ANT-WEIGHT BATTLEBOT – DC HAMMER

By

Carson Sprague

Ian Purkis

Gage Gathman

Project Proposal for ECE 445, Senior Design, Spring 2026

TA: Haocheng Yang

01 May 2026

Project No. 37

Contents

1. Introduction

Our battle Bot, “DC Hammer”, is an optimized smashing machine. Our primary design philosophy centers around efficiency; fulfilling competition weight requirements, automating attack swings, and even enabling flip detection and controls inversion all demand efficient use of resources whether those are digital, analog, or even mechanical. Our goal in designing “DC Hammer” was to develop a well-rounded battle Bot that could effectively deploy offensive and defensive measures against a wide range of diverse opponents. Many battle Bot designs fall short on balance, typically leaning on a single feature to propel the bot to victory. With emphasis on both offense and defense, and a robust/sustainable design, “DC Hammer” will match-up against any opponent and be able to continue competing match after match. The modular subsystems of our bot will allow for a clear roadmap of development. Creating a functioning bot is our priority, with tuneups and additions as the development progresses. These modular subsystems will allow for flexibility with our weapons systems in the future as we build on top of the base bot. In this document we will go over the high-level motivations and expected product of our designs, as well as cataloguing the various components we will use to realize this design. Additionally, we will cover our expectations for the various subsystems in the bot and how we will approach realizing this finished product from an ethics and safety perspective.

Ultimately, the high-level requirements of the bot are as follows:

- **User Interface:**
  - Robot will accept user control signals via Bluetooth connection/ laptop keyboard input
  - Status indicator will accurately reflect robot state and connection status
- **Sensor Subsystem:**
  - Robot will detect inversion via accelerometer readings/data
  - Ultrasonic sensor will detect object in close proximity to front of robot within accetable margin of error
  - Sensor readings will be displayed in terminal for testing/debugging
- **Hammer/Wedge Subsystem:**
  - Hammer arm will automatically adjust default position based on robot orientation after a wait period
  - Hammer arm will swing automatically based on ultrasonic sensor data, and swing manually based on keyboard input
  - Hammer arm will return to default position within an acceptable margin of error
- **Power Subsystem:**
  - Robot will operate effectively via a 3S LiPo Battery, providing sufficient current for all subsystems
  - Power system will be regulated within acceptable margin of error
  - Power subsystem will support continuous operation for a 3-minute duration
- **Software Subsystem:**
  - Robot will update motor output/controls within acceptable latency period
  - Control will operate without blocking/locking for smooth control over driving functionality
  - Software subsystem will maintain deterministic execution under all operating conditions

