard-plastic fabrication	• Possible jamming
• Useable reduction ratio	• Finer points needed for quality design are rather elusive

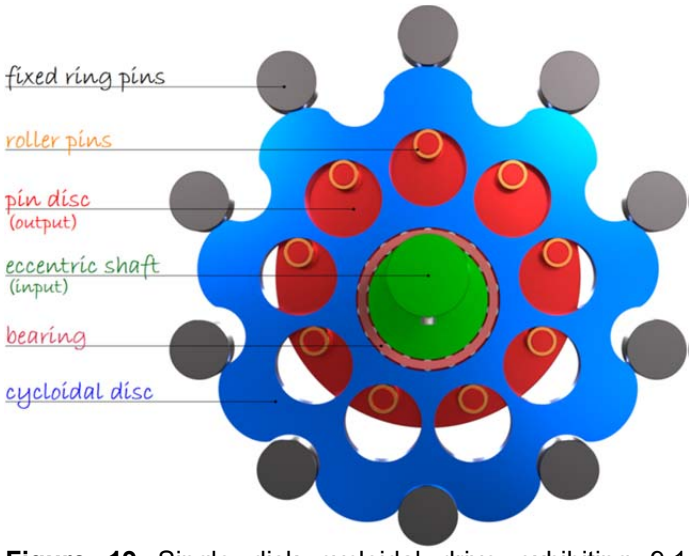


Figure 19 Single disk cycloidal drive exhibiting 9:1 reduction ratio[3]

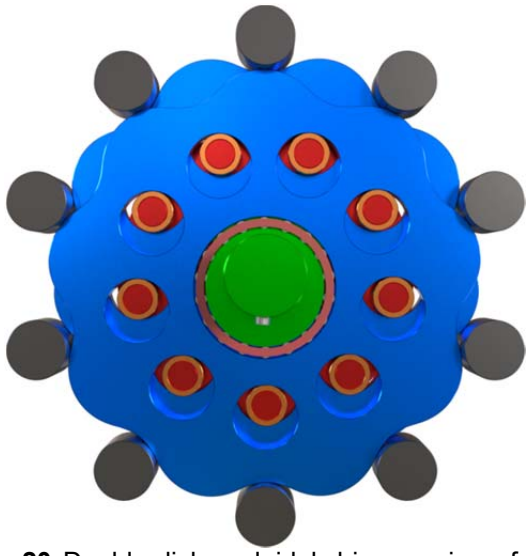


Figure 20 Double-disk cycloidal drive version of Figure 19 used to reduce wobble-vibration

3.3.2 Adopted Pulley-Based Design

I ultimately decided to choose the step-down approach based upon timing-belts and pulleys. This approach has a number of attractive features as summarized in Table 2. More details pertaining to torque and pulley options are provided in §10.

Table 2 Positives and Negatives for Pulley-Based Reduction

Positives	Negatives
Readily available timing belts and pulleys- consequently only simple machining needed	More moving parts
No reliance on slipping surfaces, whether gears or harmonic drive related gears	Larger than some approaches
More than enough torque	Requires belt tensioning
Greater than 50% of pulley circumference engaged with belts making for outstanding	Periodic error (but correctable with closed-loop control)

[3] From https://www.tec-science.com/mechanical-power-transmission/cycloidal-drive-speed-reducer-gear/how-does-a-cycloidal-gear-drive-work/

Positives	Negatives
smoothness	
Should exhibit zero backlash	
Virtually no wear	
Easily extended to azimuth axis in future versions	

My milling work was reduced by using readily-available precision timing pulleys[4] and their associated timing belts as also discussed in §10. I opted to use a cascade of two 5:1 step-downs in series thereby increasing the stall-torque by a factor up to 25x. For size considerations, a small 3-phase DC motor was chosen for the elevation axis. A picture of the adopted motor is shown in Figure 51.

Figure 21 80-tooth large pulleys with 20mm and 5mm shaft diameter, along with one of the two 16-tooth small pulleys with 5mm shaft diameter. Diameter of large pulleys is about 2".

Several cut-away looks at the pulley housing design from Fusion 360 are shown in Figure 22 through Figure 26. Getting the pulley spacing just right so that tensioners could still perform their function correctly was rather tedious in part because the mechanical details for the timing belts were not complete. I consequently had to fabricate a small jig with different center-to-center pulley distances to hone in on the final dimensioning.

Since the tensioners act to take up any extra slack in the timing belt by using only lateral force as shown in Figure 27, the starting center-to-center pulley distance must be very accurate or sufficient slack cannot be removed. The situation is very similar to increasing tension in a violin string by only applying lateral force. For example, assume the straight-belt length between the two end-points shown in Figure 27 (b) is represented by *L* but the actual distance is in error by δL . Assume further that the non-taunt

[4] I have subsequently identified a US company named Breckoflex which has a wide range of precision pulley and belt options. I will most likely turn to their product line for future efforts.

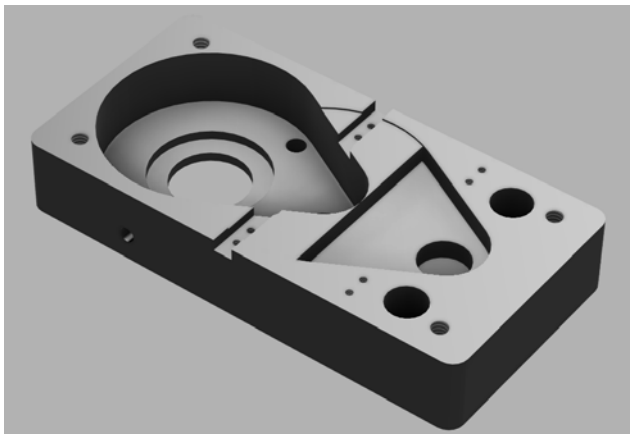


Figure 22 Bottom 1” plate

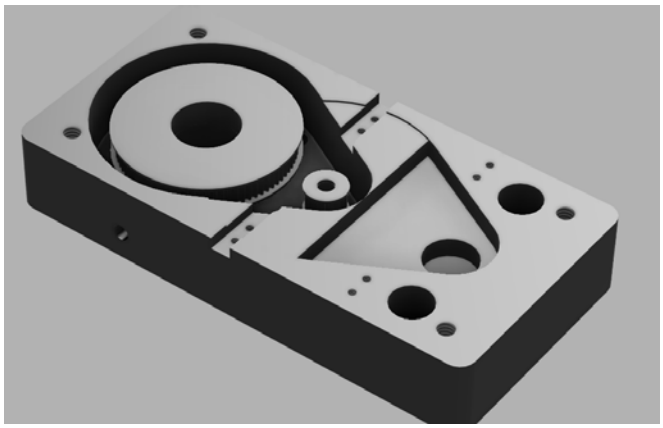


Figure 23 Bottom 1” plate with first 5:1 pulley-reduction step (80 teeth for large pulley, 16 for small)

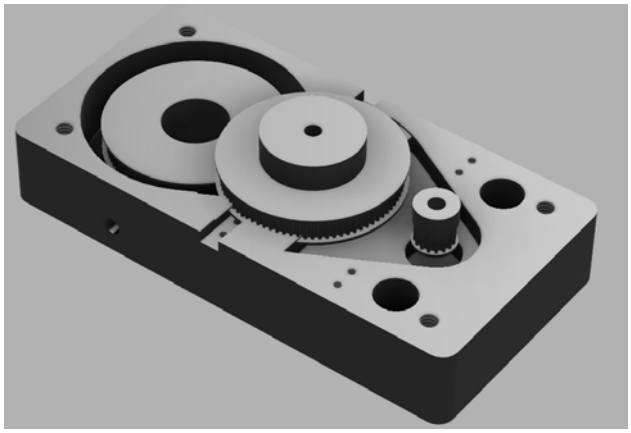


Figure 24 Bottom 1” plate along with both 5:1 pulley-reduction steps

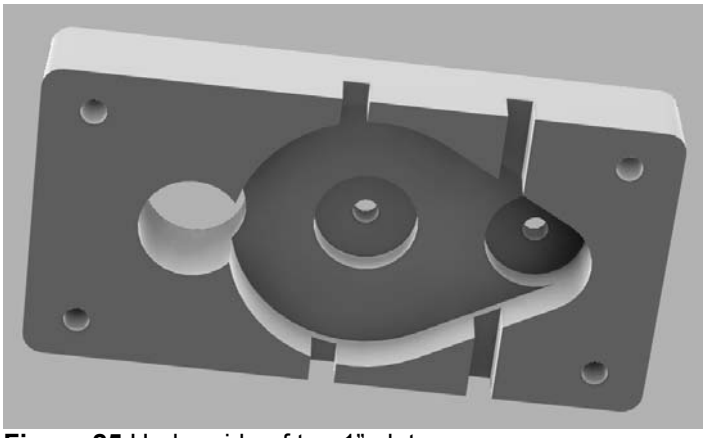


Figure 25 Under-side of top 1” plate

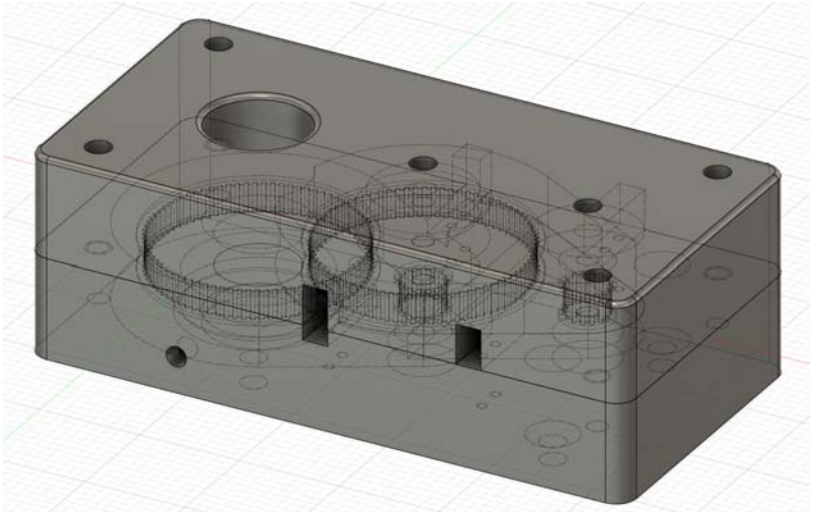


Figure 26 See-thru stack-up of entire elevation assembly less (i) shafts, (ii) tensioners, (iii) bearings, and (iv) pulley belts

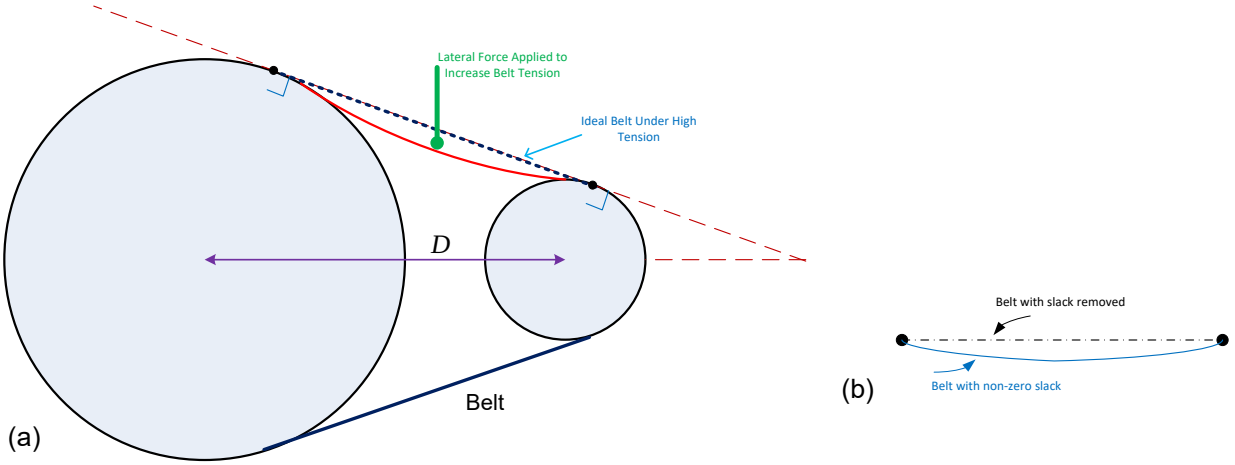


Figure 27 (a) Pulley arrangement with finite belt-slack, (b) as with a violin string, removing slack with only lateral force involves substantial lateral displacement to remove the slack[5]

belt takes the shape of a circular arc as shown having a radius given by *R*. Even a slight pulley-spacing error δL will manifest itself as a significant *sag* in the belt. More specifically,

$$\delta L = 2R\sin^{-1}\left(\frac{L}{2R}\right) - L \tag{5}$$

which leads to

$$R = \sqrt{\frac{L^3}{24\ \delta L}} \tag{6}$$

For a numerical example, if *L* = 3” and the

$$h \simeq \frac{L^2}{2R} \tag{7}$$

where *L* is the length of the ideal straight belt section and *R* is the radius of curvature for the belt having some sag. For a very tight belt, *R* should be very large compared to *L*. If *L* = 3” and *R* = 60”, *h* is still 0.075” which is still appreciably large compared to a timing belt tooth.

Prototype Work

I fabricated several incremental versions of the design Figure 26 in plastic before cutting real metal. This exercise proved to be invaluable. Some of the initial design details are provided in §10. Several pictures of this work are shown in Figure 28 through Figure 30. The metal version will be fabricated in the next few days. Once assembled, this pulley assembly will be married with the small 3-phase DC motor shown in Figure 51 to complete the mechanical portion of the elevation drive.

[5] From U27003 Pulley System.vsd.

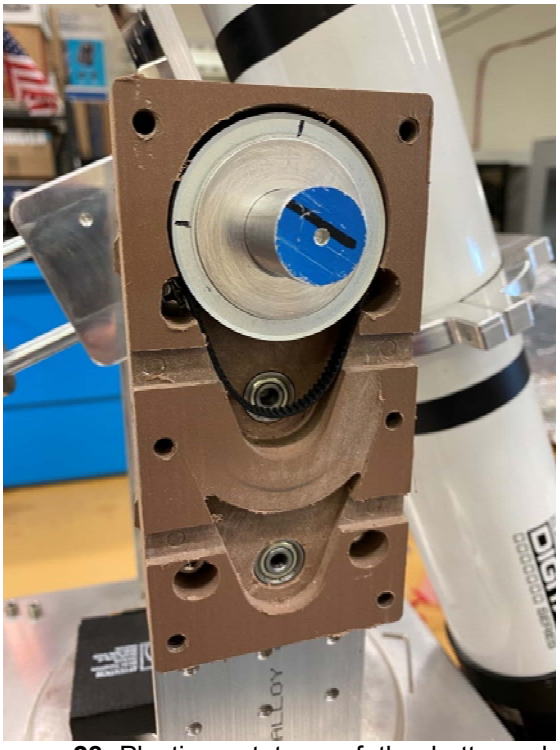


Figure 28 Plastic prototype of the bottom clam shell portion showing the first 80-tooth pulley in place

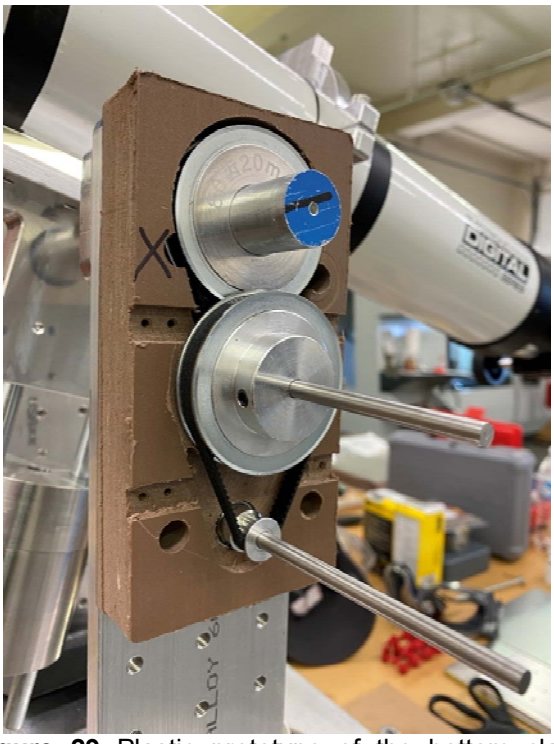


Figure 29 Plastic prototype of the bottom clam shell portion showing the first 80-tooth pulley, first 16-tooth pulley (hidden), and second 80-tooth pulley driving the second 16-tooth pulley

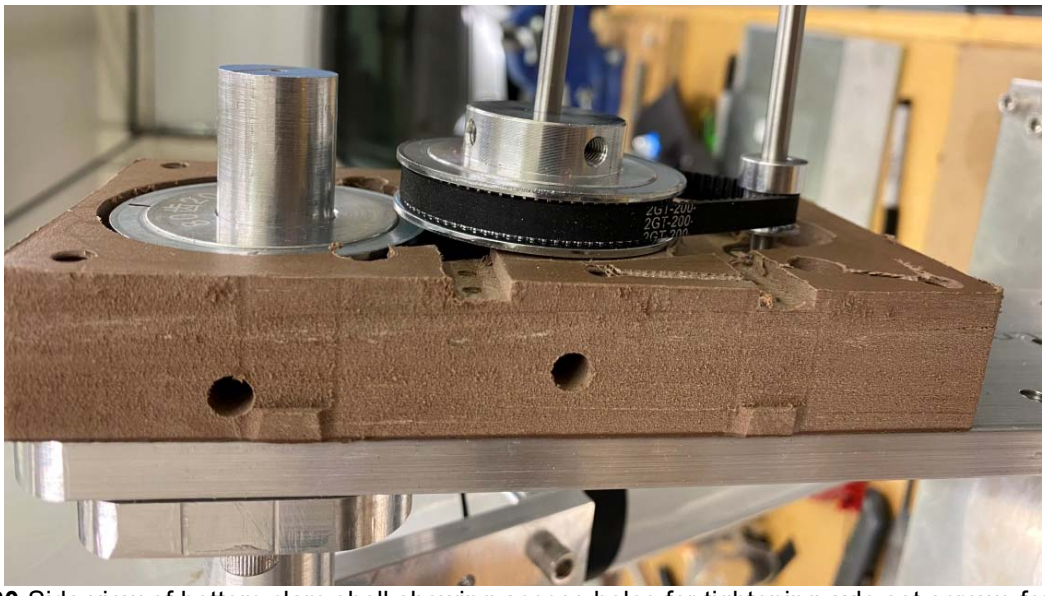


Figure 30 Side view of bottom clam shell showing access holes for tightening axle set-screws for the first 80-tooth pulley and first 16-tooth pulley

4 Optical Encoder Interface with TMS320F28379D

The optical encoder chosen for this project[6] is the AEAAT-9000-1GSH0 (Full Option) by Avago [5]. It provides 17-bit absolute angular resolution (1:131,072, equivalent to about 9.888 arc-seconds) with a readout rate as high as roughly 550 kHz. Assuming a 1 MHz clock rate, however, an ADC conversion rate of 10 ksps will be more than adequate for this project while still leaving 20,000 DSP clock cycles (at 200 MHz) in between samples to perform digital signal processing calculations.

The 2-wire SCL option will be adopted for positional readout from the encoder.

The interfacing task requires a clear understanding of both sides of the interface. Since many of the I/O pins of the TMS320F28379D will be taken up by the two 3-phase driver booster-packs, careful resource allocation is required. The encoder's SPI interface is used to setup and control the encoder (Figure 31) whereas the SSI interface is used to read back rotational position values to the DSP (Figure 32).

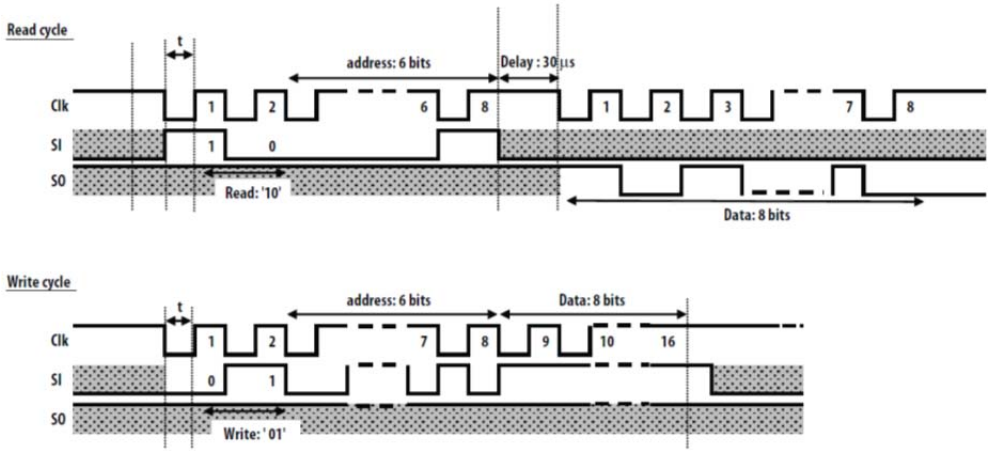


Figure 31 SPI timing diagram for encoder read and write operations [5]

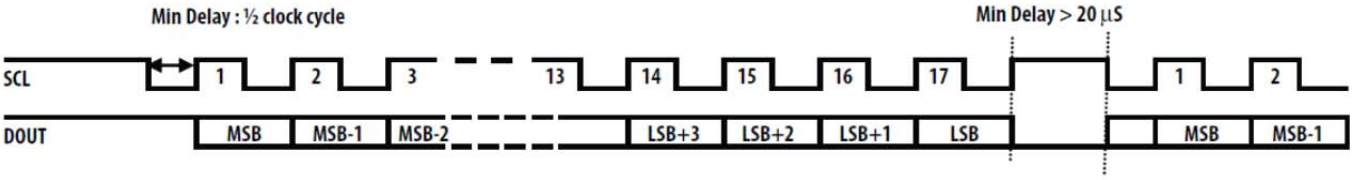


Figure 32 2-wire SSI timing diagram for single-ended control [5]

There is an alignment kit available for the AEAT-9000-1GSH0 (HEDS-8949) but it retails for about $850 from Digikey. The PC software, however, targets the Microsoft Windows XP operating system, and for this price, I am not inclined to take my chances with interoperability[7] with my Windows 10 machines. Once the encoder has been physically aligned with the azimuth or elevation axis in question, only a small subset of the interface signals listed in Table 3 will actually be needed. It is therefore advantageous to make the alignment process as simple and convenient possible.

