接口说明

1 寻找传感器

在使用视触觉传感器所有 API 接口时，需要先初始化 VTSDeviceFinder 类找到对应的传感器。

1.1 初始化 VTSDeviceFinder

VTSDeviceFinder()->VTSDeviceFinder
功能：初始化 VTSDeviceFinder 类
参数：
返回值：VTSDeviceFinder 实例

1.2 返回当前所有传感器的配置参数

get_devices()->List[VTSDeviceBaseConfig]
功能：返回当前所有传感器的配置参数
参数：
返回值：List[VTSDeviceBaseConfig]

1.3 返回当前传感器的数量

count()->int
功能：返回当前传感器的数量
参数：
返回值：int

1.4 返回当前所有传感器对应的内部索引号

indexes() ->List[int]
功能： 返回当前所有传感器对应的内部索引号
参数：
返回值：List[int]

1.5 返回当前所有传感器对应的 vendorID

get_vendorIDs() ->List[str]
功能： 返回当前所有传感器对应的 vendorID
参数：
返回值：List[str]

1.6 返回当前所有传感器对应的 SN

get_sns() ->List[str]
功能： 返回当前所有传感器对应的 SN
参数：
返回值：List[str]

1.7 返回指定序列号的传感器对应的配置参数

get_device_by_sn(sn: str) ->VTSDeviceBaseConfig
功能： 返回指定序列号的传感器对应的配置参数
参数：
  sn：请根据传感器铭牌序列号或初始化 VTSDeviceFinder 日志进行获取
返回值：VTSDeviceBaseConfig

VTSDeviceBaseConfig 用于描述设备的基础配置信息
字段：
  name: str 设备名称
  vendorID: str 设备 VID
  SN: str 设备 PID
  index:int 设备内部索引号

1.8 返回指定型号的传感器对应的配置参数

get_devices_by_vendorID(vendor_id: str) ->List[VTSDeviceBaseConfig]
功能：返回指定型号的传感器对应的配置参数
参数：
  vendor_id：传感器型号
返回值：List[VTSDeviceBaseConfig]

1.1-1.8 示例如下：

```
from pyvitaisdk import VTSDeviceFinder

if __name__ == "__main__":

    finder = VTSDeviceFinder()
    # 获取所有的序列号
    print('finder.get_sns()', finder.get_sns())

    # 获取所有的 Vendor ID
    print('finder.get_vendorIDs()', finder.get_vendorIDs())

    # 打印目前链接的所有传感器信息
    print('finder.get_devices()', finder.get_devices())

    # 打印目前链接的传感器数量
```

```
print('finder.count()', finder.count())

# 打印目前链接的传感器数量
print('finder.indexes()', finder.indexes())

# 打印指定型号的传感器信息
print('finder.get_devices_by_vendorID', finder.get_devices_by_vendorID("f225"))

# 打印指定序列号的传感器信息
print('finder.get_device_by_sn', finder.get_device_by_sn("0001"))
```

2 GF225 使用

2.1 GF225 对象初始化及释放

在使用 GF225 的所有 API 接口时，都需要先调用 GF225()方法初始化 GF225 对象，不再使用该对象时通过调用该对象的 release()方法进行释放。

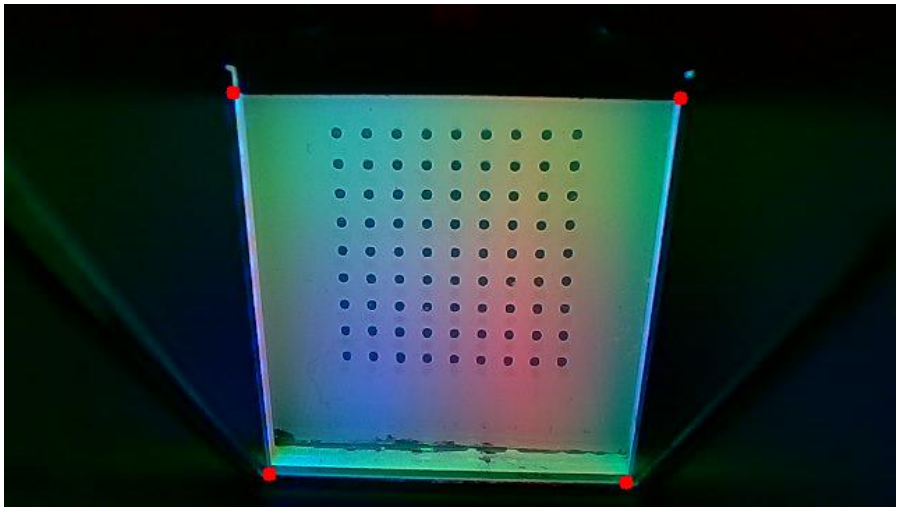
GF225(config:VTSDeviceBaseConfig，model_path: str, device:str) -> GF225
功能：初始化 GF225
参数:
config:从 VTSDeviceFinder 获取到的配置参数
model_path:深度恢复模型路径
device: ‘cpu’ or ‘cuda’
返回值：GF225 实例

