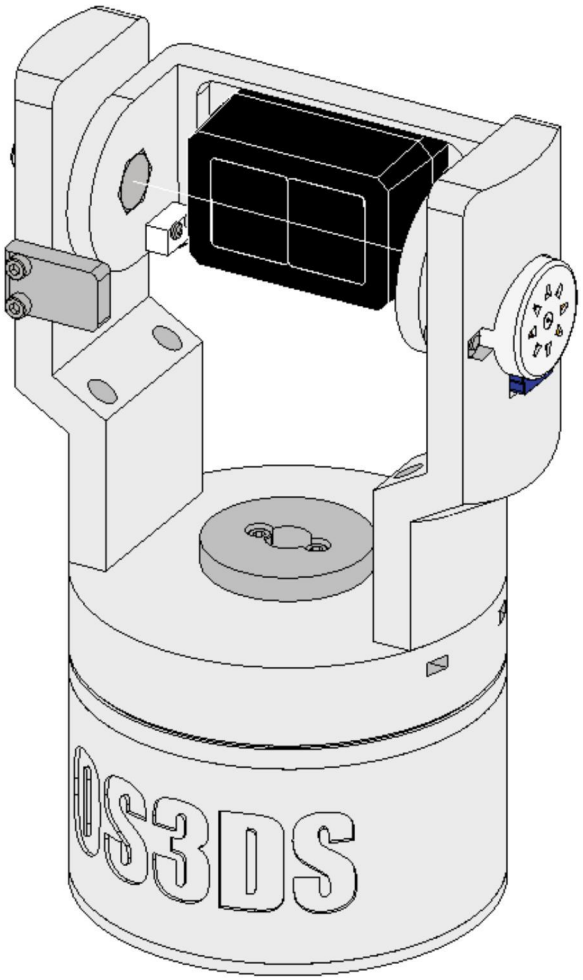
Low-Cost Open-Source 3D Scanner

Darren Paetz and Matt Kantor

Mentors: Dr. Philip Mees, Dr. Jeffrey Davis

MacEwan University



Made possible with funding by:
The Office of Research Services, MacEwan University
USRI (Undergraduate Student Research Initiative) Grant

MacEwan UNIVERSITY | OFFICE OF RESEARCH SERVICES

TABLE OF CONTENTS

TABLE OF FIGURES

1. Introduction

1.0 Abstract

The goal of this research project is to create a low-cost, open-source 3D scanner (**OS3DS**) that is capable of creating a reasonably accurate point cloud of an interior space or exterior façade of a building. This may prove useful to parties who are interested in 3D scanning, however are unable to afford a professional-grade 3D scanner. Typical professional grade scanners can cost $5 000 CAD and up. The project is open-source with the goal of encouraging future work and improvements on the base we have created. The target audience includes electronic hobbyists, digital designers, artists, archaeology students, interior designers, real-estate workers, or those looking to make a business case for the creation of a professional scan.

1.1 Review of Literature

The mapping of interior spaces with 3D scanners is a relatively common practice in the creation of modern Building Informational Models (BIMs). These models have been created to aid in the historical preservation of structures in Taiwan [1], Poland [2], and Scotland [3]. Groups of engineers have investigated the utility of interior spatial mapping in the application of autonomous interior finishing robotic platforms [4]. Other groups have created low-cost prototypes to pursue accessible historical specimen preservation in China [5]. The Arduino microcontroller platform is a common piece of open-source hardware that has a vibrant community of users around the world. The Atmel Mega 328P has been utilized to create other low-cost instrument prototypes, including an impedance meter in Kenya [6].

2. Design

2.0 Overview

The design of this device involves skills from many different disciplines of engineering. These disciplines include mechanical, electrical, software, as well as general problem solving. In this section of the report, we go into detail on the technical aspects involved in the design of this instrument.

The low-cost, open source 3D scanner (**OS3DS**) works by positioning a time-of-flight (LiDAR or SONAR) sensor using stepper motors to take distance measurements at algorithmically determined intervals. The distance and positon data from the scanner is

then sent over to a computer via USB serial communication. A Python-based software package we designed converts the data stream into a standardized open source point cloud format known as .E57. This point cloud can then be imported into appropriate software for post-processing. This processing includes the creation of a working 3D model.

2.1 Mechanical

The OS3DS has a completely 3D printed structure with maximum dimensions of 128mm diameter, and 200mm in height.

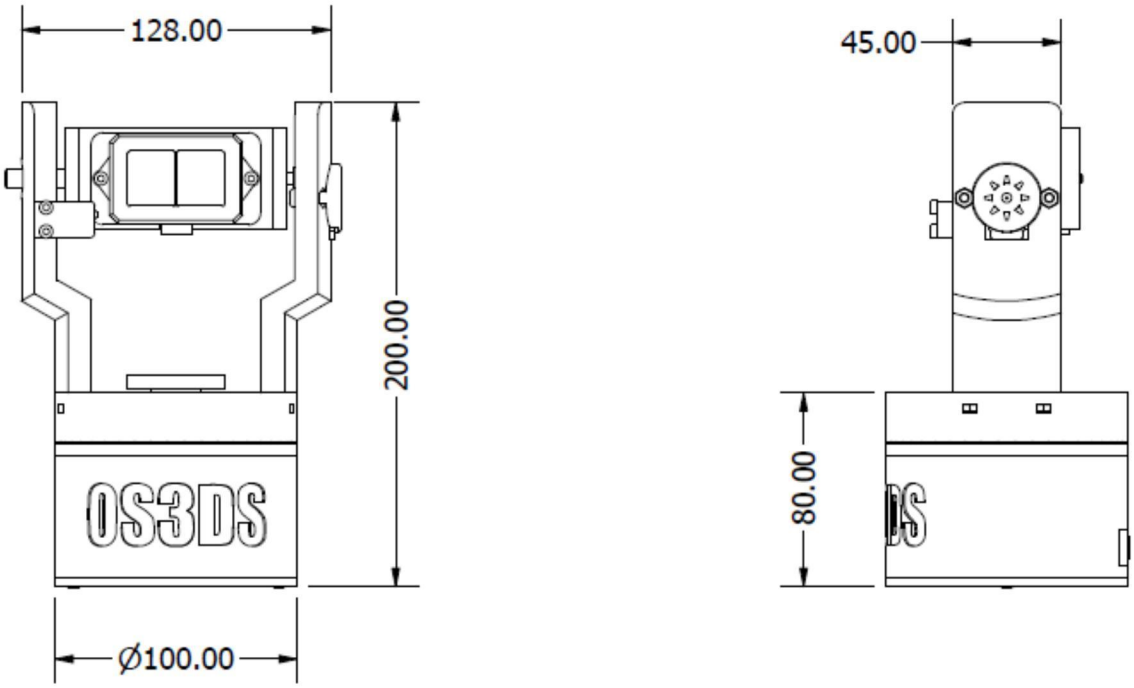


Figure 1: Physical dimensions of scanner

2.1.1 Designed for 3D printing

The OS3DS was designed to be made using extrusion-based fused deposition modelling (FDM) 3D printing. 3D printing allows for inexpensive low volume production, as there is no special tooling or setup cost involved with producing the parts. It also allows for easy modification and rapid testing of parts. This is ideal for this prototype, as it aligns with our design goals of being inexpensive, accessible, and open-source.

Utilizing 3D printing as the main method of manufacturing for this device allowed for some complex geometry to be produced that would be very expensive to produce from conventional subtractive manufacturing methods such as CNC milling. The best

example of this is the motion platform (Section 2.1.3), where we implemented an internal spur gear mechanism and 3D printed bearing race (Figure 2).

Careful consideration was taken when designing these parts to ensure they can be printed as quickly and as easily as possible. For example, the scanner head assembly was broken down into 3 small parts rather than one large one. By printing these parts separately, optimal adhesion to the printer is ensured by increasing the contact area. Additionally, this permits printing the parts in an orientation that requires minimal support material. Finally, with 3 separate parts, there is less risk of wasted time and material if a print were to fail – a substantial possibility in 3D printing.

2.1.2 Materials and mechanical components

The OS3DS is intended be made out of PETG or ABS plastic due to their relatively high glass transition temperatures of around 80° Celsius. The original design was made from PLA plastic, however, there is a large temperature rise on the stepper motors during operation. This risks softening the plastic, causing inaccurate measurements with the resulting deformation. PETG plastic was selected as it is easier to print than ABS: ABS printing requires an enclosure to prevent the print from warping.

The OS3DS uses two 5V 28BYJ-48 stepper motors; one for horizontal motion, and one for vertical motion. These motors were selected because they are some of the most commonly available and inexepensive stepper motors on the market. They also feature an internal 64:1 gear reduction. This gear reduction yields good torque, and a very precise theoretical resolution of 4096 steps per revolution (0.0879 degrees per step).

All of the hardware for the scanner are common metric sizes that should be available worldwide, with the exception of the tripod mounting nut and ball bearings. The nut is 1/4-20 UNC as per ISO 1222:2010 Photography – Tripod Connections [7] and the ball bearings are ¼" diameter. This mounting nut was chosen specifically to be compatible with a standard camera tripod.

2.1.3 Horizontal motion

The horizontal motion of the scanner is a shared responsibility between the scanner head, the scanner enclosure, and the scanner shaft. The scanner head (Fig. 2 yellow) and enclosure (Fig. 2 blue) contain v-shaped channels for ¼" ball bearings (Fig. 2 grey) to reside in. The scanner shaft (Fig. 2 red) serves the purposes of holding the scanner head on, preventing the ball bearings from coming out, and provides a shaft for the wires to travel through the center of the body.

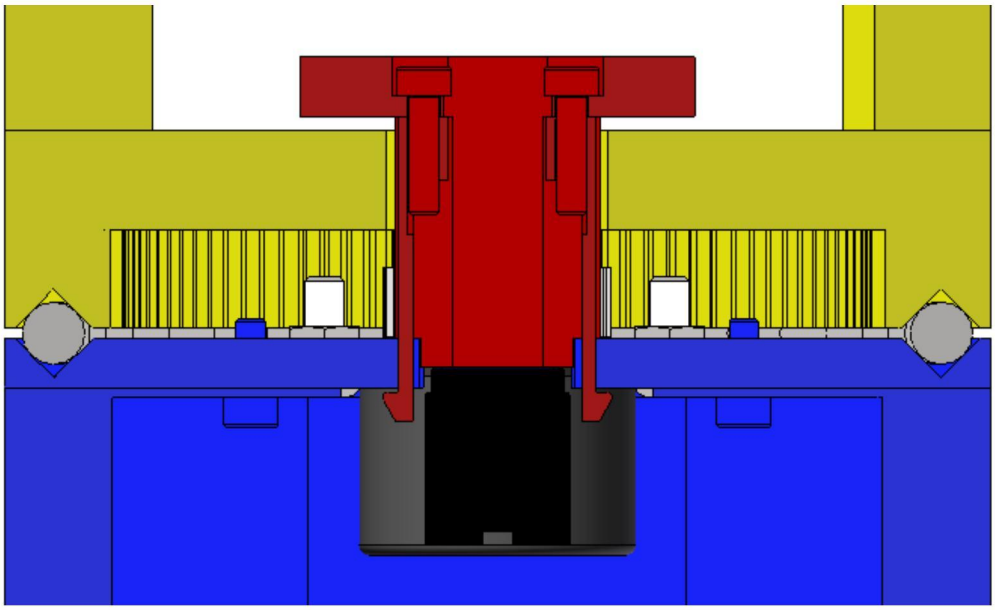


Figure 2: Cross section of motion platform

The v-shaped bearing race profile was selected after extensive testing with rounded profiles. The v-shape reduces the amount of contact between the ball and the race, which provides more reliable positioning.

The horizontal motion of the OS3DS utilizes a stepper motor and an internal spur gear system as shown in Figure 3 below to create precise horizontal motion. One of the major benefits of an internal spur gear system is greater contact between the rack and pinion gears, which reduces the amount of backlash (undesirable mechanical movement) when compared to an external spur gear. The spur gear system on the OS3DS has a reduction ratio of 3.55:1.

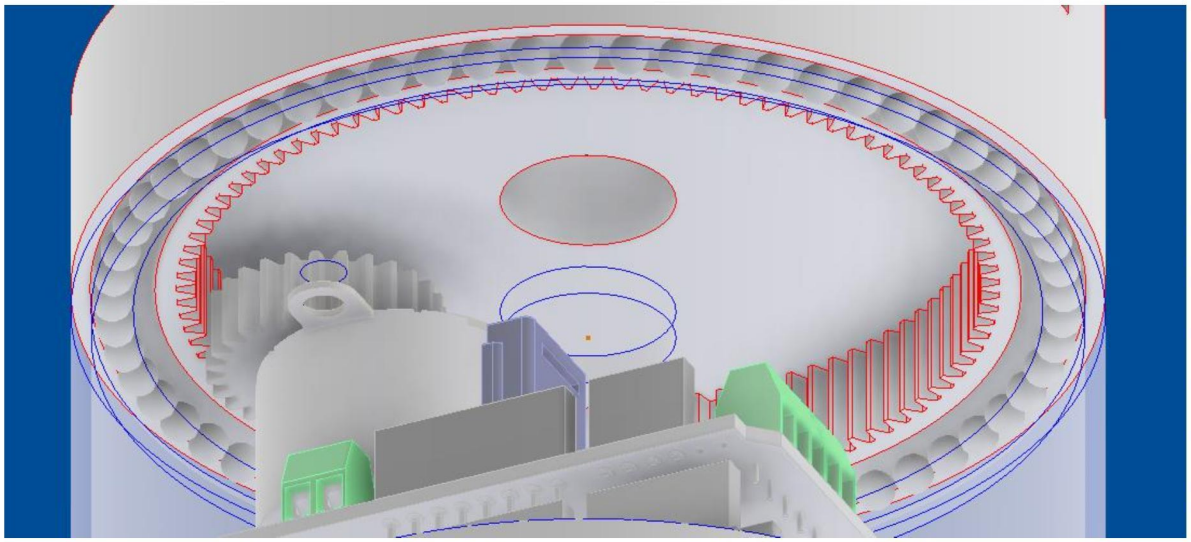


Figure 3: CAD screenshot of internal spur gear mechanism

The scanner shaft component holds the scanner together using spring clips much like a common plastic buckle (Figure 2, red). This allows for easier assembly and disassembly. The center of the shaft is hollow to allow the wires for the endstop, stepper motor, and limit switch to pass through. By running the wires through the exact center of the scanner, the effects of entanglement of the wires created by horizontal rotation is greatly reduced.

2.1.4 Vertical motion

The scanner makes vertical sweeps along the red axis shown in Figure 4. The vertical motion of the scanner uses the same stepper motor model as the horizontal motion. The vertical axis has the addition of a limit switch (Fig 4, yellow), as it is critical for the scanner to be aware of its position along the vertical sweep. The scanner has a known blind spot where it would begin to scan itself at the bottom of the sweep, therefore we must ensure that this part is not included in the scan. This blind spot occurs at 150° from the zenith.

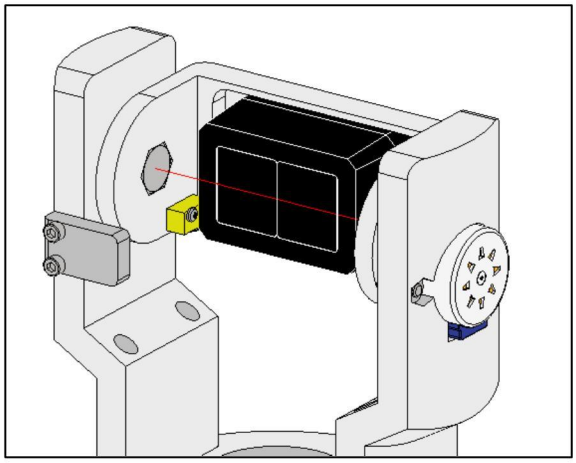


Figure 4: Vertical motion of scanner

The sensor arm is supported by the motor on one side, and the other side is supported by a 608zz bearing and M8 bolt to ensure it remains centered on the axis during rotation.

2.1.5 Mechanical backlash

During the testing phase of this project we encountered issues with backlash on the vertical axis. Every time the scanner changes direction, it takes up the backlash from the gearing inside the 28BYJ-48 stepper motor. Upon further investigation of this phenomenon, it was determined that the backlash of the motors were between 2-3°. A testing apparatus comprised of a 3D printed pointer and a protractor was attached to the stepper motor to determine this. This apparatus with the backlash can be seen in Figure 5.

Many options were considered to account for the backlash, including software calibration/compensation. The simplest solution was selected, which was to only collect scan data while traveling in the downwards direction. This ensures the backlash does not introduce differences in position to the scan data. Note this is not necessary for the horizontal motion of the scanner, as it only moves in one direction during a scan.

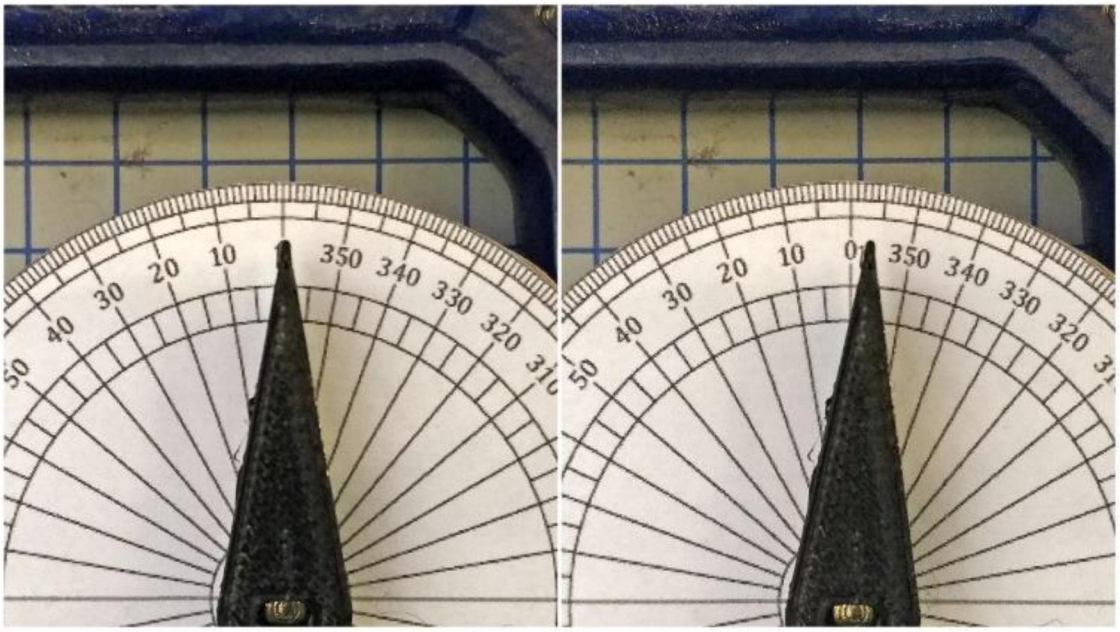


Figure 5: Mechanical backlash start and end

2.2 Electrical

The OS3DS is composed of three main electrical components. These components are the controller, the sensors, and the motion components. The scanner is built around the Arduino Uno microcontroller which houses the software and controls the instrument. Two types of sensors, LiDAR and sonar, have been provided for in electrical connections. The appropriate logic level for the LiDAR is provided with a Sparkfun Bidirectional Logic Level Converter. The scanner utilizes two 5V 28BJY-48 stepper motors for locomotion. It controls the motors via an Adafruit Stepper Motorshield V2.3.

2.2.1 Electrical components

The Arduino Uno microcontroller is an open-source package based on the Atmel Mega 328P chipset. It has an onboard voltage regulator, 14 digital I/O pins, 6 analog inputs, a USB Type B connector, and ISCP connections. It can communicate with the UART, SPI, and I2C protocols. The clock speed of the Arduino package is 16 MHz, which is created by an onboard ceramic oscillator. The electrical characteristics of the Arduino Uno are in Table 1.

Table 1: Arduino Uno Electrical Characteristics

Input Voltage Range	6-20 VDC
Operating Voltage	5 VDC
DC Current Per I/O Pin	20 mA
Maximum DC Current	200 mA
Clock Speed	16 MHz
Communication Protocols	I2C, SPI, UART
Flash Memory	32 KB, 0.5 KB used by bootloader
SRAM	2 KB
EEPROM	1 KB

Three distance sensors have been provided for in the design of the OS3DS. Two models of LiDAR sensor from Benewake, and a horn sonar from Adafruit.

The TFmini Plus single-point ranging LiDAR from Benewake was selected based on low cost, performance range, and communication capability. It can be configured to take different addresses on the I2C bus to prevent address collisions. The rangefinding light source is an LED. The electrical characteristics for the TFmini Plus LiDAR are listed in Table 2.

Table 2: Tfmini Plus LiDAR Electrical Characteristics

Input Voltage Range	4.5 – 5.5 VDC
Operating Current (Typical-Max)	≤110 – 140 mA
Sensor Operating Range	-20 °C to 60 °C
Sensor Accuracy	± 5 cm (0.1 – 6 m)
	± 1 % (6 – 12 m)
Distance resolution	5 mm
Operating range	0.1 – 12 m
Ambient light immunity	70 klux
Frame rate	1 – 1000 Hz
I²C Clock Speed (Max)	400 kHz
I²C Slave Address	0x10

The TF02-Pro single-point ranging LiDAR from Benewake was selected based on mid-tier cost, a large distance range, and communication capability. It can be configured to take different addresses on the I2C bus to prevent address collisions. The rangefinding

light source is a vertical-cavity surface-emitting laser. The electrical characteristics for the TF02-Pro LiDAR are listed in Table 3.

Table 3: TF02-Pro LiDAR Electrical Characteristics

Input Voltage Range	5 – 12 VDC
Operating Current (Typical-Max)	≤200 – 300 mA
Sensor Operating Range	-20 °C to 60 °C
Sensor Accuracy	± 5 cm (0.1 – 5 m)
	± 1 % (5 – 40m)
Distance resolution	1 cm
Operating range	0.1 – 40 m
Ambient light immunity	100 klux
Frame rate	100 Hz
I²C Clock Speed (Max)	400 kHz
I²C Slave Address	0x10

The DYP-A01 Ultrasonic Distance Measuring Controller from Adafruit was selected based on low cost, and sensor accuracy. It has a horn attachment on the transducer face to allow for better directional measurement. The electrical characteristics of the DYP-A01 Ultrasonic are listed in Table 4.

Table 4: DYP-A01 Ultrasonic Electrical Characteristics

Input Voltage Range	3.3 – 5.0 VDC
Operating Current (Typical)	<10 mA
Sensor Operating Range	-15 °C to 60 °C
Sensor Accuracy	± (1 cm + 1.003 measured)
Operating range	28 – 750 cm
Work cycle	100 – 500 ms
Baud rate	9600 bps

A Sparkfun Bidirectional Logic Level Converter was selected to convert the I2C bus from TTL to LVTTL for the LiDAR sensors. The package is built around Fairchild Semiconductor's BSS138 N-channel logic level enhancement mode field effect transistor. The datasheet for this transistor package can be found in Appendix C. Figure 1 shows the schematic for one of the four channels included in the package.

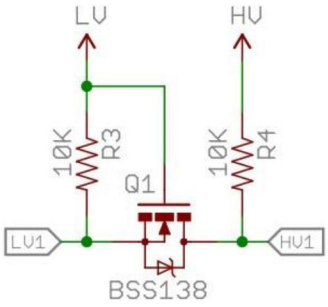


Figure 6: Electrical schematic of bidirectional LLC channel

Two 28BYJ-48 5V stepper motors were selected to provide a means of locomotion for the scanner. This selection was based on extremely low cost, market availability, as well as suitable torque and performance characteristics. The 5V model allows for a modern high-current USB adapter, which are now extremely common due to modern smartphones, to be attached to the barrel jack power supply. The stepper has an internal gearbox providing a 64:1 gearing ratio. The electrical characteristics for the 28BYJ-48 stepper motor are listed in Table 5.

Table 5: 28BYJ-48 Electrical Characteristics

Input Voltage Range	5.0 VDC
Number of Phases	4
Stride Angle	5.625° /64
Frequency	100 Hz
DC resistance	50 Ω ±7 %
Friction torque	600 – 1200 gf.cm
Pull in torque	300 gf.cm
Temperature Rise	< 40K (120 Hz)

The Adafruit Motorshield V2.3 was selected to interface with the stepper motors. It is built around Toshiba's TB6612FNG IC for driving the stepper motors. The datasheet for this package can be found in Appendix C. The motorshield can communicate with the I2C protocol. The electrical characteristics for the TB6612FNG are listed in Table 6.

Table 6: TB6612FNG Electrical Characteristics

Supply voltage (V_M)	15 V
Output voltage (V_{OUT})	15 V
Output current (I_{OUT})	1.2 A (3.2 A peak)
Operating temperature	-20 to 85 °C
Switching frequency	100 kHz
I2C Address	0x60-0x80, jumper selectable

2.2.2 Communication

The scanner utilizes an I2C bus to communicate between the controller, motorshield, and LiDAR sensors. The I2C bus operates at the TTL logic level, at 5 V. The LiDAR sensors require the LVTTL logic level, at 3.3 V. To accommodate for this, a logic level converter was used on the LiDAR portion of the bus.

The DYP-A01 Ultrasonic uses UART for communication. As the Arduino Uno only has one built-in UART controller, a second UART was added in software by utilizing the softwareSerial library. This allows the scanner to communicate all scan data to the control application via the built-in UART, while still being able to communicate with the ultrasonic sensor without necessitating bus management to prevent collisions.

2.2.3 Electrical connections

Several electrical connections were made manually to assemble the scanner. The motorshield has a suitable form factor to sit atop the Arduino with pin headers. These headers must be soldered on by the assembler. The connections from the Arduino to the bidirectional logic level converter to provide the LVTTL to TTL conversion must be made by the assembler. The 28BJY-48 stepper motors must have their leads attached to the motorshield. The horizontal direction motor is attached to motor port 1, and the vertical direction motor is attached to motor port 2. This attachment must be done by the assembler.

Finally, the motorshield requires a separate power supply than the Arduino. The Arduino can draw power over the USB connection to the computer hosting the control application. The motorshield must be provided power from a separate USB power supply. The design of the scanner utilizes a barrel-jack connection to adapt from a USB-A connection to the power terminals of the motorshield. This connection must be done by the assembler.

The details of the electrical connections can be found in the schematic diagram of the scanner. This schematic can be found in Appendix B.

2.3 Software

There are two primary pieces of software that were written to control and obtain data from the OS3DS. First, there is the firmware, which is loaded onto the Arduino Uno microcontroller and provides for data acquisition from sensors as well as locomotion functionality. Second, there is the control application, which runs on the host computer. This control application provides data acquisition from the scanner, a user interface to the scanner, as well as data processing and export functionality.

2.3.1 Firmware

The controller firmware is written for the Arduino controller in C++. The firmware has several key functions. Upon startup, the Arduino bootloader calls the `setup()` function once. It then enters the `loop()` function, which loops indefinitely until an event is registered on the serial bus. When an event is registered, the `serialEvent()` function is called. This control behavior is shown in Figure 7.

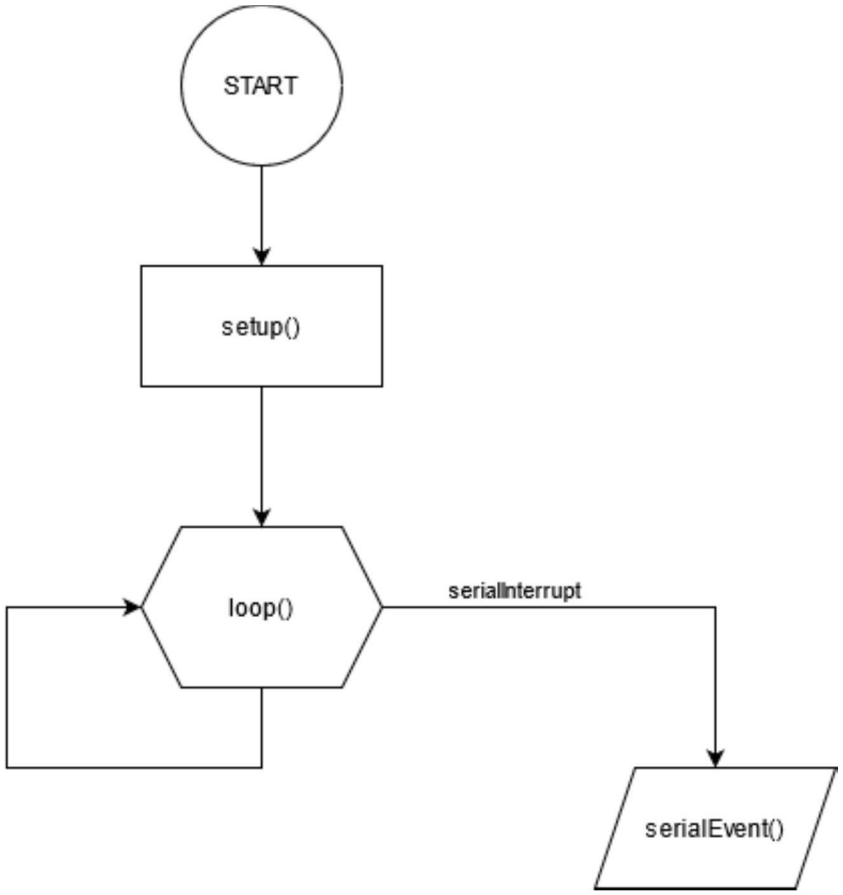


Figure 7: Startup routine

The `setup()` function initiates the primary serial connection between the controller and the USB connection to the application. It then instantiates the motorshield object in memory, and sets the stepper motor speed parameters. Finally, it configures the pins for the limit switch and software serial connection for a possible sonar sensor connection. The `setup()` function is shown in Figure 8.

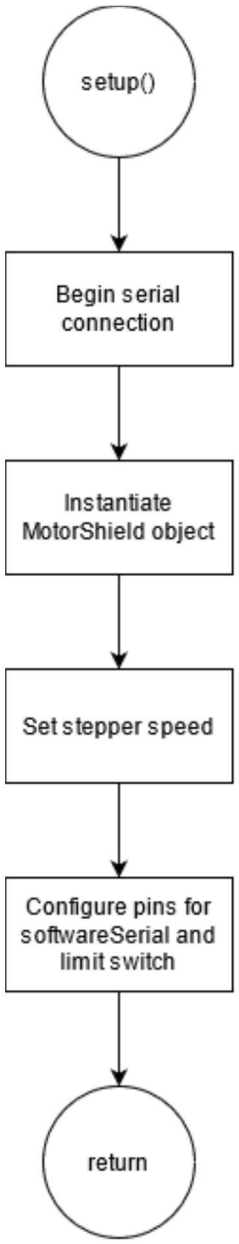


Figure 8: The setup() function

On a `serialEvent`, the controller will read one byte from the serial bus. This byte is interpreted as a command code and will initiate further behavior. This is achieved with a `switch...case` statement. There are four commands that can be triggered in this way:

- Configuration mode
- Jog mode
- Scan mode
- Rangefind mode

In Configuration mode, within the function `codeAConfigure()`, the controller calls several helper functions to read and confirm configuration data from the serial bus. `getConfiguration()` reads delimited configuration parameters from the serial bus and stores their values in global variables. `confirmScan()` then communicates those parameters back to the control application to confirm the configuration has been successfully loaded.

If the parameters have been loaded, `codeAConfigure()` continues into a `switch...case` that evalutes based on the configured sensor type. If the sonar sensor has been configured, a `softwareSerial` connection is established and set to `listen()`. If the lidar sensor has been configured, a `Wire` connection is established and a configuration frame is sent to the lidar sensor to change the measurement unit from cm to mm.

Finally, the function `calcScanPoints()` is called to determine how many points will be scanned in the final cloud and then the `homingSequence()` is called to initialize the stepper motor positions. Now that the scanner is configured, the function passes back to `loop()`. The `codeAConfigure()` function can be seen in Figure 9.

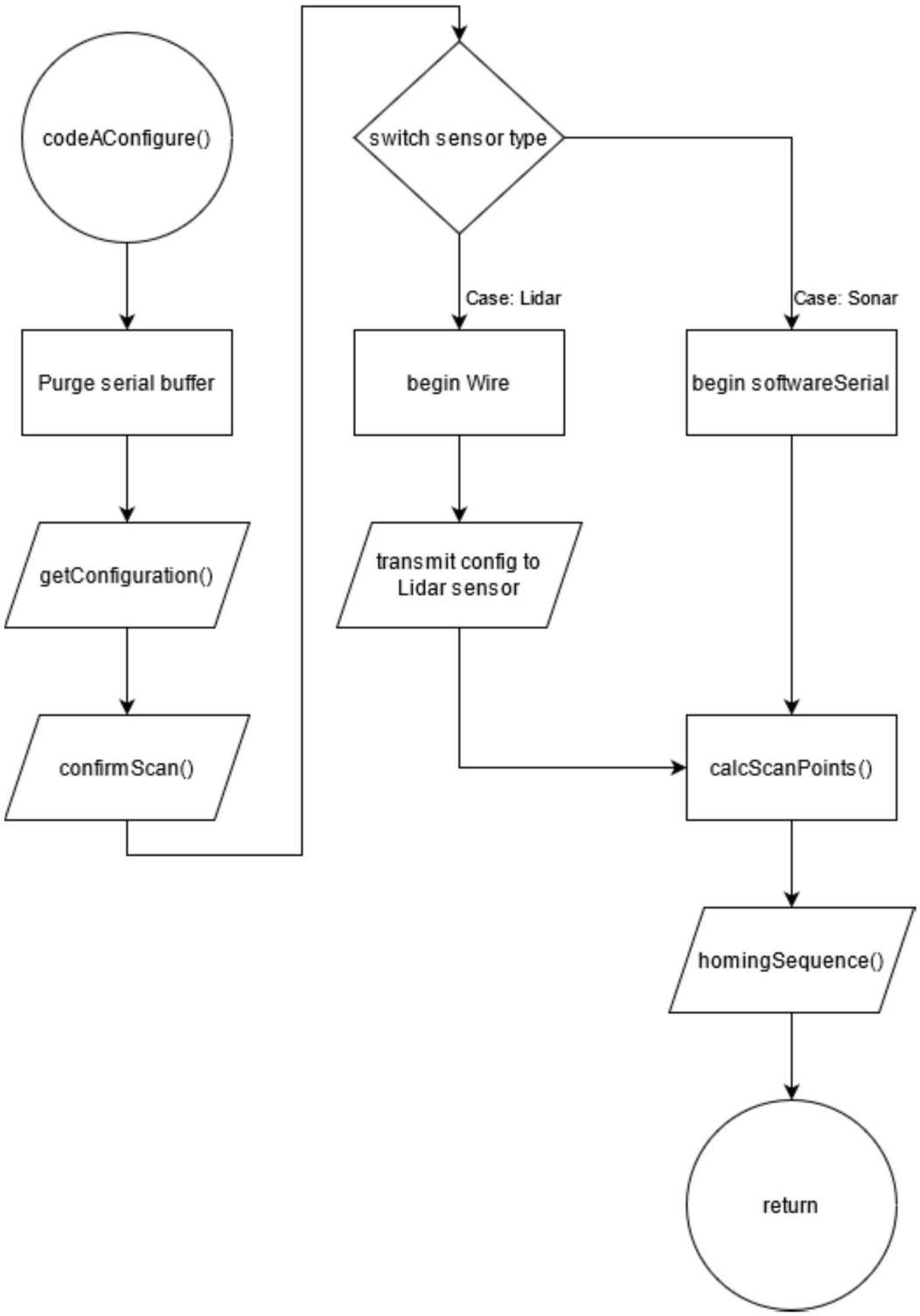


Figure 9: The codeAConfigure() function

In Jog mode, within the function `codeBJog()`, the controller reads a single byte from the serial bus. This byte is the argument for a `switch…case` that determines which direction the scanner should move. A corresponding command is then passed to the motorshield object to move the relevant stepper motor a set number of steps. The function passes back to `loop()`. The `codeBJog()` function can be seen in Figure 10.

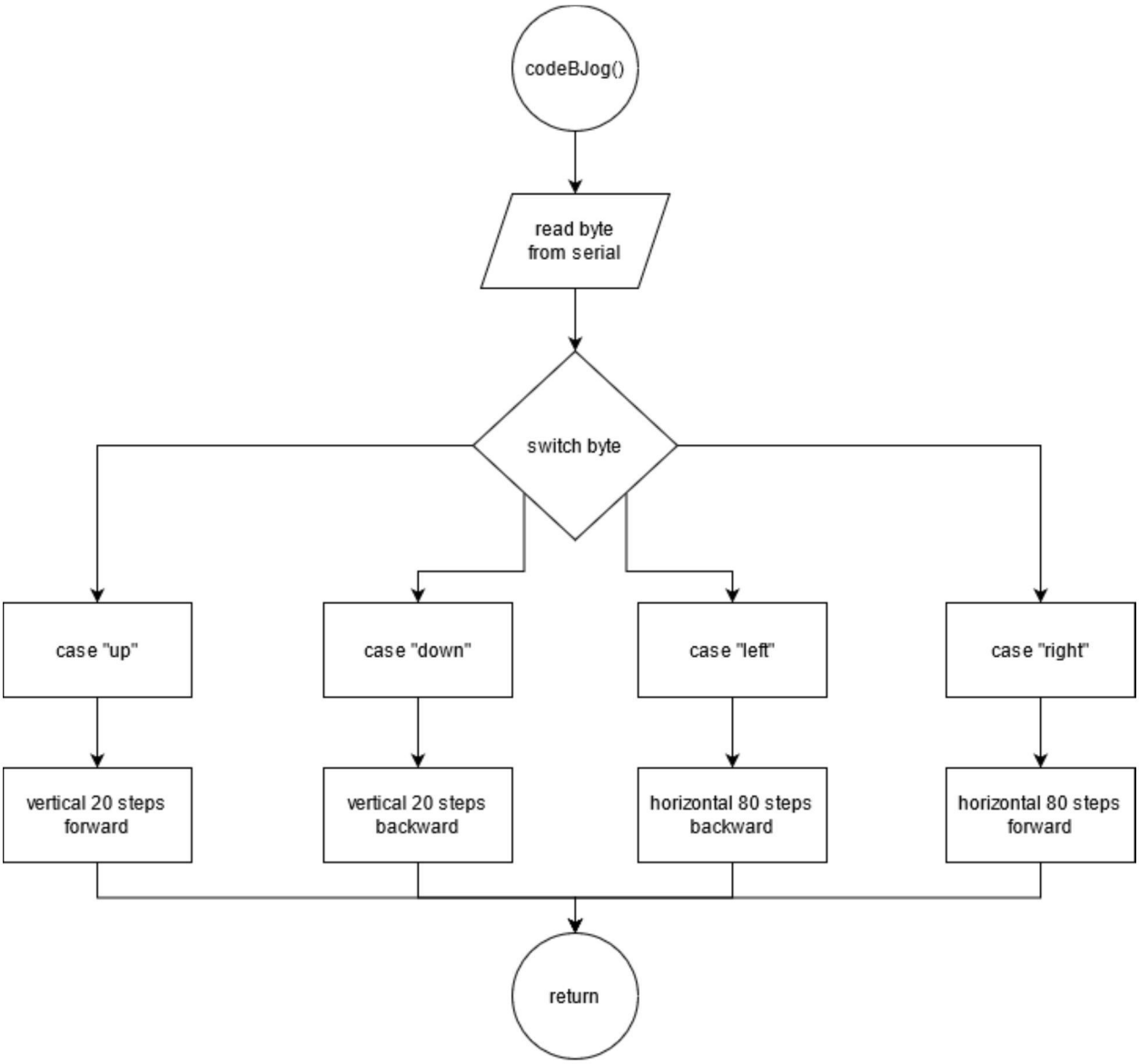


Figure 10: The codeBJog() function

In Scan mode, within the function `codeCScan()`, the controller enters a `while` loop. The loop will continue until the scan has been completed. First, the helper function `scanPoint()` is called to determine whether the scanner should register a data point at the current heading. This function only returns `false` if point interpolation has been enabled, otherwise it only returns `true`.

Following this determination, the relevant function is called to read the distance data from the configured sensor. If data validation has been enabled, distance data will only be recorded when the checksum of the sensor transmission matches the computed checksum of that transmission. The number of points scanned is updated. The current distance data and stepper motor position is then transmitted over the serial bus to the control application with a checksum of the transmission appended. Another helper function, `moveScanner()`, is called. This function moves the scanner to the next position based on the current scan's resolution and angle settings.

Once the anticipated number of points has been scanned, the `while` loop is broken. The scanner is rotated horizontally back to the initial position to prevent wire entanglement. A message is transmitted on the serial bus indicating the scan has completed. If data validation has been enabled, the statistics of failed data acquisitions is also transmitted. Finally, the stepper motors are released with the `release()` command. As the scan has been completed, the function passes back to `loop()`. The `codeCScan()` function can be seen in Figure 11.

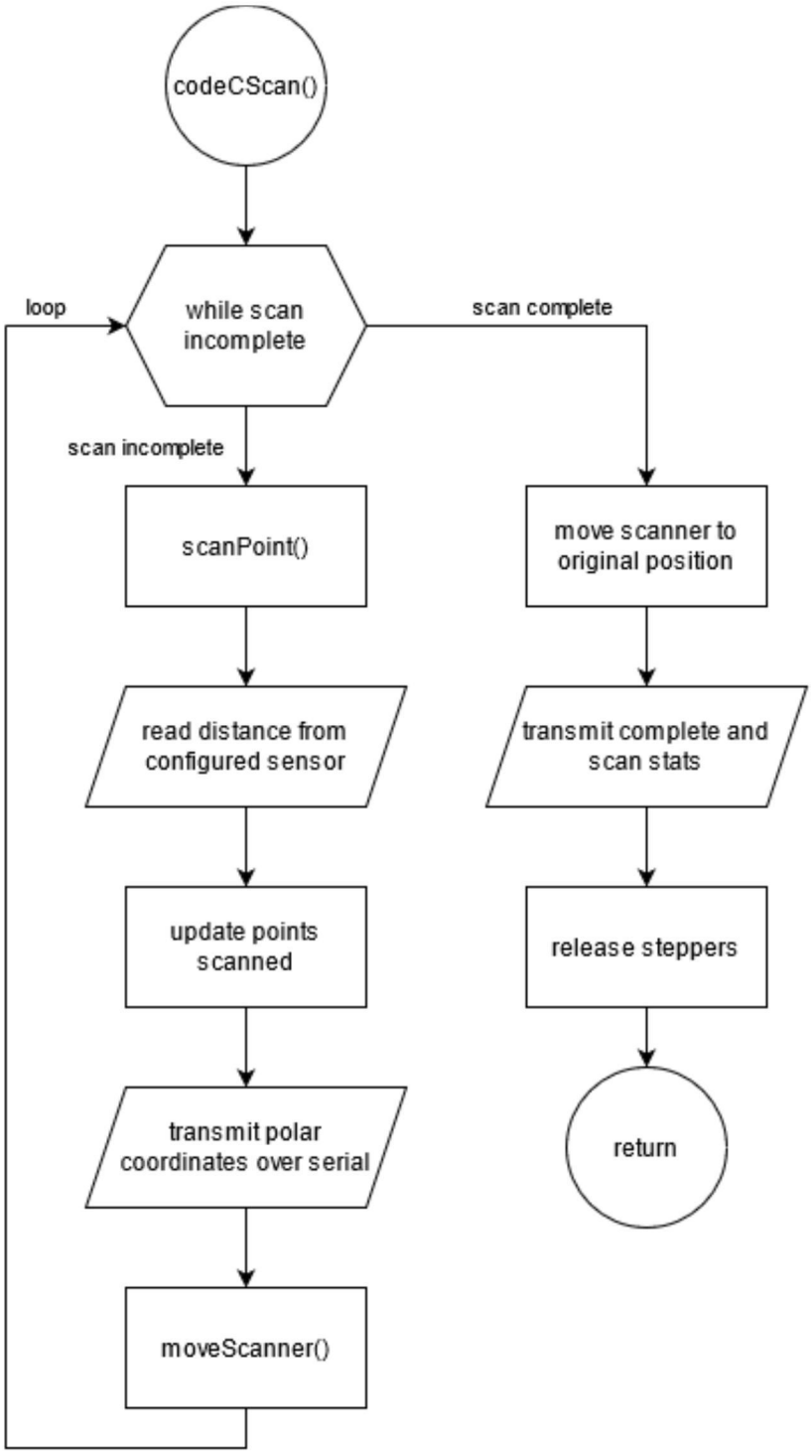


Figure 11: The codeCScan() function

In Rangefind mode, within the function `codeDRangeFind()`, the controller reads a byte from the serial bus. This byte is interpreted as the sensor the controller should read from. The controller then reads the current distance data of the selected sensor and prints it to the serial bus. The function then passes back to `loop()`. The `codeDRangeFind()` function can be seen in Figure 12.

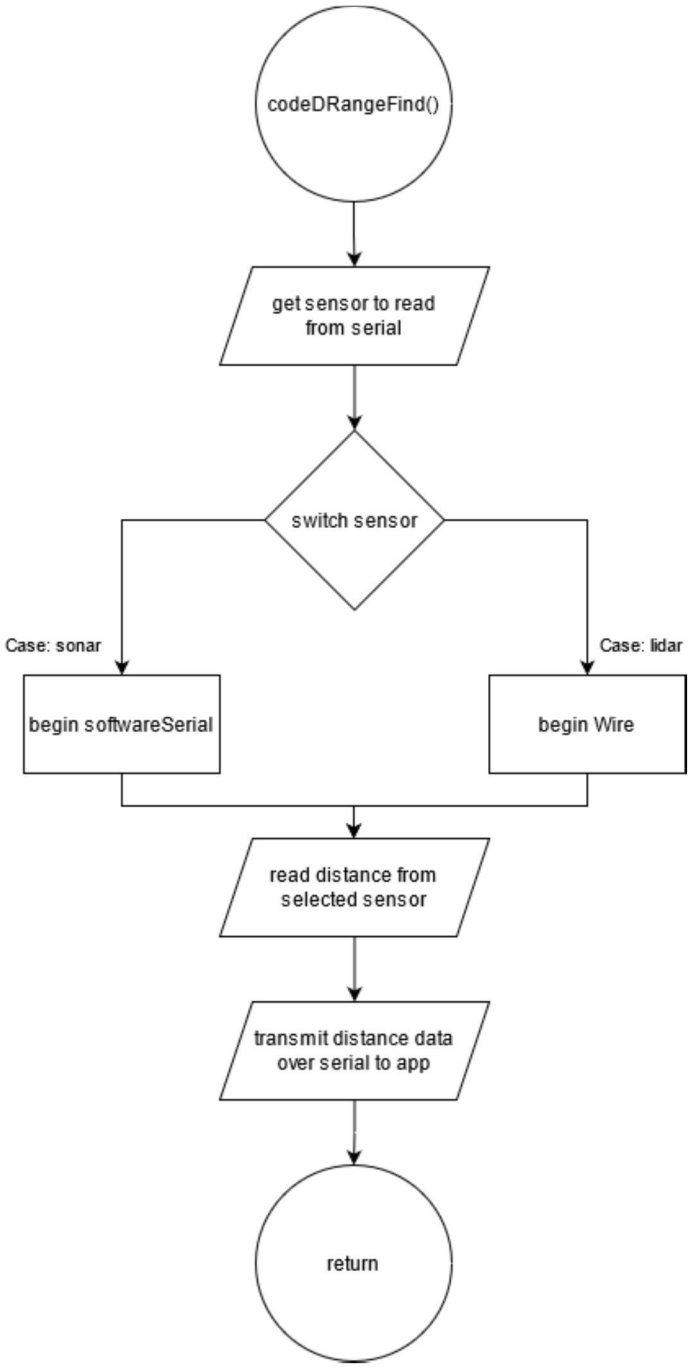


Figure 12: The codeDRangeFind() function

2.4.2 Control application

The control application is written in Python. The application has three primary functions. It connects to the scanner and subsequently reads the data stream the scanner outputs over USB. It stores the scanner data stream and outputs it in a standard format. Lastly, it provides as a direct interface to the scanner, capable of both configuration and control.

The application has a graphical user interface. This GUI was made with the `PyQt5` library from Riverbank Computing [8]. The GUI can be seen in Figure 13.

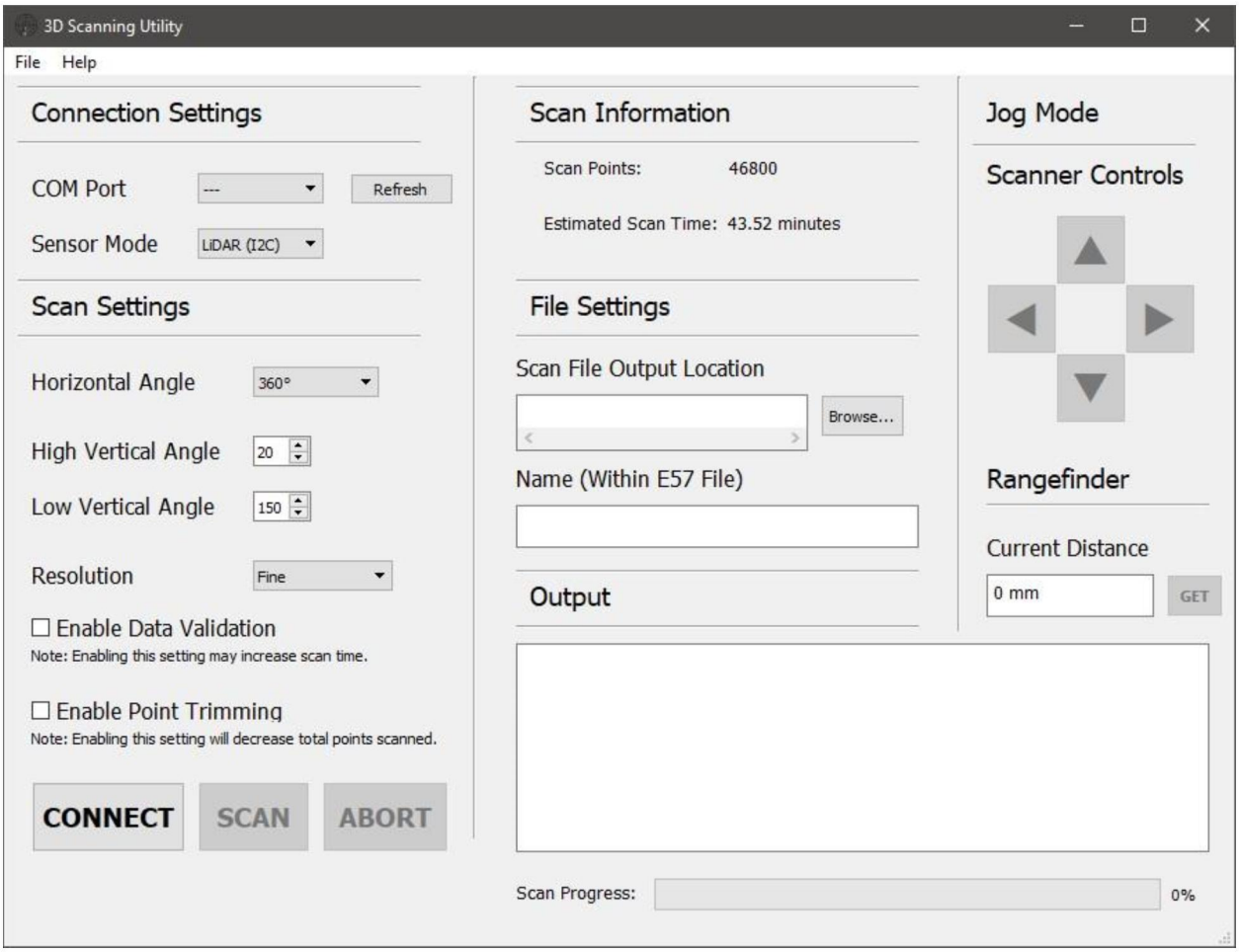


Figure 13: The control application GUI

The application provides methods to save and load the current scan settings. It saves the current configuration to a `.3DSCANCFG` file. This filetype is no different than a `.csv`, but has a custom extension to aid users in selection of configuration files.