[6] This device was unfortunately obsoleted by Broadcom/Avago in roughly June 2020. Finding a 17+ bit precision single-turn rotary encoder at the same price point has proved to be very elusive thus far.

[7] I have subsequently been able to acquire the executable file for this calibration kit and it appears to operate correctly on Windows 10 albeit without the hardware portion of the calibration kit.

Of the highlighted pins listed in Table 3, there are 5 Analog Out pins, 7 Digital In pins, and 6 Digital Out pins. With this pin count, it was decided to use an Arduino Mega 2560 to initially get the encoder operational and aligned since it is easy to program and already has a USB port for easy communication with a PC. The Mega 2560 also has a 5V output drive capability sufficient to power the optical encoder electronics.

Table 3 Encoder Interface (Highlighted Pins Brought Out with HEDS-8949 Alignment Kit per Table 4)

No.	Pin Name	Description	Function	Comments	Provided to F28379D
1	COSINE+	Analog Out	Diff Cosine+ Analog Output	Buffer with op-amp	
2	COSINE-	Analog Out	Diff Cosine- Analog Output	Buffer with op-amp	
3	SINE+	Analog Out	Diff Sine+ Analog Output	Buffer with op-amp	
4	SINE-	Analog Out	Diff Sine- Analog Output	Buffer with op-amp	
5	TiltOut	Digital Out	Tilt Alignment Output, CMOS		
6	GND	GND1		Ground pin.	
7	LocTest	Analog Out	Alignment Locate Output		
8	GND	GND2		Ground pin.	
9	NC				
10	MSBINV	Digital In	Inverted Counting	Ground pin.	
11	SPI_CLK	Digital In	SPI Clock Input		
12	Zero_RST	Digital In	Pull Down to Zero the Absolute Position		
13	SPI-SI	Digital In	SPI Data Input		
14	NSL+	Digital In	NSL+ (Grounded for 2-Wire)	Ground pin.	
15	SPI_SO	Digital Out	SPI Data Output, CMOS		
16	NSL-	Digital In	NSL- Differential (VDD for 2-Wire)	Connect to VDD	
17	GND	GND2		Ground pin.	
18	GND	GND2		Ground pin.	
19	INCB	Digital Out	B digital Output, CMOS		
20	DOUT-	Digital Out	Differential DOUT		
21	INCA	Digital Out	A Digital Output		
22	DOUT+	Digital Out	Differential DOUT		
23	DIN-	Digital In	Differential (for cascading)	Not Used.	
24	LERR	Digital Out	Error Pin (error=1, no error=0) CMOS		
25	DIN+	Digital In	Differential (for cascading)	Not Used.	
26	nRST	Digital Out	Chip Reset, CMOS, Internal Reset		
27	VDD	DC Supply		Connect to VDD	
28	SCL+	Digital In	Differential Clock +		
29	VDD	DC Supply		Connect to VDD	
30	SCL-	Digital In	Differential Clock		

While it is desirable to have the signals listed in Table 4 available for calibration and investigation purposes, once the encoder is physically aligned on its respective axis, only the SPI-related signals and several others will be needed for interfacing with the TMS320F28379D. Therefore, it is desirable to perform the calibration task with the minimum required effort and complexity.

Table 4 Pinout Provided with HEDS-8949 Alignment Kit to AEAT-9000-1GSH0

PIN	Name	Function
PIN 1	INCA	Incremental A Input
PIN 2	INCB	Incremental B Input
PIN 3		
PIN 4		
PIN 5	SCL+	SSI Shift Register Clock
PIN 6	SPI_SI	SPI Data Input
PIN 7	SPI_CLK	SPI Clock
PIN 8	NSL+	SSI Enable
PIN 9	LERR	Error pin
PIN 10	GND	IC Ground
PIN 11	Zero_RST	Zero Reset Input
PIN 12	DOUT	SSI Data Output
PIN 13	Vdd	IC supply
PIN 14	LocTest	Alignment Locate Signal
PIN 15	MSBINV	Inverted Counting Input
PIN 16		
PIN 17		
PIN 18	TiltOut	Tilt Alignment Output
PIN 19	DIN+	SSI Data Input
PIN 20	SPI_SO	SPI Data Output
PIN 21	Cosine+	Cosine+ Input
PIN 22	Sine+	Sine+ Input
PIN 23	Cosine-	Cosine- Input3
PIN 24	Sine-	Sine- Input4

4.1 Up and Running First Using an Arduino Mega 2560

The Arduino Mega 2560 is a particularly easy device to use and program. It has 54 digital I/O pins (15 able to function as PWM outputs) and 16 analog input pins. It should therefore be fairly easy to configure it to operate the AEAAT-9000-1GSH0 optical encoder. Taking this route avoids having to simultaneously come up to speed with the TMS320F28379D and the optical encoder since already have prior experience with the Arduino family.

The pinout of the optical encoder and its mapping to the Arduino Mega 2560 is tabulated in Table 5. Encoder commanding is addressed shortly.

Table 5 Optical Encoder I/O Mapping to Arduino Mega 2560

Encoder Pin Name	Pin #		Mega 2560 Pin Name	Dig/Ana	Mega I / O	Comments
COSINE+	1		A0	A	I	Truly differential
COSINE-	2		A1	A	I	Truly differential
SINE+	3		A2	A	I	Truly differential
SINE-	4		A3	A	I	Truly differential
TiltOut	5		22	D	I	Used for encoder/wheel tilt alignment. Two pulses per encoder revolution.
GND	6		Hard to GND			
LocTest	7		A4	A	I	Used for radial code wheel alignment
GND	8		Hard to GND			
Not Used	9		---			
MSBINV	10		23	D	O	
SPI_CLK	11		24	D	O	
Zero_RST	12		25	D	O	
SPI_SI	13		26	D	O	
NSL+	14		50	D	O	Connect to GND before encoder is powered on
SPI_SO	15		27	D	I	
NSL-	16		48			Connect to VDD before encoder is powered on
GND	17		Hard to GND			
GND	18		Hard to GND			
INCB	19		30	D	I	
DOUT-	20		47	D	I	
INCA	21		32	D	I	
DOUT+	22		33	D	I	
DIN-	23		38	D	O	
LERR	24		39	D	I	
DIN+	25		40	D	O	
nRST	26		41	D	O	Encoder reset
VDD	27		5V	PWR		
SCL+	28		42	D	O	
VDD	29		5V	PWR		
SCL-	30		43	D	O	

This plan of attack proved very beneficial because I ran into unexpected timing issues using the serial port routines with my C# GUI on the host PC host. I'm using an older version of Visual Studio C#, but even so, my research showed that the underlying issues were fundamental...like synchronous versus asynchronous. There was a fair amount of misinformation on the Internet, so it took some time to identify the underlying problems. In the end, I had to come up with my own serial read and write routines using only the BaseStream features of the C# software because functions like Port.BytesToRead were not reliable. More details can be found in §7.

A high-level block diagram of the hardware involved is shown in Figure 33. Once the basic encoder development work has been completed on the Arduino platform, the elevation electronics will take the form shown in Figure 34. Several useful high-level details are summarized below.

Optical Encoder:	5V, 94 mA
	17-bit precision
Mega2560:	3.3V input requires a minimum of 50 mA
DRV-8301:	6 – 24V operation
	10A RMS, 14A peak
	Built-in 1.5A regulator to supply 3.3V to Arduino
Motor:	42mm 3-phase motor
	Rated current 3.5A, peak current 10.6A
	Winding resistance 0.74Ω
	Rated voltage 24V

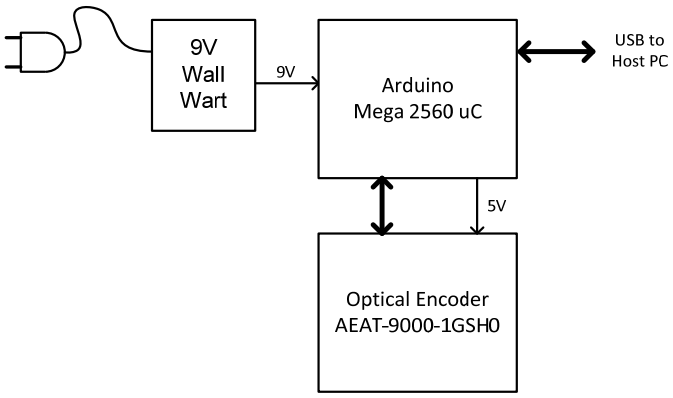


Figure 33 High-level hardware diagram for Arduino-based encoder work[8]

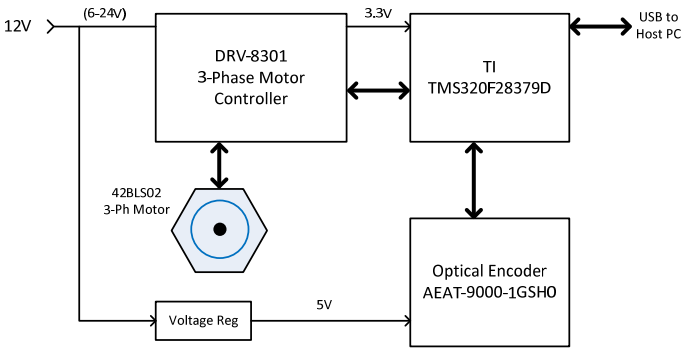


Figure 34 Ultimate direction for the elevation axis electronics

4.1.1 Software Interface

The first step involved with Figure 33 is to connect the Mega2560 Arduino board to the host computer and then load the Arduino program(presently optical_encoder_xface_11.ino) into its memory. This involves properly identifying the correct COM port for the communications to take place.

Secondly, the C# program is started on the host PC and the correct COM port and bit rate selected there. At present, the serial bit rate being used is 1.1152 Mbps as shown in the C# main screen in Figure 35.

[8] U26802 Diagrams.vsd.

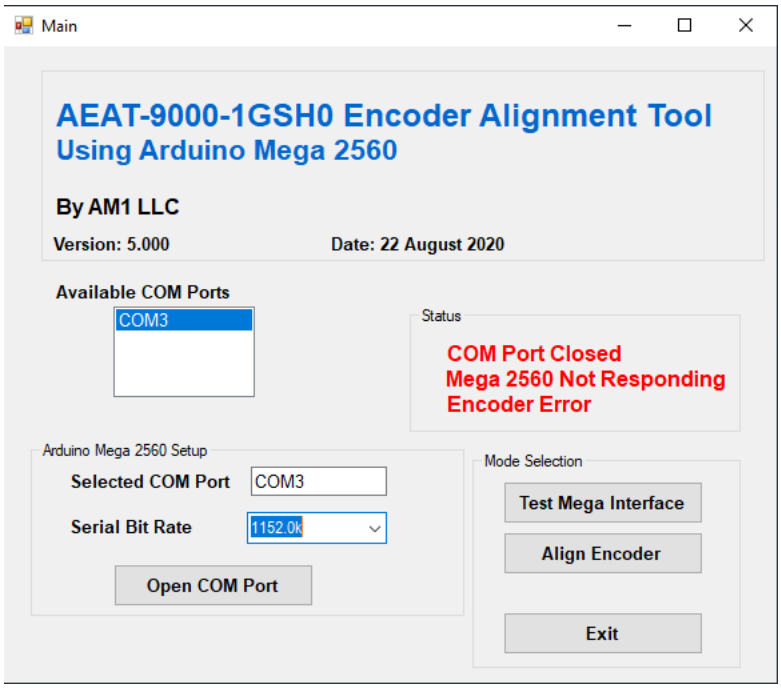


Figure 35 Main start-up page of the C# host program

Once the ***COM Port Open*** and ***Mega 2560 On-Line*** both green, the interface with the Mega can be tested using the "Test Mega Interface" button shown. An example result is shown in Figure 36

Figure 36 Mega interface test option using user-defined strings or random strings

Once proper operation has been demonstrated, the interface to the optical encoder by way of the Mega 2560 can be exercised using the "Align Encoder" button on the host computer's main screen. This launches the user screen shown in Figure 37.

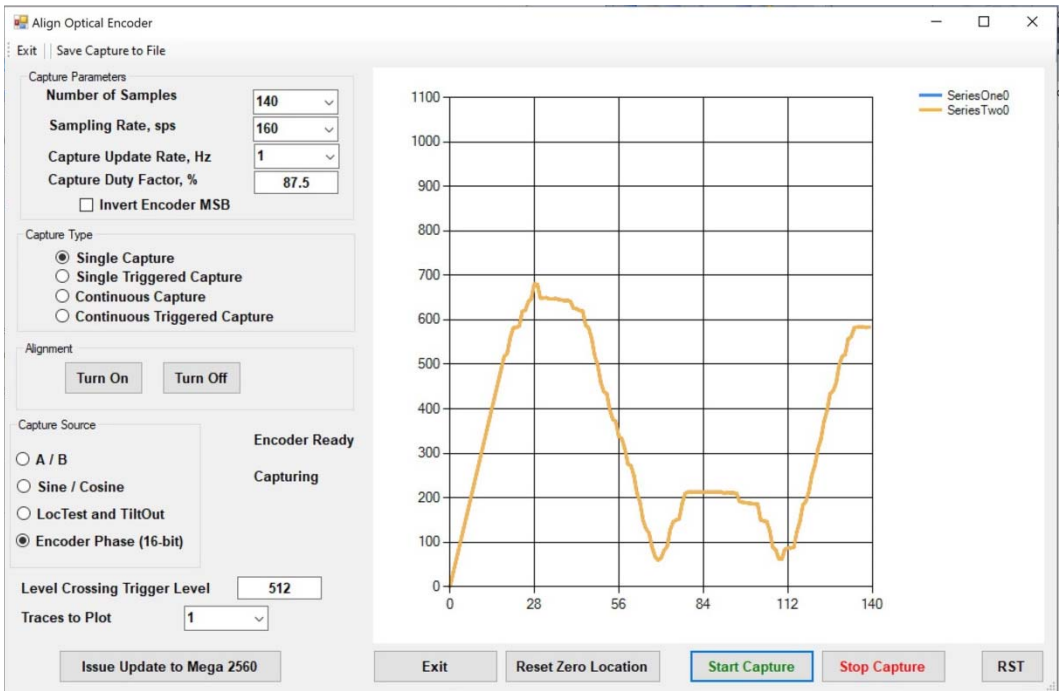


Figure 37 Optical encoder alignment screen. Optical encoder sampling rate and signal capture type are defined and up-loaded to the Mega 2560 using this screen. The sculpted waveform is caused by the jerkiness of the attached (open-circuited) 3-phase multiple-pole motor as the axle was turned manually by hand. The Arduino command details are described in the next sections.

4.1.2 Arduino State Diagram

All of the commands from the PC to the Arduino entail a serial string command, followed by a numerical value (as a string) for many of the commands. All of the commands echo back information to the PC upon receipt as this feature was very helpful for debugging purposes.

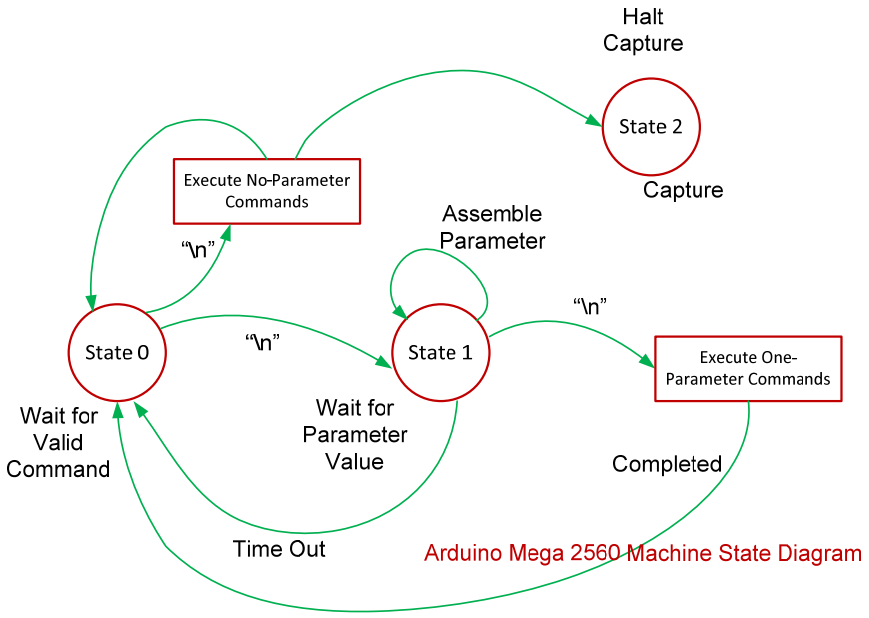


Figure 38 High-level state diagram for the Mega 2560 command interpreter

Table 6 PC Commands to Arduino

Command	Action
_echo	Echoes a random string from the PC back to the PC in order to check the USB communications path
_set_sample_rate	Dictates how frequently position information is to be sampled by the optical encoder
_set_capture_size	Number of samples per capture
_set_capture_update_rate	Rate at which encoder buffer (ultimately in the PC) is refreshed
_choose_signals	0: A/B samples 1: Sine/Cosine samples 2: LocTest & TiltOut 3: Digitized phase position
_set_capture_mode	0: Single capture 1: Single capture, triggered by level crossing 2: Continuous capture mode 3: Continuous capture mode, triggered by level crossing
_start_capture	Do single frame capture, or start continuous frame captures
_stop_capture	Stop capturing if in continuous capture mode
_reset_machine	Break out of measurement cycle; return to machine state 0
_msb_invert_true	Set MSB inversion in the encoder to true
_msb_invert_false	Set MSB inversion in the encoder to false
_restrart_timer	Setup timer parameters
_set_trigger_level	Set trigger level
_reset_encoder	Reset optical encoder
_alignment_on	Turn on optical encoder alignment mode
_alignment_off	Turn off optical encoder alignment mode
$	Reset Arduino entirely

Referring back to Figure 37, ***Capture Parameters, Capture Type,*** and ***Capture Source*** can all be defined and then uploaded to the Mega2560. A level-crossing trigger level can also be set as well as the number of signal capture curves which are to be overlaid on top of each other in the display. Capturing must be halted before any parameter update can be issued to the Mega 2560. Under proper operating conditions, the ***Encoder Ready*** and ***Capturing*** boxes will both show green.

4.1.3 Arduino SPI Communication with Encoder

Each I/O signal class identified in Table 5 is discussed in greater detail in the following subsections.

4.1.3.1 SPI Port of the Encoder

- Maximum SPI clock rate is 100 kHz
- SPI clock duty cycle $0.40 \leq \eta \leq 0.60$

SPI port timing for read and write cycles is shown in Figure 31. The address field (6 bits) comes first followed by the data field (8 bits).

During calibration mode, the SPI interface is used to perform sine/cosine gain and offset calibration. It is also used to program the EEPROM once the calibration has been done. This calibration has, however, supposedly been done at the factory so this calibration should not be subsequently required.

SPI_SO: SPI data output from encoder
SPI_SI: SPI data input to encoder
SPI_CLK: SPI clock input to encoder

Colloquial SPI terminology is shown in Figure 39. Comparing this figure to the interface signaling expected by the optical encoder Figure 31, the SPI needs to be set up with CPOL=1 and CPHA=1.

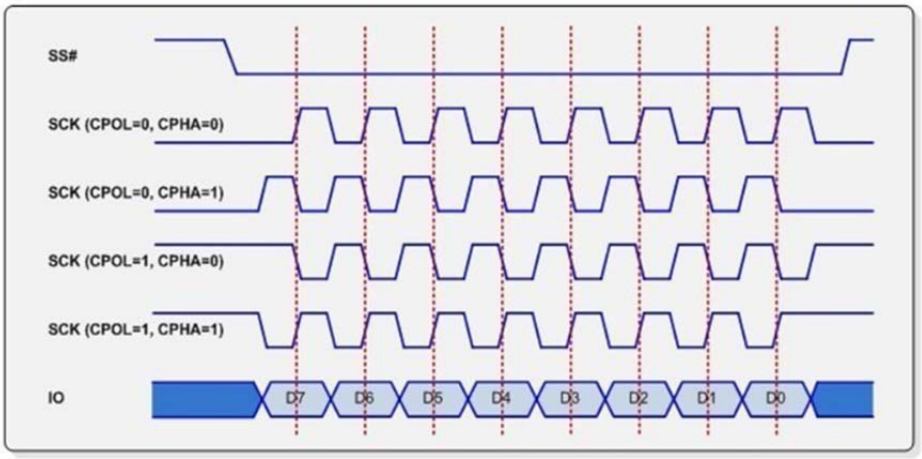


Figure 39 Colloquial SPI terminology. Optical encoder expects CPOL=1 and CPHA=1.

4.1.3.2 SPI Command: Alignment Mode

1. Write address 0x11 with 1010 1011 to unlock the associated encoder register.
2. Write address 0x10 with 0001 0001 to turn on the alignment mode.
3. Perform manual alignment of encoder to code wheel.
4. After alignment has been completed, write address 0x10 with 0000 0001.

The encoder's alignment signal (D1) will be output to the LocTest pin of the encoder. The signal's appearance will be as shown in Figure 40 and Figure 41.

For the sensor tilt alignment, the TiltOut signal pin is monitored. At the correct nominal position, the ratio *t/T* will be 0.0078 at any radial speed as shown in Figure 42.

