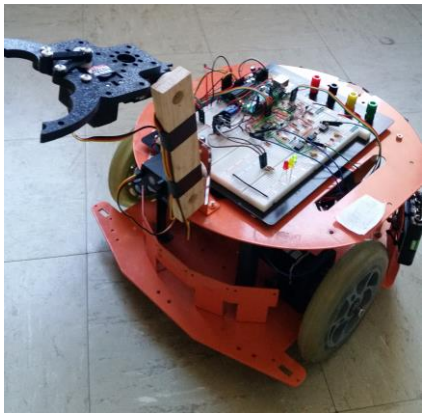

INNOVA-CLAW

Experiment 6: Final Project

ABSTRACT

For our final project, we decided to innovate on the claw project given in the example final projects. In addition to having a gripper that can open and close on objects, we wanted to raise and lower the gripper itself, and also control our robot, all through an interface utilizing an old-school video game controller.

Doan Bui, Huan Nguyen, & Josh Green

ECEN2270 Electronic Design lab

Introduction

For our final project, we had different ideas that we eventually wanted to incorporate all of.

First, we wanted to make something we knew would push us to learn more, but not so difficult that we would have nothing to present when it came time to demo. Second, we wanted to use an old-school video game controller to interface with the robot and control its movement, at the least.

The claw example project given seemed perfect to us; we did not know how to use servos, and with Arduino, though we knew that there were ways to connect the two made specifically for that purpose (libraries, we later found out). But we wanted to do *more*.

We incorporated our earlier ideas into the mix, and eventually settled upon wanting to control the robot's movement, as well as adding controllable movement to the gripper beyond merely opening and closing the arms. Our final idea was to toggle between controlling the movement of the robot or of the claw with a button, and using directional buttons (on the video game controller) to do so. Control of the gripper would include mounting it on an arm we could move in the vertical plane, and manual open/close functionality of the claw itself.

Having decided on our project, we then ordered our components.

Components

Our most essential add-on component is the old-school video game controller used to interface between the user and the robot/gripper combo: the Nintendo Entertainment System controller (NES controller).

It has four directional buttons and four push buttons; we used the directional, A, and B buttons to control our robot and gripper combination. The directional buttons were, intuitively, to control the robot's movement. The B button was to toggle between controlling the robot and the gripper setup, and the A button to open and close the gripper.

Figure 1: the NES controller; image credit here

To provide the physical force to move the gripper and the arm that we mounted it on, we used servos from Sparkfun. We used the Hitec HS-425BB in the standard size (found here, where the image also comes from):

Figure 2: the servo we used: Hitec HS-425BB

The servos have 180 degree movement capability (in which range the user can direct the servo to move to a certain angle) and require between 4.8 and 6 volts to operate. The max current draw we discovered experimentally came from the servo controlling the gripper (700mA), when it was trying to reach the position we programmed it to, but found an obstacle in the way (gripping an object, for instance).

The gripper that we used in conjunction with the servos was the Standard Gripper Kit A from SparkFun (found here, where the image is also found):

Figure 3: the Standard Gripper Kit A

Operation of the gripper is straightforward: a servo mounted into the gripper connects to two plastic rods that connect to the gripper's arms. Movement of the servo is translated mechanically into force that opens and closes the arms.

We found, experimentally, that the servo/gripper combination draws a max of about 700mA of current, when the servo is straining to reach the programmed angle but is stopped (in this case, by an object being placed between the gripper arms to prevent their full closure).

To power the servos that do this, we used a voltage regulator (from SparkFun here).

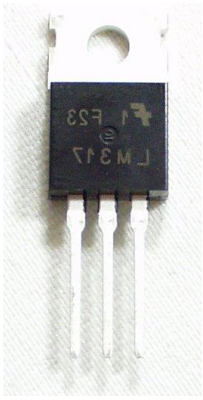

Figure 4: adjustable voltage regulator

From left to right: pin 1 is Adjust, pin 2 is V_{out}, and pin 3 is V_{in}.

It was incredibly easy to use. To get the voltage out, the equation given on the data sheet suffices:

$$V_{out} = 1.25\ Volts*(1+R_2/R_1)$$

We wanted to use ~5.8 volts for V_{out}, because of the error associated with our components (5% with resistors, and 5% for the regulator). We plugged that in, using 220Ω for R_1 and 780Ω for R2 and, when measured with an oscilloscope, got exactly 6V out. This circuit was added to our robot's breadboard, powered from the rails on it, and the servos were connected to the output of the regulator.