To obtain the current connected COM ports, the application uses the `tools.list_ports` function from the `PySerial` library. `PySerial` provides several tools for working with different operating systems and their USB connections.

The desired configuration for the scanner is simply selected from the provided combo, spin and check boxes. The horizontal angle is the angle the scanner sweeps on the horizontal axis. The high and low vertical angles are the upper and lower limit the vertical motor is bounded by, where the zenith of the scanner corresponds to 0° and the nadir is 180°. Resolution is a multiplier that is applied to both vertical and horizontal divisions. The "Fine" resolution means that one point will be acquired per degree in each axis. As the resolution increases, the number of points per degree will double with each step. A decrease in resolution causes the number to halve each step. Data validation will cause the scanner to ignore distance data from sensors if the transmission from the sensor does not pass a checksum comparison and will poll the sensor until the transmission passes. Point trimming will selectively skip points near the poles of a scan to keep point density more uniform. A representation of point trimming can be seen in Figure 14.

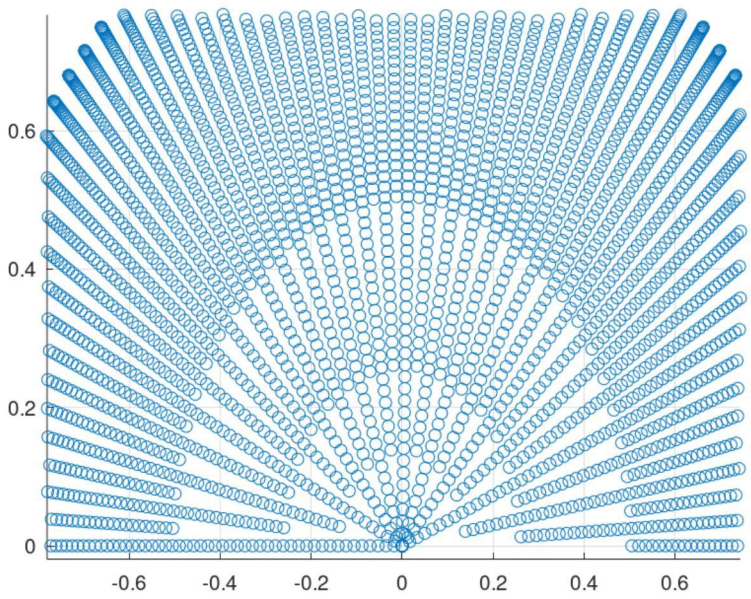


Figure 14: A view from the north pole of a unit sphere with point trimming applied

The CONNECT, SCAN and ABORT buttons are the primary points of interaction for a scan. CONNECT opens a serial connection on the host computer on the selected COM port. SCAN is enabled only when a live serial connection has been established. ABORT

is enabled only when a scan is currently taking place. The Output box displays ongoing information from the scan process. This information includes connection information, current points scanned, as well as scan statistics when a scan has completed.

Scan Information shows calculated scan properties. The estimated scan time is based on average point acquisition time from multiple scans. File Settings gives users control over where the application will export the completed scan file to, as well as the name of the point cloud within the exported `.E57` file.

Jog Mode and Rangefinder are two direct control and interaction features. When a serial connection is established, both functions are enabled. The directional buttons of Jog mode will command the scanner to move a small amount in the selected direction. When GET of Rangefind is pressed, it will command the scanner to communicate the current distance reading of the selected sensor.

The process of completing a scan has multiple steps. The `scan` method first calls helper function `sendConfiguration` to send and confirm the configuration within the scanner. A `while` loop is entered until the scan has completed. The method reads the data from the serial stream, looking for message from the scanner indicating the scan is complete. Any data that is read is stored locally to several ordered lists and is also written to a `.txt` file in the `.pts` format. If a `SerialException` is raised, the scan is aborted, or the scan completes successfully, the collected scan data is processed and exported via the `.E57` API. The outcome of the scan is communicated with the Output box.

The collected scan data is converted from polar coordinates to cartesian coordinates with the formulas:

$$x = r \cdot \sin\phi \cdot \cos\theta$$

$$y = r \cdot \sin\phi \cdot \sin\theta$$

$$z = r \cdot \cos\theta$$

Where ϕ is the angle from the z-axis and θ is the angle from the x-axis.

The data is exported to an `.E57` file with a new class, `E57Cust`, that inherits from the `pye57.E57` class [9]. E57Cust provides more control over the data within a scan header, and removes checks for certain objects within the `write_scan_raw` method. The `pye57` library is a wrapper for the `libe57` library, which is written in C++. `libe57` is an open-source API for creating `.E57` files. It was created as a part of the ASTM committee development of the `.E57` standard.

3. Testing

In order to establish the accuracy of the final design, a model space was chosen. This model space was topologically regular and has a couple of distinct features. The model space used was a portion of a room in one of the researcher's homes. The model space can be seen in Figure 15.

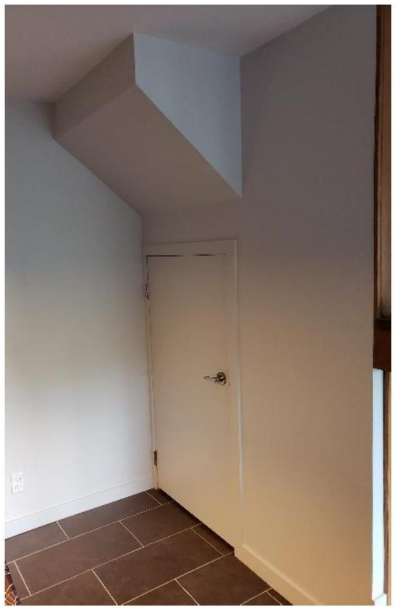


Figure 15: Model space

A "Very Fine" resolution scan was performed on the model space. A "Very Fine" resolution scan of a 90° horizontal angle yields approximately 67700 individual data points. This scan took approximately 24 minutes. The scan data acquired can be seen in Figure 16.

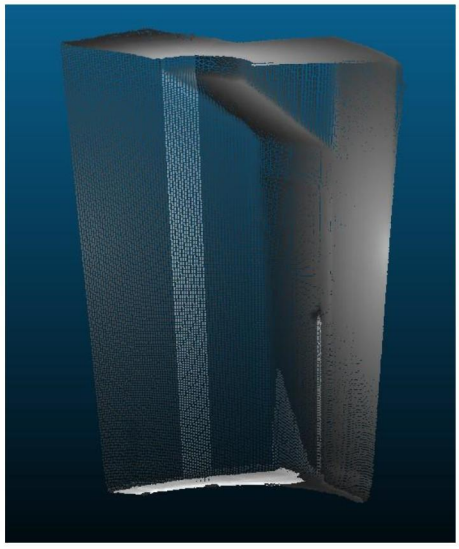


Figure 16: Scan data from model space

The model space was recreated digitally with independent measurements. This will enable a direct comparison between the digital model space and the point cloud generated by the scanner. The digital model space can be seen in Figure 17.

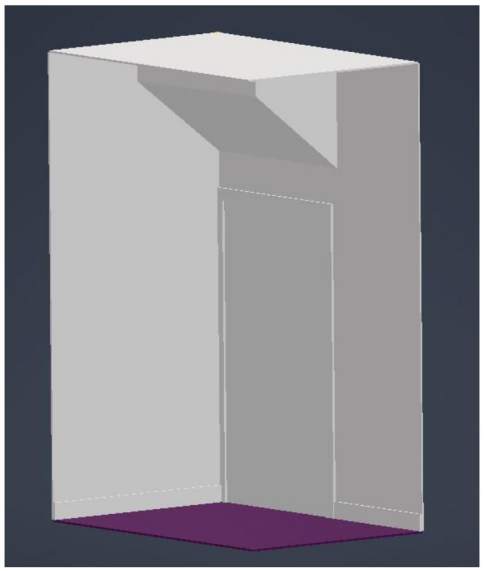


Figure 17: Digital model space

The scan object and the digital model space were overlaid in the CloudCompare software, which was obtained from [10]. The point cloud had a scalar field applied to it, which was generated from the intensity of the LiDAR sensor at that data point. The overlay of the two objects can be seen in Figure 18.

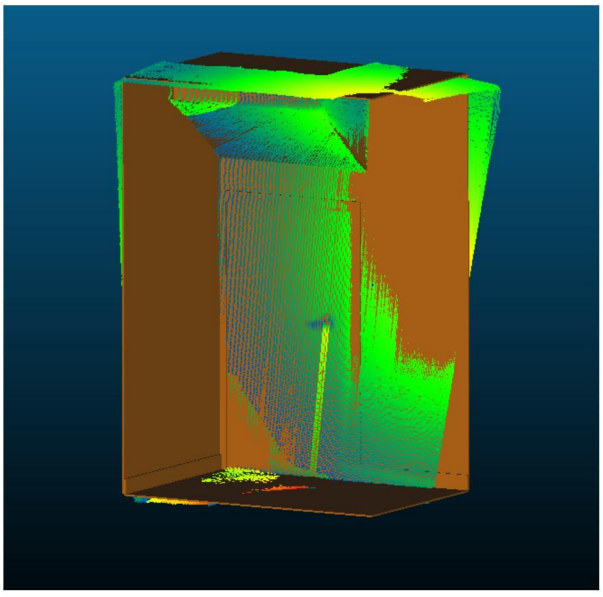


Figure 18: Overlay of scan data on digital model space

The portions of the scan data that were outside the bounds of the digital model space were removed. Once these points were removed, the normals of the point cloud were computed using an 8-neighbor minimum spanning tree algorithm. With the orientation of each point calculated, the distance from each point in the cloud to the nearest polygon on the digital reference space was computed. This was achieved with a nearest neighbor algorithm. The distance from the digital model space was applied to the point cloud as a scalar field in a heatmap. This can be seen in Figure 19.

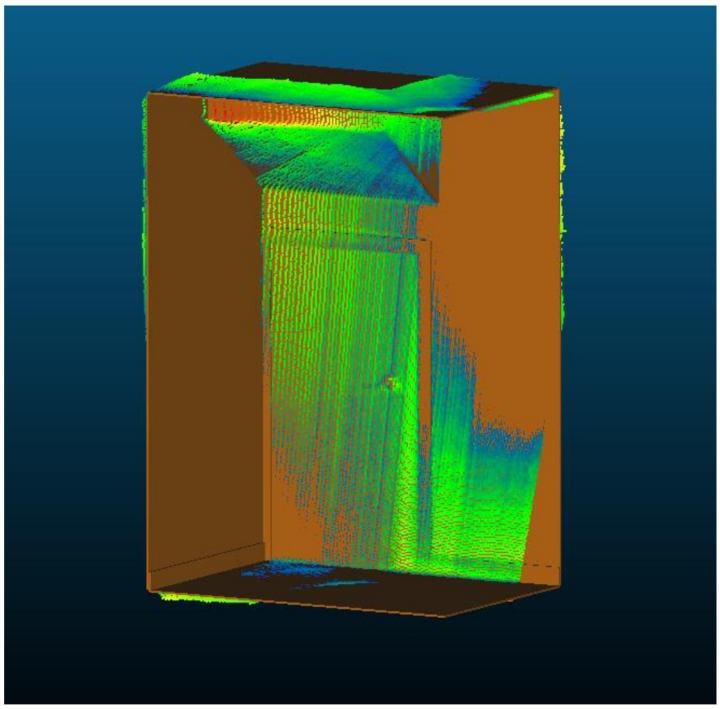


Figure 19: Scan data and digital model space with distance heatmap applied

The distances were put into a histogram and a normal distribution curve was fitted to the data. The mean distance from the digital model space was found to be 14.02 mm. The standard deviation of the distribution was found to be 12.54 mm. This histogram with the distribution curve can be seen in Figure 20.

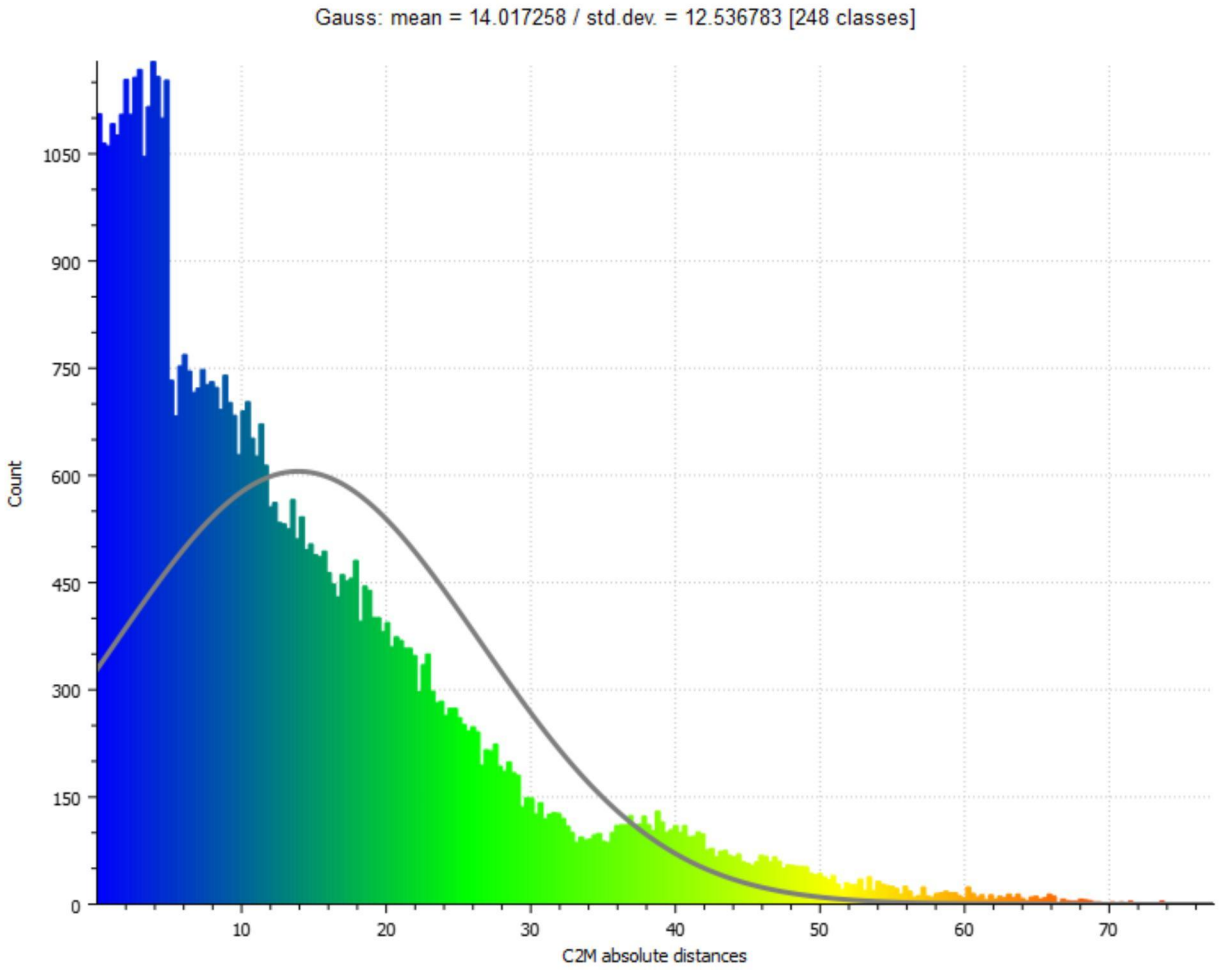


Figure 20: Normal distribution of distance data

4. Summary and Conclusions

The OS3DS is composed of many elements. It has a 3D printed body and geartrain system, which are made of appropriate filaments for the expected operating temperatures. The design of the body has been optimized to reduce 3D print failures and for easy assembly. Included in the body is an attachment thread that is compatible with standard camera tripods for ease of mounting. The 3D printed geartrain rotates atop a v-shaped ball bearing race to ensure smooth rotation. The locomotion is driven by two 5V 28BJY-48 stepper motors.

The OS3DS has an Arduino Uno microcontroller for managing data acquisition, communication, and locomotion. It can be outfitted with one of the three designed sensors: the Benewake TFmini Plus LiDAR, the Benewake TF02-Pro LiDAR, or the DYP-A01 Ultrasonic sensor. There are two pieces of software that accompany the OS3DS: the

firmware, and the control application. The firmware of the scanner is loaded onto the Arduino Uno and is written in C++. The control application is run on a host computer and is written in Python. The two pieces of software were specifically designed to interact. This software enables user-driven scan configuration, output file settings, saving scan configuration to file, jogging the scanner, and using the scanner for rangefinding. The application is able to save scan data in the form of a point cloud in the `.E57` file format, as well as an ASCII-based `.PTS` format.

All mechanical designs and software of the OS3DS are available online and are released as open-source under the GNU GPL v3.0 license. An effort has been made to ensure that all non-printed parts are widely available from major parts suppliers and distributors.

In future iterations of the design, many things can be done to improve on the OS3DS. At present, the OS3DS has been designed to be modular. New modular attachments can be designed and manufactured. In no particular order, these modules could include:

- Onboard data processing with current hardware or other computer
- Onboard battery pack for fully remote operation
- Built-in storage with SD card module
- Bluetooth integration for control from a mobile phone
- Accelerometer and gyroscope for "pose"-based scan positioning
- 12V stepper motors of similar model

During testing, scan aberrations were found in most configurations. These aberrations include spherical distortion of points near the poles of scans. This could be due to mechanical limitations of the stepper motors used, or data conversion techniques applied after the scan had completed. Diagnosing these aberrations and mitigating their influence on scans would be a valuable improvement to the present design.

The backlash in the stepper motors presented a persistent challenge through the design process. This challenge demanded a trade-off be made: scan times were elongated, to ensure accuracy. This compromise entailed only scanning in one vertical direction, from top to bottom, so as to eliminate the backlash. A further improvement in the design could be done to reduce scan times furter, while maintaining accuracy.

A current limitation is the ease by which scan data can be manipulated. Future work could also include adding data security measures to the OS3DS, to enable more robust data integrity. This improvement has the potential to pave the way to the OS3DS becoming a low-cost, professional-grade instrument.

The OS3DS was tested against a model space that had regular topology. In those tests, the scanner was compared to a digital recreation of the model space. The results of these tests yielded a mean deviation of 14.02 mm and a standard deviation of 12.54 mm from the digital model. While these tests are promising, further testing is required before a high degree of confidence in the measurement ability of the OS3DS can be claimed. More prototypes of this design should be manufactured and tested in a variety of conditions.

References

[1] W. B. Yang, Y. N. Yen, and H. M. Cheng, "Digitalized preservation and presentation of historical building – taking traditional temples and dougong as examples," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-5/W3, pp. 371-376, 2015, doi:10.5194/isprsannals-II-5-W3-371-2015.

[2] B. Kwoczynska, U. Litwin, I. Piech, P. Obirek and J. Sledz, "The Use of Terrestrial Laser Scanning in Surveying Historic Buildings," 2016 Baltic Geodetic Congress (BGC Geomatics), Gdansk, Poland, 2016, pp. 263-268, doi: 10.1109/BGC.Geomatics.2016.54.

[3] N. Brown, R. Laing, and J. Scott, "The doocots of Aberdeenshire: An application of 3D scanning technology in the built heritage," *Journal of Building Appraisal*, vol. 4, no. 4, pp. 247-254, doi: 10.1057/jba.2009.9.

[4] P. Shan et al., "A laser triangulation based on 3D scanner used for an autonomous interior finishing robot," 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO), Macau, 2017, pp. 475-480, doi: 10.1109/ROBIO.2017.8324462.

[5] Z. Zhang and L. Yuan, "Building a 3D scanner system based on monocular vision", *Applied Optics*, vol. 51, pp. 1638-1644, 2012, doi: 10.1364/ao.51.001638.

[6] N. Gift, " PROJECT 084: - MICROCONTROLLER BASED IMPEDANCE METER FOR INDUCTOR, RESISTOR AND CAPACITOR", B.Sc. thesis, Dept. Electrical and Information Engineering, University of Nairobi, Nairobi, Kenya, 2016.

[7] *ISO 1222:2010 Photography – Tripod connections*, ISO 1222:2010, 2020. [Online]. Available: https://www.iso.org/standard/55918.html

[8] Riverbank Computing, *PyQt5*, ver. 5.15.4. [Online]. Available: https://pypi.org/project/PyQt5/#files, Accessed on: May 31, 2021.

[9] D. Caron, *pye57*, ver. 0.2.3. [Online]. Available: https://pypi.org/project/pye57/#files, Accessed on: June 2, 2021.

[10] D. Girardeau-Montaut, "CloudCompare – Open Source project", CloudCompare. [Online]. Available: https://www.cloudcompare.org/, Accessed on: June 18, 2021

APPENDIX A: Code

ARDUINO FIRMWARE

```
/*
 * Open-Source 3D Scanner Project
 * Summer 2021
 * Darren Paetz, Matt Kantor, Philip Mees
 * MacEwan University, Edmonton, Alberta, Canada, 2021
 *
 * This firmware is designed to be used with the 3D Scanner Interface application.
 * That application will gather the data from the scanner and convert it to an
 * industry standard file format (E57).
 *
 *
 */

#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include <SoftwareSerial.h>

/***************
  Globals
***************/
// --Scanner--
// Parameters
unsigned int sensor_type = 0; // Parameterization from app - MEMSAVE: byte
unsigned int scan_angle = 0; // Parameterization from app
float scan_resolution = 0; // Parameterization from app
// Scanner angle limits
//  Angle phi is measured downwards from the zenith, relative to the vertical axis of the scanner
//  Straight up (Zenith) -> 0°   Straight down (Nadir) -> 180°
//  Max vertical (highside_angle): 0, but realistically should be 15 degrees.
//  Min vertical (lowside_angle): 150, limited by scanner body.
unsigned int highside_angle = 20;  // Parameterization from app
unsigned int lowside_angle = 135;  // Parameterization from app
unsigned int scan_point_trimming = 0;  // Parameterization from app
unsigned int scan_data_validation = 0; // Parameterization from app
// Scanner data collection
unsigned int distance = 0; // Distance will be used by either sensor type, and takes the same
form in memory.
unsigned int intensity = 0; // Strength will be used exclusively by the LiDAR, both TF02 Pro and
TFMini
// Scan statistics and flags
bool scan_complete = false;  // Flag for scan completion status
bool scanner_configured = false;  // Flag for scanner successful configuration
byte scanner_current_line = 1;  // Tracks current line to adjust for point interpolation.
unsigned int scan_points_acquired = 0;  // Tracks how many points have currently been scanned.
unsigned long anticipated_points = 0;  // Calculated number of points scan parameters should
yield.
unsigned int validation_fails = 0;  // Tracks amount of failed transmissions from sensors
unsigned long vertical_step_cnt = 0;  // Tracks the number of steps to travel back to the top

// ----External Devices----
// --Steppers--
// We'll be using the steppers in full step mode, meaning we get 2037 steps per revolution
// This is based on independent findings that the 28BYJ-48 stepper has 4074 steps per full
revolution in half step mode
const int revolution_steps = 2037; // 2037 steps per revolution of stepper (11.311 degrees/64:1
gear ratio)
const float horiz_deg_steps = 20.0871; // 20.0871 steps per horizontal degree (with scanner
gearing)
const float vert_deg_steps = 5.6583; // 5.6583 steps per vertical degree (only stepper gearing)
const int full_steps = 7342; // 7342 steps per full horizontal revolution of scanner
float horiz_position = 0; // Initialize horizontal position tracking variable
```

```
float vert_position = 90; // Initialize vertical position tracking variable, will be set during
homingSequence
bool vert_motor_direction = true; // Vertical motor direction: False is up, True is down

// --LiDAR--
// Default address of the LiDAR is 0x10 (DEC: 16)
// MEMSAVE: Place only in PROGMEM
int lid_addr = 0x10; // Leave as variable so it can be configured to prevent address collisions

/********************
 Stepper object setup
********************/
//Use default I2C addr 0x60 (DEC: 96)
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
//Create stepper object with (Steps/rev,Motor Port) on Motor Port 1 (M1&M2) and 2 (M3&M4)
Adafruit_StepperMotor *horiz_Stepper = AFMS.getStepper(revolution_steps,1);
Adafruit_StepperMotor *vert_Stepper = AFMS.getStepper(revolution_steps,2);

/******************
 Physical I/O setup
******************/
// Reserve serial connection for the sonar if used
SoftwareSerial sonar_Serial(2,3); // RX 2, TX 3
// Limit switch for vertical axis
const int limit_pin = 7;

/*****************
 Initial functions
*****************/
void setup() {
  // Open serial connection to the application.
  Serial.begin(115200);
  // We'll always need motorshield communication, start it immediately.
  AFMS.begin();
  // Default Stepper Speed is 10 RPM
  horiz_Stepper->setSpeed(10);  // TEST: Change to 15 rpm?
  vert_Stepper->setSpeed(10);  // TEST: Change to 15 rpm?
  // Set pins for limit switch and sonar serial connection
  pinMode(limit_pin, INPUT_PULLUP);
  pinMode(2, INPUT);
  pinMode(3, OUTPUT);
}

void serialEvent(){
  /*  The first character of any serial transmission will be a command code.
   *  A: Configure scanner
   *  B: Jog motors
   *  C: Scan
   *  D: Rangefind
   *  serialEvent is only called at the end of loop(), so we don't need to worry about
   *  potentially interfering with a configuration frame once in that function.
   */
  char command_code = (char)Serial.read();
  switch(command_code){
    case 65:
      codeAConfigure();
      break;
    case 66:
      codeBJog();
      break;
    case 67:
      codeCScan();
      break;
    case 68:
      codeDRangefind();
      break;
  }
}

/*************************
 Configuration (A) functions
```

```
**************************/

void codeAConfigure(){
  bool get_config = false, begin_scan = false;
  // Purge the serial buffer to prepare to read configuration data
  while(Serial.available()){
    char byte_dump = Serial.read();
  }
  // Until we can successfully read a configuration frame from the application, do nothing.
  while(get_config == false){
    get_config = getConfiguration();
  }
  // Until we can successfully get a confirmation message from the application, do nothing.
  while(begin_scan == false){
    begin_scan = confirmScan();
  }
  // We've configured successfully, proceed to scan. Set the serial timeout to match the
application.
  Serial.setTimeout(500);
  // sensor_type 0 -> SONAR
  //             1 -> LiDAR
  switch(sensor_type){
    case 0: // Sonar
      pinMode(2,INPUT);
      pinMode(3,OUTPUT);
      sonar_Serial.begin(9600);
      sonar_Serial.listen();
      break;
    case 1: // Lidar
      // Begin takes the argument of the Slave address for Arduino to take
      // If no address is given, it joins the I2C bus as a Master
      Wire.begin();
      // Set the wire clock to Fast mode, 400kbps
      Wire.setClock(400000);
      // Set up I2C communication mode to LiDAR
      Wire.beginTransmission(lid_addr);
      // Write the command to change the measurement unit to mm instead of cm
      // HEX: {5A, 05, 05, 06, 6A}     DEC: {90, 5, 5, 6, 106}
      byte message[] = {90, 5, 5, 6, 106};
      Wire.write(message,5);
      // TODO: Add in a message to change the slave address of the I2C if necessary
      // 210618, DP: Likely unnecessary.
      Wire.endTransmission();
      delay(10);  // Wait minimum time for LiDAR response
      break;
  }
  anticipated_points = calcScanPoints();
  homingSequence();
}

bool getConfiguration(){
  /*  Listen on the USB UART for configuration data from the application
   *  Save settings to globals
   *  If data has an invalid parameter, pass back false
   */
  String sensor_string = "", angle_string = "", hiangle_string = "";
  String loangle_string = "", resolution_string = "", valid_string = "", trim_string = "";
  // Set the serial timeout to be very long to allow for potential manual configuration
  // Get configuration data from the application
  // Leave 10ms in between reads to give time for the serial buffer to fill
  Serial.setTimeout(10000);
  sensor_string = Serial.readStringUntil('\n');
  delay(10);
  angle_string = Serial.readStringUntil('\n');
  delay(10);
  hiangle_string = Serial.readStringUntil('\n');
  delay(10);
  loangle_string = Serial.readStringUntil('\n');
  delay(10);
  resolution_string = Serial.readStringUntil('\n');
  delay(10);
```

```
  valid_string = Serial.readStringUntil('\n');
  delay(10);
  trim_string = Serial.readStringUntil('\n');
  Serial.setTimeout(1000);
  // Use string methods to pull configuration data from transmission
  sensor_type = (unsigned int)sensor_string.toInt();
  scan_angle = (unsigned int)angle_string.toInt();
  highside_angle = (unsigned int)hiangle_string.toInt();
  lowside_angle = (unsigned int)loangle_string.toInt();
  scan_resolution = (float)resolution_string.toFloat();
  scan_point_trimming = (unsigned int)trim_string.toInt();
  scan_data_validation = (unsigned int)valid_string.toInt();

  // Make sure we've changed at least one parameter
  if((scan_angle==0)){
    return false;
  }
  else{
    scanner_configured = true;
    return true;
  }
  return false; // If we've gotten here, something's gone wrong.
}

bool confirmScan(){
  // Take the configuration the scanner has and push it back to the application
  // Once permission is given from application, begin scan.
  // If permission is not granted, pass back false.
  String frame = "", response = "";
  // TODO: Add vertical angle validation
  frame += (String(sensor_type) + ','
    + String(scan_angle) + ','
    + String(highside_angle) + ','
    + String(lowside_angle) + ','
    + String(scan_data_validation) + ','
    + String(scan_point_trimming));
  Serial.println(frame);
  // Set serial timeout long for potential manual configuration
  Serial.setTimeout(10000);
  delay(10);
  response = Serial.readStringUntil('\n');
  Serial.setTimeout(1000);
  if(response == "scan"){
    return true;
  }
  else{
    return false;
  }
  return false; // If we've gotten here, something has gone wrong.
}

void homingSequence(){
  // Initialize the position of the vertical stepper motor
  // Rotate CCW until the limit switch activates
  // Once limit switch is hit, move CW to the highest possible scan angle
  bool limit_state = false;
  // limit_pin is HIGH when switch is closed and closed to ground
  limit_state = !digitalRead(limit_pin);
  // Increment the motor 1 degree in the FORWARD direction until we close the limit switch
  while(!limit_state){
    vert_Stepper->step(20,FORWARD,DOUBLE);  // TEST: Change to smaller step value
    limit_state = !digitalRead(limit_pin);
  }
  // Limit switch reached, go BACKWARD to top vertical angle
  int init_pos_steps = vert_deg_steps*highside_angle;  // TEST: Angle that scanner reaches after
taking the steps
  vert_Stepper->step(init_pos_steps,BACKWARD,DOUBLE);
  vert_position = highside_angle;
}

unsigned long calcScanPoints(){
```

```
  // Calculate the number of points we should be scanning
  unsigned long points = 0;
  // TODO: Re-introduce point trimming calculation
  // Steps : Resolution
  // (1 : 4.0), (2 : 2.0), (5 : 1.0), (11 : 0.5), (22 : 0.25), (56 : 0.1)
  unsigned int prc_adj_vert_step = vert_deg_steps/scan_resolution;
  // (4 : 4.0), (10 : 2.0), (20 : 1.0), (40 : 0.5), (80 : 0.25), (200 : 0.1)
  unsigned int prc_adj_horiz_step = horiz_deg_steps/scan_resolution;
  // Position change (Degrees) : Resolution
  // (0.1767 : 4.0), (0.3535 : 2.0), (0.8837 : 1.0), (1.944 : 0.5), (3.888 : 0.25), (9.897 : 0.1)
  float prc_adj_vert_pos = prc_adj_vert_step*(1/vert_deg_steps);
  // (0.1991 : 4.0), (0.4978 : 2.0), (0.9957 : 1.0), (1.991 : 0.5), (3.983 : 0.25), (9.957 : 0.1)
  float prc_adj_horiz_pos = prc_adj_horiz_step*(1/horiz_deg_steps);
  unsigned long vertical_positions = ((lowside_angle-highside_angle)/prc_adj_vert_pos) + 1; //
Add 1 to ensure we go over the angle limit
  unsigned long horizontal_positions = (scan_angle/prc_adj_horiz_pos) + 1; // Add 1 to ensure we
go over the angle limit
  points = vertical_positions*horizontal_positions;
  return points;
}

/*****************
 Jog (B) functions
*****************/

void codeBJog(){
  /*  Each jog button will only send a single character on UART.
   *  Wait to receive it from the application.
   */
  delay(100);
  char direct_code = (char)Serial.read();
  switch(direct_code){
    // Each button press should correspond to 4 degrees of turn
    // TODO: This amount of steps should be configurable.
    case 117:  // 'u'
      vert_Stepper->step(20,FORWARD,DOUBLE);
      break;
    case 100:  // 'd'
      vert_Stepper->step(20,BACKWARD,DOUBLE);
      break;
    case 108:  // 'l'
      horiz_Stepper->step(80,BACKWARD,DOUBLE);
      break;
    case 114:  // 'r'
      horiz_Stepper->step(80,FORWARD,DOUBLE);
      break;
  }
}

/************************************
 Scan and data acquisition (C) functions
************************************/

void codeCScan(){
  // Loop until we have enough points to satisfy the scan.
  while(!scan_complete){
    bool data_state = false;
    bool scan_the_point = false;
    // ---------------------------------
    // Gather data from configured sensor
    // ---------------------------------
    // If point interpolation enabled, check if we need to measure a point at the current scanner
position.
    scan_the_point = scanPoint();
    if(scan_the_point){
      // LiDAR routine
      if((sensor_type) && (!scan_complete)){
        data_state = readLidar();
        // If we are not using data validation, and the data is invalid, pass 0's.
        if((!scan_data_validation) && (!data_state)){
          distance = 0;
```

```
          intensity = 0;
          data_state = true; // Pass a true data state so program proceeds.
        }
        // Otherwise, we're using data validation and need to continue polling sensor until we
get a good value
        else{
          while(!data_state){
            data_state = readLidar();
          }
        }
      }
      // Sonar routine
      else if((!sensor_type) && (!scan_complete)){
        data_state = readSonar();
        // If we are not using data validation, and the data is invalid, pass 0.
        if((!scan_data_validation) && (!data_state)){
          distance = 0; // Sonar doesn't yield intensity.
          data_state = true; // Pass a true data state so program proceeds.
        }
        // Otherwise, we're using data validation and need to continue polling sensor until we
get a good value
        else{
          while(!data_state){
            data_state = readSonar();
          }
        }
      }
    }
    // Increment our point count, irrespective of data validity condition
    scan_points_acquired += 1;
    /*  Push data
     *  If data validation is enabled, and the data is bad, loop and try reading data again. App
will wait.
     *  Data frame is structured as:
     *  DATA: Distance, Itensity, Horizontal angle, Vertical angle, Checksum
     *  TYPE: unsigned int, unsigned int, float, float, unsigned long
     *  UNIT: [mm], [value], [angle°], [angle°], [integer]
     */
    if(data_state){
      unsigned long checksum = genChecksum();
      Serial.print(distance);
      Serial.print(',');
      Serial.print(intensity);
      Serial.print(',');
      Serial.print(horiz_position);
      Serial.print(',');
      Serial.print(vert_position);
      Serial.print(',');
      Serial.print(checksum);
      Serial.print('\r');
      Serial.print('\n');
    }
    else
    {
      Serial.print("0,0,0,0,0");
      Serial.print('\r');
      Serial.print('\n');
    }
    // Move the scanner to the next position.
    moveScanner();
    // Check to see if the scan has completed, set the flag if it is.
    if(scan_points_acquired >= anticipated_points){
      scan_complete = true;
    }
  }
  // Return to starting position after loop is finished, "anti-tangle"
  horiz_Stepper->step((full_steps*(horiz_position/360)),FORWARD,DOUBLE); // Rotate horizontally
clockwise
  horiz_position = 0;
  // Scan is complete, pass control back to loop() and allow serial commands again.
  Serial.print("complete,");
```

```
  if(scan_data_validation){
    Serial.println(validation_fails);
  }
  else{
    Serial.println("0");
  }
  horiz_Stepper->release();
  vert_Stepper->release();
}

void moveScanner(){
  /*  Take scan parameters and move the steppers accordingly
   *  How many steps the motor moves between data points is dependent on scan resolution
   *  When method returns true, scan is complete.
   *  (Scan resolution : Vertical points per horizontal position)
   *  (2.0 : 180), (1.0 : 90), (0.5 : 45), (0.25 : 22), (0.1 : 9)
   *  Scan resolution : Horizontal degrees in full rotation
   *  (2.0 : 720), (1.0 : 360), (0.5 : 180), (0.25 : 90), (0.1 : 36)
   * INFO: There is theoretically an available "extremely fine" scan where we take 11 data points
per degree
   *       This would mean 990 points per horizontal degree, and 3960 horizontal points
   *       A whopping total of 3.92 MILLION points!
   */
  // Steps : Resolution
  // (1 : 4.0), (2 : 2.0), (5 : 1.0), (11 : 0.5), (22 : 0.25), (56 : 0.1)
  int prc_adj_vert_step = vert_deg_steps/scan_resolution;
  // (4 : 4.0), (10 : 2.0), (20 : 1.0), (40 : 0.5), (80 : 0.25), (200 : 0.1)
  int prc_adj_horiz_step = horiz_deg_steps/scan_resolution;
  // Position change (Degrees) : Resolution
  // (0.1767 : 4.0), (0.3535 : 2.0), (0.8837 : 1.0), (1.944 : 0.5), (3.888 : 0.25), (9.897 : 0.1)
  float prc_adj_vert_pos = prc_adj_vert_step*(1/vert_deg_steps);
  // (0.1991 : 4.0), (0.4978 : 2.0), (0.9957 : 1.0), (1.991 : 0.5), (3.983 : 0.25), (9.957 : 0.1)
  float prc_adj_horiz_pos = prc_adj_horiz_step*(1/horiz_deg_steps);
  // Control the vertical motor based on the direction we need to travel
  switch(vert_motor_direction){
    case 0: // Rotate servo upwards
      vert_Stepper->step(vertical_step_cnt,FORWARD,DOUBLE);
      vert_position = highside_angle;
      vertical_step_cnt = 0;
      vert_motor_direction = !vert_motor_direction;
      break;
    case 1: // Rotate servo downwards
      vert_Stepper->step(prc_adj_vert_step,BACKWARD,DOUBLE);
      vert_position += prc_adj_vert_pos;
      vertical_step_cnt += prc_adj_vert_step;
      break;
  }
  if(vert_position >= lowside_angle){
    horiz_Stepper->step(prc_adj_horiz_step,BACKWARD,DOUBLE); // Rotate horizontally anti-
clockwise
    horiz_position += prc_adj_horiz_pos;
    vert_motor_direction = !vert_motor_direction;
    // Cycle the scanner line around the pattern
    if(scanner_current_line >= 8){
      scanner_current_line = 1;
    }
    else{
      scanner_current_line += 1;
    }
  }
  return;
}

bool readLidar() {
  /*  Read distance information from the Benewake TF02 Pro or TF Mini Plus
   *  Information is communicated over I2C bus at 400 kbps (Fast Mode), must be requested
   *  When data is read successfully, write it to the global variable
   */
  distance = 0;
  byte i = 1;  // Iterator for dataframe from lidar
  byte b = 0;  // Byte holder for dataframe from lidar
```

```
  word checksum = 0;
  bool check_pass = false;
  // Set up the delay time between request and read from the LiDAR on I2C
  const byte lid_dly = 10; // Value shouldn't change, make constant
  Wire.beginTransmission(lid_addr);
  /*  The write command will send a request to the LiDAR to obtain a data frame
   *  Measurement will change based on command
   *  HEX: {5A 05 00 01 60} will give cm
   *  DEC: {90 5 0 1 96}
   *  HEX: {5A 05 00 06 65} will give mm
   *  DEC: {90 5 0 6 101}
   */
  byte data_req_from_lid[] = {90, 5, 0, 6, 101};
  Wire.write(data_req_from_lid,5);
  // endTransmission by default ends with a "stop" message on the I2C bus - the Pro and Mini
require this message to be present
  Wire.endTransmission();
  // It's recommended that after initiating a request for data that you wait 100ms before reading
the response
  // However, this can be reduced dramatically, to near 10ms to allow for fast data acquisition
  delay(lid_dly);
  // Request the slave write the data frame onto the bus
  Wire.requestFrom(lid_addr,9);
  // Read the response from the LiDAR, interpret it and push it.
  while(Wire.available()){
    // Get the next available byte
    b = Wire.read();
    // Bytes 3 and 4 are the distance data bytes, process them
    if(i==3){
      distance = b;
    }
    if(i==4){
      // Correct for Little Endian format, and add to the total
      distance = distance + (b << 8);
    }
    // Bytes 5 and 6 are the signal strength data bytes, process them
    if(i==5){
      intensity = b;
    }
    if(i==6){
      // Correct for Little Endian format, and add to the total
      intensity = intensity + (b << 8);
    }
    // If this isn't the checksum byte from LiDAR, add it to the local checksum
    if(i != 9){
      checksum += b;
    }
    // Otherwise, we are looking at the checksum byte. Validate our checksum against it.
    // If our checksum matches, we've received the data correctly. Push it to serial.
    // Return whether the checksum matched or not to the main program.
    else{
      if(lowByte(checksum) == b){
        // If the data matches the LiDAR checksum, data is good.
        check_pass = true;
      }
      else{
        // We've failed a checksum validation, add to the statistics.
        validation_fails += 1;
      }
      // The return value of the function will dictate if we use the data or not.
      return check_pass;
    }
    // Increment byte counter
    i++;
  }
  return false; // If we've gotten here, something went wrong with the I2C. Don't use the data.
}

bool readSonar(){
  // Read distance information from theDYP-A01-V2.0 sonar
  // Information is communicated over UART at 9600 baud, unprompted
```

```
  // When data is read successfully, write it to the global variable
  distance = 0;
  word incoming_checksum = 0;
  byte frame[2];
  byte discard = 0;
  if(sonar_Serial.available()>=4){
    byte incoming_byte = byte(sonar_Serial.read());
    if(incoming_byte == 255){
      frame[0] = byte(sonar_Serial.read());
      frame[1] = byte(sonar_Serial.read());
      frame[2] = byte(sonar_Serial.read());
      distance = frame[0] << 8;
      distance += frame[1];
      incoming_checksum += frame[0] + frame[1] + 255;
      // Leave time for another piece of data to be generated
      delay(300);
      // Discard any data in the serial buffer that accumulated after we got the data frame
      while(sonar_Serial.available()>0){
        discard = byte(sonar_Serial.read());
      }
      // If checksum is valid and the data is not the maximum value (likely erroneous), data is
good.
      if(lowByte(incoming_checksum) == frame[2]){
        return true;
      }
      else{
        validation_fails += 1;
        return false;
      }
    }
  }
  return false; // If we've gotten here, something has gone wrong. Discard the data.
}

bool scanPoint(){
  // Take the current vertical position and return whether or not to measure a point based on
scanner line.
  unsigned int comparison_high_angle = 0, comparison_low_angle = 0;
  // If point trimming is disabled, scan every single point.
  if(!scan_point_trimming){
    return true;
  }
  // Set the angle limits for point interpolation.
  comparison_low_angle = lowside_angle;
  if((scanner_current_line % 8)== 0)
  {
    comparison_high_angle = highside_angle;
  }
  else if((scanner_current_line % 4)== 0)
  {
    comparison_high_angle = max(highside_angle,7);
  }
  else if((scanner_current_line % 2)== 0)
  {
    comparison_high_angle = max(highside_angle,14);
  }
  else
  {
    comparison_high_angle = max(highside_angle,30);
  }
  // If the scanner's current vertical position is within the range of angle where we want a
point on this line
  // return true to the program
  if((vert_position >= comparison_high_angle) && (vert_position <= comparison_low_angle)){
    return true;
  }
  else{
    return false;
  }
  return false;  // If we've gotten here, something's gone wrong. Default return value.
}
```

```
unsigned long genChecksum(){
  // Generate a checksum for our transmission to the application
  unsigned long checksum = 0;
  checksum += (distance + intensity + (int)horiz_position + (int)vert_position);
  return checksum;
}


/**********************
 Rangefind (D) functions
**********************/

void codeDRangefind(){
  delay(100);
  char sensor_code = (char)Serial.read();
  bool sensor_read = false;
  byte discard = 0;
  switch(sensor_code){
    case 48: // Sonar
      pinMode(2,INPUT);
      pinMode(3,OUTPUT);
      sonar_Serial.begin(9600);
      sonar_Serial.listen();
      delay(50);
      while(!sensor_read){
        sensor_read = readSonar();
      }
      sonar_Serial.end();
      break;
    case 49: // Lidar
      // Begin takes the argument of the Slave address for Arduino to take
      // If no address is given, it joins the I2C bus as a Master
      Wire.begin();
      // Set the wire clock to Fast mode, 400kbps
      Wire.setClock(400000);
      // Set up I2C communication mode to LiDAR
      Wire.beginTransmission(lid_addr);
      // Write the command to change the measurement unit to mm instead of cm
      // HEX: {5A, 05, 05, 06, 6A}     DEC: {90, 5, 5, 6, 106}
      byte message[] = {90, 5, 5, 6, 106};
      Wire.write(message,5);
      Wire.endTransmission();
      delay(50);  // Wait some time before writing the next command
      while(!sensor_read){
        sensor_read = readLidar();
      }
      Wire.end();
      break;
  }
  Serial.println(distance);
  distance = 0;
}

/*****
 Loop!
*****/

void loop() {
}
```

