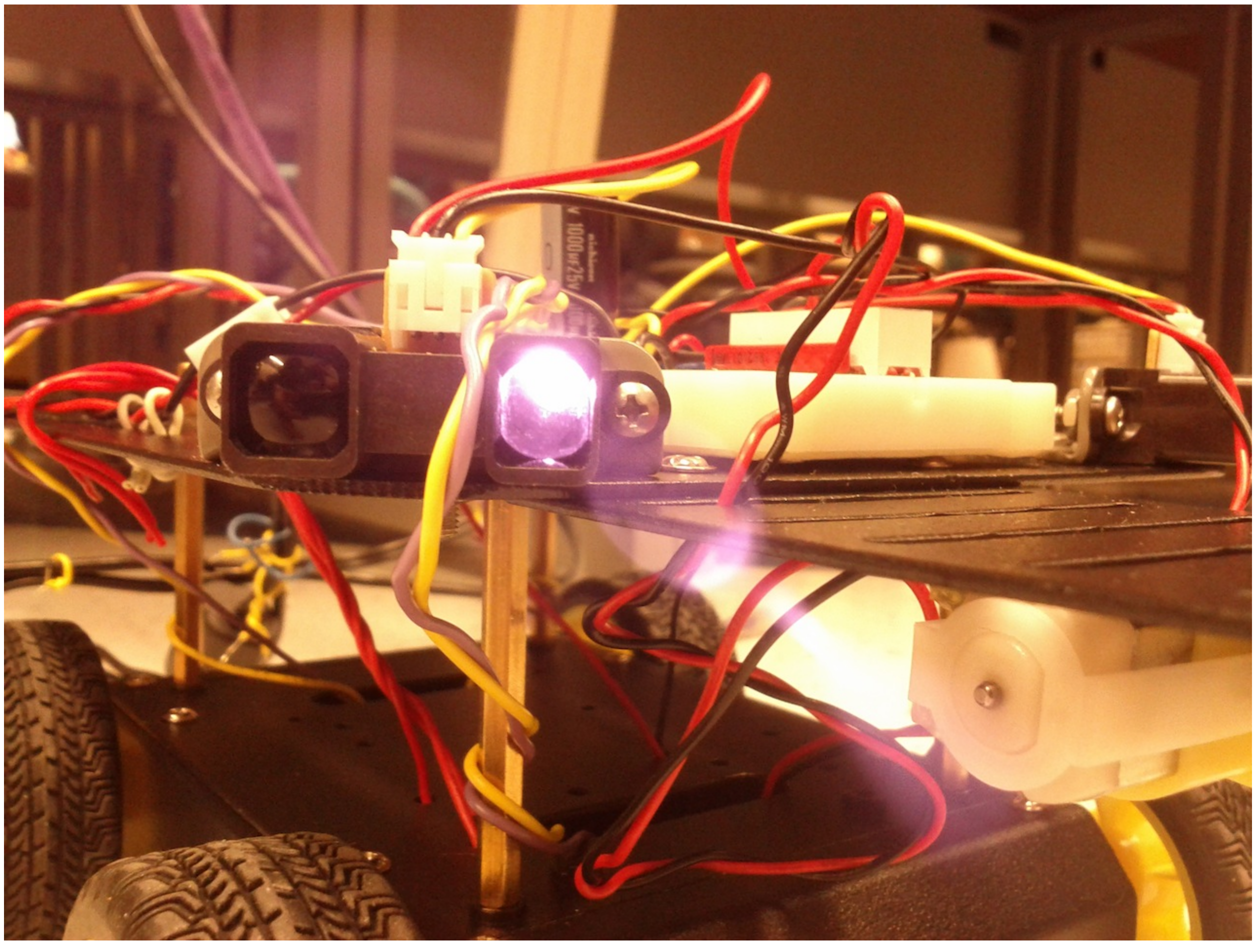
Concordia University
Department of Computer Science
and Software Engineering

Embedded Systems
SOEN 422/4 --- December 2011--- Section MM

Project Final Report

Team information	
Team Number	9
Member Name	**Member SID**
Bronson Zgeb	9402268
Eric Chan	9365079
Dragos Ionut Anton	9567399



Introduction

Project Description

This project was started to build a programmable network connected device which will accept user input as a sequence of instructions or in real time and perform them in order to use move and activate a drawing apparatus. This robot must ensure the safety of the drawing apparatus and secondly of itself. It is a real-life implementation of the Turtle Graphics program.

Purpose

The purpose of this project is to build a robot which is capable of drawing patterns on the ground. These patterns are programmable scripts, built by the user and sent over a network. The device is will perform the tasks in order, while simultaneously assuring the safety of the drawing apparatus, and itself. It does so through IR sensors which determine if there are objects nearby which present a danger to it's hardware. If so, the robot will stop moving.

Function and Performance

The unit runs a web server, which allows it to be controlled through any internet browser. Through your browser, the unit can either be controlled from the user's keyboard, or a script builder, depending on the user's preference. The intention is to allow the user to draw patterns on the ground. The script builder allows the user to build a script and run it, so that he may watch the robot autonomously perform the action. The keyboard controls present a playground for the user to manually override the robot and test out ideas while controlling the robot in real-time.

Hardware

The hardware this project uses is a robot chassis, a microcontroller with prototyping kit, a Beagleboard, a drawing apparatus, two IR sensors, and wheels for autonomous movement.. The Beagleboard is main control device, it sends and receives signals from the microcontroller. The microcontroller runs the motors, and reads information from the IR sensors. The IR sensors read how close the unit is to a collision.

Software

The software this project uses is an in house command program written for the microcontroller, a PHP server-side script, as well as an HTML/Javascript interface for the network facing aspect of the project on the Beagleboard. The interface is written to allow the user to control the robot from the browser. This includes keyboard input, as well as a script builder. The browser interface communicates with the PHP script. This then communicates with with the microcontroller. The interface is accessible because we installed the Apache2 web server on the Beagleboard. In addition, we also installed build-essentials to avoid having to cross-compile our command program. Finally we installed SSHD so that we could gain access to the unit through the ssh protocol, as well as share files with scp. The Beagleboard is running Ubuntu Linux server edition.

