机器人导论

课程作业: assignment 4

组员：16305204 郑佳豪

16326086 王润锋

提交日期：2019-10-18

Deadline：2019-10-20

1. 任务概要：

- 在给定的赛道中，实现多车道的避障和视觉巡线。
- 避障算法没有限定，可使用人工势场、RRT 等路径规划算法。
- 车体大小、赛道信息以及参考资料在此处下载。

姓名	学号	比例	具体任务
郑佳豪	**16305204**	**70%**	**搭建模型、设计并实现多车道巡线算法、实现多车道避障算法、完成实验报告**
王润锋	**16326086**	**30%**	**设计多车道避障算法、完成实验报告、录制实验视频**

2. 完成情况：

总体完成情况如下：

- 已学习并能初步使用 V-REP Python Remote API 接口，实现 Python 与 V-REP 的功能交互。
- 已实现基于视觉传感器的多车道巡线功能。
- 已实现基于视觉传感器的多车道避障功能。

1. V-REP Python Remote API

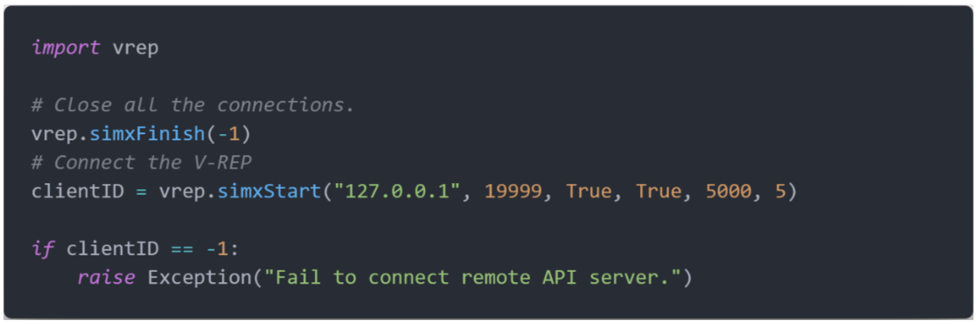
由于在上次实验中，我们已初步尝试 V-REP Remote API 接口的使用，积累了部分经验，并且考虑到本次实验任务的复杂性，因此在本次实验中，我们使用 V-REP Python Remote API 构建机器人的多车道巡线和避障算法。

首先，我们为机器人模型添加 Non-threaded Script，其具体内容如下。在 sysCall_init 函数中，我们在端口 19999 开启了 Remote API 服务。

```
function sysCall_init()
  simRemoteApi.start(19999)
end
```

1　MyBot Non-threaded 控制脚本

为了成功使用 Python 与 V-REP 交互，我们需要导入 remoteApiBindings 至项目文件夹，具体目录为 V-REP 安装目录下的 programming\remoteApiBindings，我们只需导入 vrep.py、vrepConst.py 和 remoteApi.dll 文件。我们编写简单的 Python 代码，测试 Remote API 是否调用成功，具体代码如下。

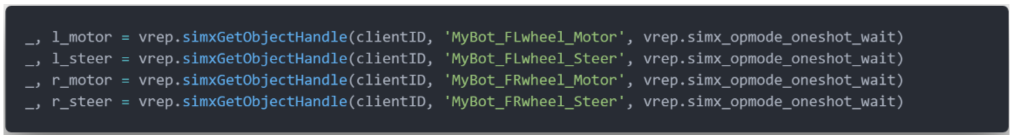


```python
import vrep

# Close all the connections.
vrep.simxFinish(-1)
# Connect the V-REP
clientID = vrep.simxStart("127.0.0.1", 19999, True, True, 5000, 5)

if clientID == -1:
    raise Exception("Fail to connect remote API server.")
```

2　测试 Remote API 是否建立成功

点击 V-REP 仿真运行按钮，随后执行上述脚本，若无发生异常，说明 Remote API 建立成功。

我们可通过 vrep.simxGetObjectHandle 获取 V-REP 仿真环境下的物体句柄。



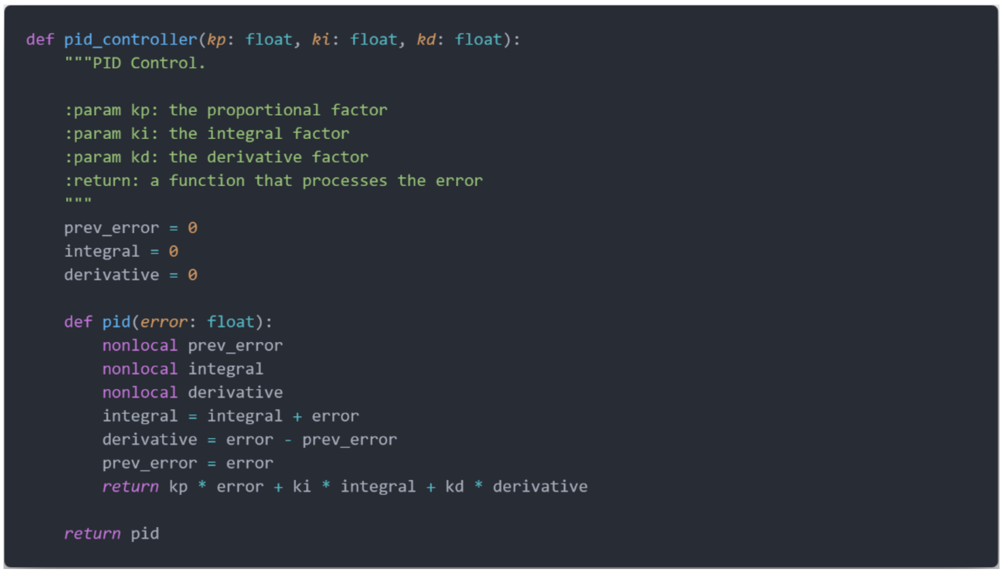
```python
_, l_motor = vrep.simxGetObjectHandle(clientID, 'MyBot_FLwheel_Motor', vrep.simx_opmode_oneshot_wait)
_, l_steer = vrep.simxGetObjectHandle(clientID, 'MyBot_FLwheel_Steer', vrep.simx_opmode_oneshot_wait)
_, r_motor = vrep.simxGetObjectHandle(clientID, 'MyBot_FRwheel_Motor', vrep.simx_opmode_oneshot_wait)
_, r_steer = vrep.simxGetObjectHandle(clientID, 'MyBot_FRwheel_Steer', vrep.simx_opmode_oneshot_wait)
```

