BaselineTech

A Mixed Reality Snowboarding Simulator

Product Development Document

By

Tejaswini Kambaiahgari - Product Owner

January 6, 2025

Project Team

Katrina Le - Project Lead

Cat Menten - Electrical Technical Lead

Bryan Pi s - Electrical Engineer

Jani Passas - Electrical Engineer Ava

Gami - Electrical Engineer

Jiamei Goodwin - Mechanical Technical Lead

Sachin Patel - Mechanical Engineer

Eli Zislis - Mechanical Engineer

Brendan Coggins - Mechanical Engineer

Alisa Nance - Mechanical Engineer

Will Rosen - Mechanical Engineer

Austin Fasnacht - Mechanical Engineer

TABLE OF CONTENTS

Executive Summary

Purpose

This project aims to build a machine that accurately simulates snowboarding and provides haptic feedback to users. Snowboarding is a seasonal based activity with limited availability in urban areas, snow sport enthusiasts may struggle to regularly engage in this sport. Access to snow mountains and resorts is restricted and requires time-consuming travel, making frequent practice less likely. This limited access can impede skill development and the entertainment of the sport. In addition to the hindrance of skill maintenance for experienced users, beginners can often feel intimidated or hesitant from starting due to it having a steep learning curve and being prone to injuries. BaselineTech proposes a snow sport simulator that provides an immersive experience for training and entertainment all year round. This also provides a safer opportunity for beginners to get more familiar with navigating the snow-sport terrain. This product will require easy installation and will allow the users to set a constant incline value and control their own momentum while they are on the board. Position data will be recorded and will be displayed to the user from an interface. Haptic feedback is provided in the form of a vibrating motor to simulate the feeling of being on an actual mountain.

Background Research

There are current alternatives on the market, but they do not offer all the ease of installation and haptic feedback that BaselineTech offers. Products such as Proleski mimic ski and snowboard conditions, and can be seen in some entertainment complexes. However, these products present significant limitations when it comes to installation and user accessibility. Additionally, their lack of haptic feedback makes them less appealing to the everyday consumer seeking a seamless and personalized experience. Therefore, there are no snowboard simulators on the market that offer user-friendly design and feedback data that BaselineTech proposes.

Mechanical Design

Full Assembly Overview

The mechanical system of this project was designed to allow the user to simulate the movements of snowboarding without the use of electronics like motors to assist with the movement. This was done using 6 main subsystems. There are three rotational movements, including roll and yaw, which are controlled by the user, and pitch, which is adjusted by the user prior to using the machine. There is also a linear movement subsystem that allows the side to side movement of the snowboarder when turning. The direction of these movements can be seen in the diagram below.

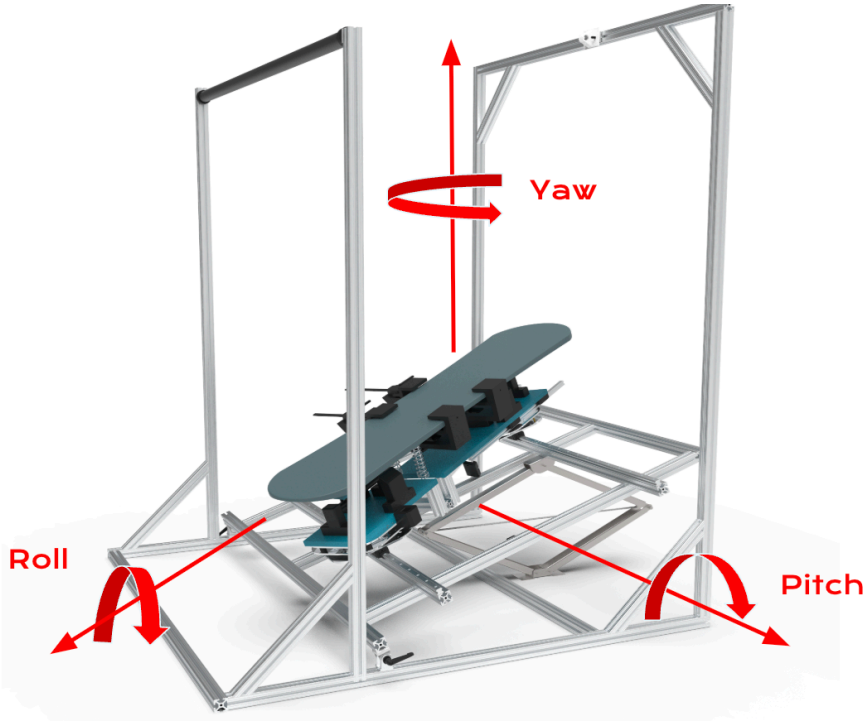


Figure 1: Full Assembly

Additionally, there is a board mounting subsystem that allows varying sizes of boards to be mounted while also accommodating the integrated load cell sensors. Lastly, there is a safety subsystem which includes a harness at the back, and a handle bar at the front.

Roll

- The roll subsystem simulates the rotation that occurs when a snowboarder shifts their weight to the back of the heels or the front of their toes in order to catch an edge.

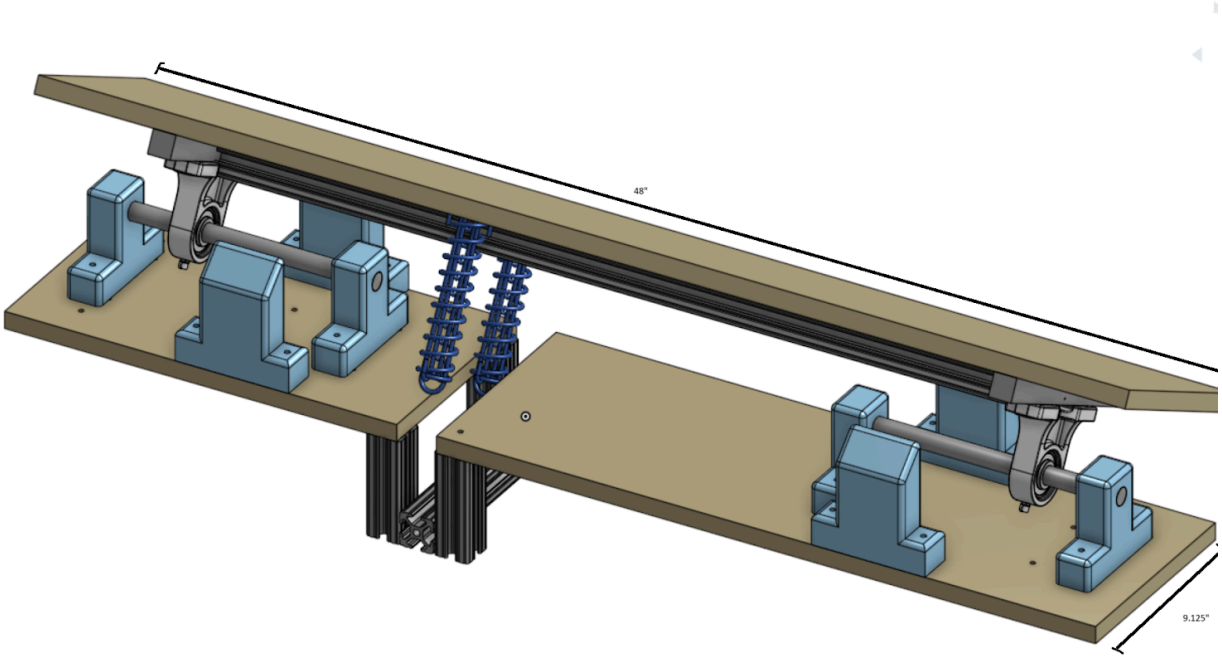


Figure 2: Roll Subassembly

- The goal for this subsystem was to mechanically emulate the “roll” rotation that occurs when properly snowboarding. To achieve this goal, the subsystem needed to allow for uniform stable rotation along the length of the board, with some external force to simulate how it feels to turn in actual snow.
- The rotation is accomplished by attaching mounted ball bearings to the top wood board. These bearings rotate around two shafts that are held in place at either end of the bottom wood board. The springs are used to both provide stability when the top board is flat, and to help the top board return to its original, flat position when it is rotated. It is hard to show in the CAD, but the springs are attached to the top board, and also to the 8020 piece that is underneath the slot in the bottom board. By attaching the springs to 8020 below the board, the entire subsystem can be significantly shorter, which saves materials, decreases moment arms, and overall makes the product less intimidating to get on. There are also four hardstops that are designed to stop the top board from going past 43.5

degrees. This angle was determined to be the maximum angle that you can carve at on a snowboard before you lose your edge and fall.

- The design ended up working well, but there were some areas for improvement. We were forced to add extra support between the two bottom boards as they were bending upwards with the springs as you rotated. After adding the supports, it was fine but these extra supports should be added to the design. It would be interesting to play around with different amounts of springs and see if there is any way to make the springs pull harder the further you rotate without being too hard to turn at the start of the rotation. The springs provide good resistance as you turn but it does not noticeably get harder the further you go.

