A WIRELESS CONTROLLED ROBOT REAL-TIME SYSTEMS LABORATORY

BY

JOSEPH FRANKLIN GIROTTI

THESIS

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science in the Graduate College of the University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Adviser:

Professor Lui Sha, Donald B. Gillies Chair
ABSTRACT

During the spring of 2011 pressure was asserted by the department on the enrollment in CS424 Real-Time Systems due to its low enrollment. The result of that pressure was a curriculum update, removal of the CS431 Embedded Systems course requirement, and the inclusion of a laboratory section similar to CS431 which had superior enrollment. This work is the result of the development of the laboratory section of CS424 Real-Time Systems.

Early development created a robot and phone based platform. The usage of Commercial-Off-The-Shelf components allowed for rapid prototyping, allowing the platform to be fielded Fall 2011. A theme was created to link both lecture and laboratory work, with students performing presentations in class on the design of their laboratory demonstrations. Issac Asimov’s three laws of robotics were modified to better match with the classes core topics of safety, reliability, and performance.
To my mother, for her patience and support, Dr. Susan Linnemeyer, for being there for me in my darkest hour, and Prof. Sha, for this excellent opportunity.
This project was a team effort on many levels and thus acknowledgement must be given to the many individuals involved.

Dr. Lui Sha and Dr. Tarek Abdelzaher are first. Dr. Sha was both my advisor and the first individual to teach with the new laboratory. His council, and the opportunities he has granted during my tenure underneath him have been invaluable. Dr. Abedelzaher oversaw the first part of the development of the platform used in the course and was a driving force behind the overall course update.

The platform itself was initially developed by Roberto de Matos. The enclosure was designed by Or Dantsker. Finally much of the modifications to the Linux configuration were done by Hongyan Wang.

Also my family and friends who have supported me patiently as I pursued my academic career, including but not limited to: my mother Rachel Girotti; my brother John Girotti; my sister Jennifer Girotti; and finally Trish Barker of the NCSA.
# TABLE OF CONTENTS

CHAPTER 1  INTRODUCTION ............................................. 1
  1.1 Motivation .................................................... 1
  1.2 Lab Requirements ........................................... 2
  1.3 Initial Results .............................................. 3
  1.4 Overview ...................................................... 3

CHAPTER 2  DESIGN ...................................................... 4
  2.1 Robot and Mobile Platform .................................. 4
  2.2 Curriculum .................................................. 5

CHAPTER 3  IROBOT CREATE .......................................... 8
  3.1 Reference Documents ......................................... 8
  3.2 iRobot Create ................................................. 8
  3.3 Create Open Interface ...................................... 9
  3.4 Sensors ...................................................... 9
  3.5 Interconnect Cable ......................................... 10

CHAPTER 4  BEAGLEBOARD ........................................... 11
  4.1 Reference Documents ......................................... 11
  4.2 BeagleBoard ................................................ 11
  4.3 Linux Configuration ......................................... 11
  4.4 Pyrobot ..................................................... 12

CHAPTER 5  ANDROID ................................................... 13
  5.1 Reference Documents ......................................... 13
  5.2 Android ...................................................... 13
  5.3 Android Software Development Kit .......................... 13

CHAPTER 6  SAMPLE LABORATORY ASSIGNMENTS ................. 14
  6.1 Machine Problem 0 ............................................ 14
  6.2 Lab 1 .......................................................... 17
  6.3 Lab 2 .......................................................... 20
  6.4 Lab 3 .......................................................... 24
  6.5 Lab 4 .......................................................... 25
  6.6 Lab 5 .......................................................... 27
CHAPTER 1
INTRODUCTION

During the 2007 to 2008 academic year I enrolled in both CS431 Embedded Systems and CS424 Real-Time Systems. Based on this performance I was offered an undergraduate research position in what is now the Cyber Physical Integrations Lab (CPLI). After my graduation in spring 2009 I continued this position as an Graduate Research Assistant, until fall 2011 when I formally joined the lab as a Masters of Science candidate. During this time I assisted with the upkeep of the CS431 lab including hardware repair and the training and assistance of graduate teaching assistants, as well as several projects.

When it became necessary to upgrade hardware for the lab first in spring 2010, then a full upgrade over the 2010 to 2011 academic year I was part of the team to develop the replacement for the old hardware and assist in updating the curriculum appropriately. Finally at the beginning of the fall 2011 semester I took over the development of a lab for CS424 which this thesis details.

1.1 Motivation

CS424 Real-Time Systems was setup as a follow up class to CS431 Embedded Systems course. These two courses combined covered the basic concepts that CPLI works with. CS431 has a lab synchronized with the lectures. CS424 class was taught as a typical lecture and theory course. The problems began with spring of 2004. If we look at Figure 1.1 [1, 2] during this time total enrollment in general follows graduate enrollment in the course, with the exception of spring 05. More over a drastic reduction in graduate enrollment led to a level of total enrollment that caused the department to question the existence of the course. In response it was proposed to perform curriculum improvement on the course.
It was decided that this curriculum improvement would be done on the course for the fall 2011 term. Though it was there was no definite cause for the drop in graduate enrollment, the fact that total enrollment followed graduate enrollment was itself a problem. Thus one of the first decisions was to make the course more like CS431 which as we can see from Figure 1.2 has a total enrollment that follows undergraduate enrollment.

