ChairGuru - A Self-Configurable Chair System

Alec Damien
Dept. of Electrical and Computer Engineering
University of Central Florida
Orlando, United States

Yvan Pierre Jr.
Dept. of Electrical and Computer Engineering
University of Central Florida
Orlando, United States

Andrew Fugate
Dept. of Electrical and Computer Engineering
University of Central Florida
Orlando, United States

Erik Barcelo
Dept. of Electrical and Computer Engineering
University of Central Florida
Orlando, United States

***Abstract*—ChairGuru aims to create a modular solution for classroom chairs, restaurant seating, and other similar seating platforms that aids in the movement of chairs positions in a defined area. ChairGuru contains Ultra-wide Band (UWB) technology that communicates with stationary anchor beacons to reconfigure a chair's position. The chair's movement is achieved using finely tuned stepper motor drivers and unique omnidirectional wheels and movements. As a final product scope, a classroom quantity of ChairGuru equipped chairs can all communicate with each other and intelligently reconfigure themselves into varying layouts at the touch of a button, without issue. Some examples of configurations can include a group of four configuration, a standard test-taking layout with all chairs facing forward, or all chairs facing inwards in a circular fashion.**

***Index Terms*—Motors, Time, UWB, ESP32, Location, Anchor, TB6600, DW1000, User, Server, Interface, Data**

I. Introduction

Classroom chairs and attached chair-desk combinations remain in high demand for public school systems and other organizations that require these platforms in learning environments. When considering the typical American grade school education, most Americans find themselves sitting in these chairs for at least thirteen years of their life from Kindergarten to senior year of high school. In higher education, the same models of chairs and desks remain in college and university classrooms where those same students are again exposed to them.

Classroom chairs and desk are frequently moved around the classroom; typically due to movement for cleaning of floor surfaces or to reconfigure their position from traditional classroom orientation to small groups to work with colleagues, or to large circles for entire class discussions. With respect to the educator that has to reconfigure a whole classroom, or the lone cleaning staff that needs to vacuum a carpet, ChairGuru aims to remove the burden of moving many chairs by hand by implementing a simpler action managed by fine-tuned hardware and a user-friendly interface.

II. Hardware Components

A. Microcontroller (ESP32)

The ESP32-WROOM-32D-N8 microcontroller unit (MCU) serves as the brain of the ChairGuru's custom printed circuit board (PCB). The MCU integrated a 40 MHz crystal, 8 MB SPI flash, Wi-Fi up to 802.11n protocol, and bluetooth v4.2 as part of its feature set. The ESP32 communicates with our Ultrawide Band chip (DW1000-derived BU01) using the Serial Peripheral Interface (SPI) over four user-selectable GPIOs to include chip select, SPI clock, Master In Slave Out (MISO), and Master Out Slave In (MOSI) to exchange high-speed data exchanges and high-rapid response times.

In addition to SPI Configurations, the ESP32's Bluetooth and Wi-Fi Capabilities facilitate data transfer to external devices and cloud servers, enabling live tracking and control over a network. This will allow for receiving commands and then sending location data to the central HTTP server. With multiple of these being parts of the board and integration of all SPI lanes, the ESP32 chip contains 21 more GPIOs to be reconfigured for stepper motor drivers and HC-SR04 ultrasonic sensors. Two 19x1 solderable headers were included for the GPIOs, with ground (GND) pins strategically allocated for attaching the peripherals. With four total stepper motor drivers ideally using two GPIOs per driver, the ESP32 maintains a sufficient amount of pin-outs for operation. Our ESP32 integrates satisfactorily on the board, utilizing the 3.3V rail provided by a LM1117-3.3 voltage regulator powering other components on the board.

This setup not only ensures reliability and longevity in operation but also allows the ESP32 to handle various tasks simultaneously without interference or power dips, maximizing efficiency in both UWB communication and peripheral management. The design thus combines high flexibility with stable connectivity and robust control, allowing ChairGuru to scale and adapt to a variety of applications in real-time positioning and automated movement.

B. UWB Microcontroller DW1000 (BU01)

The basis of communication between multiple ChairGurus and stationary beacons relies on Qorvo's Decawave DW1000

MCU. The DW1000 is a low-power and low-cost integrated circuit (IC) chip compliant with standard IEEE 802.15.4-2011. that provides for precision location tracking. The chip is primarily used in a two-way ranging location system with a precision of 10 cm, with data transfer rates of up to 6.8 Mbps and an impressive precision of 10cm. Other desirable features includes relatively low power usage, feasible PCB installation, and usage of the 3.3V rail existing on the PCB, which help it's performance on an custom PCB ideal.

The DW1000 is further built upon using a modified IC called the BU01 developed by Ai-Thinker. The BU01 has multiple enhancements that make it suited for it to be ran in difficult enviroments, such as the antenna design, radio-frequency (RF) circuit, power management, and a clock circuit. The optimized antenna allows for an improved range, stability and precision allowing for a reliable UWB signal transmissions. The impedance-matched RF circuit is given in the figure below.

