PythonSDK 用户手册

版本：V20190410

目录

1. 简介

本文档介绍 PythonSDK 的 API 详细使用方法

2. 用户接口说明

所有接口返回值的类型为元组（status,value）,status 代表调用状态，0 表示成功，-1 表示失败；value 为接口返回值。

2.1. Robot 初始化

2.1.1. Robot()

初始化一个 robot 对象。
return：返回一个机器人对象
示例：robot = Robot()

2.1.2. connect()

连接机器人。
device_name：机器人 ID，通过机器人脸部显示的 ID 号。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None
示例：result = robot.connect('Gomer_wkk')
解释：连接机器人"Gomer_wkk"，得到连接结果 result。

2.2. misc 组件

misc 包含灯光的控制接口，启动灯光前，请先设置灯光的参数，否则会按照默认参数打开灯光。

2.2.1. set_color(red, blue, green)

通过这个接口，你可以配置灯光的颜色，通过调整三原色的值，实现任意颜色的灯光，注意需要启动灯光函数"light_on"。
Red/blue/green：颜色值，范围 0~100
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：misc.set_color(50, 50, 50)
解释：设置颜色值为橙色（请自行查询 24 位色值）

2.2.2. set_blink(mode, duration, light_time, dark_time)

通过这个接口，你可以配置灯光的闪烁方式和持续时间，注意需要启动灯光函数"light_on"。
mode：0=保持常亮；1=闪烁；2=呼吸/渐变
duration：持续总时间，范围 0~30000 毫秒，当设为 65535 时，即为无限时间
light_time：闪烁模式有效，一个闪烁周期内，灯光点亮的时间，范围 0~10000 毫秒。
dark_time：闪烁模式有效，一个闪烁周期内，灯光熄灭的时间，范围 0~10000 毫秒。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：misc.set_blink(mode.BLINK, 20000, 500, 1500)
解释：设置灯光为闪烁模式，持续 20 秒，一个闪烁周期为 2000 毫秒，点亮 500 毫秒，熄灭 1500 毫秒，。

2.2.3. light_on()

启动灯光

2.2.4. light_off()

关闭灯光

2.3. motion 组件

motion，包含运动控制接口，既有开环的控制，又有闭环控制，以满足不同的需求。闭环控制，通过运动算法，精确的控制机械运动到指定位置。

2.3.1. wheels_move(speed, direction, hold_time)

车轮运动，开环方式，以时间为停止标志。
speed：运动速度，范围 0~3，0 为停止。
direction：运动方向，范围-180~180 度，0 度为正前方。
hold_time：运动时间，范围 0~10000 毫秒，当时间耗完，运动停止。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.wheels_move(3, 90, 5000)

解释：以 3 档速度向右 90 度方向运动 5000 毫秒

2.3.2. wheels_accurate_go_straight(speed, distance, wait_time)

车轮直行，闭环控制，精确到达某位置。
speed：运动速度，范围 0~3，0 为停止。
distance：运动距离，范围-1000~1000 毫米，正数为前进，负数为后退。
wait_time：等待超时时间，范围 0~10000 毫秒。当运动到达位置，立刻返回；当时间耗完，自动返回。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.wheels_accurate_go_straight(3, 200, 3000)
解释：以 3 档速度向前直行 200 毫米，最长等待 3000 毫秒

2.3.3. wheels_accurate_turn(speed, angle, wait_time)

车轮转弯，闭环控制，精确转动 X 角度。
speed：运动速度，范围 0~3，0 为停止。
angle：转弯角度，范围-360~360 毫米，正数为右转，负数为左转
wait_time：等待超时时间，范围 0~10000 毫秒。当运动到达位置，立刻返回；当时间耗完，自动返回。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.wheels_accurate_turn(3, 200, 3000)
解释：以 3 档速度向右转 200 度，最长等待 3000 毫秒

2.3.4. forearm_rotate(speed, hold_time)

小臂运动，开环方式，以时间为停止标志。
speed：运动速度，范围-3~3，0 为停止，正数为向外伸展，负数为向内收缩。
hold_time：运动时间，范围 0~10000 毫秒，当时间耗完，运动停止。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.forearm_rotate(3, 3000)
解释：小臂以 3 档速度向外伸展，持续 3000 毫秒

2.3.5. forearm_accurate_rotate(speed, angle, wait_time)