3　获取 V-REP 物体句柄

2.　工具函数

在实现中，我们封装了一些可复用的工具函数，如 pid_controller、motor、steer、get_image 等函数。

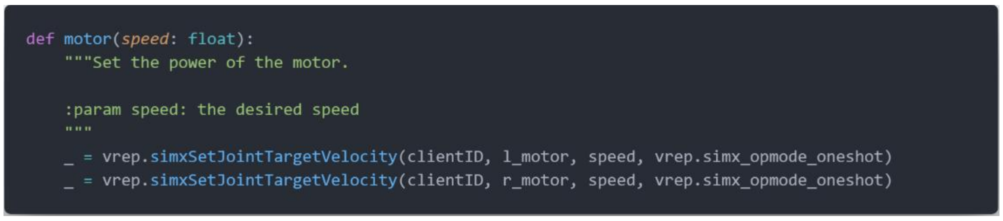
在 PID 控制中，我们输入偏差量，输出调节量，涉及到 3 个控制器的参数 kp、ki、kd。因此，我们使用 Python Clousure 技术封装 PID 控制单元，具体代码如下。在 pid_controller 函数中，我们接收 3 个控制器参数，输出一个函数，该函数接收偏差量输入，输出调节量。



```python
def pid_controller(kp: float, ki: float, kd: float):
    """PID Control.

    :param kp: the proportional factor
    :param ki: the integral factor
    :param kd: the derivative factor
    :return: a function that processes the error
    """
    prev_error = 0
    integral = 0
    derivative = 0

    def pid(error: float):
        nonlocal prev_error
        nonlocal integral
        nonlocal derivative
        integral = integral + error
        derivative = error - prev_error
        prev_error = error
        return kp * error + ki * integral + kd * derivative

    return pid
```

4　PID 控制模块

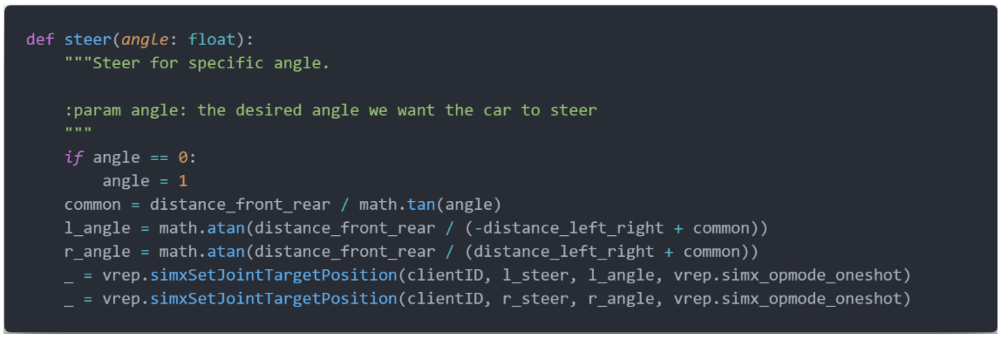
为了控制前进电机的动力输出，我们封装了 motor 工具函数，具体代码如下。该函数接收速度大小，调用 simxSetJointTargetVelocity 方法控制电机的动力输出。



```python
def motor(speed: float):
    """Set the power of the motor.

    :param speed: the desired speed
    """
    _ = vrep.simxSetJointTargetVelocity(clientID, l_motor, speed, vrep.simx_opmode_oneshot)
    _ = vrep.simxSetJointTargetVelocity(clientID, r_motor, speed, vrep.simx_opmode_oneshot)
```

5 motor 控制模块

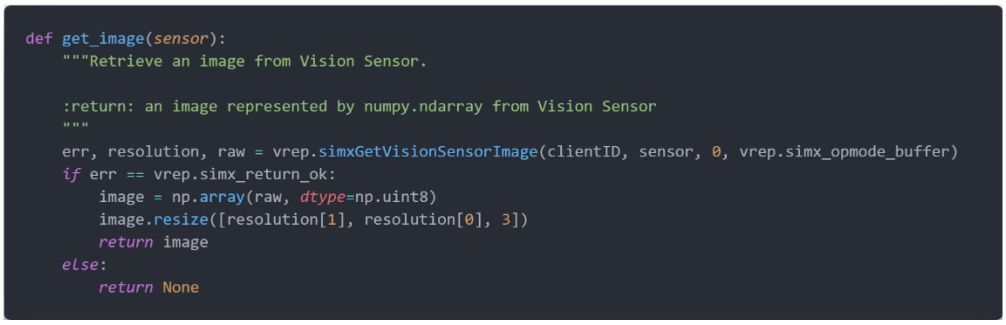
在实验中，我们使用 Ackermann 作为机器人的运动学模型。在该模型下，对于一个转向角度，两个转向轮的转向角度是需要根据运动学模型计算出来的。为此，我们实现了 steer 工具函数。



```python
def steer(angle: float):
    """Steer for specific angle.

    :param angle: the desired angle we want the car to steer
    """
    if angle == 0:
        angle = 1
    common = distance_front_rear / math.tan(angle)
    l_angle = math.atan(distance_front_rear / (-distance_left_right + common))
    r_angle = math.atan(distance_front_rear / (distance_left_right + common))
    _ = vrep.simxSetJointTargetPosition(clientID, l_steer, l_angle, vrep.simx_opmode_oneshot)
    _ = vrep.simxSetJointTargetPosition(clientID, r_steer, r_angle, vrep.simx_opmode_oneshot)
```

6 steer 控制模块

由于本次实验的目的是使用视觉传感器实现多车道的巡线和避障，因此从视觉传感器获取图像的过程是重要的，也是可复用的。这里，我们实现了 get_image 工具函数。在该函数中，我们调用 simxGetVisionSensorImage 方法，获取指定视觉传感器的图像，并调用 numpy 处理原数据，返回可供我们操作的图像数据。



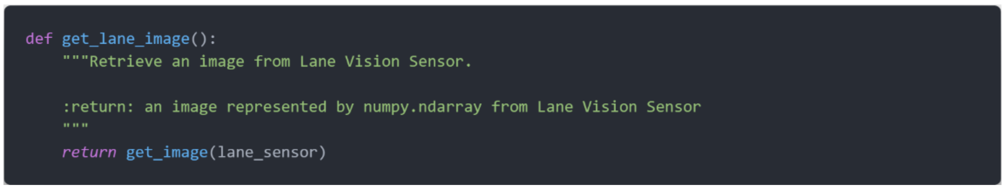
```python
def get_image(sensor):
    """Retrieve an image from Vision Sensor.

    :return: an image represented by numpy.ndarray from Vision Sensor
    """
    err, resolution, raw = vrep.simxGetVisionSensorImage(clientID, sensor, 0, vrep.simx_opmode_buffer)
    if err == vrep.simx_return_ok:
        image = np.array(raw, dtype=np.uint8)
        image.resize([resolution[1], resolution[0], 3])
        return image
    else:
        return None
```

