MALBOT!

Alpha 1.0

Introduction

MalBot! was designed for the University of California San Diego as a ECE 118 (Computer Interfacing) Project by Alexander Kissinger (Hardcodingisbetter.tumblr.com). It was inspired by such products such as the Wifi Pineapple (developed by Hak5: http://hakshop.myshopify.com/products/wifi-pineapple), and was conceived under the idea that pen-testing should be mobile, smart, automated, and on a open-source (and itself hackable) electronic platform.

Table of Contents

Goal of the Project/Notes from the Author.....05

The MalBot! Anatomy.....06

Block Diagram.....07

The Raspberry Pi Computer.....08

Battery Charging.....09

Turning MalBot! On.....09

Interfacing.....10

General Usage.....10

MalBot! Buit-in Attack.....11

Maintanence/Warnings.....12

Parts List.....13

Schematics.....14

Source Code.....15

Goal of the Project

The goal of MalBot! was to make an automated yet easily customizable pen-testing robot that can be communicated over the internet, out in the field. Because this is an open-source/baseline product there is very limited built-in functionality.

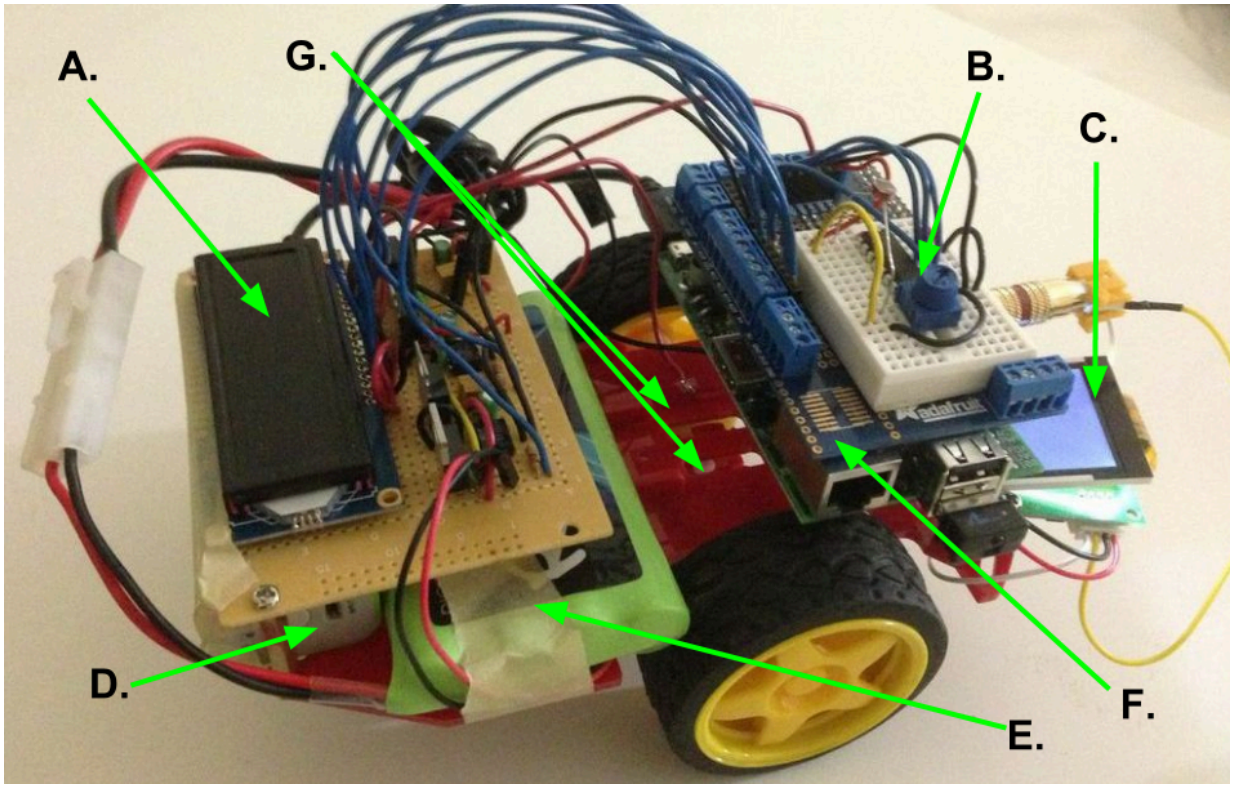
MalBot! comes pre-installed with a Raspberry Pi version 2 board, with a Adafruit Protoshield installed. MalBot! Comes with eight more actuators/peripherals: a TFT screen to debug MalBot! on the fly, an LCD character display that display current/critical information about the system, DC motors, a digital light sensor, a primary USB comm device 3G/WIFI (for comm with an end-user), a secondary Atheros chipset based USB WIFI NIC (used for pen-testing), and two power sources (one for computer and the other for the motors and TFT screen).

