Ultrasound Remote Operated Vehicle

ECE 445 - Spring 2025

Project #53
Gabriel Inojosa, Jamil Yeung, Ted Josephson
Professor: Michael Oelze
TA: Kaiwen Cao

o

Table of Contents

1. INTRODUCTION

PROBLEM

Wireless communications predominantly use electromagnetic waves as a means to communicate control and telemetry signals. However, in conductive media such as water, electromagnetic waves do not propagate well. As a result, much of the planet is inaccessible to remotely operated vehicles which communicate with their operators exclusively through electromagnetic waves.

This challenge is particularly posed towards all industries that require the use of submersibles, such as deep sea oceanography and the inspection of underwater structures. As a result, submersibles are either operated directly by a pilot, which poses a safety risk, or are operated through tethered communication. Startups such as OceanComm have explored ROVs that communicate acoustically with the controller, but these are very expensive.

SOLUTION

We intend to develop a proof of concept for a lower cost acoustically controlled ROV which operates in air, using cheap ultrasonic transducers designed for range finding.

We would like to develop a low-cost method of wireless communication using acoustics for remote control that will fit within the budget of ECE 445. For simplicity of the project, we will use the ECE 110 car as the mechanical basis of our design.

VISUAL AID

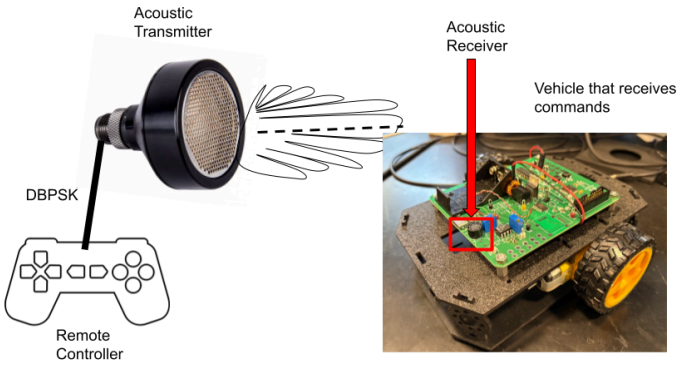


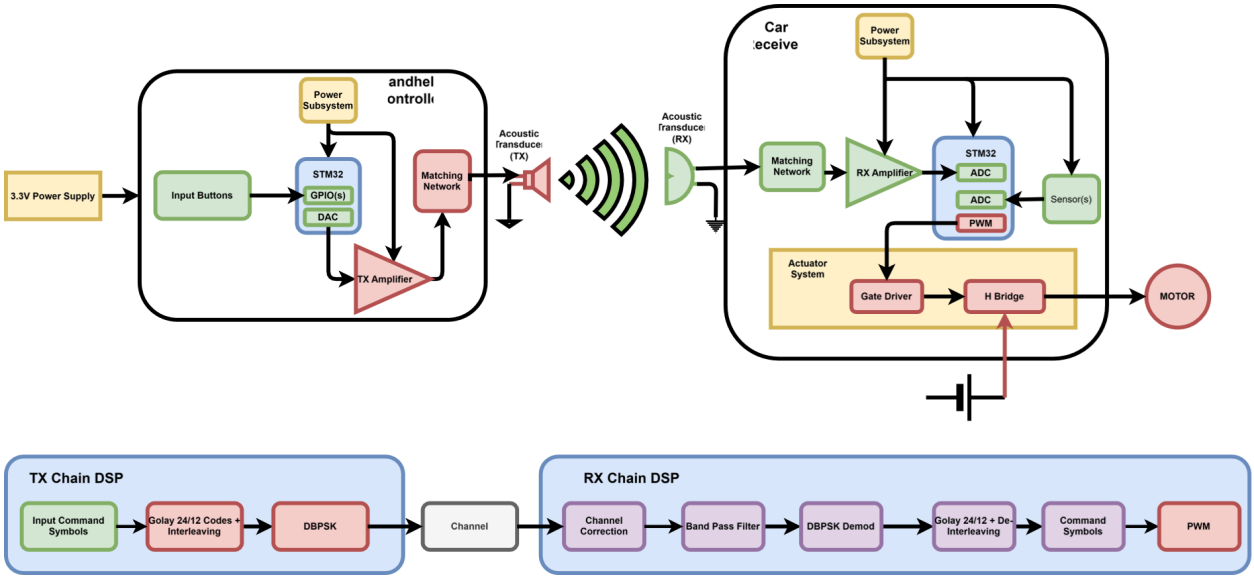
Figure 1: Visual aid of overall project

HIGH LEVEL REQUIREMENTS

- Reliable transmission of control signals over distances of at least 3 meters.
- The acoustic transmitter should be able to use differential binary phase shift keying (DBPSK) modulation at around the 40[KHz] range The resulting signal should have a bandwidth of 2 KHz.
- The vehicle should be able to demodulate and act from an instruction with a carrier SPL of 100 dB (0dB = 0.02 mPa).

2. DESIGN

Figure 2: Block Diagram of project



I. MICROCONTROLLER

The microcontroller on the transmitter will read the value of the input buttons using its GPIOs and generate throttle and steering commands to be transmitted to the car.

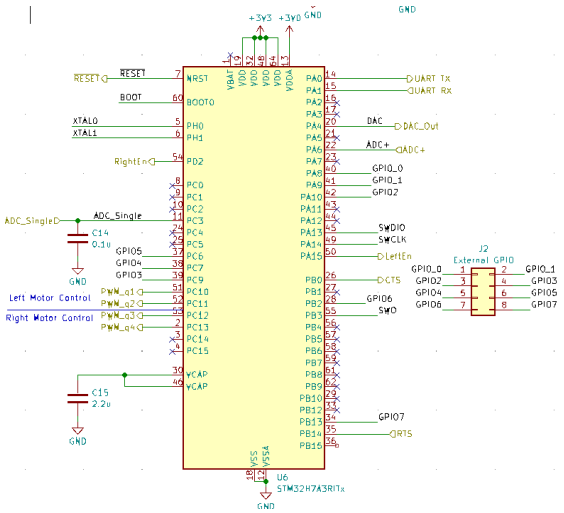


Figure 3: Schematic of STM32H7B3RITx Implemetation [3]

