RUTGERS
THE STATE UNIVERSITY
OF NEW JERSEY

Department of Electrical and Computer Engineering

332:428 Wireless Capstone Design Spring 2011

SMARTMOUNT: Controlling a Wireless Smart Home Environment

Team Members:
Christos Giannopoulos
Kristine Javier
Michael Lee
Atikur Miah

Date: May 2, 2011

Table of Contents

1. Introduction 3

2. Motivation 3

3. Parts List and Budget 4

4. Communication Design 4
4.1 Microcontroller 4
4.2 Java GUI 8

5. Mechanical Design 12
5.1 Initial Design 12
5.2 Final Design 14

6. Future Considerations 16

References 17

Appendix 1: Part Specifications 18

Appendix 2: Microcontroller Code 24

Appendix 3: GUI Code 27

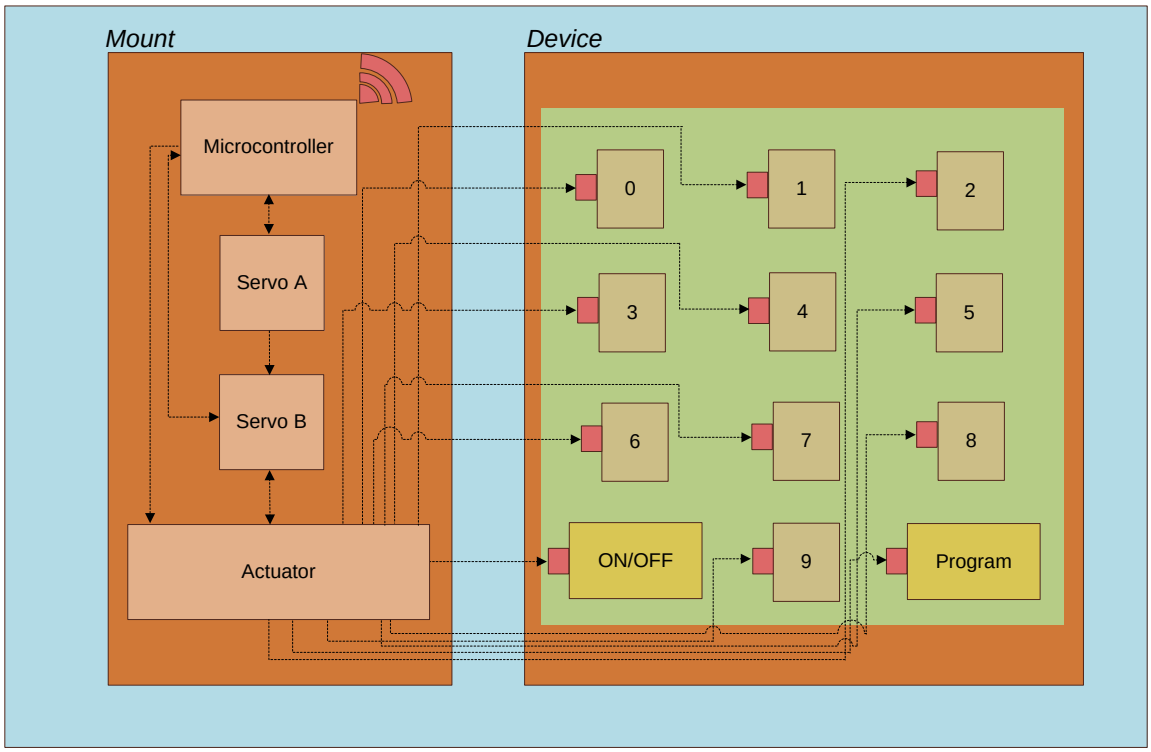
1. Introduction

The main goal of this project is to develop a "mount" that interfaces with a wide array of digital input control panels, incorporating the ability to control various appliances through a remote location. A programmable user interface will control ordinary household devices remotely based on specific user input.

2. Motivation

Current remote appliance control systems only advertise the ability to control devices that are tailored specifically to that manufacturer. Some systems have physical control units that must be installed within the home while others require mobile applications for smart phone platforms (e.g. iPhone, Android, etc.) to use as a control device. Mainstream smart home environments such as “X10” focus mainly on remote access of security devices.

The SMARTMOUNT system differs in that the controlling device consists of a physical mount that is adjustable to multiple digital input surfaces. The mount is compatible with various manufacturers’ designs and can be used for multiple household appliances. The Wi-Fi protocol is used to seamlessly integrate the developed control algorithm to readily available components within the home, where the user can then access these appliances through Internet-capable devices such as a computer or cell phone.



SMARTMOUNT Block Diagram

The desired devices that will be remotely controlled are push-button household appliances such as an oven, television, lights, and microwave. This system is targeted to those who desire access to the appliance remotely and who wish to have these devices running once they reach it. The user distance can range from an adjacent room within the home to a different town. A user can set his lights and television to turn on, set the oven to start preheating, and set the thermostat to turn on just a few minutes before he/she arrives home, essentially eliminating any waiting time. Maximizing convenience for the user is the ultimate goal.

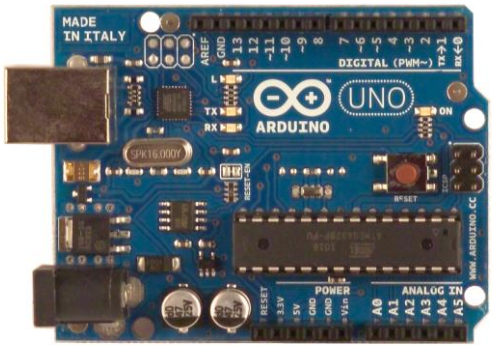
3. Parts List and Budget

Component	Manufacturer	Quantity	Price (ea.)
HS-322HD Standard Deluxe Servo Motors	Hitec	2	$9.99
Micro Maestro 6-Chan. USB Servo Controller	Polulu	1	$19.95
PQ12 –S Linear Actuator	Firgelli	1	$65.00
4 in. Wood Arm	Lowe's	2	$1.98
Wood Base Block	Lowe's	1	$5.00
1 in. Wood Screw	N/A	4	N/A
¼ in. Wood Screw	N/A	1	N/A
¼ in. Nut & Bolt	N/A	1	N/A
1 in Nail	N/A	4	N/A
Servo attachments	Hitec	2	N/A
Arduino Uno Microcontroller	Arduino	1	$30.00
Adafruit Motor Shield	Adafruit	1	$20.00
AsyncLabs WiFi Shield	AsyncLabs	1	$60.00
Total			$211.92

4. Communication Design

4.1: Arduino Microcontroller

In order to make the physical mount and Graphical User Interface (GUI) communicate with each other, the use of a microcontroller is required. A microcontroller is a self-contained embeddable system (composed of traditional computing components such as a CPU, memory, etc.) that is highly adaptable based on an inherent need. In the SMARTMOUNT system, the Arduino Uno, a programmable open-source single-board microcontroller was incorporated:



Arduino Uno

Referencing the specification sheet (detailed in the Appendices section), the Arduino Uno operates at a regulated 5V voltage, obtained from the available USB or DC input. It has a 32 KB flash memory with 14 digital and 6 analog pins available for programmable use. In order to communicate with adaptable project-specific boards (i.e. "shields," to be detailed later), an incorporated SPI (Serial-Peripheral Interface) is present, giving the microcontroller the ability to communicate with these devices quickly.

The SPI is composed of four major components: a MISO (Master-In Slave-Out), a MOSI (Master-Out Slave-In), an SCK (Serial Clock), and a SS (Slave Select). Briefly, the MISO and MOSI lines are utilized to send data between both the Arduino board and the subsequent boards that are attached to it. The SCK is a clock pulse that is used to synchronize data generated by the Arduino and the SS allows the Arduino to select which attached device it wants to interface with. The SPI is important as it provides synchronized communication between the Arduino board and the specific control boards utilized in this system.

In order to program the Arduino board, a Java-based pre-developed IDE (containing defined libraries) is utilized. After writing the desired code (based on C/C++) to control the Arduino board, the developer uploads the program (known as a "sketch") through the integrated USB interface, communicating with the board through a Serial to USB interface.

To realize the remote control aspect of the design, the integration of application-specific shields were required. In the subsequent sections, the implementation of both mechanical control and use of the mount over Wi-Fi will be discussed.

4.1.1: Wi-Fi

To develop the remote control aspect of the mount, the inclusion of a module operating at the IEEE 802.11 Wi-Fi protocol was required. The specific shield implemented in this design was the AsyncLabs WiShield v2.0:

WiShield v2.0

The WiShield contains the Microchip developed MRF24WB0MA Wi-Fi transceiver, an 802.11b Wi-Fi certified device capable of 1 to 2 Mbps speeds and multiple encryption schemes (64/128 bit WEP, WPA/WPA2). The shield draws power from the regulated 5V provided by the Arduino and utilizes the SPI connection to communicate with the Arduino board, specifically digital pins 10 – 13 for the previously defined SPI communication components.

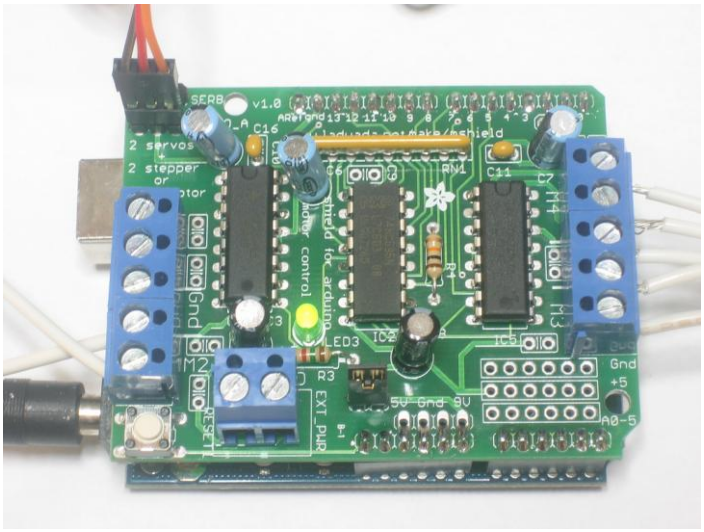
Although capable of multiple implementation schemes, the SMARTMOUNT system utilizes the WiShield as a server, using a modified version of the provided WiServer program. This sketch utilizes the incorporated Infrastructure mode, providing the capability to communicate with a remote AP device (wireless router). Unable to be dynamically assigned an IP address through the use of DHCP (Dynamic Host Configuration Protocol**)**, a defined static IP address, gateway IP, and subnet mask is programmed into the sketch, as well the encryption scheme. As detailed in the developed code (found in the Appendices section), this system utilizes WPA2, the strongest of the commercially available encryption methods. In order for a user to control the module from a remote location, their internet-capable device will have to be verified through a VPN (Virtual Private Network) incorporated in their wireless router. A VPN provides the ability to control locally hosted peripherals on a secure remote connection, assuring the user has maximum security and control over their home appliances.

