Embedded Systems (EECE 4376) Project Report

Team: Liam Kelly, (name redacted), and (name redacted)

Submitted: 9 December 2017

Submitted to: (name redacted)

Real-time Autonomous Ultrasonic Swarm Navigation (RAUSNav)

Technical Manual

Table of Contents

Summary

Real-time Autonomous Ultrasonic Swarm Navigation (RAUSNav) is a traffic control implementation that locates small robots within an area using ultrasound and controls them wirelessly. With ultrasonic pings and a system-level closed-loop speed and heading control, the robots are commanded to cleverly avoid one another with a real-time avoidance solution.

To further demonstrate the autonomous nature of the avoidance system, RAUSNav supports a user-controlled “driver". The user can drive this robot through the field unimpeded, and the avoidance system commands the autonomous robots to dynamically avoid the user’s robot and each other. Through these technical accomplishments, RAUSNav hopes to construct a small-scale solution to complex autonomous vehicle design problems.

Though this project’s environment-based positioning system does not properly correlate to the majority of agent-based modern research in autonomous vehicles, it still presents important, non-trivial challenges in wireless integration, localization, and even underdetermined robotics. Incorporation of ultrasonic transceivers, RF transceivers, and additional PIC microcontrollers into the Pololu 3pi platform necessitate custom hardware development and driver integration. Precise localization has required extensive testing and calibration of ultrasonic transceivers and their arrangement. The real-time specifications of the system demand a reliable avoidance algorithm that runs in 15 ms intervals (for vehicles n=3) on the BeagleBone Black.

NOTE: RAUSNav’s implementation is currently incomplete. As of now, avoidance algorithms are in development, but localization is functioning properly. In a future release, look for updates about avoidance algorithms with three robots (including one user-controlled robot), four robots, and many robots. This manual will cover the current implementation of RAUSNav, which features two-robot avoidance.

The Purpose Of This Document

This document will present the design concepts of RAUSNav. More specifically, it will explore RAUSNav’s design strategies, hardware specifications, and software schemes. The structure, cause, and purpose of each subsystem will be explored for the purpose of the reader’s understanding of the design of RAUSNav.

Hardware

Parts List

- 1 x Beaglebone Black
- 10 x NRF24L01 Radio Frequency Transceiver
- 4 x Pololu 3pi Robot
- 1 x AVR Dragon (ATMega Programmer)
- 4 x Pololu Ultrasonic Transmitter PCBs (*see Figures 1 and 2*)
    - 1 x Pololu Mount PCB (see *Project Files/PCBs/Pololu PCB* for Gerber and Eagle documents)
    - 1 x UT-1640K-TT-2-T Ultrasonic Transmitter
    - 1 x PIC16F1503 Microcontroller (DIP-14 Package)
    - 1 x CD4049UBE Hex Inverter
    - 1 x UA78M33 3.3V Regulator (TO-220 Package)
    - 2 x 0.22 uF Through-Hole Ceramic Capacitor (0.1" pitch, +/-5%)
    - 1 x 47 uF Through-Hole Electrolytic Capacitor (0.1" pitch, +/-5%)
    - 2 x 0.1 uF 0805 Ceramic Surface-Mount Capacitor (+/-5%)
    - 4 x 10 kΩ 0805 Resistor (+/-5%)
    - 1 x 10 Ω 0805 Resistor (+/-5%)
    - 1 x 1206 Surface-Mount LED
    - 1 x 7x2 Long Male Header
    - 4 x Plated Ultrasonic Emitter Cones (*see Figures 7 and 8*)
    - 1 x 9V Battery Mount (*see Figure 9*)
    - 1 x 9V Wired Adapter
- 4 x Ultrasonic Receiver PCBs (*see Figures 3 and 4*)

    - 1 x Receiver PCB (see *Project Files/PCBs/Receiver PCB* for Gerber and Eagle documents)
    - 1 x UT-1640K-TT-2-R Ultrasonic Receiver
    - 1 x PIC16F1503 Microcontroller (DIP-14 Package)
    - 1 x TLC277 Operational Amplifier (DBV/SOT-23 Package)
    - 1 x MMBT3904 NPN Transistor (SOT-23 Package)
    - 1 x UA78M33 3.3V Regulator (TO-220 Package)
    - 2 x 0.1 uF 0805 Ceramic Surface-Mount Capacitor (+/-5%)
    - 1 x 22 pF Through-Hole Ceramic Capacitor (0.1” pitch, +/-5%)
    - 4 x 100 kΩ 0805 Resistor (+/-5%)
    - 1 x 10 kΩ 0805 Resistor (+/-5%)
    - 2 x 560 Ω 0805 Resistor (+/-5%)
    - 1 x 10 Ω 0805 Resistor (+/-5%)
    - 1 x 1206 Surface-Mount LED
    - 1 x 9V Battery Mount (*see Figure 9*)
    - 1 x 9V Wired Adapter
- 1 x Curiosity 8-Bit Development Board by Microchip
- 1 x Enclosure, 1.2 m x 1.2 m (*see Figure 5*)
- 8 x 9V Batteries
- 16 x AAA Batteries
- 1 x Bluetooth Transceiver
- 1 x Android-Compatible Smartphone
- 1 x RAUSNav Android Application
- 1 x Jumper Wire Pack: male-to-male, male-to-female, and female-to-female
- 1 x Male and Female Header Pack: 0.1” pitch

Primary Components

Pololu 3pi Robot (pololu.com/product/975)

The Pololu 3pi is a programmable robot platform. It has an on-board ATMega328P microcontroller, which can be programmed via its ISP cable (the AVR Dragon [*see below*] was used for programming). Although the 3pi has several sensors that can be used for light detection and other specialized applications, RAUSNav only uses its motors and its open ports to its microcontroller (located on a female header jack at its top).