CONTROL APPLICATION

```
"""A module that creates the GUI application for the Open-Source Scanner.

Open-Source 3D Scanner Project
Created by: Darren Paetz, Matt Kantor, Dr. Philip Mees
MacEwan University, Edmonton, Alberta, Canada, 2021

This application is designed to be used with an Arduino-based 3D scanner.
The firmware running on the Arduino will communicate with this application,
allowing the application to convert sensor data into an ASTM standard 3D-scan
file format (E57).

This project was made possible with the following funding:
    MacEwan University's Undergraduate Student Research Initiative Grant (USRI)

"""

import sys
import pye57
import numpy as np
import math
import serial
import serial.tools.list_ports
from time import sleep
from datetime import datetime
from PyQt5 import QtWidgets, uic
import uuid

from typing import Dict
from pye57.__version__ import __version__
from pye57 import libe57

qtcreator_file = "3D_Scan_App_UI.ui"  # Load the ui file
Ui_MainWindow, QtBaseClass = uic.loadUiType(qtcreator_file)

SUPPORTED_POINT_FIELDS = {
    "cartesianX": "d",
    "cartesianY": "d",
    "cartesianZ": "d",
    "intensity": "f",
    "colorRed": "B",
    "colorGreen": "B",
    "colorBlue": "B",
    "rowIndex": "H",
    "columnIndex": "H",
    "cartesianInvalidState": "b",
}


class MainWindow(QtWidgets.QMainWindow, Ui_MainWindow):
    """Application GUI object.

    Inherits from QtWidgets.QMainWindow.

    """

    def __init__(self):
        """Initialize the application window object.

        Connect the signal and slots of the various objects within the GUI.
        Initialize the scan points and scan time labels with zeros.
```

```
        Returns
        -------
        None.

        """
        QtWidgets.QMainWindow.__init__(self)
        Ui_MainWindow.__init__(self)
        self.setupUi(self)

        # Initialize the properties of scan points and estimated time
        self.scan_point_count = 0
        self.scan_time_estimate = 0

        # Serial communication flag, put serial object in namespace
        self.connected_to_arduino = False
        self.arduino = None
        self.abort_scan = False

        # Lookup table for resolution multipliers
        # Ex Fine, Very Fine, Fine, Medium, Coarse, Very Coarse
        self.resolution_options = [4, 2, 1, 0.5, 0.25, 0.1]

        # Connect the signal slots for the file menu
        self.actionExit_2.triggered.connect(self.close)
        self.actionOpen_Settings_File.triggered.connect(self.loadSettings)
        self.actionSave_Settings_File.triggered.connect(self.saveSettings)
        self.actionAbout_OS3DS.triggered.connect(self.aboutDialog)
        self.actionAbout_Qt.triggered.connect(self.aboutQtDialog)
        self.actionLicense.triggered.connect(self.licenseDialog)

        # Initialize the com ports available to connect from OS
        self.refcomButton.clicked.connect(self.getSerialPorts)

        # Connect all combo boxes to the logic that calculates scan time
        # and number of scan points
        self.angleBox.activated.connect(self.calcScanProperties)
        self.resolutionBox.activated.connect(self.calcScanProperties)

        # Connect the browse for filename button to an open filename dialog
        self.outputfileButton.clicked.connect(self.getE57Filename)

        # Connect the scanner buttons to their respective methods
        self.connectButton.clicked.connect(self.connectToScanner)
        self.scanButton.clicked.connect(self.scan)
        self.abortScanButton.clicked.connect(self.abortScan)

        # Set the scan information with preliminary settings
        self.calcScanProperties()
        self.getSerialPorts()

        # Connect the jog buttons to the jog motor method
        self.jogscannerUp.clicked.connect(
                lambda: self.jogDirection("up"))
        self.jogscannerDown.clicked.connect(
                lambda: self.jogDirection("down"))
        self.jogscannerLeft.clicked.connect(
                lambda: self.jogDirection("left"))
        self.jogscannerRight.clicked.connect(
                lambda: self.jogDirection("right"))

        # Connect the rangefind button to the rangefind method
        self.rangefindButton.clicked.connect(self.rangeFind)

    """
    ------------------------------------------------------------
                File menu methods
```

```
    ------------------------------------------------------------
    """

    def loadSettings(self):
        """Load saved settings for the application from a CSV file.

        Returns
        -------
        None.

        """
        options = QtWidgets.QFileDialog.Options()
        # Get the filename of the settings file
        # Discard the filetype that the method returns
        fileName, _ = QtWidgets.QFileDialog.getOpenFileName(
            self, "Open a scanner configuration file", "",
            "Scanner Configuration Files (*.3dscancfg);;All Files (*)",
            options=options)
        if fileName != '':
            with open(fileName, "r") as fh:
                data = fh.readline()
                data = data.split(",")
                self.sensorBox.setCurrentIndex(int(data[0]))
                self.angleBox.setCurrentIndex(int(data[1]))
                self.highAngleBox.setValue(int(data[2]))
                self.lowAngleBox.setValue(int(data[3]))
                self.resolutionBox.setCurrentIndex(int(data[4]))
                self.datavalidationBox.setChecked(bool(data[5]))
                self.pointtrimmingBox.setChecked(bool(data[6]))
            self.calcScanProperties()

    def saveSettings(self):
        """Save current settings for the application to a CSV file.

        Returns
        -------
        None.

        """
        options = QtWidgets.QFileDialog.Options()
        fileName, _ = QtWidgets.QFileDialog.getSaveFileName(
            self, "Save scanner configuration to file", "",
            "Scanner Configuration Files (*.3dscancfg);;All Files (*)",
            options=options)
        if fileName != '':
            with open(fileName, "w") as fh:
                fh.write(repr(self.sensorBox.currentIndex()) + ",")
                fh.write(repr(self.angleBox.currentIndex()) + ",")
                fh.write(repr(self.highAngleBox.value()) + ",")
                fh.write(repr(self.lowAngleBox.value()) + ",")
                fh.write(repr(self.resolutionBox.currentIndex()) + ",")
                fh.write(repr(self.datavalidationBox.isChecked()))
                fh.write(repr(self.pointtrimmingBox.isChecked()))

    def aboutDialog(self):
        """Open the About dialog box.

        Returns
        -------
        None.

        """
        mesg = """Open Source 3D Scanning Utility \n
        This utility will allow you to connect to the OS-3D-Scanner, upload
        configurations, and then retrieve data from the scanner.\n
        This tool was developed by:
```

```
        Darren Paetz, Matt Kantor, & Dr. Philip Mees
        at MacEwan University, Edmonton, Alberta, Canada, 2021
        with the generous support of MacEwan University's USRI Grant\n
        Note:
        The OS3DS is not intended to be used in part of a formal BIM creation.
        We hope that it serves as a useful learning and visualization tool for
        any and all people that are interested in the topology of spaces.
        """
        QtWidgets.QMessageBox.about(
            self, "About the 3D Scanning Utility", mesg)

    def aboutQtDialog(self):
        """Open the aboutQt dialog box.

        Returns
        -------
        None.

        """
        QtWidgets.QMessageBox.aboutQt(self)

    def licenseDialog(self):
        """Open the aboutQt dialog box.

        Returns
        -------
        None.

        """
        dialog = QtWidgets.QDialog()
        dialog.setWindowTitle("LICENSE")
        dialog.setWindowModality(1)
        geometry = self.geometry()
        geometry.setWidth(550)
        geometry.setHeight(623)
        dialog.setGeometry(geometry)
        textarea = QtWidgets.QTextBrowser(dialog)
        textarea.setGeometry(0, 0, 550, 600)
        textarea.setFontPointSize(12)
        with open("LICENSE") as lic:
            lictext = lic.readlines()
        text = ''.join(lictext)
        textarea.setText(text)
        pb = QtWidgets.QPushButton("Okay", dialog)
        pb.setGeometry(237, 600, 75, 23)
        pb.setDefault(True)
        pb.clicked.connect(
            lambda: dialog.close())
        dialog.exec_()

    """
    ----------------------------------------------------------------
                Main window methods
    ----------------------------------------------------------------
    """

    def getSerialPorts(self):
        """Add the available COM ports to the COM port combobox.

        Returns
        -------
        None.

        """
        self.comBox.clear()
        self.comBox.addItem("---")
```

```
        ports = serial.tools.list_ports.comports()
        for port in ports:
            self.comBox.addItem(port.device)

    def calcScanProperties(self):
        """Change the value of both the scan time and scan points labels.

        Returns
        -------
        None.

        """
        pos_change_time = 0.0558  # Time is in seconds
        vertical_angle = self.lowAngleBox.value() - self.highAngleBox.value()
        scan_angles = [360, 270, 180, 135, 90, 45, 22, 10]
        angle = scan_angles[self.angleBox.currentIndex()]
        resolution = self.resolution_options[self.resolutionBox.currentIndex()]
        vert_res = round(vertical_angle*resolution)
        hor_res = round(angle*resolution)
        points = int(vert_res*hor_res)
        # TODO: This estimate needs to be updated when trimming is fully done
        #       Potentially obtain from Arduino directly before scan
        scan_time_estimate = round(points*pos_change_time/60, 2)
        self.scan_point_count = points
        self.scan_time_estimate = scan_time_estimate
        self.scanpoints_outLabel.setText(str(points))
        if scan_time_estimate < 1.0:
            self.scantime_outLabel.setText(
                str(round(scan_time_estimate*60, 2))
                + " seconds")
        elif scan_time_estimate < 90:
            self.scantime_outLabel.setText(
                str(scan_time_estimate)
                + " minutes")
        else:
            self.scantime_outLabel.setText(
                str(round(scan_time_estimate/60, 2))
                + " hours")

    def getE57Filename(self):
        """Open file dialog and use selected filename as plaintext box text.

        Returns
        -------
        None.

        """
        options = QtWidgets.QFileDialog.Options()
        default_fill = ""
        if self.sensorBox.currentIndex() == 1:
            sensortext = "LiDAR"
        else:
            sensortext = "Sonar"
        angletext = self.angleBox.currentText()[:-1]
        restext = self.resolutionBox.currentText()
        moddate = datetime.now().strftime("%y%m%d-%H%M-")
        default_fill += moddate + sensortext + "-" + angletext + "-" + restext
        fileName, _ = QtWidgets.QFileDialog.getSaveFileName(
            self, "Select a location for the E57 file", default_fill,
            "E57 File (*.e57)", options=options)
        self.outputfileEdit.setPlainText(fileName)

    def settingsScanState(self, state):
        """Change all form objects' enabled property during a scan event.

        To be used during an active scan. Will enable the Abort scan button,
```

```
        but disable all other widgets.

        Parameters
        ----------
        state : Boolean
            True -> Enabled
            False -> Disabled

        Returns
        -------
        None.

        """
        self.comBox.setEnabled(state)
        self.refcomButton.setEnabled(state)
        self.sensorBox.setEnabled(state)
        self.angleBox.setEnabled(state)
        self.highAngleBox.setEnabled(state)
        self.lowAngleBox.setEnabled(state)
        self.resolutionBox.setEnabled(state)
        self.outputfileEdit.setEnabled(state)
        self.outputfileButton.setEnabled(state)
        self.datavalidationBox.setEnabled(state)
        self.connectButton.setEnabled(state)
        self.jogMode(state)
        self.abortScanButton.setEnabled(not state)

    def jogMode(self, state):
        """Control the jog mode function of the application.

        Parameters
        ----------
        state : Boolean
            True -> Enabled
            False -> Disabled

        Returns
        -------
        None.

        """
        self.jogscannerUp.setEnabled(state)
        self.jogscannerDown.setEnabled(state)
        self.jogscannerLeft.setEnabled(state)
        self.jogscannerRight.setEnabled(state)

    def jogDirection(self, direction):
        """Write serial message to scanner containing jog direction."""
        self.arduino.write(bytes("B", encoding="ASCII"))
        sleep(0.05)
        if direction == "up":
            self.arduino.write(bytes("u", encoding="ASCII"))
        if direction == "down":
            self.arduino.write(bytes("d", encoding="ASCII"))
        if direction == "left":
            self.arduino.write(bytes("l", encoding="ASCII"))
        if direction == "right":
            self.arduino.write(bytes("r", encoding="ASCII"))

    def rangeFind(self):
        """Get the distance value from currently selected sensor."""
        self.arduino.write(bytes("D", encoding="ASCII"))
        sleep(0.05)
        sensor = str(self.sensorBox.currentIndex())
        self.arduino.write(bytes(sensor, encoding="ASCII"))
        sleep(0.1)
```

```
        response = self.arduino.readline()
        response = repr(response.decode(encoding='UTF-8'))
        self.distanceBox.setPlainText(response[1:-5] + " mm")
        return

    """
    ------------------------------------------------------------
                Connection and communication methods
    ------------------------------------------------------------
    """

    def connectToScanner(self):
        """Attempt to open a serial (COM) connection to the scanner.

        Returns
        -------
        None.

        """
        portname = self.comBox.currentText()
        if self.connected_to_arduino:
            self.connectFdbkLabel.setText("Already connected to scanner.")
            return
        if self.outputfileEdit.toPlainText() != '':
            try:
                self.arduino = serial.Serial(port=portname, baudrate=115200,
                                             timeout=3)
                self.connectFdbkLabel.setText("Successfully connected.")
                self.connected_to_arduino = True
                # If the scan button is hit immediately after connect, an
                # error will be generated. Simply pause the application for
                # one second before passing control back to user.
                sleep(1)
                self.jogMode(True)
                self.scanButton.setEnabled(True)
                self.rangefindButton.setEnabled(True)
            except serial.SerialException:
                self.connectFdbkLabel.setText(
                    """Error: Could not connect to scanner.""")
                self.connected_to_arduino = False
        else:
            self.connectFdbkLabel.setText(
                """Please choose a destination for the output file first.""")
        return

    def sendConfiguration(self):
        """Push scan configuration to Arduino.

        Returns
        -------
        bool
            True -> Configuration valid and accepted.
            False -> Configuration invalid or not accepted.

        """
        sensor = str(self.sensorBox.currentIndex())
        horiz_angle = str(self.angleBox.currentText()[:-1])
        vert_hi_angle = str(self.highAngleBox.value())
        vert_lo_angle = str(self.lowAngleBox.value())
        # Ex Fine, Very Fine, Fine, Medium, Coarse, Very Coarse
        resolution = str(
            self.resolution_options[self.resolutionBox.currentIndex()])
        validate = str(int(self.datavalidationBox.isChecked()))
        trimming = str(int(self.pointtrimmingBox.isChecked()))
        # Send configuration instruction code
        self.arduino.write(bytes("A", encoding="ASCII"))
```

```
        sleep(1)
        # Push the configuration to scanner
        self.arduino.write(bytes(sensor + '\n', encoding="ASCII"))
        sleep(0.01)
        self.arduino.write(bytes(horiz_angle + '\n', encoding="ASCII"))
        sleep(0.01)
        self.arduino.write(bytes(vert_hi_angle + '\n', encoding="ASCII"))
        sleep(0.01)
        self.arduino.write(bytes(vert_lo_angle + '\n', encoding="ASCII"))
        sleep(0.01)
        self.arduino.write(bytes(resolution + '\n', encoding="ASCII"))
        sleep(0.01)
        self.arduino.write(bytes(validate + '\n', encoding="ASCII"))
        sleep(0.01)
        self.arduino.write(bytes(trimming + '\n', encoding="ASCII"))
        total_bytes = (len(sensor)
                       + len(horiz_angle)
                       + len(vert_hi_angle)
                       + len(vert_lo_angle)
                       + len(validate)
                       + len(trimming)
                       + 5)  # commas and trim /r/n
        # Read the echo
        sleep(0.01)
        response = self.arduino.read(size=total_bytes)
        response = response.decode(encoding='UTF-8')
        parsed = response.split(",")
        print("Configuration frame from scanner: " + repr(parsed))
        # If the response matches the transmission, give clearance for scan.
        # Don't check the resolution, floating point comparison is meaningless.
        if((parsed[0] == sensor) and (parsed[1] == horiz_angle)
            and (parsed[2] == vert_hi_angle) and (parsed[3] == vert_lo_angle)
                and (parsed[4] == validate) and (parsed[5] == trimming)):
            self.arduino.write(bytes("scan\n", "ascii"))
            print("Succesfully configured.")
            print("Homing...")
            sleep(5)
            return True
        return False

    def scan(self):
        """Gather and process scan data from Arduino. Update window as needed.

        This method performs the following actions:

        Configuration
        -------------
        Pushes the current settings as a configuration frame to the scanner.
        If the scanner does not successfully read back the settings, close the
        connection. If the scanner accepts the settings, proceed to scan.

        Scan
        ----
        Reads data from the scanner over serial connection. If the serial
        connection times out, method retries until serial connection is
        re-established. If there is an exception on the serial connection, the
        data that has been gathered up until that point will be written to
        file. The user has the option to abort the scan in progress. If the
        scan is aborted, the data gathered up until that point will be written
        to file. Once the amount of points required for the scan are read, the
        data is written to file.

        Write to E57 File
        -----------------
        Once whatever data is collected and is ready for writing to file, it
        first must be processed. scan() calls the processPolar() method which
```

```
will convert the polar coordinate data that the scanner outputs to more
"human-legible" Cartesian coordinates. Once the data is converted,
scan() utilizes the E57Cust class, which has a writer method that will
create the E57 file.

Returns
-------
None.

"""
# Clear the output and update the widget
self.outputText.clear()
self.scanButton.setEnabled(False)
# Disable interaction with software while scan is taking place
# The exception is the Abort Scan button
self.settingsScanState(False)
QtWidgets.QApplication.processEvents()
# Initialize all the scan variables
# Set a zero at the beginning of every scan to give a reference
# when viewing the raw point cloud
distance = [0]
intensity = [0]
theta = [0]
phi = [0]
points_scanned = 0
outstr = ""
scan_complete = False
valid_fails = 0
self.scan_start_time = datetime.now()
fileName = self.outputfileEdit.toPlainText()
# Open the file where the data will be stored during the scan.
tmp_file = open(
    (fileName[:-4] + "_scandata.txt"), mode='w', encoding='utf-8')
# Send the configuration data to the Arduino
# On failure, cancel scan
scanner_configured = self.sendConfiguration()
if not scanner_configured:
    self.setScanResult(False, cfg=False)
    return
# Purge the serial buffer before we read data from the scanner
self.arduino.reset_input_buffer()
# Send the scan command
self.arduino.write(bytes("C", encoding="ASCII"))
# Obtain scan information from the Arduino
while not scan_complete:
    try:
        data = self.arduino.readline()
        # Strip CRLF
        data = data.decode('utf8')[:-2]
        parsed = data.split(",")
        print(repr(parsed) + " parsed")
        if parsed[0] == "complete":
            scan_complete = True
            valid_fails = int(parsed[1])
        elif(parsed[0] == "0" and parsed[1] == "0"
                and parsed[2] == "0" and parsed[3] == "0"
                and parsed[4] == "0"):
            pass
        else:
            # TODO: Should be writing these values to an interim file
            #       then getting them from that file
            #       and pushing afterwards
            # Parse and separate the sent parameters from the Arduino
            # Data frame is structured as:
            # DATA: [Distance, Itensity,
            #       Horizontal stepper angle (theta),
```

```
                #       Vertical stepper angle (phi), Checksum]
                # TYPE: [unsigned int, unsigned int, float,
                #       float, unsigned long]
                # UNIT: [mm], [value], [angle°], [angle°], [integer]
                distance.append(float(parsed[0]))
                intensity.append(float(parsed[1]))
                theta.append(float(parsed[2]))
                phi.append(float(parsed[3]))
                # Accumulate total point count
                points_scanned += 1
                # Write the scan data we received to a file
                # to preserve it in case of scanner failure
                tmp_file.write(repr(parsed[:-1]) + '\n')
            # If at any point, a serial exception is generated, abort the scan
            # This operation will preserve any scan data acquired
            except serial.SerialException:
                self.setScanResult(False, points=points_scanned)
                self.processAndExport(
                    fileName, distance, intensity, theta, phi)
                tmp_file.close()
                return
            # If the serial connection times out, keep trying
            except serial.SerialTimeoutException:
                print("Timeout!")
                pass
            # Arduino passed bad data, set at origin
            except ValueError:
                distance.append(0)
                intensity.append(0)
                theta.append(0)
                phi.append(0)
                print("Value error!")
                pass
            else:
                # If the scan got aborted by the user, break out from scan
                if self.abort_scan:
                    self.setScanResult(
                        False, points=points_scanned, usr_abt=True)
                    self.processAndExport(
                        fileName, distance, intensity, theta, phi)
                    tmp_file.close()
                    return
                # Print points completed in divisions of the modulus
                # Update the progress bar
                if (points_scanned % 10) == 0:
                    outstr = str(points_scanned) + " points scanned...\n"
                    self.outputText.setText(outstr)
                    progress = int((points_scanned/self.scan_point_count)*100)
                    self.progressBar.setValue(progress)
                # Update the application screen
                QtWidgets.QApplication.processEvents()
        # Print the final message to the output
        tmp_file.close()
        self.setScanResult(True, points=points_scanned, failedpts=valid_fails)
        self.processAndExport(fileName, distance, intensity, theta, phi)
        return

    def abortScan(self):
        """Set the abort scan flag."""
        self.abort_scan = True

    def setScanResult(self, result_success=False, points=0, failedpts=0,
                      usr_abt=False, cfg=True):
        """Update the form objects with scan progress and result.

        Will provide a feedback message with some basic scan statistics
```

```
to the user.

Parameters
----------
result_success : Boolean, optional
    True -> Scan complete.
    False -> Scan incomplete.
points : Integer, optional
    Number of points completed in scan.
usr_abt : Boolean, optional
    True -> Scan aborted by user.
    False -> Scan not aborted by user.
cfg : Boolean, optional
    True -> Scanner had been successfully configured.
    False -> Scanner had not been successfully configured.

Returns
-------
None.

"""
self.arduino.close()
self.connected_to_arduino = False
self.connectFdbkLabel.setText("Reconnect to begin another scan.")
self.scan_fin_time = datetime.now()
self.scan_completion_time = str(self.scan_fin_time
                                - self.scan_start_time)[:-7]
scan_stats = ("SCAN STATISTICS:\n"
              + "----------------\n"
              + str(points)
              + " points collected.\n"
              + str(failedpts)
              + " points failed data validation.\n"
              + "Scan time: "
              + self.scan_completion_time
              + "\nAn E57 file has been saved to the directory:\n"
              + self.outputfileEdit.toPlainText()
              + "\n\nConnection to scanner closed.")
if result_success:
    self.progressBar.setValue(100)
    self.outputfileEdit.setPlainText('')
    self.outputText.setText(
        "SCAN COMPLETE.\n\n"
        + scan_stats)
else:
    self.outputText.clear()
    self.progressBar.reset()
    if usr_abt:
        self.outputText.setText(
            "SCAN ABORTED.\n\n"
            + "The current scan data has been saved to file.\n\n"
            + scan_stats)
        self.progressBar.reset()
        self.abort_scan = False
    elif not cfg:
        self.outputText.setText(
            "ERROR!\n\n"
            + "The scanner has not been successfully configured.\n"
            + "Please attempt to upload configuration again\n"
            + "by pressing the \'CONNECT\' button.")
    else:
        self.outputText.setText(
            "ERROR!\n\n"
            + "An unexpected communication error has occurred.\n"
            + "The current scan data has been saved to file.\n\n"
            + scan_stats)
```

```
        self.settingsScanState(True)  # Re-enable the settings
        self.jogMode(False)  # Turn off jog mode
        QtWidgets.QApplication.processEvents()

    """
    ------------------------------------------------------------
                Data processing and export methods
    ------------------------------------------------------------
    """

    def processPolar(self, distance, theta, phi):
        """Convert from polar coordinates to Cartesian coordinates.

        Parameters
        ----------
        distance : List [Integer,Integer,...]
            An ordered list containing all distances obtained in scan.
        theta : List [Integer,Integer,...]
            An ordered list containing all angles theta obtained in scan.
        phi : List [Integer,Integer,...]
            An ordered list containing all angles phi obtained in scan.

        Returns
        -------
        x : List [Integer,Integer,...]
            An ordered list containing all Cartesian x coordinates of scan.
        y : List [Integer,Integer,...]
            An ordered list containing all Cartesian y coordinates of scan.
        z : List [Integer,Integer,...]
            An ordered list containing all Cartesian z coordinates of scan.

        """
        x = []
        y = []
        z = []
        theta_rad = []
        phi_rad = []
        # Convert all angles to radians
        for angle in theta:
            rad_angle = math.radians(angle)
            theta_rad.append(rad_angle)
        for angle in phi:
            rad_angle = math.radians(angle)
            phi_rad.append(rad_angle)
        i = 0
        # Do polar conversion
        for radius in distance:
            x.append(radius*math.sin(phi_rad[i])*math.cos(theta_rad[i]))
            y.append(radius*math.sin(phi_rad[i])*math.sin(theta_rad[i]))
            z.append(radius*math.cos(phi_rad[i]))
            i += 1
        return x, y, z

    def exportE57(self, fileName, cartX, cartY, cartZ, intensity=None):
        """Export the scan data to E57 file.

        Parameters
        ----------
        fileName : String
            The destination for the E57 file.
        cartX : Numpy.array
            A Numpy array of the cartesian x coordinates of scan data.
        cartY : TYPE
            A Numpy array of the cartesian y coordinates of scan data.
        cartZ : TYPE
            A Numpy array of the cartesian z coordinates of scan data.
```

```
        Returns
        -------
        None.

        """
        # Create numpy arrays of int32
        cartesianX = np.array(cartX)
        cartesianY = np.array(cartY)
        cartesianZ = np.array(cartZ)
        # If intensity data not generated (using sonar sensor), set as zeros
        if intensity is not None:
            # If we have valid intensity data, set the zero coordinate's
            # intensity value to the smallest measured value
            intensity[0] = min(intensity[1:])
            intensity = np.array(intensity)
        else:
            intensity = np.zeros_like(cartesianX, dtype="intc")
        cartesianInvalidState = np.zeros_like(cartesianX, dtype="intc")
        # Get the name for E57 file from app window
        # If it's blank, set to the name created by timestamp and settings
        if self.e57NameEdit.toPlainText() != '':
            export_file_name = self.e57NameEdit.toPlainText()
        else:
            export_file_name = fileName
        # Set the dictionary that we will pass to the writer
        data = {
            "cartesianInvalidState": cartesianInvalidState,
            "cartesianX": cartesianX,
            "cartesianY": cartesianY,
            "cartesianZ": cartesianZ,
            "intensity": intensity
            }
        # Instantiate the writer
        writefile = E57Cust(fileName, mode='w')
        try:
            writefile.write_scan_raw(data, name=export_file_name)
        # TODO: Exception handling - what happens when the file write fails?
        finally:
            writefile.close()

    def processAndExport(self, fileName, distance, intensity, theta, phi):
        """Block for processing and export.

        This method combines the processPolar and exportE57 methods for
        more simple re-use through the scan method.

        Parameters
        ----------
        fileName : String
            The destination for the E57 file.
        distance : List [Integer,Integer,...]
            An ordered list containing all distances obtained in scan.
        theta : List [Integer,Integer,...]
            An ordered list containing all angles theta obtained in scan.
        phi : List [Integer,Integer,...]
            An ordered list containing all angles phi obtained in scan.

        Returns
        -------
        None.

        """
        # TODO: Add in checks to ensure data is present.
        #       Or read from file.
        # Convert the data from the Arduino from spherical to cartesian
```

```
        cartX, cartY, cartZ = self.processPolar(distance, theta, phi)
        # Export all cartesian data as E57 file
        self.exportE57(fileName, cartX, cartY, cartZ, intensity)
        return


class E57Cust(pye57.E57):
    """Custom E57 class with overwritten writer method.

    Inherits from the E57 class in pye57.
    write_scan_raw() method has been implemented so that scan_header is not
    unnecessarily bound by header properties such as rotation and pose.

    """

    def write_default_header(self):
        """Write the header information that comes before the scan data."""
        imf = self.image_file
        imf.extensionsAdd("", libe57.E57_V1_0_URI)
        self.root.set(
            "formatName",
            libe57.StringNode(imf, "ASTM E57 3D Imaging Data File"))
        self.root.set(
            "guid",
            libe57.StringNode(imf, "{%s}" % uuid.uuid4()))
        self.root.set(
            "versionMajor",
            libe57.IntegerNode(imf, libe57.E57_FORMAT_MAJOR))
        self.root.set(
            "versionMinor",
            libe57.IntegerNode(imf, libe57.E57_FORMAT_MINOR))
        self.root.set(
            "e57LibraryVersion",
            libe57.StringNode(imf, libe57.E57_LIBRARY_ID))
        self.root.set(
            "coordinateMetadata",
            libe57.StringNode(imf, ""))
        creation_date_time = libe57.StructureNode(imf)
        creation_date_time.set(
            "dateTimeValue",
            libe57.FloatNode(imf, datetime.now().timestamp()))
        creation_date_time.set(
            "isAtomicClockReferenced",
            libe57.IntegerNode(imf, 0))
        self.root.set(
            "creationDateTime",
            creation_date_time)
        self.root.set(
            "data3D",
            libe57.VectorNode(imf, True))
        self.root.set(
            "images2D",
            libe57.VectorNode(imf, True))

    def write_scan_raw(self, data: Dict, *, name=None, scan_header=None):
        """Write raw scan data to file using C++ library libe57.

        Modified from original class so as to not require certain scan header
        properties and parameters.

        Parameters
        ----------
        data : Dict
            DESCRIPTION.
        name : String, optional, keyword argument mandatory
            DESCRIPTION. The default is None.
```

```
scan_header : scanHeader, optional, keyword argument mandatory
    DESCRIPTION. The default is None.

Raises
------
ValueError
    DESCRIPTION.

Returns
-------
None.

"""
for field in data.keys():
    if field not in SUPPORTED_POINT_FIELDS:
        raise ValueError("Unsupported point field: %s" % field)

if scan_header is None:
    scan_header = self.image_file.root()

# temperature = getattr(scan_header, "temperature", 0)

scan_node = libe57.StructureNode(self.image_file)
scan_node.set("guid", libe57.StringNode(
    self.image_file, "{%s}" % uuid.uuid4()))
scan_node.set("name", libe57.StringNode(self.image_file, name))
scan_node.set("description",
              libe57.StringNode(self.image_file,
                                "pye57 v%s" % __version__))

# Count the number of points in the scan by looking at how large
# the x entry is in the data dictionary
n_points = data["cartesianX"].shape[0]

# Set up the index bounds structure
ibox = libe57.StructureNode(self.image_file)
if "rowIndex" in data and "columnIndex" in data:
    min_row = np.min(data["rowIndex"])
    max_row = np.max(data["rowIndex"])
    min_col = np.min(data["columnIndex"])
    max_col = np.max(data["columnIndex"])
    ibox.set("rowMinimum",
             libe57.IntegerNode(self.image_file, min_row))
    ibox.set("rowMaximum",
             libe57.IntegerNode(self.image_file, max_row))
    ibox.set("columnMinimum",
             libe57.IntegerNode(self.image_file, min_col))
    ibox.set("columnMaximum",
             libe57.IntegerNode(self.image_file, max_col))
else:
    ibox.set("rowMinimum", libe57.IntegerNode(self.image_file, 0))
    ibox.set("rowMaximum",
             libe57.IntegerNode(self.image_file, n_points - 1))
    ibox.set("columnMinimum", libe57.IntegerNode(self.image_file, 0))
    ibox.set("columnMaximum", libe57.IntegerNode(self.image_file, 0))
ibox.set("returnMinimum", libe57.IntegerNode(self.image_file, 0))
ibox.set("returnMaximum", libe57.IntegerNode(self.image_file, 0))

# Don't bother setting it for now, we're not going to use it
# scan_node.set("indexBounds", ibox)

# Look in the data dictionary for the key "intensity"
# If it is in there, look in the scan header
# for the properties of intensity and set the
# intensity box appropriately
```

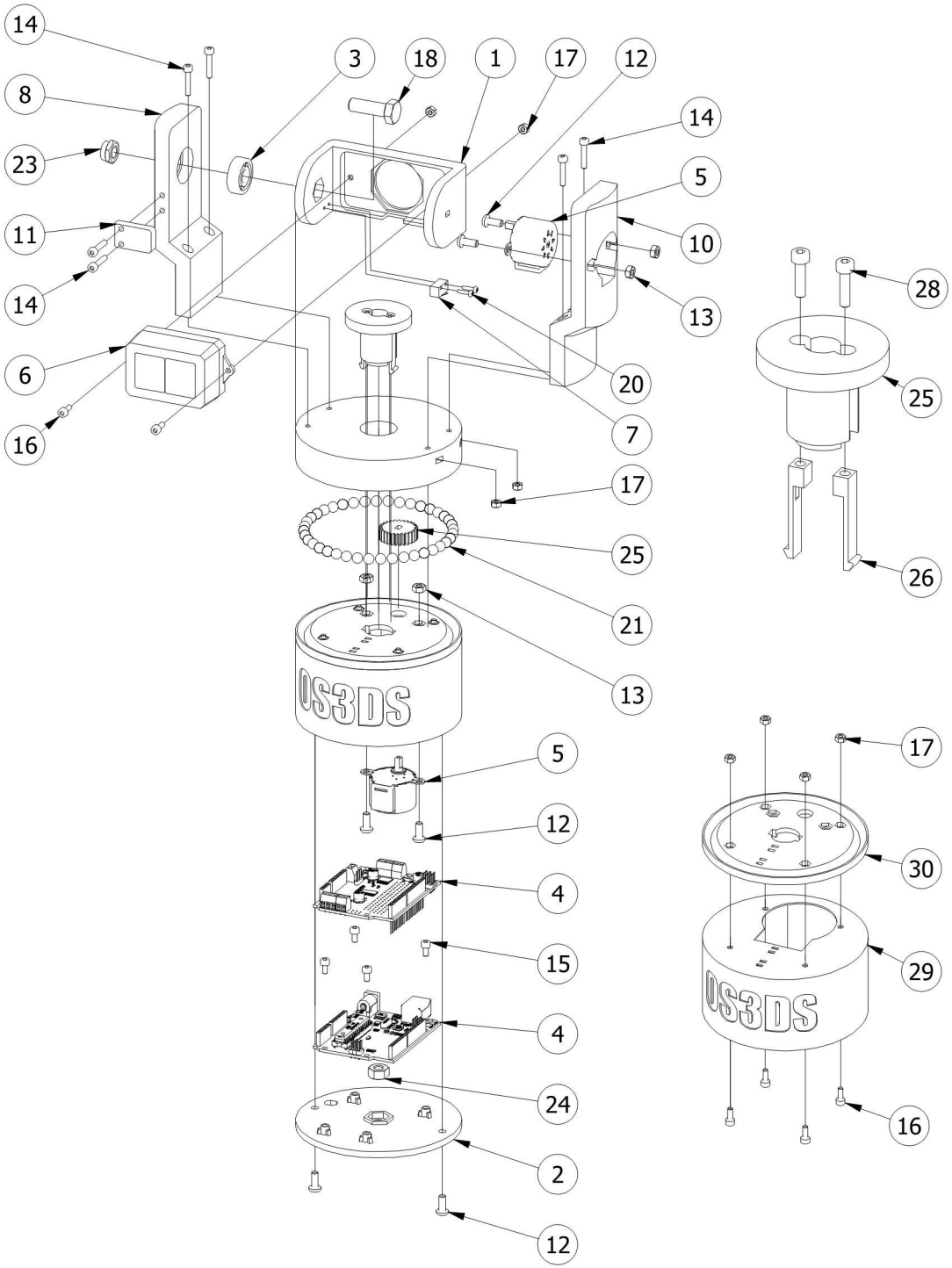
```
if "intensity" in data:
    int_min = getattr(scan_header, "intensityMinimum",
                      np.min(data["intensity"]))
    int_max = getattr(scan_header, "intensityMaximum",
                      np.max(data["intensity"]))
    intbox = libe57.StructureNode(self.image_file)
    intbox.set("intensityMinimum",
               libe57.FloatNode(self.image_file, int_min))
    intbox.set("intensityMaximum",
               libe57.FloatNode(self.image_file, int_max))
    scan_node.set("intensityLimits", intbox)

# Look for various color parameters in the data array
# Set the colorbox with their minimum and maximum
color = all(c in data for c in ["colorRed", "colorGreen", "colorBlue"])
if color:
    colorbox = libe57.StructureNode(self.image_file)
    colorbox.set("colorRedMinimum",
                 libe57.IntegerNode(self.image_file, 0))
    colorbox.set("colorRedMaximum",
                 libe57.IntegerNode(self.image_file, 255))
    colorbox.set("colorGreenMinimum",
                 libe57.IntegerNode(self.image_file, 0))
    colorbox.set("colorGreenMaximum",
                 libe57.IntegerNode(self.image_file, 255))
    colorbox.set("colorBlueMinimum",
                 libe57.IntegerNode(self.image_file, 0))
    colorbox.set("colorBlueMaximum",
                 libe57.IntegerNode(self.image_file, 255))
    scan_node.set("colorLimits", colorbox)

# Instantiate the structure node for the bounding box
bbox_node = libe57.StructureNode(self.image_file)
# Get the various cartesian elements from the data
# dictionary passed to function
x, y, z = data["cartesianX"], data["cartesianY"], data["cartesianZ"]
# Validate the data, comparing to the cartesianInvalidState key
valid = None
if "cartesianInvalidState" in data:
    # Get an array of the valid entries by
    # taking the complement of invalid
    valid = ~data["cartesianInvalidState"].astype("?")
    x, y, z = x[valid], y[valid], z[valid]
# Store minimum respective values
bb_min = np.array([x.min(), y.min(), z.min()])
bb_max = np.array([x.max(), y.max(), z.max()])
# Clear the namespace
del valid, x, y, z

bbox_node.set("xMinimum", libe57.FloatNode(self.image_file, bb_min[0]))
bbox_node.set("xMaximum", libe57.FloatNode(self.image_file, bb_max[0]))
bbox_node.set("yMinimum", libe57.FloatNode(self.image_file, bb_min[1]))
bbox_node.set("yMaximum", libe57.FloatNode(self.image_file, bb_max[1]))
bbox_node.set("zMinimum", libe57.FloatNode(self.image_file, bb_min[2]))
bbox_node.set("zMaximum", libe57.FloatNode(self.image_file, bb_max[2]))
# Write the cartesian bounds into the new structure
scan_node.set("cartesianBounds", bbox_node)

points_prototype = libe57.StructureNode(self.image_file)

is_scaled = False
precision = libe57.E57_DOUBLE if is_scaled else libe57.E57_SINGLE

center = (bb_max + bb_min) / 2

chunk_size = 5000000
```

```
x_node = libe57.FloatNode(self.image_file, center[0],
                          precision, bb_min[0], bb_max[0])
y_node = libe57.FloatNode(self.image_file, center[1],
                          precision, bb_min[1], bb_max[1])
z_node = libe57.FloatNode(self.image_file, center[2],
                          precision, bb_min[2], bb_max[2])
points_prototype.set("cartesianX", x_node)
points_prototype.set("cartesianY", y_node)
points_prototype.set("cartesianZ", z_node)

field_names = ["cartesianX", "cartesianY", "cartesianZ"]

if "intensity" in data:
    intensity_min = np.min(data["intensity"])
    intensity_max = np.max(data["intensity"])
    intensity_node = libe57.FloatNode(self.image_file,
                                      intensity_min, precision,
                                      intensity_min, intensity_max)
    points_prototype.set("intensity", intensity_node)
    field_names.append("intensity")

if all(color in data for color in
       ["colorRed", "colorGreen", "colorBlue"]):
    points_prototype.set("colorRed",
                         libe57.IntegerNode(
                             self.image_file, 0, 0, 255))
    points_prototype.set("colorGreen",
                         libe57.IntegerNode(
                             self.image_file, 0, 0, 255))
    points_prototype.set("colorBlue",
                         libe57.IntegerNode(
                             self.image_file, 0, 0, 255))
    field_names.append("colorRed")
    field_names.append("colorGreen")
    field_names.append("colorBlue")

if "rowIndex" in data and "columnIndex" in data:
    min_row = np.min(data["rowIndex"])
    max_row = np.max(data["rowIndex"])
    min_col = np.min(data["columnIndex"])
    max_col = np.max(data["columnIndex"])
    points_prototype.set("rowIndex",
                         libe57.IntegerNode(self.image_file, 0,
                                            min_row, max_row))
    field_names.append("rowIndex")
    points_prototype.set("columnIndex",
                         libe57.IntegerNode(self.image_file, 0,
                                            min_col, max_col))
    field_names.append("columnIndex")

if "cartesianInvalidState" in data:
    min_state = np.min(data["cartesianInvalidState"])
    max_state = np.max(data["cartesianInvalidState"])
    points_prototype.set("cartesianInvalidState",
                         libe57.IntegerNode(self.image_file, 0,
                                            min_state, max_state))
    field_names.append("cartesianInvalidState")

"""
other possible fields for points
    "sphericalRange"
    "sphericalAzimuth"
    "sphericalElevation"
    "timeStamp"
    "sphericalInvalidState"
```

```
            "isColorInvalid"
            "isIntensityInvalid"
            "isTimeStampInvalid"
        """

        arrays, buffers = self.make_buffers(field_names, chunk_size)

        codecs = libe57.VectorNode(self.image_file, True)
        points = libe57.CompressedVectorNode(
            self.image_file, points_prototype, codecs)
        scan_node.set("points", points)

        self.data3d.append(scan_node)

        writer = points.writer(buffers)

        current_index = 0
        while current_index != n_points:
            current_chunk = min(n_points - current_index, chunk_size)

            for type_ in SUPPORTED_POINT_FIELDS:
                if type_ in arrays:
                    arrays[type_][:current_chunk] = data[type_][
                        current_index:current_index + current_chunk]

            writer.write(current_chunk)

            current_index += current_chunk

        writer.close()


if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())
```

APPENDIX B: Drawing Package

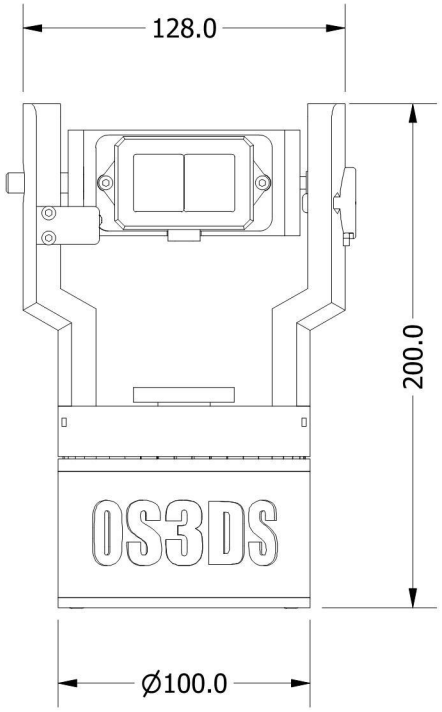
See overleaf.

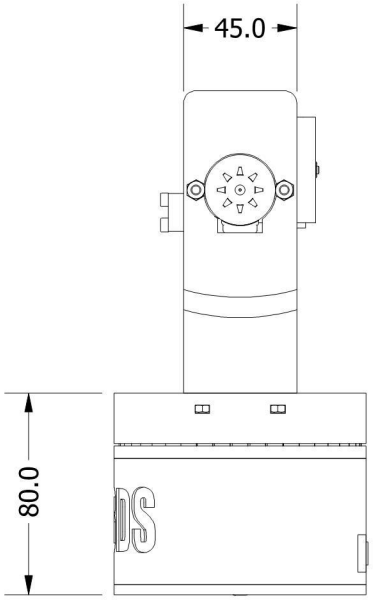


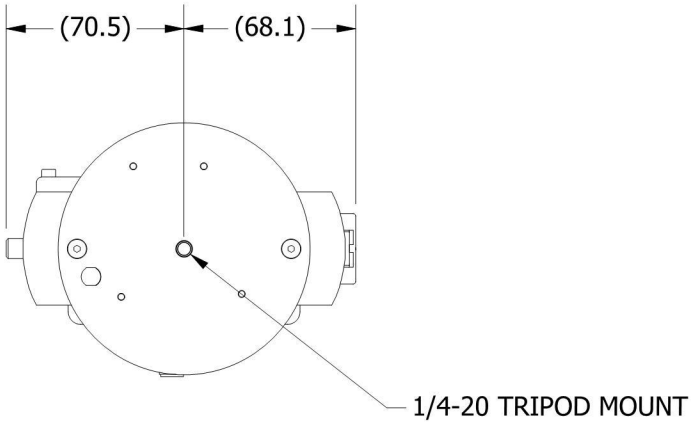
30	1002	ENCLOSURE TOP	1
29	1001	ENCLOSURE BODY	1
28	2312	SHCS M3 X 12	2
27	1004	SHAFT	1
26	1005	SHAFT HOOK	2
25	1003	SPUR GEAR	1
24	2F250	HEX NUT 1/4-20	1
23	2800F	LOCK NUT M8	1
22	3004	BARREL JACK 5.5 X 2.1 MM	1
21	2BF250	BALL BEARING 1/4 INCH	40
20	2208	MACH SCREW M2 X 8	2
18	2825	HCS M8 X 25	1
17	2300	HEX NUT M3	10
16	2308	SHCS M3 X 8	6
15	2306	SHCS M3 X 6	4
14	2316	SHCS M3 X 16	6
13	2400	HEX NUT M4	4
12	2410	BHCS M4 X 10	6
11	1009	ENDSTOP	1
10	1008	STEPPER ARM	1
9	1006	ROTATING BASE	1
8	1007	BEARING ARM	1
7	3002	LIMIT SWITCH	1
6	3003	LiDAR MODULE TF02 PRO	1
5	3001	STEPPER MOTOR	2
4	3000	ARDUINO UNO ASSEMBLY	1
3	SKF608	BALL BEARING 608ZZ	1
2	1000	BASE PLATE	1
1	1010	SENSOR BRACKET	1
ITEM	PART NUMBER	DESCRIPTION	QTY

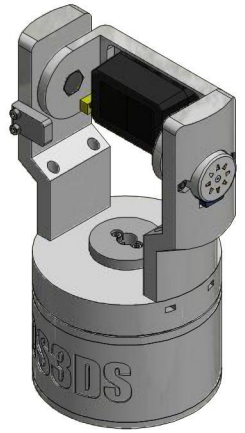
PARTS LIST

DRN BY: M. KANTOR	TITLE: OS3DS - ASSEMBLY		
DATE: 2021-06-30			
DRW. TYPE: ASSEMBLY			
SCALE: 1 / 4	PART NO: OS3DSV1		
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 1 OF 13

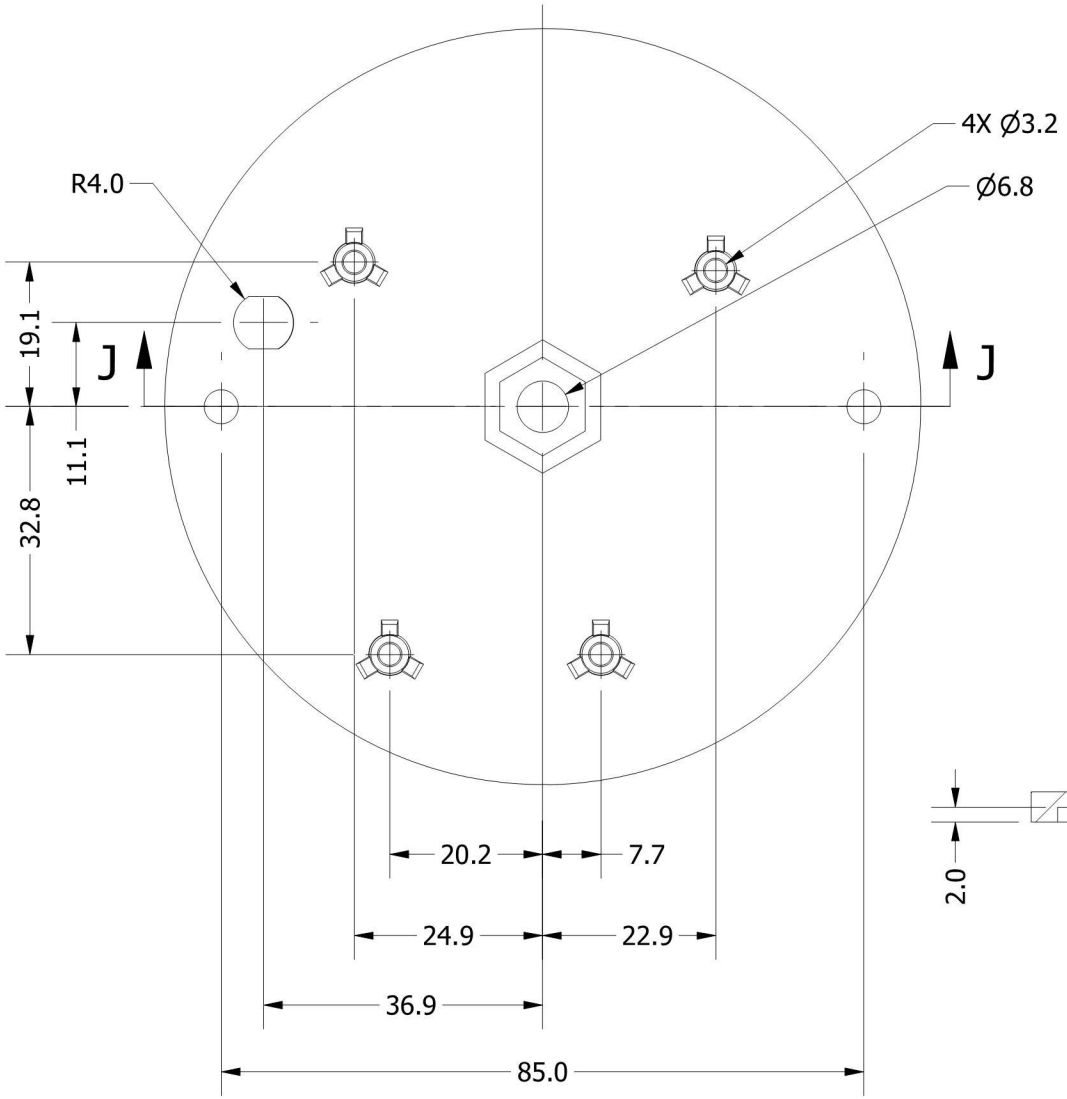


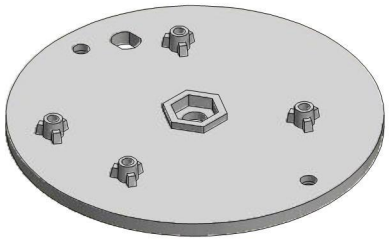


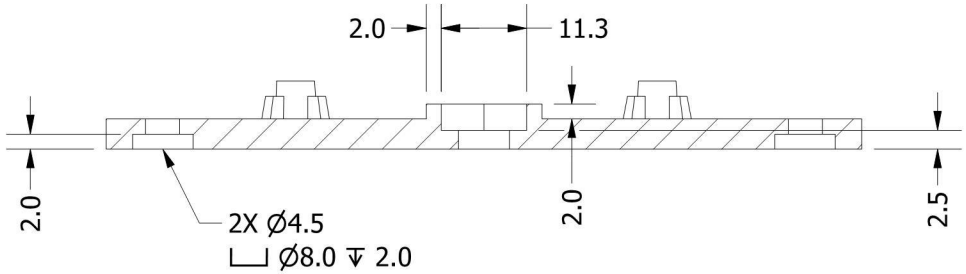




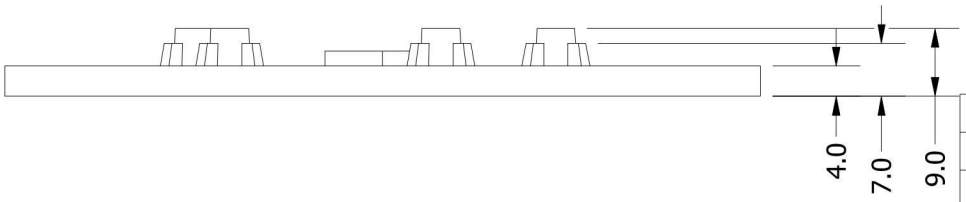
DRN BY: M. KANTOR	TITLE:			
DATE: 2021-06-30	OS3DS - ASSEMBLY			
DRW. TYPE: ASSEMBLY				
SCALE: 1 / 3	PART NO:			OS3DSV1
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 2 OF 13	



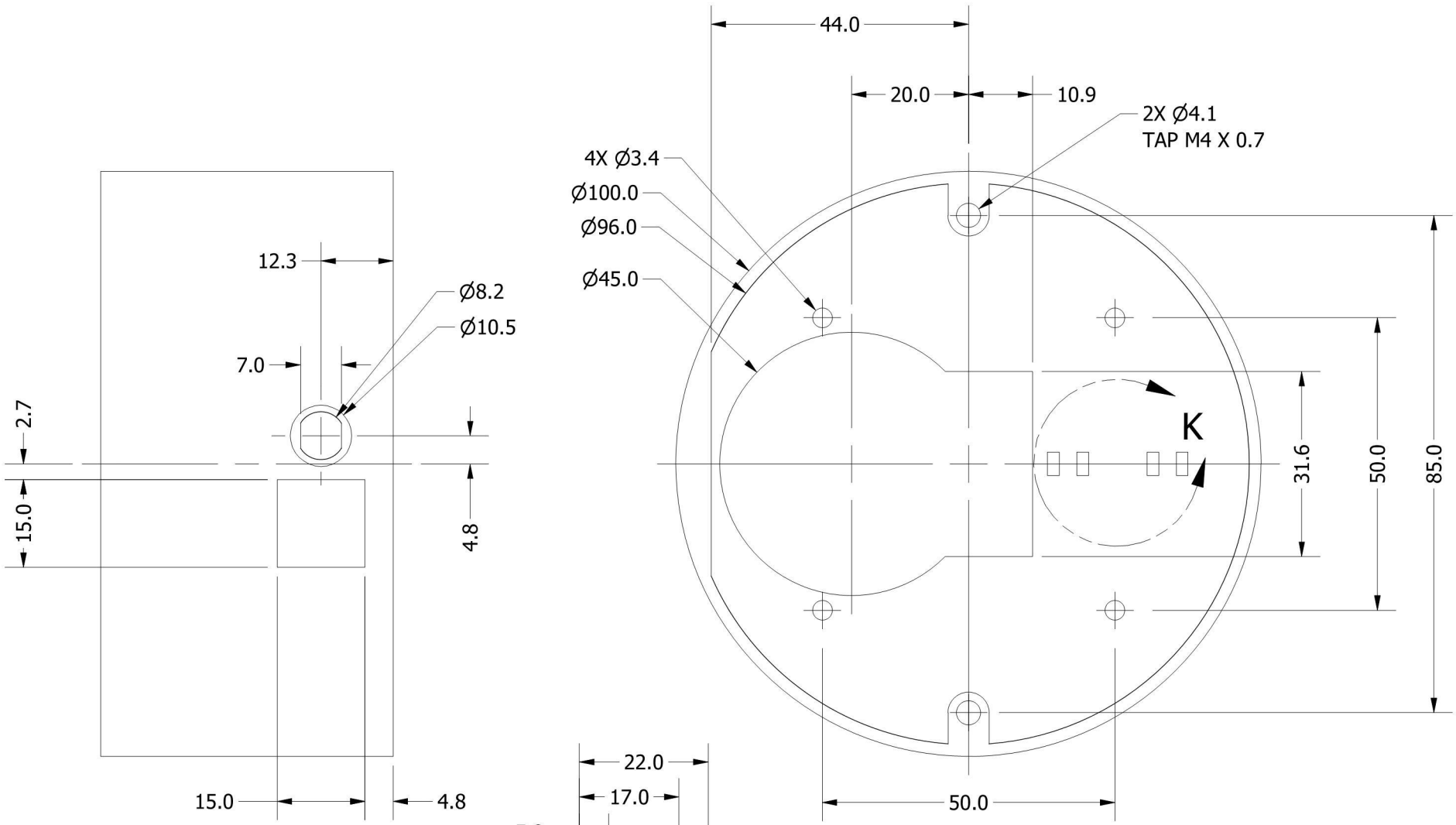


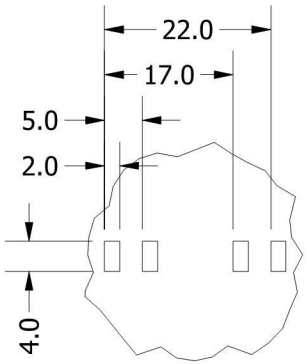


SECTION J-J
SCALE 1 : 1



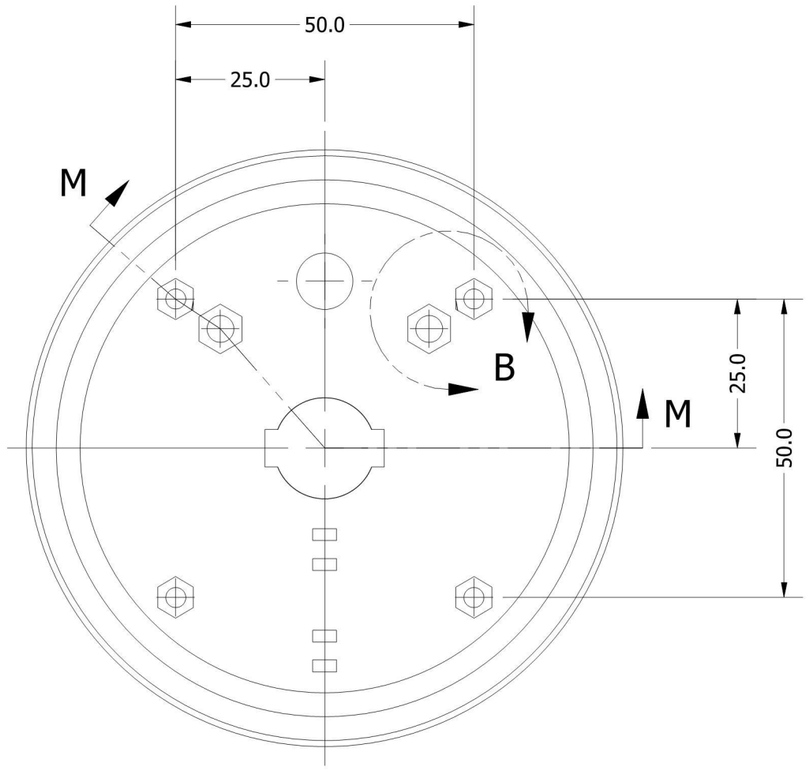
DRN BY: M. KANTOR	TITLE: BASE PLATE			
DATE: 2021-06-30				
DRW. TYPE: DETAIL				
SCALE: 1 : 1	PART NO: 1000			
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 3 OF 13	

