EEL 4924C Senior Design
December 12, 2022
Design Report

BALDUR

(Basic Autonomous Light Detection and Ultrasonic Recognition Robot)

Figure 1 Custom Logo

By: Juan Pablo Orozco
& Luis Cedeño

Table of Contents

Table of Figures

Abstract

Autonomous mobile robots require robust systems of simultaneous localization and mapping (SLAM) as these features are key characteristics for the navigation, performance, and efficiency of autonomous robotic systems. As a result, the following project consists of developing an automated mobile robot that can navigate indoor environments and construct a two-dimensional map of the area in real time. For this application, the team wishes to implement a vigorous system of object avoidance by coupling input sensing technologies using ultrasonic transducers and a LiDAR (Light Detection and Ranging) sensor to feed information about nearby objects to a centralized computing unit capable of performing SLAM algorithms and two-dimensional modeling for interfacing with the user. In addition, the robot will contain added peripherals to improve the project's features and complexity. These include: wireless autonomous charging, LED outputs, user displays, light detecting circuit and a speaker for audio capabilities as shown in Figure 1, showing the finalized design.

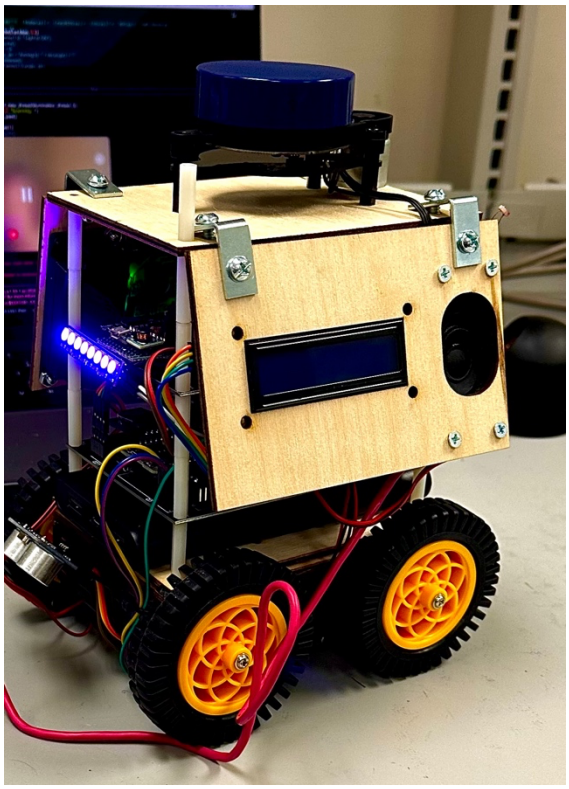


Figure 2 Finalized BALDUR Robot design

Introduction

Autonomous systems play a fundamental in a wide range of applications such as autonomous driving vehicles, humanoid robots, assistive systems, military systems, domestic systems, among others. Furthermore, sensor control and integration are critical components of a robot's ability to interface with its environment and perform its intended applications desirably. However, most autonomous systems depending on LiDAR or visual based technologies can be extremely expensive to implement. As a result, the following design incorporates a real time localization system using an affordable YDLIDAR_X2 sensor interfaced with an inexpensive ESP32 microcontroller to develop an autonomous driving vehicle capable of managing distances and recognizing landmark features using the laser's odometer to give the feedback to the user in real time using wireless fidelity connection and a Python 3.9 Graphical User Interface.

In addition, while implementation of LiDAR technologies for domestic autonomous robots exists, most of these projects depend on more complex and/or expensive microprocessors or utilize ROS software development to interpret data. Alternatively, this design contained in this report offers flexibility in its implementation by using MicroPython environment.

Scope

BALDUR project will be developed as an original integrated system design for an autonomous robotic vehicle that does not seek to offer any new technologies but instead aims to provide an alternative cost-effective solution for the integration of LiDAR sensing technologies and autonomous systems.

Technical Objectives

- Interface ESP32 microcontroller with YDLIDAR X2 sensor
- Establish Wi-fi communication between the central CPU as the server and main ESP32 microcontroller.
- Establish UART serial communication between two ESP32 microcontrollers
- Create a system of obstacle avoidance using LIDAR and ultrasonic technologies
- Create two-dimensional modeling of indoor spaces using Python 3.9 in real time
- Utilize a secondary ESP32 microcontroller to accurately move a chassis to alternate the direction of current flowing through four different DC motors.
- Utilize I2S and an AB amplifier to interface WAV files with a speaker.
- Create wireless power transfer module to charge the robot's power bank.
    - Interface power transfer transmitter with an analog buck converter to supply constant voltage to the power bank.

- Interface multiple voltage levels and supplies for the robot's various loads.
- Use WS2812 communication protocol to control RBG Neo Pixel lights.
- Create light detection circuit

Technology Selection

To accomplish an effective system of simultaneous mapping and localization multiple sensors must be implemented due to the drawbacks of using a single sensor. For instance, SLAM based robots using only vision-based sensors such as cameras can have difficulty navigating in environments with poor lighting as distinct visual features become scarcer. And robots using only LiDAR technologies are limited by motion distortions in which LiDAR readings may fail when range measurements are received at different times during continuous LiDAR rotations. Therefore, the combination of two or more sensing technologies will provide the most accurate and stable system of simultaneous localization and mapping.

Sensor selection

Some of the available sensors for autonomous vehicles include: LIDAR, infra-red, ultrasonic sensor, visual based (camera), GPS RF detection, Millimeter wavelength radar, among others. However, for this design only LiDAR and ultrasonic will be implemented based on the arguments that both sensors are inexpensive and readily available, while the infra-red sensors and cameras may fail at different light exposures. Furthermore, GPS RF detection and millimeter wavelength can be extremely costly to implement.

Microprocessor selection

Some of the available micro processing devices considered include: Raspberry Pi Pico, Raspberry PI, Nvidia Jetson Nano, Teensy 4.0, STM32, ATmega128, PIC16F877A, TI MSP430G245, ESP32.

The main microcontroller options can be summarized as follows:

Raspberry Pi Pico	**Raspberry PI**	**Nvidia Jetson Nano**	**ESP32**
✓ GPIO Ports	✓ GPIO Ports	✓ GPIO Ports	✓ GPIO Ports
✓ PWM outputs	✓ PWM outputs	✓ PWM outputs	✓ PWM outputs
✓ ADC inputs	✓ ADC inputs	✓ ADC inputs	✓ ADC inputs
✓ SPI	✓ SPI	✓ SPI	✓ SPI
✓ Speed > 100MHz	✓ Speed > 100MHz	✓ Speed > 100MHz	✓ Speed > 100MHz
✗ I2S	✓ I2S	✓ I2S	✓ I2S
✗ Wi-Fi	✗ Wi-Fi	✗ Wi-Fi	✓ Wi-Fi
✓ Dual core	✓ Dual Core	✓ Dual Core	✓ Dual Core
✓ Inexpensive	✗ Inexpensive	✗ Inexpensive	✓ Inexpensive
✓ Low power	✗ Low power	✗ Low Power	✓ Low Power

Table 1 Comparison of various microcontrollers

Moreover, from Table 1it is shown that the ESP32 microcontroller can be programmed using MicroPython, Arduino C, or C++, which allow relatively simple access coding techniques to interface the YDLIDAR X2 via UART. In addition, the ESP32 can be easily programmed, includes Wi-Fi and Bluetooth modules, as well as sufficient GPIO ports, I2S communication for audio output, PWM outputs to drive DC motors, integrated ADC converters, dual cores, and a fast clock of 160MHz for considerably affordable price. On another hand, Raspberry PI and

Jetson Nano could offer the added complexity needed; however, these microcontrollers are very costly.