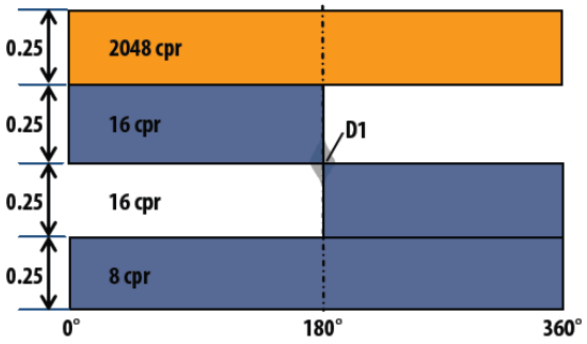


Figure 40 Code disk track 7 alignment to the diode D1 photo detector

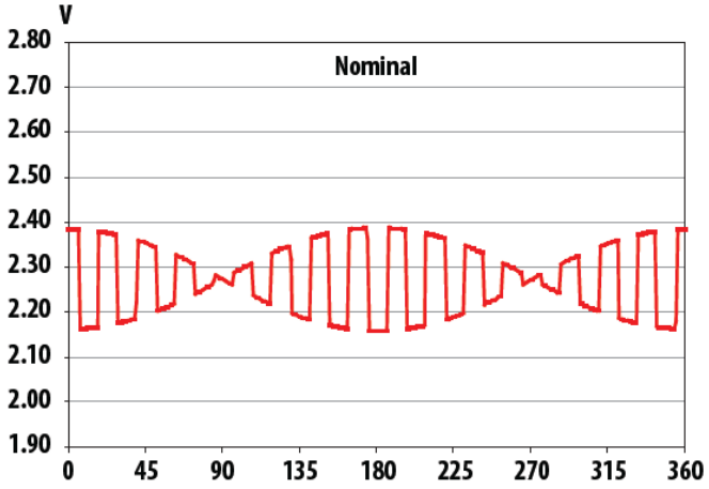


Figure 41 Internal alignment signal D1 as seen at the encoder output pin LocTest. Shown with code wheel eccentricity of 10 μm (0.4 thousandths of an inch)

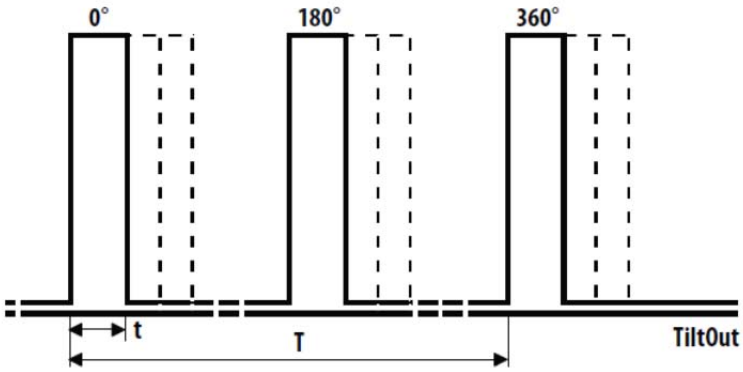


Figure 42 TiltOut signal ratio *t/T* should be 0.0078 when the encoder is properly aligned

4.1.4 Arduino Serial Port Communication with Encoder 2-Wire SSI

The absolute position is serially streamed out using the SSI protocol and timing as shown earlier in Figure 32. Each data bit is valid on the falling edge of the clock as shown.

4.1.5 Handling Encoder's Analog Signal Outputs

4.1.5.1 COSINE+, COSINE-, SINE+, SINE-

The SINE+, SINE-, COSINE+, and COSINE- signals are true differential signals.

- Nominal DC voltage is 2.5V
- Nominal voltage amplitude is 0.56V

4.1.5.2 LocTest

Use of this output signal is described in §4.1.3.2. The LocTest signal is a single-ended analog output with a nominal DC value of about 2.75V and the voltage swing of the varying signal portion is TBD and dependent upon the degree of radial misalignment between the encoder and code wheel.

4.1.6 2-Wire SSI Interface

The encoder's digital position is read using this digital interface as shown in Figure 32.

- DOUT+ is the digital data output signal
- SCL is the input digital clock signal

4.1.7 Other Encoder Digital Inputs

4.1.7.1 MSBINV

The MSBINV digital input provides the means to invert the encoder's counting.

4.1.7.2 Zero_RST

This signal is used to set the encoder's zero position at any position. The encoder stores this position value into its internal memory and all subsequent position readings are made with reference to this setting. The reset function is enabled when the Zero_RST pin is pulled to ground.

Table 7 Zero_RST

Signal Value	Action
VDD	Do Nothing
GND	Store present encoder position as the reference-zero position

4.1.8 Other Encoder Digital Outputs

4.1.8.1 TiltOut

This digital output is used as part of the encoder/code wheel alignment process as described in §4.1.3.2.

4.1.8.2 INCB and INCA

These signals are digital outputs with 2048 clocks per revolution. These A/B channels are generated from the differential sine and cosine internal signals of the encoder. As such, one signal will lead the other depending upon the direction of rotation.

4.1.8.3 LERR

This digital signal is normally a logic LOW. If the LED current becomes excessive, however, this pin is pulled to a logic HIGH. If the logic output is HIGH, this error or something else is wrong with the encoder.

4.1.9 Encoder Hardware Jig

A hardware jig was constructed to facilitate an easier working platform for the Arduino/Encoder work than the full-up telescope. This jig is shown in Figure 43 through Figure 45.

The most significant part of this jig is the encoder / encoder-wheel arrangement shown in Figure 45. I've been told that alignment between the two needs to be on the order of ±0.002" which is difficult to achieve. Denoting horizontal displacement as the x-direction and vertical displacement as the y-direction, the two gold-colored thumbscrews shown near the bottom of Figure 45 allow this alignment to be performed. Once this alignment has been achieved, the two large cap-screws shown just above the horizontal rubber band can be tightened down to anchor the alignment for good.

I have seen others use a 3-point alignment arrangement with good success as shown in Figure 46. In some respects this approach is conceptually more simple than my approach shown in Figure 45 because x- and y-direction movements are completely independent. The one down-side I see, however, is that it is difficult to precisely anchor the z-direction which will be at the whims of the set-screws butting up against the encoder's body.

The only way to adjust for Δx and or Δy misalignment in Figure 45 and Figure 47 is to adjust both gold thumbscrews such that tangent line T_1 lines up as shown. Knowing the relationships between the different parameters is important to ensure that sufficient travel range is accommodated within the jig. The solution is facilitated by using the geometric diagram shown in Figure 49.

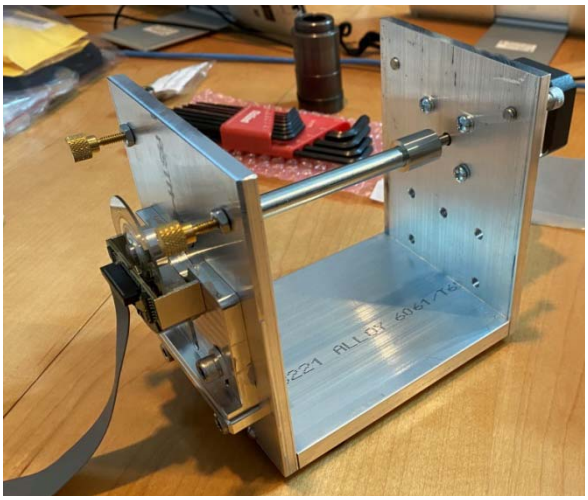


Figure 43 Encoder test jig side-view

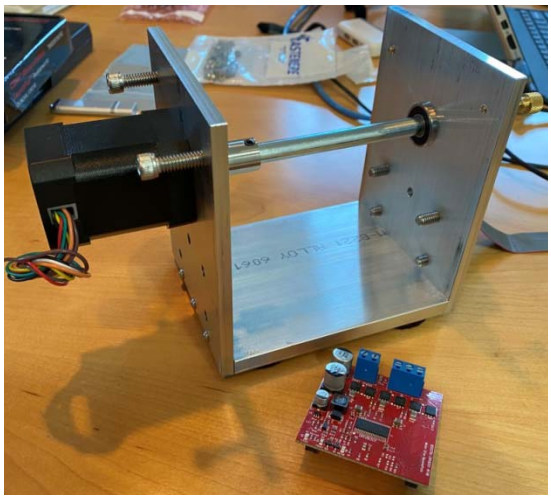


Figure 44 Opposite side view of encoder test jig with TI 3-phase motor driver (DRV-8301) in the foreground

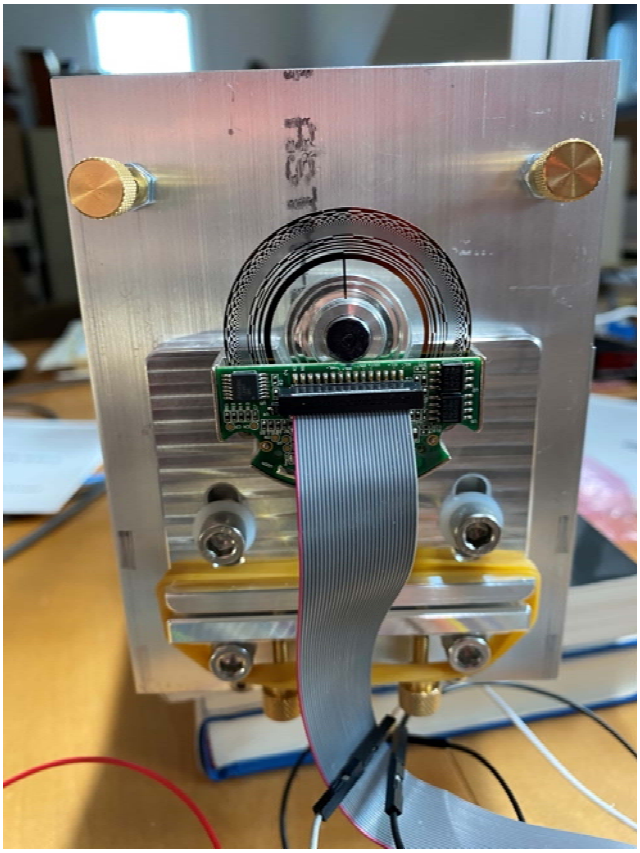


Figure 45 Key portion of the encoder jig showing the position-adjustment approach used to situate the encoder relative to the encoder wheel. The two gold thumbscrews at the bottom make it possible to adjust the (x,y) location and the tilt of the encoder with respect to the encoder wheel. The thumbscrews are 6-32 bolts thereby delivering about 0.0078” of adjustment per quarter-turn (required adjustment precision is about 0.002”).

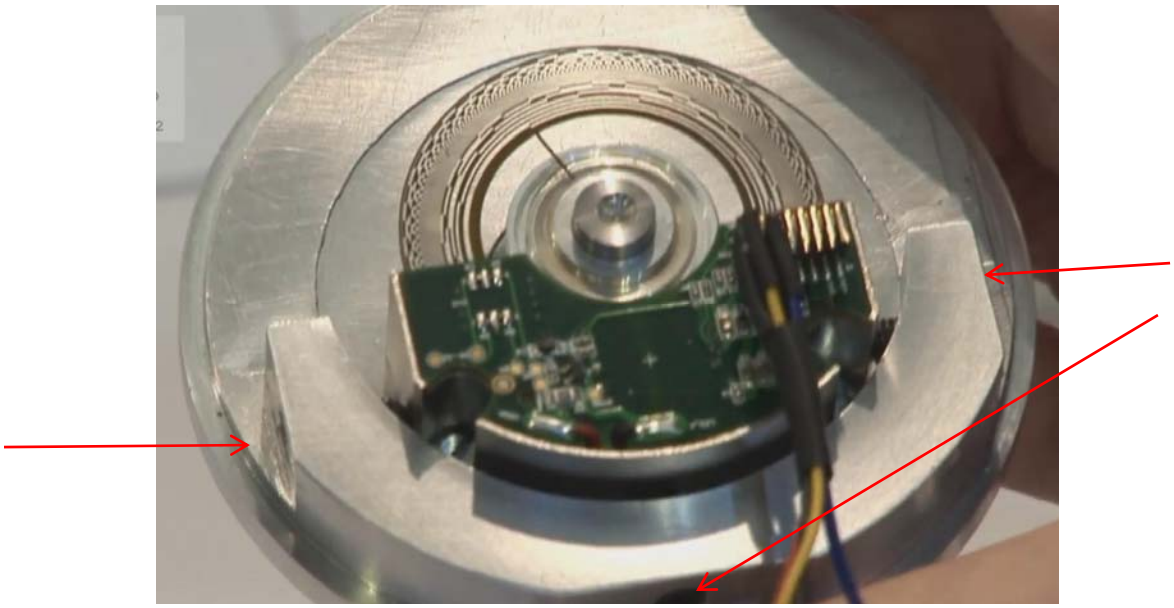


Figure 46 Three set-screw (see arrows) alignment arrangement adopted by FreeGo2 for AEAT9000 alignment

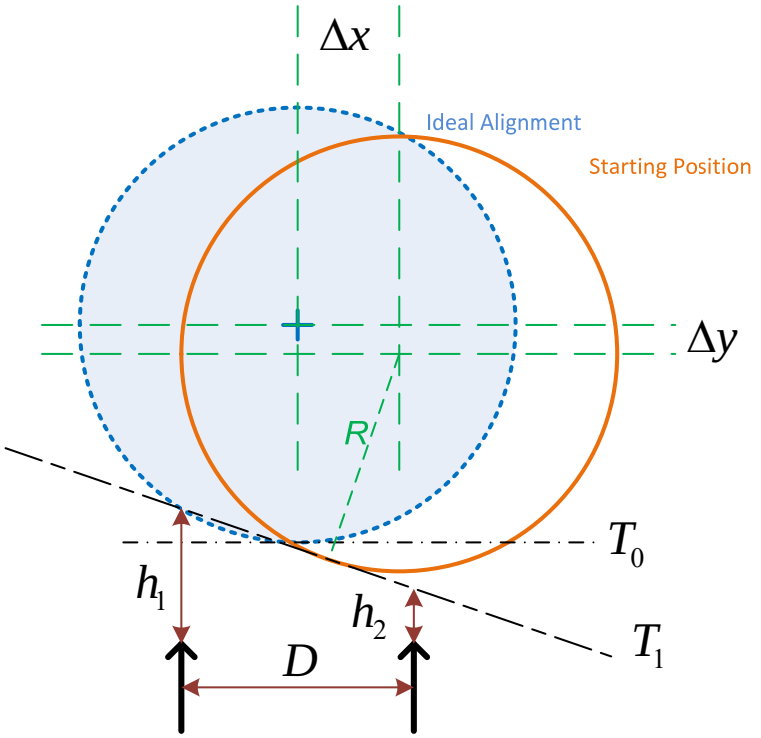


Figure 47 Encoder alignment geometry using the two-point adjustment method[9] of Figure 45. The two black arrows represent the gold adjustment screws shown near the bottom of Figure 45 whereas Δx and Δy represent the initial alignment errors of the code wheel relative to the encoder electronics. Distances h_1 and h_2 must be adjusted such that (i) the proper slope is obtained and (ii) the line T_1 is tangent to the circle.

[9] From U27233 Encoder Alignment.vsd.

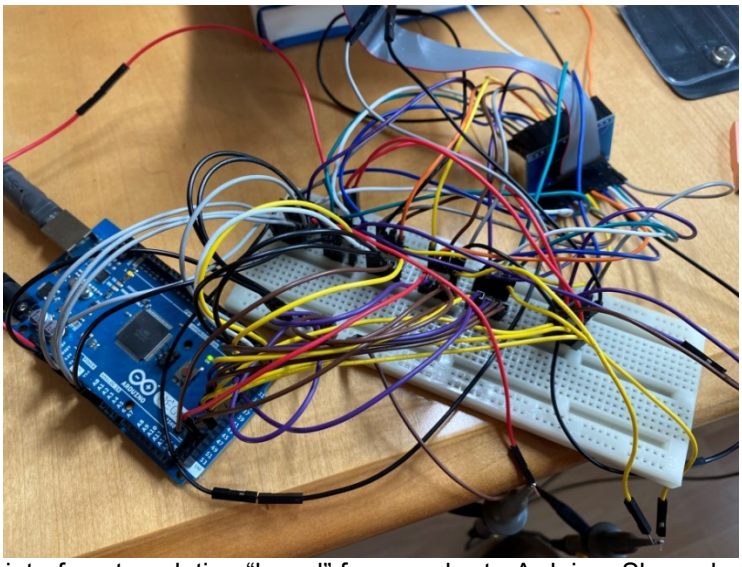


Figure 48 Make-shift interface translation "board" for encoder-to-Arduino. Sloppy but proved adequate for the low frequency signals involved.

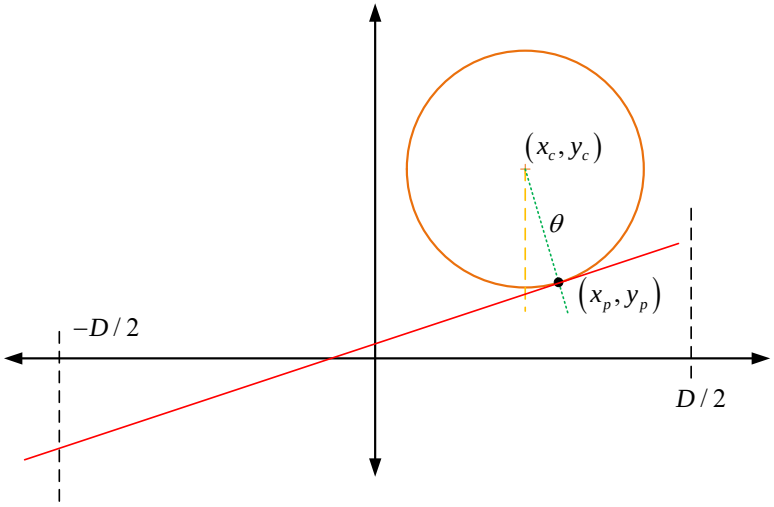


Figure 49 Geometric aid for problem solution

Any point on the circle's circumference can be represented parametrically as

$$\begin{aligned} x_p &= x_c + R\sin(\theta) \\ y_p &= y_c - R\cos(\theta) \end{aligned} \tag{8}$$

The slope of the circle at (x_p, y_p) can be found using implicit differentiation as

$$m = \frac{\frac{dy_p}{d\theta}}{\frac{dx_p}{d\theta}} = \frac{R\sin(\theta)}{R\cos(\theta)} = \tan(\theta) \tag{9}$$

Given the slope (9) and one point $\left(x_p, y_p\right)$ on the tangent line, the equation for the line is given by

$$\begin{aligned} y &= \left(x - x_p\right)\tan\left(\theta\right) + y_p \\ &= \left[x - x_c - R\sin\left(\theta\right)\right]\tan\left(\theta\right) + y_c - R\cos\left(\theta\right) \end{aligned} \tag{10}$$

From (10), it is straight forward to find the *y*-coordinates at $\pm D/2$ as

$$\begin{aligned} y_{-D/2} &= \left[-\frac{D}{2} - x_c - R\sin\left(\theta\right)\right]\tan\left(\theta\right) + y_c - R\cos\left(\theta\right) \\ y_{D/2} &= \left[\frac{D}{2} - x_c - R\sin\left(\theta\right)\right]\tan\left(\theta\right) + y_c - R\cos\left(\theta\right) \end{aligned} \tag{11}$$

From (11), the adjustment range for *y* is limited to

$$y_{-D/2} - y_{D/2} = -\frac{D}{2}\tan\left(\theta\right) \tag{12}$$

from which it also follows

$$\theta_{limit} = \pm\tan^{-1}\left(2\frac{y_{adjustment\_range}}{D}\right) \tag{13}$$

The most relevant observations primarily derive from (8) including:

- Errors in the y-direction are easily eliminated by adjusting both gold thumbscrews equally to eliminate the error; completely independent of θ
- Compensation for errors in the x-direction can be done by relatively small changes in θ since $\sin\left(\theta\right)$ is involved. Taking *R* = 0.80” and the error in *x* being 0.050”, a θ value of only 3.6° is involved. Since $R\cos\left(\theta\right)$ with this value of θ is still only 0.0016” away from the $\theta = 0$ value, errors in the *y*-dimension will dominate.

5 Plans for Next Project Installment

- Complete encoder alignment efforts
  - Substitute constant 100 RPM motor for the 3-phase motor in Figure 44 to facilitate calibration
- Transition AEAT-9000 encoder to work with TMS320 digital signal processor
  - Will require new desktop GUI written in C#
- Complete fabrication of elevation pulley-based step-down plus 3-phase motor
  - Fabricate pulley assembly
  - Mount 3-phase DC motor

6 References

1. J.A. Crawford, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft, Part II: Direct-Drive Motors,” U24933, 23 April 2019.
2. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft, Part I: Direct-Drive Motors,” U24933, 16 October 2017.
3. _____, “Photogrammetry for Non-Invasive Terrestrial Position/Velocity Measurement of High-Flying Aircraft, Part III: Direct-Drive Motor Chassis Design and Assembly,” U24933, 12 November 2019.
4. Texas Instruments, “LAUNCHXL-F28379D Overview, User’s Guide,” SPRUI77A, August 2017, U25038.
5. Avago Technologies, “AEAT-9000-1GSH0 Ultra-Precision 17-Bit Absolute Single Turn Encoder,” U26356.
6. _____, “Motor Drive BoosterPack Quick Start Guide: BOOSTXL-DRV8301,” U25064.
7. Matthew Piccoli and Mark Yim, “Cogging Torque Ripple Minimization via Position-Based Characterization,” U25085.
8. P.W. Poels, “Cogging Torque Measurement, Moment of Inertia Determination and Sensitivity Analysis of an Axial Flux Permanent Magnet AC Motor,” June 2008, U25099.
9. Hai-Lin Zhu, Hong-nen Wu, et al., “Minimal Tooth Number of Flexspline in Harmonic Gear Drive with External Wave Generator,” Gear Technology, Oct. 2013, U26858.
10. David G. Duff, “Harmonic Friction Drive,” US PAT 6,439,081, 27 Aug 2002.
11. Andrew Mosedale, “Understanding the Contact Ratio for Spur Gears with Some Comments on Ways to Read a Textbook,” U26987.
12. Nick Carter, “The Involute Curve, Drafting a Gear in CAD and Applications,” www.cartertools.com, U26966.
13. Antonio Acinapura, Gionata Fragomeni, et al., “Design and Prototyping of Miniaturized Straight Bevel Gears for Biomedical Applications*, Machines 2019*, U26981.

