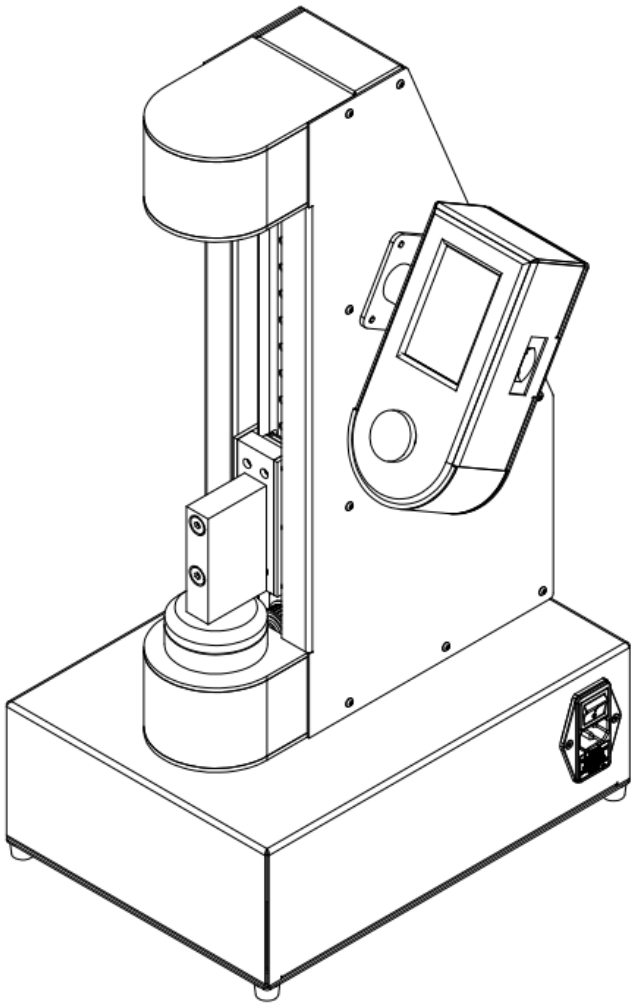
Spring Fatigue Testing Machine



OPERATOR'S MANUAL

Tristen Cutshall, Tim Hamersly

JUNE 2020

SAFETY PRECAUTIONS

Improper use of this machine can cause mild to severe injuries including but not limited to pinches, cuts, bruising and eye damage. To prevent such injuries operators should be well versed in the information contained in this manual.

In addition to operator injury, misuse may result in damage to the machine, or a dramatic reduction in the lifespan of machine components.

Operator's shall abide by the following rules to reduce both the chance of injury and damage to the machine:

1. Always wear safety glasses when the machine is under operation.
2. Avoid wearing loose clothing, jewelry, or long hair that could be caught in the machine.
3. Never put hands or fingers within the operating zone of the machine when executing movement or performing a test.
4. Never use the machine to test a spring with dimensions greater than the spring boundary zone as shown below or attach fixtures or other peripherals that fall outside of this zone.
5. When turning on the machine ensure there are no items within the operating zone of the machine and the testing arm is in the lowest possible position.
6. Never leave the machine in continuous operation without supervision.
7. When performing maintenance or making program modifications, always ensure the machine is in the OFF state, and unplugged from wall power.
8. Place the machine on a stable, flat surface free from items that could be entrapped in the bottom fan.
9. Ensure the EMERGENCY STOP button is always free from obstructions and easily accessible .

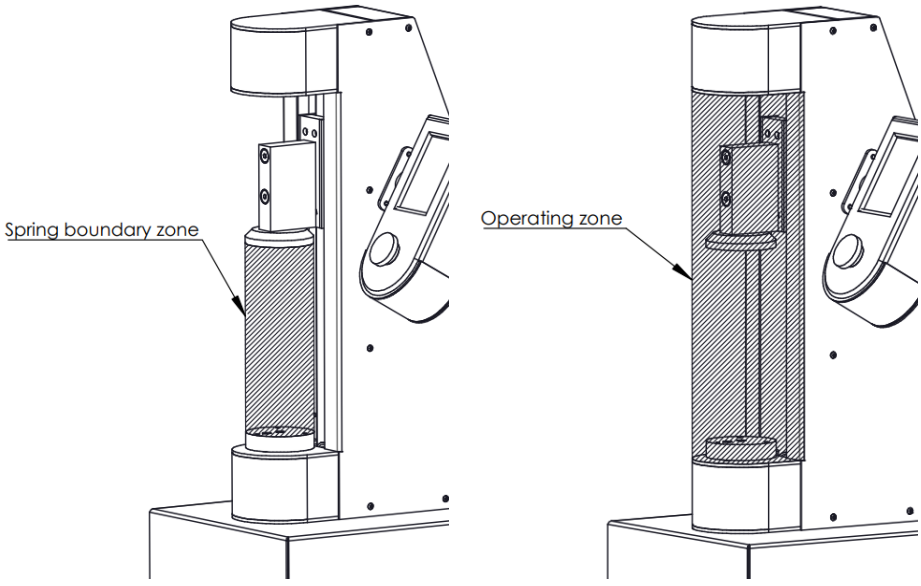


TABLE OF CONTENTS

1 MECHANICAL SYSTEM OVERVIEW

1.1 DRIVETRAIN

The testing machine is driven by a NEMA 23 framed 400W servo motor from DMM Technologies Inc. The motor shaft is 14mm in diameter with a 5mm keyed slot. A 40 tooth, 9mm wide, stepped GT3 timing pulley with 90° opposed setscrews is attached to the motor shaft.

The rear belt is a 252 tooth, 9mm wide GT3 timing belt that connects the motor shaft to the lower bearing assembly. The rear-facing pulley on the lower bearing assembly is a 50 tooth, 9mm wide, GT3 timing pulley with a 5mm keyed slot and 90° opposed setscrews. Internal to the bearing assembly are two Class 0 double shielded deep groove radial ball bearings 32mmOD x 12mmID x 10mmW. The rear facing side of the lower bearing assembly shaft is 10mm in diameter with a 5mm key slot. The front facing side of the shaft is 8mm in diameter with a 3mm key slot. The front facing pulley is a 9mm wide GT3 profile timing pulley with 28 teeth, a 3mm key slot and 90° opposed setscrews. The front drive belt is a 9mm wide GT3 profile timing belt with 900 teeth that has been cut to length to fit the front assembly. Should the user ever need to replace the front belt, any section of 9mm GT3 timing belt will work as a replacement if it is at least 2700mm in length. The upper bearing assembly has identical parts to that of the lower bearing assembly minus the rear facing pulley and shorter rear shaft.

