README - Prony Brake Dynamometer

Team Cambot (Brandon Bravo, Miguel Jiminez Gomez, Shifan Liu, Hayden Webb)

Overview:

The Rice Robotics Club (RRC) relies on precise torque and speed measurements from the motors they source from manufacturers to optimize their design projects. However, the manufacturer's specifications often differ from actual performance in robotic systems for their various projects. To address this, RRC requires an accurate dynamometer for testing their sourced motors to confirm the validity of its output. Unfortunately, high-quality dynamometer sensors are prohibitively expensive, with costs ranging between $3,500 and $5,000. The goal of this project is to adapt a generalizable low-cost, force-torque sensor design to help RRC validate motor specifications effectively that can be utilized on various motors.

Currently, RRC lacks a suitable sensor for measuring the power and speed of the motors they order online. To overcome this limitation, the team has identified the concept of a "Prony Brake" as a viable solution. This approach combines torque measurement generated from a load cell with speed detection via a photo sensor to accurately calculate motor power. Given the diverse scale of RRC’s projects, the proposed design must be adaptable to meet the varying needs of different teams through modifiable electronics and motor mounts. The developed torquemeter will offer a cost-effective and scalable solution for RRC to evaluate motor performance across their projects.

Bill of Materials:

General Tools:

Mechanical	2mm Hex Wrench[1]	4mm Hex Wrench[1]	3D Printer[1]	Sandpaper[1,2]
Electrical	Soldering Iron[1]	Desoldering Gun[1]	Lead Free Solder[1]	

Mechanical:

Item	Amount	Source	Price/Unit
M2 Heat Insert[1]	1 1	McMaster Amazon	$22.63/100 $8.99/100
M2 Screws[1]	1 1	McMaster Amazon	$15.68/100 $8.48/50
M4 Screws[1]	1 1	McMaster Amazon	$12.02/100 $7.99/25
Filament (PLA)[1]	1 (124.3g)	Amazon Bambu	$14.99/1KG $19.99/1KG

Total: $6.65 in Used Materials

Electronics:

Item	Amount	Source	Price/Unit
Photointerrupter	1	Amazon	$9.98/10
Load Cell & Driver	1	Amazon	$15.49/4
Current Sensor	1	Amazon	$7.99/4
Arduino Uno	1	Amazon	$22.08/1
Wiring[1]	1	Amazon	$6.98/1
Breadboard[1]	1	Amazon	$19.99/16
Lead Free Solder[1]	1	Amazon	$7.99/1

Total: $32.07 in Used Materials

[1] *Can be sourced from OEDK consumables/tools*

[2] *Optional tools for ease of construction in the event of mistake or modification*

Instructions to Build:

Soldering and Wiring:

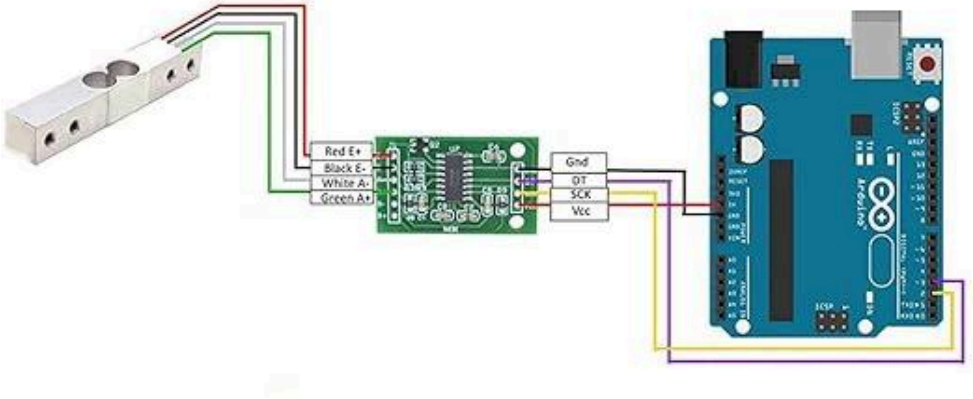
This part provides an approach for assembling and wiring a dynamometer that tracks RPM, torque, and current using an Arduino Uno R3 or compatible microcontroller. The system is built around three primary components: the HX711 load cell amplifier for torque measurement, the ACS712 current sensor for current measurement, and an interrupt-based input to measure RPM.

The load cell usually comes in separate with the HX711 amplifier. It uses four wires that need to be connected to the HX711 amplifier. Below is the correct pin mapping for the load cell to HX711 connection:

Load cell Wiring	HX711 Pin Input
Red	E+
Black	E-
White	A-
Green	A+

Red (E+): Connects to the **E+** pin on the HX711 (Excitation Positive).
Black (E-): Connects to the **E-** pin on the HX711 (Excitation Negative).
White (A-): Connects to the **A-** pin on the HX711 (Signal Negative).
Green (A+): Connects to the **A+** pin on the HX711 (Signal Positive).