Several steps were taken the first was to remove the CS431 requirement for CS424 which would allow students to directly enroll in CS424. This of course required curriculum change since Real Time scheduling was covered in CS431 and is now the single point of overlap in the two classes.

1.2 Lab Requirements

The final and most important step was to develop a lab to provide students hands on experience with implementing what they learned in class and provide a real world example for course material. The first would be accomplished through the design of an engaging platform without the need to learn the low level programming seen in CS431, and providing an interesting design space. The second was accomplished by designing labs that fit within the primary themes of CS424 which are Safety, Reliability, and Performance, Real Time Systems, Wireless and, Distributed Systems, Energy Management.

Figure 1.1: CS424 Enrollment: fall 2013 is speculative based on current enrollment in the Banner system
1.3 Initial Results

The new labs were initially fielded in fall 2011 coupled with aggressive student outreach. Student response to the lab exercises was positive. The following semester students were both informed of the curriculum change prior to course registration, as well as word of mouth by students from the previous resulting in a near 100% increase in enrollment attaining close to the 2003 enrollment. This though could be an outlier, but a key observation when taking into account registration for the fall 2013 class we can see from Figure 1.1 that the core enrollment of the class has transitioned from one that follows graduate enrollment to one that follows undergraduate enrollment much as CS431. If this trend continues the course should be on a sustainable path.

1.4 Overview

The remainder of this thesis presents the design and implementation of a new robotics and mobile platform and the associated laboratory curriculum. Chapter 2 discusses the design for the new hardware and curriculum. Chapters 3 through 5 detail the elements of the platform which is made up of an iRobot Create, BeagleBoard, and Android Phone and its use. Chapter 6 contains the new laboratory curriculum as a set of sample laboratory assignments. Appendix A is a introductory and troubleshooting manual for the platform for students to use. Finally Appendix B includes important code used on the BeagleBoard to connect to the network.
CHAPTER 2

DESIGN

Initial design for the enhanced iRobot was performed by Roberto de Matos, and the curriculum by myself. After initial fielding there was a slight rework of the design, as such I will discuss the initial design and basis for the changes in the first section. The second section details the design principles for the curriculum which was assisted with by Professor Lui Sha. The physical hardware is made up of an iRobot Create connected to a BeagleBoard with a WiFi dongle allowing it network access and connectivity to a Smartphone which in the case of the lab is an Android phone.

2.1 Robot and Mobile Platform

The initial design for the platform was most certainly inspired by the Green GPS project under Professor Tarek Abdelzaher. This project coupled real cars with smartphones to compute more fuel efficient routes for drivers. The iRobot Create mimics the car, and the Android Smartphone plays much the same role. The major difference is the addition of the BeagleBoard which is capable of both connecting to the Create as well as the network letting the Android phone stay in the users hand.

The iRobot Create is a small three wheeled robot with an impressive array of basic sensing capability. The BeagleBoard is one of the new generation of small solid state computers such as the PandaBoard, and the popular Raspberry Pi. It comes with a proprietary Linux known as Angstrom. When connected serially through the expansion header and UART protocol it can send commands to the Create through iRobots Create Open Interface which is an extension of their Serial Control Interface. A Python library was developed separately by Damon Kohler at MIT in 2007 [3]. This create objects for most of the commands that can be called rather than having to format
the serial string by the programmer greatly reducing the learning curve to command the robot. The BeagleBoards USB allows attachment of a WiFi dongle and Linux’s network manager allows the robot to logon to a WiFi network. The Android also has the ability to logon to the WiFi network and sending information to and from the BeagleBoard and Android is as simple as opening a socket to send the data through.

After initial fielding of the platform several problems were found, and we were requested to have the robots connect using a University WiFi rather than a private WiFi to prevent interference with Illinoisnet. Most of the problems stemmed from the hasty assembly of the platform. The BeagleBoard though solid state is not much more than a naked motherboard, and was originally mounted directly to the Create. This allowed possible shorts from static electricity when students handled it, as well as possible damage if the robot was dropped or generally mistreated, such as being stepped on.

Also the additional ultrasound sonar sensor attached to the system was found to be unsatisfactory for our desire for the robots to be able to detect each other. With these problems in mind the redesign included modification of the cable connecting the Create, Beagleboard, and Sonar sensor. A different model of ultrasound sensor was also used that had a wider near field vision. Finally a protective case was designed out of impact resistant plastic by Or Dantsker, which when a robot accidentally backed off a ledge did a superb job of protecting the BeagleBoard.

2.2 Curriculum

As mentioned in Chapter section 1.2 the the primary themes of CS424 are:

- Safety, Reliability, and Performance
- Real Time Systems
- Wireless and, Distributed Systems
- Energy Management

For curriculum design the goal was to cover these topics with lab exercises. It was also decided that the students would benefit from the creation of an
overarching theme to link the lecture and lab sections. This theme would be inserted into the previous lecture content.

The theme that was established was based on Asimov’s three laws of robotics, but modified to better fit the class. Asimov’s laws made human safety critical [4]:

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.