An open-source pen-testing Debian distribution, PWNPI (http://pwnpi.sourceforge.net/), comes pre-installed on the Raspberry Pi's 16GB SD card, but of course this can be changed if desired (WARNING: If you do install your own distrution other than the built-in one, you will have to find a way to interface with the motors). There are two main components to the software. There is the built-in pen-testing tools that is provided by PWNPI and there are the actuator interfacing scripts developed specifically for MalBot!. More detail on these scripts will be discussed in the Interfacing section.

If you choose to customize MalBot!, and you're encouraged to do so, the product has been simplified to abstract customization into two main layers. First, the Hardware layer can be modified to add/remove actuators from the system (proximity sensors, microphones, etc.) and can be easily installed into the Raspberry Pi's extended Protoshield and interfaced with the GPIO library. The Software layer customization was designed to to extend off of the scripts developed for MalBot! and is found in /root/Scripts. These are written in Python, use the GPIO library, and are executed via SSH. As stated before there is limited functionality, but this project was meant to get your hands dirty, so get hacking!

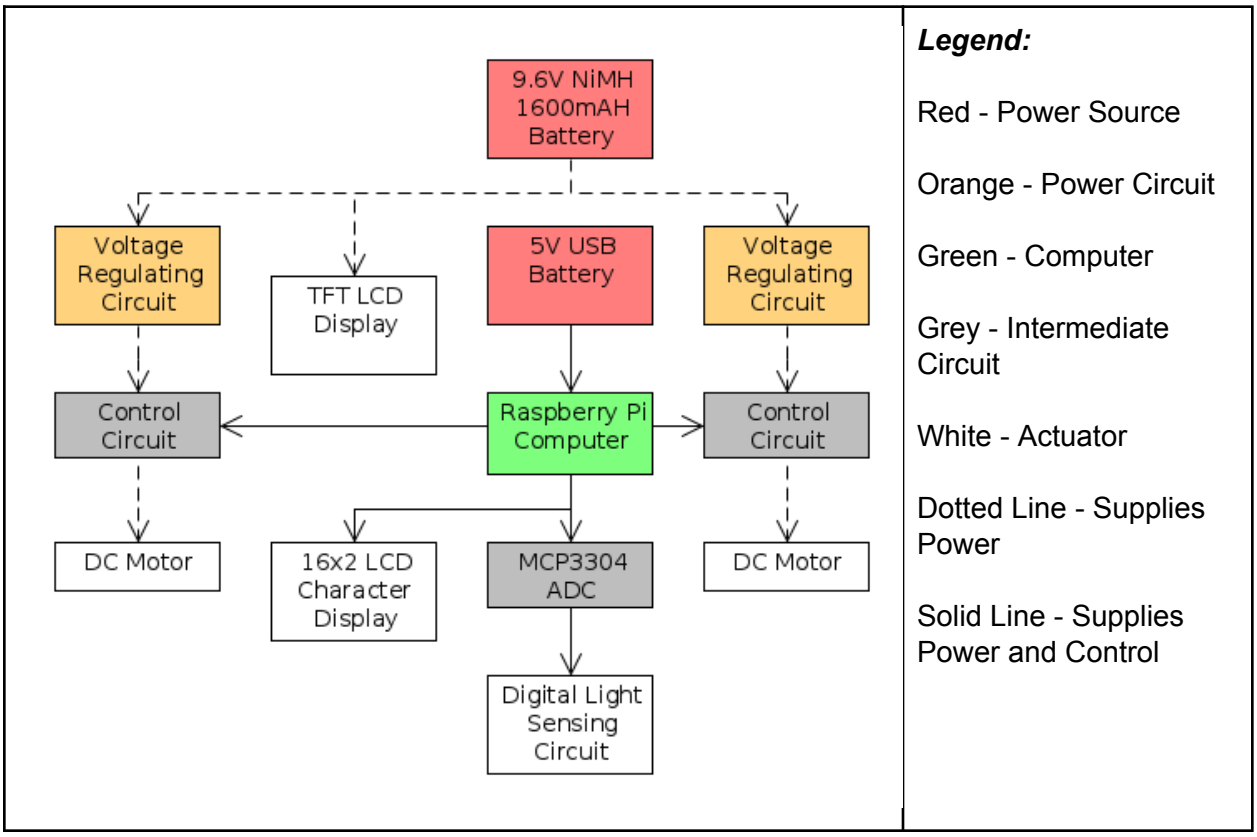
MalBot! Anatomy

Overview



a. 16x2 Character LCD Display
b. Digital Light Sensing Circuit
c. 1.5” TFT LCD Display
d. 5V USB Battery
e. 9.6V 1600mAh
f. Raspberry Pi version 2.0 Computer
g. DC Motors

Block Diagram



This is a high-level view of the MalBot! system. The 9.6V Battery supplies power to the DC Motors by running through a Voltage Regulating Circuit that outputs 5V~1A on an average load. The power from these circuits then run into a Control Circuit that is ready to provide power to the DC Motors when the Raspberry Pi sends a HIGH signal on its control line. The 9.6V Battery also provides power to the TFT LCD Display directly, which has its own internal voltage regulating circuit.