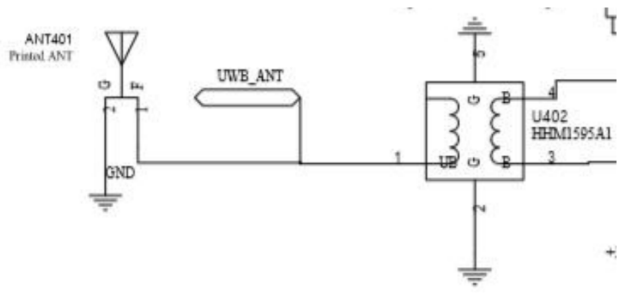


Fig. 1: Impedance-Matched Circuit for UWB

C. HC-SR04 Ultrasonic Sensor

The HC-SR04 is a popular ultrasonic sensor module designed for distance measurement, capable of detecting objects within a range of 2 cm to 400 cm with an accuracy of approximately 3 mm[4]. It operates by emitting a high-frequency ultrasonic sound wave (40 kHz) from its transmitter, which then reflects off any object within its detection range. The sensor's receiver detects the reflected sound wave, and by calculating the time it takes for the echo to return, the HC-SR04 accurately determines the distance to the object. This makes it highly useful in applications requiring proximity sensing or collision avoidance, particularly in environments where visual sensors may struggle to differentiate between obstacles.

The HC-SR04 has straightforward hardware requirements, typically interfacing with a microcontroller via two pins: a trigger pin to start the ultrasonic pulse and an echo pin to receive the pulse back. This simplicity makes it a versatile and cost-effective option for various projects, from robotics and drones to obstacle detection in automated systems. By using sound waves rather than light, it is well-suited to detect objects in environments where lighting conditions or object transparency might affect the performance of optical sensors, such as in low light, fog, or areas with reflective surfaces.

In the ChairGuru setup, where chassis tracking and navigation are primarily reliant on UWB and location-based data, there's still a need to account for nearby objects that may not be trackable through the main positioning system. For instance, stationary obstacles or transient objects like people or pets in close proximity might not always be mapped in real-time by UWB triangulation alone. This is where the HC-SR04 ultrasonic sensors add an essential layer of safety. Positioned on the chassis, these sensors act as an immediate obstacle detection system that can detect and measure distances to nearby objects in real-time.

Using ultrasonic sensors like the HC-SR04 allows the ChairGuru to continuously scan its immediate surroundings for untracked objects. If an obstacle is detected within a predefined "safety zone," the control system can halt the movement of the chassis to avoid collisions. This layer of protection is especially valuable in dynamic environments where tracked and untracked objects may coexist, and unexpected obstacles can suddenly appear. By combining the precision of UWB positioning with ultrasonic-based proximity detection, the ChairGuru achieves a comprehensive awareness of its surroundings, ensuring smooth and safe navigation even in challenging or crowded spaces. This approach maximizes safety for both the system and any nearby individuals, providing an added level of reliability to the overall positioning and movement framework.

D. Battery Configuration and Regulation

During the research phase of the project, a 12V, 9Ah sealed AGM lead-acid battery was elected to power both the PCB and motor drivers. However, two configuration changes were necessary to refine operation of the system. The first included the increase from a 12V to a 24V system for powering four motor drivers, as one 12V battery could not deliver enough current for maximum output during stress testing. Thus, prototyping included an additional sealed lead-acid battery in series with the first battery for one chassis.

In addition to maintaining 24V, our battery technology has also been modified. ChairGuru now employs a $LiFePo_4$ battery. A reduction in weight of 12 pounds for two sealed lead-acid batteries down to 2.09 pounds for one 25.6V battery significantly helped chassis rigidity. As a minor drawback, the new battery technology delivers 4Ah compared to the lead-acid battery's 9Ah, but is predicted to be sufficient for a typical use case.

The second modification for powering the chassis included isolated motor driver circuits and PCB circuits. During prototyping, CP2102N USB-to-UART ICs and ESP32 ICs were electrically damaged due to breadboard mishandling, with voltages exceeding their absolute maximum ratings. Thus, the new 25.6V battery powers four motor drivers as an isolated loop. The PCB, containing the ESP32 and DW1000 MCUs contains a four AA battery arrangement producing a peak 6V. An LM1117-3.3 steps down this voltage to a usable 3.3V for both MCUs. Additionally, 6V does not meet the absolute maximum ratings that can electrically damage the MCUs should the regulator malfunction.

E. TB6600 Motor Drivers

While researching motor drivers we could use for the project we came across the commercial TB6600 Stepper Driver. This stepper driver supports both speed and direction control while also being adaptable to several different micro step sizes and current outputs via a series of DIP switches. Initially, we chose these drivers for their "plug and play" nature and ease of setup. All of the terminals can be easily accessed and wired into our pcb. This also made the design of our code relatively simple since it relied on sending a signal to the pulse pin within a loop and toggling the direction pin.

