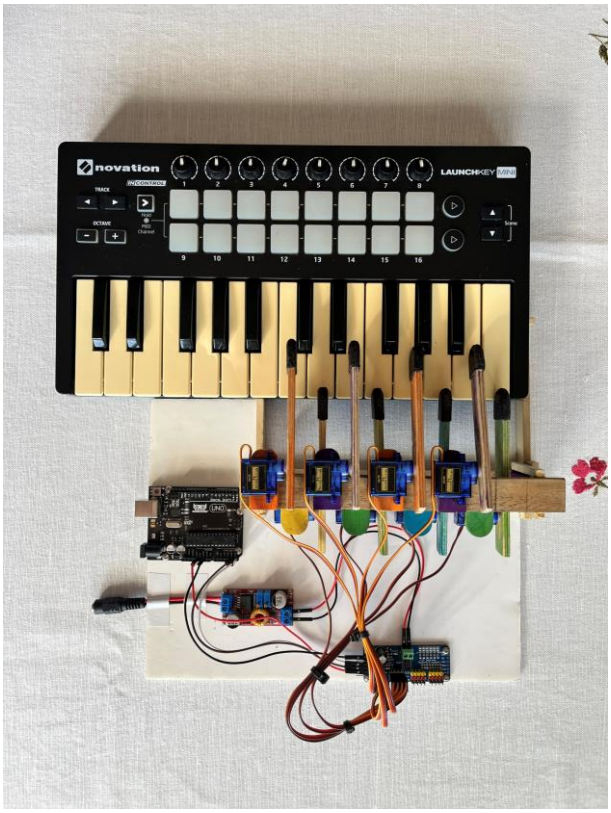


MiniPlayer

Роботски систем за свирење на синтисајзер

YouTube link: https://youtu.be/meCeByRlAr0

Вовед

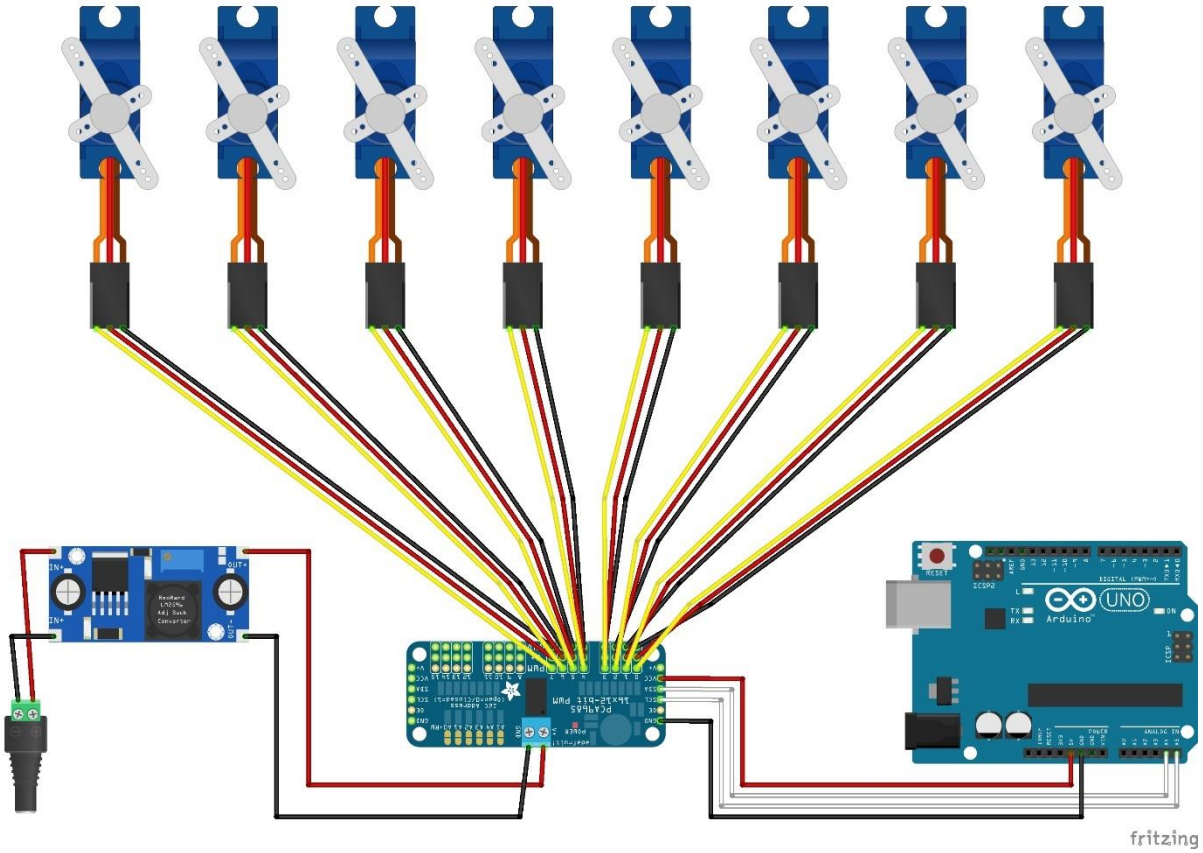
Овој систем конвертира аудио датотеки (MP3) во MIDI формат, ги обработува MIDI нотите за да се вклопат во одреден опсег на октава и ги испраќа податоците до рачно-изработен роботски систем контролиран од Arduino, опремен со серво мотори за репродукција на нотите, притискајќи на синтисајзер.

Изработиле:

Огнен Иванов | 211002
Нина Цветковска | 211089
Петар Трајкоски | 211214

Содржина

Компоненти, хардвер и процес на изработка

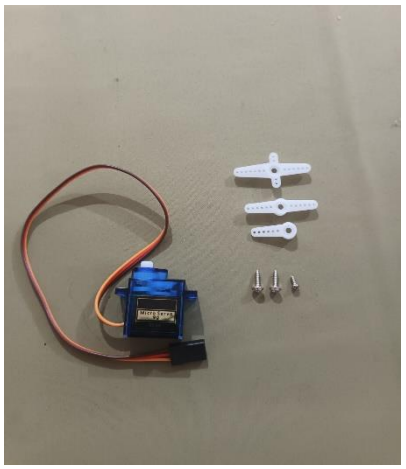


Дијаграм на поврзаност на електронските компоненти кои ги содржи роботскиот систем

1. Компоненти

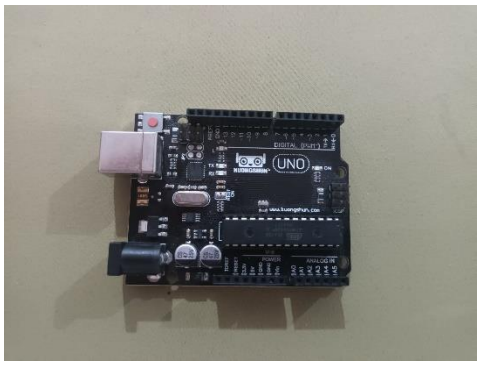
За изработка на овој роботски систем, нам ни беа потребни: синтисајзер, 8 серво мотори кои го симулираат движењето на прстите, PWM Driver на кој се поврзани моторите, напојување, step down / buck converter за подесување на напонот и контролна Arduino UNO плочка која ја регулира работата на системот.

1.1 Серво мотори



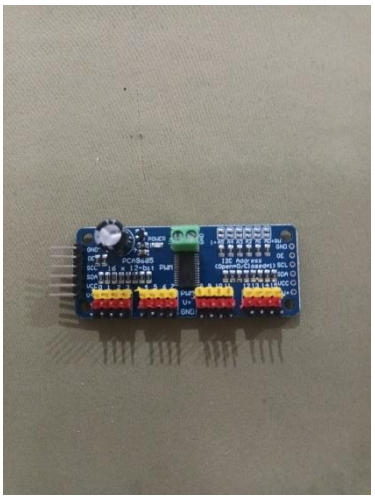
Серво моторите кои ги употребивме се модел: SG90, кои работат на напон од 5V и при најголем напор користат струја од 0.5A. Нашата цел беше да ги поставиме на фиксна позиција за да можеме да отсвириме точно една октава (8 ноти), па затоа бројот на серво мотори е 8. Тоа значи дека соодветно, напојувањето треба да е со напон од 5V, и струја од минимум 4A (8 x 0.5A). Овие серво мотори користат пластични запчаници и не се многу прецизни, но беа доволни да ја завршат работата.

1.2 Arduino UNO



Главната контролна плочка, т.е микроконтролер кој го употребивме во овој проект е Arduino UNO. Таа беше солиден избор за контрола на серво моторите и истата ја поседувавме, па не требаше да купуваме друга. Негативна страна на плочката е меморијата која ја поседува, од што зависеше и нашата имплементација, но делумно и процесирачката моќ, која ни створи предизвик во иднина.

1.3 PWM Driver – PCA9685



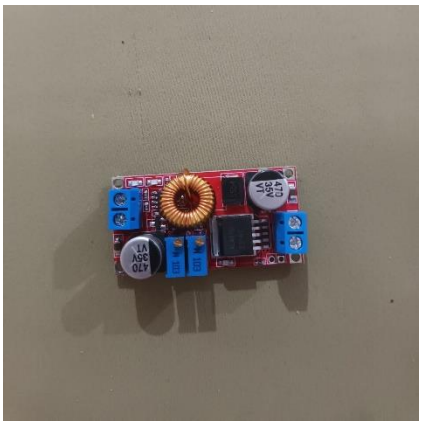
Со оглед на тоа што Arduino плочката нема доволно PWM (~) дигитални пинови за 8 серво мотори, нам ни беше потребен контролер/драјвер на кој можевме да ги поврземе сите потребни мотори. PCA9685 е плочка / PWM driver, кој има 16, 12-bit канали, на кои можат да се приклучат 16 серво мотори, а има и свој часовник (clock). Плочката се поврзува со ардуиното преку 4 пинови: VCC пин кој носи 5V од ардуиното до плочката, GND пин за да го затвори заземјувањето, SCL или system clock пин кој се поврзува на аналоген пин за ардуиното (A5) и SDA или system data пин кој е исто така поврзан со аналоген пин (A4). PCA9685 има и влез за посебно напојување кое ги придвижува моторите (V+ и GND), а во нашиот случај ни беа потребни 5V и минимум 4A за 8 серво мотори.