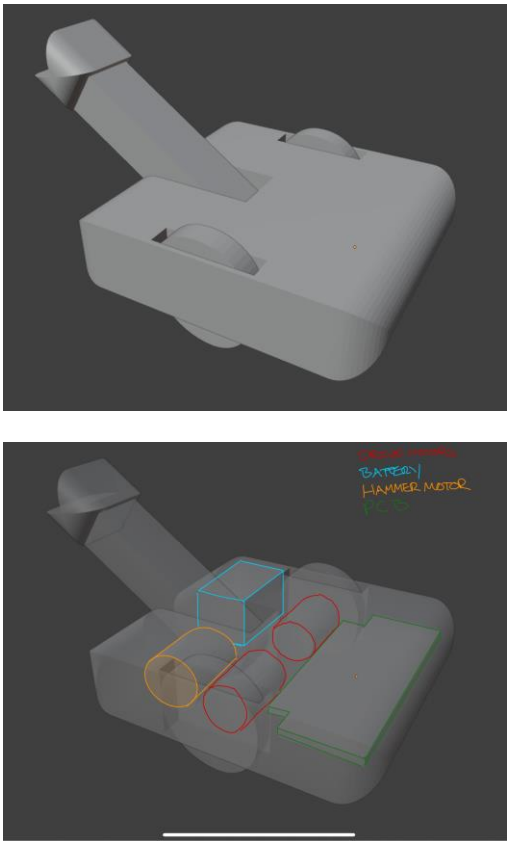

Figure 1 and 2 – Visual Aid (CAD Drawing) of Battle-Bot

2. Design

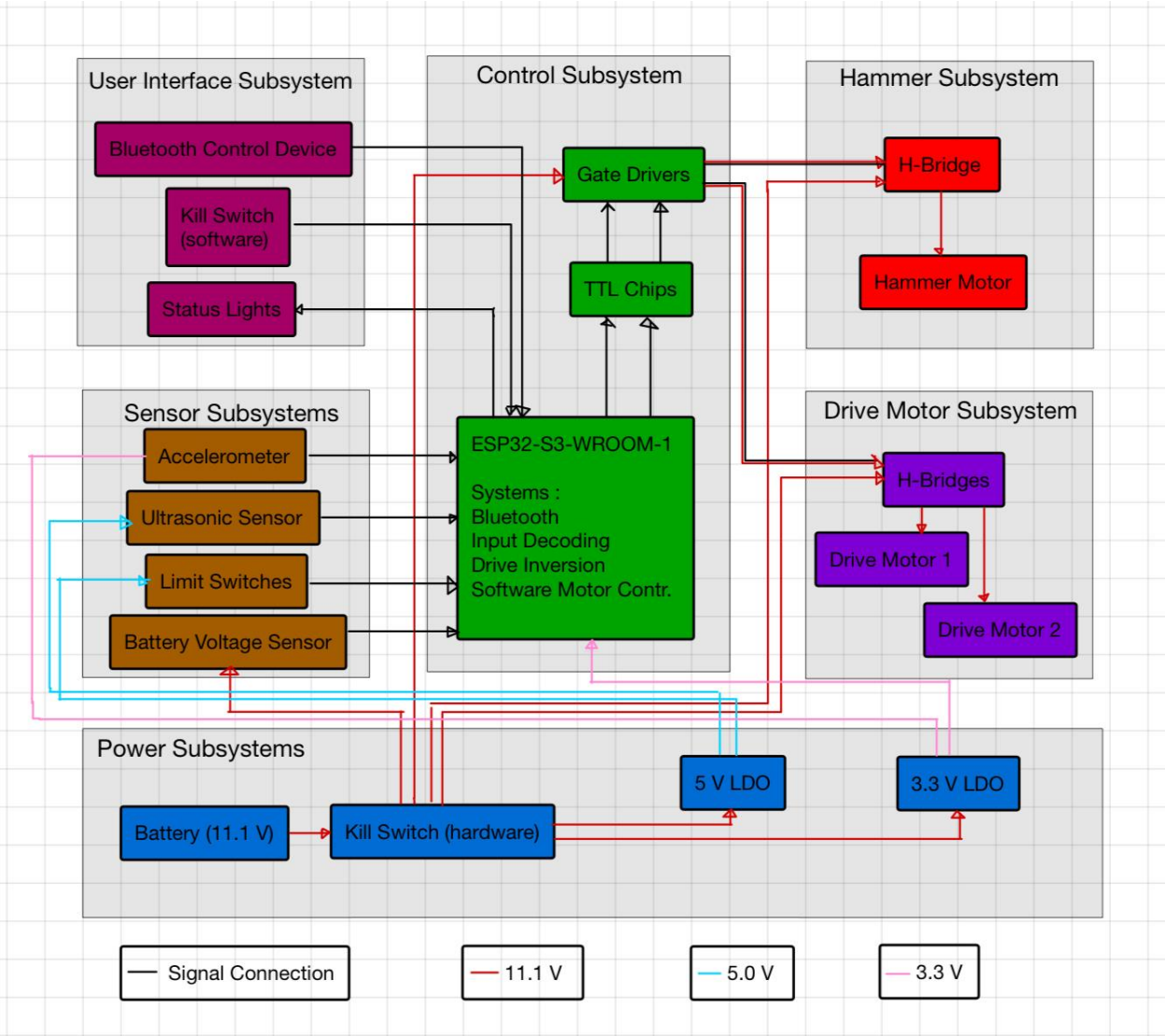

Figure 3 - Block Diagram

2.1 User Interface Subsystem

The user interface subsystem is responsible for providing a seamless interface for the user to control and monitor the battle bot. Users control the bot via a Bluetooth keyboard and can monitor the bot's status via a status LED. A kill switch is located on the bot, as required by competition specifications.

2.1.1 Microcontroller

The microcontroller (ESP32-S3-WROOM-1) has built-in Bluetooth support, supporting Bluetooth 5 and BLE. We will leverage the official Espressif Bluetooth Main API [2] for setting up the connection and reading input from the connected Bluetooth device. The MCU will run a loop that will process keyboard input. Along with taking input, the MCU will blink an LED depending on the bot status via one of its

many GPIO pins. Status codes have yet to be formally defined, but a rich enough set should be provided to promote a good user experience.

Specifically, as the MCU for our battle bot, this will interface with the ultrasonic sensor, the accelerometer, the limit switches, the battery voltage sensor, and the TTL chips on the hardware side, and will also handle the Bluetooth IO connection. This serves as the heart of our robot, to this extent it is the most important single item. Thus, it is critical it remains powered, and for which it requires a 3.0-3.6 V power supply, capable of 500 mA of current draw to function.

2.1.2 Kill Switch

A physical kill switch is required per competition guidelines, but it is important nonetheless to promote the safety and wellbeing of those nearby. We will also provide a kill switch remotely via the keyboard to further promote safety.

Requirements	Verification
Activation of the physical kill switch should disable all motor outputs within 50ms	We will create a testbench like system and run the motors at speed. At that point we will trigger the kill switch and measure the time from switch activation to the motors being disabled.
The LEDs should display all relevant Bluetooth states to the driver, including Not connected, Connected, and Fault.	We will connect, disconnect, and shut off the Bluetooth keyboard multiple times to ensure the LEDs update the status correctly and quickly to the driver.

2.2 Sensor Subsystem

The sensor subsystem collects data about the current orientation of the battle bot and position relative to the other battle bot. Namely, an accelerometer is used to keep track of the vertical acceleration of the bot, and an ultrasonic sensor is used to track when a bot is in front of us in striking range. We aim to make our control system robust to flips by automatically adjusting based on orientation.

2.2.1 Accelerometer

We use an MC3416 accelerometer as it should satisfactorily provide us with vertical acceleration data. By using an accelerometer and keeping track of the vertical acceleration, we can dynamically update our motor control system based on our orientation. This provides a seamless control interface for the user. The MC3416 uses the I2C protocol to transmit data to the MCU (ESP32-S3) via two GPIO pins and requires 3.3 volts. Pullup resistors are also required on output pins [4].

2.2.2 Ultrasonic Sensor

We use an Adafruit 4007 ultrasonic sensor not only because it should be satisfactory in terms of getting the distance to the nearest object in front of us, but also because it is functional with both 5v and 3.3v [1], which allows us more flexibility in our power subsystem design. This sensor, like most other ultrasonic sensors, transmits data via two GPIO pins, sending a trig and echo signal. We can compute the distance to the nearest object directly in front of us in software.

2.2.3 Limit Switches

Two simple switches will be used to pull down pin inputs to our MCU, to indicate an end of motion for our attack arm. These will prevent the over-currenting of our hammer motor and provide simple feedback. When operating a brushed DC motor, its speed will change with voltage, and the battery voltage will vary with the state of charge. These constitute critical components, as failure of them to detect a stop of the motor may cause component failure. This is due to the high torque of the hammer motor that may result in damage to our chassis.

2.2.4 Battery Voltage Sensor

This simple subsystem will handle battery voltage monitoring. It will function to stop our LiPo batteries from being over-discharged. The voltage will be taken into our MCU and be used as a key to turn off our motors in the case of a low battery. Failure of this component is not critical for our robot, but will be costly, as ruined batteries may never re-charge properly. This will be a passive system, with only a voltage divider and diode protection for our MCU.

