MIDI Gloves with LDR Sensors

By Mandar Deshmukh

Introduction:

This project aims to create a MIDI glove using an Arduino Nano microcontroller and Light Dependent Resistors (LDRs) to control MIDI signals based on hand movements. MIDI (Musical Instrument Digital Interface) allows musicians to create and manipulate sounds digitally, without the need for an actual instrument, and this project aims to use LDRs as light sensors to trigger MIDI notes. The gloves enable an expressive way of controlling sound without traditional instruments, offering a hands free interface for triggering musical notes and effects.

This project is inspired by other DIY projects using Piezzo (buzzer) as input sensors, but they are quite big and need filtering for precise velocity reading. We referred to the SNF Drumming MIDI Glove design on Instructables, which outlines the process of integrating light sensors into gloves to control sound through an Arduino Board.

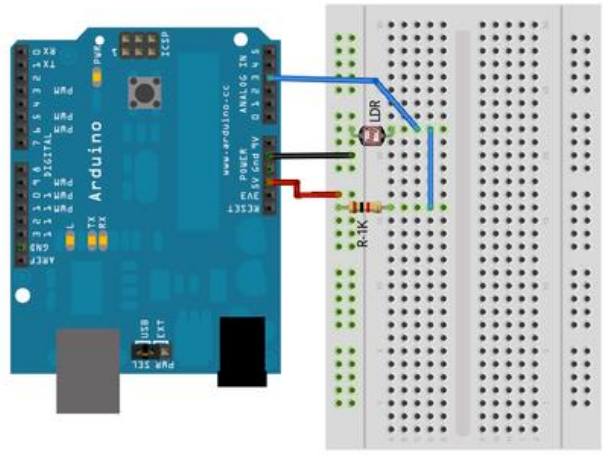
Components:

The following components were used to create the MIDI Glove:

- **Arduino Nano:** Acts as the central microcontroller to read data from the LDRs and send MIDI signals to the computer.
- **LDRs (Light Dependent Resistors):** These sensors detect changes in light intensity, which are used to trigger different MIDI signals.
- **Resistors (10kΩ):** Used to ensure proper functionality of the LDRs.
- **Glove:** To mount the LDRs on the fingertips, allowing them to detect hand movements and lighting conditions.
- **Wires:** For connecting the components.
- **USB Cable:** To connect Arduino Nano to a computer for power and MIDI data transfer.
- **Arduino IDE:** Official Arduino IDE to write and upload C code to Arduino Nano.
- **FL Studio:** Music production software to choose an instrument and input MIDI signals to produce appropriate sound.
- **LoopMIDI:** Software to create virtual loopback MIDI ports to interconnect applications on Winows (like FL Studio) that want to open hardware MIDI ports for communication.
- **Hairless MIDI Serial Bridge:** Software to connect serial devices (like Arduinos) to send and receive MIDI signals, and reroute them accordingly.
- **Styrofoam:** To fix LDRs in place on the fingertips of the glove. We used FeviKwik to stick these styrofoams, although things like an insulating tape do a decent job.

Assembly:

1. **LDR Wiring:**
   Each LDR is wired to an analog input pin on the Arduino Nano. A voltage divider inside the Arduino converts the light readings into voltage changes.
2. **Circuit Design:**
   The LDRs act as variable resistors, sending different voltage signals based on light exposure. As the hand moves and changes the amount of light on the LDRs, the voltage signals vary. The schematic is like this:



3. **Arduino Programming:**
   The Arduino Nano is programmed to interpret the LDR signals and convert them into MIDI messages. A sketch is uploaded to the Nano, mapping the LDR values to specific MIDI note or control values.
4. **MIDI output:**
   A virtual MIDI port software (LoopMIDI) is installed on the computer to interface between the Arduino and FL Studio. The Arduino sends MIDI messages through a USB connection to the computer.

Code:

There were totally two codes used for this project- one for calibration and initial recording of LDR values, and the other for actually converting these LDR values into MIDI signals.

