Chris Horn chorn4
Joseph Wightman jww2

Honors Lab Report

Introduction and Problem Statement

The initial problem statement for this project was to create a system for automatically shifting through the gears on a typical 21 speed bike. The original problem statement called for a system that shifted completely autonomously, using an accelerometer to determine the users speed and input forces to determine the proper shift point. The system would be compact enough to fit on a bike, reliable enough to be used for extended periods of time without the need to readjust the system, and have a battery life long enough to last a typical days use (around three to four hours of constant riding).

The design must also be capable of being sold as a stand alone product, which could fairly easily be added to already functioning bike. Ideally the modifications could easily be undone should the user want to remove the bike.

Any components within the ECE 110 lab kit may be used. In addition, parts from the ECE Service Center may be borrowed for no cost, and components may be purchased from the ECE Supply Center or from online suppliers. The total cost of purchased components should remain as low as possible, with a soft cap at $100.

Over the course of the semester, a prototype must be submitted and tested. Any large components should be thoroughly characterized so that their behaviors are well understood and repeatable.

Design Description and Proposed Solution

Before the problem can be directly solved, it must be broken down into small, manageable parts. From there, a solution for each portion can be developed, testing, and integrated with the rest of the design. For this particular problem, the final product was broken down into three portions: mechanical integration with the bike, sensors and sensing, and digital logic and computation (see Appendix A, Figure 1 for block diagram of final product breakdown).

The physically most straightforward portion was the digital logic and computation system. This portion was comprised solely of an Arduino, which received information from the sensor and sensing portion and sent control signals to the mechanical integration portion. The most complicated portion of this was the coding involved with determining when to shift. During the initial stages of the product design, an accelerometer was used to gather data regarding the acceleration of the bike. Using this data, the Arduino was tasked with computing the speed of the bike in real time. While, in theory, a fairly easy task, problems arose while trying to implement it. The relatively primitive and inefficient algorithms originally used caused unacceptable lag within the Arduino. This lag was felt by the user in the form of late shifts, jittery movement, and, over time, inaccurate speed readings. Additionally, running continuously results in significantly higher power draw from the Arduino. While the decision to abandon the accelerometer was made before substantial real world data could be gathered regarding the battery life impact, initial estimates show around a 40% increase in power usage.

Once the accelerometer was replaced by user actuated switches, the role of the digital logic and computation portion become substantially simpler. No real time calculations were required, and the Arduino was only responsible for ensuring that the linear actuator remained in the correct position. This improved battery life, responsiveness, and greatly simplified the design and coding required. The Arduino was now in one of two states at all time: an active, shifting state, and an idle state. During the active state, the Arduino would power the appropriate PWM output pins to drive the linear actuator. While the Arduino is moving the linear actuator, it is also measuring the voltage across the potentiometer within the actuator. When the voltage is within a threshold distance of the voltage associated with a particular destination gear, the Arduino sets all outputs to zero, and enters the idle state. While in the idle state, the Arduino consumes minimal current (in the range of tens of mA) and waits for a user input. When one of the switches is pressed by the user, the Arduino determines the new destination gear, and enters the active state once again.

Another portion of the our final product is the mechanical integration with the physical bike. The portion is comprised of the linear actuator, an H-bridge IC chip, and parts of the test bike. The linear actuator is a motorized linear potentiometer; by measuring the voltage drop across it, the Arduino is capable of determine the exact location of the actuator (see Appendix A, Figure 2 for schematic). This feedback allows the Arduino to accurately drive the actuator to the same location, plus or minus less than a millimeter, repeatedly. Initial testing showed no increase in the error distance after repeated shifts, or between restarts. The datasheet for the linear actuator lists a tolerance of 10% for potentiometer resistance at any given spot. However, while the actual resistance may be as much as 10% off from the predicated resistance, it does not change greatly as a function of time or use. Therefore, if this system was rebuilt with fresh parts from scratch, the new actuator would have to characterized to test for differences in resistance.