Major drive train components and their respective part numbers are shown below.

1.2 CARRIAGE ARM

The carriage arm is attached to two linear bearing blocks that slide on a 9mm wide single row, low-precision linear rail. The linear rail should be kept free from dirt and debris and greased with white lithium grease after prolonged use.

1.3 LOAD PLATES

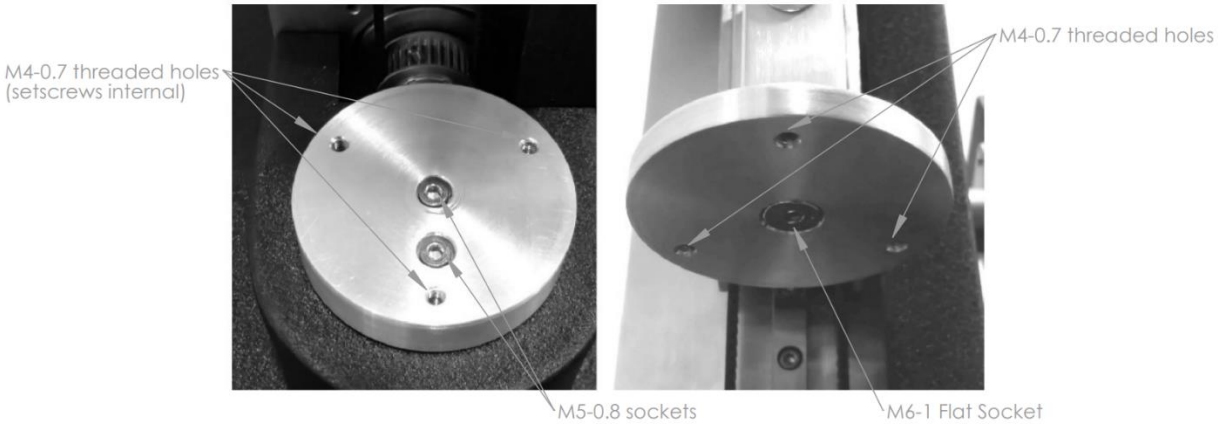
The load plates are the two cylindrical plates that touch the top and bottom faces of the spring when testing. The upper load plate is attached to the carriage arm. The lower load plate is attached to a 10kg loadcell which in turn attaches to the main frame of the machine.

The upper and lower load plates each have three M4-0.7 threaded holes that allow the user to easily attach spring fixturing.

The lower load plate is attached to the loadcell with M5-0.8 socket cap screws. These screws can be loosened to adjust the alignment of the upper and lower load plate. In addition to providing a means to attach fixturing, the three M4-0.7 threaded holes on the lower load plate have setscrews that protect the loadcell from overloading. The setscrews are currently set to contact the base with an applied load of 50N, however these may need to be adjusted after prolonged use.

****Note**** – These setscrews have Loctite applied and care will need to be taken when adjusting these to avoid stripped threads.

The upper load plate attaches to the carriage arm through a M6-1.0 flat head socket cap screw. This screw can be loosened to adjust the rotation of the upper fixture relative to that of the lower fixture.

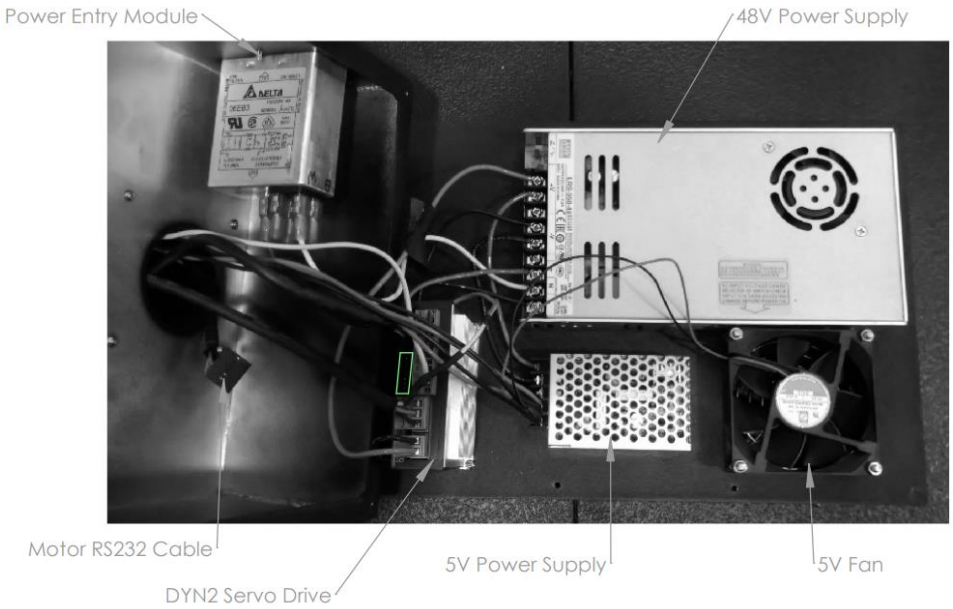


2 ELECTRICAL SYSTEM OVERVIEW

2.1 LOWER PANEL

OVERVIEW

The lower panel contains the machine's high voltage circuitry. This includes an IEC-C14 power entry module, 110VAC-48VDC-350W power supply, 48VDC-5VDC power supply, the servo motor drive, and the enclosure fan.



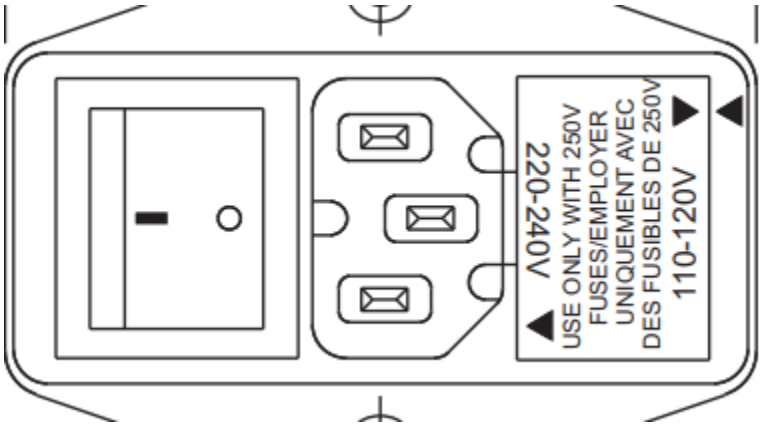
LOWER PANEL ACCESS

The following steps should be taken should the user ever wish to access this panel:

1. Disconnect power from the machine.
2. Lay the machine on the back side of the column such that the vent slots contact the worktable.
3. Remove the 10 M3-0.7 screws on the outside edge of the panel as shown below.
4. Partially open the panel by separating the front edge of the panel from the box. being careful not to suspend the weight of the panel by any cables.
5. Unplug the motor communication cable from the motor drive socket outlined in the green rectangle above.
6. Lay the panel in front of the base of the machine as shown above.

