Ice Detection & Salt-Dispensing Unit

Components of the automatic salt-dispensing robot

By

Chun-Ting Wang Lee

&&

Naman Mehta

Final Report – ECE 445

TA: Ryan May

Project No. 22

Table of Contents

1. Introduction

1.1 Purpose

Our project is an ice-detecting salt-dispensing semiautonomous robot. Our purpose can be summarized to replacing manual labor with technology in the task of dispensing rock salt over ice in in winter weather. With this project, we dive into fields we never explored before: wireless communication, power, robotics, and sensors.

1.2 Project Functions

Target Functionality:

- Remote-controlled robot movement
- Semi-autonomous robot movement
- Ice Detection
- Salt delivery on target ice (mechanical aspect)

Deliverable Functionality:

- Remote-controlled robot movement
- Semi-autonomous robot movement
- Ice Detection

1.3 Project Block Diagram

As shown in figure 1, the project is consisted of several different functional blocks:

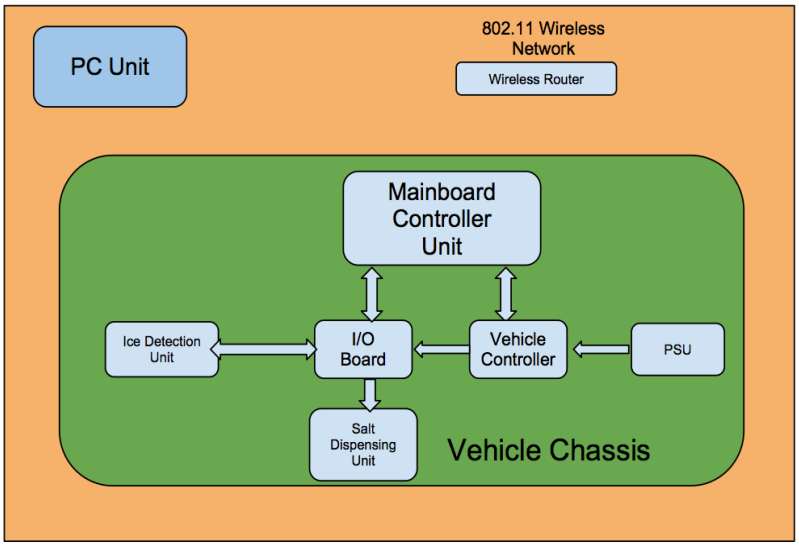


Figure 1 Project Block Diagram

The following diagram (figure 2) shows how the work is divided into two main portions; Naman was in charge of the networking and computer interfacing, Tim was in charge of the analog ice detection, salt-dispensing and input-output peripherals. The rest of the components such as vehicle control and PSU are assigned as team collaboration components.