3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

In real world systems a human may not in fact be safety critical, or at least the most critical. Also the lab robots were not capable of making the complex choices about what is dangerous to humans and what is not. Instead we rearrange the laws to more appropriately follow conventional cyber physical design of safety, reliability, and performance:

1. Safety: A robot must not violate it’s safety parameters.

2. Reliability: A robot must protect its own existence and function as long as such protection does not conflict with the First Law.

3. Performance: A robot must obey the orders given to it, except where such orders conflict with the First or Second Laws.

An important thing to note here is human is not used in the modified laws, this is because a robot could in fact be receiving commands from a non-human system.

The hardware setup and theme are best applied to the primary topic of the course Safety and Reliability which can also be difficult since unlike the other topics is an abstract concept rather than mathematical formulation. Also labs have been designed to allow students to build an understanding of the platform, rather than expecting them to pick up the intricacies of the entire platform at once. Below is an overview of the labs and their coinciding course topics.
• Machine Problem 0: an introduction to Python, several small exercises to familiarize students with the Python language preparing them for labs 1-3.

• Lab 1: Moving and Sensing with the robot, students program a fixed course that the robots must navigate using their sensors. This covers the topic area of Safety, and Reliability, or the first two laws.

• Lab 2: Commanding the Robot, students learn to connect their Android phone to command the robot. This lab prepares students for Lab 3.

• Lab 3: Simplex, students learn to put Lab 1 and Lab 2 together to create a robot that obeys the modified three laws

• Lab 4: Distributed Sign Wave, students write code where the computation of the drawing of a sign wave is distributed across multiple threads and sent to a server. This covers Real Time Systems, and Distributed Systems

• Lab 5: Wireless and Power, the idea was for students to use the robots and their WiFi to compare power costs of movement versus network communication. This would have covered the topics of Wireless, and Energy Management but unfortunately the robots removal from the private WiFi to IllinoisNet rendered the sensed, and computed results to be non-useful to the students. A replacement lab is being investigated.
CHAPTER 3

IROBOT CREATE

This chapter covers the iRobot Create, Sensors, and the custom interconnect cable that links the Create, external Sensors, and Beagleboard.

3.1 Reference Documents

Important refernce documents for the Create, it’s Open Interface, the Sonar sensor, and components of the interconnect cable are:

- iRobot Create Owners guide [5]
- iRobot Create Open Interface [6]
- MB1200 XL-MaxSonar-EZ0 [8]
- Polulu Power Board [9]
- Logic Level Converter [9]

3.2 iRobot Create

The iRobot Create is a wheeled robot derived from their popular Roomba platform. It has an array of sensors and an serial Open Interface that is decended from the previous Serial Command Interface. The Open Interface allows external systems to send commands to the robot serially through either a 7 pin Mini-Din connector or a the Cargo Bay connector a DB-25 connector.
3.3 Create Open Interface

The Create Open Interface is a serial interface that allows control of the robot and reading of it’s sensors. These commands are in the form of serial commands sent to the robot and serial responses sent to the controller in the case of sensor polling. In the CS424 lab the students use a Python library to call these commands as objects, removing the need to format the serial commands. That library will be discussed in the next chapter.

3.4 Sensors

The Roomba and Create have several sensors that are built into them, and in the case of the Create it also has four digital inputs, an analog input, three digital outputs, three high-current low side driver outputs available through the Cargo Bay Connector. In the CS424 lab we have attached a MB1200 XL-MaxSonar-EZ0 sensor to the analog input allowing the students to gauge range. The list of sensors currently used in the CS424 labs include:

1. Bump
2. Wheel Drop
3. Wall
4. Cliff (Left, Front Left, Front Right, Right)
5. Distance
6. Angle
7. Battery Charge
8. User Analog (Sonar Sensor)

The bump sensor is triggered when the robot contacts an object from the front. The wheel drop sensor is triggered when the wheels lose contact with the ground. The wall sensor returns a value that allows hugging a wall to the robot’s right side. The cliff sensors trigger when each sensor, which are located at the front of the robot, is over a ledge. Distance, Angle, and Battery Charge return appropriate values for the robots current state. Finally User
Analog returns the voltage value received from the EZ0 sonar sensor, which correlates to the robot's distance to any object in front of it.

### 3.5 Interconnect Cable

The Create, BeagleBoard, and Android phones do not directly require modification simply configuration or programming to function. A custom cable was required though to connect several components of the system:

- The Pololu Step-Down Voltage Regulator to power the BeagleBoard at 5V
- The MB1200 XL-MaxSonar-EZ0
- The Logic Level Converter which shifts the voltage to the serial connection to the BeagleBoard

Figure 3.1 shows the proper wiring for the cable. The cable requires wire, a DB-25 connector, and a 2x14 Flat Ribbon connector to connect to the BeagleBoard Expansion header.
This chapter details the BeagleBoard, Linux, and Pyrobot Python library used to control the robot.

4.1 Reference Documents

Here is a list of important reference documents for the BeagleBoard, Linux, and Pyrobot Python library.

- Ubuntu [11]
- Ubuntu Omap/Arm [12]

4.2 BeagleBoard

The BeagleBoard-xM is technically a development board for an ARM Cortex A8 processor. It is a single board containing processor, ram, etc. in solid state with the capability of using a Micro-SD card for storage. It’s small size, solid state, low power, and cost make it ideal for use as a computation platform for small projects.