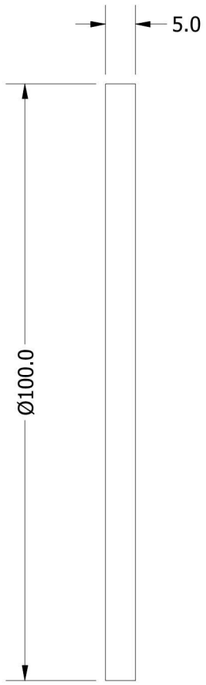


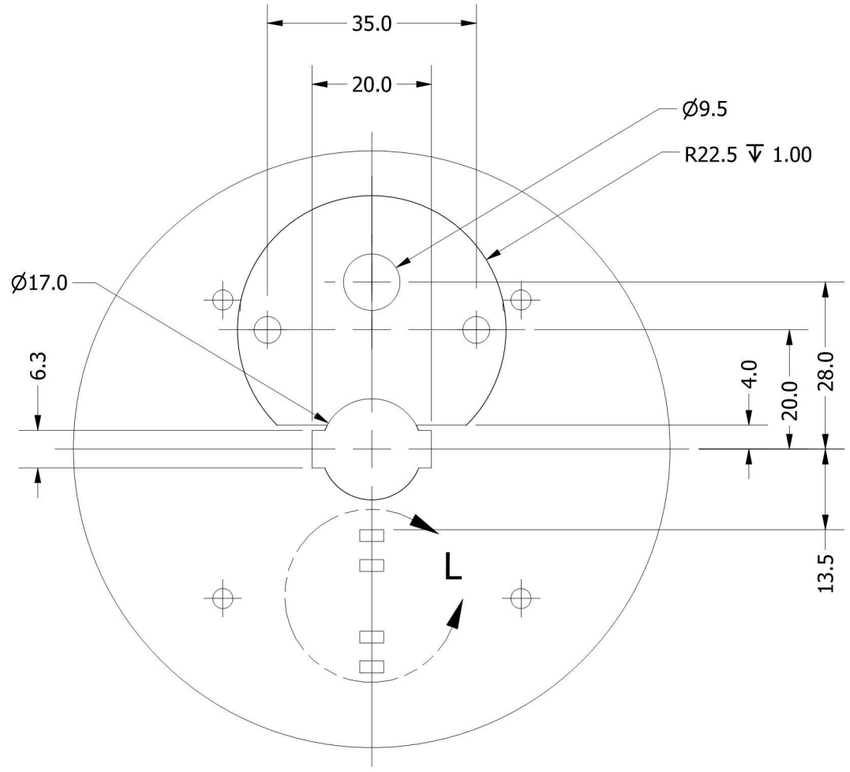


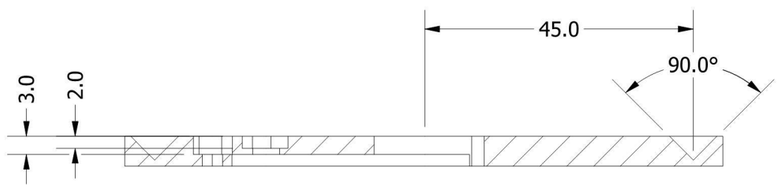
DETAIL K
SCALE 1 : 1

DRN BY: M. KANTOR	TITLE: ENCLOSURE BODY			
DATE: 2021-06-30				
DRW. TYPE: DETAIL				
SCALE: 1 : 1	PART NO: 1001			
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 4 OF 13	

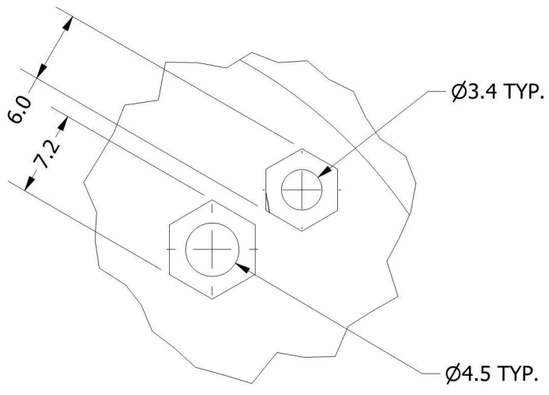




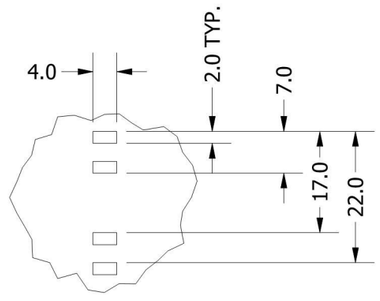




SECTION M-M
SCALE 1 : 1



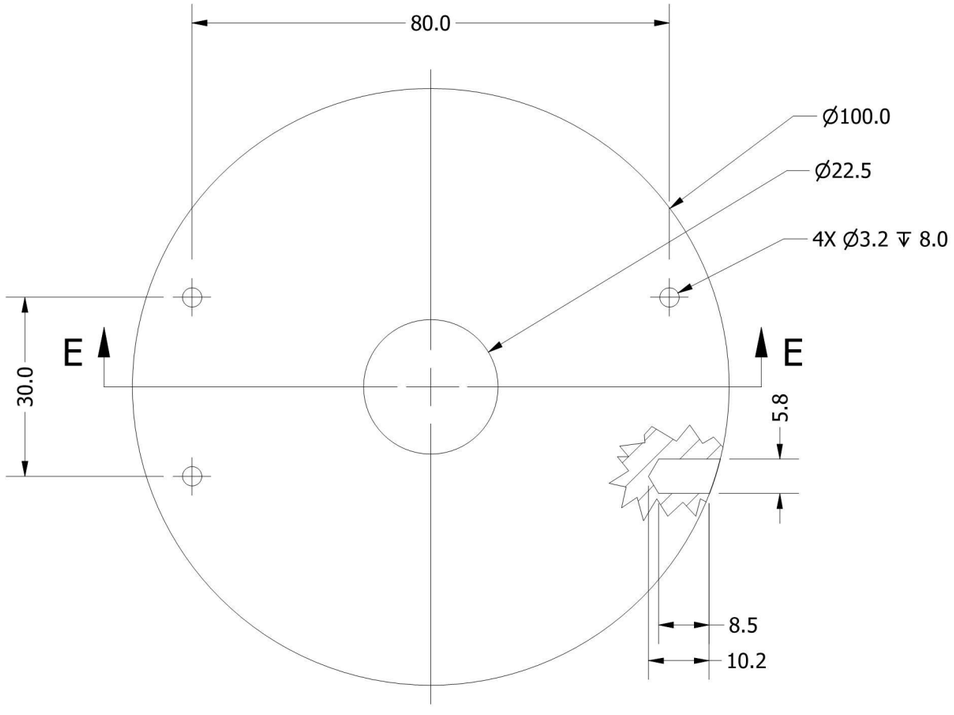
DETAIL B
SCALE 2 : 1

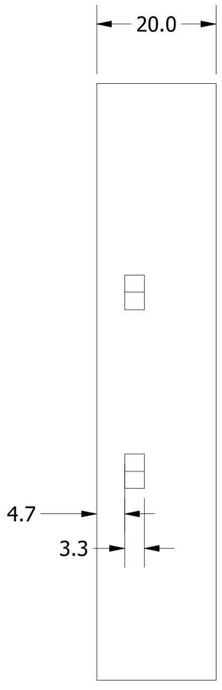


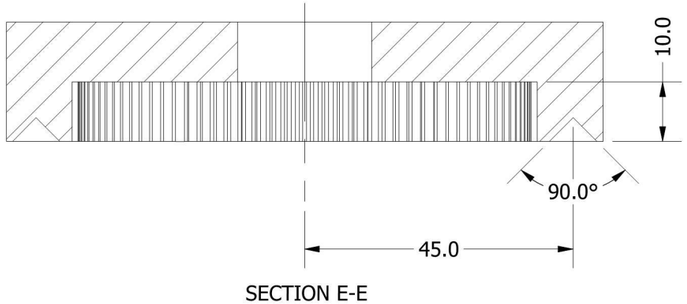
DETAIL L
SCALE 1 : 1

DRN BY: M. KANTOR	TITLE: ENCLOSURE TOP		
DATE: 2021-06-30			
DRW. TYPE: DETAIL			
SCALE: 1 : 1	PART NO: 1002		
MACEWAN UNIVERSITY	SIZE: B	REV: 0	PG: 5 OF 13

SPUR GEAR DATA	
MODULE	0.8
NUMBER OF TEETH	97
PITCH DIAMETER	77.6 mm
ROOT DIAMETER	79.6 mm
OUTSIDE DIAMETER	76 mm
PRESSURE ANGLE	20 deg



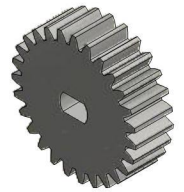


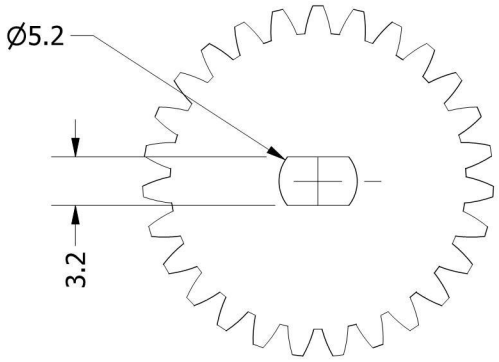


SECTION E-E
SCALE 1 : 1

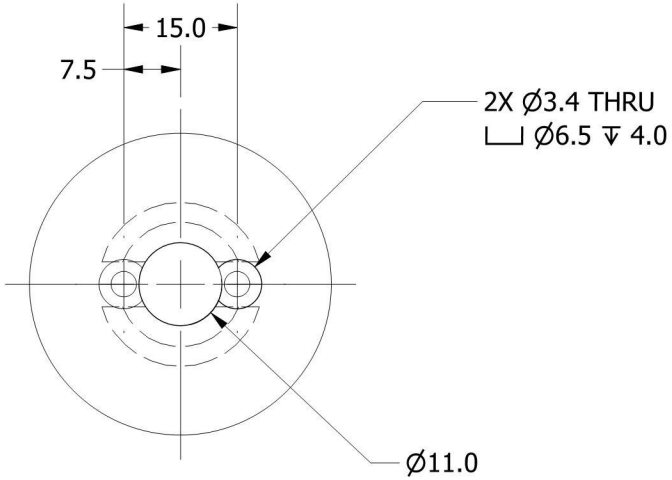
DRN BY: M. KANTOR	TITLE: ROTATING BASE		
DATE: 2021-06-30			
DRW. TYPE: DETAIL			
SCALE: 1 : 1	PART NO: 1006		
MACEWAN UNIVERSITY	SIZE: B	REV: 0	PG: 6 OF 13

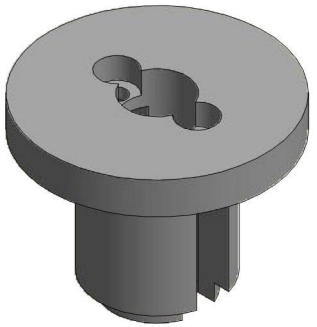
SPUR GEAR DATA	
MODULE	0.8
NUMBER OF TEETH	27
PITCH DIAMETER	21.6 mm
ROOT DIAMETER	19.6 mm
OUTSIDE DIAMETER	23.2 mm
PRESSURE ANGLE	20 deg

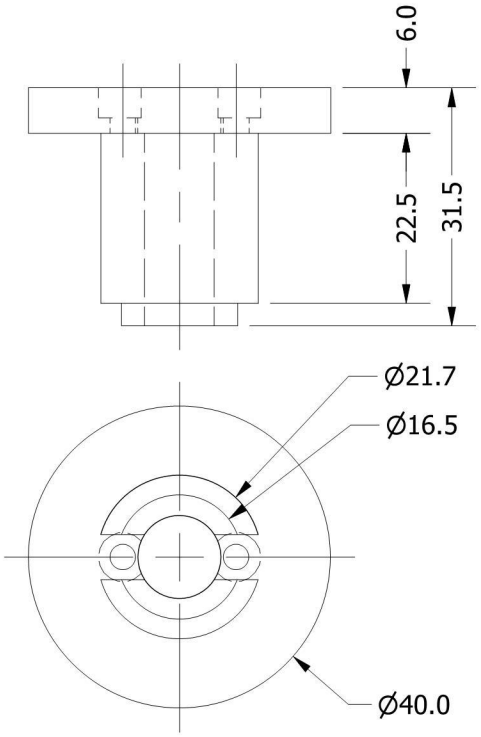


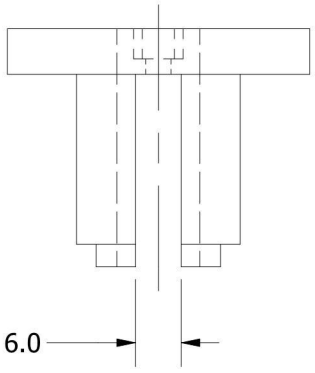


DRN BY: M. KANTOR	TITLE: SPUR GEAR		
DATE: 2021-06-30			
DRW. TYPE: DETAIL			
SCALE: 2 : 1	PART NO: 1003		
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 7 OF 13

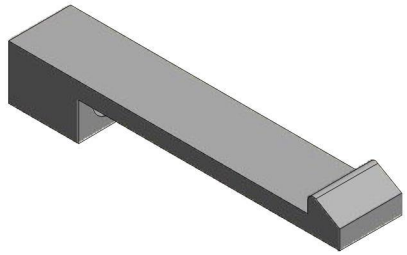


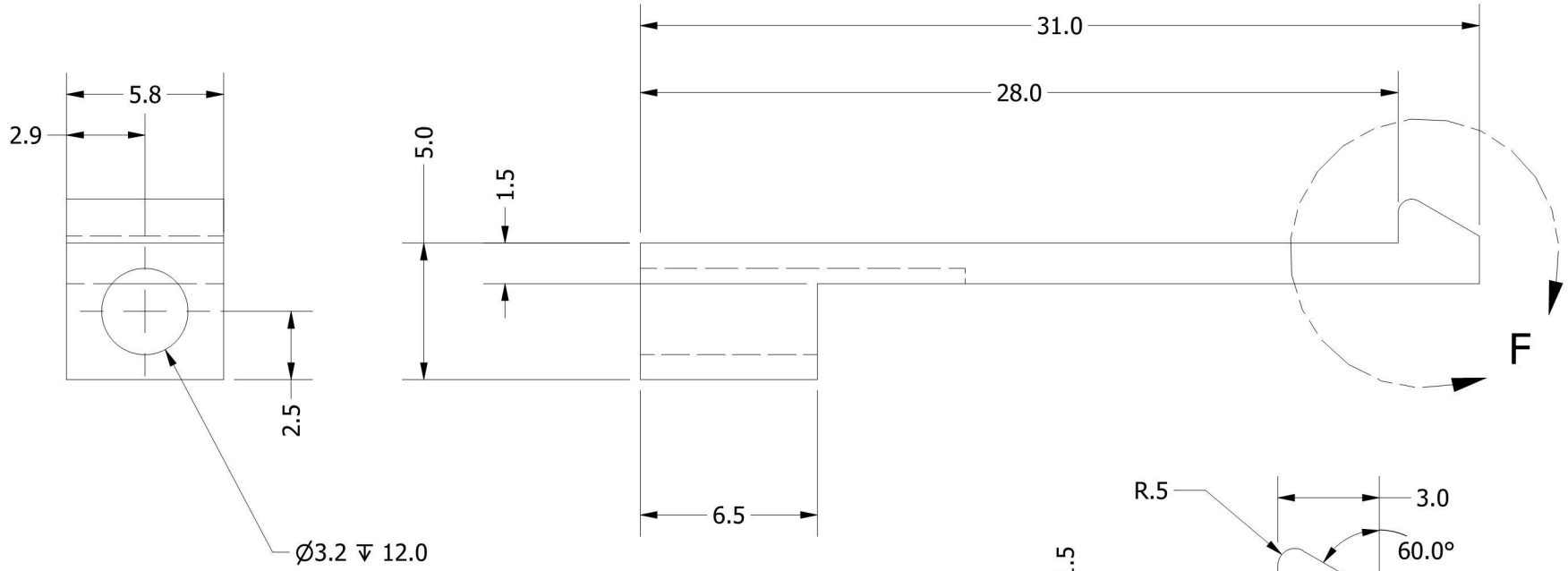


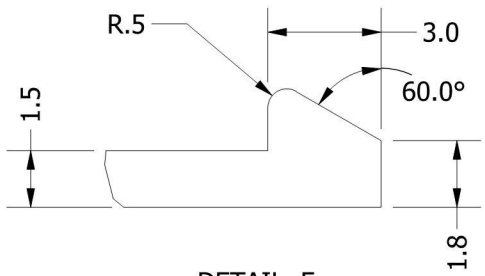




DRN BY: M. KANTOR	TITLE: SHAFT			
DATE: 2021-06-30				
DRW. TYPE: DETAIL				
SCALE: 1 : 1	PART NO: 1004			
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 8 OF 13	

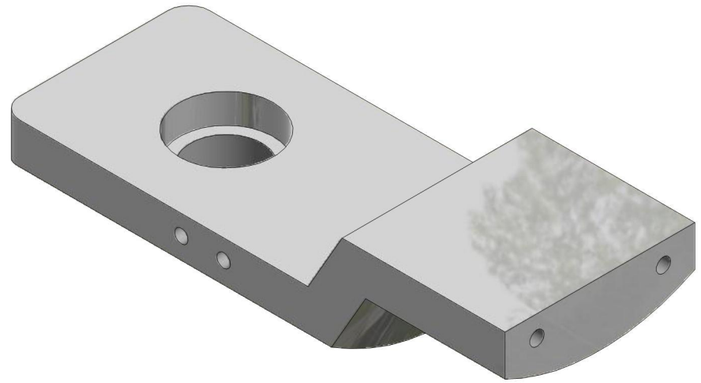


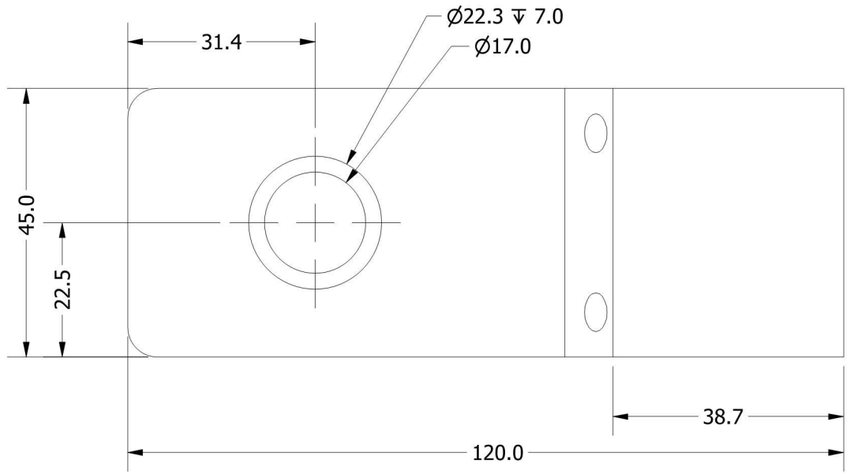


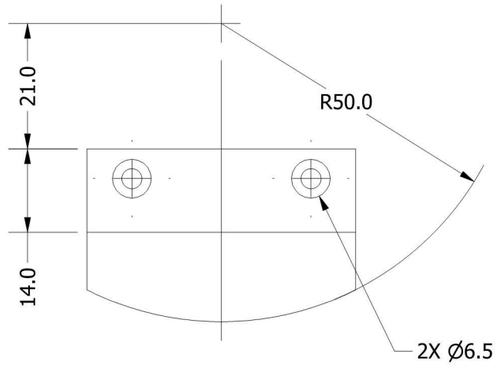


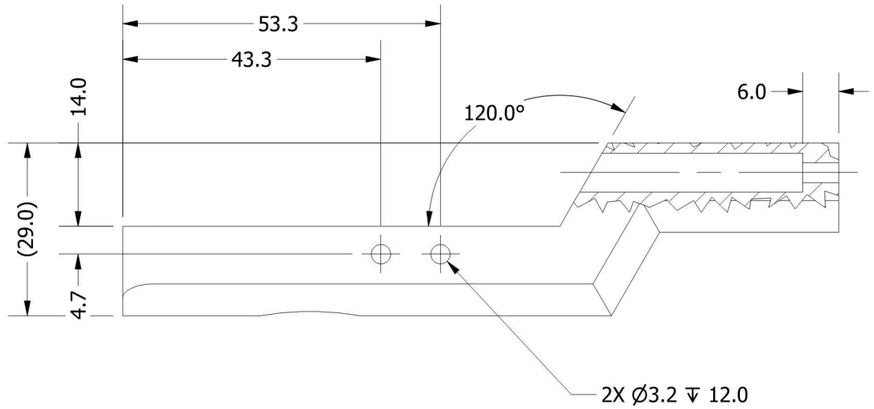
DETAIL F
SCALE 5 : 1

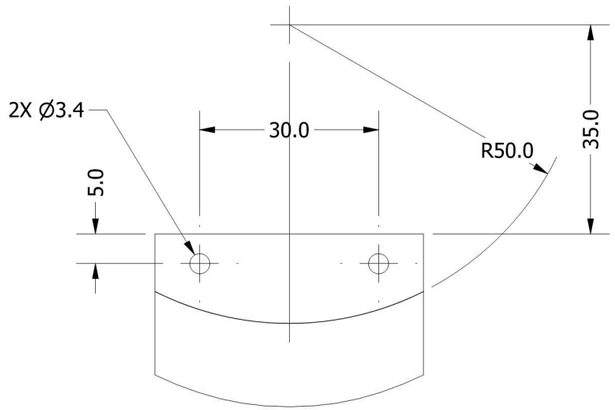
DRN BY: M. KANTOR	TITLE: SHAFT HOOK			
DATE: 2021-06-30				
DRW. TYPE: DETAIL				
SCALE: 4 : 1	PART NO: 1005			
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 9 OF 13	



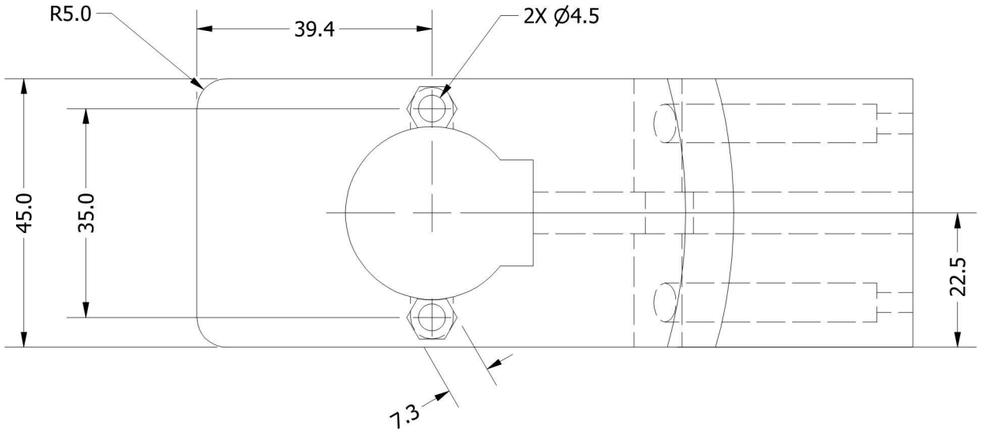


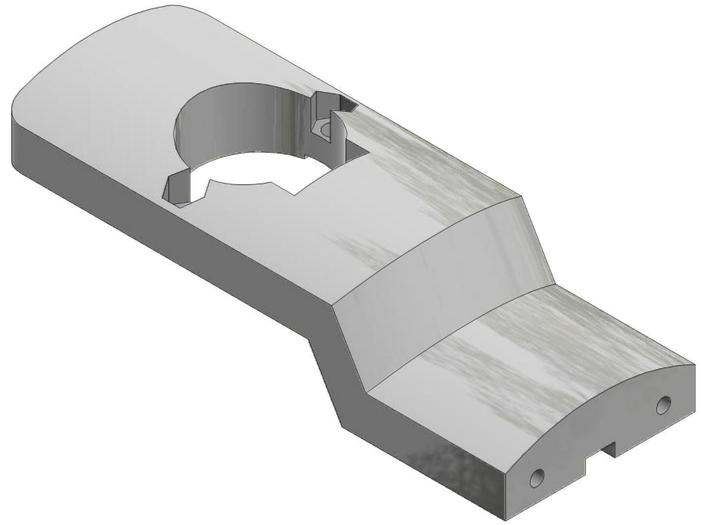


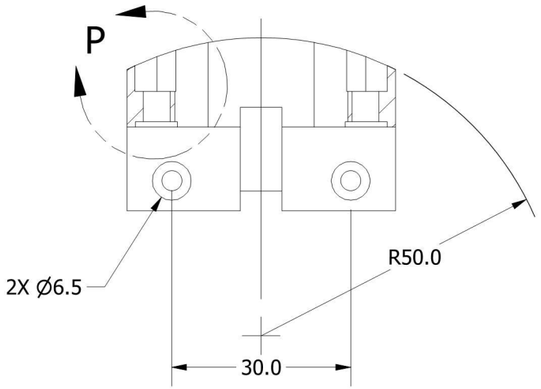




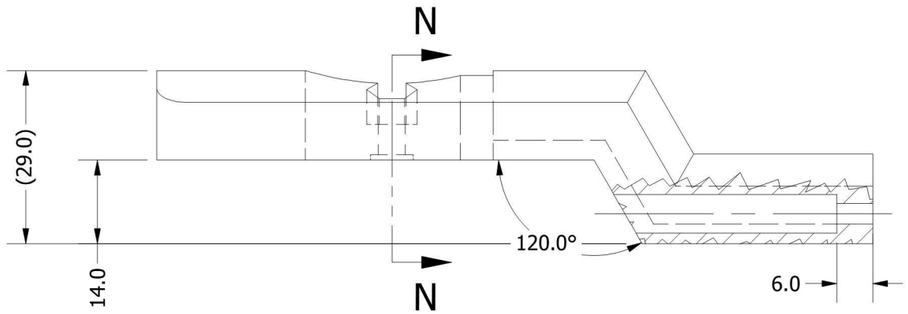
DRN BY: M. KANTOR	TITLE: BEARING ARM			
DATE: 2021-06-30				
DRW. TYPE: DETAIL				
SCALE: 1 : 1	PART NO: 1007			
MACEWAN UNIVERSITY	SIZE: B	REV: 0	PG: 10 OF 13	

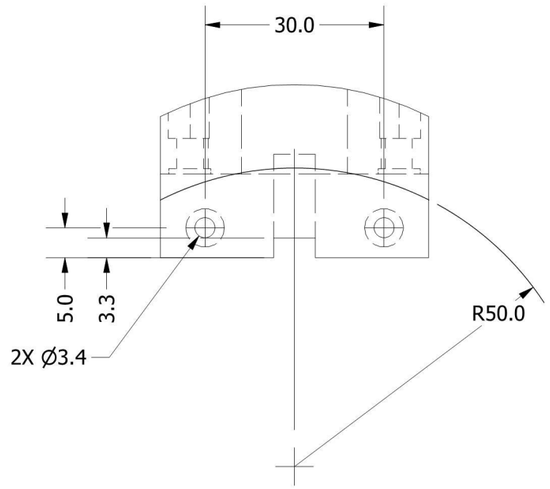


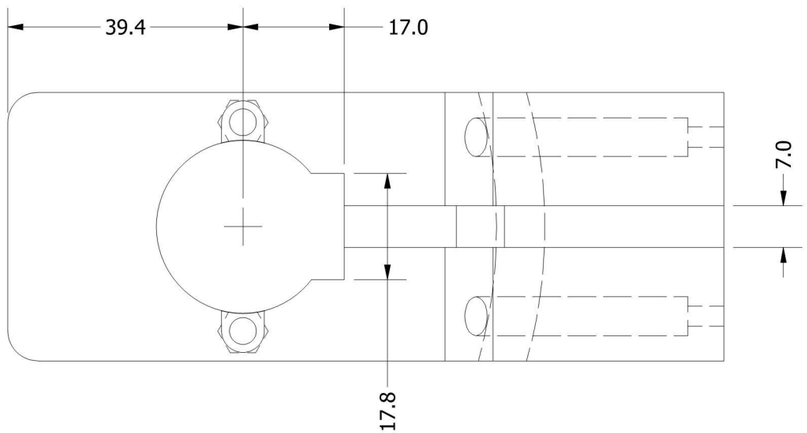


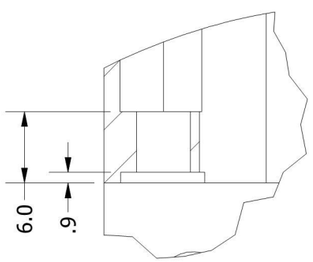


SECTION N-N
SCALE 1 : 1



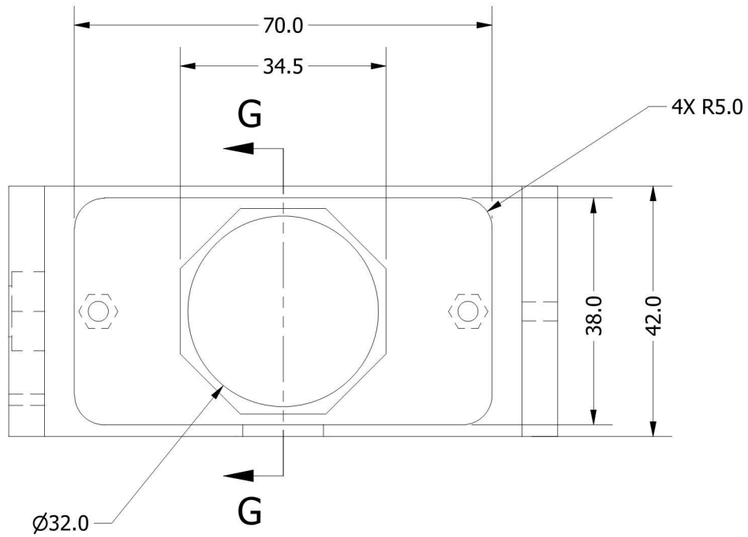


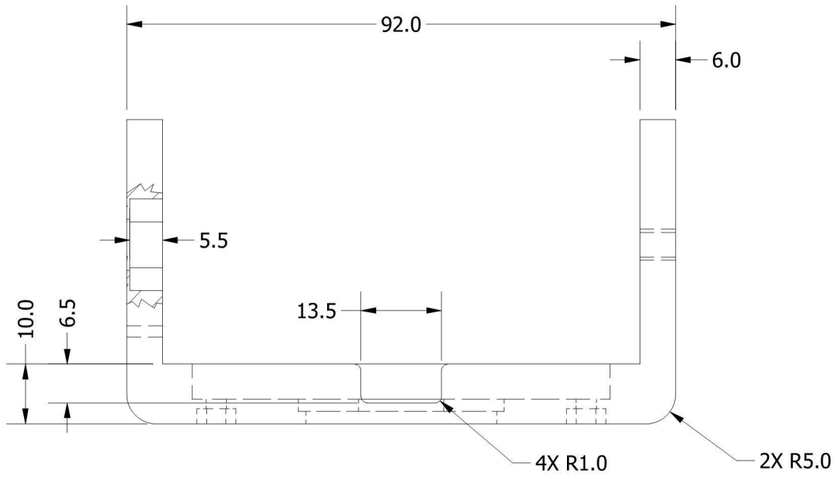


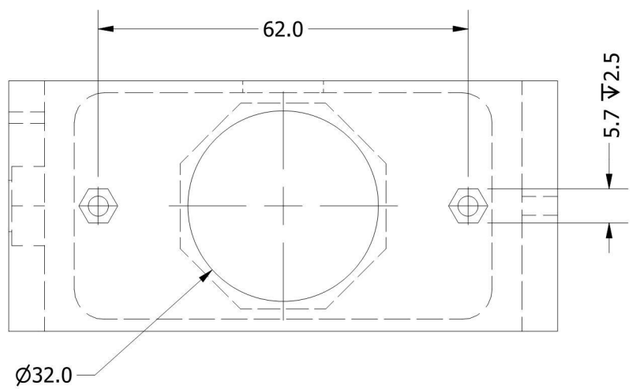


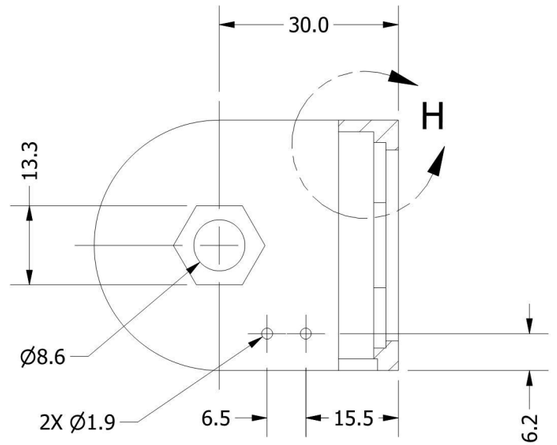
DETAIL P
SCALE 2 : 1

DRN BY: M. KANTOR	TITLE: STEPPER ARM		
DATE: 2021-06-30			
DRW. TYPE: DETAIL			
SCALE: 1 : 1	PART NO: 1008		
MACEWAN UNIVERSITY	SIZE: B	REV: 0	PG: 11 OF 13

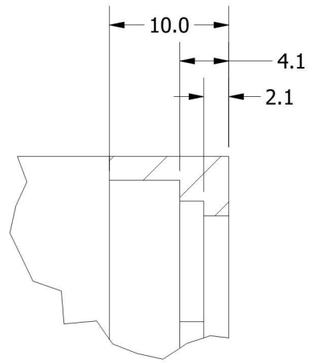




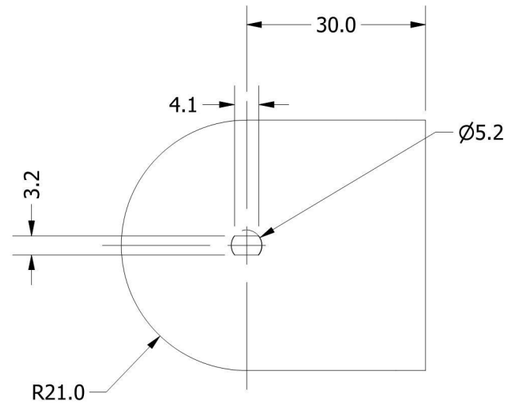


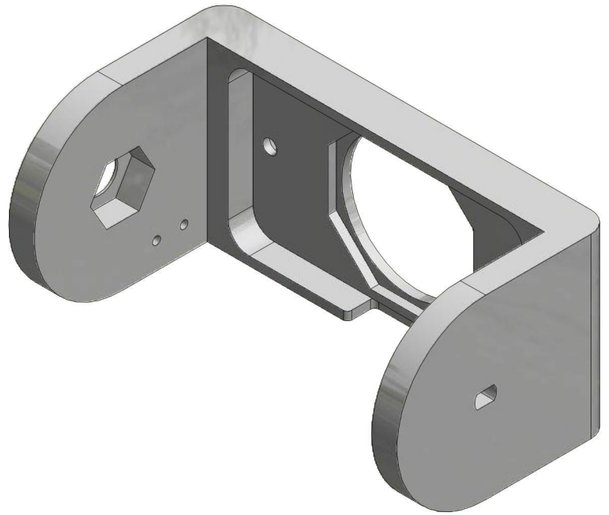


SECTION G-G
SCALE 1 : 1

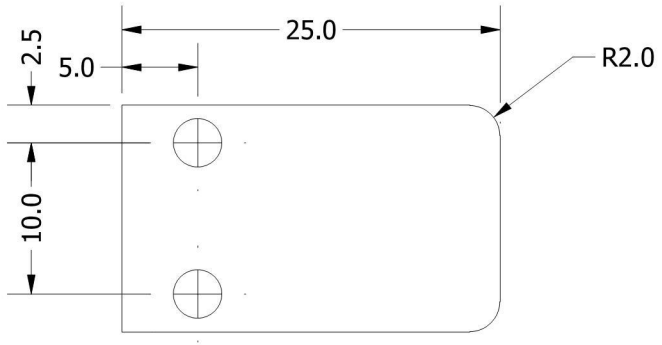


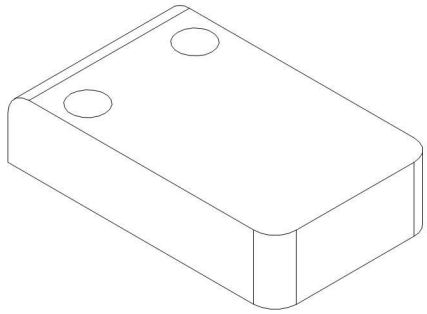
DETAIL H
SCALE 2 : 1

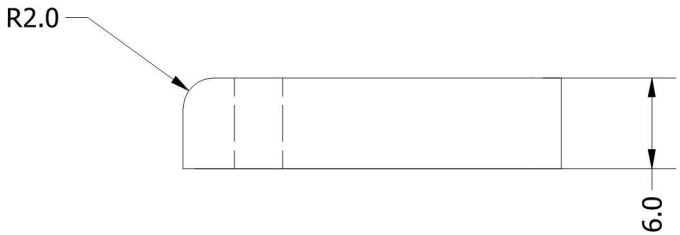




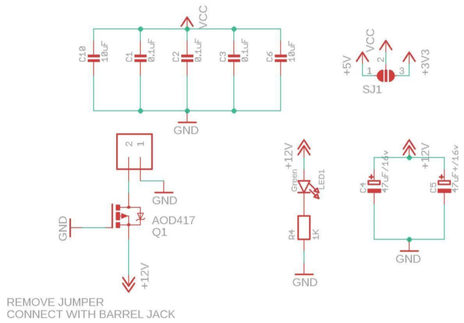
DRN BY: M. KANTOR	TITLE: SENSOR BRACKET		
DATE: 2021-06-30			
DRW. TYPE: DETAIL			
SCALE: 1 : 1	PART NO: 1010		
MACEWAN UNIVERSITY	SIZE: B	REV: 0	PG: 12 OF 13

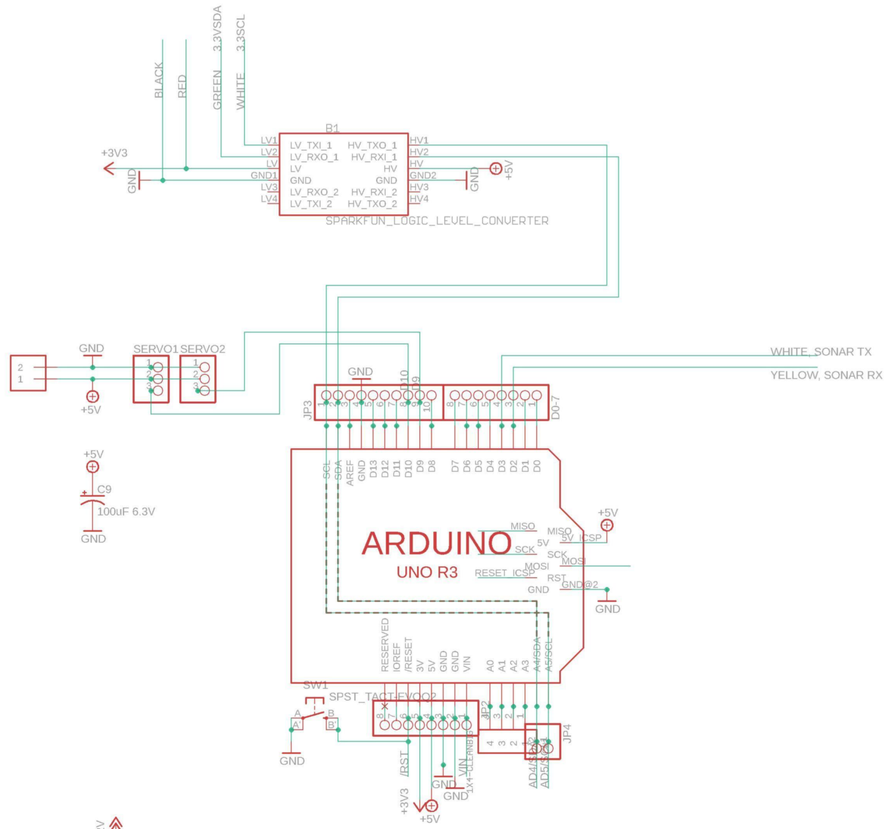


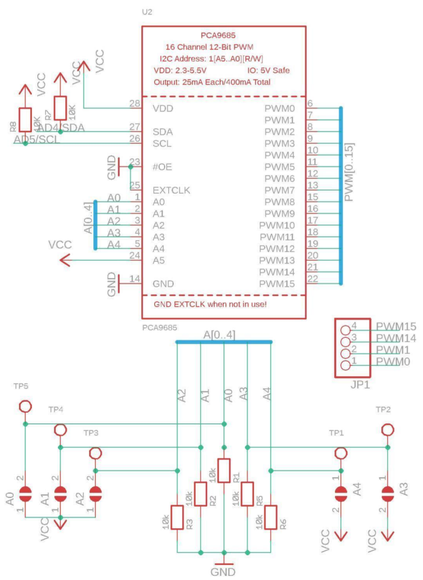


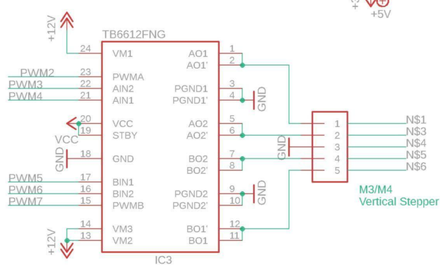


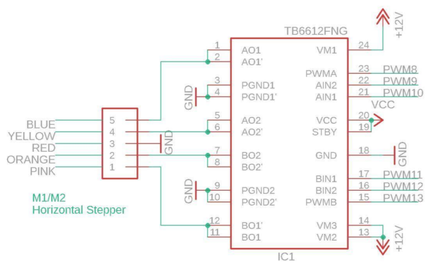
DRN BY: M. KANTOR	TITLE: ENDSTOP			
DATE: 2021-06-30				
DRW. TYPE: DETAIL				
SCALE: 2 : 1	PART NO: 1009			
MACEWAN UNIVERSITY	SIZE: A	REV: 0	PG: 13 OF 13	



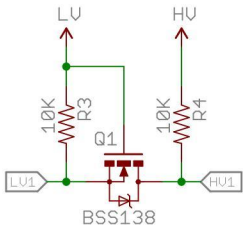


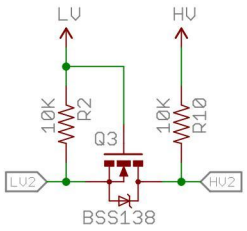


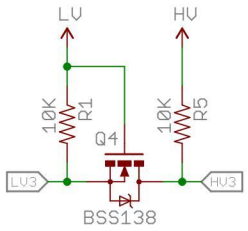


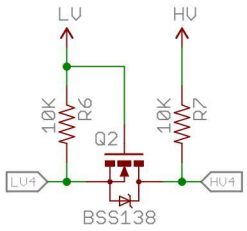


TITLE: 3DScanner_Schematic		
Document Number:		REV:
Date: 23/08/2021 12:27 PM	Sheet: 1/1	

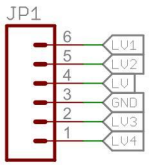




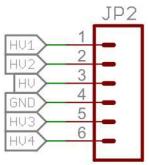




Low Voltage



High Voltage



TITLE: Logic_Level_Bidirectional

Design by: Patrick Alberts

REV: V01

Date: 9/26/2013 10:57:16 AM

Sheet: 1/1

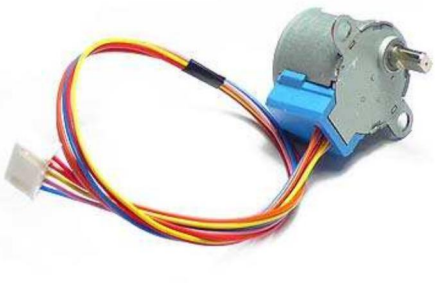
Bill of Materials - Open Source 3D Scanner							
Item Description	**Manufacturer**	**Mfg Part No**	**Vendor**	**Vendor Part No.**	**Price ($CAD)**	**Qty**	**Total**
Arduino UNO	Arduino	A000066	Mouser	782-A000066	$30.63	1	$30.63
Adafruit Motorshield	Adafruit	1438	Mouser	485-1438	$27.53	1	$27.53
I2c Logic Level Converter	Sparkfun	BOB-12009	Mouser	474-BOB-12009	$4.07	1	$4.07
5V Stepper Motor	Seeed Studio	105990072	Mouser	713-105990072	$9.52	2	19.04
TF02 PRO Lidar Sensor	Benewake	TF02PROI2C	Mouser	234-TF02PROI2C	$110.26	1	110.26
5V Stepper Motor	Seeed Studio	105990072	Mouser	713-105990072	$9.52	2	19.04
Microswitch	TE Connectivity	UP01DTANLA0	Mouser	506-UP01DTANLA04	$1.63	1	$1.63
Fasteners and Hardware							
Hex Nut - 1/4-20	N/A	N/A	CanadaBolts	IZP117	$0.17	1	$0.17
HEX NUT - M3	N/A	N/A	CanadaBolts	MZP137	$0.15	4	$0.60
HEX NUT - M4	N/A	N/A	CanadaBolts	MS1139	$0.20	10	$2.00
Lock Nut - M8	N/A	N/A	CanadaBolts	MZP243	$0.18	1	$0.18
Mach Screw M2x8	N/A	N/A	CanadaBolts	MZP111-4472	$0.31	2	$0.62
SHCS M3x6	N/A	N/A	CanadaBolts	MS133AF-4771	$0.41	4	$1.64
SHCS M3x8	N/A	N/A	CanadaBolts	MS133AF-4772	$0.42	6	$2.52
SHCS M3x12	N/A	N/A	CanadaBolts	MS133AF-4774	$0.45	2	$0.90
SHCS M3x16	N/A	N/A	CanadaBolts	MS133AF-4775	$0.47	6	$2.82
BCHS M4-10mm	N/A	N/A	CanadaBolts	MS1135-4973	$0.45	6	$2.70
608ZZ Ball bearing	N/A	N/A	BearingsCanada	608ZZ-8X22X7-CARBON-MINIATURE	$2.16	1	$2.16
1/4" Loose Bearing Balls (pack 100)	N/A	N/A	BearingsCanada	KIT16346	$8.73	1	$8.73
3D Printed Parts:							
Sensor Bracket (Dual Side)					$0.80	1	$0.80
Enclosure					$1.49	1	$1.49
Enclosure Base Plate					$0.55	1	$0.55
Scanner Head - Base Plate					$1.37	1	$1.37
Scanner Head - Arms (Both, and endstop)					$1.87	1	$1.87
Shaft + hooks					$0.25	1	$0.25
Note: 3D Printed part costs are MATERIAL COST ONLY, as pricing will vary greatly.							
						Subtotal	$243.57
						GST	$12.18
						TOTAL	$255.75

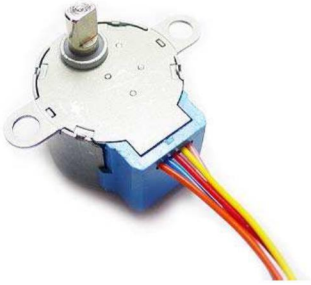
APPENDIX C: Datasheets

See overleaf.

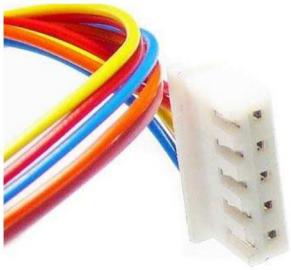
28BYJ-48 – 5V Stepper Motor

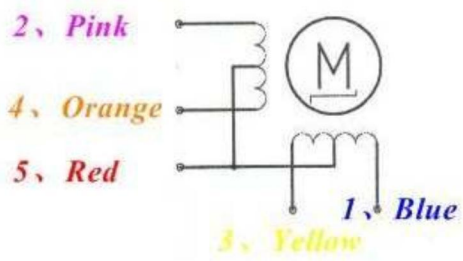
The 28BYJ-48 is a small stepper motor suitable for a large range of applications.