In order to communicate with the developed GUI, a series of HTML pages have been created that produce specific commands to control the mechanical aspect of the mount. A page has been developed for each possible position that the mechanical arm system of the SMARTMOUNT can traverse, as well as to actuate a button press. Based on the pages accessed by the GUI, the mount will position itself to a desired point and initiate a button press. The physical mapping of the GUI with the SMARTMOUNT will be detailed in the following sections.

While the WiShield provides the ability to communicate with the Arduino remotely, it has no bearing on the mechanical control of the SMARTMOUNT. In order to incorporate this aspect, another shield is required.

4.1.2: Motor Control

To control the mechanical aspects of the SMARTMOUNT, the Adafruit Motor Control shield was incorporated into the design:



Adafruit Motor Control Shield

The Adafruit Motor Control shield has the ability to control multiple mechanical devices including hobby servos, DC motors, and stepper motors. Based on the controlled device, it utilizes a different part of the shield. To control the two inputs for hobby servos, a 3-pin implementation for each servo (controlled by digital pins 9 and 10) is incorporated, deriving both voltage (regulated 5V) and the input signal from the Arduino board. Both DC motors and stepper motors are controlled by the two sets of 5 terminal blocks which implement a set of 4 H-Bridges (utilizing digital pins 4, 7, 8 and 12), allowing the motors to be controlled bi-directionally. The Motor Control shield also has the ability to derive its power from an external source through the EXT_PWR block, in case the regulated 5V from the Arduino board is insufficient.

As detailed, the SMARTMOUNT system utilizes a set of 2 servo motors along with an actuator device (with a DC motor inside). In order to implement these devices into the developed control scheme, it was first apparent to resolve an existing pin conflict between the two shields. When referencing the pin schematics of both devices (as detailed in the Appendices section), it is apparent that a servo control pin (digital pin 10) and DC motor control pins (digital pins 7 and 12) were already being used by the WiShield SPI. In order to implement these devices, digital pins 14, 15, and 19, a set of unused pins, were utilized. For the servo motors, Servo A used pin 9

(deriving the necessary power and signal without need for change) while Servo B used pin 19 as the signal pin and derived power from the EXT_PWR block. To incorporate the actuator, pins 14 and 15 were used, applied signals of opposite polarity in order to control the extension and retraction of the actuator.

When referencing the developed sketch (as detailed in the Appendices section), the mechanical control can be defined by the two distinct components that compose the mount, the servos and the actuator. The servos were controlled utilizing a pre-existing Servo library, where after a servo object was initialized; a discrete value for its angular position (with bounds of 0° to 180°) could be written and thus actualized. To control the actuator, a PWM (Pulse Width Modulated) signal was written to the relative pin, where a signal sent to pin 14 initiated an extension of the actuator and pin 15 initiated a retraction. To briefly detail, a PWM signal is a technique for obtaining analog results with digital means, defined by the Arduino library as a square wave with a user-defined duty cycle. Based on the value of the duty cycle, the speed of DC motor is altered. Utilizing the analogWrite command, the PWM signal is written to necessary pin, imitating the button press.

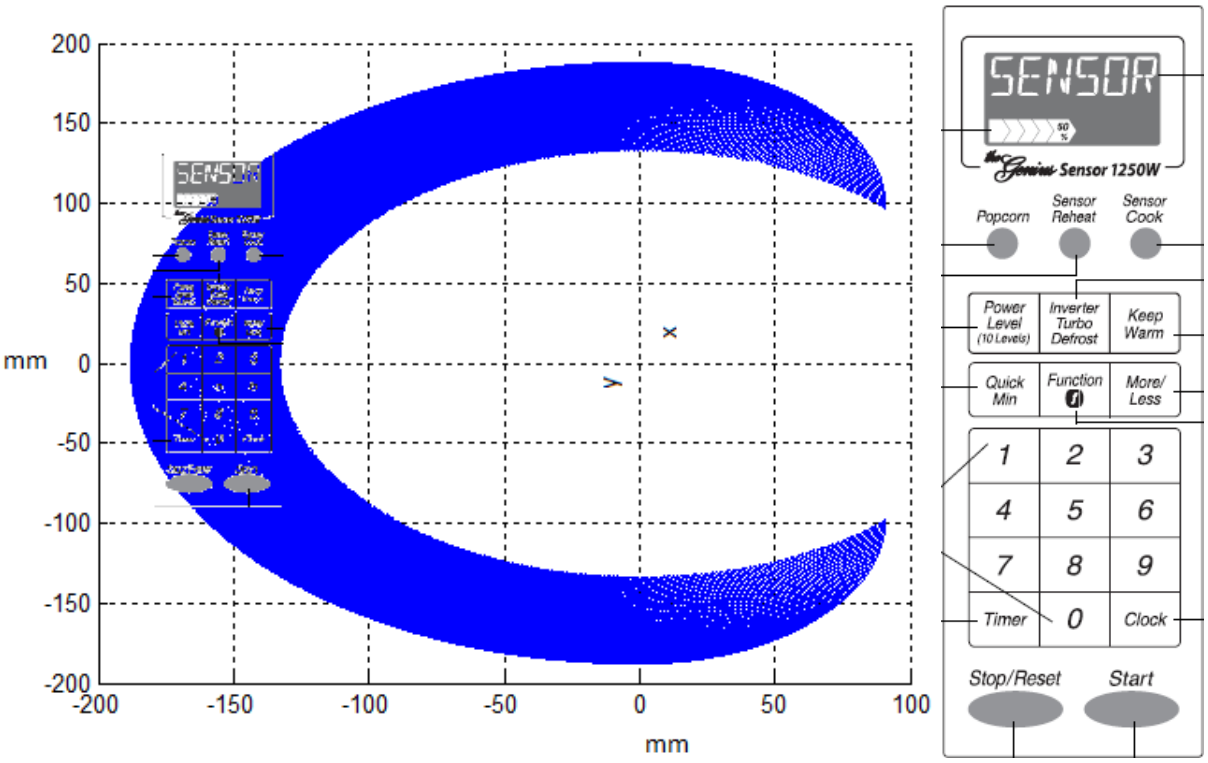
4.2: Java GUI

4.2.1: Adaptability and Mapping

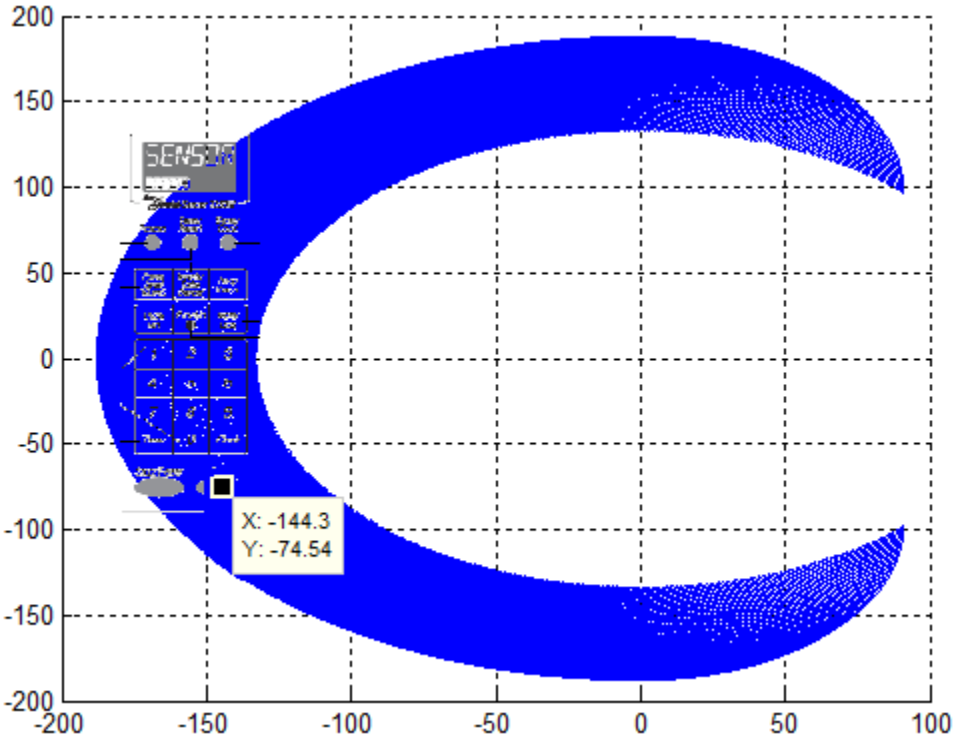
One big issue for the SMARTMOUNT device was the problem of adaptability. Due to the fact that there are so many different button layouts for devices of the same type, it is necessary to create a mapping/coordinate system to easily transition among different microwaves or other devices. A plot from MATLAB was created to visualize the mapping system and develop coordinates for buttons because the servo motors used in the prototype are limited to 180 degrees, which greatly limits the ability of the arm to hit every point in a 360 degree radius. When using polar coordinates, the outer arm of the SMARTMOUNT changed its axis and angle based on the angles of the inner arm. This plot is necessary because it allows the easy conversion between polar and Cartesian coordinates. Each blue dot on the plot represents a possible position for the actuator on the SMARTMOUNT.

4.2.2: Model NHH765BF

In the example below, the control panel from the operating instructions found on Panasonic's website has a height of 230 mm. By carefully keeping the same ratio by resizing the figure in Microsoft Paint and using transparency, it can be shown that the SMARTMOUNT can reach all of the button positions on the control panel.



By carefully overlaying the MATLAB plot over the figure above, the "mapper" can find the X and Y coordinates, and convert back into polar coordinates to determine the necessary angle for each servo motor. By measuring the farthest blue dot from the origin, the user will know exactly where the mount should be placed in order to have the correct calibration.



