**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Informatikatudományi Intézet**

**Programozáselmélet és Szoftvertechnológia Tanszék**

Szakdolgozat címe

Szerző: Témavezető:

Fegyó Benedek Dr. Tejfel Máté

Programtervező informatikus BSc. Egyetemi docens

**Budapest 2024**

Ide kerül a hivatalos témabejelentő lap.

Tartalomjegyzék

[1. Bevezetés 1](#_Toc179449177)

[2. Felhasználói dokumentáció 1](#_Toc179449178)

[2.1. Probléma rövid megfogalmazása 1](#_Toc179449179)

[2.2. Felhasznált módszerek 1](#_Toc179449180)

[2.3. Program használata 1](#_Toc179449181)

[2.3.1. Virtuális Környezet telepítése teszteléshez 1](#_Toc179449182)

[2.3.2. Blueprint telepítése 1](#_Toc179449183)

[2.3.3. Zenefelismerő használata 1](#_Toc179449184)

[2.4. Futás közben adódó tipikus hibák megoldása 1](#_Toc179449185)

[3. Fejlesztői dokumentáció 2](#_Toc179449186)

[3.1. A probléma specifikálása 2](#_Toc179449187)

[3.2. MQTT 2](#_Toc179449188)

[3.3. Crepe 3](#_Toc179449189)

[3.4. Python használata a blueprint és az alkalmazás megírásához 3](#_Toc179449190)

[3.5. Virtuális környezet 3](#_Toc179449191)

[3.5.1. Multipass 4](#_Toc179449192)

[3.5.2. SSH kulcsok generálása 4](#_Toc179449193)

[3.5.3. Cloud-Init.yaml fájl létrehozása 4](#_Toc179449194)

[3.5.4. A virtuális gépek létrehozása és konfigurálása 5](#_Toc179449195)

[3.6. Blueprint 7](#_Toc179449196)

[3.6.1. Control Node konfigurálása 8](#_Toc179449197)

[3.6.2. Node-ok konfigurálása 9](#_Toc179449198)

[3.7. Zenefelismerő alkalmazás 10](#_Toc179449199)

[3.7.1. Control Node működése az alkalmazás futása közben 10](#_Toc179449200)

[3.7.2. Node működése az alkalmazás futása közben 11](#_Toc179449201)

[3.8. Tesztelési terv és a tesztelés ereményei 13](#_Toc179449202)

[4. Összefoglalás és további fejlesztési lehetőségek 14](#_Toc179449203)

[5. Irodalomjegyzék 15](#_Toc179449204)

[6. Melléklet 16](#_Toc179449205)

# Bevezetés

1. **Bevezetés**, mely tartalmazza

* a témaválasztás indoklását,
* és a megoldandó feladat rövid, közérthető leírását.

# Felhasználói dokumentáció

## Probléma rövid megfogalmazása

## Felhasznált módszerek

## Program használata

### Virtuális Környezet telepítése teszteléshez

### Blueprint telepítése

### Zenefelismerő használata

## Futás közben adódó tipikus hibák megoldása

# Fejlesztői dokumentáció

* a probléma részletes specifikációját,
* a felhasznált módszerek leírását, a használt fogalmak definícióját,
* a szoftver logikai és fizikai szerkezetének leírását (architektúráját, komponens szerkezetét), a szoftverben megvalósított modellt: például UML modell vagy modulfelbontás, és/vagy adatkapcsolati (adatbázis) modellt vagy adatszerkezeti modellt, esetlegesen a felhasználói felület tervét,
* a tesztelési tervet és a tesztelés eredményeit.

## A probléma specifikálása

A mesterséges intelligencia alapú felvételről történő zenefelismerés egy kifejezetten körülményes, lassú és legfőképpen nehéz feladat. Telepíteni kell hozzá egy megfelelő környezetet, le kell tölteni a hozzá szükséges modulokat, választani kell egy programozási nyelvet, abban implementálni az alkalmazást, hiba és kompatibilitás ellenőrizni és futtatni ami a mai, forgalomban lévő személyi számítógépek nagy százalékán <hivatkozás> legjobb esetben is körülményes. A szakdolgozatom ezekre a problémákra ajánl megoldást, egy olyan blueprint <az micsoda> formájában ami bizonyos előfeltételek teljesítése után képes egy kattintásra felépíteni egy olyan MQTT hálózatot, ahol a személyi számítógép helyett virtuális, vagy fizikai gépeken, párhuzamosan történik a felismerés. Ezek a gépek rendelkezhetnek olyan erőforrásokkal melyek nagyban meghaladják a személyi számítógép teljesítményét, ezáltal felgyorsítva a felismerési folyamatot.

## MQTT

Az MQTT egy üzenet küldő protokol ami a dolgok internetéhez (IOT) lett kifejlesztve. A kliensek nagyon kicsik, amik vagy feliratkoznak vagy publikálnak egy témára. Az üzenetek mérete minimális, aminek következtében távoli gépeket tud összekötni minimális erőforrás és sávszélesség felhasználásával. Mivel téma alapú a publikációs séma és egy témára bárhány kliens publikálhat, így a feldolgozott szólamokat eljuttatni a control node-nak egy kis sávszélességet igénylő, a kis rendszerterhelésű feladat. Ezáltal a virtuális gépnek több erőforrása marad a felismerő futtatására, mintha például egy tradicionálisabb HTTP protokolon történne a kommunikáció. <https://www.emqx.com/en/blog/mqtt-vs-http> A hálózathoz csatlakozó eszközök száma nincs meghatározva, ami előnyös egy olyan alkalmazás esetében ami a feladatokat több felismerő között osztja szét, mivel a kommunikációs protokol nem szab fizikai korlátot a node-ok lehetséges számának. A beépített QoS biztosítja az üzenetek megbízható megérkezését.