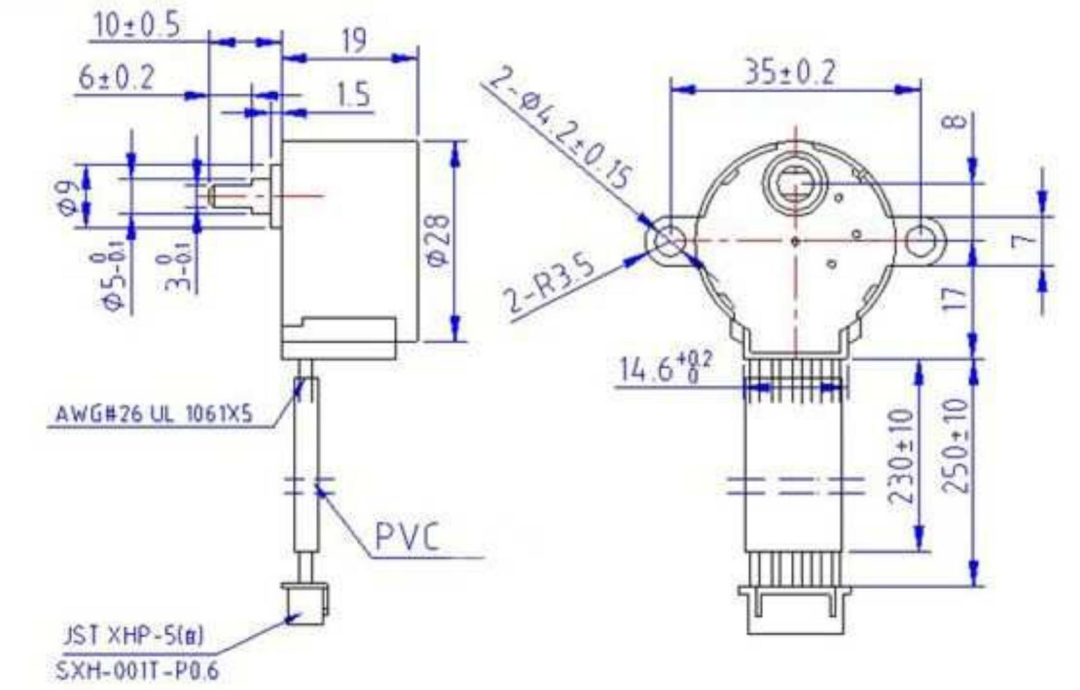




Rated voltage :	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25℃)
Idle In-traction Frequency	> 600Hz
Idle Out-traction Frequency	> 1000Hz
In-traction Torque	>34.3mN.m(120Hz)
Self-positioning Torque	>34.3mN.m
Friction torque	600-1200 gf.cm
Pull in torque	300 gf.cm
Insulated resistance	>10MΩ(500V)
Insulated electricity power	600VAC/1mA/1s
Insulation grade	A
Rise in Temperature	<40K(120Hz)
Noise	<35dB(120Hz,No load,10cm)
Model	28BYJ-48 – 5V



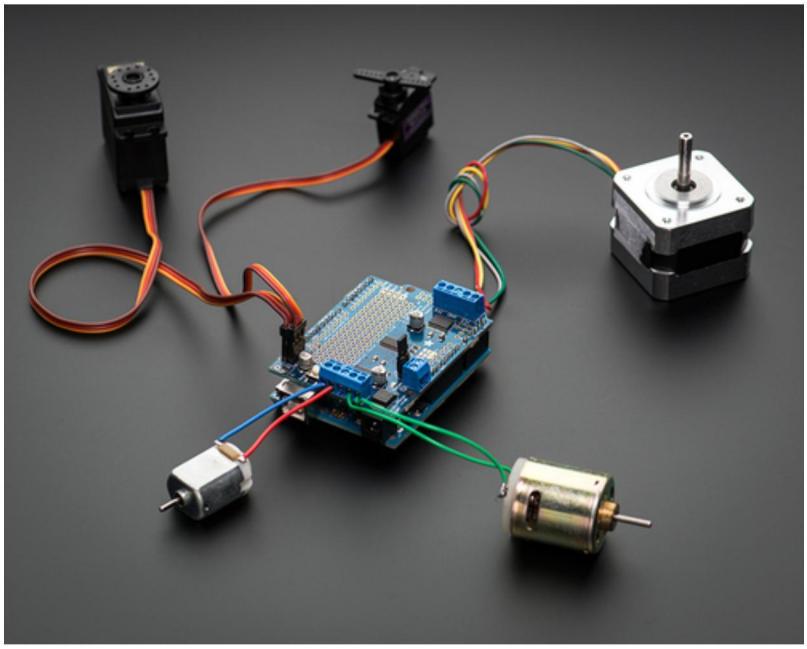




adafruit learning system

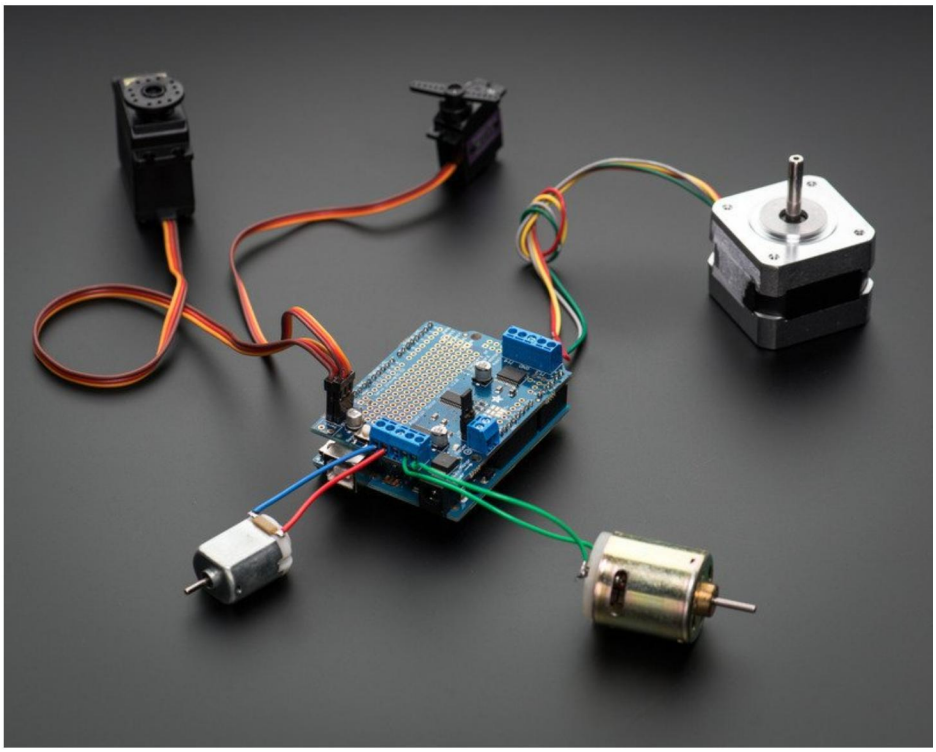
Adafruit Motor Shield V2

Created by lady ada



Last updated on 2020-01-20 05:44:57 PM UTC

Overview



The original Adafruit Motorshield kit is one of our most beloved kits, which is why we decided to make something even better. We have upgraded the shield kit to make the bestest, easiest way to drive DC and Stepper motors. This shield will make quick work of your next robotics project! We kept the ability to drive up to 4 DC motors or 2 stepper motors, but added many improvements:

Instead of a L293D darlington driver, we now have the TB6612 MOSFET drivers with 1.2A per channel current capability (you can draw up to 3A peak for approx 20ms at a time). It also has much lower voltage drops across the motor so you get more torque out of your batteries, and there are built-in flyback diodes as well.

Instead of using a latch and the Arduino's PWM pins, we have a **fully-dedicated PWM driver chip** onboard. This chip handles all the motor and speed controls over I2C. Only two GPIO pins (SDA & SCL) plus 5v and GND. are required to drive the multiple motors, and since it's I2C you can also connect any other I2C devices or shields to the same pins. This also makes it drop-in compatible with any Arduino, such as the Uno, Leonardo, Due and Mega R3.

Completely stackable design: 5 address-select pins means up to 32 stackable shields: that's 64 steppers or 128 DC motors! What on earth could you do with that many steppers? I have no idea but if you come up with something send us a photo because that would be a pretty glorious project.

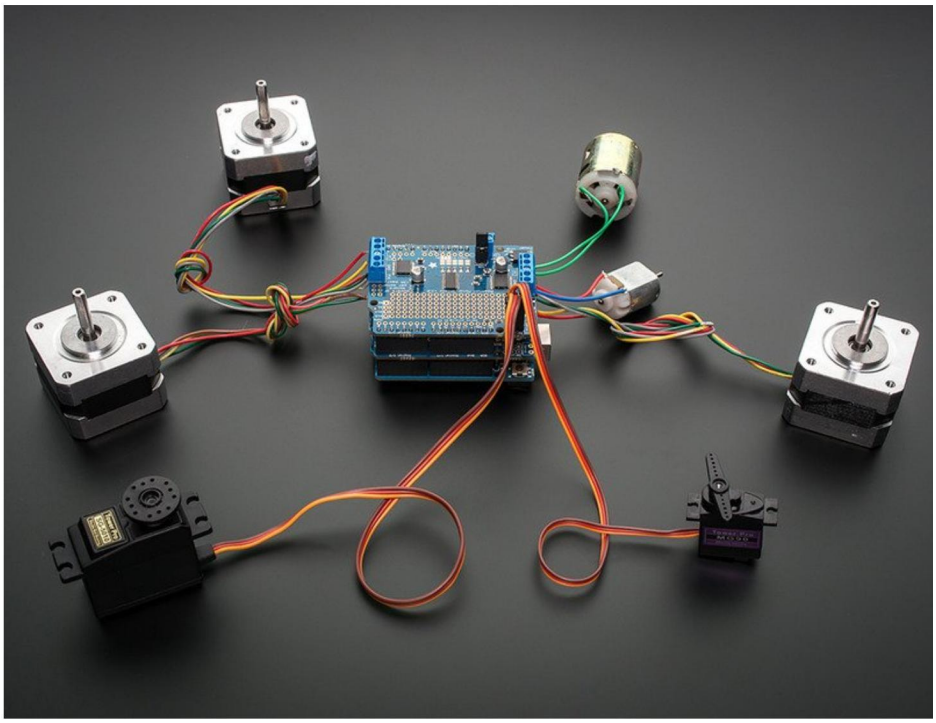
Lots of other little improvements such as a polarity protection FET on the power pins and a big prototyping area. And the shield is assembled and tested here at Adafruit so all you have to do is solder on straight or stacking headers and the terminal blocks.

Lets check out these specs again:

- **2 connections for 5V 'hobby' servos** connected to the Arduino's high-resolution dedicated timer - no jitter!
- 4 H-Bridges: TB6612 chipset provides **1.2A per bridge** (3A for brief 20ms peaks) with thermal shutdown

protection, internal kickback protection diodes. Can run motors on 4.5VDC to 13.5VDC.

- **Up to 4 bi-directional DC** motors with individual 8-bit speed selection (so, about 0.5% resolution)
- **Up to 2 stepper motors** (unipolar or bipolar) with single coil, double coil, interleaved or micro-stepping.
- Motors automatically disabled on power-up
- Big terminal block connectors to easily hook up wires (18-26AWG) and power
- Arduino reset button brought up top
- Polarity protected 2-pin terminal block and jumper to connect external power, for separate logic/motor supplies
- Tested compatible with Arduino UNO, Leonardo, ADK/Mega R3, Diecimila & Duemilanove. Works with Due with 3.3v logic jumper. Works with Mega/ADK R2 and earlier with 2 wire jumpers.
- Download the easy-to-use Arduino software library, check out the examples and you're ready to go!
- **5v** or **3.3v** compatible logic levels - jumper configurable.



As of Arduino 1.5.6-r2 BETA, there is a bug in the Due Wire library that prevents multiple Motor Shields from working properly with the Due!

FAQ

How many motors can I use with this shield?

You can use 2 DC hobby servos that run on 5V and up to 4 DC motors or 2 stepper motors (or 1 stepper and up to 2 DC motors) that run on 5-12VDC

Can I connect more motors?

Yes, by stacking shields! Every shield you stack on will add 4 DC motors or 2 stepper motors (or 1 more stepper and 2 more DC motors).
You will not gain more servo connections as the servo contacts go to pin #9 and #10 on the Arduino.

What if I also need some more servos?

Check out our lovely servo shield, also stackable with this motor shield and adds 16 free-running servos per shield
http://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield

What Arduinos is this shield compatible with?

It is tested to work with Duemilanove, Diecimila, Uno (all revisions), Leonardo and Mega/ADK R3 and higher.

It can work with Mega R2 and lower if you solder a jumper wire from the shield's SDA pin to Digital 20 and the SCL pin to Digital 21

For use with the Due or other 3.3v processors, you must configure the board for 3.3v logic levels. Find the set of 3 pads labeled "Logic". Cut the small trace between the center pad and 5v and add a jumper from 3.3v to the center.

As of Arduino 1.5.6-r2 BETA, there is a bug in the Due Wire library that prevents multiple Motor Shields from working properly!

I get the following error trying to run the example code: "error: Adafruit_MotorShield.h: No such file or directory...."

Make sure you have installed the Adafruit_MotorShield library

How do I install the library?

Check the tutorial page on the subject here http://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software

HELP! My motor doesnt work! - HELP! My motor doesnt work!...But the servos work FINE!

Is the power LED lit? The Stepper and DC motor connections will not work if the onboard green Power LED is not lit brightly!
You must connect 5-12VDC power to the shield through the POWER terminal blocks or through the DC barrel jack on the Arduino and VIN jumper.

What is the green Power LED for?

The LED indicates the **DC/Stepper motor power supply is working**. If it is not lit brightly, then the DC/Stepper motors will not run. The servo ports are 5V powered and does not use the DC motor supply

What pins are/are not used on the motor shield?

GND and either 5v (default) or 3.3v are required to power the logic on-board. (5v or 3v operation is selectable via jumper)

The shield uses the SDA and SCL i2c pins to control DC and stepper motors. On the Arduino UNO these are also known as A4 and A5. On the Mega these are also known as Digital 20 and 21. On the Leonardo these are also known as digital 2 and 3. Do not use those pins on those Arduinos with this shield with anything other than an i2c sensor/driver.

Since the shield uses I2C to communicate, you can connect any other i2c sensor or driver to the SDA/SCL pins as long as they do not use address **0x60** (the default address of the shield) or **0x70** (the 'all call' address that this chip uses for group-control)

If you want to use the servo connections, they are on pins #9 and #10. If you do not use the connector then those pins are simply not used.

You can use any other pins for any other use

Note that pins A4 and A5 are connected to SDA and SCL for compatibility with classic Arduinos. These pins are not available for use on other processors.

How can I connect to the unused pins?

All pins are broken out into 0.1" spaced header along the edges of the shield

My Arduino freaks out when the motors are running! Is the shield broken?

Motors take a lot of power, and can cause 'brownouts' that reset the Arduino. For that reason the shield is designed for seperate (split) supplies - one for the electronics and one for the motor. Doing this will prevent brownouts. Please read the user manual for information about appropriate power supplies.

I'm trying to build this robot and it doesn't seem to run on a 9V battery....

You **cannot** power motors from a 9V battery. You must use AA batteries or a lead acid battery for motors.

Can this shield control small 3V motors?

Not really, its meant for larger, 5V+ motors. It does not work for 3V motors unless you overdrive them at 5V and then they will burn out faster

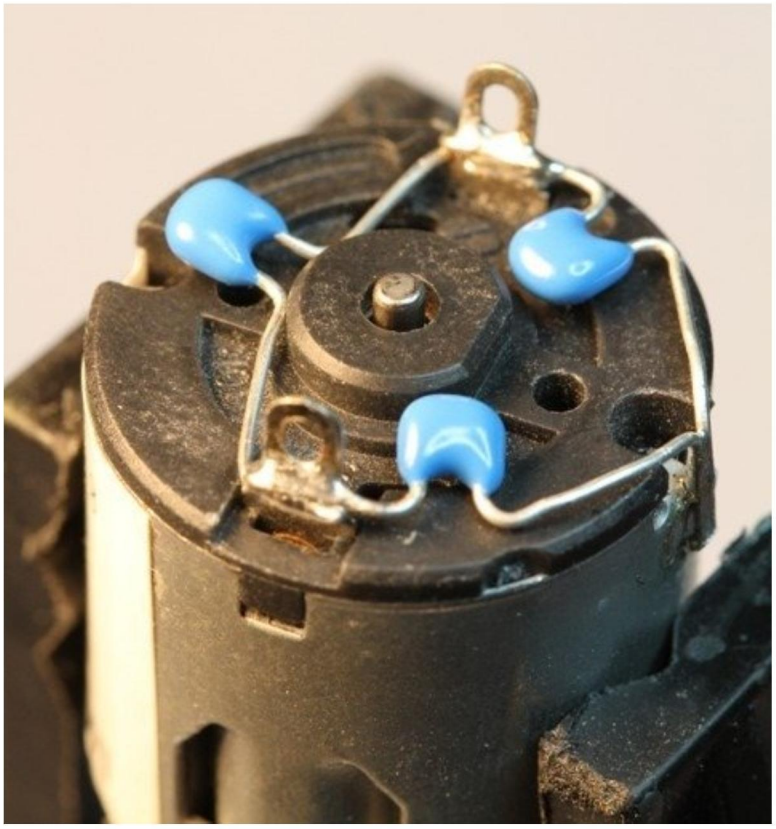
I have good solid power supplies, but the DC motors seem to 'cut out' or 'skip'.

Try soldering a ceramic or disc 0.1uF capacitor between the motor tabs (on the motor itself!) this will reduce noise that could be feeding back into the circuit (thanks macegr!)

When the motors start running nothing else works.

Many small DC motor have a lot of "brush noise". This feeds back into the Arduino circuitry and causes unstable operation. This problem can be solved by soldering some 0.1uF ceramic noise suppression capacitors to the motor.

You will need 3 total. 1 between the motor terminals, and one from each terminal to the motor casing.



But my motor already has a capacitor on it and it still doesn't work.

These motors generate a lot of brush noise and usually need the full 3-capacitor treatment for adequate suppression.

Why don't you just design capacitors into the shield?

They would not be effective there. The noise must be suppressed at the source or the motor leads will act like antennae and broadcast it to the rest of the system!

Why won't my stepper motor go any faster?

Since the shield is controlled by i2c, the maximum step rate is limited by the i2c bus speed. The default bus speed is 100KHz and can be increased to 400KHz by editing the library file in your Arduino installation folder. The file can be found in ***hardware/libraries/wire/utility/twi.h***.

Find the line with: "#define TWI_FREQ 100000L"
and change it to "#define TWI_FREQ 400000L"

Or, you can add the following code to your setup() function: (Note: this line must be inserted *after* the call to begin())

```
TWBR = ((F_CPU /400000l) - 16) / 2; // Change the i2c clock to 400KHz
```

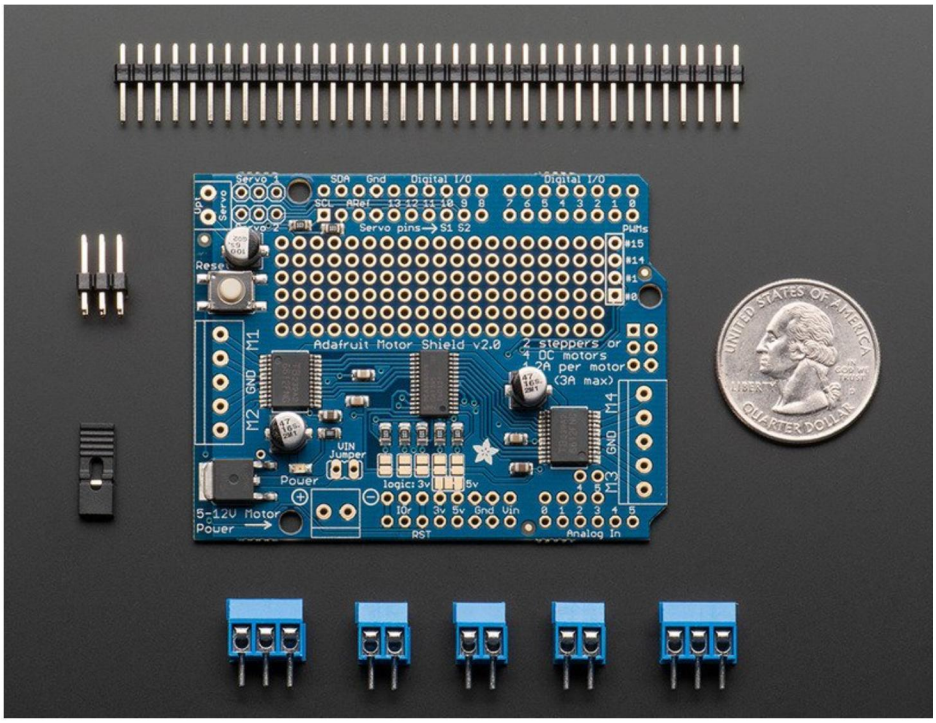
What I2C addresses are used by this shield?

The shield is addressable from 0x60-0x7F. 0x70 is an "all call" address that all boards will answer to.

My shield doesn't work with my LED backpack.

Some backpacks have a default address of 0x70. This is the "all call" address of the controller chip on the motor shield. If you re-address your backpack, it will work with the shield.

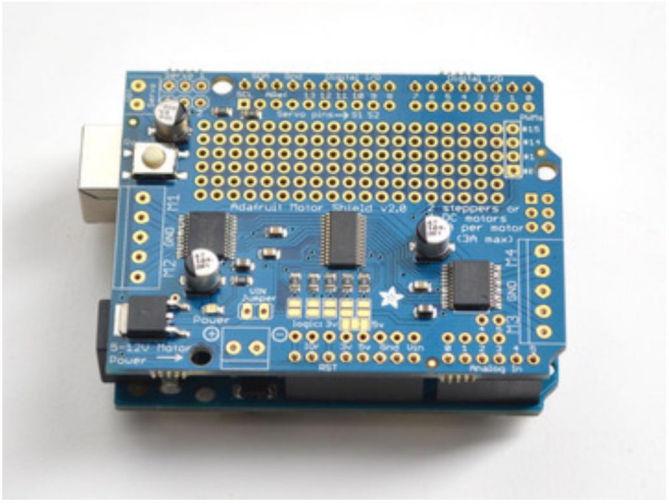
Install Headers & Terminals



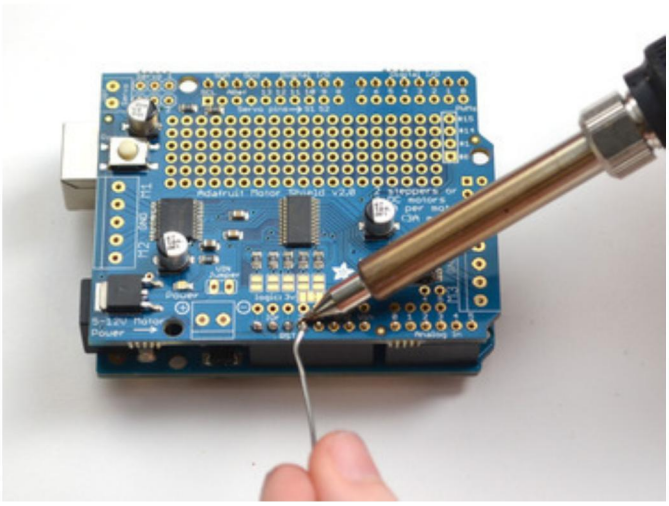
Installing Standard Headers

The shield comes with 0.1" standard header. Standard header does not permit stacking but it is mechanically stronger and they're much less expensive too! If you want to stack a shield on top, do not perform this step as it is not possible to uninstall the headers once soldered in! Skip down to the bottom for the stacking tutorial

Break apart the 0.1" header into 6, 8 and/or 10-pin long pieces and slip the long ends into the headers of your Arduino

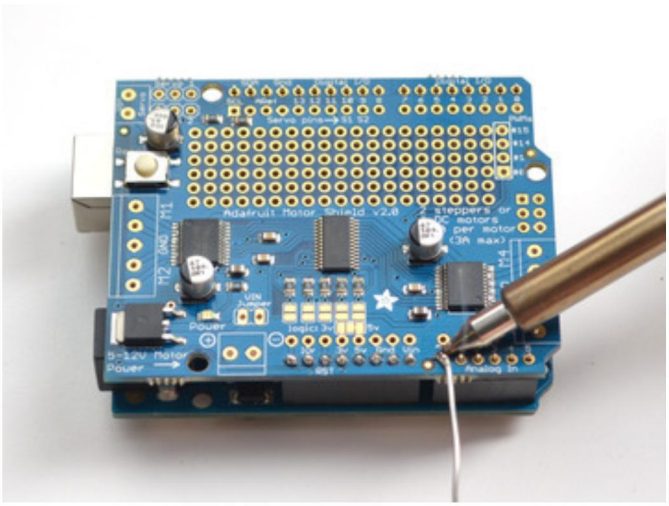


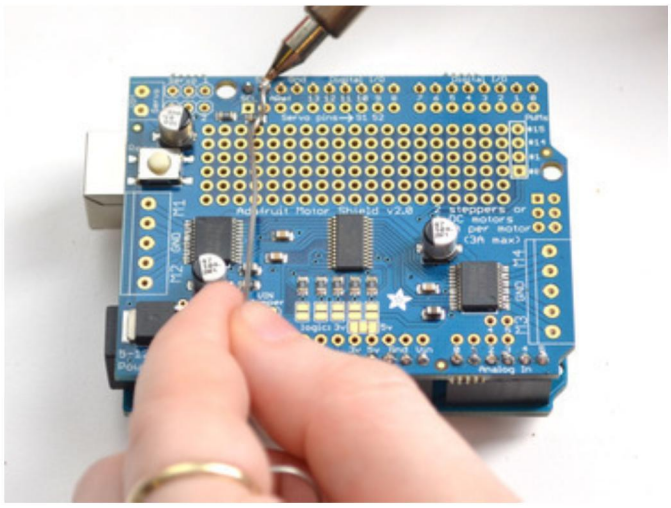
Place the assembled shield on top of the header-ed Arduino so that all of the short parts of the header are sticking through the outer set of pads

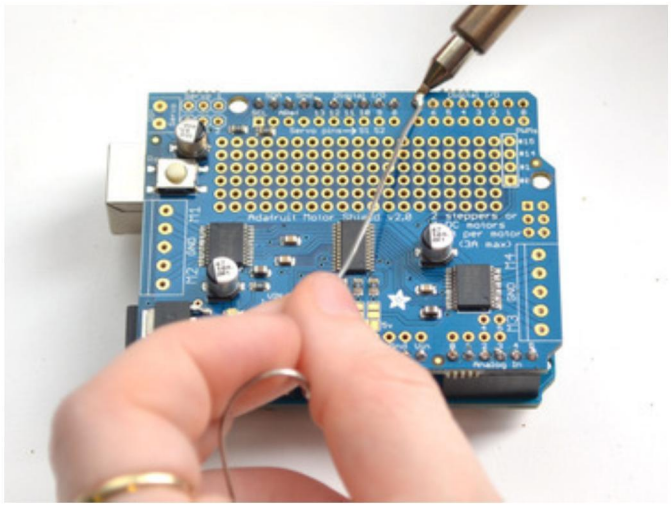


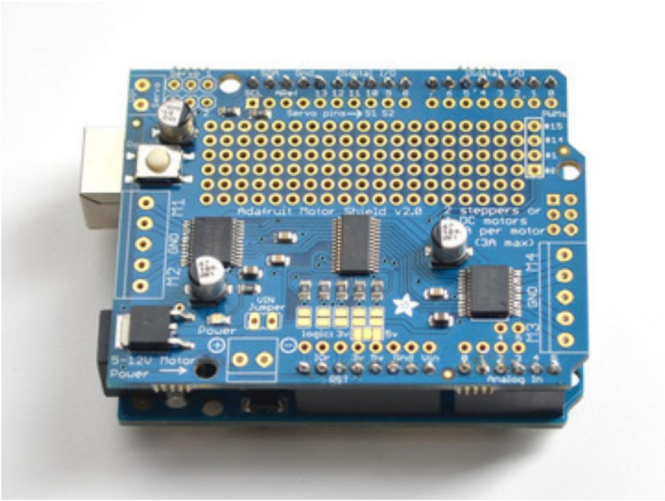
Solder each one of the pins into the shield to make a secure connection

Next, you will attach the terminal blocks, power jumper and servo connections





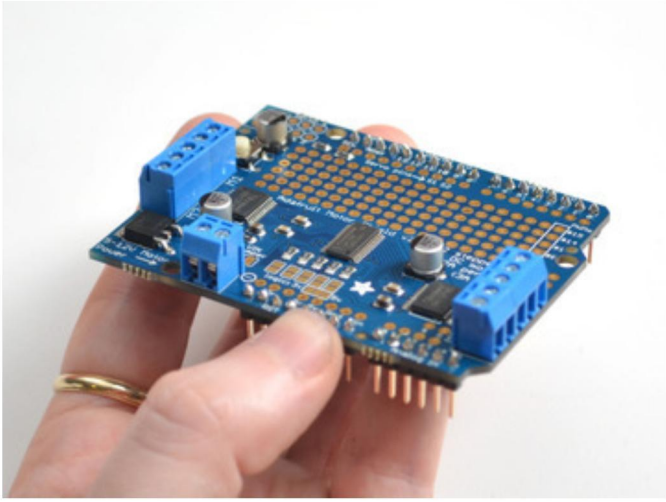




That's it! Now you can install the terminal blocks and jumper...

Installing Terminal Blocks and more

After you have installed either normal or stacking headers, you must install the terminal blocks.

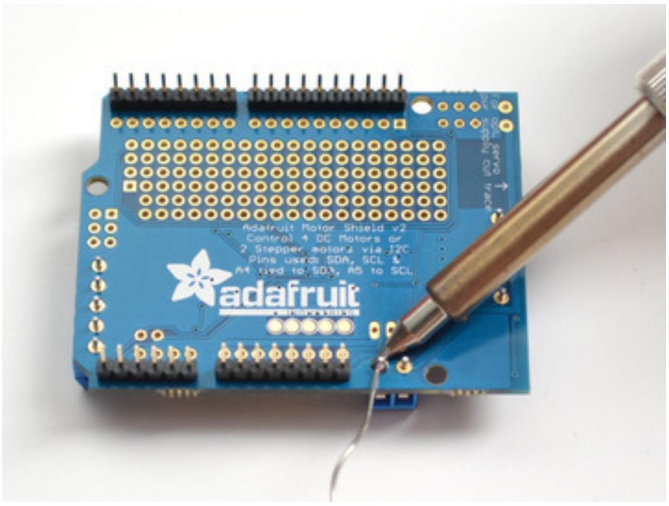


Next we will install the terminal blocks. These are how we will connect power and motors to the shield. They're much easier to use than soldering direct, just use a small screwdriver to release/attach wires!

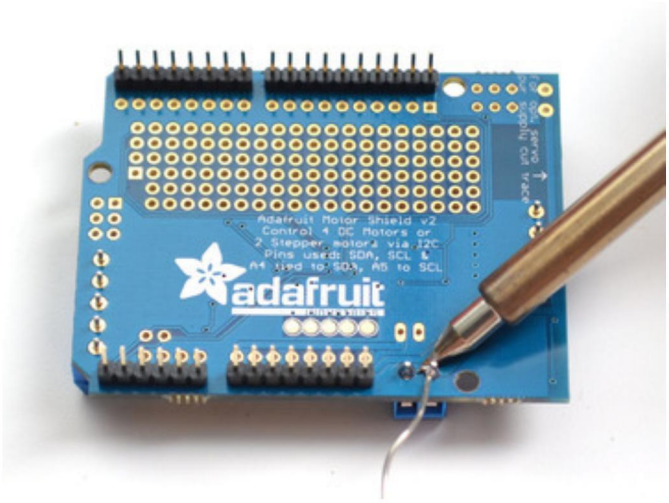
First, though, we must solder them in.

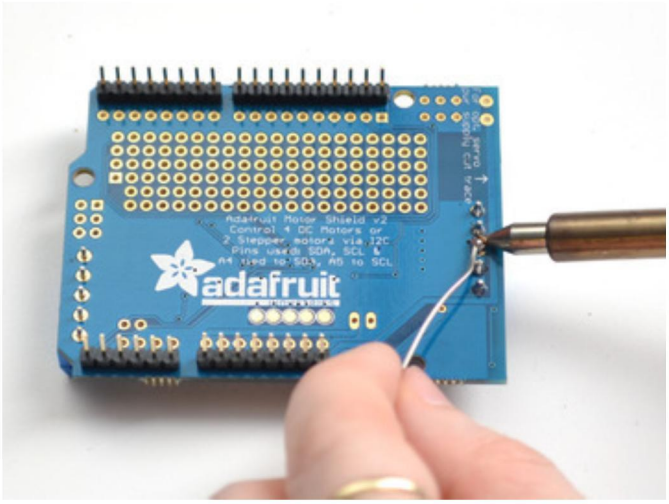
Slide the 3-pin terminal blocks into 2-pin terminal blocks so that you have 2 x 5-pin and 1 x 2-pin blocks. The two 5-pin sets go on either side. The 2-pin piece goes near the bottom of the shield. Make sure that the open holes of the terminal blocks face **out**!

Flip the board over so that you can see & solder the pins of the terminal blocks

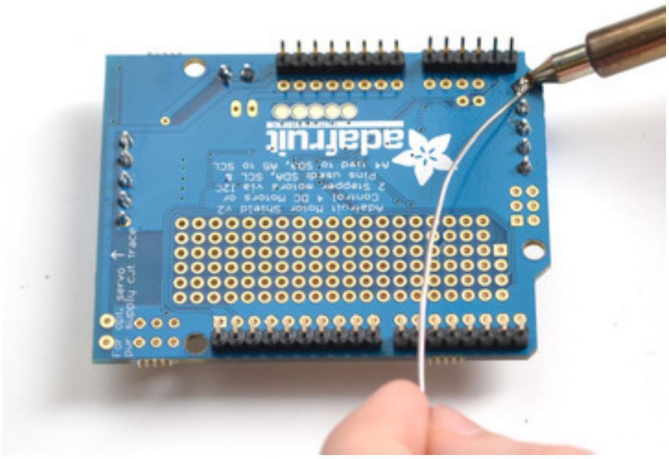


Solder in the two pins of the external power terminal-block

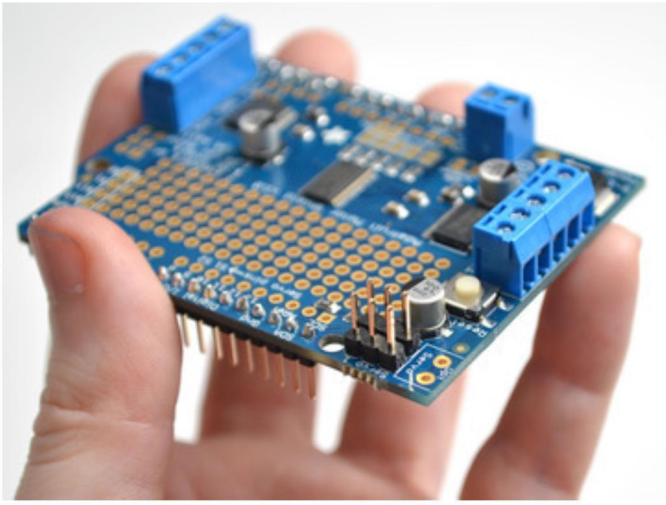




Solder in both motor blocks, 5 pads each

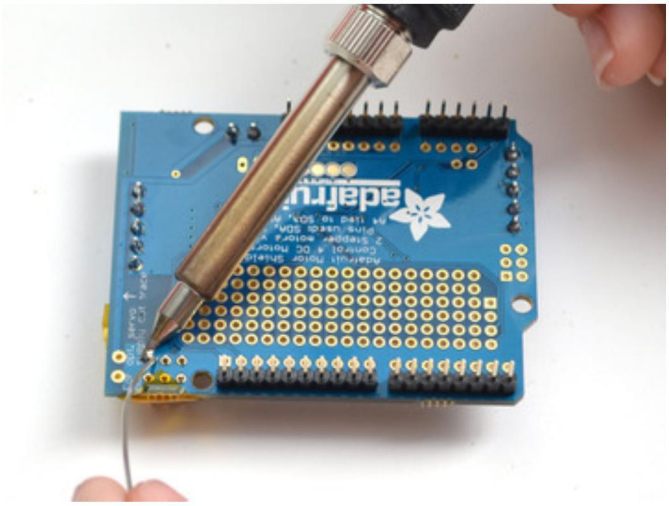


That's it for the terminal blocks. Next up, servo connections.

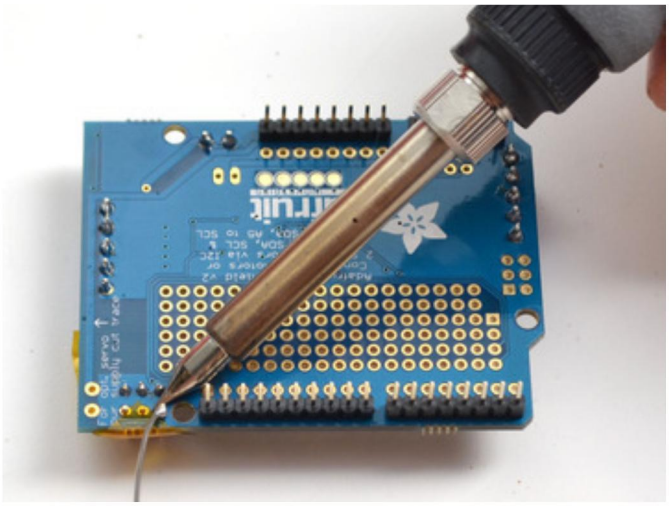


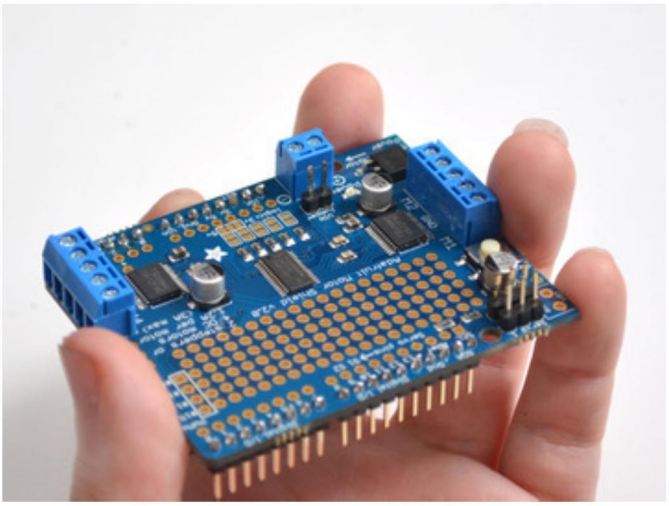
OK next up take the **2x3 pin header** and place it with the short legs down into the top corner where it says SERVO 1 and SERVO 2

You might have to sort of angle the part a little to get it to fit into both sets of 3-pin holes. we did this so it wont fall out easily when you turn it over!



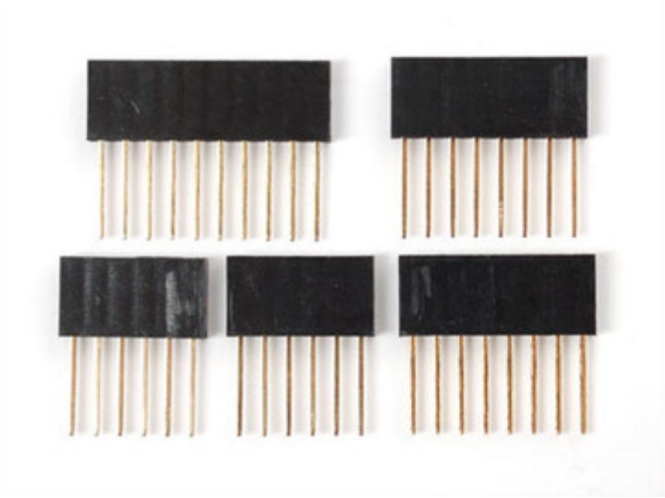
Then flip the board over and solder the 6 pins





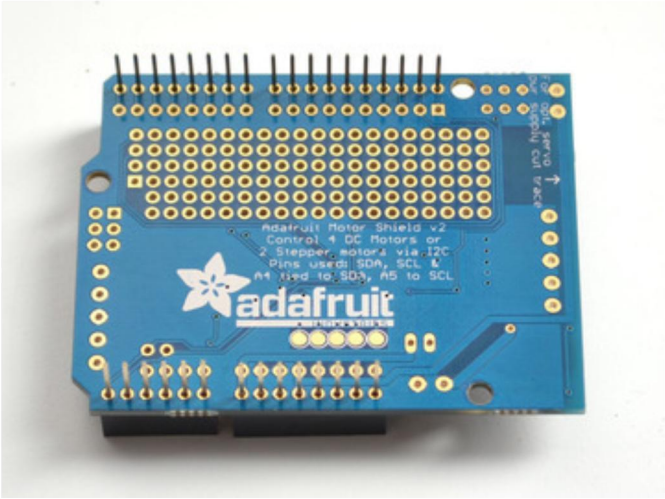
Finally, break off a 2-pin piece of header and place it next to the POWER terminal block, short legs down, tape it in place if necessary and solder it in.

Installing with Stacking Headers

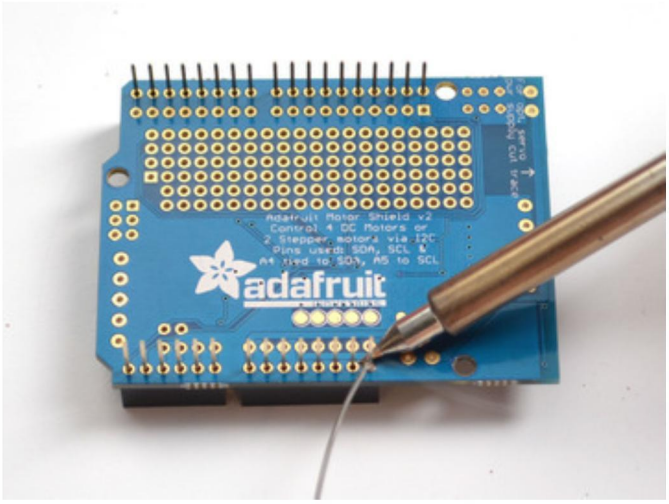


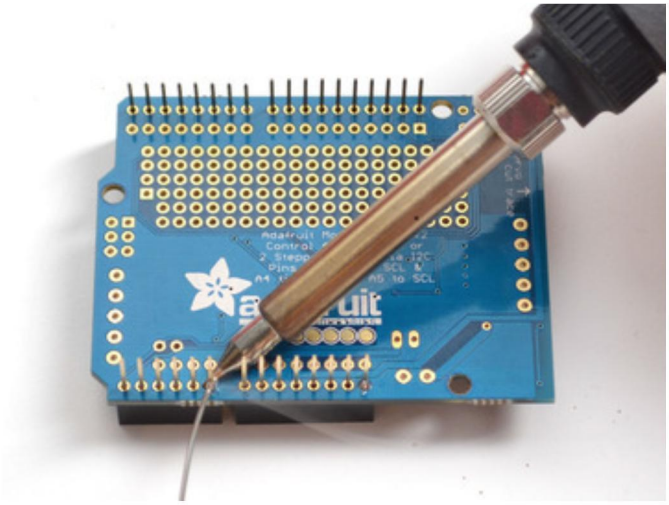
You will need to purchase Arduino stacking headers for this step, the shield does not come with them. (http://adafru.it/85)

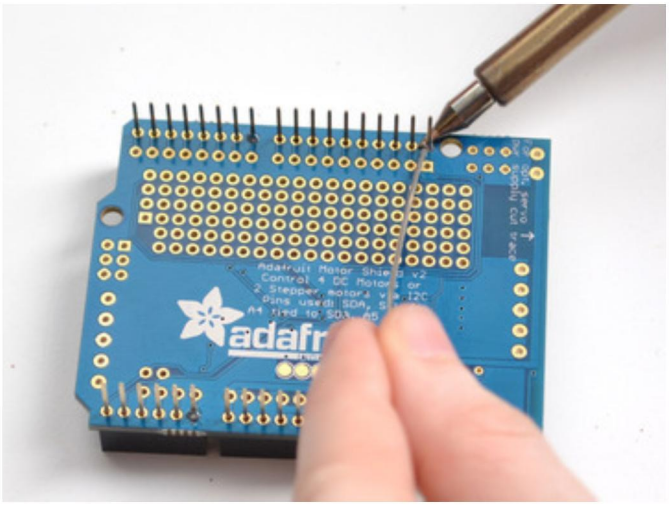
We don't show soldering in the 2x3 stacking header but you should solder that in as well - even though this shield does not use it, the one above may need those pins!



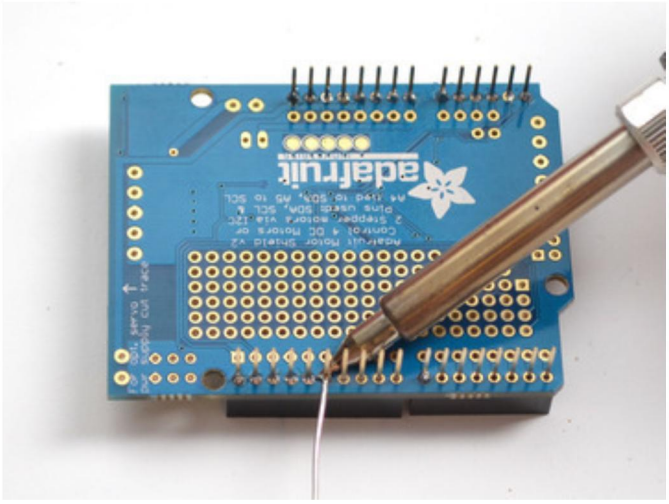
Start by sliding the 10 pin, 2 x 8 pin and 6-pin stacking headers into the outer rows of the shield from the top. Then flip the board over so its resting on the four headers. Pull on the legs if necessary to straighten them out.

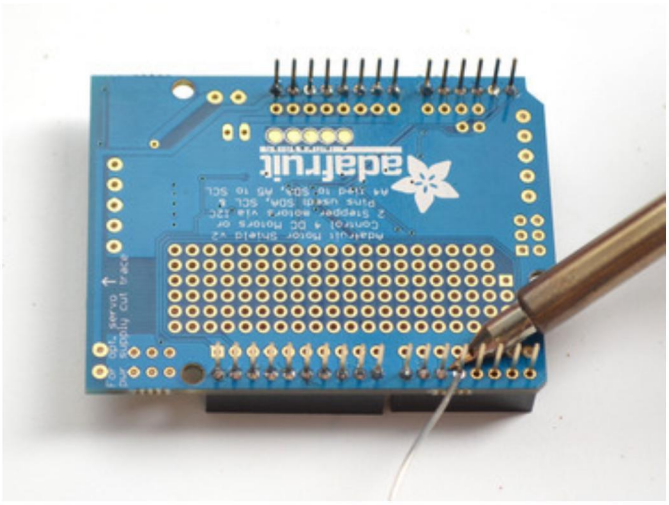






Tack one pin of each header, to get them set in place before more soldering. If the headers go crooked you can re-heat the one pin while re-positioning to straighten them up





Once you've tacked and straightened all the headers, go back and solder the remaining pins for each header.

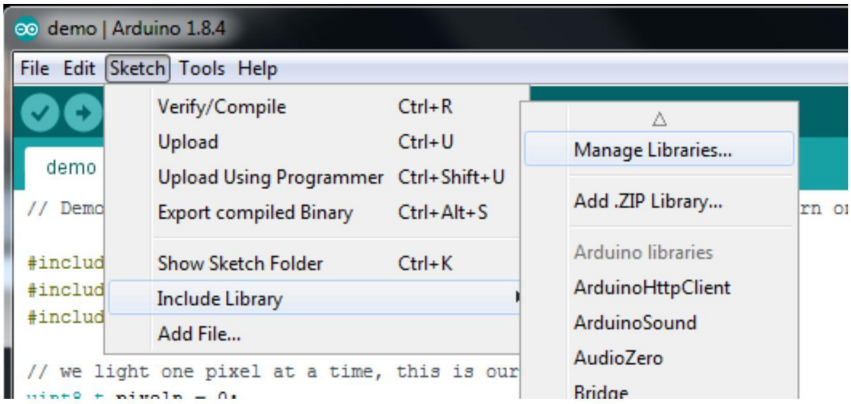
Install Software

Install Adafruit Motor Shield V2 library

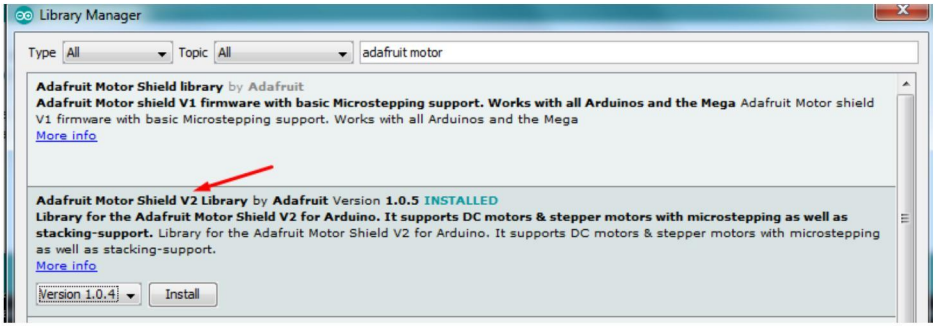
To use the shield on an Arduino, you'll need to install the Adafruit Motorshield v2 library. **This library is not compatible with the older AF_Motor library** used for v1 shields. However, if you have code for the older shield, adapting the code to use the new shield isn't difficult. We had to change the interface a little to support shield stacking, & we think its worth it!

To begin controlling motors, you will need to install the Adafruit_Motor_Shield_V2_Library library (code on our github repository) (https://adafru.it/ciN). It is available from the Arduino library manager so we recommend using that.

From the IDE open up the library manager...



And type in **adafruit motor** to locate the library. Click **Install**



If you plan to use AccelStepper for acceleration control or for simultaneous control of multiple stepper motors, you will also need to download and install the AccelStepper library:

https://adafru.it/lpB

https://adafru.it/lpB

For more details on how to install Arduino libraries, check out our detailed tutorial! (https://adafru.it/aYM)

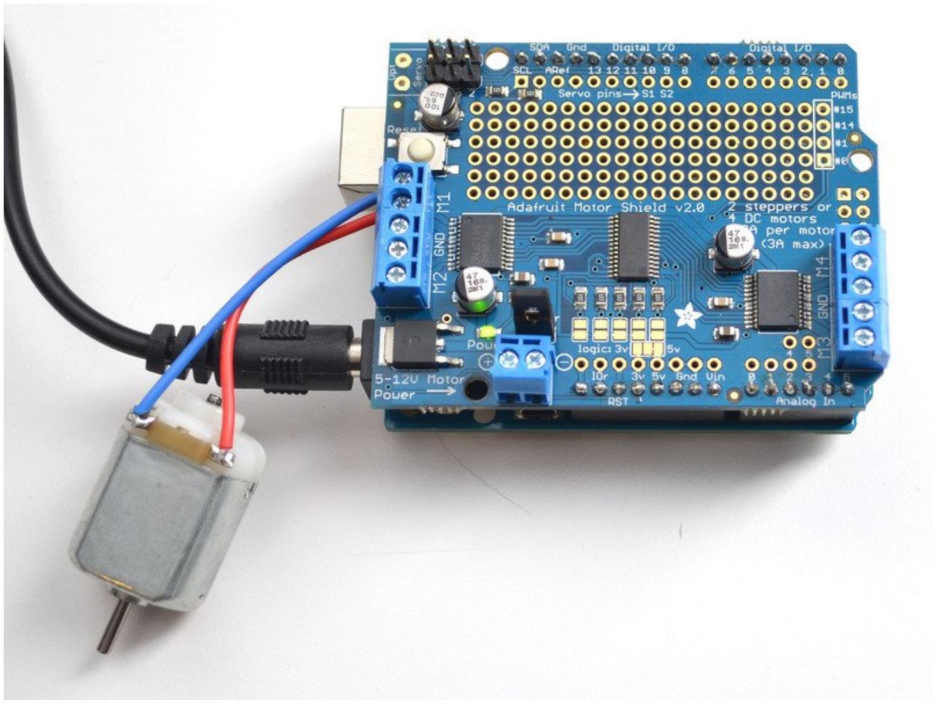
Running the Example Code

DC Motor

The library comes with a few examples to get you started up fast. We suggest getting started with the DC motor example. You can use any DC motor that can be powered by 6V-12VDC

First, restart the IDE to make sure the new library is loaded.