A text file named after the model number is needed to store all of the button positions, as well as the angles needed for the two motors. The format of an example file titled "NNH765BF" is shown in the Appendix.

4.2.3: Plan to Outsource Development and Possible Business Model

Since there are so many different control panels, the algorithm above can be used in order to outsource the work. Skilled and experienced professionals in foreign countries, such as India, have lower costs of living than in the United States. While a typical entry level programmer is paid approximately $57,000 per year, programmers in India are paid approximately $15,000 per year.[1]

By outsourcing the work overseas, the project costs will be much lower, a reduction of approximately 70 percent in programming costs.

4.2.4 Software Considerations

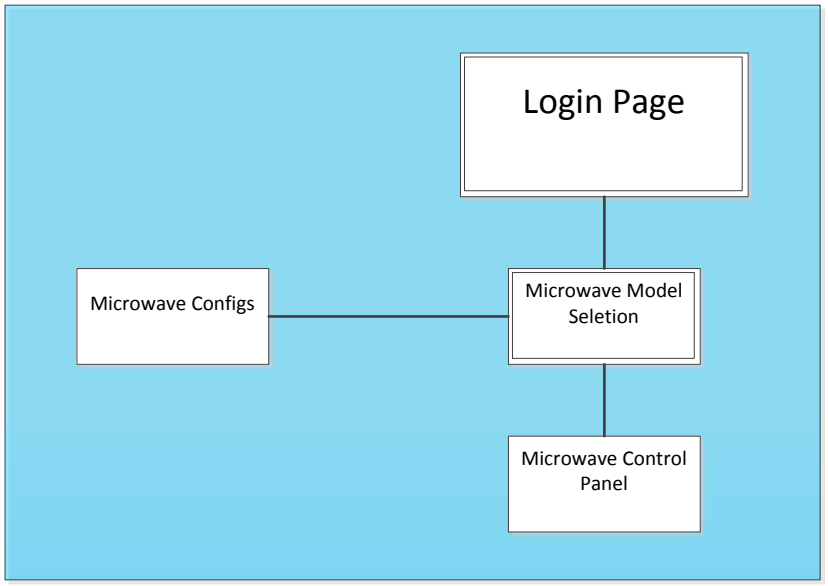
The software used for the Graphical User Interface (GUI) is a program called Netbeans, which uses the Java programming language. As mentioned above, the GUI has a folder in the program containing text files with the button coordinates and servo motor angles for each button. Since many buttons are the same for different microwave control panels, each button is identified with a number code:

…

1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
0	10
11	Start
12	Reset
13	Popcorn
14	Timer
15	Clock
16	More
17	Less
18	Power Level
19	Quick Min
20	Defrost
21	Auto Reheat
22	Keep Warm
23	Serving/Weight
24	Auto Cook (1-3)
25	Auto Cook (4-6)
26	Auto Cook (7-9)
27	Sensor Cook
28	Sensor Reheat
29	Function

…

This is much more adaptable than having to create a new frame for each control panel, as the files may be added directly into the configurations folder. A simple block diagram is shown in the figure below:



Inside the Microwave Control Panel block, the GUI attempts to connect to the static IP of the microcontroller and access the pages created on the WiServer.

5. Mechanical Design

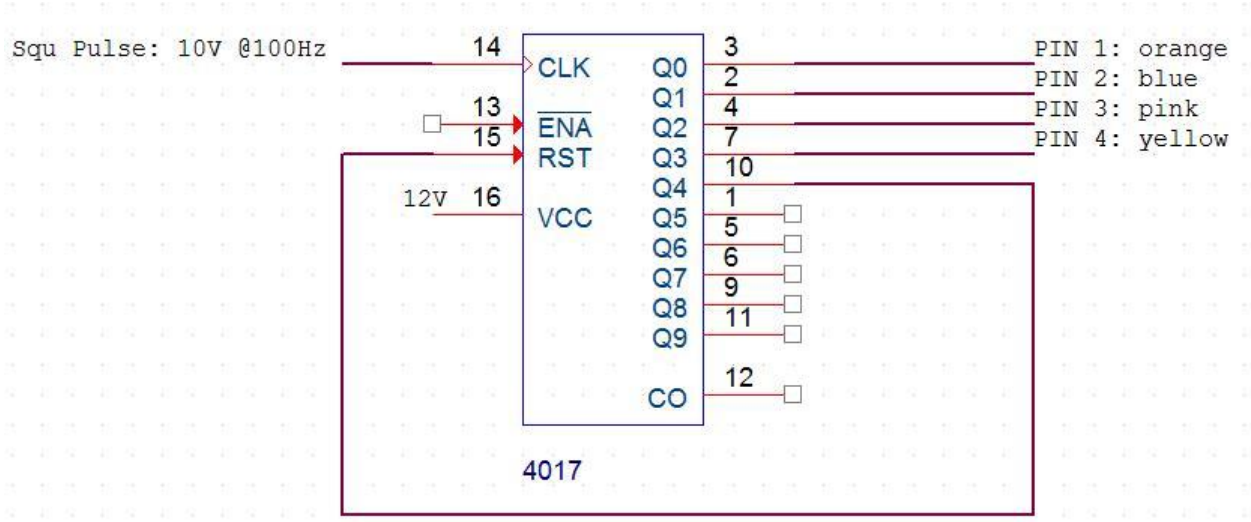
5.1: Initial Design

Initially, the mount was to have an X-axis and Y-axis system in order to place a miniature linear actuator in front of the specified appliance button. Low cost, compact size linear positioning tracks that provided both X and Y movement cost about $1,400. Instead, cheaper single-axis tracks were explored to cover the x-axis range and have a circular disk on a motor to provide the Y-position. The single axis tracks were still very costly, so the idea of a "robotic arm" evolved. The idea was for two stepper or servo motors to move two wooden blocks to the requested position, mimicking arm movement.

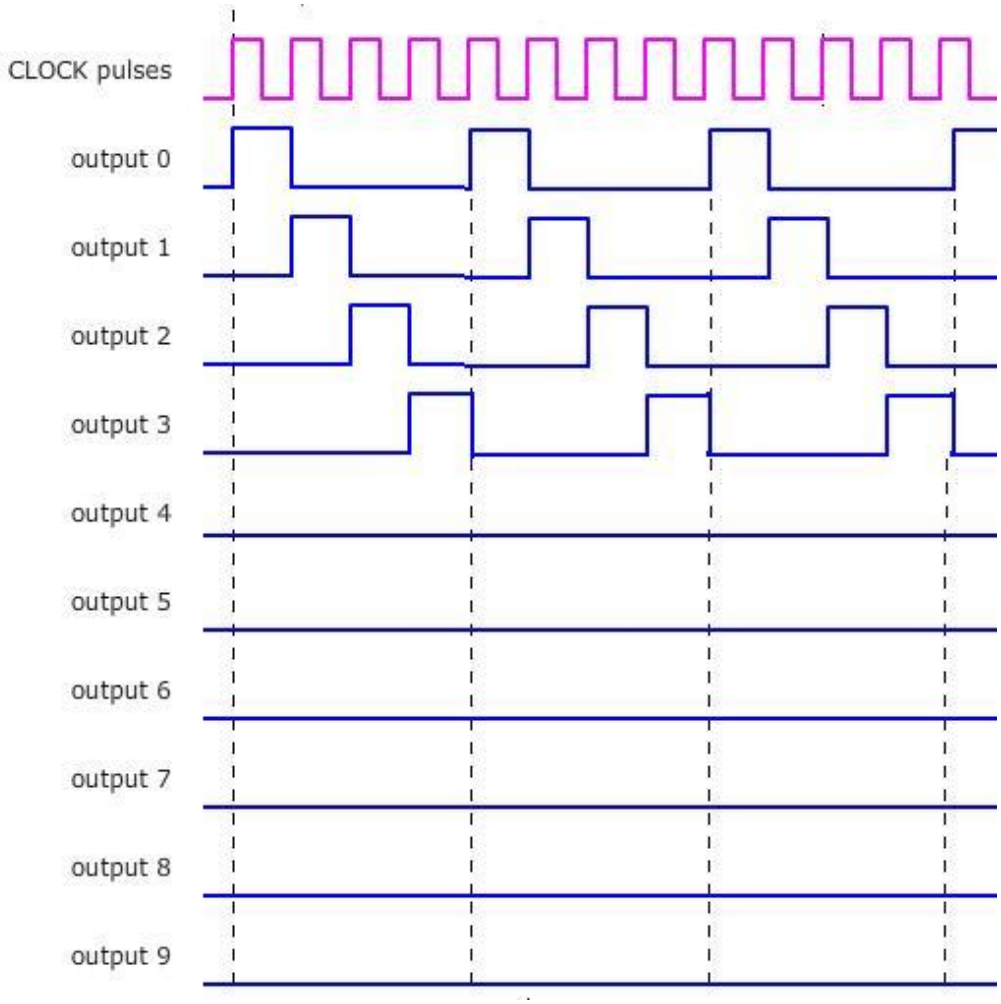
After seeking advice from hobbyists, the stepper motor was chosen to use as a translation mechanism for the wooden arm. A stepper motor steps at a specific resolution for each pulse. The MOTS 2 stepper motor was chosen since it is lightweight and moves precisely 7.5° at each pulse (equivalent to 48 steps per revolution). The stepper motor is made up of a rotor, which is the permanent magnet that rotates, and stator, which are the stationary four coils. The rotor is moved by applying a pulse DC voltage to either one or two coils at a time in sequence, depending on the type of stepping mode that is required. There are three stepping modes: single stepping (one coil at a time, 48 pulses/revolution), high torque stepping (two coils at a time, 48 pulses/revolution), and half stepping (3 coils at a time, 96 pulses/revolution).