Springs

- Porch swing springs were used to help stabilize the top board rotation and simulate the effects of gravity as you turn.

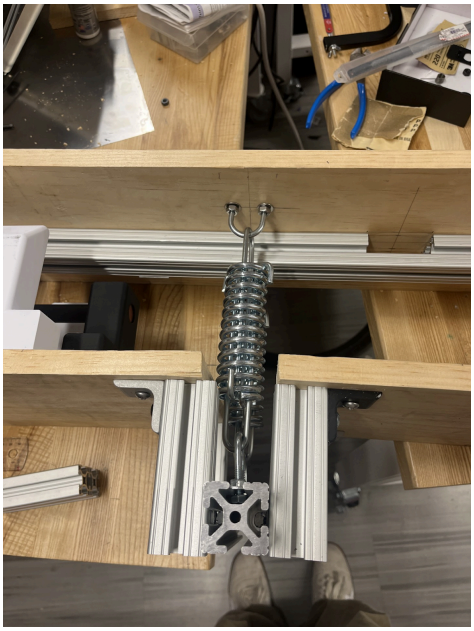


Figure 3: Spring Setup

- When a user rotates in one direction, the spring on the opposite side will pull in the opposite direction, applying more force the further that the user rotates. It also acts as a stabilizer, allowing the user to get on the board easily without it immediately rotating to one side.
- We always knew that we would need something to help the board return to its original flat position and offer some kind of resistance as you turn. At first, we were leaning towards torsional springs that could run along the shaft, but we

could not find any that were strong enough and also in our budget. Originally we were concerned about extension springs deforming if they have to follow a circular rotation path as linear extension springs. We were able to bypass this by securing the springs to U-bolts which allow the springs to move/rotate to follow the board's rotation.

- Given that the max weight was 150 pounds, we set up a problem where all the weight of the user is at the edge of the board, and the force of the spring on the other side has to counteract the torque from the weight of the user. We were able to find a set of equations that related the spring constant, the max stretch of the springs and the distance that the spring is from the center. Using these equations, we searched online but found it difficult to find many springs that were strong enough on sites like McMaster. We found more success by thinking of other products that use heavy duty springs, like punching bags and porch swings, and we were able to find these springs that fit into the range of solutions that we had found. After getting the springs we made a test setup to find the correct distance away from the center, and used the setup to find out what distance feels best to the snowboarders on our team.
    - The springs are 7.75 inches long, and have an outer diameter of 1.5625 inches. They have a spring constant of 150 lbs/in, with a max weight of 350 pounds. Two springs cost around $30, although if bought in bulk they would most likely cost much less.

Hard Stops

- The hardstops are 3D-printed and are used to prevent the board from over-rotating.

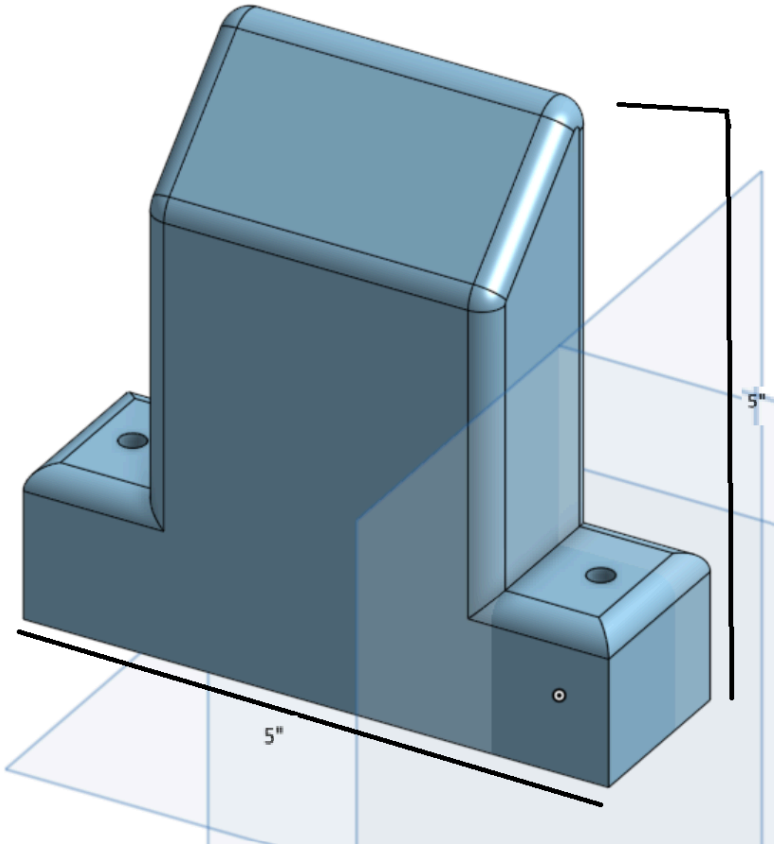


Figure 4: Hardstop CAD

- These hardstops are attached to the bottom wood board. There are four of them in total, one in each quadrant of the subsystem. As the board rotates, it will line up perfectly with the angled hard stop surface at 43.5 degrees to the ground.
- We did some research, mainly reading this explanation: http://alpinesnowboarder.com/wp-content/uploads/2018/03/Physics-of-a-Snowboard-Carved-Turn.pdf,
  which helped determine a good angle to stop the rotation at. It was found that for an average snowboarder, if their board went past 43.5 degrees in comparison to the ground, they would lose their edge and the snowboard would slip out from beneath them and they would fall. That is why we decided to make the hardstops at that angle, so that they could stop the user from rotating past a point that they usually could not go past in snow.
- The springs apply a lot of force on the board when it is fully rotated which prevents the hardstops from actually feeling much stress when they collide. This means that the material that the hardstops are made from is not super important, as long as it is fairly strong. The hardstops we used were FDM printed with 30% infill and they performed very well. They definitely could have less infill

percentage, or be made out of wood or sheet metal if that would be cheaper for the manufacturer. In order for the wood board to hit the hardstops at the exact 43.5 degree angle, the tolerances have to be pretty tight, but it is not a big deal if it does not hit it perfectly as that chosen angle is an average and it is okay if it is slightly off.

Linear Movement

- The linear movement subsystem incorporates a highly dynamic side-to-side motion to mimic the carving experience on a snowboard. This is accomplished using lubricated stainless steel linear rails and linear ball bearings.

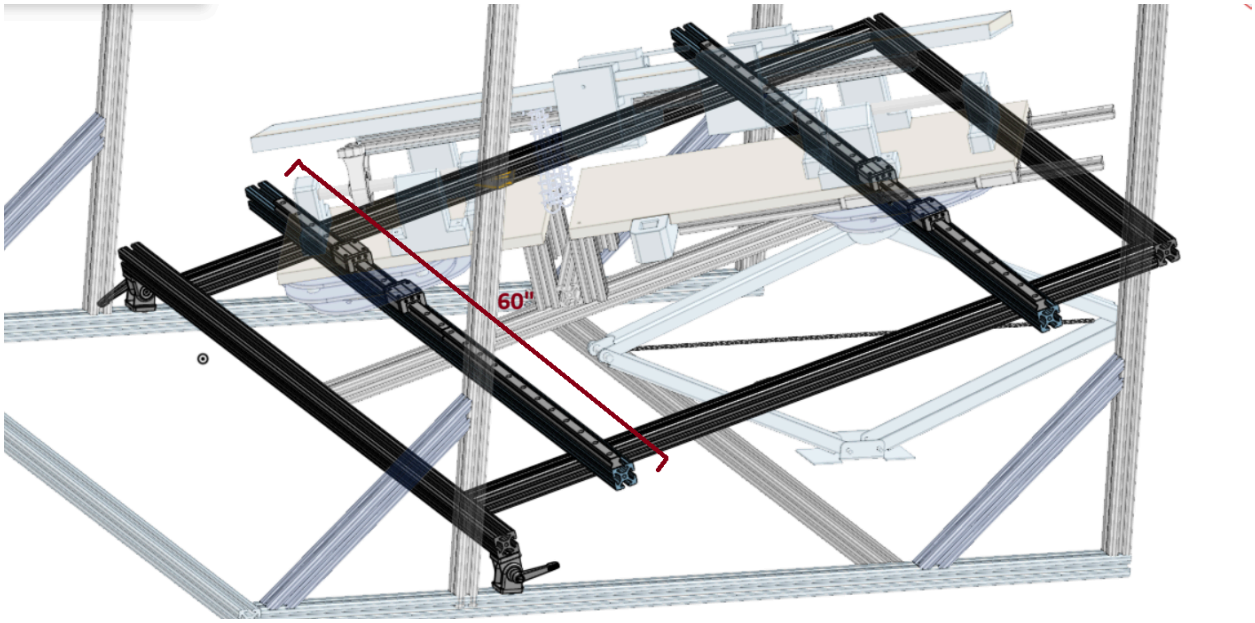


Figure 5: Linear Movement Subassembly

- This design was chosen to closely replicate the natural carving motion of snowboarding. The system enables smooth, low-friction motion thanks to its linear rails and bearings. It also uses bungee cords to self-center.
- The selected linear ball bearings and rails can support a dynamic load of approximately 300 lbs, exceeding the weight limit specified by the client. The rails are mounted on an 8020 frame, which is attached to two 8020 locking pivot joints to enable pitch adjustment. Each rail is equipped with two linear bearings, positioned together to align directly under the user's feet.