1. **Calibration Code:**
   LDRs, by default, give out their resistance values (or a quantity that is scaled to resistance). We've inverted these resistance values to measure conductance, and also, to get an idea of the thresholds of light intensity we'd need to turn the MIDI signal ON and OFF. Here's the code:

```
// Define pins for the LDRs
const int ldrPin1 = A0;  // LDR 1 connected to analog pin A0
const int ldrPin2 = A1;  // LDR 2 connected to analog pin A1
const int ldrPin3 = A2;  // LDR 3 connected to analog pin A2
const int ldrPin4 = A3;  // LDR 4 connected to analog pin A3
// Variables to store the inverted LDR values (conductance)
int LdrValue1 = 0;
int LdrValue2 = 0;
int LdrValue3 = 0;
int LdrValue4 = 0;

// Variables to store the analog values (resistance)
int ldrValue1 = 0;
int ldrValue2 = 0;
int ldrValue3 = 0;
int ldrValue4 = 0;



void setup() {
  // Start serial communication at MIDI baud rate
  Serial.begin(115200);
}

void loop() {
```

```
  // Read the analog values from the LDRs
  ldrValue1 = analogRead(ldrPin1);
  ldrValue2 = analogRead(ldrPin2);
  ldrValue3 = analogRead(ldrPin3);
  ldrValue4 = analogRead(ldrPin4);

 // Invert the values: Higher light intensity should give lower values
  LdrValue1 = map(ldrValue1, 0, 1023, 1023, 0);
  LdrValue2 = map(ldrValue2, 0, 1023, 1023, 0);
  LdrValue3 = map(ldrValue3, 0, 1023, 1023, 0);
  LdrValue4 = map(ldrValue4, 0, 1023, 1023, 0);

  // Print the inverted values to the Serial Monitor
  Serial.print("Inverted LDR 1 Value: ");
  Serial.println(LdrValue1);

  Serial.print("Inverted LDR 2 Value: ");
  Serial.println(LdrValue2);

  Serial.print("Inverted LDR 3 Value: ");
  Serial.println(LdrValue3);

  Serial.print("Inverted LDR 4 Value: ");
  Serial.println(LdrValue4);




  // Small delay for stability
  delay(100);
}
```

We define variables to store LDR values and Inverted LDR values, and set them initially to 0. The baud rate is set to match with MIDI. The map() function defines the inverse relation between Inverted LDR values and actual LDR values. Finally, the 4 inverted LDR values are printed out to the Serial Monitor. A time delay of 100ms is set for stability and ease of noting LDR values.

2. **LDR Reading to MIDI Signal:**
   This is the actual working code for the MIDI glove:

```
// Define pins for the LDRs
const int ldrPin1 = A0;  // LDR 1 connected to analog pin A0
const int ldrPin2 = A1;  // LDR 2 connected to analog pin A1
const int ldrPin3 = A2;  // LDR 3 connected to analog pin A2
const int ldrPin4 = A3;  // LDR 4 connected to analog pin A3

// Variables to store the analog values
int ldrValue1 = 0;
int ldrValue2 = 0;
int ldrValue3 = 0;
int ldrValue4 = 0;

// Variables to store the MIDI note values (0-127)
int midiNote1 = 60;  // MIDI note for LDR 1 (Middle C)
```

```
int midiNote2 = 62;  // MIDI note for LDR 2 (D)
int midiNote3 = 64;  // MIDI note for LDR 3 (E)
int midiNote4 = 67;  // MIDI note for LDR 4 (G)

// Thresholds for turning notes on and off (adjust as needed)
int noteOnThreshold = 512;   // Light intensity to turn the note ON
int noteOffThreshold = 500;  // Light intensity to turn the note OFF

// State variables to track if the note is currently on or off
bool note1On = false;
bool note2On = false;
bool note3On = false;
bool note4On = false;

void setup() {
  // Start serial communication at MIDI baud rate
  Serial.begin(115200);
}

void loop() {
  // Read the analog values from the LDRs
  ldrValue1 = analogRead(ldrPin1);
  ldrValue2 = analogRead(ldrPin2);
  ldrValue3 = analogRead(ldrPin3);
  ldrValue4 = analogRead(ldrPin4);

  // Process LDR 1
  if (ldrValue1 > noteOnThreshold && !note1On) {
    sendMIDINoteOn(midiNote1, 127);  // Send MIDI Note On with full velocity (127)
    note1On = true;  // Mark note as ON
  } else if (ldrValue1 < noteOffThreshold && note1On) {
    sendMIDINoteOff(midiNote1, 0);  // Send MIDI Note Off
    note1On = false;  // Mark note as OFF
  }

  // Process LDR 2
  if (ldrValue2 > noteOnThreshold && !note2On) {
    sendMIDINoteOn(midiNote2, 127);
    note2On = true;
  } else if (ldrValue2 < noteOffThreshold && note2On) {
    sendMIDINoteOff(midiNote2, 0);
    note2On = false;
  }

  // Process LDR 3
  if (ldrValue3 > noteOnThreshold && !note3On) {
    sendMIDINoteOn(midiNote3, 127);
    note3On = true;
  } else if (ldrValue3 < noteOffThreshold && note3On) {
    sendMIDINoteOff(midiNote3, 0);
    note3On = false;
  }

  // Process LDR 4
```

