



FINAL REPORT: ASYMMETRICAL FORCE SENSOR FOR ROWING
BIOMECHANICS

December 13, 2024

BME 400

Clients: Dr. Jill Thein-Nissenbaum, Ms. Tricia De Souza, Ms. Sarah Navin

Advisor: Dr. David Appleyard

Team Members:

Team Leader: Allicia Moeller

Communicator: Simerjot Kaur

BWIG: Neha Kulkarni

BPAG: Colin Fessenden

BSAC: Emily Wadzinski

Abstract

Elite rowers that engage in a high volume of training can suffer from a variety of injuries, most commonly occurring in the lumbar spine [1]. As rowing is a full-body movement, perfecting technique and maintaining proper form is essential to preventing such injuries and improving performance overall [2]. The UW-Madison Women's Rowing Team is seeking a way to measure force output from rowers' lower extremities to determine the presence and cause of asymmetrical loading and resulting back pain. Existing products, such as the BioRow Force Plates, are very expensive and not adaptable to specific user needs [3]. In order to achieve a more affordable and adaptable alternative, several designs were considered. The design that was deemed most suitable is the Stationary Force Plate design with load cells at each corner of the right and left ergometer foot stretchers. The signal from each of the load cells is sent to a Raspberry Pi Pico, which communicates with a laptop to generate a graphical user interface (GUI) that updates with real-time force data. Accuracy testing was performed with calibrated weights, and the force plate's accuracy was within the 5% margin of error set by the Product Design Specifications (PDS). The device will be tested for accuracy, repeatability, and reliability to a greater extent next semester, and the capacity for raw data storage will be integrated. The final prototype shows potential for broad adoption in rowing training programs, improving the effectiveness of rowing biomechanics analysis and the health and safety of athletes.

Table of Contents

Abstract	1
Table of Contents	2
I. Introduction	3
Motivation	3
Current Methods and Existing Devices	3
Problem Statement	4
II. Background	5
Relevant Physiology and Biology	5
Relevant Design Information	6
Client Information	8
Design Specifications	9
III. Preliminary Designs	10
Stationary Force Plate	10
Membrane-Bound Force Plate	11
Bearing-Guided Force Plate Design	12
IV. Preliminary Design Evaluations	15
Design Matrix	15
Design Evaluations	16
Proposed Final Design	18
V. Fabrication/Development Process	19
Materials	19
Methods	20
Final Prototype	24
Testing	24
VI. Results	26
VII. Discussion	26
VIII. Conclusions	27
IX. References	28
X. Appendix	30
Appendix A: Product Design Specifications	30
Appendix B: Materials and Expenses	39

I. Introduction

Motivation

Many members of the University of Wisconsin Women's Rowing team have reported lower back pain and other injuries, possibly due to asymmetric force output while training. Asymmetric force output causes the athlete's hips to misalign, leading to lumbar spine distress. In rowing, a slight change in body angle or stroke sequence can significantly impact joint positioning and stresses placed on the lower back [4]. Rotational twisting at the hips and torso are the lead causes for back pain in rowers, but is currently only qualitatively studied by the University of Wisconsin personal trainers through visual observation [4]. Many rowers experience back injury due to various reasons: consistently exerting force when the back is flexed, repetition of the rowing movement, and not properly adapting to the size of the ergometer or boat [5]. Despite this, current methods do not involve a way to quantitatively assess asymmetry in rowers. The University of Wisconsin Women's Rowing coaching staff is looking for a device to measure the force output female collegiate athletes produce while rowing. They want this device to be mounted on an ergometer, because most in-season training sessions, and therefore most injuries, occur on the ergometer. With this device, the athletic training staff hopes to be able to interpret differences in symmetry of a rower's force output, fix their form, and potentially reduce the risk of lower back injury by looking at quantitative values, rather than one-on-one observations.

Current Methods and Existing Devices

The University of Wisconsin Women's Rowing team currently uses an ergometer and one-on-one visual coaching and analysis to critique form and look for potential injury risks. Their current data is all qualitative, and uses the judgment of a trainer or coach to make observations and correct form. These assessments primarily occur during rowing on an ergometer. The Concept2 RowErg, which is the ergometer used by the UW Rowing Team,

displays a Force Curve that is used by rowers to track their force throughout a stroke. The ergometer's screen displays a live force-time curve for each stroke and rowers are adept at interpreting this graphical representation. However, this design focuses on force output through the handle, not the lower extremities [6]. This device helps athletes compare their real time force output to reference graphs which help understand the flaws in their form.

One potential solution to track lower extremity forces is the BioRow 2D Force Stretcher (Figure 1), produced by BioRow Ltd., which is a plate affixed to the foot stretcher of an ergometer. The plate has load cells attached to it with strain gauges that measure force in the normal and shear directions. The plate contains four load cells, two for each foot, placed on the heel and the toe locations [3]. These load cells are capable of measuring high force outputs in rowers, and can assist personal trainers and coaches with critiquing a rower's form. However, this device is outside the budget for the UW Rowing Team and not fully compatible with their ergometer setups.



Figure 1. BioRow 2D Force Stretcher [3].

The Bertec Force Plates are also capable of sensing forces from lower extremities; specifically, they sense ground reaction forces during gait, balance, and performance analysis. They contain load cells that sample at a rate of 1000 Hz, and can sense force in three directions. These force plates have large load capacities ranging from around 4500 N to 17,800 N, and come in a permanent model which can be fixed to the floor, or a portable model. Bertec also produces

custom electronics and software which are both used to process the raw data from the force plates [7]. Though they are the lab and industry standard, these force plates cannot be modified in any way in terms of size or configuration to fit an ergometer.

Problem Statement

Many college rowing athletes, particularly women, are susceptible to lifelong lower back or hip injuries due to varying weight distributions on each leg while rowing. This issue can be addressed through gathering real-time data on athlete biomechanics, but this data is often difficult to obtain. Collection and analysis of biomechanical data will enable athletes to adapt their technique towards better performance, and will assist coaches and trainers in preventing injury. The client, Dr. Jill Thein-Nissenbaum, has tasked the team with creating a force plate system that can collect biomechanical data from rowers' lower extremities at a lower cost than current market options. The team's goal is to create a force sensor system on the ergometer that will capture force output during time of use and will assess lower extremity asymmetry to establish risk stratification across all rower demographics. Additionally, the team aims to connect the force plate system to a user-friendly interface that will enable coaches and athletes to understand essential biomechanical information, thereby improving both performance and safeguarding against potential injuries.

II. Background

Relevant Physiology and Biology



Figure 2. Phases of the rowing stroke [1].

Rowing is a very high impact, fast-paced, and technical sport. Rowing requires a high magnitude of force from the entire body, but especially from the legs. As shown in Figure 2, there are four phases of the rowing stroke: the catch, the drive, the finish, and the recovery. During the catch phase, the rower's oars are fully in water, and their hips, knees, and ankles are in full flexion. The rower then moves into the drive phase, the rower extends their hips, knees, and ankles forcefully to propel the oar. During this phase, the upper body is braced so force can be transferred from the legs to the oars. During the finish, the rower is in full extension in their lower extremities and their elbows are in full flexion as they have completed the full range of motion required to move the oar. The recovery phase is the return to full flexion as the rower prepares to start the cycle of catch, drive, finish and recovery again [1].

The forces involved in the upper body can cause the spine to rotate as rowers typically only hold one oar on one side of their body in sweep rowing. This creates torque in the upper body as the spine twists to help pull and push the oar. The lumbar spine only allows for about 1.2

to 1.7 degrees of rotational movement, but most rotation happens in the mid-spine causing stress on the lumbar spine leading to back pain [8]. As a result, the most commonly cited injuries in rowers are those of the lumbar spine [9].

Relevant Design Information



Figure 3 (left) and Figure 4 (right). Footplate of two different Concept2 RowErg models used by UW Rowing Teams.



Figure 5. Concept2 RowErgs (older models) lined up to simulate a shell during sweep rowing in the UW Boathouse tank.

The two main forms of rowing are sculling and sweeping. Sculling is symmetric as rowers hold onto one handle of an oar in each hand directly in front of them and are able to pull straight back without having to twist. This form is mimicked in an ergometer. The second form, sweeping, is done on one side of the body and each rower has only one oar to manipulate. This is an asymmetric form of rowing that causes rowers to twist their upper body as they row. This form of rowing is done in a boat or tank.

The prospective design must be able to be installed on the Concept2 RowErg, which is the type of ergometer used by the UW Rowing Teams. The UW Rowing Teams use two different models of the Concept2 RowErg, whose footplates are shown in Figures 3 and 4, and the device should be modular such that it can be installed on either model. The model shown in Figure 3 is a newer model and is more commonly used during training. The model shown in Figure 4 is

configured in the tank in the UW Boathouse. The tank houses 12 bases of the Concept2 RowErg lined up in a row to simulate a shell configuration, as shown in Figure 5. It is imperative that the device can be transferred to this setup to assess asymmetry during sweep rowing as well as ergometer rowing.

The footplate on the ergometer features a detachable heel portion called the Flexfoot that allows for rowers to disconnect from the footplate and gain momentum when pulling back on the oar. Additionally, foot straps keep the rower's forefoot attached to the foot plate allowing the rower to pull back in using force generated from the front of the foot. The seat can freely move up and down along a bar, permitting the rower full extension of their legs.

Client Information

The clients for the project include Dr. Jill Thein-Nissenbaum, Ms. Tricia De Souza, and Ms. Sarah Navin. All three work with and are representing the University of Wisconsin-Madison (UW-Madison) Women's Rowing Team. Dr. Jill Thein-Nissenbaum is a professor in the UW Madison Physical Therapy Program, and is the staff physical therapist for Badger Sports Medicine. She provides consultation and rehabilitation services for all UW Madison sports and works in the Badger Athletic Performance Center analyzing athletic testing performed on UW Madison athletes [10]. Ms. De Souza is a UW-Madison Athletic Trainer; in particular, she provides athletic training services for both the Badgers Men's and Women's Rowing Teams [11]. Finally, Ms. Sarah Navin is a UW Madison Physical Therapy student. She attended UW Madison for undergraduate school and was previously on the Badger Women's Rowing team.

Design Specifications

This product has several specifications that will determine how fabrication and design is approached. Most importantly, the product must be compatible with both Concept2 RowErg models that are used by the UW-Madison Rowing Teams. This will entail taking certain dimensions into consideration, such as the standard ergometer footplate height and width of 30.7 cm by 13.3 cm. The device must not impede normal rowing motions, so it should not noticeably alter the shape of the ergometer footplates. The main goal of the design is to provide real-time, accurate measurements of rowers' magnitude of force so that any asymmetries can be corrected

in the moment. As such, the device should have a load capacity of at least 900 N [12] and the force magnitude must be measured within a limited margin of error of 5% [13]. The display component of the design should have at least a 24 Hz frame rate and no more than a 0.5 second delay to provide real-time visual feedback at least once per stroke [14, 15]. The product should be engineered to last a service life of around 10-12 years, approximately the length of an average rower's career [16]. The product should also be reproducible, with the end goal of developing 8 devices that can be positioned on the ergometers to gather data. The full Product Design Specifications (PDS) are outlined in Appendix A.

III. Preliminary Designs

Each preliminary design conceptualized by the team utilizes load cells as the force-sensing method. To keep the cost of the device low, the team opted to use single-axis load cells, which can only sense force in the normal direction. Though rowers do apply force on the footplate in the shear direction during rowing [12], it is critical that the load cells' integrity and accuracy are not damaged by these shear forces. Therefore, the team created three designs with the intent of shielding the load cells from off-axis loading to varying degrees.

Stationary Force Plate

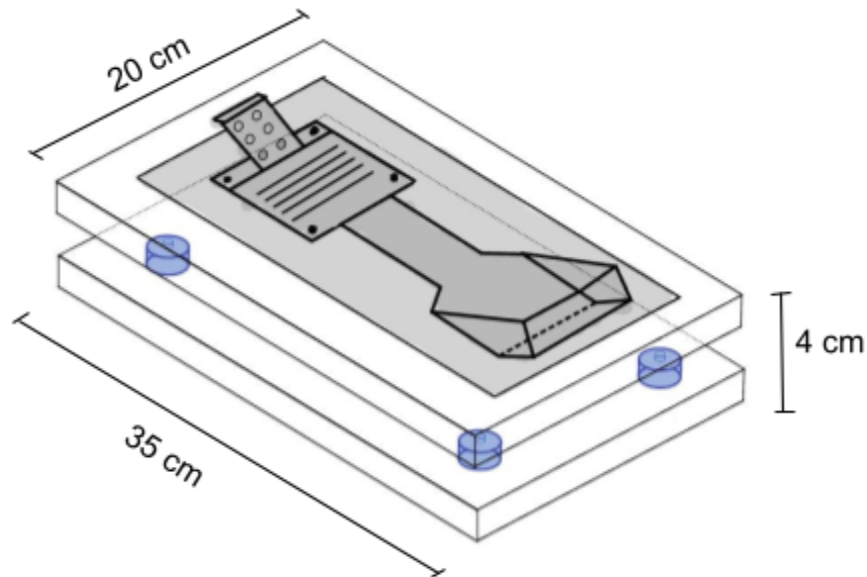


Figure 6. Stationary Force Plate Design.

The Stationary Force Plate Design, as shown in Figure 6, consists of two aluminum plates. The bottom plate is screwed into the ergometer in place of the original ergometer's footplate. Four single-axis, button load cells are secured into the corners of the bottom plate with screws. The second plate rests on top of the load cells, with divots that align to each load cell button. Foam or springs with negligible spring constants may be secured in between the plates to keep them together. The Flexfoot from the ergometer will be screwed back onto the top plate to maintain adjustability between different foot sizes.

Membrane-Bound Force Plate

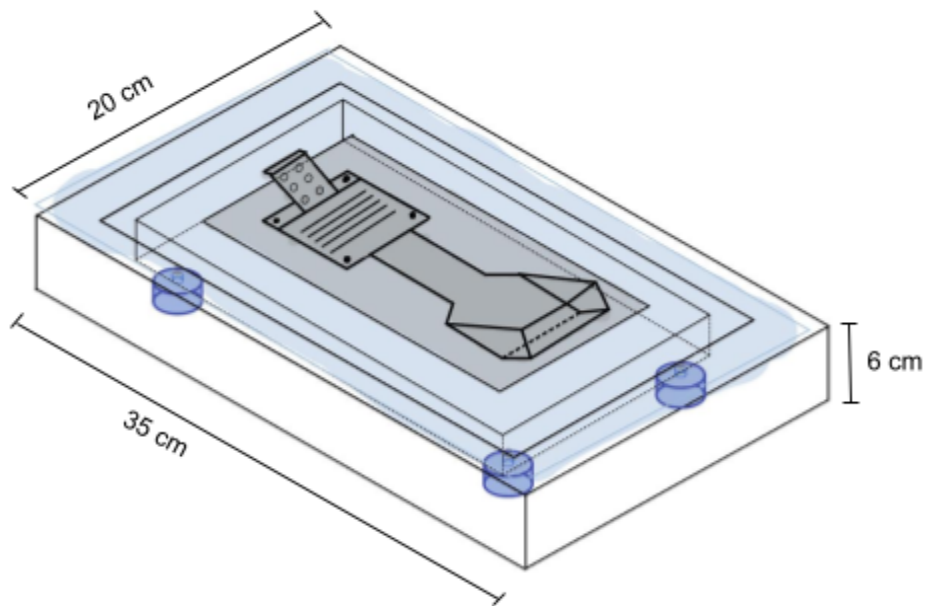


Figure 7. Membrane-Bound Force Plate Design.

The Membrane-Bound Force Plate Design, shown in Figure 7, combats potential off-axis loading with a reactionary tensile force produced by a fabric membrane. The design consists of an inner and outer aluminum plate with a slight gap between them. The outer plate will be screwed into the ergometer, with four load cells secured on top of the corners. Ball bearings will be placed on the points of the load cells to reduce friction between the load cells and inner plate. A fabric membrane will be glued to both plates and situated on top of the bearings. If the rower were to apply a force in the plane of the fabric, the fabric creates a reactionary force in the opposite direction to prevent the inner plate from moving and cancels any directional forces not normal to the plate when the athlete is rowing. The Flexfoot is also fixed to the top of the inner plate over the fabric to retain adjustability.

Bearing-Guided Force Plate Design

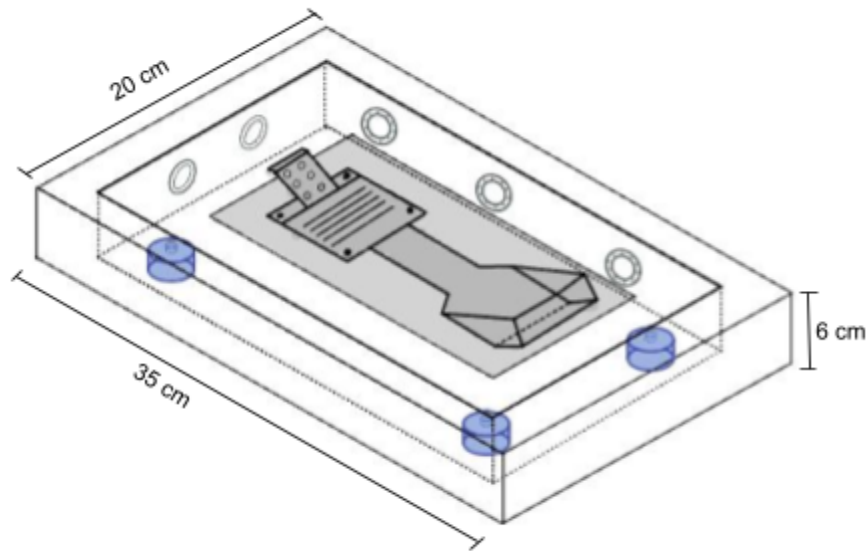


Figure 8: Bearing-Guided Force Plate Design.

The Bearing-Guided Force plate design in Figure 8 safeguards against off axis loading through the use of bearings placed on the sides of the footplate of the ergometer. The design consists of an inner and outer aluminum plate that are separated by frictionless bearings which allow for smooth movement while canceling out the horizontal and vertical force components. The outer plate will be mounted to the ergometer, and the inner plate will be placed on top of the load cells at each corner with ball bearings. The ball bearings minimize friction and ensure the inner plate only deflects normal to the ergometer footplate plane. This shields the load cells from any off-axis loading.

Graphical User Interface and Display

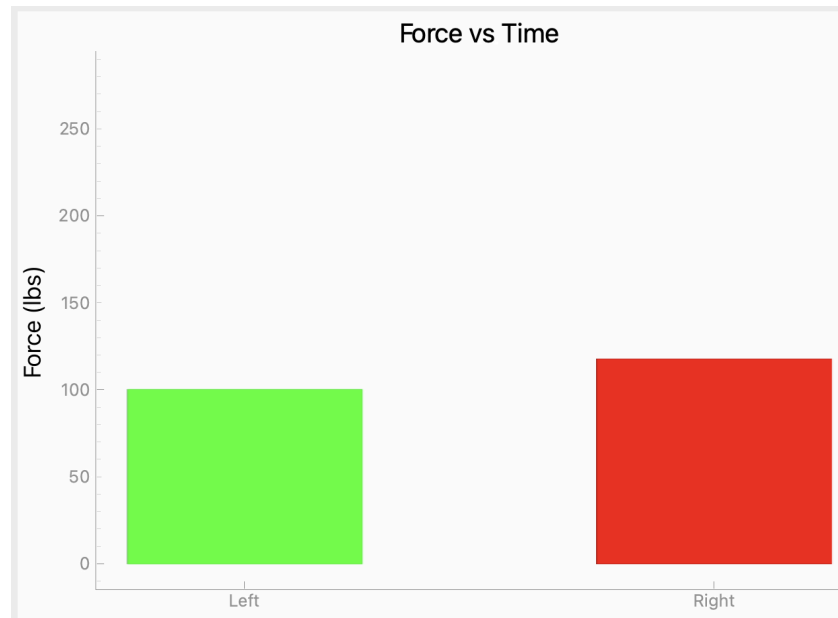


Figure 9. Absolute force output bar graph GUI design. The bar representing the leg with more force output turns red; the rower's goal is to press evenly so both bars are green.