7 获取视觉传感器图像

3. 多车道巡线

为了实现机器车的多车道巡线，我们采取以下的视觉巡线策略：**获取每一帧图像，确定图像的最大连通区域，并计算其几何中心，从而计算偏向角度。**

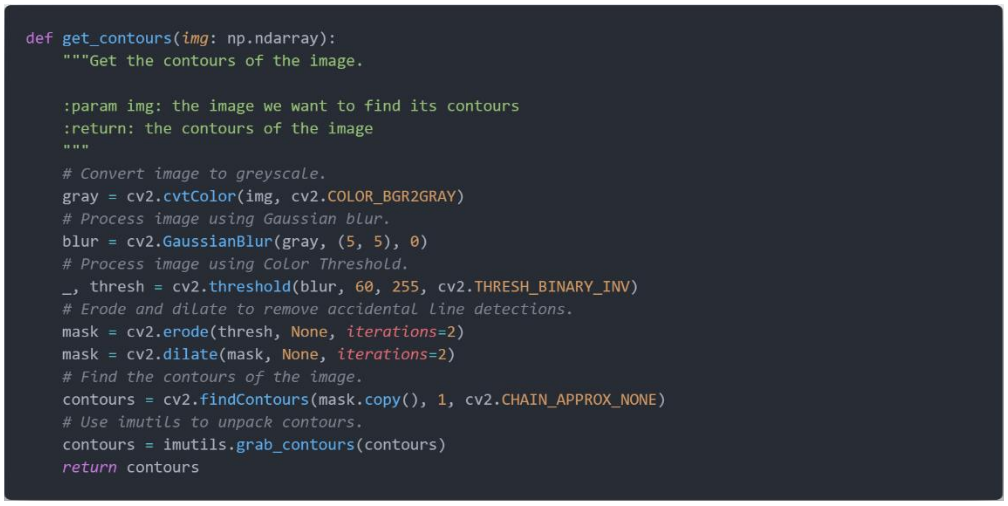
首先，我们调用 get_image 获取 Lane Vision Sensor 的图像。



```python
def get_lane_image():
    """Retrieve an image from Lane Vision Sensor.

    :return: an image represented by numpy.ndarray from Lane Vision Sensor
    """
    return get_image(lane_sensor)
```

8 获取 Lane Vision Sensor 图像

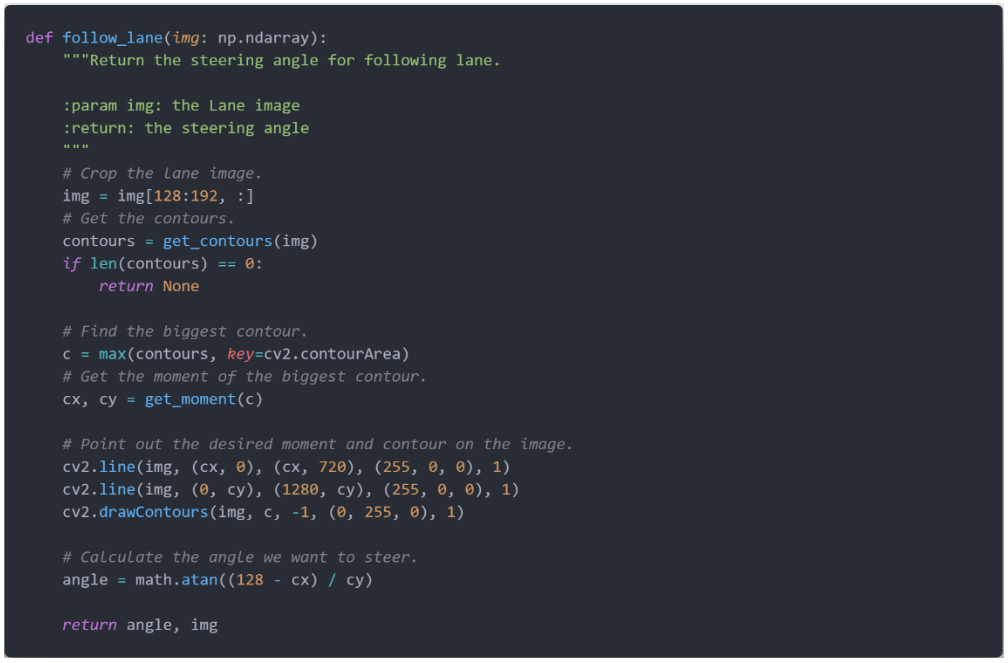
随后，我们调用 get_contours 方法确定图像上的连通区域。在此方法中，我们首先对图像进行灰度处理，随后进行高斯模糊、二值化、腐蚀和膨胀处理，最后我们调用 OpenCV 的 findContours 方法，确定图像的连通区域。注意到，我们最后使用了 imutils 处理 contours 数据，这是出于数据格式转换的目的。



```python
def get_contours(img: np.ndarray):
    """Get the contours of the image.

    :param img: the image we want to find its contours
    :return: the contours of the image
    """
    # Convert image to greyscale.
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Process image using Gaussian blur.
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    # Process image using Color Threshold.
    _, thresh = cv2.threshold(blur, 60, 255, cv2.THRESH_BINARY_INV)
    # Erode and dilate to remove accidental line detections.
    mask = cv2.erode(thresh, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)
    # Find the contours of the image.
    contours = cv2.findContours(mask.copy(), 1, cv2.CHAIN_APPROX_NONE)
    # Use imutils to unpack contours.
    contours = imutils.grab_contours(contours)
    return contours
```

9　获取图像的连通区域

在获取到图像的所有连通区域后，我们找出最大的连通区域，计算它的几何中心，随后根据几何中心的坐标，我们计算偏向角度，具体代码如下。



```python
def follow_lane(img: np.ndarray):
    """Return the steering angle for following lane.

    :param img: the Lane image
    :return: the steering angle
    """
    # Crop the Lane image.
    img = img[128:192, :]
    # Get the contours.
    contours = get_contours(img)
    if len(contours) == 0:
        return None

    # Find the biggest contour.
    c = max(contours, key=cv2.contourArea)
    # Get the moment of the biggest contour.
    cx, cy = get_moment(c)

    # Point out the desired moment and contour on the image.
    cv2.line(img, (cx, 0), (cx, 720), (255, 0, 0), 1)
    cv2.line(img, (0, cy), (1280, cy), (255, 0, 0), 1)
    cv2.drawContours(img, c, -1, (0, 255, 0), 1)

    # Calculate the angle we want to steer.
    angle = math.atan((128 - cx) / cy)

    return angle, img
```