POWER ENTRY MODULE

The power entry module to the machine contains two IEC 5 x 20mm 3A cartridge fuses for over-current protection. These fuses are undersized and should be replaced with 7A fuses that match the maximum power supply current if they ever fail. If these fuses are replaced, be careful to line up the 110-120V fuse box arrow with the bottom arrow as shown below.



2.2 CONTROL PANEL

OVERVIEW

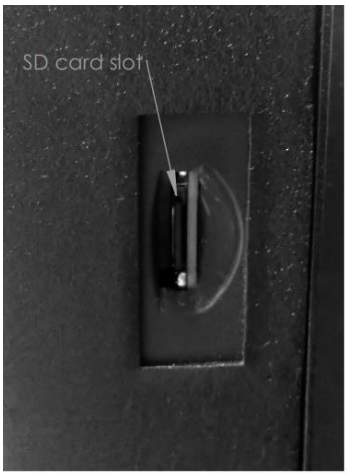
The control panel houses the machine's low voltage circuitry: the Adafruit M0 Adalogger, the HX711 loadcell amplifier board, the 3.5" TFT resistive touchscreen, a logic-level shifter board, and the emergency stop button.

MICRO SD CARD

The microSD card can be easily removed from the side of the control panel. Only the microSD card provided with the machine should be used. Should the user wish to use an alternate micro SD card, the EWU logo will no longer be displayed unless the user copies the EWU image to the alternate card. The image file must be no larger than 320px by 320px and with a 24-bit color depth.

> ****NOTE**** – The EWU logo is a hidden file on the SD Card and can only be accessed through File Explorer on Windows by selecting View -> Options -> Change folder and search options -> View Tab -> Advanced Settings -> Show hidden files, folders, and drives -> OK.

Alternate microSD cards need to be formatted to an Arduino friendly format using the SD Card Formatter application made by Tuxera. Using the "Overwrite Format" option is best. Link to the SD Card Formatter application download page here: https://www.sdcard.org/downloads/formatter/index.html

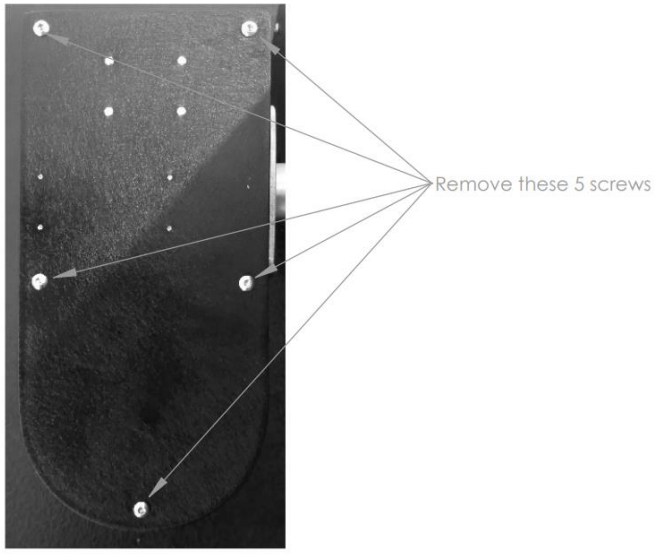


EMERGENCY STOP BUTTON

The emergency stop button has two NC contacts. One set of contacts provides a 3.3V HIGH to the A0 pin of the Arduino and the Arduino senses when this pin goes LOW when the e-stop is triggered. The other contact provides LOW to the motor drive disable pin. A 10k pull-up resistor is also wired to the motor drive disable pin so when the e-stop is triggered, the drive disable pin goes high (drive disable is active HIGH). The motor shaft will spin freely when the drive is disabled and the carriage will be free to move up and down.

CONTROL PANEL ACCESS

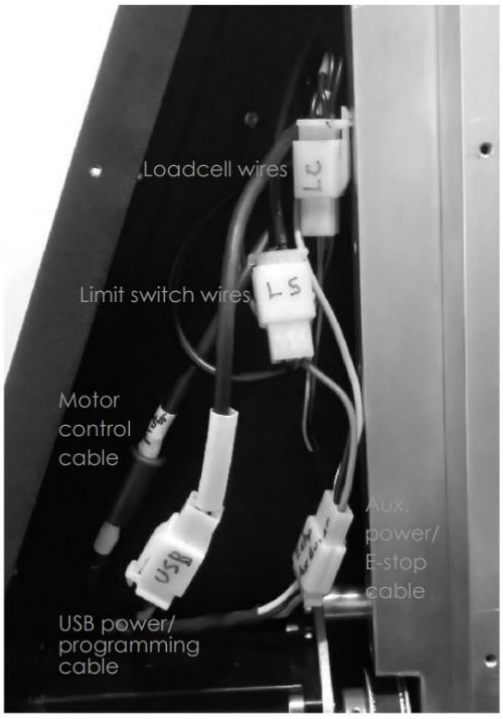
Should the user wish to access the control panel to perform a reset on the device, 5 screws must be removed from the back panel. The utmost care must be taken when doing this to avoid breaking soldered connections.