7 Appendix: C# Routines for Serial Port Communication with Arduino Mega2560

7.1 Declaration of the Serial Port in C#

```
public SerialPort myPort = new SerialPort();


myPort = new System.IO.Ports.SerialPort(comPort, baudRate, System.IO.Ports.Parity.None,
8, System.IO.Ports.StopBits.One);
```

7.2 Send String Message to Mega2560

```
public bool TxStringToMega(string mssg)
        {
            //
            // Send string mssg to the Mega 2560 processor via the serial USB port
            //
            if (!myPort.IsOpen)
            {
                Console.WriteLine("Mega serial port is not open");
                return (false);
            }
            //
            //    Send text string to Mega 2560 via USB/COM port
            //
            //  Must send over as a byte array.
            //
            int ii;
            int Lstr;
            Lstr = mssg.Length;

            byte[] cmdByteArray = new byte[Lstr+1];
            for (ii = 0; ii < Lstr; ii++)
            {
                cmdByteArray[ii] = (byte)mssg[ii];
            }
            cmdByteArray[Lstr] = 0x0;

            myPort.Write(cmdByteArray, 0, Lstr + 1);

            return (true);
        }
        //=============================================================================
```


7.3 Reading String Message from Mega2560

I set up an event handler to deal with the arrival of byte-level information from the Mega2560. Characters were assembled into strings with the usual end-of-line character denoting such.

```
public static void DataReceivedEventHandler(object sender, SerialDataReceivedEventArgs e)
        {
            SerialPort sp;
            bool loop = true;
            byte inbyte= 0;

            string buffer = "";

            sp = (SerialPort)sender;

            Program.new_line_ready = false;

            while (loop)
            {
                try
                {
                    inbyte = (byte)sp.BaseStream.ReadByte();
                }
                catch
                {
                    loop = false;
                }
                if ((int)inbyte == -1)
                {
                    loop = false;
                }
                else if (inbyte == 0 )
                {
                    Program.input_line = string.Copy(buffer);
                    Program.new_line_ready = true;
                    loop = false;
                    //Console.Write("==> {0}\n", Program.mySerialIO.myPort.BytesToRead);
                }
                else
                {
                    buffer = string.Concat(buffer, (char)inbyte);
                }
            }
         }

//============================================================================
```


8 Appendix: Arduino Code

The Arduino computer program used to act as a go-between the PC and the optical encoder is provided in its entirely in this section.

```
//
//  Arduino PC AEAT-9000 Optical Encoder Interface
//
//  Arduino Mega 2560
//
//  optical_encder_xface_11
//
//  Seems to be working reasonably well but interface with PC still gets
//  hung up on occassion.
//  TiltOut and LOCTEST still don't seem to be functioning yet. May not be
//  ultimately needed, but would still like to close that gap.
//
//  J.A. Crawford
//  22 August 2020
//
#define ledPin1 13
#define ledPin2 31

#define analogCos_plus  A0
#define analogCos_minus A1
#define analogSin_plus  A2
#define analogSin_minus A3
#define digitalTiltOut  22

#define analogLoc A4
#define MSBINV    23
#define SPI_CLK   24
#define Zero_RST  25
#define SPI_SI    26
#define NSL_plus  50
#define SPI_SO    27
#define NSL_minus 48
//
#define INCB      30
#define Dout_minus 47
#define INCA      32
#define Dout_plus  33
#define Din_minus  38
#define LERR      39
#define Din_plus   40
#define nRST      41
#define SCL_plus   42
#define SCL_minus  43

char inputBuffer[256];                  // Use to hold incoming string. Longer buffer (>64) dropped once-in-a-
while errors in string_test routine
String Command= "";
String Parameter="";
```



```
volatile int  bufferLength=0;
volatile bool stringComplete= false;    // Denote end of string encountered
volatile int  cmdIndex;
volatile int  parmValue;
volatile int  oldSampleValue= 0;

volatile int  maxBufferLength= 256;
volatile char endMarker= char(0);

volatile int  cmdState= 0;    // 0: Accumulating the command or waiting for command
                              // 1: Accumulating associated parameter
                              // 2: Execute non-measurement command
                              // 3: Ready to execute measurement command
                              // 4: Executing measurement command

volatile long xtalClock= 16000000;    // CPU clock, Hz

volatile int encoderSampleRate= 200;
volatile int encoderCaptureSize= 50;
volatile float encoderCaptureUpdateRate= 1.0;
volatile int encoderCaptureTimer= 0;
volatile int encoderSignalChoice= 0;  //  0 A/B
                                      //  1 sine/cosine
                                      //  2 LocTest & TiltOut
                                      //  3 Encoder phase
volatile int encoderCaptureMode= 0;   //  0 Single capture
                                      //  1 Single triggered capture
                                      //  2 Continuous capture
                                      //  3 Continuous triggered capture
volatile int encoderTriggerLevel= 500;
volatile bool invertMSB= false;
volatile bool doCapture= false;
volatile bool triggered= false;       // Must be true in order to do any captures
bool ledState= false;
bool diagnosticsOn= false;

int capCount= 0;    // Keep track of samples captured
String dummy;

//
//  Setup the Arduino Mega 2560
//
void setup()
{
  pinMode( ledPin1, OUTPUT );
  pinMode( ledPin2, OUTPUT );

  pinMode( analogCos_plus, INPUT );
  pinMode( analogCos_minus, INPUT );
  pinMode( analogSin_plus, INPUT );
  pinMode( analogSin_minus, INPUT );
  pinMode( digitalTiltOut, INPUT );
  pinMode( analogLoc, INPUT );
```



```
pinMode( MSBINV, OUTPUT );
pinMode( SPI_CLK, OUTPUT );
pinMode( Zero_RST, OUTPUT );
pinMode( SPI_SI, OUTPUT );
pinMode( SPI_SO, INPUT );
pinMode( INCB, INPUT );
pinMode( Dout_minus, INPUT );
pinMode( INCA, INPUT );
pinMode( Dout_plus, INPUT );
pinMode( Din_minus, OUTPUT );
pinMode( nRST, OUTPUT );

pinMode( SCL_plus, OUTPUT );
pinMode( SCL_minus, OUTPUT );

pinMode( NSL_plus, OUTPUT );
pinMode( NSL_minus, OUTPUT );

dummy.reserve(256);

cli();          // Temporarily disable all interrupts
//
//  Setup timers. Mega 2560 has 6 internal timers but don't need that many!
//  Header file iomxx0_1.h contains I/O definitions for 16-bit timers for Mega
//
//  Timer 0, 2 are 8-bit
//  Timer 1 is 16 bit
//  Timers 3, 4, 5 are 16-bit
//
//  Game Plan: Use only Timer3. Set it up so that interrupts are generated
//  at the sampling rate desired for the optical encoder. These ticks are counted
//  for as many samples as are needed for each capture.
//
//  TCCRx       Timer/counter control register
//  TCNTx       Timer/count value
//  OCRx        Output compare register
//  ICRx        Input compare register (only for 16-bit timers)
//  TIMSKx      Interrupt mask register
//  TIFRx       Interrupt flag register, indicates pending timer interrupt
//
//  Will only use Timer3 to create a tick-rate equal to the encoder sampling rate
//
//  Waveform frequency is given by
//        16e6 / (2*256) / (1 + OCR3A) = 31250/(1 + OCR3A)
//
TCCR3A= 0;
TCCR3B= 0;

TCCR3B |= (1<<WGM12);   // Mode 4, CTC
TCCR3B |= (1<<CS32) | (1<<CS30);    // Prescaler = 1024
OCR3A= 7812;            // About 1 second tick

sei();   // Enable all interrupts

//
```



```
  //  Initialize both serial ports (Mega 2560)
  //
  Serial.begin(1152000);   // Use for comm with PC C# program
  Serial.flush();
  delay(10);

  digitalWrite( nRST, HIGH );
  delay(10);
  digitalWrite( nRST, LOW );
  delay(10);
  digitalWrite( nRST, HIGH );

  digitalWrite( SPI_SI, LOW );
  digitalWrite( SPI_CLK, HIGH );

  digitalWrite( SCL_plus, HIGH );
  digitalWrite( SCL_minus, LOW );

  digitalWrite( Zero_RST, HIGH );
  delayMicroseconds(50);
  digitalWrite( Zero_RST, LOW );

  digitalWrite( NSL_plus, HIGH );
  digitalWrite( NSL_minus, LOW );
  digitalWrite( NSL_minus, HIGH );
}
//================================================================================

void loop()
{
  int jk;
  int cntr;
  int nticks;
  //
  //  Check to see if a new string is available
  //
  if( stringComplete )
  {
    //
    //  Parse command details
    //
    //  Command sent in first line; parameter (if needed) is sent secondly
    //                          Cmd
    //  _echo                   10    Send string p1 back to PC
    //  _set_sample_rate        11    Set sampling rate for the optical encoder
    //  _set_capture_size       12    Number of samples per capture
    //  _set_capture_update_rate  13    Rate at which encoder buffer is refreshed
    //  _choose_signals         14    0 = A/B samples
    //                                1 = sine/cosine samples
    //                                2 = LocTest
    //                                3 = digitized position
    //  _set_capture_mode       15    0 = single capture mode
    //                                1 = single triggered capture mode
    //                                2 = continuous capture mode
    //                                3 = continuous triggered capture mode
```



```
//  _start_capture            16    Do single frame capture, or start continuous frame captures
//  _stop_capture             17    Stop capturing if in continuous capture mode
//  _reset_machine            18    Break out of measurement cycle; return to machine state 0
//  _reset_encoder_zero       19    Reset zero position of the encoder
//  _msb_invert_true;         20    Set MSB inversion in encoder to true
//  _msb_invert_false         21    Set MSB inversion in encoder to false
//  _restart_timer            22    Setup timer parameters
//  _set_trigger_level        23    Set trigger level
//  $                         24    Reset Arduino
//  _reset_encoder;           25    Reset optical encoder
//  _alignment_on             26    Turn on optical encoder alignment mode
//  _alignment_off            27    Turn off optical encoder aligment mode
//
//  Parse the command portion out
//
if( cmdState == 0 )
{
  //
  //  Accumulate buffer elements into a command string
  //
  Command= assembleString( inputBuffer );
  //
  //  Parse command from inputBuffer
  //
  if( Command == "_echo" )
  {
    cmdIndex= 10;
    cmdState= 1;       // State to look for test string parameter
  }
  else if(Command == "_set_sample_rate")
  {
    cmdIndex= 11;
    cmdState= 1;      // State to look for test string parameter
  }
  else if( Command == "_set_capture_size")
  {
    cmdIndex= 12;
    cmdState= 1;      // State to look for test string parameter
  }
  else if( Command == "_set_capture_update_rate" )
  {
    cmdIndex= 13;
    cmdState= 1;
  }
  else if( Command == "_choose_signals" )
  {
    cmdIndex= 14;
    cmdState= 1;
  }
  else if( Command == "_set_capture_mode" )
  {
    cmdIndex= 15;
    cmdState= 1;
  }
  else if( Command == "_start_capture" )
```



```
{
  cmdIndex= 16;
  cmdState= 0;    // Out of sequence

  doCapture= true;

  if( (encoderCaptureMode==0) || (encoderCaptureMode==2))
  {
    triggered= true;
  }
  else
  {
    triggered= false;
  }

  capCount= 0;
  if( diagnosticsOn ) sendStringToPC("Capture STARTED");
}
else if( Command == "_stop_capture" )
{
  cmdIndex= 17;
  cmdState= 0;    // Out of sequence

  doCapture= false;
  triggered= false;
  Serial.flush();

  ledState= false;
  digitalWrite( ledPin1, false );
  digitalWrite( ledPin2, false);
  if( diagnosticsOn ) sendStringToPC("Capture STOPPED");
}
else if( Command == "_reset_machine" )
{
  cmdIndex= 18;
  cmdState= 0;
  doCapture= false;
}
else if( Command == "_reset_encoder_zero" )
{
  cmdIndex= 19;
  cmdState= 0;

  digitalWrite( Zero_RST, LOW );
  delay(10);
  digitalWrite( Zero_RST, HIGH );
  delayMicroseconds(50);
  digitalWrite( Zero_RST, LOW );

  if( diagnosticsOn ) sendStringToPC( "Encoder Zeroed" );
}
else if( Command =="_msb_invert_true")
{
  invertMSB= true;
```



```
  digitalWrite( MSBINV, HIGH );

  cmdIndex= 20;
  cmdState= 0;

  if( diagnosticsOn ) sendStringToPC("MSB Invert TRUE");
}
else if( Command == "_msb_invert_false")
{
  invertMSB= false;

  cmdIndex= 21;
  cmdState= 0;

    digitalWrite( MSBINV, LOW );

  if( diagnosticsOn ) sendStringToPC("MSB Invert FALSE");
}
else if( Command == "_set_timer" )
{
  cmdIndex= 22;
  cmdState= 0;

  //
  //  Set prescaler value to 64
  //  resulting in a tick-rate of 250 kHz
  //  and minimum rate with 16-bit timer of
  //  3.814697 Hz
  //
  //  Waveform Generation Mode = 4
  //
  cli();          // Disable interrupts
  TCCR3A= 0;
  TCCR3B= 0;

  TCCR3B |= (1<<WGM12);   // Mode 4, CTC
  TCCR3B |= (1<<CS32) | (1<<CS30);    // Prescaler = 1024

  //
  //
  nticks= (int)( (long)15625/encoderSampleRate);
  OCR3A= nticks;
  TIMSK3= (1<<OCIE3A);    // Enable compare A match interrupt
  sei();                  // Enable all interrupts
  //
  //  Compute timer parameters and initialize
  //
  if( diagnosticsOn ) sendStringToPC( "Resetting timer" );
}
else if( Command == "_set_trigger_level" )
{
  cmdIndex= 23;
  cmdState= 1;
}
else if( Command == "$" )
```



```
{
  cmdIndex= 24;
  cmdState= 0;
}
else if( Command == "_reset_encoder" )
{
  cmdIndex= 25;
  cmdState= 0;

  digitalWrite( nRST, HIGH );
  delay(10);
  digitalWrite( nRST, LOW );
  delay(5);
  digitalWrite( nRST, HIGH );
}
else if( Command == "_alignment_on" )
{
  cmdIndex= 26;
  cmdState= 0;

  digitalWrite( NSL_plus,LOW);
  digitalWrite( NSL_minus, HIGH );
  delayMicroseconds(50);

  digitalWrite( SPI_CLK, LOW );
  delayMicroseconds(50);
  shiftOut( SPI_SI, SPI_CLK, MSBFIRST, 0x51 );
  delayMicroseconds(50);
  shiftOut( SPI_SI, SPI_CLK, MSBFIRST, 0xab );
  digitalWrite( SPI_CLK, LOW );

  digitalWrite( NSL_plus, HIGH );
  digitalWrite( NSL_minus, LOW );
}
else if( Command == "_alignment_off" )
{
  cmdIndex= 27;
  cmdState= 0;

  digitalWrite( SPI_CLK, LOW );
  delayMicroseconds(50);
  shiftOut( SPI_SI, SPI_CLK, MSBFIRST, 0x50 );
  delayMicroseconds(50);
  shiftOut( SPI_SI, SPI_CLK, MSBFIRST, 0x11 );
  digitalWrite( SPI_CLK, LOW );
}
else
{
  //
  //  Error condition...false command, so drop it
  //
  cmdIndex= 0;
  //
  //  Still send back to PC
  //
```



```
      bufferLength= 0;
      stringComplete= false;
      doCapture= false;
      if( diagnosticsOn ) Serial.print( "Command Error" );
    }
    bufferLength= 0;
    stringComplete= false;
  }
  else if( cmdState == 1 )
  {
    //
    //  Parameter for command received.
    //  If not a measurement command, complete command execution right here
    //
    switch( cmdIndex )
    {
      case 10:    // Echo string back to PC
        //
        //  Send test string back to PC
        //
        Parameter= assembleString( inputBuffer );

        sendStringToPC( Parameter );
        cmdState= 0;
        break;
      case 11:    // Set Encoder Sample Rate
        Parameter= assembleString( inputBuffer );
        encoderSampleRate= Parameter.toInt();

        Parameter= Parameter + ":Encoder Sample Rate";
        if( diagnosticsOn ) sendStringToPC( Parameter );
        cmdState= 0;
        break;
      case 12:    // Set Capture Size
        Parameter= assembleString( inputBuffer );
        encoderCaptureSize= Parameter.toInt();

        Parameter= Parameter + ":Capture Size";
        if( diagnosticsOn ) sendStringToPC( Parameter );
        cmdState= 0;
        break;
      case 13:    // Set Capture Update Rate
        Parameter= assembleString( inputBuffer );
        encoderCaptureUpdateRate= Parameter.toFloat();

        Parameter= Parameter + ":Capture Update Rate";
        if( diagnosticsOn ) sendStringToPC( Parameter );

        encoderCaptureTimer=                                                    (int)floor(
(double)encoderSampleRate/(double)encoderCaptureUpdateRate + 0.50 );
        //Serial.println( String(encoderCaptureTimer) );

        cmdState= 0;
        break;
      case 14:    // Choose signals
```

```
      Parameter= assembleString( inputBuffer );
      encoderSignalChoice= Parameter.toInt();

      Parameter= Parameter + ":Signal Choice";
      if( diagnosticsOn ) sendStringToPC( Parameter );
      cmdState= 0;
     break;
    case 15:    // Set capture mode
      Parameter= assembleString( inputBuffer );
      encoderCaptureMode= Parameter.toInt();

      Parameter= Parameter + ":Encoder Capture Mode";
      if( diagnosticsOn ) sendStringToPC( Parameter );
      cmdState= 0;

      if( (encoderCaptureMode==0) || (encoderCaptureMode==2) )
      {
       triggered= true;
      }
     break;
    case 16:      // A no-parameter command
     break;
    case 17:      // A no-parameter command
     break;
    case 18:      // A no-parameter command
     break;
    case 19:      // Reset Encoder & Reset Zero Location
     break;
    case 20:      // A no-parameter command
     break;
    case 21:      // A no-parameter command
     break;
    case 22:      // A no-parameter command
     break;
    case 23:  //  Set trigger level
      Parameter= assembleString( inputBuffer );
      encoderTriggerLevel= Parameter.toInt();

      Parameter= Parameter + ":Trigger Level";
      if( diagnosticsOn ) sendStringToPC( Parameter );
      cmdState= 0;
      break;
    case 24:      // A no-parameter command
      break;
    case 25:      // A no-parameter command
      break;
    case 26:      // A no-parameter command
      break;
    case 27:      // A no-parameter command
      break;
    default:
      break;
  }
  bufferLength= 0;
  stringComplete= false;
```



```
    }
  }
}
//================================================================

void sendStringToPC( String xx )
{
  int cntr= 0;

  while( !Serial.availableForWrite() && (cntr < 1000) )
  {
    cntr++;
  }
  Serial.print(xx);
}
//================================================================

String assembleString( char *inx )
{
  String workString;
  int  jj;

  //
  //  When assembling the string, make sure not to have more
  //  than one endMarker
  //
  workString= "";
  jj=0;
  while( (jj<bufferLength) && ( *(inx+jj) != endMarker) )
  {
    workString= workString + *(inx+jj);
    ++jj;
  }
  workString= workString + endMarker;

  stringComplete= false;
  bufferLength= 0;

  return(workString);
}
//================================================================
//
//  SerialEvent occurs whenever a new data item comes into the hardware serial RX.
//  This routine is run between each time through the loop() running. Multiple bytes
//  of data may be available. This routine accumulates the bytes into a string until
//  such time as (i) a maximum length is accumulated, or (ii) the end-of-line character
//  is encountered.
//
//  Once a command string has been accumulated, it is acted upon.
//
void serialEvent()
{
  while( Serial.available() && !stringComplete )
  {
    //
```

```
    //  Get the new byte
    //
    byte inByte;
    inByte= byte(Serial.read());
    //
    //  Add this to the inputBuffer
    //
    inputBuffer[bufferLength++]= inByte;
    //
    //  Check to see if the newly received character is an end-of-string character
    //  Variable bufferLength is the length of the string plus 1 more for '\n'
    //
    if( inByte == endMarker )
    {
      //
      //  Quantity in inputBuffer is the new input string plus '\n'
      //
      stringComplete= true;
    }

    if( bufferLength >= maxBufferLength )
    {
      bufferLength= 0;
      Serial.println( "Buffer length exceeded in Mega");
      stringComplete= false;
    }

    if( inByte == '$' )
    {
      cmdState= 0;
      cmdIndex= 0;
      stringComplete= false;
      bufferLength= 0;
      Serial.flush();
    }
  }
}
//=============================================================================

ISR( TIMER3_COMPA_vect )    // Interrupt service routine
{
  char data[100];
  String temp;
  int ii;
  int ij;

  int v1, v2;
  unsigned long vun;
  unsigned long mask;

  //
  //  Heart beat at sample rate
  //
  if( doCapture )
  {
```

```
  ledState= !ledState;
  digitalWrite( ledPin1, ledState );
  digitalWrite( ledPin2, !ledState );
 }

 if( doCapture == false )
 {
  capCount= 0;
 }
 //
 //  Check to see if any more samples are needed
 //  to fill out this one capture frame
 //
 if( doCapture )
 {
  switch( encoderSignalChoice)
  {
    case 0:
      v1= 256*digitalRead(INCA);
      v2= 256*digitalRead(INCB);
      break;
    case 1:
      v1= analogRead(analogCos_plus) - analogRead(analogCos_minus) + 512;
      v2= analogRead(analogSin_plus) - analogRead(analogSin_minus) + 512;
      break;
    case 2:
      v1= 2*(analogRead(analogLoc) - 460) + 512;
      v2= digitalRead(digitalTiltOut)*256;
       break;
    case 3:
      //
      //  Use 2-wire SSI to retrieve 16-bit (of available 17 bits) value
      //
      vun= 0;

      digitalWrite( NSL_plus, LOW );
      digitalWrite( NSL_minus, HIGH );
      //delayMicroseconds(50);
      digitalWrite( NSL_plus, LOW );
      digitalWrite( NSL_minus, HIGH );

      digitalWrite( SCL_plus, HIGH);
      digitalWrite( SCL_minus, LOW);
      //delayMicroseconds(50);
      digitalWrite( SCL_plus, HIGH);
      digitalWrite( SCL_minus, LOW);
      digitalWrite( SCL_plus, HIGH);
      digitalWrite( SCL_minus, LOW);

      //digitalWrite( SCL_minus, HIGH );

      for (ii=1; ii<=17; ii++)
      {
        digitalWrite( SCL_plus, LOW);
        digitalWrite( SCL_minus, HIGH);
```



```
    //delayMicroseconds(50);
    digitalWrite( SCL_plus, LOW);
    digitalWrite( SCL_minus, HIGH);
    //
    if( ii <= 17 )
    {
      if( digitalRead( Dout_plus ) == HIGH )
      {
        vun= vun*2 + 1;
      }
      else
      {
        vun= vun*2 + 0;
      }
    }
    else
    {
      ij= digitalRead( Dout_plus );
      vun= vun + 0;
    }
    //
    digitalWrite( SCL_plus, HIGH);
    digitalWrite( SCL_minus,LOW);
    //delayMicroseconds(50);
    digitalWrite( SCL_plus, HIGH);
    digitalWrite( SCL_minus,LOW);
  }
  digitalWrite( NSL_plus, HIGH );
  digitalWrite( NSL_minus, LOW );

  //
  //  Do Gray code to binary conversion
  //
  if( false )
  {
    mask= vun;
    ii=31;
    while( ( mask != 0 ) && ( ii>=0 ) )
    {
      mask >> 1;

      vun= vun|mask - vun&mask;
      ii=ii-1;
    }
  }
  v1= (int)(vun/256);
  v2= (int)( vun - 256*v1 );

  digitalWrite( SCL_plus, HIGH );
  digitalWrite( SCL_minus, LOW );

  //dummy= dummy + "Encoder Phase";
  break;
 default:
  dummy= dummy + "ERROR";
```



```
      break;
    }
    if( !triggered )
    {
     if(                          ((v1<=encoderTriggerLevel)&&(v2>=encoderTriggerLevel))                          ||
((v1>encoderTriggerLevel)&&(v2<=encoderTriggerLevel) ) )
     {
      triggered= true;
     }
    }
  }
  //
  //
  if( doCapture && triggered && (capCount < encoderCaptureSize ) )
  {
    sprintf( data, "%4d%7d%7d\n", capCount, v1, v2 );

    v1= 0;
    v2= 0;
    temp= (String)data;
    temp= temp + endMarker;
    sendStringToPC(temp);
  }
  //
  //
  if( doCapture && triggered )
  {
    ++capCount;

    //
    //  Check on capCount
    //
    if( capCount >= encoderCaptureTimer )
    {
      capCount= 0;
      //
      //
      if( encoderCaptureMode < 2 )
      {
        doCapture= false;
        triggered= false;
      }
    }
  }
}
//=============================================================================
```


