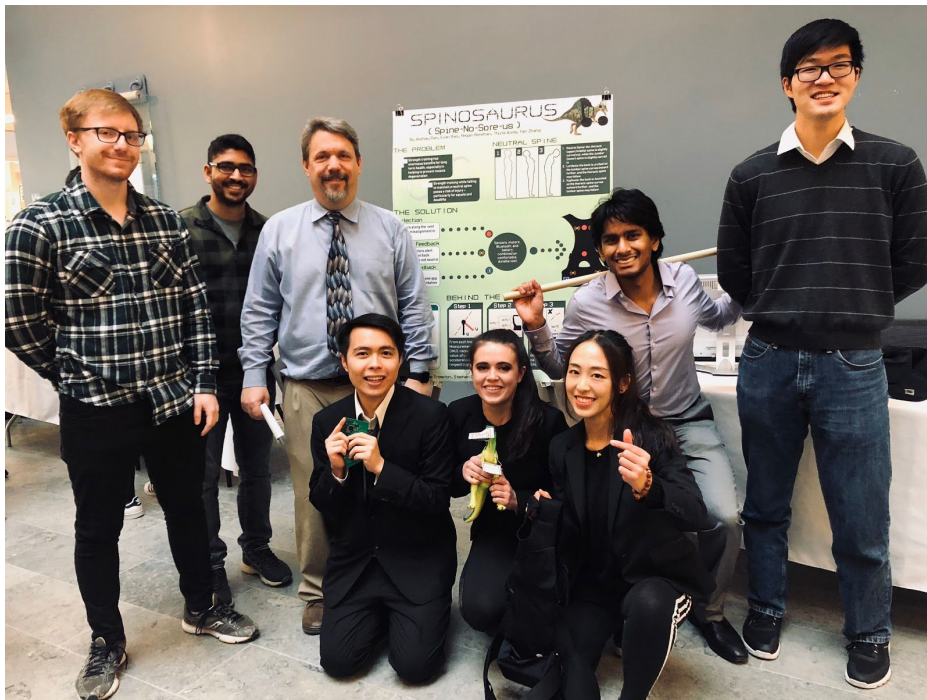


Spinosaurus

Senior Design Project Report



Akshay Rao, Evan Bao, Megan Renehan, Yan Zhang, Zack Chen

*Electrical Engineering and Computer Science
University of Michigan
{akshayro, ebao, mrenehane, sunfr, zackchen}@umich.edu*

I. INTRODUCTION AND OVERVIEW

The proposed goal of the Spinosaurus (known by codename FSAT during development) was to provide weightlifters feedback on their back form while squatting and deadlifting. Our final product met our original specifications while hitting additional stretch goals in user interface design.

Our initial research into the problem of poor exercise form demonstrated a clear need for a device like the Spinosaurus. There has been extensive research into the benefits of different types of exercise to overall fitness, and almost all of it has come to a similar conclusion: resistance strength training is beneficial for everybody's long-term health. Muscle degeneration is one of the leading causes of impairment in older adults [1]. Also, osteoarthritis (breakdown of muscle-bone ligands in the joints) is the most common chronic disability, affecting over 3 million new Americans each year. Resistance strength training has been found to combat both effects in even the oldest patients [2].

Despite the benefits of strength training, exercising with heavy weights could pose risks if you have improper form. This is especially true for two of the most important lower body exercises: squats and deadlifts. Both exercises rely heavily on maintaining a neutral spine the entire way through, else your spinal discs could be compressed unequally, leading to possible herniation over time. If weightlifters had a device that could measure the arch of their back while lifting and give them feedback on their form, they could exercise safely and effectively even without a personal trainer. This would not only be a popular product with a wider variety of exercise enthusiasts, but it would also encourage a larger portion of adults to engage in strength training, improving overall societal health. To tackle this problem, we decided to create a device that would achieve the following goals:

- Measure the relative angles between different points on a user's back while exercising to detect the overall orientation of the spine (more specifically, identify incorrect back form pertaining to the thoracic (upper) and lumbar (lower) spine individually)
- Translate our collected data on spinal orientation into user-readable feedback that indicates the current position of the spine
- Send this data wirelessly to a device that can display this information to the user in real-time
- Provide an interface that will allow the user to start or stop feedback on demand

We tackled this problem by developing the Spinosaurus, consisting of a vest (the SpineAligner) that stretches across the spine and an Android phone app. The vest has the ability to measure the orientation of the user's back with three inertial measurement units (IMUs) and then communicates with the phone app to display visual feedback.

The final demo version of the Spinosaurus was able to reliably detect incorrect form in the upper and lower back, parsing the IMU orientation data into 15 different back "states" that can be easily understood by the user. The quality of our detection system was vetted by Rosanne Crompton [3], a physical therapist at the University of Michigan University Health Service. The vest was fitted with vibration motors on the upper and lower sections, which provided haptic feedback based on the type of incorrect form.

We also developed an Android phone app which could communicate with the vest in real-time and provide feedback with a visual representation of the user's form. In addition, the user could also save a profile on the app, which they could use store previously collected calibration data. We also hit our stretch goal of being able to start and stop workout "sessions", which would tell the app to

save the the proportion of time spent in each back state, which the user can look at later to evaluate the quality of their workout as a whole.

All of these features worked without major issues during the demo. We did have trouble integrating the motors correctly on one of the vests (the smaller one), which will be clarified in a later section.

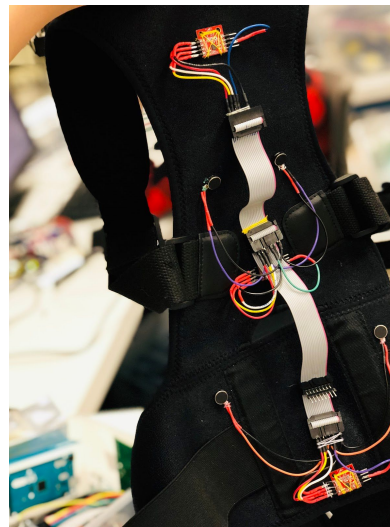
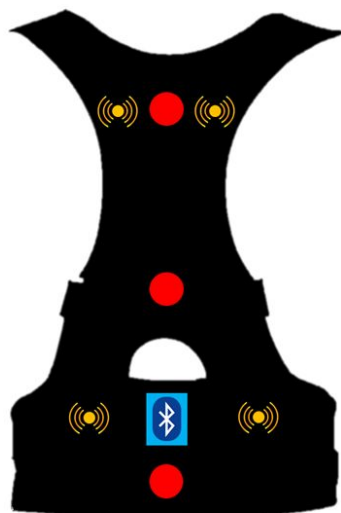
II. DESCRIPTION OF PROJECT

i. Goal, System Concept, and Feasibility

To meet the goals described above, we split up our group into two design teams: the SpineAligner team (vest module and form detection algorithm) and the app development team (Android app development and Bluetooth module implementation). Both the mechanical and software design of our product were important to meeting our goals, so we will describe both aspects in detail.

SpineAligner: Physical+Electrical Design

The SpineAligner module was built on top of a currently existing fabric back brace [cite]. We sewed on our IMUs at three locations across the back, covering the thoracic and lumbar spine of the user. The central PCB (with an AtMega processor and low-energy Bluetooth module) was placed in a fabric pouch, which was sewn between the lower and middle IMUs. All IMUs communicated with the processor through I2C. In addition, the four vibration motors were sewn into place at the top and bottom of the vest: two on both sides of the thoracic spine (on the trapezius muscles) and another two on both sides of the lumbar spine (on the latissimus dorsi muscles). All outputs and inputs were wired through a ribbon cable that ran along the spine. Finally, all wiring and components were covered with a water-resistant nylon fabric. The vest with and without the fabric is shown below.