## Crepe

A CREPE egy monofónikus hangmagasság meghatározó, amely mély konvolúciós neurális hálózaton alapul és közvetlenül idő-doménű hanghullám bemeneten dolgozik. A konvolúciós hálót több féle zenei hangon tanították, jól teljesít zajos környezetben, nyílt forráskódú és képes valós időben hangot felismerni. Ezen tulajdonságok ideálissá teszik egy olyan alkalmazáshoz, ahol ugyan annak a python kódnak kell felismernie külömböző hangszerekből származó, valószínűsíthetően zajos, koncertfelvételeket. <https://pypi.org/project/crepe/>

## Python használata a blueprint és az alkalmazás megírásához

A python egy magas szintű programozási nyelv ami egyre populárisabb a fejlesztők körében. Ezen kívűl egy gazdag és jól karban tartott könyvtárral rendelkezik, amelyben szinte bármilyen feladathoz található egy annak elvégzésére való modul. Ennek következtében egy blueprint fejlesztése közben nincs szükség arra, hogy más nyelven fusson a felismerő, a bróker, a control panel és a kommunikációért felelős script, ezek mind használhatják ugyan azt a nyelvet. Ez megkönnyíti a telepítést mivel a külömböző virtuális gépeken egyedül python-ra van szükség ami helytakarékos. A fejlesztés szempontjából is előnyös a keresztplatform támogatása miatt. A windows operációs rendszeren megírt python kód futtatható Linuxon és fordítva.

## Virtuális környezet

Az alkalmazás fejlesztése és tesztelése virtuális környezetben történt. Ennek több oka is volt. Az első, és legfontosabb, hogy a valóságban távoli gépeken, szervereken futó alkalmazások leggyakran virtuálsi gépeken futnak, annak érdekében, hogy a rendszer erőforrásai megfelelően legyenek megosztva, illetve, hogy amennyiben egy alkalmazás futása közben probléma lép fel, az ne befolyásolja a többi folyamatot. Mivel az alkalmazásunk alapból virtuális gépekre készül, így valódi szerverekre történő telepítéskor nem igényel további konfigurációt. <nagyobb cégek miket futtatnak és hol?> A második, hogy a teljes MQTT hálózat és a rajta futó alkalmazás futtatható és tesztelhető legyen egy személyi számítógépen. Ez mind a két folyamatot megkönnyíti, mivel nem kell fizikai hardvert vásárolni, a létrehozott virtuális gépeket gyorsan lehet módosítani, optimalizálni mind RAM, mind háttértár szempontjából. További előnye a módszernek, hogy megtartja a fizikai szeparációt a gépek között, így a környezet hasonló marad egy valódi fizikai gépeken létrehozott MQTT hálózathoz.

### Multipass

A multipass egy platform független eszköz, amivel felhő stílusú virtuális gépeket lehet létrehozni.<https://multipass.run/> Azon kívül, hogy ámogatja a személyes konfigurációt cloud-init interfészen keresztül, egy nagy kép könyvtárral érkezik amiből szabadon lehet külömböző célokra virtuális gépet választani. Erre egy kiváló példa a moquitto appliance, ami egy Mosquitto brókerként működő, előre elkészített virtuális gép. Olyan VM-eket hoz létre, amelyek nem befolyásolják a fizikai gépet, ennek következtében kivállóan lehet velük modellezni egy valós alkalmazás fizikai szerveren történő működését.

### SSH kulcsok generálása

A virtuális környezet létrehozásának első lépése egy SSH kulcs generálása. Ennek az a funkciója, hogy jelszó nélkül be lehessen lépni távolról a Node-okba. Ez azért fontos, mivel a blueprint telepítése közben a fizikai gépnek be kell tudnia lépni a virtuálsi gépekbe, annak érdekében, hogy külömböző folyamatokat elindíthasson, mint például a megfelelő fájlok letöltése GitHub-ról, vagy az MQTT működéséhez szükséges hálózati engedélyek beállítása. Az ssh kulcs generálása a következő kódrészlettel történik. A -C opcióval hozzáadunk egy vmuser kommentet. Ez lesz a felhasználónév amit az ssh-hoz használni fogunk, a -f opcióval pedig megadjuk, hogy a kulcs milyen nevű fájlban legyen eltárolva.

ssh-keygen -C vmuser -f multipass-ssh-key

### Cloud-Init.yaml fájl létrehozása

A Cloud-inti egy standardizált megközelítés a cross-platform virtuális gépek konfigurációjának megadására. A Multipass, képes .yaml fájlból beolvasott konfiguráció alapján létrehozni egy virtuális gépet, így amennyiben átadjuk az általunk generált ssh kulcsot, azzal lehetővé tesszük, hogy jelszó azonosítás nélkül beléphessünk a kivánt virtuális gépbe.

users:

  - default

  - name: vmuser

    sudo: ALL=(ALL) NOPASSWD:ALL

    ssh\_authorized\_keys:

    - <ssh key>

A .yaml fájl az users kulcsszóval definiálja azokat a felhasználókat, amelyeket létre kell hozni a gépen. A -default azt jelzi, hogy alapértelmezett felhasználót kell beállítani ami a multipass-al létrehozott virtuális gép esetében ubuntu. A -name adja meg az új felhasználó nevét. Ennek a névnek meg kell egyeznie az ssh kulcs generálásakor megadott névvel, vagyis amennyiben a „vmuser” nevet adtuk meg kommentben a -C opcióval az ssh-keygen-nek, akkor most is a „vmuser”-t kell használni. A sudo: ALL=(ALL) NOPASSWD:ALL jelszó nélküli sudo jogot ad a felhasználó számára, ami azt jelenti, hogy bármelyik felhasználó nevében futtathat parancsokat a rendszeren jelszó megadása nélkül. Az ssh\_authorized\_keys tartalmazza a publikus kulcsot, amely megadásával a felhasználó ssh hozzáférést nyerhet a virtuális géphez. Amennyiben az ssh kulcsot a -f multipass-ssh-key opcióval definiáltuk, akkor a multipass-ssh-key.pub fájl tartalmát kell beilleszteni ide.

### A virtuális gépek létrehozása és konfigurálása

A virtuális gépek létrehozása és konfigurálása a Virtual\_ENV/setup\_virtual\_env.bat batch fájl futtatásával történik.

#### A broker node létrehozása

A multipass-ban egy előre elérhető appliance a mosquitto ubuntu alapú virtuális gép, amely indítás után képes üzeneteket küldeni és fogadni az MQTT protokolon keresztül. Először ellenőrizzük, hogy már létezik-e a mosquitto virtuális gép, és amennyiben nem, létrehozzuk. A -n opcióval lehet nevet adni a VM-nek. Ellenkező esetben elindítjuk az előre létrehozott példányt.

multipass list | findstr /C:"MosquittoBroker" >nul

if %errorlevel% neq 0 (

    multipass launch appliance:mosquitto -n MosquittoBroker

    …

) else (

    multipass start MosquittoBroker

)

A bróker bár fut és pingelhető, a feladatát még nem képes ellátni, mivel nincs alapértelmezetten definiálva a hallgató port, nem biztos, hogy az összes IP címre hallgat.

#!/bin/bash

sudo cp /var/snap/mosquitto/common/mosquitto\_example.conf /var/snap/mosquitto/common/mosquitto.conf

echo "listener 1883 0.0.0.0" | sudo tee -a /var/snap/mosquitto/common/mosquitto.conf > /dev/null

echo "allow\_anonymous true" | sudo tee -a /var/snap/mosquitto/common/mosquitto.conf > /dev/null

sudo systemctl restart snap.mosquitto.mosquitto.service

echo "ubuntu:pwd" | sudo chpasswd

Ezen a problémáknak a kijavítására bemásoljuk a mosquitto\_example.conf fájlt a mosquitto.conf helyére amivel alapértelmezett konfigurációt hozunk létre a mosquitto számára. Ezután hozzáfűzzük a *listener 1883 0.0.0.0* sort a mosquitto.conf-hoz. Ez megadja, hogy a bróker az összes IP címre hallgasson a 1883-as porton. Engedélyezzük az anonim hozzáférést az *allow\_anonymus true* confifigurációs fájlba másolásával, hogy a brókerhez való csatlakozás során ne kelljen a node-oknak azonosítani magukat. Ez olyan megfontolásból történt, hogy semmilyen nagy kockázatú adat nem megy keresztül a hálózaton, így a biztonsági kockázat elhanyagolható. Ahhoz, hogy a módosítások érvénybe lépjenek, újra kell indítani a mosquitto szolgáltatást, ami a *sudo systemctl restart snap.mosquitto.mosquitto.service* parancs futtatásával lehetséges. A mosquitto kommunikációhoz, bár nem kapcsolódik, viszont hasznos itt beállítani, az alapértelmezett felhasználó jelszavát. Ez arra szolgál, hogy a későbbiekben, ammennyiben szükséges be lehessen lépni ssh kapcsolattal a virtuális gépre. Az előbb említett konfigurációs lépéseket a mosquitto\_broker\_setup.sh shell szkriptben találjuk, amelyet a bróker VM indítása után rámásolunk, engedélyt adunk a futására, majd futtatjuk.

multipass transfer mosquitto\_broker\_setup.sh MosquittoBroker:/home/ubuntu/mosquitto\_broker\_setup.sh

    multipass exec MosquittoBroker -- chmod +x /home/ubuntu/mosquitto\_broker\_setup.sh