- The system performed as intended, but potential issues could arise over time. If used long-term, the rails would require periodic relubrication along with regular maintenance to maintain optimal performance and ensure longevity.

Linear Rails

- The linear rails enable smooth, guided motion of the snowboard, ensuring precision and stability. The rails are made from hardened steel for durability and low friction, they measure 59” in length and 1.5” in width. The linear rails provide the structural backbone for lateral motion within the overall assembly.

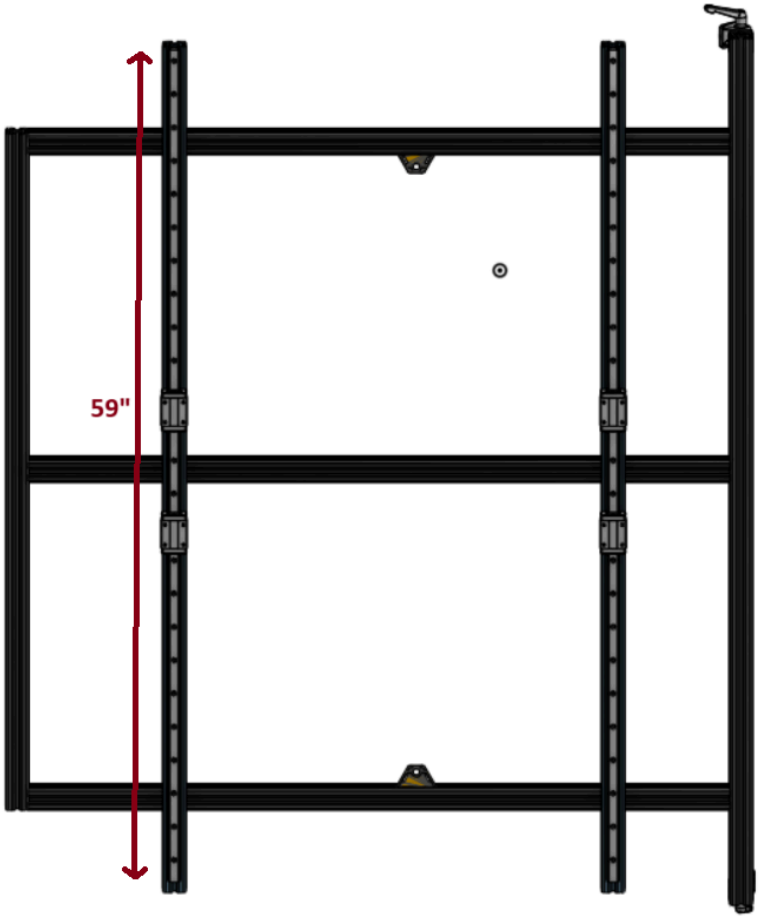


Figure 6: Linear Rails and Bearings CAD

- During the design process, we initially researched various linear motion mechanisms such as wheels, bearings, and rails. Through further research, linear rails were chosen for their superior precision and load-bearing capacity. The key features of these rails include preloaded ball-bearing carriages to minimize play and an anti-corrosion treatment for extended life. The rails are made of hardened steel which is strong, wear resistant, and

cost-effective, with each the rails costing approximately $135 as sourced from VEVOR. The rails are manufactured to within a tolerance of ±0.1 mm to ensure compatibility with carriages. For manufacturing recommendations, we advise sourcing rails from reputable suppliers with precision grinding capabilities and conducting regular lubrication to maintain optimal performance.

Bungee Cords

- The bungee cords play a vital role in providing dynamic tension to return the snowboard to its neutral position, enhancing the simulation by adding realistic resistance and feedback. The bungee cords chosen are length adjustable, and the length we chose was about 12 “. Each cord, made from rubber with a braided polyester sheath, measured 12” in length (unstretched) and can stretch up to 30”. These cords contribute significantly to the overall functionality of the simulator, ensuring a responsive and realistic experience for users.
- They are tied to U-bolts connected to the roll subsystem and hooked onto 3D printed hooks that slide into the 8020 frame.

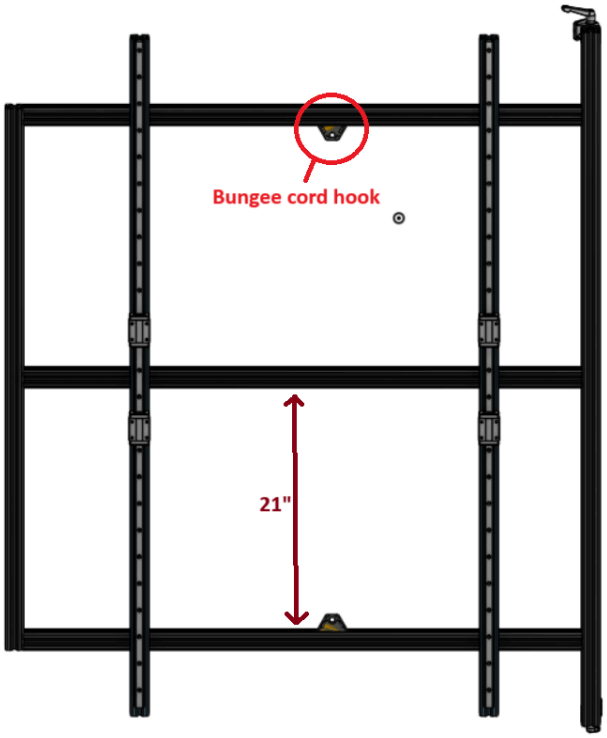


Figure 7: Bungee Cord Hooks CAD

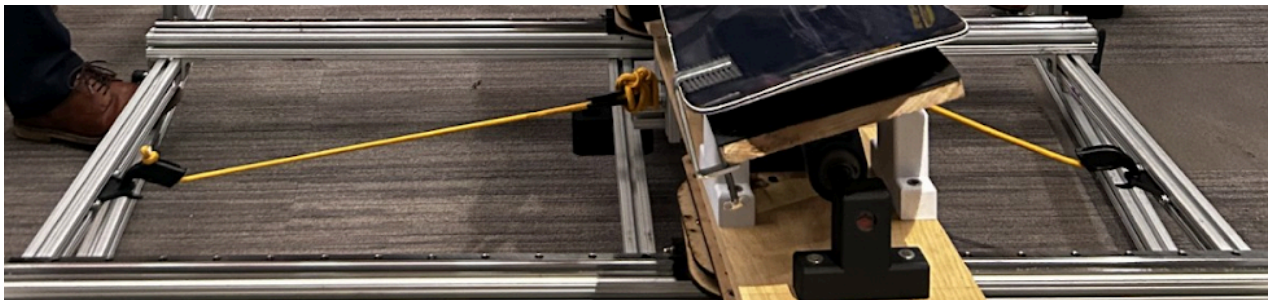


Figure 8: Bungee Cord Setup

- The design process for the bungee cords began with the research of various methods of tensioning and centering, including springs and elastic bands. After evaluating these options, bungee cords were chosen for their high elasticity and durability. Key features of the bungee cords include adjustable tension and length for customization and a durable sheath to resist wear and tear. Each cord cost approximately $7, sourced from Home Depot.

Yaw

- The yaw subsystem controls the rotational movement of the snowboard while it is in movement. The front consisted of just a Lazy Susan allowing for rotation. The back consisted of a lazy susan and linear rails on top allowing for an extra degree of freedom in the back. Below are pictures of the entire subsystem beneath the roll subsystem and each section seperately.