2.2 设置透视变化参数

set_warp_params(self, corner_points: List[List[int]]=[], offset: List[int]=[0, 0, 0, 0], scale: float=1.0, dsize: int=240, mode: str='auto')
功能：设置透视变化参数
参数：
corner_points (List[List[int]]): 原图像中希望进行透视变换的四个点.顺序依次为左上、右上、右下、左下. mode 为’manual’时有效
offset: List[int]:通过设置 offset 过滤图像边缘黑色区域 [top, bottom, left, right]
scale (float): 缩放比例
dsize (int, optional): 透视变换后输出的有效图像大小
mode: 模式， auto、manual

返回值：

manual 模式下建议四个点如下图所示：



2.3 获取一帧图像

read() -> Tuple[bool,np.ndarray, np.ndarray]
功能：获取一帧图像
参数：
返回值：Tuple[bool,np.ndarray, np.ndarray] #是否读取成功， 原始图像，变化后图像

2.4 对相机 flush

flush(nums: int)
功能：对相机 flush 指定的帧数
参数：
  nums (int): 希望 flush 的帧数
返回值：

2.1-2.4 示例如下：

```
...
finder = VTSDeviceFinder()
if len(finder.get_sns()) == 0:
    print("No device found.")
    return
sn = finder.get_sns()[0]
print(f"sn: {sn}")
config = finder.get_device_by_sn(sn)
gf225 = GF225(config=config)
# 修改参数
offset = [5, 45, 25, 25]
dsize = 240
mode = 'auto'
gf225.set_warp_params(offset=offset, dsize=dsize, mode=mode)
gf225.flush(30)

while 1:
```

```
    ret, raw_frame, warped_frame = gf225.read()
    if ret:
        cv2.imshow(f"raw_frame", raw_frame)
        cv2.imshow(f"warped_frame", warped_frame)
    key = cv2.waitKey(1) & 255
    if key == 27 or key == ord("q"):
        break
gf225.release()
...
```

2.5 启动后端

start_backend()
功能：启动后端,会启动一个后台线程持续获取图像.
参数：
返回值：

2.6 获取一帧原始图像

get_raw_frame() -> np.ndarray:
功能：获取一帧原始图像（与 start_backend() 配合使用，该接口返回最新的一帧原始图像）
参数：
返回值：np.ndarray

2.7 获取一帧变换后图像

get_warped_frame() -> np.ndarray:
功能：获取一帧变换后图像（与 start_backend() 配合使用，该接口返回最新的一帧变换后图像）
参数：
返回值： np.ndarray

2.8 停止后端

stop_backend()
功能：停止后端
参数：
返回值：

2.9 释放对象

release()
功能：释放对象
返回值：

2.5-2.9 示例如下：

```
...
```

```
finder = VTSDeviceFinder()
if len(finder.get_sns()) == 0:
    print("No device found.")
    return
sn = finder.get_sns()[0]
print(f"sn: {sn}")
config = finder.get_device_by_sn(sn)
gf225 = GF225(config=config)
# 修改参数
offset = [5, 45, 25, 25]
dsize = 240
mode = 'auto'
gf225.set_warp_params(offset=offset, dsize=dsize, mode=mode)
gf225.start_backend()
while 1:
    cv2.imshow(f"get_raw_frame", gf225.get_raw_frame())
    cv2.imshow("get_warped_frame", gf225.get_warped_frame())
    key = cv2.waitKey(1) & 0xFF
    if key == 27 or key == ord("q"):
        break
gf225.release()
gf225.stop_backend()
...
```

2.10 判断是否已经初始化 Marker

is_inited_marker() -> bool:
功能：是否已经初始化 Marker
参数：
返回值：bool

2.11 初始化 Marker

init_marker(image:np.ndarray):
功能： 初始化 Marker
参数：
image：用来进行初始化的一帧的有效图像
返回值：

2.12 对当前图像进行 Marker 的追踪

tracking(image:np.ndarray) :
功能： 对当前图像进行 Marker 的追踪
参数：
image：用来进行跟踪的一帧的变化后图像
返回值：

2.13 返回初始帧按照行列排布的 mark 点坐标

get_origin_markers() -> np.ndarray:
功能： 返回初始帧按照行列排布的 mark 点坐标
参数：

返回值：np.ndarray

2.14 返回当前帧按照行列排布的 mark 点坐标

get_markers() -> np.ndarray:
功能： 返回当前帧按照行列排布的 mark 点坐标
参数：
返回值：np.ndarray

2.15 在图像上绘制 Marker 移动流

draw_flow(frame:np.ndarray, flow)：
功能： 在图像上绘制 Marker 移动流
参数：
frame:np.ndarray
flow:移动流
返回值：