    multipass exec MosquittoBroker -- /home/ubuntu/mosquitto\_broker\_setup.sh

#### A Control Node és felismerő node-ok létrehozása

A virtuális környezet létrehozásakor a külömböző node-oknak nincs semmiféle alapvető feladata azon kívül, hogy az előfeltételeknek megfelelő konfigurációval induljanak el. Ez ebben az esetben azt jelenti, hogy lehessen rá telepíteni a megfelelő modulokat és lehessen vele SSH kapcsolatot létesíteni. Bizonyos python könyvtárak használata esetében ez különösen fontos, mivel például a Tensorflow-hoz, ami kritikus része a zenefelismerő alkalmazás működésének, az ajánlott minimális memória 8 GB. ([https://eitca.org/artificial-intelligence/eitc-ai-dltf-deep-learning-with-tensorflow/tensorflow/installing-tensorflow/examination-review-installing-tensorflow/what-is-the-minimum-amount-of-ram-recommended-for-allocating-to-the-virtual-machine-running-tensorflow/#:~:text=of%20these%20devices.-,The%20minimum%20amount%20of%20RAM%20recommended%20for%20allocating%20to%20a,and%20prevent%20memory%2Drelated%20issues](https://eitca.org/artificial-intelligence/eitc-ai-dltf-deep-learning-with-tensorflow/tensorflow/installing-tensorflow/examination-review-installing-tensorflow/what-is-the-minimum-amount-of-ram-recommended-for-allocating-to-the-virtual-machine-running-tensorflow/" \l ":~:text=of%20these%20devices.-,The%20minimum%20amount%20of%20RAM%20recommended%20for%20allocating%20to%20a,and%20prevent%20memory%2Drelated%20issues).) A személyi számítógépemen erre nincsen lehetőség, mivel ez már két hangszer esetén is több memóriát igényelne, mint amennyi rendelkezésre áll. Az előbb említett limitáció következtében kénytelen voltam kikísérletezni „trial and error” alapján, a minimálisan szükséges RAM-ot, ami bár nem elég az optimális működéshez, de megfelelő tesztelésre. Ez a szám a lapozási mechanizmus engedélyezését követően két gigabájtra jött ki. Az előbbiekben felsorolt kritériumokat figyelembe véve a node-ot a következőképpen kell elindítani:

multipass launch jammy -n %node1% --cloud-init cloud-init.yaml --disk 10G --memory 2G

Ahol a -n kapcsoló kell megadni a virtuális gép nevét, a –cloud-init után a 3.5.3 szekcióban említett konfigurációs fájlt, a –disk után a háttértár méretét és a –memory után a memória méretét.

## Blueprint

A „blueprint” azaz tervrajz általában egy olyan vázlatot jelent az informatikában, amely egy rendszer, szoftver vagy alkalmazás struktúráját és működését írja le. Ez a gyarkolatban annyit jelent, hogy egy ismeretlen, előre nem definiált környezetben, autómatikusan felállítja egy adott alkalmazás működéséhez szükséges kommunikációs hálót és telepíti a rendszer főbb komponensein a megfelelő modulokat és eszközöket, majd konfigurálja azokat.

Léteznek ugyan nyilt forráskódú, mindenki számára elérhető blueprintek, mint például az Akarino <reference> projekt, viszont ezek szoftverek nagyon ritkán vannak frissítve, aminek következtében, a használatuk legjobb esetben is körülményes. Bizonyos bennük használt modulok mára már elérhetetlenek, vagy teljesen máshogy működnek, mint ahogyan eredetileg. Ennek következtében amennyiben egy kommunkációs hálót, vagy egyedi igényű futási környezetet szeretnénk blueprint segítségével implementálni, jobban járunk, mind idő, mind energiabefektetésileg, ha megírjuk a sajátunkat. <gráf beszúrása a szakmai gyakorlatról, ami összehasonlít bizonyos blueprinteket> Bizonyos technikák azomban elleshetőek és univerzálisan használhatóak az előbb említett konstrukciókból. Ilyen például a nodelist.txt ami egy pipe („|”) szimbólummal elválasztva tartja számon az összes komponens felhasználónevét, IP címét és amennyiben van, jelszavát. Ennek a fájlnak a létezése nagyban elősegíti a skálázhatóságot mivel azzal szemben, hogy minen node-ra egyesével telepítenénk a kívánt modulokat, csak egy ciklus segítségével végigiterálunk a fájlon és végrehajtjuk a kívánt műveleteket minden elemen.

### Control Node konfigurálása

A Control Node-nak a feladata, hogy fogadja a felismerők álltal feldolgozott adatokat, majd azokat feldolgozza és eltárolja. Mivel az üzeneteket MQTT-n keresztül kapja, így szükség van olyan kliensprogramokra, illetve python könyvtárakra amelyek képessé teszik ezek küldésére és fogadására. Ilyen a mosquitto-clients és a paho-mqtt. A feldolgozni kapott adatokat musicxml formátumban kapja, és a pyhton programozási nyelv segítségével dolgozza fel, így szükség van a music21 könyvtárra is ami egy python alapú eszköztár számítógépes zeneelmélethez. < <https://www.music21.org/music21docs/about/what.html>>

Annak érdekében, hogy a telepítési folyamat skálázható legyen akár távoli, vagy fizikai szerverekre a multipass beépített shell parancsa helyett, ami csak lokálisan működik, ssh-val lépünk be a node-ba. A nodelist szöveges fájlból beolvassuk a control node IP címét és a hozzá tartozó felhasználónevet, majd a Windows-ba beépített ssh parancs segítségével belépünk távolról a node-ba. <windows ssh doksi> Ezután ha már létezik töröljük, majd GitHub-ról letöltjük a blueprint legújabb változatát, ami tartalmazza mind a telepítőket, mind a futtatható alkalmazásokat. Ezután belépünk a Blueprint mappába, engedélyt adunk a control\_node\_setup.sh-nak a futásra és futtatjuk. Ez telepíti az előző bekezdésben említett szükséges kliensprogramokat és python könyvtárakat, majd létrehozza a messages mappát és terminál.

ssh %%a@%%b -i ../Virtual\_ENV/multipass-ssh-key "if [ -d 'Szakdolgozat\_v2' ]; then rm -rf Szakdolgozat\_v2; fi && git clone %REPO\_URL% && cd Szakdolgozat\_v2 && chmod +x Blueprint/control\_node\_setup.sh && ./Blueprint/control\_node\_setup.sh"