2.3 CABLES AND CONNECTIONS

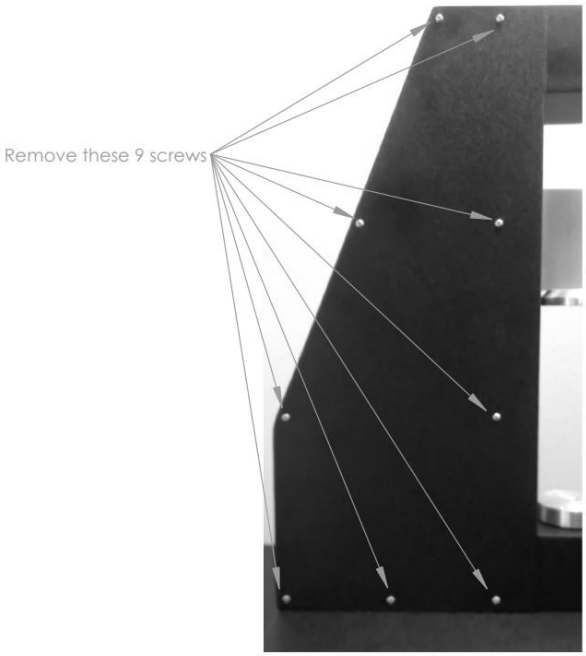
OVERVIEW

There are 5 cables (and bundles of wires) that come from the bottom panel: the motor power cable, the motor encoder cable, the motor control cable, the auxiliary power/e-stop wires, and the USB power wires for the Arduino. There are 5 cables (and bundles of wires) that come from the control panel: the motor control cable, the auxiliary power/e-stop wires, the USB power/programming cable, the load cell wires, and the limit switch wires. These cables are labeled at their connectors and shown in the image below.



CONNECTIONS ACCESS

The connections between the control panel, the bottom panel, the inductive proximity sensor, and the load cell all meet in the column of the machine. To access these connections, remove the 9 screws that retain the left side panel shown below.



LOADCELL WIRES

The load cell has 4 wires that route through the frame of the machine and terminate in a 4-pin female quick connector. There are 4 wires that meet these in a 4-pin male quick connector that come from the loadcell amplifier board internal to the control panel.

LIMIT SWITCH WIRES

The limit switch has 3 wires (power, ground, and signal) that route through the frame of the machine and terminate in a 3-pin female quick connector. There are 3 wires that meet these in a 3-pin male quick connector that come from the control panel.

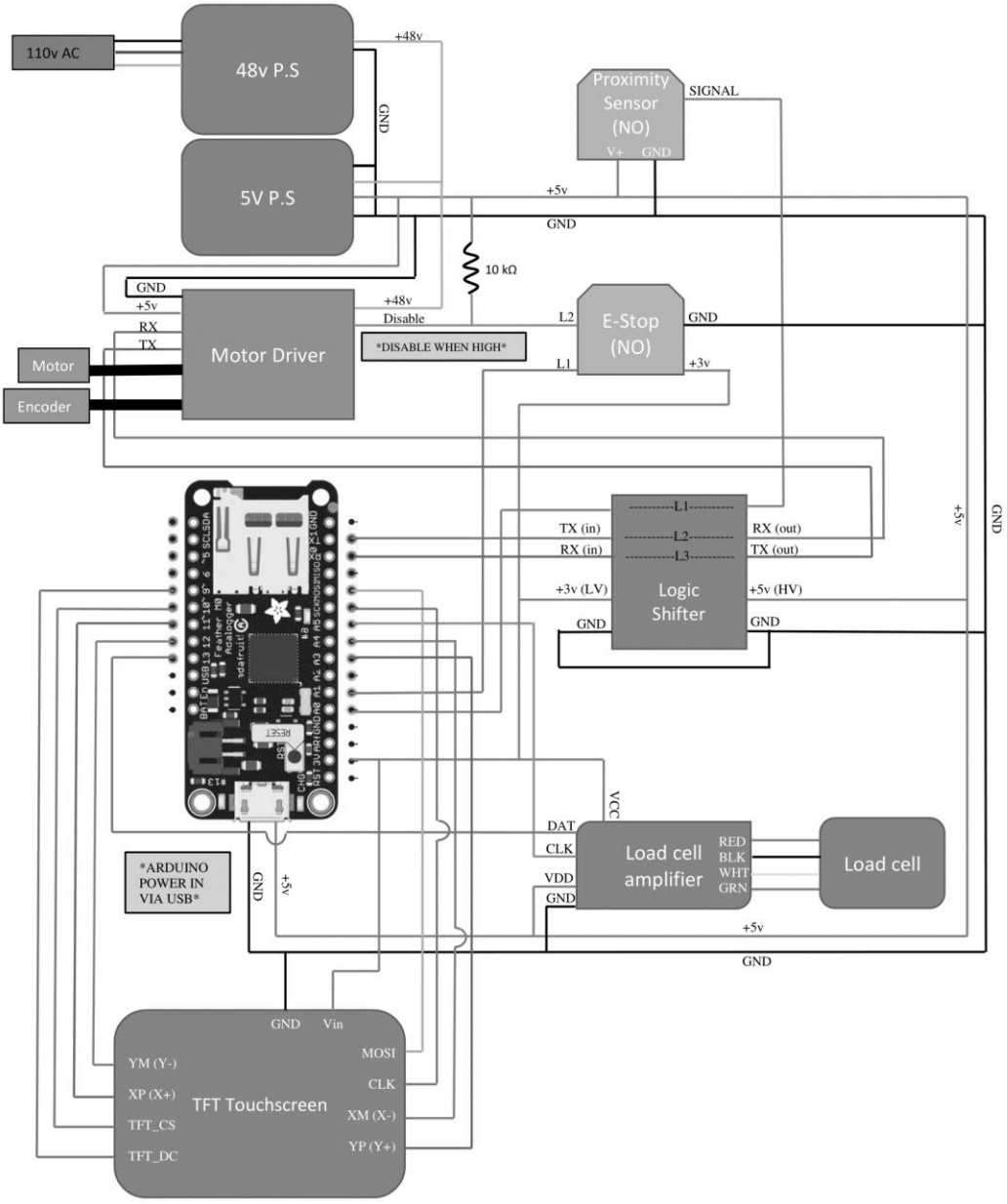
AUX. POWER / E-STOP WIRES

The aux. power/e-stop wires come from the bottom panel. The aux. power wires provide 5V power to the loadcell amplifier board, the proximity sensor, and the logic level shifter. The e-stop wire is yellow, and the power and ground wires are red and black respectively.

USB POWER / PROGRAMMING WIRES

There are 2 USB power wires that come from the bottom panel and terminate in a 4-pin male quick connector. These provide 5V power to the Arduino that steps it down to 3.3V with a regulator internal to the board when the machine is not being programmed. These wires meet a 4-pin female quick connector that has power, ground, TX, and RX wires that terminate in a male micro USB cable that directly powers the Arduino and allows for programming.