2.2.5 TTL Chips

To take our digital logic out of the MCU and limit the number of pins required, we will do some post processing using TTL chips. Using a given number of pins to control the direction of the drive motors, we will be able to control the H-bridges. This component thus takes the MCU outputs and “switches” the path on the H-bridge, more conveniently than using BJTs or other components, still it remains a 3.3 V output which will have to be stepped up by the gate drivers.

2.2.6 Gate Drivers

We have chosen DGD0211CWT-7 as gate drivers. These chips will be able to take our final PWM signals from the TTL chips at 3.3 V and apply the battery voltage ~11 V to the gates of the H-bridge MOSFETs. These devices serve a critical role in the control circuit, bridging the gap between the “low” and “high” voltage segments.

Requirements	**Verification**
The system shall detect with the bot is inverted and apply motor control state changes within 100ms.	We will place the bot upright and log the accelerometer vertical axis value. We will then flip the bot and measure how long it takes for motor controls to update. We will output logging data to the terminal.
The ultrasonic sensor shall detect objects within 5-50cm with an accuracy of +-2cm.	We will place the bot down and place another object in front of it (measured with a ruler). We will log the ultrasonic sensor data to the terminal and compute error.
Activation of a limit switch shall disable hammer motor drive within 20ms.	We will drive the hammer toward end stop, trigger the limit switch, and measure the time from switch signal change to a PWM duty of 0 on the oscilloscope.
The battery voltage sensor shall disable motor outputs when the voltage falls below 9 volts.	We will replace the battery with a bench supply and slowly lower the voltage, confirming that the motors disable at 6 volts +- 0.2 volts.

The gate drivers shall provide at or near battery voltage gate voltage when the battery voltage is normal (9-12.6V).	We will prove the MOSFET gate relative to the source, drive the PWM at 50%, and confirm the gate voltage is over 10 volts.

2.3 Hammer/Wedge Subsystem

The primary offensive tool is the hammer/wedge “attack arm”. This arm is driven by a high torque motor and activated by a proximity-detecting ultrasonic sensor for faster than human reaction time when opponents enter attack range. The attack swings will also have a user input trigger coming from a key on the laptop controlling the bot: likely the space bar. Additionally, the arm will have two default or resting positions to allow for a variety of methods for offense.

2.3.1 Hammer

The first attack mode, and namesake of the bot, is the hammer. The resting position will be high above the bot at an obtuse angle relative to the arena surface to allow for the greatest work done, or energy delivered in the resulting swing (the angle will be approximately 115° or 2 rad).

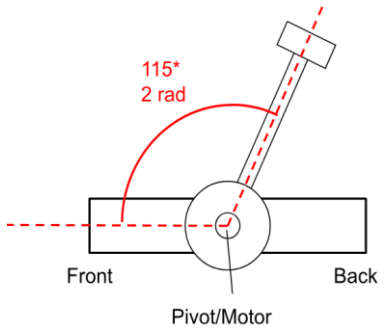

Figure 4 – Hammer Arm Action Diagram

Figure is not a visual representation of robot appearance, simply to visualize the position of the hammer relative to the rest of the bot.

A potential addition to the hammer arm that we’d like to explore is the addition of a kinetic payload to the hammer head. In this implementation of the hammer head, we’d effectively have a second impact for every hammer swing and maximize the energy transfer of the hammer impact by taking advantage of the net forces experienced by the payload during the motion. We would hollow out the inside of the hammer head to be a tube-shaped shaft, housing a steel bearing ball. The ends of the hammer head would have circular openings that allow for the bearing ball to extend partially outside of the hammer dimensions when forced toward the opening. The material, steel, of the bearing ball and more focused point of impact should allow for a more devastating transfer of energy upon impact, especially against other battle bots.

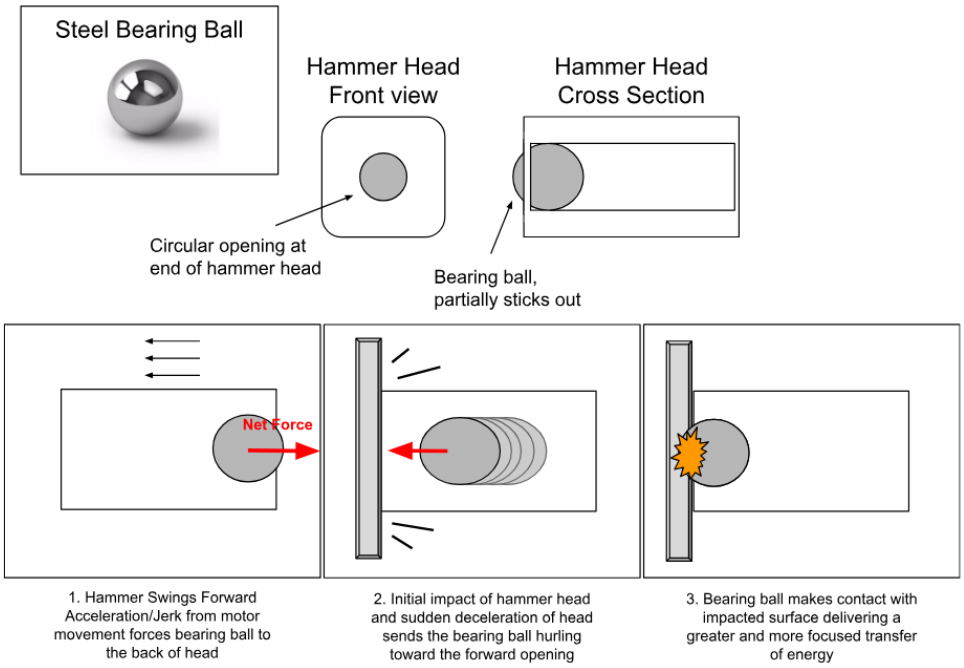

Figure 5 – Hammer Ball Bearing Mechanism

2.3.2 Wedge

The secondary attack mode will have a default resting position about -5° below the central plane of the robot (approximately -0.1 radians). Additionally, the shape of the hammer head will include a wedge on either end of the head. The goal of this mode is to get the wedge underneath the chassis/drive trains or end effectors of other robots and then execute a forceful upward swing to attempt to disable opposing bots by flipping them.