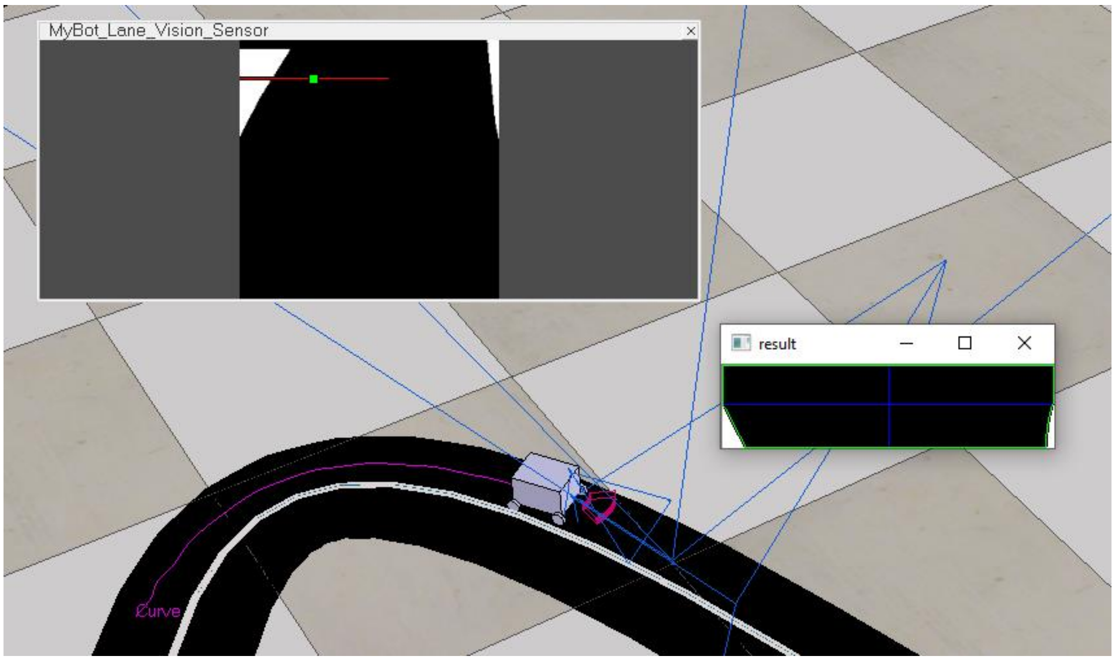
10　计算转向角度

以下是多车道巡线算法的核心流程：根据图像获取转向角度，从而实现视觉巡线。

```python
# Retrieve the image from Lane Vision Sensor.
lane_raw = get_lane_image()
# Follow the lane.
ret = follow_lane(lane_raw.copy())
# Steer for calculated angle.
angle, img = ret
steer(pid(angle))
```

11　多车道巡线算法核心流程

算法实际运行情况如下图所示，我们可以看到标题为 result 的窗口，这显示当前连通区域的几何中心。



12　多车道巡线算法的实际运行情况

4. 多车道避障

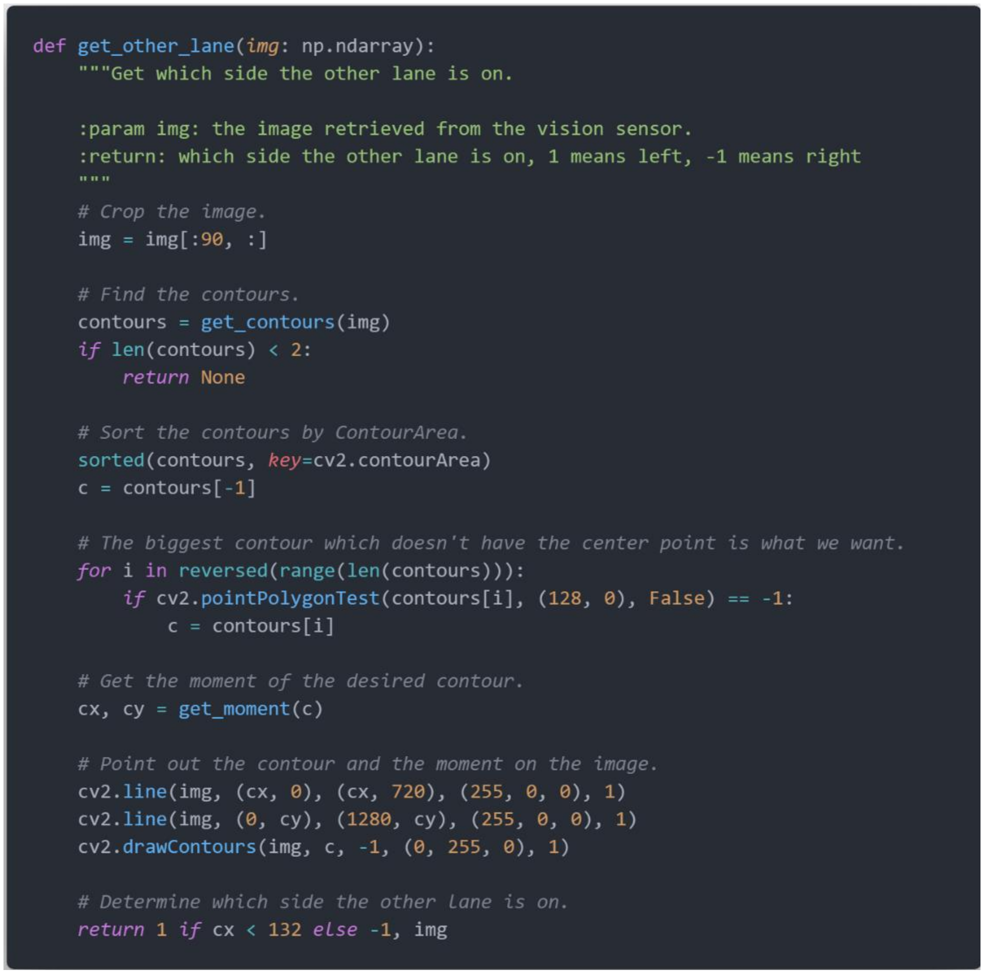
为了实现机器车的多车道避障，我们使用以下的避障策略：**获取每一帧图像，确定当前车道和其他车道。当在当前车道遇到障碍物时，转弯进入其他车道，从而避障**。