SpineAligner Module: Diagram and Final Product

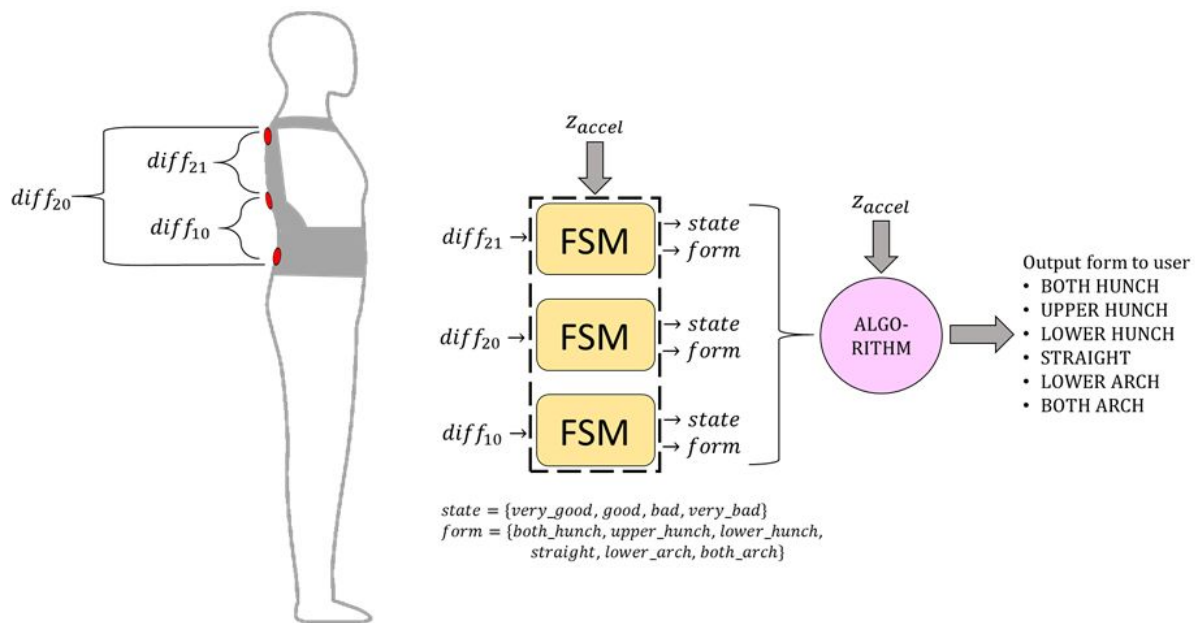
SpineAligner: PCB Design

The project utilized two iterations of the PCB. Our initial PCB design consisted of four modules: AtMega2560, rechargeable circuit, Bluetooth, and battery management. The AtMega2560's circuit was modeled after the Arduino Mega reference design, excluding most of the irrelevant features such as extra GPIO pins and UART channels [4]. As for the rechargeable circuit, we referenced a circuit found available online by GreatScott [5]. After initial testing, we discovered various issues with the PCB, such as unconnected nets, incorrect pin types, and poor placement of capacitors.

For our second PCB design, we fixed all of the previously mentioned issues and added more serial broken out pins to facilitate the debugging process. We also removed the battery management unit as the module itself was too small to solder and was shorting the I2C pins, compromising communication with the IMUs. Lastly, we included a brand new LDO as the previous LDO had power integrity issues due to an unstable current supply. Final PCB designs are posted in the reference section.

SpineAligner: Algorithm Design

The detection algorithm is summarized as follows: since three IMUs are lined up along the spine, the difference in y-axis (along the spine) acceleration between each subset of two IMUs are the inputs to the algorithm. To determine the thresholds, we tested different postures to calculate a base set of delta values. During testing, we found that the user's delta values at neutral spine alignment varied with the back's angle, so we adjusted calibration to account for various angles. The z-axis (perpendicular to spine) acceleration value of the middle IMU can be used to detect if the user's entire back is at an angle of 0° (upright), 45° (tilted) and 90° (nearly parallel to the ground). For each angle range, the thresholds and the logic on the detection are marginally different.

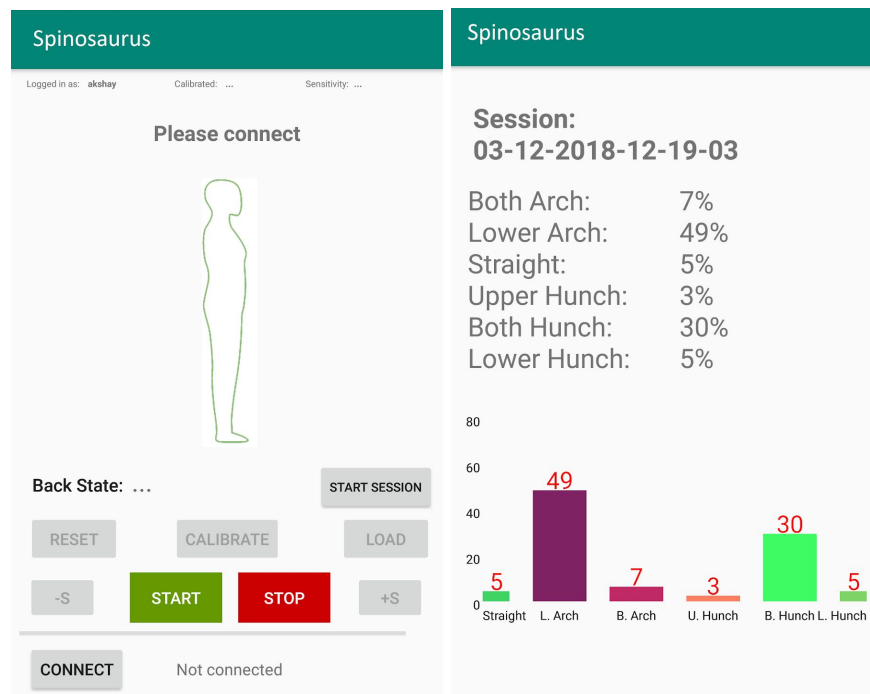


Neutral Spine Detection Algorithm

Android App Design

The Android application both controls the SpineAligner and displays feedback data that is received over Bluetooth. When you first enter the app, they enter the "Profile Creation" section, where you are prompted to create a new profile or select an existing one. All data pertaining to a profile is saved to the phone's local storage. After creating a profile, you enter the "User Menu" section, where

you can either start a new workout session or view spine data from previously saved sessions. After beginning a new workout, you enter the most important part of the app: the “Feedback” section.



Android App

In the Feedback section, the user can either hit the CALIBRATE button to enter calibration mode, or LOAD to load previously saved calibration data (if it exists). After entering calibration mode, the app enters a handshake sequence with the SpineAligner and prompts the user to hit a button while standing straight at 0°, 45°, and 90°. After saving the data to the user’s profile, the user is then allowed to hit the START button to tell the SpineAligner to send back feedback data for display. The user may also hit the START SESSION button, which begins a “workout session” in which the app saves the amount of time spent in each back state. Once the user exits the app or hits STOP SESSION, the session is saved to view later.

Bluetooth communication between the app and the BLE module on the PCB was established using a GATT (Generic Attribute Profile) protocol [6]. The AtMega processor on the SpineAligner was running as a GATT server, while the phone app was a GATT client. We created specialized GATT characteristics to implement UART communication between the client and server, allowing us to send and receive data through an easy to use TX/RX interface. Our implementation was heavily based on a tutorial written by Thejesh GN on his blog [7].