1.4 Напојување – AC/DC Adapter

Како што веќе знаеме, за придвижување на серво моторите, нам ни беше потребно напојување од 5V и минимум 4A. Бидејќи роботскиот систем е статичен и немаме мобилност, ние ги избегнавме батериите како извор на струја и се одлучивме да користиме AC/DC адаптер. Сите адаптери од 5V кои ги имаше на пазарот беа до 3A и не ги исполнуваа нашите критериуми, па затоа се одлучивме да земеме адаптер од 12V и 6A за поволна цена. Тука предизвикот не беше струјата, поголема струја од 4A не му смета на системот,

дури е препорачано да има простор од минималната граница. Напротив, предизвикот беше напонот од 12V, кој моравме да го конвертираме во 5V, со кој работат сите други компоненти. Тој проблем го решивме со step down / buck converter.

1.5 Step Down / Buck Converter - XL4015

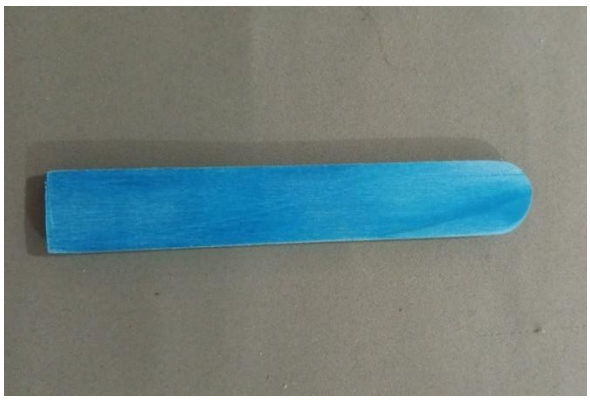


Ова компонента/плочка има за цел да го намали влезниот напон, до посакуван напон на излез, кој корисникот го подесува со потенциометар, се додека ја запазува моќмноста на влезното напојување. Ова значи дека доколку нашето напојување е од 12V и 6A (моќност: 72W | Омов Закон: P=U*I), и сакаме на излез од step down плочката да имаме напон од 5V, излезната струја ќе зависи од тоа да се запази моќноста и таа теоретски би изнесувала 14.4A (I=P/U), но карактеристиките на плочката дозволуваат излез до 5A максимум што нас ни одговараше на потребните 4. Така на излез од оваа компонента и влез во PCA9685 драјверот, ние добивме струја од 5A и напон од 5V.

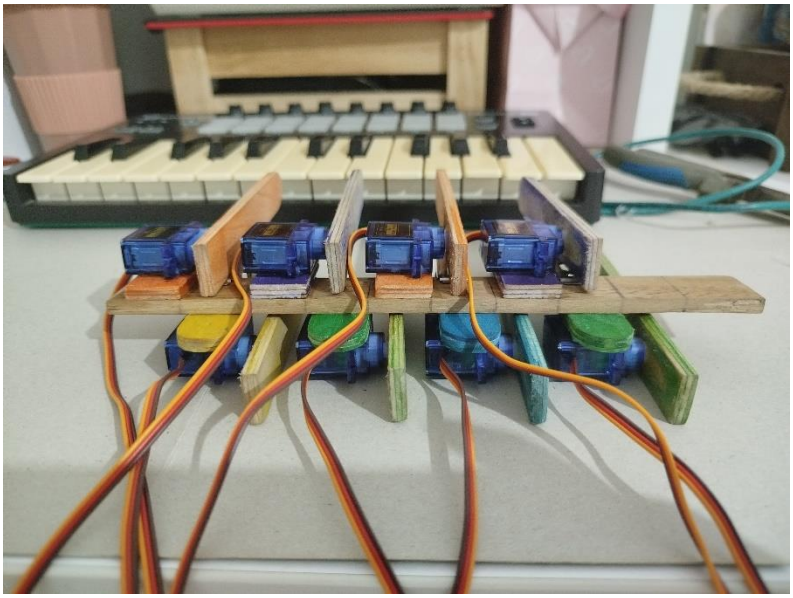
1.6 Синтисајзер - Novation Launchkey Mini MK2

2. Процес на изработка

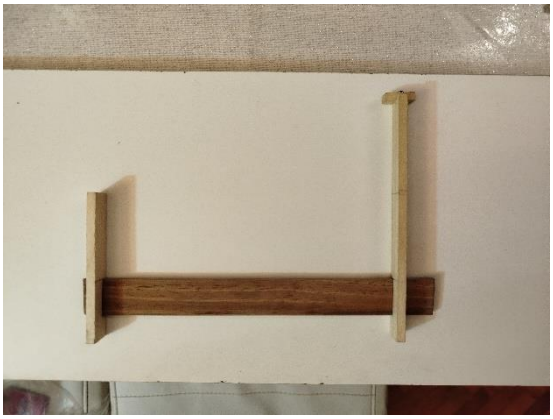
Откако ги набавивме сите потребни компоненти, следуваше изработка и конструкција на роботскиот систем. Целиот роботки систем е рачно изработен и материјалите кои ги користевме при изработка, не ги купувавме, туку ги наоѓавме по домовите и сл. Започнавме со изработката на самите прсти кои ги притискаат нотите на синтисајзерот. Тие се изработени од дрво, поточно од 3 залепени стапчиња, налик оние кои ги користат докторите на преглед или оние кои ги сретнуваме на сладолед. Залепивме 3 такви стапчиња едно врз друго со бел лепак за дрво, ги притиснавме со штипки и ги оставивме да се сушат нешто повеќе од 24 часа. Искористивме вкупно 24 стапчиња за да изработиме 8 прсти. Откако се исуши убаво лепакот, сите 8 стапчиња ги измазнивме, т.е исполиравме со брусна хартија и ги пресековме со скалпер, грубо на размер 4:1, каде подолгиот дел беше посакуваната должина за прстите, а пократкиот дел го зачувавме за друга примена понатака во изработката. Следно требаше да најдеме начин да ги прицврстиме овие прсти за серво моторите. Со самите серво мотори стигнаа и 3 различни, пластични додатоци кои се ставаат директно на оската, па одлучивме да издлабиме дупки, длабоки околу 2 слоја, во форма на еден од додатоците, за сите 8 серво мотори. Белите пластични додатоци ги залепивме со супер-лепак, па откако се исушија, ги покривме и залепивме со топен силиконски лепак. На страната на прстите која допира директно на синтисајзерот,



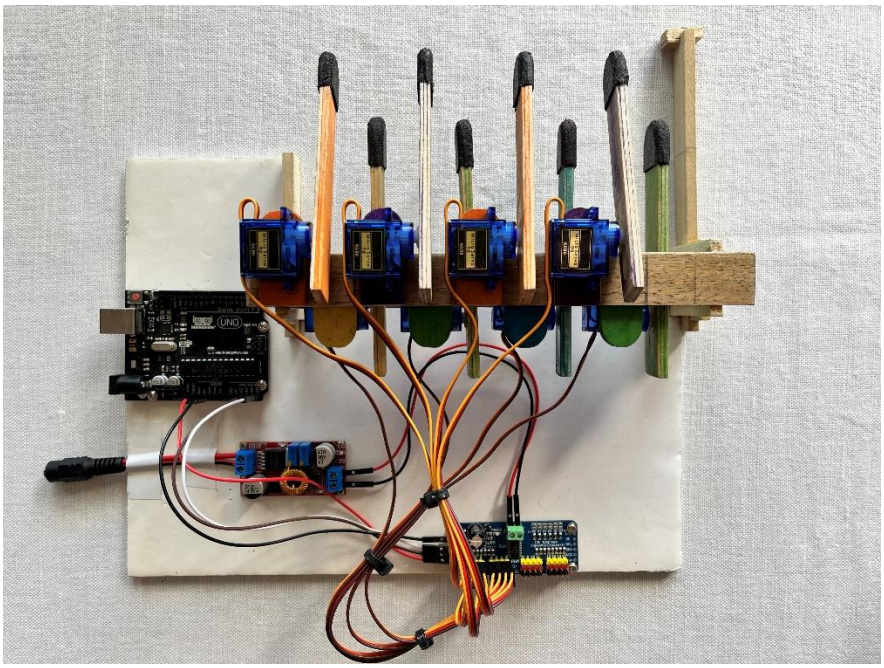
залепивме вештачка кожа како мек слој за поубав допир со подлогата. Следно, ги залепивме серво моторите на помалите, исечени делови, сочувани од претходно, за да осигураме непречено движење во просторот на работа на прстите. Потоа ја измеривме ширината на нотите на синтисајзерот за да знаеме на кое растојание да ги фиксираме моторите. Еден мотор, заедно со закачен прст, т.е дрвце, имаше ширина точно колку 2 ноти на синтисајзерот. Ова значеше дека нема да можеме да ги ставиме сите 8 серво мотори,