首先，我们调用 get_image 获取 Obstacle Vision Sensor 的图像。

```python
def get_obstacle_image():
    """Retrieve an image from Obstacle Vision Sensor.

    :return: an image represented by numpy.ndarray from Obstacle Vision Sensor
    """
    return get_image(obstacle_sensor)
```

13　获取 Obstacle Vision Sensor 图像

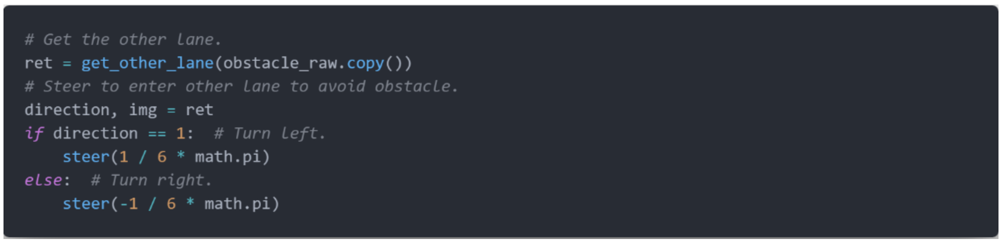
我们在 get_other_lane 函数中实现了在图像中识别当前车道和其他车道的功能，具体思路是：**确定图像的连通区域，并对其进行从大到小的排序，随后判断车辆当前点位于哪一个较大的连通区域，该区域即为当前车道，而不含车辆当前点的最大连通区域即为其他车道**。



```python
def get_other_lane(img: np.ndarray):
    """Get which side the other lane is on.

    :param img: the image retrieved from the vision sensor.
    :return: which side the other lane is on, 1 means left, -1 means right
    """
    # Crop the image.
    img = img[:90, :]

    # Find the contours.
    contours = get_contours(img)
    if len(contours) < 2:
        return None

    # Sort the contours by ContourArea.
    sorted(contours, key=cv2.contourArea)
    c = contours[-1]

    # The biggest contour which doesn't have the center point is what we want.
    for i in reversed(range(len(contours))):
        if cv2.pointPolygonTest(contours[i], (128, 0), False) == -1:
            c = contours[i]

    # Get the moment of the desired contour.
    cx, cy = get_moment(c)

    # Point out the contour and the moment on the image.
    cv2.line(img, (cx, 0), (cx, 720), (255, 0, 0), 1)
    cv2.line(img, (0, cy), (1280, cy), (255, 0, 0), 1)
    cv2.drawContours(img, c, -1, (0, 255, 0), 1)

    # Determine which side the other lane is on.
    return 1 if cx < 132 else -1, img
```

14　获取其他车道的左右位置

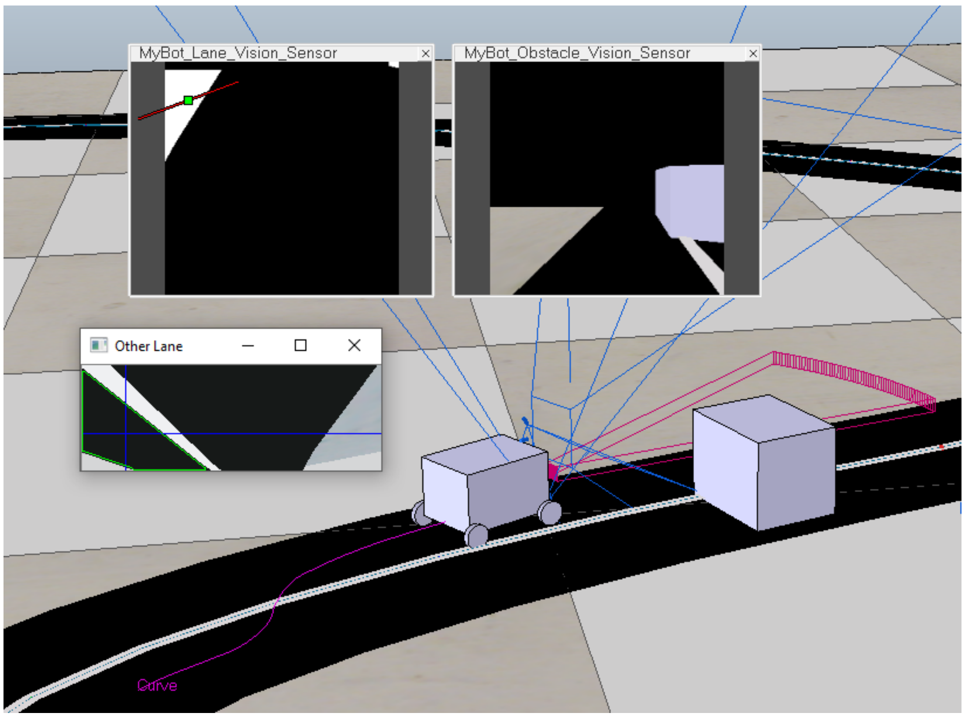
以下是多车道避障算法的核心流程：**当在当前车道检测到障碍物时，根据图像确定从当前车道转至其他车道的方向，从而实现避障**。



```python
# Get the other lane.
ret = get_other_lane(obstacle_raw.copy())
# Steer to enter other lane to avoid obstacle.
direction, img = ret
if direction == 1:  # Turn left.
    steer(1 / 6 * math.pi)
else:  # Turn right.
    steer(-1 / 6 * math.pi)
```

15　多车道避障核心流程

以下是多车道巡线和避障算法的实际运行情况。

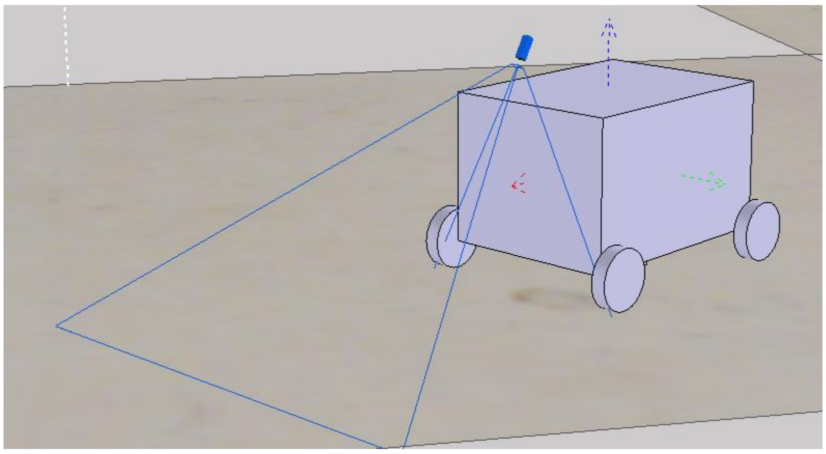


16　多车道巡线和避障算法实际运行情况

在本次实验中，我们一开始尝试使用**人工势场**方法实现视觉避障功能，但由于能力原因，并未成功将其实现。我们的基本思路是：利用**内参矩阵**对视觉传感器返回的图像进行变换，得到图像中的点在**世界坐标系**的表示，随后应用人工势场方法，目标点选取在障碍物后，得到避障路线。在实现过程中，我们在对图像中的障碍物的识别过程遇到了难以解决的问题，因此放弃了此方案。