During initial testing with the TB6600 and a NEMA 17 stepper motor, we ran into an issue where we could not get consistent rotation out of the motor with stuttering. We tried everything from changing out the driver, wires, dev board, etc. We were not able to get these drivers working for us which drove us in another direction. The DRV8825 was the next driver we tried to test with. This time we were able to get the motors turning without stuttering and were even able to breadboard with multiple motors/drivers on the same dev board. This is where a big issue came up with the DRV8825. On the DRV current output is modified by turning a set screw. This meant that it was liable to be altered unknowingly after being set. Another current issue we had was that when multiple DRV8825s and motors were connected on the same circuit, the current setting had to be altered on each driver to make sure that each motor received adequate current.

Due to the lack of reliability on the DRV8825, we circled back to the TB6600 and made another attempt at development with it. We soon figured out that the colored wires of the NEMA 17 motors were not standardized and thus we had them installed incorrectly on the TB6600. Once this was rectified the motors functioned flawlessly. This allowed us to take full advantage of all of the features present on the TB6600. We currently have it set to a micro step of 1, giving us 200 steps per revolution. One of the best features that we are using on the TB6600 is the fact that it is optically isolated. This means we can run our 24V battery exclusively on the drivers and have our pcb powered by a smaller and separate 5V source. This mitigates the risk of any high voltage issues on our custom pcb since the 24V battery never comes into contact with it.

While some might consider the TB6600 to have too large of a footprint, in our application it presents no issue. The reliability and settings that it offers is much more important than saving weight or having a smaller overall footprint for our wiring harness.

F. Nema 17 Stepper Motor

For our application we decided to use a stepper motor over a traditional DC brushless motor. Stepper motors operate by providing current to a set of coils in a specific sequence. This creates a magnetic field that attracts the rotor, allowing it to rotate. Current is provided in pulses and each pulse will advance the motor one step. In the case of the NEMA 17 this is 1.8 degrees. We can alter the sequence in which the pulses are sent and thus can control direction and speed. Given the nature of a stepper motor, they are inherently very accurate and often used for tasks such as 3D printing. The accuracy that the NEMA 17 series is capable of lends itself to our engineering requirements. When the chairs are being rearranged by the user, a high degree of accuracy is necessary to mirror a given layout.

Once we decided on using a stepper motor, we then had to choose one that would fulfill both our size specifications as well as power output specifications. The NEMA 17 series was a clear winner. Due to the way our chassis was designed and the wheels we were using, the NEMA 23 size of motors would have been too large. The first set of motors we tested output a holding torque of 0.42Nm at 1.5A. With the preliminary idea we had for our chassis and the weight of our components these motors would have been adequate for our needs. However, once we built the first prototype chassis with all of the necessary parts for the final product, it was clear this model of NEMA 17 was slightly underpowered. During testing we were able to complete all necessary movements but it was clear that the motors were running at the razor's edge of reliability. It was also noted that on higher drag surfaces like carpet, the motors had a substantial struggle moving our chassis laterally.

The solution to this problem was to invest in more powerful motors. In order to keep our chassis design the same, we opted to get higher torque output NEMA 17 motors. These new motors output 0.59Nm of torque at 2A. These output numbers should easily exceed our power requirements for the final iteration of our chair chassis.

The driving factor for us choosing to use stepper motors was their inherent accuracy+. One of the litmus tests we wanted to be able to pass with our chairs was the ability to reposition our chairs within 1-5 inches of their starting point. Being able to move our motors by the step allows us to make micro adjustments as needed to maintain positional accuracy.

The only downside we have encountered to using the NEMA 17 series of motors is that they have a lack of options when it comes to mounting hardware to attach wheels. If given more time and resources, we would have liked to custom design and fabricate a better mounting solution than what is available on the market.

G. Chassis

Our chassis was primarily designed around the size of the chair we wanted to use for our demo. Once we had the chair dimensions we were able to go about designing the chassis with CAD. The original idea was to have a "sandwich" construction with all of our components in the middle. This allowed them to be protected while also leaving a clean platform for our chair to be mounted. The panels of the "sandwich" were constructed out of 0.25" birch plywood. Birch plywood was chosen for its rigidity and weight. Birch plywood is also very easy to work with using hand tools as well as a laser cutter. The two panels are then joined together using ½" PVC pipe pieces as a spacer with carriage bolts to hold them together. These spacers were spread apart to equally distribute the weight of the chair and battery so that the motors would not be over stressed. Figure 2 below shows our CAD model for the initial chassis design. In this case it is mostly a proof of concept as we did not have our chair specifications at the time of designing it.