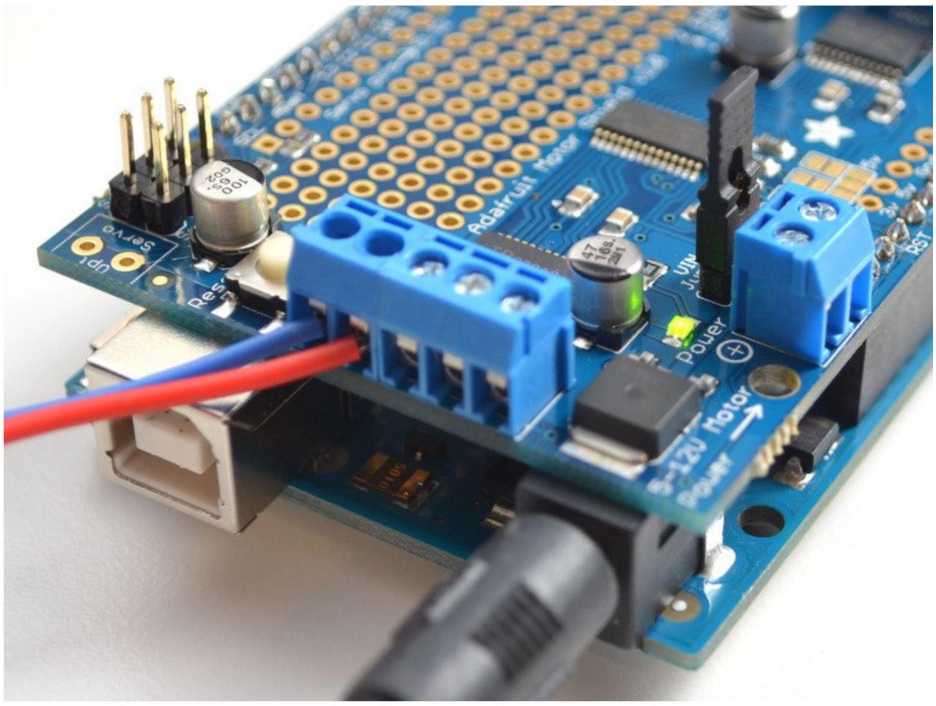
Plug the shield into the Arduino and connect a DC motor to **motor port 1** - it does not matter which wire goes into which terminal block as motors are bi-directional. Connect to the top two terminal ports, do not connect to the middle pin (GND) See the photo below for the red and blue wire example. Be sure to screw down the terminal blocks to make a good connection!



You must also supply 5-12VDC to power the motor. There are two ways to do this

1. You can power the Arduino via the **DC Barrel Jack** and **insert the VIN Jumper** shown as the tall black handle right next to the green Power LED below
2. You can power the Arduino via the DC Barrel jack **or** USB port. Then Power the shield via the 5-12VDC motor power terminal port, the double terminal block next to the green Power LED **and remove the VIN jumper**

If the Green LED next to the power terminal block isn't lit up brightly do not continue!



Once you have verified the motor is connected properly **and** you have the power LED lit up brightly, we can upload our code.

In the IDE, load **File->Examples->Adafruit_MotorShield->DCMotorTest**

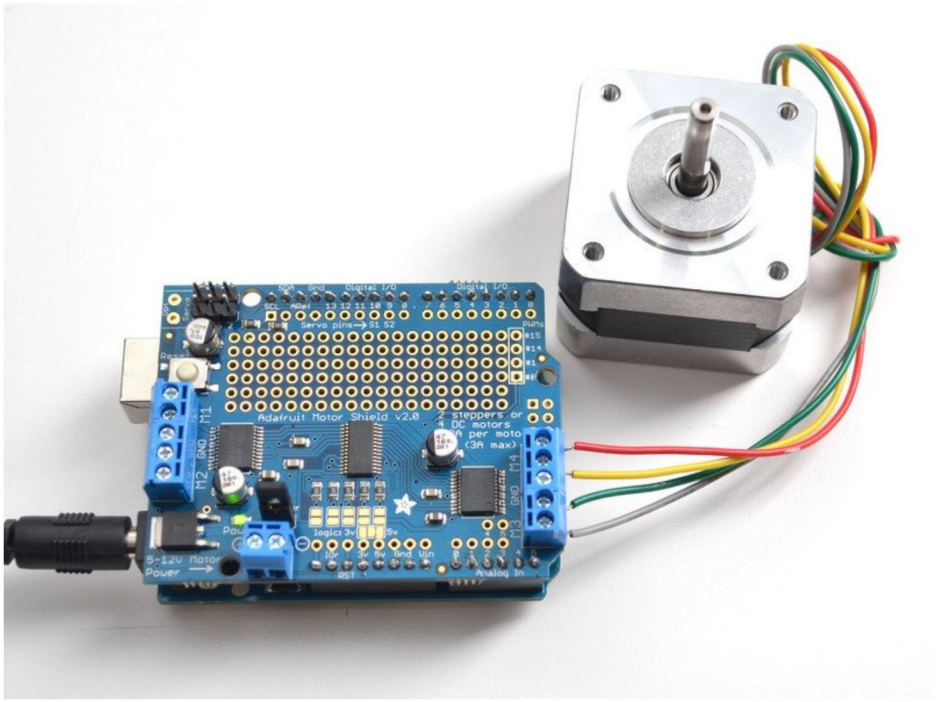
You should see and hear the DC motor turn on and move back and forth, attaching a slip of paper or tape as a 'flag' can help you visualize the movement if you have trouble seeing the movement

Stepper Motor Test

You can also test a stepper motor connection with the shield. The shield can run **unipolar** (5-wire and 6-wire) and **bipolar** (4-wire) steppers. It cannot run steppers with any other # of wires! The code is the same for unipolar or bipolar motors, the wiring is just slightly different.

Plug the shield into the Arduino and connect a stepper motor to **motor port 2** - unlike DC motors, the wire order does 'matter'. Connect to the top two terminal ports (coil #1) and the bottom two terminal ports (coil #2).

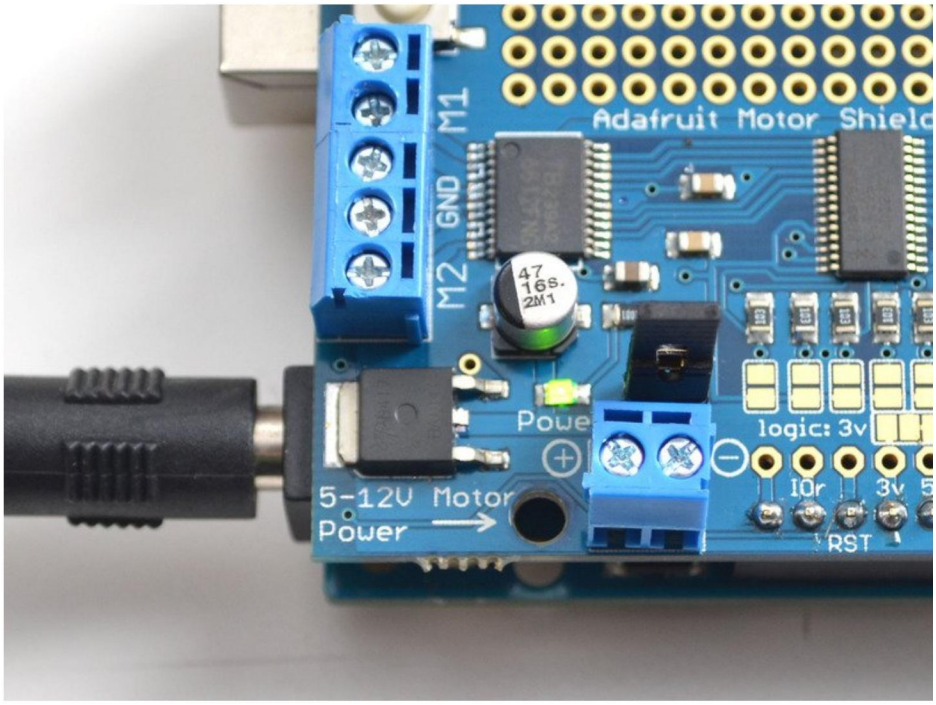
- If you have a bipolar motor, do not connect to the middle pin (GND).
- If you are using a unipolar motor with 5 wires, connect the common wire to GND.
- If you are using a unipolar motor with 6 wires, you can connect the two 'center coil wires' together to GND



You must also supply 5-12VDC to power the motor. There are two ways to do this

1. You can power the Arduino via the **DC Barrel Jack** and **insert the VIN Jumper** shown as the tall black handle right next to the green Power LED below
2. You can power the Arduino via the DC Barrel jack **or** USB port. Then Power the shield via the 5-12VDC motor power terminal port, the double terminal block next to the green Power LED **and remove the VIN jumper**

If the Green LED isn't lit up brightly **do not continue** - you must power it via the VIN jumper **or** the terminal block

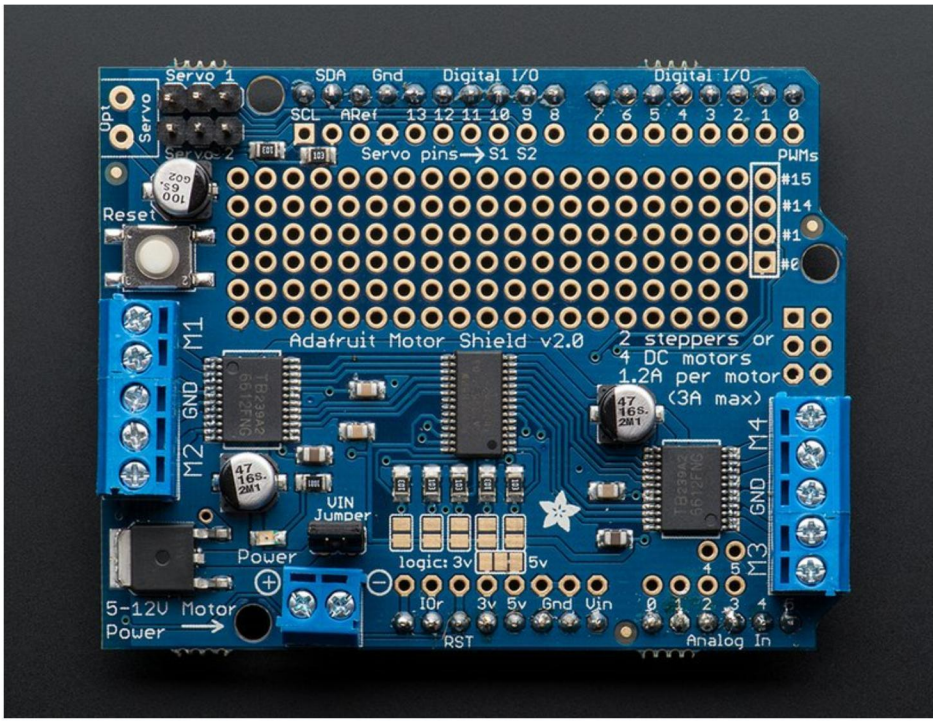


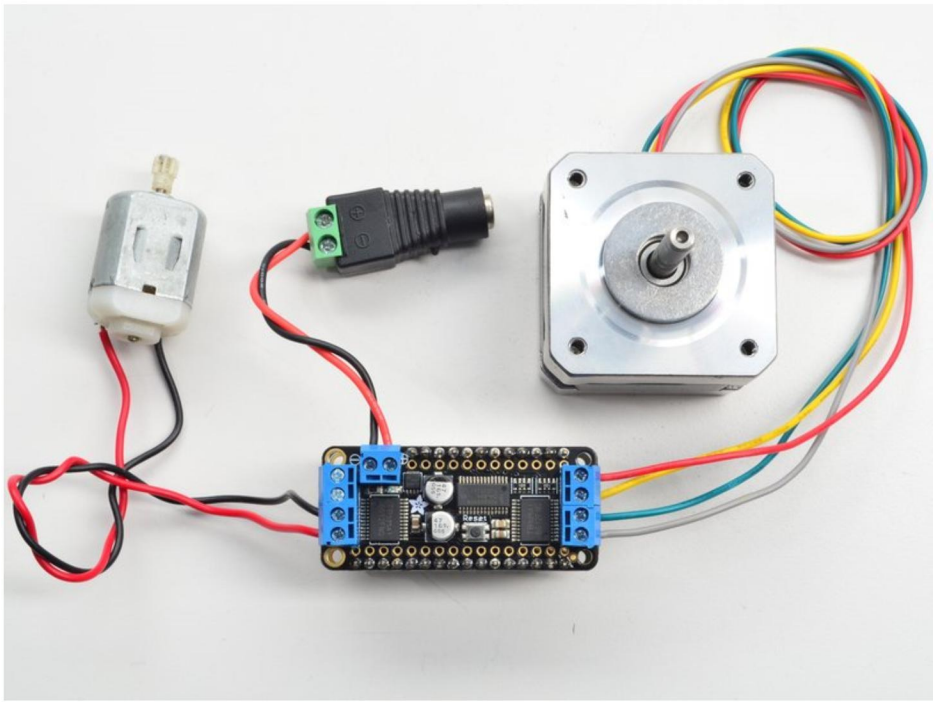
Once you have verified the motor is connected properly **and** you have the power LED lit up brightly, we can upload our

code.

In the IDE, load **File->Examples->Adafruit_MotorShield->StepperTest**

You should see and hear the stepper motor turn on and move back and forth, attaching a slip of paper or tape as a 'flag' can help you visualize the movement if you have trouble seeing the movement. There are four ways to move a stepper, with varying speed, torque and smoothness tradeoffs. This example code will demonstrate all four.





class Adafruit_MotorShield;

The Adafruit_MotorShield class represents a motor shield and must be instantiated before any DCMotors or StepperMotors can be used. You will need to declare one Adafruit_MotorShield for each shield in your system.

Adafruit_MotorShield(uint8_t addr = 0x60);

The constructor takes one optional parameter to specify the i2c address of the shield. The default address of the constructor (0x60) matches the default address of the boards as shipped. If you have more than one shield in your system, each shield must have a unique address.

void begin(uint16_t freq = 1600);

begin() must be called in setup() to initialize the shield. An optional frequency parameter can be used to specify something other than the default maximum: 1.6KHz PWM frequency.

Adafruit_DCMotor *getMotor(uint8_t n);

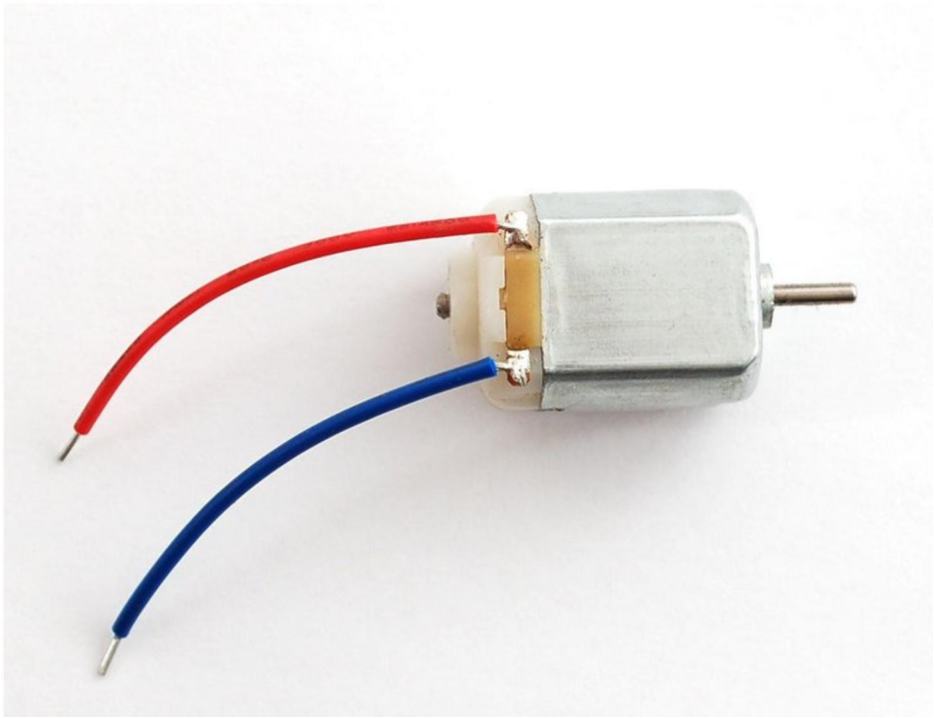
This function returns one of 4 pre-defined DC motor objects controlled by the shield. The parameter specifies the associated motor channel: 1-4.

Adafruit_StepperMotor *getStepper(uint16_t steps, uint8_t n);

This function returns one of 2 pre-defined stepper motor objects controlled by the shield.
The first parameter specifies the number of steps per revolution.
The second parameter specifies the associated stepper channel: 1-2.

void setPWM(uint8_t pin, uint16_t val);
void setPin(uint8_t pin, boolean val);

These are low-level functions to control pins on the on-board PWM driver chip. These functions are intended for internal use only.



class Adafruit_DCMotor

The Adafruit_DCMotor class represents a DC motor attached to the shield. You must declare an Adafruit_DCMotor for each motor in your system.

Adafruit_DCMotor(void);

The constructor takes no arguments. The motor object is typically initialized by assigning a motor object retrieved from the shield class as below:

```
// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);
// You can also make another motor on port M2
Adafruit_DCMotor *myOtherMotor = AFMS.getMotor(2);
```

void run(uint8_t);

The run() function controls the motor state. The parameter can have one of 3 values:

- **FORWARD** - Rotate in a forward direction
- **BACKWARD** - Rotate in the reverse direction
- **RELEASE** - Stop rotation

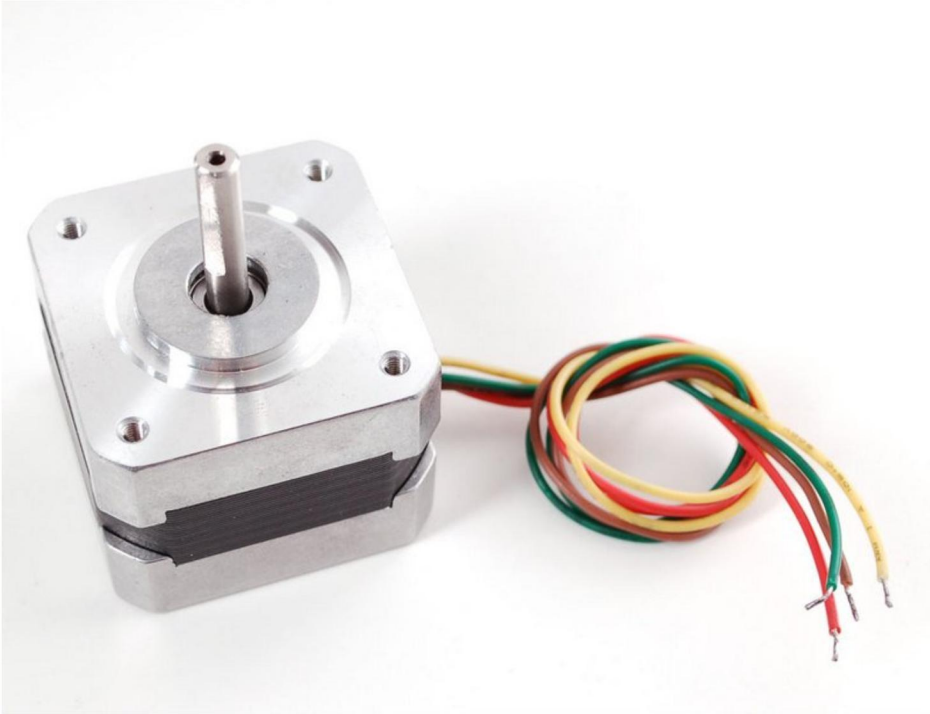
Note that the "FORWARD" and "BACKWARD" directions are arbitrary. If they do not match the actual direction of your vehicle or robot, simple swap the motor leads.

Also note that "RELEASE" simply cuts power to the motor. It does not apply any braking.

void setSpeed(uint8_t);

The setSpeed() function controls the power level delivered to the motor. The speed parameter is a value between 0 and 255.

Note that setSpeed just controls the power delivered to the motor. The actual speed of the motor will depend on several factors, including: The motor, the power supply and the load.



class Adafruit_StepperMotor

The Adafruit_StepperMotor class represents a stepper motor attached to the shield. You must declare an Adafruit_StepperMotor for each stepper motor in your system.

Adafruit_StepperMotor(void);

The constructor takes no arguments. The stepper motor is typically initialized by assigning a stepper object retrieved from the shield as below:

```
// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// to motor port #2 (M3 and M4)
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 2);
```

void step(uint16_t steps, uint8_t dir, uint8_t style = SINGLE);

The step() function controls stepper motion.

- The first parameter specifies how many steps to move.
- The second parameter specifies the direction: FORWARD or BACKWARD
- The last parameter specifies the stepping style: SINGLE, DOUBLE, INTERLEAVED or MICROSTEP

The ste() function is synchronous and does not return until all steps are complete. When complete the motor remains powered to apply "holding torque" to maintain position.

void setSpeed(uint16_t);

The setSpeed() function controls the speed of the stepper motor rotation. Speed is specified in RPM.

uint8_t onestep(uint8_t dir, uint8_t style);

The oneStep() function is a low-level internal function called by step(). But it can be useful to call on its own to

implement more advanced functions such as acceleration or coordinating simultaneous movement of multiple stepper motors. The direction and style parameters are the same as for step(), but onestep() steps exactly once.

Note: Calling step() with a step count of 1 is not the same as calling onestep(). The step function has a delay based on the speed set in setSpeed(). onestep() has no delay.

void release(void);

The release() function removes all power from the motor. Call this function to reduce power requirements if holding torque is not required to maintain position.

Arduino Library Docs

Arduino Library Docs (https://adafru.it/AvQ)

Powering Motors

Motors need a lot of energy, especially cheap motors since they're less efficient.

Voltage requirements:

The first important thing to figure out what voltage the motor is going to use. If you're lucky your motor came with some sort of specifications. Some small hobby motors are only intended to run at 1.5V, but its just as common to have 6-12V motors. The motor controllers on this shield are designed to run from **5V to 12V**.
MOST 1.5-3V MOTORS WILL NOT WORK

Current requirements:

The second thing to figure out is how much current your motor will need. The motor driver chips that come with the kit are designed to provide up to 1.2 A per motor, with 3A peak current. Note that once you head towards 2A you'll probably want to put a heat-sink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.

You can't run motors off of a 9V battery so don't waste your time/batteries!

Use a big Lead Acid or NiMH battery pack. Its also very much suggested that you set up two power supplies (split supply) one for the Arduino and one for the motors. **99% of 'weird motor problems'** are due to noise on the power line from sharing power supplies and/or not having a powerful enough supply! Even small DC motors can draw up to 3 Amps when they stall.

Setting up your shield for powering Hobby Servos

Servos are powered off of the same regulated 5V that the Arduino uses. This is OK for the small hobby servos suggested. Basically, power up your Arduino with the USB port or DC barrel jack and you're good to go. If you want something beefier, cut the trace going to the optional servo power terminal and wire up your own 5-6V supply!

Setting up your shield for powering DC and Stepper Motors

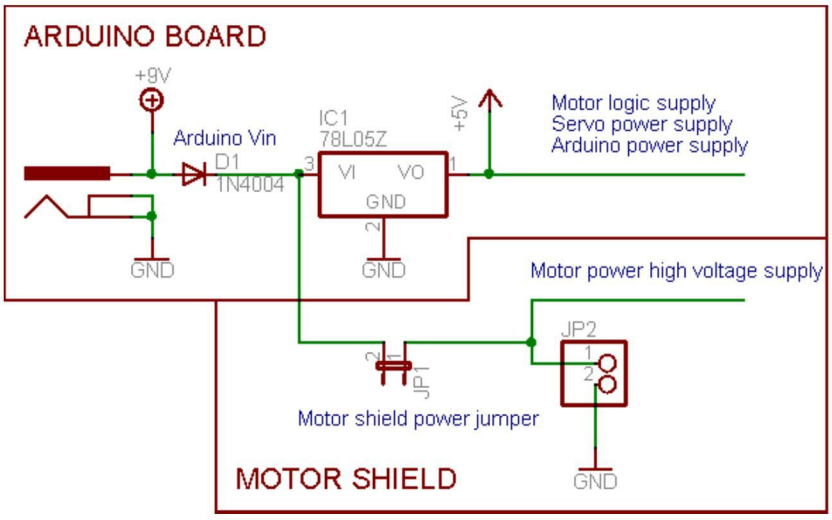
The motors are powered off of a 'high voltage supply' and NOT the regulated 5V. **Don't connect the motor power supply to the Arduino's 5V power pin**. This is a very very very bad idea unless you are sure you know what you're doing! You could damage your Arduino and/or USB port!

There are two places you can get your motor 'high voltage supply' from.

1. One is the DC barrel jack on the Arduino board
2. The other is the 2-terminal block on the shield that is labeled **DC Motor Power 5-12VDC**.

The DC Jack on the Arduino has a protection diode so you won't be able to mess things up too bad if you plug in the wrong kind of power. The terminal block has a protection FET so you will not damage the arduino/shield if you wire up your battery supply backwards, but it wont work either!

Here's how it works:



If you would like to have a **single DC power supply for the Arduino and motors**

Say a wall adapter or a single battery pack with 6-12VDC output, simply plug it into the DC jack on the Arduino or the 2-pin power terminal block on the shield. Place the power jumper on the motor shield.

Note that you may have problems with Arduino resets if the battery supply is not able to provide constant power, so it is not a suggested way of powering your motor project. You cannot use a 9V battery for this, it must be 4 to 8 AA batteries or a single/double lead acid battery pack.

If you would like to have the **Arduino powered off of USB** and the **motors powered off of a DC power supply**

Plug in the USB cable. Then connect the motor supply to the power terminal block on the shield. Do not place the jumper on the shield.

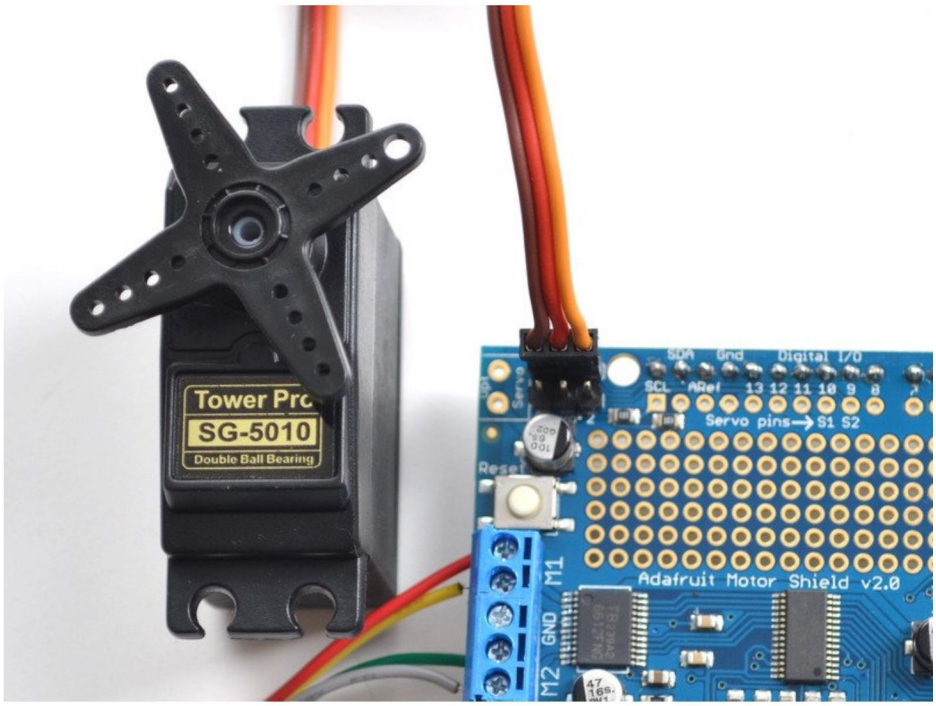
This is a suggested method of powering your motor project as it has a split supply, one power supply for logic, and one supply for motors

If you would like to have **2 separate DC power supplies for the Arduino and motors**

Plug in the supply for the Arduino into the DC jack, and connect the motor supply to the power terminal block. Make sure the jumper is removed from the motor shield.

No matter what, if you want to use the DC motor/Stepper system the motor shield LED should be lit indicating good motor power

Using RC Servos



Hobby servos are the easiest way to get going with motor control. They have a 3-pin 0.1" female header connection with +5V, ground and signal inputs. The motor shield simply brings out the PWM output lines from Arduino pins 9 and 10 to two 3-pin headers so that its easy to plug in and go. They can take a lot of power so a 9V battery wont last more than a few minutes!

The nice thing about using the onboard PWM is that its very precise and goes about its business in the background. You can use the built in **Servo** library

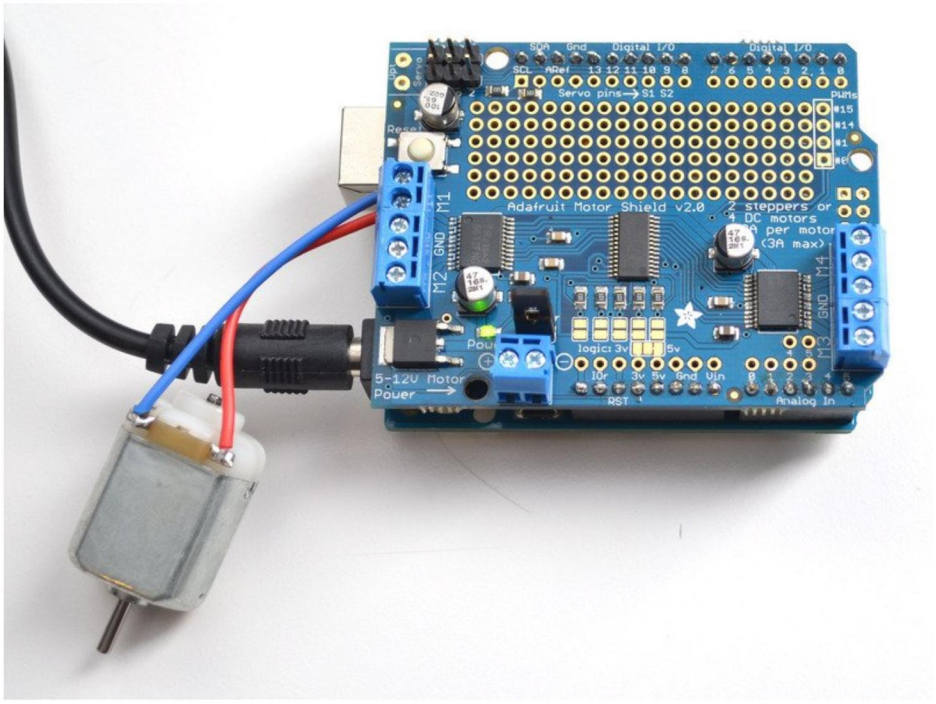
Using the servos is easy, please read the official Arduino documentation for how to use them and see the example Servo sketches in the IDE (https://adafru.it/aOD).

Powering Servos

Power for the Servos comes from the Arduino's on-board 5V regulator, powered directly from the USB or DC power jack on the Arduino. If you need an external supply, cut the 5v trace on the bottom of the board and connect a 5V or 6V DC supply directly to the **Opt Servo** power input. Using an external supply is for *advanced users* as you can accidentally destroy the servos by connecting a power supply incorrectly!

When using external servo power, be careful not to let it short out against the USB socket shell on the processor board. Insulate the top of the USB socket with some electrical tape.

Using DC Motors



DC motors are used for all sort of robotic projects.

The motor shield can drive up to 4 DC motors bi-directionally. That means they can be driven forwards and backwards. The speed can also be varied at 0.5% increments using the high-quality built in PWM. This means the speed is very smooth and won't vary!

Note that the H-bridge chip is not meant for driving continuous loads of 1.2A, so this is for small motors. Check the datasheet for information about the motor to verify its OK!

Connecting DC Motors

To connect a motor, simply solder two wires to the terminals and then connect them to either the M1, M2, M3, or M4. Then follow these steps in your sketch

Include the required libraries

Make sure you **#include** the required libraries

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_MS_PWMServoDriver.h"
```

Create the Adafruit_MotorShield object

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
```

Create the DC motor object

Request the DC motor from the Adafruit_MotorShield:

```
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);
```

with **getMotor(port#)**. ***Port#*** is which port it is connected to. If you're using M1 its**1**, M2 use **2**, M3 use **3** and M4 use **4**

Connect to the Controller

In your `setup()` function, call `begin()` on the Adafruit_MotorShield object:

```
AFMS.begin();
```

Set default speed

Set the speed of the motor using **setSpeed(*speed*)** where the ***speed*** ranges from 0 (stopped) to 255 (full speed). You can set the speed whenever you want.

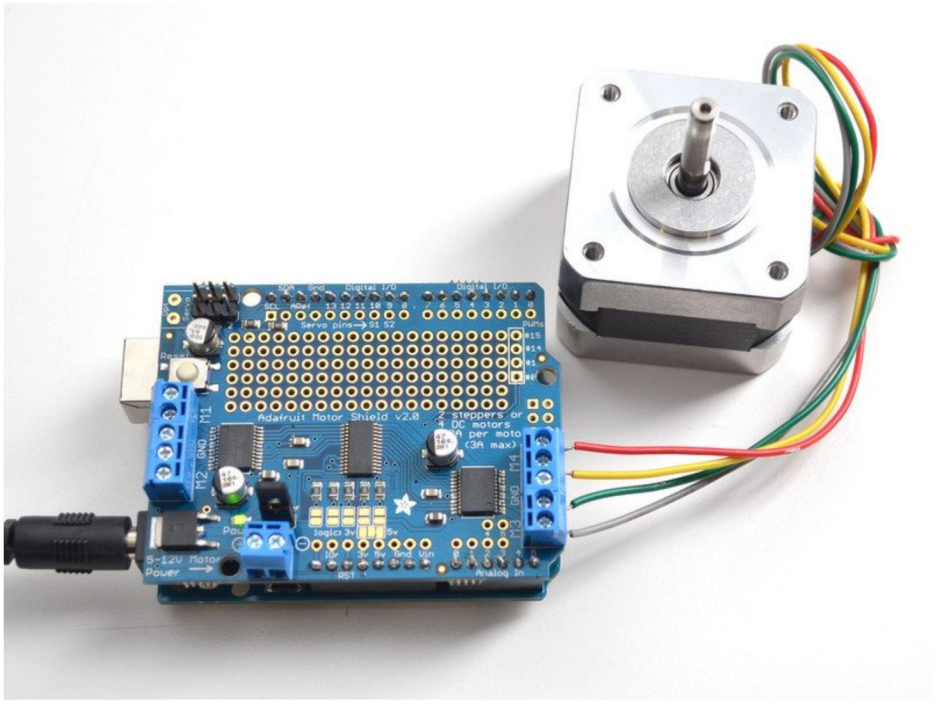
```
myMotor->setSpeed(150);
```

Run the motor

To run the motor, call **run(*direction*)** where ***direction*** is **FORWARD**, **BACKWARD** or **RELEASE**. Of course, the Arduino doesn't actually know if the motor is 'forward' or 'backward', so if you want to change which way it thinks is forward, simply swap the two wires from the motor to the shield.

```
myMotor->run(FORWARD);
```

Using Stepper Motors



Stepper motors are great for (semi-)precise control, perfect for many robot and CNC projects. This motor shield supports up to 2 stepper motors. The library works identically for bi-polar and uni-polar motors

Before connecting a motor, be sure to check the motor specifications for compatibility with the shield (https://adafru.it/r2d).

For unipolar motors: to connect up the stepper, first figure out which pins connected to which coil, and which pins are the center taps. If its a 5-wire motor then there will be 1 that is the center tap for both coils. Theres plenty of tutorials online on how to reverse engineer the coils pinout. (https://adafru.it/aOO) The center taps should both be connected together to the GND terminal on the motor shield output block. then coil 1 should connect to one motor port (say M1 or M3) and coil 2 should connect to the other motor port (M2 or M4).

For bipolar motors: its just like unipolar motors except theres no 5th wire to connect to ground. The code is exactly the same.

Running a stepper is a little more intricate than running a DC motor but its still very easy

Include the required libraries

Make sure you **#include** the required libraries

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_PWMServoDriver.h"
```

Create the Adafruit_MotorShield object

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
```

Create the stepper motor object

Request the **Stepper** motor from the **Adafruit_MotorShield**:

```
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 2);
```

...with `getStepper(steps, stepper#)`.

`Steps` indicates how many steps per revolution the motor has. A 7.5 degree/step motor has 360/7.5 = 48 steps.

`Stepper#` is which port it is connected to. If you're using M1 and M2, its port**1**. If you're using M3 and M4 indicate port **2**

Set default speed

Set the speed of the motor using `setSpeed(rpm)` where rpm is how many revolutions per minute you want the stepper to turn.

Run the motor

Then every time you want the motor to move, call the `step(#steps, direction, steptype)` procedure. **#steps** is how many steps you'd like it to take. direction is either **FORWARD** or **BACKWARD** and the step type is **SINGLE**, **DOUBLE**, **INTERLEAVE** or **MICROSTEP**.

- "Single" means single-coil activation
- "Double" means 2 coils are activated at once (for higher torque)
- "Interleave" means that it alternates between single and double to get twice the resolution (but of course its half the speed).
- "Microstepping" is a method where the coils are PWM'd to create smooth motion between steps.

Theres tons of information about the pros and cons of these different stepping methods in the resources page.

You can use whichever stepping method you want, changing it "on the fly" to as you may want minimum power, more torque, or more precision.

By default, the motor will 'hold' the position after its done stepping. If you want to release all the coils, so that it can spin freely, call `release()`

The stepping commands are 'blocking' and will return once the steps have finished.

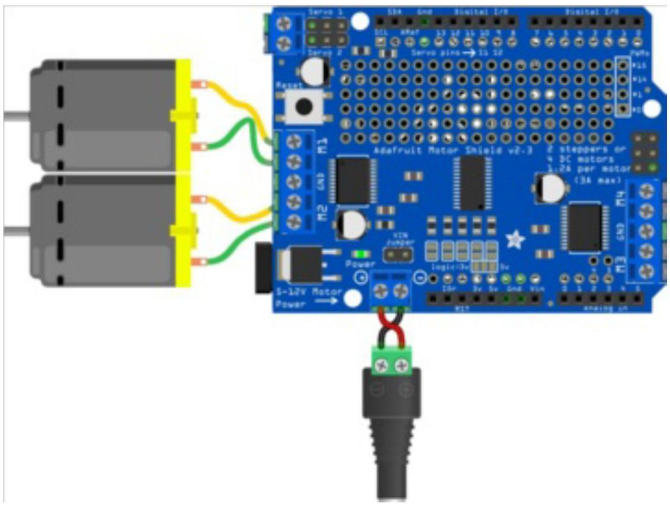
Because the stepping commands 'block' - you have to instruct the Stepper motors each time you want them to move. If you want to have more of a 'background task' stepper control, check out AccelStepper library (https://adafru.it/aOL) (install similarly to how you did with **Adafruit_MotorShield**) which has some examples for controlling three steppers simultaneously with varying acceleration

Python & CircuitPython

We've written a handy CircuitPython library for the various DC Motor and Stepper kits called Adafruit CircuitPython MotorKit (https://adafru.it/DdU) that handles all the complicated setup for you. All you need to do is import the appropriate class from the library, and then all the features of that class are available for use. We're going to show you how to import the `MotorKit` class and use it to control DC and stepper motors with the Adafruit Stepper + DC Motor Shield.

CircuitPython Microcontroller Wiring

First assemble the Shield exactly as shown in the previous pages. There's no wiring needed to connect the Shield to the Metro. The example below shows wiring two DC motors to the Shield once it has been attached to a Metro. You'll want to connect a barrel jack to the power terminal to attach an appropriate external power source to the Shield. **The Shield will not function without an external power source!**



- Connect the **two motor wires from the first motor** to **the M1 terminal on the Shield**.
- Connect the **two motor wires from the second motor** to the **M2 terminal on the Shield**.
- Connect the **positive side of the power terminal** to the **positive side of the barrel jack**.
- Connect the **negative side of the power terminal** to the **negative side of the barrel jack**.

CircuitPython Installation of MotorKit and Necessary Libraries

You'll need to install a few libraries on your Metro board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/Amd) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/uap). Our CircuitPython starter guide has a great page on how to install the library bundle (https://adafru.it/ABU).

If you choose, you can manually install the libraries individually on your board:

- **adafruit_pca9685**
- **adafruit_bus_device**
- **adafruit_register**
- **adafruit_motor**
- **adafruit_motorkit**

Before continuing make sure your board's lib folder or root filesystem has the **adafruit_pca9685.mpy, adafruit_register, adafruit_motor, adafruit_bus_device** and **adafruit_motorkit** files and folders copied over.

Next connect to the board's serial REPL (https://adafru.it/Awz) so you are at the CircuitPython >>> prompt.

CircuitPython Usage

To demonstrate the usage, we'll initialise the library and use Python code to control DC and stepper motors from the board's Python REPL.

First you'll need to import and initialize the MotorKit class.

```
from adafruit_motorkit import MotorKit
kit = MotorKit()
```

DC Motors

The four motor spots on the Shield are available as `motor1`, `motor2`, `motor3`, and `motor4`.

In this example we'll use `motor1`.

Note: For small DC motors like sold in the shop you might run into problems with electrical noise they generate and erratic behavior on your board. If you see erratic behavior like the motor not spinning or the board resetting at high motor speeds this is likely the problem. See this motor guide FAQ page for information on capacitors you can solder to the motor to reduce noise (https://adafru.it/scl).

Now to move a motor you can set the `throttle` attribute. We don't call it speed because it doesn't correlate to a particular number of revolutions per minute (RPM). RPM depends on the motor and the voltage which is unknown.

For example to drive motor M1 forward at a full speed you set it to `1.0`:

```
kit.motor1.throttle = 1.0
```

To run the motor at half throttle forward use a decimal:

```
kit.motor1.throttle = 0.5
```

Or to reverse the direction use a negative throttle:

```
kit.motor1.throttle = -0.5
```

You can stop the motor with a throttle of `0`:

```
kit.motor1.throttle = 0
```

To let the motor coast and then spin freely set throttle to `None`.

```
kit.motor1.throttle = None
```

That's all there is to controlling DC motors with CircuitPython! With DC motors you can build fun moving projects like robots or remote controlled cars that glide around with ease.

Stepper Motors

Similar DC motors, stepper motors are available as `stepper1` and `stepper2`. `stepper1` is made up of the M1 and M2 terminals, and `stepper2` is made up of the M3 and M4 terminals.

We'll use `stepper1` in our example.

The most basic function (and the default) is to do one single coil step.

```
kit.stepper1.onestep()
```

You can also call the onestep function with two optional keyword arguments. To use these, you'll need to import stepper as well.

```
from adafruit_motor import stepper
```

Then you have access to the following options:

- `direction`, which should be one of the following constant values:
  - `stepper.FORWARD` (default)
  - `stepper.BACKWARD`.
- `style`, which should be one of the values:
  - `stepper.SINGLE` (default) for a full step rotation to a position where one single coil is powered
  - `stepper.DOUBLE` for a full step rotation to position where two coils are powered providing more torque
  - `stepper.INTERLEAVED` for a half step rotation interleaving single and double coil positions and torque
  - `stepper.MICROSTEP` for a microstep rotation to a position where two coils are partially active.
- `release()` which releases all the coils so the motor can free spin, and also won't use any power

The function returns the current step 'position' in microsteps which can be handy to understand how far the stepper has moved, or you can ignore the result.

To take a double-coil step backward call:

```
kit.stepper1.onestep(direction=stepper.BACKWARD, style=stepper.DOUBLE)
```

You can even use a loop to continuously call `onestep` and move the stepper, for example a loop of `200` microsteps forward for smooth movement:

```
for i in range(200):
    kit.stepper1.onestep(style=stepper.MICROSTEP)
```

That's all there is to controlling a stepper motor from CircuitPython! Steppers are handy motors for when you need smooth or precise control of something--for example 3D printers and CNC machines use steppers to precisely move tools around surfaces.

Full Example Code

For DC motors:

```
"""Simple test for using adafruit_motorkit with a DC motor"""
import time
from adafruit_motorkit import MotorKit

kit = MotorKit()

kit.motor1.throttle = 1.0
time.sleep(0.5)
kit.motor1.throttle = 0
```

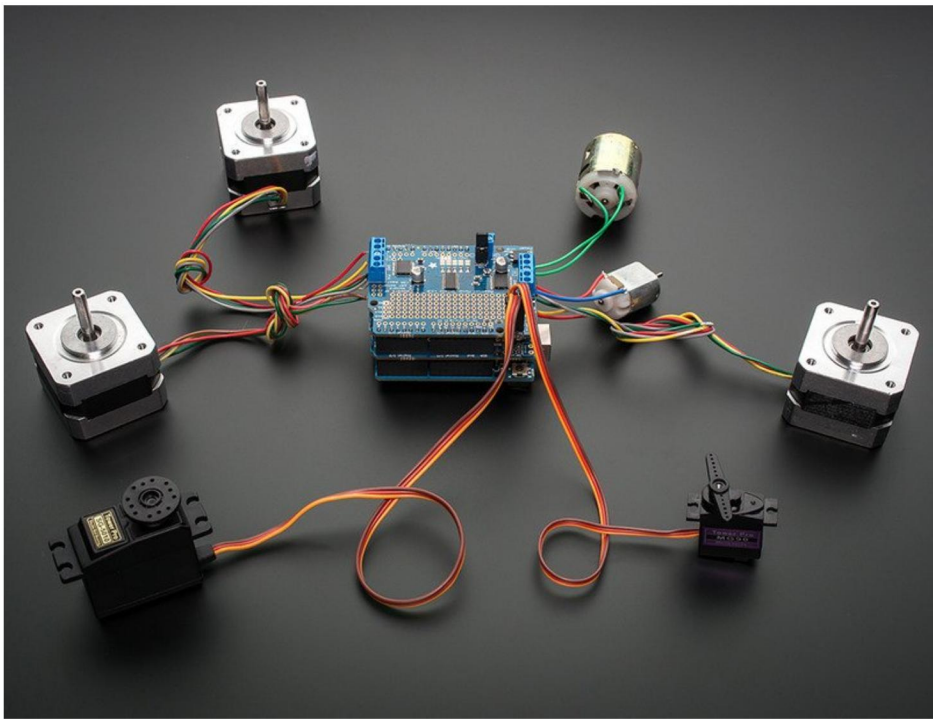
For stepper motors:

```
"""Simple test for using adafruit_motorkit with a stepper motor"""
import time
from adafruit_motorkit import MotorKit

kit = MotorKit()

for i in range(100):
    kit.stepper1.onestep()
    time.sleep(0.01)
```

Python Docs

Python Docs (https://adafru.it/DeE)

Stacking Shields



One of the cool things about this shield design is that it is possible to stack shields. Every shield you stack can control another 2 steppers or 4 DC motors (or a mix of the two)

You can stack up to 32 shields for a total of 64 steppers or 128 DC motors! Most people will probably just stack two or maybe three but hey, you never know. (PS if you drive 64 steppers from one of these shields send us a photo, OK?)

Note that stacking shields does not increase the servo connections - those are hard-wired to the Arduino digital 9 & 10 pins. If you need to control a lot of servos, you can use our 16-channel servo shield and stack it with this shield to add a crazy large # of servos. (http://adafru.it/1411)

Stacking shields is very easy. Each shield you want to stack on top of must have stacking headers installed. Check our instructions for how to do so. (https://adafru.it/ciP) The top shield does not have to have stacking headers unless you eventually want to put something on top of it.

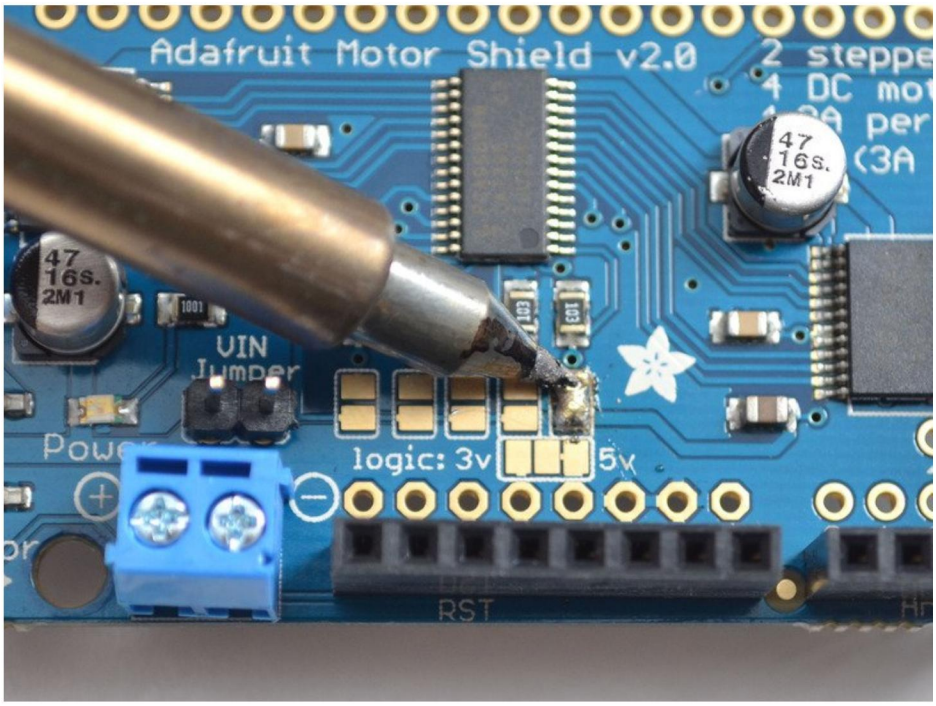
The only thing to watch for when stacking shields is every shield must have a unique I2C address. The default address is **0x60**. You can adjust the address of the shields to range from 0x60 to 0x7F for a total of 32 unique addresses.

Addressing the Shields

Each board in the chain must be assigned a unique address. This is done with the address jumpers on the lower edge of the board. The I2C base address for each board is 0x60. The binary address that you program with the address jumpers is added to the base I2C address.

To program the address offset, use a drop of solder to bridge the corresponding address jumper for each binary '1' in the address.

The right-most jumper is address bit #0, then to the left of that is address bit #1, etc up to address bit #4



Board 0: Address = 0x60 Offset = binary 0000 (no jumpers required)
Board 1: Address = 0x61 Offset = binary 0001 (bridge A0 as in the photo above)
Board 2: Address = 0x62 Offset = binary 0010 (bridge A1, to the left of A0)
Board 3: Address = 0x63 Offset = binary 0011 (bridge A0 & A1, two rightmost jumpers)
Board 4: Address = 0x64 Offset = binary 0100 (bridge A2, middle jumper)

etc.

Note that address 0x70 is the "all call" address for the controller chip on the shield. All boards will respond to address 0x70 - regardless of the address jumper settings.

Writing Code for Multiple Shields

The Adafruit_MotorShield library has the ability to control multiple shields, unlike the older AF_Motor library. First we must create a Motor Shield Controller for each shield, with the assigned address.

```
Adafruit_MotorShield AFMSbot(0x61); // Rightmost jumper closed
Adafruit_MotorShield AFMStop(0x60); // Default address, no jumpers
```

One motor shield is going to be called AFMSbot (bottom shield, so we remember) and one is AFMStop (top shield) so we can keep them apart. When you create the shield object, specify the address you set for it above.