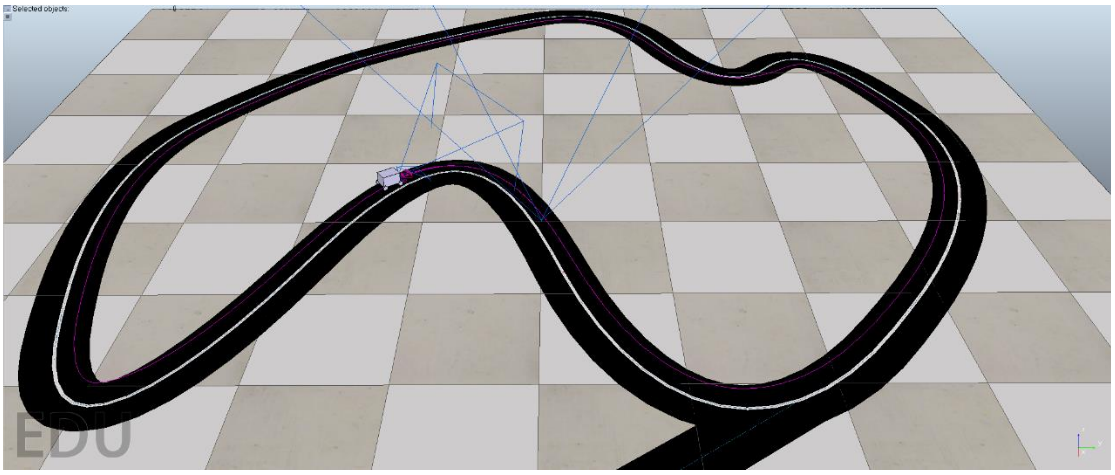
3. 效果展示：

- **多车道巡线**
  以下是机器人的静态演示图，位于车顶的是视觉传感器，用于多车道巡线功能。



17　多车道巡线静态演示

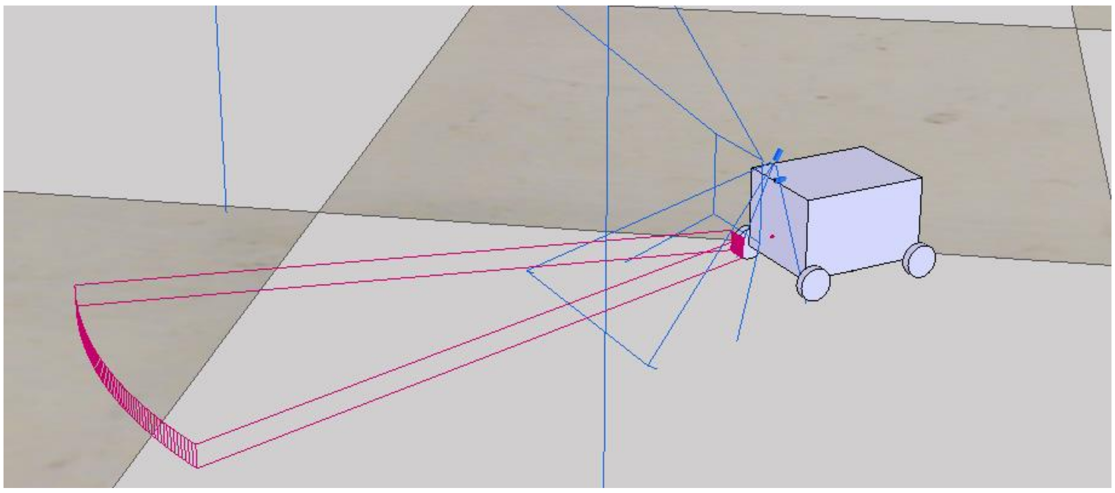
下面是机器人多车道巡线的示意图，详细细节可参考视频《机器人导论 HW4 VREP 下实现多车道巡线》，若无法跳转超链接，可复制并打开以下视频链接：https://www.bilibili.com/video/av71601540



18　多车道巡线动态演示

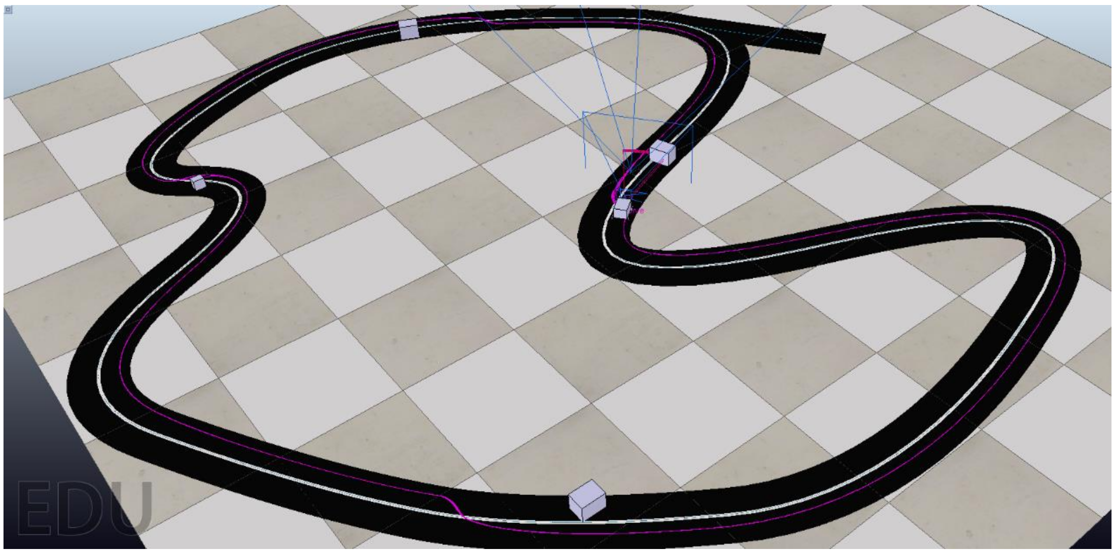
- **多车道巡线和避障**

以下是多车道巡线和避障机器人的静态演示图。



19　多车道巡线和避障静态演示

下面是机器人多车道巡线和避障的示意图，详细细节可参考《机器人导论 HW4 VREP 下实现多车道巡线和避障》，若无法跳转超链接，可复制并打开以下视频链接：https://www.bilibili.com/video/av71600529