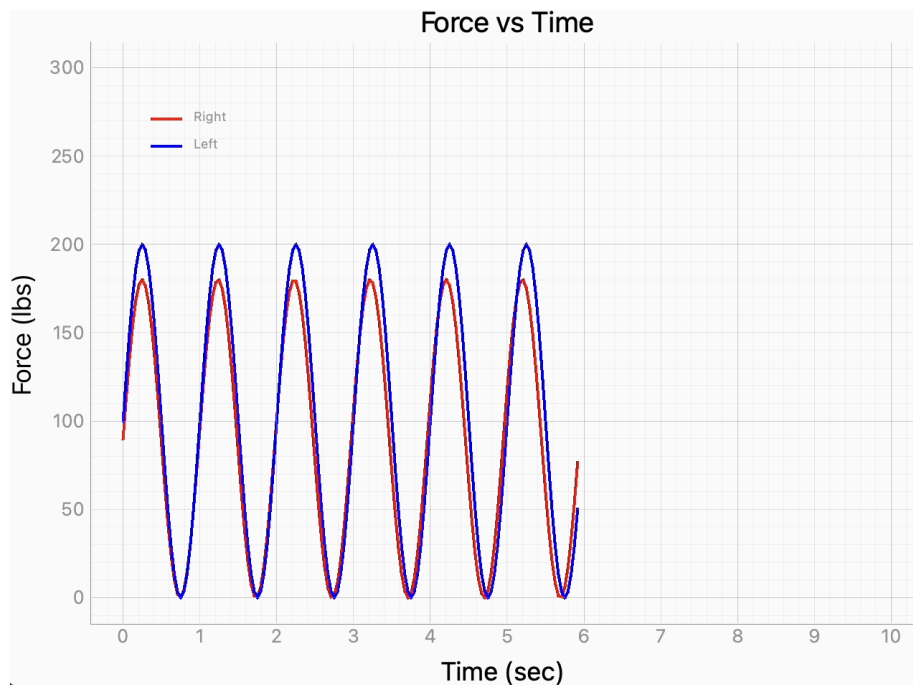


Figure 10. Absolute force output line graph GUI design. The rower's goal is to press evenly so the red and blue curves for each leg overlap.

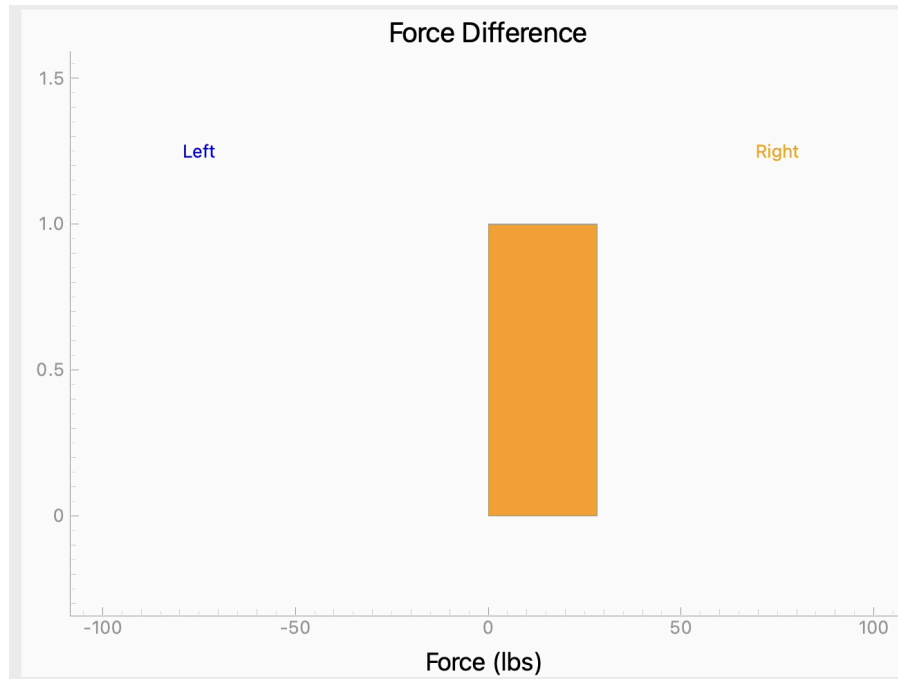


Figure 11. Force difference bar graph GUI design. The bar moves to the right or left, depending on which leg has more force output. The rower's goal is to press evenly such that the bar does not appear.

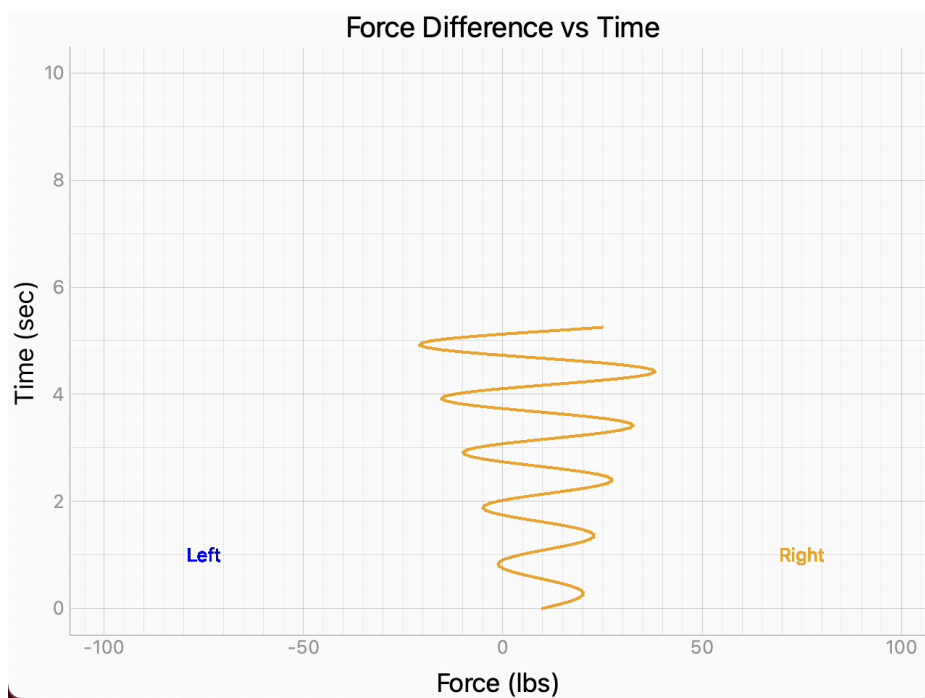


Figure 12. Force difference line graph GUI design. The line moves towards the leg with more force output. The rower's goal is to press evenly such that the line remains vertical.

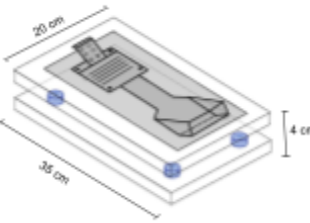
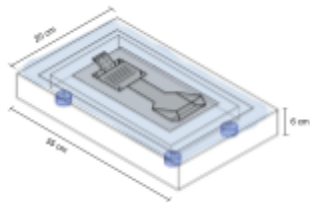
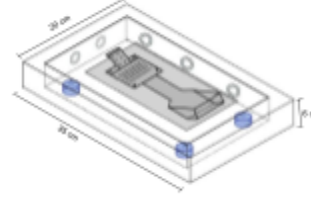
The chosen footplate design will be connected to a GUI that displays rowers' force output in real time. The user interface will allow users to choose between four graphical representations of force output. The four options consist of either the rower's pure force output or the force difference between legs on a bar graph or a line graph, as shown in Figures 9-12 above. Rowers will interpret these graphical representations of their force and adapt their technique.

The graphical user interface will be implemented using a Python script, designed using PyQt5. Python was chosen as the implementation method because it is easily accessible, highly adaptable to the client's needs, and allows for easy data storage. Because the GUI will be run in Python, a laptop will be required and will be used as the display screen. However, this laptop can be connected to a TV or larger monitor via HDMI cord for more convenient viewing per the user's preference.

IV. Preliminary Design Evaluations

Design Matrix

Table 1: Design matrix used to rank the three Load Cell Housing design ideas. Each category is rated by importance and is used to determine an overall score for each design.

		Stationary Force Plate		Membrane-Bound Force Plate		Bearing-Guided Force Plate	
							
Criteria	Weight	Score (5 max)	Weighted Score	Score (5 max)	Weighted Score	Score (5 max)	Weighted Score
Reliability	25	2	10	4	20	5	25
Ergonomics	25	5	25	4	20	4	20
Cost	20	3	12	2	8	1	4
Ease of Fabrication	15	4	12	3	9	2	6
Ease of Maintenance	15	4	12	3	9	2	6
Sum	100	Sum	71	Sum	66	Sum	61

Design Evaluations

Load Cell Housing Matrix Category Descriptions:

The design matrix to determine the best housing device includes the following criteria: Reliability, Adjustability, Cost, Ease of Fabrication, Ease of Maintenance, and Ergonomics.

Reliability refers to how accurately the device collects data despite off-axis loading, and the repeatability of collected data. Cost is a criterion to ensure that we are taking into account the budget constraints before moving forward with a design and location. The Ease of Fabrication category is how easily the design can be implemented to the Concept2 ergometer and how easily the load cell housing can be fabricated. The Ease of Maintenance criterion is a measure of the services required to maintain the device, such as calibration, cleaning, and replacement of parts. Lastly, the Ergonomics criterion refers to how the design may possibly hinder the rowing motion of the athlete. The goal of the design is to not inhibit the athlete's natural rowing motion.

Footplate Score Distributions:

The first design, The Stationary Force Plate, ultimately scored the highest out of the three with a score of 71. The stationary plate is the most simplistic design, and therefore easiest to fabricate, maintain, use, and is the least costly. However, in the Reliability category, the design scored the lowest. Due to the design's lack of off-axis load prevention, the accuracy of the data collected would be poor. In addition, with varying accuracy, the repeatability of the data would be inconsistent. In contrast, the design scored a 5/5 in Ergonomics, as it has the lowest profile and is the least obstructive to the rower since there's only a flat plate. In terms of Cost, the design was given a 3/5. Though the design requires less materials than the others, it still requires numerous costly load cells. In the Ease of Fabrication category, the stationary footplate's simple components allow for easier assembly than the others with added components. Lastly, for similar reasons, the design's Ease of Maintenance scored highly as it has fewer parts to preserve and would only require the occasional calibration. The footplate is also fully enclosed so it would not require any internal cleaning.

The second design, the Membrane-Bound Force Plate, ranked second with an overall score of 66. In terms of Reliability, the design scored moderately with a 4/5. This design provides improved accuracy in data collection over the Stationary Plate, due to its ability to limit off-axis loading. However, its complexity could still lead to some reliability concerns and is dependent on the material chosen to create the "membrane". For Ergonomics, the design scored a 4/5. While slightly more intrusive than the Stationary Plate, its overall impact on the rower's natural motion remains minimal, which is due to the membrane structure maintaining a low profile. For Cost, this design received a score of 2/5, primarily due to the additional materials and components needed like fabric and adhesive, which drive up expenses. The Ease of

Fabrication category yielded a score of 3/5, as the design, while not overly complex, is more intricate to manufacture than the Stationary Plate due to the membrane and attachment points. Similarly, for Ease of Maintenance, the design scored a 3/5, as the extra components mean there are more parts that require care, calibration, and potential replacement over time.

The third design, the Bearing-Guided Force Plate, ranked third with an overall score of 61. It scored the highest in Reliability, with a 5/5 main due to its accurate ability to minimize off-axis loading, ensuring more accurate and consistent data collection. However, this design was rated 4/5 in Ergonomics, similar to the Membrane-Bound Force Plate. While the bearings add some height to the system, they do not significantly interfere with the rower's motion. For Cost, the Bearing-Guided Plate scored the lowest, with a 1/5. Its more sophisticated design and the materials required, specifically the bearings, make it the most expensive option. In Ease of Fabrication, the design also scored a 2/5, as the additional components and precision assembly required for the bearings make it difficult to manufacture. Finally, for Ease of Maintenance, this design again received a 2/5. The bearings and associated mechanisms would require regular cleaning and maintenance to ensure smooth operation, adding complexity to its upkeep.

Proposed Final Design

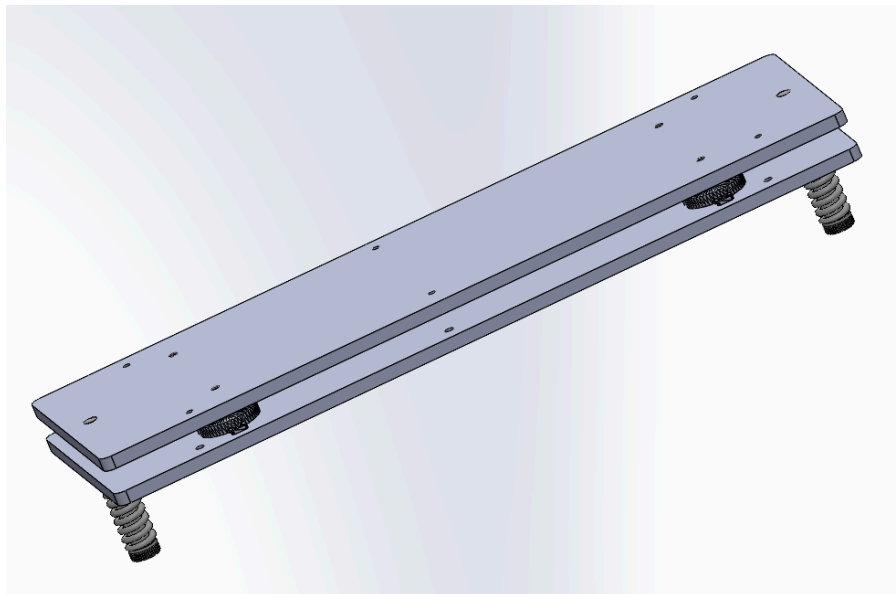


Figure 13. Isometric view of proposed final design.

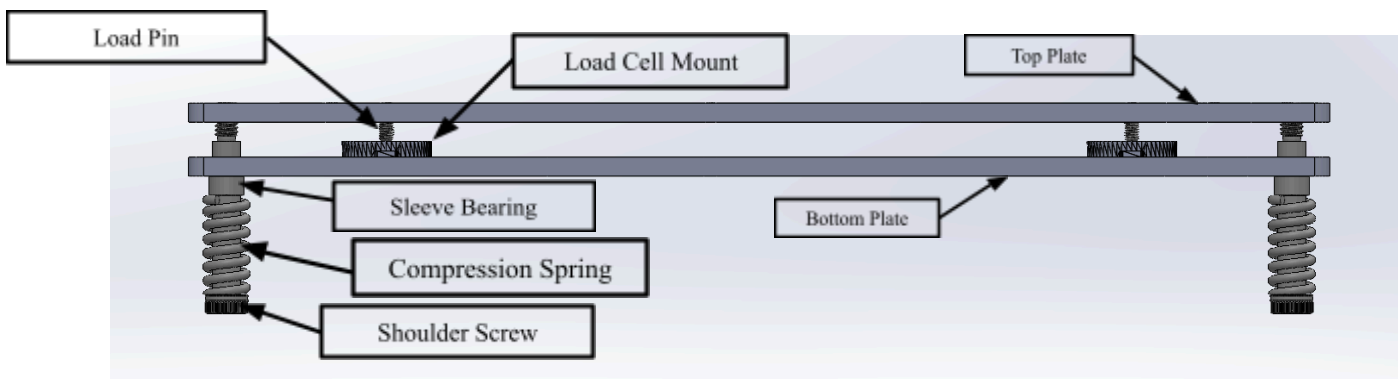


Figure 14. Side view of proposed final design, with labeled components.

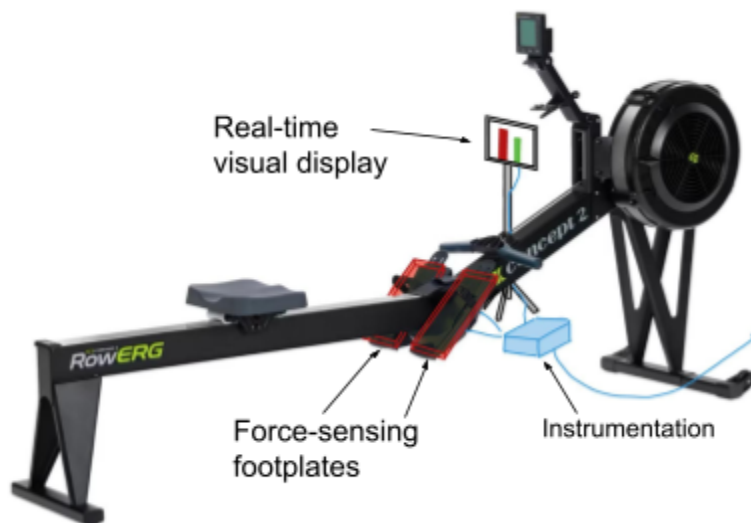


Figure 15. Placement and Integration of chosen footplate design, display, and instrumentation.

The proposed final design is the Stationary Force Plate design due to its overall simplicity and cost-effectiveness. While the design did not score particularly high in total points on the Design Matrix, its overall minimalism and core concept sufficiently provides an effective solution. The design saves expenses on additional components if off-axis loading proves to be

negligible. Otherwise, the design's components are transferable to the other proposed designs, which allows for efficient iteration.

As shown in Figures 13 and 14, the design consists of two plates with four load cells housed between them. The bottom plate will be mounted to the ergometer, while the top plate will have the Concept2 Flexfoot attached to secure the rower's feet. The plates are secured together using a shoulder screw that moves through a sleeve bearing. This assembly allows for frictionless movement of the top plate in the normal loading direction, which will place force on the load cells via set screws in the top plate that act as load pins. Compression springs on each shoulder screw apply preload by pushing the two plates together, allowing the device to sense relative tension as well as compression.

The force plates will be placed where the current footplates are located on the ergometer, as seen in Figure 15. The instrumentation hardware connected to the force plates will be compacted within a compartment below and connected to a laptop as the power source. This laptop will also act as the display for the real-time data, and will sit on a stand or in a position for the athlete to see when rowing.

V. Fabrication/Development Process

Materials

Footplate:

The team opted for aluminum as the footplate material. Aluminum plates provide an excellent yield strength of 40 MPa [20], which will ensure the plates will not yield under the load from the rower. The Flexfoot, which is the adjustable foot positioning mechanism on the Concept2 RowErg, will be secured to the aluminum top plate with six screws to ensure the rower's grip on the ergometer with the device. To secure the footplates together, two steel-shelled, PTFE-lined sleeve bearings and two steel shoulder screws will be used per plate. Six screws will be used to attach the bottom plate to the ergometer. Compression springs will be used around the shoulder screws, with a nut on the screw to compress the spring for pre-loading purposes.