2.10-2.15 示例如下：

```
...
warped_frame = gf225.get_warped_frame()
if not gf225.is_inited_marker():
    gf225.init_marker(warped_frame)
else:
    warped_frame_copy = warped_frame.copy()
    flow = gf225.tracking(warped_frame_copy)
    gf225.draw_flow(warped_frame_copy, flow)
    print(f"vts.get_origin_markers(): {gf225.get_origin_markers()}")
    print(f"vts.get_markers(): {gf225.get_markers()}")

cv2.imshow(f"tracking image", warped_frame)
...
```

2.16 获取当前标志位的三维向量

get_3d_vector(frame:np.ndarray) -> np.ndarray:
功能： 获取当前标志位的三维向量
参数：
frame: 图像
返回值： np.ndarray, shape 为 81*3 的 np 数组,依次为从左上角开始,先行后列的标志点坐标（x，y，z）

2.16 示例如下：

```
...
vector = gf225.get_3d_vector(frame)
...
```

2.17 进行标定

calibrate(nums: int)
功能：标定，设置背景信息
参数：
  nums (int): 希望标定的帧数
返回值：

2.18 进行再标定

re_calibrate(nums: int)
功能：再标定，用于更新背景信息
参数：
  nums (int): 希望标定的帧数
返回值：

2.19 判断是否已经标定

is_calibrate() -> bool
功能：判断是否已经标定
参数：
返回值：bool

2.20 启用滑动检测

enable_slip_detect()
功能：启用滑动检测
参数：
返回值：

2.21 停用滑动检测

disable_slip_detect()
功能：停用滑动检测
参数：
返回值：

2.22 查看滑动状态

slip_state() -> SlipState
功能：滑动状态
参数：
返回值：
UNKNOWN = 0 # 未知
CONTACT = 1 # 接触

```
INCIPIENT_SLIP    = 2        # 初始滑移
PARTIAL_SLIP      = 3        # 部分滑移
COMPLETE_SLIP     = 4        # 完全滑移
NO_OBJ            = 5        # 没有物体
STEADY_HOLD       = 6        # 静止保持
```

2.17-2.22 示例如下：

```
...
gf225.start_backend()
calib_num = 10
slip_state = gf225.slip_state()
gf225.calibrate(calib_num) # 启动标定
while 1:
    frame = gf225.get_warped_frame()
    if gf225.is_calibrate():
        slip_state = gf225.slip_state()
    frame_copy = frame.copy()
    put_text_to_image(frame_copy, slip_state.name)
    cv2.imshow(f"frame", frame_copy)
    key = cv2.waitKey(1) & 0xFF
    if key == 27 or key == ord("q"):
        break
    elif key == ord("e"):
        # 按 e 开启滑动检测
        gf225.enable_slip_detect()
    elif key == ord("d"):
        # 按 d 关闭滑动检测
        gf225.disable_slip_detect()
    elif key == ord('r'):
        gf225.re_calibrate(calib_num) # 重新标定

gf225.stop_backend()
...
```

2.23 设置背景图

```
set_background(image:np.ndarray)
功能：设置背景图
参数：
  image：从传感器中读取到的变换后的图像
返回值：
```

2.24 清除背景图

```
clear_background()
功能：清除背景图
参数：
返回值：
```

2.25 判断背景图是否已设置

is_background_init() -> bool 功能：判断背景图是否已设置 参数： 返回值：bool

2.26 进行 3d 重构

recon3d(image:np.ndarray) 功能：基于传入的图像进行 3d 重构 参数： 返回值：

2.27 获取深度图

get_depth_map() -> np.ndarray 功能：获取深度图,与 recon3d(image)配合使用 参数： 返回值：np.ndarray

2.23-2.27 示例如下：

```
...
gf225.start_backend()
bg = gf225.get_warped_frame()
gf225.set_background(bg)
while 1:
    frame = gf225.get_warped_frame()
    if gf225.is_background_init():
        gf225.recon3d(frame)
        depth_map = gf225.get_depth_map()
        cv2.imshow(f"depth_map", depth_map)
        cv2.imshow(f"diff image", cv2.subtract(frame, bg))

        frame_copy = frame.copy()
        put_text_to_image(frame_copy, str(np.max(depth_map)))
        cv2.imshow(f"warped_frame", frame_copy)

    key = cv2.waitKey(1) & 0xFF
    if key == 27 or key == ord("q"):
        break
    elif key == ord("e"):
        # 按 e 重新设置背景图
        gf225.clear_background()
        gf225.set_background(frame)

gf225.stop_backend()
gf225.release()
...
```