AVR Dragon (atmel.com/tools/avrdragon.aspx)

The AVR Dragon is the programmer used for the Pololu 3pi robot. It interfaces over USB with Atmel Studio, a software environment from Atmel.

NRF24L01 Radio-Frequency Transceiver (nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01)

The NRF24L01 is an RF transceiver that communicates with a protocol that sends packets over SPI. It can interface with the PIC16F1503 and the BeagleBone, and all wireless communication in RAUSNav is accomplished with the NRF transceiver.

PIC16F1503 Microcontroller (microchip.com/wwwproducts/en/PIC16F1503)

The PIC16F is a family of 8-bit microcontrollers. In RAUSNav, the PIC is used for SPI communication with the BeagleBone over the NRF24L01 modules, as well as UART communication, done with bit-banging, with the Pololu on the Transmitter PCBs and time-of-flight measurements on the Receiver PCBs with its hardware timer.

Curiosity 8-Bit Development Board (mouser.com/new/microchip/microchip-8bit-curiosity/)

The Curiosity board from Microchip is a programming board for the PIC16F microcontroller family. It was used to program the PIC16F1503 on the Pololu PCB and the Receiver PCB. It interfaces over USB with the MPLAB development environment by Microchip.

BeagleBone Black (beagleboard.org/black)

The BeagleBone Black is an open-source system-on-chip device that runs a Linux operating system. RAUSNav uses it as its hub for all traffic control, Bluetooth communication, and RF communication.

Transmitter Boards

The Transmitter Boards give the Pololu robots the ability to transmit ultrasonic sound signals. The inspiration for the transmitter circuit came from a document by Texas Instruments about measuring distance with an MSP430 microcontroller (see *Project Files/Other Documents*). It is designed to fit into the female headers on the top of the Pololu Board, as it takes input from the ATMega to transmit ultrasound. *See Figures 1 and 2* below for overviews. Note that the in-house PCB milling machine that the RAUSNav team had available did not have any way to through-hole plate holes, so vias had to be created by hand, and most of the board's traces were routed on the bottom layer.

The schematic layout is in Figure 2. Though it says in the top left that a AAA battery pack should be used, the AAA pack was later replaced with a 9V battery pack. The header connecting to the 3pi is on the far-left; it utilizes the PD7 pin for ultrasonic transmission and the GND pin for a common ground. The PD7 pin feeds into a logic converter circuit, which feeds into a hex inverter. Whenever the PD7 pin goes to HIGH, then a logic converter circuit outputs a HIGH 15V-signal to the hex inverter, which supplies enough current to swing the transmitter terminals 30V (since one end is on 0V LOW and the other end is on 15V HIGH, or vice-versa).

Meanwhile, the PIC receives signals from the NRF transceiver and outputs UART signals (9600 baud) to the 3pi's UART pin on PD0. (Note that the PIC does not interact with the transmitter at all; instead, the Pololu receives commands indirectly from the BeagleBone and outputs a 40 kHz signal on its PD7 pin when instructed to.) Since the PIC does not have a designated UART pin, the UART signal from the PIC had to be bit-banged; this feature is included in the software package.