Hardware:

Figure 16. TE Connectivity FX29 100 lb load cell. [17]

For the load sensing component, the team purchased the FX29 Compact Compression load cells from TE Connectivity, shown in Figure 16. The load cells have a capacity of at least 900 N and are at most 2.5 cm thick [17] so they are thin enough to be housed within an aluminum plate. The load cells will be secured using 3D printed housing out of PLA that is screwed into the bottom plate.

The microcontroller used to process the signal from the load cell will be the Raspberry Pi Pico due to its ability to interface with the sensors and GUI. A signal processing circuit will be fabricated onto a printed circuit board (PCB), using TLV274 op amps to filter, buffer and amplify the voltage output from the load cell, and the MCP3208 analog-to-digital converter to sample the amplified analog signal and send it to the Raspberry Pi Pico via SPI communication.

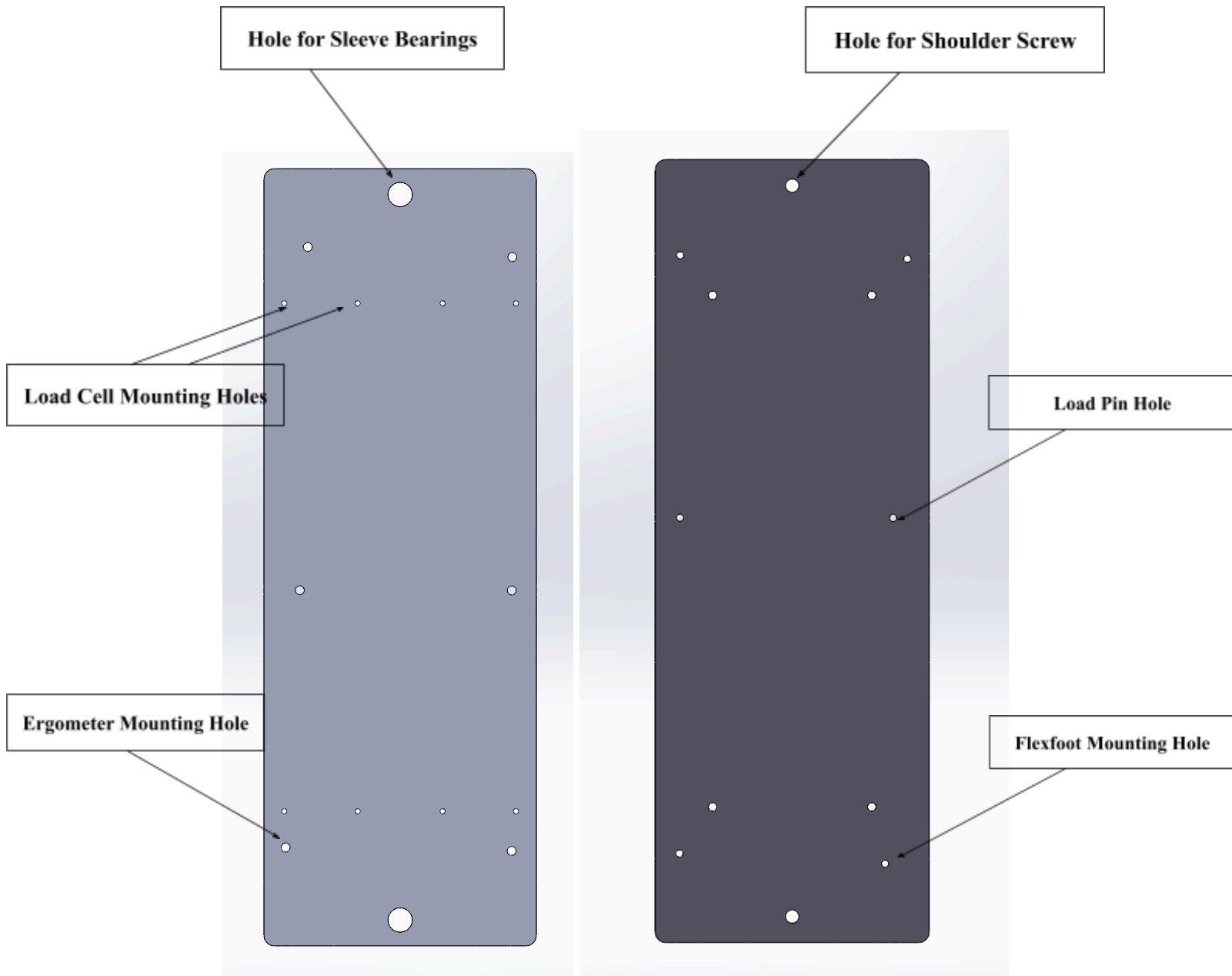
Display:

The hardware described above is connected to a laptop via USB. The laptop will act as both a display screen for the GUI and a storage location for the data collected during rowing trials. A laptop is necessary because the GUI and data storage methods will be implemented using a Python script or application. However, for easier viewing, the laptop could be connected to a TV or larger monitor, depending on the rowers' preference. The UW Boathouse has access to these display options if needed.

For a full materials and expenses list, refer to Appendix B.

Methods

Footplates:



Figures 17 (left) and 18 (right): Solidworks models of bottom plate (left) and top plate (right) with labeled features.

The footplates were fabricated by cutting aluminum plate stock down to size and using a CNC mill to locate and drill holes for assembling and mounting the plates. The bottom plate in Figure 17 contains holes for the sleeve bearings, load cell fixture mounting screws, and ergometer mounting screws. The top plate in Figure 18 contains holes for the shoulder screws and set screw load pins, as well as the Flexfoot mounting screws. Dimensioned Solidworks drawings with hole and tap sizes can be found in Appendix C.

Load Cell Mounts:

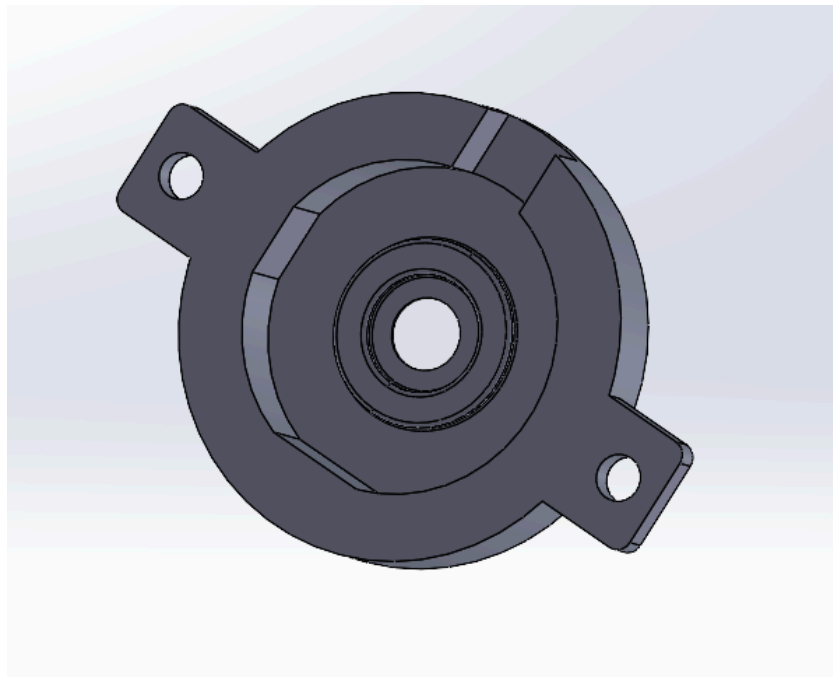


Figure 19. SolidWorks part of the load cell housing.

Between the two plates, 3D-printed fixtures shown in Figure 19 attached to the bottom plate house each load cell to secure them in place and expose the button of each cell to a set screw that is attached to the top plate.

Hardware:

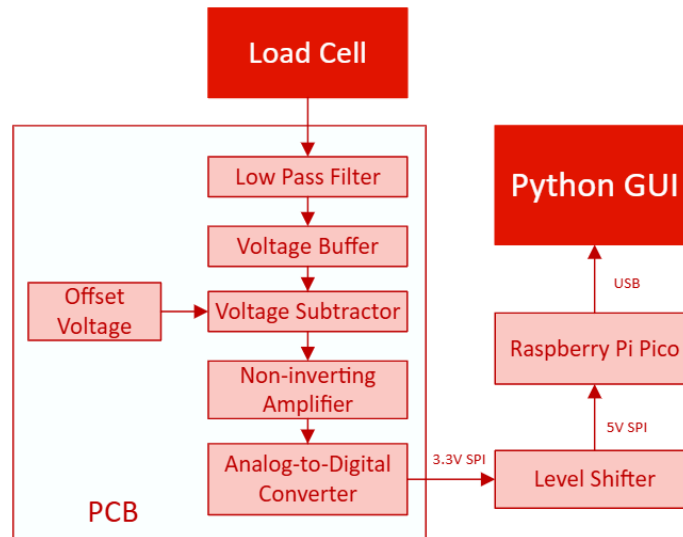


Figure 20. Hardware block diagram.

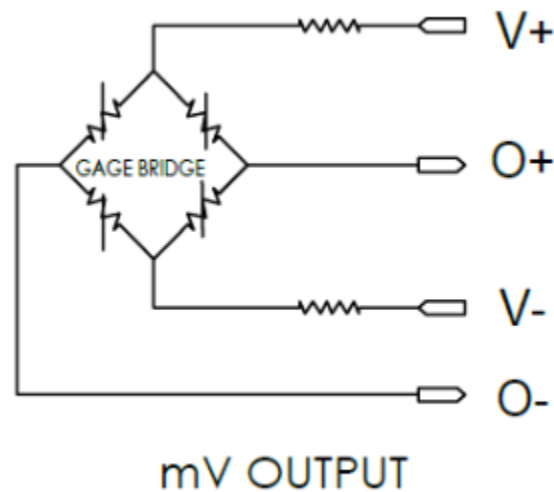


Figure 21. FX29 load cell circuit diagram [17]. V is the supplied voltage, and O is output voltage.

Figure 20 outlines the entire signal processing methodology to be utilized. The load cells used to fabricate this design will be the TE Connectivity FX29 100 lb load cells, which use a

wheatstone bridge circuit configuration with a strain gauge, shown in Figure 21. The load cells have an operating voltage of 5V and their data output is the difference between the positive and negative output lines. To implement eight load cells into our design (four for each foot plate), the Raspberry Pi Pico will draw power from a computer via USB connection and will power each load cell from its VBUS pin. The two output signals will be filtered with a low pass filter with a cutoff frequency of 7.23 Hz then passed through voltage buffers. Then, the two buffered signals will be subtracted with an offset voltage of roughly 104 mV and amplified with a gain of 23 by a non-inverting amplifier shown in Figure 19. The analog outputs of the non-inverting amplifier will be connected to the MPU3208 analog to digital converter which will send the data sequentially to the Raspberry Pi Pico digital pins via SPI communication. Because the operating voltage of the Raspberry Pi Pico GPIO pins is 3.3V, and the output of the ADC digital pins is 5V, a level shifter will be used to shift the voltage down from 5V to 3.3V. The Raspberry Pi Pico will send the data serially to the computer through the USB connection and a python script will calculate the total force on each plate using the load cell voltage data and update a GUI.

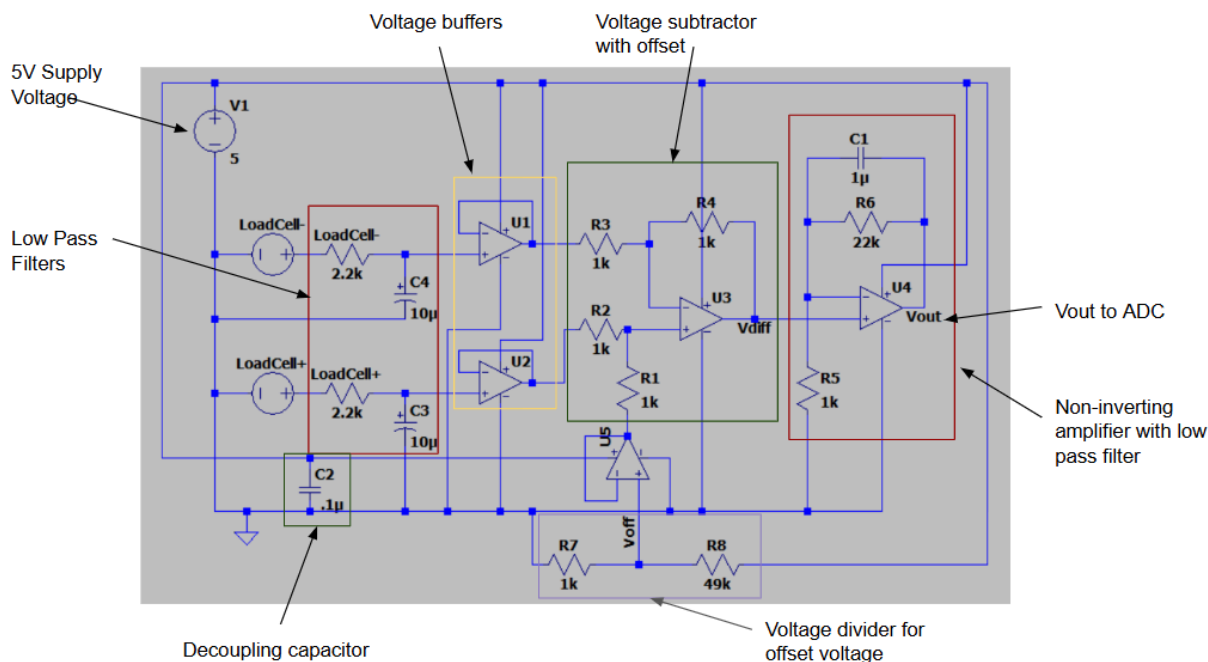


Figure 22. Differential amplifier subcircuit schematic.

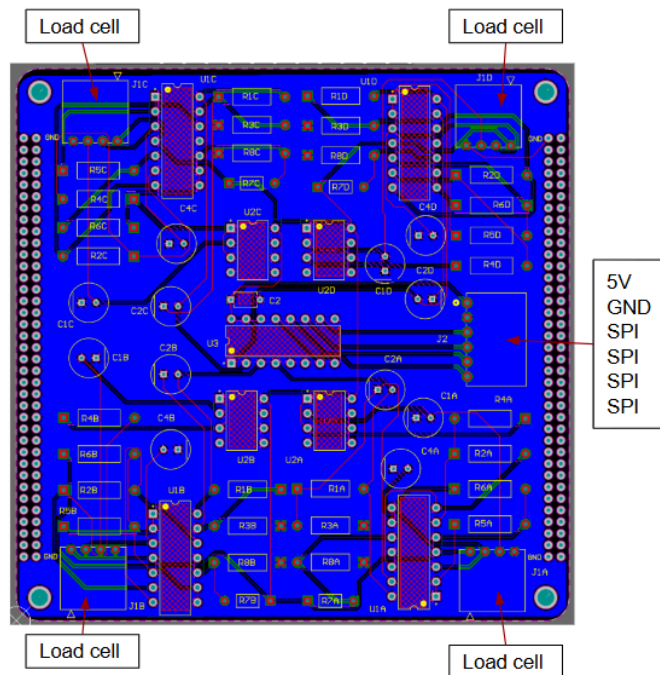


Figure 23. Printed circuit board layout.

To make the hardware integration more manageable, a PCB layout of the circuit detailed in Figure 22 was made (Figure 23). This PCB layout has socket connectors for four load cells and a 6-pin socket connector for the 5V power line, the ground connection, and 4 wire SPI connection. The PCB was designed to go in between the aluminum plates of each foot plate. The PCB provides power and ground connections from the raspberry pi pico to the load cells, and includes all the signal processing hardware and the analog-to-digital conversion. All the components on the PCB are through-hole components and were soldered onto the boards after receiving them.

Display:

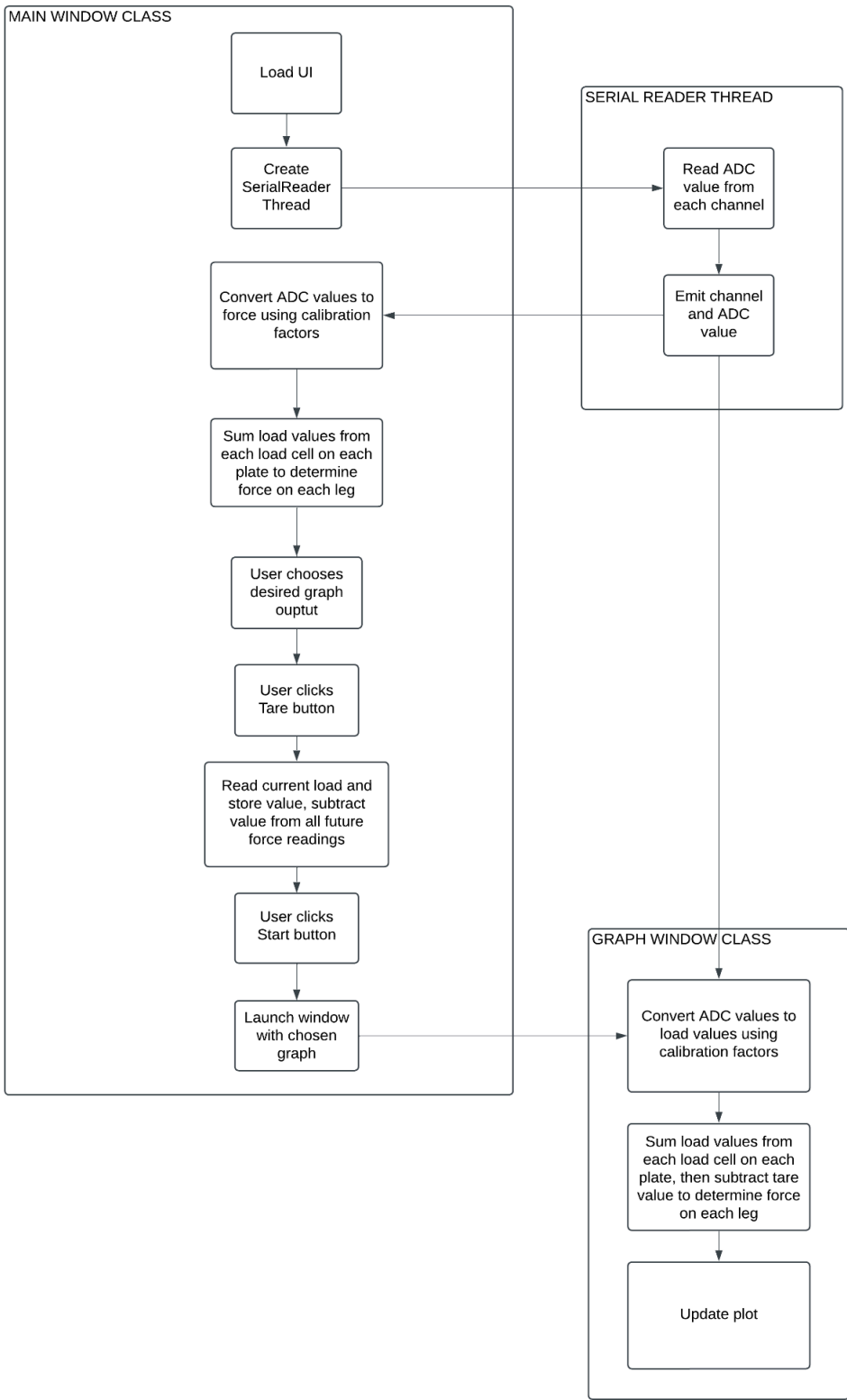


Figure 24. Block diagram of code handling signal processing and graphical user interface.