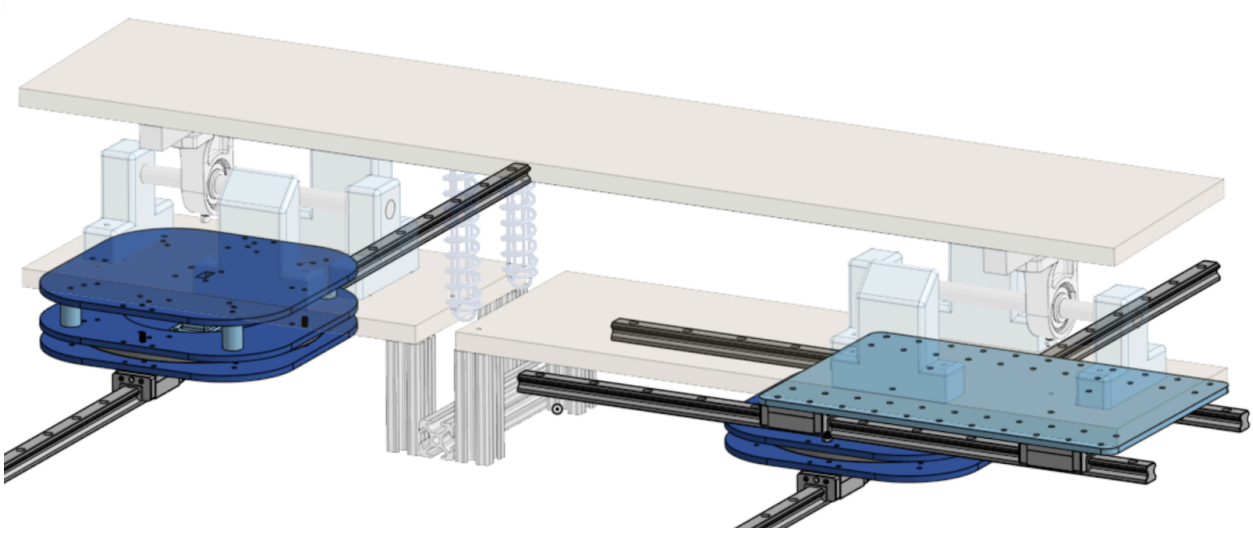


Figure 9: Entire Yaw Assembly

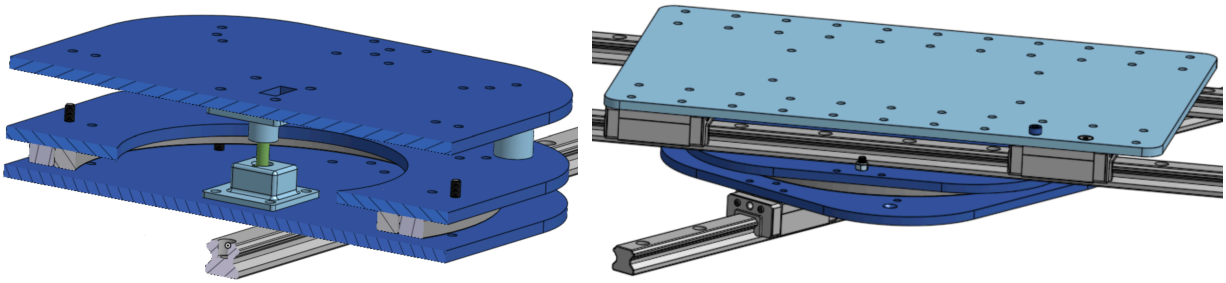


Figure 10: Front Assembly (left) and Back Assembly (right)

- The choice of the lazy susan's (7.8 in ID, 9.8 in OD):
  - I chose to use a lazy susan as my device to give the snowboard its rotational ability. A lazy susan is a large rotational bearing coming in many different sizes. Through searching, I was able to find one that was 10 inches in diameter and could withstand 150 pounds. With the use of two of these and forces being distributed evenly, the motion was smooth and could withstand the weight of the subsystems above and a person.
- The linear rails of the back assembly:
  - To allow for the snowboard to even be able to rotate, the mounts of the board need to be able to extend and contract to different lengths. Since the two lazy susan's used are on fixed tracks, two linear rails were used on the top of the back assemblies lazy susan to allow for that additional degree of freedom.
- Mounting the rotary encoder:
  - To measure the rotation, a rotary encoder was mounted in the center of the front assembly on the bottom plate which did not rotate. To mount it, a 3D printed enclosure was printed that kept it in place. An additional piece was then glued to the rotating part of the enclosure and mounted to the top board (which did rotate). This allowed us to easily implement the rotary encoder and give us precise values.
- Mounting plates used (270mm x 270mm):
  - The plates used throughout this subsystem were 1/8 inch MDF (all colored blue in the figures above). MDF is a very versatile material that allows for easy prototyping and iteration as it could be laser cut within minutes. Additionally, it is able to withstand the forces that would be applied to it by the snowboarder.

Future Improvements:

Some future improvements to this subsystem could be to lessen the weight. Additionally, there could have been a better prepared plan for the mounting of this subsystem to the roll

subsystem mounted above. To be mass distributed, I feel that aluminum would be a better material to use instead of MDF as it is more durable and an industry standard.

Pitch

- The **pitch subsystem** is designed to control the inclination of the snowboarding simulator, utilizing a scissor jack mechanism with a lead screw. This design allows for the smooth adjustment of the simulator's height, increasing the platform's vertical angle when needed. The system consists of two parallel members with interlocking gears at the top and bottom. As the lead screw is tightened, the parallel members move closer together, causing the platform to raise. The entire system is mounted on top of the device using 80/20 aluminum extrusions.

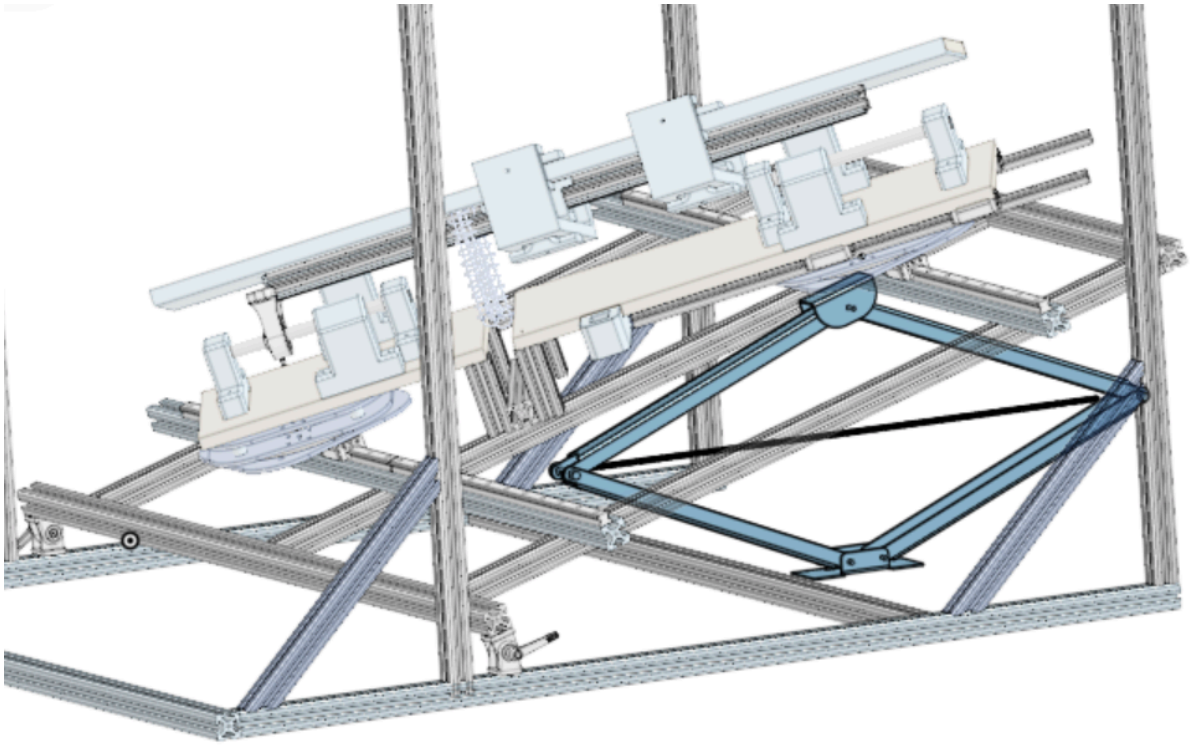
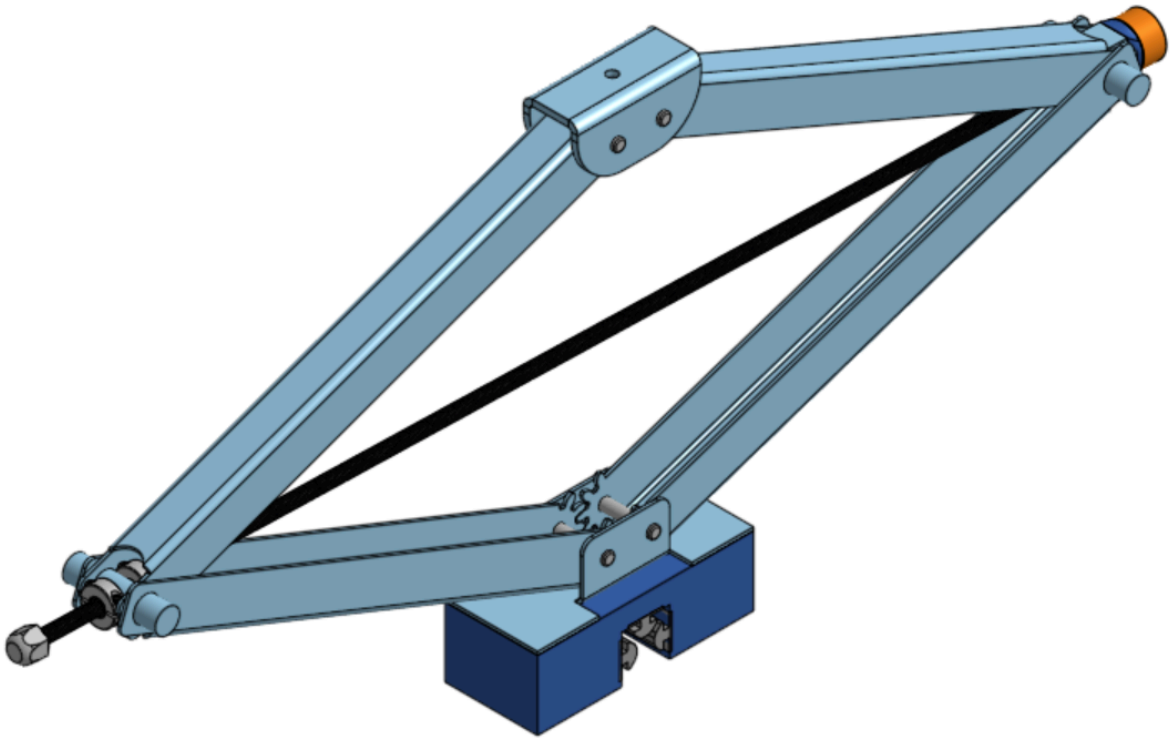