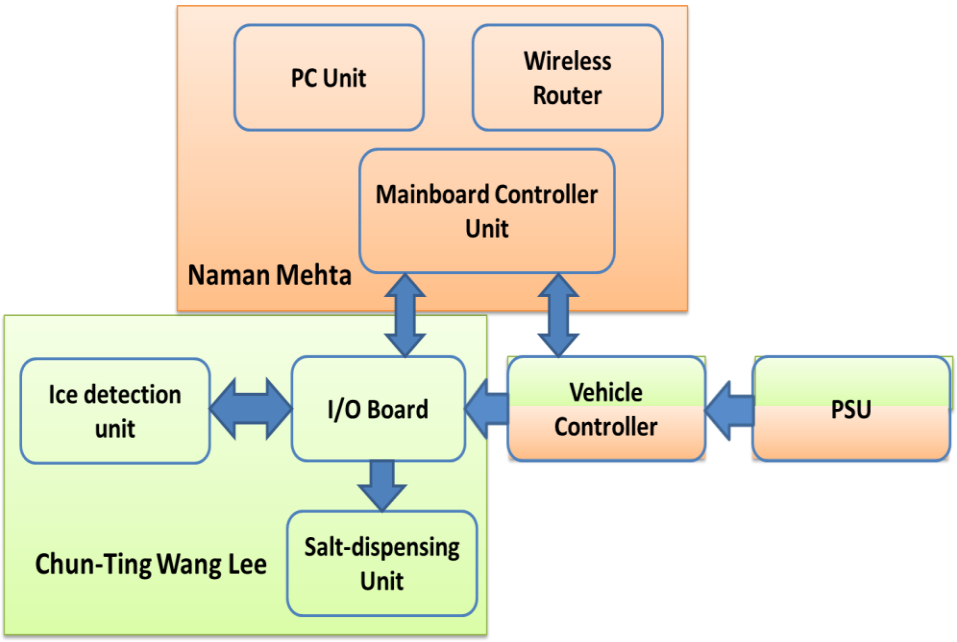


Figure 2 Work Division Diagram

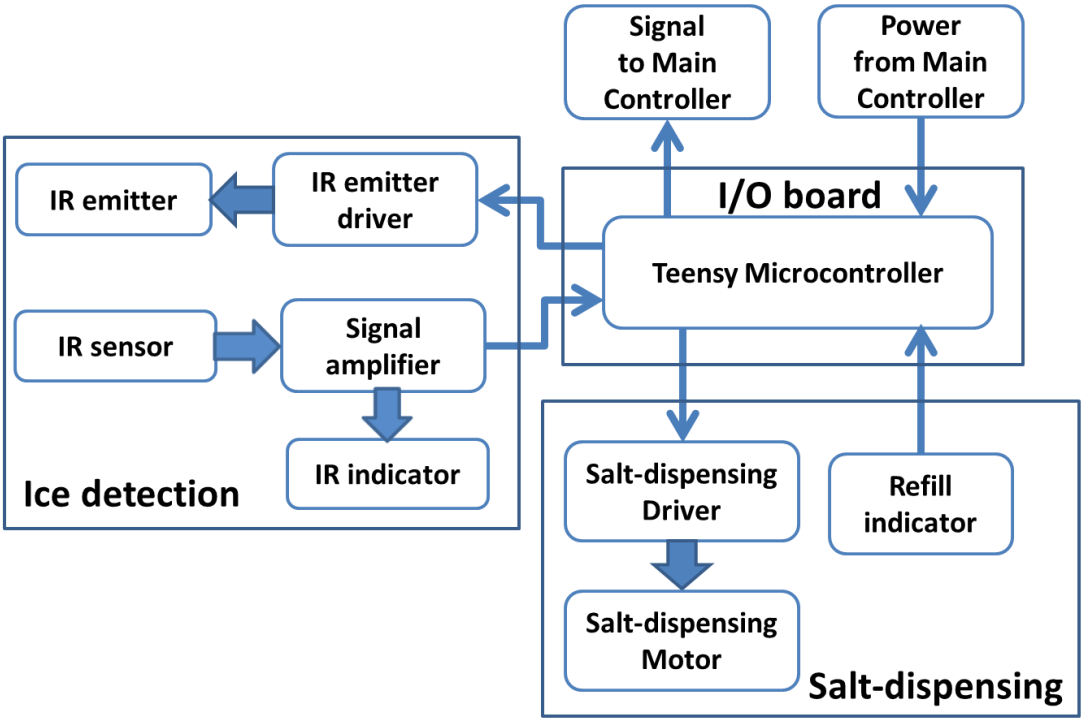


Figure 3 Detection & Dispensing Sub-Diagram

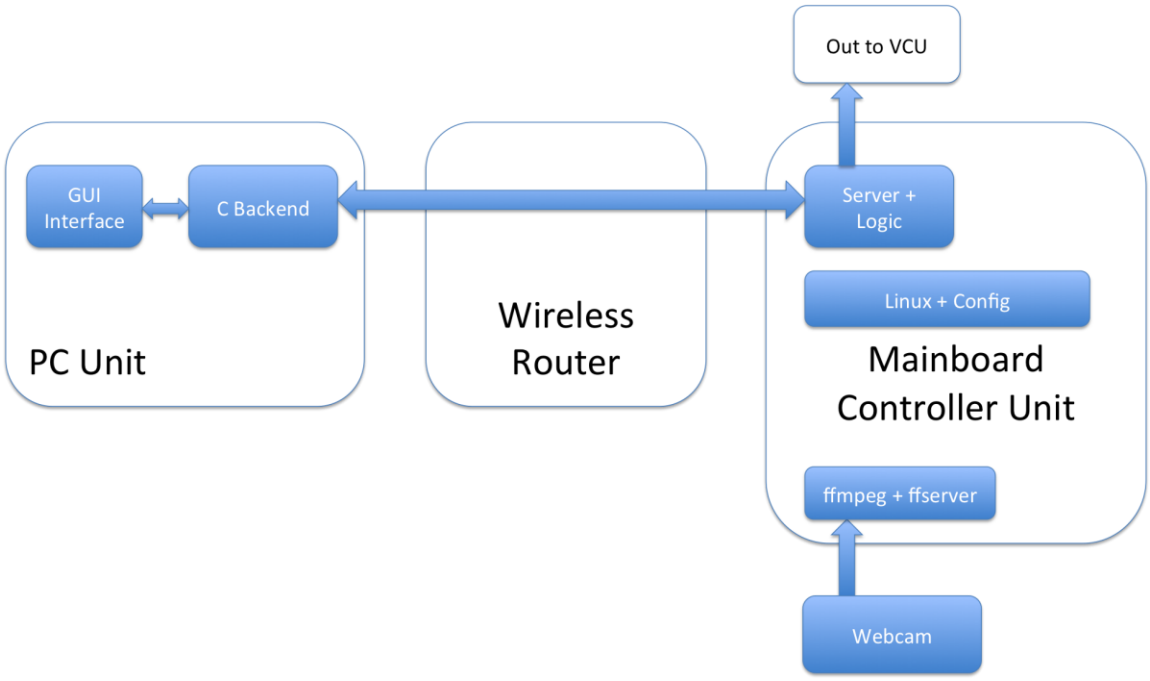


Figure 3.5 – PC Unit, Wireless Router, and MCU

I/O Board

The I/O Board consists of a Teensy microcontroller. We chose to do our I/O on a separate microcontroller then the one included in the VCU because of the amount of I/O pins we need. The ideal case would have been to send ice detection results back to the Mainboard controller unit, which forwards the information. It would also control the salt dispensing mechanism. The connection to the mainboard would be USB. In our deliverable, we kept this as a separate unit for demonstration purposes, so we did not need to interface with the Mainboard controller unit.

Salt Dispensing Unit

The unit is responsible for mechanically dispensing the salt. It would consist of a container for the rock salt to be dispensed, a motor to control the outflow of the rock salt, and a refill indicator circuit. Rock salt would be placed inside a container that has an extended opening that is tilted and narrow. The outflow of rock salt would be determined by how fast the trap door by the opening is being lifted. This unit was not implemented in the final deliverable.

Ice Detection Unit

Mounted at the front of the robot, the unit notifies the I/O Board whether there is ice in front of the vehicle or not. Consisting of IR emitters, an array of IR sensor and some circuitry that protect the integrity of the signal, the unit will determine the presence of ice by comparing the amount of IR signal detected by the sensor array. The IR emitters will send out light signals that will then be reflected by the ground below. The sensor array, monitoring the IR signals reflected by the ground below, should pick up sufficient IR reflection, amplify it, and deliver the information to the I/O Board.