The Raspberry Pi itself is powered via USB by a 5V~1A MAX USB Battery. Both the 16x2 LCD Character Display and Digital Light Sensing Circuit, as well as its intermediate circuits, are powered and controlled by the Raspberry Pi's internal power via GPIO/SPI/I2C/etc.

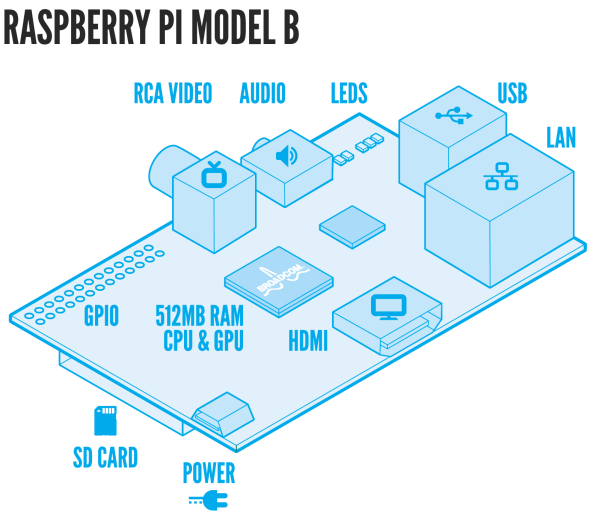
The DC Motors are controlled via GPIO (HIGH/LOW values) and are isolated away from the Raspberry Pi by means of a NTE3040 Optoisolator. A control signal will be sent from the Raspberry Pi to the optoisolator on the Control Circuit that will instruct the DC Motors to move or not. Be advised that there is not an H-Bridge unit on the Control Circuit, ergo the motors can/will only move in one direction (this reflects on the Python code).

The Digital Light Sensing Circuit is interpreted by the MCP3004 ADC, which is powered by the

Raspberry Pi's internal 5V power (via GPIO). The Raspberry Pi interfaces with the MCP3004 via the SPI data protocol. The Digital Light Sensing Circuit is composed of an LDR and a Variable Resistor that can be used to customly set the desired sensitivity.
16x2 Character LCD Display is also powered by the Raspberry Pi's 5V GPIO and directly interfaces with the Raspberry Pi via GPIO

TFT LCD Display has its own on board "buck converter" that allows an input between 6V-12V and is directly powered by the 9.6V battery.

The Raspberry Pi Computer



The Raspberry Pi is a full ARM powered Linux computer. It comes preinstalled with PWNPI, an open-source Debian-based pen-testing distribution. The Raspberry Pi works on two main levels: the Hardware and the Software layer. The software layer can be accessed directly (using keyboard, any of the video ports, or built-in TFT screen) or via the internet. The hardware layer can be accessed through the GPIO Python library that is currently installed (SPI and I2C Python libraries are also available).