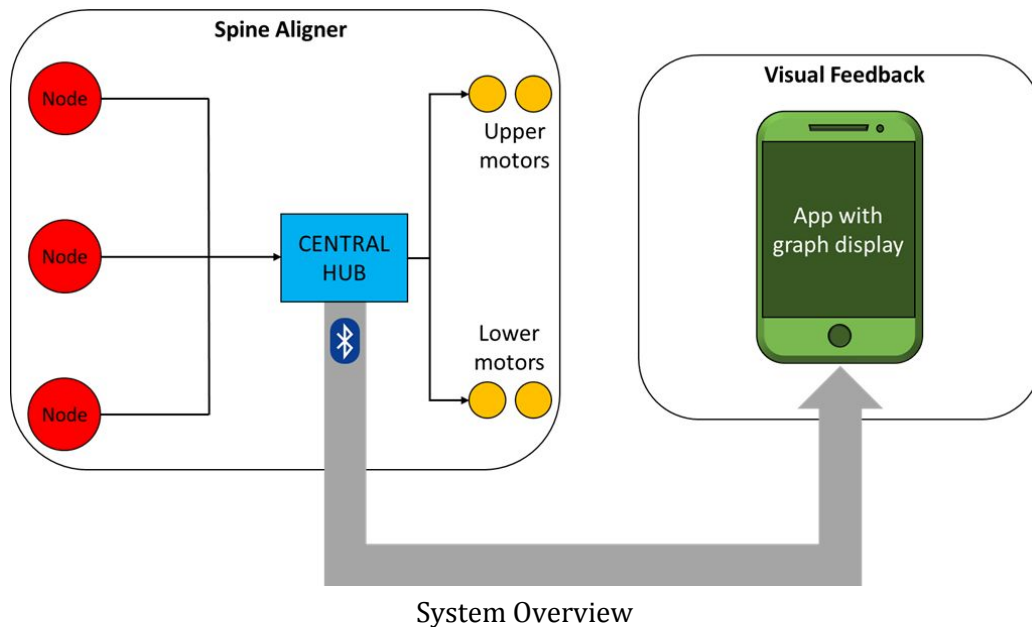
ii. Design Constraints

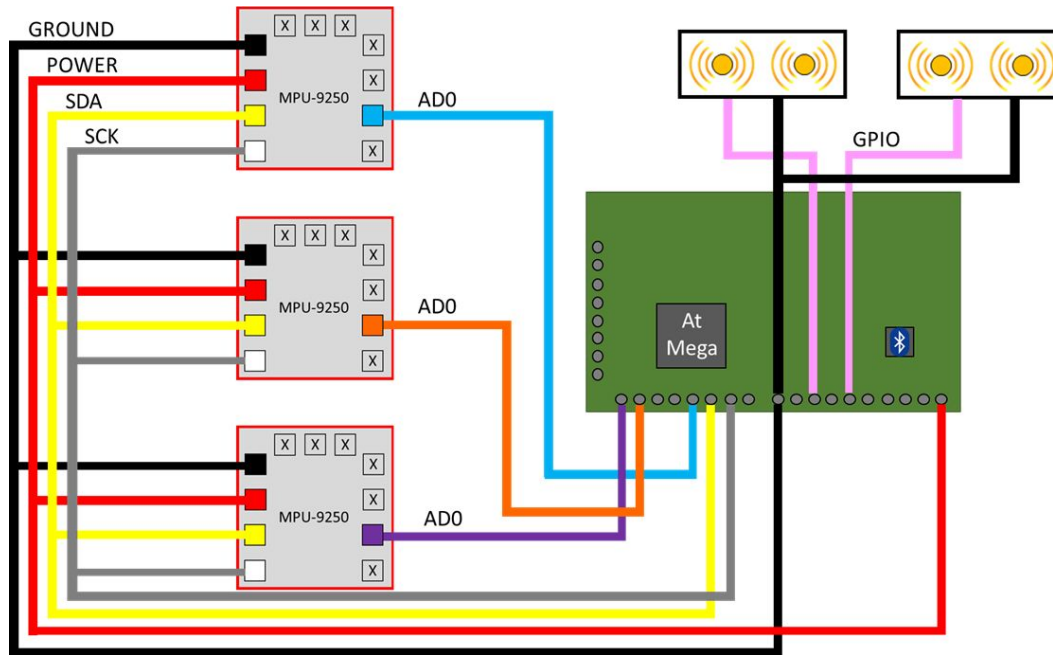
Constraint	Specification	Details
Size	Fits within lower back section	The PCB was a comfortable size which sat within the lower back of the vest without impacting the ability to squat or deadlift.

User Feedback	Haptic and Visual	Motors were included in the upper and lower areas of the vest for immediate feedback.
Battery Life	> 2 Hours per charge	The vest should last for a minimum of one workout session per full charge. The final vest had a battery life of over 3 hours on a full charge.
Accessibility	"User-friendly"	The system included a 5V USB charging port and a physical on/off button. The casing for the PCB can be easily taken off from the vest, and the PCB can be recharged.
Water Resistance	Sweatproof	The PCB was enclosed within a plastic case to reduce the chance of being damaged by moisture. Furthermore, the vest was covered by a moisture resistant cloth.


iii. System Architecture

Below is a generic diagram of the system architecture. The system reads accelerometer data from three IMUs, which is processed at the "central hub," the AtMega chip on our PCB. From here, the data is communicated to the user via vibration motors (controlled by setting GPIO pins on the PCB low/high) and via a phone application, which receives information over Bluetooth.





SpineAlginer Connections

	Character Sent	System response
	c (calibrate)	Y-accelerometer values are recorded at 0, 45, and 90 degrees, and used as offsets in the algorithm; these values are also sent back to the app via Bluetooth
	l (load)	Previous calibration (offset) values are sent via Bluetooth to the AtMega, and are stored and used as offsets in the algorithm
	g (go)	System output is enabled so that the user can see their phone in the app, and vibration motors are enabled
	s (stop)	System output is disabled, and vibration motors are disabled
	r (reset)	System output and vibration motors are disabled, and offset values are recalculated
	i (increase)	Sensitivity is uniformly increased, meaning threshold windows are narrowed (system allows for an increase three times)
	d (decrease)	Sensitivity is uniformly decreased, meaning threshold windows are widened (system allows for an decrease three times)

Android App Communication Protocol

III. MILESTONES, SCHEDULE AND BUDGET

We stuck to our original schedule fairly closely for most of the semester, up until the first batch of PCBs arrived. Testing and breadboarding the individual components (IMU prototyping, algorithm formulation, Bluetooth communication, and phone app back-end) worked well, but we encountered a multitude of problems upon testing the PCB, thus postponing system integration. The schedule of the major sections and milestone deliverables are shown below.

i. IMU

Milestone 1: Demonstrate a working calibration method for a person's neutral spine; show the IMU data for aligned spine posture can be recorded and improper postures with deviated data can be detected; use working IMUs + processing algorithm on devkit

For Milestone 1, the IMU algorithm was shown by two IMUs installed at the ends of a flexible ruler as a spine simulation. The algorithm only produced binary results, good or bad. We planned to calculate the quaternion and roll-pitch-yaw from the acceleration, angular momentum, and magnetic field to obtain angle information, but these values were too fluctuant and were subject to drift, so ultimately we decided to use only acceleration values. To filter out spikes in data, we used a finite state machine (Very Good - Good - Bad - Very Bad). The basic algorithm was effective enough to distinguish a bent ruler from a straight ruler, demonstrating that our project idea had potential.

Milestone 2: Tinker with processing algorithm as needed to work on human spines

At Milestone 2, the prototyped SpineAligner was almost complete. Three IMUs were sewn to the vest and the algorithm could differentiate between the postures. The calibration consisted of the user standing straight with a neutral spine and form was detected by subtracting the offsets from the instant acceleration values and checking how close the values were to zero (neutral). At this stage, we ran into the issue of spine conditions changing as the user bent over, causing even the neutral spine values to diverge (since human spines are not as simple as rulers). This stage was a roadblock that needed to be solved for the project to proceed successfully.

To potentially fix this issue, we decided to increase our data size by collecting posture data from all five members. This additional data helped improve the algorithm's accuracy, but the issue still persisted. The next potential fix was to categorize data into three different angles: 0°, 45°, and 90°, tripling the amount of data we collected. By utilizing the increased dataset, we were able to determine new threshold values. This combined with adding 45° and 90° to our initial calibration meant our algorithm was finally working on humans.

After Milestone 2, the SpineAligner module included basic Bluetooth communication. The processor constantly listened to commands from a phone app via Bluetooth, executed corresponding tasks, and then sent back the instant spine forms to the app. The functions included calibrating, starting, stopping, changing sensitivity, etc.

On Nov. 20, we met physical therapist Rosanne Crompton at UHS to show her the working vest on a high-level athlete, and she approved the effectiveness. By the time the PCBs were delivered, the two vests were refined to the point where the algorithm would no longer require major changes. On Nov. 28, we met assistant research scientist Stephen Cain to talk about more advanced methods to potentially increase accuracy. The main focus was showing us the correct usage of quaternion values, but this proved to be too much to implement with our time being limited by PCB debugging.

ii. Bluetooth Module

Milestone 1: Test with Bluetooth communication

For Milestone 1, testing with the Bluetooth breakout board went well. We wrote a simple C++ backend for the Arduino to send and receive test messages through BLE to a generic BLE UART phone app. Overall we did not run into any major issues with the bluetooth module. The code written in this portion was used for the final product to communicate with the phone app.

iii. Phone App

Milestone 1: Demonstrate a test version of the phone app, using data that we expect to obtain from the SpineAligner module

At Milestone 1, the phone app was a basic mock visual mockup that showed how data would be displayed to the user, tested with dummy data. This included basic profile creation and loading, as well as our initial idea of having circles for each IMU that the user would need to line up to achieve a straight back.