20　多车道巡线和避障动态演示

4. 存在问题：

本次实验，内容是多车道巡线和避障的结合，还是颇有挑战性的。我们遇到了几个具备挑战性的问题，最终大多数都被解决了，但仍有遗留问题。

在模型构建上，我们使用了 V-REP Python Remote API 实现机器人的巡线和避障算法，由于在上次实验中，我们已经踩过 Remote API 的坑，所以本次实验我们并未在 Remote API 部分耗费太多时间。

我们将本次实验任务划分为了两部分：多车道巡线、多车道避障。在多车道巡线部分，我们是一开始就确定要从连通区域的几何中心切入。但我们在计算连通区域时，遇到了 OpenCV 接口不兼容的问题，在查阅几篇技术文章后，我们引入了 imutils 包，对 findContours 返回数据进行了封装处理，这才解决了问题。

在多车道避障部分，我们一开始是打算使用内参矩阵对图像进行坐标变换，从而使用人工势场路径规划，计算出避障路线。但由于我们能力原因，并未将其成功实现。但在查阅人工势场相关资料的过程中，我们对该方法有了更为深入的理解，这也算是本次实验的一个大收获。我们使用另一种避障方式：根据 Proximity Sensor 判断是否在当前车道检测到障碍物，若检测到则先确定图像中其他车道的位置（是当前车道的左边还是右边），随后进行避障。在成功将该算法实现后，我们发现其效果挺不错的。但由于我们使用了 Proximity Sensor 探测障碍物，而并未使用视觉障碍物检测，这是我们本次实验没有完成的任务，很遗憾。

5. 附录：

MyBot Non-threaded 控制代码

```
function sysCall_init()
  simRemoteApi.start(19999)
end
```

lane_follower.py 多车道巡线控制代码

```
from utils import *


def main():
    # Initialize the simulation.
    init()

    # PID Parameters
    # Motor Kp Ki Kd
    # 5 0.01 0 0
    # 10 0.0001 0.0005

    pid = pid_controller(1, 0.0001, 0.0005)
    # Set the Power!
    motor(10)

    # Start the main loop.
    while vrep.simxGetConnectionId(clientID) != -1:
        # Retrieve the image from Lane Vision Sensor.
        lane_raw = get_lane_image()
        if lane_raw is None:
            continue

        # Follow the lane.
        ret = follow_lane(lane_raw.copy())
        if ret is None:
            continue

        # Steer for calculated angle.
        angle, img = ret
        steer(pid(angle))

        # Show the processed image.
        cv2.imshow("result", img)

        # When we press "q", it quits the simulation.
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


if __name__ == '__main__':
    main()
```

lane_follower_with_obstacle_detection.py 多车道巡线和避障控制代码

```
from utils import *


def main():
    # Initialize the simulation.
    init()

    # PID Parameters
    # Motor Kp Ki Kd
    # 10 1 0.00001 0.0001

    pid = pid_controller(1, 0.00001, 0.0001)
    # Set the Power!
    motor(10)

    # It is used to avoid obstacle.
    avoid_obstacle_time = 0

    # Start the main loop.
    while vrep.simxGetConnectionId(clientID) != -1:
        simulation_time = vrep.simxGetLastCmdTime(clientID)
        if simulation_time - avoid_obstacle_time <= 1000:
            continue

        # Read the Proximity Sensor to determine whether we detect the
obstacle.
        ret = vrep.simxReadProximitySensor(clientID, proximity_sensor,
vrep.simx_opmode_streaming)
        if ret[1] is False:  # When there is no obstacle.
            # Retrieve the image from Lane Vision Sensor.
            lane_raw = get_lane_image()
            if lane_raw is None:
                continue

            # Follow the lane.
            ret = follow_lane(lane_raw.copy())
            if ret is None:
```

```python
                continue

            # Steer for calculated angle.
            angle, img = ret
            steer(pid(angle))

            # Show the processed image.
            cv2.imshow("Lane", img)
        else:  # When there is an obstacle.
            obstacle_raw = get_obstacle_image()
            if obstacle_raw is None:
                continue

            # Get the other lane.
            ret = get_other_lane(obstacle_raw.copy())
            if ret is None:
                continue

            # Steer to enter other lane to avoid obstacle.
            direction, img = ret
            if direction == 1:  # Turn left.
                steer(1 / 6 * math.pi)
            else:  # Turn right.
                steer(-1 / 6 * math.pi)

            # Set the time for avoiding obstacle.
            avoid_obstacle_time = simulation_time

            # Show the processed image.
            cv2.imshow("Other Lane", img)

        # When we press "q", it quits the simulation.
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break


if __name__ == "__main__":
    main()
```

