机器人导论

课程作业: assignment 2

组员：16305204 郑佳豪

16326086 王润锋

提交日期：2019-09-23

Deadline：2019-09-23

1. 任务概要：

- 参考文档中 Tutorial – Line Following BubbleRob 部分，学习：
  - 巡线小车任务描述
  - 自定义线路径的添加与编辑
  - 光感巡线小车的简单逻辑与巡线效果
- 参考文档 Vision Sensors 章节，学习：
  - 视觉传感器的概念
  - 视觉传感器的参数设置
  - 视觉传感器相关 API
- 学习 PID 算法原理
  - https://en.wikibooks.org/wiki/Control_Systems/Controllers_and_Compensators
  - P、PI、PID 控制等方案的异同
- 学习数字图像处理相关知识
- 在任务 1 的基础上，调整单目摄像头的角度，使之能够看到地面上的路线
- 结合小车底盘模型设计合理的 PID 算法，编写脚本实现基于单目图像的巡线功能
  - 思路：
    - 调节量是什么？轮速？舵机转角？是调节增量还是位置？
    - 偏差量是什么？左半边跟右半边黑色像素的数量？线中心的位置？斜率？

姓名	学号	比例	具体任务
郑佳豪	**16305204**	**60%**	**搭建机器人模型、完成 PID 控制模型、完成实验报告**
王润锋	**16326086**	**40%**	**完成 PID 控制模型、完成实验报告、录制实验视频**

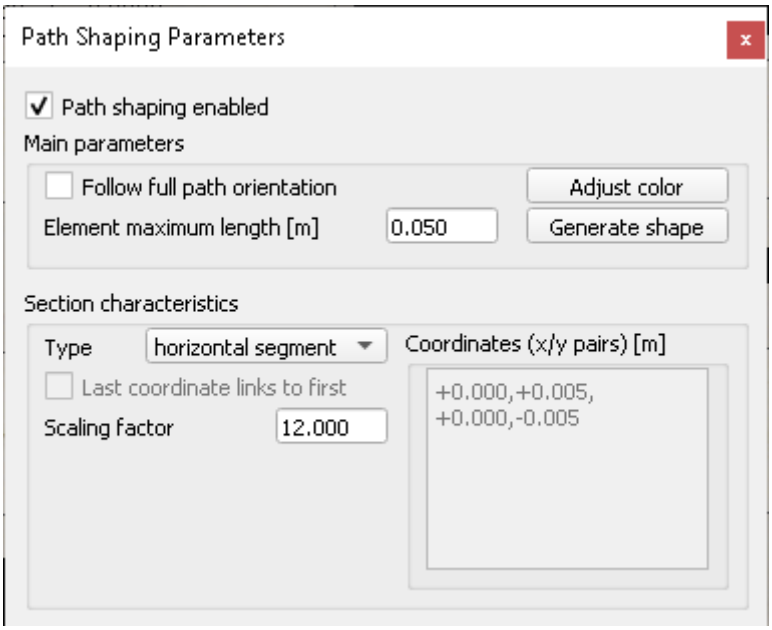
2. 完成情况：

总体完成情况如下：

- 已熟悉巡线小车的任务描述，并熟悉 V-REP 软件的基本使用，如车身与轮子的物理引擎的设计、动力学模型的参数设置、传感器的添加与使用、驱动关节的设计与控制、控制脚本的编写、自定义线路径的添加与编辑等功能，已了解视觉传感器的概念、参数设置和相关 API。
- 已学习 PID 算法原理，大致理解 P、PI、PID 控制等方案的异同。
- 已在任务 1 的基础上，调整单目摄像头的角度，使其能看到地面上的路线，并实现了基于 PID 控制理论的巡线功能。