Milestone 2: Have an integrated SpineAligner + Visual Feedback prototype, which demonstrate calibration and basic use with minimal issues

The largest roadblock occurred while preparing the device for Milestone 2. Writing the Bluetooth backend for the app was much more complicated than expected. Even though we referenced various BLE android app tutorials, issues always occurred when trying to scan and connect. Finally, after close to a week of trial and error, we were able to fix the issue by referencing the Android Studio sample BLE project. The issue was partially due to insufficient permissions and partially due to incorrectly calling the BLE API.

At this point, the basic functionalities of the phone app were complete. Multiple profiles could be created on the app and any necessary calibration data could be saved to a file associated with that profile. Furthermore, the app would spawn a separate thread to check for nearby Bluetooth devices, prompting the user to select the device with the MAC address corresponding to the SpineAligner and connect after a “handshake” signal was returned. When connected, the app was able to continuously read data from the SpineAligner.

After the Milestone, we made several iterations to the visual feedback of the app. Originally we had planned to display the user’s spine as a series of dots moving back and forth along a line, but we feared this would not work as well with only three IMUs. Instead, we decided to display static graphs showing the current posture and compare it to the correct posture. These postures were custom drawn for our project to ensure clarity. As the project continued, more and more features were added, such as buttons for calibration, begin monitoring, stop monitoring, changing sensitivity level, recording and showing session performance, re-calibration, and more.

iv. PCB and Integration

Milestone 1: Should have a good idea of how the PCB with the main processor will be laid out

At Milestone 1, we had a basic layout of the first version of PCB. The major components of the PCB included the microprocessor, the Bluetooth module, the charging module, the voltage regulator module, and the power monitor module. The power integrity and general layout was verified with Steven Rogachi, senior engineer in research at Space Physics Research Lab.

Milestone 2: PCB design should be in the mail; Plan ways it can be protected to be durable/waterproof

For Milestone 2, the first version of PCB was sent out for manufacturing. We found a plastic casing with perfect size of the PCB and holes for wiring, so we decided to case the PCB in it. We also bought waterproof cloth to protect the PCB and other components on the vest.

The project fell into great difficulties after we began working with the PCB. The first PCB could not burn the bootloader with all the components already soldered. Our team spent five days (over 60 hours) debugging the PCB, and finally got it working after replacing the on-board Bluetooth chip with the Bluetooth breakout board, replacing the LDO with a diode, and abandoning the charging module and the power monitor module. However, we could not use the USB serial port to upload the program, so we were forced to use the SPI ports on board. Due to all these issues, we needed to order a second PCB, which was sent out on November 26th.

The second PCB arrived on November 30th, which is when our team began testing the board. This board also ran into issues, resulting in our team camping the MESH lab to try and fix them in time for the Design Expo. The board could not upload programs because the Bluetooth SPI lines were conflicting with the lines for uploading. The motors were drawing too much current so we needed to edit the board to utilize a separate battery solely to power them. The MOSFETS used to control the motors were also of the wrong type (needed to be PMOS, not NMOS), which caused the motors to not receive enough power. The LDO also broke a IMU as the circuit was not correct so the output voltage was higher than expected.

Because of all these issues, the actual schedule deviated greatly from our plan. It wasn't until the day before the Expo that the PCB with all components fully integrated finally began working properly. Somehow, the stars aligned and the project could be completed on time.

v. Wiring and Durability

Milestone 2: PCB design should be in the mail; Plan ways it can be protected to be durable/waterproof

For Milestone 2, we ordered ribbon cables to clean up the wiring, waterproof cloth to protect all the components and obtained a casing to hold the PCB. The implementation of wiring and durability stalled until we managed to debug the PCB. A ribbon cable was used to cleanly connect the IMUs and vibration motors with the PCB. This ribbon cable took a while to get working properly due to the need to adjust the PCB to accept the header, but it was not too difficult. Beyond that, the vest was also covered with sewn on waterproof cloth and the PCB casing was placed in a pocket sewn to the vest. With these adjustments, the final product was portable and durable in time for the expo..

vi. Budget

The budget was basically consistent with the plan. In the last period of the project, we ordered a second version of PCB and several parts like the ribbon cables and clamps, but the whole budget was acceptable. The total budget was \$938.43.

IV. LESSONS LEARNED

It was quite a journey from the group formation meeting, when the project idea was born, to demo day, when the team (finally) finished. Let us look back and reflect on all that they have learned.

Right away, the project proved itself to be difficult. The team members in charge of developing the algorithms for the IMUs, Yan and Megan, thought it would be simple to read from multiple IMUs on the same bus, yet they spent an entire night debugging this basic, beginning stage of the project, which taught them to read data sheets very carefully, as it turned out that the IMUs came with the address pin already soldered to ground, which was a major issue since it was necessary to drive all but one of the address pins high in order to read from multiple IMUs on the same I2C. If more attention had been placed on reading the data sheet, Megan and Yan would have saved themselves many hours of grief.

After this initial issue, getting simple readings from multiple IMUs was not too difficult; however, once the team turned their attentions to more complex values, such as quaternions and Euler angles, they were faced with a multitude of problems. The readings were slow to update, changing erratically, and often did not make any sense. The poor souls spent about a week leading up to Milestone 1 attempting to use these values to detect whether or not two IMUs were in a straight line. Then, the group's savior (Sharan) suggested that the team simply use the y-accelerometer values, which ended up being the values used in the final version of the algorithm. Group members learned that sometimes the more simple solution works better, at least as a jumping-off point (though, in this scenario, they chose to stick with the simple values), and that Sharan is the best.

After Milestone 1, the team began to develop the IMU algorithm very gradually, often feeling like they went to lab and worked so many hours, yet still accomplished nothing. However, slowly but surely, they developed the algorithm and found appropriate threshold values, and this algorithm ended up working pretty well on team members Megan and Yan; the team felt rewarded for their hard work and patience. Unfortunately, after trying the algorithm on fellow team members Zack, Akshay, and Evan, it was evident that adjustments needed to be made. First, the calibration method was altered to include initial values at three different angles, rather than just one (standing up straight). Then began the data collection. Many hours in lab were spent collecting and analyzing data to find new threshold values and adjust the algorithm accordingly. The new threshold values, which were only adjusted small amounts for the final version, worked much better. Thus the team learned the importance of collecting a plethora of data, as their initial, garbage threshold values were found with only a few data points per angle per position on one person, whereas the new and improved threshold values were found on five different people with around 200 data points per angle per position.

With the algorithm working, the team turned their sights to Bluetooth communication. Surprisingly, communication with the Bluetooth breakout board went well, and adding Bluetooth functionality to the code required some debugging, but much less than other parts of the project. We were hopeful that our Android app development would move along smoothly, leaving us plenty of time to debug the SpineAligner itself. Alas, this was not so.

Our initial goal for the app was simply to establish the app as a GATT client and scan for all surrounding devices. Unbeknownst to us, Android recently updated all of their built in BLE interface code, so most of the tutorials we found on the subject were obsolete. For days we swam through an ocean of blog posts, articles, and documentation on implementing Android GATT

servers, usually to no avail. Eventually, we came across a blog post that gave a detailed explanation of implementing a pseudo UART protocol as an Android GATT server, which ended up working flawlessly. From that point on, app development was quite smooth -- Akshay continued to update the user interface little by little until the demo day. However, this would only be a small moment of respite before the storm.

Now we enter the darkest chapter in the team's journey: integrating the PCB. Hardware became their downfall. Once the PCB work began, they felt as though they were cursed. The first PCB required an enormous amount of debugging. As one team member, Yan, liked to say, every step took the team eight hours. First, just bootloading Arduino on the board was an issue. When we finally managed that, it took us another eight hours to upload a simple sketch to blink an LED -- the group members recall that this was finally completed at 4am with Sharan still helping them debug. After that, it again took endless hours to communicate with the IMUs via I2C. Once that was done, getting the Bluetooth chip to work was discovered to be impossible, and the team simply used the Bluetooth breakout board, though even getting that to work was a battle. All of these steps required many hacks on the board to get anything to work. Almost all of this debugging would have been impossible without Sharan (the savior) spending all his time with the group in lab. Due to all the issues with the first PCB, its garbage silkscreen, and the inability to get the on-board Bluetooth working, the team was unfortunately forced to order a second board. From their first PCB, the team learned that debugging, especially with hardware, is extremely time intensive and requires patience and careful thinking. In creating the second board, the team learned to be a little more careful with the design of the PCB, and they also ensured that the silkscreen was readable.