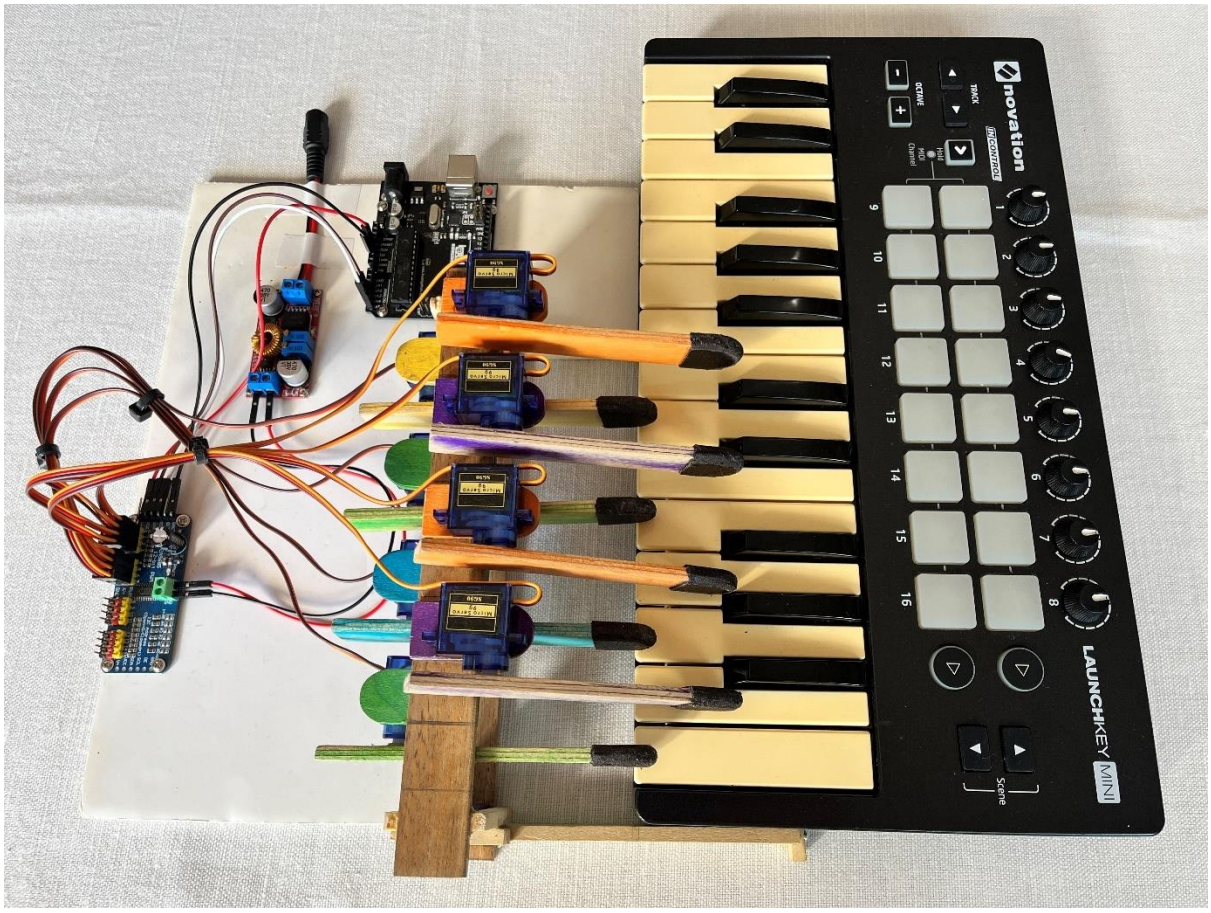


едни до други, на една основа, туку дека ќе треба да групираме по 4 серво мотори на две различни основи/висини. Ние ова го решивме така што искористивме две страни од една дрвена основа, но тука дојдовме до друг предизвик. Оската на движење на горните и долните мотори мора да е различна, за основата на која се залепени да не претставува пречка при нивното движење. Со други зборови, оската на горните мотори мора да е испакната од работ околу кој тие ротираат, на напред, за основата да не пречи при ротација околу таа оска. Истото важи и за долните мотори, т.е оската на движење мора да е е испакната позади работ околу кој тие ротираат. Тоа е причината зошто горните мотори се напред, а долните назад, релативно на основата на која ги залепивме. Во оваа фаза, роботот доби некаков првичен изглед за тоа како би личел крајниот производ. Следниот предизвик беше да направиме конструкција, со која ќе ја фискираме висината и растојанието на моторите до синтисајзерот. Започнавме од долниот дел на конструкцијата, кој го дизајниравме да се закачува/фиксира за вдлабнатините, т.е дизајнот на самиот синтисајзер. Потоа додадовме столбови за моторите да бидат на посакуваната висина. На столбовите ја залепивме даската со серво моторите и добивме една целина, т.е целосна конструкција која ги позиционира прстите до синтисајзерот. Следно, исековме стиропорна плоча во форма на сталажата и ја залепивме за истата, на која можевме да ги закачиме компонентите и да ги фиксираме со шрафови.



Во следниот чекор, пред да ги поврземе компонентите, требаше да го штелуваме step down / buck converter на 5V излезна струја. Овој уред имаше дефект, по што набавивме нов, од истиот модел. Вториот конвертер беше исправен и со помош на мултиметар и свртување на потенциометарот, успеавме да ја штелуваме излезната струја на 5V. Откако ги поврзавме сите компоненти со посакуваната струја, пробавме да ги придвижиме серво моторите преку библиотеката за PCA9685 драјверот (Adafruit library). Овој обид беше неуспешен, по што решивме да го детектираме проблемот со помош на мултиметар. По сите мерења, сите компоненти ја добиваа посакуваната струја, но PCA9685 драјверот на излез за серво моторите испраќаше значително мала струја, која не беше доволна за нивно придвижување. Заклучокот од ова беше дека PCA9685 драјверот е неисправен, по што отидовме и го заменивме за нов, но и новата плочка ги даде истите резултати. Повторно ги направивме сиите можни мерења и тестови за да го детектираме проблемот, и заклучокот беше ист – неисправна компонента. По вториот неуспешен обид, PCA9685 драјвер нѝ позајми колега кој го нарачал од Интернет, и со истото поврзување, истото напојување и истиот код како и претходно, овој пат се работеше како што треба и можевме да преминеме на софтверската имплементација. За овој процес на изработка на хардверските делови и набавувањето и поврзувањето на електронските компоненти, нам ни беа потребни од 3 до 4 недели и сите компоненти не чинеа околу 4000 денари.





MIDI to Arduino: Преглед на работниот тек

1. Python апликација

Кодот дефинира апликација користејќи PyQt5 (или PySide) која обработува MP3-датотека, ја претвора во MIDI-датотека, овозможува понатамошни трансформации на ноти и комуникација со Arduino плоча.

1.1 Main Structure

Апликацијата се состои од три главни елементи:

1. **MP3ToMIDIApp class**: Го дефинира корисничкиот интерфејс и се справува со интеракциите за избор на датотеки, директориум за излез и статус за време на процесот на конверзија од MP3 во MIDI.
2. **Функции за обработка на MIDI**: Вклучува различни функции за трансформација на MIDI-датотека, како што се транспонирање на ноти во одреден опсег на октава, отстранување на ноти со повишувалка(диез) и справување на ноти што се преклопуваат.
3. **WorkerThread class**: Нишка за справување со обработката на MIDI-датотека додека се ажурира интерфејсот со пораки за статус и препраќање на обработените ноти до Arduino плочата.

1.2 Main Application Class (MP3ToMIDIApp)

Оваа класа го претставува главниот прозорец на апликацијата. Ги дефинира распоредот и интеракциите на апликацијата, ракувањето со внесување/излез на датотека и управување со корисничкиот интерфејс.

```
class MP3ToMIDIApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("MidiPlayer")
        self.resize(720, 480)
        self.setAcceptDrops(True)

        self.central_widget = QWidget()
        self.setCentralWidget(self.central_widget)
        self.layout = QVBoxLayout()
        self.central_widget.setLayout(self.layout)
```

1.2.1 Клучни компоненти на MP3ToMIDIApp

- *Елементи на корисничкиот интерфејс:*
  - ❖ QLabel: Се користи за прикажување информации на корисникот, како на пример каде да се влечат и пуштаат датотеки или директориуми.

```
        # Labels and buttons
        self.input_label = QLabel("Drag & Drop MP3 File Here", self)
        self.input_label.setAlignment(Qt.AlignCenter)
        self.input_label.setStyleSheet(self.get_default_stylesheet())
        self.layout.addWidget(self.input_label)

        self.output_label = QLabel("Drag & Drop Output Directory Here", self)
        self.output_label.setAlignment(Qt.AlignCenter)
        self.output_label.setStyleSheet(self.get_default_stylesheet())
        self.layout.addWidget(self.output_label)
```

  - ❖ QPushButton: Копчиња кои им овозможуваат на корисниците да изберат MP3-датотека, да изберат директориум за излез или да го започнат процесот на конверзија.

```
        self.select_input_btn = QPushButton('Select MP3 File', self)
        self.select_input_btn.clicked.connect(self.select_input_file)
        self.layout.addWidget(self.select_input_btn)

        self.select_output_btn = QPushButton('Select Output Directory', self)
        self.select_output_btn.clicked.connect(self.select_output_directory)
        self.layout.addWidget(self.select_output_btn)

        self.process_button = QPushButton('Convert and Send', self)
        self.process_button.clicked.connect(self.start_conversion)
        self.layout.addWidget(self.process_button)

        self.process_again_button = QPushButton('Process Again', self)
        self.process_again_button.clicked.connect(self.process_again)
        self.process_again_button.hide()  # Hide it initially
        self.layout.addWidget(self.process_again_button)
```