ANALOG REFERENCE CIRCUIT, ADC RESOLUTION

A 3.0 V clamping circuit is set to define VDDA on the microcontroller at 3.0 V in order to make resolution mathematically easier. The ADC and DAC for potentiometer sensing will both be 12 bits, therefore allowing the resolution scale to be:

$$\frac{V_{in}}{2^{12}} = \frac{3[V]}{4096[INT\ MAX]} = 0.7324 * 10^{-4}[\frac{V}{int}] = V_{min}$$

TRANSMITTER DSP

The STM32 will apply error control coding to the user input. It will then generate a differential BPSK signal centered at 40 KHz containing coded control signals. If the transmitter is on the car, it will apply error control coding to the telemetry data produced by its sensors. The data from each will be transmitted to the other using the Differential BPSK scheme described below.

To a 1 bit, the transmitter will send a carrier with the same phase as the carrier during the previous symbol time. For example if the previous symbol was:

$$s(t) = cos(\omega_c t)$$

then the current symbol will be:

$$s(t) = cos(\omega_c t)$$

To send a 0 bit, the transmitter will send a carrier with the opposite phase as the carrier during the previous symbol. For example, if the previous symbol was:

$$s(t) = -cos(\omega_c t)$$

then the current symbol will be:

$$s(t) = cos(\omega_c t)$$

The error control code is a 12/24 Golay code. The codeword is computed by the application of a generator polynomial. Our implementation is borrowed from LiquidDSP.

RECEIVER DSP

The ADC on the STM32 microcontroller will sample the incoming signal at 250 KHz. The sampled signal will then be band-pass filtered with a hard coded filter generated in python using the window method. The bits are recovered directly from the

filtered signal by delaying it and multiplying it with itself. The exact delay used is very important for the proper functioning of the receiver. It must be the closest integer multiple of the carrier period to the sample period. The delay d is computed using the following formula:

$$d = [\frac{F_c(SPS)}{F_s}][\frac{F_s}{F_c}]$$

where [x] denotes x rounded to the nearest integer.

The output of the demodulator can be expressed as

$$y(t) = s(t)s(t-d)$$

Whenever the input the receiver transitions from

$$s(t) = cos(\omega_c t) \qquad \text{to} \qquad s(t-d) = -cos(\omega_c t)$$

the output of the demodulator is

$$y(t) = -\frac{1}{2}(1 + cos(2\omega_c t))$$

and when the input does not transition, the output of the demodulator is

$$y(t) = \frac{1}{2}(1 + cos(2\omega_c t))$$

in both cases, the $cos(2\omega_c t)$ term is removed by the application of a low pass pulse shaping filter, leaving behind the +/- 1/2 term. This low pass filter is implemented as a cascaded integrator-comb (CIC), and has a rectangular impulse response. The following recurrence describes the CIC implementation:

$$y[n] = y[n-1] + x[n] - x[n-SPS]$$

where sps is the number of samples per symbol (250).

MICROCONTROLLER DSP/UNIT TEST PROCEDURE

The majority of the DSP unit testing was performed using a loopback configuration, where the DAC output is connected to the ADC input of the same microcontroller. This allows both the TX and TX DSP chains to be tested simultaneously with only one board. To test the TX and RX chains in isolation, the Analog Devices M2K module can be used to sample the DAC output, which is used with a simulated receiver in python.

II. AMPLIFIER, TRANSDUCER AND COMMUNICATION SYSTEM

A resistive network and Op-Amp or transistor will be used to amplify the signal from the DAC on the microcontroller. This amplifier has peak to peak output voltage of 9v from battery bias voltage. This will be limited by the maximum voltage swing of our amplifier, which is limited to the supply voltage. The 1 V headroom is fit for the TL082 amplifier. We will use the Murata MA40S4S/R transducers for their wide gain pattern and high sensitivity.

Recent tests performed by this group indicate that we can expect the output voltage of the MA40S4R to be ~ 50 mV when the transmitter is driven with 10v peak-to-peak sine wave at 40 kHz with the ADALM M2K. We will use the TL082 Op-Amp with a 1 [MΩ] negative feedback resistor to amplify the signal from the MA40S4R, which results in roughly 2V peak-to-peak output when tested as described.

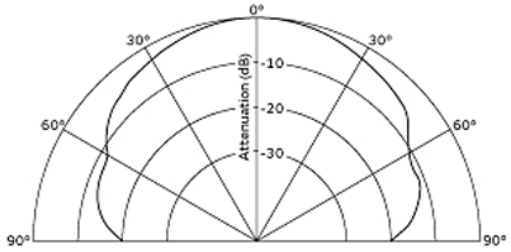


Figure 5: Directivity pattern of MA40S4S [1]