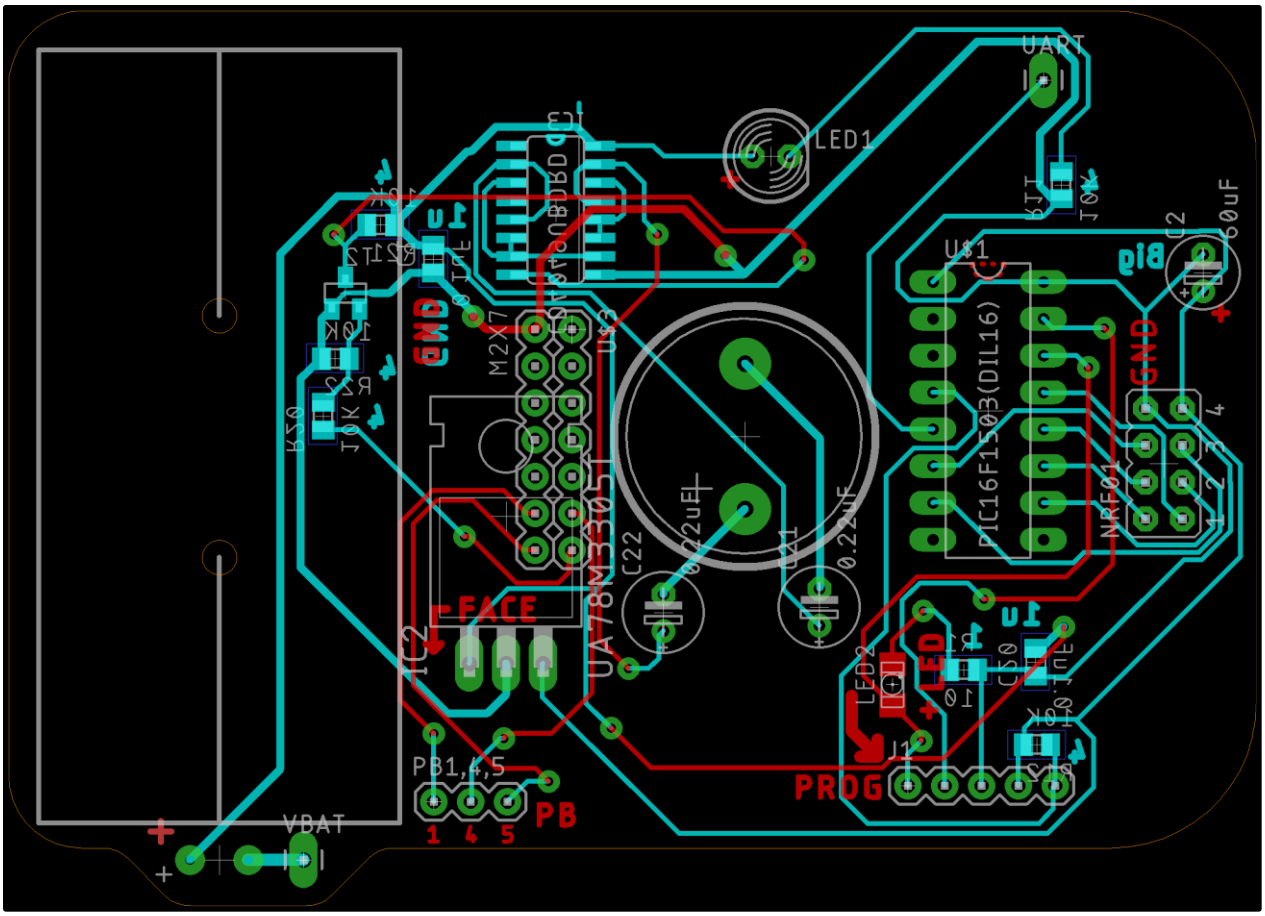

Figure 1: The Pololu Ultrasonic Transmitter PCB layout. Note the location of the ultrasonic emitter (the big circle in the middle of the layout); it is centered above the 3pi when inserted properly into its female header.

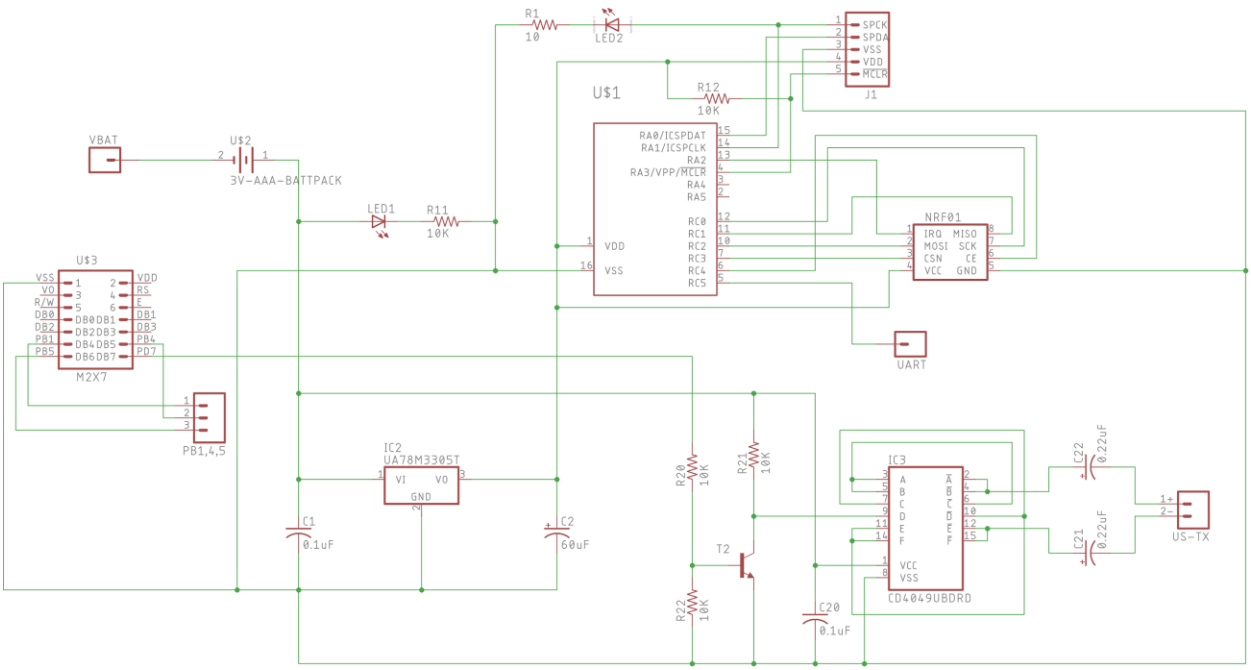

Figure 2: The Pololu Ultrasonic Transmitter PCB schematic. Note the UART pin connected to RC5 on the PIC.

Receiver Boards

The Receiver Boards receive ultrasonic pings from the Pololu Ultrasonic Transmitters as well as record the time it takes for signals to be received. They can clock time-of-flight delays with the hardware timers on their PICs. As the PICs interface with the NRF transceivers, they can begin monitoring time when the 3pis are instructed to emit an ultrasonic signal (the time difference between the two received signals is negligible). The board layout is shown in *Figure 3*, and the schematic is shown in *Figure 4*.