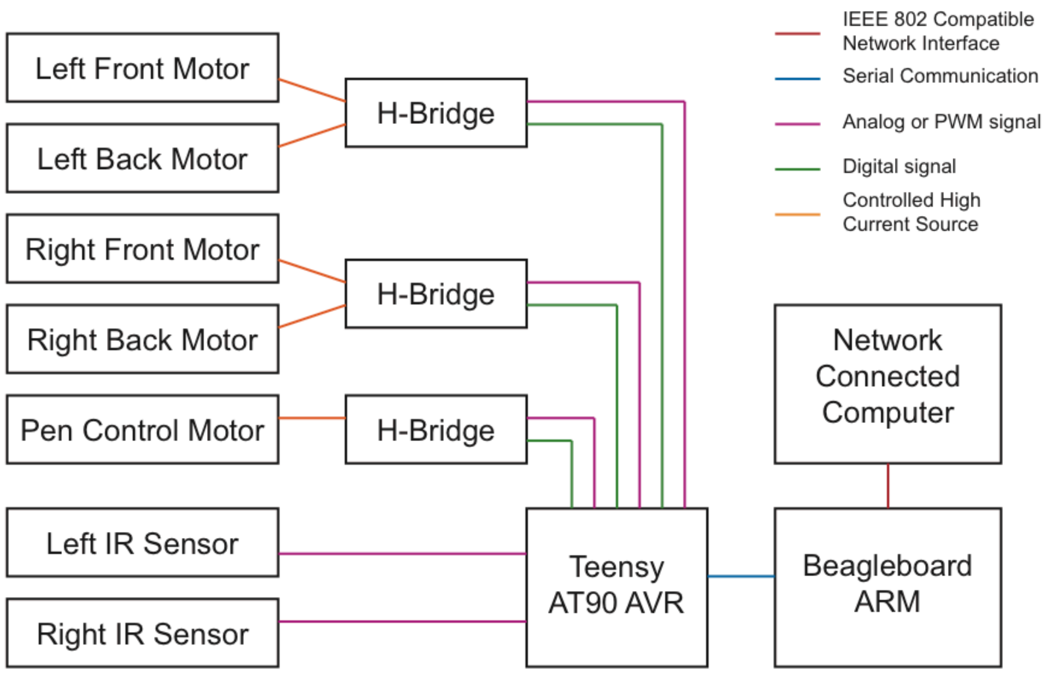
Scope of Completion

The project has met its expectations in terms of basic software. The hardware of the device needs to be tuned and made sturdier. This includes the drawing apparatus, which is quite fragile at the moment. Furthermore, the unit only has IR sensors on the front, so it's unable to determine if there are dangers behind it. The software needs to have its web interface improved for added usability and needs proper concurrency support as there is none at the moment. We intended to have built-in patterns for demonstrative purposes, but this has yet to be implemented. The unit does not currently support real-time. Unfortunately, this means the reaction time when encountering an obstacle is not sub-optimal, and could cause damage to the hardware.

Hardware Design

System Design

The hardware is divided into two main categories. The machine control mechanics and logic as well as the network accessibility and main control logic. The mechanics and logic is considered to be the robotics assembly chassis, motor drivers, microcontroller prototyping board and Beagleboard for main control logic. The Beagleboard receives messages from a network controlled computer, through a web server running off the device. The user can access the control interface using a standard web browser. The board interprets the messages received from the user, which were passed through either an RJ45 or Wi-Fi connection. Signals are passed from the Beagleboard, over the serial port to the Teensy Microcomputer.



Subsystem Design

The network connected computer sends signals using the HTTP protocol, over an RJ45 or Wi-Fi connection.

Figure 18 is the high level block diagram of the BeagleBoard-xM.