Project Description

Our final intended design comprises of three main "blocks": first, we had to figure out how to communicate with the Arduino using the NES controller as an interface; next, we had to extend that capability to controlling the robot's movement using the NES controller interface; lastly, we had to incorporate the control of the arm/gripper setup using the NES controller interface.

Interface Using NES Controller

Starting off, we had little idea of how the NES controller communicated with the gaming system by the Nintendo company, let alone of how to communicate with it using the Arduino microcontroller.

We found out, with online research (mostly here), that the NES controller has five wires (of its seven wires total) of interest to us: one +5V power wire (white), one ground wire (brown), and the clock (red), latch (orange), and data (yellow) wires.

To communicate with the NES controller (assuming it is powered and grounded), the latch pin must be set high, then low; next, eight pulses must be sent through the clock wire. For each pulse, the data pin will output logic low if the corresponding button is pressed (the order is: A B Start Select Up Down Left Right).

From this webpage, we found a function controllerRead()—included in our code in the appendix—that makes use of this configuration to read the data from the controller. It sets the latch high, then low, sending what can be thought of as a 'start' signal to the controller. Then, the function runs a 'for' loop eight times, by basically advancing the clock 'count' and checking to see if the corresponding button is depressed. If it is the 0 bit representing logic low gets written into the byte-size variable that we store the controller data in. For example, using the order of buttons given above: a 01111111 means that the A button is pressed and no others.

Using that input, we can then run 'if' or 'while' loops based on checking against the specific buttons being pressed—one pseudo-code example: if (controller data == 01111111) {open/close claw}.

To make sure this was working, we output the controller data variable via serial, and checked to see if the correct buttons were showing logic low; it worked, and a screen capture below shows that (along with some extraneous data we were just checking for our own curiosity; the pertinent lines are the first two):

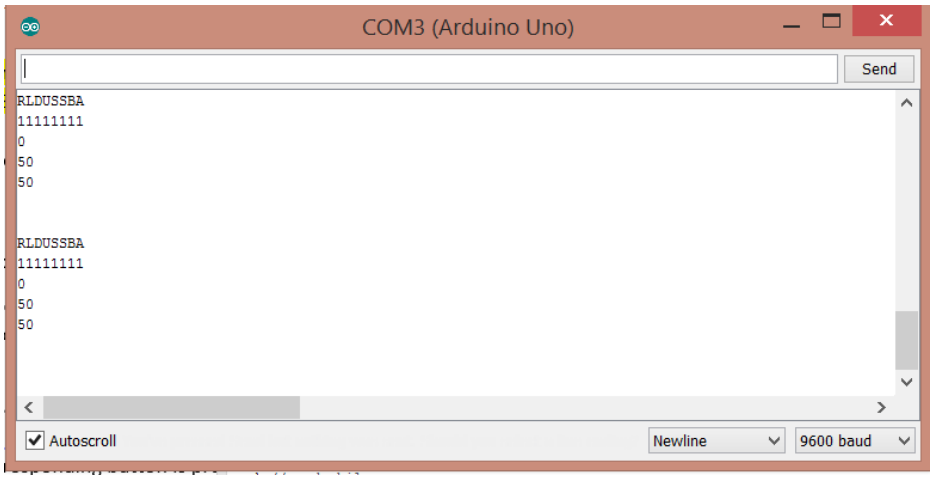

Figure 4: serial output of our testing the NES controller

Challenges

Connecting the NES controller to the robot was a bit of a challenge at first; we did not know what the wires inside the cable would look like, and we did not want to make irreparable changes to one of our key components. Eventually, we found out online that there are five distinct wires in the cable, and we cut it open to find the five color-coded wires, which we soldered to male to male jumper cables we ordered, and that made it almost incredibly easy to connect to the Arduino (and disconnect, for storage purposes).

User-controlled Movement—Robot

We defined the commands as human-readable names (example: UP = 11101111) and then used 'if' statements to check for each case; for each of the four directions, we called the four functions we wrote in lab 5, the directional movement functions. These functions rely on the rest of the circuits we built in the previous four labs, like the speed sensor and motor drivers. The functions (Forward(), Left(), Right(), etc) are found in the code included in our appendix.

Later on, once we added more features, we also added a green LED to our circuit; if we were in 'drive' mode, where we control the robot's movement, the green LED would light up.

Challenges

The challenge that reared its head was one that forced us to spend some time researching online, but finally found the solution to in the documentation for the Arduino language: the library Servo.h, that allows us to easily interface with the servos, disables the PWM functionality of digital pins 9 and 10. Those were, unfortunately, our analogWrite pins that set the speed reference voltage, and so our wheels refused to work, no matter what value we set for the reference.

By printing the controller data we read to the computer via serial, we knew that we were reading our commands sent through the controller just fine; we double-checked our code and were confident that it was working.

