Digital projects - Door lock

EITF40 - LTH

George Ryrstedt - dt07gr1
Fredrik Annerstedt - dt07fa8

Advisor: Bertil Lindvall

March 1, 2011

Abstract

The aim of this project was to construct a door lock that responded to a set of predefined knock sequences. The idea was that it is all too easy to forget your keys and for example lock yourself out of your apartment. Having to remember a certain sequence instead is much easier and eliminates the need for any physical keys. For additional security Bluetooth reception of the owners cellphone is also required.

Contents

1 Introduction

1.1 About the course

The goal of the course is to simulate real industrial development. More specifically to learn the necessary skills needed in order to build a working prototype for further production. First and foremost it provides an insight on how software and hardware integrates and work together. The course is without any scheduled lectures and requires much individual work and time management.

1.2 About the project

The project in this report is a door look that responds to a preprogrammed sequence of knocks. The intended function is that the user knocks the sequence on the door and if it is correct the door should open automatically. To make it easier for the user the knocks should not be based on an absolute time frame. Rather the relative time between each knock is the important factor. In case someone accidentally happens to produce the correct sequence the users cellphone must be in Bluetooth range for the door to open. The lock is reprogrammable and have a LCD-screen which outputs status messages and debug information.

2 Requirements

In the beginning of the project a basic requirements specification was created.

- The lock should be constantly ready to receive the key sequence.
- The lock should only accept the correct knock sequence, based on relative timing.
- The lock should only open the door if both the knock sequence and the predefined cellphone is in range.
- The lock should be easily reprogrammable with a new knock sequence.
- The predefined cellphone should be easily changeable.
- The lock should have a LCD to display current state and other information.
- The lock should have status LEDs to indicate its current state.

3 Hardware

3.1 Arduino BT with ATMega168

The whole project is based around an AVR ATMega168 microprocessor which is integrated in an Arduino board. The ATMega is a complete microcontroller with RAM, flash memory and EEPROM. The initial thought was to just use the ATMega16 but the Arduino was chosen for its simple Bluetooth integration. Otherwise Bluetooth would probably have been too complex to integrate in the specified timeframe. Both the ATMega and the Arduino board are well

known as among hobbyists which makes it easy to find documentation and other information. The Arduino also makes it possible to do everything in C++ instead of just using C.

3.2 LCD Hitachi 44780

The display consists of spaces for 2x16 characters and is based on the Hitachi 44780 chip. This project uses it for outputting information about the state of the lock. Also debug information in the development phase of the project.

3.3 H-bridge and motor

An ordinary LEGO-motor is used in the project to simulate the opening of the lock. When the door opens the motor first rotates in one direction for a couple of seconds then wait for the user to get inside and then rotates in the other direction, simulating the locking. To be able to turn the motor both ways a simple H-bridge is used. This also enables the ability to connect an external power source to the motor so that in a real world scenario a more powerful motor can be used.

3.4 Other hardware

A trimmable potentiometer (trimpot) is used to vary the contrast on the LCD. LEDs are used to indicate basic states, but was mostly used during development.

Buttons are used to simulate door knocks and to reprogram the device. The original idea was to use a piezo speaker to detect knocks but due to availability buttons are used instead. There were unexpected problems with button debouncing and a low pass filter consisting of resistors and capacitors had to be implemented.

4 Software design

The first problem with the software was the fact that the user had to be able to set a knock sequence of variable length. This immediately ruled out using a standard array as if the user entered a knock sequence that was longer then the array was designed to hold it would result in undefined and potentially dangerous behavior. To get around this a simple vector class was implemented that internally stored the values in an array giving O(1) fetches of data and O(n) insertion time. The worst case insertion time occurs when an object is placed after the last position in the array, in which case a new and bigger array was created. The old one copies to the new one and the old one gets deleted. The second problem was how to implement code to detect when a knock occurred, or the programming button was pressed. Had polling been used the program might potentially miss the event if it occurred faster then the main loops execution time. Instead the ATMega's two available interrupts bound to the two events and store the time the knock occurred or to enter programming mode based on the event that occurred. If the event was of a knock type the "knockDetected"-function was called and the current time stored in the knocks vector. However if the programming button was pressed the knock handling routine was changed

from the default "knockDetected" to "programKnocks" which essentially does the same thing as "knockDetected" except instead of storing the time values in the "knocks "vector it stores them in the "openSeq" vector. The main loop of the program continuously runs and checks whether the "knocks" and "openSeq" vector are equal in length; if they are it compares the relative time intervals that the knocks occurred at in the array and if they are equal to each other within the defined tolerances, the door opens. The software was implemented using C++ as there were no hard realtime constraints on the system and the features of C++ enabled design choices that would not have been available had C been used e.g. implementing a custom vector class.