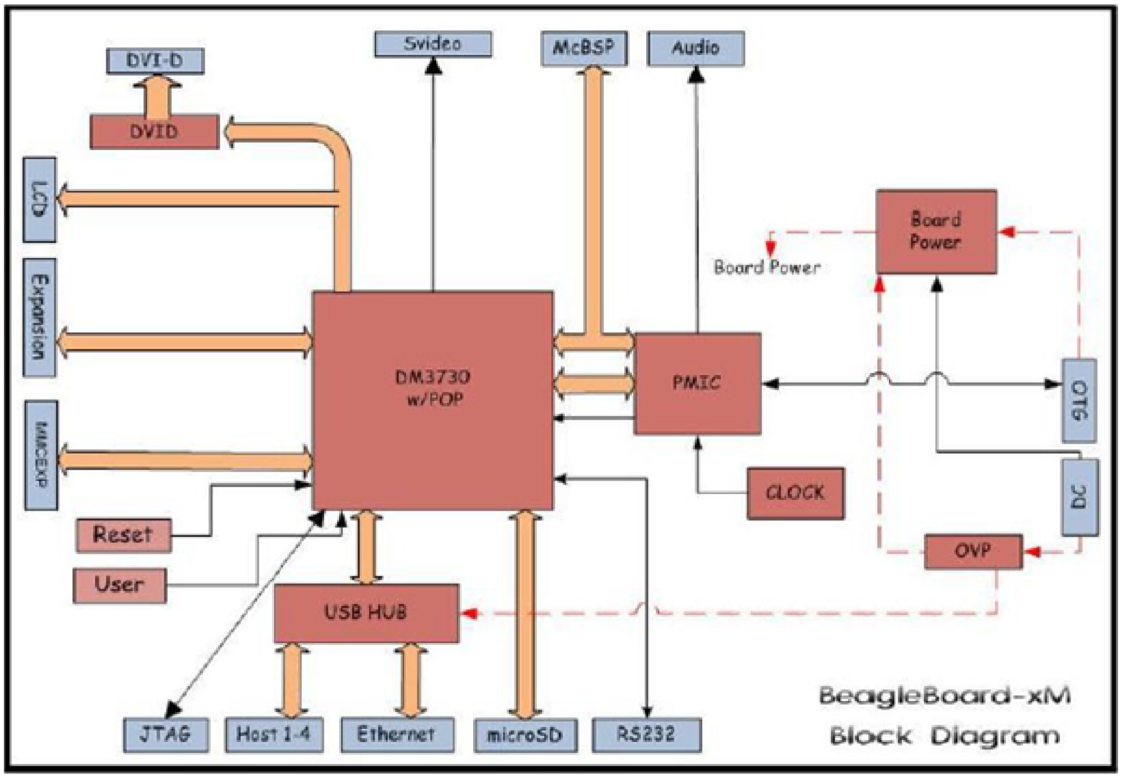


Figure 18. BeagleBoard-xM High Level Block Diagram

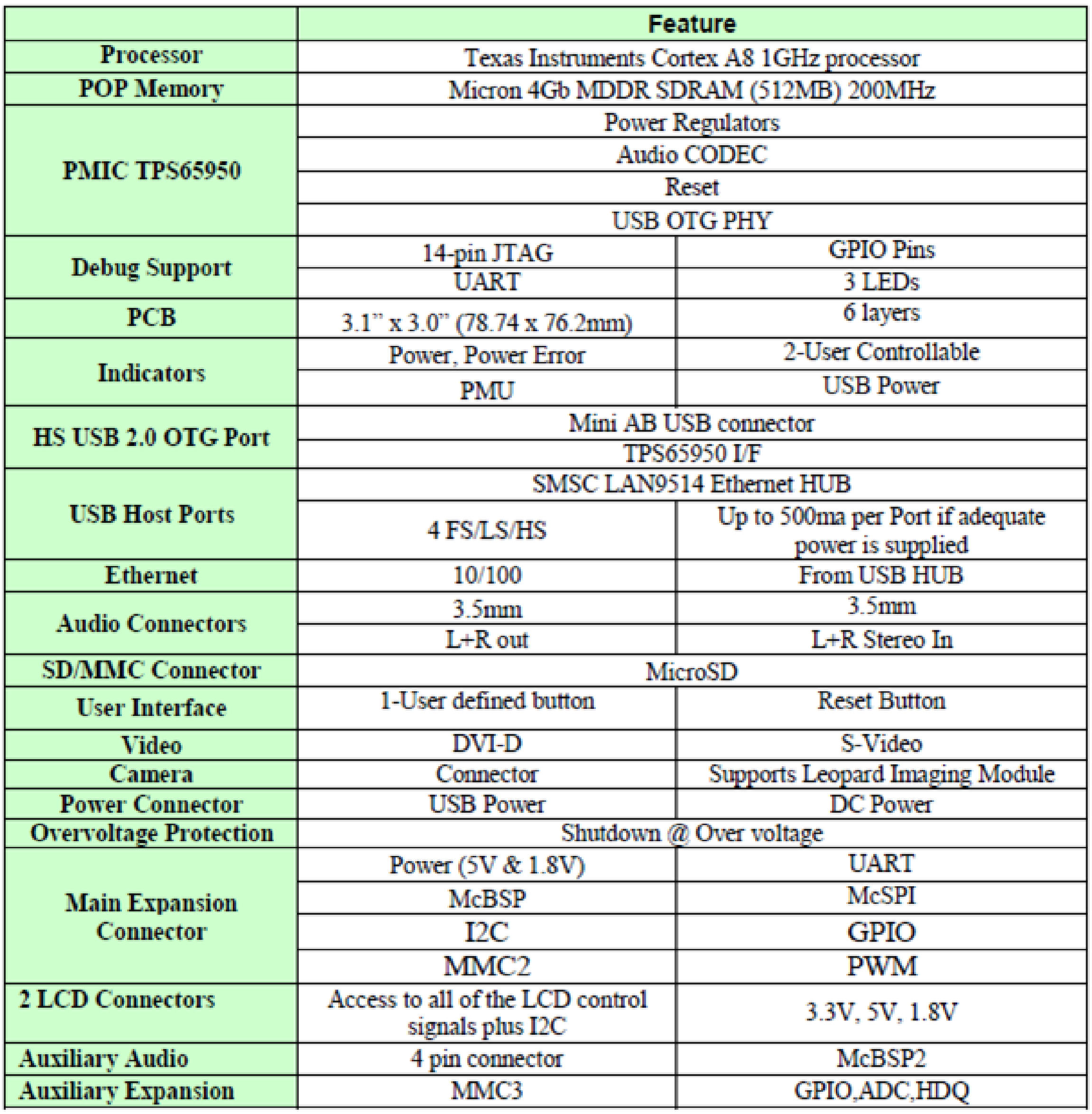
In order to receive the messages, the Beagleboard uses either USB or an Ethernet port, depending on how the user connected to the interface. To this end, we used a USB Wi-Fi adapter to receive the signals, and pass them into the device. As seen in the block diagram, Ethernet messages are also passed into the device through the USB Hub. The Beagleboard contains a Texas Instruments Cortex A8 1GHz ARM processor. The ARM family of processor is excellent for low power consumption, and moderate processing power. This allows us to power the device through a simple USB port if necessary, which is desireable on this sort of embedded device.

Messages are created by the processor and sent from the Beagleboard over the emulated RS232 port. This is a USB port on the side of the Beagleboard, and an RS232 port in the Teensy Microcontroller.

The Teensy Microcontroller communicates with the motors using PWM signals. PWM signals are a way for the microcontroller to fake analog signals using digital signals. In addition to these analog signals, each motor has a digital signal used to control the direction of the motor. The analog pins of the Teensy Microcontroller are able to send PWM signals, while the digital pins are used to send the direction control signals. Each motor is connected to an H Bridge, which connects to the Teensy board. The H Bridge acts as an amplifier from the Teensy board to the motors. This is because the motors require a large amount of power to run, more than the Teensy can offer. To power the motors, we have an extra external 5v power supply.

In addition to the motors, the Teensy Microcontroller has to control the IR sensors. The IR sensors are powered directly from the Teensy. These are connected to analog pins, which allow the Teensy to read analog values from the sensor with the help of the analog-to-digital converter.