Then we can request the motors connected to each one

```
// On the top shield, connect two steppers, each with 200 steps
Adafruit_StepperMotor *myStepper2 = AFMStop.getStepper(200, 1);
```

```
Adafruit_StepperMotor *myStepper3 = AFMStop.getStepper(200, 2);

// On the bottom shield connect a stepper to port M3/M4 with 200 steps
Adafruit_StepperMotor *myStepper1 = AFMSbot.getStepper(200, 2);
// And a DC Motor to port M1
Adafruit_DCMotor *myMotor1 = AFMSbot.getMotor(1);
```

You can request a stepper or DC motor from any port, just be sure to use the right AFMS controller object when you call **getMotor** or **getStepper**!

Then, both shields must have **begin** called, before you use the motors connected

```
AFMSbot.begin(); // Start the bottom shield
AFMStop.begin(); // Start the top shield
```

You can try out this code for yourself by setting up two shields and running the **File->Examples->Adafruit_MotorShield->StackingTest** example

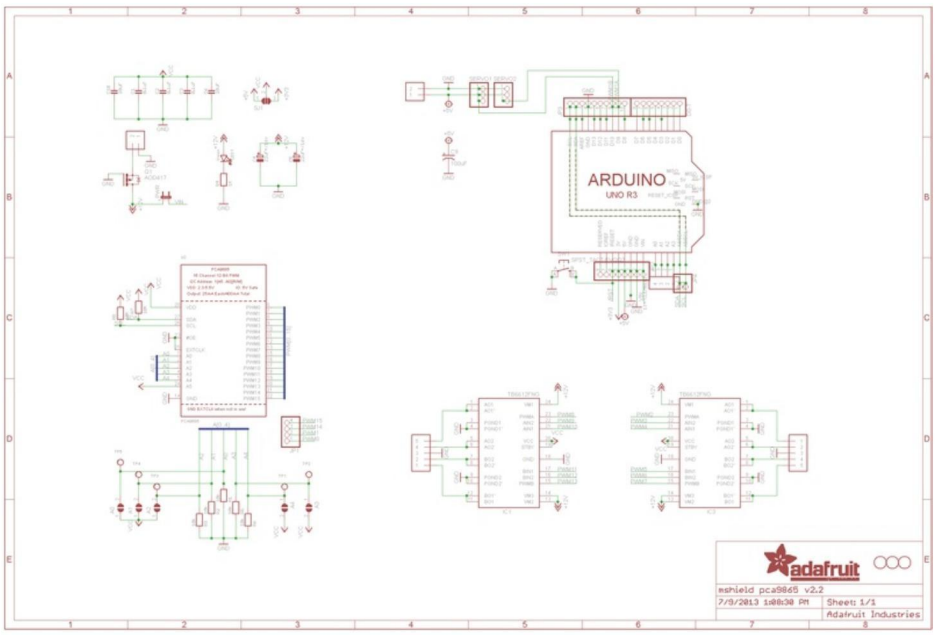
Resources

Motor ideas and tutorials

- Wikipedia has tons of information (https://adafru.it/aOF) on steppers
- Jones on stepper motor types (https://adafru.it/aOH)
- Jason on reverse engineering the stepper wire pinouts (https://adafru.it/aOI)

PCB files are on GitHub (https://adafru.it/DeF)

Schematic, click to embiggen

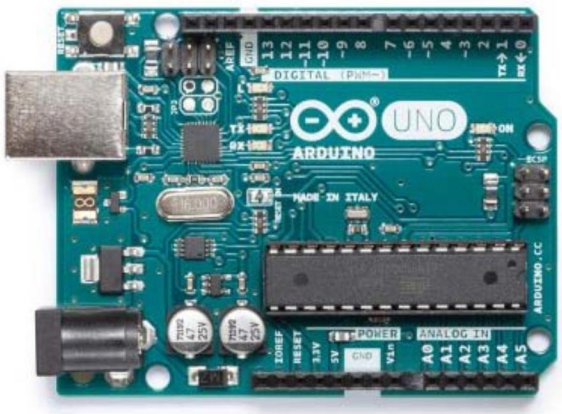


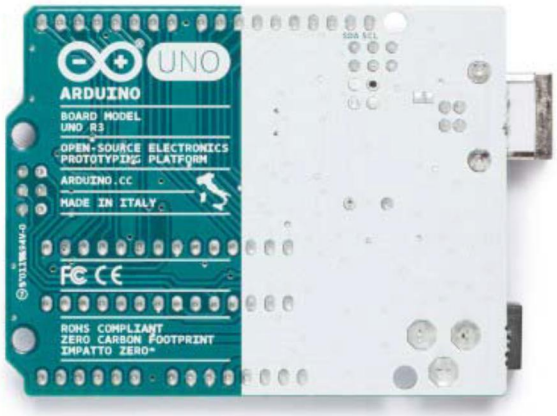
Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Adafruit:
171 1438





ARDUINO UNO REV3

Code: A000066

The UNO is the best board to get started with electronics and coding. If this is your first experience tinkering with the platform, the UNO is the most robust board you can start playing with. The UNO is the most used and documented board of the whole Arduino family.

- **Arduino Uno** is a microcontroller board based on the ATmega328P (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.. You can tinker with your UNO without worring too much about doing something wrong, worst case scenario you can replace the chip for a few dollars and start over again.

  "Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The Uno board and version 1.0 of Arduino Software (IDE) were the reference versions of Arduino, now evolved to newer releases. The Uno board is the first in a series of USB Arduino boards, and the reference model for the Arduino platform; for an extensive list of current, past or outdated boards see the Arduino index of boards.

  You can find here your board warranty informations.
  https://www.arduino.cc/en/Main/warranty

Getting Started

You can find in the Getting Started section all the information you need to configure your board, use the Arduino Software (IDE), and start tinker with coding and electronics.
https://www.arduino.cc/en/Guide/HomePage

TECH SPECS

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA

Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
LED_BUILTIN	13
Length	68.6 mm
Width	53.4 mm
Weight	25 g

- ## OSH: Schematics

Arduino Uno is open-source hardware! You can build your own board using the following files:

EAGLE FILES IN .ZIP
https://www.arduino.cc/en/uploads/Main/arduino_Uno_Rev3-02-TH.zip
SCHEMATICS IN .PDF
https://www.arduino.cc/en/uploads/Main/Arduino_Uno_Rev3-schematic.pdf
BOARD SIZE IN .DXF
http://arduino.cc/documents/ArduinoUno.dxf

Programming

The Arduino Uno can be programmed with the (Arduino Software (IDE)). Select "Arduino/Genuino Uno from the Tools > Board menu (according to the microcontroller on your board). For details, see the reference and tutorials.

The ATmega328 on the Arduino Uno comes preprogrammed with a bootloader that allows you to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files).

You can also bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header using Arduino ISP or similar; see these instructions for details.

The ATmega16U2 (or 8U2 in the rev1 and rev2 boards) firmware source code is available in the Arduino repository. The ATmega16U2/8U2 is loaded with a DFU bootloader, which can be activated by:

- On Rev1 boards: connecting the solder jumper on the back of the board (near the map of Italy) and then rese ing the 8U2.
- On Rev2 or later boards: there is a resistor that pulling the 8U2/16U2 HWB line to ground, making it easier to put into DFU mode.

You can then use Atmel's FLIP software (Windows) or the DFU programmer (Mac OS X and Linux) to load a new firmware. Or you can use the ISP header with an external programmer (overwriting the DFU bootloader). See this user-contributed tutorial for more information.

Warnings

The Arduino Uno has a resettable polyfuse that protects your computer's USB ports from shorts and overcurrent. Although most computers provide their own internal protection, the fuse provides an extra layer of protection. If more than 500 mA is applied to the USB port, the fuse will automatically break the connection until the short or overload is removed.

Differences with other boards

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converter.

Power

The Arduino Uno board can be powered via the USB connection or with an external power supply. The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the GND and Vin pin headers of the POWER connector.

The board can operate on an external supply from 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may become unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

The power pins are as follows:

- Vin. The input voltage to the Arduino/Genuino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). You can supply voltage through this pin, or, if supplying voltage via the power jack, access it through this pin.

- 5V.This pin outputs a regulated 5V from the regulator on the board. The board can be supplied with power either from the DC power jack (7 - 12V), the USB connector (5V), or the VIN pin of the board (7-12V). Supplying voltage via the 5V or 3.3V pins bypasses the regulator, and can damage your board. We don't advise it.

- 3V3. A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.

- GND. Ground pins.

- IOREF. This pin on the Arduino/Genuino board provides the voltage reference with which the microcontroller operates. A properly configured shield can read the IOREF pin voltage and select the appropriate power source or enable voltage translators on the outputs to work with the 5V or 3.3V.

Memory

The ATmega328 has 32 KB (with 0.5 KB occupied by the bootloader). It also has 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).
https://www.arduino.cc/en/Reference/EEPROM

Input and Output

See the mapping between Arduino pins and ATmega328P ports. The mapping for the Atmega8, 168, and 328 is identical.

PIN MAPPING ATmega328P
https://www.arduino.cc/en/Hacking/PinMapping168

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode() https://www.arduino.cc/en/Reference/PinMode, digitalWrite() https://www.arduino.cc/en/Reference/DigitalWrite, and digitalRead() https://www.arduino.cc/en/Reference/DigitalRead functions. They operate at 5 volts. Each pin can provide or receive 20 mA as recommended operating condition and has an internal pull-up resistor (disconnected by default) of 20-50k ohm. A maximum of 40mA is the value that must not be exceeded on any I/O pin to avoid permanent damage to the microcontroller.

In addition, some pins have specialized functions:

- Serial: 0 (RX) and 1 (TX). Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.
- External Interrupts: 2 and 3. These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value. See the attachInterrupt() function for details.
- PWM: 3, 5, 6, 9, 10, and 11. Provide 8-bit PWM output with the analogWrite() function.
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). These pins support SPI communication using the SPI library.
- LED: 13. There is a built-in LED driven by digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.
- TWI: A4 or SDA pin and A5 or SCL pin. Support TWI communication using the Wire library.

The Uno has 6 analog inputs, labeled A0 through A5, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and the analogReference() function. There are a couple of other pins on the board:

- AREF. Reference voltage for the analog inputs. Used with analogReference().
- Reset. Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

Communication

Arduino/Genuino Uno has a number of facilities for communicating with a computer, another Arduino/Genuino board, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega16U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The 16U2 firmware uses the standard USB COM drivers, and no external driver is needed. However, on Windows, a .inf file is required. The Arduino Software (IDE) includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-to-serial chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

A SoftwareSerial library allows serial communication on any of the Uno's digital pins.
https://www.arduino.cc/en/Reference/SoftwareSerial

The ATmega328 also supports I2C (TWI) and SPI communication. The Arduino Software (IDE) includes a Wire library to simplify use of the I2C bus; see the documentation for details. For SPI communication, use the SPI library.
https://www.arduino.cc/en/Reference/Wire

https://www.arduino.cc/en/Reference/SPI

Automatic (Software) Reset

Rather than requiring a physical press of the reset button before an upload, the Arduino/Genuino Uno board is designed in a way that allows it to be reset by software running on a connected computer. One of the hardware flow control lines (DTR) of the ATmega8U2/16U2 is connected to the reset line of the ATmega328 via a 100 nanofarad capacitor. When this line is asserted (taken low), the reset line drops long enough to reset the chip. The Arduino Software (IDE) uses this capability to allow you to upload code by simply pressing the upload button in the interface toolbar. This means that the bootloader can have a shorter timeout, as the lowering of DTR can be well-coordinated with the start of the upload.

This setup has other implications. When the Uno is connected to either a computer running Mac OS X or Linux, it resets each time a connection is made to it from software (via USB). For the following half-second or so, the bootloader is running on the Uno. While it is programmed to ignore malformed data (i.e. anything besides an upload of new code), it will intercept the first few bytes of data sent to the board after a connection is opened. If a sketch running on the board receives one-time configuration or other data when it first starts, make sure that the software with which it communicates waits a second after opening the connection and before sending this data.

The Uno board contains a trace that can be cut to disable the auto-reset. The pads on either side of the trace can be soldered together to re-enable it. It's labeled "RESET-EN". You may also be able to disable the auto-reset by connecting a 110 ohm resistor from 5V to the reset line; see this forum thread for details.

Revisions

Revision 3 of the board has the following new features:

- 1.0 pinout: added SDA and SCL pins that are near to the AREF pin and two other new pins placed near to the RESET pin, the IOREF that allow the shields to adapt to the voltage

provided from the board. In future, shields will be compatible with both the board that uses the AVR, which operates with 5V and with the Arduino Due that operates with 3.3V. The second one is a not connected pin, that is reserved for future purposes.

- Stronger RESET circuit.
- Atmega 16U2 replace the 8U2.

https://store.arduino.cc/usa/arduino-uno-rev3 12-8-17

FAIRCHILD SEMICONDUCTOR

October 2005

BSS138

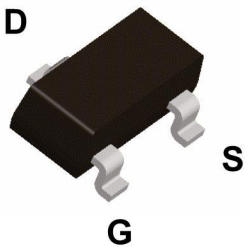
N-Channel Logic Level Enhancement Mode Field Effect Transistor

General Description

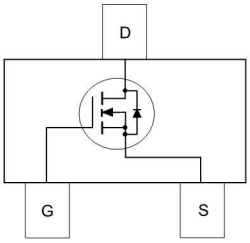
These N-Channel enhancement mode field effect transistors are produced using Fairchild's proprietary, high cell density, DMOS technology. These products have been designed to minimize on-state resistance while provide rugged, reliable, and fast switching performance.These products are particularly suited for low voltage, low current applications such as small servo motor control, power MOSFET gate drivers, and other switching applications.

Features

- 0.22 A, 50 V. $R_{DS(ON)}$ = 3.5Ω @ V_{GS} = 10 V
  $R_{DS(ON)}$ = 6.0Ω @ V_{GS} = 4.5 V
- High density cell design for extremely low $R_{DS(ON)}$
- Rugged and Reliable
- Compact industry standard SOT-23 surface mount package



SOT-23



Absolute Maximum Ratings

T_A=25°C unless otherwise noted

Symbol	Parameter		Ratings	Units
V_{DSS}	Drain-Source Voltage		50	V
V_{GSS}	Gate-Source Voltage		±20	V
I_D	Drain Current – Continuous	(Note 1)	0.22	A
	– Pulsed		0.88	
P_D	Maximum Power Dissipation	(Note 1)	0.36	W
	Derate Above 25°C		2.8	mW/°C
T_J, T_{STG}	Operating and Storage Junction Temperature Range		−55 to +150	°C
T_L	Maximum Lead Temperature for Soldering Purposes, 1/16" from Case for 10 Seconds		300	°C

Thermal Characteristics

$R_{\theta JA}$	Thermal Resistance, Junction-to-Ambient	(Note 1)	350	°C/W

Package Marking and Ordering Information

Device Marking	Device	Reel Size	Tape width	Quantity
SS	BSS138	7''	8mm	3000 units

Electrical Characteristics

T_A = 25°C unless otherwise noted

Symbol	Parameter	Test Conditions	Min	Typ	Max	Units
Off Characteristics						
BV_{DSS}	Drain–Source Breakdown Voltage	V_{GS} = 0 V, I_D = 250 μA	50			V
$\frac{\Delta BV_{DSS}}{\Delta T_J}$	Breakdown Voltage Temperature Coefficient	I_D = 250 μA,Referenced to 25°C		72		mV/°C
I_{DSS}	Zero Gate Voltage Drain Current	V_{DS} = 50 V, V_{GS} = 0 V			0.5	μA
		V_{DS} = 50 V, V_{GS} = 0 V T_J = 125°C			5	μA
		V_{DS} = 30 V, V_{GS} = 0 V			100	nA
I_{GSS}	Gate–Body Leakage.	V_{GS} = ±20 V, V_{DS} = 0 V			±100	nA
On Characteristics (Note 2)						
$V_{GS(th)}$	Gate Threshold Voltage	V_{DS} = V_{GS}, I_D = 1 mA	0.8	1.3	1.5	V
$\frac{\Delta V_{GS(th)}}{\Delta T_J}$	Gate Threshold Voltage Temperature Coefficient	I_D = 1 mA,Referenced to 25°C		–2		mV/°C
$R_{DS(on)}$	Static Drain–Source On–Resistance	V_{GS} = 10 V, I_D = 0.22 A V_{GS} = 4.5 V, I_D = 0.22 A V_{GS} = 10 V, I_D = 0.22 A, T_J = 125°C		0.7 1.0 1.1	3.5 6.0 5.8	Ω
$I_{D(on)}$	On–State Drain Current	V_{GS} = 10 V, V_{DS} = 5 V	0.2			A
g_{FS}	Forward Transconductance	V_{DS} = 10V, I_D = 0.22 A	0.12	0.5		S
Dynamic Characteristics						
C_{iss}	Input Capacitance	V_{DS} = 25 V, V_{GS} = 0 V, f = 1.0 MHz		27		pF
C_{oss}	Output Capacitance			13		pF
C_{rss}	Reverse Transfer Capacitance			6		pF
R_G	Gate Resistance	V_{GS} = 15 mV, f = 1.0 MHz		9		Ω
Switching Characteristics (Note 2)						
$t_{d(on)}$	Turn–On Delay Time	V_{DD} = 30 V, I_D = 0.29 A, V_{GS} = 10 V, R_{GEN} = 6 Ω		2.5	5	ns
t_r	Turn–On Rise Time			9	18	ns
$t_{d(off)}$	Turn–Off Delay Time			20	36	ns
t_f	Turn–Off Fall Time			7	14	ns
Q_g	Total Gate Charge	V_{DS} = 25 V, I_D = 0.22 A, V_{GS} = 10 V		1.7	2.4	nC
Q_{gs}	Gate–Source Charge			0.1		nC
Q_{gd}	Gate–Drain Charge			0.4		nC
Drain–Source Diode Characteristics and Maximum Ratings						
I_S	Maximum Continuous Drain–Source Diode Forward Current				0.22	A
V_{SD}	Drain–Source Diode Forward Voltage	V_{GS} = 0 V, I_S = 0.44 A(Note 2)		0.8	1.4	V

Notes:

1. $R_{\theta JA}$ is the sum of the junction-to-case and case-to-ambient thermal resistance where the case thermal reference is defined as the solder mounting surface of the drain pins. $R_{\theta JC}$ is guaranteed by design while $R_{\theta CA}$ is determined by the user's board design.

a) 350°C/W when mounted on a minimum pad..

Scale 1 : 1 on letter size paper

2. Pulse Test: Pulse Width ≤ 300 μs, Duty Cycle ≤ 2.0%

Typical Characteristics

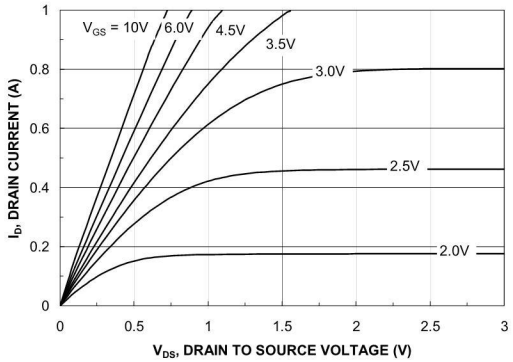


Figure 1. On-Region Characteristics.

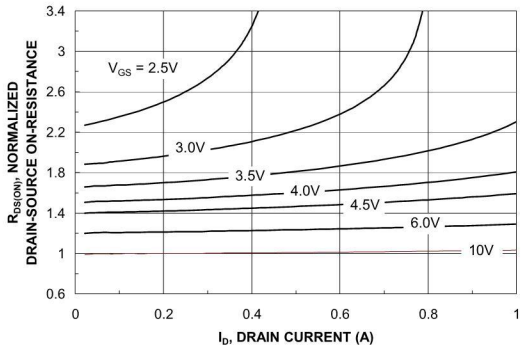


Figure 2. On-Resistance Variation with Drain Current and Gate Voltage.

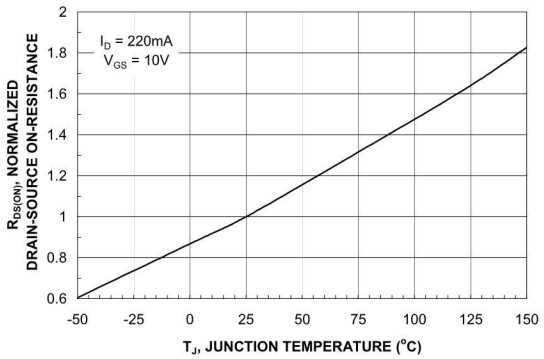


Figure 3. On-Resistance Variation with Temperature.

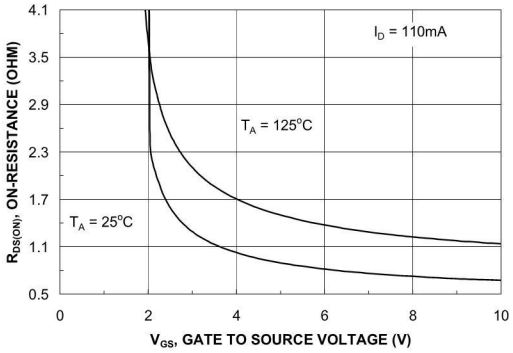


Figure 4. On-Resistance Variation with Gate-to-Source Voltage.

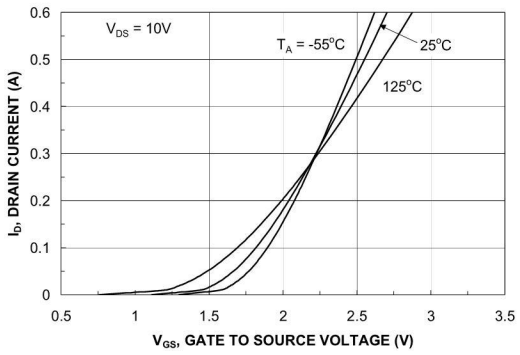


Figure 5. Transfer Characteristics.

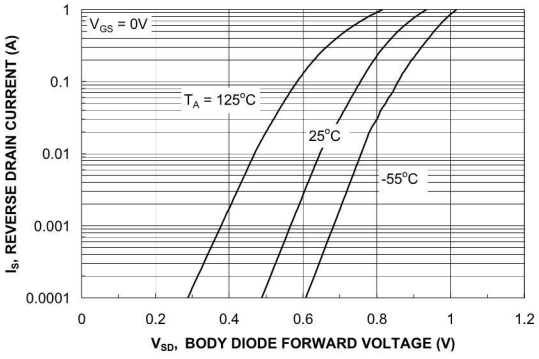


Figure 6. Body Diode Forward Voltage Variation with Source Current and Temperature.

Typical Characteristics

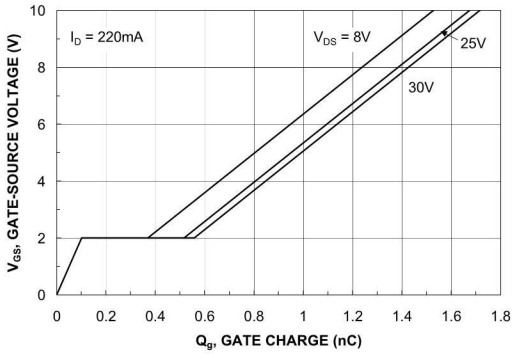


Figure 7. Gate Charge Characteristics.

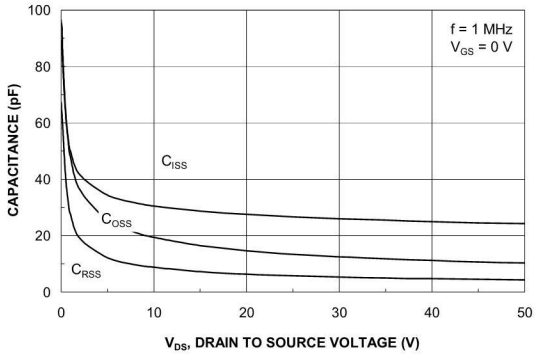


Figure 8. Capacitance Characteristics.

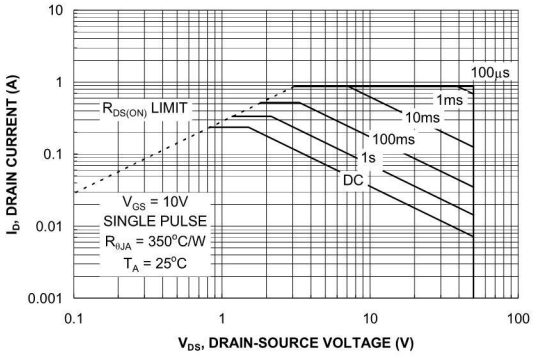


Figure 9. Maximum Safe Operating Area.

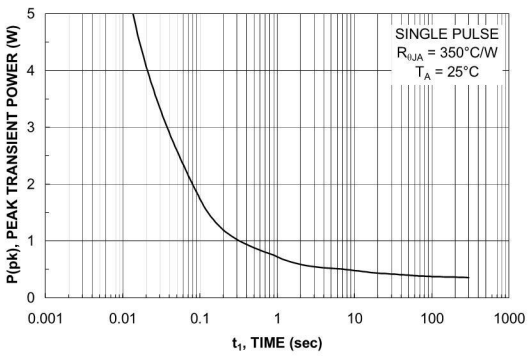


Figure 10. Single Pulse Maximum Power Dissipation.

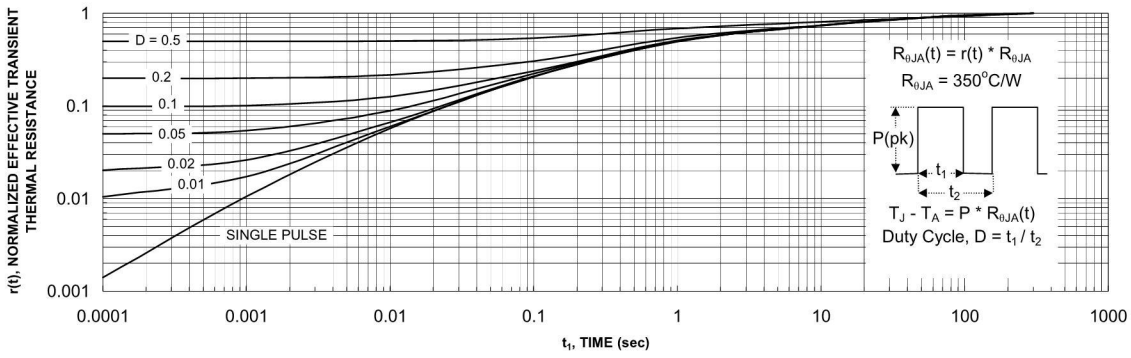


Figure 11. Transient Thermal Response Curve.

Thermal characterization performed using the conditions described in Note 1a.
Transient thermal response will change depending on the circuit board design.

TRADEMARKS

The following are registered and unregistered trademarks Fairchild Semiconductor owns or is authorized to use and is not intended to be an exhaustive list of all such trademarks.

ACEx™
ActiveArray™
Bottomless™
Build it Now™
CoolFET™
CROSSVOLT™
DOME™
EcoSPARK™
E²CMOS™
EnSigna™
FACT™
FACT Quiet Series™
FAST®
FASTr™
FPS™
FRFET™
GlobalOptoisolator™
GTO™
HiSeC™
I²C™
i-Lo™
ImpliedDisconnect™
IntelliMAX™
ISOPLANAR™
LittleFET™
MICROCOUPLER™
MicroFET™
MicroPak™
MICROWIRE™
MSX™
MSXPro™
OCX™
OCXPro™
OPTOLOGIC®
OPTOPLANAR™
PACMAN™
POP™
Power247™
PowerEdge™
PowerSaver™
PowerTrench®
QFET®
QS™
QT Optoelectronics™
Quiet Series™
RapidConfigure™
RapidConnect™
µSerDes™
ScalarPump™
SILENT SWITCHER®
SMART START™
SPM™
Stealth™
SuperFET™
SuperSOT™-3
SuperSOT™-6
SuperSOT™-8
SyncFET™
TinyLogic®
TINYOPTO™
TruTranslation™
UHC™
UltraFET®
UniFET™
VCX™
Wire™

Across the board. Around the world.™
The Power Franchise®
Programmable Active Droop™

DISCLAIMER

FAIRCHILD SEMICONDUCTOR RESERVES THE RIGHT TO MAKE CHANGES WITHOUT FURTHER NOTICE TO ANY PRODUCTS HEREIN TO IMPROVE RELIABILITY, FUNCTION OR DESIGN. FAIRCHILD DOES NOT ASSUME ANY LIABILITY ARISING OUT OF THE APPLICATION OR USE OF ANY PRODUCT OR CIRCUIT DESCRIBED HEREIN; NEITHER DOES IT CONVEY ANY LICENSE UNDER ITS PATENT RIGHTS, NOR THE RIGHTS OF OTHERS.

LIFE SUPPORT POLICY

FAIRCHILD'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF FAIRCHILD SEMICONDUCTOR CORPORATION.
As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, or (c) whose failure to perform when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in significant injury to the user.

2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

PRODUCT STATUS DEFINITIONS

Definition of Terms

Datasheet Identification	Product Status	Definition
Advance Information	Formative or In Design	This datasheet contains the design specifications for product development. Specifications may change in any manner without notice.
Preliminary	First Production	This datasheet contains preliminary data, and supplementary data will be published at a later date. Fairchild Semiconductor reserves the right to make changes at any time without notice in order to improve design.
No Identification Needed	Full Production	This datasheet contains final specifications. Fairchild Semiconductor reserves the right to make changes at any time without notice in order to improve design.
Obsolete	Not In Production	This datasheet contains specifications on a product that has been discontinued by Fairchild semiconductor. The datasheet is printed for reference information only.

User Manual of DYP-A01-V2.0

Waterproof Ultrasonic Distance Measuring Controller

一. Description:

Waterproof ultrasonic distance measuring controller of DYP-A01Y –V2.0 uses the integrated and enclosed waterproof ultrasonic transducer, which has a certain dust waterproof level, suitable for the wet and poor measurement occasion. It is with a special horn mouth, suitable for the larger testing range requirements. And it has different types of output mode, it's a commercial grade module with high-performance and high reliability.

二. Features:

1. Power Supply: DC3.3V to 5.0V;
2. Standby current can be below 10uA;
3. UART auto-output;
4. UART controlled output;
5. PWM auto-output;

6. PWM controlled output;
7. Digital output;
8. With integrated and enclosed waterproof ultrasonic transducer;
9. With temperature compensation;
10. Working Temperature: -15℃ to +60℃;
11. Storage Temperature: -25℃ to +80℃;
12. Measurement Accuracy: ±(1cm + S*0.3%) ,S is the measured value ;
13. Electrostatic protection design, the enclosure of sensor and the I/O PIN are added the electrostatic protection device, up to the standard of IEC61000-4-2;

三. Advantages:

1. With high degree of protection;
2. With strong anti-interference;
3. Data output, stable and reliable;
4. Lower power dissipation;
5. With strong anti - static ability;
6. High measurement accuracy；
7. Small size, easy installation;
8. With automatic output mode, release the user processor;
9. With controlled output mode, could minimize power consumption according to the actual application;

四. Applications:

1. Level distance measurement;
2. Liquid level measurement;
3. Parking management system;
4. Detect the objects close to and the existence of external objects;
5. Intelligent bin management system;
6. Robot obstacle avoidance、 automatic control;

7. Monitoring of water level in sewer and bottom hole;

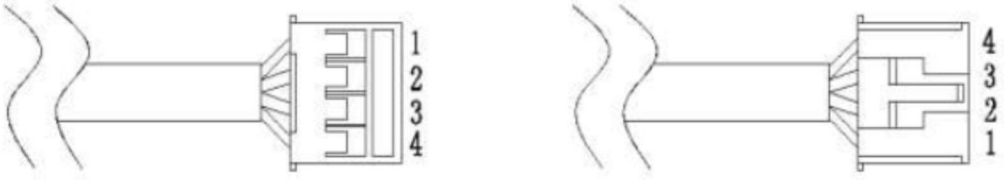
五.Basic Parameters:

Items	(A01A serial)For plane distance measurement	(A01B serial)For human body measurement	(A01C serial)For rubbish measurement	Unit	Remarks
Working voltage	3.3-5.0	3.3-5.0	3.3-5.0	V	DC
Average working current	< 10	< 10	< 10	mA	(1)
Blind distance	≤28	≤28	≤28	cm	
The measuring range (without horn mouth)	28-450	28-450	28-450	cm	(2)
The measuring range (with horn mouth)	28-750	28-750	28-750	cm	(2)
Detection Angle (without horn mouth)	-	≈75	-	Degree	(3)
Detection Angle (with horn mouth)	≈40	≈65	-	Degree	(3)
Output modes	UART auto output UART controlled output PWM auto output PWM controlled output Digital Output	UART auto output UART controlled output PWM auto output PWM controlled output Digital Output	UART auto output PWM auto output	-	

Note:

1. Above measured typical data bases on the power supply is 5V and work cycle is 100ms, which is at room temperature;
2. At room temperature, the tested object is a 50cm*60cm plane carton, and the detection direction should be as vertical as possible to the tested object;
3. At room temperature, the tested object is φ7.5cm * 100cm white PVC pipe, which is with 100cm distance; Different measured distance there are also differences in measuring angles;

六. Wiring definition:



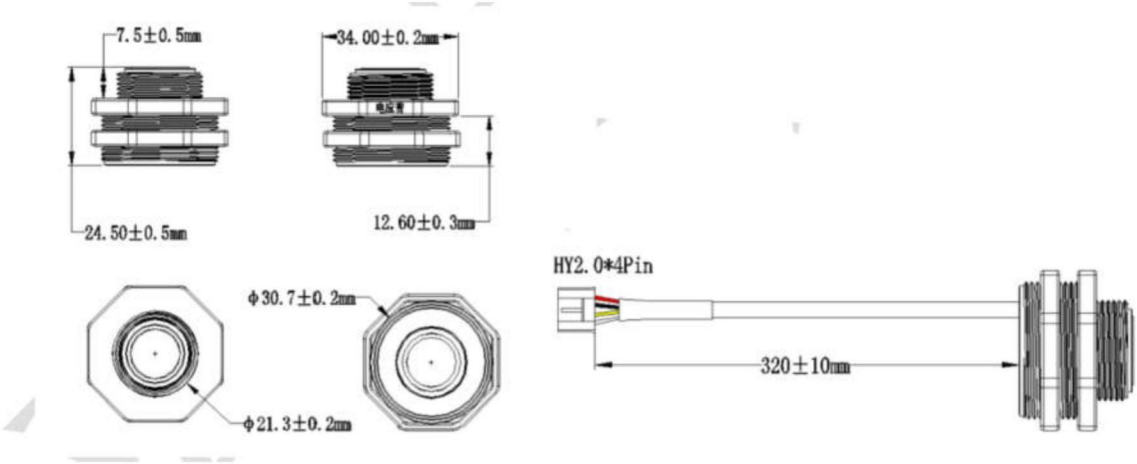
Wire number	Wire name	Wire description	Remarks
1	VCC	3.3-5V power supply access line	
2	GND	Power supply ground wire	
3	RX	Function wire	(1)
4	TX	Function wire	(1)

Remarks:

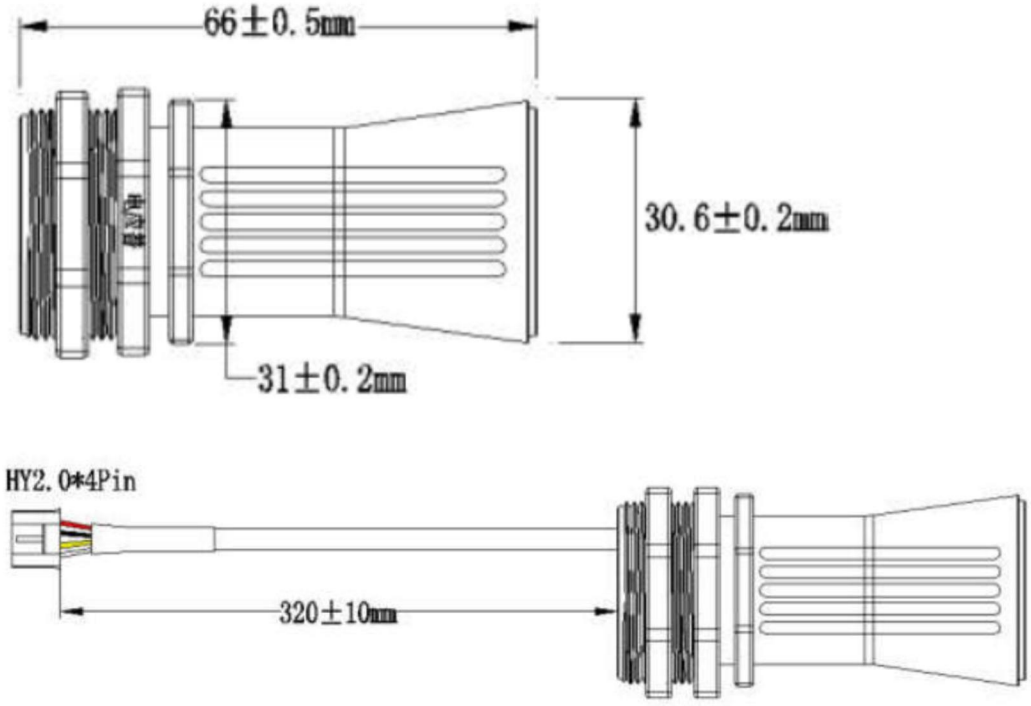
(1) The user shall choose the output mode as their practical use, the product cannot be compatible with two or more output modes at the same time.

七. Mechanical Features:

Without horn mouth:



With horn mouth:



八. Module series description

According to different characteristics and advantages, modules are divided into three series

For A01A serial, mainly is for plane distance measurement;

For A01B serial, mainly is for human body measurement;

For A01C serial, mainly is for rubbish measurement;

1. The description of A01A:

For A01A serial, mainly is for plane distance measurement: it can be used for targeted measurement of planar objects with long distance and high precision; There are 5 output modes, they are UART auto output, UART controlled output, PWM auto output, PWM controlled output and digital output;

2. The description of A01B:

For A01B serial, mainly is for human body measurement: sensitive to human body detection, human target measurement more stable, the blind area to measure the stability of the object. It can measure the upper body stably within 200cm without horn mouth and 350cm with horn mouth; There are 5 output modes, they are UART auto output, UART controlled output, PWM auto output, PWM controlled output and digital output;

3. The description of A01C:

For A01C serial, mainly is for rubbish measurement: through the special algorithm intelligent filter bin borders and interference objects and accurately measure the overflowing state of garbage in the garbage box. There are 2 output modes, they are UART auto output and PWM auto output;

九. Output Format

1. Working parameters of output mode

Items	UART auto	UART controlled	PWM auto	PWM controlled	Digital output	Unit	Remarks
Standby current	-	≤10	-	≤10	-	uA	
Work cycle	100~500	Controlled	250~500	Controlled	100	ms	(1)
Output mode	UART serial port	UART serial port	PWM pulse width	PWM pulse width	TTL digital output	-	
Measurement accuracy of flat objects	± (1+S*0.3%)	± (1+S*0.3%)	± (1+S*0.3%)	± (1+S*0.3%)	± (1+S*0.3%)	cm	(2)
temperature compensation	compensation	compensation	compensation	compensation	compensation	-	

Remarks:

(1) Regarding A01A and A01B serials, the work cycle is 250ms for UART auto output and PWM auto output; And the work cycle is 500ms for UART auto output and PWM auto output regarding A01C serial.

(2) At room temperature, the tested object is a 50cm*60cm plane carton. S represents the measurement distance, and the detection direction should be as vertical as possible to the tested object

2. The instructions of UART auto output

(1) Output cable definition

No. of cables	Cable name	Cable description	Remarks
1	VCC	Power input cable 3.3V~5V	
2	GND	Power ground cable	
3	RX	Processing value and real time value output selection cable	(1) (2)
4	TX	UART output cable	(1)

Remarks:

(1) The function of the lead wire corresponds to the output mode of the product model and cannot coexist with the output mode of other products.

(2) For A01C serial module which "RX" cable is empty.

(2) Communication instructions of UART auto output

For UART auto output of A01A and A01B serials, when the lead wire "RX" is hung in air or input high level, sensor will output according to the processing value, which data is more stable and the response time is about 300-500ms; When the lead wire "RX" inputs low level, sensor will output according to the real-time value, and the response time is 100ms.

For UART auto output of A01C serial, when the lead wire "RX" is hung in air, which is intelligent algorithm processing value output, and the response time is 500ms.

UART	Data bits	Stop bit	Parity check	Baud rate
TTL level	8	1	None	9600bps

(3) Output Format of UART

Frame data	Instruction	Byte
Frame header	Fixed to 0xFF	1 byte
Data_H	High 8 bits of distance data	1 byte
Data_L	Low 8 bits of distance data	1 byte
SUM	Checksum of communication	1 byte

(4) Example of UART output

Frame Data	Data_H	Data_L	SUM
0XFF	0X07	0XA1	0XA7

Note: the checksum retains the low 8 bits of accumulative value only;

SUM=(Frame data + Data_H + Data_L)&0x00FF

= (0XFF + 0X07 + 0XA1) &0x00FF

= 0XA7

Distance value = Data_H *256 + Data_L = 0X07A1;

Convert to decimal is 1953;

Means the current measured distance value is 1953mm;

4. UART controlled output

(1) Output cable definition

No. of cables	Cable name	Cable description	Remarks
1	VCC	Power input cable 3.3V~5V	
2	GND	Power ground cable	
3	RX	Trigger input cable	(1)
4	TX	UART output cable	(1)

Remarks:

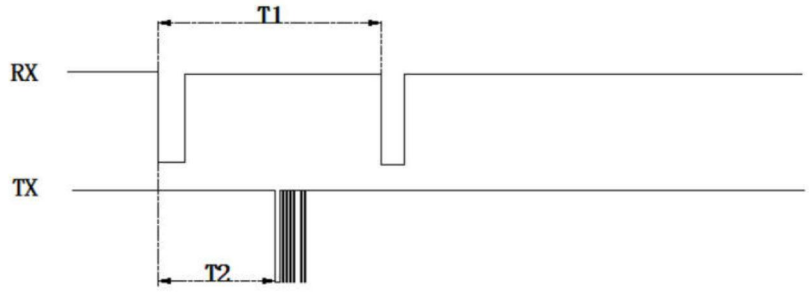
(1)The function of the lead wire corresponds to the output mode of the product model and cannot coexist with the output mode of other products.

(2) Communication instructions of UART controlled output

When the cable “RX” receives a trigger pulse with falling edge or any serial port data, the module makes a measurement, the cable “TX” will output a measurement distance value when the measurement is finished every time. The output mode can control the measurement period and reduce the power consumption. The trigger period of the module must be greater than 70ms

UART	Data bits	Stop bit	Parity check	Baud rate
TTL level	8	1	None	9600bps

(3) Timing diagram of UART controlled output



Note： T1 > 70ms, T2= 50~60ms;

(4) Output format of controlled UART

Frame data	Instruction	Byte
Frame header	Fixed to 0xFF	1 byte
Data_H	High 8 bits of distance data	1 byte
Data_L	Low 8 bits of distance data	1 byte
SUM	Checksum of communication	1 byte

(5) Example of controlled UART output

Frame Data	Data_H	Data_L	SUM
0XFF	0X07	0XA1	0XA7

Note: the checksum retains the low 8 bits of accumulative value only;

SUM=(Frame data + Data_H + Data_L)&0x00FF

= (0XFF + 0X07 + 0XA1) &0x00FF

= 0XA7

Distance value = Data_H *256 + Data_L = 0X07A1;

Convert to decimal is 1953;

Means the current measured distance value is 1953mm;

5.PWM auto output mode

(1) Output cable definition

No. of cables	Cable name	Cable description	Remarks
1	VCC	Power input cable 3.3V~5V	
2	GND	Power ground cable	
3	RX	Empty	(1)
4	TX	PWM output cable	(1)

Remarks: (1)The function of the lead wire corresponds to the output mode of the product model and cannot coexist with the output mode of other products.

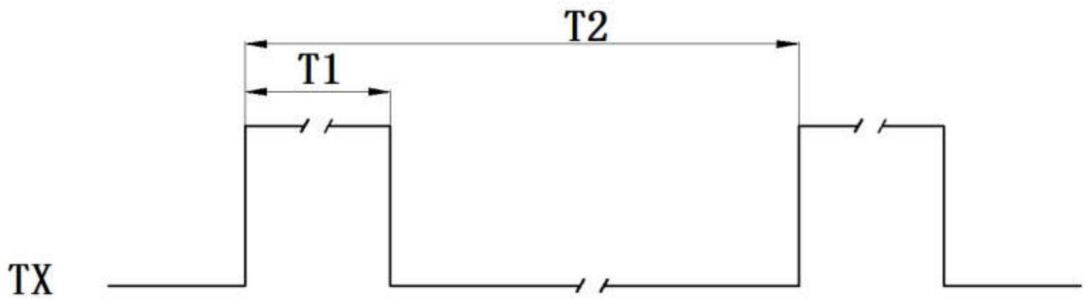
(2) Working instructions of PWM auto output

For PWM auto output of A01A and A01B serials, ranging is performed once every 250ms, and the range value detected by the module is converted to PWM high level pulse width, which is output by the "TX" wire. If no object is detected, the "TX" lead outputs a fixed pulse width of about 45ms.

For PWM auto output of A01C serial, ranging is performed every 500ms. If no object is detected, the "TX" wire will output a fixed pulse width of about 15ms.

PWM pulse width has temperature compensation, ranging is more accurate in different temperature environment.

(1) Timing diagram of PWM auto output



Note: For the PWM auto output of A01A and A02B serial: T1= 1.4~45ms; T2=250ms. For the PWM auto output of A01C: T1= 1.4~15ms; T2=500ms.

(2) Compute mode

Formula: S = T*V/2 (S is distance value, T is the time of PWM high level pulse width, V is sound travels in air)

The sound velocity V is 348M/S at room temperature, and the formula could be simplified to S=T/57.5 (now the unit of distance “S” is cm, unit of time “T” is us)

For example, when the output cable “TX”, which time of PWM high level pulse width T3 is 10000us, and S=T/57.5= 10000/57.5≈ 173.9(cm),means the current value of measured distance is 173.9cm.

6. PWM controlled output

(1) Output cable definition

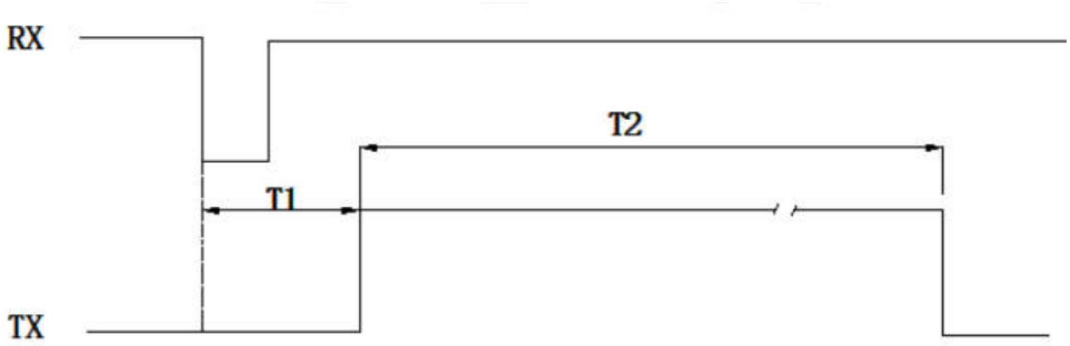
No. of cables	Cable name	Cable description	Remarks
1	VCC	Power input cable 3.3V~5V	
2	GND	Power ground cable	
3	RX	Trigger cable	(1)
4	TX	PWM output cable	(1)

Remarks: (1)The function of the lead wire corresponds to the output mode of the product model and cannot coexist with the output mode of other products.

(2) Working instructions of PWM auto output

The "RX" wire of the module receives a trigger pulse with falling edge or any serial port data, and the module will make a measurement once. The "TX" wire will output a TTL high level PWM pulse width signal, and the trigger period of the module must be greater than 70ms. If the module does not detect the object, the output wire "TX" will output a fixed pulse width of about 45ms.

(3) Timing diagram of PWM controlled output



Note: T1= 1 ~ 8ms; T2= 1.4 ~ 45ms (the time of PWM high level pulse width);

(4) Compute mode

Formula: S = T*V/2 (S is distance value, T is the time of PWM high level pulse width, V is sound travels in air)

The sound velocity is 348m/s at room temperature, and the formula could be simplified to S=T/57.5 (now the unit of distance “S” is cm, unit of time “T” is us)

For example, when the output wire “TX”, which time of PWM high level pulse width T3 is 10000us, and S=T/57.5= 10000/57.5≈ 173.9(cm),means the current value of measured distance is 173.9cm.

7. Digital output

(1) Output cable definition

No. of cables	Cable name	Cable description	Remarks
1	VCC	Power input cable 3.3V~5V	
2	GND	Power ground cable	
3	RX	Negative output wire of digital output	(1)
4	TX	Positive output wire of digital output	(1)

Remarks: (1)The function of the lead wire corresponds to the output mode of the product model and cannot coexist with the output mode of other products.

(2)The function instructions of digital output

The module factory will set a threshold value of 1.5m by default. The module conducts ranging every 100ms. When the detected target distance value is less

than the set threshold value, the "TX" wire outputs high level and the "RX" wire outputs low level. The current detected distance value is greater than the set threshold value, the "TX" wire outputs low level and the "RX" wire outputs high level. In order to improve the stability, the factory defaults to the threshold value set by the small residual of the detected target distance for 5 consecutive times to be the threshold value set by the small residual of the detected target distance. It is determined that the detected target distance is greater than the set threshold value for 10 times in a row. The lead of module "TX" and "RX" can only output high and low level signals without driving ability. If there are any special requirements to modify the threshold value or other settings, special instructions should be given at the time of purchase.