Unfortunately for the group, the second PCB didn't have a visible silkscreen, which provided an additional challenge of always needing to triple check the eagle files to make sure connections were correct. Debugging on the second PCB went smoother and did not require the hacks that the first board needed. However, there were issues with the crystal oscillator, which were also present on the first PCB, so the team turned to the wise and mighty Sir James, who educated them on what capacitor values were needed. When the team managed to get the on-chip Bluetooth working at 4am one night, they were overjoyed, and spirits were high. Approaching demo day, the group was feeling as though they would finish debugging their technical issues and have time to make the project look nice. However, here we meet the true villain of this tale: the LDO. This beastly component, which was thought to be fixed but was actually adjustable, initially tricked the team into thinking it worked; little did the group know, the board was actually running at 4V instead of 3.3V. So, at 6:00 am on Wednesday, the great and mighty Spinosaurus fell. Our first line of defense, the IMUs, suffered a loss, and several GPIO pins also burnt out, causing debugging to be severely difficult. At this point, everyone in the group was extremely tired and felt completely crushed. The despair is evident in an email to the professor at 7:00am, as seen in Appendix I. The group felt that all hope was lost, and although the professor issued a retreat, the team pressed on, determined to complete the project by Thursday. They knew, deep down perseverance was the key. With hard work, determination, caffeine, a great GSI wizard named Tan, and three resistors (to fix the LDO issue), the team managed to get everything working by 6:00am on demo day, and spent the few remaining hours before the demo making their project look nice.

Our hardworking team was rewarded, as the demo went smoothly, without anything deciding to break in the middle. At the end of this tale, we can look upon lessons learned by the group (some were mentioned during the story, but we feel it is important to recap):

- Developing a complex algorithm takes time, patience, and a large amount of data and data analysis
- For all debugging, the oscilloscope is one's greatest tool; when in doubt, look at the signal on the scope
- Always ensure that ground is ground, for the entire PCB
- Naming nets is key when creating a PCB
 - Note: one issue that took the team quite a while to debug was due to a disconnection between two nets named BAT+ and +BAT
- Be extremely cautious and aware of what voltage you're supplying to your components, as well as the safe voltage ranges at which each component can operate
- Testing in small pieces and building from there is often a critically essential part of the debugging process
- Patience is a virtue
- Have hope, even when all is lost

Finally, we hear from each member of the group on what technical lessons they personally learned:

Akshay	I learned the ins and outs of developing an Android app from scratch, as well as how to interface the software and hardware aspects of low-energy Bluetooth by developing a GATT client and server. I also learned quite a bit about the limitations of IMUs, and the math required to convert data from IMUs into useful orientation vectors (quaternions, Kalman filters, etc.). Finally, I gained a lot of experience on surface mount soldering techniques (more importantly, ways to correct soldering mistakes after the fact).
Evan	By far the greatest improvement was my soldering knowledge. There are many tricks for soldering, such as knowing when to apply solder paste versus hand soldering components (typically smaller footprints). Furthermore, I learned much about the proper steps to debug hardware, with the first step to always ensure the circuit connections are properly set up with no shorts.
Megan	The technical knowledge I gained came from debugging the PCB. I am now much better at using the oscilloscope (previously, I was awful). I'm now very familiar with bootloading Arduino. I worked extensively with I/O, including SPI and I2C.
Yan	The most important things I learned was from debugging the PCB and integrating all the components. I have learned how to shoot problems when there seems no problem - tearing things down to small parts and testing everything step by step. Also reading the datasheet carefully was essential to guarantee we are using the right parts. I became much more familiar with the circuit stuff, the serial protocols (I2C, Bluetooth), human engineering (IMUs) and the debugging tools (oscilloscope, multimeters).
Zack	A good silkscreen is essential for soldering. Resoldering several times could potentially destroy some traces which would render the whole board useless. It is important to test out each module in PCB before integrating all of them. There are four major things to notice when debugging hardware: 1) Voltage levels for power and Ground 2) Serial Communication and digital signal levels 3) Soldering or wiring communications 4) The chip could be fried.

V. Contributions of each member of team

This is just for fun!

Team Member	Title	Contribution	Effort**
Sharankumar Huggi	The Savior		99%
Akshay Rao	Android Boy	IMU algorithm, app development, PCB debugging, moral support	0.2% (20%)
Evan Bao	Sir MOSFET	Bluetooth interface, app development, soldering, PCB debugging, Netflix account	0.2% (20%)
Megan Renehan	SPI Master	Logo, poster design, diagrams, app pictures, IMU algorithm, PCB debugging, junk food	0.2% (20%)
Yan Zhang	The Spinosaurus	IMU algorithm, sewing, wiring, PCB debugging, iconic emails, ramen	0.2% (20%)
Zack Chen	Solderman	PCB design, soldering, PCB debugging, WoW (BFA expansion, power leveling)	0.2% (20%)

**percentages in parentheses indicate contribution if Sharan is not included

Evan - My first role was to write a bluetooth interface so messages could be sent from the Arduino to a mobile app. This involved writing C++ code to utilize the Bluetooth Low Energy library to establish a connection with a phone app and emulate UART over BLE GATT to send and receive messages. While waiting for the PCB, I began work with Akshay on the mobile app for a short time, mostly researching how to write a backend using the BLE API. After the PCB arrived, my main role transitioned fully into PCB related work with Zack, primarily entailing populating PCBs, debugging the PCB, and soldering on demand.

Zack - I was responsible for a variety of things including but not limited to designing the entire PCB, soldering, PCB debugging, parts ordering and selection, wiring. Initially, I worked on parts selection and ordering for our project with Evan and Akshay. Then after deciding what parts to order I began to design the PCB over the course of two weeks. After our first PCB arrived, I populated the PCB and started debugging with Sharan, Yan and Megan. After realizing that we needed more debugging functionality and many small nitpicky hacks on our board, I designed a second PCB the following week and shipped it out. The process repeats for hardware debugging, soldering and burning sketches onto our PCB. Then when everything finally came together, I helped Yan with wiring the vest up and decide how different parts of our casing should go together on the vest. Lastly, I did a small amount of sewing on the demo day to help get the vests out on the demo as soon as possible.

Megan - At the start of the project, I was primarily responsible for working with the IMUs and developing the algorithm to detect neutral spine. Getting the IMUs to work gave me experience with I2C. While not technically challenging, coming up with the algorithm was conceptually difficult and forced me to think both logically and algorithmically. Following the completion of the IMU algorithm, I was heavily involved in debugging the PCB, including bootloading, uploading sketches,

and trying to get the IMUs and Bluetooth to work on it. This process continued until demo day. Other contributions I made were reaching out to the physical therapist and getting her to come to the design expo. Since I enjoy design-type tasks, I was in charge of creating all diagrams for all reports, drawing the pictures used in the app, and designing the amazing poster and cool logo.

Yan - I devoted the majority of the semester into the IMU algorithm. I dealt with the I2C communication protocols and developed the algorithm to identify spine forms based on the IMU acceleration values. Having tried all 9 degrees of the IMUs, the quaternion and the roll-pitch-yaw, I learned the restrictions of the IMUs that the measurements are inaccurate and subject to drift, so I focused on the acceleration solely. I collected 225 sets of data from all the five team members, plotted and compared the data, revised the calibration method, and developed the IMU algorithm to be universally applicable. I also designed the branch predictor in forms of finite state machine to filter the acceleration spikes from movements. After meeting with the physical therapist and polished the algorithm, I moved to the PCB debugging. Having been working on the Bluetooth and SPI, the ICSP pins, the MOSFET and vibration motors, etc., I helped make the two versions of PCBs work. Besides these, I changed the normal solid wires to ribbons and copper wires to clean up the vests. I sewed and fixed all components on the vests and also made a portable pocket for the central hub on the back.