4.3 Linux Configuration

A major issue though is that ARM is a different architecture than the standard x86, therefore operating system support is limited. Angstrom a version of Linux is the default image for the BeagleBoard. There is also community
support for several other Linux flavors. It was decided to use Ubuntu 12.04 LTS since unlike Angstrom it is open source, rather than proprietary, and has a large amount of community support. This image does not include a GUI to minimize space and computation requirements.

As such several steps must be taken to get the robots to connect to Illinoisnet upon boot. The first is to configure the network interfaces, and wpa_supplicant to access the network. Next we need the robot to report its IP to an external source so that students can SSH into them. A SQL database was created that’s contents are reported to a PHP page that students can view. To get the robots to report their IP to this database we need a shell script that will run automatically and code to access the database.

All these files are available in Appendix C.

4.4 Pyrobot

The Linux OS coupled with the Pyrobot library allows us to write and run Python programs to communicate and control the Create. The Pyrobot library creates an object Roomba and then extends it to a Create Object. This object holds class objects allowing calls to specific functions of the Create Open Interface. Rather than require students to alter this file, which could result in errors, instead an additional class Rtbot is created by extending the Create class.
CHAPTER 5

ANDROID

This chapter details the Android platform used in class.

5.1 Reference Documents

Important information for the Android platform and it’s SDK can be found at:


5.2 Android

The Android Open Source Project is an open source project led by Google. The purpose of which is to design a software stack for mobile devices specifically phones. There are many Android devices available from phones, to tablets, and with the release of the Ouya game consoles. It is more a family of hardware that supports the Android environment than a particular architecture.

5.3 Android Software Development Kit

The Android Software Development Kit(SDK) provides access to libraries and developer tools necessary to develop Android Applications. By default it uses the Eclipse Integrated Development Environment. It also includes an Android Emulator to test applications.
CHAPTER 6
SAMPLE LABORATORY ASSIGNMENTS

This chapter contains one introductory machine problem, and five sample laboratory assignments intended to be assigned over the course of the semester. Students are generally given two weeks per exercise, with the machine problem being assigned the first week of the course and laboratory exercises following. The MP0, Lb1, Lb2, and Lb3 build on each other and interact with the course through student discussion and presentations.

A special note on Lab 5. The intent is to have a lab covering wireless and power as per the lectures. Unfortunately the current platform does not lend itself well to the format of L5. The movement from a single private WiFi network to IllinoiNet, and the small lab space make it incapable of receiving interesting data in the lab, effectively negating it’s purpose. Therefore it is the intent to create a different lab to fill this curriculum slot. Lab 5 is included in this Thesis as a matter of record.

6.1 Machine Problem 0

6.1.1 Overview

In this lab you will be introduced to using Python.

You must write a Python program that does all of the following.

1. print the string ”Hello World!”

2. Receive an integer from the command line and compute the Fibonacci sequence up to that number. It should look like

   username@computername:PATH$ python Lab0.py 4
   Hello World!
   {0, 1, 1, 2, 3}
3. Contains an internal dictionary

```python
seat_roster = {1: 'Ted', 2: 'Bob'}
```

4. Prompt the user to input an integer with

"Enter an integer: "

- Catch if the input is a valid integer if so continue
- If it is not reply with

"Please enter a valid integer"

and prompt the user again for an integer

5. Use the integer to check if that seat is filled.

- If it is reply with

"Seat # X is in the roster. Y is sitting there"

where X is the seat number and Y the name of the person

- If it is not reply with

"Seat #X is not in the roster"

where X is the seat number

6. Prompt user to enter command, and evaluate the string as a Python command. You should check that it is a valid Python command.

- If it is evaluate the command and prompt for another command
- If it is not reply with

"Please enter a valid command"

and prompt for another command
6.1.2 Procedure

1. Before getting started, see sections 1 through 5 in the Python tutorial (available at Python.org).

2. Prepare for Lab 0.
   - Open your preferred editor for Python. Any text editor will do, but you may want one that at least provides code highlighting
   - Save the file as Lab0.py

3. The print() command will print a string below are some examples
   - print "Hello World!"
   - print "My shoe size is %i" % shoe_size
   - print "I am %i foot %i inches tall" % (height_ft, height_inch)
   - print "I am", height_ft, "foot", height_inch, "inches tall"

4. Use import sys at the top of your file. This will give you access to sys.argv[] function which will allow you to capture command line arguments. The first element sys.argv[0] will be the file name you open with the when you call python filename argv[1] argv[2] etc.

5. See section 4.6 of the Python Tutorial for two examples of computing the list of Fibonacci numbers. Make sure to follow the formatting given above.

6. Copy the dictionary definition from Overview #3

7. The raw_input() function allows you to both print a prompt and receive a string from the user, and is the suggested method for user input.

   s = raw_input("Prompt Input")

8. The input from a input() is a string. Converting a string to an integer can be done with the int() function.
9. The int() function can fail so you will need to catch the exception. The 
try and except construct allows this and is covered in Section 8.3 of the 
Python tutorial.

```
try:
    some_function(value)
except:
    handle_exception()
```

10. Since you want to prompt again for an integer if int() fails you may 
wish to create a function definition so that it may be called again in 
case of an exception. Function definition is covered in section 4.6 of 
the Python tutorial. Generically function definitions look like:

```
def function_name(args)
    ... function code
```

11. Dictionaries can be checked for the presence of a particular key with 
the dictionary_name.has_key(key_val) function which returns a boolean

12. The exec() function takes a string and executes it in Python, allowing 
dynamic input of Python commands midway through the program. It 
can fail like the int() function so make sure you catch any exceptions.