  - ❖ QProgressBar: Лента за напредок што го прикажува статусот на процесот за конверзија.

```
    # Progress Bar
    self.progress_bar = QProgressBar(self)
    self.progress_bar.setAlignment(Qt.AlignCenter)
    self.layout.addWidget(self.progress_bar)
    self.progress_bar.hide()
```

- *Ракување со влезни и излезни датотеки/директориуми:*
  - ❖ Корисниците можат да влечат и испуштаат MP3-датотеки и директориуми за излез во интерфејсот или да ги користат копчињата за рачно да ги изберат. Етикетите даваат инструкции и визуелни повратни информации.

```
    self.input_file = None
    self.output_dir = None
```

- *Настани со влечење и испуштање:*
  - ❖ dragEnterEvent and dropEvent: Функции за „drag-and-drop“ за прифаќање MP3-датотеки и директориуми и соодветно ажурирајќи ги етикетите.

```
def dragEnterEvent(self, event: QDragEnterEvent):
    if event.mimeData().hasUrls():
        event.accept()
        self.input_label.setStyleSheet(self.get_hover_stylesheet())
        self.output_label.setStyleSheet(self.get_hover_stylesheet())
    else:
        event.ignore()

def dropEvent(self, event: QDropEvent):
    urls = event.mimeData().urls()
    if urls:
        file_path = urls[0].toLocalFile()
        if os.path.isfile(file_path) and file_path.lower().endswith('.mp3'):
            self.input_file = file_path
            self.input_label.setText(f"Selected MP3: {file_path}")
        elif os.path.isdir(file_path):
            self.output_dir = file_path
            self.output_label.setText(f"Output Directory: {file_path}")

    # Restore default appearance after drop
    self.input_label.setStyleSheet(self.get_default_stylesheet())
    self.output_label.setStyleSheet(self.get_default_stylesheet())
```

- *Останати методи на класата:*
  - ❖ **select_input_file**: Отвара „file dialog“ за избор на MP3-датотека.

```
1 usage
def select_input_file(self):
    self.input_file, _ = QFileDialog.getOpenFileName(self, "Select MP3 File", "", "MP3 Files (*.mp3)")
    if self.input_file:
        self.input_label.setText(f"Selected MP3: {self.input_file}")
```

  - ❖ **select_output_directory**: Отвара „directory dialog“ за избор на излезен директориум.

```
1 usage
def select_output_directory(self):
    self.output_dir = QFileDialog.getExistingDirectory(self, "Select Output Directory")
    if self.output_dir:
        self.output_label.setText(f"Output Directory: {self.output_dir}")
```

- **start_conversion**: Го започнува процесот на конверзија кога корисникот ќе го притисне копчето „Convert and Send“. Исто така, крие и други елементи на интерфејсот за корисникот да се фокусира на лентата за напредок за време на обработката.

```
def start_conversion(self):
    if not self.input_file or not self.output_dir:
        print("Please select both MP3 file and output directory.")
        return

    # Hide other UI elements
    self.progress_bar.show()
    self.input_label.hide()
    self.output_label.hide()
    self.select_input_btn.hide()
    self.select_output_btn.hide()
    self.process_button.hide()

    self.progress_bar.setValue(0)
    self.progress_bar.show()

    self.worker = WorkerThread(self.input_file, self.output_dir)
    self.worker.update_message.connect(self.show_message)
    self.worker.progress.connect(self.update_progress)
    self.worker.start()
```

- **process_again**: Го ресетира корисничкиот интерфејс за уште еден круг на обработка по завршувањето на првата конверзија.

```
1 usage
def process_again(self):
    self.input_label.setText("Drag & Drop MP3 File Here")
    self.output_label.setText("Drag & Drop Output Directory Here")
    self.input_file = None
    self.output_dir = None

    # Show all original UI elements again
    self.input_label.show()
    self.output_label.show()
    self.select_input_btn.show()
    self.select_output_btn.show()
    self.process_button.show()
    self.progress_bar.hide()
    self.process_again_button.hide()  # Hide process again button
```

- **update_progress**: Ја ажурира лентата за напредок.

```
1 usage
def update_progress(self, value):
    self.progress_bar.setValue(value)
    if value == 100:
        # Show all elements again after processing is done
        self.process_again_button.show()  # Show process again button
```

- **show_message**: Ја ажурира етикетата за прикажување пораки за време на процесот на конверзија.

```
1 usage
def show_message(self, message):
    self.input_label.setText(message)
    self.input_label.repaint()  # Update the label immediately
```

1.3 Функции за обработка на MIDI

Овие функции се одвоени од класата MP3ToMIDIApp и се справуваат со обработка на MIDI-датотеки по конверзија од MP3. Тие вклучуваат операции за менување на музичката содржина, како што се префрлање ноти во одреден опсег на октава, отстранување на ноти со повишувалка(диез) или избегнување на преклопувачки ноти со помош на „music21“ библиотеката.

1.3.1 Клучни MIDI функции

1. *transpose_to_octave(score, min_note='C4', max_note='C5'):*
   - ❖ Оваа функција ги транспонира нотите во MIDI-датотеката за да се вклопат во одреден опсег на октава (C4 до C5). Нотите надвор од овој опсег или се поместуваат нагоре или надолу за октава за да се задржат во границите. Нотите под минималната нота (C4) се транспонирани нагоре за една октава (12 полутони). Нотите над максималната нота (C5) се транспонираат надолу за една октава (-12 полутони). Со акордите се постапува слично со прилагодување на секој тон во акордот.

```
def transpose_to_octave(score, min_note='C4', max_note='C5'):
    lower_pitch = note.Pitch(min_note)
    upper_pitch = note.Pitch(max_note)
    original_notes = []

    for element in score.flat.notesAndRests:
        if isinstance(element, note.Note):
            if element.pitch < lower_pitch:
                # Transpose up to the nearest note within the octave range
                transposition = 12  # One octave up
                element.pitch = element.pitch.transpose(transposition)
            elif element.pitch > upper_pitch:
                # Transpose down to the nearest note within the octave range
                transposition = -12  # One octave down
                element.pitch = element.pitch.transpose(transposition)
            else:
                original_notes.append(element)  # Keep original notes
        elif isinstance(element, chord.Chord):
            new_pitches = []
            for pitch in element.pitches:
                if pitch < lower_pitch:
                    pitch = pitch.transpose('P8')  # Transpose up
                elif pitch > upper_pitch:
                    pitch = pitch.transpose('-P8')  # Transpose down
                else:
                    original_notes.append(note.Note(pitch))  # Keep original notes
                new_pitches.append(pitch)  # Always add the transposed pitch
            element.pitches = new_pitches

    return original_notes
```

2. *remove_repeating_chords(score):*
   - ❖ Оваа функција ги отстранува последователните повторувачки акорди за да се избегне вишок. Ги споредува акордите врз основа на нивните MIDI звучни вредности и ги задржува само единствените во низа.

```python
def remove_repeating_chords(score):
    """Remove consecutive repeating chords."""
    unique_chords = []
    prev_chord = None

    for element in score.flat.notesAndRests:
        if isinstance(element, chord.Chord):
            chord_pitches = sorted([p.midi for p in element.pitches])  # Use sorted MIDI values to compare
            if chord_pitches != prev_chord:
                unique_chords.append(element)
                prev_chord = chord_pitches
        else:
            unique_chords.append(element)

    return stream.Stream(unique_chords)
```

3. *remove_sharps(score):*
   - Оваа функција ги отстранува сите ноти со повишувалка. Итерира низ нотите и акордите и ги отстранува елементите што содржат „#“. За акорди, ги отстранува поединечните такви ноти, а ако акордот се испразни по отстранувањето, целиот акорд се брише.

```python
def remove_sharps(score):
    """Remove all sharp notes from the score."""
    notes_to_remove = []

    print("Identifying sharp notes to remove...")
    for element in score.flat.notesAndRests:
        if isinstance(element, note.Note):
            if '#' in element.nameWithOctave:
                notes_to_remove.append(element)
                print(f"Found sharp note: {element.nameWithOctave}")
        elif isinstance(element, chord.Chord):
            # Remove pitches from chord that are sharps
            new_pitches = [pitch for pitch in element.pitches if '#' not in pitch.nameWithOctave]
            if len(new_pitches) != len(element.pitches):
                chord_pitches = ', '.join(p.nameWithOctave for p in element.pitches)
                print(f"Chord {chord_pitches} had sharps and is being modified.")
            element.pitches = new_pitches

            # If no pitches remain in the chord, mark it for removal
            if not element.pitches:
                notes_to_remove.append(element)
                print(f"Chord {chord_pitches} has no remaining pitches and will be removed.")
```

```python
    # Remove sharp notes from the score
    for note_to_remove in notes_to_remove:
        if isinstance(note_to_remove, note.Note):
            print(f"Removing note: {note_to_remove.nameWithOctave}")
        else:
            print(f"Removing chord with pitches: {', '.join(p.nameWithOctave for p in note_to_remove.pitches)}")

        if note_to_remove in score.flat.notesAndRests:
            score.remove(note_to_remove)
        else:
            print(f"Note or chord not found in score.")
```

4. *shift_overlapping_notes(score):*
   - Оваа функција го прилагодува времето на белешките кои се преклопуваат за да се осигура дека тие нема да се судираат. Ако две ноти се преклопуваат, втората нота се префрла за да започне веднаш по завршувањето на првата.

```
def shift_overlapping_notes(score):
    """Shift overlapping notes instead of cutting them off."""
    for i in range(len(score.flat.notesAndRests) - 1):
        current_note = score.flat.notesAndRests[i]
        next_note = score.flat.notesAndRests[i + 1]

        if isinstance(current_note, (note.Note, chord.Chord)) and isinstance(next_note, (note.Note, chord.Chord)):
            # Calculate the end time of the current note
            current_end_time = current_note.offset + current_note.quarterLength

            # Check if the next note starts before the current note ends
            if next_note.offset < current_end_time:
                # Move the next note's start time to the current note's end time
                next_note.offset = current_end_time

    return score
```

1.4 Worker Thread Class

Класата WorkerThread наследува од QThread, дозволувајќи ѝ да извршува задачи во посебна нишка за да го одржува интерфејсот одговорен и ги испраќа обработените MIDI податоци на Arduino за репродукција.

