

TECHNICAL UNIVERSITY OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE

DEPARTMENT OF AUTOMATION

TECHNICAL REPORT

CONTROLLING INFRARED CAPABLE DEVICES WITH ARDUINO

Student: Florentin-Mircea Pietrar

Contents

| | |
|--|----|
| 1.Introduction..... | 3 |
| 2.Project Summary..... | 3 |
| 2.1 About Infrared communication..... | 3 |
| 2.2 Hardware design | 5 |
| 2.3 Software design..... | 8 |
| 2.4 Improvements for final product | 9 |
| Webography..... | 9 |
| Appendix-Arduino code..... | 10 |

1.Introduction

This report presents the whole process of creating a universal infrared controlling device based on Arduino microcontroller. The goal of this project is to have the ability to learn as many remotes from different devices and the ability to control the devices with Arduino. In final form the goal is that a raspberry pi sends a command via Serial communication to Arduino and it transmits the desired infrared command.

2.Project Summary

2.1 About Infrared communication

Infrared (IR) communication is a widely used and easy to implement wireless technology that has many useful applications. The most prominent examples in day to day life are TV/video remote controls, motion sensors, and infrared thermometers.

What is infrared?

Infrared radiation is a form of light similar to the light we see all around us. The only difference between IR light and visible light is the frequency and wavelength. Infrared radiation lies outside the range of visible light, so humans can't see it:

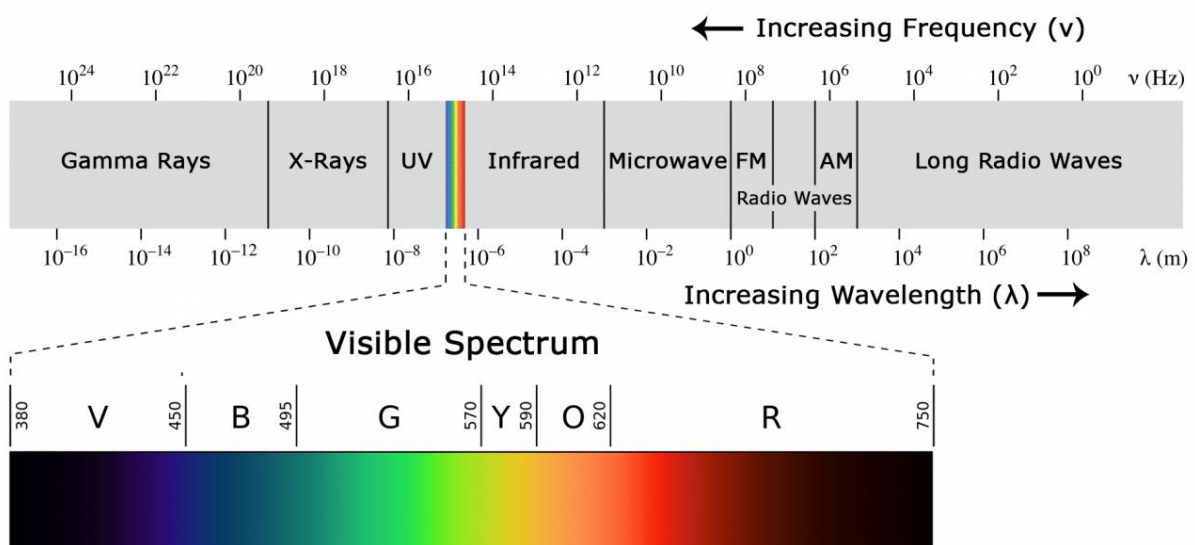


Figure 1 Electromagnetic spectrum

Because IR is a type of light, IR communication requires a direct line of sight from the receiver to the transmitter. A typical infrared communication system requires an IR transmitter and an IR receiver.

IR SIGNAL MODULATION

IR light is emitted by the sun, light bulbs, and anything else that produces heat. That means there is a lot of IR light noise all around us. To prevent this noise from interfering with the IR signal, a signal modulation technique is used.