6.2 Lab 1

6.2.1 Overview

In this lab you will be introduced to using the PyRobot library. 
You must write a Python program that does all of the following.

1. Run an obstacle course as per your TA’s design which will include

   • Moving the robot forward, backward, and turning.

   • Moving the robot dependent on sensors including the cliff, ultrasonic 
     rangefinder, bump, wheel drop, distance, and angle sensors.
6.2.2 Procedure

1. Before getting started, see section 9 "Classes" in the Python tutorial (available at Python.org).

2. Prepare for Lab 1.
   - Download Pyrobot.py and Rtbot_blank.py from the resources page of the class webpage, and Lab01_blank.py from the labs page.
   - Open your preferred editor for Python. Any text editor will do, but you may want one that at least provides code highlighting.
   - Open the Rtbot_blank.py and Lab01_blank.py files provided on the site.

3. As you can see Rtbot extends the Create class (which extends the Roomba class) from Pyrobot. This is where you will want to place any additional functions that you create. Lab01_blank.py should be used to run functions to perform the actions necessary to run the course.

4. To connect to the robot find your robots IP by going to http://cs424.web.cs.illinois.edu/

5. Use the ssh command to connect to a robot with the passphrase demoroomba

   ssh student@192.168.1.<WiFi IP>

6. Use the scp command to download files to the robot. Note: most Linux machines do not allow you to scp files to them. The robots are configured to allow this, so ”push” files from your computer to the robot.

   /your/directory# scp file.name root@192.168.1.<WiFi IP>:/some/directory/on/robot/

7. When ready you can run your code by using the following command
python Lab01_blank.py /dev/ttyO1
argv[2] argv[3] etc...

NOTE: that is ”O” not zero. the additional argv[x] arguments may be blank or additional commands for your code. e.x. you could pass the distance you want the robot to travel as argv[2] like this

python Lab01_blank.py /dev/ttyO1 300

8. The Drive(self, velocity, radius) in PyRobot will move the Create as per its description. You can call it in your Lab01_blank.py code using the following syntax

    robot.Drive(velocity, radius)

    Drive(robot, velocity, radius)

you may call it in a function in the Rtbot class using the following syntax

    self.Drive(velocity, radius)

    Drive(self, velocity, radius)

the following functions are merely sub-functions of Drive but are useful

    • the DriveStraight(self, velocity) function drives at an angle of zero.
    • the TurnInPlace(self, angle) function turns without changing its displacement.
    • the Stop() and SlowStop() functions stop the robot.

9. The sensor functions can be called with the following syntax

    robot.sensors.sensorfunction(argvs)
you may call it in a function in the Rtbot class using the following syntax

```python
self.sensors.sensorfunction(argvs)
```

- **GetDistance()** returns the distance the robot has traveled and resets the counter.
- **GetAngle()** returns the angle the robot has traveled and resets the counter.
- **GetAnalogInput()** returns an integer value for the analog to digital encoder and resets it.
- **GetBump()** returns a boolean dependent on the state of the bump sensor and resets it.

10. There are no functions to get the just the cliff or wheel drop sensors so you may want to write them using the `GetAll()` command. After using the `GetAll()` command you can access sensor values using the following syntax.

```python
self.sensors.data['sensorname']
robot.sensors.data['sensorname']
```

you can find the sensor names by looking at the `DecodeGroupPacket0` or `DecodeGroupPacket6` functions as seen in `pyrobot.py`

### 6.3 Lab 2

#### 6.3.1 Overview

In this lab you will be introduced to using the android phone and communication. You must write Python and Java code that does all of the following.

1. Connect an Android phone to the Create and send and receive data.

   - Command the robot to go forward, backward, left, right, and stop
   - Commad the robot to kill its threads
6.3.2 Procedure

Robot

1. Before getting started, see section 7.8 queue, 16.2 threading, 20.19 socketserver, and 17.2 sockets in the Python library reference (available at Python.org).

2. Download and configure Eclipse and the Android SDK.

   http://www.vogella.de/articles/
   Android/article.html#installation

   has a short installation guide and an intro to Adroid projects on Eclipse

3. some definitions will prove useful so add

   HOST = ''
   PORT = 50000

   to the top of lab01_blank.py and name the file Lab02.py

4. and add

   BUFFER_SIZE = 1024
   COMMANDS = []
   ACK = "ACK\n"
   CONFIG_CMD = "__cfg__"

   to the top of your rtbot.py file

5. We will need to run two threads at a time, the SocketServer and Robot-Controller. The SocketServer handles message passing, while the Robot-Controller handles commanding the robot.