More information on the Raspberry Pi's GPIO functionality can be found here:
http://elinux.org/RPi_Low-level_peripherals

The main site, along with the Raspberry Pi community, can be found here:
http://www.raspberrypi.org/

Battery Charging

MalBot! comes with two batteries:

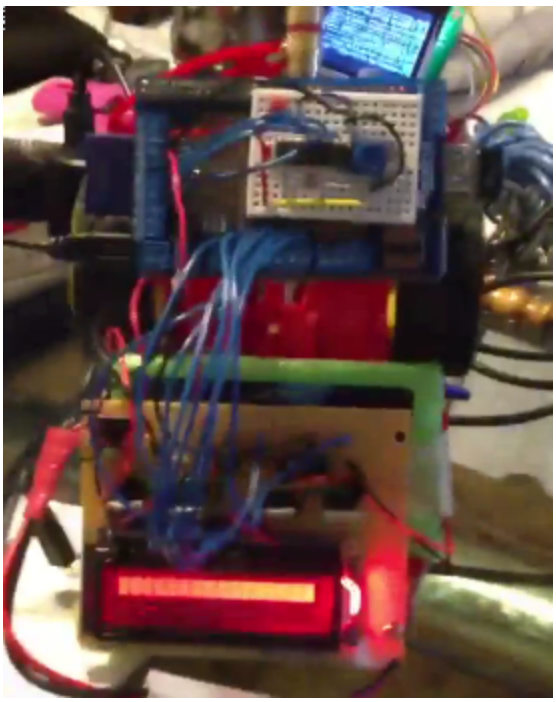
1x 9.6V 1600mAh RC Car Battery
Powers: the DC Motors, TFT screen

1x 5.0V ~ 1A USB Battery
Powers: the Raspberry Pi, Char LCD display, Digital Light Sensor

Each Battery is replaceable with anything in the same voltage range, but have been specifically chosen for it's size dimensions and weight. The 9.6V battery pack comes with it's own charger and the USB battery stick can be plugged into any USB power supply on the input side. Charge for at least three hours for the most optimal usage.

Turning MalBot! On

To turn on MalBot! on plug in the 9.6V battery pack's male receiver into the corresponding female receiver on the motor board, then plug a micro USB cable into the USB Battery Stick on the output side, and plug in the micro receiver into the Raspberry Pi. If both batteries are charged you will immediately see the LEDs on the Raspberry Pi start flashing, the TFT will display show Linux booting, and the Char display light up.



If everything on the software-side is working properly, then Char display will output it's default information consisting of the current environment's luminosity and it's network information and Keanu Reeves will display on the TFT screen.

Interfacing

When MalBot! boots up it's eth0 interface will have a short 10 second period where it will assign it's an ip of 10.10.0.1 for debugging perposes. If you have a computer interfacing with MalBot!'s ethernet port it will hold that IP until restarted. This can be double-check for the Char display will output eth0's ip information.

After that 10 second period, MalBot! will try to attach itself onto any insecure network and get an IP, which can be verified by looking at the Char display. If instead, you installed a 3G card as the primary network interface, it will be dished an IP according to that. If you then register an IPv6 for MalBot!, no matter what internal IP (varying) MalBot! may have, you will always be able to communicate via your IPv6 address. The primary means of communication is SSH, but of course a webserver can be installed on MalBot! and communicated via (or something even more clever). The login credentials are as follows:

Username: root
Password: toor

If all other means of interfacing do not suffice, you can always directly interface with MalBot!'s USB ports and use it's built in display.

General Usage

The general usage flow for MalBot! works in the following:

1. Turn MalBot! on.
    a. When both batteries are plugged in you should see both LCD displays light up and PWNPI boot up on the TFT Screen.
2. Tune the variable resistor on the digital light sensor to the desired sensitivity.
    a. The digital light sensor is based on an 10-bit range which that gives ambient light in values between 0 to 1023. This means that the variable resistor (potentiometer) is quite sensitive.
    b. The best way to tune the sensor is to put the cover (disguise) over MalBot!, get the reading, and tune it to result a luminous reading of around 800, in its cover, in a generally well lit room. This is because the demo code bases sets the movement cut-off reading at 800.
    c. You can find the luminos reading on the character display after bootup.

3. Log into MalBot!.
    a. After PWNPI boots, Keanu Reeves should display on the TFT screen. This indicates that MalBot! is ready to receive communication from an end-user.
    b. MalBot! will then try to log into an insecure network and display its network information on the character display.
    c. Use one of the above methods in the “Interfacing” section to log in.
4. Hack away!
    a. You can start testing out MalBot!’s pen-testing abilities by either using PWNPI’s built-in computer security testing software suite, using the built-in programmed demo scripts, or by making your own scripts based off of the demo scripts.