The design choice to use a linear actuator in lieu of a servo was done for two reasons: power consumption and cost. Servos, unlike motors, consume power even when they are not actively moving to a new location. This means that the servo would be a constant power drain, possibly reducing battery life and creating additional strain on the system. Additionally, servos are substantially more expensive. The original problem statement called for a low cost system, ideally marketed to college students and hobbyists. While adopting a servo for a more professional tier, entry level automated gear shifters could achieve a far lower price by using motor driven linear actuators. With some feedback, a motor driven linear actuator could achieve tolerances similar to that of a high cost servo.

The motor is driven by a half H-bridge IC chip. This chip's logic is powered by a 5V pin on the Arduino, and the motors are driven by a 6V line directly pulled from the batteries. There are three inputs to the chip from the Arduino: 2 PWM inputs that determine motor speed and direction, and a digital enable signal. The enable signal was not hardwired as an additional safety feature. For specific faults, the Arduino sets the enable signal low, completely disconnecting any power to the motor.

The final portion of the design is the sensors and sensing portion. Initially this section was comprised of an accelerometer, reporting to Arduino. However, for reasons outlined in the digital logic and computation section, the decision to move to switches was made. The switches were connected to a digital high voltage on one end, and to an Arduino input pin on the other. In addition, on the Arduino side, a pull down resistor was used to prevent floating voltages could be cause unwanted gear shifts. These switches, for the prototype, were mounted on a model handlebar. By placing them on the underside of the handlebar, and close to the hand, the user is able to easily depress the button with their thumb. Within a second or two, the entire system would react and shift gears the exact number of times a button was pressed. This allows the user to quickly scale up or down a specific number of gears, without risk of over or under shooting their desired destination gear.

In order to install the product onto an existing bike, the user must disconnect the gear shift cable from the handle mounted selector. This cable is then connected to the linear actuator, which mimics the tension that the hand operated selector normal puts on the cable. By fighting or loosening the cable, by moving linearly, the actuator is capable of changing gears. This system does require some tools and basic bike knowledge. However, it is completely reversible in the event that the user wants to remove the product from their bike. Additionally, the hand operated selectors are completely disconnected while the product is in use. This means in the event of a failure, there is no way to change gears without stopping and removing the product, potentially causing danger to the rider or others around them. The product does not interfere or inhibit the brakes in any way whatsoever, so the rider can safely come to a stop regardless of the condition of the automated gear shifter.

Measurements and Testing

Due to the low tolerances of the linear potentiometer, testing was required to find the exactly resistances for the various points needed. Testing was fairly simple: place the potentiometer in series with a 10kΩ resistor, and drive 5V across them. According to the data sheet for the potentiometer, the maximum resistance was approximately 10kΩ, and occurred at maximum displacement from the motor. Therefore, as the linear potentiometer slide away from the motor, its resistance would increase from 0Ω up to 10kΩ, and the voltage drop it would range from 0V to 2.5V. Initial testing showed that this was true. With the basic principles down, the voltage drop across the potentiometer at various predetermined points was measured and recorded. These values were then inputed into the Arduino code as the target voltages for various destination gears. Note that the actual values coded in are not voltages, but rather integers from 0 to around 512. This is because the Arduino reads the analog input on a scale from 0 to 1023, with 1023 being 5V. Due to the fact that the largest voltage drop across the linear actuator is 2.5V, the largest value ever measured at the analog pin is ~512.

With target voltages determined, testing began. Initial testing showed that the linear actuator would oscillate around the target position for a few cycles before finally resting into its steady state position. In order to reduce this wobble, a threshold value was added. When the voltage is with the threshold distance from the target value, the Arduino says “close enough” and stops moving the actuator. This improves stability, decreases power consumption, and improves reliability. It did, however, increase the area in which the linear actuator ended up in by a millimeter. This distance was too small to cause any trouble with the bike's drivetrain, so it is considered acceptable.