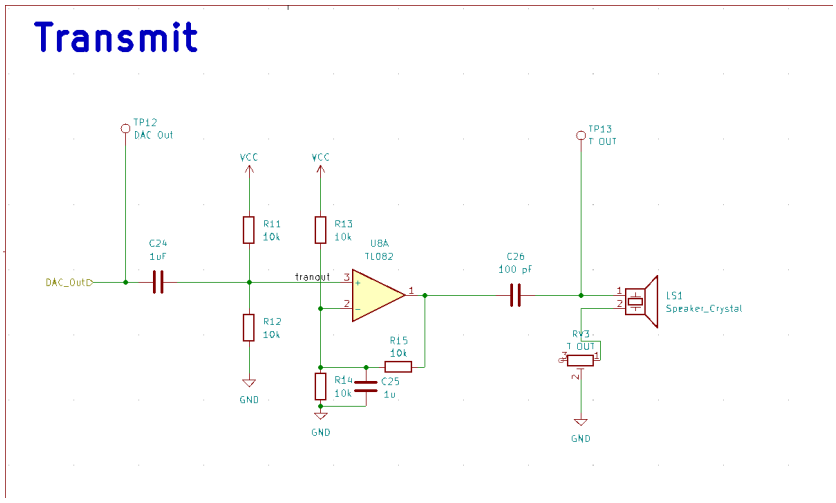


Figure 6: Schematic of Transmitter Circuit

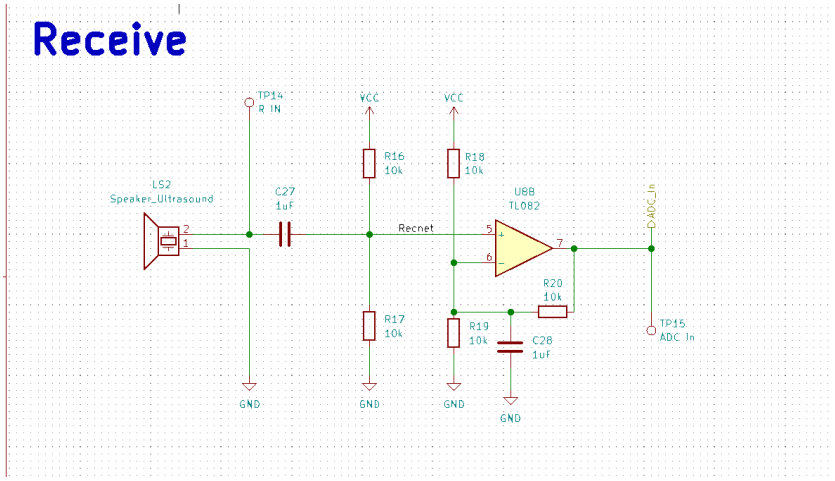


Figure 7: Schematic of Receiver Circuit [10]

Receive Amplifier Output

The receiver piezo is matched to an amplifier whose output is sampled by the ADC on the microcontroller

Unit Testing Procedure

The output of the receiver can be observed on an oscilloscope while the transmitter is being used as described above.

III. POWER DISTRIBUTION

A non-isolated buck DC-DC converter will be used to step down the input voltage of a 9V battery to power the STM32 at 3.3VDC. This will then be cascaded into the

In the following iteration of using the 9V to 0V rail, The negative bias for the MAX1044 charge pump [2] in the design document became unnecessary for the amplifiers. In order to mitigate the susceptibility of switching noise to the power rail, a choke is designed with a 12uH inductor with various capacitors placed in parallel.

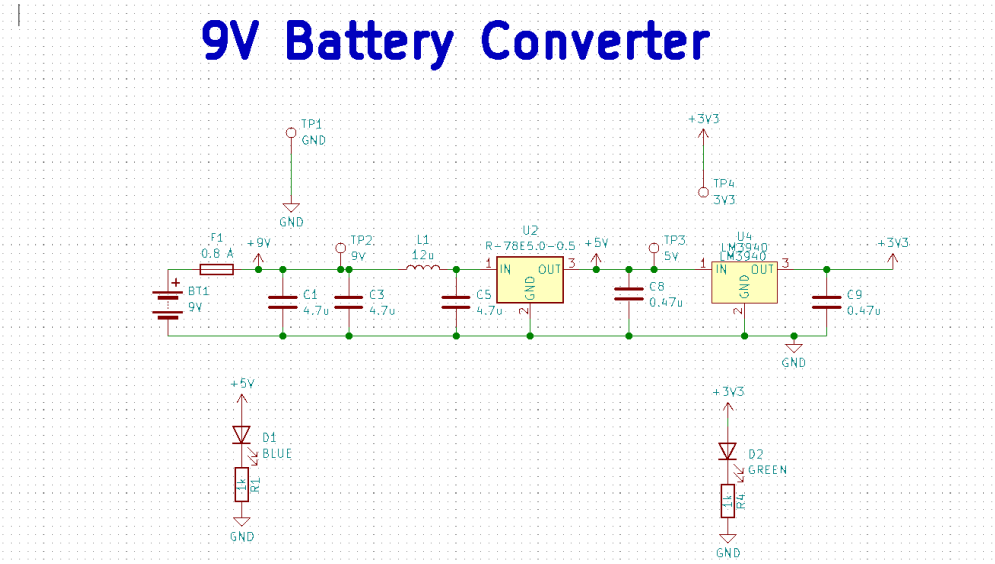
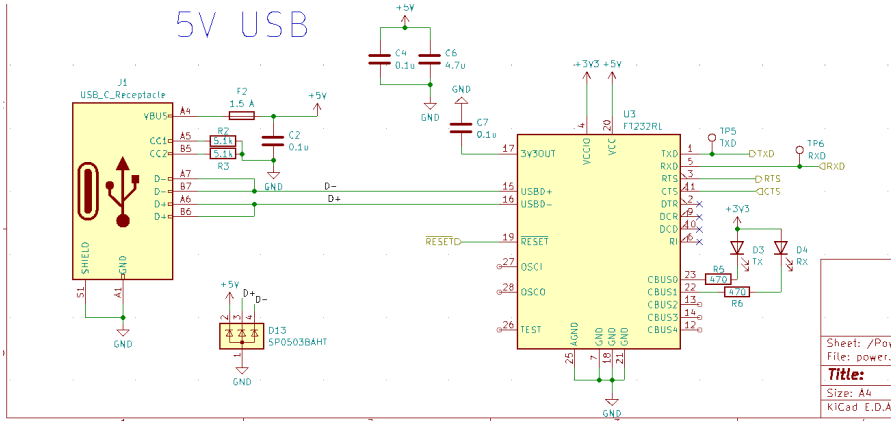


Figure 10 (Above): DC-DC Buck Converter Circuit cascaded with a 3v3 Linear Voltage Regulator.

Figure 11 (Below): USB C Receptacle Circuit with FT232RL RS232 Serial to USB module and schematic errata.



For debugging purposes, a USB-C port was designed with power delivery negotiation on the CC pins, transient voltage suppression, and connection to the FT232RL to send

IV. ACTUATOR

The DC Motors are driven using the L293 H-Bridge in normal operation with rectifier flyback diodes. The STM32 will drive the H-Bridge through logic latch control and PWM for motor angular velocity control.