Operating System selection

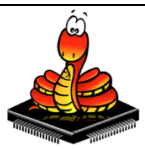
  **Python and OS**	  **Ubuntu and ROS**
Pros: ○ User friendly interface and previous experience in coding these environments ○ Available LIDAR repositories ○ More versatile environment to create user interfaces ○ Compatible with IOS **Cons:** ○ Application and interface must be developed from scratch. ○ Limited flexibility with some microcontrollers	**Pros:** ○ Open-source software ○ Most features for LIDAR interface have been developed ○ Common software in robotics and automation. **Cons:** ○ No previous experience in these environments ○ Requires access bootcamp of personal computers.

Table 2 Comparison of various Operating Systems

As a result of the former analysis, the ESP32 microcontrollers will be programmed using MicroPython. Additionally, Python 3.9 in Mac OS software will be implemented and libraries such as Pygame and Web socket suitable for two-dimensional modeling and wireless

communication will be employed. Moreover, advanced machine learning capabilities for implementation of SLAM algorithm are also available in Python 3.9.

Communication Protocol Selection

- IEEE 802.11 2.4GHz Wireless communication must be used between the ESP32 controllers and the CPU module. As a result, Wi-Fi will be implemented since Bluetooth and Radio Frequency communication can limit the communication that can occur between devices simultaneously.
- I2S protocol for audio output. The speaker will be driven from an I2S AB amplifier since is best suited for communicating digital audio devices. Contrastingly, robustness of SPI or other serial protocols makes them less desirable for clean DAC for audio output.
- UART will be used for communication between both ESP32 microcontrollers since integers and strings variables must be transmitted between them, in addition UART uses a minimal number of wires.
- WS2812 communication will be implemented between the RBG LED strip and the ESP32 microcontroller since it is a serial protocol of communication that requires the least number of shared signals between two devices and allow easy interface between the Neo Pixel LED strip and the ESP32 microcontroller.
- Parallel digital communication will be used between the Raspberry PI Pico and the secondary ESP32 microcontroller since only one signal is required to be communicated.
- I2C will be used to interface with the external MPU9250 accelerometer.

Components and Circuitries

BALDUR contains multiple circuits and power sources to allow a smooth integration of multiple systems, they are outlined as follows and the Hardware Diagram is shown in Figure 2.

- Power Supply
    - Five 1.5 AA batteries to power DC motors
    - 5V, 2000mA, two-port USB Power bank to energize microcontrollers and sensors.
- Robot movement
    - 4 6VDC motors connected to chassis
    - 2 H-bridge circuits using NMOS and PMOS FETs
    - ESP32 controller for PWM signal emission.
    - I2C serial communication. With MPU-9250 accelerometer
- Audio
    - ESP32 containing audio WAV files
    - I2S DAC and AB amplifier module for ESP32
- LiDAR interface
    - YDLIAR X2 sensor
    - ESP32 microcontroller to receive hex data from LiDAR through UART
    - Wi-Fi Router to create wireless server-client communication between ESP32 controller and main CPU
    - LCD screens for user interaction as well as GUI on main CPU
- Ultrasonic sensors
    - Raspberry Pi Pico for PWM emission

    - AB amplifier and speaker for audio output
- Light sensing circuit
    - Operational amplifier to create a light sensing detecting circuit
    - Photoresistor
    - RBG LED light to output light corresponding to input light intensity
- Wireless power transmitter
    - Two identical coils and capacitors to couple energy transfer
    - 555 Timer for PWM emission
    - H-bridge circuitry using PMOS and NMOS FETs to output AC signal through coil
    - CMOS Dual type flip flop to drive H-bridge
- Wireless power receiver
    - Diode rectifier bridge
    - 555 Timer for PWM generation
    - Buck converter
        - NMOS to drive buck conversion
        - 100mH inductor
        - Zener diode
        - Resistors
        - Coupling and bypass capacitors

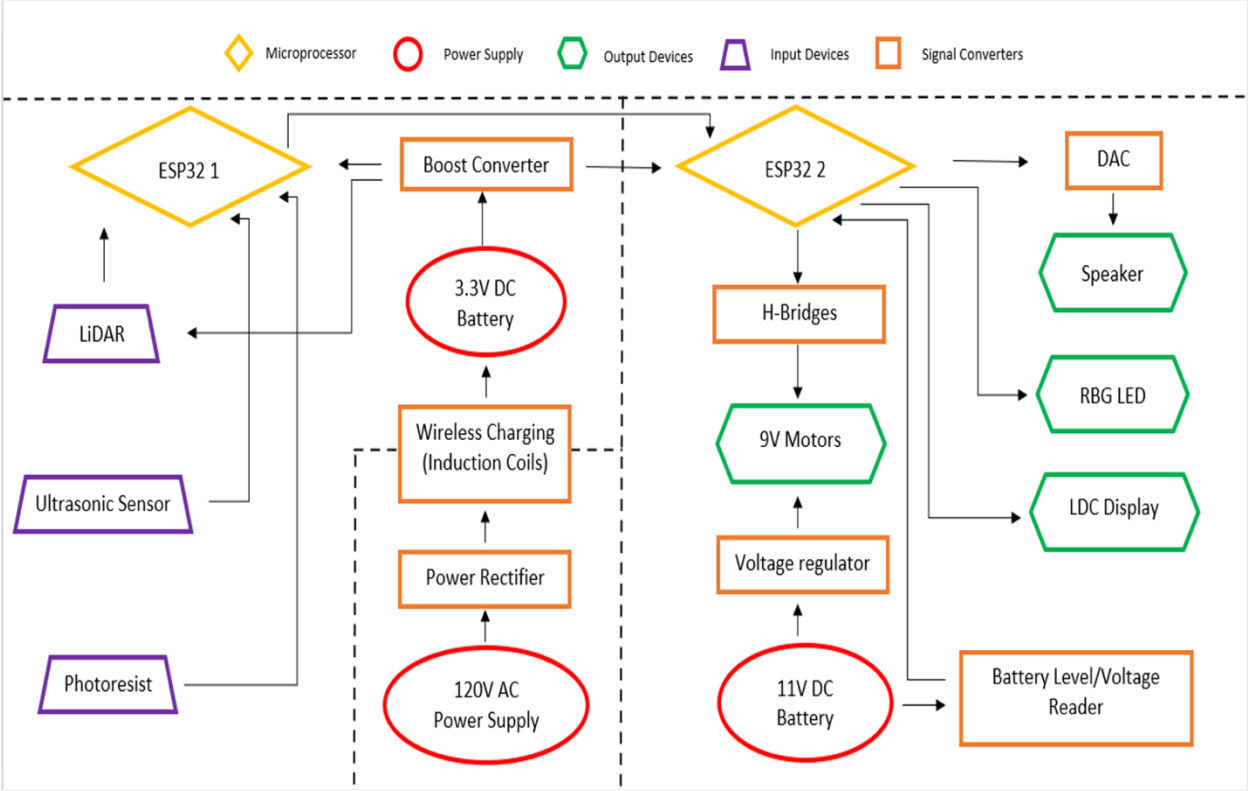


Figure 3 Hardware Block Diagram

System Process

Overall, BALDUR will have the following a system functionality as described by the following sequential steps and the Software Block Diagram shown in Figure 3.