9 Appendix: Arduino Mega 2560

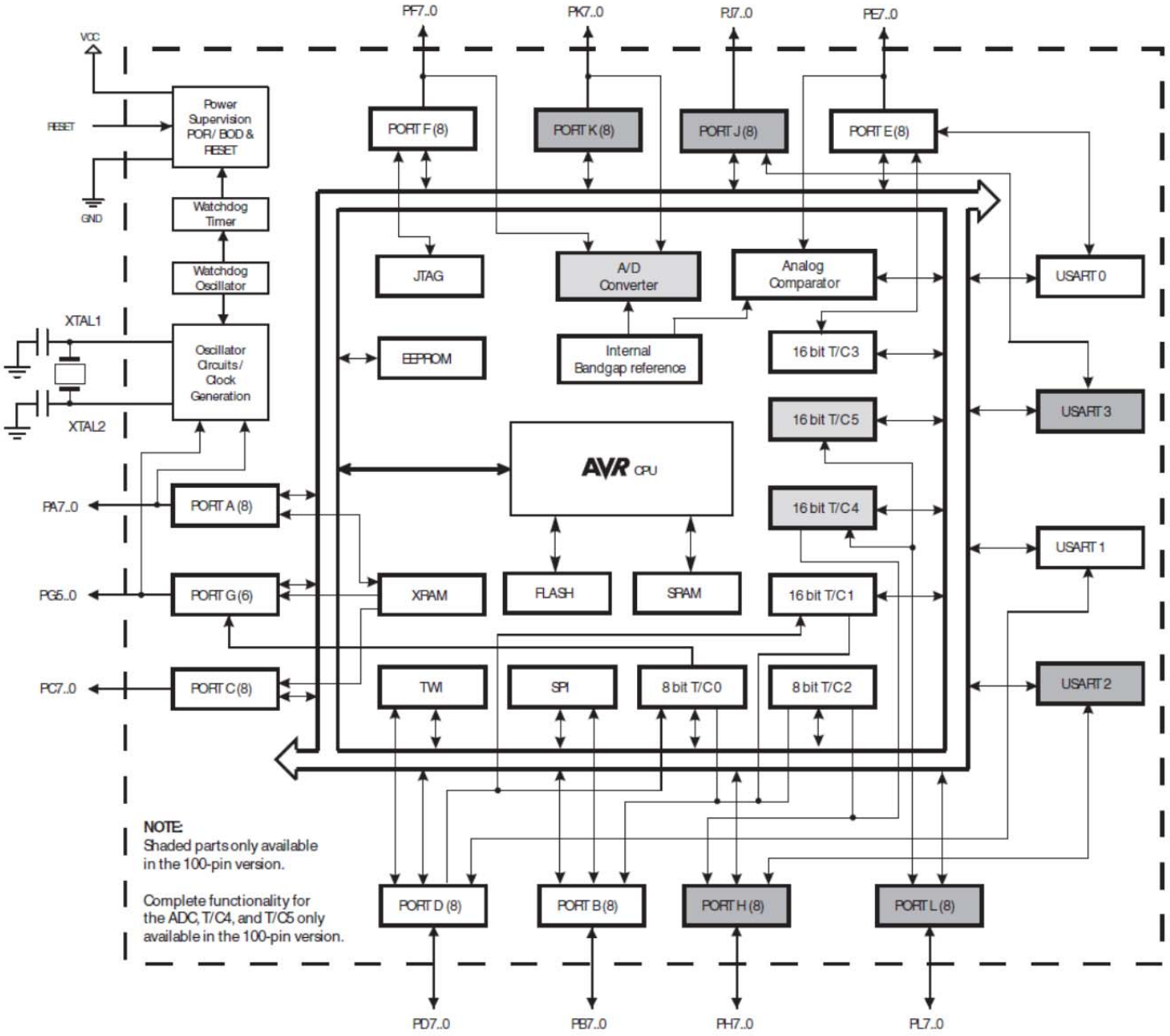


Figure 50 Internal block diagram[10] of the ATmega640/1280/1281/2560/2561 low-power CMOS 8-bit microcontroller from Atmel

[10] Atmet ATmega640/.../V-2560/... 8-bit Microcontroller, U26671.

10 Appendix: Elevation Drive Motor

A much smaller motor was chosen for the elevation drive portion of the project for physical size/weight as well as electrical power reasons. As such, its stand-alone torque capabilities were inadequate for the project and a torque-multiplication by way of a mechanical means (e.g., gears) subsequently required. The need for zero backlash prevailed of course, so multiple avenues were explored. The approach described here is based upon pairs of timing-belt pulleys.

10.1 Torque Requirements

Explicit torque requirements are rather elusive since the telescope system will be purposely balanced, growth for presently unknown telescope loads needs to be accommodated, and torque primarily dictates only how quickly the telescope load can be positioned with is rather arbitrary as well. The purchased brushless 3-phase motor (is shown in) is part number 42BLS02 (shown in Figure 51) with the following key parameters:

Rated Torque = 0.125 Nm
Stall Torque= 0.15 Nm
Rated RPM = 4000
Rated Power (@ Rated Torque and Speed)= 52.4 W

Figure 51 The 42BLS02 brushless 3-phase DC motor for elevation drive

This information leads to the computed stall torque in terms of the more familiar foot-pounds as shown below.

$$0.125Nm \times 0.2248\frac{lb}{N} \times 3.2808\frac{ft}{m} = 0.0922\ \ ft\,lb \tag{14}$$

$$RatedPower = 0.0922\ 2\pi \frac{RPM}{60\,\text{sec}} = 38.617 \frac{ft\ lb}{\text{sec}}$$
$$= 38.61738.617 \frac{ft\ lb}{\text{sec}} \times \frac{1}{0.73756 \frac{ft\ lb}{W}} = 52.36W \quad (15)$$

$$StallTorque = 0.15Nm = 0.1106\ ft\ lb \quad (16)$$

Given the size of telescope loads envisioned, even if carefully balanced, the computed stall torque is unacceptably small. A value between 20x and 25x is deemed more advisable.

10.2 Torque Multiplication

A mechanical means for torque multiplication is required for the elevation drive. Although the adopted motor can deliver substantial torque, the torque is tied to the rotational speed which for the elevation drive will usually be zero. Several different mechanical means were described in §3 and the cascaded pulley approach is considered here.

In the pulley system envisioned here, a cascade of a 5x and 5x (or 4x) pulley-reduction steps will be used. A single pulley reduction step is shown in Figure 52. Timing belt/pulley combinations are used so the reduction factor is given by the ratio of pulley diameters (or equivalently the ratio of pulley teeth between the two pulleys).

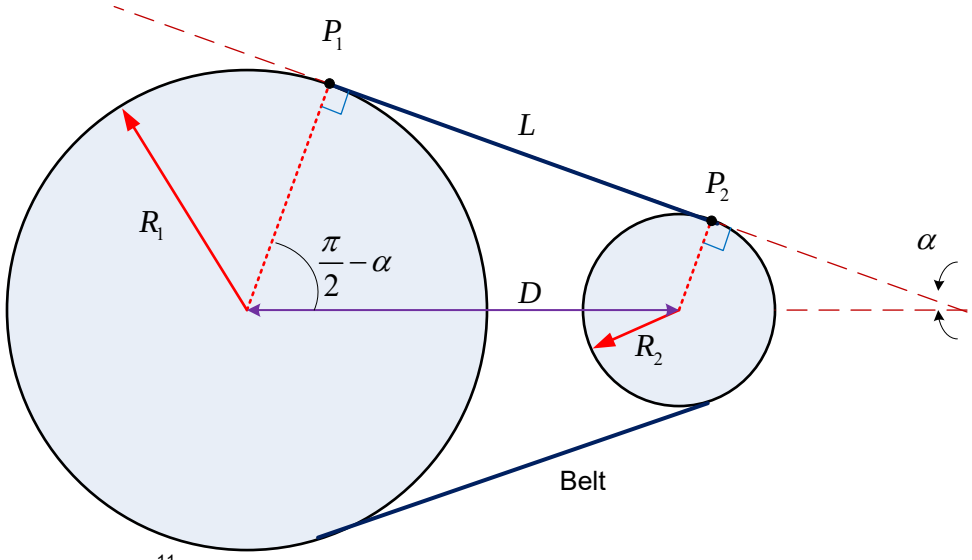


Figure 52 Pair of pulleys[11]

A bit of trigonometry with Figure 52 gives

$$L = \sqrt{D^2 - \left(R_1 - R_2\right)^2} \quad (17)$$

with the angle α following as

[11] From U27003 Pulley System.vsd.

$$\alpha = \tan^{-1}\left(\frac{R_1 - R_2}{L}\right) = \tan^{-1}\left[\frac{R_1 - R_2}{\sqrt{D^2 - (R_1 - R_2)^2}}\right] \tag{18}$$

This angle plays a role in determining what proportion of each pulley's circumference is engaged with the belt at all times. From this, the tangent points P_1 and P_2 in the figure are given in vector (x,y) form as

$$P_1 = \left[R_1 \cos\left(\frac{\pi}{2} - \alpha\right), R_1 \sin\left(\frac{\pi}{2} - \alpha\right)\right] \tag{19}$$

$$P_2 = \left[D + R_2 \cos\left(\frac{\pi}{2} - \alpha\right), R_2 \sin\left(\frac{\pi}{2} - \alpha\right)\right] \tag{20}$$

where the center of the larger pulley is assumed to be situated at the origin (0,0). These formula were used to construct the nomographs shown in Figure 53 and Figure 54.

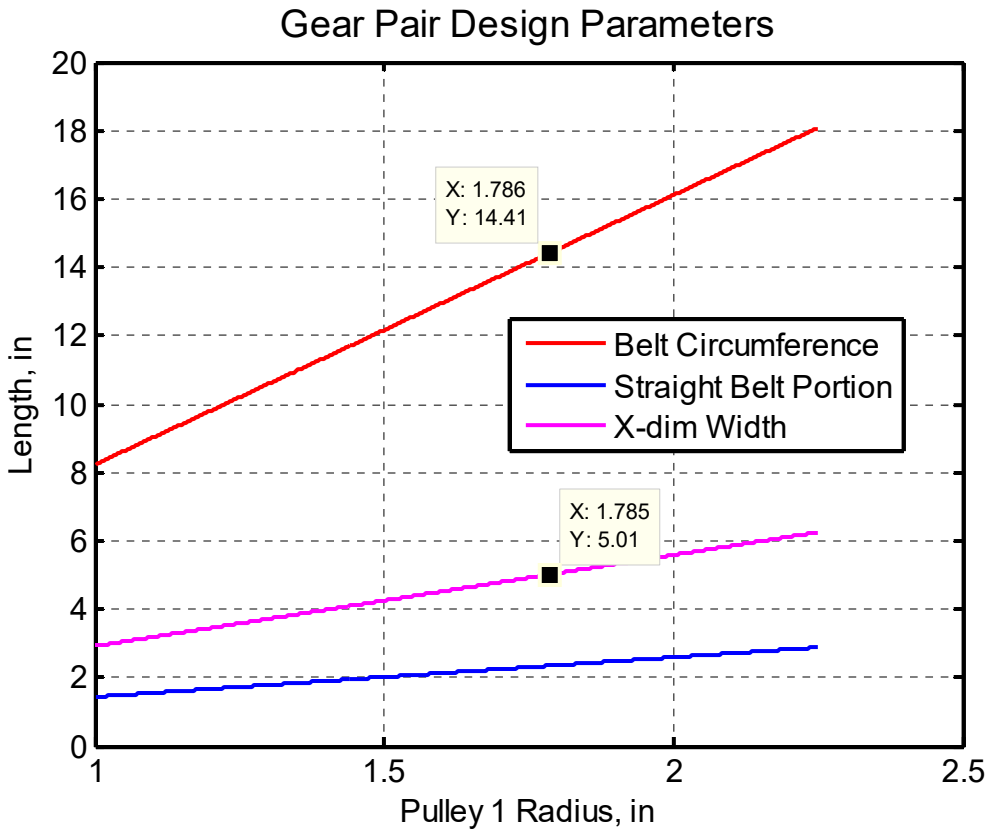


Figure 53 Gap between pulleys = 0.25" with reduction factor of 3.0. Belt circumference[12] associated with total width of 5" is 14.41" or equivalently 366mm.

[12] From u26994_pulley_system.m.

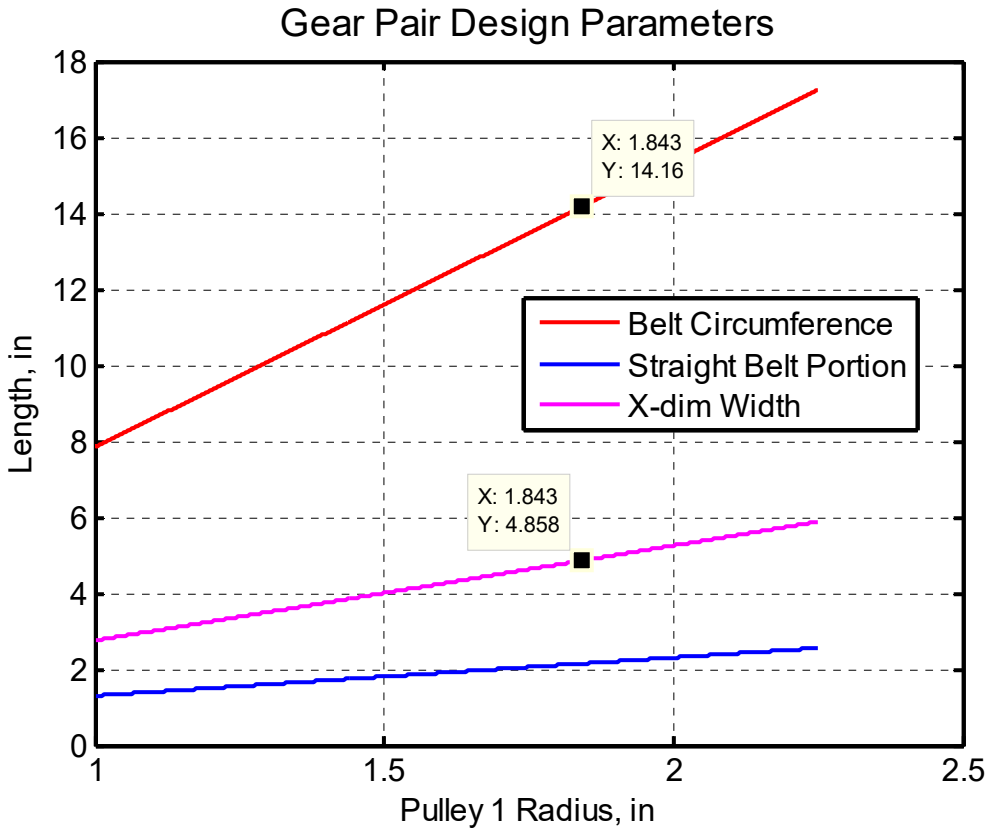


Figure 54 Gap between pulleys = 0.25" with reduction factor of 4.0. Belt circumference associated with total width of 4.858" is 14.16" or equivalently 360mm.

One of the most common timing belt tooth spacings available is 2mm (0.07874") which is fairly small to mill on my Haas TM1-P machine. Wishing to complete this portion of the project sooner rather than later, it is more convenient to procure the belts and pulleys on-line from Amazon or the equivalent. The most common large pulley sizes are 40T, 60T, and 80T where T denotes *teeth*. For the smaller pinion gear, the most common sizes are 16T and 20T. These details are summarized in Table 8.

Table 8 Commonly Available Timing Pulleys

Number of 2mm Teeth	Pitch Diameter, inches		Number of 2mm Teeth	Pitch diameter, inches
40	1.00		16	0.401
60	1.504		20	0.5013
80	2.0051			

Table 9 Ideal Torque Using 1-Stage and 2-Stage Pulley Reductions

Pulley 1 Teeth	Pulley2 Teeth	Stall Torque, 1 Stage, ft-lb	Stall Torque, 2 Stages, ft-lb
60	16	0.41475	1.555
"	20	0.3318	0.9954
80	16	0.553	2.765
"	20	0.4424	1.7696

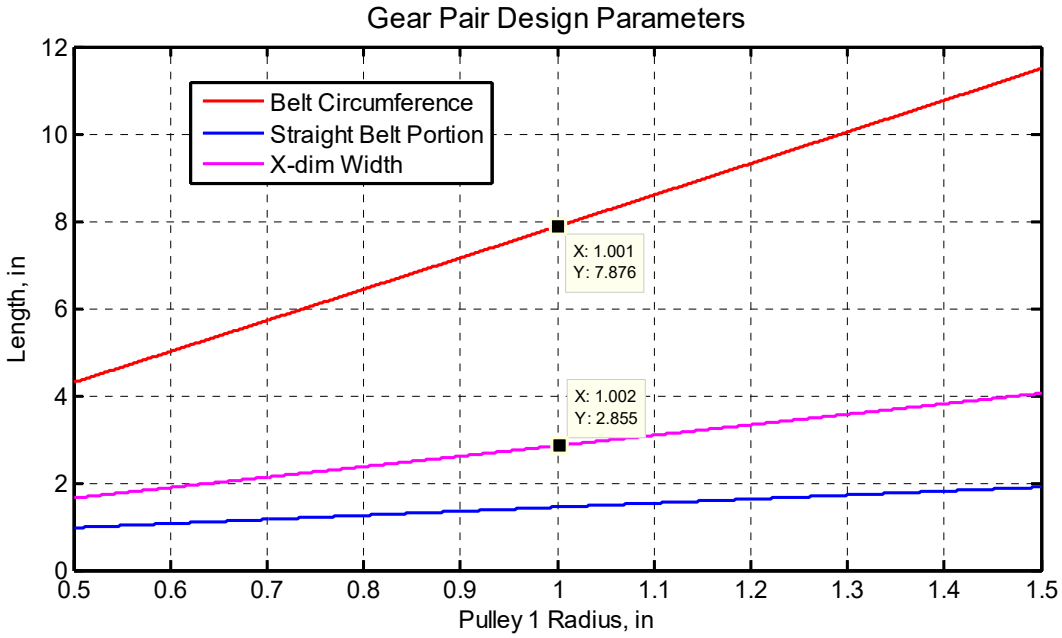


Figure 55 Design nomographs[13] with pulley separation of 0.45" appropriate for 200 mm belt length associated with final design choices

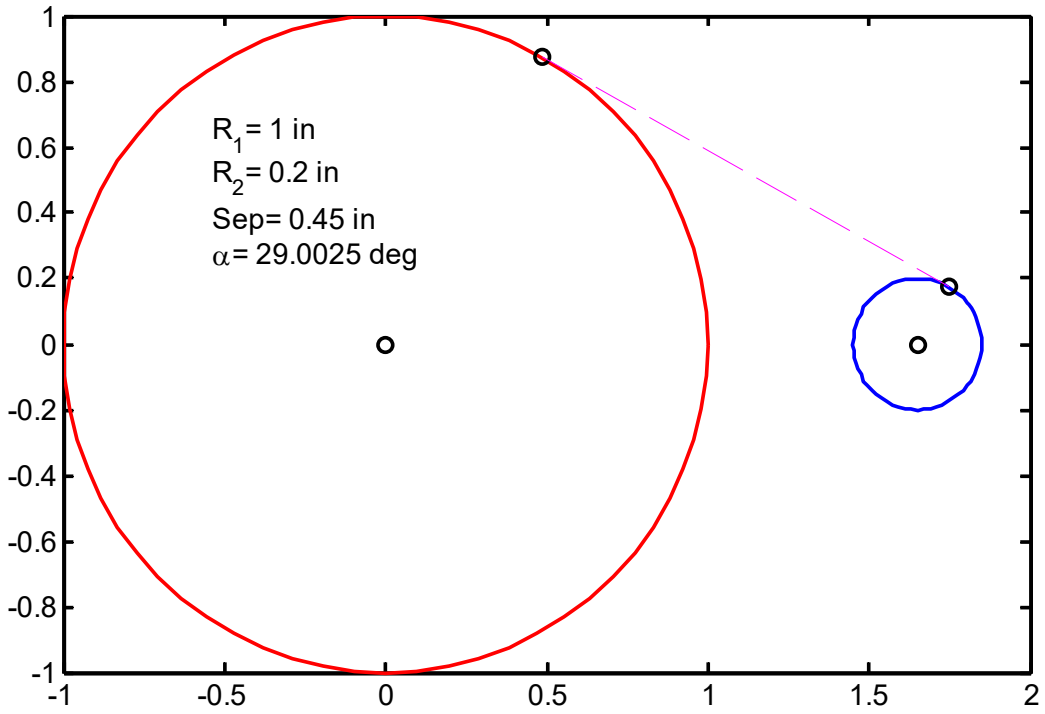


Figure 56 Geometry for single stage pulley reduction exhibiting 5:1 reduction associated with final design choices

[13] From u26994_pulley_system.m.