小臂运动，闭环控制，精确转动 X 角度。

speed：运动速度，范围 0~3，0 为停止。
angle：目标绝对角度，范围 0~210 度，0 度为完全收缩位置，210 度为完全伸展位置
wait_time：等待超时时间，范围 0~10000 毫秒。当运动到达位置，立刻返回；当时间耗完，自动返回。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.forearm_accurate_rotate(3, 90, 3000)
解释：小臂以 3 档速度转动到 90 度位置，最长等待 3000 毫秒

2.3.6. arm_rotate(speed, hold_time)

大臂运动，开环方式，以时间为停止标志。
speed：运动速度，范围-3~3，0 为停止，正数为向后收缩，负数为向前伸展。
hold_time：运动时间，范围 0~10000 毫秒，当时间耗完，运动停止。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.arm_rotate(3, 3000)
解释：大臂以 3 档速度向前伸展，持续 3000 毫秒

2.3.7. arm_accurate_rotate(speed, angle, wait_time)

大臂运动，闭环控制，精确转动 X 角度。
speed：运动速度，范围 0~3，0 为停止。
angle：目标绝对角度，范围 0~160 度，0 度为完全收缩位置，160 度为完全伸展位置
wait_time：等待超时时间，范围 0~10000 毫秒。当运动到达位置，立刻返回；当时间耗完，自动返回。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.arm_accurate_rotate(3, 90, 3000)
解释：大臂以 3 档速度转动到 90 度位置，最长等待 3000 毫秒

2.3.8. head_rotate(speed, hold_time)

头部运动，开环方式，以时间为停止标志。
speed：运动速度，范围-3~3，0 为停止，正数为向上抬头，负数为向下低头。
hold_time：运动时间，范围 0~10000 毫秒，当时间耗完，运动停止。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.head_rotate(3, 3000)
解释：头部以 3 档速度向上抬起，持续 3000 毫秒

2.3.9. head_accurate_rotate(speed, angle, wait_time)

头部运动，闭环控制，精确转动 X 角度。
speed：运动速度，范围 0~3，0 为停止。
angle：目标绝对角度，范围-15~65 度，-15 度为低头极限位置，0 度为正视前方，65 度为抬头极限位置。
wait_time：等待超时时间，范围 0~10000 毫秒。当运动到达位置，立刻返回；当时间耗完，自动返回。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.head_accurate_rotate(3, 40, 3000)
解释：头部以 3 档速度转动到 40 度位置，最长等待 3000 毫秒

2.3.10. paw_rotate(speed, hold_time)

手爪运动，开环方式，以时间为停止标志。
speed：运动速度，范围-3~3，0 为停止，正数为向内闭合，负数为向外张开。
hold_time：运动时间，范围 0~10000 毫秒，当时间耗完，运动停止。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.paw_rotate(3, 3000)
解释：手爪以 3 档速度向内闭合，持续 3000 毫秒

2.3.11. paw_accurate_rotate(speed, angle, wait_time)

手爪运动，闭环控制，精确转动 X 角度。
speed：运动速度，范围 0~3，0 为停止。
angle：目标绝对角度，范围 0~100 度，0 度为张开极限位置，100 度为闭合极限位置。
wait_time：等待超时时间，范围 0~10000 毫秒。当运动到达位置，立刻返回；当时间耗完，自动返回。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：motion.paw_accurate_rotate(3,100, 3000)
解释：手爪以 3 档速度闭合到 100 度位置，最长等待 3000 毫秒

2.4. audio 组件

audio，包含与声音相关的接口，TTS、音乐制作、变声、播放、停止等功能。

2.4.1. make_tts(text, file_name)

tts 合成，并保存为文件。
text：文本信息，英文 128 个字符，中文 30 个字。
file_name：保存的文件名，文件将保存在 gomer 大脑，可用作变声，或者播放。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：audio.make_tts("Hello world!", "first_file")
解释：将文本“hello world”转换为音频，保存名为 first_file 的音频文件

2.4.2. make_music(speed, music, file_name)

音乐合成，并保存为文件。
speed：节拍速度，速度 1~8
music：音乐简谱信息，最多 128 个数字。
file_name：保存的文件名，文件将保存在 gomer 大脑，可用作变声，或者播放。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：audio.make_music(2, "517334245151", "first_file")
解释：将简谱“517334245151”转换为音频，节拍速度为 2，保存名为 music_file 的音频文件

2.4.3. convert_audio(src_file, convert_type, dst_file)

音频变声，将源音频按照某类型变声为目标音频。
src_file：源音频文件
convert_type：变声类型。
dst_file：保存的文件名，文件将保存在 gomer 大脑，可用作其他用途。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：audio.convert_audio("src_audio", 2, "convert_file")
解释：将音频文件“src_audio”以类型 2 变声，保存名为 convert_file 的音频文件