1. Initially, the main CPU containing the GUI is started with the purpose of interacting with the user prior to the initialization of BALDUR. The GUI offers the user the option to start the robot automatic or manual mode. The former initializes a robot that is controlled autonomously while the latter gives the user control of the robot while maintaining the mapping features unaffected by the mode of operation.
2. Next, the GUI initializes the main CPU as a server on the local network created by an internet router under the IEEE 802.11 Wi-Fi protocols for 2.4GHz connection on the IP address 192.168.1.189.

3. Consequently, BALDUR can be turned on by plugging in the USB cable of the ESP32 microcontroller into the on-board power bank. Furthermore, upon starting, BALDUR will connect to the Wi-Fi connection as a client under the same Wi-Fi communication protocol as the server to establish a secure connection.
4. Once connected, BALDUR will initialize systems by testing the LCD screens, reproducing an audio WAV file, and emitting light using the RBG LED.
5. Lastly, the LiDAR sensor is initialized and with each rotation and "infinite" stream of hex data is sent to the main ESP32 controller via UART communication. This data is processed according to the YDLIDAR X2 Development Manual protocol to obtain a dictionary containing corresponding angles and distances of nearby objects according to the laser's odometer. Lastly, the dictionary is sent to the main CPU server via Wi-Fi.
6. Once the GUI receives data it is processed to obtain the angle corresponding to the biggest distance to dictate the robot's next direction of motion. In addition, the data is processed via a basic SLAM algorithm to display points in the GUI visualizer as small rectangles to recreate walls and features of the surrounding environment of BALDUR. The CPU sends back the corresponding angle for the robot to move next.
7. The main ESP32 controller receives the next angle of motion and communicates this data via a secondary UART channel to the secondary ESP32 microcontroller which processes the angle of motion into a specified time vector use to drive the PWM signals of the 4-DC motors.
8. The DC motors receive PWM signals and propel BALDUR into the desired direction while independently the ultrasonic sensor being driven by the Raspberry Pi Pico searches

for nearby objects below the LiDAR's field of vision and will send a stop signal to the secondary ESP32 in case of obstacles to avoid collision.

9. The second core of the main ESP32 controller drives the LCD screen with updates about angle of motion and sets the light intensity of the RBG light corresponding to the light intensity received by an ADC pin that changes its input according to the conductivity of a photoresistor located near the LiDAR sensor. Similarly, the secondary core of the secondary ESP32 is used to output desired audio files to the speaker and manage the secondary LCD screen.
10. The process repeats again once the robot arrives to the specified location as dictated by the main CPU.

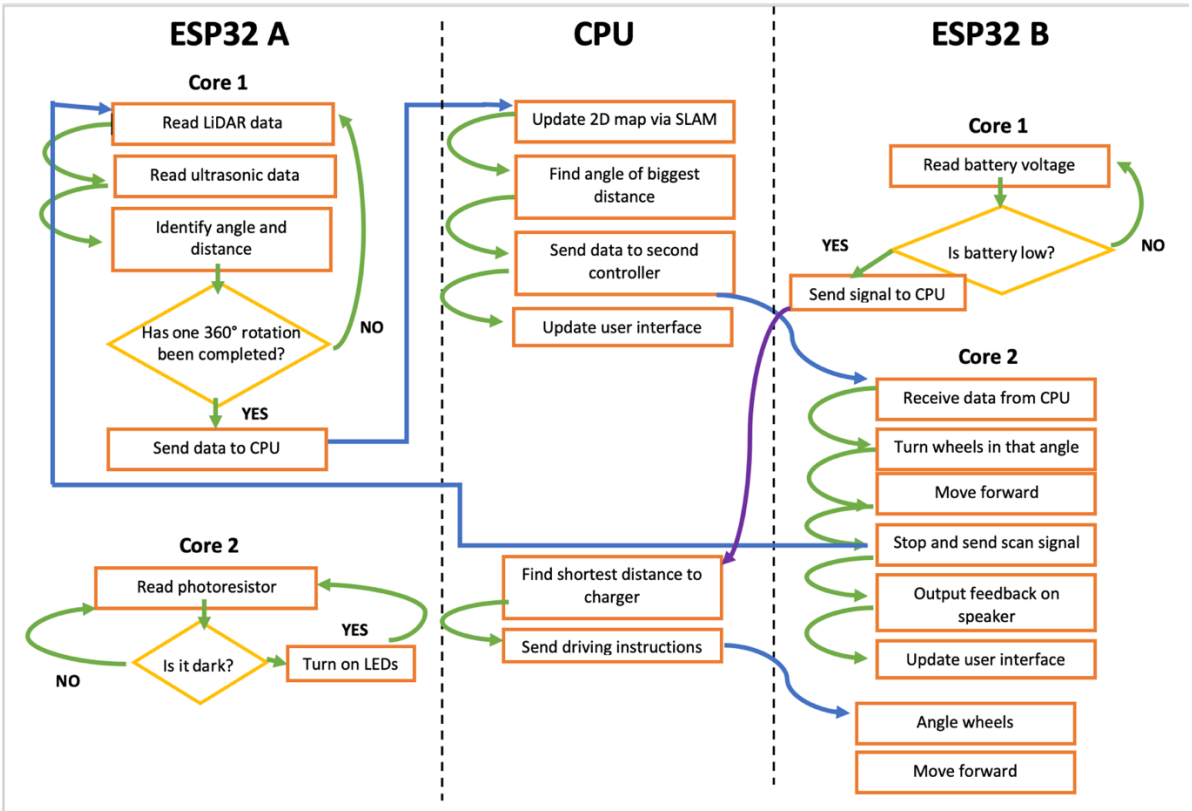


Figure 4 Software Block Diagram for BALDUR

Timeline

The following Gantt chart outlines the desired timeline for project design completion:

ID	Task Mode	Task Name	Duration	Start	Finish
1		Design 2 Project	78 days	Wed 8/24/22	Fri 12/9/22
2		Brainstorm	6 days	Wed 8/24/22	Wed 8/31/22
3		Ordering parts	5 days	Sun 8/28/22	Thu 9/1/22
4		Abstract	3 days	Wed 8/31/22	Fri 9/2/22
5		Presentation	6 days	Wed 8/31/22	Wed 9/7/22
6		Altium designs	17 days	Thu 9/8/22	Fri 9/30/22
7		Testing circuits	12 days	Fri 9/9/22	Mon 9/26/22
8		Ordering First boards	6 days	Fri 9/30/22	Fri 10/7/22
9		Testing comunications between componets	10 days	Mon 10/3/22	Fri 10/14/22
10		Reciving boards	4 days	Fri 10/14/22	Wed 10/19/2
11		Testing board	10 days	Mon 10/17/2	Fri 10/28/22
12		Trouble shouting and re-designing	10 days	Mon 10/31/22	Fri 11/11/22
13		Ordering second Board	5 days	Fri 11/11/22	Thu 11/17/22
14		Reciving second boards	5 days	Wed 11/23/22	Tue 11/29/22
15		Testing and building	6 days	Thu 12/1/22	Thu 12/8/22
16		Demo and compleation	3 days	Wed 12/7/22	Fri 12/9/22

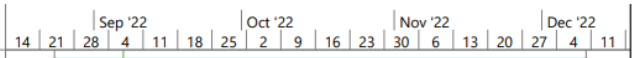


Figure 5 Gantt Chart

Responsibilities

The following outlines the responsibilities of each team member:

Luis Cedeño

- Movement:

  Design and codding of all the motor drivers for the movement of the robot

  - H-bridge

    Needed to give power to the motors and allow invert rotation

  - Signal divers

PWM signals and circuits that allow the right voltage to go through the MOSFETS to open them and close them the in their entirety.