The graphical user interface (GUI) is integrated with the circuit with the code described by the block diagram in Figure 24. Using the PyQt library, a main window is initialized with a user interface allowing the user to choose their desired graph style, tare the device, and start the live plot. The Main Window class communicates with the Raspberry Pi Pico through the Serial Reader thread, which continuously reads ADC values from each channel and emits them back to the Main Window class. Then, inside the Main Window class, raw force values from each sensor are computed using calibration factors determined in testing and added to determine total raw force applied on each plate. When the user presses the tare button, the current raw force value is saved as the tare value; this value is subtracted from all future force readings to ensure accurate force measurements. When they are ready, the user clicks the start button, triggering a new Graph Window class to initialize. This class contains the axes for the chosen graph style and updates the plot using force values calculated by converting emitted ADC values to raw force, multiplying by the appropriate calibration factor, summing the sensors on each plate, and subtracting the tare value. For the full code utilized in the final prototype, refer to Appendix F.

Assembly:

To assemble the prototype, all capacitors, resistors, op amps, and the ADC and connectors were soldered to the PCB. Each PCB was adhered to the inside face of the bottom plate using electrical tape. Each load cell was placed inside its housing and screwed into the bottom plate. Then, the load cell wires and Raspberry Pi were connected to the PCBs. The Raspberry Pi is adhered to the central shaft of the ergometer using electrical tape. With all the electronics fixed on each bottom plate, each bottom plate was mounted to the ergometer.

On the top plate, the set screw load pins were threaded in, ensuring they were protruding evenly from the underside of the top plate. After the set screws were secured, the Flexfoot for each foot was screwed into each top plate. A nut and compression screw was placed onto each shoulder screw before threading the shoulder screw through the sleeve bearing on the bottom plate and into the top plate. Once the plates were secured together with the shoulder screws, and the Raspberry Pi Pico was plugged into the computer via USB, assembly was complete.

Final Prototype



Figure 25. Assembled footplates mounted onto ergometer.

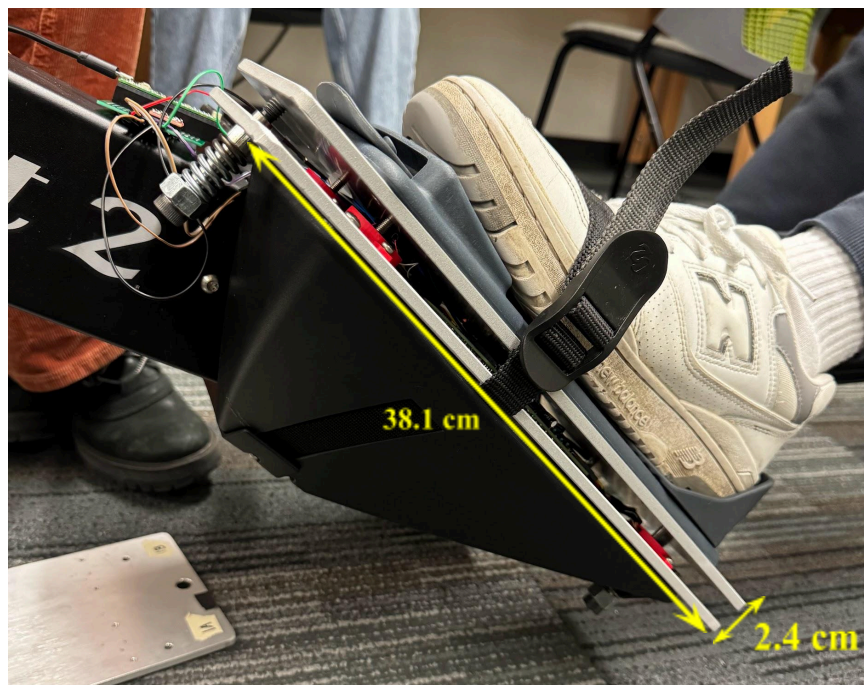


Figure 26. Side view of assembled prototype mounted on ergometer during rowing.

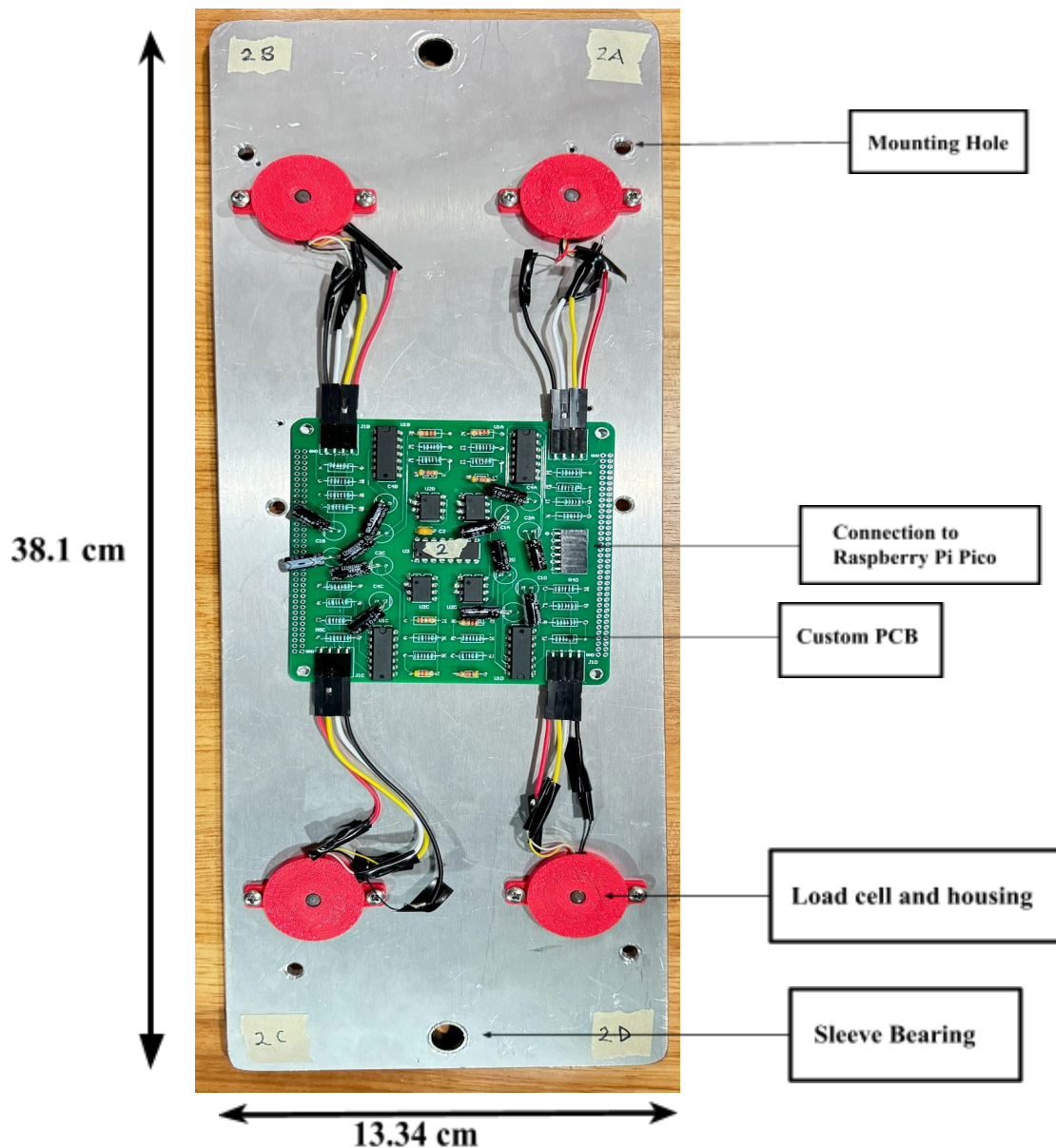


Figure 27. View of inside of bottom plate with components labeled.

The final prototype shown in Figures 25, 26, and 27 is fully assembled with all components integrated. The overall design consists of two plates with load cells and a PCB housed between them to measure applied force. The load cells are housed in 3D-printed fixtures as shown in Figure 27, and a set screw threaded into the top footplate acts as a load pin which contacts the load cell when force is applied. The top and bottom aluminum plates are secured together with a shoulder screw seated inside a sleeve bearing. This assembly allows the top plate to translate in the normal loading direction without friction, ensuring that the entirety of applied

normal loads are transmitted to the load cells. Also fitted onto the shoulder screws are compression springs, which apply pre-load by pushing the plates together. Pre-loading the device allows the load cells to measure compression and relative tension. The entire plate assembly is mounted on the footplate locations on the Concept2 RowErg. The top plate of the assembly has the Concept2 Flexfoot attached, to maintain the adjustable foot placement rowers are accustomed to.

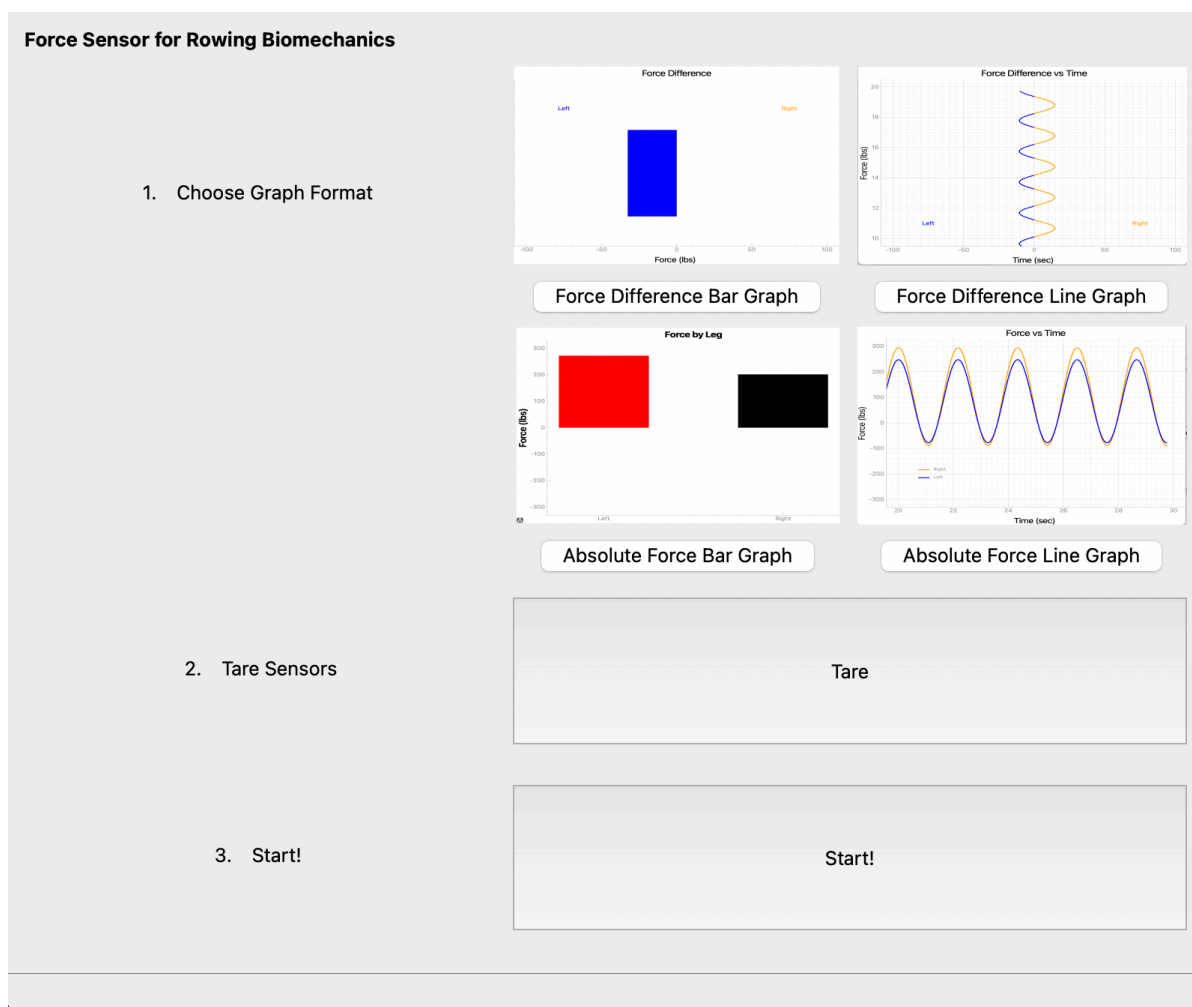


Figure 28. GUI home screen with buttons for user to choose desired graph, tare sensors, and start.

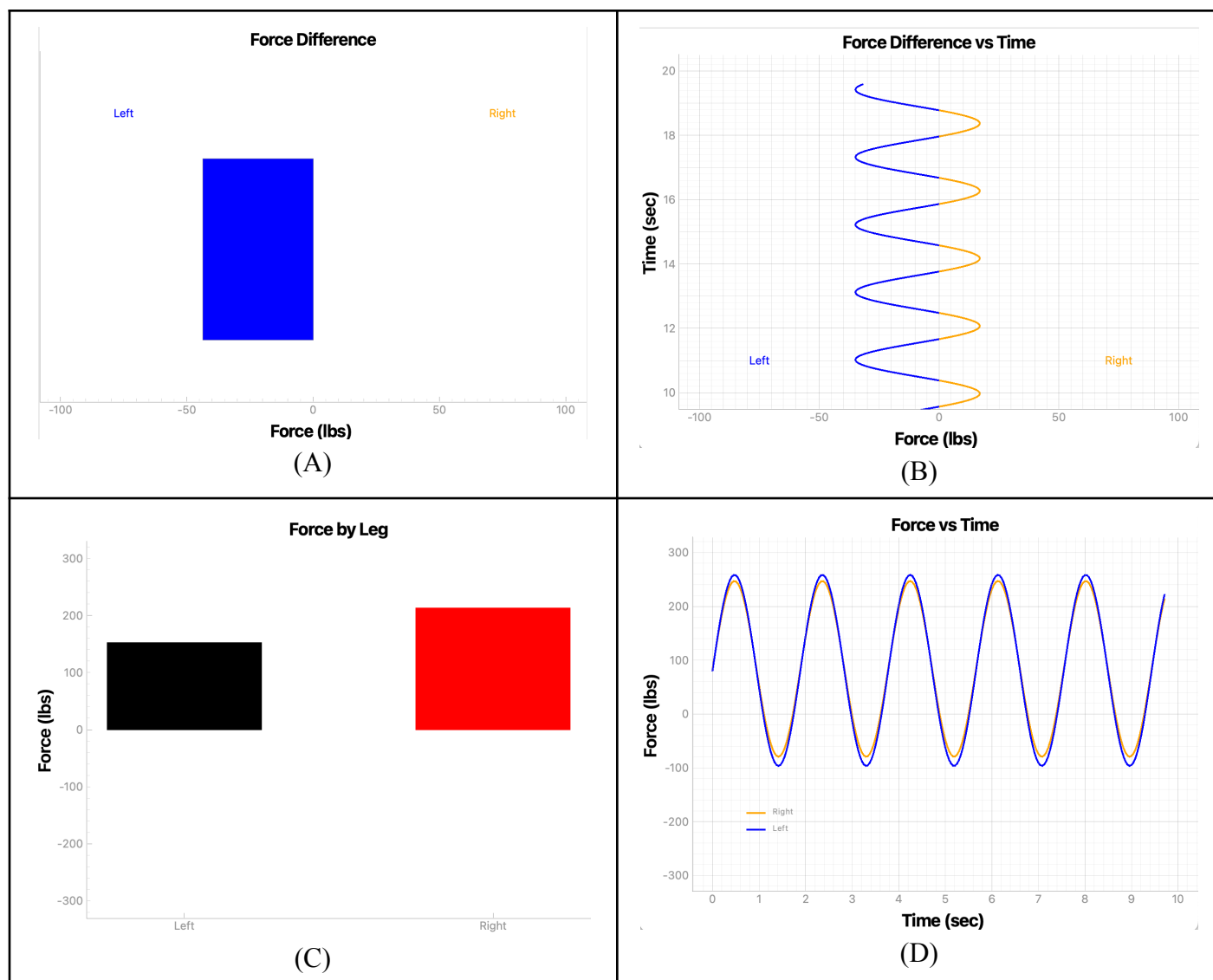


Figure 29 (A-D). Graph output options for displaying live force readings. (A) is a force difference bar graph, (B) is a force difference line graph, (C) is an absolute force bar graph, and (D) is an absolute force line.

Figures 28 and 29 above show the GUI. The GUI allows the rower to choose the graph output they would like from the options in Figure 28. These graphs display either force difference, where a bar or line moves proportionally towards the side applying more force, or absolute force, where bars or lines for each leg plot the absolute force for each leg. On the absolute bar graph, the bar for the leg applying more force turns black. If the force on both legs is within a 10% difference, both bars turn green to signify the rower is symmetric as defined by the clinical 10% threshold. For the other graphs, if the force on the right and left legs are not

within 10% of each other, the graph background flashes red to prompt the user to correct themselves. All of these plots respond in real-time and can be run until the user closes the graph window and returns to the homescreen.

Testing

Load Cell Calibration:

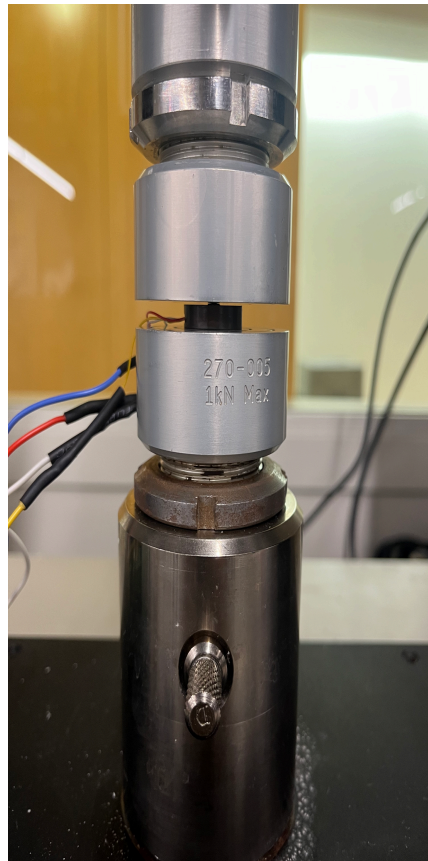


Figure 30. Load cell calibration setup on MTS Machine.

Each load cell to be used on the device was calibrated individually on the MTS machine, using the setup shown in Figure 30 with a 1kN MTS load cell and compression platen. A known load was applied using the machine, and the ADC value output from the load cell was recorded. This process was repeated five times at five different loads between 44.5 to 445 N to create a calibration curve relating load and ADC value for each load cell. The slope of this curve is the calibration factor for each load cell (see Appendix E: Testing Protocol - Calibration Protocol).

Accuracy Testing:



Figure 31. Accuracy testing setup using point load application mechanism.