2.4.4. get_volume()

获取当前音量。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时，类型为 list，无返回值时为 None

示例：ret = audio.get_volume()

```
if ret[0] < 0:
    logger.info("get volume failed!")
else:
    logger.info("volume is:", ret[1][0][0])
```

解释：获取音量大小，如果失败，提示失败，如果成功，显示音量的大小

2.4.5. set_volume(vol)

设置音量。
vol：设置的音量值，范围 0~5
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时，类型为 list，无返回值时为 None

示例：ret = audio.set_volume(3)

```
if ret[0] < 0:
    logger.info("set volume failed!")
else:
    logger.info("volume is:", ret[1][0][0])
```

解释：设置音量大小，如果失败，提示失败，如果成功，显示当前音量的大小

2.4.6. get_player_status()

获取当前播放状态。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None
空闲 value 为[(0, None)]
正忙 value 为[(1, file_name)]

示例：ret = audio.get_player_status()

```
if ret[0] < 0:
    logger.info("get status failed!")
else:
    file = ret[1][0][1]
    logger.info("now is playing:", file)
```

解释：获取播放状态，如果失败，提示失败，如果成功，显示正在播放的文件

2.4.7. play_audio(audio_file)

播放音频文件。
audio_file：播放的文件名
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.4.8. stop_player()

停止音频播放器。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.5. screen 组件

本章介绍屏幕显示接口，提供了多样化显示方式，包括用户自定义图案显示。

2.5.1. show_picture(img_file, hold_time)

将用户的图片展示在屏幕上。
img_file：图片路径，图片在 PC 上的位置，图片必须是 320x240 分辨率 jpg 格式。
hold_time：图片显示时间，范围 1000~10000 毫秒。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：screen.show_picture("C:\sdk\gomer\my_screen.jpg", 3000)
解释：在屏幕上显示用户的 my_screen.jpg 图片文件，保持 3000 毫秒

2.5.2. show_status()

获取显示屏状态
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None
空闲 value 为[(0, None)]
正忙 value 为[(1, file_name)]

示例：
```
ret = screen.show_status()
is_busy = ret[1][0][0]
if is_busy > 0:
    logger.info("screen is busy")
    current_img = ret[1][0][0]
    logger.info(current_img)
else:
    logger.info("screen is idle")
```
解释：获取显示状态，如果繁忙，显示正在显示的图片，如果空闲，提示空闲状态

2.5.3. mirror_camera()

将摄像头的画面投影到屏幕上，打开后将会一直保持，直到用户关闭。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.5.4. stop_mirror_camera()

停止摄像头投影
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.5.5. stop_showing()

停止屏幕的一切显示。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.6. emotion 组件

本章介绍情绪相关接口，提供了多样化展示情绪接口，包括用户自定义情绪的编辑接口。

2.6.1. emotion_show(name, expression_switch, voice_switch, action_switch, paw_safe)

情绪展示，且可以自由控制各个元素的开关。
name：情绪、情绪类的名称，如:情绪"wonderful"，或者"happy"，将会随机一个 happy 类中的情绪。
expression_switch：表情的开关。
voice_switch：声音的开关。
action_switch：动作的开关。
paw_safe：手爪安全，action_switch=1 时有效，当启用时，会禁止情绪中的手爪动作。

return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：emotion.emotion_show("wonderful", True, False, True, False)
解释：展示 wonderful 情绪，并关闭声音，允许手爪动作。

2.6.2. emotion_status()

获取当前情绪展示状态
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None
空闲 value 为[(0, None)]
正忙 value 为[(1, emotion_name)]

示例：
```
ret = emotion.emotion_status()
is_busy = ret[1][0][0]
if is_busy > 0:
    logger.info("emotion is busy")
    emotion_name = ret[1][0][0]
    logger.info(emotion_name)
else:
    logger.info("emotion is idle")
```
解释：获取情绪状态，如果正在展示情绪，显示正在展示的情绪，如果空闲，提示空闲

2.6.3. stop_emotion()

停止当前的情绪展示
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.7. event 组件

本章介绍已有的事件消息，事件消息是机器人自动反馈的，用户自行处理事件。

2.7.1. open_manager()

启动事件管理器，将启动监听模块，用户可使用注册接口，加入监听事件。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.7.2. close_manager()

关闭事件管理器，将关闭监听模块，不再对事件有响应。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

2.7.3. bind_event_callback(event_type, callback_func)