2.4 WIRING DIAGRAM



3 OPERATION

3.1 STARTUP

1. Turn on the machine using the switch on the back right of the bottom housing.
2. Wait for the startup sequence to begin. Wait until the motor has finished moving and remains at the top position. This will take about 30 seconds. The screen will now go to the **Main** menu.

    ****Note**** – The startup sequence of the machine establishes the maximum and minimum boundaries of machine travel. These boundaries are not stored in memory and must be re-established every time the machine is turned on. The carriage arm first moves up by a small amount to unload the scale and then moves down very slowly until it contacts the lower load plate and a sufficient force is registered. The machine then stores this position as the lower boundary of machine travel. It then moves up until the upper limit switch is triggered and stores this position as the upper boundary of machine travel. If the machine is jarred during this time or something touches the scale to cause a sufficient force reading before the carriage arm reaches the lower load plate, a false lower boundary will be established and the machine must be reset.

3.2 TEST

1. From the **Main** menu, navigate to the **Jog** menu and select **Auto**. This will auto-detect the top of the spring for use in the testing phase. Follow prompts on the screen and place the spring on the bottom load plate. Once the spring is on the load plate, select **Ok** and the auto calibration will begin.

    ****Note**** – The **Auto** function works similar to the startup sequence outlined above. The machine moves down slowly until the scale registers 0.1N of force, it then moves up until it registers 0N force and stores this position as the spring top. The auto function must be used to store the spring top position before testing. The machine will refer to this position as the spring top for subsequent tests. If springs of different length are tested, or the machine power is cycled, the Auto function must be executed again to reinstate the spring top position.

2. After the auto calibration of the spring the screen will automatically return to the **Main** menu.
3. Navigate to the **Test** menu.
4. Select desired limits of force (**Fy**), displacement (**Uy**) and number of cycles (**N**). Select **+** or **–** to increment or decrement the selected value. Select **Save.**

****Note**** – The system will use whichever limit (Fy or Uy) it detects first to establish the lower spring boundary, so if a certain Uy is desired, set a much higher Fy than is expected for the given Uy. If a certain Fy is desired, set a much higher Uy than is expected for the given Fy.

If N = 10^0, a single cycle spring-rate test will commence. The machine will move down in 0.1mm increments and record the force registered by the scale. Once Fy or Uy is reached, the machine will store the spring bottom position and the machine will move up in 0.1mm increments until it is back at the spring top position. Since the motion of the machine occurs in 0.1mm steps, the machine may move beyond the preset Fy value.

If N > 10^0, a slow single cycle spring-rate test will commence to ensure the machine will not crash. For the following cycles, the machine will move between the spring top position and spring bottom position and record the force registered by the scale at these postions.

5. Once **Fy**, **Uy**, and **N** have all been set, a **Begin** button should appear.
6. Make sure the spring is within the spring boundary zone and the operating zone is clear.
7. Select **Begin**.
8. Once the test has finished, select **Back** to return to the **Main** menu.

3.3 MANUAL JOG

1. From the **Main** menu, navigate to the **Jog** menu.
2. Press and hold **UP** or **DOWN** to jog UP or DOWN.
   Upon release of either button the motor will STOP. If for any reason, the motor does not STOP, the **STOP** button will stop the motor.
3. The **Back** button will return to the **Main** menu.

3.4 FILES MENU / DELETE FILES

1. From the **Main** menu, navigate to the **Files** menu. This will display the current list of files on the SD card.
2. To delete one file, select **Delete**.
   a. The **Select** button will select the next file on the list.
   b. Press **Delete** to delete the file.
3. To delete all files on the SD card, select **Delete All**.

4 PROGRAMMING

START HERE

The programming for this machine was done in C/C++ using Arduino IDE. The GitHub repository for our code can be found here:

MASTER CODE DOWNLOAD:

https://github.com/3dprintedspringsEWU2020/testingmachine

4.1 REQUIRED SUPPORTING SOFTWARE

To reprogram the Arduino, you must first install the required **board support package** and **libraries** as well as the **Arduino IDE**.

ARDUINO IDE

Go to https://www.arduino.cc/en/main/software and download your preferred version of the Arduino IDE software.

BOARD SUPPORT PACKAGE:

Since our Arduino is produced by Adafruit, board support for “Adafruit SAMD Boards” which include our board, the Adafruit Feather M0, need to be installed using Arduino IDE’s board manager. To install, open Arduino IDE and go to:

Tools -> Board: boardname -> Board Manager -> Search: “Adafruit SAMD Boards”.

Click install.

INSTALL LIBRARIES:

Our code uses the following Arduino libraries:

SPI.h	serial protocol for use with touchscreen
Adafruit_GFX.h	library for buttons, and colors
Adafruit_HX8357.h	HX8357 - display specific functions
TouchScreen.h -	Touchscreen functionality
SD.h -	SD functionality
Stdlib.h -	Standard math functions
HX711.h -	Functionality for the HX711 load cell

These libraries must be installed in the same directory as Arduino IDE. Arduino will automatically do this if the libraries are installed using the library manager.

Tools-> Manage Libraries…

Search for each library name as listed above and click install.

> ****NOTE**** – A minor modification must be made to the TouchScreen.h library to allow for reliable button presses with our touchscreen. THIS MODIFICATION IS REQUIRED FOR STABLE MACHINE OPERATION.

1. Navigate to your TouchScreen.cpp file. The default install directory is under: C:\...\...\...\Documents\Arduino\libraries\Adafruit_TouchScreen\TouchScreen.cpp
2. Open TouchScreen.cpp with any text editor, Notepad is preferred.
3. Look for the line `#define NUMSAMPLES 2` and change it to `#define NUMSAMPLES 5`
4. Save the modified file and close.

The touchscreen library averages the result of multiple touchscreen readings to determine the X and Y coordinates of the touch event. Changing this line results in more reliable touch readings.

4.2 UPLOADING TO THE ARDUINO

After a modification to the code is complete and it has compiled without errors, the code can be uploaded to the Arduino board.

> ****NOTE**** – The board's Serial connections can interfere with the Serial USB connection to the computer during the PC-to-USB 'handshake'. For this reason, a 5 second delay has been added at the beginning of the boot-up sequence for the Arduino.

For best results, upload the code during this 5 second boot-up delay.