In order to individually turn each coil on, a decade counter was used to send a pulse to each coil in sequence. The LM4017 is a decade counter that takes a clock pulse in and steps the output from negative to positive in a series of ten steps, where only one of the individual outputs is on at a time. This is different from a binary coded decimal (BCD) counter, which has four

outputs A,B,C,D, where A is the LSB. The outputs of a BCD counter go in sequence from binary 0000 to 1001, which is equivalent to decimal numbers 0 to 9. The LM4017 consists of a 3-stage Johnson counter with a decoder to convert the Johnson counter outputs into decimal numbers. In the case of the stepper motor, each of the four coils must be energized individually in a certain order. The LM4017 counts to a specified number and restarts the count or it can count to a certain number and stop. In order for the stepper motor to turn, a clock pulse must be applied to each of the four stepper motor pins separately. Each successive coil in the motor must be energized. If the coils are incorrectly connected, the motor will not rotate. In order to keep the motor rotating, the first four output pins (LM4017 pins 3,2,4,7) are connected to the four pins of the stepper motor in order (orange, blue, pink, yellow) so that each pin goes high after another. To continuously count through this four pin sequence, the fifth output pin (LM4017 pin 5) is connected to reset (LM4017 pin 15). The LM4017 is powered by connecting 12V to V_{DD} (LM4017 pin 16) since it operates between 3VDC and 15VDC. The circuit diagram for this connection is shown in the figure below:



With the stepper connected as in the above schematic, the counting sequence for each output pin will look like this:



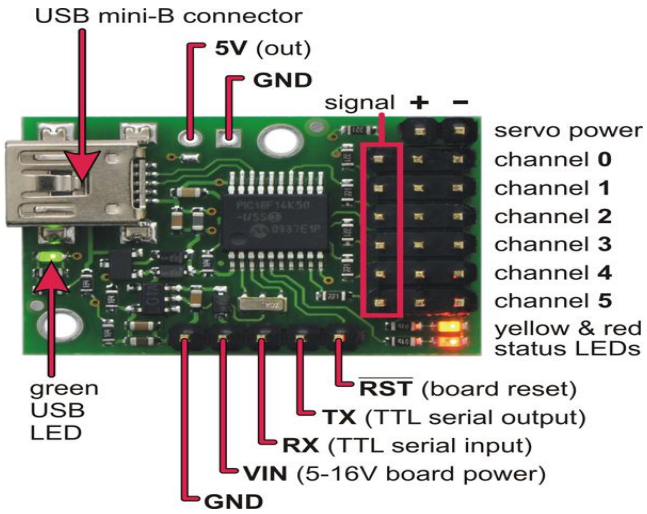
5.2: Final Design

The final design of the mount consisted of servo motors rather than stepper motors. HiTEC's HS-322HD Deluxe servo motor is not only more durable but it can also generate higher torque and speed. The torque is 3kg-cm at 4.8V and 3.7kg-cm at 6v. The servo can operate at a speed of 0.19 seconds/60° at 4.8V or 0.15 seconds/60° at 6V. This equates to a servo with 66.7 rpm, whereas the MOTS 2 stepper max speed is 11.8 rpm. Additionally, this servo weighs 43g, which is ¼ lighter than the stepper at 59g. The servo's resin outer gear shaft allowed a setup that made attaching and detaching the wooden arms easy. Most servos rotate up to about 180°, the HS-322HD rotates up to 210°, particularly. Although full 360° rotation motors are available, they are pricier. The servos are active devices, so that when they are commanded to move to a position they also actively hold their position. If there is an external force pushing the servo, it will resist with up to the maximum torque rating. The servo only maintains position for as long the position pulse is repeated. The maximum holding time is the command repitition rate of 20ms. Two HS-322HD servos were used in the SMARTMOUNT design. One servo was

attached to a piece of 2” by 4” wooden block. For the first arm, a 0.25” x 1.8” x 4.5” piece of wood was attached to the motor shaft. Another 0.25” x 1.8” x 4.5” was then attached to the first wooden arm. Lastly, the actuator was attached the second wooden arm to carry out the pressing function. The placements of the wooden arms onto the motor shafts were adjusted accordingly to allow the actuator to press all the buttons on a mock oven. Depending on the appliance being used, each of the wooden arms on the mount can be attached in a position which will optimize the performance. The adjustment is only needed due to the use of non-360° rotation motors.

The servo motor only contains a few parts which include an electric motor, a potentiometer, and a control board. A controlled Pulse-Width Modulated signal is received by the servo from an outside source. The servo controller board interprets the signal input and turns on the geared motor inside. This is processed by a Pulse Width to Voltage Converter, which demodulates this signal to obtain a voltage that relates to the width of the input pulse signal. The width of the pulse is proportional to the position of the motor. The output voltage is buffered and so does not decay drastically between pulses. Therefore, the length of time between pulses is not important, though the general period of the signal is 20ms. The circuit is tuned to produce a useful voltage over a 1ms to 2ms period. Most servos rotate up to about 180°, the HS-322HD rotates up to 210°. In this case, the 1ms would represent 0°, 2ms would represent 210°, and 1.5ms would rotate the motor 105°. The motor, using a series of gears, turns the output shaft and the potentiometer simultaneously. The potentiometer is fed into the servo control circuit and when the control circuit detects that the position is correct, it stops the motor.

Pololu Micro Maestro Servo Controller board was used to test the design. The following is a pinout of the controller:



The Micro Maestro consists of 6 channels, permitting the use of up to six servos simultaneously; however, the SMARTMOUNT design requires only two. Using a USB A to mini-B cable, the Micro Maestro is connected directly to a computer. Using the schematic and control program from the Pololu website, the servos set up and were moved to desired positions. The Micro Maestro requires 5V to operate and the servos require 6V. An LED light attached to the Micro Maestro pertains to the status of the controller. The green light indicates the USB connection to the computer. When the LED flashes yellow light slowly, it indicates that the

controller has not yet determined the baud rate. When the light blinks faster, it has detected the servo period, and blinks proportionally to the servo period. The red light on the LED indicates error in the setup. The mock oven design was used for testing purposes. Different position sequences were generated along with variations in motor speed to test for precision, practicality and durability.

The PQ12-S Linear Actuator used to press buttons is easily controlled applying a DC voltage to extend the actuator. To retract the actuator, the polarity is reversed. It operates at a current of 550mA at 6V or 220mA at 12V. The maximum force 9N and the speed are adjustable by alternating the applied load. The limit switches will turn the power off when the actuator reaches 1mm of the end of the stroke.

6. Future Considerations

After full realization of the SMARTMOUNT design, there are a few aspects that can be revised if provided the opportunity.

First, the prototype construction of the mount is physically made out of wood. While easily obtained and constructible, there is a decrease in both the aesthetics and lifespan of the device. In future revisions, a more applicable material such as steel or iron would be appropriate as it is more physically and visually adaptable to a home environment. Second, a consideration exists on the usage of the motor control shield on the microcontroller device. Based on the amount of pin conflicts that existed and the availability of multiple digital pins even after implementing the WiShield, it would be applicable to remove the motor shield all together and save both power and cost. Lastly, the concept of user feedback is a component that was not implemented in this prototype. In future revisions, it would be highly appropriate to incorporate an attachable sensor device that could notify the user when a button has been pressed, assuring that the desired commands have been executed in succession.

These considerations are a product of previous attempts at implementation. The lessons learned from constructing the prototype for the SMARTMOUNT are highly valuable for potential revisions.

References

http://www.firgelli.com/Uploads/PQ12_datasheet.pdf Firgelli, Linear Actuator PQ12-S datasheet.

http://www.pololu.com/docs/pdf/0J40/maestro.pdf Pololu, Maestro Servo Controller User's Guide.

http://www.servocity.com/html/hs-322hd_standard_deluxe.html Hitec, HS-322HD Servo Specifications.

http://www.velleman.eu/distributor/products/view/?country=be&lang=en&id=75006 Velleman, Mots2 Stepper Specifications.

http://www.doctronics.co.uk/pdf_files/HEF4017B.pdf 4017 datasheet.

http://www.expertsfromindia.com/designerdevloper.htm

http://www.panasonic.com/ Appliance schematics.

http://www.doctronics.co.uk/4017.htm LM4017 Specifications.

http://electronics-diy.com/electronics/stepper_motors.php Controlling Stepper Motors with a Parallel Port.

http://shieldlist.org/adafruit/motor Motor Control Shield pin layout.

http://shieldlist.org/asynclabs/wishield-v2 WiShield pin layout.

http://www.asynclabs.com/wiki/index.php?title=WiServer WiServer documentation.

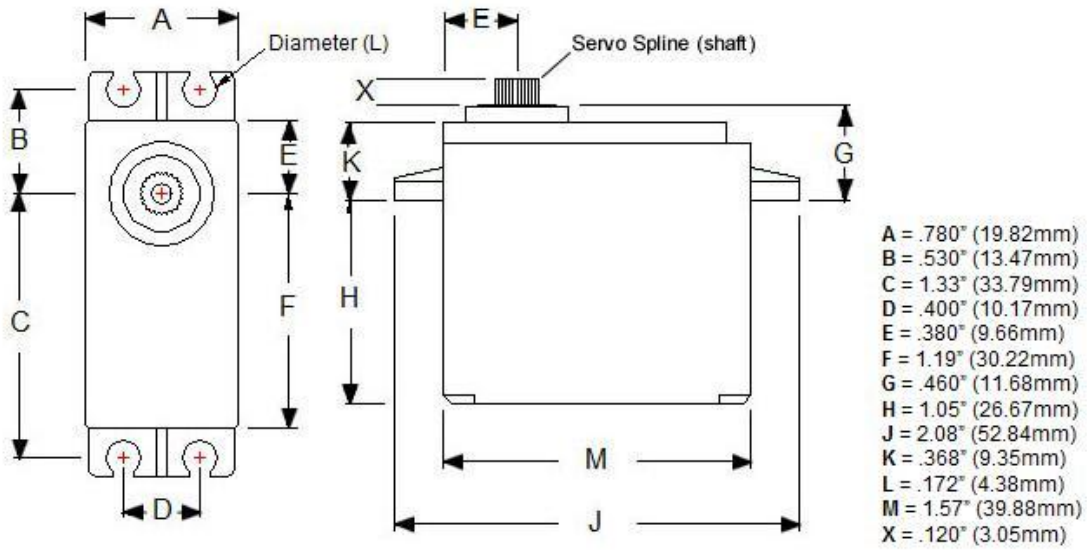
http://www.ladyada.net/make/mshield/ Motor Shield documentation.

http://arduino.cc/ Arduino documentation.