PC Unit

The PC unit in the setup consisted of a laptop with a unix environment, wxPython, Python, and wireless card. The PC would present a GUI interface to the user for control of the robot. The GUI interface would have its own logic that would do some high-level tasks and then depend on the C Backend to do the network communications to the server running on the Mainboard Control Unit.

Wireless Router Unit

This is a stock 802.11 wireless router, a Linksys WRT54GL in our case. Providing a network in which the Mainboard Controller Unit can register itself with a static IP, the router serves only as a communications medium.

Mainboard Controller Unit

This is the brains of the robot. Specifically, a TI OMAP platform named “Pandaboard.” The MCU has a flavor of Linux installed (Ubuntu 11.10) and runs a server which talks to the C Backend of the PC Unit. The server logic also forwards vehicular commands to the VCU. At one stage of development, we had attached a webcam to the MCU and streamed live feed using ffserver and

ffmpeg, however, the webcam didn't make it to the final deliverable due to reasons explained in the design details.

2 Design

2.1 Ice detection Unit

Different detection methods are considered for ice detection, namely, temperature sensor and infrared sensor. With temperature sensor, the implementation of ice detection would be more difficult since most of the temperature sensor entails physical contact with the object under test. The infrared sensor, on the other hand, could provide a reading without contact. In addition, according to Meitzler et al [5], IR emitter and sensor network have been successfully implemented to detect the thickness of ice. With numerous modifications, ice detection could be implemented as well. Therefore, an ice detection system will be implemented with IR emitter, and IR sensor (a photo-transistor).

2.1.1 Ice Detection Design-emitter

Before the actual detection of ice, the operation of major components (emitter, sensor) is tested. To test the IR emitter (LTE-302), a general purpose amplifier, 2N3904, is in place to provide a 20mA current to the IR emitter (see figure 4). To bias the transistor, a digital high (5 V) is assumed to be Vcc, and I_E is assumed to be close to I_E which is about 20mA when transistor is on. To calculate resistor values needed (R_1 and R_2) for biasing, the following equations are used, and the result simulation is shown in figure 5.

Assumed: $ie \approx ic = 20mA$, $Vcc = 5V$ $Vbe = 0.65V$ [3]

Given: $\beta = 100$ at $Vce = 1V$ $VonLED = 1.2V$ [1], $\beta \cdot ib = ic$

To keep V_{ce}=1V, $Vc - VonLED = 1V$ $Vc = 2.2V$

$Vcc - V_{R1} = 2.2V$ $V_{R1} = 2.8V = ic \cdot R1$

$R1 = 2.8V / 20mA = 140\Omega$, Since $R2 \cdot ib = R2 \cdot ic / \beta = Vcc - Vbe - VonLED = 3.15V$

$R2 = 3.15V \cdot 100 / 20mA = 1.575k\Omega$

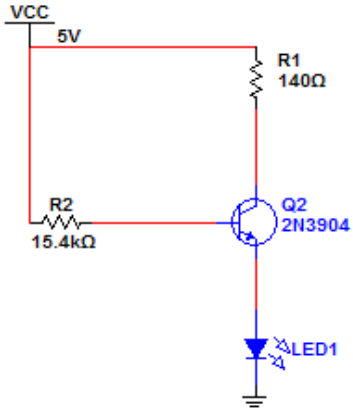


Figure 5 Transistor Circuit

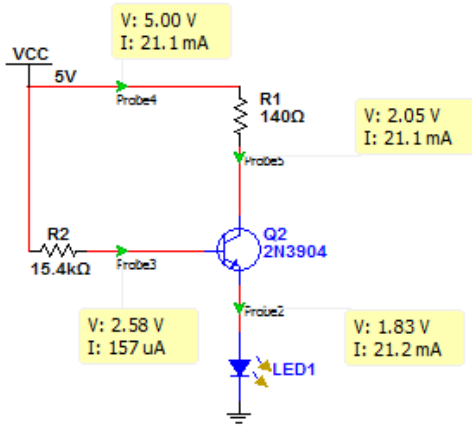


Figure 4 Transistor Circuit Simulation

As shown in the simulation, figure 5, the current provided to LED1 is about 20mA (21.2mA) and LED1 is on. However, because of using a different model than LTE-301 in modeling, voltage at LED1 is not 1.2V.

2.1.2 Ice Detection-sensor

To detect the ice, a sensor that responses to different intensities of IR signal is required. To implement, the counter part of the LTE-302 emitter, LTR-301 is used. The original setup of the sensor is shown in figure 6.

