# RoboCup Project

**Documentation**

Cornelius Braun, Seraphin Zunzer, Emilia Schulenburg

---

## Table of contents



---

# 1. Introduction

In our RoboCup project, we want to implement a dancing robot that classifies music into genres and selects a suitable keyframe.

### 1.1 Motivation/Goal

The robot should be able to record audio files with the computer microphone. Afterwards, the robot analyzes the recorded audio files with a machine learning classifier for genre detection. Then, the robot selects a suitable keyframe and executes it.

### 1.2 Organization

We split the whole project into three parts: sensing, thinking and action. In sensing, we want to record music with the microphone, or load an audio file from the storage. The thinking part

was about implementing the genre classification. Based on the results from this step, the robot selects and executes a suitable keyframe in the action step. The dance moves for each music genre were pre-defined, so the robot only has to decide for a keyframe depending on the recorded music.

### 1.3 Tools

To implement our program, we used different tools. Here we want to give a short introduction to each tool, details can be found on the linked webpage or by a short research. Each part has its own difficulties, thus we had to select the tools depending on the situation. After having some trouble with earlier selected libraries, we even had to replace them during the implementation process by better ones.

1. **sound-device**, replaced by **pyaudio**:
   Both libraries provide bindings for the PortAudio library and a few functions to record music from a selected microphone. Sound-device is a smaller library and easier to use, but it has fewer functionalities, so we decided to replace it with pyaudio.

2. **librosa**:
   A python package for music and audio analysis. In general, it provides everything that is necessary to create music information retrieval systems.

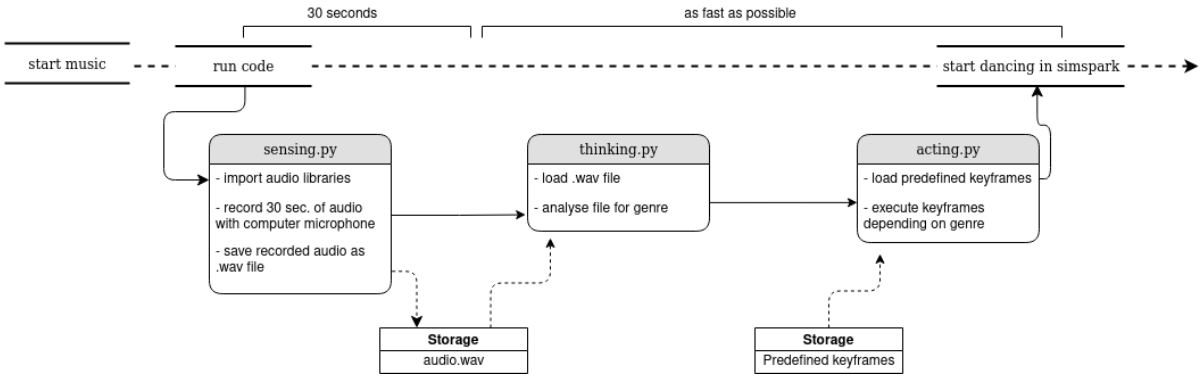
3. **scipy**/ **keras**:
   Both are very popular and powerful libraries used for scientific computing and the implementation of different classifiers.

4. **threading.py**:
   A python library for threading. We used this to listen for start and end times of music continuously during the execution of the application.

## 2. Project planning, draft and procedure

### 2.1 Project planning and draft

To understand what we have to implement, we've created a data flow diagram.
This chart includes all three main components we had at the beginning, the data transmission between the components and all memory accesses. We added a timeline on top to illustrate what the robot should do after starting the music. This draft is only sequential and does not regard the parallelized program flow that we actually need. A detailed explanation follows in section *3. Implementation*.




### 2.2 Working procedure

We split the project into three parts, so that everyone could start with implementing one of the three components.
After we finished the core functionalities, we encountered some problems. To fix these errors and improve the performance, especially the results of the genre classification caused by the bad microphone quality, we decided to work more together.
Think of scrum: we had many cycles to finish a part of the project. Then we had long discussions about problems that occurred, current results, what is working and what is too hard to realize. Furthermore, we thought about new improvements we can make until our next meeting.
To get the best results we were not only working on our own part but helping each other with abstract ideas or concrete implementation ideas.

# 3. Implementation, Problems and Results

We separated the implementation into three parts: Sensing, Thinking and Acting. Below, we describe the final implementation structure, the implementation of the individual parts, the results themselves and all difficulties that occurred during the implementation process.