The only other problem encountered was with the pulldown resistors in the sensor and sensing portion. Originally, 1MΩ resistor was used. This sometimes caused the Arduino to read a high value for longer than expected, limiting the rate at which the user could move through the gears (the Arduino would only shift one gear per digital high). Replacing the 1MΩ resistor with a 100 kΩ solved this problem.

Conclusion

Overall, after the abandonment of the accelerometer and adoption of user pressed switches, the project went fairly smoothly. Problems that arose were not overly difficult to solve, and their solutions presented few, if any, complications. There are some inherit restrictions to using the linear actuator, such as limited torque and a possibly high wear rate. However, for an entry level prototype, the design is solid and should provide multiple hours of use before failing. The total cost of components was well under $50, and the MSPR of retail counterparts are often in the range of $150 to $200. Most currently available version of this design require the purchase of a specialized bike, and can not be added to an already existing platform. The base platform developed for this project may be the solution that the market is looking for. If specialty parts could be machined, and a few modifications made, the platform could likely be durable enough to go to market.

By breaking down the problem statement into easily solved parts, a team of two with 14 weeks and $50 is capable of designing a prototype of a design that does not currently exist on the open market. With a larger team, a high budget, and longer time frame, a to go market product and plan could be developed and acted upon.

Appendix A

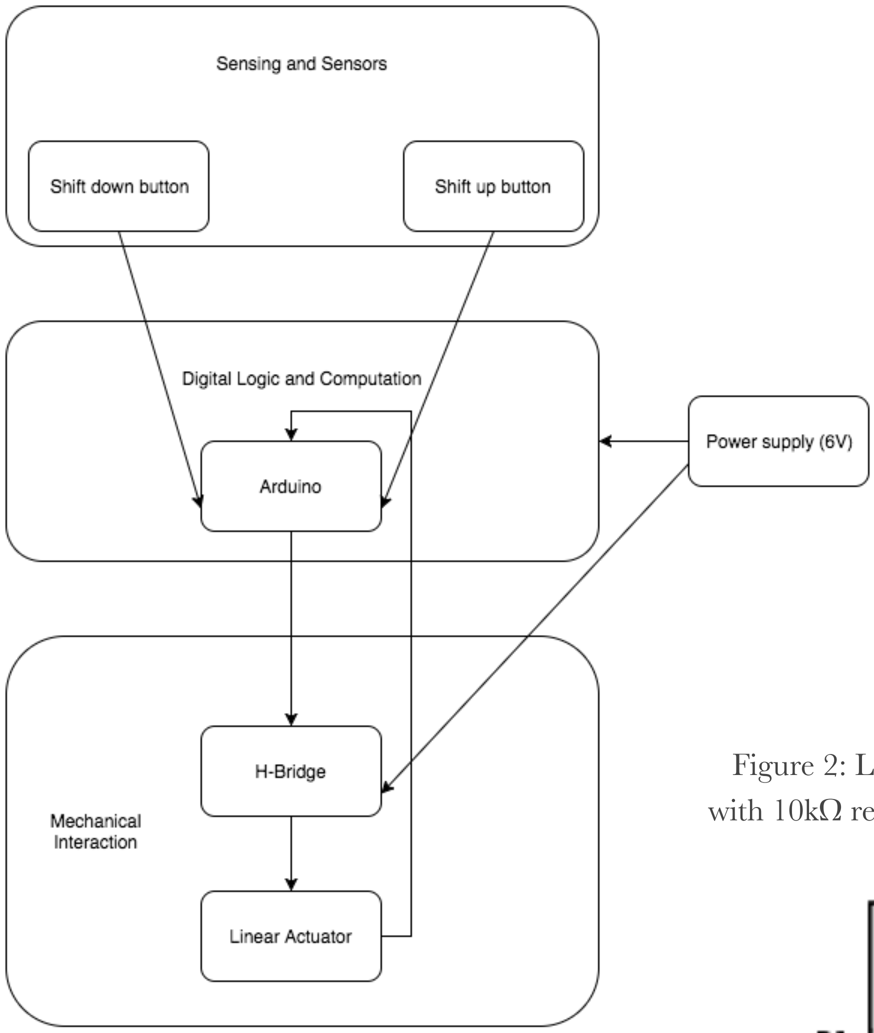
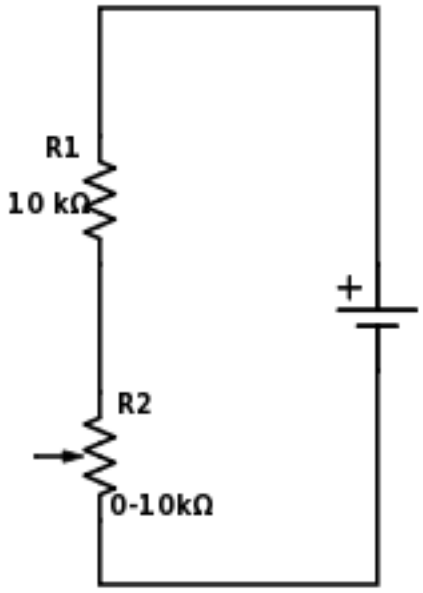