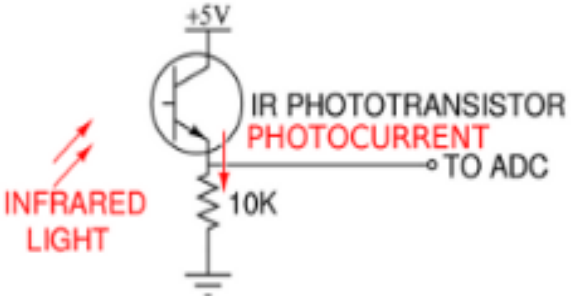


Figure 7 Original Sensor Setup

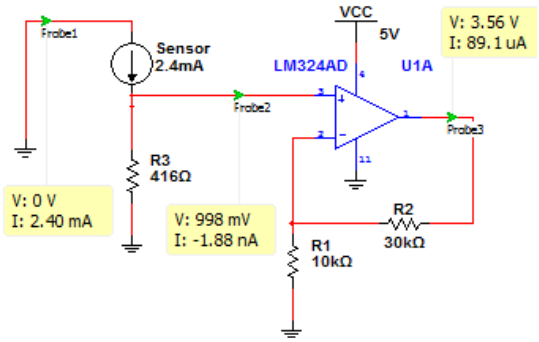


Figure 6 Sensor Setup Modified

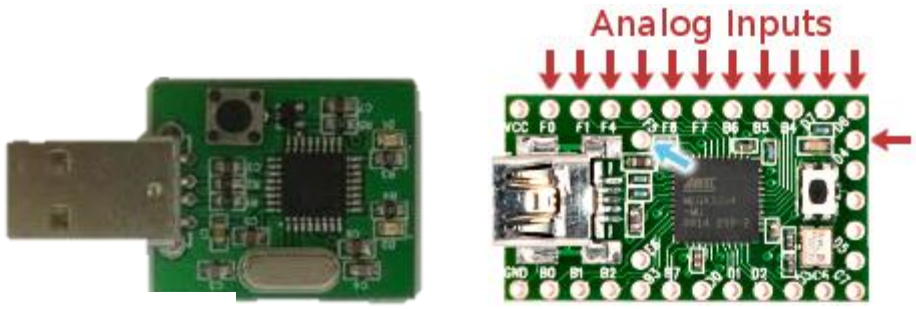
However, to accommodate the analog reading ability of teensy microcontroller (up to 5 volt), a resistor with lower value is been used. Moreover, to capture the change in the collector current more accurately, an amplifier, LM324, and an LED are used to amplify the subtle voltage change, as shown in figure 7. R2 and R1 are chosen so that a voltage gain of 4 is achieved by the amplifier. According to the LM324 datasheet, $Gain = 1 + R2/R1 = 1 + 3 = 4$ [4]. R3 is chosen so that the maximum collector current under 1mW illumination (2.4mA) times the voltage gain (4) is within 5V [6].

2.1.3 Ice Detection-system

The system for ice detection has not been tested. However, after the emitter and sensor are built, the system will be constructed by using different surface to see how much IR light is reflected.

2.3 Teensy Microcontroller

The teensy microcontroller is the brain of the detection and dispensing units. It is a complete USB-based microcontroller development system, in a very small footprint. All programming is done via the USB port. Specifically, in this project, teensy 2.0 is used because its ability to read analog input which is not possible for its ancestor teensy 1.0, see figure 8.



Teensy 1.0 **Teensy 2.0**

Figure 8 Comparison Between 2.0 and 1.0

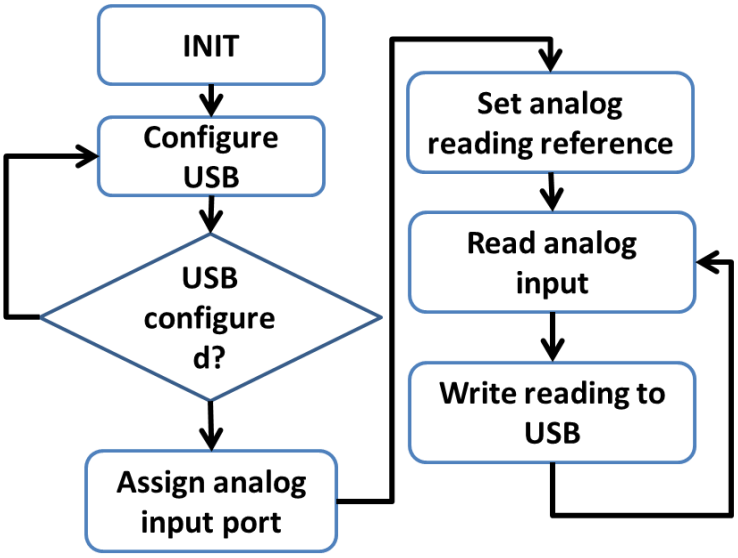


Figure 9 Analog Input Reading

2.3.1 Teensy Microcontroller Flowchart-analog input reading2.4 Vehicle Controller

Wild-thumper controller is our vehicle controller. It has dual 15A continuous H bridges to control the motor and the ability to receive command from Panda-board.