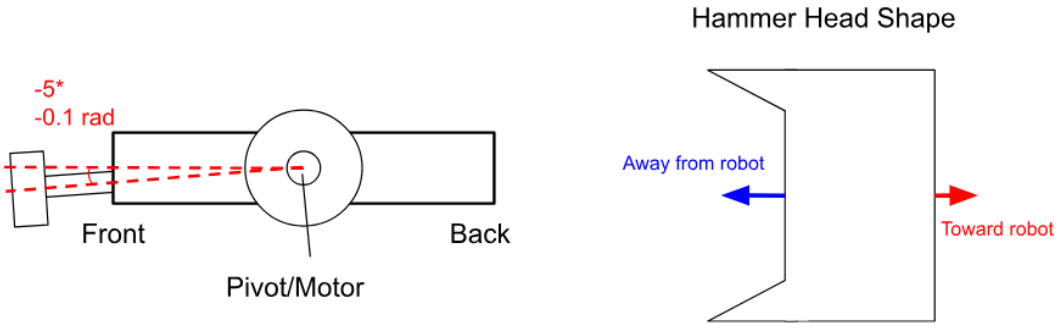

Figure 6 – Wedge Diagram

2.3.3 High Torque Motor

We have selected a 12 V, 45 RPM, brushed DC geared motor, with a rated stall torque of 50 kg-cm. This motor will conduct any flipping or smashing rotations of the hammer and should be adequate for any roughly 1 kg battle bots it should face. For this purpose, this motor requires 0.68 to 2.19 A of current,

rated and stall respectively. In combination with the drive motors, this will compromise the majority of the current required by our robot.

Requirements	**Verification**
Hammer default position will change based on the orientation of the robot. After 1 sec of inversion beyond 90 degrees, the default arm position should invert to account for orientation change.	A lot of the expected hammer behavior is the result of multiple sub modules working as intended (i.e. sensor trigger/accuracy, motor control, Bluetooth connection, etc.). To test the rotation threshold for the robot, we could output accelerometer readings to a display (perhaps via Bluetooth to a window on piloting laptop). This would help us see that the inversion is triggered after exceeding greater than 90 degrees of inversion. We can also test the delay in the same way, by inverting the bot beyond the threshold and observing the arm position adjustment within a particular time delay (1 second as of now).
Hammer/wedge swing will be triggered by sensor input when object is within range of the hammer arm. Hammer/wedge swing will be triggered by user/keyboard input	As in the previous design requirement, Success of the swing triggers is the result of multiple subsystems working correctly. Testing this specific behavior is as simple as observing a swing as a result of user/keyboard input, and moving a detectable object closer and closer to the ultrasonic sensor until a swing is triggered. Using a ruler or meter stick we can check at what distance the swing is triggered. Currently this distance is unknown, as we intend to try several variations of arm design and will depend on the arm's pivot point relative to the rest of the robot's chasse.
Hammer arm will return to initial position (after swinging action) within +/- 2 degrees	We can utilize the user input for discrete testing of hammer swing rotation. The model of motor we are planning on utilizing does not have the ability to measure rotation. Rotational control would be relative to motor speed (input current magnitude) and duration of activation (input current duration). To verify the actual rotation amount, we could use a protractor and manual swing to observe the correct rotations (forward and backward swing).

2.4 Power Subsystem

To give life to our battle bot, a power system is needed to supply necessary power to all components, including drive motors, hammer motor, MCU, sensors, etc. A variety of components are used to ensure a safe and robust system to deliver power where necessary.

2.4.1 Battery

The main battery is to provide requisite power for the entire system. For this purpose, a 3s (11.1v) LiPo battery of sufficient capacity is required. This battery will connect directly to the kill switch, through which there will be connections to all major components, particularly the voltage regulators and to the motors. This is a critical component due to the fact that it must provide sufficient current and voltage to three 12v DC motors which represent the majority of our battle bot's power consumption. Particularly, this battery must be able to store sufficient energy to power these motors for 3-minute bouts.

2.4.2 Voltage Regulator – 5.0 V

We have conducted a redesign since our proposal. After subsequent thermal capacity calculations, it was obvious to see that our original LDO scheme, even under resistive paralleling, would pose significant risk of issues. Instead, after consideration, we instead decided to leverage existing integrated MOSFET buck converter ICs. To this extent, we decided to leverage Ti's TPS563252DRLR, which is a 3-amp rated 94% efficient, low resistance buck converter. This was configured to have less than 3% voltage ripple output directly at 5V.

2.4.3 Voltage Regulator – 3.3 V

Multiple components on our battle bot will require a 3.3 V power supply. Principally, the MCU, an ESP32-S3 chip, will require a rated 3.3 V and 0.5 A of power to operate with a minimum and maximum operating voltage of 3.0 V and 3.6 V respectively. To this constraint, we have chosen the AZ1117C-3.3, whose datasheet indicates a ±1% accuracy, minimal regulation, and a minimum of 1 amp of load.

Requirements	**Verification**
Battery must be able to provide for sustained operation of Battlebot for the duration of one bout of at least 3 minutes in length.	We will conduct test bouts with our robot, activating all subsystems frequently and continuously in the case of motors to validate our loading calculations. Assessing battery voltage before, during, and after using datalogging on our ESP32-S3.
5.0 Volt Regulator will need to be able to output consistent load in line with supply for subsystems requiring lower-than-battery voltage.	We will connect an artificial test load to our circuit of equivalent 1.0 Amp draw, using an electronic DC load or equivalent. During this test we will also assess the voltage ripple and load regulation of our circuit, to ensure it is within the 3% spec in voltage tolerance via multimeter.
3.3 Volt Regulator will need to consistently supply a reliable 0.5 amp at steady-state to supply our ESP32-S3 for its operations.	We will once again test our circuit with an artificial test load of equivalent 0.5 Amp draw, using an electronic DC load or equivalent.