向事件管理器注册一个事件，并绑定事件回调函数。
event_type：事件类型
    button_click
    button_click_twice
    very_low_power
    low_power
    normal_power
    paw_issue
    forearm_issue
    arm_issue
    head_issue
    foot_issue
    posture_change。
callback_func：事件回调函数。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：event.bind_event_callback("button_click", button_click_func)
解释：注册按键单击事件，绑定按键单击事件处理函数 button_click_func。

2.8. face 组件

本章介绍人脸识别接口，将诸多人脸算法数据开放给用户，可利用数据进行再次运算和创造新的想法。

2.8.1. detect_face(image_type, min_face_size, max_face_size, min_face_score, display)

进行一次人脸识别，可以选择数据源，如从本地图片、摄像头，可以设定人脸标准。将会返回识别到的人脸坐标等信息。
image_type：选择数据源，默认摄像头"camera",图像数组"image_array",图像文件"file"。
min_face_size：最小人脸大小，范围 10~300，默认 10
max_face_size：最大人脸大小，范围 10~300，默认 300
min_face_score：最小人脸得分，范围 0~100，默认 10，该分数是对人脸的完整性和清晰度进行评分。
display：将本次检测画面显示在屏幕上。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

返回值：(0, [{"face0":[x,y,w,h,score]}, {"face1":[x,y,w,h,score]}])
返回值解释：status = 0
value = [{"face0":[x,y,w,h,score]}, {"face1":[x,y,w,h,score]}]
value 列表中是字典类型，一个人脸的参数有 5 个，起点坐标 x、y，宽高 w、h，人脸得分值 score。
示例：ret = face.detect_face("camera", 30, 300, 10, True)

```
x = ret[1][0]["face0"][0]
y = ret[1][0]["face0"][1]
width = ret[1][0]["face0"][2]
height = ret[1][0]["face0"][3]
score = ret[1][0]["face0"][4]
logger.info("face0 :", x, y, width, height, score)
```

解释：从摄像头识别一次人脸，设定人脸大小范围 30-300，人脸得分最小为 10，并在屏幕上显示图像，获取人脸的参数值并显示。

2.8.2. get_face_feature(face_index, display)

获取人脸的特征点，如眼睛、鼻子、眉毛、嘴巴的坐标，这个接口依赖于"Face.detect_face"的结果。
face_index：选择人脸索引名称,如"face0"。
display：将本次检测画面显示在屏幕上。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

返回值：(0, [{"point0":[lx,ly,rx,ry,nx,ny,lmx,lmy,rmx,rmy]}])
返回值解释：status = 0
value = [{"point0":[lx,ly,rx,ry,nx,ny,lmx,lmy,rmx,rmy]}]
value 列表中是字典类型，一个人脸的特征参数有 10 个，左眼坐标 lx、ly，右眼坐标 rx、ry，鼻子坐标 nx、ny，左嘴角坐标 lmx、lmy，右嘴角坐标 rmx、rmy。
示例：ret = face.get_face_feature("face0", True)

```
left_eye_x = ret[1][0]["point0"][0]
left_eye_y = ret[1][0]["point0"][1]
right_eye_x = ret[1][0]["point0"][2]
right_eye_y = ret[1][0]["point0"][3]
nose_x = ret[1][0]["point0"][4]
nose_y = ret[1][0]["point0"][5]
left_mouth_x = ret[1][0]["point0"][6]
left_mouth_y = ret[1][0]["point0"][7]
right_mouth_x = ret[1][0]["point0"][8]
right_mouth_y = ret[1][0]["point0"][9]
```

解释：获取人脸 face0 的特征点，并在屏幕上显示图像。

2.8.3. face_register(face_index, face_name)

将人脸起名，并录入到数据库，这个接口依赖于"Face.detect_face"的结果。
face_index：选择人脸索引名称,如"face0"。
face_name：人脸名称。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：face.face_register("face0", "Tom")
解释：将人脸 face0 录入机器人数据库，并命名为 Tom。

2.8.4. identity_recognize(face_index, display)

将人脸与机器人数据库比对，识别身份，这个接口依赖于"Face.detect_face"的结果。
face_index：选择人脸索引名称,如"face0"。
display：将本次检测画面显示在屏幕上。
返回人脸姓名，如果未知，返回"unkown"
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

```
示例：ret = face.identity_recognize("face0", True)
      if ret[0] < 0:
          logger.info("recognize face failed")
      else:
          name = ret[1][0]
          logger.info("recognize face :", name)
```
解释：将人脸 face0 与机器人数据库比对，返回结果为 Tom。