Table 2. BeagleBoard-xM Features



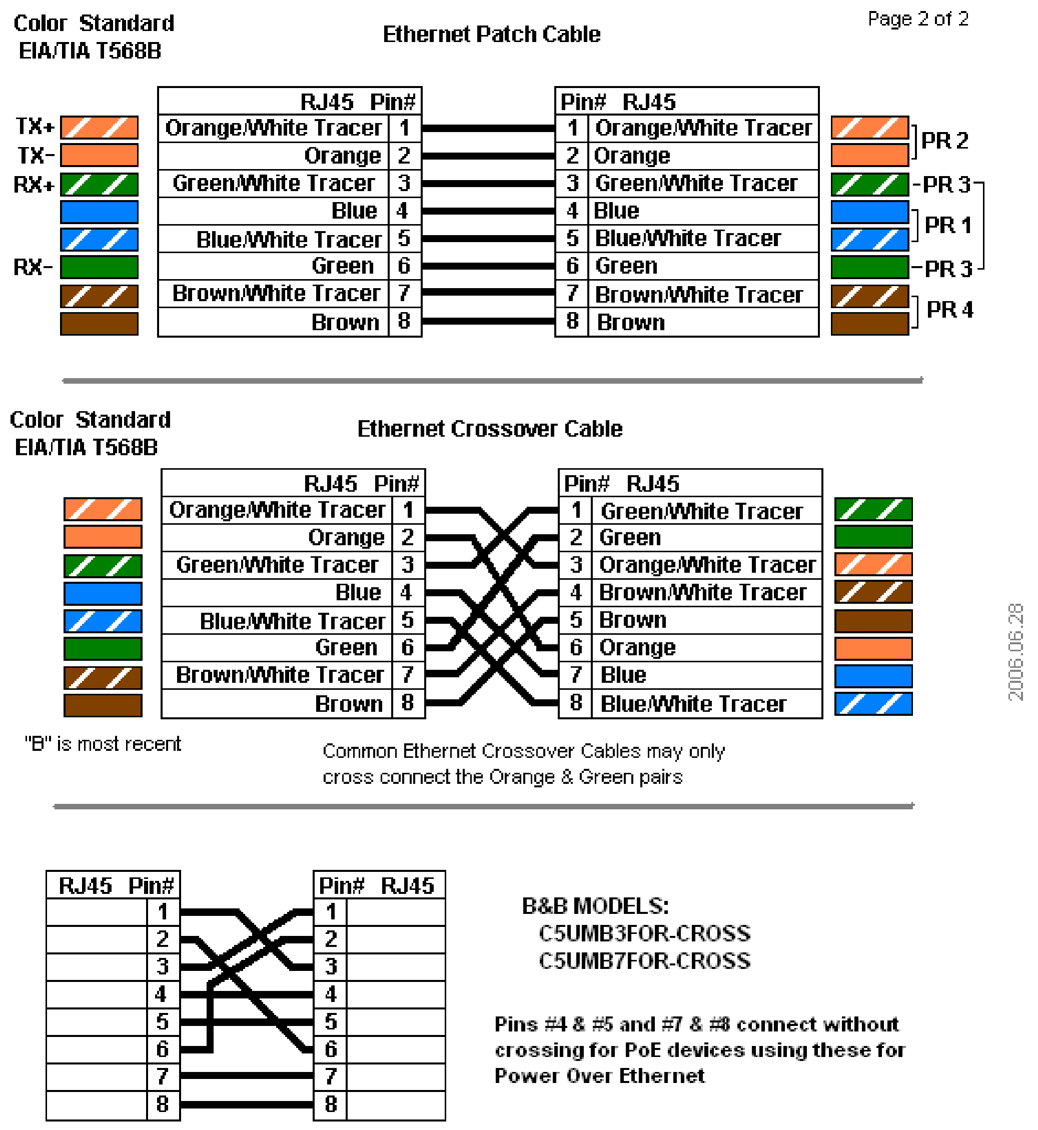
	Feature	
Processor	Texas Instruments Cortex A8 1GHz processor	
POP Memory	Micron 4Gb MDDR SDRAM (512MB) 200MHz	
PMIC TPS65950	Power Regulators	
	Audio CODEC	
	Reset	
	USB OTG PHY	
Debug Support	14-pin JTAG	GPIO Pins
	UART	3 LEDs
PCB	3.1" x 3.0" (78.74 x 76.2mm)	6 layers
Indicators	Power, Power Error	2-User Controllable
	PMU	USB Power
HS USB 2.0 OTG Port	Mini AB USB connector	
	TPS65950 I/F	
USB Host Ports	SMSC LAN9514 Ethernet HUB	
	4 FS/LS/HS	Up to 500ma per Port if adequate power is supplied
Ethernet	10/100	From USB HUB
Audio Connectors	3.5mm	3.5mm
	L+R out	L+R Stereo In
SD/MMC Connector	MicroSD	
User Interface	1-User defined button	Reset Button
Video	DVI-D	S-Video
Camera	Connector	Supports Leopard Imaging Module
Power Connector	USB Power	DC Power
Overvoltage Protection	Shutdown @ Over voltage	
Main Expansion Connector	Power (5V & 1.8V)	UART
	McBSP	McSPI
	I2C	GPIO
	MMC2	PWM
2 LCD Connectors	Access to all of the LCD control signals plus I2C	3.3V, 5V, 1.8V
Auxiliary Audio	4 pin connector	McBSP2
Auxiliary Expansion	MMC3	GPIO,ADC,HDQ

http://beagleboard.org/static/BBxMSRM_latest.pdf - Table 2

System Intercommunication

To communicate between the Beagleboard computer and the Teensy Microcontroller a USB RS-232 emulation was used.

We communicated from our workstations to the Beagleboard computer using an RJ45 connection, or, a USB2-to-RS232 converter.



http://www.bb-elec.com/tech_articles/EthernetCables_illustration.asp

Software Design

System Design
Our main control logic is running on the Beagleboard, which is running the Ubuntu 11.10 server edition operating system. Installed on the Beagleboard we also have development essentials, for compiling our C code. Our control interface is dependent on the Apache2 web server which serves our interface as a website, which can be accessed by any regular browser. Inside Apache2 we have the PHP5 module installed, which allows our server to run PHP code. This is necessary because the main control script is written in PHP.

The Teensy Microcontroller depends on the Arduino libraries, which allow to write high level code for hardware control.

Subsystem Design
The network connected computer is the entry point into the system. The user can control the unit using a regular web browser, pointing at an html site on the Beagleboard computer. The website is served with the Apache2 web server. Once the user has access to the control interface, he can send messages to the Beagleboard computer through keyboard commands, or our script builder interface. Both of these interfaces use Javascript for programming logic.

Once a user sends a command to the Beagleboard, it is interpreted by our PHP control script. This is made possible through the PHP5 module running on the Apache2 web server. Once the PHP control script has the commands, it interprets them into commands which can be understood by the Teensy Microcontroller. The interpreted commands are sent from PHP to our C Microcontroller command program. The command program then passes these commands to the Teensy Microcontroller through the serial port.

The microcontroller controls the motors using the PWM functions of the Arduino library. Each motor is connected to the microcontroller through an H Bridge, which allow us to safely power high wattage devices. The microcontroller sends a PWM signal through the H Bridge to set the speed and direction of the motor.

While the motors are running, our PHP control script polls the IR sensors. To accomplish this, the control script sends periodic commands to the C command program, and checks the results. The command program sends a recognizable message over the serial port, and in reply the microcontroller sends a reading from the IR sensor. Once this is done, the PHP control script determines if the result is in the danger zone. If so, it immediately sends a message to shut off all the motors, and stops any dangerous user input.