1.4.1 Клучни компоненти

- *Сигнали:*
  - ❖ update_message: Емитува статусни пораки за ажурирање на интерфејсот за тековната задача (e.g., converting files, processing MIDI).
  - ❖ progress: Емитува цел број за да го покаже напредокот на транформацијата на датотеката (0 до 100).

```
class WorkerThread(QThread):
    update_message = pyqtSignal(str)
    progress = pyqtSignal(int)
```

- *Иницијализација (__init__ method):*
  - ❖ Ги зема input_file и output_dir како аргументи. Ја иницијализира сериската врска со Arduino на наведената COM порта со брзина од 9600. Чека 2 секунди за да дозволи Arduino да се ресетира.

```
    def __init__(self, input_file, output_dir):
        super().__init__()
        self.input_file = input_file
        self.output_dir = output_dir
        self.arduino = serial.Serial( port: "COM13",  baudrate: 9600, timeout=1)
        time.sleep(2)  # Wait for Arduino to reset
```

- *Run методот (run):*
  - ❖ Конвертира MP3 во MIDI: Повикува **convert_mp3_to_midi** и емитува напредок.
  - ❖ Трансформира на MIDI ноти во опсегот на октавата: Повикува **fit_midi_to_octave_range** и емитува напредок.
  - ❖ Испраќа MIDI податоци до Arduino: Го повикува **send_midi_to_arduino_updated_timing** за да ги пренесе вградените MIDI ноти додека го одржува времето.

```
def run(self):
    self.update_message.emit("Converting MP3 to MIDI...")
    midi_file = self.convert_mp3_to_midi(self.input_file, self.output_dir)
    self.progress.emit(50)  # Update progress

    self.update_message.emit("Fitting MIDI notes to octave range...")
    fitted_midi_file = self.fit_midi_to_octave_range(midi_file, os.path.join(self.output_dir, 'adjusted_music.mid'))
    self.progress.emit(100)  # Update progress

    self.update_message.emit("MIDI notes processed")
    # Here you can add the code to send MIDI to Arduino

    # Send MIDI to Arduino
    # Updated function with better timing
    # self.send_midi_to_arduino_updated(fitted_midi_file)
    self.send_midi_to_arduino_updated_timing(fitted_midi_file)  # Timing 2.0
```

1.4.2 Користени методи во Run Method

1. *convert_mp3_to_midi (input_dir, output_dir):*
   - ❖ Користи претходно обучен модел за конвертирање на MP3-датотека во MIDI-датотека. Ја враќа патеката на генерираната MIDI-датотека.

```
@staticmethod
def convert_mp3_to_midi(input_dir, output_dir):
    basic_pitch_model = Model(ICASSP_2022_MODEL_PATH)
    midi_filename = os.path.splitext(os.path.basename(input_dir))[0] + "_basic_pitch.mid"

    predict_and_save(
        audio_path_list=[input_dir],
        output_directory=output_dir,
        model_or_model_path=ICASSP_2022_MODEL_PATH,
        save_midi=True,
        save_model_outputs=True,
        sonify_midi=True,
        save_notes=True
    )
    return os.path.join(output_dir, midi_filename)
```

2. *fit_midi_to_octave_range (midi_file, output_file, min_note='C4', max_note='C5'):*
   - ❖ Зема MIDI-датотека и ги прилагодува нотите да се вклопат во одреден опсег (C4 до C5).
   - ❖ Извршува неколку функции:
     - **Transpose Notes**: Преместува ноти за да се вклопат во опсегот на октавата.
     - **Remove Repeating Chords**: Ги елиминира дупликатите акорди за да ја поедностави репродукцијата.
     - **Shift Overlapping Notes**: Ги прилагодува преклопувачките ноти наместо да ги отстранува.
     - **Remove Sharp Notes**: Ги елиминира нотите со повишувалка.
   - ❖ Ја зачувува прилагодената MIDI-датотека на одредената излезна патека.

```
@staticmethod
def fit_midi_to_octave_range(midi_file, output_file, min_note='C4', max_note='C5', gap_duration=0.2,
                             tempo_factor=2.5, duration_extension=0.5):
    score = converter.parse(midi_file)

    original_notes = transpose_to_octave(score, min_note, max_note)

    # Remove repeating chords
    unique_score = remove_repeating_chords(score)

    # Shift overlapping notes instead of cutting them off
    smooth_score = shift_overlapping_notes(unique_score)

    # Removing sharp notes
    remove_sharps(smooth_score)

    mf = midi.translate.music21ObjectToMidiFile(smooth_score)
    mf.open(output_file, 'wb')
    mf.write()
    mf.close()
    return output_file
```

3. *send_midi_to_arduino_updated_timing (self, midi_file, min_note_duration=200):*
    - ❖ Ги испраќа податоците од MIDI до Arduino, придржувајќи се до оригиналното време.
    - ❖ Чита MIDI пораки и ги обработува:
        - Го прилагодува времето врз основа на оригиналното темпо на песната.
        - Собира ноти во акорди за испраќање.
        - Ги испраќа акордите до Arduino со мало задоцнување за да се одржи тајмингот.

```
def send_midi_to_arduino_updated_timing(self, midi_file, min_note_duration=200):
    """
    Sends MIDI data to Arduino while following the original timing and slowing down the tempo as needed.
    :param midi_file: Path to the MIDI file
    :param min_note_duration: Minimum duration in milliseconds for any note, regardless of MIDI timing.
    """
    try:
        mf = MidiFile(midi_file)
        print(f"Loaded MIDI file: {midi_file}")

        ticks_per_beat = mf.ticks_per_beat
        tempo = 500000  # Default tempo in microseconds per beat (120 BPM)
        current_time = 0
```

```
        for i, track in enumerate(mf.tracks):
            print(f"Processing track {i + 1}/{len(mf.tracks)}")

            notes_to_send = []  # To store notes in a chord

            for msg in track:
                print(f"Message: {msg}")

                if msg.type == 'set_tempo':
                    tempo = msg.tempo

                # Convert ticks to milliseconds and apply the tempo scaling
                delta_time_ms = msg.time * (tempo / ticks_per_beat) / 1000.0
                current_time += delta_time_ms

                if msg.type == 'note_on' and msg.velocity > 0:
                    pitch = msg.note

                    # Adjust the duration using tempo scale and ensure a minimum duration
                    duration = max(int(delta_time_ms), min_note_duration)

                    notes_to_send.append((pitch, duration))
```

```
                # If there's a delay or an end of the track, send all collected notes as a chord
                if (msg.type == 'note_off' or msg.time > 0) and len(notes_to_send) > 0:
                    print(f"Sending chord with {len(notes_to_send)} notes")

                    # Send all notes in the chord
                    self.send_chord_to_arduino(notes_to_send)
                    notes_to_send.clear()  # Clear the notes buffer for the next chord

                    # Wait for the correct timing before sending the next chord/note
                    time.sleep(delta_time_ms / 1000.0)

        print("MIDI file processed successfully. Closing connection.")
        self.arduino.close()

    except serial.SerialException as se:
        print(f"Serial communication error: {se}")
    except FileNotFoundError as fnfe:
        print(f"MIDI file not found: {fnfe}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
    finally:
        atexit.register(close_arduino_connection)
        if self.arduino.is_open:
            self.arduino.close()
        print("Serial connection closed.")
```

4. *send_chord_to_arduino (self, notes):*
   - Испраќа збирка ноти (акорд) до Arduino. Секоја нота се испраќа со своето времетраење, претворена во формат од два бајти.
   - Имплементира механизам за повторен обид за добивање потврда (ACK) од Arduino за да се потврди успешната репродукција.

```python
def send_chord_to_arduino(self, notes):
    """
    Sends a chord (multiple notes) to the Arduino. Each note is sent along with its duration.
    """
    try:
        for pitch, duration in notes:
            # Convert duration to 2 bytes
            duration_bytes = [duration >> 8, duration & 0xFF]
            # Send the pitch and duration bytes to Arduino
            self.arduino.write(bytes([pitch] + duration_bytes))
            print(f"Sent note {pitch} with duration {duration}")
            # Add a small delay to avoid overloading the Arduino with many notes at once
            time.sleep(0.05)  # 50 ms delay between notes

        # Retry mechanism for ACK
        retries = 3
        ack_received = False
        while retries > 0:
            start_time = time.time()
            while time.time() - start_time < 2:  # 2-second timeout
                ack = self.arduino.read()  # Read one byte
                if ack == b'\x06':  # ACK received
                    print("ACK received, chord played successfully.")
                    ack_received = True
                    break
            if ack_received:
                break
            else:
                retries -= 1
                print(f"ACK timeout, retrying... ({3 - retries}/3)")

        if retries == 0:
            print("Failed to receive ACK after 3 retries, moving to next notes...")

    except Exception as e:
        print(f"Error sending chord to Arduino: {e}")
```

5. *close_arduino_connection (self):*
   - Безбедно ја затвора врската со Arduino, осигурувајќи дека ресурсите се ослободуваат кога нишката е завршена.

```python
def close_arduino_connection(self):
    if self.arduino:
        print("Closing Arduino connection safely.")
        self.arduino.close()
```

1.5 Различни варијанти на midi_to_arduino

1.5.1 Функција: send_midi_to_arduino_bulk(midi_file, max_notes=64)

I. Цел

- Ги испраќа сите MIDI податоци до Arduino, придржувајќи се до одреден максимален број на ноти.