6. use the SocketServer python command to create a TCPServer

   server = SocketServer.TCPServer((HOST, PORT), TCPHandler)

   • you will need to write the TCPHandler class in rtbot.py to receive commands. We will discuss this later in the procedure.
- The server should also allow reuse address

7. Next we need to create a thread for the `SocketServer`.

   - The `threading.Thread` python command
   - It should serve_forever
   - `setDaemon` and `start` the thread

8. It may be helpful to create a command queue to store commands using the `Queue` class.

   - The TCPHandler should place commands in the queue.
   - The RobotController should read commands from the queue.

9. In `rtbot.py` create a `RobotController` class

   ```python
class Robot_Controller(threading.Thread):
    def __init__(self, robot):
        threading.Thread.__init__(self)
        self.rtbot = robot
    def run(self):
        run(self) should check the command queue and execute commands
```

10. In your lab file instantiate a `RobotController` and `start()` it.

11. You can use the `threading.join()` command with the `SocketServer` to keep the main thread blocking until the server closes.

### Android

The following walkthrough is meant for Eclipse, if you do not use eclipse you may need to change a few of the steps.

1. Create a new Android project.

2. Open and configure `AndroidManifest.xml`

3. `res/values/strings.xml` is an excellent place to define constants such as string values.
4. Open res/layout/main.xml and configure it to have a EditText to input the IP address of the robot and Button to connect.

5. Open your main activity in /src and configure it. You should have the definition of a public class already in it:

   - Use setContentView to attach the correct layout R.layout.yourlayout
   - You will need to define a private EditText variable to hold the IP Address and assign (EditText) findViewById(R.id.edittextfieldid)
   - Define a button using Button variablename for the connect button and assign it to (Button) findViewById(R.id.yourbuttonid)
   - create an onclick listener by: yourbuttonvar.setOnClickListener(new Button.OnClickListener(){ });

6. Create two new Activities (class). One should function as the TCP handler, and ther other should send commands to the robot.

7. Go back to your main activity and in your onclicklistener create a Bundle to pass the IP address. A new Intent to call the activity to send commands to the robot. Finally run startActivity to start it.

8. Create a new layout for the robot command activity, it should have buttons for: forward, reverse, left, right, and stop

9. In the robot command activity create a private View for each button.

10. Create a variable of the type of the TCPHandler and set it to null

11. In the onCreate retrieve the Bundle information and use it to assign a new TCPHandler to the variable you created.

12. Check if the TCPHandler to make sure the socket is okay (you will define this later in the TCPHandler)

   - If the socket is bad then you should .finish() the robot commander
   - otherwise continue on

13. Create a function to send commands. It should try and catch exception (catch(Exception e)) on the TCPHandler message send function.
14. Define each button and create onClickListeners for each that use the send command function.

15. Make sure to close the TCPHandler socket when you stop or kill the thread.

16. Go back to the TCPHandler activity and define a

   - final static int Port = 50000
   - a Socket
   - a PrintWriter and set it to null
   - a boolean socketOK and set it to true

17. Use Try and catch to create a Java Socket and Java PrintWriter and assign them to your variables. If it fails set socketOK to false. You can refer to them in the Java documents.

18. You should create class functions to close the socket, return the value of socketOK, and send a message.

Instructor Bot

The robot with the Instructor sticker is designed to allow you to test your Android TCPHandler. To do so power up the robot, you should hear two beeps (one on power up, and one on the soft reset). After this connect to it via your android. You can log in and check that the script is running using guest and rtbl as username and password, then run ps -aux | grep python you should see that it was initialized in init.d. If you need to reboot you may use the soft reset button on the top right of the board.

6.4 Lab 3

6.4.1 Overview

In this lab you will make a class presentation, and an implementation of a set of safety requirements using a robot controller.
Requirements

1. Safety: A robot must not violate it’s safety parameters.

2. Reliability: A robot must protect its own existence and function as long as such protection does not conflict with the First Law.

3. Performance: A robot must obey the orders given to it, except where such orders conflict with the First or Second Laws.

6.4.2 Procedure

Class Presentation

You will make a group class presentation on October 6, see the Initial Design Guide for details on the presentation.

Android and Robot

Your existing Android project from Lab 2 should be able to be reused for this lab. You should write python code to create a Robot Controller that fulfills the requirements. Some things to note:

- It is not safe to run in reverse except on a bump or cliff where you MUST backup until the sensors reads false.

- On wheel drop the robot should stop and wait for manual intervention.

- The ultrasound sensor should allow the robot to avoid bumps, except from small, or short objects.

6.5 Lab 4

6.5.1 Overview

In this lab you will create a demonstration for real time communication outputing a sine wave.
Requirements

1. Create a server in the Processing language that will draw a sine wave given inputs through a socket. The inputs should be a magnitude, and an RGB value.

2. Create at least 3 clients that calculate the appropriate output values for the sine wave, each controller should send a unique RGB value.

6.5.2 Procedure

Overview

You should download processing from http://processing.org/. After installing you can find an example of drawing a sine wave by going to File – Examples – Math – SineWave. Processing has existing Server() and Client() classes in its Api as well as other networking functions that you can find at http://processing.org/reference/libraries/net/Client.html. It also has timing functions such as millis() http://processing.org/reference/millis_.html. Processing is Java based so many Java functions work naturally within the framework. You will want to use the java timers to extend TimerTask a good example can be located here http://www.abstractmachine.net/blog/timertask/

Processing Server

1. The server MUST be written in Processing.

2. In processing you must have a setup and draw functions. When the code executes the setup function initializes the state, and draw is called and runs in an infinite loop.