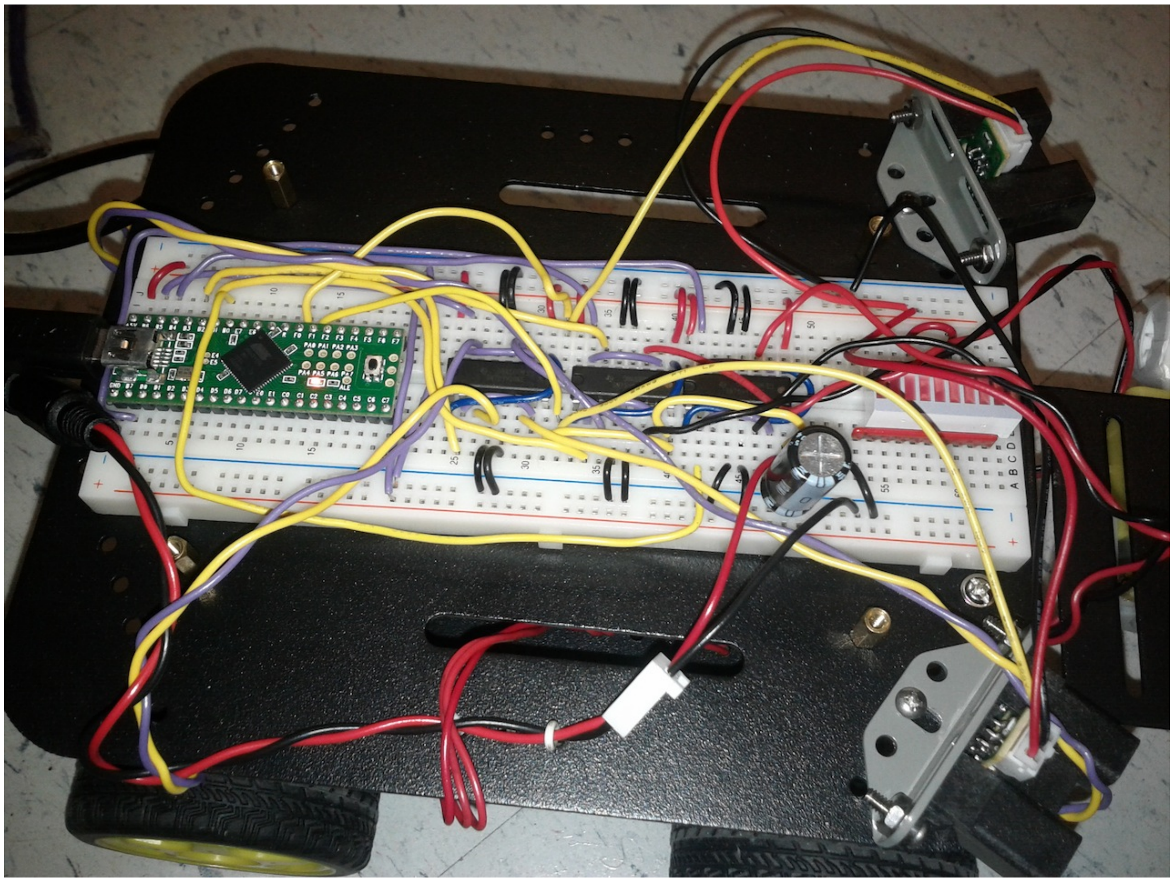
Arduino Functions Used
In the Arduino software, we use analog read and analog write to interface with the motors and IR sensors. These allow us to perform PWM functions in a high level manner. Furthermore, we use serial read, and serial write to communicate from the Arduino to the Beagleboard. The Arduino reads commands from the Beagleboard through the serial port, therefore using serial read. Then, after performing the requested tasks, the Arduino sends it’s reply back to the Beagleboard with the serial write function.

System Intercommunication

User commands are sent to our unit through a web browser using the HTTP protocol. This allows the user to send and receive instructions through a simple web server.

Commands sent to the web server are communicated through simple system calls. In other words, our PHP control script calls the C command program directly through the OS interface.

The C command program sends commands to the Teensy Microcontroller over a serial interface. In this case, the serial interface is emulated over USB2. When the Teensy is done, it returns the results back over the serial interface, where the C command program can read them. For PHP can read the results by asking the C Command program to read and return data in the Serial buffer.