II. Параметри

- **midi_file**: патека до датотеката MIDI што треба да се обработи.
- **max_notes**: Максимален број ноти за испраќање одеднаш (default: 64).

III. Чекори

1. **Воспоставете сериска врска**: Се поврзува со Arduino преку одредена COM порта. Чека Arduino да се ресетира. Ја потврдува врската.

2. **Вчитување MIDI датотека**: Ја вчитува MIDI-датотеката користејќи MidiFile. Иницијализира празен список note_data за складирање на информации за нотите и времетраење.

3. **Процесирање MIDI траки**: Итерија низ секоја трака во MIDI датотеката.
   За секоја порака во траката:

   3.1 Ако се работи за порака note_on (со брзина > 0), ги извлекува висината и времетраењето.
   3.2 Ја проверува валидноста на вредностите на висината и времетраењето.
   3.3 Го конвертира времетраењето во два бајта и ги додава бајтите на висината и времетраењето во note_data.
   3.4 Проверува дали акумулираните ноти достигнуваат max_notes и ја прекинува јамката ако е така.

4. **Испракање податоци до Arduino**: Ако има ноти за испраќање, го праќа note_data до Arduino плочата. Изборно чека потврда (ACK) од Arduino.

5. **Затварање конекција**: Ја затвара сериската врска и се справува со исклучоци за сериски грешки и сценарија за ненајдена датотека.

```
def send_midi_to_arduino_bulk(midi_file, max_notes=64):  # Set a maximum number of notes to send
    try:
        # Initialize serial connection to Arduino
        print("Attempting to connect to Arduino...")
        arduino = serial.Serial( port: 'COM13',  baudrate: 9600, timeout=1)  # Ensure the correct port is used
        time.sleep(2)  # Allow time for Arduino to reset
        print("Connected to Arduino!")

        mf = MidiFile(midi_file)
        print(f"Loaded MIDI file: {midi_file}")

        note_data = []  # Store the notes and their corresponding durations
        # Loop through each track and message in the MIDI file
        for i, track in enumerate(mf.tracks):
            print(f"Processing track {i + 1}/{len(mf.tracks)}")

            for msg in track:
                print(f"Message: {msg}")

                # Handle 'note_on' messages
                if msg.type == 'note_on' and msg.velocity > 0:
                    pitch = msg.note
                    duration = msg.time  # Duration in ticks

                    # Validate pitch
                    if 0 <= pitch <= 127:  # Ensure pitch is within MIDI range
                        # Ensure duration is within acceptable limits
                        if 0 <= duration <= 65535:  # Check if duration can be represented in 2 bytes
                            # Split duration into two bytes and store in the list
                            duration_bytes = [duration >> 8, duration & 0xFF]
                            note_data.extend(
                                [pitch] + duration_bytes)  # Append the pitch and duration bytes to the data list

                            # Check if we have reached the maximum number of notes
                            if len(note_data) // 3 >= max_notes:  # Each note consists of 3 bytes (pitch + duration)
                                break  # Exit the loop if max notes reached

            if len(note_data) // 3 >= max_notes:
                break  # Exit the outer loop if max notes reached
```

```
    # Send all accumulated note data in one go
    if note_data:
        print(f"Sending {len(note_data) // 3} notes in bulk to Arduino...")
        arduino.write(bytes(note_data))

        # Wait for acknowledgment (optional)
        ack = arduino.read_until(b"ACK")
        if ack == b"ACK":
            print("Arduino received all notes successfully.")
        else:
            print("No acknowledgment received from Arduino.")
    else:
        print("No note data found to send.")

    print("Closing connection.")
    arduino.close()

except serial.SerialException as se:
    print(f"Serial communication error: {se}")
except FileNotFoundError as fnfe:
    print(f"MIDI file not found: {fnfe}")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
finally:
    if arduino.is_open:
        arduino.close()
    print("Serial connection closed.")
```

1.5.2 Функција: send_midi_to_arduino(midi_file)

I. Цел

- Испраќа MIDI податоци до Arduino плочата една по една нота, осигурувајќи се дека секоја нота е потврдена пред да се испрати следната.

II. Параметри

- **midi_file**: патека до датотеката MIDI што треба да се обработи.

III. Чекори

1. **Воспоставете сериска врска**
2. **Вчитување MIDI датотека**
3. **Процесирање MIDI траки**: Итерија низ секоја трака во MIDI датотеката.
   За секоја порака во траката:
   - 3.1 Ако се работи за порака note_on (со брзина > 0), ги извлекува висината и времетраењето и ги препраќа.
   - 3.2 Имплементира циклус за да чека ACK од Arduino пред да ја испрати следната нота.
4. **Справување со акорди**: Проверува дали има акорди и ги обработува слично како поединечните ноти.
5. **Затварање конекција:** Ја затвора врската и се справува со исклучоците.

```
def send_midi_to_arduino(midi_file):
    try:
        # Initialize serial connection to Arduino
        print("Attempting to connect to Arduino...")
        arduino = serial.Serial( port: 'COM13',  baudrate: 9600, timeout=1)  # Ensure the correct port is used
        time.sleep(2)  # Allow time for Arduino to reset
        print("Connected to Arduino!")

        mf = MidiFile(midi_file)
        print(f"Loaded MIDI file: {midi_file}")
```

```
        for i, track in enumerate(mf.tracks):
            print(f"Processing track {i + 1}/{len(mf.tracks)}")

            for msg in track:
                print(f"Message: {msg}")

                # Handle 'note_on' messages
                if msg.type == 'note_on' and msg.velocity > 0:
                    pitch = msg.note
                    duration = msg.time  # Duration in ticks

                    print(f"Sending note {pitch} with duration {duration}")
                    try:
                        # Split duration into two bytes
                        duration_bytes = [duration >> 8, duration & 0xFF]
                        arduino.write(bytes([pitch] + duration_bytes))

                        # Wait for ACK from Arduino before continuing
                        while True:
                            ack = arduino.read()  # Read one byte
                            if ack == b'\x06':  # 0x06 is the ASCII code for ACK
                                print("ACK received, sending next note...")
                                break  # Exit the loop once ACK is received
                            else:
                                print("Waiting for ACK...")

                    except Exception as e:
                        print(f"Error sending note to Arduino: {e}")
```

```
                # Example case for handling a chord
                elif msg.type == 'chord':
                    chord_pitches = [note.note for note in msg.notes]
                    duration = msg.time

                    print(f"Sending chord {chord_pitches} with duration {duration}")
                    try:
                        chord_bytes = [pitch for pitch in chord_pitches]
                        duration_bytes = [duration >> 8, duration & 0xFF]
                        arduino.write(bytes(chord_bytes + duration_bytes))

                        # Wait for ACK from Arduino before continuing
                        while True:
                            ack = arduino.read()  # Read one byte
                            if ack == b'\x06':  # 0x06 is the ASCII code for ACK
                                print("ACK received, sending next note...")
                                break  # Exit the loop once ACK is received
                            else:
                                print("Waiting for ACK...")

                    except Exception as e:
                        print(f"Error sending chord to Arduino: {e}")

        print("MIDI file processed successfully. Closing connection.")
        arduino.close()
```

```
    except serial.SerialException as se:
        print(f"Serial communication error: {se}")
    except FileNotFoundError as fnfe:
        print(f"MIDI file not found: {fnfe}")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
    finally:
        if arduino.is_open:
            arduino.close()
        print("Serial connection closed.")
```

1.5.3 Функција: send_midi_to_arduino_updated(midi_file)

I. Цел

- Испраќа MIDI податоци до Arduino плочата додека го одржува оригиналното време и го зголемува темпото.

II. Параметри

- **midi_file**: патека до датотеката MIDI што треба да се обработи.