5 Execution

The original plan was to build a complete system which would be ready to be installed in a standard door. However due to several mechanical obstacles the project was scaled down somewhat. The actual fitting of the lock and everything surrounding that e.g. the motor torque required to open a lock were not problems that felt relevant to the course. It was decided to skip those parts and replace them with simulated components in order to focus on the electronic parts and the software. A LEGO-motor was used to simulate a real high torque motor but it could easily be replaced with a real one should anyone wish to implement the system in an actual door.

A piezo speaker would have provided the input for the knock sequence but since no one was available and and it was possible to do it just as well with buttons, no further time was spent investigating that. Buttons replaced the piezo entirely and while a few possible problems disappeared with that others appeared instead. A substantial amount of time was spent trying to eliminate debouncing. The buttons would trigger irregularly and in effect became totally useless in that state. However after a simple low-pass filter was implemented better results were noted. They were still not up to the task of emulating knocks though. With additional code to further eliminate debouncing an acceptable reliability was reached. Just as with the motor the buttons should be somewhat easily replaceable with a pizeo speaker in a real system.

6 Results

The prototype is working as intended and simulates opening of the door when the correct sequence is entered. It should be relatively easy to modify this prototype to a working production model. The basic operation is there only the form factor and some of the components need revision. It satisfies all of the requirements that were set up in the beginning. In addition to this it is also possible to open the door with only the specified cellphone.

7 Discussion

During the project several problems where encountered in regards to the analog components. The first problem we encountered was that sometimes when pressing the button repeatedly in rapid succession the software would register

two button presses when the button was only pressed once. The first solution to this problem we tried was implement a low pass filter between the button and the input pin, this reduced the problem some what but the software would still sometimes register one too many presses so to argument the hardware solution a software fix was implemented where button presses occurring within less then 100ms from the last registered one are ignored. The second major hardware problem we encountered was that at what appeared to random times the Arduino stopped responding to input or giving output as well as behaving erratically when buttons were pressed with no apparent pattern. After a while we realized that this was occurring every time we tried to run the motor through the H-bridge and due to the induced current spike the Arduino entered an undefined state, the solution to this was as simple as adding a capacitor between ground and Vcc before the H-bridge.

References

[1] AVR ATMega16
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Processors/ATmega16.pdf

[2] Hitachi 44780
http://www.eit.lth.se/fileadmin/eit/courses/edi021/datablad/Display/hd44780.pdf