1. Make sure machine is powered **OFF** before continuing.
2. Remove the left side panel. (see **Section 2.3**)
3. Disconnect the USB power/programming cable.
4. Open Arduino IDE and ready the code for upload as soon as the cable is connected.
5. Plug the provided USB programming cable into the female (control panel side) of the USB power/programming cable.
6. Press **Upload** immediately upon connecting the USB cable to try and upload within the 5 second delay.
    a. If the screen boot-up sequence has already started, disconnect and try again.

7. Arduino IDE will display a “Upload Successful” message and will power cycle the Arduino CPU.
8. Unplug the USB programming cable and reconnect the upper and lower Arduino power harnesses of the machine.
9. Reinstall the left side panel.
10. Done.

4.3 TOUCHSCREEN

The display is an Adafruit HX8357 3.5” TFT Color LCD touchscreen.

The X and Y axes of the touchscreen are measured using the resistance between the X+ (XP) pin and X- (XM) pin. The Y resistance is automatically determined from the X resistance.

> ****NOTE**** – Use light to moderate pressure and hold ½ second long per button press for best touchscreen results.

The Arduino uses SPI to communicate with the HX8357. SPI pins include **SCK** or **CLK** (Serial Clock), **MISO** (Master In Slave Out) which we do not need, **MOSI** (Master Out Slave In), **DC** (Data), and **CS** (Chip Select).

See the beginning of the code for Arduino pin assignments.

All buttons except the **UP** and **DOWN** buttons in the **Jog** menu are triggered once when pressed. The **UP** and **DOWN** buttons are momentary and are triggered until release.

The display of the screen uses the Adafruit_GFX and Adafruit_HX8357 libraries.

The touchscreen functionality uses the modified Touchscreen library (see **Section 4.1**)

For documentation and more information see the **Appendix.**

4.4 LOADCELL

The loadcell is a 10kg beam loadcell that uses the HX711 loadcell amplifier board from Sparkfun.com.

The maximum mass used with this scale should not exceed 10kg (about 100N of force).

The scale has been calibrated to read in Newtons. It uses an average gravitational acceleration of 9.80665 m/s^2. The scale can be recalibrated using a known weight, and the following code:

https://github.com/sparkfun/HX711-Load-Cell-Amplifier/blob/master/firmware/SparkFun_HX711_Calibration/SparkFun_HX711_Calibration.ino

The calibration factor obtained is stored in the code in the **calibration_factor** variable.

The loadcell also uses SPI for communication.

The **scale.get_units()** command will request a reading from the load cell and scale it by the calibration factor. The **scale.tare(n)** command will tare the scale using a zero calculated from the average of **n** readings.

The **startup_scale()** function should be called before getting the scale readings. This function will begin the SPI with the scale, zero the scale, and allow for scale readings to settle before continuing. This is important because the first 5 or so readings from the scale tend to fluctuate wildly.

The output signal from the scale is routed to the amplifier which connects to the Arduino.

The maximum reading frequency that can be obtained from the amplifier board is 80Hz.

For documentation and more information see the **Appendix.**

4.5 MOTOR / MOTOR DRIVER

The motor is a 400W NEMA23 servo motor. It is controlled with a DYN2 motor driver.

The Arduino communicates to the DYN2 via DMM Technology's RS232 serial protocol.

The program uses 3 main functions provided in this code to execute movement and read motor position.

ReadMotorPosition32(char ID)
move_abs32(char ID, long pos)
move_rel32(char ID, long pos)

The **ReadMotorPosition32(char ID)** function will return the absolute position of the motor and store it in the **Motor_Pos32** variable, a signed long. The argument is the motor ID which in our case is always **0**. The motor position returned is the absolute position of the motor relative to the starting position. The starting position of the motor is always 0 regardless of the motor shaft position. One full rotation of the shaft represents a value of 65,536. The limits of this variable are -134,217,728 : 134,217,727 so the motor drive knows the absolute position within 2048 rotations clockwise and 2048 rotations counterclockwise. Calling this command continuously may disrupt the RS232 communications and freeze the Arduino. Use native Arduino command **delay(20)** to

allow for a 20 millisecond delay. Use this between calling this function and using the motor position variable **Motor_Pos32** to allow time for the value to update.

> ****NOTE**** The values obtained from **ReadMotorPosition32** and movement commands **move_rel32** / **move_abs32** differ by a factor of 4. For this reason, values obtained from **ReadMotorPosition32()** MUST BE DIVIDED BY 4 before being used in subsequent movement commands.

The **Turn_const_speed(char ID, long rpm)** function SHOULD NEVER BE USED. This function will effectively reset the origin of the encoder to the position when the function was stopped.

The **move_rel32(char ID, long pos)** function moves a relative number of steps in either direction from the current shaft position. The value of **pos** here is signed. A value of 16,384 represents one full shaft rotation. Positive is UP, and negative is DOWN. The motor drive dynamically updates the target position. This means if **move_rel32** is called before motion for a previous movement command is finished, the drive will only move relative to the current shaft position.

The command **move_rel32(0, 0)** is used to STOP the motor.

The **move_abs32(char ID, long pos)** function moves to an absolute position **pos**. The starting position of the motor is always 0 regardless of the motor shaft position. Our machine records position relative to the top and bottom limits so the location of the starting position does not matter. One full rotation of the shaft represents a value of 16384.

> ****NOTE**** For both **move** functions, the motor will move at the max speed and acceleration set in the DYN2 setup. To change various DYN2 settings including the max acceleration, max_speed, and PID gain settings, the DYN2 setup can done using the DYN2-to-USB tuning cable provided by DMM Tech (must be purchased) and downloading the DMMDRV installer package here: http://www.dmm-tech.com/Downloads.html. Changing settings requires unplugging the motor-to-driver control cable and plugging in the USB tuning cable.

For documentation and more information see the **Appendix**.

4.6 SD CARD STORAGE

SD card files are opened using the line

dataFile = SD.open(filename, O_CREAT | O_APPEND | O_WRITE);

If the filename does not exist, a new file is created. Writing information to the SD card is done using the line

dataFile.print(data, int precision);

The value of **precision** determines the number of digits after the decimal point.

Use the line

dataFile.print(',');

To change columns (.csv = comma separated values)

And the line

dataFile.print('\n');

To change rows

Always use

dataFile.flush();

After writing a set of data and

dataFile.close();

After finishing writing to a file. For more information see the **Appendix.**

4.7 PROGRAM STRUCTURE

The program's main **setup()** function is executed first. This function initializes serial communication, scale readings, and the touchscreen. The EWU logo is drawn on the screen along with various messages before calling the **motor_calibrate()** function.