Figure 11: [Pitch in Full Assembly]

- I chose this design because it offers a **simple, reliable**, and **cost-effective** solution for controlling the simulator's pitch. The scissor jack mechanism is widely used for lifting applications, providing the flexibility needed to adjust the platform's angle smoothly. The design also allows for easy maintenance and adjustments using the lead screw and dowel pins.

Scissor Lift

- The scissor jack mechanism comprises two parallel members with interlocking gears driven by a lead screw. The lead screw moves between the members, which are forced to come closer together, ultimately raising the platform. The system is supported by dowel pins to prevent unwanted movement once the device is raised.



- The function of this component is to **raise and lower the platform** of the snowboarding simulator, allowing adjustments to the simulator's pitch angle. When the lead screw is rotated, the interlocking gears force the parallel members to move, causing the platform to become more vertical.
- The design began with a 3D-printed prototype to test the mechanics of the scissor jack. Once the proof of concept was validated, I ordered the parts from **SendCutSend**, a sheet metal fabrication company, which laser cut and bent the parts to the required shape. The remaining components (lead screw, shaft collar, and dowel pins) were sourced from **McMaster-Carr**.
    - **Scissor Jack Design**: Simple but effective lifting mechanism for smooth pitch adjustments.

        - **Lead Screw Mechanism**: Provides precise control over the lifting and lowering process.
        - **Dowel Pins for Stability**: Added to prevent unwanted movement and secure the system once the platform is raised.
    - **Manufacturing Recommendations**:
        - **Laser Cutting & Bending**: Ideal for creating the sheet metal parts, providing high precision.
        - **Lead Screw Assembly**: Requires careful alignment to ensure smooth motion.
        - **Dowel Pins**: Should be reinforced for secure locking of the system.
    - **Cost Estimate**: The total cost for the pitch subsystem was **$335**, which covers the fabrication of the sheet metal parts from **SendCutSend**, along with the components sourced from **McMaster-Carr**.

Board Mounting

- The **board mounting subsystem** is designed to secure the user's board to the top level mounting platform utilizing custom 3D printed assemblies that conform to the width and thickness of the board. This design accounts for constricting all dimensions of the board without applying a compressive load on the board which would thus disrupt the load cells. The system consists of a rubber matted top platform and two compliant clamps, stationary and adjustable, that utilize a detachable dog clamp to constrain the board. One clamp uses a custom slider to glide along the 80/20 support bar under the top layer of the board.

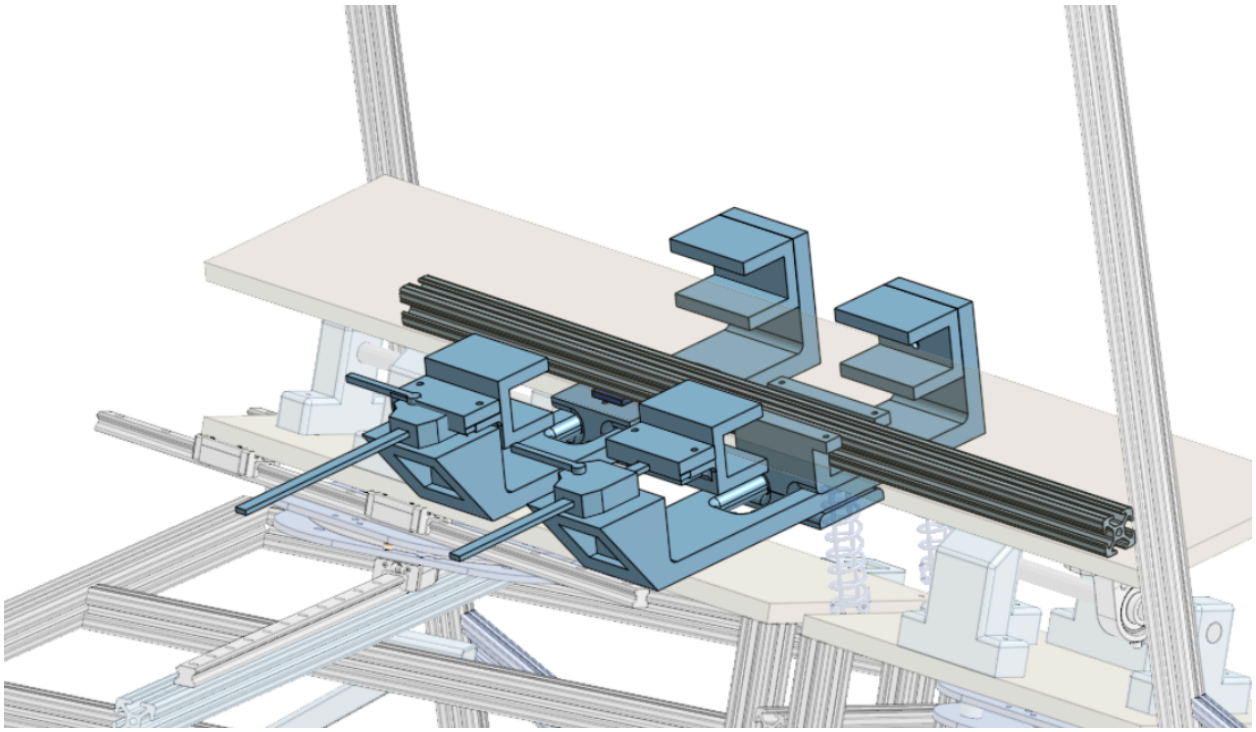


Figure 12: [Board Mounting in Full Assembly]

- I chose this design because it offers the most amount of **adjustability** for the user's board while on the simulator as well as remaining quite **cost-effective**. The main trouble of non-compressive constrainment is the **issue of a robust mechanism** that can support the weight of the user. The use of dog clamps is widely used in word working and applied compressive and stationary forces parallel to the load cells so as to not affect the readings. The strength of the FDM printed components stood the test and remained very cheap to produce. Potential shortcomings of this design stem from its uncertainty of robustness. To fully constrain the users board a **stronger material selection** must be purchased and machined finely as this option was not a possibility with our budget and resources. An alternative solution could be to **use compressive clamps and tare** the readings the load cells receive while being compressed by the clamps.

Non-compressive Clamps

- The clamping mechanism has two versions serving constring on two locations on the user's board. The adjustable clamp, which has a range from the springs to the back side hard stops, comprises seven FDM printed pieces, an off the shelf slider, and an off the shelf dog clamp. The clamp is comprised of 4 components, the "small C" which holds the one end of the platform and board, the "dog clamp assembly" which has 3D printed pieces that attach onto the end of the clamp to form another "small C" holding in the

other side of the mounting platform and board, the “pin” which moves along the 80/20 support, and finally the “large C” facing upwards which houses the other three mechanisms. The stationary clamp is composed of the same components but remains fixed in between the springs and frontside hard stops and has a pin that is screwed into the board.