Appendix 1: Part Specifications

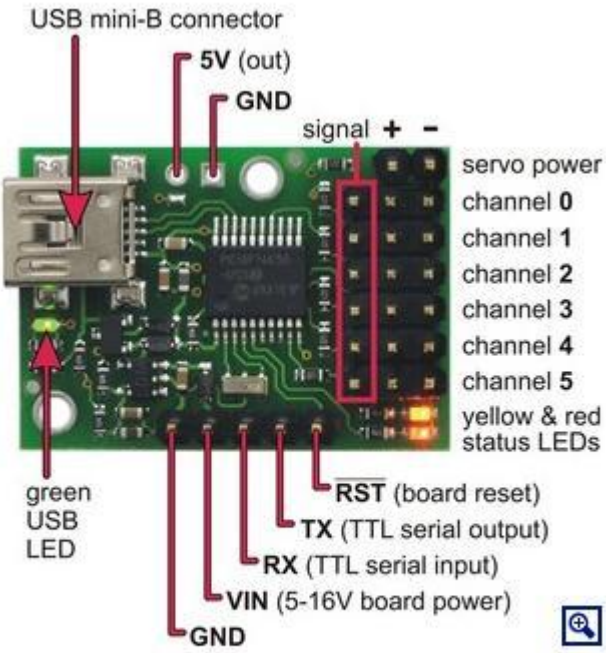
HS-322HD Standard Deluxe Servo Motors

Control System: +Pulse Width Control 1500usec Neutral
Required Pulse: 3-5 Volt Peak to Peak Square Wave
Operating Voltage: 4.8-6.0 Volts
Operating Temperature Range: -20 to +60 Degree C
Operating Speed (4.8V): 0.19sec/60° at no load
Operating Speed (6.0V): 0.15sec/60° at no load
Stall Torque (4.8V): 42 oz/in (3.0 kg/cm)
Stall Torque (6.0V): 51 oz/in (3.7 kg/cm)
Current Drain (4.8V): 7.4mA/idle and 160mA no load operating
Current Drain (6.0V): 7.7mA/idle and 180mA no load operating
Dead Band Width: 5usec
Operating Angle: 40 Deg. one side pulse traveling 400usec
Direction: Clockwise/Pulse Traveling 1500 to 1900usec
Motor Type: Cored Metal Brush
Potentiometer Drive: 4 Slider/Direct Drive
Bearing Type: Top/Resin Bushing
Gear Type: Heavy Duty Resin
360 Modifiable: Yes
Connector Wire Length: 11.81" (300mm)
Dimensions: See Schematics
Weight: 1.52oz (43g)



Micro Maestro 6-Chan. USB Servo Controller

Size: 0.85" x 1.20"
Weight: 4.8 g
Channels: 6
Baud: 300 - 200000 bps
Minimum operating voltage: 5 V
Maximum operating voltage: 16 V
Supply current: 30 mA
Partial Kit: N

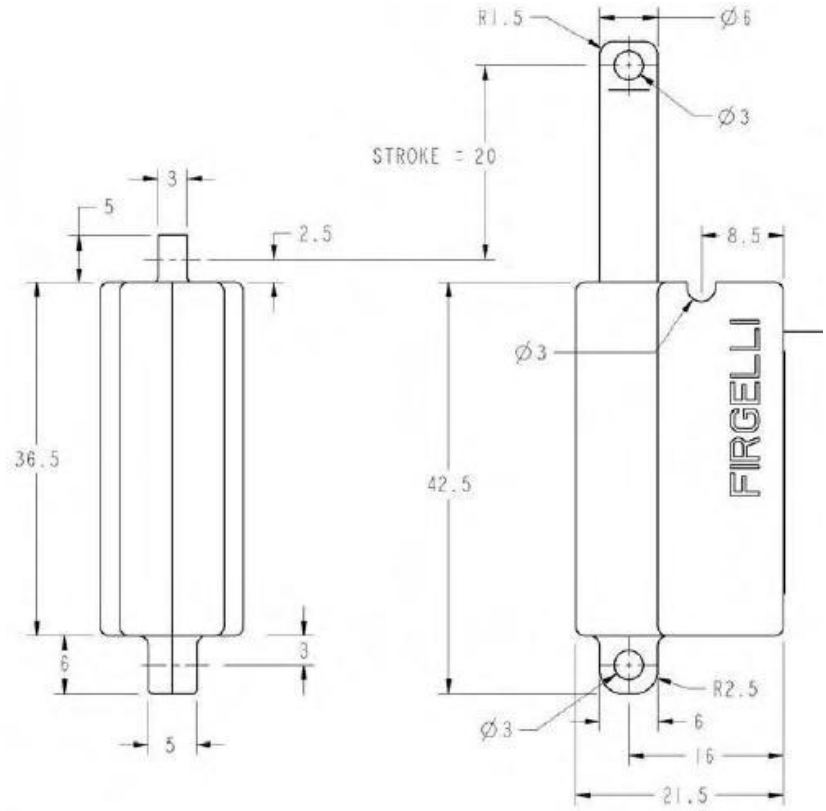


Micro Maestro 6-channel USB servo controller (fully assembled) labeled top view.

PQ12 –S Linear Actuator

Stroke: 20mm
Maximum Force: 9-35N
No-Load Speed: 25-9mm/s
Weight: 15 grams
Voltage: 6V or 12V
Stall Current: 550 mA @ 6V, 220 mA @ 12 V
Control Options: Limit Switches (-S)

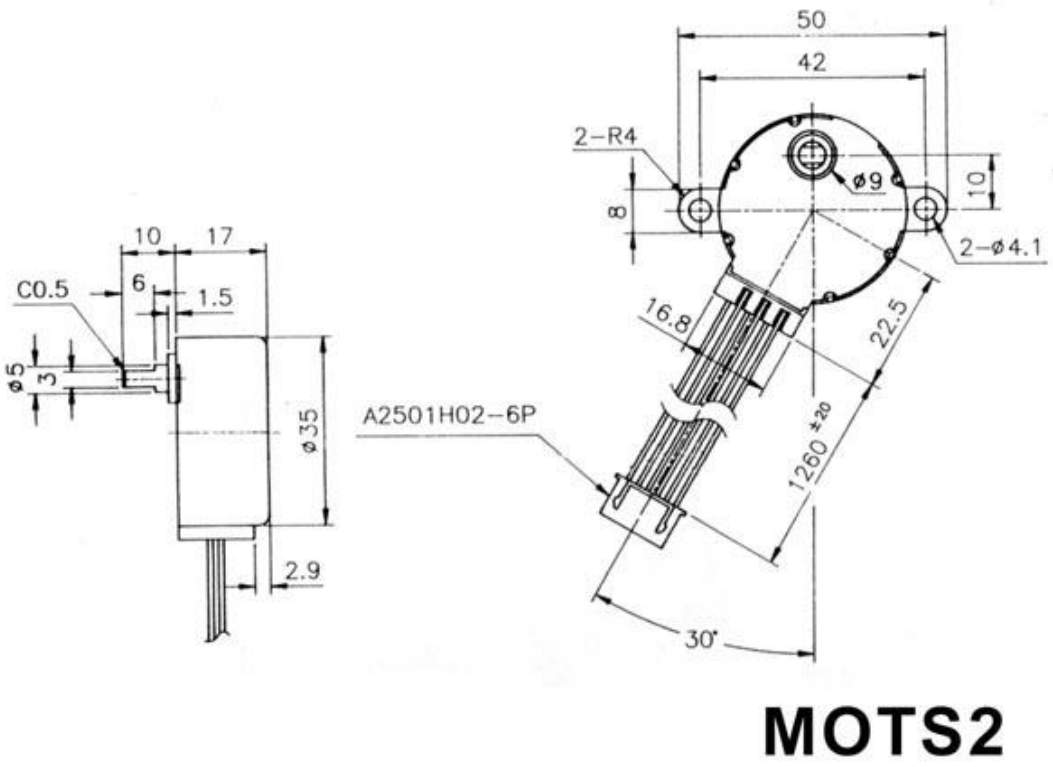
PQ12 Wiring (Pin Connections)	
Pin 1	Limit Detection (optional)
Pin 2	Actuator Power
Pin 3	Actuator Power
Pin 4	Not Connected
Pin 5	Not Connected



MOTS 2 Stepper Motor 12V 60mA (75° angle/45 steps)

resistance: 170 ohms
rated voltage: 12Vdc
current: 60mA
impedance: 200ohm
phase: 4
step angle / step: 7.5° / 48
reduction ratio: 1/85
detent torque: 500gfcm
pull-in torque: 650gfcm
max. starting pulse rate: 350pps
max. slewing pulse rate: 800pps
temperature range: 45°C
noise: 40dB
cable: 450mm AWG 1007#26
terminal: JST SXH-001T-P0.6
insulation strength: AC 600V - 1 sec cut-off current: 10mA
colours

1: B1 : pink
2: A1 : orange
3: A2 : yellow
4: B2 : blue
5: GND : red
6: GND2 (for MOTS2) : brown