MalBot! Built-in Attack

Performing an Automated Wifi Discovery and Sniffing Attack

Automated wifi discovery and sniffing are the two main scripts that are provided to test MalBot!. They are located in the /root/Scripts/ folder and are named “findsignals” and “triangulation”. The other files are dependencies for these two scripts, and will show themselves useful when writing custom scripts. A basic auto-sniffing attack is done by performing the following, and can be seen by the demo video: http://hardcodingisbetter.tumblr.com/post/52013137447/malbot-demo

1. Turn on MalBot!.
2. Log into to system via SSH.
3. Change into the “Scripts” directory and execute the “findsignals” script.

```
$ cd /root/Scripts/
$ ./findsignals
        Cell 01 - Address: 00:00:00:00:00:20
                Quality=61/70  Signal level=-49 dBm
                Encryption key:on
                ESSID:"HACK ME"
        Cell 02 - Address: E0:00:00:00:00:FF
                Quality=24/70  Signal level=-86 dBm
                Encryption key:on
                ESSID:"DoctorWOW”
```

4. MalBot! will then return you a list of wifi access points within the proximity. Simply choose the ESSID you wish to attack and proceed to the next step.
5. Execute the “triangulation” script and enter in the scanning wifi NIC’s network interface and the ESSID to be attacked.
    a. If the ambient light exceeds 800 (verified by checking the character display), the script will warn that the room is currently bright and unsafe to move in. Typing ‘n’ will exit the script and ‘y’ will continue even though it is not necessarily incognito.
    b. If MalBot! is reading less than 80dBm from the victim access point, in any given

current location, the script will inform the user that it is “HUNTING SIGNAL!”. MalBot! will try to move itself and triangulate the access point.

c. When MalBot! reaches 80dBm, or better, but is not in a darkly concealed area it will try to move itself into a dark area informing the user by printing “GOOD SIGNAL\n LOOKING FOR HIDING SPOT”.
d. Only finally after both the above conditions are met will MalBot! siese from moving and start sniffing wifi packets from the victim access point, informing the user by printing “ATTACKING ACCESSPOINT”.

```
$ ./triangulation
Enter the interface you want to use for scanning: wlan0
Enter the ESSID: HACK ME
!!!THE ROOM IS BRIGHT, ARE YOU SURE?: y
HUNTING SIGNAL!
HUNTING SIGNAL!
GOOD SIGNAL
LOOKING FOR HIDING SPOT
ATTACKING ACCESSPOINT
ATTACKING ACCESSPOINT
ATTACKING ACCESSPOINT
```

**The source code can be found in the “SOURCE CODE” section.*

Maintenance/Warnings

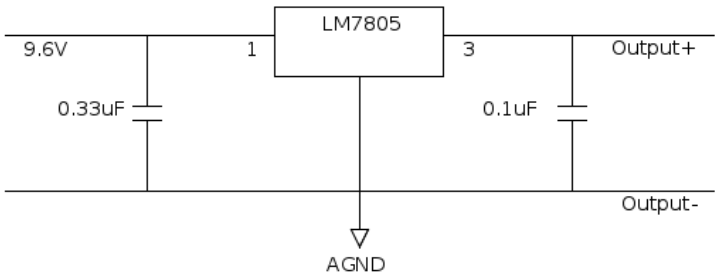
- Always make sure you charge the batteries with the proper charger, any misuse may cause the battery cells to leak or explode.
- Make sure if you replace the built-in batteries that they are within the proper voltage range.
- Any environmental factors that may cause the DC motors to become overly stress may drain the battery at an extreme rate.
- Keep MalBot! away from water
- REMEMBER THAT MALBOT! SHOULD ONLY BE USED FOR EDUCATIONAL PURPOSES. ANY ILLEGAL USE WILL NOT REFLECT ON UCSD OR ON THE DEV.