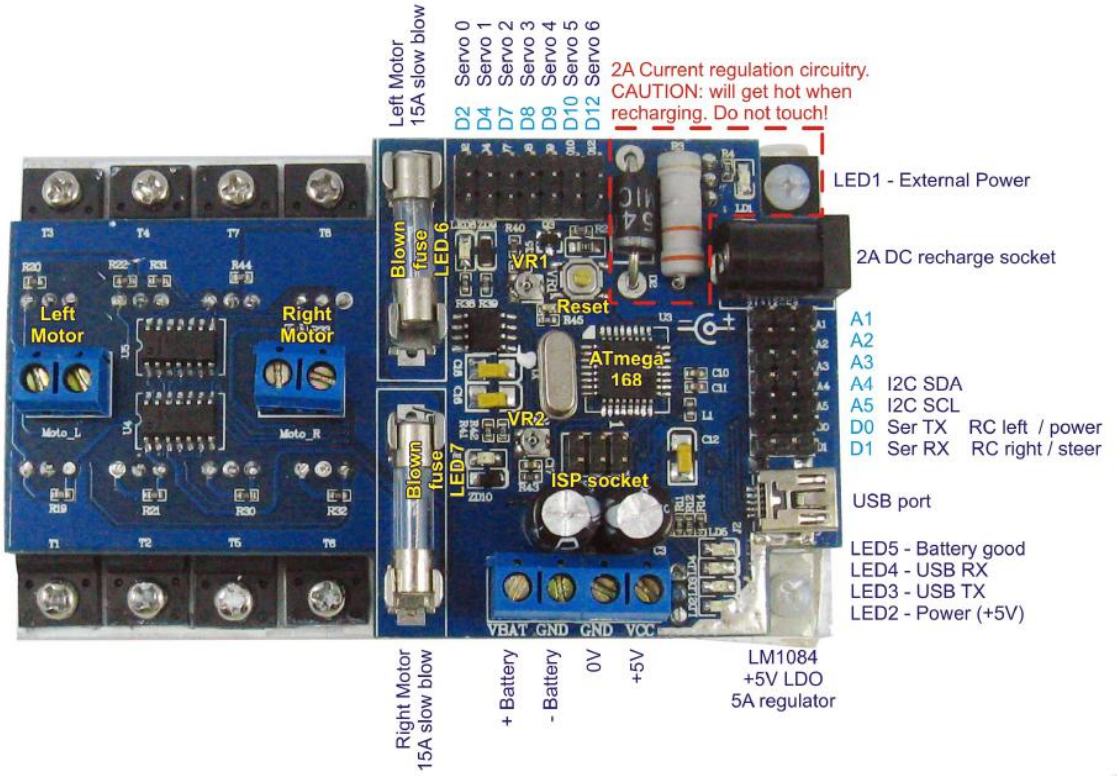
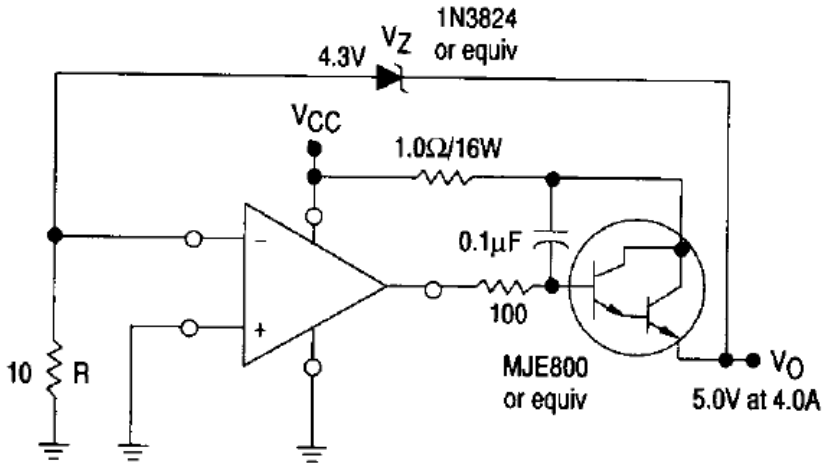


Figure 10

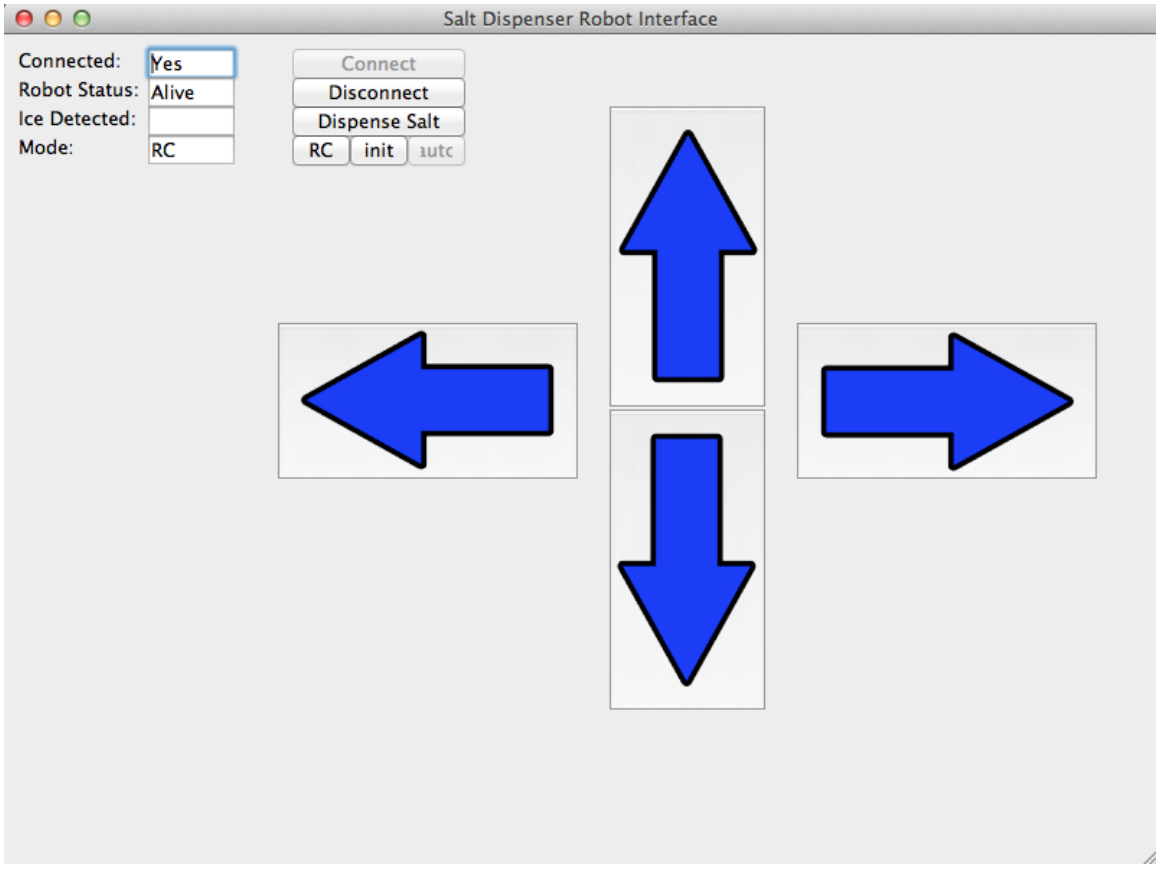
2.5 Power Supply Unit

Power supply unit consists of a darlinton pair transistor to increase current gain, it will supply 5V at 4A.