The **motor_calibrate()** function executes motion and scale reading functions to establish the lower and upper limits of machine travel. The lower limit is stored in the **bottom_limit** variable. The upper limit is stored in the **top_limit** variable.

The program then commences execution of the **loop()** function. This function displays the main menu buttons and scans for button presses. If the JOG button is pressed, function **process_jog_menu()** is called, if the TEST button is pressed, **process_test_menu()** is called, if the FILE button is pressed, **displayfiles()** is called.

In **process_jog_menu()** the jog menu buttons are initialized and button presses are continuously scanned for in a loop. LOW, MEDIUM, and HIGH buttons change the **jog_speed** variable to the presets defined in the top of the code LOWSPD, MEDIUMSPD, and HIGHSPD respectively. The UP and DOWN buttons call **ReadMotorPosition32()** continuously to ensure the position is within the upper and lower boundaries, and call **move_rel32(0, jog_speed)** continuously with a small delay to move incrementally up and down. In this way, LOW, MEDIUM, and HIGH speeds

dictate the step size that the machine moves at, which in turn effects the average velocity of the movement.

In **process_test_menu()** buttons are initialized to set the force, displacement, and cycle count. To change the values of these variables, **process_mod_value_menu()** is called to display the “+”, “-” and “SAVE” buttons. The max displacement is stored in the **DISP** variable. The max force is stored in the **FORCE** variable. The number of cycles is stored in the **num_cycles** variable. If all 3 parameters are set, the TEST button is displayed, and upon pressing the TEST button, the **TESTING()** function is called.

The **TESTING()** function creates a new file, prints data to the screen, and prints the .csv heading labels. The machine moves to the top of the spring position if it is not already there. The static test is executed and then the fatigue test is executed under the condition that **N** (number of elapsed cycles) < **num_cycles**. **N** is incremented after each cycle. When the machine is executing its downward movement for the static test, the machine moves down until either **Fy** or **Uy** is reached as mentioned in **Section 3.2** above. Once either of these conditions is met, the position is stored in the **spring_bottom** variable.

The **process_auto_setup()** function handles the function to locate the top of the spring. The top of the spring gets stored in the **spring_top** variable.

Numerous secondary functions are executed within the main functions outlined above.

5 MAINTENANCE

BEARINGS

The bearings should be checked for signs of wear about every 10^{14} cycles. Signs of uneven wear include an audible scraping sound or visual “sticky spots” during their rotation.

The bearings used in this machine are pre-packed with grease and double shielded, so no greasing should be necessary.

Both upper and lower bearings can be replaced with B6201ZZ bearings from MISUMI or any suitable bearings that match these bearings dimensions. This requires removing the upper and lower bearing assemblies from the machine frame.

LINEAR RAIL

The rail should be inspected for any signs of uneven wear about every 10^{14} cycles.

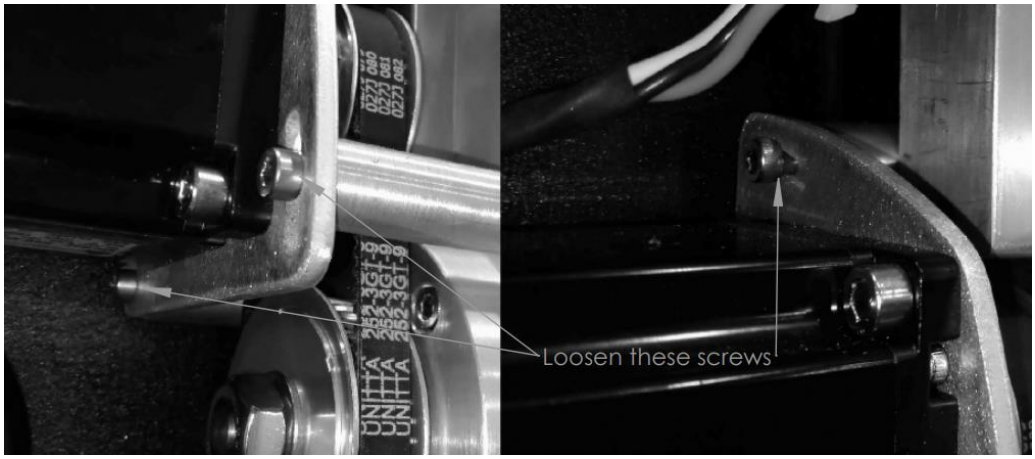
Apply a dab of white lithium grease to the linear rail and slide the carriage blocks along it every 10^{6} cycles.

The rail and carriage blocks can be replaced with SE2BSZ10-275 linear rail and carriages from MISUMI. To replace the linear rail remove the front belt and and the 14 M3-0.7 socket screws that attach the rail to the machine’s frame.

PULLEYS AND BELTS

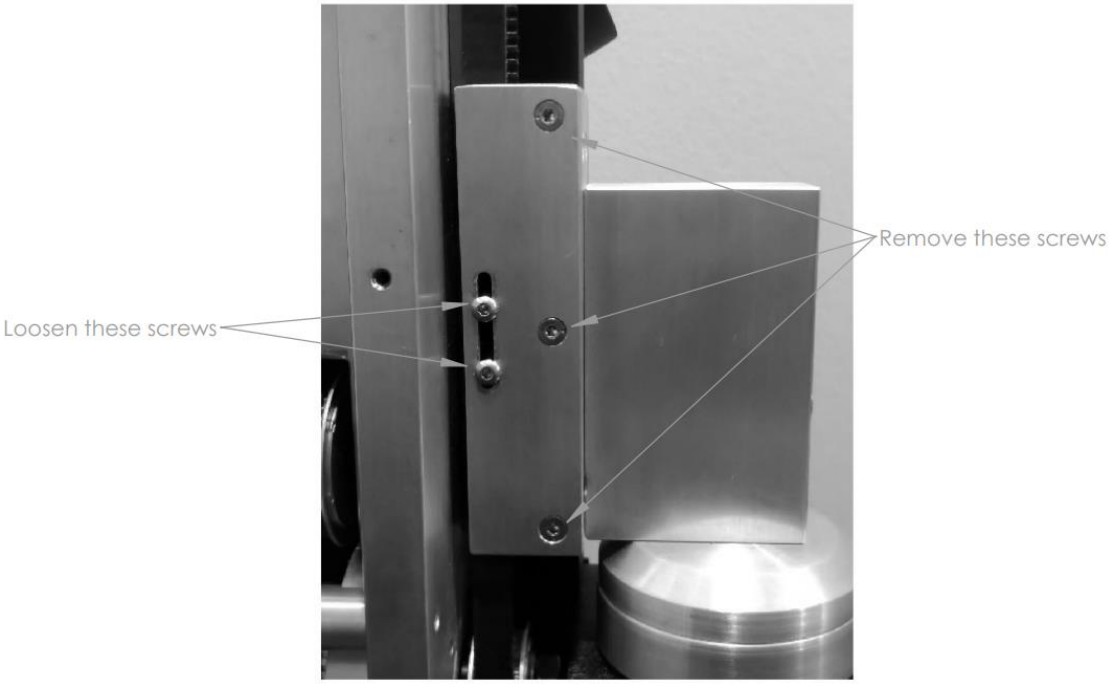
All pulleys and belts should be inspected for wear every 10^{14} cycles. Symptoms may include slipping belts, slipping pulleys or uneven wear on other parts.