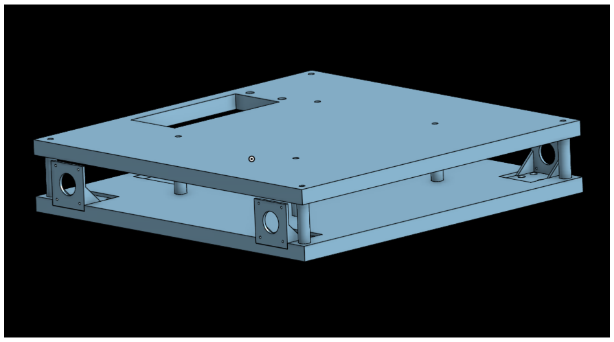


Fig. 2: CAD Model of Initial ChairGuru Chassis

After our initial prototype was constructed, we noticed that when the batteries and chair were mounted, the plywood panels were bowing inwards slightly. This caused the chassis to drag on the ground and not give enough clearance for the wheels to spin freely. This also added a camber to the motors and wheels which meant our chassis would not drive in a straight line. At this point we knew that the theory behind the design was successful but needed a better execution. To solve our stability problem we elected to add aluminum ribs to the plywood in order to prevent the wood from bending under load. For this we used 1/16 thickness angled aluminum that was mounted on the front and back of each panel. While aesthetically pleasing, the angled aluminum stiffened the areas where the chair directly imparts force onto the chassis. After installing the aluminum we tested the chassis under load and saw a significant improvement. This was still using our original motors which are underpowered, thus once we assemble the final product with the higher torque output motors it should run flawlessly.

Testing our chassis design was also crucial to identifying some shortcomings of the components we selected. Foremost was the type of batteries we were using. The two 12V/9AH lead-acid batteries weighed in at 12 lbs. This caused a significant strain on our motors which, in hindsight, was unnecessary given the much lighter weight options for a 24V battery. It also illustrated how the motors we originally selected were on the edge of being reliable. Even though our design could work with the original motors, it made more sense to go with a higher torque option that would comfortably perform at the levels we require.

The major aspect that we neglected when designing our chassis was the actual weight of the components we were selecting. It became apparent very quickly that managing weight is a huge part of creating a reliably functioning motorized device. It would have been ideal to know these things before we started committing to a design but, as has been relevant with a lot of the subsystems, we didn't know until we started working with them hands on.

III. Software Components

Software is the key to ChairGuru and its' intelligent functionality, transforming it from a simple motorized device resembling a large remote control car, to an autonomous system that assists us. As we enable ChairGuru and its' features at the hardware level by providing the necessary components to achieve our goals, the software we implement here makes these components operate how we intend them to operate for our user's experience and overarching system goals. The overall software system is stacked at various levels to support a custom communication network that has no need for internet access, real-time positioning and location tracking, and dynamic movement control that makes full use of our hardware. As we begin discussing each component of our software system, we will see why some of our software decisions are best suited for these goals.

A. *Local Network*

Establishing back and forth communication and data exchange is the backbone of all of ChairGuru's technology; without it, we have no way of controlling movements, interpreting data, and interacting with the device(s) other than through supplying power. Knowing this, we established a pseudo local area network (LAN) running on one of the three anchor points that remain a ”permanent” structure put into place with ChairGuru's installation. This LAN allows our sever (discussed later in the User Interface section) to communicate between all the anchors and each chair in the present area. This allows us to assign names and profiles to each chair and device present in a room and allow for direct and intentional ”targeting” of each device with instructions. By assigning these names via IP address allotting, we are then able to have our server read information to and from our chairs. This means we can offload the heavy processing power to our central server (room and setup specific) and return commands at a faster rate with higher accuracy as the server sees the entire space and spatial relation to the other chairs in the vicinity.

With these benefits being the main driver of this decision, there also exists enterprise reasoning to utilize this LAN setup. As we intend for the ChairGuru to be implemented into public places like school, restaurants, and other places, the ability to control network security and/or integrate the LAN into pre-existing networking established by a facilities IT department. This allows for even tighter integration into the user's domain and current technology setup; or allows for a network to be configured in an environment lacking pre-existing network connectivity. This is ran off of an ESP32 acting in access point operation with specific IP allotting instructions flashed onto it.

Finally, the last benefit to the LAN is that it benefits both the user and ChairGuru themselves as it keeps latency and bandwidth concerns to a minimum. As the ChairGuru is ”self-contained”, there is no need to use bandwidth on already questionably stable networks, for example, or worry that communication could be slowed between the ChairGuru server and on-board devices.

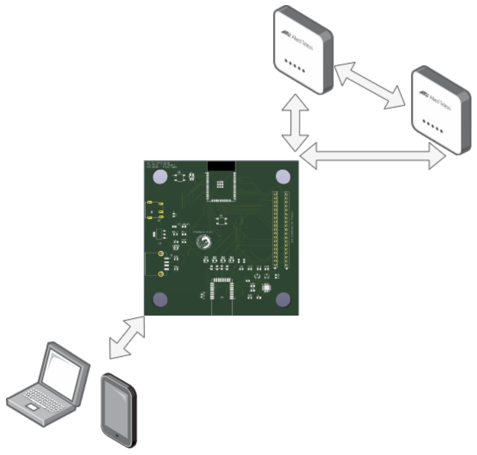


Fig. 3: ChairGuru Network Design

B. Triangulation

In order to calculate and produce location data, we utilized a Makerfabs chip that builds on an ultra-wide-band (UWB) module produced by Qorvo, the DW1000. This allows us to utilize UWB with a specially designed antenna to get the most range and accuracy out of the UWB protocol. This setup utilizes a minimum of two DW1000 anchor points and at least one ChairGuru PCB to determine and transmit location data. In our setup, we are using two main anchor points, a number of ChairGurus as determined by the end-user (in our demonstrations case, two), and a DW1000 module operating as a third anchor on the LAN PCB board. This LAN PCB allows for the positioning to rely on an extra relation point from the DW1000 and communicate information directly to LAN ESP32 via SPI as a redundancy.