Finally, after reading about the fact that Servo.h disables some functionality of pins 9 and 10, we moved the pins around so that we were using those for digital output only, and everything worked the moment we uploaded our code and powered our robot and circuit.

User-controlled Movement—Gripper & Arm

Moving the gripper and arm was not the hard part of our idea; making the movement look as smooth as possible was. We wanted the servo that angles the gripper in the vertical plane to move in conjunction with the gripper that angles the arm in the vertical plane as well, so that the overall movement from a raised gripper to a lowered gripper (angled to avoid scraping the ground or stick too far into the air) is smooth and pleasing to watch.

To overcome this obstacle, we wrote program to move both the servo controlling the angle of the claw and the servo controlling the angle of the arm to move only a few degrees each, and put that sequence of commands into a 'for' loop. Technically, the servos still move separately, but they only move a few degrees each time the loop runs, and that fact in conjunction with the fact that the commands are executed quickly by the Arduino combine to make the movement look smooth to the human eye. This is in the 'automatic' raise and lower mode, where the user presses UP or DOWN once, and the claw raises to its highest or lowest programmed position, respectively.

In the 'manual' raise and lower mode (toggled between by pressing START), the user can press UP or DOWN to manually raise or lower the arm; the movement here is jerky, but is a mechanical problem in this case. The weight of the arm/claw combo is a bit much for the servo we chose; with a stronger servo (that we would use next time—most likely a high-torque servo), the movement would look smooth.

To help users distinguish between the two modes, we added yellow and red LEDs: the yellow LED—basically the inverted green LED—denotes when the user is in 'claw' mode, where they control the arm and claw. The red LED lights up when the claw mode is selected, and the manual raise and lower mode is also selected.

The three LEDs are placed next to one another on the board, making almost a stoplight-lookalike that we feel adds to the features of the robot as well, and makes it more fun to use.

Challenges

We constructed the arm and attached the servos and made a great-looking arm; when we went to hook it up to the Arduino, however, we hit a wall: we found that we had already used most of the available digital pins for the rest of our robot's circuits, like the speed sensors.

This problem kept us at a standstill for a while, because we kept trying to see if we could move around the wires and inputs and outputs to make room for the three signal lines going to the servos, but the inevitable truth remained: we had less pins available than we needed.

The solution was simple, once we turned to the internet: with a little research and Googling, we found that the Analog In pins on the Arduino are capable of performing as digital input and output pins.

We moved three lines of the controller (clock, latch, and data), and the three data lines to the three servos, over to the six available analog pins on the Arduino, and it worked perfectly.

In retrospect, it was great for our project that the NES controller communicates mostly serially, rather than in parallel. That meant we had some available pins, which were taken up by the servo data lines.

Potential Improvements & Changes

After we constructed the arm and got all the servos operating correctly we realized we did not account for the torque in the arm since the servo did not have enough power to raise the arm and claw back up. The servo we used (HS_A42BB) has a torque capacity of 45.82/56.93 oz-in from 4.8/6.0 V.

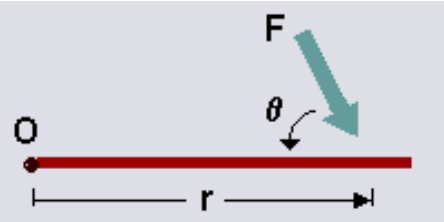

Figure 5: the physics behind our claw/arm combo

Each servo has a mass of 45.5 and the claw has a mass of 56.9 g. So overall the mass at the end of the arm is $2(45.5) + 56.9 = 147.9\ g$. We used a mass of 180 g in our calculations to account for the mass of any small object the claw can pick up. We also used an angle of 90 degrees to maximize the torque.

$$\tau = 0.180kg * \left(\frac{9.8m}{s^2}\right) * 0.3048 * \sin(90) = \ 0.5377\ Nm = 76.1447\ oz - in$$

76.1447 oz-in is more torque than the servo can handle. So in order to reduce this we calculated the torque for half the length of the original arm.

$$\tau = 0.180kg * \left(\frac{9.8m}{s^2}\right) * 0.1524 * \sin(90) = \ 0.2688\ Nm = 38.0653\ oz - in$$

This value of 38.0653 is well under the capacity that the servo can handle so not only will the servo be able to raise the arm back up but we will also be able to pick up objects with a heavier mass than anticipated.