3. You can change how the sine wave is drawn by changing the values used to calculate it namely xspacing, w, theta, amplitude, period, dx, and yvalues

4. It is suggested you familiarize yourself with the sample sine code mentioned above, you should be able to merely modify it slightly to have it draw from I/O instead of doing the calculation itself.
5. You will need to have the server send synchronization messages to the clients occasionally to minimize drift and jitter.

6. You will need to find how often synchronization pulses need to be sent. They may need to be sent every 3, 10, 100, etc.

7. There should never be more than two items in the queue (one being read, and one being written). If there is your server is not running fast enough.

Clients

8. The clients may be written in your language of choice, but make sure to comment them well for grading.

9. Each client should have its own color to draw the sine dots with

10. They should receive and process synchronization messages from the server

11. Compute the proper value and send to the server along with the clients appropriate color

6.6 Lab 5

6.6.1 Overview

In this lab you will create a setup to test wifi, and movement power requirements.

Requirements

Phone

1. Create a program to launch connect to the robot and begin experiment

Robot
1. Maintain safety as defined in previous labs

2. Send a series of 10,000 UDP packets and measure the resulting power usage

3. Travel 1m and measure the resulting power usage

4. Repeat three times while making an equilateral triangle

5. return values to the terminal

6.2 Procedure

To complete this lab you will need to create an Android package and a python robot server, that can be based off your previous code.

Robot

1. For moving in the equilateral triangle you can use your previous code developed in previous labs.

2. To send the UDP packets you will need to send the UDP packets you can do this by using the following. Where msg is a string, ip_client is the IP of the phone, and PACKETS_PORT is the appropriate port.

   ```python
   send_udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
   send_udp.sendto(msg,(ip_client, PACKETS_PORT))
   ```

3. To get the amount of energy used you can use the

   ```python
   robot.sensors.GetBatteryCharge()
   ```

Phone

You should be able to create an application that connects to the robot and sends a "start" command similar to code used in previous labs.
A.1 Setup

A.1.1 The Robot

The robot is made up of an iRobot create, and a Beagleboard -xM with wifi dongle.

The iRobot Create:

The Beagleboard -xM (revision C):
http://beagleboard.org/static/BBxMSRM_latest.pdf

A.1.2 The Software

The Beagleboard is loaded with Ubuntu 12.04, and runs Python 2.7.

You can interface with the Create using the modified pyrobot.py library:
https://wiki.engr.illinois.edu/display/cs424/Resources

This library interfaces through the Create’s Serial Control Interface called
the Open Interface:
http://www.irobot.com/filelibrary/pdfs/hrd/create/create%20open%20interface_v2.pdf

A.1.3 Networking

The robots are currently designed to logon to Illinoisnet on boot, and send
their current IP to
http://cs424.web.cs.illinois.edu/
A.1.4 Sensors

The Create has many internal sensors that are supported in the pyrobot.py library. In addition we have hooked a MaxSonar MB1200 up the User Analog Input


A.2 Best Practices

A.2.1 System Overview

If we look at the top of the robot to the left of Figure: A.1 there are three points of interest from top to bottom we have:

- User Button: configures the boot order DO NOT PRESS!
- Reset Button: this allows a soft reset of the machine
- Cable Connector: a DB-25 connector that connects the The DB-25 connector connects to three points the first two of which can be seen in on the right of Figure: A.1 from top to bottom:
  1. Power Adapter for Beagleboard to Create power
  2. TTY Serial port on Beagleboard to Create Serial Control Interface
  3. Sonar Sensor to Create User Analog Pin
A.2.2 Powering the Board

To power the board the Cable Connector must be seated properly. Then you may connect the adapter tail into the appropriate connector on the Beagleboard.

A.2.3 Turning off the Board

To power off the board simply remove the connector from back of the Beagleboard. Careful not to yank on the connector as you may cause the tail to become separated from the power adapter board. To safely remove:

1. Pinch your thumb and forefinger on the tail as close to the enclosure box as possible as in Figure: A.2.

2. Lever your fingers towards the enclosure to unseat the tail from the connector

3. Lightly pull out and up as in Figure: A.2

A.2.4 Reseating the Cable Connector

For various reasons you may need to reseat the Cable Connector.
ALWAYS UNPLUG THE BEAGLEBOARD BEFORE RESEATING THE CABLE CONNECTOR

If you do not do so there is a chance that you disconnect the ground pin before the power pin and short the Beagleboard. The easiest way to loosen it is:

1. Put your right index finger on the right side of the Cable Connector as in Figure: A.2.
2. Use it as a lever to loosen it.
3. Rock the connector back and forth and gently lift up.
4. Re-align and push firmly in onto the connector.
5. Pull it about an inch up. Going further could cause you to disconnect the Serial Connector, or Power Adapter.
6. Align and push firmly down

A.2.5 Charging the Robot

When you are done please plug the robot back in to charge. You should make sure the Beagleboard is unplugged, and that the robot is actually charging. When the robot is charging you should see the LED next to the power button as a flickering red as in Figure: A.2.