Parts List

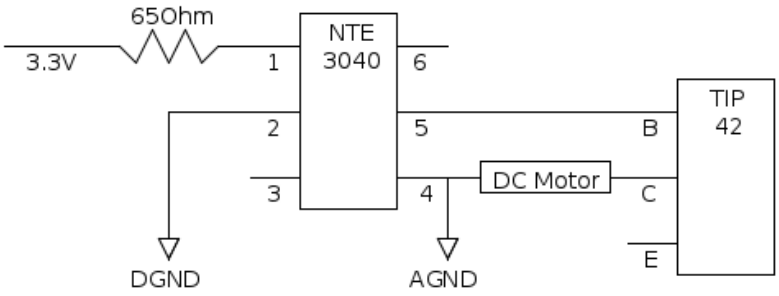
Part	Count	Supplier	Cost
Tenergy 2600mAh Portable Power Bank Item#51034	1	Amazon.com	$19.99
NTSC/PAL TFT Display 1.5”	1	Adafruit.com	$39.95
RGB Backlight Negative LCD 16x2 + extras	1	Adafruit.com	$13.95
Sparkfun Magician Chassis	1	Amazon.com	$26.95
Adafruit Prototyping Pi Plate	1	Adafruit.com	$15.95
Raspberry Pi Model B	1	Element14.com	$35.00
NTE3040 Optoisolator	2	Frys.com	$2.00
TIP42 PNP Transistor	2	Radioshack.com	$2.00
LM7805 Linear Voltage Regulator	2	Radioshack.com	$3.00
0.33microF Capacitor	2	Radioshack.com	$0.20
0.1microF Capacitor	2	Radioshack.com	$0.20
65ohm Resistor	2	Radioshack.com	$0.10
848ohm Resistor	1	Radioshack.com	$0.05
9.6V 1600mAh RC Battery, Charger	1	Radioshack.com	$30.00
MCP3004 A/D Converter	1	Adafruit.com	$3.75
Breadboard Trim Potentiometer 10K	1	Adafruit.com	$1.25
Photo cell (CdS photoresistor)	1	Adafruit.com	$1.00
Total			$195.34

Schematics

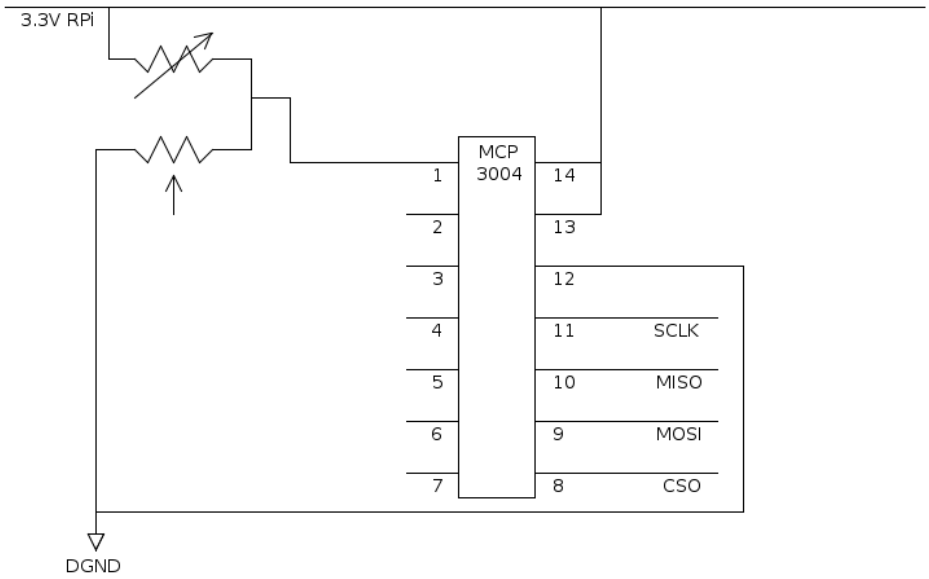
Voltage Regulating Circuit



Motor Control Circuit



Digital Light Sensor



Source Code

"findsignals"

```
#!/bin/bash
iwlist wlan0 scan > .signaltmp
awk '/Cell|ESSID|Quality|Encryption/' .signaltmp > signallist
rm .signaltmp
cat signallist
```