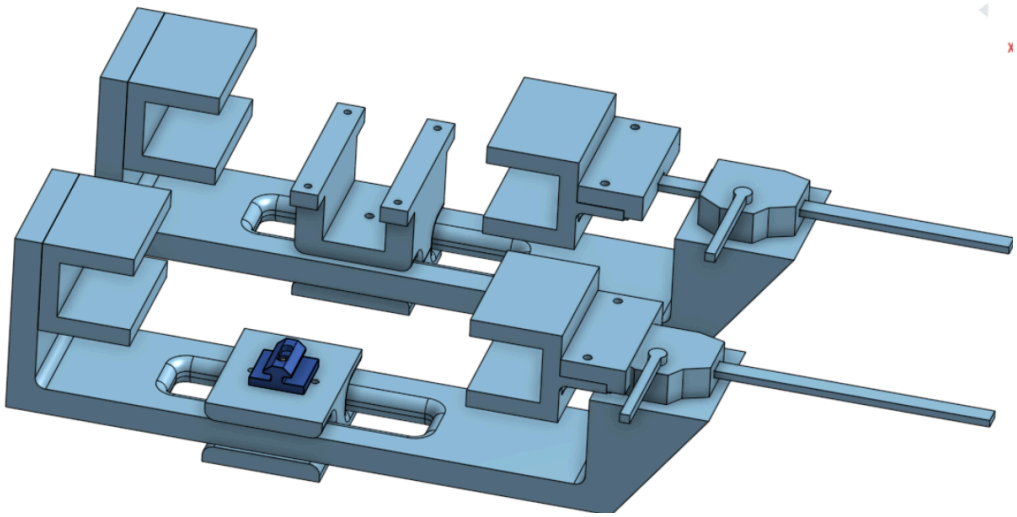


Figure 13: (Back) Stationary and (front) Adjustable Non-compressive Clamps

- The function of this component is to **constrain** the board onto the mounting platform with just the movement of the adjustable clamp to the inner position of the elevated/back binding and to ensure the stationary clamp is contacting the outer portion of the lowered/front binding. With pushing the dog clamp side of the assembly in and pulling the lever the mechanism will snap into place and remain immovable till otherwise acted on.
- The design began with initial sketches of research into parallel constrainment in workshops, wood-working, etc.. A proof of concept was achieved using dog clamps sourced from **Amazon** and a 3D-printed slot and pin mechanism. I then solved the adjustability problem by using an 80/20 support beam with a slider ordered off of McMaster-Carr. Rubber matting was purchased from **Amazon** to increase friction between the board and platform. The remaining components were 3D-printed and sourced from **McMaster-Carr**.
- **Manufacturing Recommendations**:
  - **3D-printing**: This method allows for full control over the adjustability and high level tolerancing though the use of stronger materials would be recommended in the fabrication process to provide an added level of robustness.

    - **Screw Assembly**: Through the use of screws the components were attached and assembled.
- **Cost Estimate**: The total cost for the board mounting subsystem was **$67**, which covers the purchase of the T-slotted glider from **McMaster-Carr** and the EWORK Bench Dog Clamp and Rubber Scraper Mat from **Amazon**, along with the in-house fabrication of the FDM printed components.

Safety

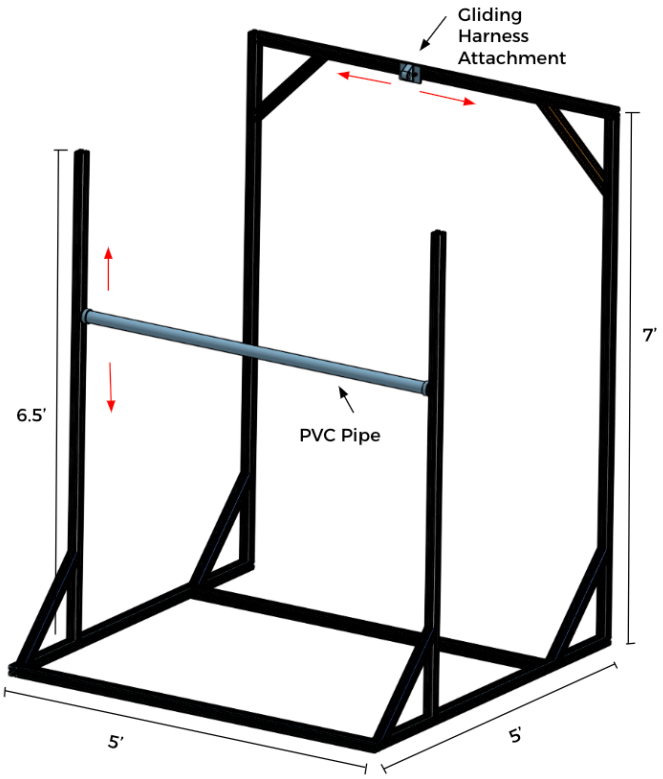


Figure 14: Overall Safety Assembly

Dimensions: W: 5ft x L: 5ft x H:7ft

This design serves two purposes. First, it provides a frame for the entire structure. Made of 80/20, the frame's width and length is kept at 5ft to stay within budget. The second purpose is to ensure safety in case of a fall and provide stabilization. Two components serve this purpose: a 1 1/4-inch PVC pipe along the front of the system, which the user can hold onto for stability, and a sliding rail in the back that attaches to a chest harness worn by the user. Both stabilization methods are essential for ensuring user safety.

The ideation process for this design hinged heavily off of a ski leash, a chest harness worn by kids while they learn how to ski, as well as other skiing and snowboarding prototypes. Therefore the

safety aspect of the design has two main parts. The first is a 6.5ft high adjustable PVC pipe in front of the user to grab onto for stabilization. The second is a 7ft high gliding pipe with a chest harness attachment to it that slides horizontally across the back piece of 8020. The harness provides safety in case of falls and simulates the backwards force faced by wind resistance while snowboarding, allowing for the user to balance better at higher inclines.

Figure 15: Harness Demonstration

Future improvements include adding clamps to the front bar to prevent it from moving unpredictably. Additionally, if the budget would allow, extra aluminum extrusions are highly recommended between the front and back of the subsystem to add more stability to the safety mechanism.

Locking

- The locking mechanism used in the simulator is a simple yet effective design that ensures stability and alignment during use. It consists of two 3D-printed parts both with a square profile that fits precisely into a 1” 8020 aluminum extrusion. One of the 3D-printed parts is mounted onto the roll subsystem, while the other is securely attached to the 8020 frame.

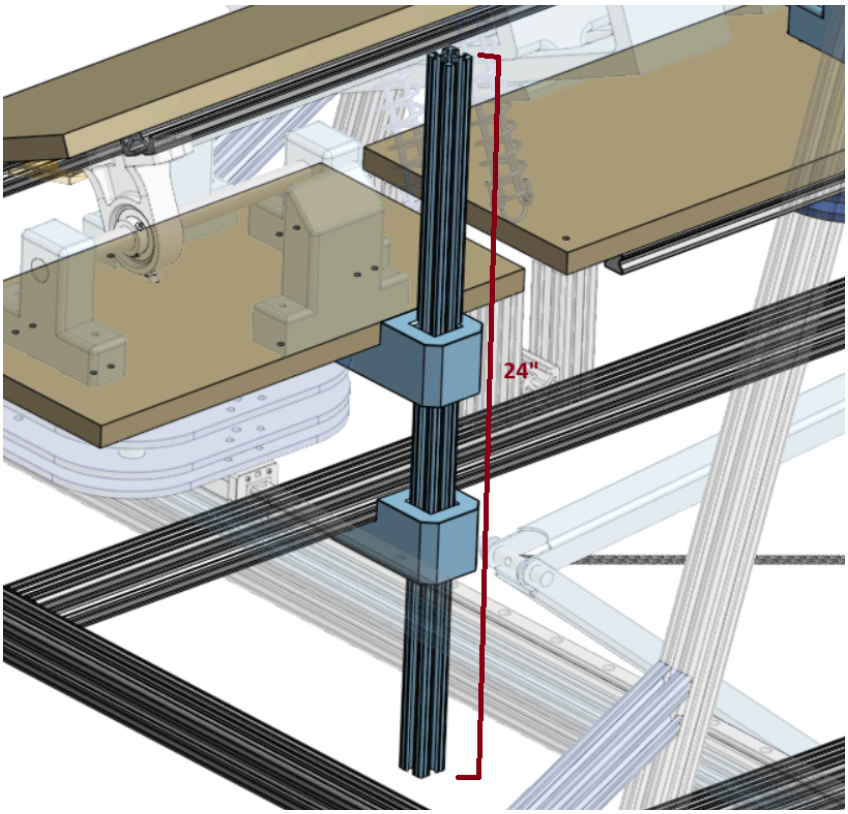
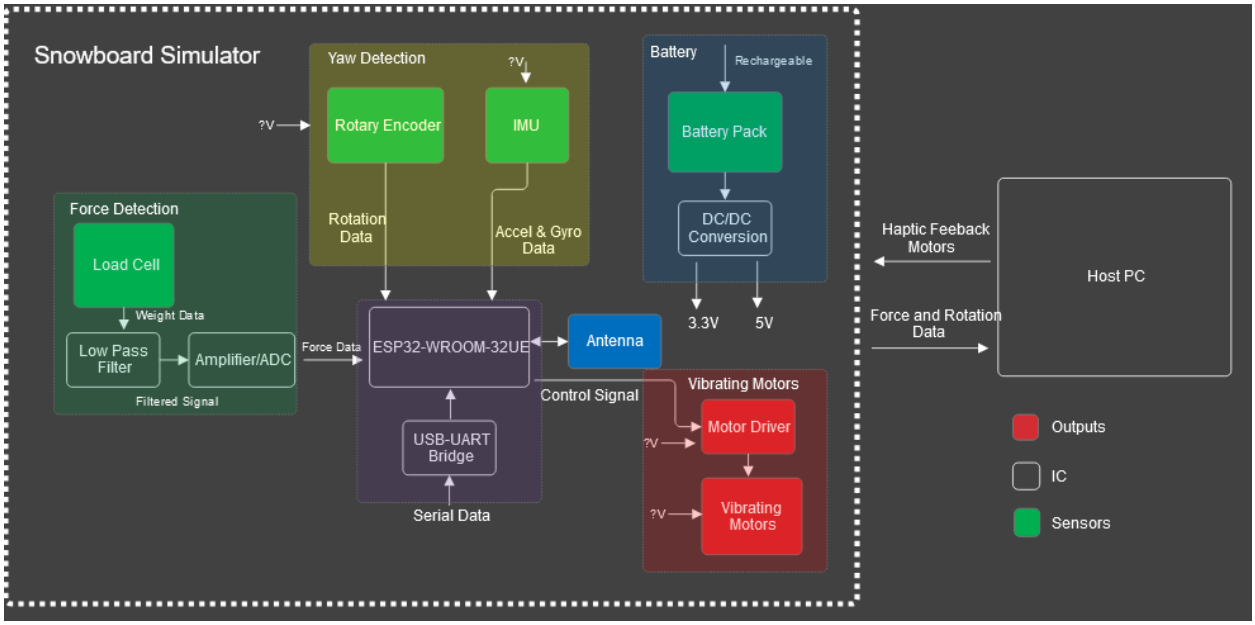