	During this test we will also assess the voltage ripple and load regulation of our circuit, to ensure it fits within expectations via multimeter. We will also use a multimeter thermocouple to assess temperature and compare with calculated loaded temperature rise.

2.5 Software Subsystems

2.5.1 Bluetooth

One of the core functionality features that's required for our battle bot is wireless control. We'll leverage the ESP32-S3's Bluetooth (Low Energy) capabilities to achieve this. Specifically, we'll utilize the Apache Newt NimBLE Bluetooth host stack, which the ESP-IDF supports [8]. We chose to go down this route rather than a traditional Bluetooth stack because of the lower RAM footprint, event-driven design, suitability for resource constrained systems, and that it integrates well with FreeRTOS (which runs on the MCU).

The bulk of the code written will just go into initializing the Bluetooth LE stack, defining interfaces, and registering for subscriptions that go along with that. Core to BLE functionality are the Generic Access Profile (GAP) and the Generic Attribute Profile (GATT) [7]. These act as interfaces, where GAP describes how the device behaves during scanning and connection events, and GATT defines what data is accessible to clients and how (read, write, etc). In our case, the ESP32 is a client to the Bluetooth keyboard. Aside from being mostly pedantic system initialization, these interfaces allow us to define and register our own callbacks when an event or notification occurs (hence event-driven design), instead of constantly polling for new data, which wastes CPU cycles and drains power.

The GAP callback is responsible for decoding the GAP event that the MCU just received and handling it accordingly. It will fire when the ESP32 connects or disconnects to the Bluetooth keyboard. These events and the Generic Access Profile as a whole are responsible for just setting up the BLE connection. The event will be one of the following and will be handled as such [8].

```
BLE_GAP_EVENT_CONNECT

BLE_GAP_EVENT_DISCONNECT

BLE_GAP_EVENT_CONN_UPDATE
```

The GATT callback is more interesting as the GATT notifications will contain the key/control info from the Bluetooth keyboard. This callback fires when the GATT server (keyboard) sends data to the GATT client (ESP32) containing the current state of one or more keys. The input decoder will then be responsible for parsing the pressed/released keys and driving the motors accordingly.

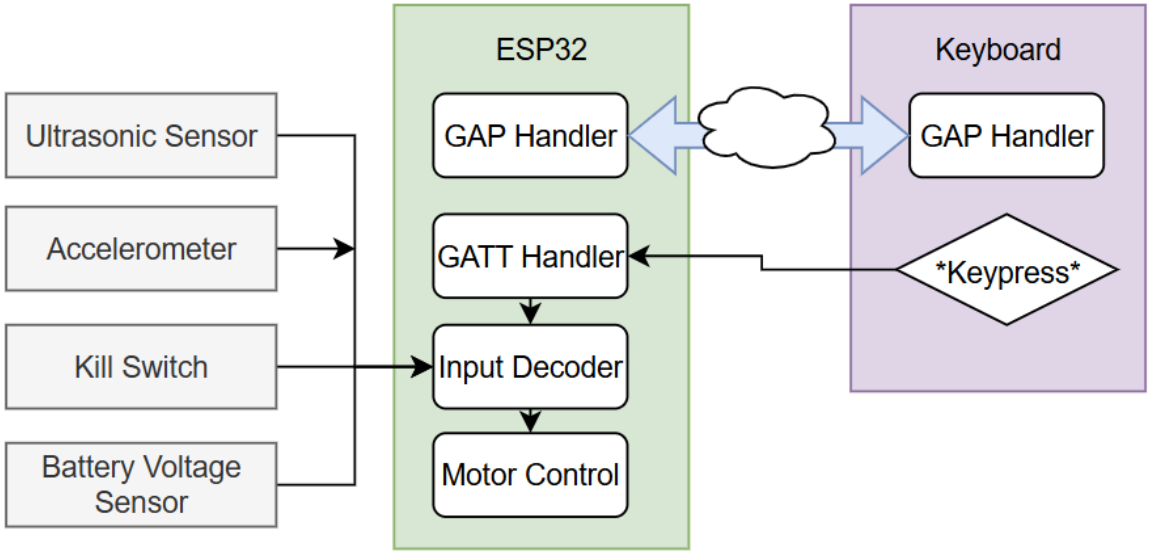

Figure 7 - Software Architecture

2.5.2 Input Decoder

The input decoder takes in keyboard input (passed in from the Bluetooth handler) as well as sensor inputs from the board. These include the ultrasonic sensor, accelerometer, kill switch, and battery voltage sensor. The data packets coming from the keyboard will come in a HID (Human Interface Device) packet.

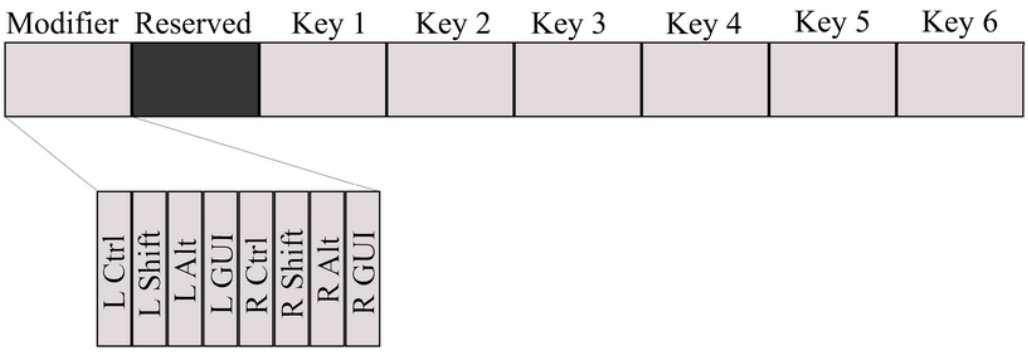

Figure 8 - HID Packet [6]

This data format allows us to keep track of up to 6 key presses at a time – far more than the controller will need at one time. We tentatively plan to use the following keys for control:

Key	Function
W	Forwards
A	Left
S	Backwards
D	Right

Shift Modifier	'Sprint'
Space	Hammer
Tab	Change Mode
K	Kill Switch

The input decoder also takes in a signal from the accelerometer over I2C. Fortunately the ESP-IDF has ample support for I2C so configuring the GPIO pins to handle this protocol is straightforward. We're only really interested in the vertical acceleration which tells us if the bot is upright or upside down. At any point, if the vertical acceleration switches directions, we will swap what 'forward' and 'backward' means from a motor control perspective. E.g. instead of forward being clockwise when upright, it will be counterclockwise when inverted.