### 3.1 Implementation

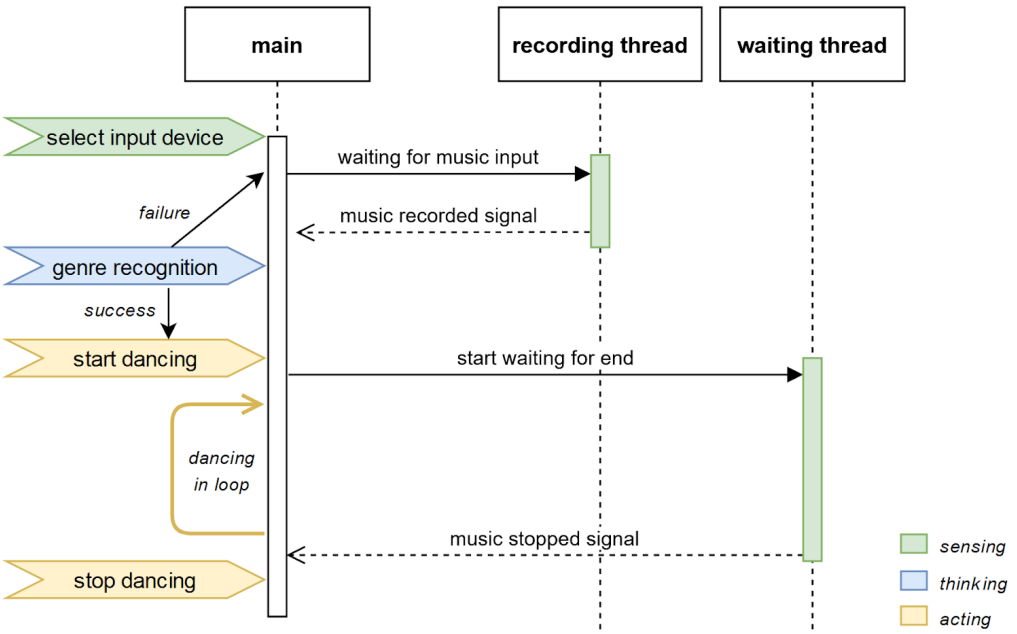
In this section, we want to make a short introduction into the general implementation structure and the user interface.
The user can select different functionalities via the command line input, for example, what kind of microphone the robot should use or if the robot should load music or record it with the microphone.

```
----------------------------record device list---------------------------
Input Device  0  -  Microsoft Sound Mapper - Input
Input Device  1  -  Stereomix (Realtek(R) Audio)
Input Device  2  -  Microphone Array (Realtek(R) Au
------------------------------------------------------------------------
Select a microphone before starting! Enter device number and press return: 1

Setup done!

------------------------------ Select Input ------------------------------
   1: Record song live with selected microphone.
   2: Load already recorded song from storage.
   3: Always use option 1.
   4: Always use option 2.
------------------------------------------------------------------------
Enter a number:
```

We needed to change the code structure a bit, because we want to control the robot and record music simultaneously. For this, we used the threading library in python.
The following sequence diagram shows the final implementation of the whole project.




### 3.1.1 Sensing

For the sensing part itself, we used the pyaudio library that can record music from a selected microphone and apply different functions and filters on the received audio array.

First, the user can select a microphone to record the music. This is done by a predefined function of the pyaudio library. Afterwards, a new recording thread is started where the user can choose between recording new music or loading a pre-recorded file from storage.

If selected, the music recording should start automatically. Depending on the microphone, the threshold in our implementation could be too low or too high, so you have to change it.

While recording with the microphone, we use something similar to a silence remover so that silence at the beginning or at the end of the audio will be cut off. After recording 30 seconds, we additionally apply a normalization filter, so that the volume of all files is high enough for the classifier to recognize the genre.

This normalized audio is saved as a .wav file in the processing folder, so that the classifier can access the file. If everything is done, the main program will receive a signal, and it continues with the genre recognition. A detailed explanation of how we implemented the classifier follows in section 3.1.2 Thinking.

For now, we skip the recognition and the dancing part, and take a look at how we allow the user to interrupt the execution of the current keyframes. Therefore, we have implemented the waiting thread, the only purpose of this thread is to detect if the music has stopped, or if the user has interrupted the keyframe execution. To achieve this, we check if the current volume of the last recorded audio sample is below a certain threshold. If that is the case, the child thread prints a warning that the music has stopped, and asks the user to increase the volume to continue the dance. If the music has stopped for too long, the thread sends a signal to the main program to stop the dance and restart the process.