- Power distribution

  Design and processing of different voltages in the same circuit

- Proximity alarm / safety control
  - Ultrasonic

    Setting up the ultrasonics and stablishing the parameters for the needed operations

  - AB amplifier

    Needed for the audio output of the DAC to increase the signal

  - DAC decoding

    Decoding hex code into a specific sound

- Entertainment and display
  - LCD screen

    Needed to display the status of the Communication between PCBs and current song playing

  - Memory expansion

    A microSD reader to increase the memory of the ESP32 to store several wav files for the music.

  - I2S configuration

    I2S system to control and amplify the audio been read of the wav files

Juan Pablo Orozco

- Interface YDLIDAR X2 with ESP32 microcontroller
- Create Wireless connectivity between ESP32 as a client and the main CPU as a server
- Interface Neo pixel light with ESP32 Microcontroller
- Interface and configure MPU9250 accelerometer with ESP32 Microcontroller
- Create light detection circuit using a photoresistor and an instrumentation amplifier
- Develop GUI and SLAM algorithm in main CPU using Python 3.9 to analyze LiDAR data.
- Create UART communication protocol between two ESP32 microcontrollers
- Interface 4x20 LCD screen with main ESP32 microcontroller
- Create main ESP32 program
- Develop wireless power transmitter circuit using NE555 Timer analog PWM oscillations in the inverter
- Develop Buck Converter and power receiving circuit for wireless power transmission.

Material List

System	Part	Quantity	Price	Total Cost
Microprocessors	Espressif ESP32 Dev/Kit	2	9	18
	Raspberry Pi Pico	1	$4.00	$4.00
Lighting	Adafruit NeoPixel Stick RBG LED	1	$5.95	$5.95
Audio	4Ω Speaker	2	$3.95	$7.90
	I2S/DAC converter MAX98307	1	$5.95	$5.95
	MicroSD breakout board	1	$7.50	$7.50
	MicroSD card	1	$8.08	$8.08
Range Sensors	YDLIDAR X2	1	$69.99	$69.99
	Ultrasonic Sensor HC-SR04	1	$3.50	$3.50
Light sensor	LTC1632 -Op/amp	1	$5.35	$5.35
	Photoresistor	1	$1.50	$1.50
	22k resistor	2	$0.15	$0.30
	1k resistor	1	$0.15	$0.15
	2.2k resistor	1	$0.15	$0.15
	680Ω resistor	1	$0.15	$0.15
	1uF Capacitor	1	$0.40	$0.40
	100nF Capacitor	1	$0.40	$0.40
Movement	Chassis and 4 DC Motors DIY amazon	1	$18.99	$18.99
	FQP27P06 PMOS	4	$1.57	$6.28
	IRLZ44N NMOS	4	$1.58	$6.32
	LM311 Comparator	2	$0.58	$1.16
Power Supply	1.5 AA batteries	6	$1.05	$6.30
	Power Bank 25800mAH	1	$29.99	$29.99
	2A/3A Fuses	2	$1.17	$2.34
	1000uF Capacitor	5	$0.55	$2.75
	100nF Capacitor	5	$0.40	$2.00
Wireless Transmitter	NE555 Timer	1	$0.44	$0.44
	FQP27P06 PMOS	2	$1.57	$3.14
	IRLZ44N NMOS	2	$1.58	$3.16
	AWCCA 24uH Coil	1	$8.30	$8.30
	0.1uF Capacitor surface mount	4	$0.89	$3.56

	IDX609 PI Gate Driver	2	$0.30	$0.60
	CD4013 CMOS Flip-Flop	1	$1.85	$1.85
	47Ω resistor	2	$0.10	$0.20
	1kΩ resistor	2	$0.10	$0.20
			Total	236.85

Appendix A: Problems / Solutions / Issues / Design Changes

LiDAR change from YDLIDAR X4 to YDLIDAR X2

The main feature of BALDUR consists of its ability to detect features based on LiDAR data. As a result, interfacing the LiDAR sensor with the ESP32 microcontroller was a fundamental aspect of the project. Initially, it was attempted to use the YDLIDAR X4 sensor which has different modes of operation that can be controlled via UART communication. However, there were multiple issues with transmitting bytes from ESP32 microprocessor to the LiDAR X4 sensor. Data baud rate and information was revised using the DAD board analog sensor and voltage levels were verified as well as per the YDLIDAR X4 datasheet. Nonetheless, the YDLIDAR X4 was an older used model and failed to interface properly as the serial data stream did not match any of the required bytes highlighted by the datasheet documents.

Fortunately, the University of Florida faculty had a spare YDLIAR X2 sensor with no previous usage. The YDLIDAR X2 sensor is smaller and less robust than the X4 LiDAR sensor, for instance, the X4 sensor has two-way serial UART communication while the X2 sensor cannot receive data from the microcontroller and is constantly in a default mode of transmitting an infinite array of bytes that can be parsed to extract the information of the surrounding object's angles and corresponding distances. The data sheet was then used to construct a library in MicroPython coding environment that would be able to manage the PWM signal that dictates the

LiDAR's rotating speed and receive the infinite data stream of bytes that can be parsed, deconstructed, and analyzed to produce a two-dimensional array with angles and distances obtained from the YDLIDAR X2 sensor. Data was able to be verified by printing the array results on the terminal and comparing these measurements to controlled features in a testing environment.

Addition of CMOS-Dual D-Flip-Flop for Wireless Power Transmitter

The development of the wireless power transmitter charger was conceptualized to be completely analog to fulfill project requirements. In this manner, an inverter circuit had to be developed to convert DC power into a 100kHz sinusoidal signal that would correspond to the central frequency of oscillation of the selected capacitive-inductive circuit made for power transfer. To accomplish this, a PWM signal had to be generated and switched at opposite input-ends of an H-bridge circuit. Originally, two square oscillator circuits with similar characteristics were going to be built using astable multivibrator circuits using the TI-NE555 timer chip. However, it was found that the phase of the square signals was overlapping in too many undesired instances which would result in the activation of the symmetrically installed NMOS and PMOS FETs of the H-bridge circuit. In other words, overlapping PWM signals were shorting the entire circuit constantly and failing to reproduce the desired 100kHz signal.

To resolve the former issue, a single oscillator circuit was used to produce a 200kHz square wave signal that was then fed into CMOS D-flip flop from which two square wave signals at 50% duty cycle, 100kHz and 180° out of phase shift are obtained and fed into the H-bridge circuit for power transmission.