The ultrasonic sensor provides us information about what's directly in front of us. We will fire the ultrasonic sensor and then use the roundtrip time to calculate the distance to said object in front of us (a rival battle bot). Given that sound travels at around 343 m/s, or more useful to us, 0.034cm/microsecond, we can use the following equation to calculate the distance in software.

$$D = \frac{T * 0.034}{2}$$

Where D is in centimeters, and T is in microseconds. We can adjust the 'kill zone' in which the hammer will automatically strike in software, so we will tune accordingly once we have a working prototype.

2.5.3 Software Motor Control

Now that we've collected input from the user, and all the signals have been handled, we can finally drive the motors. To handle motor control, we will use Espressif's LED Control library (LEDC) [5] which generates PWM signals (for more than just LEDs). We'll take advantage of a couple of APIs to set and update the duty cycle on the fly, such as [5]:

```
ledc_set_duty()

ledc_update_duty()
```

This flexibility allows us to tune our parameters in software to achieve whatever speed we deem fit for competition. This also allows us to support a crawl, walk, and sprint speed if we find it advantageous. To handle turning left and right, we will increase the duty cycle of one motor while decreasing the duty of the other. We'll use a simple case-switch statement to handle these different states, and again tune our parameters until we're happy with how the bot maneuvers.

Requirements	**Verification**
Must achieve a latency of under 1ms from receiving a key press to driving the motors	When a key press is received, it will trigger the GATT handler. We will start a timer in this handler and then stop the timer once we update the duty cycle in the motor control handler. We will write this to the terminal.

Note that this is not the time from physical keypress, but the time from which the signal is received at the MCU.	
Spamming keypresses while driving should not cause the bot to lock up or have controls 'lag' behind. In competition we will be pushing our bot to the limits. We cannot have our control system breaking down or lagging under high strain - otherwise we are at a massive disadvantage.	We will alternate through WASD as fast as we can on the keyboard for 5 or so seconds, then we'll stop. If the bot continues driving (meaning our handler is too slow), or the control becomes seemingly random (signals are being corrupted), we need to streamline our software to be more efficient.

2.6 Drive Subsystem

2.6.1 H-Bridge NMOSFETs

For our motor to operate properly we will need high-power MOSFETs for our H-Bridge. Instead of purchasing integrated ICs, we have designed an H-Bridge that can be triplicated across all three motors. The chosen devices are as follows, for NMOSFETs Onsemi's FDC655BN, which are 30 V, 6.3 A rated MOSFETs, for PMOSFETs CPH6350-TL-W, which are 30 V 6.0 A rated, both with low on-state resistances and high switching frequency capabilities, for our diode protection we have chosen 80SQ045NG, which are 45 V 8 A, 0.55 forward voltage diodes, which should provide current paths while our MOSFETs are switching. These should be overrated for our application to ensure safety in our circuit. If one of these devices were to fail, it would not only constitute a robot failure but may be critical to battery safety. However, since these devices are overrated, they should not be a limiting element in our design. Like any other critical path electrical component, failure will be caused by loss of voltage, or improper operation.

2.6.2 Drive Motors

Our main two drive motors are 12 V, 600 RPM, brushed DC geared motors. These will propel our battle bot, which is a critical feature for obvious reasons. Their current requirements are 100-1600 mA for the no-load to stall current operating range. This helps us inform how we must compute our battery size. These will interface with the H-bridge modules and the ground. These motors will need to be durable, as this robot will likely experience high-g forces being tossed around.

Requirements	**Verification**
H-Bridge functionality will be determined by its ability to properly control the motor connected to it, and effectively handle the heat generated.	Temperature stability tests will be performed where a load is connected until steady-state temperature is observed, ensuring safe temperatures limits. Frequency tests should also be included, to determine our maximum switching frequency achievable, to do this, with no load connected and an oscilloscope measuring voltage, verify the

	maximum frequency for a minimum 10% duty cycle for motor control.
Motors will need to be functional and highly reliable for long durations and under stall/stuck rotor conditions.	When motors arrive, drive duration tests will occur, with continuous drive for up to 3 minutes (battle duration) with load, and up to 10 minutes no load. For these tests, a thermocouple will be used to determine maximum case temperature. Short duration tests will be conducted for stalled rotor conditions to verify maximum current draw, to verify accuracy to schematics. Finally, DC motor characterization will be carried out to determine optimal switching frequency, this can be simultaneously conducted with the stalled rotor and no-load tests.

2.7 Tolerance Analysis – Critical Subsystem

In our IC heavy PCB, we are unlikely to have many issues with individual components, or small system failures. More than likely, improper cooling or lack of power will be a bottleneck. For the purpose of handling high current for our motors, this is easy, we can simply over-rate our H-bridge components in terms of current, making the circuit safer and more functional for a wider variety of brushed DC motors. For this purpose, a current overhead factor of 3x was selected for greater margin of safety, so our lowest rated component in our H-bridge is our 6 A PMOSFET (roughly 2-amp locked rotor current expected).

However, our power supply is not so fortunate. With a battery voltage of 9.0-12.4 volts, we need a robust system to provide reliable power to our MCU and accessories, operating around 5.0 and 3.3 V. In terms of acceptable supply voltages, our MCU datasheet requires the following,

Symbol	Parameter		Min	Typ	Max	Unit
VDD33	Power supply voltage		3.0	3.3	3.6	V
I_{VDD}	Current delivered by external power supply		0.5	—	—	A
T_A	Operating ambient temperature	65 °C version	−40	—	65	°C
		85 °C version			85	
		105 °C version			105	

To this purpose, we have selected for the 3.3 V supply the AZ1117CH-3.3TRG, whose output parameters are shown below, from which it is clear to see that the nominal output including loading effects has this device clearly being able to output the necessary current and voltage to the specifications of our MCU.