With the plates fully assembled, a point load application mechanism shown in Figure 31. was used to apply varying weights to the center of the device while recording the measured load output. The team collected 13 data points with loads ranging from 0-578 Newtons. By comparing the known applied load to the measured load, the team was able to calculate the device's percent error and confirm if it fell within the 5% specified in the PDS.

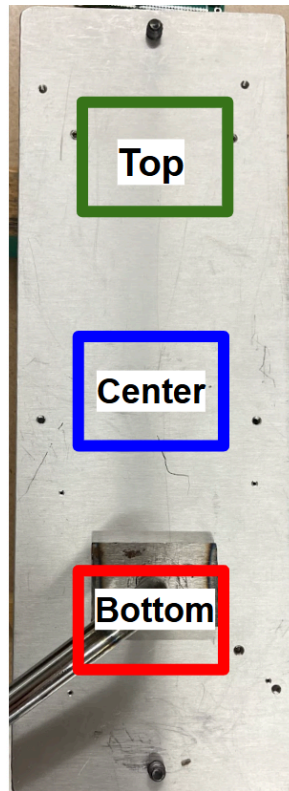


Figure 32. Locations of applied load in testing.

The team also conducted a separate test to assess the effect of the location of the applied load on the measured value. In this test, the load application mechanism was moved to the top, center, and bottom of the plate as seen in Figure 32. The same known load was applied to all three locations and the measured load was recorded. This process was repeated three times, using three different known loads.



Figure 33. Shear loading test setup.

To test how shear components of force affect the force plate, a pulley was used to apply a known shear load to the device, as seen in Figure 33. The pulley had a known weight on one end, and the other end was fixed to the point of application of a known normal load. This process was repeated using two different weights on the pulley providing shear (22 N and 50 N), and with five normal loads ranging from 182-557 N. With this data, the team could analyze whether shear loading has a significant effect on the normal force reading (see Appendix E: Testing Protocol).

VI. Results

Load cell performance was demonstrated to be reliable under standard conditions while also being sensitive to specific factors. The load cell calibration showed strong linear relationships between force and ADC values as seen in Figure 34 for all of the load cells with R^2 values ranging from 0.999978 to 0.999998. The average load cell calibration slope was 0.0516216. Force readings at the center position of the device (plotted in Figure 35) were not

significantly different from the actual weights with a p value of 0.49. Across 12 different trials (12 different weights) the average percent error was 0.476% and the worst-case percent error was -1.506%, meeting the PDS criterion of 5% margin of error. Additionally, testing across different load positions (top, center, bottom) showed no significant variation in readings as shown in Figure 36. This test was supported by an ANOVA p-value of 0.518. Despite reliable load cell readings with normal force testing, an introduction of a shear force of 22.7 Newtons caused a statistically significant difference in readings, with a p value of 0.0063 which indicates error induced from lateral forces. The percent error under shear loading, while statistically significant, still fell within the margin of error set by the PDS. Percent errors of the load cell readings with and without shear force is shown in Figure 37.

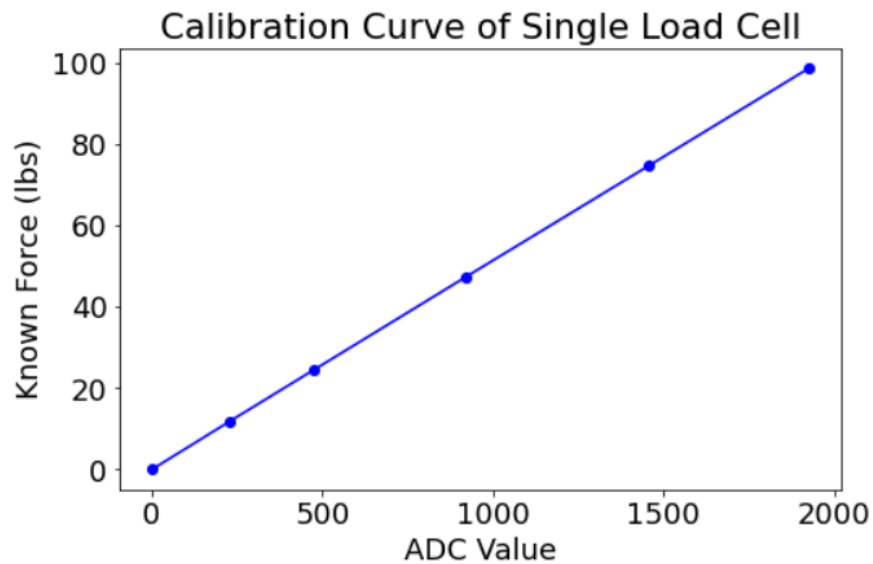


Figure 34. Force vs ADC Value for a single load cell, recorded during calibration.

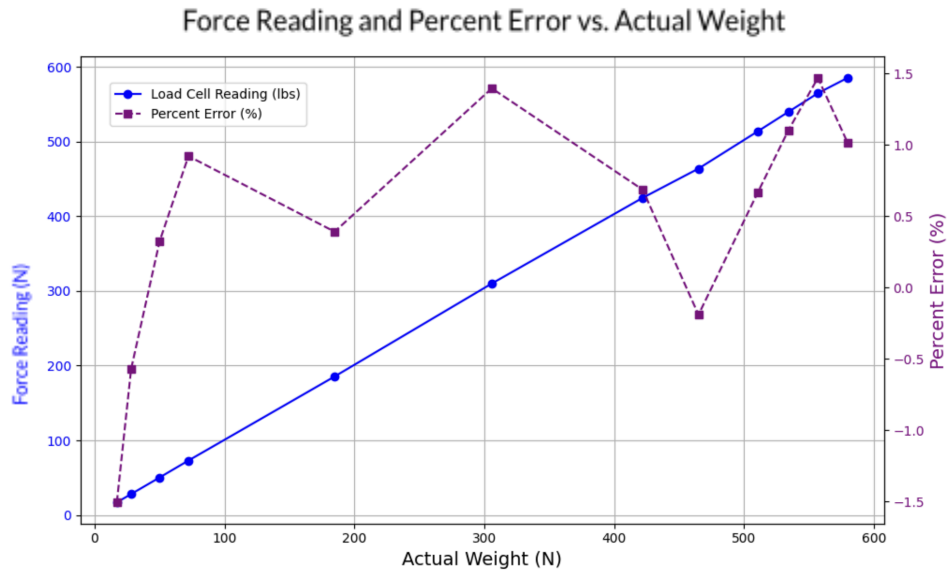


Figure 35. Force reading and percent error vs actual weight loaded at center of force plate.

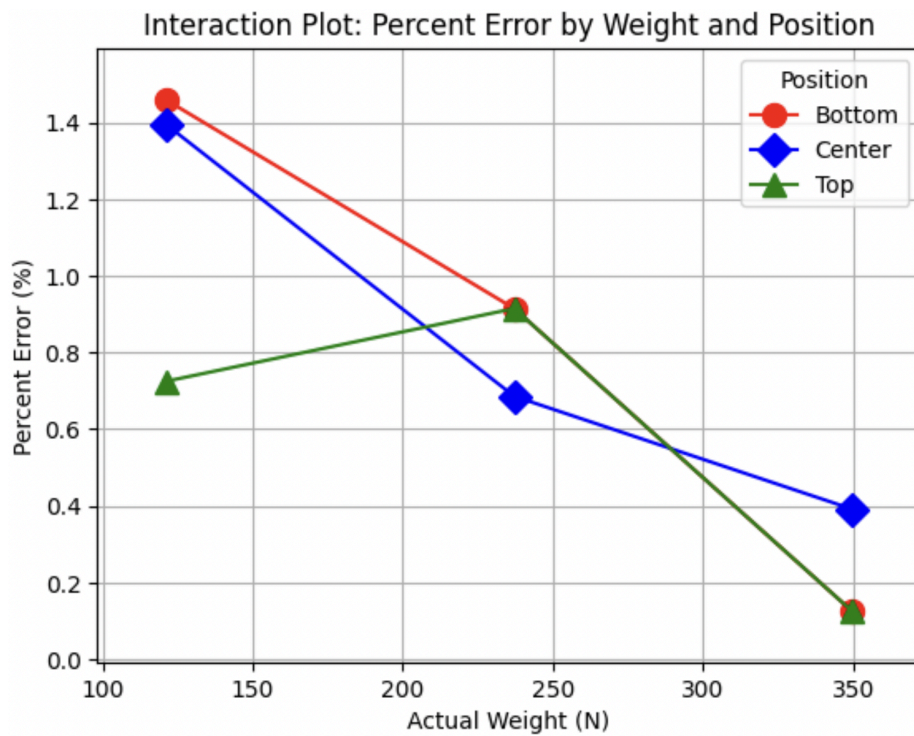


Figure 36. Device percent error at different loads applied in 3 different locations to the top plate.

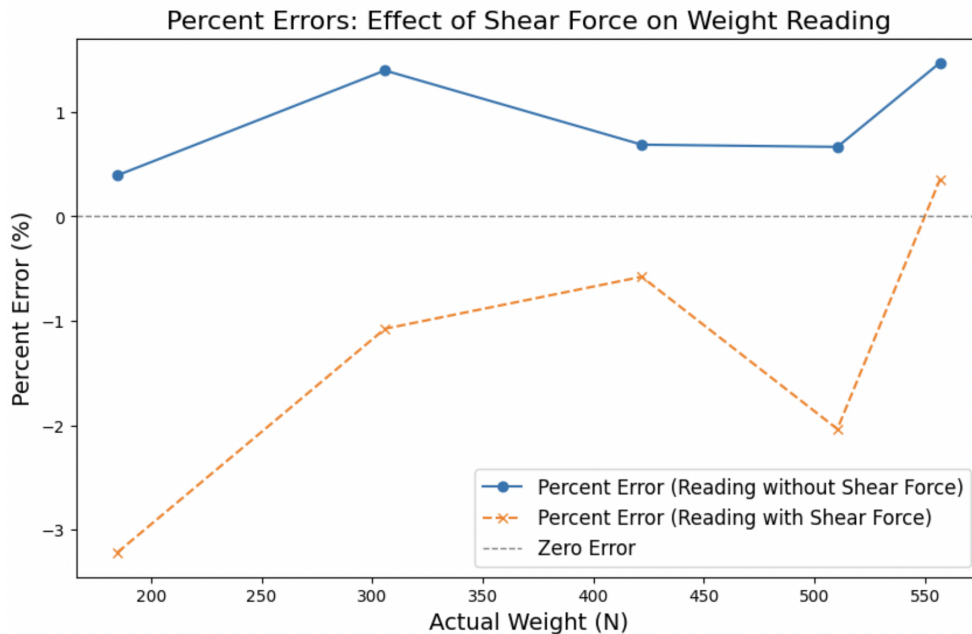


Figure 37. Percent errors of device force reading and actual weight with and without applied shear force.

VII. Discussion

The testing results provide important data relevant to the validity of the force plate design. The force plates are able to measure force accurately to the precision of 1 lb (the client prefers the use of lb rather than N) to an accuracy within the 5% margin of error set by the PDS, even when tested under eccentric and shear loading conditions. These findings confirm the prototype meets client requirements, but additional testing, including repeatability and higher shear load evaluations, is needed for robust validation.

Potential sources of error include non-linearity and hysteresis (up to 1%) inherent to the load cells, as specified by the manufacturer [17]. Although calibration curves show a highly linear relationship between applied force and output, these factors remain relevant as the load cells are dynamically loaded during rowing. Another potential source of error is electrical noise. Electrical noise in the load cells' analog output mitigated by low-pass filters, and minor signal delays caused by capacitor charging times are additional concerns. The testing results showed that shear loads did have a significant effect on the accuracy of the normal force measurement

(while remaining within 5% margin of error). Shear loads induced by the rowing motion could be a significant source of error, which is why extensive testing on the effect of shear loads on accuracy must be more thoroughly investigated. Finally, the friction between the shoulder screw and the sleeve bearing has the potential to distort the force readings by absorbing some of the applied force.

Upon further validation, the prototype could track both short- and long-term force output data for the UW Rowing Team, supporting performance monitoring and providing quantitative benchmarks for injured rowers' safe return to activity. It could also facilitate research on factors like weight class, stroke rate, and experience level in rowing performance, as well as the link between lower extremity asymmetry and common rowing injuries such as lower back pain.

The team has also considered ethical considerations in its design. The design does not infringe on Bylaw 10 in NCAA Division 1 Legislation [18] as it cannot be used to give improper financial aid or banned substances to athletes, and cannot be used in sports wagering. In addition, the device fits well within NCAA regulation on practices or athletically-related activities [19]. The design will also take into account confidentiality of rowers' data in accordance with HIPAA, as rowers can be considered patients of the athletic trainers they work with. HIPAA guarantees that patient data will remain confidential between a patient and their provider [20]. Therefore, rowers' data will be stored in a secure format, only available to only athletes and authorized personnel.

VIII. Conclusions

Asymmetrical load distributions can contribute to lower extremity injuries among competitive, collegiate rowers. Our objective is to design a low-cost force plate that provides real-time force data for the UW-Madison Rowing Team as the athletes train on the ergometer. Our chosen design, the Stationary Force Plate Design, consists of two aluminum plates. The bottom plate will be securely screwed into place, with four single-axis button load cells mounted at each corner. The upper plate, resting on top of the load cells, features set screws that align with each load cell point. To hold the plates together, two shoulder screws go through frictionless sleeve bearings in the center of the bottom plate and screw into the threaded holes in the top plate. The FlexFoot from the ergometer is reattached to the top plate, allowing for adjustability to

accommodate different foot sizes. As force is applied through the foot, the load cells will generate a signal to the Raspberry Pi Pico microcontroller, which will relay the data to a laptop via USB connection showing a detailed GUI. Each load cell used in the prototype was calibrated by placing a known load and linearly correlating it to the ADC value output by the circuit. The device was tested as a whole by loading it in the normal direction with varying force, location of applied load, and applied shear force. Analysis of testing data found that the device meets the PDS criteria of measuring force within a 5% error, but shear loading has a statistically significant effect on accuracy; therefore, further testing is required to fully understand the effects of shear loading on the device. The design will be installed in the ergometers at the UW-Madison Porter Boathouse, enabling athletes and coaches to visualize and analyze real-time data through the GUI. In the future, the team plans to integrate data storage capabilities into this design by adding an optional checkbox to the GUI that will automatically save the force and time data to a file on the user's computer. Further testing of the device under various loading conditions will be conducted to validate the accuracy, repeatability, and reliability of the design before it is adopted by the UW Rowing Team training and physical therapy staff.

IX. References

- [1] S, Arumugam, et al. “Rowing Injuries in Elite Athletes: A Review of Incidence with Risk Factors and the Role of Biomechanics in Its Management.” *Indian Journal of Orthopaedics*, vol. 54, no. 3, Jan. 2020. pubmed.ncbi.nlm.nih.gov, <https://doi.org/10.1007/s43465-020-00044-3>
- [2] Buckeridge, E. M., et al. “Biomechanical Determinants of Elite Rowing Technique and Performance: Rowing Technique and Performance.” *Scandinavian Journal of Medicine & Science in Sports*, vol. 25, no. 2, Apr. 2015, pp. e176–83. DOI.org (Crossref), <https://doi.org/10.1111/sms.12264>.
- [3] “2D_Stretcher,” Biorow. https://biorow.com/index.php?route=product/product&path=61_115&product_id=109 (accessed Sep. 21, 2023).
- [4] Clay, Helen & Mansell, Jamie & Tierney, Ryan. (2016). ASSOCIATION BETWEEN ROWING INJURIES AND THE FUNCTIONAL MOVEMENT SCREEN™ IN FEMALE COLLEGIATE DIVISION I ROWERS. *International journal of sports physical therapy*. 11. 345-349.
- [5] “Lower Back Pain Relief for Rowers.” *Performance Health*, www.performancehealth.com/articles/lower-back-pain-relief-for-rowers. Accessed 11 Oct. 2023.
- [6] Cheri, “Using the Force Curve,” *Concept2*, May 18, 2012. <https://www.concept2.com/indoor-rowers/training/tips-and-general-info/using-the-force-curve> (accessed Sep. 29, 2023).
- [7] “Force Plates,” *Bertec*. <https://www.bertec.com/products/force-plates> (accessed Sep. 13, 2023).
- [8] E. Wittich, “Symmetry - bat logic,” www.batlogic.net, <https://batlogic.net/wp-content/uploads/2017/08/Row360-Issue-008-Symmetry-of-Sweep.pdf> (accessed Oct. 12, 2023).
- [9] R. E. Boykin et al., “Labral injuries of the hip in rowers,” *Clinical Orthopaedics & Related Research*, vol. 471, no. 8, pp. 2517–2522, 2013. doi:10.1007/s11999-013-3109-1

- [10] “Jill Thein-Nissenbaum, Physical Therapy (PT) Program,” UW School of Medicine and Public Health. Accessed: Oct. 06, 2023. [Online]. Available: <https://www.med.wisc.edu/education/physical-therapy-program/faculty-and-staff/jill-thein-nissenbaum/>
- [11] “Tricia De Souza | Women’s Rowing Coach,” Wisconsin Badgers. Accessed: Oct. 06, 2023. [Online]. Available: <https://uwbadgers.com/sports/womens-rowing/roster/coaches/tricia-de-souza/1617>
- [12] “Instrumentation of an ergometer to monitor the reliability of rowing performance.” Accessed: Jan. 30, 2024. [Online]. Available:
- [13] Q. Liu, Y. Dai, M. Li, B. Yao, Y. Xin and J. Zhang, "Real-time processing of force sensor signals based on LSTM-RNN," 2022 IEEE International Conference on Robotics and Biomimetics (ROBIO), Jinghong, China, 2022, pp. 167-171, doi: 10.1109/ROBIO55434.2022.10011703.
- [14] S. Allison, Y. Fujii, and L. M. Wilcox, “Effects of Motion Picture Frame Rate on Material and Texture Appearance,” IEEE Transactions on Broadcasting, vol. 67, no. 2, pp. 360–371, Jun. 2021, doi: 10.1109/TBC.2020.3028276.
- [15] “Service,” Concept2. Accessed: Feb. 05, 2024. [Online]. Available: <https://www.concept2.com/service/monitors/pm3/how-to-use/understanding-stroke-rate>
- [16] R. Taylor, “How to Become an Olympic Rower: Our Ultimate Guide from an Olympian,” www.rowingcrazy.com, Apr. 27, 2023. <https://www.rowingcrazy.com/how-to-become-an-olympic-rower/> (accessed Sep. 22, 2023).
- [17] “Compact Compression Load Cell | FX29,” TE Connectivity. Accessed: Oct. 07, 2024. [Online]. Available: <https://www.te.com/en/product-CAT-FSE0006.html>
- [18] “Legislative Services Database - LSDBi.” Accessed: Oct. 11, 2023. [Online]. Available: <https://web3.ncaa.org/lsdbi/search/bylawView?id=11561>
- [19] “Legislative Services Database - LSDBi.” Accessed: Oct. 11, 2023. [Online]. Available: <https://web3.ncaa.org/lsdbi/search/bylawView?id=9024>
- [20] “Health Insurance Portability and Accountability Act of 1996,” ASPE. <https://aspe.hhs.gov/reports/health-insurance-portability-accountability-act-1996> (accessed Sep.

21, 2023).