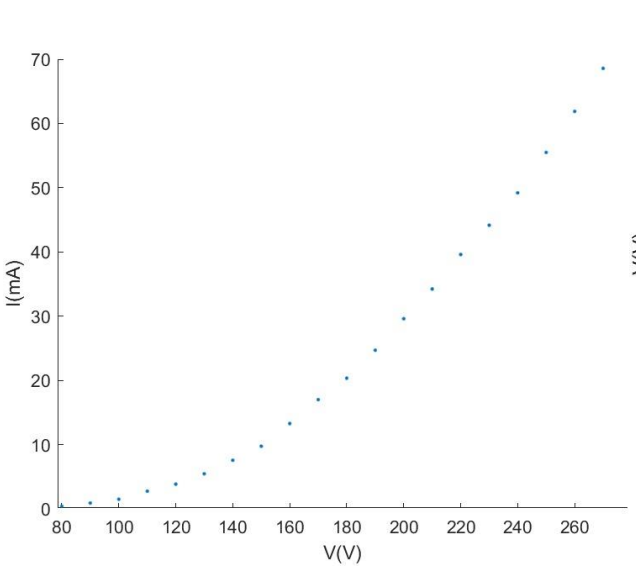
```
  if (ldrValue4 > noteOnThreshold && !note4On) {
    sendMIDINoteOn(midiNote4, 127);
    note4On = true;
  } else if (ldrValue4 < noteOffThreshold && note4On) {
    sendMIDINoteOff(midiNote4, 0);
    note4On = false;
  }

  // Small delay for stability
  delay(10);
}

// Function to send a MIDI Note On message
void sendMIDINoteOn(byte note, byte velocity) {
  Serial.write(0x90);    // MIDI Note On message, Channel 1 (0x90)
  Serial.write(note);    // Note to play (e.g., 60 is middle C)
  Serial.write(velocity);  // Velocity (0-127)
}

// Function to send a MIDI Note Off message
void sendMIDINoteOff(byte note, byte velocity) {
  Serial.write(0x80);    // MIDI Note Off message, Channel 1 (0x80)
  Serial.write(note);    // Note to stop (e.g., 60 is middle C)
  Serial.write(velocity);  // Velocity (0 to turn off)
}
```

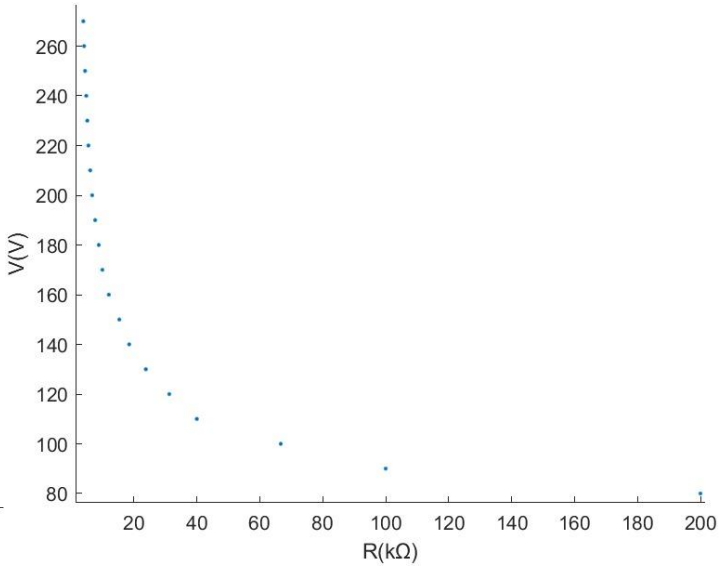
Here, we start off by defining variables to store LDR values (resistance) and setting them initially to 0. We then define the MIDI note we want to assign to each of the LDRs. Based on our previous calibration data, we set an ON and OFF threshold for different values of light intensity (ambient light for our case was around 400 units), then ran it through an if-else condition to change our Boolean variable to ON and OFF respectively. For both, we define two functions to send out the MIDI signal, with parameters for channel, note, and amplitude. We also add a small time delay of 10ms for stability and smoothness.