### Node-ok konfigurálása

A felismerő node-ok konfigurálása hasonló a control node-hoz abból a szempontból, hogy a nodelist szöveges fájlból kiolvassuk a node IP címét és felhasználónevét majd ssh kapcsolattal belépünk rá, letöltjük GitHub-ról a blueprintet majd végrehajtjuk a node\_setup.sh-t. Azonban node-okból bármennyi lehet, aminek következtében az előbb említett folyamatot beágyazzuk egy ciklusba, ami végigiterál a hátralévő sorokon és midegyikre végrehajtja a konfigurációs folyamatot. Ezen kívül a node-okon történik a zenék felismerése, python programozási nyelv segítségével, aminek következtében a control node-hoz szükséges a tensorflow a crepe működéséhez és a crepe a zenefelismeréshez. Továbbá, annak érdekében, hogy a node ne fogyjon ki a memóriából a tensorflow telepítése közben, a 3.5.4.2 szekcióban említettek alapján be kell állítani a lapozási mechanizmust.

#!/bin/bash

#lapozási mechanizmus

sudo fallocate -l 1G /swapfile

sudo chmod 600 /swapfile

sudo mkswap /swapfile

sudo swapon /swapfile

sudo apt install mosquitto-clients -y

sudo apt install python3-pip -y

sudo pip install --upgrade tensorflow

pip install crepe

pip install music21

pip install paho-mqtt

## Zenefelismerő alkalmazás

A zenefelismerő alkalmazás szétosztja a hangszereket és azok szólamait tartalmazó wav fájlokat a felismerő node-ok között, amelyek feldolgozzák és továbbítják az így keletkezett musicxml dokumentumot a control node felé, aki ezeket egy végleges kottává olvasztja össze és eltárolja. Ez után az alkalmazás terminál. Ezt a viselkedést két módon lehet elérni. Az első, ami autómatikusan elindul, akkor következik be, amikor a blueprint minden, az alkalmazás futásához szükséges, modult föltelepített a node-okra. Ekkor autómatikusan elindul a *lampion.py* amely bekéri a felhasználótól a hangszerek adatait, majd továbbküldi őket feldolgozásra. A második, ugyan ezt a viselkedést eredményezi, viszont a felhasználónak manuálisan kell elindítania. Ebben az esetben az Application mappában kell futtatni a *full\_lampion.bat*-et ami az előzőekben említett módon hajtódik végre.

### Adatok bekérése a felhasználótól

Az adatok bekérése a felhasználótól a *lampion.py* python szkript futtatásával történik. A *lampion.py* fájl egy Python szkript, amely különböző eszközök (instruments) és csomópontok (nodes) közötti kommunikációt valósít meg MQTT protokoll segítségével. A szkript két fő funkciót tartalmaz: az egyik a vezérlő csomópont (control node) inicializálása, a másik pedig az eszközök csomópontjainak indítása és a szükséges fájlok küldése. Ezeket a fájlokat a felhasználótól kéri be ellenőrzött módon. Először a hangszerek számát, majd az adott hangszerek nevét, skáláját majd azt, hogy hol található a felismerni kívánt wav fájl. Ezt követően a felhasználónak meg kell erősítenie a hangszer paramétereit, mivel ezt visszamenőlegesen nem lehet módosítani. Az így bekért adatokat egy *Instrument* típusú objektumban tárolja minden hangszerhez.

temp\_instrument = Instrument(instrument\_name, i, instrument\_scale, wav\_file\_location)

Az *initiate\_controll\_node* függvény felelős a vezérlő csomópont inicializálásáért. Ez a függvény egy alprocesszt indít, amely a *mqtt\_publish\_message.py* szkriptet futtatja, és egy üzenetet küld a vezérlő csomópontnak arról, hogy hány eszköz van csatlakoztatva. 3.7.2 A függvényt egy külön szálban (control\_thread) futtatják, hogy ne blokkolja a fő program futását.

def initiate\_controll\_node():

    subprocess.run(["python3", "mqtt\_publish\_message.py", broker\_ip, "ControlNode/initiate\_control\_node", str(len(instruments))])

A *start\_instrument\_node* függvény minden egyes eszközhöz és csomóponthoz külön-külön fut le. Először egy üzenetet küld az adott csomópontnak, hogy indítsa el az eszközt az előzőekben bemutatott módon. Ezután létrehoz egy ideiglenes fájlt, amely tartalmazza az eszköz nevét és skáláját, majd ezt a fájlt és az eszközhöz tartozó WAV fájlt is elküldi az adott csomópontnak. Végül egy üzenetet küld, hogy jelezze az eszköz inicializálásának végét.

A szkript több szálat használ a párhuzamos végrehajtáshoz. Minden eszközhöz és csomóponthoz létrehoz egy új szálat, amely a *start\_instrument\_node* függvényt futtatja. Ezeket a szálakat egy listában (*threads*) tárolja, és mindegyiket elindítja. A program végén minden szálat megvár (*join*), hogy biztosítsa, hogy minden eszköz inicializálása befejeződjön, mielőtt a vezérlő csomópont inicializálása is befejeződik.

for thread in threads:

    thread.join()

control\_thread.join()

### Üzenet és file küldése MQTT-n Python szkripttel

Annak érdekében, hogy az alkalmazás minél könyebben lehessen futtatva külömböző operációs rendszereken az MQTT-n történő üzenetek és fájlok küldése a felhasználó személyi számítógépén python szkripttel lett megvalósítva. Az MQTT kliens kapcsolódik a szerverhez, majd közzéteszi az üzenetet és lecsatlakozik.

client.connect(broker, port)

# Start the network loop

client.loop\_start()