X. Appendix

Appendix A: Product Design Specifications



PRODUCT DESIGN SPECIFICATIONS: ASYMMETRICAL FORCE SENSOR FOR ROWING BIOMECHANICS

BME 400, Section 304

Clients: Dr. Jill Thein-Nissenbaum and Ms. Tricia De Souza

Advisor: Dr. David Appleyard

Team Members:

Team Leader: Allicia Moeller

Communicator: Simerjot Kaur

BSAC: Emily Wadzinski

BWIG: Neha Kulkarni

BPAG: Colin Fessenden

Function:

Force sensors have been widely used in sports biomechanics to measure load distribution and center of pressure for the purpose of correcting form and mitigating injuries. However, getting real-time data during rowing is often difficult to obtain in non-clinical settings and may be very expensive to implement, especially due to environmental and equipment-related constraints. Rowing is a rigorous sport that can lead to injuries in the lumbar spine, the shoulders, the knees, and the hips when the right and left lower extremities generate asymmetrical forces [1]. Additionally, this asymmetry is difficult to quantify visually, and current methods include using stationary rowing simulation machines that underestimate the mechanical power required against water currents [2]. Specifically, these current methods of evaluating rowing form focus mainly on upper body metrics such as stroke power and involve studies outside of the rowing environment. Our design aims to provide accurate real-time data of rowers' lower extremities by integrating a force sensor system on an ergometer base to transduce force measurements that can be viewed while rowing against current in a tank or on the stationary ergometer. The application of our design will allow athletes and coaches to assess and adapt athlete performance, identify risk factors for injury, and assess return to injury metrics.

Client Requirements:

- The device must be strong enough to withstand the force exerted by rowers during the drive phase of the stroke, which peaks at 900 N [3].
- The device must accurately measure the load transmitted through each leg and translate the data to an interface that provides real-time data viewing while rowing.
 - The device must display real-time data on the amount of force transmitted by the toe and heel (separately) of each foot onto the tank footplate.
 - The device must store relevant performance metrics from a trial, such as peak force per stroke and time to peak force.
- The frequency and duration of force data storage during rowing sessions must be adjustable.
- The client desires an easily integrated force measuring system that should operate without requiring change in rowing technique or excessive modification of current rowing equipment.
- The device must alert the rower when force exerted by the right and left foot are asymmetrical.

Design Requirements:

1. Physical and Operational Characteristics:

a. Performance Requirements:

- The product must track the degree to which rowers are exerting symmetric force through their entire lower extremity, to track any asymmetry present.
 - The device should quantify the degree of asymmetry using the magnitude of relative force between limbs in Newtons.
- The product should display real-time data during a rower's trial so they can monitor any fluctuations as they occur.
 - The real-time display must be easily interpretable by the user(s) using simple visual cues like colors, lights, figures, and text.
- The product should be able to store data so coaches and rowers can see the data in real time and analyze it later.

b. Safety:

- This product should not disrupt the motion of the rower or the ergometer as a stroke is completed.
- This product should not cause any electrical shocks to the rower's and have minimal large cords in close proximity to the rower. The device needs to be plugged into an outlet with standard voltage of 120 V [4].
- This product should be able to be cleaned between uses with alcohol-based solution or soap and water. Bleach and/or hydrogen peroxide should be avoided [5].
- This product should not have any sharp edges.

c. Accuracy and Reliability:

- The device should be made with easily available parts such that they are replaceable in the event of malfunction or failure.
- The product should display and store data with high accuracy with a margin of error at 5% [6].
- The product must have no more than a 0.5 second delay between a rower's stroke and the real-time display so as to provide feedback at least once per stroke [7].

d. Life in Service:

- The NCAA in-season hourly practice limitation is no more than 20 hours per week and roughly 8 months out of the year or about 34 weeks [8].
- The product should remain functional for the duration of a full collegiate rowing career. The typical career of a collegiate rower is 4 years. This equates to roughly 6,800 - 8,160 hours.
- The Concept2 RowERG® requires all screws and connections to be thoroughly checked every 250 hours of use [7]. The product's connections and integrity should be checked concurrently.

e. Shelf Life:

- The average lifespan of a load cell is around 10 years with proper usage, maintenance, and protection [9].
- The appropriate range of ambient temperature for load cell storage is from -10°C - 40°C [10].

f. Operating Environment:

- The client would like this device to be compatible with the ergometer next to the tank, as well as ergometers in the training room, which exist in room temperature conditions. These conditions are around 20-22° C and low humidity.
- An outlet or extension cord should be provided in the room to power the device.

g. Ergonomics:

- Display
 - The display will be at eye level from the rower as they are rowing, roughly 1.1 m from the ground [11].
 - The feedback will be easy to interpret quickly, so that the rower can quickly adjust their form.
- Force Plate
 - The plates will not add any unnatural feeling for the rowers, and therefore they will not have to change their technique in order to use them.
 - The force plate will be mounted flat onto the existing ergometer footplate.
 - The force plate must be compatible with different foot sizes.

h. Size:

- Display
 - The visual display should be at least 12 cm wide and 6.75 cm tall so that the screen size allows alphanumeric text to be 10 mm tall (*see Standards and Specifications*).

- Force Plate
 - The width of a singular footplate of the 2005 Concept2 Ergometer Model D in the rowing tank is 13.3 cm and the height is 30.7 cm. The force plate must be the same size or smaller than these dimensions to fit on top of the foot plate.
 - The average 200kg load cell thickness is between 10-35 mm [12][13]. Therefore the thickness of the product should not be thicker than 35mm in order to maintain a relatively level surface and not impede upon the toe or heel straps of the Flexfoot.

i. Weight:

- Maximum user weight for the RowERG is 227 kg [1]. The weight range of a woman crew athlete is on average 50 - 84 kg [14]. To not exceed this scale, the product weight should not exceed 143 kg.

k. Materials:

- A strain gauge load cell will be used for measuring force in a force plate to provide a greater surface area for force distribution applied by the foot. The chosen strain gauge load cell will operate by measuring electrical resistance changes in response to applied strain or pressure on the load cell. This load cell should accurately assess and withstand weights of 200 kg applied while rowing based on surface strain. [15]
- Additionally, housing material for load cells should be safe to use in a sports testing environment and be in compliance with the Sports and Recreational Equipment General Safety Requirements (*see Standards and Specifications*)
- A load cell amplifier compatible with the chosen strain gauge load cells will be utilized and have an operation voltage of 5 Volts.
 - Will be used to amplify signals from the load cells for accurate weight measurements. It will also be compatible with microcontrollers for data acquisition. [16]
- A display screen such as a TV monitor, tablet, or laptop will be used to display rowers' data, as these screens are readily available in the UW Boathouse.

l. Aesthetics, Appearance, and Finish:

- Display
 - The visual display must have a frame rate of at least 24 Hz, which is the standard frame rate of motion pictures, so that changes on the display appear continuous to the human eye [17].

- Force Plate
 - The constructed force plate should have clean lines and match the neutral gray and black colors of the ergometer so that it blends in as an attachment.
- Any hardware or electronics used to connect the force plates to the display should be hidden in an electronics box, to maintain a neat appearance.

2. Product Characteristics:

a. Quantity:

- The team aims to fabricate one functioning prototype this semester, consisting of a right and left force plate connected to a display screen. In the future, the client would like a total of 8 prototypes for the 8 ergometers fit to the tank.

b. Target Product Cost:

- The budget for this design project is \$500. The budget may be increased with approval from the UW Athletic Department.

3. Miscellaneous:

a. Standards and Specifications :

- The device must not interfere with the construction of the Concept2 RowErg® such that it fails to comply with the ASTM Standard Specifications for Fitness Equipment (ASTM F2276 – 23) [18].
 - Specifies that edges should be free of burrs and sharp edges, and corners should be chamfered
 - Specifies that the ergometer should withstand 1560 on/off cycles
 - Specifies that the footplate should be slippage-resistant
 - Specifies that the ergometer should be able to withstand 136 kg or the maximum user weight, whichever is greater
- The device must also comply with the ASTM Standard Specification for Universal Design of Fitness Equipment for Inclusive Use by Persons with Functional Limitations and Impairments (ASTM 3021-17), such that rowers with functional limitations and impairments can use the device [19].
 - Specifies that color contrast on any visual display must be greater than or equal to 70%

- Specifies that font size should be at least 10 mm
- Specifies that the display should continue to display visual feedback at least 5 seconds after exercise has stopped.
- The device must comply with the Sports and Recreational Equipment General Safety Requirements (ISO 20957) to enhance safety and reliability of athletic testing equipment [20].
 - It includes guidelines for mechanical strength and endurance testing to ensure material can withstand forces applied during athlete testing.

b. Customer:

- The primary target customer for the product is the Physical Therapist and Athletic Training Staff for the University of Wisconsin Rowing Team.
 - University of Wisconsin collegiate rowers will be the primary operators of the device during use.
 - The device will also be used by the coaching staff of the University of Wisconsin Rowing Team.
- The customer(s) will use the device for routine evaluation of rowers' form, diagnosis of injury, and assessing progress during rehabilitation and return from injury.
 - Quantitative markers of asymmetry are required for determining the degree of injury and stage of progress during rehabilitation.
 - Positional placement must be adjustable between the ergometer and port or starboard sides of the tank, as well as between different models of ergometers.

c. Patient-Related Concerns:

- The device should not interfere with proper rowing technique or injure the athlete in any way.
- The device should not interfere with the ergometer or boat such that they begin to degrade or malfunction.
- The device should be accompanied by a data storage drive or other technology that allows for patient performance data to be stored confidentially, in compliance with HIPAA [19].
 - The storage drive must be able to store multiple runs of longer rowing sessions between 40-100 minutes.

d. Competition:

- Bertec® produces portable force plates for gait, balance, and performance analysis [21].
 - The load cells contained inside utilize strain gauges and transducers to measure forces and moments in the x, y, and z directions

- The portable force plates have a sampling frequency of 1000 Hz.
 - The portable force plates have loading capacities of 4440, 8880, or 17760 N.
- Biorow produces a 2D force sensor that uses four load cells fixed to a plate, and the plate is screwed between the foot straps of the ergometer and the foot stretchers [22].
 - The load cells can measure from -800 to +3200 N.

References:

- [1] S. Arumugam, P. Ayyadurai, S. Perumal, G. Janani, S. Dhillon, and K. A. Thiagarajan, "Rowing Injuries in Elite Athletes: A Review of Incidence with Risk Factors and the Role of Biomechanics in Its Management," *Indian J Orthop*, vol. 54, no. 3, pp. 246–255, Jan. 2020, doi: 10.1007/s43465-020-00044-3.
- [2] G. Treff, L. Mentz, B. Mayer, K. Winkert, T. Engleder, and J. M. Steinacker, "Initial Evaluation of the Concept-2 Rowing Ergometer's Accuracy Using a Motorized Test Rig," *Frontiers in Sports and Active Living*, vol. 3, Jan. 2022, doi: <https://doi.org/10.3389/fspor.2021.801617>.
- [3] "Instrumentation of an ergometer to monitor the reliability of rowing performance." Accessed: Jan. 30, 2024. [Online]. Available: <https://www.tandfonline.com/doi/epdf/10.1080/026404197367434?needAccess=true>
- [4] "Site Home - Global Site," *Stanford.edu*, 2017. <https://simtk-confluence.stanford.edu:8443/> (accessed Sep. 22, 2023).
- [5] "How should I clean or disinfect the Force Plate?," *success.spartascience.com*. <https://success.spartascience.com/en/knowledge/how-should-i-clean-or-disinfect-the-force-plate> (accessed Sep. 22, 2023).
- [6] Q. Liu, Y. Dai, M. Li, B. Yao, Y. Xin and J. Zhang, "Real-time processing of force sensor signals based on LSTM-RNN," *2022 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Jinghong, China, 2022, pp. 167-171, doi: 10.1109/ROBIO55434.2022.10011703.
- [7] "Service," Concept2. Accessed: Feb. 05, 2024. [Online]. Available: <https://www.concept2.com/service/monitors/pm3/how-to-use/understanding-stroke-rate>
- [8] "What is the rowing training volume of elite programs? - Sparks." Accessed: Feb. 07, 2024. [Online]. Available: <https://sparksrowing.com/blog/what-is-the-rowing-training-volume-of-elite-programs>
- [9] "Aluminum Alloy Load Cell, Steel Alloy Load Cell, Stainless Steel Load Cell - www.mavin.cn." Accessed: Feb. 07, 2024. [Online]. Available: https://www.mavin.cn/blog/the-life-span-and-wiring-code-of-load-cell_b6
- [10] G. Mattingly, J. Garner, and M. Whitaker, "Observed Temperature Effects on Load Cells," Oak Ridge National Laboratory (ORNL), Oak Ridge, TN (United States), Jul. 2017. Accessed: Feb. 07, 2024. [Online]. Available: <https://www.osti.gov/biblio/1479805>
- [11] "Concept2 RowErg." Accessed: Feb. 05, 2024. [Online]. Available: <https://shop.concept2.com/rowergs/298-model-d-with-pm5.html>
- [12] "TinyTronics." Accessed: Feb. 08, 2024. [Online]. Available: <https://www.tinytronics.nl/shop/>
- [13] "Sentran LLC | Load cells, force transducers and weighing systems solutions." Accessed: Feb. 08, 2024. [Online]. Available: <https://www.sentranllc.com/products/load-cells/low-profile.htm>
- [14] "The Height & Weight of Female Rowers," SportsRec. Accessed: Feb. 07, 2024. [Online]. Available: <https://www.sportsrec.com/6665309/the-height-weight-of-female-rowers>

- [15] M. #210, M. #779596, and M. #1287021, “Load cell - 200kg, disc (TAS606),” SEN-13332 - SparkFun Electronics, <https://www.sparkfun.com/products/13332> (accessed Feb. 7, 2024)
- [16] “Sparkfun load cell amplifier - HX711,” SEN-13879 - SparkFun Electronics, <https://www.sparkfun.com/products/13879> (accessed Feb. 7, 2024).
- [17] S. Allison, Y. Fujii, and L. M. Wilcox, “Effects of Motion Picture Frame Rate on Material and Texture Appearance,” *IEEE Transactions on Broadcasting*, vol. 67, no. 2, pp. 360–371, Jun. 2021, doi: 10.1109/TBC.2020.3028276.
- [18] “Standard Specification for Universal Design of Fitness Equipment for Inclusive Use by Persons with Functional Limitations and Impairments” <https://www.astm.org/f3021-17.html> (accessed Sep. 20, 2023).
- [19] “Health Insurance Portability and Accountability Act of 1996,” ASPE. <https://aspe.hhs.gov/reports/health-insurance-portability-accountability-act-1996> (accessed Sep. 21, 2023).
- [20] “ISO/DIS 20957-1(en) General safety requirements and test methods,” ISO, <https://www.iso.org/obp/ui/en/#iso:std:81908:en>
- [21] “Force Plates,” Bertec. <https://www.bertec.com/products/force-plates> (accessed Sep. 13, 2023).
- [22] “2D_Stretcher,” Biorow. https://biorow.com/index.php?route=product/product&path=61_115&product_id=109 (accessed Sep. 21, 2023).

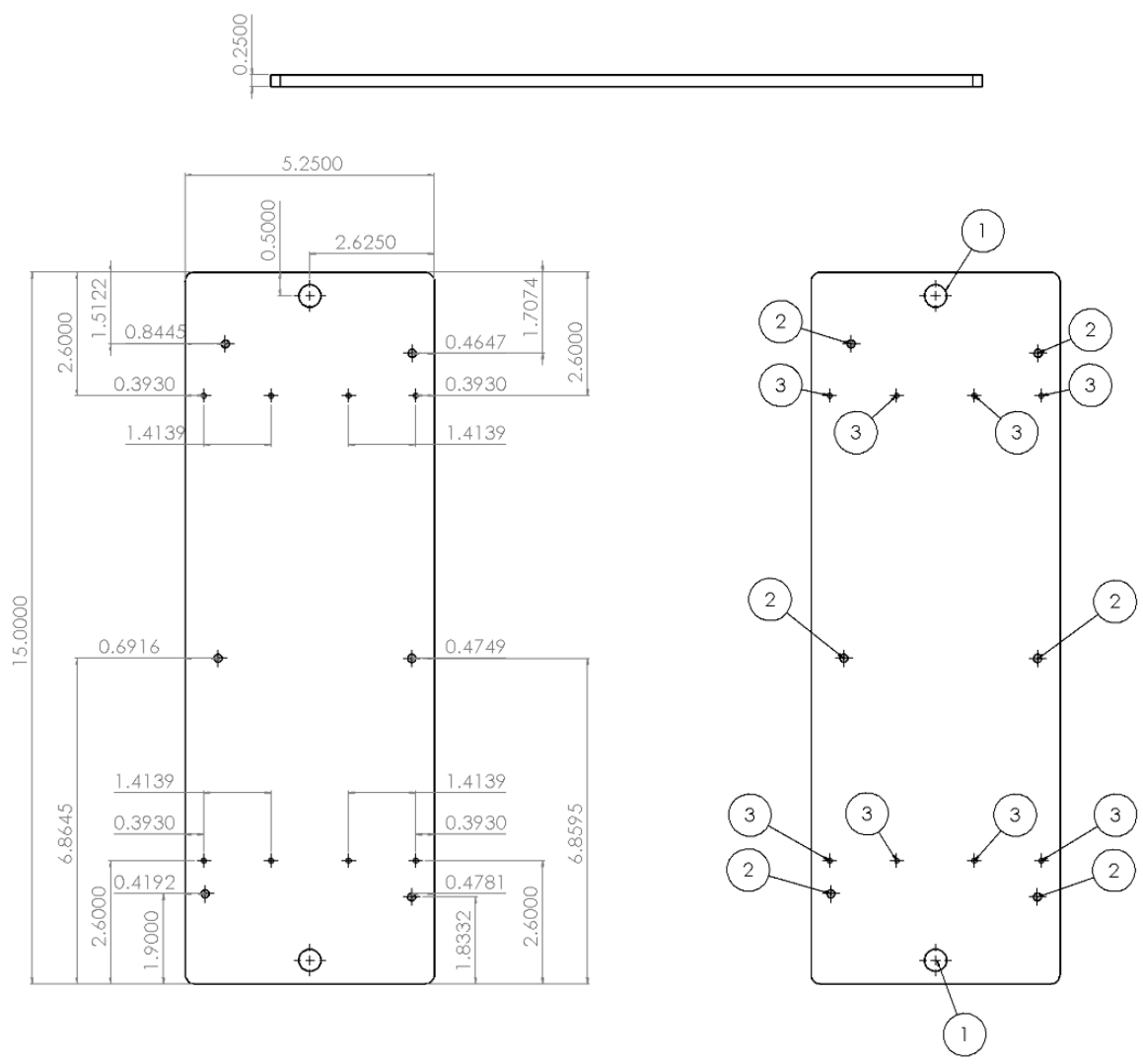
Appendix B: Materials and Expenses

Table 1. Materials and Expenses for Final Design.