Addition of Zener diode to Buck-Boost converter circuit

The wireless power receiving circuit required the rectification of the AC received power transfer signal, low pass filtering stage and a buck converter circuit for 5V voltage regulation. The complexity of the circuit was exposed whenever the two wireless-transfer coils were not physically positioned correctly. Small offsets from the centered position of the coils would result in high voltage spikes of up to 20V with low current signals which could result in damaging consequences for the electronic load at the end of the buck converter circuit. As a result, a robust voltage regulation circuit at the final stage of the charging circuit needed to be implemented. The team opted for using a commercial power-boost circuit at the output for voltage regulation and to avoid exceeding the maximum voltage ratings of the boost converter a 9V rated Zener diode and 100uF parallel capacitor was added.

Replacement of MPU-6050 Accelerometer

Initially, BALDUR's next direction of movement would be selected based on the LiDAR's angle data. However, there is no access to the LiDAR's X2 on board accelerometer, for this reason, to match the angles seen by the LiDAR sensor an external accelerometer was designed to be interfaced using I2C serial communication to obtain the corresponding robot's orientation. However, after the installation of the MPU-650 it was discovered that the MPU6050 lacked a magnetometer sensor which makes it impossible to obtain accurate measurements of the vertical axis of rotation of the robot. A gyroscope and accelerometer can be used to keep track of the transverse and longitudinal axis of a body using the respective three-dimensional components. However, the vertical axis of rotation depends on the center of gravity of robot and this data can only be extracted with an onboard magnetometer sensor that can give feedback based off the changes of the gravitational vector components as it rotates from its center of mass.

For this reason, the MPU-6050 had to be replaced with the MPU-9250 which does contain a magnetometer and give reliable results when measuring the yaw angle.

Wireless Network Configuration & Connectivity Changes

BALDUR's design is aimed to realize heavy processing of LiDAR sensor data using a remote CPU communicated via Wi-Fi to the ESP32 microcontroller. The original design consisted of utilizing both ESP32 microcontrollers as clients connected to the CPU's server at a specified IP address. However, the ESP32 Dev/Kit 1 board that were used in BALDUR's design require the utilization of 5 GPIO ports to be assigned as an individual virtual port for Wi-Fi communication. This detail mentioned in the ESP32's datasheet was ignored during the initial stages of the project's design. Nonetheless, as the Wi-Fi configuration stage of the project development approached, the team realized that the secondary ESP32 microcontroller did not have sufficient ports to control all its peripherals and the PWM signals for the robot's motors as well Wi-Fi. For this reason, the team decided to connect only the main ESP32 microcontroller to the remote CPU server and relay the transmitted information form the CPU via UART communication to the secondary ESP32 microcontroller since UART communication required the least number of connections between the two-boards.

Power Supply Changes

The original design of BALDUR consisted of using Lithium Polymer (Li-Po) batteries to supply the ESP32 microcontrollers and a separate voltage supply using 9V or AA batteries to supply power to the DC motors. However, a single cell Lithium Polymer battery was not able to supply enough power for both ESP32 microcontrollers and its peripheral systems. This is due to the fact the LiDAR sensor has a momentary inrush current component of 1.2A and other

peripherals and systems for both ESP32 microcontrollers require 5V and at least 2.4A to adequately supply the robot. In addition, using two Li-Po cells in series would require changing the charging circuit of the batteries and accounting for voltage shifts in the batteries charging circuit which would have added unaccounted complexity to the robot's final design. Thus, the team opted for using a commercial power bank with two USB ports that could be charged with constant 5V input voltage and supply up to 3A of power continuously for extended periods of time. This decision was more costly than using Li-Po batteries but resolve issues of time management and power supply management that were fundamental for the finalized robot design.

Appendix B: Problems / Solutions / Issues / Design Changes

Issues were encountered with the H-bridges since the transistors weren't opening and closing all the way making a short by having the forward and reverse bias active at the same time. These issues took us some time to discover and to fix it we needed to implement a separate signal drivel method with a comparator circuit that allow the full voltage to go to the MOSFETS and still maintain the PWM signal needed to regulate the velocity of the motors.

These problems were consequently resolved by lowering the frequency of rthe PWM signals.

Results

The following section highlights the finalized design files for BALDUR's implementation.