For the HX711 output pins to the Arduino UNO, the wiring is listed as follows:



HX711 Pin Output	Connection Description	Microcontroller Pin
VCC	Power to load cell	5V (Microcontroller)
GND	Ground of load cell	GND (Microcontroller)
DT	Data from load cell to microcontroller	Pin 4 (Digital Pin)
SCK	Clock for HX711	Pin 3 (Digital Pin)

Using the upper process as a reference, the ACS712 and the photo interrupter can also be wired with the following scheme:

ACS712 Pin	Connection Description	Microcontroller Pin
VCC	Power to Photo Interrupter	5V (Microcontroller)
GND	Ground of Photo Interrupter	GND (Microcontroller)
OUT	Data from ACS712 to microcontroller	A0 (Analog Pin)

Photo Interrupter Pin	Connection Description	Microcontroller Pin
VCC	Power to Photo Interrupter	5V (Microcontroller)
GND	Ground of Photo Interrupter	GND (Microcontroller)
OUT	Data from Photo Interrupter to microcontroller	Pin 2 (Digital interrupt Pin)

Troubleshooting (FMEA)

If users are experiencing issues where no data is outputting or the Arduino is not taking in data, it's important to systematically check the potential causes. Below is an FMEA that helps identify common issues and guide troubleshooting for situations where the data isn't coming through properly.

Load cell not outputting Torque

Failure Mode	Possible Causes	Effect	Mitigation/Action
No output from HX711	Incorrect wiring between HX711 and Load Cell	No data from load cell	Double-check the connections to ensure the load cell’s wires (Red, Black, White, Green) are correctly connected to the HX711 pins (E+, E-, A-, A+).
	HX711 not powered or misconnected	HX711 not operational	Ensure that the HX711 is connected to the power and ground rails (5V and GND) correctly. Confirm the microcontroller is powered and supplying the correct voltage.
	HX711 initialization failure in code	Arduino doesn't read the load cell	Check that the scale.begin() function in the code is called properly and verify the pin assignments for LOADCELL_DOUT_PIN and LOADCELL_SCK_PIN.
	Load cell damaged or defective	No readings	Test the load cell on a different system or try replacing the load cell to see if readings are outputted.
	Incorrect load cell calibration	Wrong data output	Adjust the MULTIPLIER value in the code according to the specific calibration for your load cell. Use a known weight to calibrate it.

Arduino Not Receiving Data (RPM or Current)

Failure Mode	Possible Causes	Effect	Mitigation/Action
No RPM signal detected	Incorrect wiring of the RPM signal pin	No RPM count or timing data	Verify the wiring of the interrupt signal (Pin 2 on the Arduino) from the RPM sensor. Ensure the sensor is outputting a pulsed lighting.
	Interrupt pin not configured correctly	Arduino does not react to RPM signal	Ensure the attachInterrupt() function is used correctly and the pin mode is set as INPUT_PULLUP for the interrupt pin. Check if the interrupt is enabled in code.
	Debouncing issue (frequent false interrupts)	Inaccurate RPM readings	Ensure the debounce logic in the interrupt service routine (ISR) is implemented correctly. Check the debounce delay (debounceDelay) for appropriate timing.
No current data from ACS712	Incorrect analog pin connection	No current readings	Check the connection of the ACS712 output pin to the correct analog pin (A0) on the Arduino. Verify the analogRead() function is correctly reading the data.
	Incorrect power supply to ACS712	ACS712 not operational	Verify that the ACS712 is correctly powered (5V) and ground is connected to the Arduino.
	Faulty ACS712 sensor	No current data	Replace the ACS712 sensor with a known working one to verify the sensor is functioning.
	Incorrect calibration of ACS712	Wrong current reading	Double-check the offset and sensitivity values in the code. Adjust ACS712_OFFSET and ACS712_SENSITIVITY according to your specific sensor and setup.

Arduino Code Issues

Failure Mode	Possible Causes	Effect	Mitigation/Action
Arduino code not running correctly	Incorrect pin assignments in code	Data from sensors not read or output	Verify that the pin assignments in the code match the actual hardware wiring. Ensure all pins are declared correctly in the setup() function.
	Incorrect serial communication setup	Data not output to Serial Monitor	Check that Serial.begin(115200); is initialized in the setup() function. Confirm that the Serial Monitor baud rate is set to 115200.
	Missing or incorrect data processing functions	No output in serial monitor	Ensure functions like getIsrData() or readSensors() are being called within the main loop, and check that each sensor's data is being processed and printed.
	Unstable or blocking code (infinite loops, delays)	Arduino unable to process other tasks	Avoid using delay() in your main loop, use millis() for non-blocking timing. Review your code for infinite loops or blocking calls that might halt data collection.