# Publish the message

result = client.publish(topic, message)

# Wait for the message to be published

result.wait\_for\_publish()

# Stop the network loop and disconnect

client.loop\_stop()

client.disconnect()

Fájl küldése esetén ez a folyamat annyiban módolsul, hogy a fájl bináris módon kerül megnyitásra (rb) majd publikálásra.

with open(file\_path, 'rb') as file:  # Open the file in binary mode

        file\_content = file.read()

        client.publish(topic, file\_content)

### Control Node működése az alkalmazás futása közben

A Control Node az alkalmazás indulásakor nem tudja, hogy hány darab feldolgozott musicxml fájlt vár, amelyek az egyesítésre váró szólamokat tartalmazzák. Ezt a számot, amit a továbbiakban hívjunk „n”-nek ( n∈ ℕ+) akkor fogja megkapni, amikor a felhasználó már megadta és megerősítette a felismerni kívánt szólamokat. Ez után „n” darab musicxml file-t vár, amiket ideiglenesen eltárol, majd amikor megérkezett az összes egyesít. Utolsó lélsben a kottát elmenti és törli a temporális fájlokat, majd terminál.

#### Hallgatózó mód

A hallgatózási folyamat a *listen\_control\_node.py* python szkript futtatásával valósul meg. Először ellenőrizzük, hogy a könyvtár ahová menteni szeretnénk az ideiglenes fájljainkat létezik-e. Amennyiben nem, létrehozzuk. Ezután létrehozzuk a *callback* függvényeket azokra az esetekre, amikor csatlakozunk, vagy üzenetet kapunk. A csatlakozás *callback* függvénye kiírja, hogy milyen visszatérési értékkel sikerült csatlakozni a szerverhez, majd feliratkozik a *ControlNode/+* témára. A „+” jel, jelen esetben egy *wildcard* ami azt jelenti, hogy a *ControlNode* téma minden altémájára érkező üzenetet olvasson be.

def on\_connect(client, userdata, flags, rc):

    print(f"Connected with result code {rc}")

    client.subscribe(TOPIC)

A beérkezett üzenetek *callback* fügvénye kétféle módon működik, amelyek között a „first” logikai változó igazra, vagy hamisra változtatásával tudunk kapcsolni. Az első üzenet előtt ez a logikai változó igaz, ami azt jelenti hogy a *CotrolNode/* *+* témáról egy számot szeretnénk beolvasni. Ez a szám lesz az „n” amely meghatározza, hogy hány darab feldolgozott szólamot kell a továbbiakban várni. Ez után a „first” változó értékét hamisra állítjuk. Ez után „n” darab fájlt olvasunk ki a *ControlNode/+* témából és a tartalmukat a messages mappába mentjük. Annak érdekében, hogy minden szólamnak a neve egyedi legyen, a beérkezett musicxml fájloknak a téma nevét adjuk amelyen érkeztek. Ennek az egyediségét a felismerő node-ok biztosítják azáltal, hogy a saját felhasználónevüket használják publikálásnál altémának. (3.7.2.5) A kívánt számú beérkezett file után a hallgatózó módért felelős python szkript lecsatlakozik az MQTT hálózatról és terminál.

#### A beérkezett feldolgozott anyagok egyesítése, tárolása és az ideiglenes fájlok törlése

A beérkezett mscxml fájlokat a *listen\_control\_node.py* a messages mappába írta, olyan módon, hogy a nevük egy egyedi azonosíó amely magába folgalja, hogy melyik node-ról érkeztek. Ezeket a fileokat a *unification.py* segítségével beolvassuk, a music21 könyvtár segítségével ellenőrzi, hogy csak egy szólamot tartalmaznak és amennyiben igen, akkor hozzáadja a „combined\_score” *stream.Score()* típusú változóhoz. Amikor a messages mappa összes musicxml fájlján végigért a szkript, a „combined\_score” változóban eltárolt teljes kotta tartalmát a kimeneti a musescore mappába és terminál. Ezután töröljük a messages mappa tartalmát, hogy felkészüljünk a következő kombinálási feladatra.

### A felismerő node működése az alkalmazás futása közben

A felismerő node feladata, hogy a beérkezett wav formátumú hanganyagot a megadott paraméterek alapján, mint a hangszer és annak skálája, a Crepe mély neuron háló segítségével felismerje, feldolgozza, musicxml formátummá konvertálja, majd továbbítsa a Control Node felé az MQTT hálózaton keresztül. Mivel nem hozunk létre új node-okat minden feladathoz, így a node-oknak univerzális felismerőként kell funkciónálniuk, amelyek megfelelő paraméterezés mellett képesek feldolgozni külömböző hangszerekből és stílusokból származó hanganyagokat.