2.8.5. expression_recognize(face_index, display)

将人脸的表情，这个接口依赖于"Face.detect_face"的结果。
face_index：选择人脸索引名称,如"face0"。
display：将本次检测画面显示在屏幕上。
返回人脸的表情，如果未知，返回"normal"
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

```
示例：ret = face.expression_recognize("face0", True)
      if ret[0] < 0:
          logger.info("recognize face failed")
      else:
          expression = ret[1][0]
```

```
logger.info("recognize face expression:", expression)
```

解释：识别人脸 face0 的表情，返回结果为 happy。

2.8.6. get_face_identity(face_index, identity_name, display)

获取人脸的身份特征数据，这个接口依赖于"Face.detect_face"的结果。
face_index：选择人脸索引名称,如"face0"。
identity_name：将数据命名并保存为临时文件(在机器人大脑中)。
display：将本次检测画面显示在屏幕上。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：face.get_face_identity ("face0", "id_tom", True)
解释：获取人脸 face0 的身份特征数据，保存为"id_tom"文件，并显示图像。

2.8.7. get_face_similarity(identity_name1, identity_name2)

比对两个人脸的身份特征数据，这个接口依赖于"Face.detect_face"的结果。
identity_name1：身份特征数据文件名称，"Face.get_face_identity"的结果。
identity_name2：身份特征数据文件名称，"Face.get_face_identity"的结果。
返回相似度，范围 0~100。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

```
示例：ret = face.get_face_similarity("id_sam", "id_tom")
      if ret[0] < 0:
          logger.info("compare faces failed")
      else:
          similarity = ret[1][0]
          logger.info("face similarity:", similarity)
```

解释：比对人脸 id_sam 和 id_tom 的身份特征数据，返回相似度 80。

2.8.8. face_rename(old_name, new_name)

将人脸重新命名，不能与数据库中的人名相同。
old_name：已有的人脸名称。
new_name：新的人脸名称。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

示例：face.face_rename("tom", "sam")
解释：将数据库中的人脸 tom，重新命名为 sam。

2.8.9. get_face_list()

获取机器人数据库中的所有人脸。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

返回值：(0, [{"list":["tom","tom"]}])
返回值解释：status = 0
value = [{"list":["tom","tom"]}]
value 列表中是字典类型，字典中包含已经保存的人脸名称。

示例：ret = face.get_face_list()
name1 = ret[1][0]["list"][0]
name2 = ret[1][0]["list"][1]
解释：获取数据库中的人脸列表，得到返回"tom", "sam"。

2.9. pattern 组件

本章介绍图案识别接口，将诸多图案算法数据开放给用户。

2.9.1. detect_pattern(image_type, min_size, max_size, min_score, display)

进行一次图案识别，可以选择数据源，如从本地图片、摄像头，可以设定图案标准。将会返回识别到的图案信息。
image_type：选择数据源，默认摄像头"camera",图像数组"image_array",图像文件"file"。
min_size：最小图案大小，范围 30~300，默认 30
max_size：最大图案大小，范围 30~300，默认 300
min_score：最小图案得分，范围 0~100，默认 10，该分数是对图案的完整性和清晰度进行评分。
display：将本次检测画面显示在屏幕上。
return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

返回值：(0, [{"pattern0":[lux,luy,ldx,ldy,rdx,rdy,rux,ruy,score]}])

返回值解释：status = 0

value = [{"pattern0":[lux,luy,ldx,ldy,rdx,rdy,rux,ruy,score]}]

value 列表中是字典类型，一个图案的特征参数有 9 个，左上角 x、y，左下角 x、y，右下角 x、y，右上角 x、y，图案得分。

示例：ret = pattern.detect_face("camera", 30, 300, 10, True)

```
left_up_x = ret[1][0]["pattern0"][0]
left_up_y = ret[1][0]["pattern0"][1]
left_down_x = ret[1][0]["pattern0"][2]
left_down_y = ret[1][0]["pattern0"][3]
right_down_x = ret[1][0]["pattern0"][4]
right_down_y = ret[1][0]["pattern0"][5]
right_up_x = ret[1][0]["pattern0"][6]
right_up_y = ret[1][0]["pattern0"][7]
pattern_score = ret[1][0]["pattern0"][8]
```

解释：从摄像头识别一次图案，设定图案大小范围 30-300，图案得分最小为 10，并显示图像。

2.9.2. add_pattern(pattern_path, width, hight)