The ultrasonic piezo is placed on the 1.2m x 1.2m stage at a designated corner, pointed horizontally toward the center of the board. Its signals are amplified by an op amp (gain ≈ –360) and fed into the RA4 pin on the PIC. When a LOW is read on RA4, then the value of the designated hardware timer (i.e. some multiple of the time that it took the ultrasonic signal to travel to the receiver) in the PIC is sent to the BeagleBone for processing.

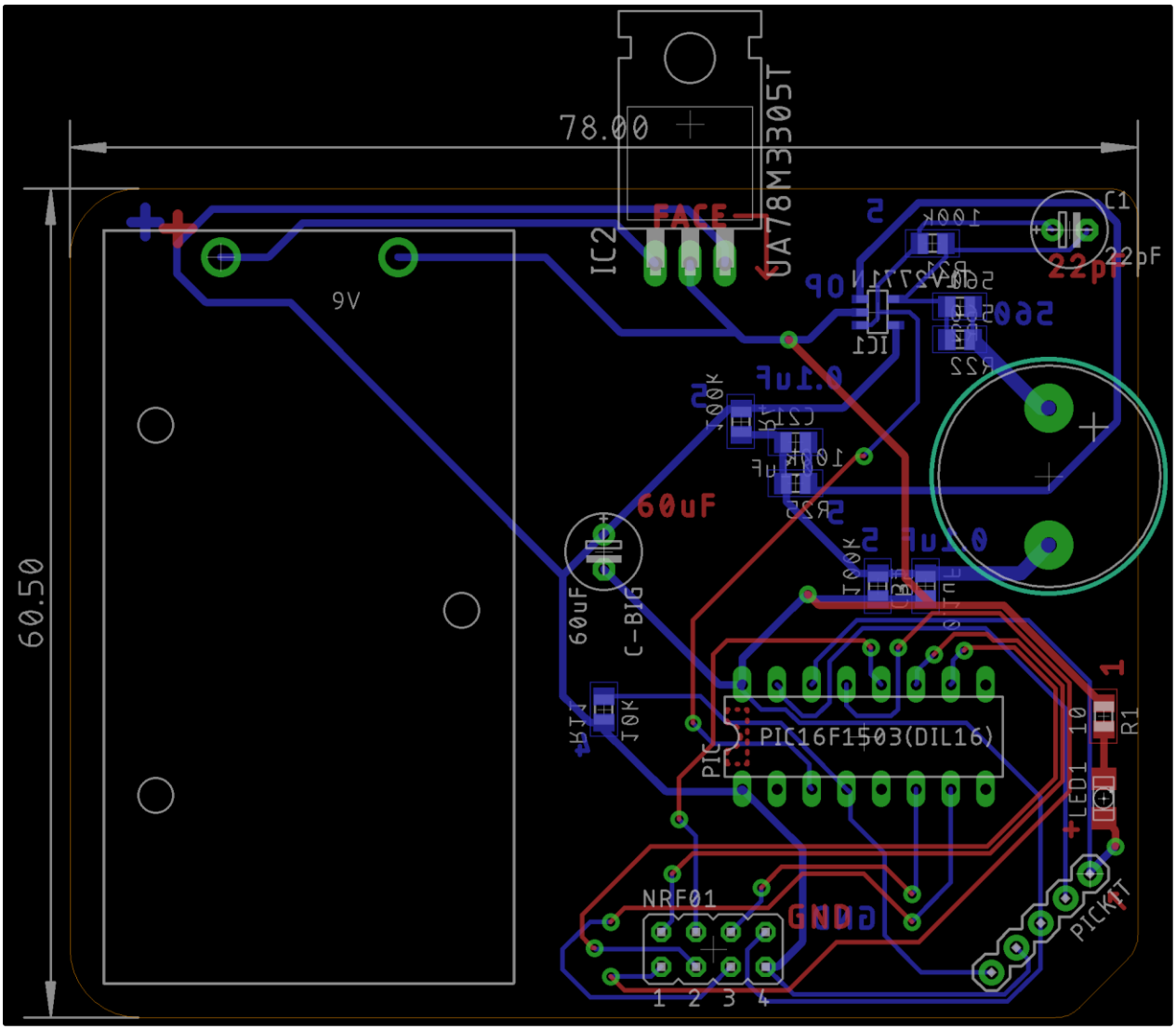

Figure 3: The board layout for the Ultrasonic Receivers. The ultrasonic receiver piezo is not soldered directly to the board; rather, it is extended with jumper wires to rest on a corner of the 1.2m x 1.2m wood stage.