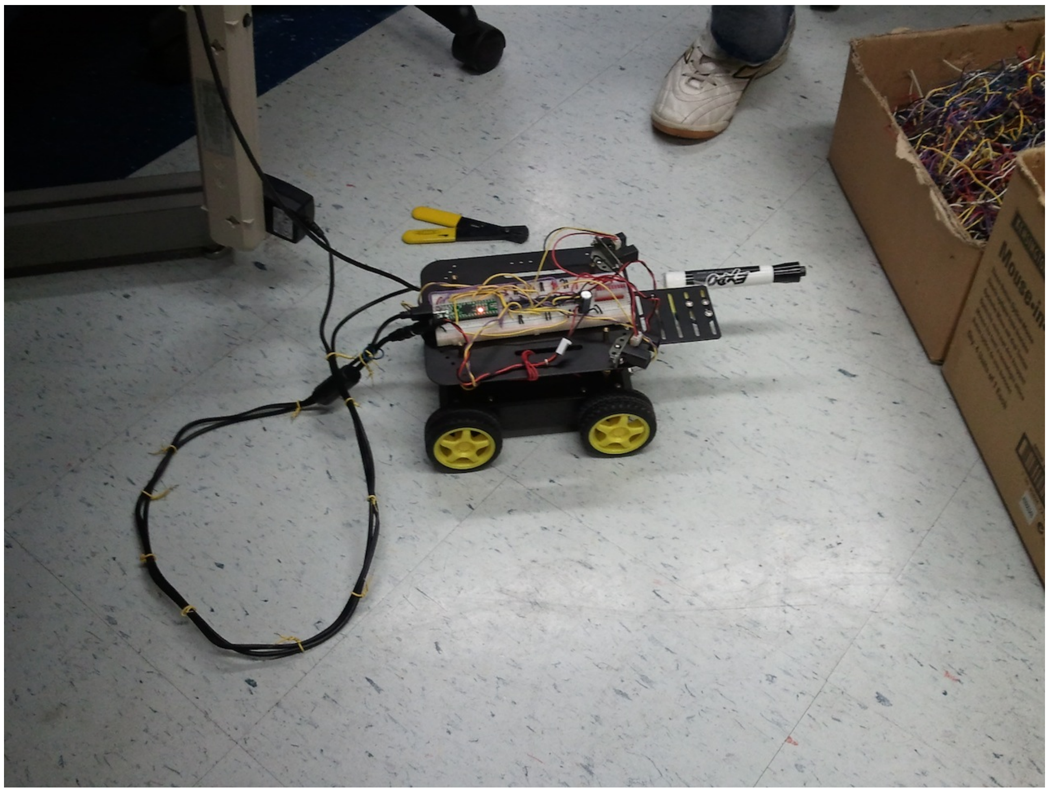
Development Software

To develop on the Teensy Microcontroller, we used the Arduino software with the Teensy library. This simplifies Teensy development. The Arduino software allows us to write and compile code for the Teensy microcontroller. This was installed on each of our personal workstations.

The Beagleboard was loaded with the Ubuntu 11.10 server edition operating system. To access the operating system running on the Beagleboard, we used the screen program, which is available in any Unix-based operating system. Furthermore, we installed sshd on the Beagleboard, which allowed us to connect to the device, and share files through the ssh protocol. We also installed development-essentials, which allowed us to compile our C code directly on the Beagleboard.

To allow users to control the device, we needed a web server to host our control interface, so we installed the Apache2 web server. In addition, we installed the PHP5 module for the given web server, which allowed us to write our control script in PHP.

We used vi as a text editor to write our code. This is available on every Unix-based operating system. We also used git for version control, as well as to share our code. Git was installed on the Beagleboard so that we could make changes on our local workstations, and pull that code directly to the Beagleboard.



System Performance

Component Testing

While developing the system, we tested each component individually. That is to say, each software module was self-contained in a testing environment. First off, we made sure to be able to control the speed and direction of each set of motors. This one done through the Arduino software connected to a simple set of motors. Once complete, we tested sending and receiving commands over the serial port.

Once this was accomplished, we moved onto the C command program. The C command was an interface to send and receive commands over the serial port. This was a simple program to write and test.

Next we decided to build a simple PHP script to receive information from the user control interface. This component was merely there to allow us to develop the control interface, and leave the main control program for last. Once this was done, a simple javascript/html ui was built and tested.

Finally, with all the components in place, we began working on the main PHP control script, which would link the system together. The PHP control script was slowly developed and tested, one function at a time. Once we overcame certain hurdles, we were able to try the entire system, controlling the motor from the HTML control interface.

System Test

With every part of the system in place, we were able to start testing the system as a whole. This helped us to discover all the possibilities which we had overlooked during development, including issues of timing. In this phase, we discovered that the polling function was not as fast as we had hoped. As a result, the system had a hard time performing it's tasks on time, as well as stopping on time before hitting an obstacle. As a result we refined our methods. We were able to bring down the polling the time slightly, as well as taking the timing into account. This allowed us to avoid obstacles more often, and also fixed the issue of performing tasks on time. However, this could still use improvement.

The system test essentially boiled down to a process of identifying bottlenecks, and refining them. However, we feel that the system will still be sub-optimal by the delivery date, given our time constraints.

Here (http://www.youtube.com/watch?v=-zZ_JO5q48g) you can see an amusing video of our early system tests in which we lost control of the unit.

Here (http://www.youtube.com/watch?v=CR_lO8Gk8nI) is another video of our early system tests in which we managed to get the robot to write on the floor.

More videos of our demo can be accessed at these links:

- http://www.youtube.com/watch?v=GFnwlkFPICs
- http://www.youtube.com/watch?v=JFpDOef9nc0

User Manual

System Installation

To install our system on a new device running Ubuntu Linux:

- Install Apache2:
    - sudo apt-get install apache2
- At this point, if you access the device's IP through a web browser while on the same network, you should see a page that says "It Works". This file is located at /var/www and can be altered as you wish.
- If you wish to access our code, install Git:
    - sudo apt-get install git
- Now get the latest version:
    - git clone git://github.com/bzgeb/so422.git /var/www/robot
    - You will likely need root permission to install in /var/www
- Access this directory from [device IP]/robot in your web browser.
- Make sure that the apache user has permission to write to the serial ports.
    - Add users www-data and/or apache to group that owns the serial tty file.
- You should now be able to control the robot through your web browser.

System Initialization

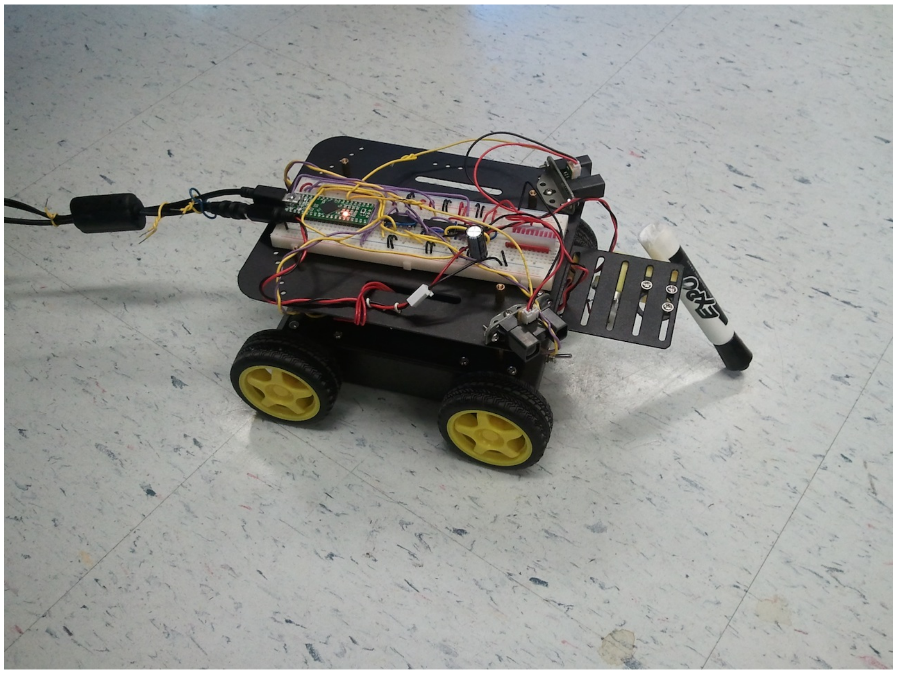
Our system requires little installation. The system can be controlled through any generic web browser with javascript enabled. To initialize the system, the Beagleboard and Teensy must be powered and booted. Afterwards, the user can connect to the system using either a wireless or Ethernet connection. Once a connection is chosen, the user can connect to the device using his browser and plugging in the IP address of the unit.