11 Appendix: Gear Terminology[14]

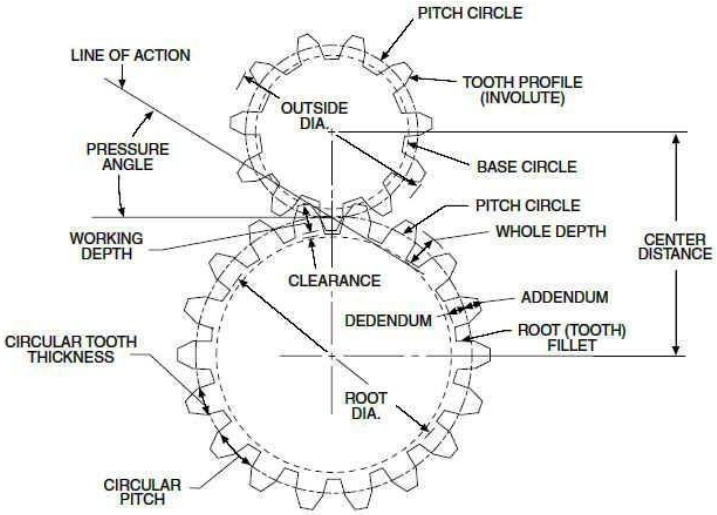


Figure 57 Gear terminology[15]

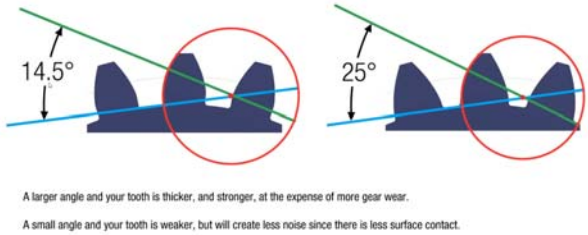


Figure 58 Pressure angle is the angle between the tangent to the tooth and the tangent to the root fillet circle. Standard angles are 14.5°, 20°, and 25°.

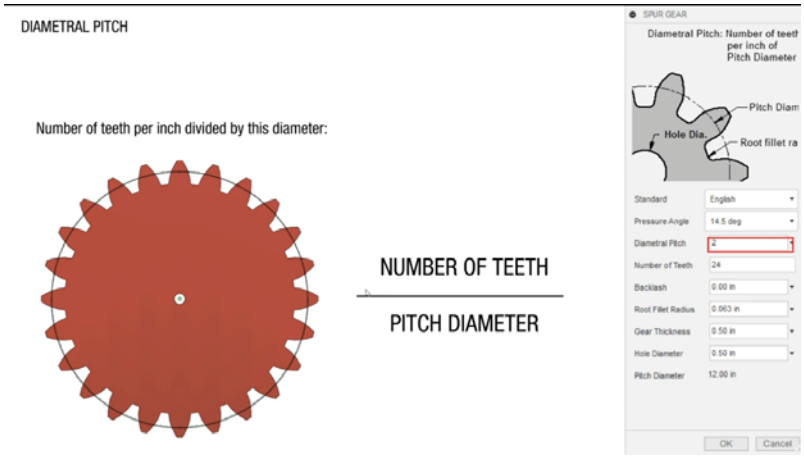


Figure 59 Diametral pitch from Fusion 360

[14] Spur Gear Parameters in Fusion 360, on Youtube
[15] As found at www.quora.com under *diametrical pitch*.

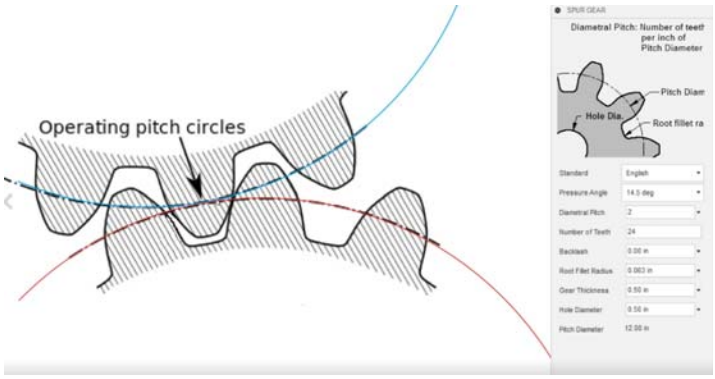


Figure 60 Operating pitch circles from Fusion 360

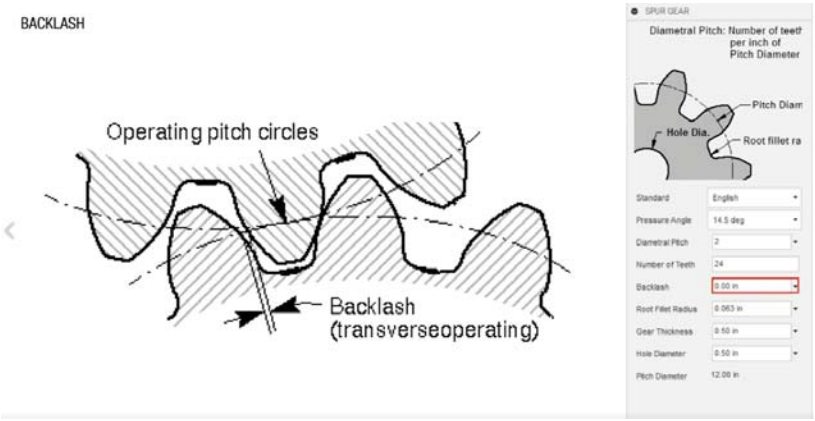


Figure 61 Gear backlash from Fusion 360

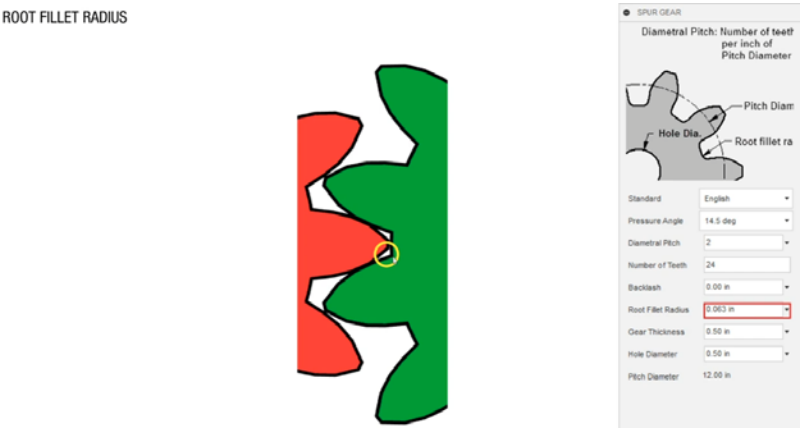


Figure 62 Gear fillet radius must be such that the tip of the tooth does not touch the bottom of the root fillets from Fusion 360

12 Appendix: Involute Gear Design Mathematics

I had my first real exposure to *involute gear design* while working on this project—and I unnecessarily got over-infatuated with it. In hindsight, I now realize this often-chosen gear tooth profile *is not* a good choice for my flattened harmonic drive concept after all. The intellectual investigation was quite fun, however, and I have little doubt but that I may need these concepts in a future project. Consequently, even though there is a great deal of material available on the internet, I will document some of my findings here.

12.1 Not Suitable for Flattened Harmonic Drive Concept

The involute tooth profile is a standard approach to use in general gear design because (i) it permits gears having a wide range of tooth-count to be easily integrated together, (ii) it results in high gear torque efficiency, and (iii) low noise for starters. Perhaps the most striking reason for not choosing the involute shape for my harmonic drive is that one of its attributes is that gear-pairs can be designed to have a contact ratio (CR) between roughly 1.2 and ≥ 2.0.

> Gear contact ratio is the average number of teeth in contact between two gears as they rotate together.

A large contact ratio improves load handling power of a gear train while reducing noise and potential backlash. A detailed derivation for CR can be found in [11]. This reference gives the contact ratio as

$$CR = \frac{\sqrt{r_{ap}^2 - r_{bp}^2} + \sqrt{r_{ag}^2 - r_{bg}^2} - c\sin(\phi)}{p_b} \tag{21}$$

in which

r_{ap}, r_{ag}	addendum radii of the mating pinion and gear
r_{bp}, r_{bg}	base circle of the mating pinion and gear
c	center-to-center distance between the two gears
p_b	base pitch
N	number of teeth
d_b	diameter of the base circle
ϕ	pressure angle
P	diametrical pitch = number of teeth per inch of gear diameter

This terminology is further clarified in §11, and in Figure 63 through Figure 65.

It is natural to ask how large *CR* can be. For standard gears, the diametrical pitch is given by

$$P = \frac{1}{r_a - r} = \frac{\pi}{p_b \cos(\phi)} \tag{22}$$

In the limit as $P \to \infty$,

$$CR_{\max} = \frac{4}{\pi \sin(2\phi)} \tag{23}$$

which equates to 1.981 for the standard pressure angle ϕ = 20°.

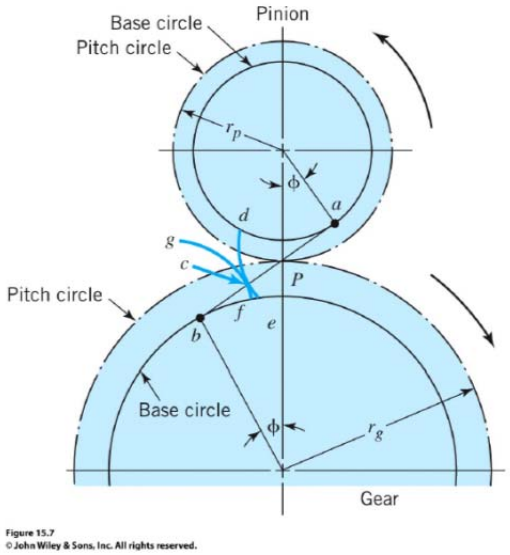


Figure 15.7
© John Wiley & Sons, Inc. All rights reserved.

Figure 63 From [11] showing the relationships between base and pitch circle radii and pressure angle

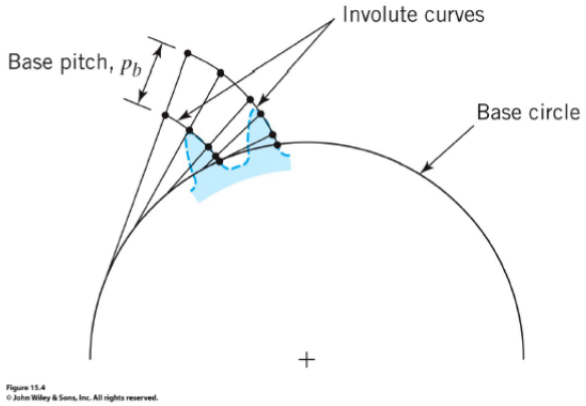


Figure 15.4
© John Wiley & Sons, Inc. All rights reserved.

Figure 64 From [11] showing the base pitch, p_b, which is measured as the length of a single tooth measured along the base circle

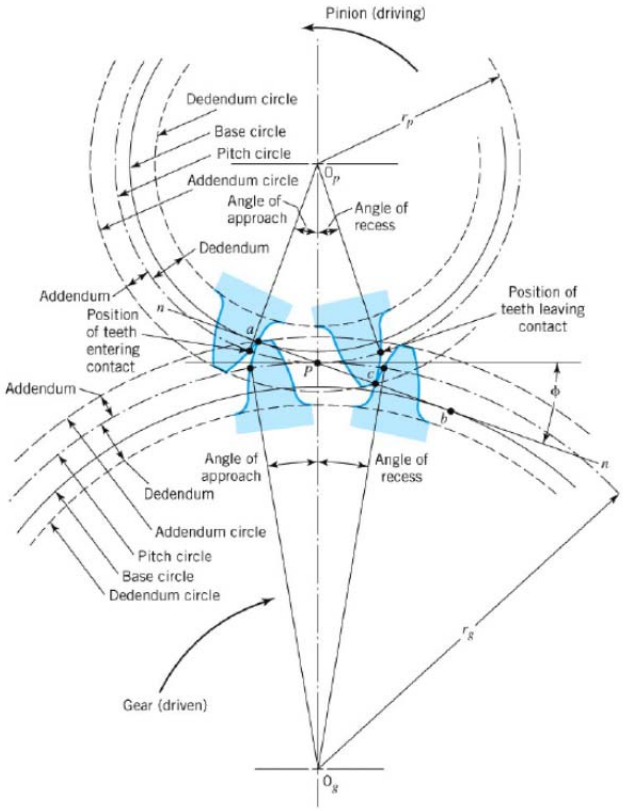


Figure 15.8
© John Wiley & Sons, Inc. All rights reserved.

Figure 65 Showing two pairs of teeth in contact from From [11]. The pinion tooth at *a* is just beginning to push along the pressure line on its gear tooth and that at *c* is just finishing pushing on its gear tooth.

As shown here, the maximum *CR* value is on the order of 2 implies an average of 2 gear teeth are in contact with each other as the gears are rotated. In sharp contrast, up to roughly 30% of the teeth used in a harmonic drive are always in contact! This observation is one of the hallmarks of harmonic drives over more standard gear trains.

12.2 The Involute

The involute of a circle (as given by Wikipedia) is the locus of points on a piece of taut string as the string is either unwrapped from or wrapped around the circle as shown in Figure 66.

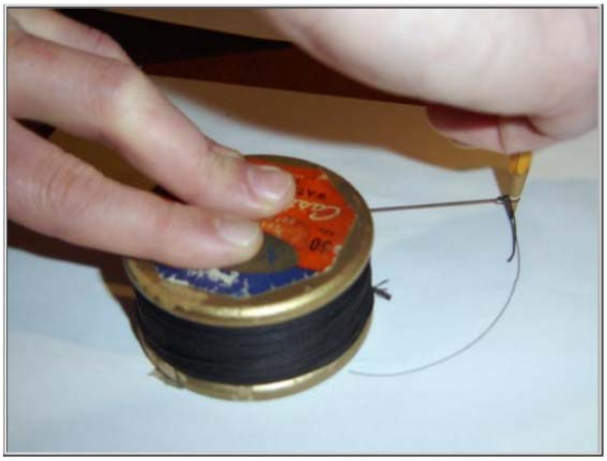


Figure 66 Involute of a circle from [12]

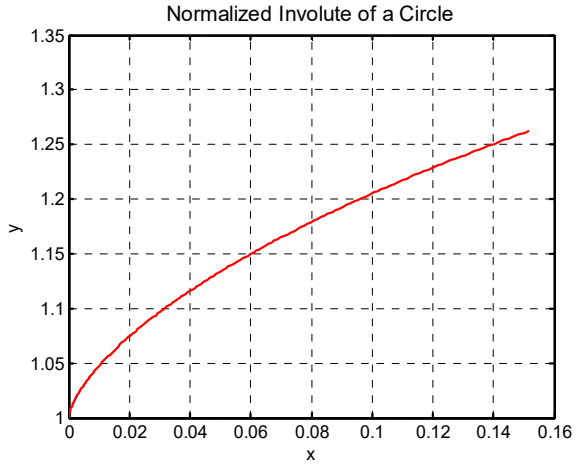


Figure 67 Normalized involute of a circle[16]

The mathematical equations for the involute of a circle is given in Cartesian coordinates as

$$\begin{aligned} x_{eq} &= r_{b,eq}\left[\sin(\theta)-\theta\cos(\theta)\right] \\ y_{eq} &= r_{b,eq}\left[\cos(\theta)+\theta\sin(\theta)\right] \end{aligned} \tag{24}$$

where $r_{b,eq}$ is the radius of the fundamental circle of the equivalent gear and θ that spans from zero to θ_t where

$$\theta_t = \sqrt{\left(\frac{r_{t,eq}}{r_{b,dq}}\right)^2 - 1} \tag{25}$$

A plot of (24) is shown in Figure 67. The involute curve is important in gear design because it forms the basis for common gear tooth design.

The involute curve can be seen in action[17] by observing the motion of a rack and pinion gear assembly as shown in Figure 68. This perspective was offered by Dr. Rainer Hessmer[18] in which each lateral move of the rack Δx was followed by a reverse rotation of the gear by $\Delta\theta = \Delta x / r_{pinion}$ thereby making the teeth trajectories along the gear's circumference look stationary (i.e., gear not rotating). The

[16] From u26956_involute_gear_design.m.
[17] from Measurement, U26967.
[18] www.hessmer.org.

graphical result for a moderate-sized $\Delta\theta$ is shown in Figure 69 with the finer-step case shown in Figure 70. The curved portion of each tooth silhouette inside the circle is the involute curve.

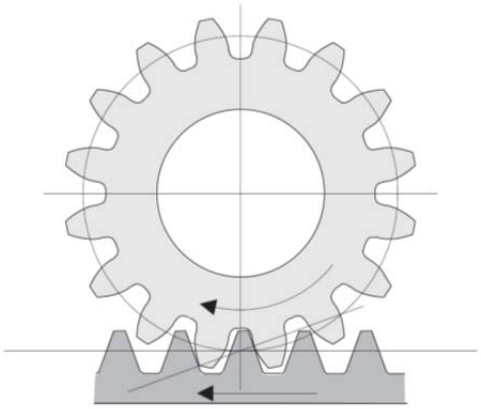


Figure 68 Sketch of a rack (lower straight gear) and pinion (upper gear)

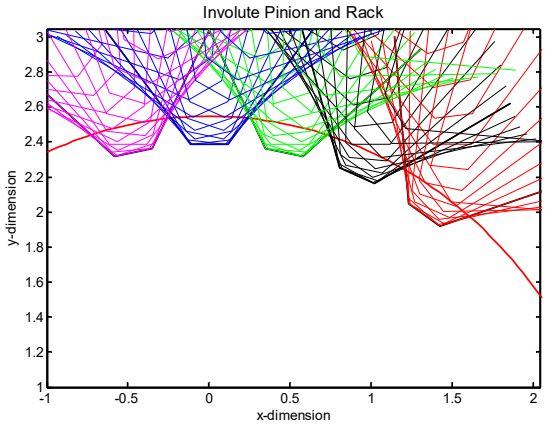


Figure 69 The involute curve shape can be seen[19] by tracing out the gear's tooth position as it is rolled across a rack gear

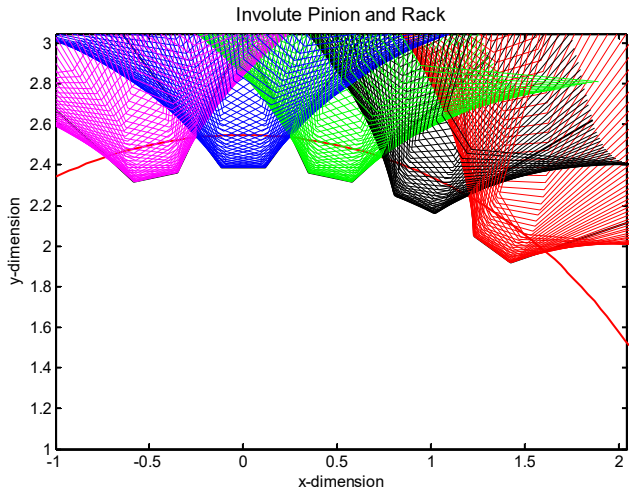


Figure 70 Finer step version of Figure 69

[19] u26983_rack_pinion_sim.m.

13 Appendix: Gear Design Techniques in Fusion 360

Caution: Although the gear discussion in this section is termed bevel gear, the parameter design techniques used to convert the original spur gear into a so-called bevel gear are completely ad-hoc. So although the Fusion 360 methods used in this discussion are certainly useful, a more rigorous technique should be followed for the bevel design parameter.

Fusion 360 so happens to have an add-in tool for designing involute spur gears. While I was researching the suitability of the involute gear tooth shape for my project, I was simultaneously exploring how Fusion 360 could be used to physically design them. As a result, the cart got ahead of the horse a bit and I learned how to design with the Fusion 360 tool before I discovered the unsuitability of the involute tooth shape for my project as described in §12.

Nevertheless, this adventure into using Fusion 360 proved to be very helpful. This exercise proved to be rather elusive, however, in that one very key almost undocumented feature of Fusion 360 had to be employed. I discovered this technique on a Youtube video. More specifically, although Fusion 360 includes a very nice macro for the design of involute gears, there is no way to break the resultant sketch/body into the features needed to make a bevel gear. An example end-result of my efforts is shown in Figure 71. One key feature of the design was making the teeth slanted at an angle.

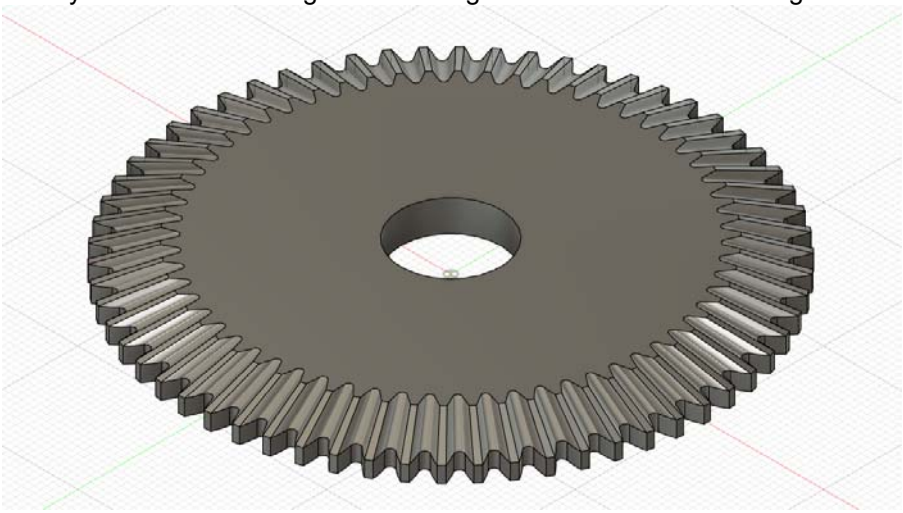
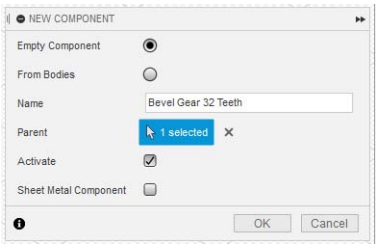


Figure 71 Example involute bevel gear designed in Fusion 360.