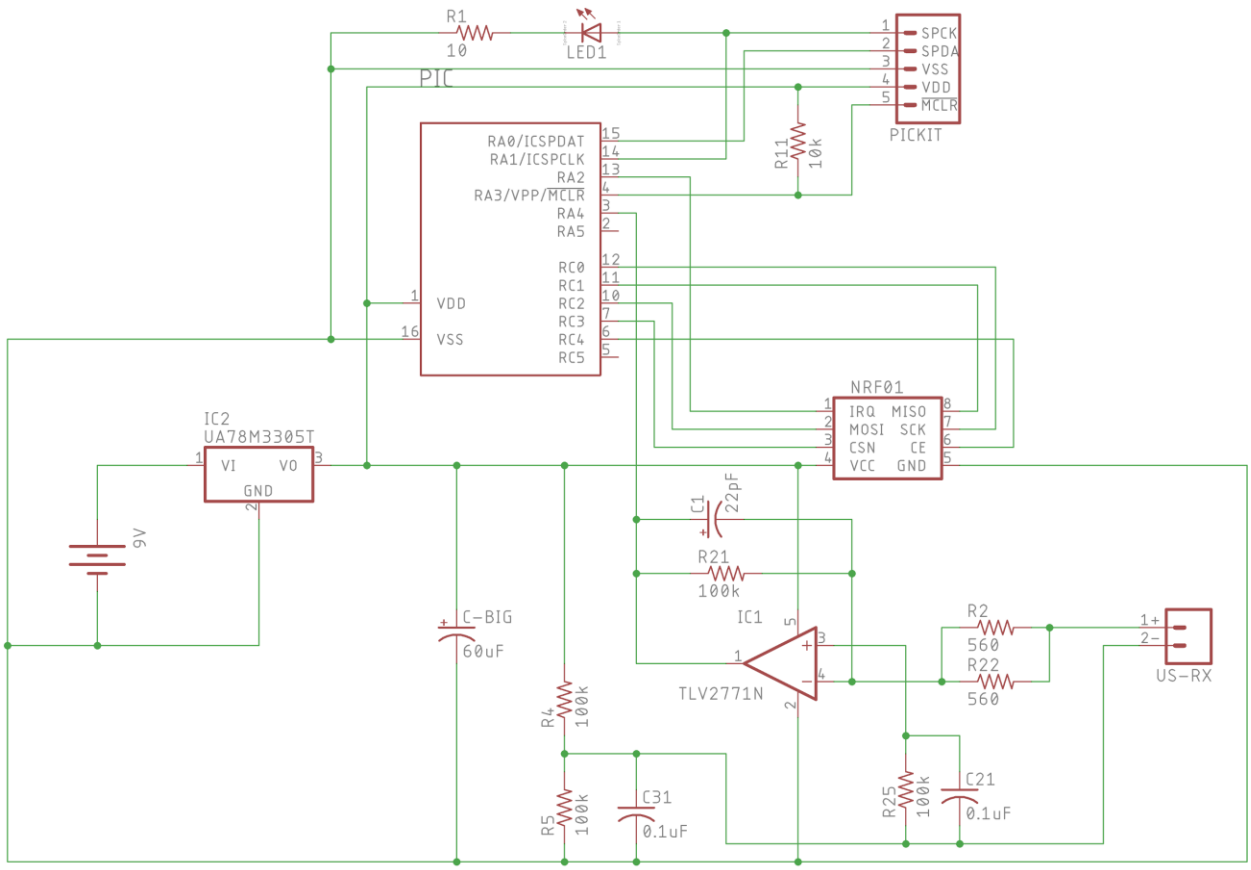

Figure 4: The schematic layout for the Ultrasonic Receivers. As shown, the input from the ultrasonic receiver piezo (right) is amplified (gain = –360) through the op amp and registered onto the RA4 pin on the PIC. When the PIC reads a LOW on RA4, then an ultrasonic ping has been received.

A Note About Gerber Files

The Gerber files used to mill the PCBs are in the *Project Files/<PCB>/Gerber Files* directory. These, along with the BRD and SCH files themselves, were generated from the Eagle Autodesk software. Their file extensions determine their function:

Extension	**Layer**	**Mnemonic**
.CMP	Top layer	“Component” side
.SOL	Bottom layer	“Solder” side
.TXT	Holes	--
.DRI	Drill information file	“Drill” file
.GML	Board outline	--
.GPI	Board outline information file	--
.CRB	Top layer rubout	“Component-side RuBout” file

Auxiliary Parts

The Field

RAUSNav’s Field is composed of four pieces of 1.5” x 3” wood planks and two fitted styrofoam boards, marked in increments of 10 cm (*see Figure 5*). Its inner dimensions are about 1.2 x 1.2 m. All four receivers are placed on each corner of the field, and each piezo receiver is oriented horizontally to point toward the center of the field. Tape is used to secure them in place (*see Figure 6*). The robots are placed somewhere in the field where the piezos can detect them.

Styrofoam was used as a base to prevent accidental multipath from the ultrasound; it has a higher sound absorption quality than wood, and it is a little easier for the 3pis to drive over (albeit a little fragile).