Schematics

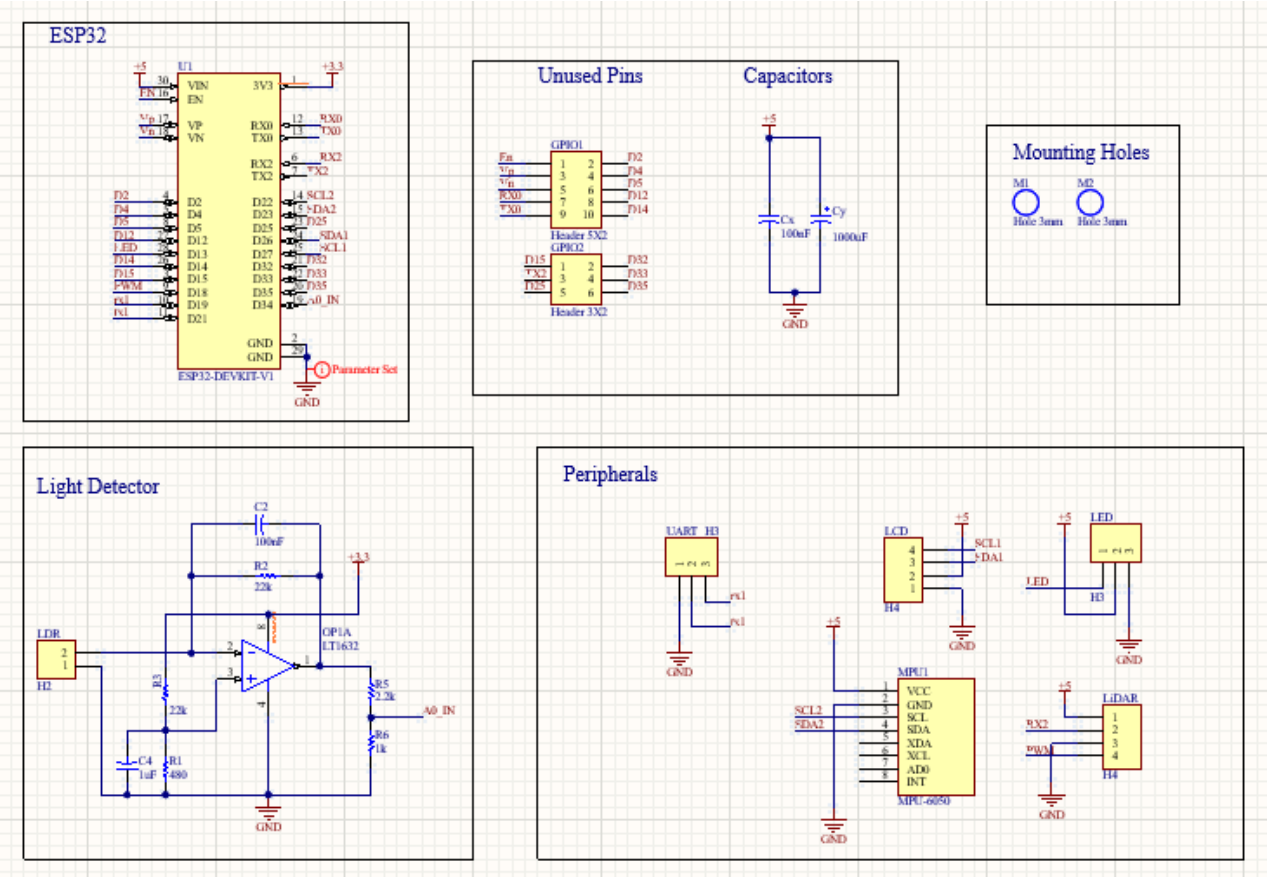


Figure 6 Main ESP32 Schematic

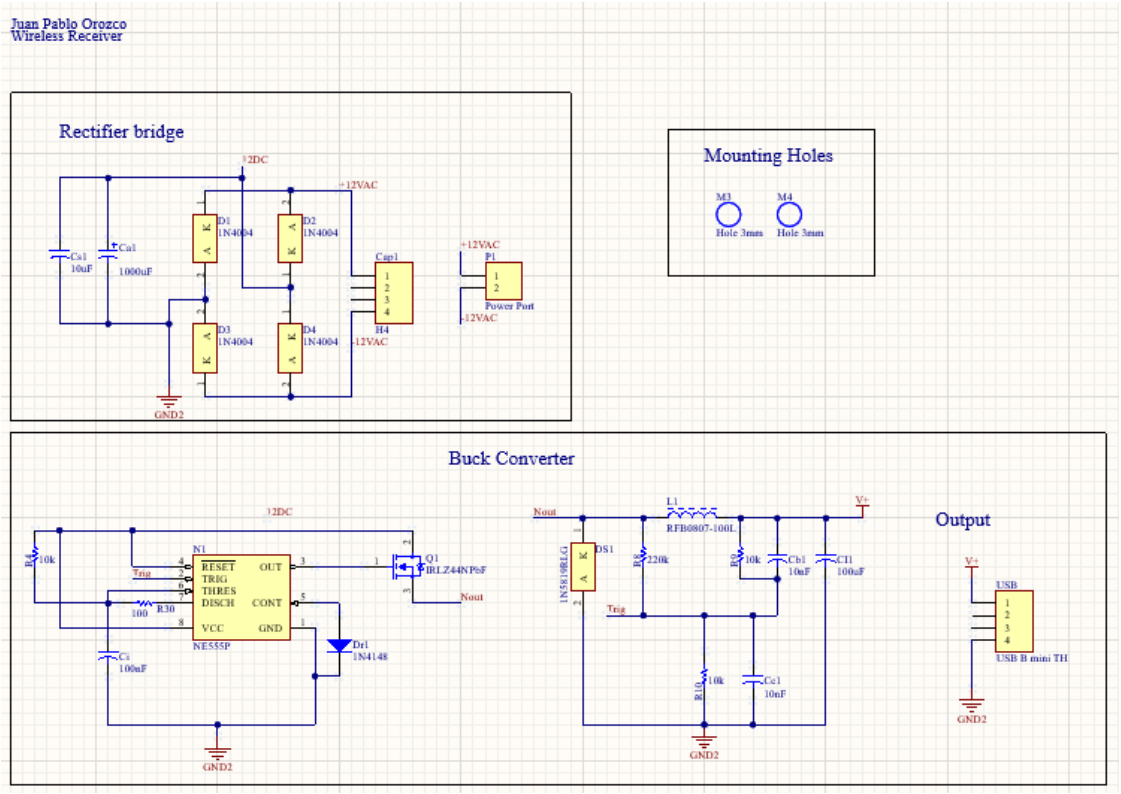


Figure 7 Wireless Receiver Schematic

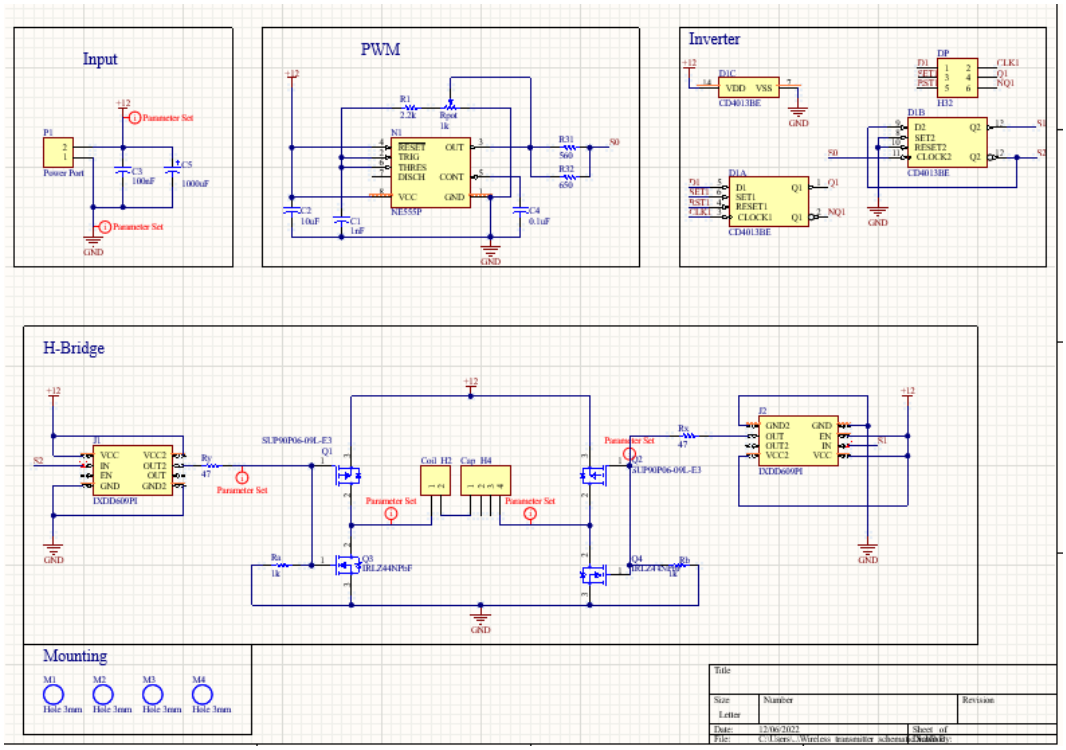


Figure 8 Wireless Charger Schematic

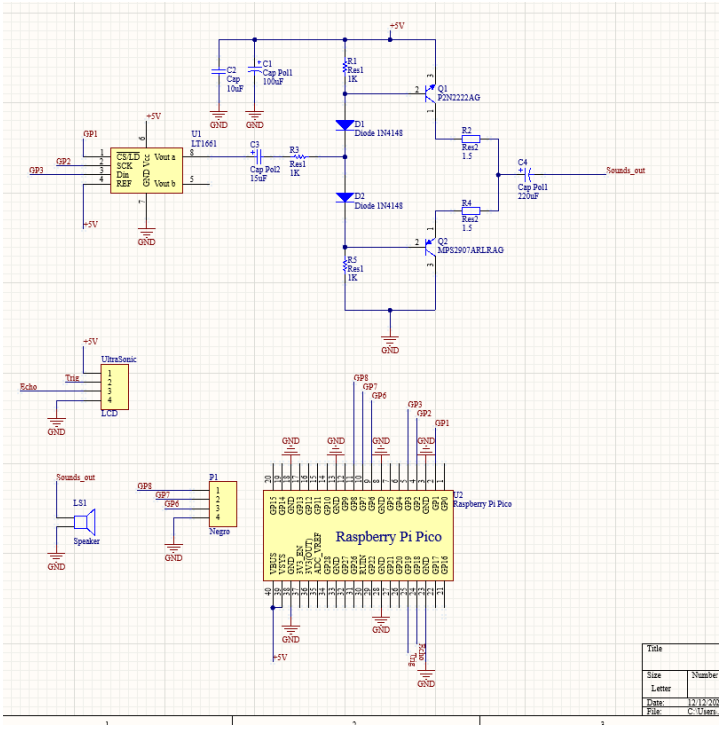


Figure 9 Ultrasonic AB Amplifier and Raspberry Pico Schematic