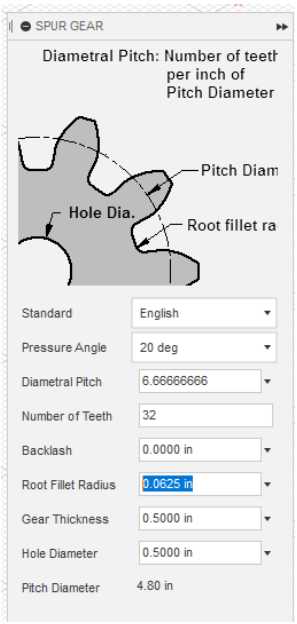
The construction steps in Fusion 360 took a while to sort out so they are recorded here for my possible future re-use.

In most cases, be sure that Selection Mode has its priority set to Face and that the Solid rather than Surface mode has been selected.

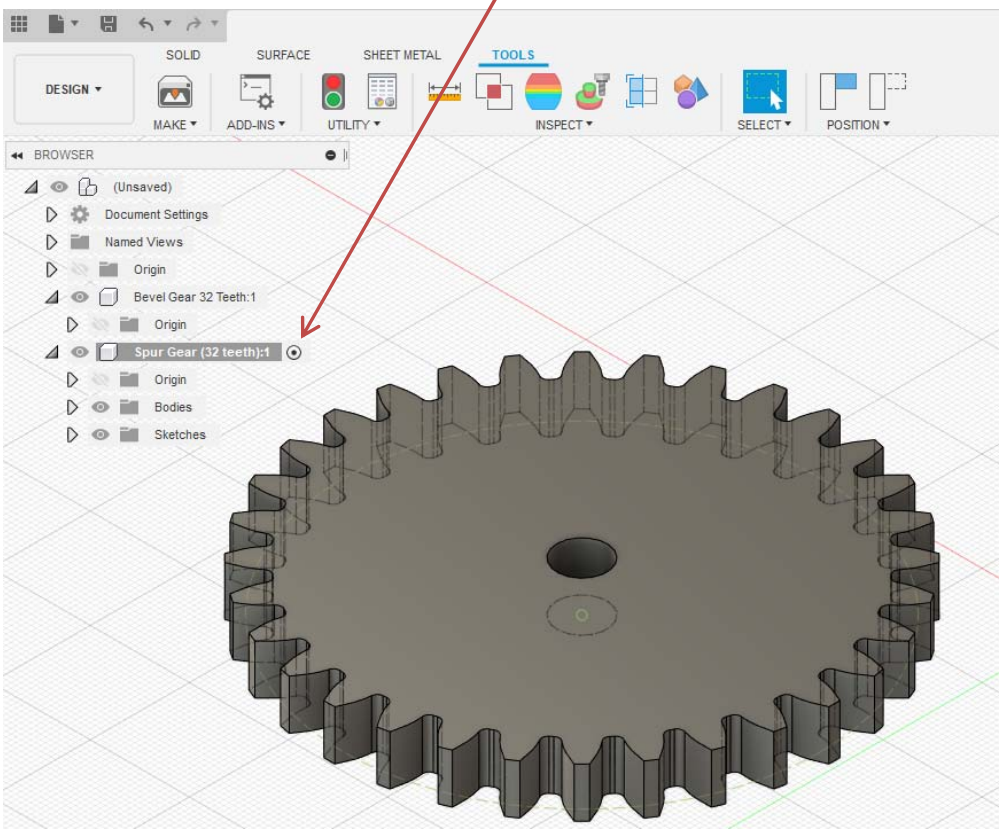
- Create a new component. Call it **Bevel Gear 32 Teeth**



- Use Spur Gear macro to design the base gear of interest. This is a Fusion 360 tool under ADD-INS/Scripts and Add-Ins named **Spur Gear.** Specify the design parameters as shown below.

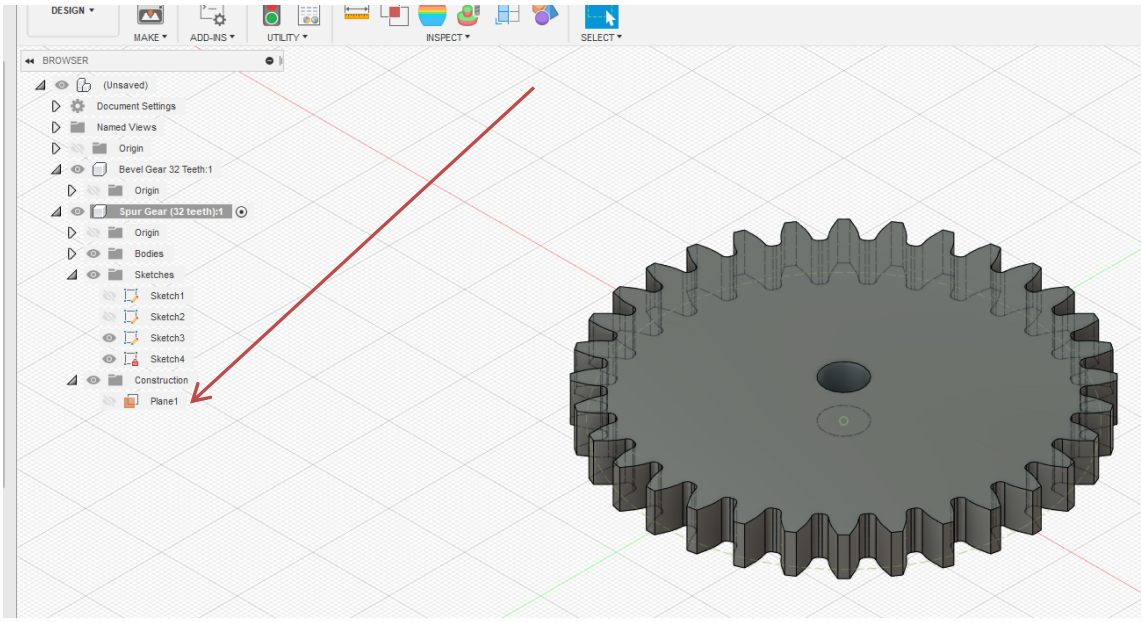


- After executing the Run command in Spur Gear, a new gear will be created as shown below. Activate it as shown by the arrow for easier book-keeping later.

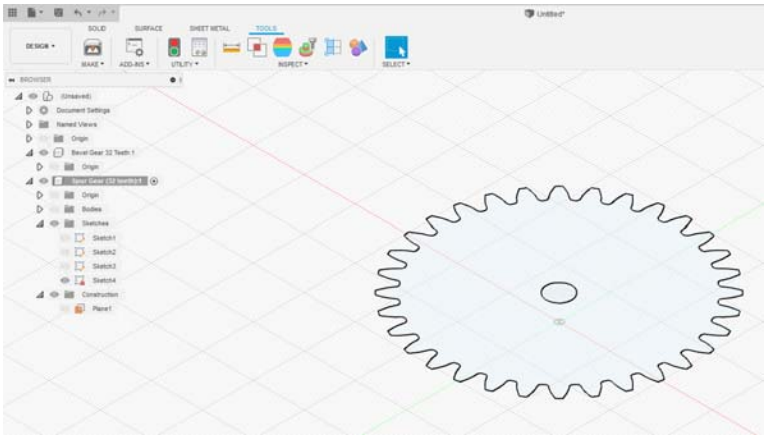


- Select/high light the spur gear body using a left-click of the mouse on the part. This will high-light the top-face of the component.
- Right-click on the face of the gear body, and create a Sketch on the face of the component.
- Finish the sketch.

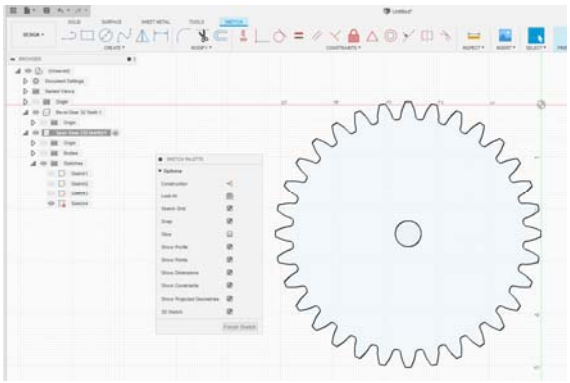
- Right-click on the just-saved sketch and add an offset plane at the pre-designed distance, in this case 0.125 inches.
- Hit return.
- Temporarily hide the offset plane as shown below.



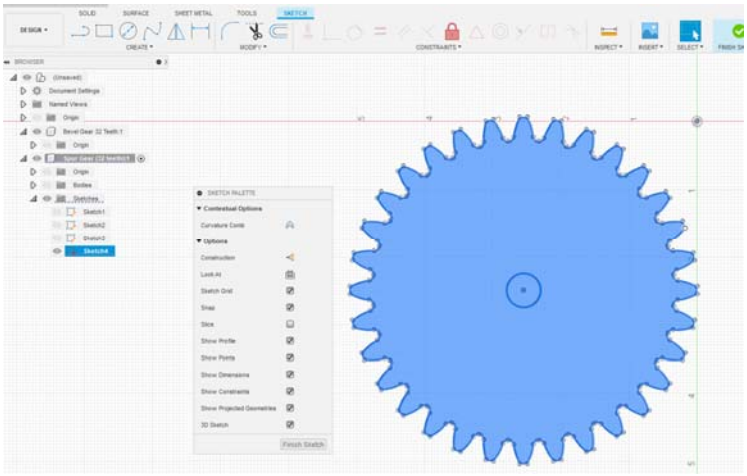
- The gear body is no longer needed, so hide it from view also. Notice the sketch which has been made visible. The gear silhouette only gets its bold outline when the mouse is hovering over it. Hiding the body is convenient later as it doesn't confuse matters when trying to copy sketch-only features.

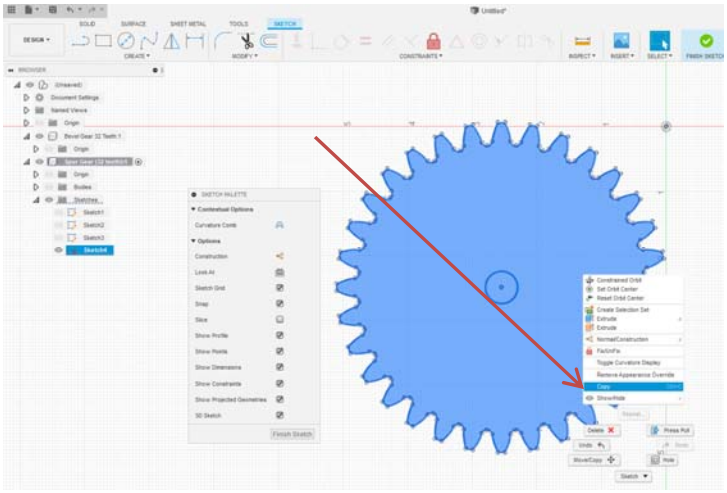


- Open the first sketch again for editing. Upon first opening, only a shadow-like appearance will be present, but when doing a screen copy-and-paste, the bold outline is also shown as below.



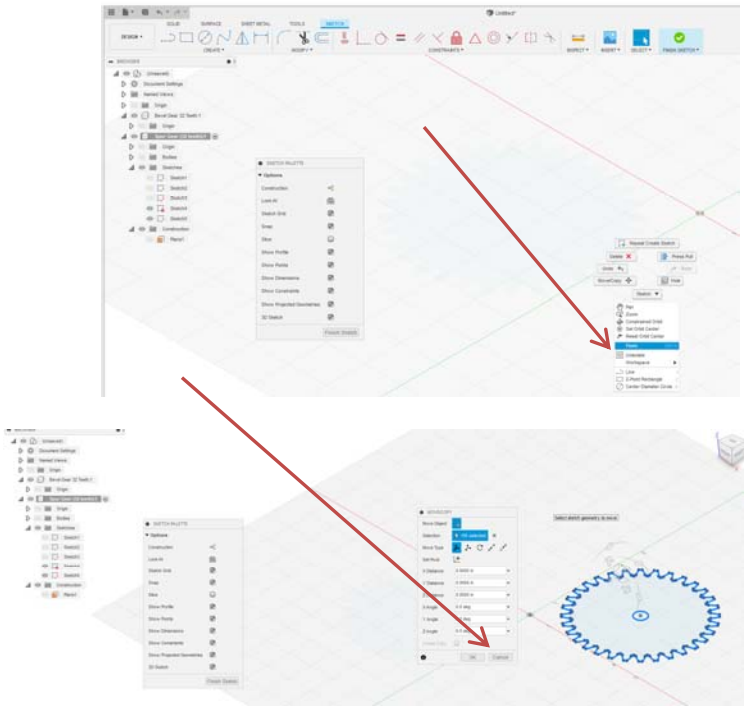
- Select all in the edit window by dragging the mouse to enclose the entire gear as shown below.
- Right-click to copy as also shown below.
- Finish the sketch.



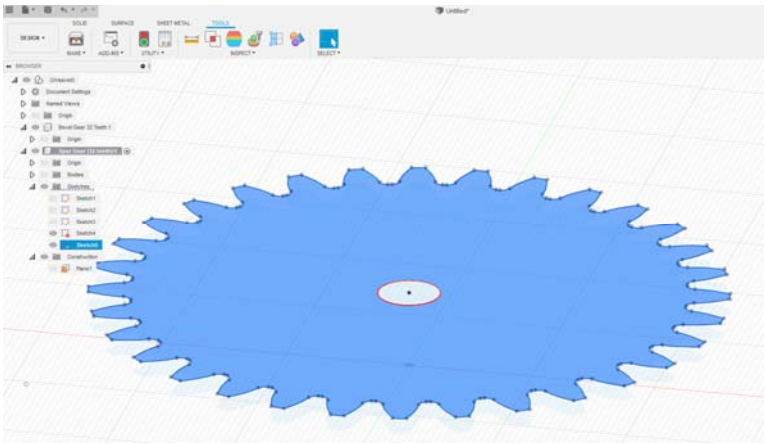


- Make the offset plane visible again by clicking its visibility icon.
- Create a new sketch on the offset plane.
- Right click, and select Paste to paste the gear pattern on the offset plane.

- A Move/Copy window will appear--- Cancel it. Otherwise, you will move the contents of the first sketch to the offset plane rather than just make a copy of it.
- Finish the sketch.



- With both sketches visible as below, the newly copied gear on the offset plane will be directly over the original gear's silhouette as shown below.



- To make the bevel gear, the top sketch is selected for editing,
- Under the MODIFY tab, Sketch Scale is selected as shown below.
- Using the mouse, all of the gear items are selected.
- A non-zero point on the sketch must also be selected for the scaling operation to work. Doing so can shift the location of the sketch figure undesirably, but if this occurs, simply use the MOVE command to correctly reposition the object.
- I will use a scaling factor of 0.80

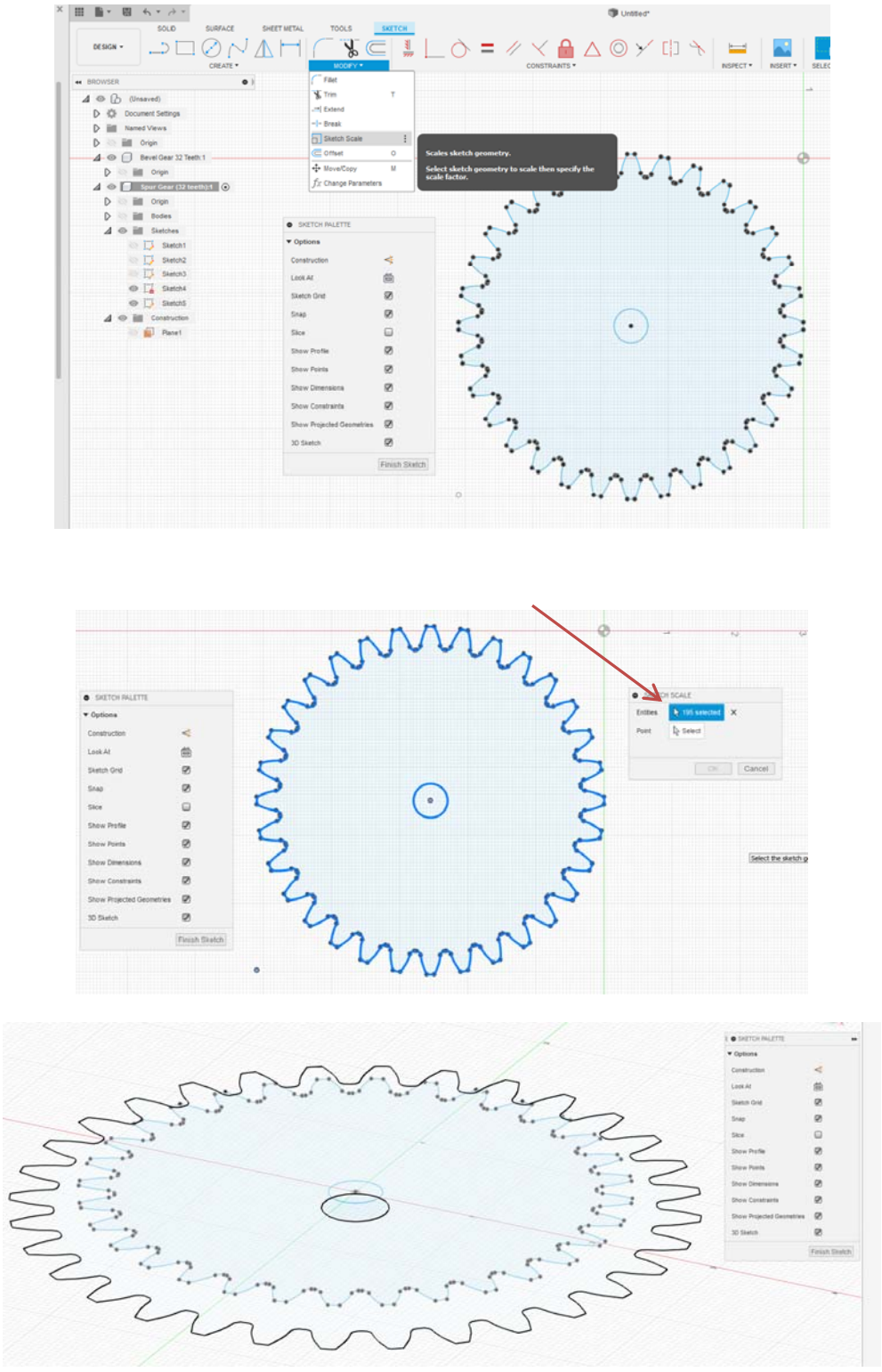


Figure 72 Original gear sketch and just-scaled gear on the offset plane both made visible

- Use the LOFT tool to combine the two sketches into the desired 3-D bevel gear. The final result is shown in

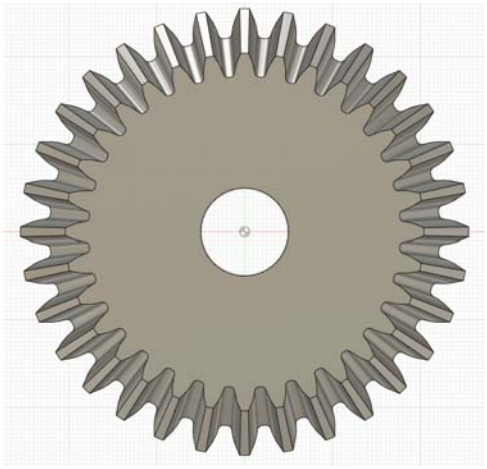


Figure 73 Top-view of the newly designed gear

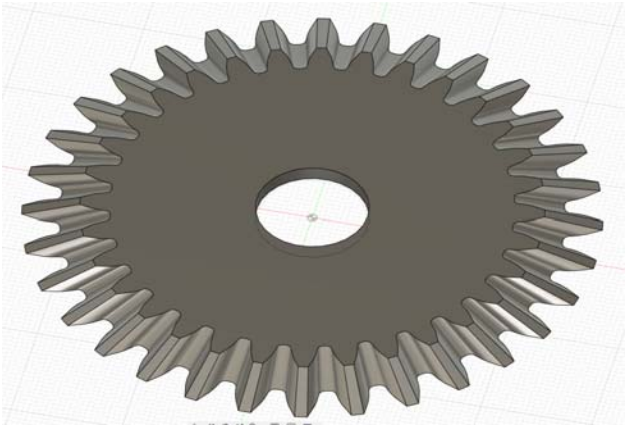


Figure 74 Offset view of the newly designed gear

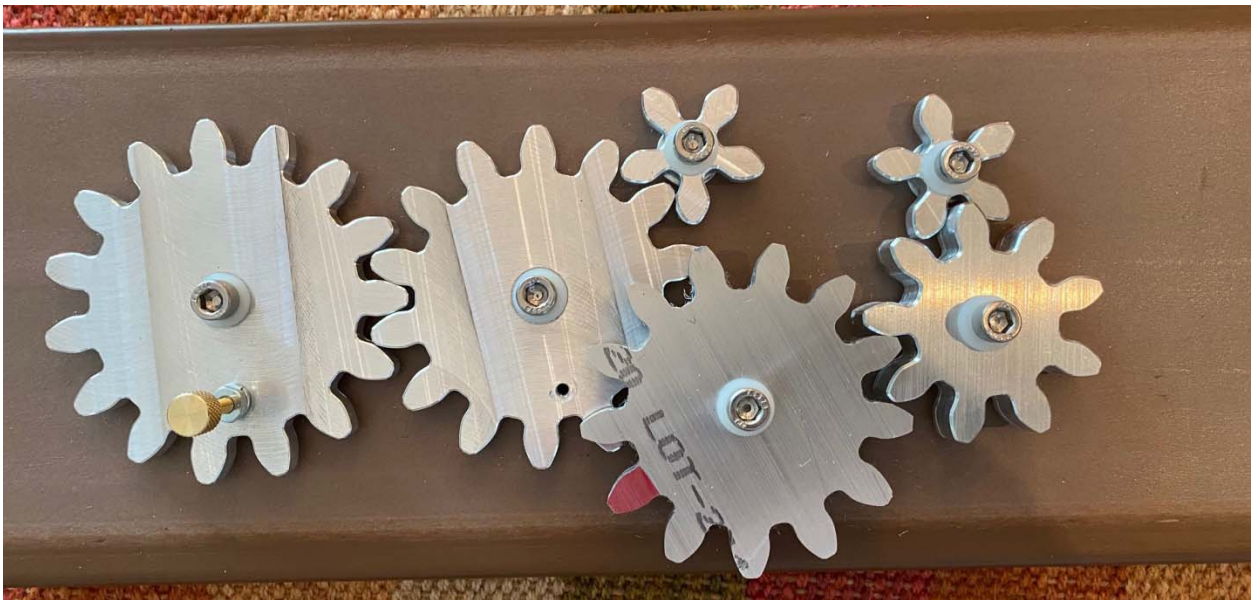


Figure 75 Involute gear train made of for a child's toy. The gear with the red tooth is actually a stacked-gear. Building such a train gives one a true perspective about how close the mechanical tolerances need to be in order to have no binding and little to no backlash!