2.6 The PC Unit

Designing the GUI Interface



wxPython and the GUI

Python is a simple, high-level programming language. wxPython is a wrapper for Python of the GUI toolkit wxWidgets. I chose these because I believe Python is suitable for quickly coding high-level objects and without dealing with a lot of things seen in low level languages like memory management; wxPython is meant to quickly push objects onto the screen with simple parameters. Instantiating a button can be as simple as creating it in one line of code and binding it to a event handling function in another line of code.

The ability to switch modes is coded in Python. During the RC mode, arrow-key presses and arrow button clicks are send to the robot. During the initialization phase, the mode is similar to RC but actions are recorded. Finally, the autonomous mode is a playback of the directional commands send during the initialization phase, hence the robot being semi-autonomous.

For the PC to know whether the robot is still "alive," I send periodic "Are you Alive?" messages and wait for acknowledgement messages. I believe this system works as I give an adequate time of 6 seconds in case of network issues. If the 6 seconds elapse without acknowledgement, the robot is considered unresponsive and connection is terminated.

Designing the C Backend (client program)

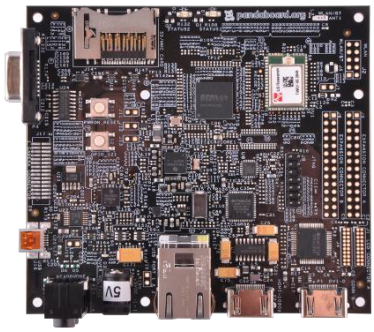
The backend of the Python/wxPython GUI is C code. It is there to connect to the robot over the network and handle communication. It is labeled as a client program as the robot is constantly running a server program that the client program can connect to. Messages such as the periodic "Are you Alive?" are passed from the Python frontend down to the C client backend and out through the network, and vise-versa. The program is mostly network socket setup and sending/receiving over the network socket.

2.7 The Wireless Router Unit

As this is a stock router with no real configuration other than setting up a WPA2 encrypted network, there is no real design details. All that is needed is the ability to allow static IP configurations on hosts, which is enabled by default.

2.8 The Mainboard Controller Unit (Pandaboard)

Setting up the Pandaboard



The contents of the Pandaboard box only include the board itself. Required components for setting up the Pandaboard include:

1. A 5V Power Supply.

The Pandaboard requires a 5V source of power, and a recommended device is http://search.digikey.com/us/en/products/PSAC30U-050/993-1019-ND/2384432.

5V/4A needs to be delivered, and the power input barrel specs are 2.1mm ID, 5.5mm OD with a positive tip.

2. An SD Card

A minimum 4GB SD card is recommended to hold the Linux OS.

We are using a 4GB SD/microSD card by Kingston:

http://www.newegg.com/Product/Product.aspx?Item=N82E16820134527

3. A Programming/Testing Desktop Environment

A simple HDMI cable is needed for video out to a display for programming and testing purposes. A USB keyboard and mouse are needed, as well. The final setup will idealy not include these, as these are not needed in the embedded system.

Getting the OS Image

In our setup, we chose the Ubuntu 11.10 distribution of Linux. The OS image needs to be made for Texas Instruments' OMAP platform. These can be found through http://omappedia.org/wiki/Ubuntu_flashing .

Putting the OS Image onto the SD Card

After downloading, the file needs to be unarchived if it was downloaded from Ubuntu, as they distribute gzipped images. Once unzipped, it is time to copy to the SD card. The commands used are OS-specific, and I since I did this on OSX, I provide OSX-specific instructions:

1. (insert SD Card)
2. Unmount the SD card:
    a. First, find out which device it is under /dev by running `diskutil list` and figuring out which "n" disk number it is
    b. Run `diskutil unmountDisk /dev/disk`"n" where is determined in the above step
3. Run the following: `dd if=`(the ubuntu image) `of=/dev/disk`"n" . The process can take quite a while, so don't assume it is hanging.

Look at http://www.embeddedarm.com/support/faqs.php?item=10 for similar instructions on Linux or Windows.

Powering the Pandaboard (test harness, not standalone battery power config)

The next part is powering up the Pandaboard. We used a 5V ~3amp power supply courtesy of the parts shop. My partner, Tim, did the wiring from AC outlet to the power supply's power-in/GND pins and DC power out wires to a barrel-style connector since the Pandaboard has a barrel-style power input. I would like to note that a 4 Amp supply is definitely recommended, even though sources on the internet may say that less power can be adequate. Less power is adequate if the Pandaboard isn't powering the development environment we have: HDMI out, wireless keyboard, wireless mouse. These consumed too much power that the Pandaboard would reset during OS installation, so I had to resort to only plugging in either keyboard or mouse.

OS and Network Related Setup

Once powered, the OS should boot and installation configuration options should be shown. You can go through these without much guidance as it only involves simple things like setting up time-zones. Once the desktop is reached, the right software must be present:

1. GCC toolchain. The Ubuntu OS came with this installed.
2. Text Editor. I installed Vim as it's terminal-friendly, and since I needed to make a choice of using a keyboard over a mouse. If you really want to use a mouse and also have inadequate amperage from your PSU like us, you can use an on-screen keyboard with the mouse to type.

Ensure the Pandaboard is connected to your home network. You will need to assign it a static IP on your home network so the PC will know what IP to connect to instead of telling it everytime in the case of a dynamic IP.