UWB transmits information across a wide bandwidth of 500 MHz or greater. UWB generates radio energy at specific time intervals and occupies a large bandwidth, enabling time modulation. The time modulation can be modulated on UWB signals (pulses) by encoding the polarity of the pulse, the amplitude, or by using orthogonal pulses. UWB pulses can be sent at relatively low pulse rates for supporting time or position modulation. High-Grade Pulse-UWB systems have been capable of channel pulse rates in excess of 1.3 billion pulses per second. This set of protocols helps enable our triangulation and positioning system.

With each ChairGuru chassis treated as a mobile tag in the triangulation setup, each unit continuously sends UWB location data to the main server. This data is transmitted through an HTTP server to collect and analyze each ChairGuru's coordinates in real time. By calculating the relative distance of each mobile tag from the primary anchor points, we are able to pinpoint the exact position of each chair within a given space. The DW1000 modules used as anchors on the LAN PCB establish a stable communication framework, allowing each ChairGuru to determine its location with respect to other tagged units.

To ensure that each ChairGuru remains in a defined, non-overlapping area, our system uses real-time distance checks between each chair's coordinates. This safeguard prevents any two ChairGurus from occupying the same physical space, which could result in conflicting positional data or interference. The server processes these location updates, cross-references them with pre-set distance thresholds, and flags any potential overlap zones. Through this layered configuration, we achieve reliable multi-point triangulation, allowing each ChairGuru to operate independently while maintaining spatial awareness within a shared environment.

To maximize accuracy and minimize the loss of data, the ChairGuru setup incorporates redundancy through the additional DW1000 anchor on the LAN PCB, which supplements the data from the primary anchors. This redundancy provides an extra layer of reliability, ensuring that each position reading is cross-verified and refined, even in environments with potential signal interference or obstructions. Each ChairGuru tag leverages the UWB protocol's high-precision timing to send pulse signals back to the anchors, which then compute the tag's exact distance and position through multilateration. This precise time-of-flight data, when triangulated from multiple reference points, enables the server to calculate each chair's orientation and location within a few centimeters of accuracy. This allows the system to scale seamlessly in larger spaces with additional ChairGurus and anchor points, while maintaining low latency and high positioning accuracy.

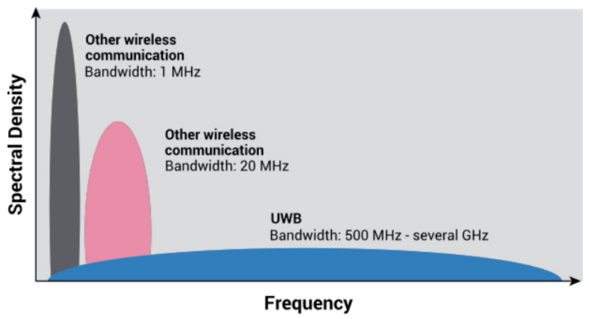


Fig. 4: Spectral Density versus Frequency

The UWB radio system is used to determine the "time of flight" of the transmission at various frequencies, where the speed of flight is that of light.

$$Distance = T_{prop} * Light \tag{1}$$

For a basic UWB ranging system, we can consider single-sided or double-sided two-way ranging as shown in the image below.

A message (TX) is sent from device A and the time is recorded. When device B receives the message, the message time is recorded once more. After a delay, T_{reply}, Device B sends a message (TX) to device A and the time is recorded another iteration. From this flight loop, we derive:

$$T_{prop} = (1/2)(T_{round} - T_{reply}) \tag{2}$$

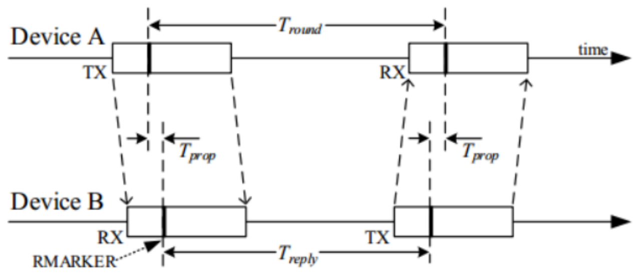


Fig. 5: Visualization of Ranging

For reliability, a four-message mode is employed using the UWB setup as shown in the figure below.