Figure 16: Locking Mechanism Subassembly

- This design was chosen for its simplicity, ease of manufacturing, and cost efficiency. The square profile ensures a snug and reliable fit, minimizing any unwanted movement during operation. Additionally, the use of 3D printing allows for rapid prototyping and customization of the parts to ensure optimal performance. The material for the 3D-printed parts is PLA, selected for its adequate strength and low cost, making it an ideal choice for this application.
- The locking mechanism contributes significantly to the overall assembly by maintaining a firm connection between the snowboard and the frame. During the design process, we evaluated several alternatives, including clamps and adjustable fasteners, but ultimately selected this solution due to its balance of simplicity and effectiveness. The key features of the locking mechanism include its lightweight construction and ease of integration into the existing assembly. For manufacturing recommendations, we suggest using a high-resolution 3D printer to ensure accurate dimensions and tight tolerances. The estimated cost of manufacturing the locking mechanism is approximately $10, including material and labor. We also recommend incorporating two locking points instead of one, as a single locking point may fail under sufficiently high loads.

Electrical Design

Design Overview

- Flowchart of how components and subsystems interact
- 

- Explanation of decisions made in using general technologies and concepts
    - Load cells were chosen to measure the roll and the front and back foot pressure. This is because it was cost-effective and was easy to hide compared to attaching a sensor to the springs. For yaw detection, we mainly used a rotary encoder because it was attached to the actual physical motion of the board, so when the board moved, it moved the encoder. The IMU was added as backup data collection as it can measure many forms of movement, however, it is dependent on the IC internal sensors to keep track of movement which was less reliable.
- I/O of the electrical systems in relation to mechanical, software, etc.

Rotary Encoder

The EMS22A Non-Contacting Absolute Encoder is a rotary encoder used as the primary yaw-sensing system; it measures the horizontal rotation of the mounted snowboard. Data is sent via binary absolute, meaning the position is sent in a series of 10 bits followed by status bits and a parity bit. The rotary encoder uses a magnet to find its position and send it using 1024 pulses per revolution. The sensor can measure up to 0.35° accuracy with ± 1.4° error. The sensor uses 5V, so level shifting is required to gather and send data to and from the sensor.

A rotary encoder was chosen as the primary yaw system because it has a long lifespan and is unlikely to fail if mounted properly--where no vertical force is applied, but near the circular ball bearing so it may rotate freely and collect the snowboard's yaw data. Additionally, it is not a sensor which accumulates error. In testing, the sensor sometimes recorded small jumps in recorded position which may be accounted for by the ± 1.4° error. This may be smoothed and mitigated in code, since it can be assumed the user must turn continuously; the snowboard will not jump from position to position. Finally, the rotary encoder only measures position in one plane and will not be affected by changes in pitch or roll, unlike the IMU.

Rotary encoder code in Python creates a plot meant to emulate the motion of the snowboarder down a mountain, found in Appendix A. This code creates a graph which uses the rotary encoder's position to direct the 'snowboarder' down the mountain, moving with constant speed but variable direction. Speed is adjustable by the adjustable pitch of the simulator.

Rotary encoder code in Arduino collects raw positional data from the rotary encoder and prints that position. Python sometimes has discrepancies which does not collect data smoothly; Arduino does not have this issue.

IMU (Inertial Measurement Unit)

The BMI088 6-Axis IMU is a secondary yaw system for redundancy as well as a general data collection sensor so any programmer may gather position and orientation data from the snowboard apparatus as it is used. It is primarily intended for the possible addition of a virtual environment to the snowboarding experience by programmers outside this project.

The BMI088 IMU's technical specifications are best found on pages 9 and 10 of the datasheet. The gyroscope's zero-rate error is ±1°/s, but this may be mitigated by comparing values from the IMU to those of the rotary encoder and load cells to correct error buildup over time. The IMu can be configured using I2C or SPI, but I2C was chosen for this project for simplicity. It operates at a 3.3V logic level, so no level shifting is necessary.

This IMU was chosen due to the logic level it operates on and team members' past project experience. A 6-axis sensor is required; 3-axis does not send the necessary data and 9-axis is unnecessary for the needs of this project. The BMI088 has four interrupt pins, but they are not used on the PCB as the sensor is only intended to collect data; there are no systems in place which could react to any sudden motions deserving of an interrupt, so no interrupt is necessary. It should be mounted close to the snowboard to gather the snowboard's yaw, pitch, roll, and translational movement across the x, y, and z axis.

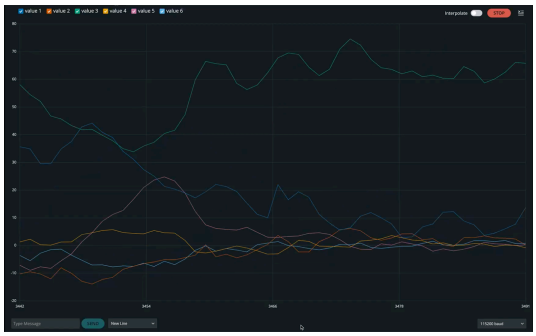


Figure 17: Raw data from IMU in motion plotted using Serial Plotter in Arduino.

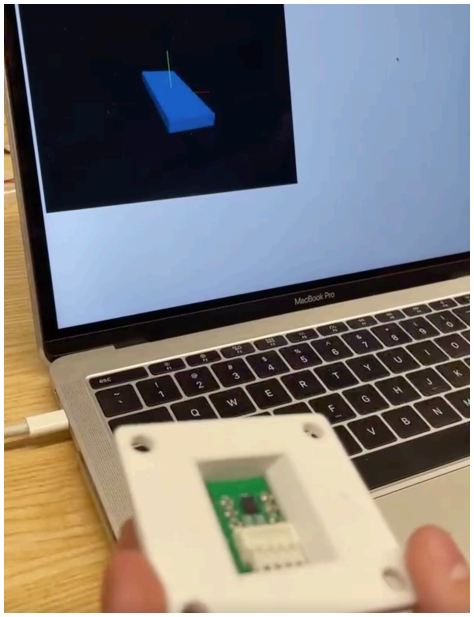


Figure 18: IMU being simulated as a rectangular prism which uses accelerometer and gyroscope data to replicate the motion of the IMU.

Load Cells

To record the riders' force distribution across the snowboard, an array of load cells were used. The load cells are positioned so that there's a load cell positioned for the heel and ball of each foot. The load cell selected was the FX1901-0001-0200-L. This was the load cell of choice due to the high operating force of 200 pounds. This means that the array of four load cells are rated for up to 800 pounds, allowing for the weight of the rider, board and mounting components.

The load cell is a push-button compression load cell, and it operates at a +5V supply level. The load cell is a wheatstone bridge, with the red and black wires being the power wires and the white and yellow wires being the signal wires. When compressed, there is a voltage increase across the white and yellow differential signals, due to the changing resistance in the wheatstone

bridge. This differential signal is proportional to the compressive force of the load cell and needs to be amplified using the load cell amplifier.

- Force being distributed on load cell

Figure 19: Load Cell

Load Cell Amplification

The selected load cell amplifier/Analog-Digital-Converter was the ADS1231 from Texas Instruments. The output of the ADC is a 24-bit SPI-esque digital output such as data output, clock, power-down and speed. 24 bits allows for a theoretical resolution of 5.40 milligrams of compression per bit. The load cell amplifier is capable of outputting at a rate of 10 or 80 samples per second, however 10 Samples per second was selected to optimize accuracy and decrease load cell error. However, when the load cell and load cell amplifier are setup, there is a plateau of roughly 20 pounds or so. We weren't able to pinpoint the cause of the problem with extensive troubleshooting.