Akshay - For the first month of the class, I did a lot of testing with our IMUs, as we tried to discover their attributes and limitations when it comes to measures angular differences. I helped implement a complementary filter algorithm that combine accelerometer and gyroscope data to capture IMU orientation -- however, we ended up scrapping this method since we decided to only use the accelerometer data in our final algorithm. Over the next few months of the, I primarily worked on implementing hardware aspects of low-energy Bluetooth communication (with Evan), and then started to learn Android development (Java) to start working on the phone app. I developed the entire phone app (GATT client/server), and continued working on it until demo day. In the days before demo day, I helped out with last minute soldering and PCB fixes. I helped fix the LDO issue we were having by soldering on three resistors onto the output and feedback pins, which ensured that we could have a stable 3.3V output.

VI. COST OF MANUFACTURE

Note: Within the projection of the price for the Spinosaurus, the majority of the passive components such as inductors, resistors and capacitors have been omitted. In order to account for their costs, one can add around 20 dollars at most for all the quantities up till 200 units (see Appendix II for more information).

We believe the best price break is at 25 units which would cost around \$67 per unit including the passive components. This cost can be compared to the actual price within the market and against our competitor which for UPRIGHT GO currently stands at about \$75 per unit. If we intend to make profit with the product, a more reasonable price break would be at 100 units which would result in each unit costing about \$40. Moreover, since the Spinosaurus is in its first prototype stage, there could be cheaper and better major components that can replace our current design. For instance, the Atmega2560 and nrf8001 chips can be replaced with a processor that includes a built in bluetooth chip, decreasing the costs. Furthermore, the software could be greatly optimized to

decrease processing requirements, cutting costs on the processor. Overall, the cost given our time constraint seems perfectly reasonable for an initial Spinosaurus prototype.

VII. PARTS

Description of the part	Unit cost (\$)	Quantity	Total Cost (\$)
Amazon Order 1			
Arduino Mega 2560	33.00	1	33.00
MPU-9250 Breakout Board	15.95	2	31.90
Back Brace	29.99	2	59.98
Mouser Order 1			
MPU-9250 Breakout Board	14.95	3	44.85
1572-1625-ND Rechargeable battery	18.86	4	75.44
1528-1199-ND Bluetooth Breakout Board	19.95	3	59.85
Vibration Motor Disc	1.20	8	9.60
Digikey Order 1			
FT232RL-REEL IC USB FS SERIAL UART	4.50	5	22.50
MPU-9250 Breakout Board	14.95	5	74.75
ATMega2560	12.21	5	61.05
NRF8001-R2Q32-T Bluetooth LE chip	4.26	5	21.30
AMCA31-2R450G-S1F-T3 RF Antenna	0.48	10	4.80
61729-1011BLF USB B connector	1.04	5	5.20
Digikey Order 2			
PTC RESET FUSE 15V 500MA	0.508	20	10.16
FERRITE BEAD 120 OHM	0.0976	25	2.44
FIXED IND 3.9NH 300MA 210 MOHM	0.023	20	0.46
LED GREEN DIFFUSED	0.254	25	6.35
LED RED DIFFUSED	0.202	25	5.05
RES SMD 1K OHM	0.0394	100	3.94
RES SMD 10K OHM	0.0394	100	3.94
RES SMD 330 OHM	0.0333	100	3.33
RES SMD 2K OHM	0.1173	100	11.73

RES SMD 1K OHM	0.0333	100	3.33
RES SMD 100 OHM	0.0394	100	3.94
RES 150 OHM	0.0273	100	2.73
RES SMD 4.7K OHM	0.3	30	9.00
RES SMD 10K OHM	0.0394	100	3.94
RES SMD 22K OHM	0.0376	100	3.76
IC REG LINEAR 3.3V	0.409	20	8.18
Mouser Order 2			
INDUCTOR 5.6nH	0.225	30	6.75
INDUCTOR 15nH	0.085	30	2.55
INDUCTOR 10uH	0.33	30	9.90
LED GREEN	0.396	30	11.88
16MHz 10ppm OSCILLATOR	0.275	30	8.25
16MHz 30ppm OSCILLATOR	0.325	30	9.75
6x6 PINS	0.248	30	7.44
N-CHANNEL 60V 200mA	0.107	100	10.70
Digikey Order 3			
CAP CER 0.1UF	0.036	120	4.32
CAP CER 22PF	0.0458	100	4.58
CAP CER 1UF	0.125	20	2.50
CAP CER 10UF	0.453	20	9.06
CAP CER 2200PF	0.144	20	2.88
CAP CER 0.033UF	0.098	30	2.94
CAP CER 1000PF	0.079	30	2.37
CAP CER 10UF	0.422	20	8.44
CAP CER 1.8PF	0.092	30	2.76
CAP CER 1.2PF	0.279	20	5.58
CAP CER 1.5PF	0.078	20	1.56
CAP CER 0603 1UF	0.125	20	2.50
CAP CER 10000PF	0.327	20	6.54
Digikey Order 4			

CAP CER 0.1UF	0.216	20	4.32
CAP CER 15PF	0.203	30	6.09
FIXED IND 10UH	0.242	20	4.84
FIXED IND 15NH	0.066	20	1.32
IC MCU 8BIT 256KB FLASH 100TFQP	12.21	3	36.63
Digikey Order 5			
MOSFET N-CH 20V 6A SOT-23	0.513	15	7.7
IC REG LIN POS ADJ 100MA 8SOIC	0.548	10	5.48
CAP CER 2.2UF 10V X7R 0805	0.206	20	4.12
RES 330K OHM 5% 1/8W 0805	0.0089	100	0.89
RES SMD 100 OHM 1% 1/8W 0805	0.0193	100	1.93
AIIPCB Order			
First PCB order	11.1108	10	111.108
Second PCB order	3.686	10	36.686
LCSC Order			
PMIC	0.2266	15	3.4
PCB connectors headers	0.0051	50	0.26
Battery Protection ICs	0.1451	15	2.18
DC-DC converters step up	0.1147	15	1.73
TOTAL			938.43

VIII. REFERENCES AND CITATIONS

i. Interface Codes

In order to show a clear view of the codes, we published the interface codes to the following git repository: <https://github.com/Sunfer-Z/spinosaurus.git>. `imu.h` and `imu.cpp` provide the IMU interface. `params.h` provides most parameters and global variables. `imu_specific.ino` is the Arduino sketch that incorporates the whole algorithm. The codes are also attached in Appendix V.

ii. PCB

See Appendix III.

iii. Acknowledgements

Thanks to the `Adafruit_nRF8001` library and the `MUP9250` library which we based our algorithm on [8] [9].

Thanks to Dr. Mark Brehob, Matthew Smith, Sharankumar Huggi, James Connolly for their instruction, guidance and wisdom in our darkest hours.

Thanks to Jielun Tan, Rosanne Crompton, Stephen Cain for their selfless help and inspiration on our project.

Thanks to Arduino for their mega reference design and GreatScott an online youtuber for his rechargeable circuit design. Links to their resources can be found at [3][4].

Appendix I: The Email of Despair

Spine Fails

Inbox x



Yan Zhang

Wed, Dec 5, 7:05 AM (6 days ago)



to Mark, me, Zack, Akshay, ebao ▾

Hi Prof. Brehob,

The more we tried, the more it breaks. We originally had two vests - one with simple common wires (a little ugly), and the other one with the ribbons and clamps (beautiful). After we first got the beautiful vest working with everything, we found that the signal change in the motors would cause the program to stop (totally froze). Then we tried our ugly vest and it worked perfectly with everything. Then we tried our beautiful vest without the vibration motors, it could work. Then I put the beautiful vest on, and the program couldn't run, so we thought there were connection issues. Then I took off the beautiful vest. We tried strengthen all the connections. Then the beautiful vest began generating random results. Even when we put it on the table without touching it, it just generated random results. We checked the wirings multiple times. Then we changed to the ugly vest without connecting the vibration motors, and the results were the same. The previous perfect vest are generating only random numbers. Then we re-uploaded to the ugly vest, and nothing changed.

We don't know what else we could do. We spent the day in 2334, and then spent the night in Mesh Lab, and the more we tried, the less we have working. Because we thought the wiring was so important, we tried our best to make the ribbons work. Sharan said you would like it because it looks so clean, but now we couldn't even have an ugly vest work. We are still in the Mesh Lab trying everything we can. We are gonna fail. We left Mesh Lab 6am the day before yesterday, 7am yesterday and today we don't leave, but we don't have any more time or energy. We are so tired. We are so desperate. We initially wanted an A in this class, so we worked so hard in the lab that other groups just said "you are the group that's in lab every day?" But now we could only get an ugly grade just like our not working vests. Please tell us what else we can do and what the results will be. We are sorry to be such failures.