In IR signal modulation, an encoder on the IR remote converts a binary signal into a modulated electrical signal. This electrical signal is sent to the transmitting LED. The transmitting LED converts the modulated electrical signal into a modulated IR light signal. The IR receiver then demodulates the IR light signal and converts it back to binary before passing on the information to a microcontroller:

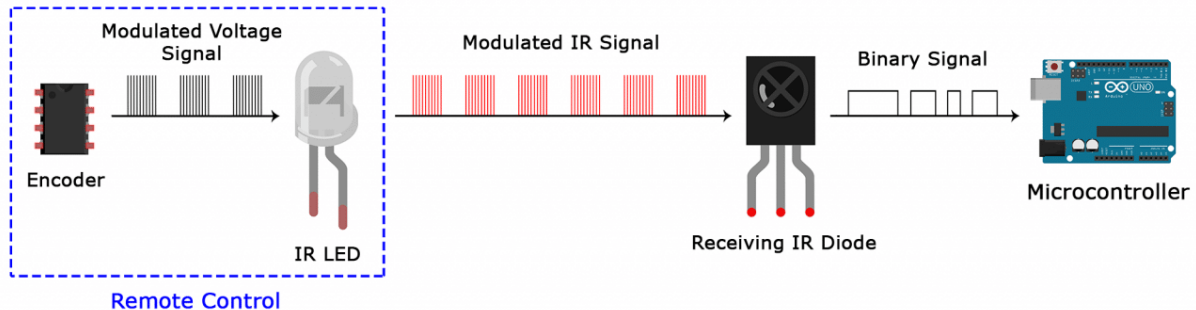


Figure 2 IR signal modulation

The modulated IR signal is a series of IR light pulses switched on and off at a high frequency known as the carrier frequency. The carrier frequency used by most transmitters is 38 kHz, because it is rare in nature and thus can be distinguished from ambient noise. This way the IR receiver will know that the 38 kHz signal was sent from the transmitter and not picked up from the surrounding environment.

The receiver diode detects all frequencies of IR light, but it has a band-pass filter and only lets through IR at 38 kHz. It then amplifies the modulated signal with a pre-amplifier and converts it to a binary signal before sending it to a microcontroller.

IR TRANSMISSION PROTOCOLS

The pattern in which the modulated IR signal is converted to binary is defined by a transmission protocol. There are many IR transmission protocols but NEC, and RC5 are the more common protocols.

IR CODES

Each time a button is pressed on the remote control, a unique hexadecimal code is generated. This is the information that is modulated and sent over IR to the receiver.

2.2 Hardware design

For this project I used an Arduino UNO R3 as the main processing unit. The other two most important hardware components are the IR receiver and the IR LED. These hardware components are not enough for what we want because the Arduino cannot supply sufficient current to drive an IR LED and to solve this drawback, I used a NPN 2N2222 transistor and a 470-ohm resistor. The electrical scheme of the connection of the transistor, resistor and IR LED is the following:

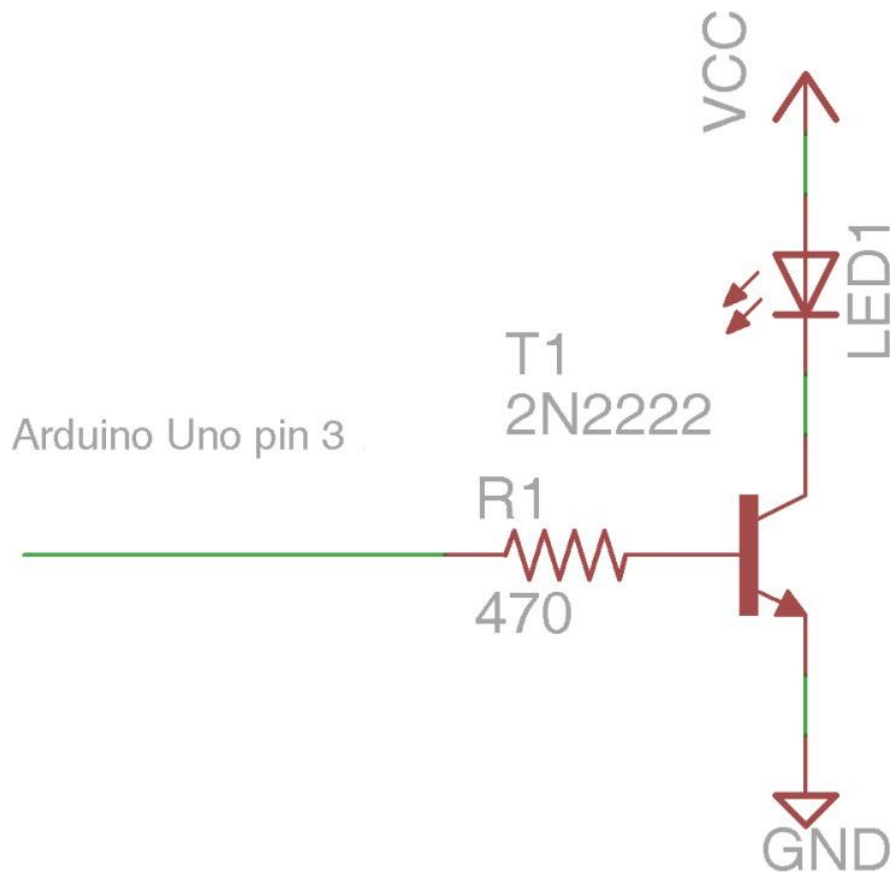


Figure 3 Electrical connection of IR LED

The final connections of the whole circuit are the following:

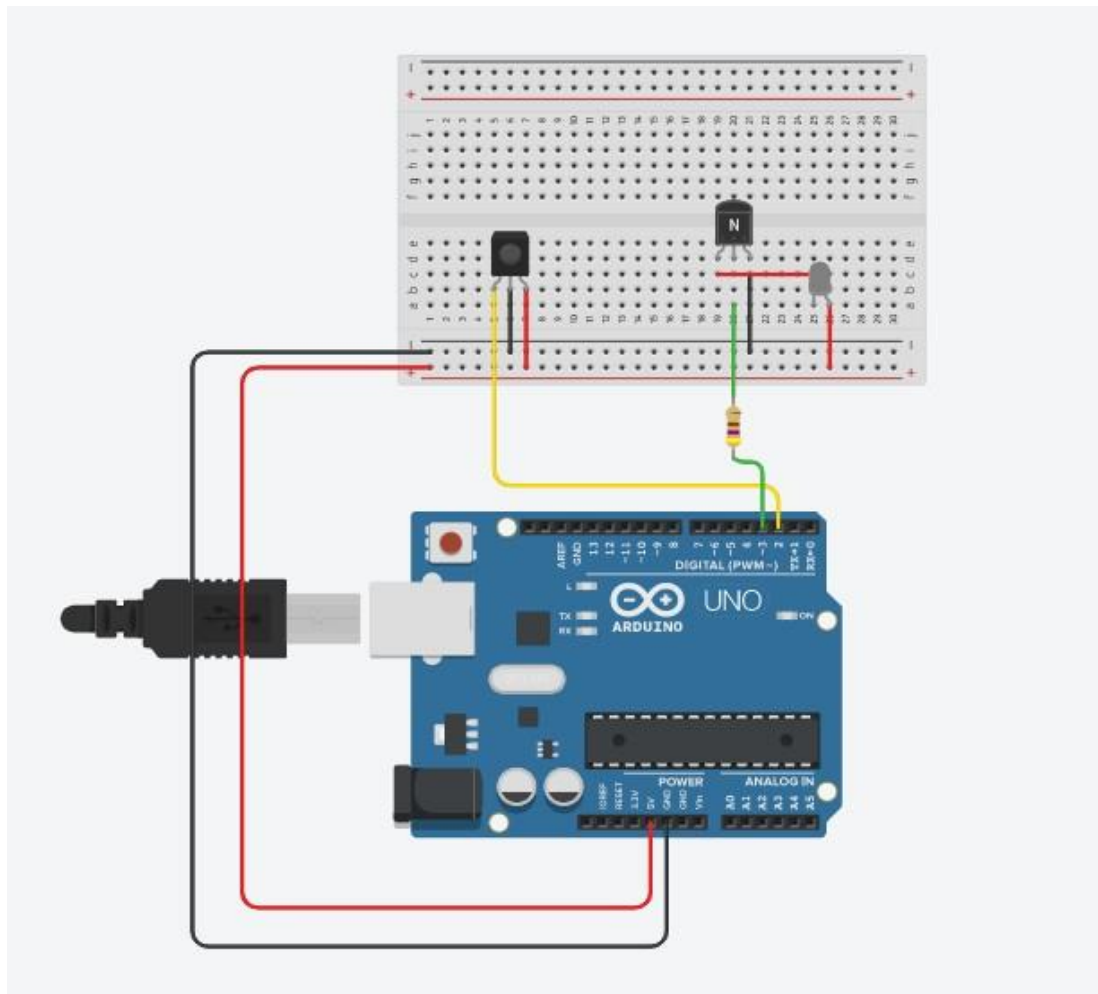


Figure 4 Circuit connection

Yellow wire-connected to the out terminal of the IR receiver and to the digital pin 2

Green wire-connected to the base of the transistor and through resistor to the digital pin 3

The IR LED anode is connected to the Vcc and the cathode to the collector of the transistor.
The emitter of the transistor is connected to the ground.

Physical build of the circuit:

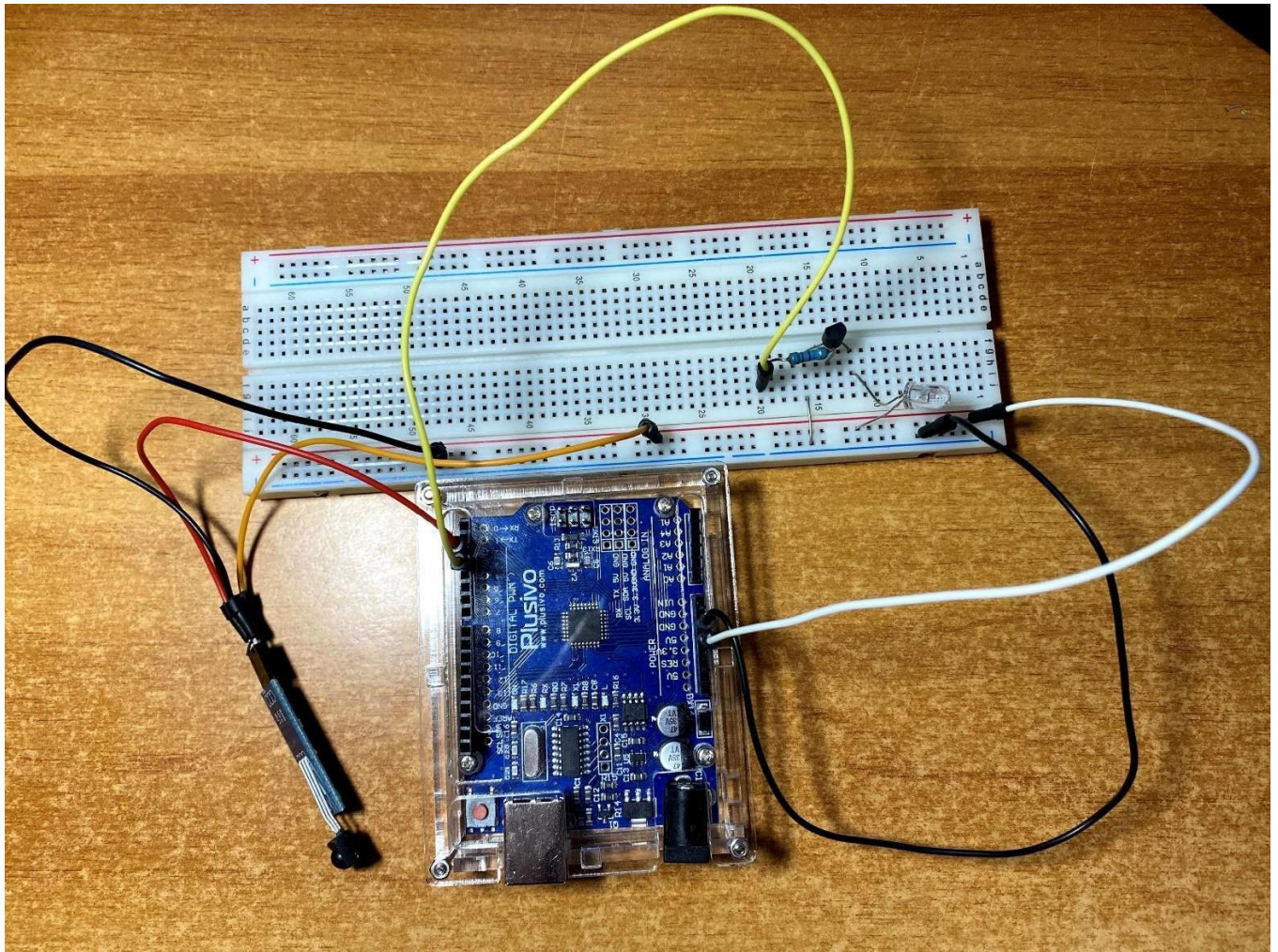


Figure 5 Physical circuit

2.3 Software design

For the software implementation I used a library called IRLib2 which simplifies a lot the implementation of the receiving and sending IR signals.

The logic of the software is the following: if it doesn't have any hexadecimal codes from a remote, it enters in a learning mode in which it starts the IR receiver and start storing the codes sent by the remote. After learning all the desired buttons, they are saved in the EEPROM memory of the Arduino and from the serial monitor we can choose which signal to be sent.

Currently it only stores 5 buttons but this can be modified.

```
Received NECx Value:0xE0E0D02F
Received NECx Value:0xE0E0E01F
Received NECx Value:0xE0E048B7
Received NECx Value:0xE0E008F7
Received NECx Value:0xE0E016E9
```

☒ Autoscroll ☐ Show timestamp

Newline ▼ 9600 baud ▼ Clear output

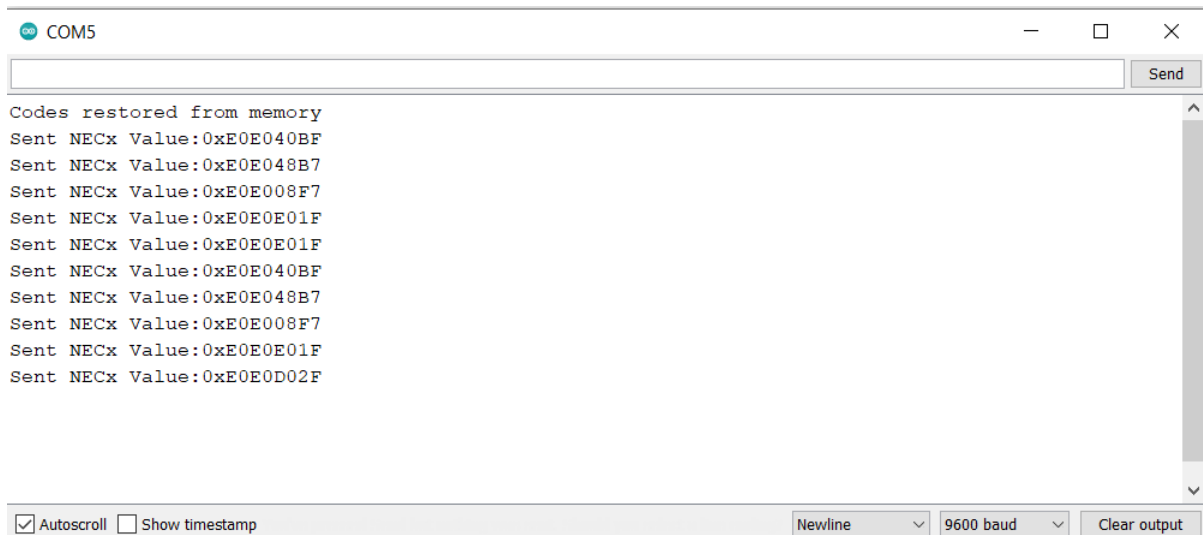
Figure 6 Learning of the remote

```
Codes restored from memory
Received NECx Value:0xE0E040BF
Received NECx Value:0xE0E048B7
Received NECx Value:0xE0E008F7
Reject code
Received NECx Value:0xE0E0E01F
Received NECx Value:0xE0E0D02F
```