Microcontroller

The selected microcontroller for this project was the ESP32-Wroom32UE from Espressif Systems. This was used for its 3.3V input, I2C and SPI (for load cells) interfaces, compatibility with Arduino IDE, over 25 GPIO pins, and using an external Antenna 2.4-WRT-CCC for better bluetooth and wifi connectivity. Additional connections included a USB to UART bridge, reset buttons, and a USB-C connection.

Haptic Feedback

The purpose of haptic feedback is to allow the user to get instantaneous feedback from the software depending on the output from the various sensor systems. Haptic Feedback in the product is achieved with two vibrating motors. The vibrating motors are simple DC motors with an eccentric rotating mass. The motors are rated for 5 VDC and a typical current draw of 0.55 Amps. For comparison, these motors are typically used in various consumer products such as video game controllers and massage guns.

For the haptic feedback system to be effective, the vibrating motors were placed as close to the user as possible, on the back of the ski boots.

To drive the motors, the motor driver DRV8830DGQR was selected due to the components ability to draw from a battery. The motor driver's communication protocol is I^2C, with the ability to vary the addresses to have independent control of each motor when using two different drivers. The drivers are rated for up to 6.8 Volts and 1.3 Amps, so a more powerful motor can be used in place of the selected ones if desired.

Battery Management System

Based on our power consumption estimates, we selected a 3.7V, 10Ah battery which allows our system to have a lifespan of around 12 hours. On top of this, we use an Adafruit USB charger to easily charge our system by plugging in a USB-C connector. We also included a rugged push button as an enable button to allow users to turn on and off the system easily. With this battery voltage, we were able to easily use a boost converter to go up to 5V for the rotary encoder and load cells, and a Linear Drop Out regulator to go down to 3.3V for everything else.

Software Design

Explanation of Overall Software Decisions

The idea behind the overall system is to collect sensor data on the esp32, which sends this data over bluetooth to another computer which can process and interpret the data into visualizations and other interfaces. The data from the load cells, IMU, and rotary encoder would allow the simulation to accurately create a model that emulates the movement of the snowboarder if on a slope moving downwards.

For our scope of the project, we focused the hardware and decided on making code that functions locally on a single device plugged in. Currently, we have rotary encoder data that is sending serial data to a python script which creates a graph of the snowboarder's location, specifically its horizontal movement with a fixed speed. What it does is take the horizontal motion of the board, scales it, and interprets this as the horizontal motion of a line on a python graph. Note that the vertical speed is constant but could be changed based on load cell or IMU data. Currently, this code does not include data from the IMU or the load cells, but since these individual systems work, with more time they can be integrated into a larger simulation/UI.

Next Steps

Mechanical

- Run more user testing
- Continue to grease the rails to minimize friction
- Add extrusions to the front and back of the assembly to add more stability

Electrical

- Combine sensor and tracking code
- Troubleshoot load cell weight limit
- Integrate bluetooth to sent data results to PC
- Merge hardware code with VR simulation

Appendix

Arduino IDE Rotary Encoder Coder to get Data:

```
//for Arduino testing:
const int PIN_CS = 4;    //blue
const int PIN_CLOCK = 6;  //yellow
const int PIN_DATA = 7;   //green

// //for ESP32:
// #define PIN_CS 29      //blue
// #define PIN_CLOCK 28    //yellow
// #define PIN_DATA 27     //green

void setup() {
 Serial.begin(115200);
 pinMode(PIN_CS, OUTPUT);
 pinMode(PIN_CLOCK, OUTPUT);
 pinMode(PIN_DATA, INPUT);

 digitalWrite(PIN_CLOCK, HIGH);
 digitalWrite(PIN_CS, LOW);
}

//byte stream[16];
void loop() {

 digitalWrite(PIN_CS, HIGH);      //CS must be driven high once to get data
 digitalWrite(PIN_CS, LOW);
 int pos = 0;
 for (int i=0; i<10; i++) {
  digitalWrite(PIN_CLOCK, LOW);   //get position data from bit corresponding to CLK signal
  digitalWrite(PIN_CLOCK, HIGH);

  pos = pos | digitalRead(PIN_DATA);    //binary to int
  if(i<9) pos = pos << 1;
 }
 for (int i=0; i<=6; i++) {       //parity/status bits, ignore
```

```
   digitalWrite(PIN_CLOCK, LOW);
   digitalWrite(PIN_CLOCK, HIGH);
  }
  Serial.println(pos);
  delay(100);
}
```

Python Rotary Encoder Graphing:

```
import serial
import time
import sys
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from collections import deque
import threading
import numpy as np

#Rotary encoder BaselineTech code
#creates graph simulating the snowboarder's position on the snow
#for testing purposes separate of snowboarding simulator, align blue tick mark on encoder with
tick in casing. first reading should be ~605 if aligned properly

# Set up the serial connection
try:
   ser = serial.Serial('/dev/cu.usbmodem101', 115200, timeout=1)  # update port as needed
   time.sleep(2)
except serial.SerialException as e:
   print(f"Error: Could not open serial port. {e}")
   sys.exit(1)


pitch_factor = 1.0        # USER: define the pitch factor based on snowboard pitch


# Shared data buffers
x_data = deque(maxlen=500)
y_data = deque(maxlen=500)
```

```
start_time = time.time()
running = True  # Flag to control the serial thread

# Add a thread lock
data_lock = threading.Lock()

# Add a variable to track the last valid value
last_valid_value = None

set_origin = 0
origin = 0
pos = 0

# Function to calculate y-axis value
def calculate_y_value(time_elapsed):
  return pitch_factor * time_elapsed

# Function to calculate moving average
def moving_average(data, window_size=5):
  if len(data) < window_size:
    return data  # Not enough data to smooth
  return np.convolve(data, np.ones(window_size)/window_size, mode='valid')

# Serial reading in a separate thread
def read_serial():
  global last_valid_value, set_origin, origin, pos
  while running:
    try:
      data = ser.read(ser.in_waiting).decode('utf-8').strip()  # Read all available data
      lines = data.split("\n")
      with data_lock:
        for line in lines:
          if line.isdigit():
            raw_value = int(line)
            print(raw_value-origin)

            if set_origin == 0:    #get original value, use as central point
              origin = raw_value
              set_origin = 1     #origin only gets set once, at beginning of run
```

```
                if last_valid_value is None:       #init last valid value at begin of run
                    last_valid_value = raw_value

                #check for unexpected readings and ignore if bad (difference over 200 but below
900 to preserve 0 to 1024 continuity)
                if abs(last_valid_value - raw_value) > 200 and abs(last_valid_value - raw_value) <
900:
                    raw_value = last_valid_value
                else:
                    last_valid_value = raw_value       #if it was within expected range, update the last
valid value

                #adjust the scale factor as needed for sensitivity, likely after testing. 25-50 range
seems optimal
                pos = (pos) + (raw_value-origin)/30     #calculate pos based off previous position and
deviation of encoder from 'origin'
                if pos < -99:
                    pos = -99
                if pos > 99:
                    pos = 99

                # Append valid data to be plotted
                x_data.append(pos)
                y_data.append(calculate_y_value(time.time() - start_time))
            except Exception as e:
                if running:
                    print(f"Error reading serial data: {e}")
            time.sleep(0.002)
# Start the serial reading thread
serial_thread = threading.Thread(target=read_serial, daemon=True)
serial_thread.start()

# Function to update the plot
def update(frame):
    with data_lock:  # Lock access to shared data
        x_data_plot = list(x_data)[-50:]  # Last 50 points
        y_data_plot = list(y_data)[-50:]
```

```
    if len(x_data_plot) > 0 and len(y_data_plot) > 0:
        # Apply smoothing to y_data
        y_data_smooth = moving_average(y_data_plot, window_size=5)

        # Ensure x_data matches the length of smoothed y_data
        x_data_smooth = x_data_plot[-len(y_data_smooth):]

        ax.clear()
        ax.plot(x_data_smooth, y_data_smooth)
        ax.set_title("Real-Time Snowboard Simulation")
        ax.set_xlabel("Left-Right Position (X-axis)")
        ax.set_ylabel("Speed based on Pitch (Y-axis)")
        ax.set_xlim(-100, 100)  # Fixed x-axis range
        ax.grid()

# Function to safely close the serial port and stop the thread
def close_serial():
    global running
    running = False  # Signal the thread to stop
    serial_thread.join()  # Wait for the thread to finish
    ser.close()  # Close the serial port
    print("Serial port closed.")

# Set up the plot
fig, ax = plt.subplots()
ani = FuncAnimation(fig, update, interval=1)    #update every 1ms

# Hook into the closing event of the plot
def on_close(event):
    close_serial()  # Clean up when the plot window is closed

fig.canvas.mpl_connect('close_event', on_close)

plt.show()
```