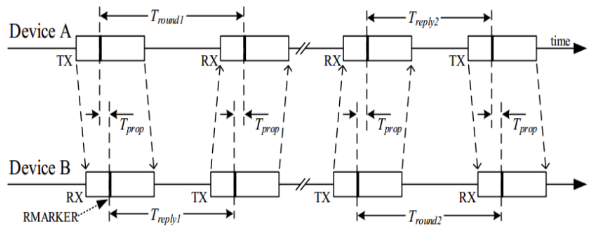


Fig. 6: Visualization of Four-Message Ranging

With the implementation of four-message mode ranging, the following time propagation equation is derived below.

$$T_{prop} = \frac{(T_{round1} * T_{round2} - T_{reply} * T_{reply2})}{(T_{round1} + T_{round2} + T_{reply} + T_{reply2})} \tag{3}$$

With a baseline orientation of two development kits as stationary anchors, and our custom PCB (the ChairGuru platform) as a tag, the three devices can communicate using the propagation algorithm given earlier. To determine the position of the tag, we use the law of cosines to get cosine of A and the sine of A using the following formulas:

$$cos\alpha = \frac{b^2 + c^2 - a^2}{2 * b * c} \tag{4}$$

$$sin^2\theta + cos^2\theta = 1 \tag{5}$$

From these equations, we can determine the angle and distance of the tag as shown in the triangulation visuals above and return this data as a useful piece of information to our server for position processing.

C. Motor Control

In order to successfully control the Nema 17 stepper motors, another part of our software systems had to be created for this sub-system. Not only do we need special movement classes to take in movement and how much, but we also must ensure the right movement is being outputted and that the correct GPIO pins are being activated for each wheel.

Since we are using omni-directional wheels, we have extra movement cases; the traditional four we are accustomed to

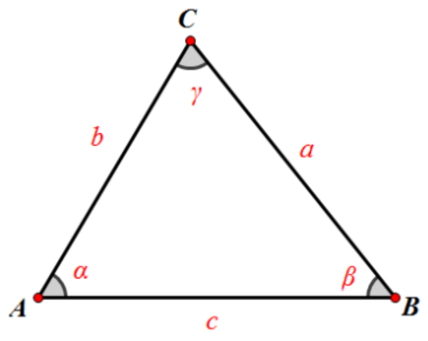


Fig. 7: Triangulation

being forwards, backwards, left, and right, but also having access to lateral left and right movement, centered rotations, and four more diagonal movements. Once configured, these can all be put into custom functions to be called by our devices as information is sent to them from the server. By being able to pass in a directional variable (where ”fwd” might mean ”forward” and ”brd” might mean ”backward right-diagonal”, etc.) along with a inches amount, we can offer precise control of our ChairGuru’s that is in a constant feedback loop with the positioning system to achieve the results.

As for how we are controlling the motors, it’s simply output via the GPIO pins on our custom ChairGuru PCB that is controlling the motors enable, direction, and pulse pins. These GPIO’s are connected to our TB6600 motor driver and deliver the electrical pulses to make the motors turn. This section was discussed above in the Hardware Components section, but is again a relevant topic for how we send signals for movement and how they are ultimately executed, from server to physical movement.

D. User Interface & Server

Despite all of these software subsystems working to produce usable and reliable data, their total sum is useless to us without some way to tie them all together. With this, we showcase out user interface and the server that runs everything as the overall brains of this reconfiguration system. Taking in data from the triangulation subsystem, managing various devices connected to the LAN, producing results from this data, conveying said data to the user on an intuitive user interface, and then transmitting back onto the ChairGuru movement field, was our big software undertaking.

To implement this all, we first began by breaking down what we wanted our software flow to look like. The biggest layout for this process was what we wanted our software class diagram to look like and what info/data needed to be where within the system. As seen in figure 8, the class diagram illustrates the overarching software setup for the ChairGuru system. It illustrates how the chair exists, what kind of information the antennas will communicate to discern each chair, where its’ spatial existence is within a map and its’ boundary, as well as what type of motor controls we will need to call for each chair.

Following our class diagram, it was important to understand how we wanted the system to operate and how we could handle the expected results. For this, figure 9 illustrates our ideal use case flowchart.

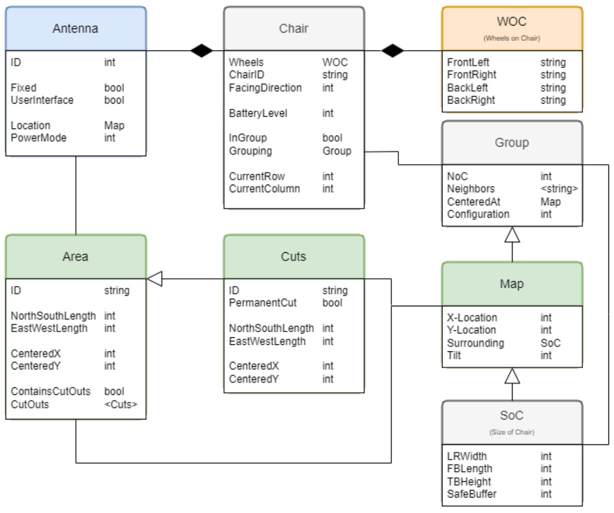
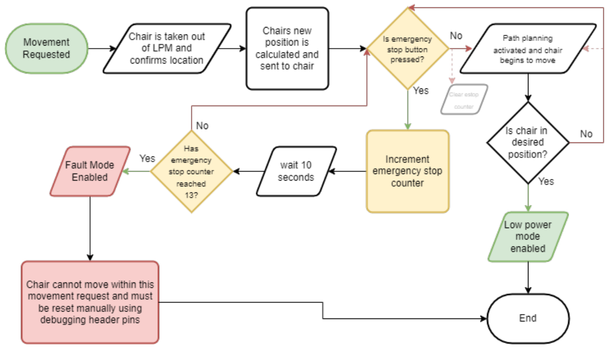