Best,

Yan and Group

Appendix II: Manufacturing cost information

Cost for 25 units

	Quantity	Unit Price	Total
☰ PCB	25	\$9.27	\$231.69
⚙️ Parts	25	\$25.11	\$627.79
🔧 Assembly	25	\$32.23	\$805.84
Total	25	\$66.61	\$1,665.32

Cost for 50 units

	Quantity	Unit Price	Total
☰ PCB	50	\$5.54	\$276.87
⚙️ Parts	50	\$22.72	\$1,136.22
🔧 Assembly	50	\$21.43	\$1,071.30
Total	50	\$49.69	\$2,484.40

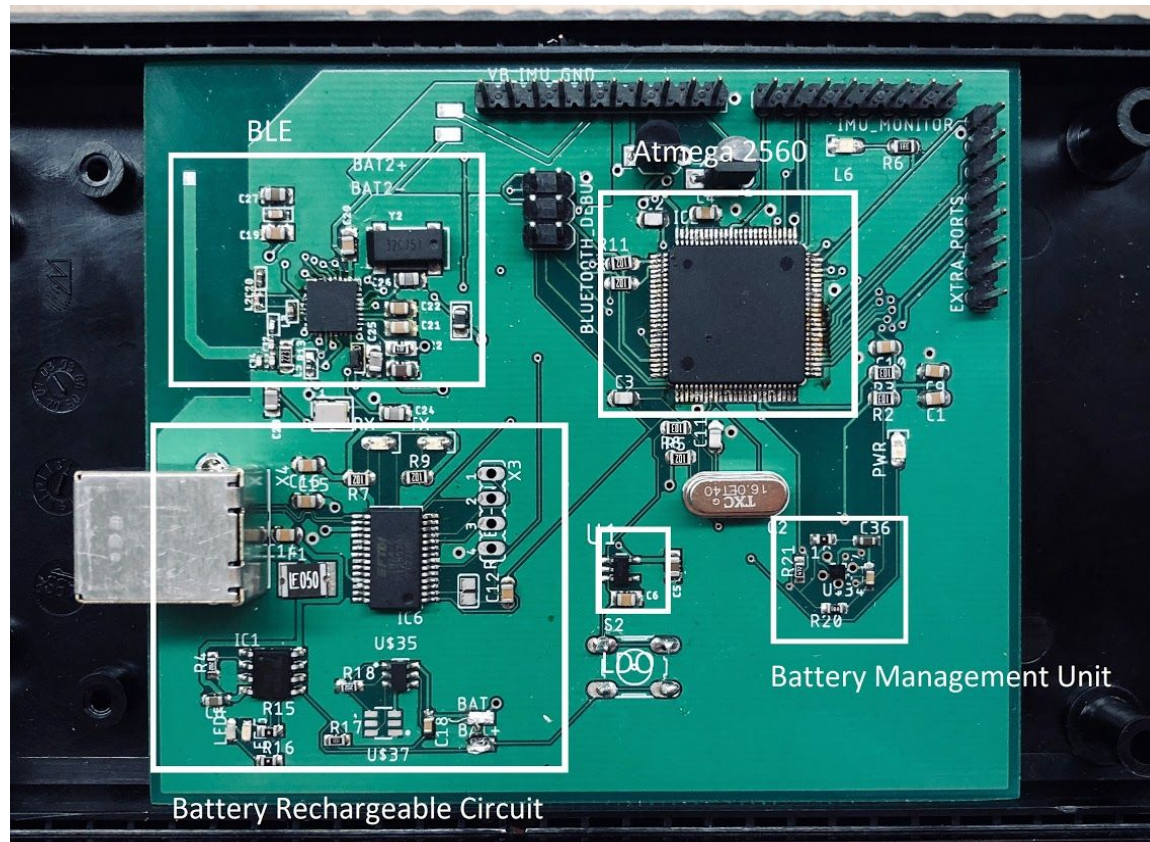
Cost for 100 units

	Quantity	Unit Price	Total
☰ PCB	100	\$3.59	\$359.04
⚙️ Parts	100	\$20.23	\$2,023.24
🔧 Assembly	100	\$15.46	\$1,546.28
Total	100	\$39.29	\$3,928.56

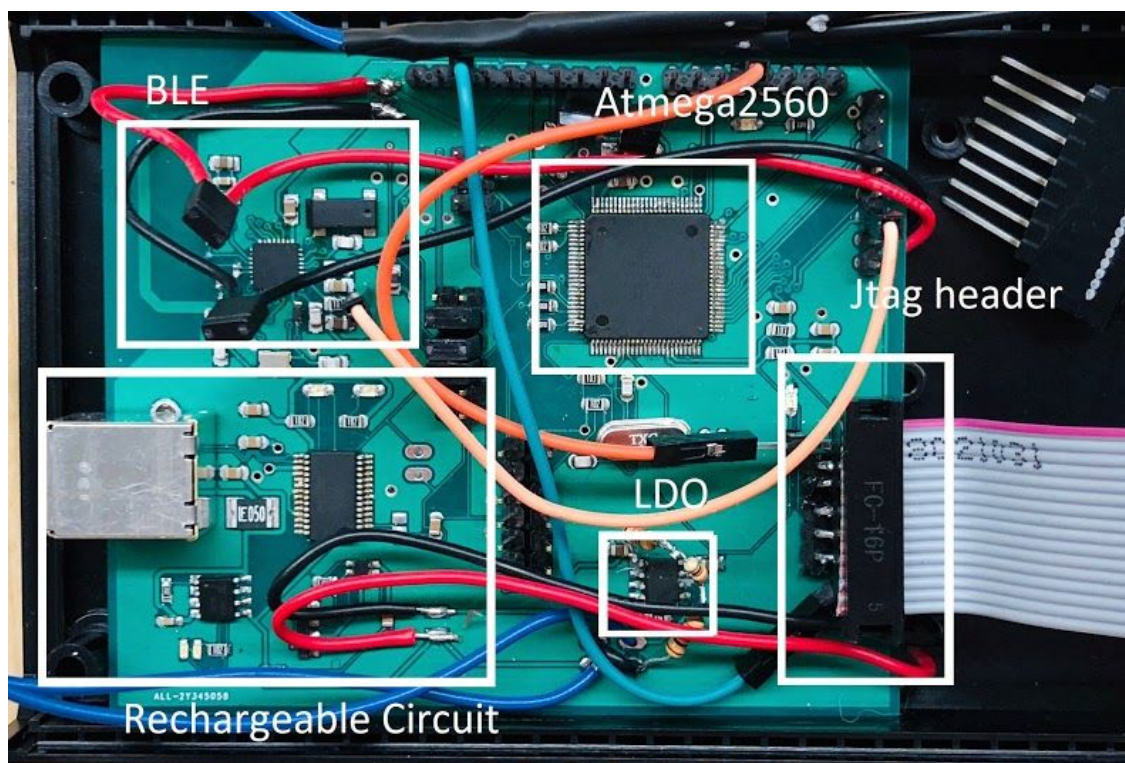
Cost for 200 units

	Quantity	Unit Price	Total
☰ PCB	200	\$2.56	\$512.64
⚙️ Parts	200	\$19.85	\$3,969.74
🔧 Assembly	200	\$12.10	\$2,420.63
Total	200	\$34.52	\$6,903.02