☒ Autoscroll ☐ Show timestamp

Newline ▼ 9600 baud ▼ Clear output

Figure 7 Rejection of multiple codes for long press on remote



```
COM5
Codes restored from memory
Sent NECx Value:0xE0E040BF
Sent NECx Value:0xE0E048B7
Sent NECx Value:0xE0E008F7
Sent NECx Value:0xE0E0E01F
Sent NECx Value:0xE0E0E01F
Sent NECx Value:0xE0E040BF
Sent NECx Value:0xE0E048B7
Sent NECx Value:0xE0E008F7
Sent NECx Value:0xE0E0E01F
Sent NECx Value:0xE0E0D02F
```

Figure 8 Codes restored from EEPROM and send example

2.4 Improvements for final product

An improvement could be to store as many remotes and as many keys as we want and to have 4 IR LEDs such that we have a full 360-degree range of controlling IR devices.

Webography

How to Send Arduino IR Remote Signals-<https://www.arrow.com/en/research-and-events/articles/how-to-send-arduino-ir-remote-signals>

HOW TO SET UP AN IR REMOTE AND RECEIVER ON AN ARDUINO-<https://www.circuitbasics.com/arduino-ir-remote-receiver-tutorial/>

Sending IR Codes-<https://learn.adafruit.com/using-an-infrared-library/sending-ir-codes>IRLib2-<https://github.com/cyborg5/IRLib2>

Receiving and Decoding IR Software installation-<https://learn.adafruit.com/using-an-infrared-library/hardware-needed>

User's Manual for IRLib2 –

<http://courses.cs.tau.ac.il/embedded/projects/spring2018/ShAir/Arduino/libraries/IRLib2/manuals/IRLibReference.pdf>