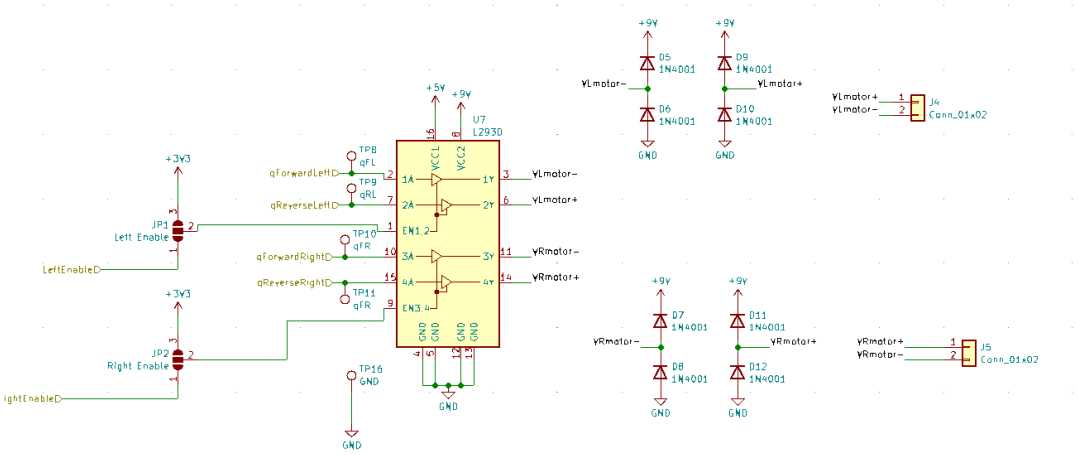


Figure 13: Motor Controls using the L293 H-Bridge integrated circuit with digital PWM control from the STM32H7.

The desired switching frequency for the H-Bridge does not consider the switch strain or sink current of the device, instead with consideration for the switching noise that may be picked up by the STM32 ADC for the receiver. In order to be effectively avoided from the ADC, we chose a switching frequency to be half of the carrier frequency of the digital communication

State	Definition	Motor Directions	STM32 Timer Channels Driven
0	Immobile	N/A	
1	Forward	Left CW, Right CCW	T1Ch2, T1Ch3
2	Reverse	Left CCW, Right CW	T1Ch1, T3Ch4
3	Left Pivot Forward	Left CW, Right Neutral	T1Ch3
4	Left Pivot Reverse	Left CCW, Right Neutral	T3Ch4
5	Right Pivot Forward	Left Neutral, Right CW	T1Ch1
6	Right Pivot Reverse	Left Neutral, Right CCW	T1Ch2

Table 1 (Above): List of states and H-bridge motor direction outputs.

STM32 PWM FIRMWARE

In order to program the STM32 to adjust the duty cycle and switching frequency for motor control PWM, the firmware must consider the CCR and the ARR. The CCR defines the threshold value of a digital comparator to an increasing counter to set a signal pin high or low. The ARR sets the maximum integer value the counter will reach before resetting. Given the STM32 has an internal 64 MHz clock, we determine our values for a 20 kHz switching frequency given.

$$f_{sw} = \frac{f_{clk}}{ARR + 1} \Rightarrow ARR = \frac{f_{clk}}{f_{sw}} - 1 = \frac{64E6}{20E3} - 1 = 3200 - 1 = 3199 = ARR$$

PWM is limited at maximum duty cycle of 95%, or, $CCR \in (0, 3039)$. The duty cycle is updated in the main loop of the STM32's firmware and passed by reference to the HAL_TIM_PWM_Start function.

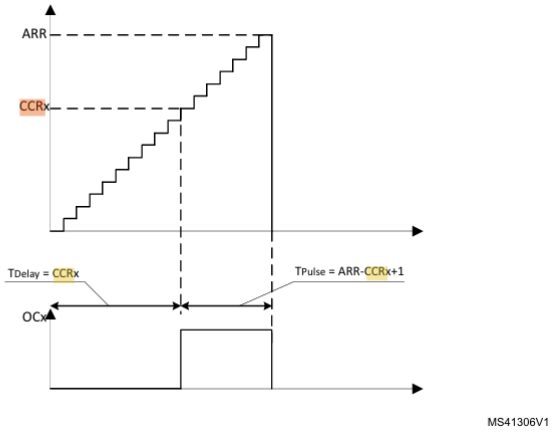


Figure 14 (above): Diagram of STM32 PWM generation through comparator register and output. [11]

V. LAYOUT

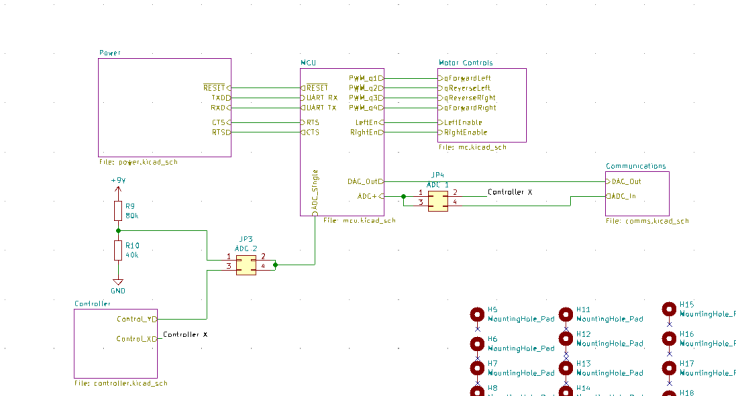


Figure 15: Main schematic of all system sheets in KiCAD

The PC Board is divided into 9 volt, 5 volt, and 3v3 net regions. The 9 volt region is the location of the battery and the H bridge, the 5 volt region is the region of the transducer elements and USB-C power and serial communication for debugging with UART. The 3v3 net is set for the microcontroller and other circuitry required for the ADC for sensing.

The PCB was designed with intended flexibility to be soldered as either the controller or the receiver of the commands. Changing the jumpers allows a quick change in the PCB's hardware to either work like a controller or a vehicle.

