Multiple Pepper synchronisation tasks

Scenario Case:

In a stage scene, multiple Peppers are controlled to perform a task synchronously by a unified command or instruction, for example, when multiple Peppers perform a neat dance on the stage, they start to perform the same dance movement synchronously by a single command or signal only, so as to achieve the purpose of cooperation among multiple Peppers.

If you want to implement a function similar to the above scenario, you can refer to the following project: pepper-task-synchroniser

Project description:

This project provides Android libraries in order to synchronise the execution of the same task or action on multiple Pepper robots, it relies on Chirp a third-party solution, where Chirp sends encrypted hypergolic waves through a transmitting device with a speaker (e.g., a laptop), and an Android application (Pepper) receives the signals. Pepper) receives the signal and performs the task.

You can just clone the project, open android receiver in Android Studio and create a new project in android-receiver, android-receiver) in Android Studio and run it directly on Pepper. At the same time, build python-sender on your computer to send signals.

Advantages (compared to traditional network synchronisation methods):

- Both sender and receiver can be in flight mode, it only requires a device with a speaker.
- The limitation of the strength of the signal is proportional to the capacity of the speaker.
- The transmission is via ultrasound, which cannot be detected by the human ear.
- The solution is scalable and supports an unlimited number of robots, as long as the robots are able to receive the signals synchronously.
- If you want to stop the task, you can stop it by simply sending the STOP signal, which is not possible with the traditional network approach.

Reference steps:

You can refer to the following steps to integrate this feature into your project.

Transmitter:

Step 1: Install external dependencies (different systems)

Mac OS

```
brew install portaudio libsndfile
```

Ubuntu

```
sudo apt-get install python3-dev python3-setuptools portaudio19-dev libffi-dev libsndfile1
```

Windows

```
# Install sounddevice from the link below
# https://www.lfd.uci.edu/~gohlke/pythonlibs/#sounddevice
pip3 install sounddevice.whl
```

Step 2: Install the chirpsdk package

The requirements.txt file contains python packages that the robot does not have by default. These packages need to be included in order to be used in the application. Cut to the folder containing the requirements.txt file and execute the command:

```
pip3 install -r requirements.txt -t external_dependencies
```

Step 3: Embed Secret Key

Generate a secret key for your application, fill in the key according to the .chirprc file template, and put this file into the root directory ~/.chirprc.

Step 4: Run

Switch to the scripts directory containing the main.py file. You can run the script in two ways.

```
- ./main.py
- python3 main.py
```

This script -u option is used to send string data. It depends on the settings of the receiving application, but here we use the following command

- go - tells the bot to execute the command
- stop - tells the bot to stop executing commands and return to listening mode
- close - disconnects from the SDK running on the robot and stops listening for any commands.

For example:

```
- ./main.py -u go
- ./main.py -u stop
- ./main.py -u close
```

Receiving end:

First step: build Android project

- Create a new Android project, if you want to use QiSDK, you need to make sure the minimum version number is API 23 Marshmellow
- Use JitPack to install the synchronisation libraries, command from the Releases tab here and select the latest version.
- Add it to the allprojects, repositories module in your project's build.gradle file:

```
allprojects {
    repositories {
        ...
        maven { url "https://maven.chirp.io/release" }
    }
}
```

- Synchronise the gradle file and the sync library will be imported into the project

Step 2: Embed the key

Generate secret key for your application, add apikey properties file (same level as gradle properties) to the root directory in your project and write it in the following format:

```
CHIRP_KEY=""
CHIRP_SECRET=""
CHIRP_CONFIG=""
```

Once the apikey properties file has been added, the following needs to be added to the build.gradle file (under plugins in the header of the file).

```
def apikeyPropertiesFile = rootProject.file("apikey.properties")
def apikeyProperties = new Properties()
apikeyProperties.load(new FileInputStream(apikeyPropertiesFile))
```

Add in the default configuration:

```
android {
    ...
    defaultConfig {
        ...
        buildConfigField("String", "APPKey", apikeyProperties['CHIRP_KEY'])
        buildConfigField("String", "APPSecret", apikeyProperties['CHIRP_SECRET'])
        buildConfigField("String", "APPConfig", apikeyProperties['CHIRP_CONFIG'])
    }
}
```

After adding, you need to sync the gradle file.

Step 3: Add speaker permissions

On the Android side of Pepper, you need to add a runtime microphone permission grant, which is added in the onResume() method:

```
private val REQUEST_RECORD_AUDIO = 1

override fun onResume() {
    super.onResume()

    if (ContextCompat.checkSelfPermission(this, Manifest.permission.RECORD_AUDIO) != PackageManager
        ActivityCompat.requestPermissions(this, arrayOf(Manifest.permission.RECORD_AUDIO), REQUEST_
    }
}
```

The following callback function is used to check if the permission is authorised by the user:

```
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<String>, grantResults:
    when (requestCode) {
        REQUEST_RECORD_AUDIO -> {
            if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED)
                // Permission granted
            }
            return
        }
    }
}
```

Step 4: Installation

In the activity, initialise TaskSynchroniser using the context of the activity:

```
private val taskSynchroniser: TaskSynchroniser by lazy {
TaskSynchroniser(this, BuildConfig.APPKey, BuildConfig.APPSecret, BuildConfig.APPConfig)
}
```

The first step requires linking the callback function. Call the initialiseSDK() method in the onCreate() method.

```
private fun initialiseSDK() {
    taskSynchroniser.chirpConnect.onReceiving { channel: Int ->
        Log.v("Synchroniser", "onReceiving on channel: $channel")
    }

    taskSynchroniser.chirpConnect.onReceived { payload: ByteArray?, channel: Int ->
        val hexData = payload?.let { String(it) }
        Log.v("Synchroniser", "onReceived: $hexData on channel: $channel")
    }
}
```

When there is a signal, these callback functions will be accepted, and if accepted successfully, the signal can be parsed into a string to perform the appropriate action.

Step 4: Start and Stop

The best practice is to add buttons in the UI to start and stop the SDK and track the status. Start the SDK in the button listener method startSDK():

```
private fun startSDK() {
if (taskSynchroniser.start()) {
// Callback here if the SDK was started successfully.
} else {
// Callback here if the SDK failed to start.
}
}
```

Stop the SDK in the button listener method stopSDK():

```
private fun stopSDK() {
    if (taskSynchroniser.stop()) {
        // Callback if SDK successfully stopped
    } else {
        // Callback if SDK failed to stop
    }
}
```

Best practice is to end the SDK in the onStop() method, called in onDestroy():

```
taskSynchroniser.close()
```

Caution:

Please note the following points during the actual runtime:

- Laptops using a MacBook Pro 13 (2018) need to have the volume adjusted to the maximum at a range of 20-25 metres where the robot can hear it, with background noise in the house at 40-50 dB.
- The robot side must be using a flat panel microphone and make sure it is not covered.
- The Pepper robot cannot be used as a transmitting device, there may be a mismatch between the microphone and the speaker.

It also supports external speakers, or have the transmitter as close to the robot as possible.