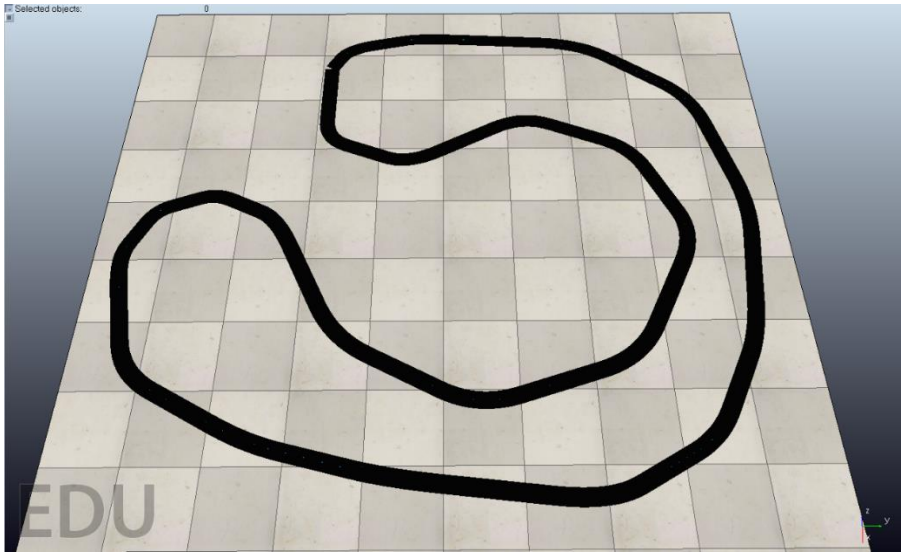
1. 自定义路线

巡线小车必不可少的就是路径，在 Tutorial – Line Following BubbleRob 有比较详细的介绍。我们在[Menu bar-->Add-->Path-->Circle type]添加 path，打开 Object Movement with the Mouse 开关，这样我们就可以用鼠标控制 path，然后在左边工具栏打开 Path Edit Mode，这样我们就可以比较方便调整路径的形状。



1 路径宽度设置

最终，我们设计并添加了如下图所示的路径。

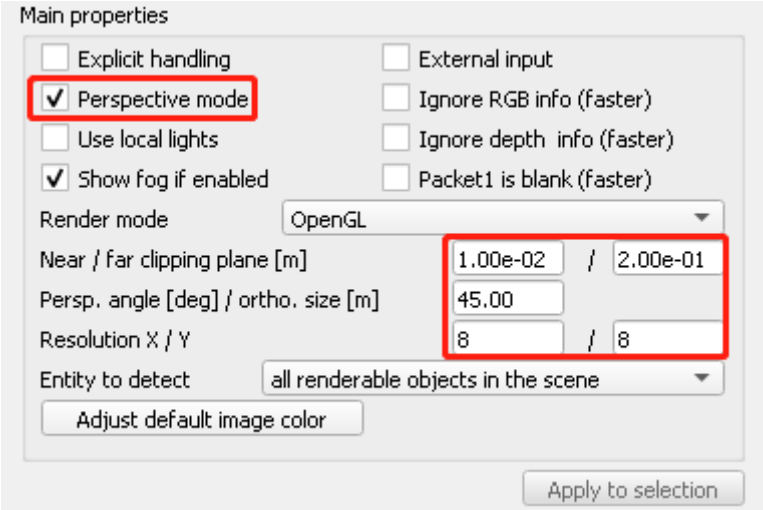


2 路径鸟瞰图

2. 传感器

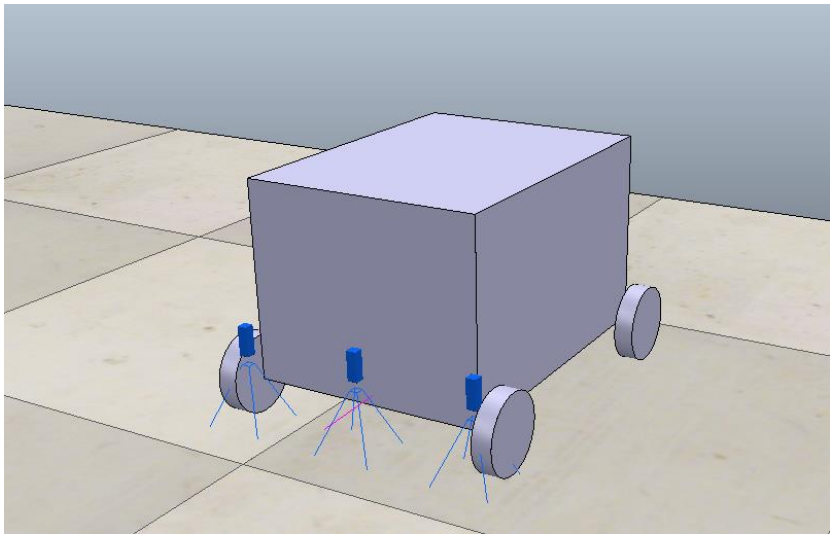
传感器是巡线小车的核心组件,为实现巡线，我们为其添加了 3 个 Vision Sensor，分别是 MyBot_Left_Sensor、MyBot_Middle_Sensor、MyBot_Right_Sensor。这三个传感器可以在[Menu bar --> Add --> Vision sensor --> Orthographic type]添加，并设置角度为[180;0;0]，并这三个传感器拖进我们的小车中。值得注意的是，我们需要将这三个传感器设置为 Perspective mode，并设置分辨率和角度。我们将分辨率设置为 8*8，将拍摄角度设置为 45