Figure 1: Block Diagram of System

Figure 2: Linear Potentiometer in Series with 10kΩ resistor (voltage measured across R2)



Appendix B: Arduino Code

```
//Constants

int GearPositions[] = {100, 150, 200, 250, 300, 350, 400};
int UpGearPin = 13;
int DownGearPin = 8;
int currentGear = 3;
int voltageReadPin = A0;
int delayT = 10;
int Threshold = 4;
int motorSpeed = 240;
int MotorPin1 = 11;
int MotorPin2 = 10;



void setup() {
  // put your setup code here, to run once:
  pinMode(voltageReadPin, INPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(UpGearPin, INPUT);
  pinMode(DownGearPin, INPUT);
  Serial.begin(9600);

  //                      A4
  //Primary 7-segment display pins    ________
  pinMode(3, OUTPUT);     //     |       |
  pinMode(2, OUTPUT);     //  A3 |       | 2
  pinMode(7, OUTPUT);     //     |   7   |
  pinMode(4, OUTPUT);     //      -------
  pinMode(A4, OUTPUT);        //        |       |
  pinMode(A3, OUTPUT);        //     A2 |       | 3
  pinMode(A2, OUTPUT);        //        |   4   |
  pinMode(2, OUTPUT);        //        -------      . <-- 2



}

void loop() {

  Serial.println(analogRead(A0));
  Serial.println(currentGear);
  if (digitalRead(UpGearPin) || digitalRead(DownGearPin))
  {
    if (digitalRead(UpGearPin) && currentGear < 7)
    {
      shiftGear("up");
      displayGear();
      while(digitalRead(UpGearPin))
      {
        delay(10);
      }

    }
    else if (digitalRead(DownGearPin) && currentGear > 1)
    {
      shiftGear("down");
      displayGear();
      while(digitalRead(DownGearPin))
      {
        delay(10);
      }
    }
  }
}

void shiftGear(String shift_direction)
{
  float targetVoltage;
  if(shift_direction.equalsIgnoreCase("up"))
  {
    currentGear = currentGear + 1;
    targetVoltage = GearPositions[currentGear - 1];
  }
  else if(shift_direction.equalsIgnoreCase("down"))
  {
    currentGear = currentGear - 1;
    targetVoltage = GearPositions[currentGear - 1];
  }
  float currentVoltage = analogRead(voltageReadPin);
  float distanceFromTargetVoltage = targetVoltage - currentVoltage;

  while (abs(distanceFromTargetVoltage) > Threshold)
  {
    if (distanceFromTargetVoltage < 0)
    {
      moveMotor(motorSpeed, "term3");
      currentVoltage = analogRead(voltageReadPin);
      distanceFromTargetVoltage = targetVoltage - currentVoltage;
    }
    else if (distanceFromTargetVoltage > 0)
    {
      moveMotor(motorSpeed, "term1");
      currentVoltage = analogRead(voltageReadPin);
      distanceFromTargetVoltage = targetVoltage - currentVoltage;
    }
  }
  stopMotor();