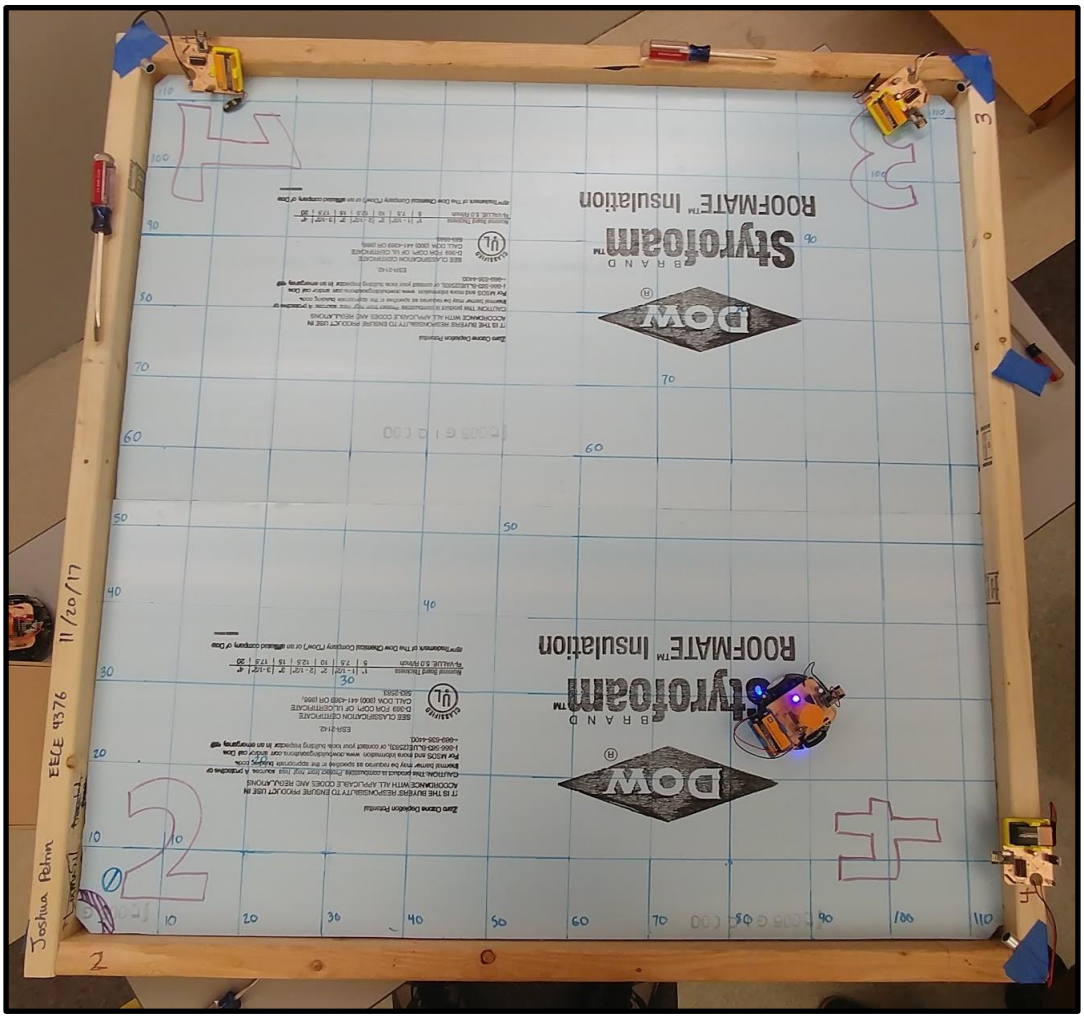

Figure 5: The RAUSNav field setup. The grid is spaced 10 cm. There is a piezo receiver at each corner, numbered as shown.

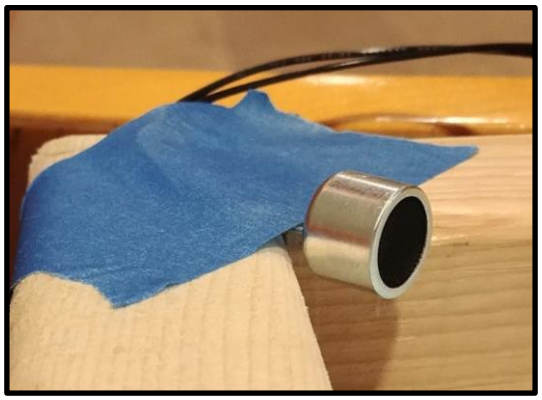

Figure 6: The ultrasonic piezo receiver attached with tape to the wood frame.

Transmitter Cones

The transmitter cones go on top of the transmitter piezos on the Pololu Transmitter Boards. They were designed in Autodesk Inventor, and their STL and PRT files can be found under *Project Files/3D Prints*. Ultrasonic sound hits their ceilings at a 50° angle with respect to the vertical. A sheet metal cone was inserted onto the ceiling to decrease sound absorption by increasing the acoustic impedance; it was created from a 300° sector of a circular piece of sheet aluminum with a radius of 12 mm (*see Figures 7 and 8*).

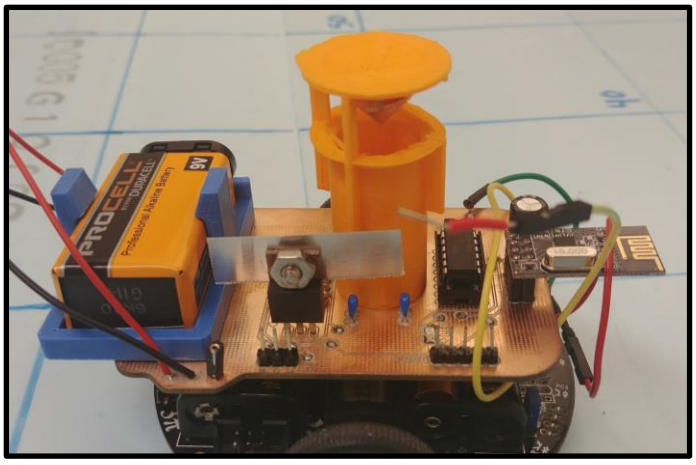

Figure 7: The Pololu 3pi with attached Transmitter PCB and Transmitter Cone. One of the supports on the cone broke when the metal fill was being glued into the interior.