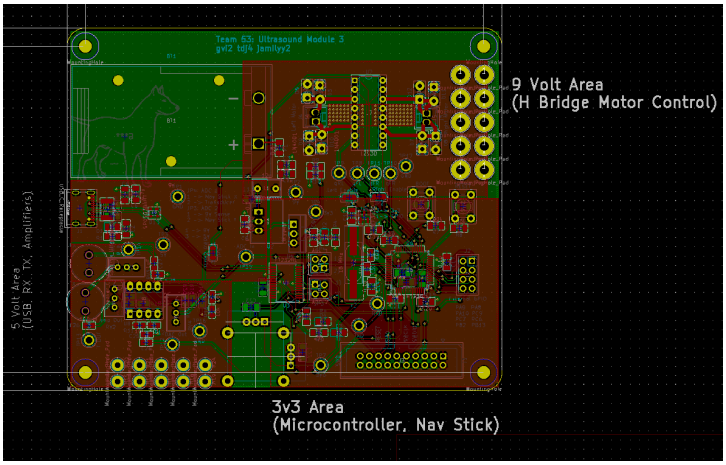


Figure 16 (above): Layout of the Fourth Round PCB order.

Further accommodations included eight additional STM32 pin ports to a header on the PCB. This is a failsafe for manufacturing errors in case that traces are stripped while soldering or features of pins are no longer supported.

In order to fit on the prints of the ECE 110 car, the dimensions of the PCB are 12[cm]x10[cm] with 3.5 [mm] diameter mounting holes tapped at 11[cm]x9[cm].

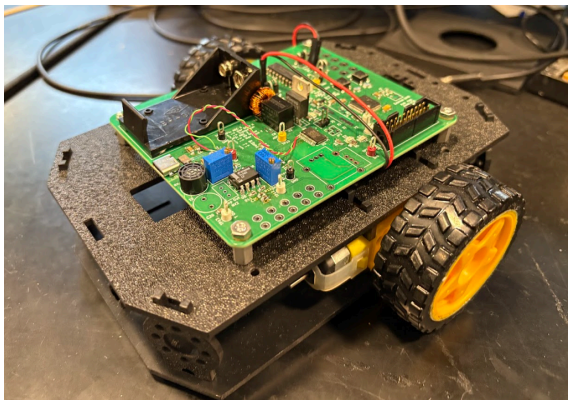


Figure 17 (above): The final PCB design fastened with spacers on the vehicle chassis.

3. VERIFICATION

I. MICROCONTROLLER

Acceptance Criteria	Testing	Criterion Met
STM32H7 flashes over SWD	Using the ST-Link connections and harness, wire to 3v3, GND, !RESET, SWCLK, SWDIO	Microcontroller on PCB is able to flash LED blinking firmware, confirmed by observing LED
STM32H7 shall output full functionality of PWM Outputs, ADC, and DAC implementation in PCB	Testpoints of STM32H7 pin outputs will be probed using the oscilloscope to observe for proper waveforms.	Not fully met. DAC and ADC was not tested on the STM32H7 PCB. PWM with timer interrupts weremet, however.

Table 2 (above): Verification criteria for the microcontroller implementation

II. AMPLIFIER, TRANSDUCER, COMMUNICATION SYSTEM

The full acoustic communication system should be able to achieve a bit error rate of less than 10^3 in poor to average channel conditions at a distance of 3 meters. The required SNR to achieve the specified error rate is difficult to compute for DBPSK. This is because it requires finding the cumulative distribution function of the product of normally distributed random variables. Instead, we can simulate the receiver at different SNR values.

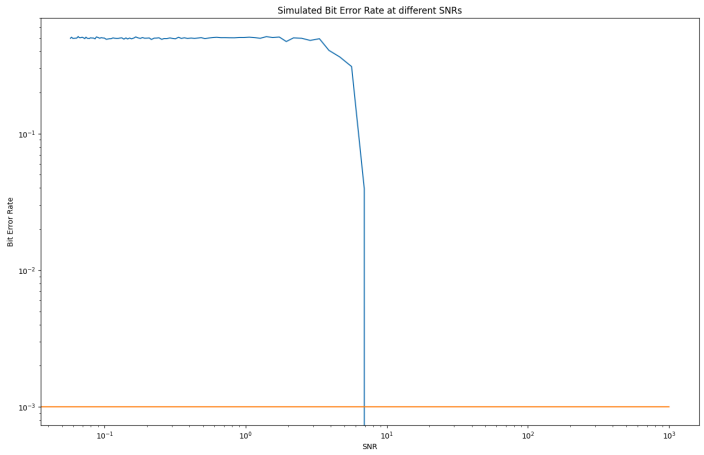


Figure 18: Simulated BER at various SNRs.

The receiver BER was measured using software on the microcontroller which generates a sequence of packets where the 12 bit payload is incremented by 1 from each packet to the next. The receiver computes the hamming distance between each received

packet and its corresponding expected packet, then sums all the hamming distances to find the total number of bit errors, which it can use to find the average error rate.

The raw data transmission rate bits per second. This requirement exists to ensure that 20 12-bit update packets can be sent per second, guaranteeing a latency of 50 ms, not including propagation delay, to ensure drivability. The latency can be measured by using an oscilloscope to plot the control inputs and the corresponding GPIO outputs, then comparing the delay between when the control input is applied and when there is a corresponding change in pin voltage. The data rate requirement can be verified using test patterns known by both the receiver and transmitter. The data rate test will be implemented in software on the microcontroller.