Fig. 8: Software Class Diagram

The flowchart shows the higher level process of how our class diagram should work together. By requesting a movement from the user interface, the user has just initiated the antennas for all ChairGuru's and they have began reporting their location data from the triagulation subsystem. From here the data is sent to the server via our LAN network subsystem and the server begins data processing. As soon as movement is allowed from our flowchart condtions, the server displays the movements to the user interface and sends motor control commands again back over the antennas, to the specific ChairGuru, and through the motor control functions. All of this being constantly monitored by our built in software collision avoidance and path planning systems, and ready to be stopped at a moments notice.



Movement Requested
Chair is taken out of LPM and confirms location
Chairs new position is calculated and sent to chair
Is emergency stop button pressed?
No
Yes
Path planning activated and chair begins to move
Is chair in desired position?
No
Yes
Low power mode enabled
End
Increment emergency stop counter
wait 10 seconds
Has emergency stop counter reached 13?
No
Yes
Fault Mode Enabled
Chair cannot move within this movement request and must be reset manually using debugging header pins

Fig. 9: Movement Flowchart

From here, the ChairGuru system will finalize movements, and enter low-power mode, ensuring battery levels are optimized to make it through a day and that the system is ready for its next command without being a hindrance to the activities around it.

To illustrate this at a more complete subsystem level, figure 10 below shows the way we have our various subsystems connected.

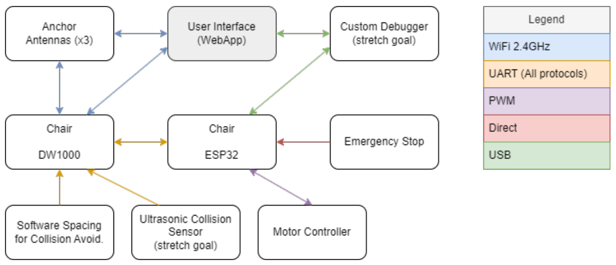


Fig. 10: Communication Block Diagram

Seeing the communication block diagram tie together nine various subsystems via five different methods helps us understand how this product pulls the subsystems together to produce a functional and useful device.

With the software implementation and use cases being described, we may be concerned with how exactly a user will input data? We don't expect the user enter a precise xyz steps in the correct orientation obviously. Instead, we offer a simple graphical user-interface (GUI) that allows them to create their layouts, plan their space, view current layouts, and initial reconfiguration; all at the users finger tips with no backbend knowledge of the system required! While this is great for an end-users experience, what exactly are they initiating with their points of contact?

Taking a better look now at how users can interact with the software systems, the figure below shows their three main points of interaction:

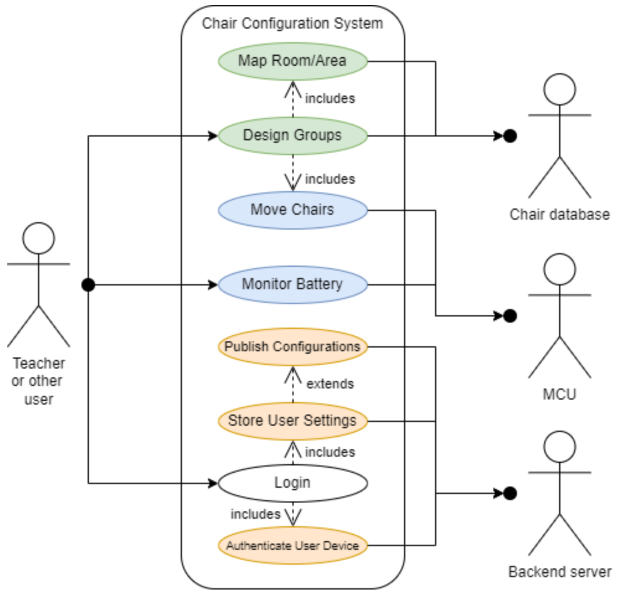


Fig. 11: Use Case Diagram

Firstly, we see the areas in green representing database interaction. This is the most important part as the user should spend upwards of 95% of their time here when working with

ChairGuru. This interaction with designing groups and setting up their space in the mapping areas allows them to create a custom solution to their exact room dimensions and obstacles. From here users add or remove chairs from their room, create their various layouts to rely on certain conditions such as room location (for rotating teachers or times on the daily schedule) and edit old layouts.