Description	Item #	Specs	Link	Price	Qty	Item Total
Alloy Steel Sleeve Shoulder Screw	91259A632	3/8" Shoulder Diameter	Link	\$2.86	4	\$11.44
PTFE Sleeve Bearing Shell	60695K2	3/8" OD 0.5" Length	Link	\$2.44	4	\$9.76
TI Connectivity Compression Load Cells	824-FX292X-1 00A0100L	100lb Operating Force	Link	\$28.43	8	\$227.44
12 BIT MCP3208 ADC	MCP3208-CI/P -ND	12 Bit IC ADC	Link	\$4.97	2	\$9.94
TLV274IN	296-14379-5-N D	Op Amp 4 Circuit	Link	\$1.06	10	\$10.58
1K Ohm Resistors	RNF14FTD1K0 0	1k ohm resistors	Link	\$0.03	100	\$3.15
Raspberry Pi Pico H	2648-SC0917- ND	microcontroll er	Link	\$5	1	\$5.00
Stainless Steel Flat-Tip Set Screws (Pack of 25)	94355A337	10-32 0.5" Long	Link	\$5	1	\$5.41
Aluminum Footplates	6061 T651	12x16x.25	Link	\$31	2	\$61.16
Custom Printed Circuit Boards		95x95mm (5 copies)	Link	\$10.61	1	\$10.61
Compression Spring	9657K374	124 max load, 1.75" L	Link	from last year	12	
					Total	\$354.49

Appendix C: SOLIDWORKS Drawings

Bottom Plate Drawing:



HOLES:

- 1: 0.46785" DIAMETER THRU HOLE
- 2: #8/32 CLEARANCE HOLE
- 3: M3X0.5 TAPPED HOLE

Figure 1. SolidWorks drawing of bottom plate with hole dimensions.

Top Plate Drawing:

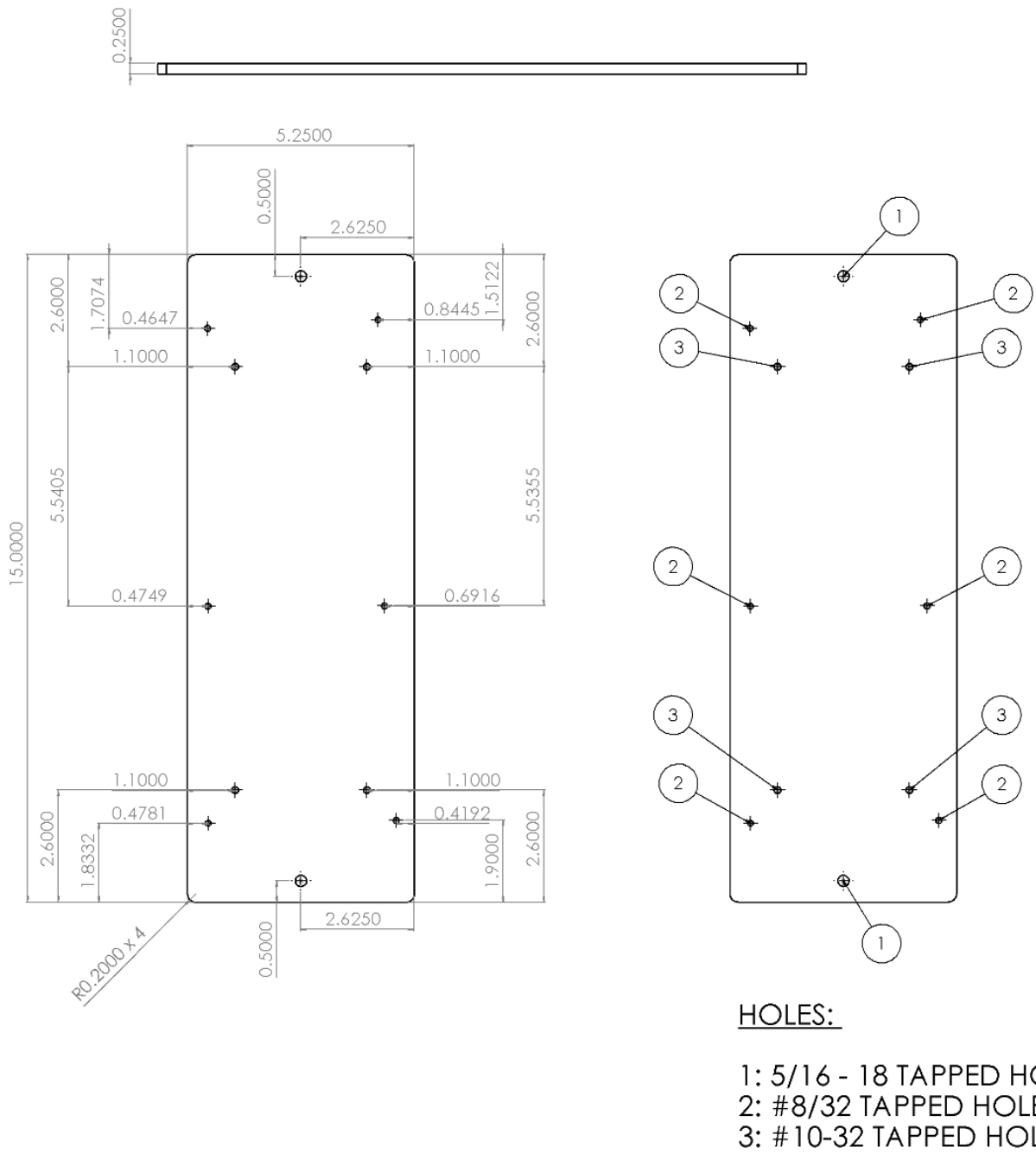


Figure 2. SolidWorks drawing of top plate with hole dimensions.

Appendix D: Fabrication Protocol

For a detailed list of materials, refer to Appendix B.

Footplates:

1. Use a jigsaw to cut the aluminum plates to 15" x 5.25" x 0.25" as shown in the drawings in Appendix C.
2. Deburr and fillet the edges of the plates using the radius specified in Appendix C.
3. Use a CNC mill to drill and tap the holes specified by the drawings in Appendix C for two bottom plates and two top plates.

Circuitry:

1. Obtain a soldering iron and solder wire.
2. Turn on the iron to 750 degrees Fahrenheit.
3. Solder the components to the PCB boards at their corresponding locations (refer to Figure 23).
4. Trim excess wire from the resistors and capacitors.
5. Flatten the capacitors against the board to minimize the height of the board.
6. Repeat with a second PCB board.

Assembly:

1. Unscrew the left and right Concept 2 RowErg Model D Flexfoot from its base.
2. Attach the PCB boards to the center of each bottom aluminum plate using loops of electrical tape.
3. Plug in four load cells to their corresponding spots on each board.
4. Place the load cell housings over each load cell and screw them in with eight M3-0.5 screws per plate.
5. Attach the bottom plates with the circuitry to the existing holes on the ergometer using the screws from the removed FlexFoot.
6. Place the Raspberry Pi Pico between the feet of the erg and connect it to both PCBs.
7. Attach the removed FlexFoots onto the two top aluminum plates with twelve total #10-32 screws.
8. Place the top plates over the bottom plates.
9. Lubricate the shoulder screws with WD-40 and insert them through the bearings of the bottom plate and screw them into the top plate.
10. Connect the Pico to a computer or monitor.

Appendix E: Testing Protocol

Calibration Protocol:

1. Place a 1 kN load cell attachment into the MTS machine. Attach the top and bottom compression platens.
2. Move the MTS machine down until the compression platens are 0.2 inches apart (measure with a ruler) and set the emergency stopper.
3. Open the MTS software. Right click on the current force reading and click zero sensor.
4. Attach the FX29 load cell you intend to calibrate to the PCB into the intended channel of use. Attach the PCB to power so that it can print out the raw ADC values of all 4 channels.
5. Make note of the ADC value measured by the load cell when no force is applied to it. This will be the offset value.
6. Move the MTS machine up a few inches and place the FX29 load cell on the bottom compression platen.
7. Very carefully, and very slowly move the top platen down with the fine adjustment wheel while checking the force reading and stop moving the wheel downward once some force is applied. The displacement of the FX29 load cell at 100 lb is 0.03 mm, so it is essential that you move the fine adjustment wheel very carefully as it does not take much displacement to put large amounts of force on the load cell.
8. Use the fine adjustment wheel to adjust the MTS machine until the software reads a force value that is close to 44.5 N. Record the precise force measurement from the MTS software as well as the ADC value of the FX29 load cell.
9. Repeat step 8 with the additional force values listed in Table 3.
10. Once data has been collected for all force values, subtract all the ADC values by the offset value from step 5. Then, run a linear regression using the actual force readings as y-values and the ADC values as x-values, so that the slope of the best fit line provides a calibration coefficient in the form of force/ADC.
11. Repeat steps 4-10 on the remaining 7 sensors, taking care to attach every sensor to its intended channel of use so that the calibration accounts for differences in gain across different channels due to resistor tolerance.

Table 2. Calibration data recording table.

Load Cell:		ADC Offset:	
Ideal Force Values (N)	Actual Force Values (N)	ADC Value	ADC Value - Offset
44.4822			

111.2055			
222.411			
333.6165			
444.822			

Testing Protocol:

1. Obtain a set of calibrated weights.
2. Set up one assembled plate on a table with supports, connected to a computer with the code printing force to the nearest tenth.
3. Place a pointed bar onto the middle of the plate that can apply loads while weights are adjusted.
4. Zero out the plate with the bar on it.
5. Place a 0.539 kg weight onto the bar and record the weight.
6. Add another weight of 1.175kg and record the reading.
7. Continue adding weights to the loading mechanism and record the printed readouts in a spreadsheet.
8. Apply a shear load of 2.319 kg using a cable and pulley at the same point of the normal force at five various total weights: 18.841 kg, 31.18 kg, 43.023 kg, 52.044kg, 56.772 kg. Record any differences in printouts.
9. Apply three various weights to the top of the plate and record the plate's readings of 12.339 kg, 24.182 kg, 35.652 kg.
10. Repeat with at the bottom of the plate.

Table 3. Weights Used in Testing Consecutively.

Weight Added (kg)	Total Weight (kg)
0.539	0.539
1.175	1.75
1.16	2.874
2.262	5.109
2.262	7.371
11.47	18.841

12.399	31.18
11.843	43.023
4.377	47.4
4.644	52.044
2.421	54.465
2.307	56.772
2.319	59.091

Appendix F: Code

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import sys
import math
from random import randint
import pandas as pd
from PySide6 import QtGui, QtWidgets, QtCore
from PySide6.QtCore import QThread, Signal, QMutex, QMutexLocker
from PySide6.QtWidgets import QApplication, QMainWindow, QDialog, QLabel, QVBoxLayout,
QWidget, QGraphicsPathItem, QMessageBox, QErrorMessage
from PySide6.QtCore import QTimer # Change to PySide6
import pyqtgraph as pg
from pyqtgraph.Qt import QtCore # This is fine as it uses the appropriate PyQtGraph module
import numpy as np
import serial
# from PyQt5.QtCore import QThread, pyqtSignal
# from PyQt5.QtWidgets import QMainWindow, QVBoxLayout, QWidget, QPushButton, QLabel,
QMessageBox
# from PyQt5.QtCore import QTimer
import serial.tools.list_ports
import time

# In[2]:

# Important:
# You need to run the following command to generate the ui_form.py file
# pyside6-uic form.ui -o ui_form.py, or
# pyside2-uic form.ui -o ui_form.py
from ui_form import Ui_MainWindow

# Serial Reader Thread

# In[3]:
```



```
class SerialReaderThread(QThread):
    data_received = Signal(str, int) # Signal for channel and value

    def __init__(self, port, baudrate):
        super().__init__()
        self.port = port
        self.baudrate = baudrate
        self.running = False

    def run(self):
        try:
            self.serial = serial.Serial(self.port, self.baudrate, timeout=1)
            self.running = True
            while self.running:
                if self.serial.in_waiting:
                    line = self.serial.readline().decode('utf-8').strip()
                    if line:
                        try:
                            channel, value = line.split(':')
                            channel = channel.strip()
                            value = int(value.strip())
                            # print("data available")
                            self.data_received.emit(channel, value)
                            # print("data emitted")
                        except ValueError:
                            continue
        except serial.SerialException as e:
            print(f"Error: {e}")
            # error_dialog = QMessageBox()
            # error_dialog.showMessage("Serial Connection Lost")
            # error_dialog.exec_()
            # self.running = False

    def stop(self):
        self.running = False
        if self.serial and self.serial.is_open:
            self.serial.close()

### ABS LINE

# In[4]:
```

```

class AbsLineMainWindow(QMainWindow):
    closed_signal = Signal()
    def __init__(self):
        super().__init__()

        # Create lists to store the last 10 seconds of data along with timestamps
        self.force_left = []
        self.force_right = []

        global left_tare
        global right_tare

        # Calibration factors for each channel (adjust as needed)
        self.calibration_factors = {
            '1A': 0.05128183369,
            '1B': 0.05140563617,
            '1C': 0.05226255289,
            '1D': 0.05096242992,
            '2A': 0.05121516838,
            '2B': 0.05140285445,
            '2C': 0.05292458815,
            '2D': 0.05250629783
        }

        self.serial_thread = None
        self.channel_values = {}

        self.plot_graph = pg.PlotWidget()
        self.setCentralWidget(self.plot_graph)
        self.plot_graph.setBackground("w")
        self.plot_graph.setTitle(
            "<span style='color: black; font-size: 20pt; font-weight: bold;'>Force vs Time</span>"
        )
        self.plot_graph.showGrid(x=True, y=True)

        timestyles = {"color": "black", "font-size": "20px", "font-weight": "bold"}
        forcestyles = {"color": "black", "font-size": "20px", "font-weight": "bold"}
        self.plot_graph.setLabel("left", "Force (lbs)", **forcestyles)
        # self.plot_graph.setLabel("bottom", "Time", units=None, **timestyles)

        # Set axes range
        self.plot_graph.setXRange(0, 10)
        self.plot_graph.setYRange(-100, 300)

```

```

# Define pens for graph lines
rightpen = pg.mkPen(color=(255, 165, 0), width=4)
leftpen = pg.mkPen(color=(0, 0, 255), width=4)

# Initialize graph lines
self.right_line = self.plot_graph.plot([], [], name="Right", pen=rightpen)
self.left_line = self.plot_graph.plot([], [], name="Left", pen=leftpen)

# Add legend
legend = pg.LegendItem(offset=(50, -50))
legend.setParentItem(self.plot_graph.getViewBox())
legend.addItem(self.right_line, "Right")
legend.addItem(self.left_line, "Left")

# Timer setup to update plot
self.plot_timer = QTimer(self)
self.plot_update_freq = 24
self.plot_timer.setInterval(1000 / self.plot_update_freq) # 10 Hz
self.plot_timer.timeout.connect(self.update_plot)
self.plot_timer.start()

self.index = 0
self.display_window = 10 # Duration in seconds to show on the graph
self.start_monitoring()

def autodetect_com_port(self):
    """Auto-detect the Raspberry Pi Pico's COM port."""
    ports = serial.tools.list_ports.comports()
    for port in ports:
        if "USB" in port.description:
            return port.device
    return None

def start_monitoring(self):
    com_port = self.autodetect_com_port()
    if not com_port:
        # QMessageBox.critical(self, "Error", "No Raspberry Pi Pico detected!")
        return
    print("port found")

    baudrate = 115200

# Start the serial reader thread

```

```

self.serial_thread = SerialReaderThread(com_port, baudrate)
self.serial_thread.data_received.connect(self.update_channel_value)
self.serial_thread.start()

def update_channel_value(self, channel, value):
    """Update the value for a specific channel and compute sums for each group with
    calibration."""
    if channel in self.calibration_factors:
        calibrated_value = value * self.calibration_factors[channel]
        self.channel_values[channel] = calibrated_value
        self.compute_sums()

def compute_sums(self):
    global left_tare
    global right_tare
    """Compute the sum of values for each group of channels and maintain only the last 10
    seconds of data."""
    group1_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '1') - right_tare, 4)
    group2_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '2') - left_tare, 4)

    # Append new data with timestamps
    current_time = time.time()
    self.force_right.append((current_time, float(group1_sum)))
    self.force_left.append((current_time, float(group2_sum)))

    # Keep only the last 10 seconds of data
    self.force_right = [(t, val) for t, val in self.force_right if current_time - t <=
self.display_window]
    self.force_left = [(t, val) for t, val in self.force_left if current_time - t <= self.display_window]

def update_plot(self):
    """Update the plot to show the last 10 seconds of data."""
    if self.force_left and self.force_right:
        # Extract timestamps and values for plotting

        times_right = [t for t, _ in self.force_right]
        values_right = [val for _, val in self.force_right]
        times_left = [t for t, _ in self.force_left]
        values_left = [val for _, val in self.force_left]

        # Ensure times are sorted
        times_right, values_right = zip(*sorted(zip(times_right, values_right)))

```

```

times_left, values_left = zip(*sorted(zip(times_left, values_left)))

# Set x-axis range to the last 'display_window' seconds
if times_right and times_left:
    start_time = max(times_right[0], times_left[0])
    end_time = times_right[-1] if times_right[-1] > times_left[-1] else times_left[-1]
    self.plot_graph.setXRange(start_time, end_time)

# Update the plots
self.right_line.setData(times_right, values_right)
self.left_line.setData(times_left, values_left)

self.plot_graph.getAxis('bottom').setTicks([])

# Check for asymmetry and update background color
raw_asym = abs(values_left[-1] - values_right[-1])
if max(values_left[-1], values_right[-1]) != 0:
    if (values_left[-1] > 30) & (values_right[-1] > 30):
        percent_asym = raw_asym / max(values_left[-1], values_right[-1])
        if percent_asym >= 0.10:
            self.plot_graph.setBackground("r")
        else:
            self.plot_graph.setBackground("w")
    else:
        self.plot_graph.setBackground("w")
else:
    self.plot_graph.setBackground("w")

def closeEvent(self, event):
    """Handle application close."""
    self.closed_signal.emit()
    if self.serial_thread and self.serial_thread.isRunning():
        self.serial_thread.stop()
        self.serial_thread.wait()
    event.accept()

# ## ABS BAR

# In[5]:

class absbar_MainWindow(QMainWindow):
    closed_signal = Signal()

```

```

def __init__(self):
    super().__init__()

    # Create lists to store the last 10 seconds of data along with timestamps
    self.force_left = None
    self.force_right = None

    global left_tare
    global right_tare

    # Calibration factors for each channel (adjust as needed)
    self.calibration_factors = {
        '1A': 0.05128183369,
        '1B': 0.05140563617,
        '1C': 0.05226255289,
        '1D': 0.05096242992,
        '2A': 0.05121516838,
        '2B': 0.05140285445,
        '2C': 0.05292458815,
        '2D': 0.05250629783
    }

    self.serial_thread = None
    self.channel_values = {}

    # set up central widget
    self.central_widget = QWidget()
    self.setCentralWidget(self.central_widget)
    self.layout = QVBoxLayout(self.central_widget)

    # create plot widget
    self.plot_widget = pg.PlotWidget()
    self.layout.addWidget(self.plot_widget)
    self.plot_widget.setBackground("w")
    self.plot_widget.setTitle("<span style='color: black; font-size: 20pt; font-weight: bold;'>Force  
by Leg</span>")
    forcestyles = {"color": "black", "font-size": "20px", "font-weight": "bold"}
    self.plot_widget.setLabel("left", "Force (lbs)", **forcestyles)

    self.plot_timer = QTimer(self)
    self.plot_update_freq = 24
    self.plot_timer.setInterval(1000 / self.plot_update_freq) # 10 Hz
    self.plot_timer.timeout.connect(self.update_plot)
    self.plot_timer.start()

```