#### Hallgatózó mód

A node univerzális mivolta következtében szüksége van bizonyos paraméterekre a felismerési folyamat elvárt végrehajtásához. Ezek a paraméterek a hangszer neve, skálája és a felismerni kívánt wav hangfájl. Ezeket az adatokat MQTT-n keresztül a felhasználótól kapja meg. A *node\_action.sh* amit az alkalmazás indításánál futtatunk a felismerő node-on (3.7) először létrehoz egy ideiglenes mappát, azon belül az *out.txt* temporális kimeneti fájlt, majd elindítja a *listen\_node.py*-t a *$(hostname)* bemeneti paraméterrel. Ez azért fontos, mivel ezen a témán fog hallgatni a felismerő bejövő üzenetekre, és mivel a node neve egy egyedi azonosító, ezzel tudjuk biztosítani, hogy ne keveredjenek össze az üzenetek. A node elkezd halgani a „*system/$(username)/+*”témán és addig vár, amíg nem kap egy „start” üzenetet. Amikor ezt megkapta, az ezután következő üzeneteket fájlként eltárolja az ideiglenesen létrehozott mappába (TMP) addíg amíg nem érkezik egy „end” ami után terminál. Annak érdekében, hogy a beérkezett fájlok helyes formátumban legyenek eltárolva, az eredeti nevén létrehozott témában publikáljuk a tartalmát. Ezután a témát adjuk a fájl nevének a felismerő node-on. Például a „beres.wav” hangfájlt a „*system/Node1/beres.wav”* témán publikáljuk amit aztán a node beolvas és a „*TMP/system\_Node1\_beres.wav*” néven ment.

topic = msg.topic

            payload = msg.payload

            file\_path = os.path.join(SAVE\_DIR, topic.replace('/', '\_'))

            print(f"{topic} {file\_path}")

            with open(file\_path, 'wb') as f:

                f.write(payload)

#### Beérkezett hanganyag felismerése a paraméterek alapján

A mentett wav fájl felismerése a *pitch\_detecion.py TMP/\*.wav* szkript végrehajtásával történik. Mivel az ideiglenes mappát a felismerési folyamat végén töröljük és csak egy audió fájlt kapunk node-onként, így biztosak leheteünk benne, hogy a *TMP* mappa egy .wav kitejesztésű elemet tartalmaz. A *scipy.io* könyvtárat használva beolvassuk a mintavételezési rátát és a hanganyagot a *sr* és *audio* változókba.

filename = sys.argv[1]

sr, audio = wavfile.read(f"{filename}")

Ez után meghívjuk a *crepe*.*predict(...)* függvényt, paraméterül adjuk neki a hanganyagot, a mintavételezési rátát és bekapcsoljuk a viterbi simítást. A függvény egy rendezett négyessel tér vissza, ami tartalmazza az időt, a jósolt frekvenciát, a konfidenciát és a nyers aktivációs mátrixot. A *numpy* könyvtár használatával egy két dimenziós mátrixba rendezzük a rendezett négyes első háőrom elemét, mivel az aktivációs mátrixra nem lesz szükség a feldolgozáshoz. Ez után az időt kicseréljük mintavételezési rátára, olyan módon, hogy a léterhozott *numpy.colum\_stack* első oszlopát egyenlővé tesszük két egymást követő idő külömbségével.

time, frequency, confidence, activation = crepe.predict(audio, sr, viterbi=True)

data = numpy.column\_stack((time, frequency, confidence))

rate = data[1][0] - data[0][0]

for i in range(len(data)):

    data[i][0] = rate

Ez után a nyers felismerési adatot mentjük a *TMP/out.txt* fájlba tabulátorokkal elválasztva, 3 tizedes jegyre kerekítve, majd a program terminál.

numpy.savetxt('TMP/out.txt', data, fmt='%.3f', delimiter='\t')

#### Felismert hanganyag feldolgozása

Miután a *pitch\_detection.py* terminál a *client\_action.sh* meghívja a *process.py* python szkriptet, aminek a feladata, hogy feldolgozza a nyers adatokat. Először létrehoz egy tömböt a lehetséges frekvenciákkal egy oktávon. Ez olyan módon történik, hogy alapul vesz egy kromatikus skálát ahol az „A” hang felel meg a 440Hz-nek majd kiszámolja, hogy a bizonyos hangok hol helyezkednek el a 0Hz-től számított első 8 oktávban. Ez a lista lesz az, ami az elérhető hangokat tárolja. Ennél azért nincs szükség szélesebb frekvencia halmazra, mivel az emberi fül ennél magasabb frekvenciát már nem hall. <link?>

notes = [ 'A', 'A#', 'B','C', 'C#', 'D', 'D#', 'E', 'F', 'F#', 'G', 'G#']

octaves = np.arange(0, 9)

frequencies = []

for octave in octaves:

    for note in notes:

        frequency = 440 \* (2 \*\* (octave - 4 + (notes.index(note) / 12)))

        frequencies.append(frequency)

note\_freq\_array = np.array(frequencies)

note\_freq\_array\_2 = list(zip(9\*notes,note\_freq\_array))

note\_freq\_array = np.append(note\_freq\_array,0)

Ez után beolvassuk a felismerés végén eltárolt fájlt és elkezdjük a feldolgozást. Az adatokat egy *numpy* tömbbe tároljuk el. Amennyiben egy felismert frekvenciának a konfidenciája 0.5 alatt van akkor egyenlővé tesszük a hozzá legközelebbi értékkel. Ezután megkeressük az érvényes frekvenciák közül, hogy melyik van a legközelebb az adott méréshez egyenlővé tesszük. Például ha a felismerő 439.83Hz-t mért, akkor ezt lecseréljük a hozzá abszolúlt értékben legközelebb lévő frekvenciára, a 440Hz-re.

def find\_closest\_frequency(number):

    absolute\_diff = np.abs(note\_freq\_array - number)

    closest\_index = np.argmin(absolute\_diff)

    closest\_frequency = note\_freq\_array[closest\_index]

    return closest\_frequency

Annak érdekében, hogy megkapjuk, mennyi ideig tartott egy hang az időben, összegezzük az egymás után következő azonos frekvenciájú mérések mintavételezési idejét, és átlagoljuk a konfidenciájukat.