The rear drive belt can be replaced or tensioned by loosening the motor standoff screws shown below. Complete removal of these screws may be easiest to replace the rear belt.



The front linear-rail-side belt can be replaced by removing the 3 screws on the left side of the carriage shown below to free the left side carriage plate from the carriage and loosening the belt clamp screws shown below. To adjust the front belt tension, loosen the belt clamp screws, squeeze the screws together, and retighten the screws.

Should the belt clamps ever need replacement, completely remove the belt clamp screws and reinstall new belt clamps. See image below for belt clamp removal.



6 TROUBLESHOOTING

The most notable problem we came across when putting the machine into service was noise being generated in the motor control cable when the motor was under load. This caused the **ReadMotorPosition32()** command to override the Arduino with garbage serial data and caused the Arduino to freeze. We fixed this problem by permanently soldering the motor control cable connection and adding a ferrite bead over the cable. This seems to have fixed the problem; however this will be the most likely source of machine malfunctions in the future. This only occurred when **ReadMotorPosition32()** was continuously being called in a loop. If the Arduino freezes randomly check to make sure this cable is not touching the motor housing.

Tips for troubleshooting the machine:

TOUCHSCREEN

Use light to moderate pressure and hold ½ second long per button press for best touchscreen results.

- Screen will not turn on:
  - Make sure Emergency Stop Button is reset (pulled out).
  - Make sure Arduino power/USB programming cable is plugged in.
  - Check continuity on all relevant wires from Arduino to touchscreen.
- Screen freezes mid-operation
  - Power cycle the machine
- Buttons flicker on and off
  - Increase the number of touchscreen samples outlined in **Section 4.1**

ARDUINO

- Arduino not recognized on COM port
  - Check programming cable connections.
  - Remove the control panel and double-press the reset button on the Arduino board.
- Arduino will not turn on
  - Make sure Emergency Stop Button is reset (pulled out).
  - Check Arduino power/USB programming cable is plugged in.
  - Check continuity on all Arduino power wires.
  - Check voltage from 5v power supply.
- Arduino freezes
  - Power cycle the machine
- Arduino not logging data

    - Ensure the microSD card:
        - is plugged in correctly
        - has free space
        - is properly formatted
    - Ensure the SPI pins used by the SD card are the same as specified in the master code
- Arduino not detecting limit switch
    - Check limit switch indicator lights to see if it has power
    - Check limit switch wires and connector
    - Cross reference code with master code
        - Check for proper pinMode()
        - Check for proper pin state

LOADCELL

- Loadcell does not read at all
    - Check loadcell pins to Arduino for discontinuity
- Loadcell is not reading correct values
    - Check loadcell readouts USB with the USB programming cable and using the calibration sketch (see **Section 4.4**).
    - Make sure the set screws on the bottom load plate are not contacting the frame of the machine.

MOTOR

- Motor not moving
    - Motor freewheeling
        - Check Emergency Stop button is reset (pulled out).
    - Motor not freewheeling
        - Check all connections to and from motor driver
        - Check continuity on wires to Arduino from driver
        - Check motor is functional using PC tuning cable and DMMDRV software (see **NOTE** in **Section 4.5**).
- Motor moving, but not to correct position
    - Did you use the **Turn_const_speed()** command? If so, replace this with the use of **move_rel32** and **move_abs32**. Never use **Turn_const_speed()**.
    - Cross reference motor movement with master code
- Motor stops mid-test
    - Possible that one of the built-in motor-driver disable's was activated
        - Motor can be disabled by torque overload, heat overload, speed too high and other factors. See supporting DYN2 documentation in the **Appendix** for details.
    - Power cycle the machine

LIMIT SWITCH

- Limit switch never triggers
    - Check for small green power light.
        - If no green light, switch does not have power
            - Check limit switch power wires and LS connector
        - If yes light, maybe Arduino is not detecting switch
            - Check for red detection light upon switch trigger
            - Check code
            - Cross reference with master code

APPENDIX

REFERENCE DOCUMENTS

ARDUINO

https://www.adafruit.com/product/2796

https://learn.adafruit.com/adafruit-feather-m0-adalogger/downloads

LOADCELL

https://cdn.sparkfun.com/assets/learn_tutorials/5/4/6/hx711F_EN.pdf

https://github.com/sparkfun/HX711-Load-Cell-Amplifier/tree/master/firmware

https://github.com/sparkfun/HX711-Load-Cell-Amplifier/tree/master/firmware/SparkFun_HX711_Calibration

TOUCHSCREEN

https://github.com/adafruit/Adafruit_HX8357_Library

https://www.adafruit.com/product/2050

MOTOR DRIVER

http://www.dmm-tech.com/Dyn2_v2.html

LIMIT SWITCH

https://yaboo.pl/data/include/cms/sn04.pdf

SD CARD

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

48V POWER SUPPLY

https://www.meanwell.com/Upload/PDF/LRS-350/LRS-350-SPEC.PDF

5V POWER SUPPLY

https://www.meanwell.com/Upload/PDF/SD-15/SD-15-SPEC.PDF

SOFTWARE

Master code https://github.com/3dprintedspringsEWU2020/testingmachine

Arduino IDE https://www.arduino.cc/en/main/software

SD card formatter https://www.sdcard.org/downloads/formatter/index.html

DMMDRV (DYN2 tuning software) http://www.dmm-tech.com/Downloads.html