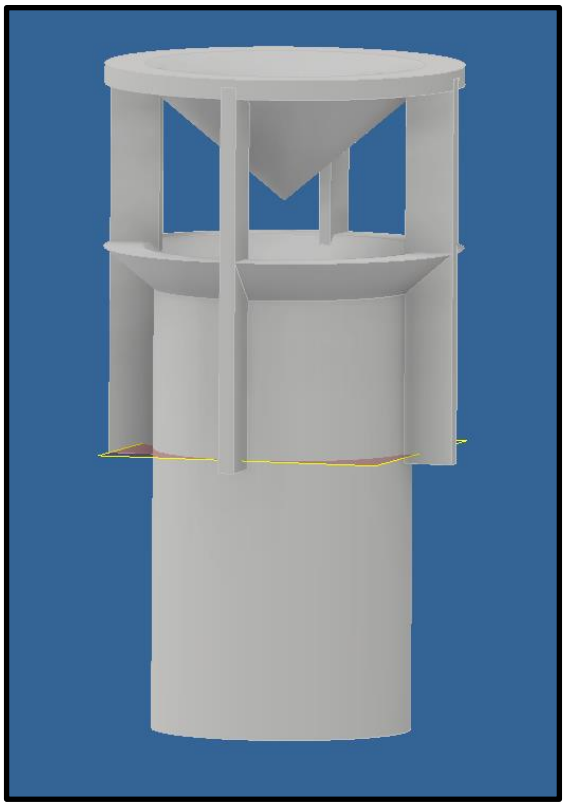

Figure 8: A 3D realization of the Transmitter Cone (created in Autodesk Inventor software). The metal cone was glued over the pointed ceiling, as shown.

9V Battery Holder

RAUSNav's 9V Battery Holder was found on Thingiverse (thingiverse.com/thing:336319). Its creator is ahmetcemturan (thingiverse.com/ahmetcemturan/about). The current implementation of RAUSNav uses six copies of it. The 3D model of the holder is in Figure 9.

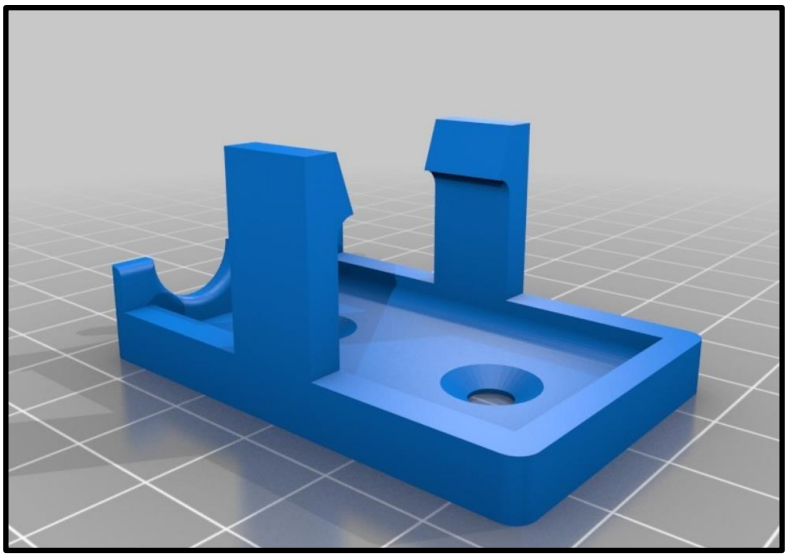

Figure 9: The 9V battery holder from Thingiverse, by user ahmetcemturan.

Software

Android App

The Android application was developed in Android Studio, written in Java and XML. There is a single Activity which uses a custom view, a class called DriveView, as its content view. Before setting the content view, it will attempt to connect to the HC-06 Bluetooth Transceiver. After accomplishing this, the view will load, displaying a thin, black arc. When one places their finger on the arc, a red dot will appear in the location they pressed their finger. Its location will indicate the direction of the robot. For example, if one places their finger right of center of the arc, the robot will turn right. While one must place their finger on the arc to move the robot initially, as long as your finger remains on the screen, the robot will continue to move; it will only stop once you lift your finger. If you place your finger on the arc, and then move it off the arc while still keeping it on the screen, the location of the red dot will be where the line through the center of the circle which the arc belongs to and your finger intersects the arc -the robot will move as though you placed your finger where the red dot ends up. In order to avoid sending packets at an excessively high rate, a new packet will only be sent when your finger is initially placed on the arc, removed from the arc, or moves significantly along the arc. In fact, there are only 15 defined x values and 9 defined y values on the arc. When a new data is to be sent, a

packet will be crafted into a byte array following a specific protocol which the BBB expects and is then sent to the HC-06 over Bluetooth.

Towers

The tower code is written in C. The PICs on the towers first configure SPI, pins, and a hardware timer. Then, they begin to listen for a packet. If they receive a packet with the start listening header, they will start the hardware timer and wait until they receive an active low signal from the ultrasonic receiver. As soon as the signal is received, they will read the timer's value (which cannot overflow for valid data based on the speed of sound, prescale value, and 16-bit register size), and then configure transmission mode. The raw count from the timer will be sent to the BBB, and they will return to listening mode.

Rather than implementing a timeout on the towers for receiving a ping, they will simply continue to count, but remain in receive mode. If they receive a new start timing ping, they will reset the timer -this means that they missed an ultrasonic signal on the last ping, but it is okay because a new ping should be arriving, and their timers will have been reset.

Robots

For both the PICs and the AVRs, the robot code is written in C.

PICs on the Robots

The PICs on the robots first configure SPI, bit banging UART, and pins. Then they begin to listen for packets. They expect two packet headers: either a start ping packet, or a direction packet. If they receive a start ping packet, they will send a one-byte UART signal to the AVR telling it to send an ultrasonic ping. If they receive a direction packet, they will craft a UART packet containing x and y direction to the AVR.