}

void moveMotor(int Speed, String move_direction)
{
  if(move_direction.equalsIgnoreCase("term3"))
  {
    analogWrite(MotorPin1, Speed);
    delay(20);
    //analogWrite(MotorPin1, 0);
    stopMotor();
  }
  else if(move_direction.equalsIgnoreCase("term1"))
  {
    analogWrite(MotorPin2, Speed);
    delay(20);
    //analogWrite(MotorPin1, 0);
    stopMotor();
  }
}
void stopMotor()
{
  analogWrite(MotorPin1, 0);
  analogWrite(MotorPin2, 0);
  delay(10);
}
```

```
void displayGear(){
 if(currentGear == 0)
 {
  digitalWrite(2, LOW);
  digitalWrite(3, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(4, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A3, LOW);
  digitalWrite(A2, LOW);
 }
 else if(currentGear == 1)
 {
  digitalWrite(3, LOW);
  digitalWrite(2, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(A4, HIGH);
  digitalWrite(A3, HIGH);
  digitalWrite(A2, HIGH);
 }
 else if(currentGear == 2)
 {
  digitalWrite(3, HIGH);
  digitalWrite(2, LOW);
  digitalWrite(7, LOW);
  digitalWrite(4, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A3, HIGH);
  digitalWrite(A2, LOW);
}
else if(currentGear == 3)
{
  digitalWrite(3, LOW);
  digitalWrite(2, LOW);
  digitalWrite(7, LOW);
  digitalWrite(4, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A3, HIGH);
  digitalWrite(A2, HIGH);
}
else if(currentGear == 4)
{
  digitalWrite(3, LOW);
  digitalWrite(2, LOW);
  digitalWrite(7, LOW);
  digitalWrite(4, HIGH);
  digitalWrite(A4, HIGH);
  digitalWrite(A3, LOW);
  digitalWrite(A2, HIGH);
}
else if(currentGear == 5)
{
  digitalWrite(3, LOW);
  digitalWrite(2, HIGH);
  digitalWrite(7, LOW);
  digitalWrite(4, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A3, LOW);
  digitalWrite(A2, HIGH);
}
```

```
else if(currentGear == 6)
{
  digitalWrite(3, LOW);
  digitalWrite(2, HIGH);
  digitalWrite(7, LOW);
  digitalWrite(4, LOW);
  digitalWrite(A4, HIGH);
  digitalWrite(A3, LOW);
  digitalWrite(A2, LOW);
}
else if(currentGear == 7)
{
  digitalWrite(3, LOW);
  digitalWrite(2, LOW);
  digitalWrite(7, HIGH);
  digitalWrite(4, HIGH);
  digitalWrite(A4, LOW);
  digitalWrite(A3, HIGH);
  digitalWrite(A2, HIGH);
}
else if(currentGear == 3)
{
  digitalWrite(3, LOW);
  digitalWrite(2, LOW);
  digitalWrite(7, LOW);
  digitalWrite(4, LOW);
  digitalWrite(A4, LOW);
  digitalWrite(A3, LOW);
  digitalWrite(A2, LOW);
}
else if(currentGear == 9)
```

```
 {
   digitalWrite(3, LOW);
   digitalWrite(2, LOW);
   digitalWrite(7, LOW);
   digitalWrite(4, HIGH);
   digitalWrite(A4, LOW);
   digitalWrite(A3, LOW);
   digitalWrite(A2, HIGH);
 }
}
```