[3] Arduino BT
http://arduino.cc/en/Main/ArduinoBoardBluetooth

A Schematics

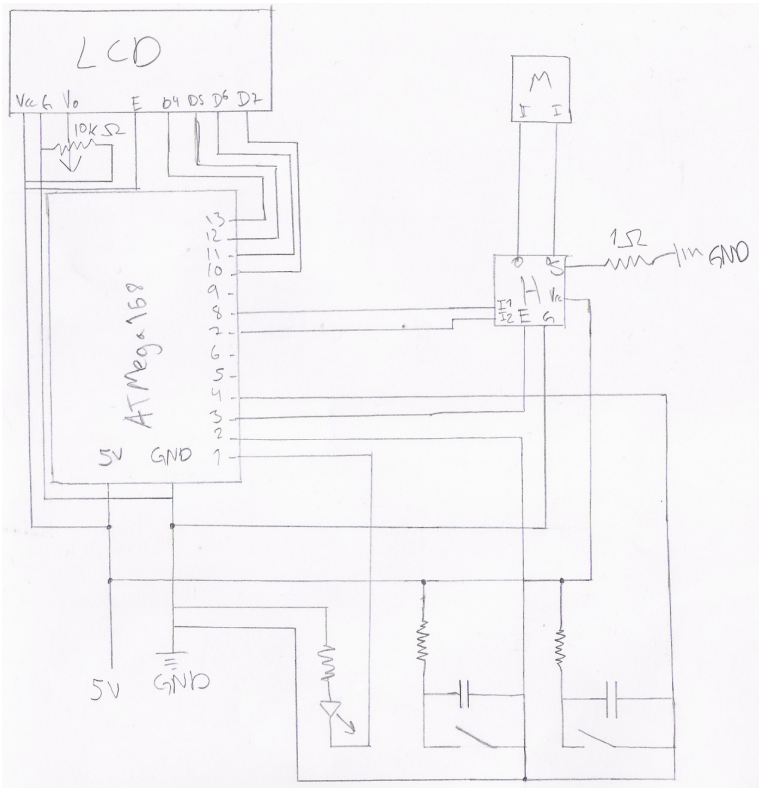

Figure 1: Basic overview

B Source code

```
#define TOLERANCE 0.5 //tolerance ie 0.5 = plus/minus 50%
#define TIMEOUT 5000 //Timeout between knocks in milliseconds
#define DEBOUNCE 100 //minimum amount of time to wait between button presses in ms


//////////////////////////Class declarations/////////////////////////////////
template<typename T>
class Vector
{
public:
  Vector(int size) : mySize(size), myLastValue(-1)
  {
        array = (T *)malloc(mySize * sizeof(T));
  }

  ~Vector()
  {
        free(array);
```

```
  }

  void set(int position, T value)
  {
        if(position > myLastValue)
        {
    myLastValue = position;
        }
        if(position > (mySize - 1))
        {
    T* tmp = array;
    array = (T *)malloc((position + 1) * sizeof(T));
    memcpy(array, tmp, mySize);
    free(tmp);
    mySize = position + 1;
        }
        array[position] = value;
  }

  void push(T value)
  {
        set((myLastValue + 1), value);
  }

  T get(int position)
  { return array[position]; }

  int size()
  { return (myLastValue + 1); }

  T lastValue()
  { return array[myLastValue]; }

  void clear()
  { myLastValue = -1; }

private:
  int myLastValue;
  int mySize;
  T* array;
};


////////////////////////////////////////////////////////////////////////////////

volatile int motorstate = LOW;
volatile bool programMode = 0;
int knockpin=2;
int programpin=3;
int motorEnablePin = 8;
int motorForwardsPin = 7;
int motorBackwardsPin = 6;
int ledpin=13;
int ledState = LOW;
Vector<float> knocks(10);
Vector<float> openSeq(10);
int nothing = 0;

void setup() {
  attachInterrupt(0, knockDetected, FALLING);
  attachInterrupt(1, enterProgramMode, FALLING);
  pinMode(programpin, INPUT);
```

```
  pinMode(knockpin, INPUT);
  pinMode(ledpin, OUTPUT);
  pinMode(motorEnablePin, OUTPUT);
  pinMode(motorForwardsPin, OUTPUT);
  pinMode(motorBackwardsPin, OUTPUT);
  Serial.begin(115200);
  Serial.println("Started");
  openSeq.push(0);
  openSeq.push(1);
  openSeq.push(2);
}

void loop() {
  //Serial.println(Serial.list().length);
  if(Serial.available() > 0)
        {
        byte data = Serial.read();
        if(data == 'o')
    {
      openDoor();
    }
    }
  if(openSeq.size() != 0 && knocks.size() == openSeq.size())
        {
        openDoor();
        /*
        Serial.print("running detection code\n");
        toggleLed(); //give some debugging output
        float distanceInput;
        float distanceExpected;
        float totalTimeInput = knocks.lastValue() - knocks.get(0);
        float totalTimeExpected = openSeq.lastValue() - openSeq.get(0);
        int correct = 0;
        int nmbrDatapoints = openSeq.size() - 1;
        for(int i = 0; i < nmbrDatapoints; i++)
    {
      distanceInput = (knocks.get(i + 1) - knocks.get(i)) / totalTimeInput;
      distanceExpected = (openSeq.get(i + 1) - openSeq.get(i)) / totalTimeExpected;
      //Serial.print("Distnance");
      //Serial.print(distanceInput, 3);
      //Serial.print(" expected value: ");
      //Serial.print(distanceExpected, 3);
      //Serial.print("\n");
      if(distanceInput > (distanceExpected * TOLERANCE) && distanceInput < (distanceExpected
        {
        correct++;
        }
      else
        {
        break;
        }
    }
        if(correct == nmbrDatapoints)
    {
      openDoor();
    }
        */
    knocks.clear(); //reset position to reenable knockDetection
        }
}

void knockDetected()
```

```
{

  float time = millis();
  if(knocks.size() < openSeq.size() && (knocks.size() == 0 || time > (knocks.lastValue() + DE
        {
        Serial.println("knock detected");
        if(knocks.size() != 0 && time > (knocks.lastValue() + TIMEOUT)) //if timeout since la
    {
      Serial.println("New sequence");
      knocks.clear();
    }
        knocks.push(time);
        }
}

void programKnocks()
{
  float time = millis();
  if(openSeq.size() == 0 || time < (openSeq.lastValue() + TIMEOUT))
        {
        if( time > (openSeq.lastValue() + DEBOUNCE))
    {
      Serial.println("adding knock");
      openSeq.push(time);
    }
        }
  else
        {
        Serial.println("exiting programming mode");
        attachInterrupt(0, knockDetected, FALLING);
        knockDetected();
        }
}

void enterProgramMode()
{
  Serial.println("Entering programming mode");
  openSeq.clear();
  knocks.clear(); //prevent loop code from running once we recivie a knock or two
  attachInterrupt(0, programKnocks, FALLING);
}

void toggleLed()
{
  ledState = !ledState;
  digitalWrite(ledpin, ledState);
}

void openDoor(){
  int waitDelay = 5000;
  digitalWrite(motorForwardsPin, HIGH);
  digitalWrite(motorEnablePin, HIGH);
  delay(waitDelay);
  digitalWrite(motorEnablePin, LOW);
  digitalWrite(motorForwardsPin, LOW);
  delay(waitDelay);
  digitalWrite(motorBackwardsPin, HIGH);
  digitalWrite(motorEnablePin, HIGH);
  delay(waitDelay);
  digitalWrite(motorEnablePin, LOW);
  digitalWrite(motorBackwardsPin, LOW);
}
```