In the future, to mitigate this problem, we could choose from a number of potential solutions. First, we could shorten the arm to shorten the moment arm, and therefore put less total torque on the servo that has to hold it up. Second, we could make the arm out of lighter materials, and also lessen the torque that way, though we would also have to make sure the material was still strong enough to withstand the weight placed on it. Third, we could create some sort of pulley system so that the servos could all be mounted on the robot but still have control over the arm and claw—that would have the biggest effect on weight, as having two servos on the end of the arm added nearly 100g of mass to the end of the moment arm. Lastly, we could implement a combination of two or more of these proposed solutions.

Another potential improvement we would want would be the ability to control both the claw and the robot's movement simultaneously. The NES controller provided for a limited number of inputs (and the number of pins available on the Arduino board was also a limiting factor), and therefore we had to be economical and control the robot by toggling between movement and claw modes.

To make this feature feasible, we predict that using a more recent game controller would increase the number of possible inputs—perhaps the next generation of NES controller, the Super Nintendo Entertainment System controller (SNES) would suffice. We also would use another Arduino board to communicate with the one controlling the robot, and that would guarantee an increase in the number of pins available; we could also accomplish the same via using a larger Arduino board (at the expected increase in cost).

Conclusion

Our robot was, by our stated goals, a smashing success!

Not only we were able to control the robot's movement and the angle of the claw as well as its opening and closing, we were able to add extended features to our project: manual control of the angle of the claw, *and* manual control of the angle of the arm, as well as the ability to switch between automatic control of and manual control of the arm's angle on-the-fly, without having to modify code.

Driving the robot around is a pleasure, and other users who tried it had lots of fun as well.

This entire semester has been very educational, somewhat stressful (in an educational way), and ended with a great time making our final project—we consider it a great learning experience.

Thank you.