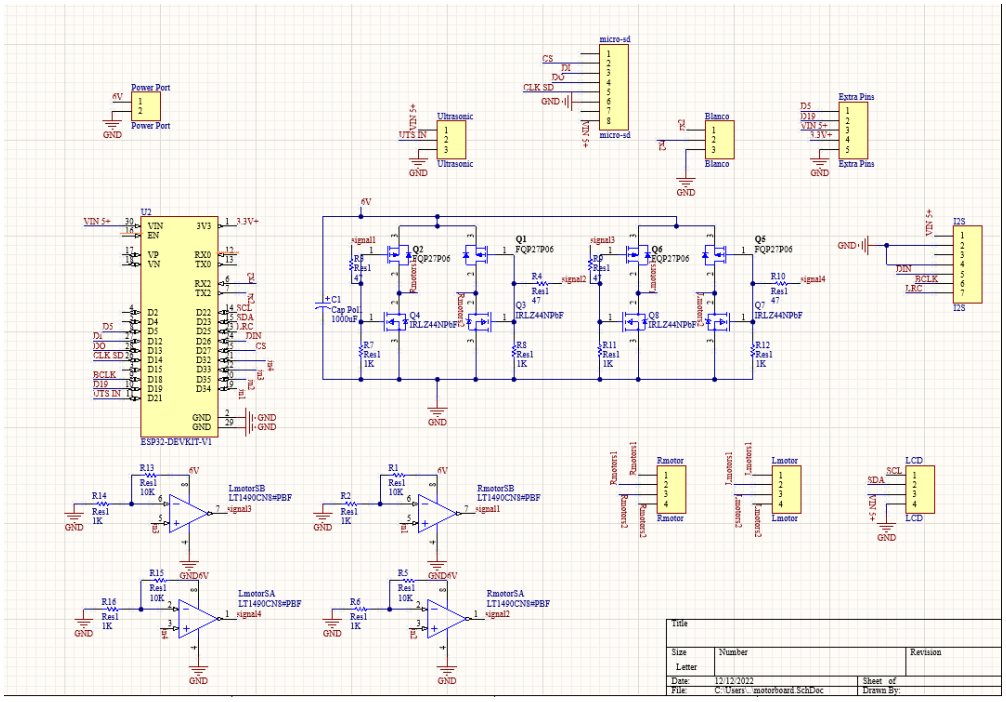
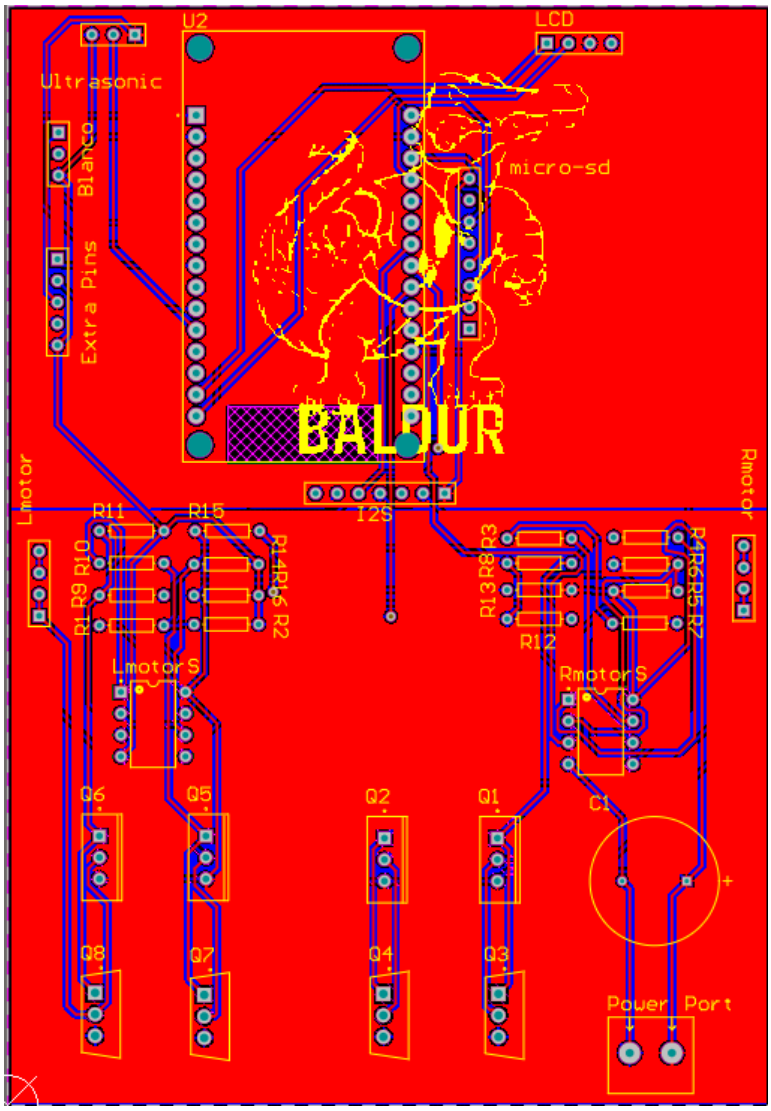


Figure 10 Secondary ESP32 Schematic

PCB design


U2
LCD
Ultrasonic
Blanco
Extra Pins
micro-sd
BALDUR
Lmotor
Rmotor
I2S
R11
R15
R10
R9
R14
R16
R2
R13
R8
R3
R12
R7
LmotorS
RmotorS
Q6
Q5
Q2
Q1
C1
Q8
Q7
Q4
Q3
Power Port