To assign a static IP, you can look at detailed instructions found here:

http://www.howtogeek.com/howto/19541/how-to-assign-a-static-ip-to-an-ubuntu-10.04-desktop-computer/

Note: your gateway is simply the IP of your router, or whatever is the last device in your network before going out to the internet. Whatever you set your static IP to, record it and add an entry to your PC's hosts file like this:

panda 192.168.1.150

Your PC's hosts file is simply a list of hostnames and what IPs they translate to you that you can personally define. Usually on unix and unix-like OSes, it is /etc/hosts. To confirm you've set it up correctly, have both PC and Pandaboard on the network, and ping "panda" or whatever hostname you used for your Pandaboard from your PC.

Setting up the Webcam

During our development phase, we had made progress with streaming webcam feed from the robot to the PC Unit. We worked with two different webcams, one rather ancient one (D-Link DSB C300) and one Microsoft LifeCam. The first webcam, when connected to the Pandaboard, was detected as a webcam type device in the Linux system message log, but no appropriate

driver was loaded. We could not get the webcam software v4l2 (video4linux2) to play with such an old device with the incorrect driver. After doing some research, I found out that it had an OV 511 chip inside, and found the newest drivers for it, which dated to 2006. These drivers were outdated as they cannot compile for the newer Linux kernel used in Ubuntu 10.10. After attempting to rewrite the old outdated drivers, my partner gave me a Microsoft LifeCam. Linux detected correct drivers for this right away, and I was able to move on to the second stage. Here I used ffmpeg to capture data from the webcam and stream it with ffserver. After a day's worth of playing around with settings, network bandwidth, and rethinking, I was able to reduce the streaming lag from the webcam to around twenty seconds. This was a completely unacceptable lag that forced us to drop the webcam component. The webcam driver, ffserver/ffmpeg, or both were performing too poorly on the embedded platform to deliver anything close to a live video feed.

3. Design Verification

In the design verification phase, the expected performance of each component is quantified and organized. Verification phase is a critical phase that ensures basic component-wise requirements are met. By doing verification from bottom up, debugging should be kept at a minimum.

3.1 Ice detection Unit Verifications & Results

Please see appendix A for verification and results.

3.2 Teensy Microcontroller Verifications & Results

Different analog inputs are successfully read, see figure 10, 11, 12, 13.

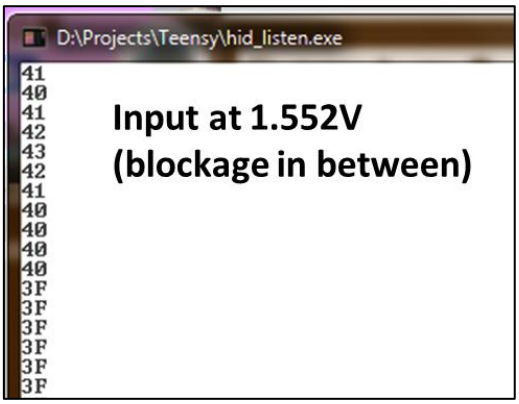


Figure 12 Input At 1.552V

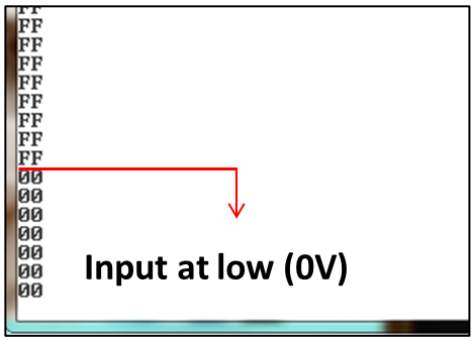


Figure 11 Input At 0V

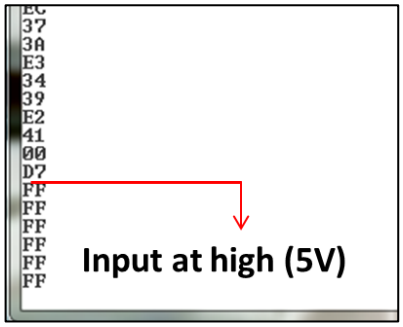


Figure 14 Input At 5V

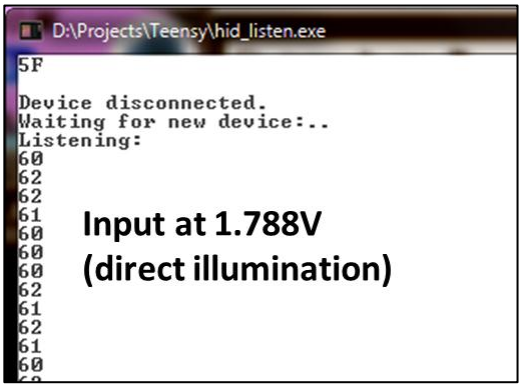


Figure 13 Input At 1.778V

An easy conversion can be made to obtain the correct analog value, using hex value FF and the corresponding voltage 5V as a base for comparison. First FF_{hex} is converted into 255 in decimal. Second, the analog reading is calculated using the following formula,

$$Vout = 5 \cdot \frac{reading\ converted\ to\ decimal}{255}$$

The reading for input of 1.552V is then calculate to be 1.255V and 1.88V for input of 1.788V

3.3 PC Unit, MCU, and Communication Verifications & Results

Modular Testing Procedures

Condition	Testing Procedure
1. Commands recorded correctly during initialization phase	1. Does the data recorded match keys pressed on keyboard?
2. Python code can talk to backend C code	2. Similar to UNIX pipe, can I pipe stdin/stdout?
3. Does the PC establish a connection with Pandaboard and communicate well?	3. Send 100 commands and see how many are received
4. Auto mode path is correct?	4. Make sure the commands send during auto mode are the same as initialization mode

Quantitative Results (of above testing procedures)

Test 1

Yes, positive results. Commands saved to text file and comparison shows every directional command sent is recorded. The test was tried three times with varying amounts of directional commands.