Appendix

Arduino Code

```
/*****************************************
 ** ECEN2270 Electronic Design Lab **********
 ** Doan Bui, Huan Nguyen, Josh Green ********
 ** Final Project: NES controller + gripper **
 *****************************************/
```

```
#include <stdarg.h>
#include <Servo.h>

// define pins


// NES pins, defines
const int latch = A4; // orange
const int clk = A3; // red
const int datin = A5; // yellow
byte controller_data = 0;
const byte UP = B11101111;
const byte DOWN = B11011111;
const byte LEFT = B10111111;
const byte RIGHT = B01111111;
const byte SELECT = B11111011;
const byte START = B11110111;
const byte A = B11111110;
const byte B = B11111101;

// Servos & servo pins
const int s0pin = A0;
const int s1pin = A1;
const int s2pin = A2;
Servo servo0;
Servo servo1;
Servo servo2;

// motor pins
const byte pinON = 4;         // connect pin 6 to ON/OFF switch, active
HIGH
const byte pinCW_Left = 7;    // connect pin 7 to clk-wise PMOS gate
const byte pinCC_Left = 8;    // connect pin 8 to counter-clk-wise PMOS
gate
const byte pinSpeed_Left = 5; // connect pin 9 to speed reference
const byte pinCW_Right = 11;
const byte pinCC_Right = 12;
const byte pinSpeed_Right = 6;
const byte vRefLeft = 250;
const byte vRefRight = 250;

// other defines
byte toggle = 0;
byte gripper = 0;
byte adjust = 0;
int count1 = 0;
int count2 = 0;
int start1 = 0;
int start2 = 0;
int vert = 0;
const byte LEDpin = 10;

// setup pins and initial values
```

```
void setup() {
  // Setup motors
  pinMode(pinON, INPUT);
  pinMode(pinCW_Left, OUTPUT);
  pinMode(pinCC_Left, OUTPUT);
  pinMode(pinSpeed_Left, OUTPUT);
  pinMode(pinCW_Right, OUTPUT);
  pinMode(pinCC_Right, OUTPUT);
  pinMode(pinSpeed_Right, OUTPUT);
  pinMode(13, OUTPUT);            // on-board LED
  digitalWrite(13, LOW);          // turn LED off
  digitalWrite(pinCW_Left, LOW);  // stop clkwise
  digitalWrite(pinCC_Left, LOW);  // stop counter-clkwise
  digitalWrite(pinCW_Right, LOW);
  digitalWrite(pinCC_Right, LOW);
  analogWrite(pinSpeed_Left, vRefLeft); // set speed reference, duty-cycle
= 50/255
  analogWrite(pinSpeed_Right, vRefRight);

  // Setup NES interface
  pinMode(latch, OUTPUT);
  pinMode(clk, OUTPUT);
  pinMode(datin, INPUT);

  digitalWrite(latch, HIGH);
  digitalWrite(clk, HIGH);

  // Setup servos

  servo0.attach(s0pin);
  servo1.attach(s1pin);
  servo2.attach(s2pin);


  // Setup serial transmission
  Serial.begin(9600);
  for (start1 = servo1.read(); start1 < 180; start1++) {
    servo1.write(start1);
  }
  for (start2 = servo2.read(); start2 < 155; start2++) {
    servo2.write(start2);
  }

  pinMode(LEDpin, OUTPUT);
}

void loop() {
  while (digitalRead(pinON)) {    // wait for ON switch

    controllerRead();

    if (toggle == 0) {
      while (controller_data == UP) {
        Forward();
```

```
    controllerRead();
  }
  while (controller_data == DOWN) {
    Back();
    controllerRead();
  }
  while (controller_data == LEFT) {
    Left();
    controllerRead();
  }
  while (controller_data == RIGHT) {
    Right();
    controllerRead();
  }
  while (controller_data == START) {
    Stop();
    controllerRead();
  }
}
else {
  if (controller_data == UP) {
    count1 = servo1.read();
    count2 = servo2.read();
    for (vert = 0; vert < 180; vert++) {
      servo1.write(count1);
      servo2.write(count2);
      if (count1 != 180) {
        count1++;
      }
      if (count2 != 155) {
        count2++;
      }
      delay(25);
      if ( (count1 == 180) && (count2 == 155) ) {
        break;
      }
    }
  }
  if (controller_data == DOWN) {
    count1 = servo1.read();
    count2 = servo2.read();
    for (vert = 0; vert < 180; vert++) {
      servo1.write(count1);
      servo2.write(count2);
      if (count1 != 3) {
        count1--;
      }
      if (count2 != 35) {
        count2--;
      }
      delay(25);
      if ( (count1 == 3) && (count2 == 35) ) {
        break;
      }
```

```
    }
  }
  if (controller_data == RIGHT) {
    adjust = servo2.read();
    if (adjust < 150) {
      adjust += 3;
    }
    servo2.write(adjust);
  }
  if (controller_data == LEFT) {
    adjust = servo2.read();
    if (adjust > 6) {
      adjust -= 3;
    }
    servo2.write(adjust);
  }
}

if (controller_data == A) {
  if (gripper == 23) {
    gripper = 90;
    servo0.write(gripper);
  }
  else {
    gripper = 23;
    servo0.write(gripper);
  }
}

if (controller_data == B) {
  if (toggle == 0) {
    toggle = 1;
  }
  else toggle = 0;
}

if (toggle == 1) {
  digitalWrite(LEDpin, HIGH);
}
else {
  digitalWrite(LEDpin, LOW);
}

if (controller_data == SELECT) {
  delay(1000);
  if (servo1.read() < 160) {
    for (start1 = servo1.read(); start1 < 180; start1++) {
      servo1.write(start1);
    }
  }
  while (controller_data != SELECT) {
    for (start2 = servo2.read(); start2 < 155; start2++) {
      servo2.write(start2);
      delay(50);
```

```
        }
        for (start2 = servo2.read(); start2 < 20; start2--) {
          servo2.write(start2);
          delay(50);
        }
        controllerRead();
      }
    }



    Stop();
    delay(200);
  } // end while
} // end loop


void controllerRead() {
  digitalWrite(latch, HIGH);
  digitalWrite(latch, LOW);
  for (int x = 0; x <= 7; x++) {
    bitWrite(controller_data, x, digitalRead(datin));
    digitalWrite(clk, HIGH);
    digitalWrite(clk, LOW);
  }
}

void Stop()//Stops Robot
{
  digitalWrite(pinCW_Left, LOW);
  digitalWrite(pinCC_Left, LOW);
  digitalWrite(pinCW_Right, LOW);
  digitalWrite(pinCC_Right, LOW);
}

void Forward()//Moves Forward
{
  digitalWrite(pinCW_Left, HIGH);
  digitalWrite(pinCC_Left, LOW);
  digitalWrite(pinCW_Right, HIGH);
  digitalWrite(pinCC_Right, LOW);
}

void Left()//Rotate Left
{
  digitalWrite(pinCW_Left, LOW);
  digitalWrite(pinCC_Left, HIGH);
  digitalWrite(pinCW_Right, HIGH);
  digitalWrite(pinCC_Right, LOW);
}

void Right()//Rotate Right
{
```

```
  digitalWrite(pinCW_Left, HIGH);
  digitalWrite(pinCC_Left, LOW);
  digitalWrite(pinCW_Right, LOW);
  digitalWrite(pinCC_Right, HIGH);
}

void Back()//Moves Back
{
  digitalWrite(pinCW_Left, LOW);
  digitalWrite(pinCC_Left, HIGH);
  digitalWrite(pinCW_Right, LOW);
  digitalWrite(pinCC_Right, HIGH);
}
```