count=0

sum = 0

confidence = 0

last\_value = -1

data\_2 = np.array([])

for i in range(0, len(data)):

    value = data[i,1]

    time = data[i,0]

    confidence\_tmp = data[i,2]

    if value == last\_value or last\_value == -1:

        sum+=time

        confidence += confidence\_tmp

        count += 1

        last\_value = value

    else:

        data\_2 = np.append(data\_2,[last\_value, sum, (confidence / count)],axis = 0)

        count=0

        sum = 0

        confidence = 0

        last\_value = -1

A jelenlegi adatban még előfordulhatnak felismerési hibák, olyan hangok amelyek valamilyen külső vagy belső tényező következtében tévesen lettek mérve. Ezeknek a soroknak két jellemzője van, irreálisan rövid ideig tartanak (három századmásodpercnél rövidebbek), vagy nagyon alacsony átlagos konfidencia szinttel rendelkeznek. Annak érdekében, hogy ne rövidítsük meg a sorokat, mivel így elveszne a szólamok közötti összhang, az ilyen hibás adatok helyére szünetet illesztünk be. A feldolgozott adatot ami a kezdeti mintavételezési idő, frekvencia, konfidencia helyett most már a hang hossza, hangmagasság, átlagos konfidencia sorokat tartalmazza elmentjük a *TMP/processed.txt* szöveges fájlba az utolsó oszlopot elhagyva, mivel arra a továbbiakban nem lesz szükségünk.

to\_delete = []

for i in range(1,len(data\_2)-1):

    if((data\_2[i, 1] < 0.1) and (data\_2[i-1, 0] == data\_2[i+1, 0])):

        …

    elif((data\_2[i, 1] < 0.05)): … #0.1 > s

    elif(data\_2[i, 2] < 0.70 and round(data\_2[i,0]) != 0):

         …

#### A feldolgozott hanganyag konvertálása mscxml formátummá

A feldolgozott fájlt, a *processed.txt*-t átadjuk a *convert\_to\_musescore.py* python szkriptnek, ami a music21 könyvtár segítségével mscxml formátumúvá konvertálja a felismert adatokat. A hanghosszokat, a *processed.txt* első oszlopát, zeneileg értelmezhető hosszokra konvertálja. Ezek közül a legkisebb a másodperc egy-harmincketted része (0.03125s), ennek következtében töröltünk minden mérést ami három századmásodpercnél rövidebb volt. 3.7.2.3

durations = [1/32, 1/16, 1/8, 1/4, 1/2, 1, 2, 4, 8, 16]  # Available durations

closest\_duration = min(durations, key=lambda x: abs(x - timestamp))

Ez után létrehozunk egy music21 *stream.Score()* objektumot amelybe beleírjuk a hangokat vagy amennyiben 0 volt a frekvencia a szüneteket és a hozzájuk tartozó időintervallumokat, majd az eredményt musicxml formátumban mentjük.

def generate\_musicxml(durations, frequencies, output\_file):

    # Create a music21 Stream object to store the musical elements

    score = stream.Score()

    # Create a Part object to hold the notes

    part = stream.Part()

    score.append(part)

    # Iterate over the durations and frequencies arrays

    for dur, freq in zip(durations, frequencies):

        if freq == 0:

            # Create a Rest object for the pause

            r = note.Rest()

            # Create a Duration object with the corresponding duration value

            d = duration.Duration(dur)

            # Set the quarter length of the Rest object

            r.quarterLength = d.quarterLength

            # Add the Rest object to the Part object

            part.append(r)

        else:

            # Create a Note object with the corresponding frequency

            n = note.Note()

            n.pitch.frequency = freq

            # Create a Duration object with the corresponding duration value

            d = duration.Duration(dur)

            # Set the quarter length of the Note object

            n.quarterLength = d.quarterLength

            # Add the Note object to the Part object

            part.append(n)

    # Convert the score to MusicXML format and save to file

    score.write('musicxml', fp=output\_file)

#### Az eredmény publikálása a Control Node felé

A felismerő node miután felismerte a beérkező hangfájlt, feldolgozta és eltárolta musicxml formátumban közzé teszi ezt a saját témáján az MQTT hálózaton annak érdekében, hogy a Control Node beolvashassa és feldolgozhassa. 3.7.1.1 Ezt a mosquitto\_pub parancssori eszköz segítségével hajtjuk végre. A -h kapcsoló meghatározza a bróker elérési címét, a -t a téma nevét éf a -f a küldésre szánt fájl helyét.

mosquitto\_pub -h MosquittoBroker -t "ControlNode/$hostname" -f TMP/\*.musicxml

A „\*” helyettesítő karakter a *\*.musicxml* fájlnévben olyan megfontolásból lett használva, hogy a shell szkriptet ne kelljen node-specifikussá tenni. Minden node egy musicxml fájlt hoz létre, viszont ezeket nem biztos, hogy azonos néven mentik el. A „\*” használatával kiküszöbölhető minden lehetséges ebből adódó probléma.

#### Az ideiglenes fájlok törlése

Annak érdekében, hogy az alkalmazás következő futásakor ne lépjen fel konfliktus a régi és az új fájlok között, a *node\_action.sh* shell szkript végén az ideiglenes mappát töröljük.

rm -rf TMP

Ezzel végetér a felismerési és publikálási folyamat és a *node\_action.sh* terminál.

## Tesztelési terv és a tesztelés ereményei

# Összefoglalás és további fejlesztési lehetőségek

# Irodalomjegyzék

# Melléklet