Acceptance Criteria	Testing	Results
BER of < 0.001	Software comparison of received bits with a test pattern. Software will continue to loop the test pattern until 30 bit errors are observed, or the test times out.	Required BER is obtained at distances of up to 3 meters only when the space around the receiver is clear enough to prevent multipath interference.
BER remains within limits when the relative speed between the transmitter and receiver falls between [0, 1] m/s.	Timer and meter stick are used to measure relative velocity.	Required BER is obtained when the transmitter moves away from the receiver at ~0.5 m/s.
Maximum Data Transmission Rate of > 640 error free bits per second	Transmission rate is fixed at 1000 bits per second and other functions are confirmed working	Software comparison of received bits with a test pattern at 3 meters, yields error-free transmission of 1000 bits per second

Table 2 (above): Verification standards and results of Wireless communication

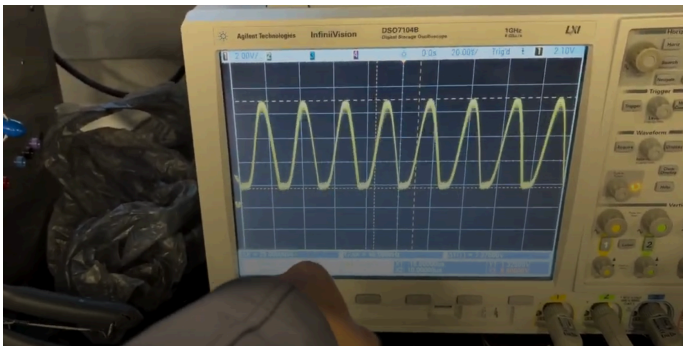


Figure 19: Oscilloscope results of receive amplifier circuit in figure 7 from 80mVp-p excitement signal

III. POWER DISTRIBUTION SYSTEM

Acceptance Criteria	Testing	Criterion Met
The circuit will receive 5V from the USB-C receptacle at a limit 0.5 A.	Probe the 5V terminal and ground with a multimeter	Yes, serial communication over RS232 and USB was also successful
The voltage of buck cascade shall have a peak-to-peak ripple voltage no greater than 90 millivolts at the STM32	This will be checked using an oscilloscope to probe 3v3 waveform	Yes. The STM32H7 on the PCB was able to be properly powered.

Table 4 (above): Verification standards and

IV. ACTUATOR

Acceptance Criteria	Testing	Criterion Met.
H bridge is able to vary the speed of the motor with the duty cycle.	Observe the motor RPM while adjusting the duty cycle of the PWM waveform. At a 0% duty cycle, the motor should not be moving	Yes the RPM of the motor changes with the change in PWM.
The duty cycle of the switching waveform will be limited at 90% under the control unit	There is a function that limits the PWM to a percentage of the waveform and can be tested by measuring the waveform.	Yes the waveform changes accurately to the change in percentage of the PWM.

Table 5: Acceptance Criteria and testing for the Actuator subsystem

4. COSTS

I. Bill Of Materials

Total Bill: $58.14

Manufacturer Part Number	Link	Price (USD)	Part Description	Amount For Project	Total Price of each part
MA40S4S	Mouser	$5.06	40 kHz Ultrasonic Transducer	4	20.24
STM32H7B3RIT6	Digikey	8.673	Microcontroller	2	17.346
R-78E5.0-0.5	LCSC	$2.58	9-5V DC-DC BUCK Converter	1	2.58
YF16-DFL7.2-B5K0(45-10)B5K0(55)-RG-A22	LCSC	$0.45	Joystick	1	0.45
12BH611-GR	Mouser	1.35	9V Battery Holder	1	1.35
MAX1044	Analog Devices	6.18	Charge Pump	2	12.36
REF3030	LCSC	0.88	3.0 V Ref for Analog	2	1.76
SF-1206F080-2	Digikey	0.57	0.8 A Fuse	1	0.57
MF-MSMF150/24X-2	Digikey	0.22	PTC RESET FUSE 1.5A	1	0.22
USB4105-GF-A	Digikey	0.78	USB C PORT	1	0.78
22272021	Digikey	0.24	MOTOR CONNECTOR	2	0.48

Table 6: Bill of Materials for project budget for parts that were sourced from suppliers that were not the ECE E-Shop

It took about 6 hours to fully solder a receiver board. Assuming that the time it takes to solder the transmitter takes equally, it takes a total of 12 hours to assemble the full hardware on the PCB. While 3D printing the chassis may be automated, the time it takes to tap the standoffs to the chassis takes roughly half an hour. Assuming a wage ranging from $15 an hour to $30 an hour, cost of labor can range from $180 to $360.

3. Conclusion

I. Results

Overall, the ultrasonic communication hardware and firmware worked and the ROV hardware and firmware worked. However, the combination between the two parts of the project was not successful. The ultrasonic transmitter and receiver was able to transmit usable data packets over 3 meters while stationary and in movement, which was an important goal in our high level requirements. The ROV was able to support the ultrasonic communication through hardware and was able to have its motors turn via commands sent to the microcontroller with data packets, however the firmware for the ultrasonic receiver never made it onto the ROV. Additionally, the design document called for the use of a dual potentiometer navigation stick, but without an implementation of the ADC for the voltage divider, buttons were used instead.

II. Reflection

There were many communication issues throughout the project such as different board designs that were functionally similar, board revision and schematic revisions that were not adequately communicated. This led to two boards being produced. What should have been done was have scheduled meetings earlier in the semester rather than later in the semester. Having meetings in person ensured that communication issues were resolved as waiting for confirmation on a specification was not an issue in those situations.

The design document originally called for the use of GFSK modulation, but only DBPSK was used instead as our modulation scheme. This is because the limited bandwidth of the transducers led to worsened performance with GFSK, which requires more bandwidth than (D)BPSK for the same symbol rate. Modular circuits proved to be very time consuming, so a switch was made to use an IC. There were two separate boards that functioned similarly, so only one board made it onto the report.

There were errors that occurred throughout the design process. The first design of the PCB round did not have the right pinouts for flashing the STM32H7 over serial wire. (i.e. PA13 for SWDIO was left with no trace), as a result the microcontroller could not flash for the first week. Another minor error was the lack of consideration for interrupts required in PWM. Layout assumed that PWM could be implemented through any GPIO pin without the need for interrupt, and so bodges were required to the proper timer pins. The last error was flipping USB D+ and D- by accident. Stripping the trace and bodging a twist pair fixed this. Lack of peer review is a clear culprit in this.