From here the user can initiate the ChairGuru movements by accessing the database and start the path planning process. The server is the key to all user initiated controls; and it stems here from the database access that is ”hidden” on the back-end.

Next, the user has some access to monitor chair status details. As we wanted to implement a battery monitor and autonomous, inactive hours, re-charging, the user could log in and see the battery level along with other various maintenance related items. These other related items could be minor warnings such as if a certain motor was not performing as well as intended, if the network connection was unstable, or if the chair collision sensors were tripped and sent into fault mode; to name a few. This interaction is largely a stretch goal but key to maintaining the ChairGuru fleet when we as developers are no longer around their deployments to diagnose the issues. This also work more hand-in-hand with figure 10 above where we talk about a custom debugger as another stretch goal. If we were to estimate a percent amount of time a user should spend in this part of the ChairGuru software system, we would ideally like to be as close to 0% as possible. With this, we'd like to see a user spend 1% or less time here as we aim to make maintenance as unneeded as possible.

Lastly, we have the last 4% of the time a user should spend interacting with the ChairGuru. Here we see the user settings and ways to actually access the software. We wanted to create a seamless user experience that didn't require constant logging in and out and instead should know who is logged in and what permissions, settings, and preferences they are entitled to. Our ideal use case would be to run ChairGuru as a computers service that runs on startup and initializes the chairs without manual input. This will pull user login data from their user account on their computer, ideally being an enterprise account such as a Microsoft enterprise domain.

This user interaction area enables the system to be used by various users and store configurations in multiple ways, including to the users account or to a room account where multiple users can share layouts OR maintain their own set of layouts for the same room. This also allows users to authenticate new devices to control the system in the event of wanting to use a different interface device, or to block potential wrongdoers.

E. MongoDB within MERN Stack

Knowing now the majority of our software systems, we keep referring to our server; but what exactly is it? Well to start, it is a MERN stack that makes our ChairGuru website fluid and interactions with various software components seamless. The letters stand for **M**ongoDB, the database, **E**xpress, a back-end framework, **R**eact, a front-end JavaScript framework, and **N**ode.js, a package manager and front to back end interface for the server commands.

React and Express offer great and industry standard frameworks that work very well for our purposes; thought the most important parts of the MERN stack for us are the Node.js and MongoDB portions. Node allows us to work with preexisting libraries and deploy open source code to use within our server side code. Some of the most important packages we utilize in this project include the environmental variable package and the MongoDB library that allows easy access to our data; streamlining data transfers and ensuring the focus is on good data practices rather than simply transferring the data in the first place.

MongoDB though is an integral part of the ChairGuru as it is a free and open source database that is able to run online via their Atlas component of MongoDB, or via the downloadable version. The ChairGuru is intended to operate with both versions of Mongo and is able to switch with a simple change in environmental variable links; something an IT department would have no issues accessing upon setup. For our final vision though, we utilize MongoDB downloaded onto our Windows machine. This is because Mongo is configurable as a system service that will run on startup and take login details directly from the ChairGuru user settings, similar to how the ChairGuru takes user details from a computers user information; these two systems work hand-in-hand with data retrieval.

This Mongo setup allows for our system to maintain usability while offline and wouldn't depend on specific computers if the service was setup on the user enterprise account. This means multiple things, first, we wouldn't see users needing to login into separate accounts multiple times and rather sharing a single (ideally) one-time login, and second, allowing for a complete omission of internet latency. This means cutting out the time delay of sending requests, receiving them back through limited bandwidth, and avoiding potential for packet-loss along the way. Keeping a local database ensures the latency is as minimal as possible; restricted to computer speed rather than other uncontrollable factors.

F. Acknowledgment

The authors would like to acknowledge Dr. Chan and Dr. Wei for guiding our research and advising us along the entirety of our project. We'd also like to acknowledge Dr. Weeks for the hours of technical help he provided while prototyping and constructing our project in the lab.

References

[1] ”ESP32 UWB Indoor Positioning Test.” Makerfabs, https://www.makerfabs.cc/article/esp32-uwb-indoor-positioning-test.html

[2] ”ESP32-WROOM-32-Datasheet.” Espressif Systems, https://www.makerfabs.com/desfile/files/esp32-wroom-32-datasheet-en.pdf

[3] ”DWM1000 IEEE 802.15.4-2011 UWB Transceiver Module.” Qorvo Inc., https://www.qorvo.com/products/d/da007946

[4] ”How HC-SR04 Ultrasonic Sensor Works & Interface It With Arduino” Last Minute Engineer, https://lastminuteengineers.com/arduino-sr04-ultrasonic-sensor-tutorial/

[5] ”11 Types of Networks Explained: VPN, LAN, & More” Belden, Inc., https://www.belden.com/blogs/network-types
[6] ”TB6600 Stepper Motor Driver Specifications” , https://bulkman3d.com/wp-content/uploads/2019/06/TB6600-Stepper-Motor-Driver-BM3D-v1.1.pdf?srsltid=AfmBOopXwRe3Nbok9UG-6AGUbimYIsjV8TxSAm9Hfx2LrT5EbJaacc3A