"triangulation"

```
#!/usr/bin/env python
#IO18 -> 12
#IO04 -> 7
import os
execfile("ECE118_PHOTO.py")
execfile("pickessid")
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(7, GPIO.OUT, GPIO.LOW)
GPIO.setup(12, GPIO.OUT, GPIO.LOW)

pickessid_init()
ok_continue = True
sig = get_signal( get_list( user_if ), essid )
lum = readadc(0)

if lum > 400:
        global ok_continue
        ok_continue = raw_input("!!!THE ROOM IS BRIGHT, ARE YOU SURE?: ")

while ok_continue:
        sig = get_signal( get_list( user_if ), essid )
        lum = readadc(0)

        sig_in_bounds = sig < 80
        lum_in_bounds = lum < 400

        if not sig_in_bounds:
                print "HUNTING SIGNAL!"
                displaycmd = open('displaycmd.dat', 'w')
                displaycmd.write('HUNTING SIGNAL!')
                displaycmd.close()
                #DO SOME MOVEMENT
                GPIO.output(7, GPIO.HIGH)
                GPIO.output(12, GPIO.HIGH)
        elif not lum_in_bounds:
                print "GOOD SIGNAL"
                print "LOOKING FOR HIDING SPOT"
                displaycmd = open('displaycmd.dat', 'w')
                displaycmd.write('GOOD SIGNAL\nHIDING')
                displaycmd.close()
                GPIO.output(7, GPIO.HIGH)
                GPIO.output(12, GPIO.HIGH)
                sleep(1)
            #DO SOME MOVEMENT
                GPIO.output(7, GPIO.LOW)
                GPIO.output(12, GPIO.HIGH)
        else:
```

```
                print "ATTACKING ACCESSPOINT"
                displaycmd = open('displaycmd.dat', 'w')
                displaycmd.write('ATTACKING\nACCESSPOINT')
                displaycmd.close()
                GPIO.output(7, GPIO.LOW)
                GPIO.output(12, GPIO.LOW)
            #CALL PACKET SNIFFING SOFTWARE
        sleep(2)
```

“pickessid”

```
root@pwnpi:~/Scripts# cat pickessid
#!/usr/bin/env python
import subprocess
import re
from time import sleep

user_if = ""
essid = ""

def pickessid_init():
        global user_if
        global essid
        user_if = raw_input("Enter the interface you want to use for scanning: ")
        essid = raw_input("Enter the ESSID: ")

def get_list( wlan_if ):
        proc = subprocess.Popen('sudo iwlist '+ wlan_if + ' scan 2>/dev/null', shell=True, stdout=subprocess.PIPE, )
        stdout_str = proc.communicate()[0]
        stdout_list=stdout_str.split('\n')
        stdout_list = stdout_list[::-1]
        return stdout_list

def get_signal( iwlist_list, this_essid ):
        print this_essid
        found = False
        match = ""
        for line in iwlist_list:
                line = line.strip()
                if not found:
                        match = re.search( this_essid, line )
                        if match:
                                found = True
                else:
                        match = re.search( "Signal level=-(\d+)", line )
                        if match:
                                return match.group(1)
                                break
```

“robot_char”

```
#!/bin/sh
nohup /root/Scripts/ECE118_CHAR.py &
```

“ECE118_CHAR.py”

```
#!/usr/bin/python

from Adafruit_CharLCD import Adafruit_CharLCD
from subprocess import *
from time import sleep, strftime
```

```
from datetime import datetime
execfile("/root/Scripts/ECE118_PHOTO.py")

lcd = Adafruit_CharLCD()

cmd0 = "ip addr show eth0 | grep inet | awk '{print $2}' | cut -d/ -f1"
cmd1 = "ip addr show wlan0 | grep inet | awk '{print $2}' | cut -d/ -f1"

lcd.begin(16,1)

def run_cmd(cmd):
    p = Popen(cmd, shell=True, stdout=PIPE)
    output = p.communicate()[0]
    return output

while 1:
        lcd.clear()
        readinfile = open("/root/Scripts/displaycmd.dat", 'r')
        displaycmd = readinfile.read()
        readinfile.close()

        if displaycmd == '' or displaycmd == '\n':
                eth0ipaddr = run_cmd(cmd0)
                wlan0ipaddr = run_cmd(cmd1)
                #lcd.message(datetime.now().strftime('%b %d  %H:%M:%S'))
                lcd.message('eth:%s\n' % ( eth0ipaddr ) )
                lcd.message('wln:%s\n' % ( wlan0ipaddr ) )
                sleep(2)

                lcd.clear()
                lcd.message('luminos:%i\n' % ( readadc(0) ) )
        else:
                lcd.message(displaycmd)
        sleep(2)
```

Dependencies:
https://github.com/adafruit/Adafruit-Raspberry-Pi-Python-Code/tree/master/Adafruit_CharLCD