A.3 Troubleshooting

A.3.1 Beagleboard Power

The BeagleBoard does not receive power when plugged in. Check that the Create has power. Push the power button and cycle the robot till the Green LED next to it is on. If the robot plays a funeral dirge then it is running out of power. Please plug it in to charge and select another robot. If the robot has power then the Cable Connector may be loose try reseating it. If this fails to work after several tries it is suggested that you try using a different robot.
A.3.2 No IP Updates

The robot can take several minutes to boot, and get onto the network, but if it has been some time and it has not sent a message to update it’s IP. The robot may have failed to send they message after receiving the IP, or it may have failed to get onto the network. It is suggested that you hard reboot the robot. After several reboots it is suggested that you try using a different robot.

A.3.3 Serial Not Available

Python Script starts to run, but hangs on sending opcodes or throws a serial not available error. The Cable Connector is most likely loose(it should beep when sent the soft reset), attempt to reseat it. After several attempts it is suggested that you try using a different robot.
This appendix contains example code discussed elsewhere in the document.

B.1 File: /etc/network/interfaces:

```bash
auto lo
iface lo inet loopback

auto wlan0
iface wlan0 inet dhcp
pre-up echo > /var/tmp/current_ip
pre-up hwclock -s
pre-up wpa_supplicant -iwlan0
-c/etc/wpa_supplicant/wpa_supplicant.conf -B
#pre-up sleep 5
up dhclient wlan0
post-up /etc/network/email.sh &
#make sure to sudo chmod 700 email.sh unless you want
#everyone to see your email login credentials
post-up hwclock -w --utc
```

B.2 File: /etc/wpa_supplicant/wpa_supplicant.conf

You may need to create this file, afterwards run chmod(700) this is important to keep the credentials secret from students. You will need to input the correct identity and password as per CITES/EWS direction.

```bash
ctrl_interface=/var/run/wpa_supplicant
```
network={
    ssid="IllinoisNet"
    eap=TTLS
    key_mgmt=WPA-EAP
    pairwise=CCMP
    group=CCMP
    identity="CITES_ID"
    password="CITES_PWD"
    phase2="auth=MSCHAPV2"
}

B.3 File: /etc/network/email.sh

You may need to create this file, afterwards run chmod(700) this is important to keep the credentials secret from students. You will need to input the correct identity and password as per CITES/EWS direction.

#!/bin/sh
ROOMBA=\$(uname -n)
IPADDRESS=\$(/sbin/ifconfig wlan0 |
sed -n 's/.*inet addr:([\^ ]\*\(\)\.*\(\)/p')
FILE=/var/tmp/current_ip
LOG=/var/tmp/email.log

fn_updatedb()
{
    date >> ${LOG}
    python /etc/network/login.py ${ROOMBA} ${IPADDRESS} 2>&1 |
    tee -a ${LOG}
    echo ${IPADDRESS} > ${FILE}
}

fn_checkip()
{
    IPADDRESS=\$(/sbin/ifconfig wlan0 |
sed -n 's/.*inet addr:([\^ ]\*\(\)\.*\(\)/p')

35
OLDIP=$(cat ${FILE})
if [ "${IPADDRESS}" == "${OLDIP}" ]
    echo "No change in IP"
fi
}

fn_updatedb Booting up

i=1
while [ "$i" -ne 0 ]
do
    sleep 150
    fn_checkip
done

B.4 File: /etc/network/login.py

You may need to create this file, afterwards run chmod(700) this is important to keep the credentials secret from students. You will need to input the correct username and password which can be found in the course SVN.

from mechanize import Browser
from sys import argv

login_url = "http://cs424.web.cs.illinois.edu/login.php"
update_url = "http://cs424.web.cs.illinois.edu/update_ip.php"

def update_ip(roomba, ip):
    browser = Browser()

    """ login through a form """
    browser.open(login_url)
    browser.select_form(name="login")

    browser["username"] = "UPDATE_USER"
    browser["password"] = "UPDATE_PASSWORD"
browser.method = "POST"
response = browser.submit()
# print response.read()

""" update the database through GET parameters """
# response = browser.open(update_url)
response = browser.open(update_url + "?roomba=" + roomba + "&ip=" + ip)
print response.read()

if __name__ == "__main__":
    if len(argv) < 3:
        print "\tUSAGE: ", argv[0], "<Roomba>, "<IP>"
        exit(1)
    else:
        update_ip(argv[1], argv[2])

B.5 File: Rtbot.py

Rtbot extends the Create class object so that students can write their own
class methods without modifying Pyrobot.py which could result in students
creating bugs that are hard to track.

from pyrobot import *
import sys
import logging
import time
#=============================================================
# put defines here e.x.
# define_name = define value
#=============================================================
# define the Rtbot class to init and start itself
class Rtbot(Create):
    def __init__(self, tty='/dev/ttyUSB0'):
        super(Create, self).__init__(tty)
        self.sci.AddOpcodes(CREATE_OPCODES)
self.sensors = CreateSensors(self)
self.safe = False  # Use full mode for control.

def start(self):
    logging.debug('Starting up the Rtbot."
    self.SoftReset()
    self.Control()

#=================================================================================

#place further functions in the Rtbot class e.x.
# def somefunction(some_argvs):
#    some code
REFERENCES