Finalized Robot

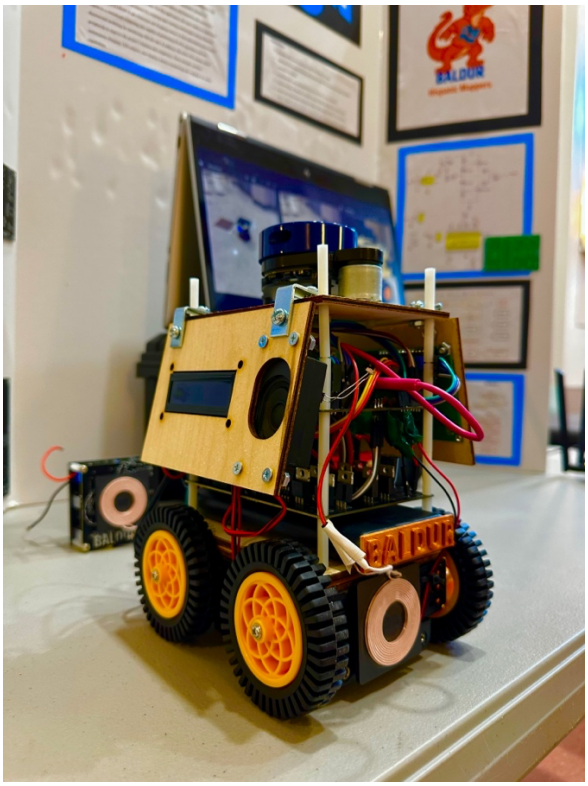


Figure 11 Back of BALDUR

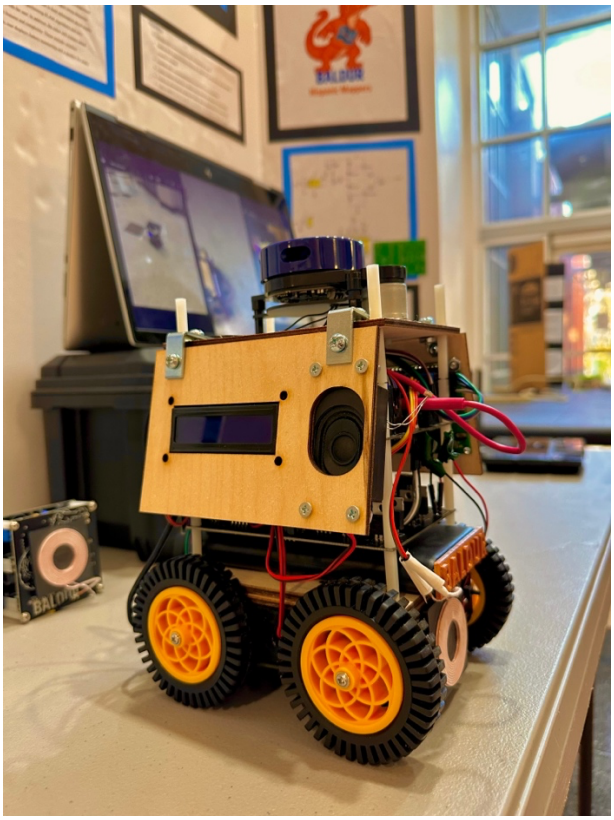


Figure 12 Final BALDUR design

Code

The main ESP32 file depending on additional original and commercial libraries was used as follows:

```
# Main BALDUR
# Juan Pablo Orozco
# Created: September 21 2022
# Last Updated: Dec 02 2022
#=================================================================
====================================
import _thread
from machine import Pin, UART
from time import sleep
from LiDAR_X2 import *
from Neopixel import *
from Light_Sensor import *
from WiFI import *
from LCD import *
from Motion import *


#================================================ Initialize
LCD ========================================
lcd_scl  = 27
lcd_sda = 26
rows = 4
col = 20
LCD = LCD( rows, col, lcd_scl, lcd_sda)
LCD.clear()
LCD.write(4,0,"Welcome")
LCD.write(0,1,"Scanning")
LCD.write(0,2, "WiFi networks")
sleep(0.5)
LCD.clear()
LCD.write(0,2, "please wait...")
#================================================ Initialize
WI-FI ====================================
LCD.clear()
LCD.write(1,0,"Connecting...")
client = wifi_client()

connect = client.connect_station()
LCD.clear()
LCD.write(0,1,"Connected to WiFi")
```

```
LCD.write(1,2, "Creating client...")
client.create_client()
client.send(client.CONNECT_MSG)
LCD.clear()
LCD.write(2,0, "Success!")
#================================================ Initialize
LiDAR ====================================
RXD2 = 16
TXD2 = 17
UART_channel = 2
M_CTR_pin = 18
LiDAR = YDLIDAR_X2(port = UART_channel,  tx = TXD2, rx = RXD2,
m_ctr = M_CTR_pin)
LCD.clear()
LCD.write(0,0, "LiDAR Initialized")
#================================================ Initialize
NeoPixel  ====================================
LCD.clear()
eye_pin = 13
pixels = 8
Eye = RBG_light(pixels, eye_pin)
LCD.write(1,0, "RBG Initialized")
#================================================ Initialize
GPIO ========================================
LED  = Pin(2, Pin.OUT)
light_sensor_pin = 34
light_sensor = Light_Sensor(light_sensor_pin)
LCD.clear()
LCD.write(1,0, "RBG Initialized")
#================================================ Initialize
Accelerometer ==============================
mpu_scl = 22
mpu_sda = 23
Motion_sensor = Motion_Sensor(mpu_sda, mpu_scl)
LCD.write(1,0, "MPU Initialized")
#================================================================
================================
#================================================ Initialize
ESP32 Communication ========================
uart_channel = 1
uart_baudrate = 128000
uart_rx =19
uart_tx = 21
LCD.clear()
negro = UART(uart_channel, baudrate = uart_baudrate, tx =
uart_tx, rx = uart_rx)
```

```
direction = 0
lock = _thread.allocate_lock()
#========================================= Local Functions
=========================================
def lightIntro():
    Eye.bounce(Eye.red, 0.2)
    Eye.turn_off()
    Eye.cycle(Eye.blue, 0.2)
    Eye.turn_off()
    Eye.rainbow()
    sleep(2)
    Eye.rainbow_cycle(0.01)
    Eye.turn_off()
    sleep(2)

def read_pitch():

    ax = Motion_sensor.get_acc_x()
    ay = Motion_sensor.get_acc_y()
    az = Motion_sensor.get_acc_z()
    pitch = Motion_sensor.get_pitch_deg(ax,ay,az)

    return pitch

def read_roll():

    ax = Motion_sensor.get_acc_x()
    ay = Motion_sensor.get_acc_y()
    az = Motion_sensor.get_acc_z()
    roll = Motion_sensor.get_roll_deg(ax,ay,az)

    return roll

def read_yaw():

    roll = read_roll()
    pitch = read_pitch()
    yaw = Motion_sensor.get_yaw_deg(magx,magy,magz,pitch,roll)

    ax = Motion_sensor.get_acc_x()
    ay = Motion_sensor.get_acc_y()
    az = Motion_sensor.get_acc_z()
    yaw2 = Motion_sensor.get_yaw2_deg(ax,ay,az)

    gz = Motion_sensor.get_g_z()
```

```
    return round(gz,2)

def send_LiDAR_data():

    LiDAR_points  = LiDAR.get_Data()
    if LiDAR_points is not  None:
        LiDAR_points_length_str = "Length: , " +
str(len(LiDAR_points))
        client.send(LiDAR_points_length_str)
        for key in range(len(LiDAR_points)):
            client.send(LiDAR_points[key])
            print("Sent " , LiDAR_points[key])
        return client.receive()

def illumination_thread():
    lightIntro()
    global direction
    while True:
        p = light_sensor.read()
        Eye.set_brightness(Eye.violet, p)
        brightness_str = "Brightness is: " + str(100-p) + "%"
        LCD.write(0,2,brightness_str)
        lock.acquire()
        direction_tmp = direction
        lock.release()
        direction_list = direction_tmp.split(',')
        angle_tmp = direction_list[0]
        angle_tmp_str = "Going to: " + str(angle_tmp) + " deg"
        LCD.write(0,3, angle_tmp_str)



#======================================== Main
============================================
# ============================================
============================================

_thread.start_new_thread(illumination_thread, ())
LCD.write(2,0, "Hi I'm Baldur!")
LCD.write(0,1,"Scanning...")
yaw = read_yaw()
while True:
    LED.value(1)
    lock.acquire()
    direction = send_LiDAR_data()
```

```
lock.release()
angle_str = str(direction)
negro.write(angle_str)
LED.value(0)
sleep(0.5)
```