System Operation

To operate the system, the user must open a web browse and connect to the IP address of the unit, currently this is staticly defined as 169.254.69.69/16. Given this, he will be able to choose from two different control interfaces.

The first control interface is controlled using the keyboard. In this mode, the user controls the robot's movement using the standard keyboard arrow keys. This allows the robot to move forward, backward, turn left, and turn right. Furthermore, the user can raise or lower the unit's drawing apparatus. This allows the user to stop or begin drawing. This is controlled using the 'Z' and 'X' keys.

The second control interface is the script builder. This allows the user to create a script and run it in sequence on the unit. The script interface has several buttons, which correspond to opcodes, as well as a textbox to supply the argument. In this case, the arguments correspond to the amount of time the unit should perform each task. The user can build a script but inputting these commands, and finally clicking the submit button. The unit will then begin execution of the script. The commands include forward, backward, turn left, turn right, pen up and pen down.


EXPO

Discussion

During the course of this project we ran into a few obstacles. Some resource related and some management related.
Due to an over packed course load and a busy work week, our team was unable to find an appropriate meeting time regularly and this caused our project to suffer in terms of the goals we could set. It also caused us to fall behind with our goals.

Initially we had intended to use Bluetooth to communicate with our unit. However, due to technical difficulties, and a lack of resources, we were unable to get Bluetooth to function as desired. As a result, much time was wasted attempting to implement this technology.

After deciding to drop Bluetooth, we decided to use Wi-Fi to connect to our unit. To do this, we had to order a wireless adapter, because there is none available on the Beagleboard. Unfortunately, the Wi-Fi adapter took much longer than expected to arrive. As such, we did all of our development using an Ethernet cable as an umbilical card. As a result, we never had a chance to fully test and implement the Wi-Fi controlled interface.

We also ran into a major roadblock with our Apache2 web server. We wasted many hours trying to debug why, initially, our control interface was unable to communicate with our unit. This was especially frustrating since we were able to control the device by calling the PHP control script directly. Eventually, we discovered that commands run by the Apache2 user didn't have permission to write to a serial device. As a result of this issue, we lost a lot of time, but luckily we overcame the problem.

We ran into a number of problems trying to control a delicate device using non-realtime software. Unfortunately, we discovered too late that our initial software design for the robot was sub-optimal in terms of responsiveness. As a result, we cannot guarantee that our unit will be able to detect obstacles before it's too late, the unit succeeds on a case-by-case basis. This demonstrates the importance of real-time systems when controlling delicate or dangerous software.

In addition, we ran into some issues with the number of bytes being passed back from the Teensy. As it turns outs, when reading from a serial port we can never expect an EOF character. As such, it was not uncommon for our device to get stuck reading from the serial port forever. To overcome this challenge, we decided to reduce our response from the Teensy so a single byte. This meant that no matter what, we could expect at most one byte, which would alleviate the issue of waiting for extra bytes that would never arrive.

This project accelerated our learning knowledge of embedded computing. Two out of our three team members are strictly computer science students, and generally unfamiliar with hardware. As such, this project was the perfect bridge between our knowledge of software, and learning about hardware, due to the balance required.

Finally, the lessons learned in regards to real-time computing couldn't have been more clear. Due to the difficult issues faced in terms of timing and responsiveness, we will never forget the dangers of controlling a delicate piece of hardware without considering these concerns.

Appendix I

Apache2 Web Server

http://httpd.apache.org/

Ubuntu 11.10 Server Edition

http://www.ubuntu.com/

PHP Manual

http://www.php.net/manual/en/index.php

Our Project Hosted on GitHub:

This is useful for the most up-to-date versions of our code

https://github.com/bzgeb/so422

Teensy Code:

```
/*
Protocol
  -A - left motors control
  -B - right motors control
  -C - pen motor control
  -R - read analog value from sensor
*/
void setup()
{
  //Motor Driving Pins
  pinMode(26, OUTPUT);
  pinMode(25, OUTPUT);
  pinMode(24, OUTPUT);
  pinMode(27, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);

  // Direction Control Pins
  pinMode(21, OUTPUT);
  pinMode(22, OUTPUT);
  pinMode(23, OUTPUT);

  //Set all pins to off
  analogWrite(0, 0);
  analogWrite(1, 0);
  digitalWrite(21, 0);
  digitalWrite(22, 0);
  digitalWrite(23, 0);
  analogWrite(24, 0);
  analogWrite(25, 0);
  analogWrite(26, 0);
  analogWrite(27, 0);

  // Turn on LED
  pinMode(6, OUTPUT);
  digitalWrite(6, HIGH);

  // Initialize Serial
  Serial.begin(9600);
  //Serial.println("Start");
  int incomingByte=0;

}

int reg = 0;
bool negative = false;
//array sensor will keep the values read at one moment from the IR sensors
int sensor[3]={0,0,0};


void loop()
{
  if (Serial.available()){
    char ch = Serial.read();

    if (ch == '-') {
      negative = !negative;
      return;
    }
    if ((ch >= '0') && (ch <= '9')) {
      reg = reg * 10 + ch - '0';
      return;
    }
    if (ch == 'A') { // left motor control
     // set left motor

        if (negative)//go backward
            {digitalWrite(22, 1);
           analogWrite(25, (255-reg));
            }
          else //go forward
            {digitalWrite(22, 0);
           analogWrite(25, reg);
            }
      reg = 0;
      negative = false;

    }
    if (ch == 'B') { // right motor control
     // set right motor

        if (negative)//go backward
        {digitalWrite(23, 1);
         analogWrite(26, (255-reg));
       }
      else //go forward
        {digitalWrite(23, 0);
    analogWrite(26, reg);}

  reg = 0;
  negative = false;

}
if (ch == 'C') { // left motor control
 // set left motor

    if (negative)//go down
        {
          digitalWrite(21, 1);
          analogWrite(27, (255-reg));
      }
      else //go up
        {   digitalWrite(21, 0);
           analogWrite(27, reg);
        }

  reg = 0;
  negative = false;

}
if (ch == 'R'){
 switch(reg){
 case 1:
 sensor[0]=analogRead(A0);
 Serial.write(sensor[0] / 4);
 break;
 case 2:
 sensor[1]=analogRead(A1);
 Serial.write(sensor[1]/4);
 break;
 case 3:
 sensor[2]=analogRead(A2);
 Serial.write(sensor[2]/4);
 break;
 default:
 //"No such sensor number exists!"
 Serial.write(-1);

 break;

 }
```

```
      //reset the values in sensor
      memset(&sensor, 0, sizeof(int) * 3);
      //reset input value
      reg = 0;
     }


  }
}
```