III. Чекори

1. **Воспоставете сериска врска**
2. **Вчитување MIDI датотека:** Ја вчитува датотеката MIDI и ги враќа информациите за темпото.
3. **Процесирање MIDI траки**: Итерија низ секоја трака во MIDI датотеката.
   За секоја порака во траката:
   3.1 Го прилагодува времето за секоја нота врз основа на оригиналното MIDI темпо.
   3.2 Собира ноти за испраќање како акорди доколку е потребно.
4. **Испраќање ноти до Arduino плочата**
5. **Затварање конекција**

```
def send_midi_to_arduino_updated(midi_file):
    try:
        print("Attempting to connect to Arduino...")
        arduino = serial.Serial( port: 'COM13',  baudrate: 9600, timeout=1)  # Adjust port as necessary
        time.sleep(2)  # Allow time for Arduino to reset
        print("Connected to Arduino!")

        mf = MidiFile(midi_file)
        print(f"Loaded MIDI file: {midi_file}")

        ticks_per_beat = mf.ticks_per_beat
        tempo = 500000  # Default tempo in microseconds per beat (120 BPM)
        current_time = 0
```

```
    for i, track in enumerate(mf.tracks):
        print(f"Processing track {i + 1}/{len(mf.tracks)}")

        notes_to_send = []  # To store notes in a chord

        for msg in track:
            print(f"Message: {msg}")

            if msg.type == 'set_tempo':
                tempo = msg.tempo

            # Convert ticks to milliseconds
            delta_time_ms = msg.time * (tempo / ticks_per_beat) / 1000.0
            current_time += delta_time_ms

            if msg.type == 'note_on' and msg.velocity > 0:
                pitch = msg.note

                # Increase the duration slightly to slow down servos
                duration = int(delta_time_ms * 1.5)

                # Set a minimum duration of 100ms to prevent rapid movement
                if duration < 100:
                    duration = 100

                notes_to_send.append((pitch, duration))
```

```
            # If there's a delay or an end of the track, send all collected notes as a chord
            if (msg.type == 'note_on' and msg.velocity == 0) or (len(notes_to_send) > 0 and msg.time > 0):
                for pitch, duration in notes_to_send:
                    print(f"Sending note {pitch} with duration {duration}")
                    try:
                        duration_bytes = [duration >> 8, duration & 0xFF]
                        arduino.write(bytes([pitch] + duration_bytes))

                        # Add a small delay between sending notes to prevent servo overload
                        time.sleep(0.05)  # 50 ms delay between notes

                    except Exception as e:
                        print(f"Error sending note to Arduino: {e}")
```

```
                # Retry mechanism for ACK
                retries = 3
                ack_received = False
                while retries > 0:
                    start_time = time.time()
                    while time.time() - start_time < 2:  # 2-second timeout
                        ack = arduino.read()  # Read one byte
                        if ack == b'\x06':  # ACK received
                            print("ACK received, sending next notes...")
                            ack_received = True
                            break
                    if ack_received:
                        break
                    else:
                        retries -= 1
                        print(f"ACK timeout, retrying... ({3 - retries}/3)")

                if retries == 0:
                    print("Failed to receive ACK after 3 retries, moving to next notes...")

                notes_to_send.clear()  # Clear the notes buffer for the next chord or note

    print("MIDI file processed successfully. Closing connection.")
    arduino.close()
```

1.6 Како е меѓусебно поврзано

- **Работен тек на UI**:

    1. Корисникот прикачува MP3-датотека и директориум за излез или ги користи копчињата за рачно да ги избере.
    2. Со притискање на копчето „ Convert and Send “, апликацијата ги крие елементите на корисничкиот интерфејс и ја прикажува лентата за напредок, што укажува на почеток на процесот на конверзија.
    3. Работничката нишка (WorkerThread) ја обработува MP3-датотеката, ја претвора во MIDI и ги применува функциите за обработка.
    4. Откако ќе заврши конверзијата, лентата за напредок достигнува 100%, а корисникот може да избере да обработи друга датотека со кликнување на копчето „ Process Again “, со што се ресетира интерфејсот.

- **Комуникација со Arduino плочата:** Обработените MIDI податоци се испраќаат до Arduino со помош на една од дефинираните функции за испраќање, обезбедувајќи соодветно време и комуникација.

2. Arduino код

2.1 SingleNotes_v0.1

Оваа скица на Arduino контролира осум серво мотори користејќи го Adafruit PWM Servo Driver и PCA9685 модул. Секое серво претставуваат по една музичка нота во опсег од C4 до C5, а скицата прима MIDI податоци преку сериска врска, предизвикувајќи одредени серво мотори да се движат врз основа на MIDI податокот.

2.1.1 Клучни компоненти

1. **Иницијализација на серво драјвер:**

    - Серво моторите се контролираат со помош на Adafruit PWM Servo Driver, иницијализиран со фреквенција од 60 Hz. Вредностите SERVOMIN и SERVOMAX ја дефинираат ширината на пулсот за минимални и максимални агли.

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

#define SERVOMIN  150  // Minimum pulse length count (adjust if needed)
#define SERVOMAX  600  // Maximum pulse length count (adjust if needed)
#define SERVO_FREQ 60   // Analog servos run at ~60 Hz
```

2. **Поставување агол за секој серво мотор:**

    - На секој серво (што одговара на ноти C4 до C5) му се доделува минимален и максимален агол:

        ❖ c0_angle_min до c7_angle_min: Почетни позиции.

- c0_angle_max до c7_angle_max: Конечни позиции по активирањето на нотата.

```
// Define min and max angles for each servo
int c0_angle_min = 173;
int c1_angle_min = 176;
int c2_angle_min = 171;
int c3_angle_min = 172;
int c4_angle_min = 174;
int c5_angle_min = 162;
int c6_angle_min = 164;
int c7_angle_min = 175;

int c0_angle_max = 163;
int c1_angle_max = 163;
int c2_angle_max = 160;
int c3_angle_max = 160;
int c4_angle_max = 162;
int c5_angle_max = 149;
int c6_angle_max = 151;
int c7_angle_max = 163;
```

3. **Сериска комуникација:**

- Скицата чита по три бајти од серискиот влез за секоја нота:
  - **Pitch:** Определува кој серво да се движи врз основа на MIDI нотата.
  - **Duration High Byte** и **Duration Low Byte:** Определува колку долго сервото ќе остане во положбата „притиснато“.

```
void loop() {
  // Check if data is available from the serial
  if (Serial.available() >= 3) {  // Make sure we have 3 bytes per note
    int pitch = Serial.read();
    int duration_high = Serial.read();  // Second byte is the high byte of duration
    int duration_low = Serial.read();   // Third byte is the low byte of duration
    int duration = (duration_high << 8) | duration_low;  // Combine high and low bytes

    // Map MIDI pitch to servo index
    int servo_index = mapMidiPitchToServo(pitch);

    if (servo_index != -1) {
      moveServo(servo_index, duration);  // Move servo based on note
    }
  }

  // Check for active servos and return them to their starting position after the duration
  for (int i = 0; i < 8; i++) {
    if (activeServo[i]) {
      unsigned long currentTime = millis();
      if (currentTime - startTime[i] >= servoDuration[i]) {
        resetServo(i);  // Reset the servo to the start position
        activeServo[i] = 0;  // Mark the servo as inactive
      }
    }
  }
}
```

4. **MIDI Pitch мапирање:**
    - MIDI белешките (C4 до C5) се мапирани на специфични сервоси.

```
// Function to map a MIDI pitch to the corresponding servo index
int mapMidiPitchToServo(int pitch) {
  switch (pitch) {
    case 60: return 0;  // C4
    case 62: return 4;  // D4
    case 64: return 1;  // E4
    case 65: return 5;  // F4
    case 67: return 2;  // G4
    case 69: return 6;  // A4
    case 71: return 3;  // B4
    case 72: return 7;  // C5
    default: return -1;  // Return -1 if the pitch is outside the servo range
  }
}
```

5. **Преместување и ресетирање на серво мотори:**
    - Кога ќе се прими MIDI нотата, соодветното серво се преместува во својата максимална позиција и започнува тајмерот.

```
// Function to move the servo for a given duration
void moveServo(int channel, int duration) {
  int end_angle = get_end_angle(channel);
  if (end_angle != -1) {
    int pulseWidth = map(end_angle, 0, 180, SERVOMIN, SERVOMAX);
    pwm.setPWM(channel, 0, pulseWidth);

    startTime[channel] = millis();  // Store the start time
    activeServo[channel] = 1;       // Mark this servo as active
    servoDuration[channel] = duration;  // Store the duration for this servo
  }
}
```

    - По одреденото времетраење, сервото се ресетира на својата минимална положба.

```
// Function to reset the servo back to the start position
void resetServo(int channel) {
  int start_angle = get_start_angle(channel);
  if (start_angle != -1) {
    int pulseWidth = map(start_angle, 0, 180, SERVOMIN, SERVOMAX);
    pwm.setPWM(channel, 0, pulseWidth);
  }
}
```

    - Активноста на сервосот се следи со помош на низи за почетни времиња, времетраење и активни состојби.

```
// Timing variables
unsigned long startTime[8];  // To store start times for each servo
int activeServo[8] = {0};    // Track which servos are active
int servoDuration[8];        // Track duration for each active servo
```

6. **Останати потребни функции:**

- setup(): Неговата цел е да ја иницијализира комуникацијата, да го конфигурира двигателот на серво моторот и да ги постави сите серво мотори на нивните почетни агли. По поставувањето, таа паузира неколку секунди пред да започне да работи главната програма.

```
void setup() {
  Serial.begin(9600);
  pwm.begin();
  pwm.setPWMFreq(SERVO_FREQ);  // Set PWM frequency to 60 Hz

  // Initialize all servos to their starting position
  for (int channel = 0; channel < 8; channel++) {
    int start_angle = get_start_angle(channel);
    int pulseWidth = map(start_angle, 0, 180, SERVOMIN, SERVOMAX);
    pwm.setPWM(channel, 0, pulseWidth);
    Serial.print("Channel "); Serial.print(channel); Serial.println(" initialized to starting position");
    delay(600);  // Short delay between initializing each servo
  }

   delay(3000);  // Wait 3 seconds before starting the loop
  Serial.println("Wating for input: ");

}
```

- get_start_angle(channel) и get_end_angle(channel): Враќање на аглите за почеток и крај за секој серво врз основа на неговиот индекс.