utils.py 工具函数

```python
import time

import cv2
import imutils
import math
import numpy as np

import vrep

# Close all the connections.
vrep.simxFinish(-1)
# Connect the V-REP
clientID = vrep.simxStart("127.0.0.1", 19999, True, True, 5000, 5)

if clientID == -1:
    raise Exception("Fail to connect remote API server.")

_, l_motor = vrep.simxGetObjectHandle(clientID, 'MyBot_FLwheel_Motor',
vrep.simx_opmode_oneshot_wait)
_, l_steer = vrep.simxGetObjectHandle(clientID, 'MyBot_FLwheel_Steer',
vrep.simx_opmode_oneshot_wait)
_, r_motor = vrep.simxGetObjectHandle(clientID, 'MyBot_FRwheel_Motor',
vrep.simx_opmode_oneshot_wait)
_, r_steer = vrep.simxGetObjectHandle(clientID, 'MyBot_FRwheel_Steer',
vrep.simx_opmode_oneshot_wait)
_, lane_sensor = vrep.simxGetObjectHandle(clientID, 'MyBot_Lane_Vision_
Sensor', vrep.simx_opmode_oneshot_wait)
_, obstacle_sensor = vrep.simxGetObjectHandle(clientID, 'MyBot_Obstacle
_Vision_Sensor', vrep.simx_opmode_oneshot_wait)
_, proximity_sensor = vrep.simxGetObjectHandle(clientID, 'MyBot_Proximi
ty_Sensor', vrep.simx_opmode_oneshot_wait)

# Display processed image, just for debugging!
_, debug = vrep.simxGetObjectHandle(clientID, 'Debug_Display', vrep.sim
x_opmode_oneshot_wait)

# The distance between left wheels and right wheels, 28 is the truth di
stance(cm), 0.2 is a factor.
distance_left_right = 28 * 0.2
# The distance between front wheels and rear wheels, 30 is the truth di
stance(cm), 0.2 is a factor.
distance_front_rear = 30 * 0.2


def init():
    """Initialize the simulation.
    """
    vrep.simxGetVisionSensorImage(clientID, lane_sensor, 0, vrep.simx_o
pmode_streaming)
```

```
    vrep.simxGetVisionSensorImage(clientID, obstacle_sensor, 0, vrep.si
mx_opmode_streaming)
    time.sleep(1)


def pid_controller(kp: float, ki: float, kd: float):
    """PID Control.

    :param kp: the proportional factor
    :param ki: the integral factor
    :param kd: the derivative factor
    :return: a function that processes the error
    """
    prev_error = 0
    integral = 0
    derivative = 0

    def pid(error: float):
        nonlocal prev_error
        nonlocal integral
        nonlocal derivative
        integral = integral + error
        derivative = error - prev_error
        prev_error = error
        return kp * error + ki * integral + kd * derivative

    return pid


def steer(angle: float):
    """Steer for specific angle.

    :param angle: the desired angle we want the car to steer
    """
    if angle == 0:
        angle = 1
    common = distance_front_rear / math.tan(angle)
    l_angle = math.atan(distance_front_rear / (-
distance_left_right + common))
    r_angle = math.atan(distance_front_rear / (distance_left_right + co
mmon))
    _ = vrep.simxSetJointTargetPosition(clientID, l_steer, l_angle, vre
p.simx_opmode_oneshot)
    _ = vrep.simxSetJointTargetPosition(clientID, r_steer, r_angle, vre
p.simx_opmode_oneshot)
```

```python
def motor(speed: float):
    """Set the power of the motor.

    :param speed: the desired speed
    """
    _ = vrep.simxSetJointTargetVelocity(clientID, l_motor, speed, vrep.
simx_opmode_oneshot)
    _ = vrep.simxSetJointTargetVelocity(clientID, r_motor, speed, vrep.
simx_opmode_oneshot)


def get_image(sensor):
    """Retrieve an image from Vision Sensor.

    :return: an image represented by numpy.ndarray from Vision Sensor
    """
    err, resolution, raw = vrep.simxGetVisionSensorImage(clientID, sens
or, 0, vrep.simx_opmode_buffer)
    if err == vrep.simx_return_ok:
        image = np.array(raw, dtype=np.uint8)
        image.resize([resolution[1], resolution[0], 3])
        return image
    else:
        return None


def get_lane_image():
    """Retrieve an image from Lane Vision Sensor.

    :return: an image represented by numpy.ndarray from Lane Vision Sen
sor
    """
    return get_image(lane_sensor)


def get_obstacle_image():
    """Retrieve an image from Obstacle Vision Sensor.

    :return: an image represented by numpy.ndarray from Obstacle Vision
 Sensor
    """
    return get_image(obstacle_sensor)
```

```python
def print_image(image: np.ndarray):
    """Print the image on the V-REP.

    :param image: the image we want to display on "Debug_Display"
    """
    vrep.simxSetVisionSensorImage(clientID, debug, image.ravel(), 0, vr
ep.simx_opmode_oneshot)


def get_contours(img: np.ndarray):
    """Get the contours of the image.

    :param img: the image we want to find its contours
    :return: the contours of the image
    """
    # Convert image to greyscale.
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    # Process image using Gaussian blur.
    blur = cv2.GaussianBlur(gray, (5, 5), 0)
    # Process image using Color Threshold.
    _, thresh = cv2.threshold(blur, 60, 255, cv2.THRESH_BINARY_INV)
    # Erode and dilate to remove accidental line detections.
    mask = cv2.erode(thresh, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)
    # Find the contours of the image.
    contours = cv2.findContours(mask.copy(), 1, cv2.CHAIN_APPROX_NONE)
    # Use imutils to unpack contours.
    contours = imutils.grab_contours(contours)
    return contours


def get_moment(contour):
    """Get the moment of the contour.

    :param contour: the contour we want to get its moment
    :return: the moment of the contour
    """
    m = cv2.moments(contour)
    cx = int(m['m10'] / m['m00'])
    cy = int(m['m01'] / m['m00'])
    return cx, cy


def follow_lane(img: np.ndarray):
    """Return the steering angle for following lane.
    :param img: the Lane image
    :return: the steering angle
    """
    # Crop the lane image.
    img = img[128:192, :]
    # Get the contours.
    contours = get_contours(img)
    if len(contours) == 0:
        return None

    # Find the biggest contour.
    c = max(contours, key=cv2.contourArea)
    # Get the moment of the biggest contour.
    cx, cy = get_moment(c)

    # Point out the desired moment and contour on the image.
    cv2.line(img, (cx, 0), (cx, 720), (255, 0, 0), 1)
    cv2.line(img, (0, cy), (1280, cy), (255, 0, 0), 1)
    cv2.drawContours(img, c, -1, (0, 255, 0), 1)

    # Calculate the angle we want to steer.
    angle = math.atan((128 - cx) / cy)

    return angle, img


def get_other_lane(img: np.ndarray):
    """Get which side the other lane is on.

    :param img: the image retrieved from the vision sensor.
    :return: which side the other lane is on, 1 means left, -
1 means right
    """
    # Crop the image.
    img = img[:90, :]

    # Find the contours.
    contours = get_contours(img)
    if len(contours) < 2:
        return None

    # Sort the contours by ContourArea.
    sorted(contours, key=cv2.contourArea)
    c = contours[-1]
```

```python
    # The biggest contour which doesn't have the center point is what we want.
    for i in reversed(range(len(contours))):
        if cv2.pointPolygonTest(contours[i], (128, 0), False) == -1:
            c = contours[i]

    # Get the moment of the desired contour.
    cx, cy = get_moment(c)

    # Point out the contour and the moment on the image.
    cv2.line(img, (cx, 0), (cx, 720), (255, 0, 0), 1)
    cv2.line(img, (0, cy), (1280, cy), (255, 0, 0), 1)
    cv2.drawContours(img, c, -1, (0, 255, 0), 1)

    # Determine which side the other lane is on.
    return 1 if cx < 132 else -1, img
```