多Pepper同步任务

场景案例：

在某舞台现场，通过统一命令或者指示来控制多台 Pepper 同步执行一个任务，比如多台 Pepper 在舞台上进行表演整齐舞蹈时，仅通过一个口令或者信号，开始同步执行同一个舞蹈动作，达到多台 Pepper 合作的目的。

如需实现类似上述场景的功能，可以参考以下项目： pepper-task-synchroniser

项目简介：

此项目提供了 Android 库，为了在多台 Pepper 机器人上同步执行同一个任务或者动作，它依赖于 Chirp 第三方解决方案，Chirp 通过带有扬声器的发送设备（比如笔记本）来发送加密的超生波，同时 Android 应用（Pepper）来接收信号，从而执行任务。

您可直接clone项目，在 Android Studio 内打开 android-receiver 并在Pepper上直接运行它，同时在电脑端搭建 python-sender 用于发送信号。

优势（相比传统网络同步方式）：

- 发送者和接收者都可以处于飞行模式，它仅仅需要带有扬声器的设备。
- 信号的强弱限制和扬声器的能力或正比。
- 超声波的发送是通过超声波，不能被人耳检测到。
- 此解决方案可以扩展，它支持无限多的机器人。只要机器人能够同步接收到信号。
- 如果想要停止任务，只需要简单的发送 STOP 信号就可以停止，这是传统网络方式不能做到的。

参考步骤：

可以参考下列步骤将该功能集成到您的项目里。

发送端：

第一步：安装外部依赖（不同系统）

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
```

第二步：安装 chirpsdk 包

在 requirements.txt 文件中包含了机器人默认没有的 python 包。为了在应用中使用，需要包含这些包。 切到包含 requirements.txt 文件的文件夹，并执行命令：

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

第三步：嵌入密匙

为自己的应用生成密匙，按照 .chirprc 文件模版填写密匙，并把此文件放到根目录 ~/.chirprc

第四步：运行

切换到包含 main.py 文件的 scripts 目录， 可以通过两种方式运行脚本。

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

此脚本 -u 选项用于发送字符串数据。它取决于接收应用的设置，但是在这里我们使用以下命令

- go - 告诉机器人执行命令
- stop - 告诉机器人停止执行命令并返回到监听模式
- close - 与运行在机器人的 SDK 断开链接，并停止监听任何命令

比如：

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

接收端：

第一步：搭建 Android 工程

- 创建新的 Android 工程，如果要使用 QiSDK，需要确认最小版本号是 API 23 Marshmellow
- 使用 JitPack 来安装同步库，命令从这里的发布标签，选择最新版本。
- 在项目的 build.gradle 文件中的 allprojects, repositories 模块中添加：

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

- 同步 gradle 文件，同步库将被导入到工程中

第二步：嵌入密匙

为自己的应用生成密匙，在项目中根目录添加 apikey.properties 文件 （与 gradle.properties 同一个层级），并按照以下格式书写：

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

apikey.properties 文件添加完成，需要在 build.gradle 文件添加如下 （在文件头部的 plugins 下面）：

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

在默认配置中添加：

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

添加完成，需要同步 gradle 文件。

第三步：添加扬声器权限

在 Pepper 的 Android 端，需要添加运行时的麦克风权限授予，在 onResume() 方法中添加：

```
private val REQUEST_RECORD_AUDIO = 1

override fun onResume() {
    super.onResume()

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

下面的回调函数用于检查权限是否被用户授权：

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

第四步：安装

在 activity 中，使用 activity 的 context 初始化 TaskSynchroniser：

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

第一步需要链接回调函数， 在 onCreate() 方法中调用 initialiseSDK() 方法。

```
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")
    }
}
```

当有信号的时候，这些回调函数将被接受，如果接受成功，可以把信号解析成字符串，执行相应的动作。

第四步：开启和停止

最佳实践方式时在 UI 中添加按钮来开始和停止 SDK，开源跟状态。 在按钮监听方法 startSDK() 中开启 SDK：

```
private fun startSDK() {
    if (taskSynchroniser.start()) {
        // 如果 SDK 成功开启，在此回调
    } else {
        // 如果 SDK 开启失败，在此回调
    }
}
```

在按钮监听方法 stopSDK() 中停止 SDK：

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

最佳实践是在 onStop() 方法中结束 SDK，在 onDestroy() 中调用：

```
taskSynchroniser.close()
```

注意事项：

在实际运行过程中，请注意一下几点：

- 使用 MacBook Pro 13 (2018) 的笔记本需要把音量调节到最大，在机器人能听到的 20-25 米范围， 屋内的背景噪音在 40-50 分贝。
- 机器人端必须是使用平板的表定风， 开销定没有被覆盖。
- 不能使用 Pepper 机器人作为发送设备，可能出现麦克风和扬声器错乱

它也支持外部扬声器，或者让发送器尽量靠近机器人。