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{OUT}	Output Voltage	1.5V ≤ V_{IN}-V_{OUT} ≤ 10V		3.267	3.3	3.333	V
				3.235	3.3	**3.365**	
V_{RLINE}	Line Regulation	1.5V ≤ V_{IN}-V_{OUT} ≤ 10V		—	0.5	6	mV
				—	—	**10**	
V_{RLOAD}	Load Regulation	—		—	2	15	mV
V_{DROP}	Dropout Voltage	ΔV_{OUT} = 1%, I_{OUT} = 0.8A	SOT223	—	1.2	1.3	V
			TO252-2 Series	—	1.3	1.4	V
I_{LIMIT}	Current Limit	—		1	1.35	—	A

What is not clear from this simple picture is how we get from our 12.4 V battery to a suitable input for this AZ1117CH-3.3TRG. It is true that the AZ1117CH-3.3TRG can handle up to 15 nominal input, but it is also obvious to anyone who understands LDOs that this is infeasible. Low Drop Out regulators are simple devices, their design means that their waste energy is proportional to the voltage drop across them, doing some simple calculations as shown below clarifies that this device may, or more accurately, will overheat.

(12.4 V (in) – 3.3 V (out)) x 0.5 A (minimum required) = 4.55 Watts

4.55 W * 15 C / W (Junction to Case Thermal resistance) = ~ 70 C rise

Clearly a 70-degree Celsius rise just from junction to case thermal resistance means that this is simply a non-starter. To mitigate this effect, we can instead employ an intermediate level conversion. Since we also need 5 volts, this is the optimal voltage to select as it reduces the number of components required. To this extent, we selected an integrated MOSFET buck converter IC, TI's TPS563252DRLR. Using approximately TI's suggested design characteristics, we can see that this configuration will significantly reduce thermal losses in our LDO.

(5.0 V (in) - 3.3 V (out)) x 0.5 A = 0.85 Watts

Importantly this configuration has excess current capacity, up to 3 A, and relatively small voltage tolerances under normal conditions. These two things give us more freedom when designing the circuit. Fundamentally, we will also manually validate this circuit as a precursor to the final board revisions, to ensure reliability.

3. Cost and Schedule

The table below outlines the cost for all the major individual components we plan on using, with a total of $101.69 before tax and shipping. Assuming shipping adds about 5% and sales tax adds another 10%, we can expect to pay just around $120 for components. As ECE graduates from Illinois, we can expect a salary of about $40/hr. Thus, we can calculate the labor cost to be about $40/hr * 2.5 * 40 hrs, roughly $4,000 per team member. With three team members, that's $12,000 in labor costs. The total cost of this project comes out to be just about $12,120.

Component	Component ID	Link	Price per	Quantity	Total

Functionality			component		Cost
PMOSFET	CPH6350-TL-W	https://www.digikey.com/en/products/detail/onsemi/CPH6350-TL-W/4847613	0.71	6	4.26
NMOSFET	FDC655BN	https://www.digikey.com/en/products/detail/onsemi/FDC655BN/979810	0.74	6	4.44
Diodes	80SQ045NG	https://www.digikey.com/en/products/detail/onsemi/80SQ045NG/1475457	0.63	12	7.56
Robot Off Switch	FingerTech Mini Power Switch	https://www.fingertechrobotics.com/proddetail.php?prod=ft-mini-switch	9	1	9
Accelerometer	MC3479	https://www.digikey.com/en/products/detail/memsic-inc/MC3479/15292802	1.11	1	1.11
Buck Converter IC	TPS563252DRLR	https://www.digikey.com/en/products/detail/texas-instruments/TPS563252DRLR/20415402?s=N4IgTCBcDaICoAUDKBWAbAZjCsARASgDL4gC6AvkA	0.49	1	0.49
Buck Converter Inductor	DR74-2R2	https://www.digikey.com/en/products/detail/eaton-electronics-division/DR74-2R2-R/667219	0.57	1	0.57
Main MCU	ESP32-S3-WROOM-1	https://www.digikey.com/en/products/detail/espressif-systems/ESP32-S3-WROOM-1-N8/15200089	5.49	1	5.49
Ultrasonic Sensor	Adafruit 4007	https://www.digikey.com/en/products/detail/adafruit-industries-llc/4007/9857020	3.95	1	3.95
Gate Drivers	DGD0211CWT-7	https://www.digikey.com/en/products/detail/diodes-incorporated/DGD0211CWT-7/12702560	0.55	12	6.6
Hammer Motor	12V 50RPM 694 oz-in Brushed DC Motor	https://www.robotshop.com/products/12v-50rpm-694-oz-in-brushed-dc-motor?pr_prod_strat=e5_desc&pr_rec_id=1e033aea8&pr_rec_pid=7487305908385&pr_ref_pid=7487308398753&pr_seq=uniform	15.88	1	15.88
Wheel Motors	12V 600RPM 50:1 Gearmotor	https://www.robotshop.com/products/dyna-engine-12mm-diameter-501-micro-metal-gearmotor-12v-600rpm	6.95	2	13.9
3V3 Regulator	AZ1117CH-3.3TRG	https://www.digikey.com/en/products/detail/diodes-incorporated/AZ1117CH-3-3TRG1/4470985	0.16	1	0.16
Battery	Turnigy 450mAh 3S 70 LiPo pack	https://hobbyking.com/en_us/turnigy-nano-tech-plus-450mah-3s-70c-w-xt30.html	6.39	2	12.78
Connectors (Assorted)	JST-VH series	-	0.25	10	2.5

Resistor (Assorted)	Varying Sizes	-	5	1	5
Capacitors (Ceramic)	Varying Sizes	-	5	1	5
Capacitors (electrolytic)	Varying Sizes	-	3	1	3

We aim to follow the schedule below to stay on track but understand that this is just an estimate and some things may take more or less time than allotted.