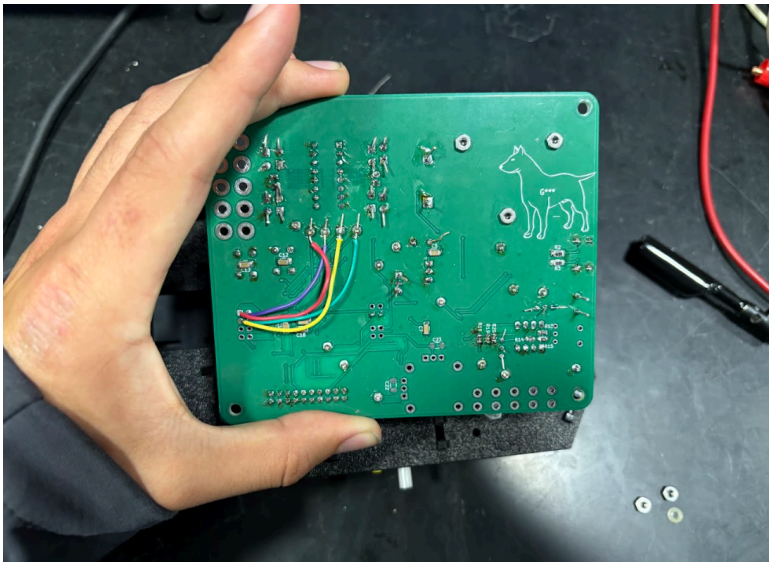


Figure 20: Bodges of points for leftover STM32 pins that supported timer outputs to the L293 H-Bridge gate control inputs on the bottom side of the PCB.

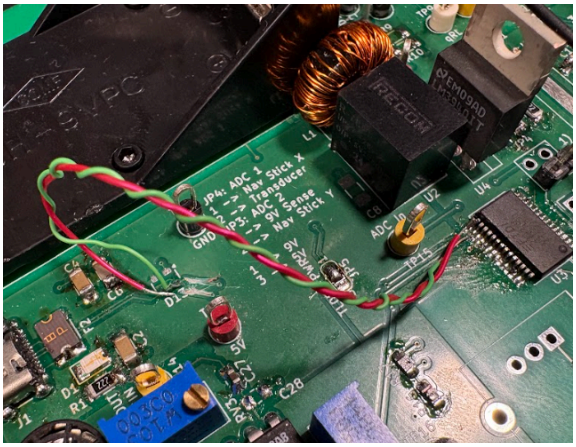


Figure 21: Bodge of the USB D+ and D- and corrected connection to the FT232RL USB to UART transceiver. [13]

III. Further work

Programming the receiver board with the communication firmware would finish all high level requirements. This would mean merging the firmware from the driver controllers, and the firmware from the motor drivers. The controller board was also not finished, but it would be a replication of the receiver board. Implementation of a coherent demodulation so that higher order modulation and a linear demodulation scheme can be used can also be considered. Higher order coherent modulation will enable higher data rates with the same bandwidth, and linear modulation will enable proper channel/ISI correction.

APPENDIX A: LIST OF ACRONYMS

ARR: Auto Reload Register
BER: Bit Error Rate
CCR: Contrast Control Register
CIC: Cascaded Integrator-Comb
CW: Clockwise
CCW: Counter-Clockwise
DBPSK: Differential Binary Phase Shift Keying
DSP: Digital Signal Processing
EMC: Electromagnetic Compliance
GPIO: General Purpose Input Output
PCB: Printed Circuit Board
PWM: Pulse Width Modulation
ROV: Remotely Operated Vehicle
SNR: Signal to Noise Ratio
TxChx: Timer X Channel X
UART: Universal Asynchronous Receiver Transmitter

APPENDIX B: STM32H7 FIRMWARE

Main Loop Implementation of Duty Cycle sweeping with UART print statements.

```
MX_GPIO_Init();
MX_ADC1_Init();
MX_TIM1_Init();
MX_TIM3_Init();
MX_UART4_Init();
uint16_t dutyCycle = 0;
uint16_t minDutyCycle = 0; // Setting minimum constraint on duty cycle
uint16_t maxDutyCycle = htim1.Init.Period * 95 / 100;
uint8_t state = 1;
HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);  // Right Motor Forward
HAL_TIM_PWM_Start(&htim1,  TIM_CHANNEL_3);  // Left Motor Forward
HAL_TIM_PWM_Start(&htim1,  TIM_CHANNEL_1);  // Right Motor Reverse
HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_4);  // Left Motor Reverse
uint8_t tx_buffer[27];
while (1){
        /* Set the duty cycle */
        sConfigOC.Pulse = dutyCycle;
        MotorFSM_Handle(state, dutyCycle, dutyCycle);
                // Sweep duty cycle from 10% to 90%
        while (dutyCycle < maxDutyCycle){
             dutyCycle += 100;
             MotorFSM_Handle(state, dutyCycle, dutyCycle);
             uint8_t tx_buffer[32] = " ";
             HAL_UART_Transmit(&huart4, tx_buffer, sprintf(tx_buffer, "State:
%u , Duty Cycle: %lu\r\n", state, dutyCycle * 100 / htim1.Init.Period), 10);
             HAL_Delay(100);
        }
        dutyCycle = htim1.Init.Period / 10;
                // Cycle through FSM states (0 to 6)
        state = (state + 1) % 7;  // Cycle through states 0-6
 }
```

Motor Control Method

```
void MotorFSM_Handle(uint8_t state, uint16_t dutyCycleLeft, uint16_t
dutyCycleRight)
{
   // First, stop all channels (set duty cycle to 0)
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0); // Right Forward
   __HAL_TIM_SET_COMPARE(&htim1,  TIM_CHANNEL_3, 0); // Left Forward
   __HAL_TIM_SET_COMPARE(&htim1,  TIM_CHANNEL_1, 0); // Right Reverse
   __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, 0); // Left Reverse
   switch(state)
```

```
    {
        case 1: // Both forward
            __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, dutyCycleRight);
            __HAL_TIM_SET_COMPARE(&htim1,  TIM_CHANNEL_3, dutyCycleLeft);
            break;
        case 2: // Both reverse
            __HAL_TIM_SET_COMPARE(&htim1,  TIM_CHANNEL_1, dutyCycleRight);
            __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, dutyCycleLeft);
            break;
        case 3: // Left forward
            __HAL_TIM_SET_COMPARE(&htim1,  TIM_CHANNEL_3, dutyCycleLeft);
            break;
        case 4: // Left reverse
            __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_4, dutyCycleLeft);
            break;
        case 5: // Right forward
            __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, dutyCycleRight);
            break;
        case 6: // Right reverse
            __HAL_TIM_SET_COMPARE(&htim1,  TIM_CHANNEL_1, dutyCycleRight);
            break;
        default: // 0 or unknown → Stop all
            break;
    }
}
```