(PHP Main Control Program)
Robot.php:

```php
<?php
   $script = $_POST["script"];
   //var_dump($script);

   header("Content-type: text/plain");

   foreach ($script as $operation) {
      $crash  = 0;
      $opcode = $operation["opcode"];
      $arg    = $operation["arg"];

      switch ($opcode) {
         case "fd":
            echo "Forward ho!</br>";
            send("250A250B");
            $crash = wait($arg);
            send("AABB");
            break;
         case "bw":
            echo "Backward ho!</br>";
            send("-250A-250B");
            $crash = wait($arg);
            send("AABB");
            break;
         case "rt":
            echo "Right Turn ho!</br>";
            send("250A-250B");
            $crash = wait($arg);
            send("AABB");
            break;
         case "lt":
            echo "Left Turn ho!</br>";
            send("-250A250B");
            $crash = wait($arg);
            send("AABB");
            break;
         case "pu":
          echo "Pen Up!</br>";
                send("255C");
                $crash = wait($arg);
                send("CC");
          break;
        case "pd":
          echo "Pen Down!</br>";
                send("-255C");
                $crash = wait($arg);
                send("CC");
          break;
        default:
          echo "DERP!!</br>";
          break;
    }

        if ($crash == 1) {
          echo "CRASH";
          return;
        }
}

function wait($time) {

    $timeInMicroSeconds = $time * 1000;
    $timeWaited = 0;
    while ($timeWaited <= $timeInMicroSeconds)
    {
          $timeTaken = -1;
        $timeTaken = poll();

          if ($timeTaken == -1) {
                return 1;
          }

        usleep(250 - $timeTaken);
        $timeWaited += 250;
    }

        return 0;
}

function send($argument) {
    $ret = system("teensyWrite /dev/ttyACM0 " . $argument);
}

function poll() {
        $time_start = microtime(true);
        $sensor1 = 0;
        $sensor2 = 0;
        // Poll sensor 1
        $sensor1 = system("teensyPoll /dev/ttyACM0 1R");

        // Poll sensor 2
        $sensor2 = system("teensyPoll /dev/ttyACM0 2R");
          $time_end = microtime(true);

          if ($sensor1 > 400) {
              return -1;
          }

          if ($sensor2 > 400) {
           return -1;
        }

          return $time_end - $time_start;
    }
?>
```

(Script Builder Interface)
Robot.html:

```
<!DOCTYPE html>
    <head>
        <meta http-equiv="Content-type" content="text/html; charset=utf-8">
        <title>Turtle Power</title>
        <script src="js/jquery-1.7.min.js"></script>
        <script type="text/javascript">
            var $script = [];
            var $scriptString = [];
            $(document).ready(function() {

                // Submit Button Junk
                $("#submit").click(function() {
                    var $robotUrl = "robot/testRobot.php";

                    $.post($robotUrl, {script: $script}, function(data) {
                        console.log(data);
                        $("#script").html("RESULTS! </br>" + data);
                        $(".command").removeAttr("disabled");
                        $("#arg").removeAttr("disabled");
                        $("#submit").removeAttr("disabled");
                    });

                    $("#script").html("Sending Script!");

                    $script = [];
                    $scriptString = [];

                    $(".command").attr("disabled", "true");
            $("#arg").attr("disabled", "true");
            $("#submit").attr("disabled", "true");
          });

          // Command Buttons Junk
          $(".command").click(function(data, element) {
            var $command = $(data.target).attr("name");
            var $arg = $("#arg").val() || "42";

            $scriptString.push([$command, $arg].join(" "));
            $script.push({opcode:$command, arg:$arg});
            console.log($scriptString);
            $("#script").html($scriptString.join("</br>"));
          });
        });
      </script>
    </head>
    <body>
      <input type="button" id="fd" name="fd" value="forward"    class="command">
      <input type="button" id="bw" name="bw" value="back"       class="command">
      <input type="button" id="rt" name="rt" value="Right Turn" class="command">
      <input type="button" id="lt" name="lt" value="Left Turn"  class="command">
      <input type="button" id="pu" name="pu" value="Pen Up"     class="command">
      <input type="button" id="pd" name="pd" value="Pen Down"   class="command">

      <input type="text"   id="arg" name="arg" value="">

      <p id="script">SCRIPT!</p>
      <input type="button" id="submit" name="submit" value="submit">
    </body>
</html>
```

(Keyboard Interface)
Magicpanel.html:

```
<!DOCTYPE html>
<html>
<head>
        <title>My Page</title>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <meta name="Author" content="Eric Chan" />
        <script type="text/javascript" src="js/jquery-1.7.min.js"></script>
        <script type="text/javascript">

$(function() {
        $("#stop").click(function() { $.get('logo_stop.php') });
        $("#fd").click(function() { $.get('logo_fd.php') });
        $("#bk").click(function() { $.get('logo_bk.php') });
        $("#lt").click(function() { $.get('logo_lt.php') });
        $("#rt").click(function() { $.get('logo_rt.php') });
        $("#pu").click(function() { $.get('logo_penup.php') });
        $("#pd").click(function() { $.get('logo_pendown.php') });

})


window.onkeydown = function(e) {
      var key = e.keyCode;

      if (key == 37) $.get('logo_lt.php');
      if (key == 38) $.get('logo_fd.php');
      if (key == 39) $.get('logo_rt.php');
      if (key == 40) $.get('logo_bk.php');
        if (key == 88) $.get('logo_penup.php');
        if (key == 90) $.get('logo_pendown.php');
}
window.onkeyup = function(e) {
        $.get('logo_stop.php');
}

        </script>
        <style type="text/css">
        </style>
</head>
<body>

<button id="stop">STOP</button>
<button id="fd">Forward</button>
<button id="bk">Backward</button>
<button id="lt">Left</button>
<button id="rt">Right</button>
<button id="pu">Move</button>
<button id="pd">Draw</button>

</body>
</html>
```