Week	Tasks
Feb 22 – Feb 28 **PCB Round 1 – Feb 26**	Carson: Begin work on Bluetooth code (and design doc) Gage: Gage Finish a preliminary board for power rail and H-bridge verification Ian:
Mar 1 – Mar 7 **PCB Round 2 – Mar 5**	Carson: Continue work on Bluetooth code and prep for presentation, help with 3d printing & PCB Gage: Create an initial complete schematic for design review. Ian:
Mar 8 – Mar 14 **PCB Round 3 – Mar 12**	Carson: Complete Bluetooth code / Begin input decoder and motor control code, help with 3d printing & PCB Gage: Complete initial PCB Ian:
Mar 15 – Mar 21 *Spring Break*	Carson: Traveling Gage: Staying Home Ian: Traveling
Mar 22 – Mar 28 **PCB Round 4 – Mar 26**	Carson: Continue input decoder and motor control code, help with 3d printing & PCB Gage: Validate H-Bridge and Power Rail PCBs. Ian:
Mar 29 – Apr 4	Carson: Complete input decoder and motor control code, help with 3d printing & PCB Gage: Create 3d model for shell or begin testing PCBs Ian:
Apr 5 – Apr 11 **Progress Demo**	Carson: Integration, prep for demo, test, tune parameters. Gage: Create 3d model for shell or begin testing PCBs Ian: Integration, prep for demo, test, tune parameters.
Apr 12 – Apr 18	Carson: Bug fixes, prep for demo/presentation Gage: Polish or create work arounds for any final issues. Ian:
Apr 19 – Apr 25 **Mock Demo/Presentation**	Carson: Final bug fixes, prep for demo/presentation Gage: Final bug fixes, prep for demo/presentation

	Ian:
Apr 26 – May 2 **Final Demo/Presentation**	Carson: Prep for demo/presentation Gage: Prep for demo/presentation Ian: Prep for demo/presentation
May 3 – May 9 **Final Papers Due – May 6** **Lab Notebook Due**	Carson: Cleanup and submission Gage: Cleanup and submission Ian: Cleanup and submission

4. Ethics, Safety, and Societal Impact

Despite the proposed system being a small-scale combat robot/battle bot designed for a controlled competition environment, it still involves mechanisms with intent to do harm. Thus, it's important and necessary to describe our responsibility as engineers to uphold the highest standards regarding ethics, safety, and impact.

4.1 Ethics

The ethical responsibility in this context revolves around containment, preventing misuse, and playing by the rules described in the competition handbook. We want to emphasize that this system is designed strictly for the end of year battle-bot competition, which takes place in an enclosed and supervised testing environment, and is not at all intended to harm people, animals, property, etc. As referenced in the IEEE Code of Ethics I.1 [3], we hold the safety, health, and welfare of the public as the upmost priority. We commit ourselves to creating a safe and contained system and will take measures along the way to ensure that the power of our battle-bot is never in the wrong hands.

4.2 Safety

This battle-bot includes rotating motors and potentially high-speed mechanical components, which would introduce injury risk if improperly handled. Furthermore, all components are powered by electricity, which could pose a threat if safe procedures are not followed during development. We commit ourselves to firstly follow the competition handbook and ensure that our bot meets competition regulations and adheres to the safety standards as defined within them. Secondly, but equally as importantly, we commit ourselves to following lab and general electrical safety standards during development. We understand that, as defined in the IEEE Code of Ethics I.6 [3], if we encounter tasks beyond our training or qualifications, we will reach out to someone who is qualified. Examples include using the machine shop or using high-voltage components. Thirdly and finally, we commit ourselves to implementing control safety. Should one of our systems go out, such as a loss of Bluetooth signal, we will implement fail-safe behavior to completely shut down our system should we ever lose control.

4.3 Societal Impact

Although this is an educational toy battle-bot, the broader societal implications contain the development and/or normalization of autonomous weapon systems. This project is limited and does not pursue autonomous targeting nor serious offensive capabilities, but it is important for us to remain aware of the broader implications during development. We as young engineers use this project to

contribute to our hands-on education, safe electronics design, and control systems development. This battle-bot and the skills acquired during development are transferable to a number of constructive applications, such as search and rescue robotics. We hope that by the completion of this project, we've inspired others to use and develop more robots to contribute to the good of society, rather than take away from it.

References

[1] Adafruit Industries, “Ultrasonic Distance Sensor – 3V or 5V – HC-SR04 compatible – RCWL-1601,” 2019. Available: https://mm.digikey.com/Volume0/opasdata/d220001/medias/docus/307/4007_Web.pdf. (Accessed: Feb. 9, 2026).

[2] Espressif Systems, “Bluetooth® Main API,” *ESP-IDF Programming Guide*. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-reference/bluetooth/esp_bt_main.html#api-reference. (Accessed: Feb. 9, 2026).

[3] IEEE, “IEEE Code of Ethics,” 2020. Available: https://www.ieee.org/about/corporate/governance/p7-8. (Accessed: Feb. 12, 2026).

[4] Memsic Semiconductor Co., Ltd., “MC3416 3-Axis Accelerometer,” 2021. Available: https://www.digikey.com/en/products/detail/memsic-inc/MC3416/15292804. (Accessed: Feb. 9, 2026).

[5] “Led control (LEDC),” Espressif. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/ledc.html (Accessed Feb. 25, 2026).

[6] L. Babun, *USB Human Interface Device (HID) Report Examples Fig 1.* 2020. Available: https://www.researchgate.net/figure/USB-Human-Interface-Device-HID-report-examples_fig1_339632610 (Accessed Feb. 25, 2026).

[7] “How gap and GATT work - bluetooth low energy basics,” Punch Through. Available: https://punchthrough.com/how-gap-and-gatt-work/ (Accessed Feb. 25, 2026).

[8] “Nimble-based host apis,” Espressif. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-reference/bluetooth/nimble/index.html (Accessed Feb. 25, 2026).