Example Instantiation of the 20 kHz switching frequency in Timer 1 with IOC code generation:

```
static void MX_TIM1_Init(void)
{
 TIM_MasterConfigTypeDef sMasterConfig = {0};
 TIM_OC_InitTypeDef sConfigOC = {0};
 TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
 htim1.Instance = TIM1;
 htim1.Init.Prescaler = 0;
 htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
 htim1.Init.Period = 3200;
 htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
 htim1.Init.RepetitionCounter = 0;
```

```
 htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
 if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
 {
   Error_Handler();
 }
.
.
.
}
```

Receiver Implementation in C++:

```
/*
 *  From receiver.cpp
 *
 *  Created on: Mar 11, 2025
 *      Author: Ted Josephson
 */


void receiver::process(float *x) {

    for (unsigned i = 0; i < blocksize; i++) {
        // compute demodulator output
        float dbpsk_out = ac_delay_buf[ac_delay_i] * x[i];
        // update auto correlation buffer with new value, now that old one
is no longer needed
        ac_delay_buf[ac_delay_i] = x[i];

        // compute output of pulse shaping filter
        float next_psf_state = psf_state + dbpsk_out -
psf_delay_buf[symbol_phase];
        // update psf buffer with new value, now that old is no longer
needed
        psf_delay_buf[symbol_phase] = dbpsk_out;

        // zero crossing detection
        if (psf_state * next_psf_state < 0) {
            target_phase = (symbol_phase + SPS / 2) % SPS;
        }

        // resampling and timing recovery
```

```
        if (symbol_phase == target_phase) {
            push_sample(next_psf_state);
            time_since_last_zc = 0;
        } else {
            time_since_last_zc ++;
        }

        psf_state = next_psf_state;

        // update periodic indices
        ac_delay_i = (ac_delay_i + 1) % ac_delay;
        symbol_phase = (symbol_phase + 1) % SPS;
    }
}

void receiver::push_sample(float b) {

    //print(std::to_string(b) + "\n\r");

    bool bit = b > 0 ? true : false;

    float score = 0;

    for (unsigned i = 0; i < preamble_length; i++) {
        unsigned si = (bit_index + i) % frame_size;
        if(bits[si]) {
            score += preamble[i];
        } else {
            score -= preamble[i];
        }
    }

    if(score > 14) {
        unsigned payload = 0;
        std::string outstr = "";
        unsigned msg_start = bit_index + 2 * preamble_length;

        // assemble message bits into payload
        for (unsigned i = 0; i < payload_size; i++) {
            payload |= bits[(msg_start + i) % frame_size] << i;
```

```
            outstr += std::to_string(bits[(msg_start + i) % frame_size]);
        }

        uint32_t decoded_message = fec_golay2412_decode_symbol(payload);

        if (decoded_message == 0x1000) {
            //print("error decoding message");
        } else {
            print("payload: " + std::to_string(decoded_message) + " bits:
" + outstr + " score: " + std::to_string(score) + "\n\r");
        }
        last_payload = decoded_message;
    }



    bits[bit_index] = bit;

    bit_index = (bit_index + 1) % frame_size;


};
```

REFERENCES

[1] Murata, *MA40S4S Datasheet*, 2022, [Online] Available: https://www.mouser.com/datasheet/2/281/MA40S4S-792899.pdf

[2] Analog Devices, Maxim Integrated, *MAX1044 Switched-Capacitor Voltage Converters*, 2019, [Online] Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ICL7660-MAX1044.pdf

[3] ST Microelectronics, *STM32H7B3RI Datasheet*, 2022 [Online] Available: https://www.st.com/resource/en/datasheet/stm32h7b3ri.pdf

[4] WCH, *CH340N*, [Online] Available: https://aitendo3.sakura.ne.jp/aitendo_data/product_img/ic/inteface/CH340N/ch340n.pdf

[5] RECOM, *R-78E-0.5 DC-DC Buck Converter,* 2024 [Online] Available: https://recom-power.com/pdf/Innoline/R-78E-0.5.pdf

[6] Shenzhen Yatelien Technology Co, *YF16-DFL7.2-B5Ko(45-10)B5Ko(55)-RG-A22*, [Online] Available: https://www.lcsc.com/datasheet/lcsc_datasheet_2408061713_YTL-YF16-DFL7-2-B5Ko-45-10B5Ko-55-RG-A22_C37323741.pdf

[7] P Krein, A Banerjee, *ECE 469 Laboratory Notes*, 2024 [Online] Available: https://powerece469.web.illinois.edu/wp/experiment-4-dc-dc-conversion-part-ii-converters-for-motor-drives/

[8] S. B. Dewan, G. R. Slemon, A. Straughen, *Power Semiconductor Drives*. New York: John Wiley, 1984.

[9] L Kinsler, A Frey, A Coppens, J Sanders, et al, *Fundamentals of Acoustics*, 4th Ed, New York: John Wiley, 2000.

[10]"Ultrasonic Transducer Impedance | David Pilling," *Davidpilling.com*, 2021. https://www.davidpilling.com/wiki/index.php/Transimp (accessed Mar. 07, 2025).

[11] ST Microelectronics, *AN4476 General Purpose Timer Cookbook for STM32 Microcontrollers*, July 2019 [Online] Available: https://www.st.com/resource/en/application_note/an4776-generalpurpose-timer-cookbook-for-stm32-microcontrollers-stmicroelectronics.pdf

[12] Texas Instruments, *L293x Quadruple Half -H Drivers*, January 2016 [Online] Available: https://www.ti.com/lit/ds/symlink/l293.pdf

[13] Future Technologies Devices International Ltd. (FTDI), *FT232 UART USB IC Datasheet*, May 2025, [Online] Available: https://ftdichip.com/wp-content/uploads/2020/08/DS_FT232R.pdf