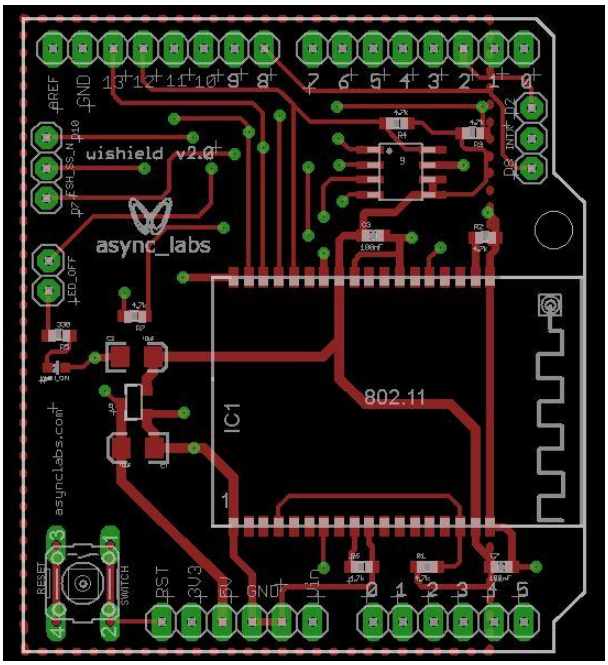


MOTS2

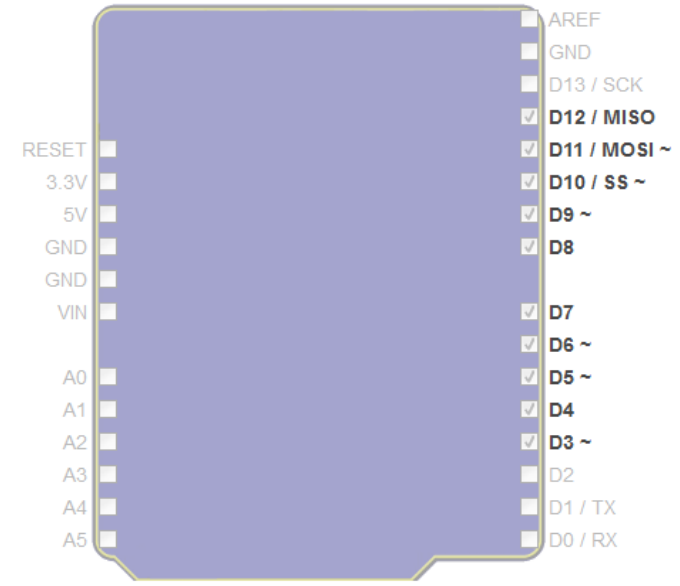
Arduino Microcontroller

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

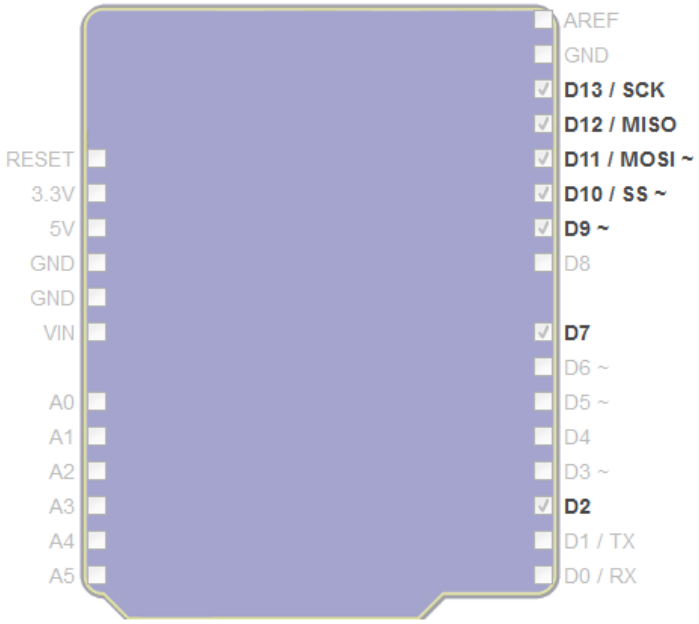
Arduino Specification Sheet



WiShield Pin Schematic



Motor Shield Pin Usage



WiShield Pin Usage

Appendix 2: Microcontroller Code

```
//SMARTMOUNT

#include <WiServer.h> // WiServer Library
#include <Servo.h> // Servo Control Library

#define WIRELESS_MODE_INFRA     1
#define WIRELESS_MODE_ADHOC     2

// Wireless configuration parameters ----------------------------------------
unsigned char local_ip[] = {192,168,1,120}; // IP address of WiShield
unsigned char gateway_ip[] = {192,168,1,1}; // router or gateway IP address
unsigned char subnet_mask[] = {255,255,255,0}; // subnet mask for the local network
const prog_char ssid[] PROGMEM = {"Arduino2"};  // max 32 bytes

unsigned char security_type = 3; // 0 - open; 1 - WEP; 2 - WPA; 3 - WPA2

// WPA/WPA2 passphrase
const prog_char security_passphrase[] PROGMEM = {"projectmack"}; // max 64 characters

// WEP 128-bit keys
prog_uchar wep_keys[] PROGMEM = {
  0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, // Key 0
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Key 1
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Key 2
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00  // Key 3
};

// setup the wireless mode
// infrastructure - connect to AP
// adhoc - connect to another WiFi device
unsigned char wireless_mode = WIRELESS_MODE_INFRA;

unsigned char ssid_len;
unsigned char security_passphrase_len;

// End of wireless configuration parameters ----------------------------------------

Servo myservo;  // Servo object to control a servo
Servo myservo2;

int posa = 0;    // Variable to store the servo position
int posb = 0;

// This is our page serving function that generates web pages
boolean sendMyPage(char* URL) {

  // Check if the requested URL matches "/"
  if (strcmp(URL, "/") == 0) {
    // Use WiServer's print and println functions to write out the page content
    WiServer.print("<html>");
    WiServer.print("Enter a position for the mount to move!");
    WiServer.print("</html>");

    // URL was recognized
    return true;
  }
 // Sample cases for position control of both servo motors
  else if (strcmp(URL, "/movea45") == 0 ) {
    WiServer.print("<html>");
    WiServer.print("Servo A has moved to 45 degrees!");
```

```
  WiServer.print("</html>");
  posa = 45;
  myservo.write(posa);
  delay(100);
  return true;
}
else if (strcmp(URL, "/movea90") == 0 ) {
  WiServer.print("<html>");
  WiServer.print("Servo A has moved to 90 degrees!");
  WiServer.print("</html>");
  posa = 90;
  myservo.write(posa);
  return true;
}
else if (strcmp(URL, "/movea135") == 0 ) {
  WiServer.print("<html>");
  WiServer.print("Servo A has moved to 135 degrees!");
  WiServer.print("</html>");
  posa = 135;
  myservo.write(posa);
  return true;
}
else if (strcmp(URL, "/movea180") == 0 ) {
  WiServer.print("<html>");
  WiServer.print("Servo A has moved to 180 degrees!");
  WiServer.print("</html>");
  posa = 180;
  myservo.write(posa);
  return true;
}
else if (strcmp(URL, "/moveb45") == 0 ) {
  WiServer.print("<html>");
  WiServer.print("Servo B has moved to 45 degrees!");
  WiServer.print("</html>");
  posb = 45;
  myservo2.write(posb);
  return true;
}
else if (strcmp(URL, "/moveb90") == 0 ) {
  WiServer.print("<html>");
  WiServer.print("Servo B has moved to 90 degrees!");
  WiServer.print("</html>");
  posb = 90;
  myservo2.write(posb);
  return true;
}
else if (strcmp(URL, "/moveb135") == 0 ) {
  WiServer.print("<html>");
  WiServer.print("Servo B has moved to 135 degrees!");
  WiServer.print("</html>");
  posb = 135;
  myservo2.write(posb);
  return true;
}
else if (strcmp(URL, "/moveb180") == 0 ) {
  WiServer.print("<html>");
  WiServer.print("Servo B has moved to 180 degrees!");
  WiServer.print("</html>");
  posb = 180;
  myservo2.write(posb);
```

```
  }
  // Initializes actuator button press
  else if (strcmp(URL, "/act") == 0 ) {
    WiServer.print("<html>");
    WiServer.print("Button Press!");
    WiServer.print("</html>");
    analogWrite(14, 150); // Extends actuator
    delay(2500);
    analogWrite(14, 0);
    analogWrite(15, 150); //Retracts actuator
    delay(500);
    analogWrite(15, 0);

    return true;
  }
  // URL not found
  return false;
}

// Actions that occur when Arduino is powered ON

void setup() {

  pinMode(14, OUTPUT);
  pinMode(15, OUTPUT);

  analogWrite(15, 255); // Calibration for actuator
  delay(2500);
  analogWrite(15, 0);



  myservo.attach(9);  // attaches Servo A on pin 9 to the servo object
  myservo2.attach(19); // attaches Servo B on pin 19 to the servo2 object

  // Initialize WiServer and have it use the sendMyPage function to serve pages
  WiServer.init(sendMyPage);

}

// Actions that occur continuously until Arduino is powered OFF
void loop(){

  // Run WiServer
  WiServer.server_task();

}
```

Appendix 3: GUI Code

"NNH765BF.txt"

```
NNH765BF
19,30,170,108,36
29,90,170,115,20
16,150,170,121,8
18,30,140,101,40
20,90,140,107,24
22,150,140,112,11
13,30,110,85,55
28,90,110,94,35
27,150,110,99,20
1,30,200,121,8
2,90,200,124,19
3,150,200,129,7
4,30,220,121,35
5,90,220,130,19
6,150,220,134,9
7,30,240,126,37
8,90,240,134,22
9,150,240,140,10
10,90,260,139,27
11,150,300,146,29
12,30,300,131,54
14,30,260,129,43
15,150,260,146,15
```

"convertback.m"

```
function [theta1,theta2]=convertback(findx,findy)

theta1=-1;
theta2=-1;

t = 0 : pi/360 : pi;
r1 = 97;
r2 = 91;
xc = 0;
yc = 0;

x = r1*cos(t) + xc;
y = r1*sin(t) + yc;

for i = 1:length(x)
    xc1 = x(i);
    yc1 = y(i);
    t1 = 3*pi/2+t(i) : pi/360 : 5*pi/2+t(i);
    x1 = r2*cos(t1) + xc1;
    y1 = r2*sin(t1) + yc1;
        for z=1:length(x1)
           x2 = x1(z);
           y2 = y1(z);
           if (abs(x2-findx)<.05 && abs(y2-findy)<.05)
                  theta1 = t(i)*180/pi;
                  theta2 = (t1(z)-(3*pi/2+t(i)))*180/pi;
           end
        end
end
```

“MountRange.m”

```
t = 0 : pi/360 : pi;
r1 = 97;
r2 = 91;
xc = 0;
yc = 0;

x = r1*cos(t) + xc;
y = r1*sin(t) + yc;

figure();
hold on

for i = 1:length(x)
    xc1 = x(i);
    yc1 = y(i);

    t1 = 3*pi/2+t(i) : pi/360 : 5*pi/2+t(i);

    x1 = r2*cos(t1) + xc1;
    y1 = r2*sin(t1) + yc1;

    plot(-y1,x1)
end
grid on
hold off
```