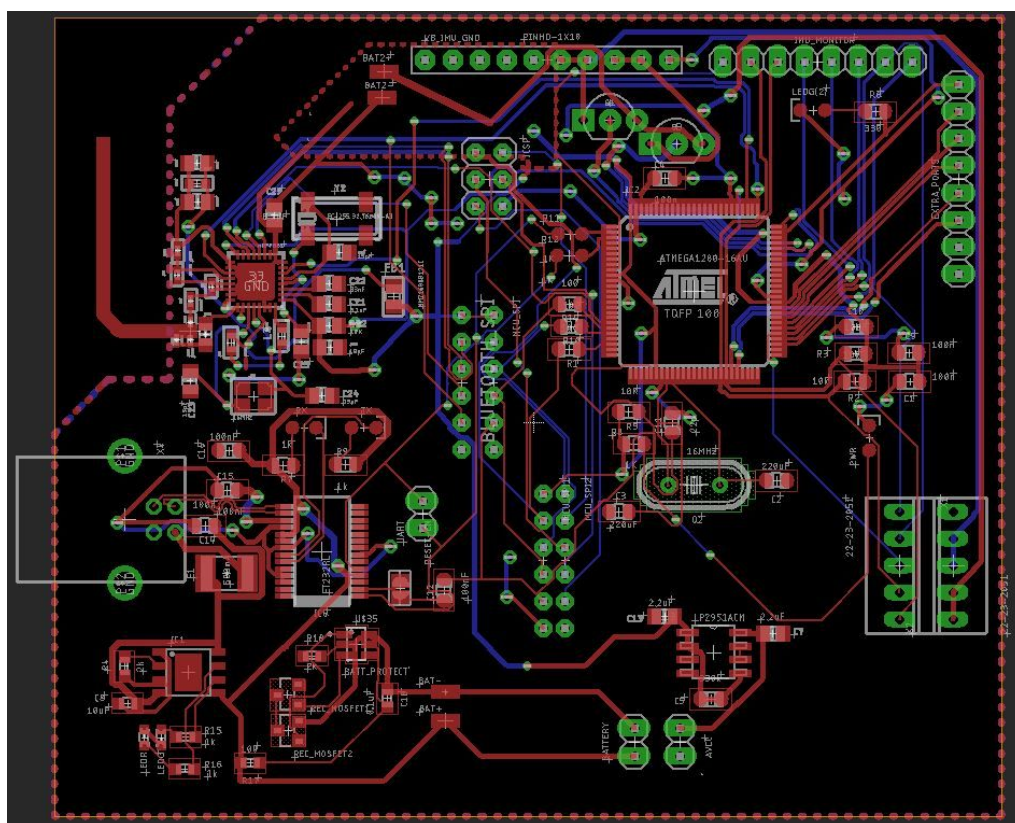
Appendix III: PCB



Spinosaurus version 1



Spinosaurus version 2



Spinosaurus version 2 board

Appendix IV: Bibliography

- [1] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2791257/>
- [2] <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3606891/>
- [3] <https://www.uhs.umich.edu/profile/rosanne-crompton>
- [4] <https://store.arduino.cc/usa/arduino-mega-2560-rev3>
- [5] https://easyeda.com/GreatScott/LiPoChargeProtectBoost_copy-3d9f4c3ddc7347d7861776340b9b90b7
- [6] <https://developer.android.com/guide/topics/connectivity/bluetooth-le>
- [7] <https://thejeshgn.com/2016/12/11/uart-gatt-server-peripheral-on-android/>
- [8] https://github.com/adafruit/Adafruit_nRF8001.git
- [9] <https://github.com/bolderflight/MPU9250.git>

Appendix V: Interface Codes

The following pages show the header files of `params.h`, Bluetooth interface and `imu.h`.

```

1  #ifndef PARAMS_H
2  #define PARAMS_H
3
4  #define RUNNING_AVG_SIZE 20
5
6  // Bluetooth pins
7  #define REQ_PIN 30    // SPI select pin
8  #define RDY_PIN 18    // Sends interrupt to MCU when data is ready (Needs to be an
  interrupt pin)
9  #define RST_PIN 32    // Resets the bluetooth board (Used during startup)
10
11 // SpineAligner on/off
12 #define ON 1
13 #define OFF 0
14
15 // motor pins
16 #define UPPER_MOTOR 28
17 #define LOWER_MOTOR 29
18
19 // number of sensors on vest
20 #define NUM_IMUS 3
21
22 // state of spine
23 #define VERY_GOOD 0
24 #define GOOD 1
25 #define BAD 2
26 #define VERY_BAD 3
27
28 // form of spine
29 #define STRAIGHT 0
30 #define LOWER_ARCH -1
31 #define BOTH_ARCH -2
32 #define UPPER_HUNCH 1
33 #define LOWER_HUNCH 3
34 #define BOTH_HUNCH 2
35
36 // z angle thresholds
37 #define DEG20 -1
38 #define DEG45 -3.5
39 #define DEG90 -9
40
41 // loop threshold value
42 #define T_VAL 25
43
44 // change amounts for 20 degree zone
45 #define TMP_LA20 -1.5
46 #define TMP_UH20 1.2
47 #define TMP_BH20 1.2
48
49 // sensitivity
50 #define INCR_DECR 0.2
51 #define MAX_CHANGE 3
52
53 #endif

```

```

1  #ifndef BLUETOOTH_H
2  #define BLUETOOTH_H
3
4  #include <SPI.h>
5  #include "Adafruit_BLE_UART.h"
6
7  /*******
8  //
9  //*****
10
11 // FSAT interface to utilize Adafruit_BLE_UART.h functions
12 // One class instance per device
13 class NRF{
14     public:
15         // Constructs Adafruit_BLE_UART object with the given pin numbers
16         NRF(int req, int rdy, int rst);
17
18         // Starts BLE configuration
19         void nrf_begin();
20
21         // Updates status of chip and updates private members last_status and current_status
22         bool update_status();
23
24         // Returns the private member last status of the device
25         bool get_last_status();
26
27         // Returns the private member current status of the device
28         bool get_current_status();
29
30         // Writes a String to the serial line
31         // Max 20 characters per line. Longer strings will be broken into multiple lines
32         void write_string();
33
34         // Returns how many bytes are available to be read
35         int data_available();
36
37     private:
38         // Adafruit_BLE_UART, used to interface with the device
39         Adafruit_BLE_UART BTLEserial;
40
41         // Statuses:
42         // ACI_EVT_DEVICE_STARTED - Device begins advertising
43         // ACI_EVT_CONNECTED - Device is connected
44         // ACI_EVT_DISCONNECTED - Device is disconnected
45
46         // The last status of the device
47         aci_evt_opcode_t last_status;
48
49         // The current status of the device
50         aci_evt_opcode_t current_status;
51 };
52
53 #endif

```

```

1  #ifndef IMU_H
2  #define IMU_H
3
4  #include <Arduino.h>
5  #include <Wire.h>
6  #include "MPU9250.h"
7  #include "params.h"
8
9  struct FSM {
10     int imu_bottom, imu_top;
11     int state, form;
12     float BA[3], LA[3], UH[3], BH[3];
13     int THRE_GOOD, THRE_BAD;
14     int count_g, count_b;
15 };
16
17 struct AxisVals {
18     float readings[RUNNING_AVG_SIZE] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
19     float total = 0;
20     float avg = 0;
21     float offset[3] = {0.0, 0.0, 0.0};
22 };
23
24 class IMU_T {
25 public:
26     int AD0_pin;    // LSB of slave address of the IMU
27     int imu_id;     // unique id of each IMU
28     int accel_index;
29     AxisVals accel_xyz[3];
30
31     /* ----- Setup Functions ----- */
32
33     /* Constructor. Initializes data collection for a particular IMU.
34      * Tells the IMU what data it should be sampling and placing in its buffer.
35      */
36     IMU_T(int imu_id_arg = 0);
37
38     void init_pin(int pin_num);
39
40     /* Sets the full scale of the accelerometer values:
41      * 0 = ±2g
42      * 1 = ±4g
43      * 2 = ±8g
44      * 3 = ±16g
45      */
46     void set_accel_scale(MPU9250 &imu_device, int val);
47
48     /* Sets the full scale of gyroscope values:
49      * 0 = ±250dps
50      * 1 = ±500dps
51      * 2 = ±1000dps
52      * 3 = ±2000dps
53      */
54     void set_gyro_scale(MPU9250 &imu_device, int val);
55
56     /* ----- Data Functions ----- */
57
58     /* Gets accelerometer data axis x from I2C line. */
59     float get_accel_data_x(MPU9250 &imu_device);
60
61     /* Gets accelerometer data axis y from I2C line. */
62     float get_accel_data_y(MPU9250 &imu_device);
63
64     /* Gets accelerometer data axis z from I2C line. */
65     float get_accel_data_z(MPU9250 &imu_device);
66
67     /* ----- Calibration Functions ----- */
68
69     /* Allows initial calibration with threshold for calibration allowance specified in

```

```
70     * percentage or degrees of freedom.  
71     */  
72     void calibrate_IMU(MPU9250 &imu_device, int angle);  
73  
74     void update_accel_averages(MPU9250 &imu_device);  
75 };  
76  
77 #endif
```