十. Limit Parameter

1. Rated Environmental Condition

Items	Min. Value	Typical Value	Max. Value	Unit	Remark
Storage Temperature	-25	25	80	℃	
Storage Humidity		65%	90%	RH	(1)
Working Temperature	-15	25	60	℃	
Working Humidity		65%	80%	RH	(1)

Notes:

A, When then environment temperature is in 0 to 39℃, the max. value of humidity is 90% (no condensation);

B, When the environment temperature is in 40℃ to 50℃, the max. value of humidity is the high humidity in the natural world under current temperature(no condensation);

2. Rated Electrical Condition

Items	Specification			Unit	Remark
	Min. Value	Typical Value	Max. Value		
Working Voltage	3.1	5.0	5.25	V	
Peak Current	50		75	mA	Peak to peak value
Input Ripple			50	mV	Peak to peak value
Input Noise			100	mV	Peak to peak value
ESD			±200K/±2K	V	(1)
ESD			±4K/±8K	V	(2)

Note:

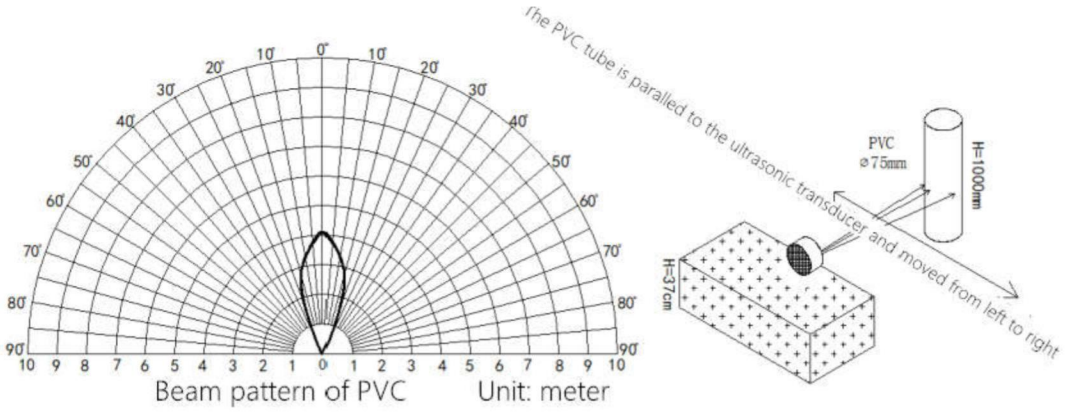
4. For the electrostatic specifications of assemble lead, the contact electrostatic could not be more than ±200V, and the air electrostatic could not be more than ±2KV;

5. The shell of ultrasonic transducer and the lead wire meets the standard of IEC61000-4-2;

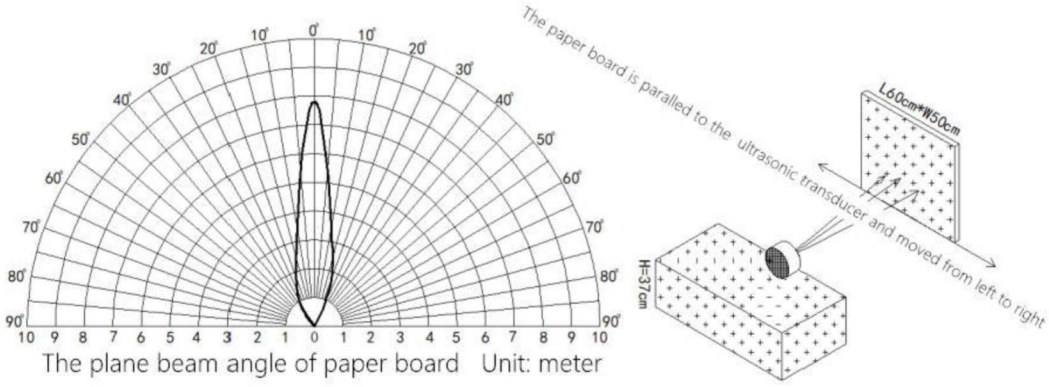
十一. The Effective Detection Range:

1. The reference beam image of A01A serial:

(1) The tested object is the white cylindrical tube, material is PVC, height is 100cm, diameter is 7.5cm:

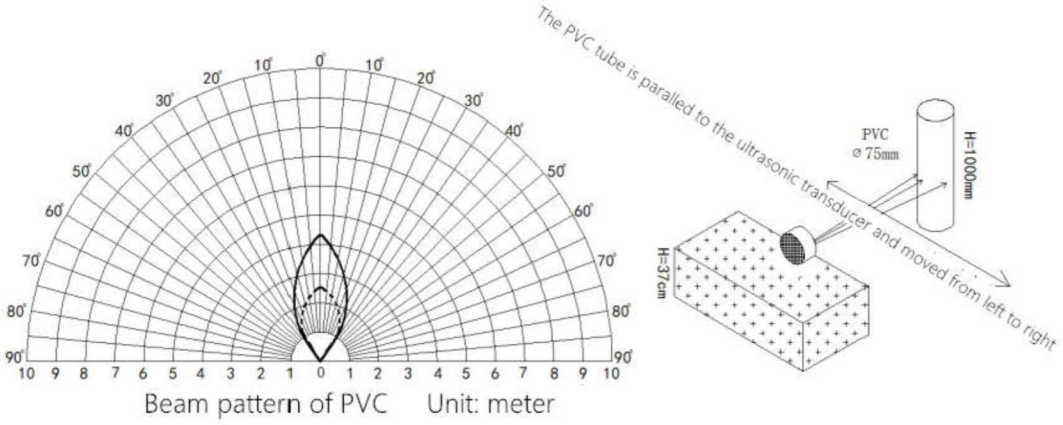


(2) The tested object is the corrugated case, perpendicular to 0° axle wire, length is 60cm, width is 50cm:

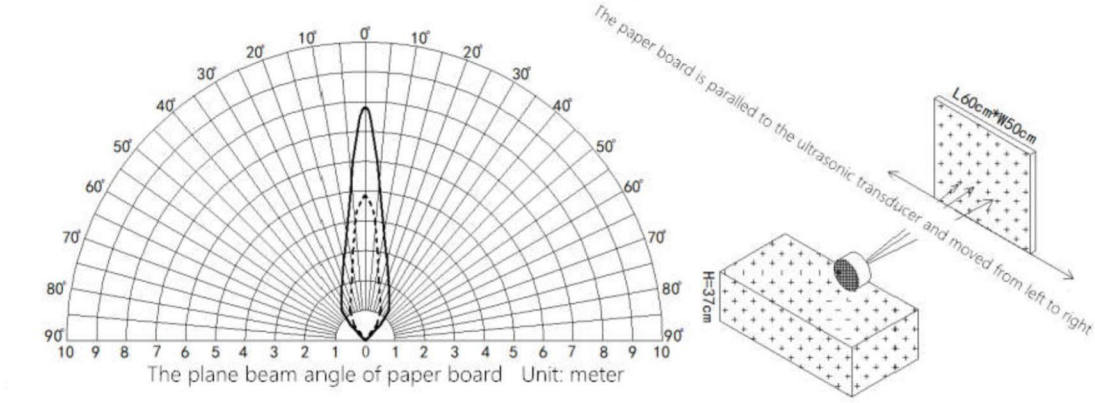


2. **The reference beam image of A01B serial:**

(1) The tested object is the white cylindrical tube, material is PVC, height is 100cm, diameter is 75mm (The solid line is the data from the one with horn mouth, and the dotted line is the data from the one without horn mouth):



(2)The tested object is the corrugated case, perpendicular to 0° axle wire, length is 60cm, width is 50cm(The solid line is the data from the one with horn mouth, and the dotted line is the data from the one without horn mouth):



Note: Above testing data is from our company testing room, in practical application different installation and use environment, the final data will be some difference with ours, please refer to the actual application test.

十二. Instructions of model selection

This product can be divided into three series according to different application scenarios, and the output format is also divided into various. Users choose the corresponding model according to their actual application needs:

No.	Application	Features	Output mode	Item No.
A01A serial	For flat object measurement	1.Waterproof case + horn mouth; 2.The measuring range is 28cm to 750cm; 3.Distance measuring angle≈40°	UART auto output	DYP-A01ANYUB-V2.0
			UART controlled output	DYP-A01ANYTB-V2.0
			PWM auto output	DYP-A01ANYWB-V2.0
			PWM controlled output	DYP-A01ANYMB-V2.0
			Digital output	DYP-A01ANYGDB-V2.0
A01B serial	For human body measurement	1. Waterproof case; 2.The measuring range is 28cm to 450cm; 3.Distance measuring angle≈75°	UART auto output	DYP-A01BNYUW-V2.0
			UART controlled output	DYP-A01BNYTW-V2.0
			PWM auto output	DYP-A01BNYWW-V2.0
			PWM controlled output	DYP-A01BNYMW-V2.0
			Digital output	DYP-A01BNYGDW-V2.0
		1.Waterproof case+ horn mouth;; 2.The measuring range is 28cm to 750cm; 3. The upper body was measured stably within 350cm; 4.Distance measuring angle≈65°	UART auto output	DYP-A01BNYUB-V2.0
			UART controlled output	DYP-A01BNYTB-V2.0
			PWM auto output	DYP-A01BNYWB-V2.0
			PWM controlled output	DYP-A01BNYMB-V2.0
			Digital output	DYP-A01BNYGDB-V2.0
A01C serial	For rubbish measurement	1.Waterproof case+ horn mouth;; 2.The measuring range is 28cm to 250cm;	UART auto output	DYP-A01CNYUB-V2.0
			PWM auto output	DYP-A01CNYWB-V2.0

Note:

For the measuring range (flat object): at room temperature, the tested object is the corrugated case which is with 60cm(L) * 50cm (w).

For the distance measuring angle: at room temperature, which tested object is the white PVC tube withφ7.5cm*100cm and the sensor was placed on the 100cm;

十三. Reliability Test Condition:

Item	Test Items	Experiment Conditions	Sample Quantity	Remark
1	High temperature and humidity working	65℃,85%RH,power on@5V, 72hrs	3	
2	Low temperature working	-20℃, power on @5V, 72hrs	3	
3	High temperature and humidity storage	80℃,80% RH, storage, 72hrs	3	
4	Low temperature storage	-30℃, storage, 72hrs	3	
5	Vibration test	10-200Hz,15min.,2.0G,XYZ, three axial and each one is 0.5h	3	
6	Drop test	1.2m free fall, 5times@wooden floor	3	

Remark: After testing, the sensor passes the function test will be determined as ok, and the performance attenuation rate ≤10%.

十四、Matters needing attention：

1) Pay attention to structural tolerances when designing, due to unreasonable structure design may cause transient abnormality of module function;

2) Pay attention to EMC evaluation when designing, due to unreasonable system design may cause transient abnormality of module function;

3) Involving the boundary application of product limit parameters, please contact with our FAE to confirm the matters needing attention;

4) We reserve the right to change this document, features updated without notice;

十五、Package specification:

1) Default as our regular packing mode;

2) Packaging materials can be customized according to customer IQC standards;

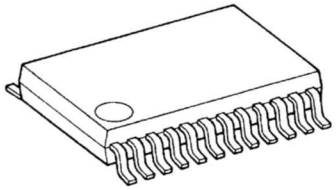
3) Container transport way staggered LCL is required, at the same time, the outer edge of a single stack should be coated with reinforced corner plate to provide sufficient support;

Toshiba Bi-CD Integrated Circuit Silicon Monolithic

TB6612FNG

Driver IC for Dual DC motor

TB6612FNG is a driver IC for DC motor with output transistor in LD MOS structure with low ON-resistor. Two input signals, IN1 and IN2, can choose one of four modes such as CW, CCW, short brake, and stop mode.



SSOP24-P-300-0.65A

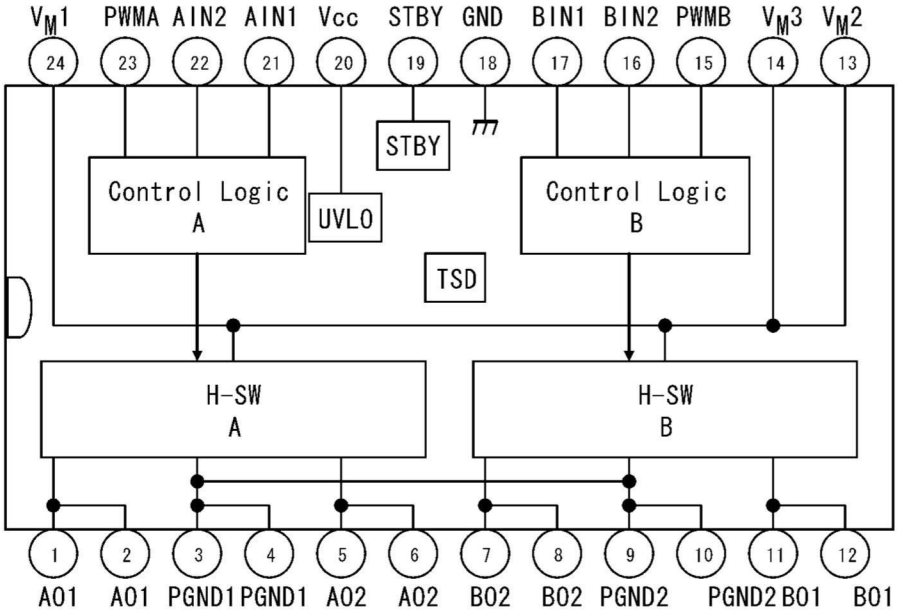
Weight: 0.14 g (typ.)

Features

- Power supply voltage: VM = 15 V(Max)
- Output current: I_{OUT} = 1.2 A(ave)/3.2 A (peak)
- Output low ON resistor: 0.5Ω (upper+lower Typ. @ VM ≥ 5 V)
- Standby (Power save) system
- CW/CCW/short brake/stop function modes
- Built-in thermal shutdown circuit and low voltage detecting circuit
- Small faced package(SSOP24: 0.65 mm Lead pitch)

* This product has a MOS structure and is sensitive to electrostatic discharge. When handling this product, ensure that the environment is protected against electrostatic discharge by using an earth strap, a conductive mat and an ionizer. Ensure also that the ambient temperature and relative humidity are maintained at reasonable levels.

Block Diagram



Pin Functions

No.	Pin Name	I/O	Function
1	AO1	O	ch A output 1
2	AO1		
3	PGND1	—	Power GND 1
4	PGND1		
5	AO2	O	ch A output 2
6	AO2		
7	BO2	O	ch B output 2
8	BO2		
9	PGND2	—	Power GND 2
10	PGND2		
11	BO1	O	ch B output 1
12	BO1		
13	VM2	—	Motor supply
14	VM3		
15	PWMB	I	ch B PWM input/200 kΩ pull-down at internal
16	BIN2	I	ch B input 2/200 kΩ pull-down at internal
17	BIN1	I	ch B input 1/200 kΩ pull-down at internal
18	GND	—	Small signal GND
19	STBY	I	“L” = standby/200 kΩ pull-down at internal
20	Vcc	—	Small signal supply
21	AIN1	I	ch A input 1/200 kΩ pull-down at internal
22	AIN2	I	ch A input 2/200 kΩ pull-down at internal
23	PWMA	I	ch A PWM input/200 kΩ pull-down at internal
24	VM1	—	Motor supply

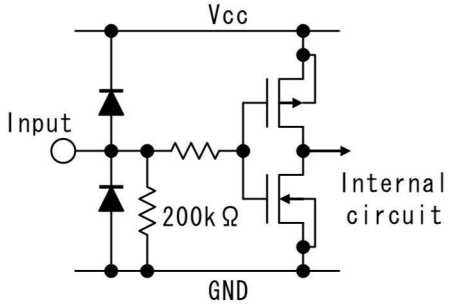
Absolute Maximum Ratings (Ta = 25°C)

Characteristics	Symbol	Rating	Unit	Remarks
Supply voltage	VM	15	V	
	V_{CC}	6		
Input voltage	V_{IN}	-0.2 to 6	V	IN1,IN2,STBY,PWM pins
Output voltage	V_{OUT}	15	V	O1,O2 pins
Output current	I_{OUT}	1.2	A	Per 1 ch
	I_{OUT} (peak)	2		tw = 20 ms Continuous pulse, Duty ≤ 20%
		3.2		tw = 10 ms Single pulse
Power dissipation	P_D	0.78	W	IC only
		0.89		50 mm × 50 mm t = 1.6 mm Cu ≥ 40% in PCB mounting
		1.36		76.2 mm × 114.3 mm t = 1.6 mm Cu ≥ 30% in PCB monting
Operating temperature	T_{opr}	-20 to 85	°C	
Storage temperature	T_{stg}	-55 to 150	°C	

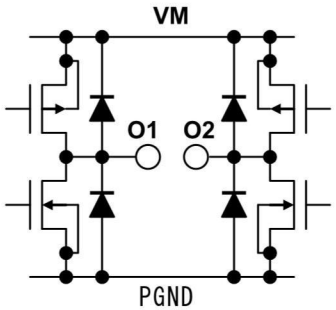
Operating Range (Ta = -20 to 85°C)

Characteristics	Symbol	Min	Typ.	Max	Unit	Remarks
Supply voltage	V_{CC}	2.7	3	5.5	V	
	VM	2.5	5	13.5	V	
Output current (H-SW)	I_{OUT}	—	—	1.0	A	VM ≥ 4.5 V
		—	—	0.4		4.5 V > VM ≥ 2.5 V Without PWM Operation
Switching frequency	f_{PWM}	—	—	100	kHz	

Input pin: IN1, IN2, PWM, STBY



Output pin: O1, O2

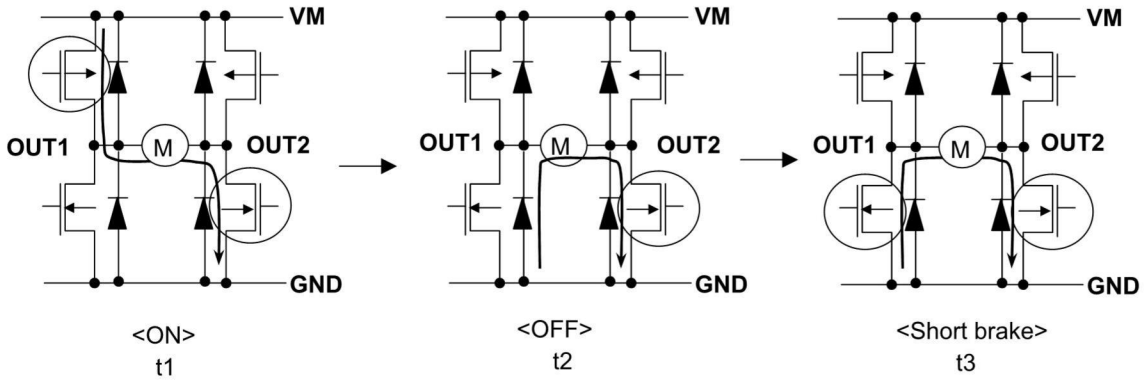


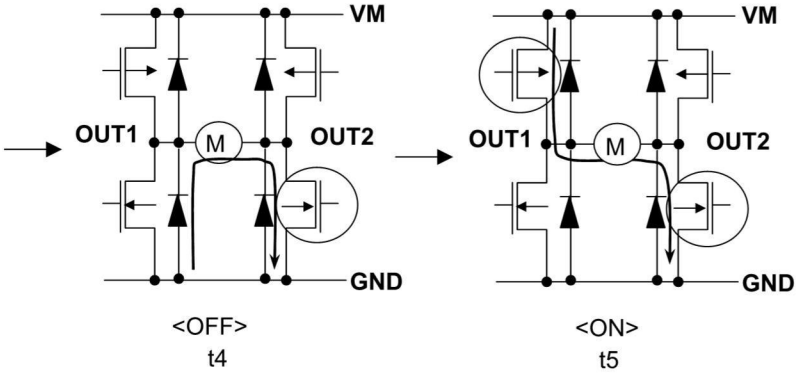
H-SW Control Function

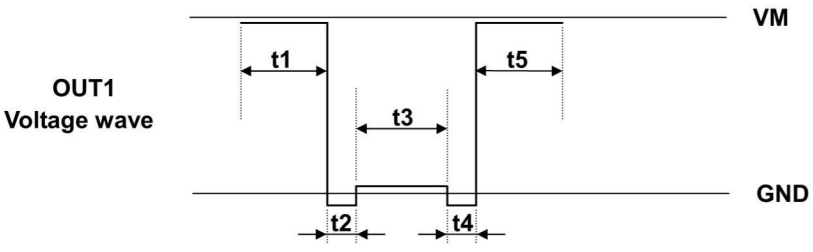
Input				Output		
IN1	IN2	PWM	STBY	OUT1	OUT2	Mode
H	H	H/L	H	L	L	Short brake
L	H	H	H	L	H	CCW
		L	H	L	L	Short brake
H	L	H	H	H	L	CW
		L	H	L	L	Short brake
L	L	H	H	OFF (High impedance)		Stop
H/L	H/L	H/L	L	OFF (High impedance)		Standby

H-SW Operating Description

・To prevent penetrating current, dead time t2 and t4 is provided in switching to each mode in the IC.



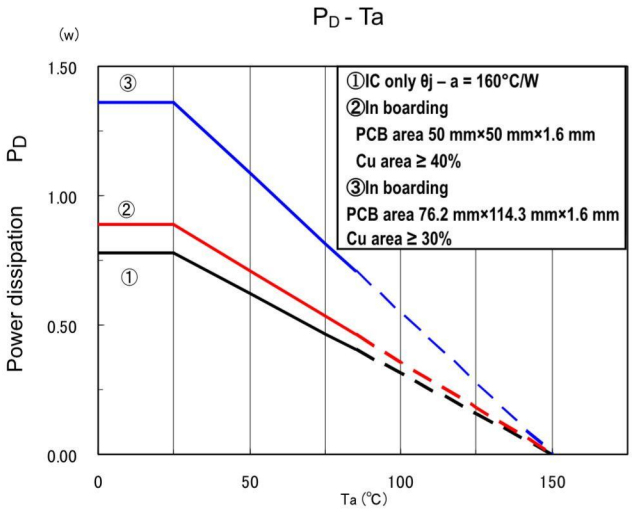


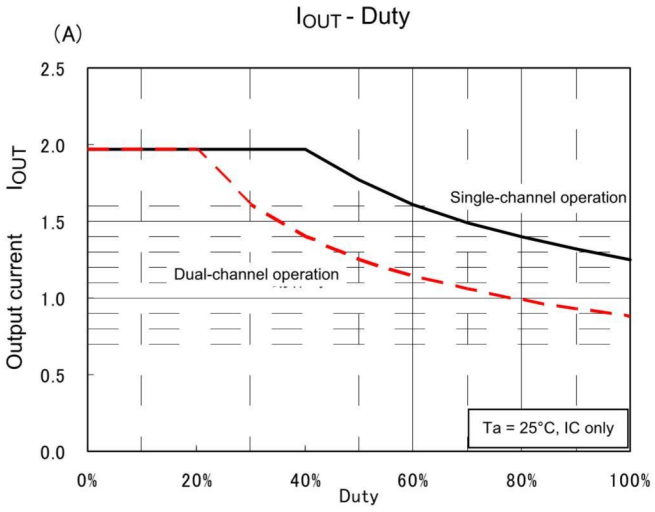


Electrical Characteristics (unless otherwise specified, Ta = 25°C, Vcc = 3 V, VM = 5 V)

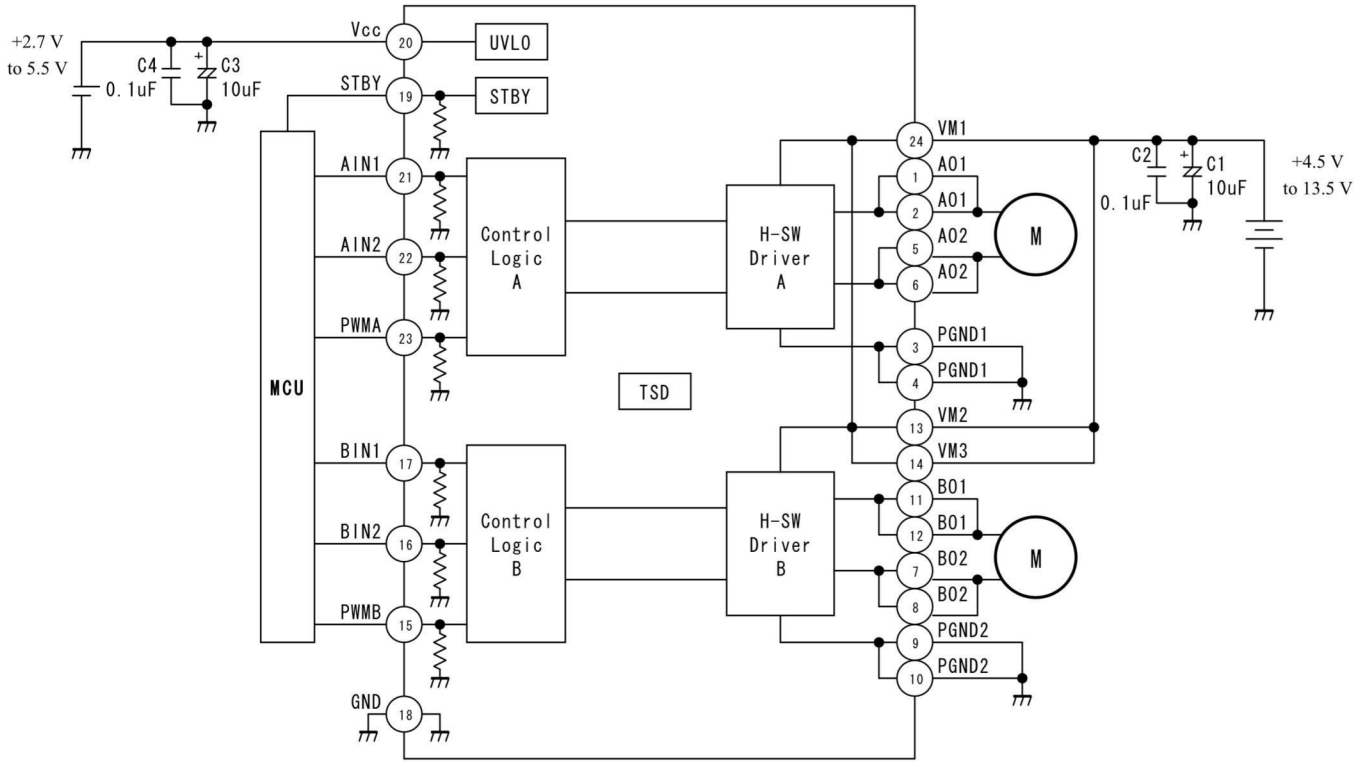
Characteristics	Symbol	Test Condition	Min	Typ.	Max	Unit
Supply current	I_{CC}(3 V)	STBY = Vcc = 3 V, VM = 5 V	—	1.1	1.8	mA
	I_{CC}(5.5 V)	STBY = Vcc = 5.5 V, VM = 5 V	—	1.5	2.2	
	$I_{CC(STB)}$	STBY = 0 V	—	—	1	μA
	$I_{M(STB)}$		—	—	1	
Control input voltage	V_{IH}	—	Vcc×0.7	—	Vcc+0.2	V
	V_{IL}		-0.2	—	Vcc×0.3	
Control input current	I_{IH}	V_{IN} = 3 V	5	15	25	μA
	I_{IL}	V_{IN} = 0 V	—	—	1	
Standby input voltage	$V_{IH(STB)}$	—	Vcc×0.7	—	Vcc+0.2	V
	$V_{IL(STB)}$		-0.2	—	Vcc×0.3	
Standby input current	$I_{IH(STB)}$	V_{IN} = 3 V	5	15	25	μA
	$I_{IL(STB)}$	V_{IN} = 0 V	—	—	1	
Output saturating voltage	$V_{sat(U+L)1}$	I_O = 1 A, Vcc = VM = 5 V	—	0.5	0.7	V
	$V_{sat(U+L)2}$	I_O = 0.3 A, Vcc = VM = 5 V	—	0.15	0.21	
Output leakage current	$I_{L(U)}$	VM = V_{OUT} = 15 V	—	—	1	μA
	$I_{L(L)}$	VM = 15 V, V_{OUT} = 0 V	-1	—	—	
Regenerative diode VF	$V_{F(U)}$	I_F = 1A	—	1	1.1	V
	$V_{F(L)}$		—	1	1.1	
Low voltage detecting voltage	UVLD	(Design target only)	—	1.9	—	V
Recovering voltage	UVLC		—	2.2	—	
Response speed	t_r	(Design target only)	—	24	—	ns
	t_f		—	41	—	
	Dead time H to L	Penetration protect time	—	50	—	
	Dead time L to H	(Design target only)	—	230	—	
Thermal shutdown circuit operating temperature	TSD	(Design target only)	—	175	—	°C
Thermal shutdown hysteresis	ΔTSD		—	20	—	

Target characteristics





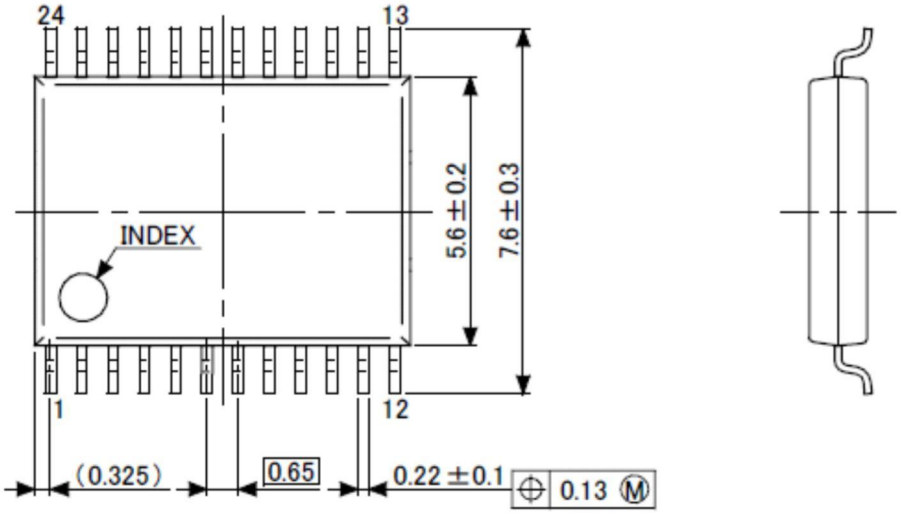
Typical Application Diagram

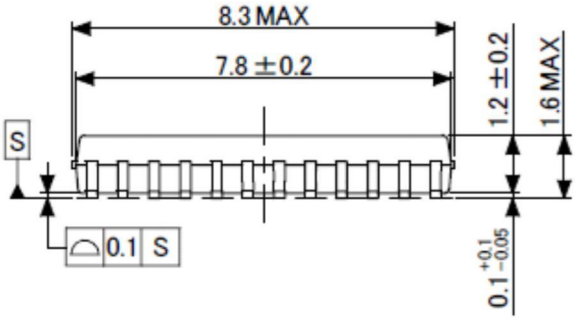


Note: Condensers for noise absorption (C1, C2, C3, and C4) should be connected as close as possible to the IC.

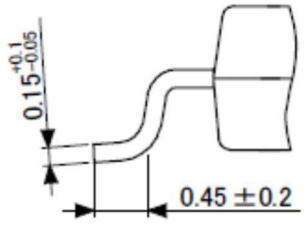
Package Dimennsions

SSOP24-P-300-0.65A "Unit : mm"





Detail of a terminal



Weght: 0.14 g (typ)

Notes on Contents

1. Block Diagrams

Some of the functional blocks, circuits, or constants in the block diagram may be omitted or simplified for explanatory purposes.

2. Equivalent Circuits

The equivalent circuit diagrams may be simplified or some parts of them may be omitted for explanatory purposes.

3. Timing Charts

Timing charts may be simplified for explanatory purposes.

4. Application Circuits

The application circuits shown in this document are provided for reference purposes only. Thorough evaluation is required, especially at the mass production design stage.
Toshiba does not grant any license to any industrial property rights by providing these examples of application circuits.

5. Test Circuits

Components in the test circuits are used only to obtain and confirm the device characteristics. These components and circuits are not guaranteed to prevent malfunction or failure from occurring in the application equipment.

IC Usage Considerations

Notes on handling of ICs

[1] The absolute maximum ratings of a semiconductor device are a set of ratings that must not be exceeded, even for a moment. Do not exceed any of these ratings.
Exceeding the rating(s) may cause the device breakdown, damage or deterioration, and may result injury by explosion or combustion.

[2] Use an appropriate power supply fuse to ensure that a large current does not continuously flow in case of over current and/or IC failure. The IC will fully break down when used under conditions that exceed its absolute maximum ratings, when the wiring is routed improperly or when an abnormal pulse noise occurs from the wiring or load, causing a large current to continuously flow and the breakdown can lead smoke or ignition. To minimize the effects of the flow of a large current in case of breakdown, appropriate settings, such as fuse capacity, fusing time and insertion circuit location, are required.

[3] If your design includes an inductive load such as a motor coil, incorporate a protection circuit into the design to prevent device malfunction or breakdown caused by the current resulting from the inrush current at power ON or the negative current resulting from the back electromotive force at power OFF. IC breakdown may cause injury, smoke or ignition.
Use a stable power supply with ICs with built-in protection functions. If the power supply is unstable, the protection function may not operate, causing IC breakdown. IC breakdown may cause injury, smoke or ignition.

[4] Do not insert devices in the wrong orientation or incorrectly.
Make sure that the positive and negative terminals of power supplies are connected properly.
Otherwise, the current or power consumption may exceed the absolute maximum rating, and exceeding the rating(s) may cause the device breakdown, damage or deterioration, and may result injury by explosion or combustion.
In addition, do not use any device that is applied the current with inserting in the wrong orientation or incorrectly even just one time.

Points to remember on handling of ICs

(1) Thermal Shutdown Circuit

Thermal shutdown circuits do not necessarily protect ICs under all circumstances. If the thermal shutdown circuits operate against the over temperature, clear the heat generation status immediately.

Depending on the method of use and usage conditions, such as exceeding absolute maximum ratings can cause the thermal shutdown circuit to not operate properly or IC breakdown before operation.

(2) Heat Radiation Design

In using an IC with large current flow such as power amp, regulator or driver, please design the device so that heat is appropriately radiated, not to exceed the specified junction temperature (Tj) at any time and condition. These ICs generate heat even during normal use. An inadequate IC heat radiation design can lead to decrease in IC life, deterioration of IC characteristics or IC breakdown. In addition, please design the device taking into considerate the effect of IC heat radiation with peripheral components.

(3) Back-EMF

When a motor rotates in the reverse direction, stops or slows down abruptly, a current flow back to the motor's power supply due to the effect of back-EMF. If the current sink capability of the power supply is small, the device's motor power supply and output pins might be exposed to conditions beyond absolute maximum ratings. To avoid this problem, take the effect of back-EMF into consideration in system design.

RESTRICTIONS ON PRODUCT USE

- Toshiba Corporation, and its subsidiaries and affiliates (collectively "TOSHIBA"), reserve the right to make changes to the information in this document, and related hardware, software and systems (collectively "Product") without notice.
- This document and any information herein may not be reproduced without prior written permission from TOSHIBA. Even with TOSHIBA's written permission, reproduction is permissible only if reproduction is without alteration/omission.
- Though TOSHIBA works continually to improve Product's quality and reliability, Product can malfunction or fail. Customers are responsible for complying with safety standards and for providing adequate designs and safeguards for their hardware, software and systems which minimize risk and avoid situations in which a malfunction or failure of Product could cause loss of human life, bodily injury or damage to property, including data loss or corruption. Before customers use the Product, create designs including the Product, or incorporate the Product into their own applications, customers must also refer to and comply with (a) the latest versions of all relevant TOSHIBA information, including without limitation, this document, the specifications, the data sheets and application notes for Product and the precautions and conditions set forth in the "TOSHIBA Semiconductor Reliability Handbook" and (b) the instructions for the application with which the Product will be used with or for. Customers are solely responsible for all aspects of their own product design or applications, including but not limited to (a) determining the appropriateness of the use of this Product in such design or applications; (b) evaluating and determining the applicability of any information contained in this document, or in charts, diagrams, programs, algorithms, sample application circuits, or any other referenced documents; and (c) validating all operating parameters for such designs and applications. **TOSHIBA ASSUMES NO LIABILITY FOR CUSTOMERS' PRODUCT DESIGN OR APPLICATIONS.**
- **PRODUCT IS NEITHER INTENDED NOR WARRANTED FOR USE IN EQUIPMENTS OR SYSTEMS THAT REQUIRE EXTRAORDINARILY HIGH LEVELS OF QUALITY AND/OR RELIABILITY, AND/OR A MALFUNCTION OR FAILURE OF WHICH MAY CAUSE LOSS OF HUMAN LIFE, BODILY INJURY, SERIOUS PROPERTY DAMAGE AND/OR SERIOUS PUBLIC IMPACT ("UNINTENDED USE").** Except for specific applications as expressly stated in this document, Unintended Use includes, without limitation, equipment used in nuclear facilities, equipment used in the aerospace industry, medical equipment, equipment used for automobiles, trains, ships and other transportation, traffic signaling equipment, equipment used to control combustions or explosions, safety devices, elevators and escalators, devices related to electric power, and equipment used in finance-related fields. **IF YOU USE PRODUCT FOR UNINTENDED USE, TOSHIBA ASSUMES NO LIABILITY FOR PRODUCT.** For details, please contact your TOSHIBA sales representative.
- Do not disassemble, analyze, reverse-engineer, alter, modify, translate or copy Product, whether in whole or in part.
- Product shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable laws or regulations.
- The information contained herein is presented only as guidance for Product use. No responsibility is assumed by TOSHIBA for any infringement of patents or any other intellectual property rights of third parties that may result from the use of Product. No license to any intellectual property right is granted by this document, whether express or implied, by estoppel or otherwise.
- **ABSENT A WRITTEN SIGNED AGREEMENT, EXCEPT AS PROVIDED IN THE RELEVANT TERMS AND CONDITIONS OF SALE FOR PRODUCT, AND TO THE MAXIMUM EXTENT ALLOWABLE BY LAW, TOSHIBA (1) ASSUMES NO LIABILITY WHATSOEVER, INCLUDING WITHOUT LIMITATION, INDIRECT, CONSEQUENTIAL, SPECIAL, OR INCIDENTAL DAMAGES OR LOSS, INCLUDING WITHOUT LIMITATION, LOSS OF PROFITS, LOSS OF OPPORTUNITIES, BUSINESS INTERRUPTION AND LOSS OF DATA, AND (2) DISCLAIMS ANY AND ALL EXPRESS OR IMPLIED WARRANTIES AND CONDITIONS RELATED TO SALE, USE OF PRODUCT, OR INFORMATION, INCLUDING WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, ACCURACY OF INFORMATION, OR NONINFRINGEMENT.**
- Do not use or otherwise make available Product or related software or technology for any military purposes, including without limitation, for the design, development, use, stockpiling or manufacturing of nuclear, chemical, or biological weapons or missile technology products (mass destruction weapons). Product and related software and technology may be controlled under the applicable export laws and regulations including, without limitation, the Japanese Foreign Exchange and Foreign Trade Law and the U.S. Export Administration Regulations. Export and re-export of Product or related software or technology are strictly prohibited except in compliance with all applicable export laws and regulations.
- Please contact your TOSHIBA sales representative for details as to environmental matters such as the RoHS compatibility of Product. Please use Product in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. **TOSHIBA ASSUMES NO LIABILITY FOR DAMAGES OR LOSSES OCCURRING AS A RESULT OF NONCOMPLIANCE WITH APPLICABLE LAWS AND REGULATIONS.**

TF02-Pro LiDAR （Mid-range distance sensor）

1. Product Description

TF02-Pro is a single-point ranging LiDAR based on TF02 upgrade. The performance and accuracy of different reflectivity are improved, it can achieve stable, accuracy, sensitive and high frequency range detection.

Main features of product:

- The range up to 40 meters
- Ambient light resistance(Up to 100Klux)
- High frame rate(up to 1000Hz)
- Low power consumption

Main applications:

- Intelligent traffic
- Intelligent parking lots
- Material level monitoring
- UAV

2. Technical Specifications and Parameters

Table 1 Main characteristic parameters of TF02-Pro

Parameter Name		Value
Product performance	Operating Range	0.1~40m@90%reflectivity 0.1~13.5m@10%reflectivity 0.1~40m@90% reflectivity (100Klux) 0.1~13.5m@10% reflectivity (100Klux)
	Accuracy[1]	±5cm (0.1~5m), ±1% (5~40m)
	Distance Resolution	1cm
	frame rate	100Hz
	Repeatability	1σ: <2cm (0.1~35m@90% reflectivity)
	Ambient light resistance	100Klux
	Operation t temperature	-20~60℃
	Protection Level	IP65
Optical Parameters	Light source	VCSEL
	Central wavelength	850nm
	FOV	3°
	Photobiological safety	Class 1 (EN60825)
Electrical Parameters	Supply voltage	DC 5V~12V
	Average Current	≤200mA
	Power consumption	≤1W
	Peak Current	300mA
	Communication level	LVTTL (3.3V)
	Communication interface	UART/I2C
Other Parameters	Dimensions	69mmx41.5mmx26mm (L*W*H)
	Weight	50g (with cables)
	Enclosure material	PC/ABS
	Storage temperature	-30~80℃
	Cable length	80cm

[1] Accuracy was calculated based on a standard white board with 90% reflectivity in indoor condition(25℃), changes in conditions may cause errors to increase

3. Product Appearance and Structure

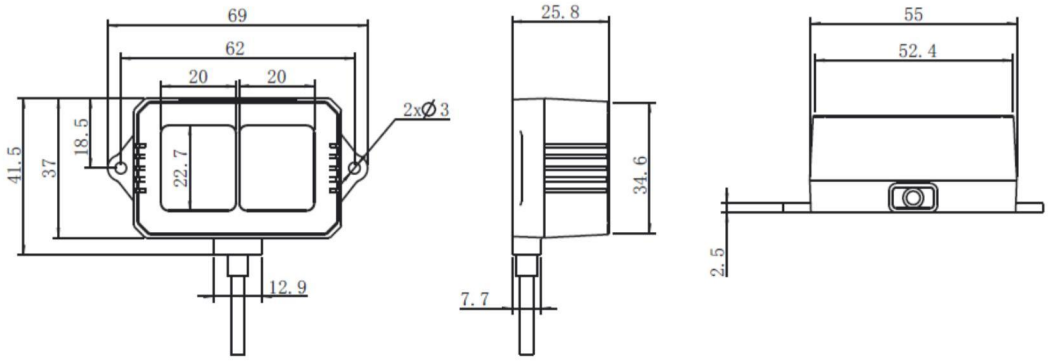


Figure 1 Dimension of TF02-Pro （Unit:mm）

4. Communication Interface

Table 2 Communication Interface--UART

Default Baud rate	115200
Data bits	8
Stop bit	1
Parity	None

Table 3 Communication Interface--I^2C

Max transmission rate	400kbps
Master/Slave mode	Slave
Default address	0x10
Address range	0x01~0x7F

5. Configurable Parameters

Table 4 Configurable parameters

Parameters	UART	
	Description	**Default setting**
Communication interface	UART, I^2C and I/O	UART
Frame rate	Adjustable,1~1000Hz	100Hz
Baud rate	Adjustable,9600~921600bps	115200bps
Reset to default	Reset all the settings to default	/

6. Product Certification

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Benewake:

TF02 Pro Uart TF02 Pro I2C

TFmini Plus single-point ranging LiDAR

1. Product Description

TFmini Plus is a milestone of Benewake in the process of promoting the cost-effective -LiDAR. Apart from low-cost, small-size and low-power-consumption, TFmini Plus also improves the frame rate, introduces IP65 enclosures and optimizes various compensation algorithms. These new characters greatly expand the application fields and scenarios of TFmini Plus.

2. Technical Specifications and Parameters

Table 1 Main characteristic parameters of TFmini Plus

Parameter		Value
Product parameters	Operating Range	0.1m~12m①
	Accuracy	±5cm@ (0.1-6m)
		±1%@ (6m-12m)
	Distance resolution	5mm
	Ambient light immunity	70klux
	Operating temperature	-20~60°C
	Frame rate	1~1000Hz（adjustable）②
	Enclosure rating	IP65
Optical parameters	Light source	LED
	Central wavelength	850nm
	FOV	3.6°③
Electrical parameters	Supply voltage	5V±0.5V
	Average current	≤110mA
	Power consumption	550mW(low power mode
	Peak current	140mA
	Communication level	UART，I2C，I/O
Miscellaneous	Wire length	30cm
	Material of enclosure	ABS+PC
	Weight	12g
	Storage temperature	-20°C~75°C

① *Range based on a standard whiteboard with reflectivity 90% in indoor condition*

② *Only frame rates meet the formula – 1000/n (n is positive integer) can be set.*

③ *This is the theoretical number,the is some offset for the real number.*

3. Product Appearance and Structure

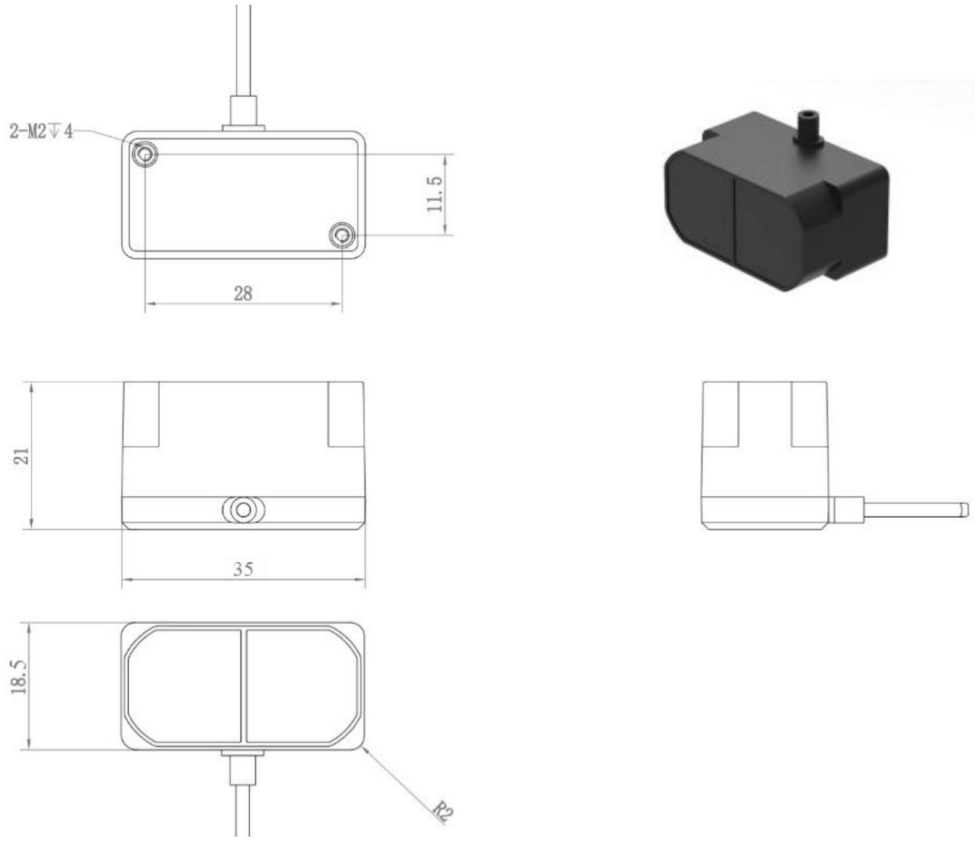


Fig. 1 Dimensions of TFmini Plus module

4. Wiring Guide

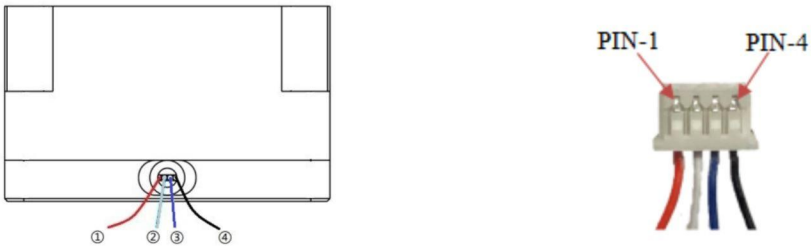


Fig. 2 Wiring diagram of TFmini Plus

No.	Color	Corresponding PIN	PIN	Function
①	Red	PIN-1	+5V	Power supply
②	White	PIN-2	RXD/SDA	Receiving/Data
③	Blue/Green	PIN-3	TXD/SCL/IO	Transmitting/Clock/IO
④	Black	PIN-4	GND	Ground

5. Communication Protocol

Table 2 Communication Protocol--UART

Communication port	UART
Default Baud rate	115200
Data bits	8
Stop bit	1
Parity	None

Table 3 Communication Protocol--I^2C

Communication port	I^2C
Max transmission rate	400kbps
Master/Slave mode	Slave
Default address	0x10
Address range	0x01~0x7F

6. Data format

The data frame contains 9 bytes, 2 bytes of frame head, 2 bytes of distance value (Dist_L and Dist_H), 2 bytes of signal strength (Strength_L and Strength_H), 2 bytes of temperature (Temp_L and Temp_H) and 1byte of checksum. All the data and commands are transmitted in hexadecimal format. See the Table 4 for more details.

Table 4 Standard Data Code Format and Description

Byte0-1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
0x59 59	Dist_L	Dist_H	Strength_L	Strength_H	Temp_L	Temp_H	Checksum
Data code explanation							
Byte0	0x59, frame header, same for each frame						
Byte1	0x59, frame header, same for each frame						
Byte2	Dist_L distance value lower by 8 bits						
Byte3	Dist_L distance value higher by 8 bits						
Byte4	Strength_L low 8 bits						
Byte5	Strength_L high 8 bits						
Byte6	Temp_L low 8 bits (suit for version later than V1.3.0)						
Byte7	Temp_H high 8 bits (suit for version later than V1.3.0)						
Byte8	Checksum is the low 8 bits of the cumulative sum of the numbers of the first 8 bytes.						

7. Configurable parameters

Table 5 Configurable parameters list

Configurable item	Description	Factory setting
Comunication interface	UART,I^2C and I/O	UART
Frame rate	1~1000Hz	100Hz
Baud rate setting	9600~921600bps	115200
Trigger source	Measure automatically or by trigger	auto
Reset to factory	All of setting reset to factory	/

8. Common configuration commands

8.1 Convention

(1) Little endian transmission has been applied in multi byte data,i.e. low byte of data will be saved in lower address

(2) Downlink frame:data from master computer to LiDAR

(3) Uplink frame: data from LiDAR to master computer or other terminal

8.2 Frame definition

Byte	0	1	2	3~Len-2	Len-1
Description	Head	Len	ID	Payload	Checksum

Head: frame head of command frame(0x5A)

Len: length of the frame, head and checksum included

ID: identifier code of command

Payload: data segment. **Little endian format**

Checksum: sum of all bytes from Head to payload. Lower 8 bits.

8.3 Commands

Table 6 Common commands of TFmini Plus

Commands	Downlink frame	Uplink frame	Description
Obtain firmware version	5A 04 01 5F	5A 07 01 01 02 03 SU	Represent V3.2.1
System reset	5A 04 02 60	5A 05 02 00 SU	00-Succeeded 01-Failed
Set update rate	5A 06 03 00 00 SU	5A 06 03 00 00 SU	Set Frame rate (1~1000Hz)①
Set measurement unit	5A 05 05 01 SU	5A 05 05 01 SU	01-cm 06-mm
Set baud rate	5A 08 06 00 00 00 00 SU	5A 08 06 00 00 00 00 SU	Set baud rate②
Enable/Disable output	5A 05 07 00 SU	5A 05 07 00 SU	0-Disable 1-Enable
Communication interface mode	5A 05 0A MODE SU	/	MODE 0：UART，1：IIC

Modify IIC slave address	5A 05 0B ADDR SU	5A 05 0B ADDR SU	Change the I^2C slave address(default 0x10)
Obtain data frame	5A 05 00 01 60	Data Frame(9 Bytes – cm)	Only works in I^2C Mode
	5A 05 00 06 65	Data Frame(9 Bytes – mm)	
Restore factory settings	5A 04 10 6E	5A 05 10 00 SU	00-Succeeded 01-Failed
Save settings③	5A 04 11 6F	5A 05 11 00 SU	00-Succeeded 01-Failed

Note:Bytes with yellow undertone represents checksum. Bytes with blue undertone represents data segment.

① The default frame rate is 100Hz. The customized frame rate should be calculated by the formula: 1000/n (n is positive integer). Data stability will decrease with frame rate increasing.

② Only standard baud rates are supported. When setting a high frame rate, a high baud rate is recommended to ensure data security.

③ Please always send the command of save settings when try to modify parameters of TFmini Plus,otherwise the settings will not take effect after power off.