“Main.java”

```
package gui;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URLConnection;
import java.util.ArrayList;

public class Main {
   public static void robotCommand(String command) {
      try {
         URL url = new URL("http://192.168.1.120/" + command);
         URLConnection con = url.openConnection();
         BufferedReader in = new BufferedReader(new
InputStreamReader(con.getInputStream()));
         String data;
         while ((data = in.readLine())!=null)
            System.out.println(data);
         in.close();
      } catch (Exception ex) {
         System.out.println(ex.getMessage());
      }
   }

   public static void main(String[] args) {
      microwaveOven oven = new microwaveOven();
      demoOven oven1 = new demoOven();
      DeviceSelection device = new DeviceSelection();
      LoginScreen login = new LoginScreen();

      ArrayList<OvenConfigs> ovens = OvenConfigs.readAllConfigs(oven);

      oven.setscreens(device);
      device.setscreens(oven,oven1,ovens);
      login.setscreens(device);

      device.jComboBox1.removeAllItems();
      for(int i=0; i<ovens.size(); i++)
         device.jComboBox1.addItem(ovens.get(i).OvenName);
      login.setVisible(true);
   }

}
```

“microwaveOven.java”

```
package gui;

import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;

/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * ConvectionOven.java
 *
 * Created on Feb 24, 2011, 5:07:00 AM
 */

/**
 *
 * @author Michael
 */
public class microwaveOven extends javax.swing.JFrame {
    public int m[],n[];
    JFrame device;
    /** Creates new form ConvectionOven */
    public microwaveOven() {
        initComponents();
        this.setSize(280, 450);
    }
    public void setscreens(JFrame device){
        this.device = device;
    }
    public void setButton(int button_num, boolean enabled, int x, int y) {
        JComponent jComponent = jButton1;
        switch (button_num) {
            case 1:
                jComponent = jButton1;
                break;
            case 2:
                jComponent = jButton2;
                break;
            case 3:
                jComponent = jButton3;
                break;
```

```
        case 4:
           jComponent = jButton4;
           break;
        case 5:
           jComponent = jButton5;
           break;
        case 6:
           jComponent = jButton6;
           break;
        case 7:
           jComponent = jButton7;
           break;
        case 8:
           jComponent = jButton8;
           break;
        case 9:
           jComponent = jButton9;
           break;
        case 10:
           jComponent = jButton10;
           break;
        case 11:
           jComponent = Start;
           break;
        case 12:
           jComponent = reset;
           break;
        case 13:
           jComponent = popcorn;
           break;
        case 14:
           jComponent = timer;
           break;
        case 15:
           jComponent = clock;
           break;
        case 16:
           jComponent = more;
           break;
        case 17:
           jComponent = less;
           break;
        case 18:
           jComponent = powerLevel;
           break;
        case 19:
```

```
            jComponent = quickMin;
            break;
        case 20:
            jComponent = defrost;
            break;
        case 21:
            jComponent = autoReheat;
            break;
        case 22:
            jComponent = keepWarm;
            break;
        case 23:
            jComponent = servingWeight;
            break;
        case 24:
            jComponent = autoCook13;
            break;
        case 25:
            jComponent = autoCook46;
            break;
        case 26:
            jComponent = autoCook79;
            break;
        case 27:
            jComponent = sensorCook;
            break;
        case 28:
            jComponent = sensorReheat;
            break;
        case 29:
            jComponent = function;
            break;
    }
    jComponent.setVisible(enabled);
    jComponent.setLocation(x, y);
}

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {
```

```
        jButton3 = new javax.swing.JButton();
        jButton4 = new javax.swing.JButton();
        jButton5 = new javax.swing.JButton();
        jButton6 = new javax.swing.JButton();
        jButton7 = new javax.swing.JButton();
        jButton8 = new javax.swing.JButton();
        jButton9 = new javax.swing.JButton();
        jButton10 = new javax.swing.JButton();
        jButton2 = new javax.swing.JButton();
        jButton1 = new javax.swing.JButton();
        Start = new javax.swing.JButton();
        jButton12 = new javax.swing.JButton();
        display = new javax.swing.JTextField();
        reset = new javax.swing.JButton();
        more = new javax.swing.JButton();
        popcorn = new javax.swing.JButton();
        autoReheat = new javax.swing.JButton();
        defrost = new javax.swing.JButton();
        keepWarm = new javax.swing.JButton();
        quickMin = new javax.swing.JButton();
        servingWeight = new javax.swing.JButton();
        autoCook79 = new javax.swing.JButton();
        autoCook13 = new javax.swing.JButton();
        autoCook46 = new javax.swing.JButton();
        timer = new javax.swing.JButton();
        powerLevel = new javax.swing.JButton();
        less = new javax.swing.JButton();
        clock = new javax.swing.JButton();
        function = new javax.swing.JButton();
        sensorReheat = new javax.swing.JButton();
        sensorCook = new javax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        getContentPane().setLayout(null);

        jButton3.setText("3");
        jButton3.setMargin(new java.awt.Insets(0, 0, 0, 0));
        jButton3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton3ActionPerformed(evt);
            }
        });
        getContentPane().add(jButton3);
        jButton3.setBounds(170, 80, 60, 20);

        jButton4.setText("4");
```

```
jButton4.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton4.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton4ActionPerformed(evt);
   }
});
getContentPane().add(jButton4);
jButton4.setBounds(30, 110, 60, 20);

jButton5.setText("5");
jButton5.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton5.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton5ActionPerformed(evt);
   }
});
getContentPane().add(jButton5);
jButton5.setBounds(100, 110, 60, 20);

jButton6.setText("6");
jButton6.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton6.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton6ActionPerformed(evt);
   }
});
getContentPane().add(jButton6);
jButton6.setBounds(170, 110, 60, 20);

jButton7.setText("7");
jButton7.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton7.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton7ActionPerformed(evt);
   }
});
getContentPane().add(jButton7);
jButton7.setBounds(30, 140, 60, 20);

jButton8.setText("8");
jButton8.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton8.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton8ActionPerformed(evt);
   }
});
```

```
getContentPane().add(jButton8);
jButton8.setBounds(100, 140, 60, 20);

jButton9.setText("9");
jButton9.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton9.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton9ActionPerformed(evt);
   }
});
getContentPane().add(jButton9);
jButton9.setBounds(170, 140, 60, 20);

jButton10.setText("0");
jButton10.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton10.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton10ActionPerformed(evt);
   }
});
getContentPane().add(jButton10);
jButton10.setBounds(100, 170, 60, 20);

jButton2.setText("2");
jButton2.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton2.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton2ActionPerformed(evt);
   }
});
getContentPane().add(jButton2);
jButton2.setBounds(100, 80, 60, 20);

jButton1.setText("1");
jButton1.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton1.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton1ActionPerformed(evt);
   }
});
getContentPane().add(jButton1);
jButton1.setBounds(29, 80, 60, 20);

Start.setText("Start");
Start.setMargin(new java.awt.Insets(0, 0, 0, 0));
Start.addActionListener(new java.awt.event.ActionListener() {
```

```
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        StartActionPerformed(evt);
    }
});
getContentPane().add(Start);
Start.setBounds(170, 170, 60, 20);

jButton12.setText("Previous Screen");
jButton12.setMargin(new java.awt.Insets(0, 0, 0, 0));
jButton12.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        jButton12ActionPerformed(evt);
    }
});
getContentPane().add(jButton12);
jButton12.setBounds(30, 360, 140, 20);

display.setEditable(false);
display.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        displayActionPerformed(evt);
    }
});
getContentPane().add(display);
display.setBounds(30, 50, 200, 20);

reset.setText("Reset");
reset.setMargin(new java.awt.Insets(0, 0, 0, 0));
reset.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        resetActionPerformed(evt);
    }
});
getContentPane().add(reset);
reset.setBounds(30, 170, 60, 20);

more.setText("More");
more.setMargin(new java.awt.Insets(0, 0, 0, 0));
more.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        moreActionPerformed(evt);
    }
});
getContentPane().add(more);
more.setBounds(100, 310, 60, 20);
```

```
popcorn.setText("Popcorn");
popcorn.setMargin(new java.awt.Insets(0, 0, 0, 0));
popcorn.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      popcornActionPerformed(evt);
   }
});
getContentPane().add(popcorn);
popcorn.setBounds(30, 240, 60, 20);

autoReheat.setText("Auto Reheat");
autoReheat.setMargin(new java.awt.Insets(0, 0, 0, 0));
autoReheat.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      autoReheatActionPerformed(evt);
   }
});
getContentPane().add(autoReheat);
autoReheat.setBounds(100, 210, 60, 20);

defrost.setText("Inverter Turbo Defrost");
defrost.setMargin(new java.awt.Insets(0, 0, 0, 0));
defrost.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      defrostActionPerformed(evt);
   }
});
getContentPane().add(defrost);
defrost.setBounds(170, 210, 60, 20);

keepWarm.setText("Keep Warm");
keepWarm.setMargin(new java.awt.Insets(0, 0, 0, 0));
keepWarm.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      keepWarmActionPerformed(evt);
   }
});
getContentPane().add(keepWarm);
keepWarm.setBounds(30, 290, 60, 20);

quickMin.setText("Quick Min");
quickMin.setMargin(new java.awt.Insets(0, 0, 0, 0));
quickMin.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      quickMinActionPerformed(evt);
   }
```

```
        });
        getContentPane().add(quickMin);
        quickMin.setBounds(100, 290, 60, 20);

        servingWeight.setText("Serving/Weight");
        servingWeight.setMargin(new java.awt.Insets(0, 0, 0, 0));
        servingWeight.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                servingWeightActionPerformed(evt);
            }
        });
        getContentPane().add(servingWeight);
        servingWeight.setBounds(180, 290, 60, 20);

        autoCook79.setText("Auto Cook (7-9)");
        autoCook79.setMargin(new java.awt.Insets(0, 0, 0, 0));
        autoCook79.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                autoCook79ActionPerformed(evt);
            }
        });
        getContentPane().add(autoCook79);
        autoCook79.setBounds(180, 270, 60, 20);

        autoCook13.setText("Auto Cook (1-3)");
        autoCook13.setMargin(new java.awt.Insets(0, 0, 0, 0));
        autoCook13.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                autoCook13ActionPerformed(evt);
            }
        });
        getContentPane().add(autoCook13);
        autoCook13.setBounds(30, 270, 60, 20);

        autoCook46.setText("Auto Cook (4-6)");
        autoCook46.setMargin(new java.awt.Insets(0, 0, 0, 0));
        autoCook46.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                autoCook46ActionPerformed(evt);
            }
        });
        getContentPane().add(autoCook46);
        autoCook46.setBounds(100, 270, 60, 20);

        timer.setText("Timer");
        timer.setMargin(new java.awt.Insets(0, 0, 0, 0));
```

```
        timer.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                timerActionPerformed(evt);
            }
        });
        getContentPane().add(timer);
        timer.setBounds(180, 230, 60, 20);

        powerLevel.setText("Power Level (10 Levels)");
        powerLevel.setMargin(new java.awt.Insets(0, 0, 0, 0));
        powerLevel.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                powerLevelActionPerformed(evt);
            }
        });
        getContentPane().add(powerLevel);
        powerLevel.setBounds(30, 310, 60, 20);

        less.setText("Less");
        less.setMargin(new java.awt.Insets(0, 0, 0, 0));
        less.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                lessActionPerformed(evt);
            }
        });
        getContentPane().add(less);
        less.setBounds(180, 310, 60, 20);

        clock.setText("Clock");
        clock.setMargin(new java.awt.Insets(0, 0, 0, 0));
        clock.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                clockActionPerformed(evt);
            }
        });
        getContentPane().add(clock);
        clock.setBounds(180, 250, 60, 20);

        function.setText("Function");
        function.setMargin(new java.awt.Insets(0, 0, 0, 0));
        function.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                functionActionPerformed(evt);
            }
        });
        getContentPane().add(function);
```

```
    function.setBounds(30, 210, 60, 20);

    sensorReheat.setText("Sens Reheat");
    sensorReheat.setMargin(new java.awt.Insets(0, 0, 0, 0));
    sensorReheat.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            sensorReheatActionPerformed(evt);
        }
    });
    getContentPane().add(sensorReheat);
    sensorReheat.setBounds(30, 340, 60, 20);

    sensorCook.setText("Sens Cook");
    sensorCook.setMargin(new java.awt.Insets(0, 0, 0, 0));
    sensorCook.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent evt) {
            sensorCookActionPerformed(evt);
        }
    });
    getContentPane().add(sensorCook);
    sensorCook.setBounds(100, 340, 60, 20);

    pack();
}// </editor-fold>

private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    display.setText(display.getText()+jButton5.getText());
    Main.robotCommand("movea" + m[4]);
    Main.robotCommand("moveb" + n[4]);
    Main.robotCommand("act");

}

private void jButton12ActionPerformed(java.awt.event.ActionEvent evt) {
    device.setVisible(true);
    this.setVisible(false);
}

private void displayActionPerformed(java.awt.event.ActionEvent evt) {
}

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton1.getText());
    Main.robotCommand("movea" + m[0]);
    Main.robotCommand("moveb" + n[0]);
```

```
    Main.robotCommand("act");
}

private void StartActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[10]);
    Main.robotCommand("moveb" + n[10]);
    Main.robotCommand("act");
}

private void resetActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[11]);
    Main.robotCommand("moveb" + n[11]);
    Main.robotCommand("act");
    display.setText("");
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton2.getText());
    Main.robotCommand("movea" + m[1]);
    Main.robotCommand("moveb" + n[1]);
    Main.robotCommand("act");
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton3.getText());
    Main.robotCommand("movea" + m[2]);
    Main.robotCommand("moveb" + n[2]);
    Main.robotCommand("act");
}

private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton4.getText());
    Main.robotCommand("movea" + m[3]);
    Main.robotCommand("moveb" + n[3]);
    Main.robotCommand("act");
}

private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton6.getText());
    Main.robotCommand("movea" + m[5]);
    Main.robotCommand("moveb" + n[5]);
    Main.robotCommand("act");
}

private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton7.getText());
```

```
    Main.robotCommand("movea" + m[6]);
    Main.robotCommand("moveb" + n[6]);
    Main.robotCommand("act");
}

private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton8.getText());
    Main.robotCommand("movea" + m[7]);
    Main.robotCommand("moveb" + n[7]);
    Main.robotCommand("act");
}

private void jButton9ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton9.getText());
    Main.robotCommand("movea" + m[8]);
    Main.robotCommand("moveb" + n[8]);
    Main.robotCommand("act");
}

private void jButton10ActionPerformed(java.awt.event.ActionEvent evt) {
    display.setText(display.getText()+jButton10.getText());
    Main.robotCommand("movea" + m[9]);
    Main.robotCommand("moveb" + n[9]);
    Main.robotCommand("act");
}

private void moreActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[15]);
    Main.robotCommand("moveb" + n[15]);
    Main.robotCommand("act");
}

private void popcornActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[12]);
    Main.robotCommand("moveb" + n[12]);
    Main.robotCommand("act");
}

private void autoReheatActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[20]);
    Main.robotCommand("moveb" + n[20]);
    Main.robotCommand("act");
}

private void defrostActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[19]);
```

```
    Main.robotCommand("moveb" + n[19]);
    Main.robotCommand("act");
}

private void keepWarmActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[21]);
    Main.robotCommand("moveb" + n[21]);
    Main.robotCommand("act");
}

private void quickMinActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[18]);
    Main.robotCommand("moveb" + n[18]);
    Main.robotCommand("act");
}

private void servingWeightActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[22]);
    Main.robotCommand("moveb" + n[22]);
    Main.robotCommand("act");
}

private void autoCook79ActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[25]);
    Main.robotCommand("moveb" + n[25]);
    Main.robotCommand("act");
}

private void autoCook13ActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[23]);
    Main.robotCommand("moveb" + n[23]);
    Main.robotCommand("act");
}

private void autoCook46ActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[24]);
    Main.robotCommand("moveb" + n[24]);
    Main.robotCommand("act");
}

private void timerActionPerformed(java.awt.event.ActionEvent evt) {
    Main.robotCommand("movea" + m[13]);
    Main.robotCommand("moveb" + n[13]);
    Main.robotCommand("act");
}
```

```
    private void powerLevelActionPerformed(java.awt.event.ActionEvent evt) {
        Main.robotCommand("movea" + m[18]);
        Main.robotCommand("moveb" + n[18]);
        Main.robotCommand("act");
    }

    private void lessActionPerformed(java.awt.event.ActionEvent evt) {
        Main.robotCommand("movea" + m[16]);
        Main.robotCommand("moveb" + n[16]);
        Main.robotCommand("act");
    }

    private void clockActionPerformed(java.awt.event.ActionEvent evt) {
        Main.robotCommand("movea" + m[14]);
        Main.robotCommand("moveb" + n[14]);
        Main.robotCommand("act");
    }

    private void functionActionPerformed(java.awt.event.ActionEvent evt) {
        Main.robotCommand("movea" + m[28]);
        Main.robotCommand("moveb" + n[28]);
        Main.robotCommand("act");
    }

    private void sensorReheatActionPerformed(java.awt.event.ActionEvent evt) {
        Main.robotCommand("movea" + m[27]);
        Main.robotCommand("moveb" + n[27]);
        Main.robotCommand("act");
    }

    private void sensorCookActionPerformed(java.awt.event.ActionEvent evt) {
        Main.robotCommand("movea" + m[26]);
        Main.robotCommand("moveb" + n[26]);
        Main.robotCommand("act");
    }
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new microwaveOven().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton Start;
    private javax.swing.JButton autoCook13;
    private javax.swing.JButton autoCook46;
```

```
    private javax.swing.JButton autoCook79;
    private javax.swing.JButton autoReheat;
    private javax.swing.JButton clock;
    private javax.swing.JButton defrost;
    private javax.swing.JTextField display;
    private javax.swing.JButton function;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton10;
    private javax.swing.JButton jButton12;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JButton jButton4;
    private javax.swing.JButton jButton5;
    private javax.swing.JButton jButton6;
    private javax.swing.JButton jButton7;
    private javax.swing.JButton jButton8;
    private javax.swing.JButton jButton9;
    private javax.swing.JButton keepWarm;
    private javax.swing.JButton less;
    private javax.swing.JButton more;
    private javax.swing.JButton popcorn;
    private javax.swing.JButton powerLevel;
    private javax.swing.JButton quickMin;
    private javax.swing.JButton reset;
    private javax.swing.JButton sensorCook;
    private javax.swing.JButton sensorReheat;
    private javax.swing.JButton servingWeight;
    private javax.swing.JButton timer;
    // End of variables declaration

}
```