```
// Function to get the start angle for a given channel
int get_start_angle(int channel) {
  switch (channel) {
    case 0: return c0_angle_min;
    case 1: return c1_angle_min;
    case 2: return c2_angle_min;
    case 3: return c3_angle_min;
    case 4: return c4_angle_min;
    case 5: return c5_angle_min;
    case 6: return c6_angle_min;
    case 7: return c7_angle_min;
    default: return -1;
  }
}

// Function to get the end angle for a given channel
int get_end_angle(int channel) {
  switch (channel) {
    case 0: return c0_angle_max;
    case 1: return c1_angle_max;
    case 2: return c2_angle_max;
    case 3: return c3_angle_max;
    case 4: return c4_angle_max;
    case 5: return c5_angle_max;
    case 6: return c6_angle_max;
    case 7: return c7_angle_max;
    default: return -1;
  }
}
```

2.2.2 Работен тек

1. Програмата ги иницијализира сите серво мотори на нивните почетни позиции.
2. Чека сериски влез од страната на Python програмата (MIDI ноти).
3. Кога се примаат MIDI податоци, соодветното серво се преместува врз основа на нотата, се задржува за одреденото времетраење, а потоа се ресетира.
4. Јамката постојано ги проверува и ресетира сите активни серво мотори чие времетраење е истечено.

2.2 SingleNotes_v0.2

Двете скици контролираат сет на серво мотори засновани на MIDI влез. Сепак, изменетата скица вклучува некои подобрувања и оптимизации, особено при ракување со MIDI податоци и серво контрола.

2.2.1 Клучни разлики

1. **Ракување со сериски податоци:**
   - **Претходна верзија:** Јамката проверува дали се достапни најмалку три бајти податоци пред да се прочита MIDI нотата.
   - **Модифицирана верзија:** Условот *if (Serial.available() > 0)* се користи за иницирање на читање, проследено со циклус за while за обработка на сите достапни податоци се додека има најмалку три бајти. Оваа промена овозможува поефикасно читање на повеќе MIDI ноти последователно.

```
void loop() {
  // Check if data is available from the serial
  if (Serial.available() > 0) {  // Make sure we have 3 bytes per note
    while (Serial.available() >= 3){
      int pitch = Serial.read();
      int duration_high = Serial.read();  // Second byte is the high byte of duration
      int duration_low = Serial.read();   // Third byte is the low byte of duration
      int duration = (duration_high << 8) | duration_low;  // Combine high and low bytes

      // Map MIDI pitch to servo index
      int servo_index = mapMidiPitchToServo(pitch);

      if (servo_index != -1) {
        moveServo(servo_index, duration);  // Move servo based on note
      }
    }

  }

  // Check for active servos and return them to their starting position after the duration
  for (int i = 0; i < 8; i++) {
    if (activeServo[i]) {
      unsigned long currentTime = millis();
      if (currentTime - startTime[i] >= servoDuration[i]) {
        resetServo(i);  // Reset the servo to the start position
        activeServo[i] = 0;  // Mark the servo as inactive
      }
    }
  }
```

2. **Отстранети сериски принтања:**
   - **Претходна верзија:** Изјавите за сериско печатење беа вклучени за да дадат повратни информации за време на фазата на иницијализација и да покажат чекање за внесување.
   - **Модифицирана верзија:** Овие печатени изјави се отстранети.
3. **Сигнал за потврда:**
   - **Модифицирана верзија:** По проверката за активни серво мотори, изменетиот код испраќа сигнал за потврда назад до испраќачот, што покажува дека MIDI податокот е примен и обработен. Оваа функција не беше присутна во претходната верзија, овозможувајќи подобрена комуникација помеѓу испраќачот и примачот, што може да помогне во дебагирање или синхронизација во поставувањето на повеќе уреди.

```
  //Send Acknowledgment back
  Serial.write(0x06);
}
```

2.3 SingleNotes_v0.3

Генерално, новата скица се подобрува во однос на претходната верзија со подобрување на организацијата, читливоста и логиката на серво контрола.

2.3.1 Клучни разлики

1. **Ракување со сериски податоци:**
   - Предходно, серискиот влез се проверуваше со *if (Serial.available() > 0)* проследено со вгнездена јамка за читање податоци. Новата скица го поедноставува ова со *while (Serial.available() >= 3)*, со што станува појасно дека програмата ќе продолжи само ако има најмалку три бајти достапни за читање. Ова може малку да ја подобри ефикасноста со избегнување дополнителна проверка.

```
void loop() {
  // Check if data is available from the serial
  while (Serial.available() >= 3) {  // Ensure we have at least 3 bytes
    int pitch = Serial.read();
    int duration_high = Serial.read();  // Second byte is the high byte of duration
    int duration_low = Serial.read();   // Third byte is the low byte of duration
    int duration = (duration_high << 8) | duration_low;  // Combine high and low bytes

    // Map MIDI pitch to servo index
    int servo_index = mapMidiPitchToServo(pitch);

    if (servo_index != -1) {
      moveServo(servo_index, duration);  // Move servo based on note
    }
  }

  // Check for active servos and return them to their starting position after the duration
  unsigned long currentTime = millis();
  for (int i = 0; i < 8; i++) {
    if (activeServo[i] && (currentTime - startTime[i] >= servoDuration[i])) {
      resetServo(i);  // Reset the servo to the start position
      activeServo[i] = 0;  // Mark the servo as inactive
    }
  }

  // Send acknowledgment back
  Serial.write(0x06);
}
```

2. **Логика за движење на серво:**

   - Новата скица додава min_duration од 200 милисекунди во рамките на функцијата moveServo(), осигурувајќи дека сите серво мотори се активни барем за ова времетраење, дури и ако се прими пократко времетраење. Оваа промена може да помогне да се спречат многу кратки движења кои можеби не се забележливи или ефективни.

```
// Function to move the servo for a given duration
void moveServo(int channel, int duration) {
  int end_angle = get_end_angle(channel);
  if (end_angle != -1) {
    int pulseWidth = map(end_angle, 0, 180, SERVOMIN, SERVOMAX);
    pwm.setPWM(channel, 0, pulseWidth);

    startTime[channel] = millis();  // Store the start time
    activeServo[channel] = 1;       // Mark this servo as active
    int min_duration = 200;
    servoDuration[channel] = max(duration, min_duration);  // Store the duration for this servo
  }
}
```

3. **Логика за ресетирање на серво:**

   - **Модифицирана верзија:** Во функцијата resetServo() се додава доцнење(300) по поставувањето на ширината на пулсот на почетната позиција. Ова овозможува кратка пауза за сервото да ја достигне својата позиција пред да продолжи циклусот, што може да помогне да се спречат проблеми со брзите последователни команди кои влијаат на способноста на сервото да реагира правилно.

```
// Function to reset the servo back to the start position
void resetServo(int channel) {
  int start_angle = get_start_angle(channel);
  if (start_angle != -1) {
    int pulseWidth = map(start_angle, 0, 180, SERVOMIN, SERVOMAX);
    pwm.setPWM(channel, 0, pulseWidth);
    delay(300);
  }
}
```

2.4 SingleNotes_v0.4

Финалната скица ја подобрува организацијата и читливоста на кодот со користење на глобални променливи за заеднички податоци, воведување стандардна константа за минимално времетраење и поедноставување на контролниот тек на серво движењата. Додека основната функционалност останува иста, овие промени го прават кодот почист и потенцијално полесен за управување во идните модификации.

2.4.1 Клучни разлики

1. **Промени во декларацијата на променливите:**

   - Во новата скица, некои променливи како start_angle, end_angle, pulseWidth, pitch, duration_high, duration_low, uration, servo_index, and currentTime се глобално декларирани на врвот на скицата. Оваа промена го намалува вишокот со отстранување на потребата од нивно декларирање внатре во функциите, правејќи го кодот почист и потенцијално подобрувајќи ги перформансите бидејќи до овие променливи може директно да се пристапи.

```
int start_angle;
int end_angle;
int pulseWidth;
int pitch;
int duration_high;
int duration_low;
int duration;
int servo_index;
unsigned long currentTime;
```

2. **Константи:**
    - Дефинирана е нова константа MIN_DURATION (поставена на 200) за стандардизирање на минималното времетраење за серво движење.

```
#define MIN_DURATION 200
```

Недостатоци и планови за идно подобрување

1. Серво Мотори

Главен недостаток е квалитетот на серво моторите. SG90 серво мотори се со пластични запчаници и се прилично непрецизни. Подобрување во ова поле ќе бидат попрецизни, метални мотори. Вообичаено вакви серво мотори користат поголем напон и поголема струја што значи и промена на напојувањето, за некое помоќно.

2. Придвижување на роботската конструкцијата

Со самото притискање на синтисајзерот се поместува основата на која што се наоѓаат типките/прстите од роботот. За да се подобри овој дел потребно е да се додаде тежина на основата, за да биде потешко за придвижување, или да се залепи конструкцијата за синтисајзерот, но ова решение не е педантно и се губи модуларноста и мобилноста на синтисајзерот.

3. Ограничување со една окатава

Со самото тоа што на располагање имаме само 8, фиксни серво мотори, кои можат да свират на една фиксна октава на синтисајзерот, и софтверската имплементација се сведе на тоа ние песната да ја трансформираме и да ја сведеме на една октава. Со ова крајниот резултат би се разликувал од првичната песна. Од софтверски аспект, најелегантно подобрување е промена на октава за секоја нота да одговара на октавата во која треба да се отсвири, но со оглед на тоа што користиме готова програма за синтисајзерот, не можеме да правиме промена на октави, софтверски. Решение што е на хардверско ниво е додавање на повеќе серво мотори за свирење на повеќе октави, односно користење на целосната клавијатура. Иако синтисајзерот кој е употребен во овој проект има копчиња за зголемување и намалување на октави, физичко притискање на истите додека се запазува времето и моментот на свирење е невозможно.

4. Недостаток на меморија кај Arduino UNO

Поради ограничен капацитет на меморија нотите на песната мора да се праќаат една по една. Со ова се губи дел од темпото и единствено решение е да се префрлат сите ноти одеднаш. Со додавање на надворешна меморија, ќе може да се префрлат сите ноти, т.е целата песна и темпото и тајмерот да ги регулира самото ардуино.