Appendix-Arduino code

```
#include <IRLibDecodeBase.h> //We need both the coding and
#include <IRLibSendBase.h> // sending base classes
#include <IRLib_P01_NEC.h> //Lowest numbered protocol 1st
#include <IRLib_P02_Sony.h> // Include only protocols you want
#include <IRLib_P03_RC5.h>
#include <IRLib_P04_RC6.h>
#include <IRLib_P05_Panasonic_Old.h>
#include <IRLib_P07_NECx.h>
#include <IRLib_HashRaw.h> //We need this for IRsendRaw
#include <IRLibCombo.h> // After all protocols, include this
// All of the above automatically creates a universal decoder
// class called "IRdecode" and a universal sender class "IRsend"
// containing only the protocols you want.
// Now declare instances of the decoder and the sender.
IRdecode myDecoder;
IRsend mySender;

// Include a receiver either this or IRLibRecvPCI or IRLibRecvLoop
#include <IRLibRecv.h>
#include <EEPROM.h>

IRrecv myReceiver(2); //pin number for the receiver
// emitter is connected on pin 3

// Storage for the recorded code
//uint8_t codeProtocol[5]; // The type of code
//uint32_t codeValue[5]; // The data bits if type is not raw
//uint8_t codeBits[5]; // The length of the code in bits
```

```
// index of the stored codes
```

```
//unsigned index = 0;// index must be 0
```

```
struct AppSettings
```

```
{
```

```
    unsigned index = 0;
```

```
    uint8_t codeProtocol[5]; // The type of code
```

```
    uint32_t codeValue[5]; // The data bits if type is not raw
```

```
    uint8_t codeBits[5]; // The length of the code in bits
```

```
};
```

```
AppSettings mySettings;
```

```
void SaveSettings()
```

```
{
```

```
    int eeAddress = 0;
```

```
    EEPROM.put(eeAddress, mySettings);
```

```
}
```

```
void RestoreSettings()
```

```
{
```

```
    int eeAddress = 0;
```

```
    EEPROM.get(eeAddress, mySettings);
```

```
}
```

```
void ClearEEPROM()
```

```
{
```

```
    for (int i = 0; i < EEPROM.length(); i++)
```

```
    {
```

```
        EEPROM.write(i, 0);
```

```
    }
```

```
}
```

```
void setup() {  
  RestoreSettings();  
  if (mySettings.index == 0)  
  {  
    for (unsigned i = 0; i < 5; i++)  
    {  
      mySettings.codeValue[i] = 0;  
      mySettings.codeProtocol[i] = UNKNOWN;  
    }  
  }  
  Serial.begin(9600);  
  delay(2000); while (!Serial); //delay for Leonardo  
  if (mySettings.index == 0)  
    Serial.println(F("Send 5 codes from your remote and we will record it."));  
  else  
    Serial.println(F("Codes restored from memory"));  
  
  myReceiver.enableIRIn(); // Start the receiver  
}
```

```
// Stores the code for later playback
```

```
void storeCode(unsigned i) {  
  mySettings.codeProtocol[i] = myDecoder.protocolNum;  
  Serial.print(F("Received "));  
  Serial.print(Pnames(mySettings.codeProtocol[i]));  
  if (mySettings.codeProtocol[i] == UNKNOWN) {  
    Serial.println(F(" saving raw data."));  
    myDecoder.dumpResults();  
  }
```

```

        mySettings.codeValue[i] = myDecoder.value;
    }
    else {
        mySettings.codeValue[i] = myDecoder.value;
        mySettings.codeBits[i] = myDecoder.bits;
    }
    Serial.print(F(" Value:0x"));
    Serial.println(mySettings.codeValue[i], HEX);
}

void sendCode(unsigned i)
{
    mySender.send(mySettings.codeProtocol[i],mySettings.codeValue[i],
mySettings.codeBits[i]);
    if (mySettings.codeProtocol[i] == UNKNOWN) return;
    Serial.print(F("Sent "));
    Serial.print(Pnames(mySettings.codeProtocol[i]));
    Serial.print(F(" Value:0x"));
    Serial.println(mySettings.codeValue[i], HEX);
}

void loop() {

while (mySettings.index < 5)
{
    if (myReceiver.getResults()) {
        myDecoder.decode();

        if (mySettings.index >= 1)
        {

```

```

    if (myDecoder.value == mySettings.codeValue[mySettings.index - 1])
    {
        Serial.println("Reject code");
    }
    else
    {
        storeCode(mySettings.index);
        mySettings.index++;
    }
}
else
{
    storeCode(mySettings.index);
    mySettings.index++;
}
myReceiver.enableIRIn(); // Re-enable receiver

}
}

```

```

if (mySettings.index >= 5)
    SaveSettings();
while (mySettings.index >= 5)
{
    if (Serial.available()) {
        uint8_t C = Serial.read();
        if (C == 'a')
        {
            sendCode(0);
        }
    }
}

```

```
if (C == 's')
{
    sendCode(1);
}
if (C == 'd')
{
    sendCode(2);
}
if (C == 'f')
{
    sendCode(3);
}
if (C == 'g')
{
    sendCode(4);
}
if (C == 'r')
{
    ClearEEPROM;
    mySettings.index = 0;
    for (unsigned i = 0; i < 5; i++)
    {
        mySettings.codeValue[i] = 0;
        mySettings.codeProtocol[i] = UNKNOWN;
    }
    myReceiver.enableIRIn(); // Re-enable receiver
}

}
```