```

self.index = 0
self.start_monitoring()

def autodetect_com_port(self):
    """Auto-detect the Raspberry Pi Pico's COM port."""
    ports = serial.tools.list_ports.comports()
    for port in ports:
        if "USB" in port.description:
            return port.device
    return None

def start_monitoring(self):
    com_port = self.autodetect_com_port()
    if not com_port:
        # QMessageBox.critical(self, "Error", "No Raspberry Pi Pico detected!")
        return
    print("port found")

    baudrate = 115200

    # Start the serial reader thread
    self.serial_thread = SerialReaderThread(com_port, baudrate)
    self.serial_thread.data_received.connect(self.update_channel_value)
    self.serial_thread.start()

def update_channel_value(self, channel, value):
    """Update the value for a specific channel and compute sums for each group with
    calibration."""
    if channel in self.calibration_factors:
        calibrated_value = value * self.calibration_factors[channel]
        self.channel_values[channel] = calibrated_value
        self.compute_sums()

def compute_sums(self):
    global left_tare
    global right_tare
    """Compute the sum of values for each group of channels and maintain only the last 10
    seconds of data."""
    group1_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '1') - right_tare, 4)
    group2_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '2') - left_tare, 4)

```

```

# Append new data with timestamps
self.force_right = float(group1_sum)
self.force_left = float(group2_sum)

def update_plot(self):
    # Simulate streaming data by updating the index
    threshold = 20
    if (self.force_left != None) & (self.force_right != None):
        self.plot_widget.clear()
        self.plot_widget.setYRange(-100, 300)
        self.bar_graph = pg.BarGraphItem(x=[2, 3], height=[self.force_left, self.force_right], width
= 0.5)
        self.asym = self.force_right - self.force_left
        if self.force_left - self.force_right > threshold:
            brushes = ['r', 'black']
            pens = brushes
        elif self.force_right - self.force_left > threshold:
            brushes = ['black', 'r']
            pens = brushes
        else:
            brushes = ['g', 'g']
            pens = brushes
        self.bar_graph.setOpts(brushes=brushes, pens=pens)
        self.plot_widget.addItem(self.bar_graph)
        self.plot_widget.getAxis("bottom").setTicks([(2, "Left"), (3, "Right")])

def closeEvent(self, event):
    """Handle application close."""
    self.closed_signal.emit()
    if self.serial_thread and self.serial_thread.isRunning():
        self.serial_thread.stop()
        self.serial_thread.wait()
    event.accept()

# ## DIFF LINE

# In[6]:

class diffline_MainWindow(QMainWindow):
    closed_signal = Signal()
    def __init__(self):
        super().__init__()

```



```

# Create lists to store the last 10 seconds of data along with timestamps
self.force_left = []
self.force_right = []

global left_tare
global right_tare

# Calibration factors for each channel (adjust as needed)
self.calibration_factors = {
    '1A': 0.05128183369,
    '1B': 0.05140563617,
    '1C': 0.05226255289,
    '1D': 0.05096242992,
    '2A': 0.05121516838,
    '2B': 0.05140285445,
    '2C': 0.05292458815,
    '2D': 0.05250629783
}

self.serial_thread = None
self.channel_values = {}

# Set up graph
self.plot_graph = pg.PlotWidget()
self.setCentralWidget(self.plot_graph)
self.plot_graph.setBackground("w")
self.plot_graph.setTitle("<span style='color: black; font-size: 20pt; font-weight: bold;'>Force
Difference vs Time</span>")

# Identify styles for axes
timestyles = {"color": "black", "font-size": "20px", "font-weight": "bold"}
forcestyles = {"color": "black", "font-size": "20px", "font-weight": "bold"}
# self.plot_graph.setLabel("left", "Time (s)", **timestyles)
self.plot_graph.setLabel("bottom", "Force (lbs)", **forcestyles)

# Show grid
self.plot_graph.showGrid(x=True, y=True)

# Initialize graph line
self.line = self.plot_graph.plot([], [])

# Set axes range
self.plot_graph.setYRange(0, 10)

```

```

self.plot_graph.setXRange(-100, 100) # Set based on your force diff data
self.add_regional_labels()

self.sample_rate = 24
# Add a timer to update the plot
self.timer = QtCore.QTimer()
self.timer.setInterval(1000/self.sample_rate)
self.timer.timeout.connect(self.update_plot)
self.timer.start()

self.index = 0
self.display_window = 10 # Duration in seconds to show on the graph
self.start_monitoring()

def add_regional_labels(self):
    self.right_label = pg.TextItem(text="Right", angle=0, anchor=(0.5, 0.5), )
    self.right_label.setColor('#FFA500')
    self.left_label = pg.TextItem(text="Left", angle=0, anchor=(0.5, 0.5))
    self.left_label.setColor('b')

def autodetect_com_port(self):
    """Auto-detect the Raspberry Pi Pico's COM port."""
    ports = serial.tools.list_ports.comports()
    for port in ports:
        if "USB" in port.description:
            return port.device
    return None

def start_monitoring(self):
    com_port = self.autodetect_com_port()
    if not com_port:
        # QMessageBox.critical(self, "Error", "No Raspberry Pi Pico detected!")
        return
    print("port found")

    baudrate = 115200

    # Start the serial reader thread
    self.serial_thread = SerialReaderThread(com_port, baudrate)
    self.serial_thread.data_received.connect(self.update_channel_value)
    self.serial_thread.start()

def update_channel_value(self, channel, value):

```

```

    """Update the value for a specific channel and compute sums for each group with
    calibration."""
    if channel in self.calibration_factors:
        calibrated_value = value * self.calibration_factors[channel]
        self.channel_values[channel] = calibrated_value
        self.compute_sums()

    def compute_sums(self):
        global left_tare
        global right_tare
        """Compute the sum of values for each group of channels and maintain only the last 10
        seconds of data."""
        group1_sum = round(sum(value for channel, value in self.channel_values.items() if
        channel[0] == '1') - right_tare, 4)
        group2_sum = round(sum(value for channel, value in self.channel_values.items() if
        channel[0] == '2') - left_tare, 4)

        # Append new data with timestamps
        current_time = time.time()
        self.force_right.append((current_time, float(group1_sum)))
        self.force_left.append((current_time, float(group2_sum)))

        # Keep only the last 10 seconds of data
        self.force_right = [(t, val) for t, val in self.force_right if current_time - t <=
        self.display_window]
        self.force_left = [(t, val) for t, val in self.force_left if current_time - t <= self.display_window]

    def update_plot(self):
        cm = pg.ColorMap([0.0, 1.0], ['b', 'orange'])
        pen = cm.getPen(span=(-0.05, 0.05), orientation='horizontal', width=4)

        if self.force_left and self.force_right:
            # Extract timestamps and values for plotting
            times_right = [t for t, _ in self.force_right]
            values_right = [val for _, val in self.force_right]
            times_left = [t for t, _ in self.force_left]
            values_left = [val for _, val in self.force_left]

            # Ensure times are sorted
            times_right, values_right = zip(*sorted(zip(times_right, values_right)))
            times_left, values_left = zip(*sorted(zip(times_left, values_left)))

            self.asym = [r - l for r,l in zip(values_right,values_left)]

```

```

# Set y-axis range to the last 'display_window' seconds
start_time = max(times_right[0], times_left[0])
end_time = times_right[-1] if times_right[-1] > times_left[-1] else times_left[-1]
self.plot_graph.setYRange(start_time, end_time)

self.line.setData(self.asym, times_right)
self.line.setPen(pen)
self.plot_graph.getAxis('left').setTicks([])

# self.right_label.setPos(75, start_time + 1)
# self.plot_graph.addItem(self.right_label)
# self.left_label.setPos(-75, start_time + 1)
# self.plot_graph.addItem(self.left_label)
if (max(values_right[self.index]), values_left[self.index]) != 0:
    percent_asym = abs(self.asym[self.index]) / (max(values_right[self.index],
values_left[self.index])
    if (values_right[self.index] > 30) & (values_left[self.index] > 30) & (percent_asym >
0.1):
        self.plot_graph.setBackground('r')
    else:
        self.plot_graph.setBackground('w')
    else:
        self.plot_graph.setBackground('w')

def closeEvent(self, event):
    """Handle application close."""
    self.closed_signal.emit()
    if self.serial_thread and self.serial_thread.isRunning():
        self.serial_thread.stop()
        self.serial_thread.wait()
    event.accept()

# ## DIFF BAR

# In[7]:

class diffbar_MainWindow(QMainWindow):
    closed_signal = Signal()

```

```

def __init__(self):
    super().__init__()

    # Create lists to store the last 10 seconds of data along with timestamps
    self.force_left = None
    self.force_right = None

    global left_tare
    global right_tare

    # Calibration factors for each channel (adjust as needed)
    self.calibration_factors = {
        '1A': 0.05128183369,
        '1B': 0.05140563617,
        '1C': 0.05226255289,
        '1D': 0.05096242992,
        '2A': 0.05121516838,
        '2B': 0.05140285445,
        '2C': 0.05292458815,
        '2D': 0.05250629783
    }

    self.serial_thread = None
    self.channel_values = {}

    # set up central widget
    self.central_widget = QWidget()
    self.setCentralWidget(self.central_widget)
    self.layout = QVBoxLayout(self.central_widget)

    # create plot widget
    self.plot_widget = pg.PlotWidget()
    self.layout.addWidget(self.plot_widget)
    self.plot_widget.setBackground("w")
    self.plot_widget.setTitle("<span style='color: black; font-size: 20pt; font-weight: bold;'>Force
Difference</span>")
    timestyles = {"color": "black", "font-size": "20px", "font-weight": "bold"}
    forcestyles = {"color": "black", "font-size": "20px", "font-weight": "bold"}
    self.plot_widget.setLabel("bottom", "Force (lbs)", **forcestyles)

    # Hide the y-axis scale and labels
    self.plot_widget.getAxis('left').setTicks([]) # Remove tick marks
    self.plot_widget.getAxis('left').setStyle(showValues=False) # Hide values

```

```

# Add a timer to simulate new temperature measurements
self.timer = QtCore.QTimer()
self.sample_rate = 24
self.timer.setInterval(1000/self.sample_rate)
self.timer.timeout.connect(self.update_plot)
self.timer.start()

self.index = 0
self.display_window = 10 # Duration in seconds to show on the graph
self.start_monitoring()

def autodetect_com_port(self):
    """Auto-detect the Raspberry Pi Pico's COM port."""
    ports = serial.tools.list_ports.comports()
    for port in ports:
        if "USB" in port.description:
            return port.device
    return None

def start_monitoring(self):
    com_port = self.autodetect_com_port()
    if not com_port:
        # QMessageBox.critical(self, "Error", "No Raspberry Pi Pico detected!")
        return
    print("port found")

    baudrate = 115200

    # Start the serial reader thread
    self.serial_thread = SerialReaderThread(com_port, baudrate)
    self.serial_thread.data_received.connect(self.update_channel_value)
    self.serial_thread.start()

def update_channel_value(self, channel, value):
    """Update the value for a specific channel and compute sums for each group with
    calibration."""
    if channel in self.calibration_factors:
        calibrated_value = value * self.calibration_factors[channel]
        self.channel_values[channel] = calibrated_value
        self.compute_sums()

def compute_sums(self):

```

```

global left_tare
global right_tare
"""Compute the sum of values for each group of channels and maintain only the last 10
seconds of data."""
group1_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '1') - right_tare, 4)
group2_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '2') - left_tare, 4)

# Append new data with timestamps
self.force_right = float(group1_sum)
self.force_left = float(group2_sum)

def add_regional_labels(self):
    right_label = pg.TextItem(text="Right", angle=0, anchor=(0.5, 0.5))
    right_label.setPos(75, 1.25) # Position at y=0.5 (adjust as necessary)
    right_label.setColor('#FFA500')
    self.plot_widget.addItem(right_label)
    left_label = pg.TextItem(text="Left", angle=0, anchor=(0.5, 0.5))
    left_label.setPos(-75, 1.25) # Position at y=0.5 (adjust as necessary)
    left_label.setColor('b')
    self.plot_widget.addItem(left_label)

def update_plot(self):
    # Simulate streaming data by updating the index
    if self.force_left and self.force_right:
        self.asym = self.force_right - self.force_left
        self.plot_widget.clear()
        self.plot_widget.setRange(yRange=[-0.25, 1.5], xRange=[-100, 100])
        if max(self.force_left, self.force_right) != 0:
            percent_asym = self.asym / max(self.force_left, self.force_right)

            if (self.force_left > 30) & (self.force_right > 30) & (percent_asym > 0.1):
                self.plot_widget.setBackground('r')
            else:
                self.plot_widget.setBackground('w')

            if self.asym > 0:
                brush = ['#FFA500']
            else:
                brush = ['b']
        else:
            self.plot_widget.setBackground('w')

```

```

self.bar_graph = pg.BarGraphItem(x0=0, width=self.asym, height = 1, brushes=brush)
self.plot_widget.addItem(self.bar_graph)
self.add_regional_labels()

def closeEvent(self, event):
    """Handle application close."""
    self.closed_signal.emit()
    if self.serial_thread and self.serial_thread.isRunning():
        self.serial_thread.stop()
        self.serial_thread.wait()
    event.accept()

### MAIN APP

# In[8]:

class MainWindow(QMainWindow):
    def __init__(self, parent=None):
        super().__init__()
        # print("Main window initialized")
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        self.ui.countdownLabel.hide()

        self.ui.absbar_button.clicked.connect(self.set_last_graph_clicked)
        self.ui.absline_button.clicked.connect(self.set_last_graph_clicked)
        self.ui.diffbar_button.clicked.connect(self.set_last_graph_clicked)
        self.ui.diffline_button.clicked.connect(self.set_last_graph_clicked)
        self.last_clicked_graph_button = None

        self.ui.tare_button.clicked.connect(self.tare)
        self.ui.start_button.clicked.connect(self.start)

        self.countdown_timer = None # Initialize the countdown timer

# Create lists to store the last 10 seconds of data along with timestamps
self.force_left = 0
self.force_right = 0

```



```

global left_tare
global right_tare

left_tare = 0
right_tare = 0

self.tare_offset = {'1': 0, '2': 0} # Offset to subtract for tare

# Calibration factors for each channel (adjust as needed)
self.calibration_factors = {
    '1A': 0.05128183369,
    '1B': 0.05140563617,
    '1C': 0.05226255289,
    '1D': 0.05096242992,
    '2A': 0.05121516838,
    '2B': 0.05140285445,
    '2C': 0.05292458815,
    '2D': 0.05250629783
}

self.serial_thread = None
self.channel_values = {}

self.start_monitoring()
self.count = 0

def autodetect_com_port(self):
    """Auto-detect the Raspberry Pi Pico's COM port."""
    ports = serial.tools.list_ports.comports()
    for port in ports:
        if "USB" in port.description:
            return port.device
    return None

def start_monitoring(self):
    com_port = self.autodetect_com_port()
    if not com_port:
        QMessageBox.critical(self, "Error", "No Raspberry Pi Pico detected!")
        return
    print("port found")

    baudrate = 115200

```

```

# Start the serial reader thread
self.serial_thread = SerialReaderThread(com_port, baudrate)
self.serial_thread.data_received.connect(self.update_channel_value)
self.serial_thread.start()

def update_channel_value(self, channel, value):
    """Update the value for a specific channel and compute sums for each group with
    calibration."""
    if channel in self.calibration_factors:
        calibrated_value = value * self.calibration_factors[channel]
        self.channel_values[channel] = calibrated_value
        self.compute_sums()

def compute_sums(self):
    group1_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '1'), 4)
    group2_sum = round(sum(value for channel, value in self.channel_values.items() if
channel[0] == '2'), 4)

    # Append new data with timestamps
    self.raw_force_right = float(group1_sum)
    self.raw_force_left = float(group2_sum)

    self.force_right = self.raw_force_right - right_tare
    self.force_left = self.raw_force_left - left_tare

def closeEvent(self, event):
    """Handle application close."""
    if self.serial_thread and self.serial_thread.isRunning():
        self.serial_thread.stop()
        self.serial_thread.wait()
    event.accept()

def stop_monitoring(self):
    if self.serial_thread and self.serial_thread.isRunning():
        self.serial_thread.stop()
        self.serial_thread.wait()
        self.serial_thread = None

def tare(self):
    global left_tare
    global right_tare
    left_tare = self.raw_force_left

```

```

right_tare = self.raw_force_right
print("right tare", right_tare)
self.change_button_color()

def set_last_graph_clicked(self):
    button = self.sender()
    if button:
        self.reset_graph_buttons()
        button.setStyleSheet("background-color: green; color: white;")
        self.last_clicked_graph_button = button
        print(self.last_clicked_graph_button.text())

def start(self):
    self.change_button_color()
    self.start_countdown(5)

def restart_monitoring(self):
    self.start_monitoring()

def start_countdown(self, seconds):
    self.countdown_time = seconds
    self.ui.countdownLabel.show()
    self.ui.countdownLabel.setText(f"Starting in: {self.countdown_time}")
    self.countdown_timer = QTimer(self)
    self.countdown_timer.timeout.connect(self.update_countdown)
    self.countdown_timer.start(1000) # Update every second

def update_countdown(self):
    self.countdown_time -= 1
    if self.countdown_time > 0:
        self.ui.countdownLabel.setText(f"Starting in: {self.countdown_time}")
    else:
        self.ui.countdownLabel.setText("Starting now!")
        self.countdown_timer.stop() # Stop the countdown
        self.begin_graph() # Start the graph after countdown ends

def reset_graph_buttons(self):
    for button in [self.ui.absbar_button, self.ui.absline_button, self.ui.diffbar_button,
self.ui.diffline_button]:
        button.setStyleSheet("background-color: white; color: black;")

```

```

def set_unclicked(self):
    for button in [self.ui.absbar_button, self.ui.absline_button, self.ui.diffbar_button,
                  self.ui.diffline_button, self.ui.tare_button, self.ui.start_button]:
        button.setStyleSheet("background-color: white; color: black;") # Reset the style
        button.setChecked(False) # This line is more relevant for toggle buttons

def change_button_color(self):
    button = self.sender()
    if button:
        button.setStyleSheet("background-color: green; color: white;")

def begin_graph(self):
    QTimer.singleShot(5000, self.begin_graph)

def begin_graph(self):
    if self.last_clicked_graph_button.text() == 'Force Difference Bar Graph':
        self.stop_monitoring()
        self.diffbar_window = diffbar_MainWindow()
        self.diffbar_window.closed_signal.connect(self.restart_monitoring)
        self.diffbar_window.show()
    elif self.last_clicked_graph_button.text() == 'Force Difference Line Graph':
        self.stop_monitoring()
        self.diffline_window = diffline_MainWindow()
        self.diffline_window.closed_signal.connect(self.restart_monitoring)
        self.diffline_window.show()
    elif self.last_clicked_graph_button.text() == 'Absolute Force Bar Graph':
        self.stop_monitoring()
        self.absbar_window = absbar_MainWindow()
        self.absbar_window.closed_signal.connect(self.restart_monitoring)
        self.absbar_window.show()
    elif self.last_clicked_graph_button.text() == 'Absolute Force Line Graph':
        self.stop_monitoring()
        self.absline_window = AbsLineMainWindow()
        self.absline_window.closed_signal.connect(self.restart_monitoring)
        self.absline_window.show()

    self.ui.countdownLabel.hide()
    self.set_unclicked()

# In[9]:

if __name__ == "__main__":

```

```
app = QApplication(sys.argv)
widget = MainWindow()
widget.show()
sys.exit(app.exec())
```