14 Appendix: Pulley / Timing Belts

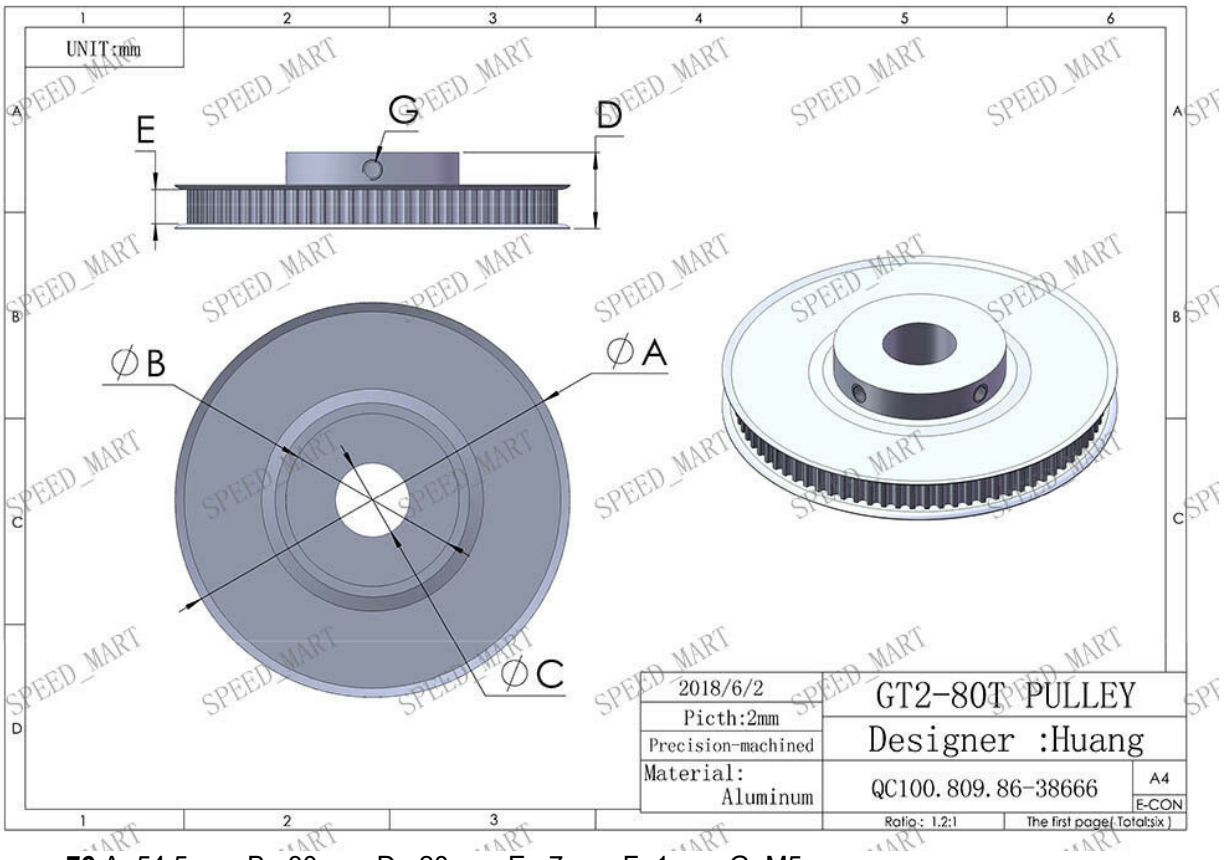


Figure 76 A=54.5mm, B= 30mm, D= 20mm, E= 7mm, F=1mm, G=M5

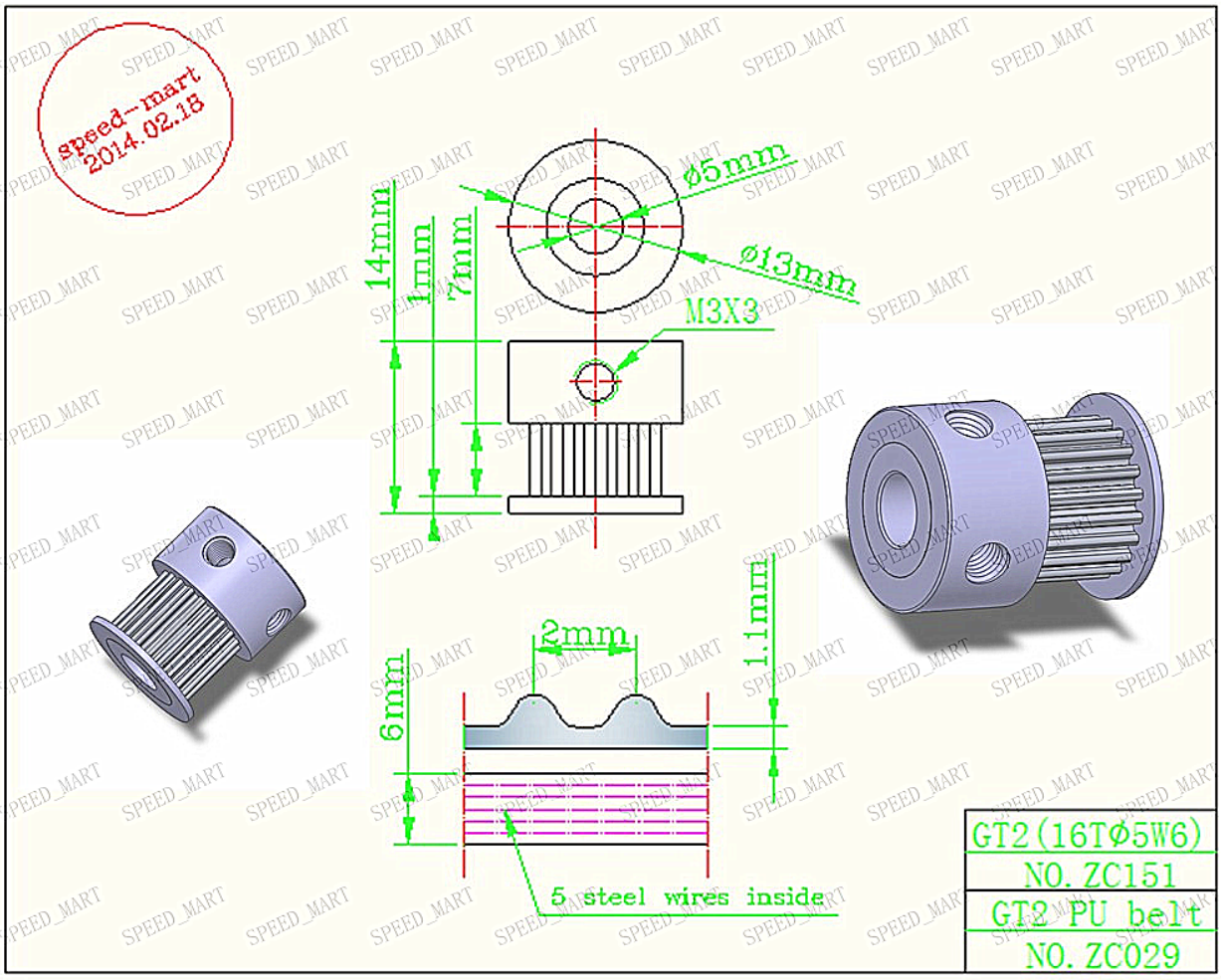

speed-mart
2014.02.18
ø5mm
ø13mm
14mm
1mm
7mm
M3X3
2mm
1.1mm
6mm
5 steel wires inside
GT2(16TØ5W6)
NO. ZC151
GT2 PU belt
NO. ZC029

Figure 77

15 LAUNCHXL-F28379D with BOOSTXL-DRV8301

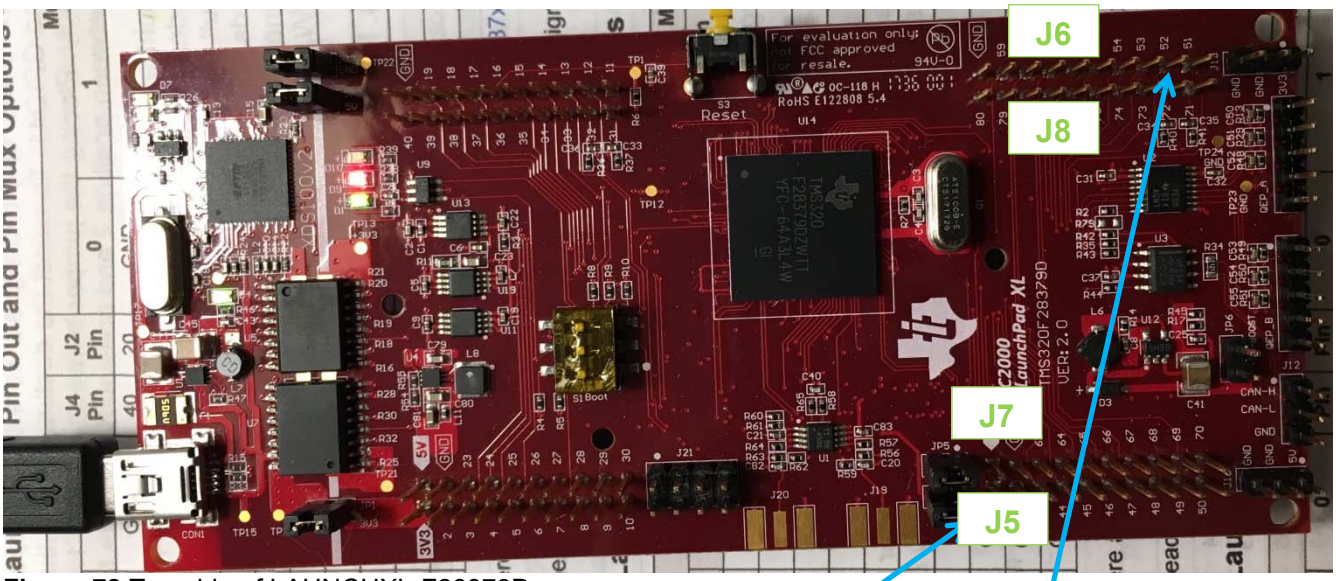
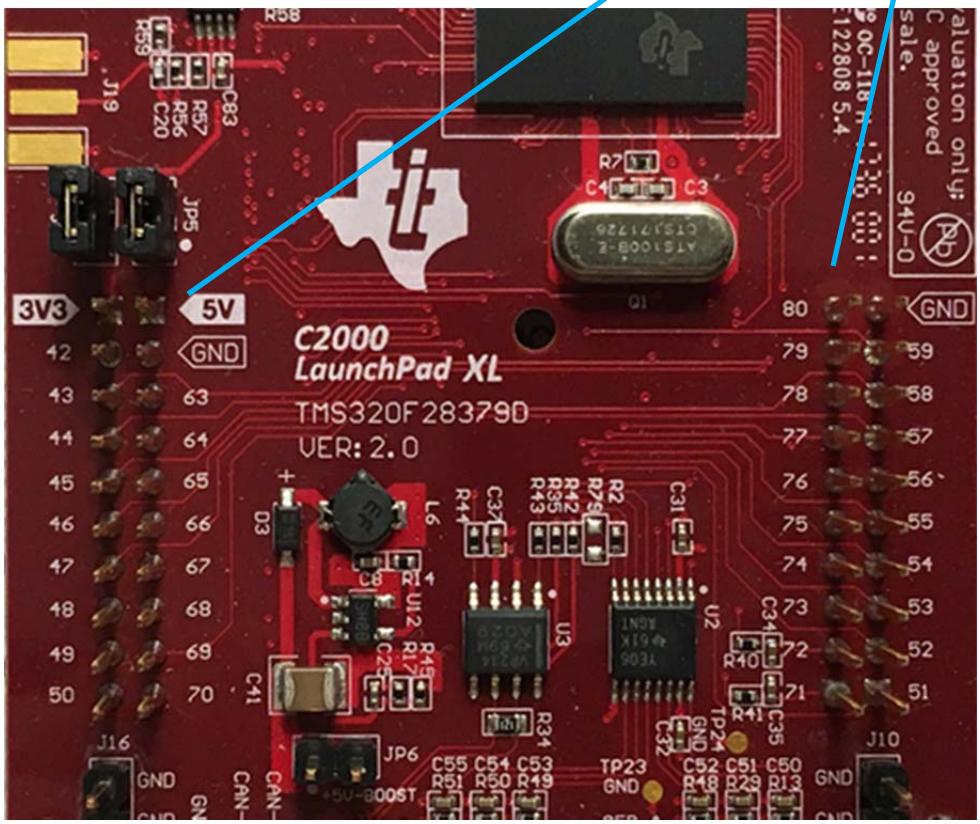


Figure 78 Top-side of LAUNCHXL-F28379D



Mux Value				J5 Pin	J7 Pin	Mux Value			
X	2	1	0			0	Alt Function	2	X
			3.3V	41	61	5V			
			GPIO95	42	62	GND			
SCIRXDC[1]			GPIO139	43	63	ADCIN15	CMPIN4N		
SCITXDC[1]			GPIO56	44	64	ADCINC5	CMPIN5N		
			GPIO97	45	65	ADCINB5			
			GPIO94	46	66	ADCINA5	CMPIN2N		
SPICLKB[1]			GPIO65	47	67	ADCINC4	CMPIN5P		
			GPIO52[2]	48	68	ADCINB4			
SCLB[1]			GPIO41[2]	49	69	ADCINA4	CMPIN2P		
SDAB[1]			GPIO40[2]	50	70	ADCINA1	DACOUTB		

Figure 79 LAUNCHXL-F28379D Pin Out and Pin Mux Options- J5 & J7 [4]

Mux Value				J8 Pin	J6 Pin	Mux Value			
X	2	1	0			0	1	2	X
		EPWM4A	GPIO6	80	60	GND			
		EPWM4B	GPIO7	79	59	GPIO66			
		EPWM5A	GPIO8	78	58	GPIO131			SD2_C1[1]
		EPWM5B	GPIO9	77	57	GPIO130			SD2_D1[1]
		EPWM6A	GPIO10	76	56	RST			
		EPWM6B	GPIO11	75	55	GPIO63			SPISIMOB[1]
OUTPUTXBAR3[1]			GPIO14	74	54	GPIO64			SPISOMIB[1]
OUTPUTXBAR4[1]			GPIO15	73	53	GPIO26			SD2_D2[1]
			DAC3	72	52	GPIO27			SD2_C2[1]
			DAC4	71	51	GPIO25			OUTPUTXBAR2[1]

Figure 80 LAUNCHXL-F28379D Pin Out and Pin Mux Options- J6 & J8 [4]

Mux Value				J1 Pin	J3 Pin	Mux Value			
X	2	1	0			0	Alt Function	2	X
			3.3V	1	21	5V			
			GPIO32	2	22	GND			
	SCIRXDB		GPIO19	3	23	ADCIN14	CMPIN4P		
	SCITXDB		GPIO18	4	24	ADCINC3	CMPIN6N		
			GPIO67	5	25	ADCINB3	CMPIN3N		
			GPIO111	6	26	ADCINA3	CMPIN1N		
SPICLKA[1]			GPIO60	7	27	ADCINC2	CMPIN6P		
			GPIO22	8	28	ADCINB2	CMPIN3P		
		SCLA	GPIO105[2]	9	29	ADCINA2	CMPIN1P		
		SDAA	GPIO104[2]	10	30	ADCINA0	DACOUTA		

Figure 81 LAUNCHXL-F28379D Pin Out and Pin Mux Options- J1 & J3 [4]

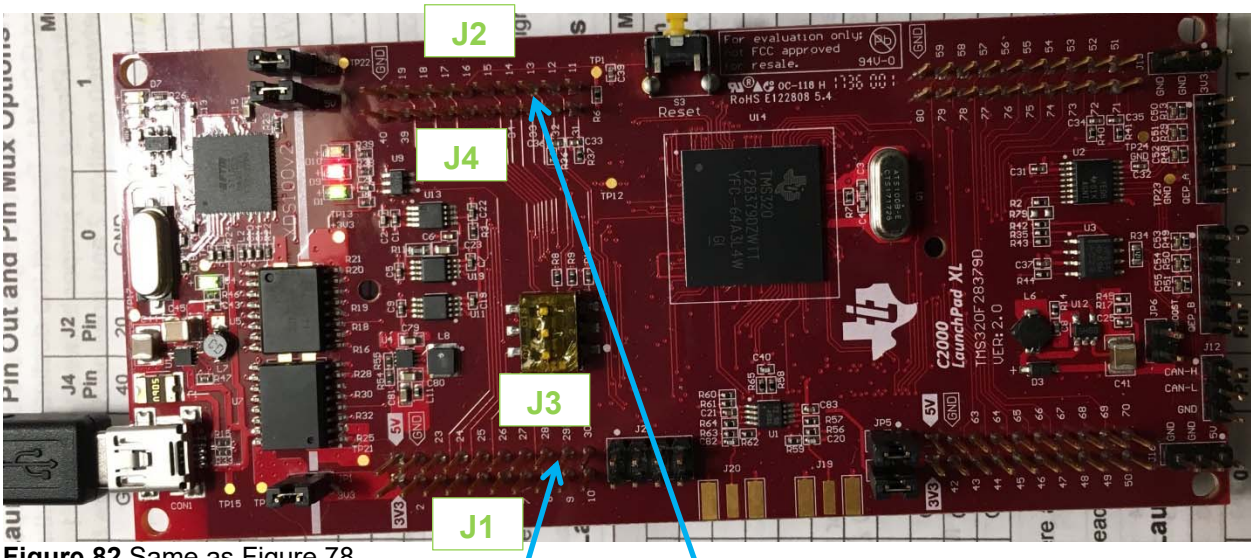
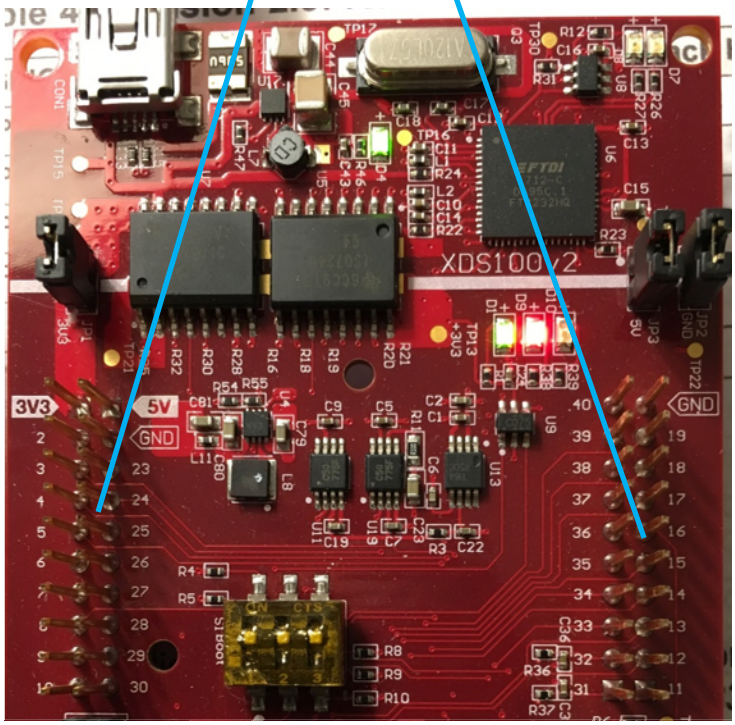


Figure 82 Same as Figure 78



Mux Value				J4 Pin	J2 Pin	Mux Value			
X	2	1	0			0	1	2	X
		EPWM1A	GPIO0	40	20	GND			
		EPWM1B	GPIO1	39	19	GPIO61			
		EPWM2A	GPIO2	38	18	GPIO123			SD1_C1(1)
		EPWM2B	GPIO3	37	17	GPIO122			SD1_D1(1)
		EPWM3A	GPIO4	36	16	RST			
		EPWM3B	GPIO5	35	15	GPIO58			SPISIMOA(1)
		OUTPUTXBAR1	GPIO24	34	14	GPIO59			SPISOMIA(1)
OUTPUTXBAR7(1)			GPIO16	33	13	GPIO124			SD1_D2(1)
			DAC1	32	12	GPIO125			SD1_C2(1)
			DAC2	31	11	GPIO29(2)			OUTPUTXBAR6(1)

Figure 83 LAUNCHXL-F28379D Pint Out and Pin Mux Options- J2, J4 [4]

Notes to Self

Motor Power

$$P_{watts} = Torque_{oz-in} RPM \times 0.00074 \tag{26}$$

$$1W = 44.2537 \frac{ft-lb}{\min} \tag{27}$$

Work= Force x Distance

115 oz-in @ 1200 RPM is about 102 W

(28)

$$Torque_{oz-in} \frac{1\,lb}{16\,oz} \frac{1\,ft}{12\,in} RPM \times 2\pi \frac{1}{44.2537} = Torque_{oz-in} RPM \times 0.00073948 \tag{29}$$

Odrive—Python, open-source
Trinamic Motion
Teknic with Clearpath TMC5160 motors