"OvenConfigs.java"

```
package gui;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;
import javax.swing.JFrame;

public class OvenConfigs {
   microwaveOven window;
   String OvenName;
   public class Component {
      int x, y, m, n;
      boolean enabled;
   }
   Component list[];
   OvenConfigs(microwaveOven window) {
      this.window = window;
      list = new Component[30];
      for (int i=0; i<list.length; i++) {
         list[i] = new Component();
         list[i].enabled = false;
      }
   }

   /*
    * ovenname2
    * 2,button x,button y,degA,degB
    * 3,button x,button y,degA,degB
    */
   void readConfig(String filename) {
      try {
         BufferedReader file = new BufferedReader(new FileReader(filename));
         OvenName = file.readLine();
         String button_info;
         while ((button_info = file.readLine()) != null) {
            System.out.println(button_info);
            String button_parts[] = button_info.split(",");
            int button_number = Integer.parseInt(button_parts[0]);
            int button_x = Integer.parseInt(button_parts[1]);
            int button_y = Integer.parseInt(button_parts[2]);
            int angleA = Integer.parseInt(button_parts[3]);
            int angleB = Integer.parseInt(button_parts[4]);
            list[button_number].x = button_x;
```

```
            list[button_number].y = button_y;
            list[button_number].m = angleA;
            list[button_number].n = angleB;
            list[button_number].enabled = true;
        }
    } catch (Exception e) {e.printStackTrace();}
}

void setButtons() {
    int m[] = new int[list.length];
        int n[] = new int[list.length];
    for (int i=0; i<list.length; i++){
        window.setButton(i, list[i].enabled, list[i].x, list[i].y);
    }
            for (int i=0; i<list.length; i++){
                m[i] = list[i].m;
                n[i] = list[i].n;
    }
        window.m = m;
        window.n = n;
    }
static ArrayList<OvenConfigs> readAllConfigs(microwaveOven window) {
    ArrayList<OvenConfigs> configs = new ArrayList<OvenConfigs>();
    File dir = new File("configs");
    String[] children = dir.list();
    if (children != null) {
        for (int i=0; i<children.length; i++) {
            String filename = children[i];
            System.out.println(filename);
            OvenConfigs o = new OvenConfigs(window);
            o.readConfig("configs/"+filename);
            configs.add(o);
        }
    }
    return configs;
}
}
```