从 PC 端录入一个图案。

pattern_path：图像文件的绝对路径,图案名称应小于 16 字符。

width：图案的真实宽度，单位厘米

hight：图案的真实高度，单位厘米

return：(status, value),status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None

示例：pattern.add_pattern("C:\sdk\demo\pattern.jpg", 30, 30)

解释：将图案 pattern.jpg 录入机器人数据库，设定图案大小 30x30。

2.9.3. get_3d_position(pattern_index, display)

获取当前画面中图案的 3D 姿态，这个接口依赖于"Pattern.detect_pattern"的结果。

pattern_index：图案的索引名称。

display：将本次检测画面显示在屏幕上。

return：(status, value),status 是状态码，成功为 0，失败为-1；value 为 None

返回值：(0, [{"pattern0":[x,y,z,theta_x,theta_y,theta_z]}])

返回值解释：status = 0

value = [{"pattern0":[x,y,z,theta_x,theta_y,theta_z]}]

value 列表中是字典类型，一个图案的 3D 参数有 6 个，空间坐标 x、y、z，theta 值 theta_x、theta_y、theta_z。

示例：ret = pattern.get_3d_position("pattern0", True)

```
x = ret[1][0]["pattern0"][0]
y = ret[1][0]["pattern0"][1]
z = ret[1][0]["pattern0"][2]
theta_x = ret[1][0]["pattern0"][3]
theta_y = ret[1][0]["pattern0"][4]
theta_z = ret[1][0]["pattern0"][5]
```

解释：获取图案 pattern0 的 3D 坐标，并显示画面。

2.9.4. move_to_pattern(pattern_name, distance)

移动到图案的正面，并保持 X 距离，该接口会返回运动参数，由用户控制到达指定位置。
pattern_name：图案的名称。
distance：与图案的目标距离，单位厘米。
return：(status, value)，status 是状态码，成功为 0，失败为-1；value 是返回值，有返回值时类型为 list，无返回值时为 None
返回需要调整的路径：
value 值为[prm1, prm2, prm3, prm4, prm5]

prm1 代表是否到达位置，如果值为 101 表示已经到达位置，否则未到位
prm2 代表头部还需移动的位置，如果值为 255 表示头部已到位
prm3 代表车轮第一次还需转弯的位置，如果值为 0 表示转向到位
prm4 代表车轮还需直行的位置，如果值为 0 表示直行到位
prm5 代表车轮第二次还需转弯的位置，如果值为 0 表示转向到位

示例：

```
while True:
    # Detect or not of Pattern
    value = pattern.move_to_pattern("G", 9)
    if value[0] == 0:
        if value[1][0] == 101:
            break
        if value[1][1] != 255:
            motion.head_accurate_rotate(2, value[1][1], 3000)
        if value[1][2] != 0:
            motion.wheels_accurate_turn(1, value[1][2], 3000)
        if value[1][3]  != 0:
            motion.wheels_accurate_go_straight(2, value[1][3], 3000)
        if value[1][4] != 0:
            motion.wheels_accurate_turn(2, value[1][4], 3000)
```

解释：移动到图案 G 的正面，并保持 9 厘米距离，会返回运动路径，直到到达目标位置。

3. 范例讲解

3.1. 范例 04_play_audio.py

```
# -*- coding:utf-8 -*-

from sdk.gomer.robot import Robot


def say_hello_world(robot):
    robot.make_tts('hello world', 'test1.wav')
    robot.set_volume(1)
    robot.play_audio('test1.wav')
    robot.make_tts('hello python', 'test2.wav')
    robot.set_volume(5)
    robot.play_audio("test2.wav")


if __name__ == "__main__":
    robot = Robot()
    robot.connect('Gomer_wkk')
    say_hello_world(robot)
```

1、 首先导入 robot 接口。
2、 创建一个机器人对象，SDK 将会初始化一个机器人控制系统。
3、 启动连接机器人"Gomer_wkk"，SDK 将会搜索并连接指定的机器人。
4、 如果连接成功，你就可以通过文档中的 API，任意创作 Python 程序。
5、 范例中，将会使用"make_tts"生成一个“hello world”语音，将机器人音量设为 1，通过播放语音接口"play_audio"，你可以听到 “hello world” 声音。

4. 版权说明

5. 后记

版本日期	制定/修订人	修订记录
V20190131	魏凯凯	创建初始版本
V20190225	魏凯凯	增加详细示例代码
V20190410	魏凯凯	修改整体架构，修改接口、示例