AVRs on the Robots

The AVRs first configure PWM, the ultrasonic emitter (i.e. a timer), interrupts, and UART. Then, the robot simply waits for interrupts: either UART or timer. If it receives a UART interrupt, it reads the packet and checks if it is a send ping or a set direction packet. If it is a set direction packet, it sets a flag so that when timer interrupts are received, the robot knows to flip the ultrasonic pin so as to create an AC signal which in turn creates ultrasonic sound. Note that after 5ms of pinging ultrasonic, the flag will be flipped so that it stops pinging. If it receives the head of a direction packet, it will set another flag and start to fill a 4-byte buffer. Once the buffer is full, it will verify that the packet was valid. If so, it will send the x and y direction to a steering function.

The steering function must set the speed of two parallel motors. To translate x and y, where the y axis is the forward direction for the robot, to speed of two parallel motors, the following principle is applied: for the robot to move at a given y speed, both motors must necessarily have at least that same speed. Since y-direction is the forward direction for the robot, the x-direction can be thought of as rotation. To rotate, there must be non-zero torque, and therefore one motor must have an excess speed over the other -this is based on x.

BBB

The BBB is the centralized controller for the system. In the main function, it initializes a mutex and three threads. The three threads are:

Manual Control

The manual control thread first opens a UART1 file on the BBB. It then runs an infinite loop which polls the UART file for data. When it receives data, it ensures that the first data packet it received is a direction packet header. Then, expecting a specific packet structure, it will set flags to indicate which portions of the packet it has received so that it can properly handle the packet data. Once it has read a complete packet, it will craft an RF packet with a set direction header to be sent to the manually controlled robot over the NRF. After this, it will reset flags and continue to poll for new UART data from the HC-06.

Send Ping

The send ping thread runs an infinite loop which, every 100ms, will transmit a send ping packet first to the manually controlled robot and then to the autonomous robot. This is performed by a function called pingCoords. In pingCoords, the header passed to the function is placed in a transmission packet. This header is so that the correct robot knows to ping. Then two separate NRF packets are sent: one to the towers, and another to the robot that is being triangulated. The function then enters a loop that will continue until either three towers respond with valid data (i.e. data between 0 and MAX_LENGTH cm) or until the loop times out (10ms, which is slightly more than the maximum time it could take for ultrasound to traverse the diagonal of the enclosure). Finally, the function will check to see if at least three towers have sent data in the last 10 ping attempts. If they have, it will perform triangulation computations for that robot and place its coordinates in a coordinate array. The triangulation computations are done through a deterministic calculation *without* square roots (this is only possible because any three towers form a right angle and therefore two-dimensional Euclidean coordinate system. The simple statement of the mathematics is as follows:

Consider a point at the origin and at a point on the y axis in $\mathbb{R}^2$, where the second point has y coordinate p. Suppose each point has a circle centered around it, with radii r_1 and r_2, respectively. Suppose that the following restrictions hold: $p \neq 0$, $r_1 < p + r_2$, $r_1 > p - r_2$, and $p < r_2 - r_1$.

Theorem: *If, given the restrictions noted above, the two circles intersect at the points* (x_1, y_1) *and* (x_2, y_2) *then* $y_1 = y_2$ *and* $x_1 = -x_2$ *where* $y_1, x_1 \in \mathbb{R}$

The first circle can be defined by the following equation: $(y-p)^2 + x^2 = r_2^2$ and the second can be defined by $y^2 + x^2 = r_1$. By subtracting the second equation from the first, we have:

$$-2yp + p^2 = r_2^2 - r_1^2$$

$$y = -\frac{(r_2^2 - r_1^2 - p^2)}{2p}$$

Clearly, then, y has a single solution. Note that, nicely, this is a closed form polynomial for y. Since $p \neq 0$, this is defined. Substituting back in yields that:

$$x^2 = r_2^2 - (-p + \frac{p^2 + r_1^2 - r_2^2}{2p})^2$$

Therefore if x_1 is a solution to the above equation, $x_2 = -x_1$ must be a solution, as $a^2 = (-a)^2$. Now we want to make sure that we have two such real valued solutions. If the RHS is positive and non-zero, then we will have two real-valued solutions.

Autonomous Control

The autonomous control thread contains an infinite loop that checks to see whether the changeDirection flag was set by send ping. This flag is set whenever new location data becomes available. Based on the current locations of the robots and the autonomous robot's heading, a new heading for the autonomous robot will be chosen. The algorithm for this is described in the following section on the Python simulation.

Python Simulation (Avoidance)

A python file, avoid.py, was written to simulate the avoidance algorithm. The simulation uses the Pygame interface to capture key presses as the inputs to control the manually controlled robot and displays the locations of the manually controlled and autonomous robots as circles on a display. The algorithm works as follows: the autonomous robots starts at the origin (top left corner) and the manually controlled robot starts in the center of the display. The robot tries to get to the diagonally opposite corner while avoiding the manually controlled robot. Ideally, the robot should take the path directly connecting these two corners because, should it encounter the manually controlled robot, being in the center will give it the most room to maneuver away. Its first priority, however, is to avoid the manually controlled robot, so if it gets within a critical radius, it will change its heading. It can either turn left or right, so a rotation matrix

is applied to its heading vector in both directions. Then, its movement in that direction is simulated through several steps to see which rotation will take it further away from the manually controlled robot. Once it is outside the critical radius, it will attempt to change its heading so that it can get back on the diagonal path so that if it encounters the manually controlled robot again, it will, as noted, have the most space to maneuver.