### 3.1.2 Thinking

The genre of the recordings was analyzed based on 30 second long music snippets from the *sensing* part, like described in the chapter before. Now we want to implement a classifier that analyzes the music and returns the genre, and throws an error if no genre is recognizable. For the classification itself, we compared 3 classifiers:

a. A perceptron with 100 hidden units
b. A support vector machine with RBF kernel
c. A convolutional neural network

All models were trained on the GTZAN data set[1]. The features we use are the normalized Mel-frequency cepstral coefficients (MFCC). These are typically used in speech and music processing[2]. Because the recording quality of most microphones is worse than the audio quality of the GTZAN data, we integrate a **compressor** to improve performance. The compressor we use is a classic Mu-compressor as implemented in *librosa.*

Genre classification itself is a really difficult problem that we chose. If it works well, the work can be published, hence it was a very ambitious task for the project. Nevertheless we managed to solve it with an accuracy of 96% of our classifier.

A problem with genre classification is that even humans argue about the correct genre of their favorite songs. However, we want our robot to be able to recognize some music genres and to dance to it, so we had to think of some workarounds. As the genre classification itself is very hard, we decided to focus on three genres on which our robot is trained:

- pop/disco,
- classic and
- metal/rock.

### 3.1.3 Acting

As soon as the robot recognizes the music genre, he chooses the corresponding dance from the defined dictionary and performs its keyframes. While the music is playing, the robot will keep dancing and repeat the same dance again and again.

As there are two possible ways of recognizing the music, we needed different ways of stopping the music and the robot.

---

[1] https://www.tensorflow.org/datasets/catalog/gtzan
[2] https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd for an explanation

On the one hand, you can play music and record it with the microphone. It will be analyzed and the robot will start to dance. As soon as you turn the music off, the robot will stop dancing and he will take a bow.
On the other hand, you can choose to directly input a song. It will be analyzed and the robot will dance. You can interrupt him by pressing enter. Again he will stop dancing and bow.

All keyframes were generated with the software Choregraphe [3] and afterwards exported in Bezier to Python. These keyframes are stored in the directory “dance_keyframes” and imported in the according *dancing.py*.

### 3.2 Problems

The first problem we solved was the poor quality of the audio from the computer microphone, which made the classification difficult because the audio waves from the training data was really different. We solved this by using different quality enhancers, for example to filter out silent parts. Additionally, directly after recording the music we apply a volume normalization filter and before the classification step, we are using a compressor for all audio files. With this method, we improved the results of the classification by a lot.

And we had another problem that could not be solved in the end, but that has no bad side effects on the results. Our classifier needs more than 30 seconds long music files as input, making the direct recording process a bit slow. However, the classification result is very convincing, so we decided to accept the restrictions.

The biggest challenge was to merge all components together and create a functional threading concept. The robot should stay in the simspark simulator while recording an audio, and execute keyframes while waiting for the user to interrupt the dance moves. With our implementation already described in 3.1 Implementation everything works great.

Another difficulty regarding the generation of the keyframes was the usage of the lower part of the robot. It was very hard to make the robot use its legs without making him fall. To solve that we focused our dances on the upper body half.

### 3.3 Results

For the promising results, please take a look at our videos [4].

## 4. Conclusion

Even though we had some difficulties sometimes using different libraries or software, we were eventually able to manage everything.
The project was very fun and we learned a lot. We are very happy with the results, because we reached all of our goals and made the robot dance!

[3] http://doc.aldebaran.com/2-1/getting_started/installing.html
[4] https://tubcloud.tu-berlin.de/s/J9jYsGp4WJbxTEZ

# Appendix

Due to the bad audio quality from microphones, we recommend to use internal recorded sound. If it is not possible for you to set up an internal microphone successfully, you can choose to load a .wav file from the sound directory.

**Please execute the dancing.py file after installing all libraries, and notice the command line output.**

## Installation tipps:

WINDOWS:

1. Don't use headphones if you want to record with stereomix.

2. If you have trouble with the installation of pyaudio, try[5]:
   **pip install pipwin**
   **pipwin install pyaudio**

3. Set up the stereomix device in windows.[6]

4. Select stereomix microphone as "Standardaufnahmegerät", run this code and choose Stereomix at the beginning

LINUX:

1. With conda or pip install **portaudio** and **pyaudio.**

2. For using a similar device like stereomix in windows, install voice control[7] using **sudo apt-get install pavucontrol** and run it by using **pavucontrol.**

3. Run the dancing file in another terminal and select input device "pulse".

4. Select "Monitor of built-in audio" from register card "recordings" in voice control panel.

---

[5] https://stackoverflow.com/questions/55936179/trying-to-install-pyaudio-using-pip

[6] https://www.howtogeek.com/howto/39532/how-to-enable-stereo-mix-in-windows-7-to-record-audio/

[7] https://stackoverflow.com/questions/65079325/problem-with-alsa-in-speech-recognitionpython-3