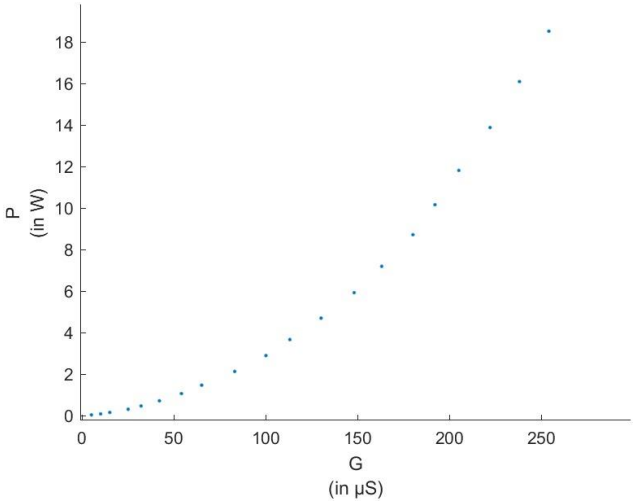
Testing and Calibration:

While testing, adjustments are made to the light intensity threshold to ensure that the gloves respond appropriately to hand movements and environmental lighting. The values of the resistors connected to the LDRs are fine-tuned to get optimal sensitivity. Calibration is done to prevent false triggering under normal lighting conditions and to ensure that only specific gestures (like covering or exposing the LDRs with the hand) would result in MIDI notes being sent.

Moreover, we used this opportunity to verify the relation of our result using the apparatus for the photoelectric effect. We plotted for the voltage of the light source vs the resistance and more importantly, power (proportional to intensity) vs conductance. The graphs are attached here:







We can verify here that the Power increases with the conductance.

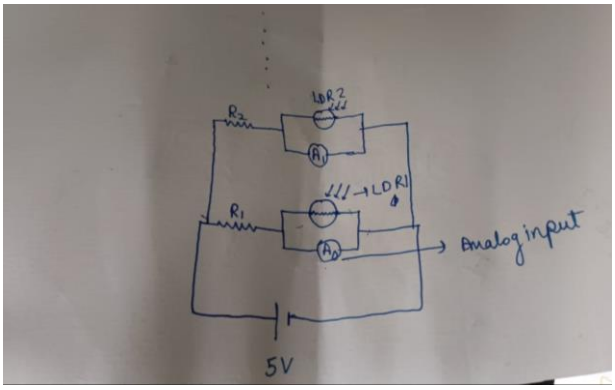
Functionality:

1. **Working of the LDR:** A photoresistor, also known as a Light Dependent Resistor (LDR), operates based on the photoconductive effect in semiconductor materials. Our LDRs were made of Cadmium sulfide (CdS), which is a typical material for photoresistors. The maximum operable temperature was 800 ˚C. It had a cross-section area of 5mm x 5mm, with a dark resistance of around 80 MΩ.
    - Semiconductors have a specific band gap, which is the energy difference between the valence and conduction bands. In the absence of light, few electrons have enough energy to jump from the valence to the conduction band, so the material has high resistance.
    - When light shines on the semiconductor surface of a photoresistor, photons are absorbed by the material. It the photon has higher energy than the band gap, it can excite an electron from the valence to the conduction band. This process creates electron-hole pairs in the conduction and valence bands respectively.
    - Excited electrons and resulting holes lead to an increase in the available number of charge carriers, increasing the conductance. In darkness, the LDR can have very high resistance (in MΩ), while in bright light, it can drop to just a few kΩ.
    - The relationship between light intensity and resistance is typically nonlinear. The resistance doesn't decrease linearly with increasing light; instead, it often follows an exponential or logarithmic relationship.
    - Like most semiconductor devices, the conductivity of an LDR is temperature-dependent. At higher temperatures, some electrons gain enough thermal energy to jump to the conduction band even

without light, slightly decreasing the resistance. This temperature dependence is usually small compared to the effect of light, but it can be a factor in precision applications.