Test 2

Yes, positive results. Interface can read messages form backend and write out commands. Flushing buffers is required.

Test 3

Yes, positive results. Thanks to TCP, commands send were received in-order and all 100 were received.

Test 4

Yes, positive results as expected after Test 1. File I/O is almost guaranteed to not fail. Commands recorded to file were successfully read back during auto mode.

4. Costs

4.1 Parts

Part	Unit Cost	Quantity	Actual Cost
Teensy Microcontroller	$16.00	1	$16.00
Breadboard	$15.00	1	$15.00
LTE-302 IR Emitter	$1.95	2	$3.90
LTR-301 IR Photo-detector			
2N3904 General purpose transistor	$0.10	2	$0.20
DC motor	$1.95	1	$1.95
LM324 Amplifier	$0.95	1	$0.95
Pandaboard	$250.00	1	$250.00
Resistors	$0.10	15	$1.50
MC3401 Amplifier	$0.50	1	$0.50
MJE800	$1.00	1	$1.00
Thumper-controller	$150.00	1	$150.00
		Total	**$441.00**

4.2 Labor

Member	Salary/hour	Hours	Total
Naman	$50	80	$8000
Chun-Ting	$50	80	$8000

4.3 Total Cost

Labor cost	Part Cost	Total
$16,000.00	$441.00	$16441.00

5. Conclusion

Intensive research is done on methodology of ice detection to justify the approach to ice differentiation. Temperature sensing was first considered, but not implemented due to the contact requirement of the device. IR sensing, on the other hand, was proven valid by Moss et al. The basic emitter and sensor were built and tested for their proper operation. A teensy microcontroller captures analog output from the sensor, and digitized it for further processing. The wireless communications between controllers is established and tested.

5.1 Accomplishments

The approach of ice detection is confirmed, basic detection circuitry is constructed and microcontroller's analog reading is successful. Detection circuit is able to differentiate between objects with high reflectivity (potential ice) and objects that are not so reflective. Wireless communication is established between MCU and the PC Unit, and software on the MCU is able to forward commands to the VCU successfully.

5.2 Uncertainties

Although ice detection circuit is successfully constructed, the detection of ice is not completely robust since our design only focuses on ice sheet (ice that has a flat surface), any curvature on the surface will significant effect the result of the detection.

The semiautonomous mode operation still does not make the robot fully autonomous since it requires initialization every time the course/path is altered. However, for the same environment (a driveway, for example) the operation is sufficient.

5.3 Ethical Considerations

On the ethical side of things, we have to think about the safety of the end user. The end user should not be harmed by usage of our robot. For example, exposed wires should be reduced so to not accidently shock the end user. The software required to control the robot should not contain harmful code such as malware. What is promised to the end user should be delivered within the reasonable bounds of advertisement, such as accuracy of the ice detection.

5.4 Wrap-up and Future work

Building on the current deliverable, we can see the addition of the mechanical dispensing unit to the robot as well as integrating the ice detection unit onto the robot. Bigger steps might be to make the robot fully autonomous using a lot more processing power, code, and time. Image processing and logic behind a fully autonomous mode would be a research topic of its own. True ice detection (at the material science level) would be a research project of its own, as well. After reading the report of traffic Light de-icer, we believe we can modify our ice detection circuitry to be more accurate and durable. The improvements would involve an IR camera and software to do the distinguishing of the ice from dirt and other various outdoor elements. This way, we avoid the calibration involved without current setup.

References

[1] *LTE-302 Datasheet*, Lite-on electronics, Inc.

[2] *LTR-301 Datasheet*, Lite-on electronics, Inc.

[3] *2N3904 NPN General Purpose Amplifier Datasheet,* Fairchild semiconductors. 2011

[4] *LM324 Low Power Quad Operational Amplifiers,* Texas Instrument. 2011

[5] *An Infrared Solution to a National Priority NASA Ice Detection and Measurement Problem*. Thomas Meitzler, MI. 2007

[6] Teensy microcontroller, http://www.pjrc.com/teensy/

[7] The wxPython API found at http://www.wxpython.org/docs/api/

[8] Python 2.7.2 Documentation found at http://docs.python.org/tutorial/inputoutput.html

[9] Beej's Guide to Network Programming http://beej.us/guide/bgnet/

[10] Rochkind, Marc J. *Advanced Unix Programming.* Publication Date: May 9, 2004 | ISBN-10: 0131411543. Addison-Wesley Professional; 2nd Edition.

[11] Pandaboard Ubuntu Pre-built Binaries Guide. http://omappedia.org/wiki/Ubuntu_Pre-built_Binaries_Guide

[12] Python Subprocess information found at http://www.doughellmann.com/PyMOTW/subprocess/

[13] C Programming Reference found at http://www.cplusplus.com/reference/

[14] Kernighan, W. Brian. Ritchie, M. Dennis. *The C Programming Language.* Publication Date: April 1, 1988 | ISBN-10: 0131103628. Prentice Hall; 2nd Edition.

Appendix A Requirement and Verification Table

System Requirements and Verifications

Requirement	Verification	Verification status (Y or N)
1. Ice detection a. Emitter Von ~1.2V measured b. Vbe ~0.65V measured	1. Verification a. 1.19V b. .67V	 Y Y Y
2. Teensy microcontroller a. ~5V at Vcc	2. Verification a. 5.15V	 Y