- Due to its ability to vary resistance with light, the photoresistor is used in a wide range of light-sensitive applications, such as automatic lighting, light meters in cameras, alarm systems, etc.

2. **Working of the setup:**
    - When light falls on the LDR, it generates electron-hole pairs in the semiconductor material, lowering its resistance proportionally to the light intensity. The LDR is connected to the Arduino as part of a voltage divider circuit (often with a fixed resistor), where the LDR and the fixed resistor divide the input voltage based on the LDR's resistance. This change in resistance alters the voltage at the point where the LDR connects to the Arduino's analog input pin.
    - The Arduino reads the varying voltage from the LDR circuit as an analog signal (typically on pins A0-A5 for analog inputs). The analog signal is read using `analogRead()`, which converts the voltage level to a digital value ranging from 0 to 1023. For example, in bright light, the voltage across the LDR may be low, resulting in a high analog value (close to 1023), while in darkness, the analog reading may be low (close to 0).
    - In the Arduino code, the LDR reading (0–1023) is processed to create a MIDI-compatible output. If you want to invert the value so that higher light intensity results in a lower MIDI value, you can use the `map()` function to scale the range inversely. The final processed value will likely be in the range of 0–127 (the standard MIDI value range).
    - Once the Arduino has mapped the LDR's analog reading to the MIDI range, it sends the data over serial to the Hairless MIDI-Serial Bridge. This is done using `Serial.write()` commands in the Arduino code, which writes bytes of data corresponding to a MIDI message. MIDI data often includes:
        - MIDI Channel: Specifies the channel (1–16) on which the message is sent.
        - MIDI Command: Defines the type of MIDI message (e.g., Note On, Control Change).
        - MIDI Value: Represents values such as note velocity or controller position
    - The Hairless MIDI-Serial Bridge is software that translates the serial data received from the Arduino into standard MIDI messages. It reads the raw bytes sent by the Arduino, interprets them as MIDI commands, and forwards them to a Virtual MIDI Port. This virtual port emulates a MIDI connection, allowing applications like FL Studio to receive MIDI messages without physical MIDI hardware.
    - FL Studio responds to the MIDI messages in a way that reflects the change in light detected by the LDR: Gradual changes in light (like moving a hand in front of the LDR) produce smooth changes in the controlled parameter. Sharp changes in light intensity produce quick parameter changes, like sudden volume increases or filter cutoffs. If multiple LDRs are used, each can control a different parameter by sending MIDI on separate channels or using different Control Change numbers.
    - Here's the practical circuit diagram for the project:



Troubleshooting:

1. We faced most problems in understanding the workings of FL studio. This can be figured out by trying different instruments and by watching tutorials online.

2. Another area of difficulty can be setting the thresholds. While both the ON and the OFF threshold should be higher than ambient light, the difference in both of them should be narrow or the instrument will not work for the given code. We also tried setting both thresholds equal, and this gave us the best results. Interestingly, keeping the OFF threshold higher than the ON threshold produces a reverb effect in the instrument, which can be used for musical purposes.

Utilities:

While this project is a mere prototype, it can be improved and modified in many ways for several applications:

1. While MIDI is one of the applications of LDRs, we can also utilise them for sign language translation. LDRs can be arranged at specific locations on the hand and their collective intensity variation and orientation can be used to translate sign language. This can be used by disabled people and they can mingle better in society.
2. MIDI gloves can be used by musicians or performers in live settings to control sound effects, lighting, and visuals, all through mere hand movements, allowing for more expressive and dynamic performances.
3. They can provide physical interactions within virtual reality environments. This has applications in VR art and gaming.
4. For people with disabilities or limited mobility, MIDI gloves can offer an accessible way to interact with music software. For example, someone with limited hand dexterity could use a few basic gestures to trigger complex sounds or control music production elements. In case a musician loses a dominant hand for, say, a guitar, the non-dominant hand can be used with the glove with ease to improve performance.
5. They can be used for interactive music learning for beginners and can be a non-traditional approach to music education
6. These can also be used by people with neurodivergent disorders like ADHD to fidget around or increase focus, or to help with situations of anxiety, or motor skill development.

Conclusion:

The project, overall, uses LDRs and MIDI and might appear fun, but has very practical applications and relevance in today's society. It also has several advantages over a photodiode, in terms of less power and more adaptability. While this project taught me a lot about the workings of an Arduino board, it also helped put ideas to practicality, while enabling me to explore further improvements for it.