度。



3　视觉传感器的设置

以下是我们在任务 1 的机器人基础上，添加完 3 个视觉传感器的静态效果图。



4　添加视觉传感器的机器人静态演示

3. 工具函数

我们实现了 motor 工具函数，该函数用于两个前进电机的动力设置，其相关代码如下。

```
-- Get FL Motor.
FLwheel_Motor = sim.getObjectHandle('MyBot_FLwheel_Motor')
-- Get FR Motor.
FRwheel_Motor = sim.getObjectHandle('MyBot_FRwheel_Motor')
-- Motor Tool Function
motor = function(speed)
    sim.setJointTargetVelocity(FLwheel_Motor, speed)
    sim.setJointTargetVelocity(FRwheel_Motor, speed)
end
```

5　motor 工具函数

我们使用的是 Ackermann 转向运动学模型。根据该运动学模型，车辆在转弯时，其两个转向轮的转向角度是不一致的，但其角度可由数理公式推导而出。为此，我们实现了 steer 转向工具函数，实现两个转向轮能够按照模型进行不同角度的转向。

```
-- d = 0.2 * distance (cm) between left and right wheels.
-- l = 0.2 * distance (cm) between front and rear wheels.
d = 28 * 0.2
l = 30 * 0.2
-- Get FL Steering Motor.
FLwheel_Steering = sim.getObjectHandle('MyBot_FLwheel_Steering')
-- Get FR Steering Motor.
FRwheel_Steering = sim.getObjectHandle('MyBot_FRwheel_Steering')
-- Steer Tool Function
steer = function(angle)
    -- Conversion
    angle = angle * math.pi / 180
    leftAngle = math.atan(l / (-d + l / math.tan(angle)))
    rightAngle = math.atan(l / (d + l / math.tan(angle)))
    sim.setJointTargetPosition(FLwheel_Steering, leftAngle)
    sim.setJointTargetPosition(FRwheel_Steering, rightAngle)
end
```

6　steer 工具函数

4. 基于非 PID 的巡线功能实现

我们先实现非 PID 控制的巡线功能，我们的思路很简单：通过在主循环里，我们通过 readVisionSensor 读取视觉传感器的数据，根据左右传感器的数据情况，进行转向:若只有左边的传感器检测到线，则向左转向；若只有右边的传感器检测到线，则向右转向;若左右传感器都检测到线，则直行。

在读取视觉传感器数据之前，我们需要获取传感器实例，具体代码如下。注意到，我们用数组维护传感器实例，这是为了后面，我们能够方便地读取数据。

```
-- Get Vision Sensors
Vision_Sensors = {-1, -1, -1}
-- Get Left Vision Sensor.
Vision_Sensors[1] = sim.getObjectHandle('MyBot_Left_Sensor')
-- Get Middle Vision Sensor.
Vision_Sensors[2] = sim.getObjectHandle('MyBot_Middle_Sensor')
-- Get Right Vision Sensor.
Vision_Sensors[3] = sim.getObjectHandle('MyBot_Right_Sensor')
```

7　获取视觉传感器实例

我们在 Threaded 脚本的主循环中，通过调用 sim.readVisionSensor 获取各传感器的数据，并跟阈值（0.6）进行比较，若小于阈值，则表明已探测到线，否则表明未探测到线。

```
-- The Main Loop
while sim.getSimulationState() ~= sim.simulation_advancing_abouttostop do
    -- Read sensors' data.
    results = {-1, -1, -1}
    for i = 1, 3, 1 do
        result, data = sim.readVisionSensor(Vision_Sensors[i])
        if (result >= 0) then
            -- When data is lower than 0.6,
            -- it means the presence of the line.
            results[i] = (data[11] < 0.6)
        end
    end
    -- ......
end
```

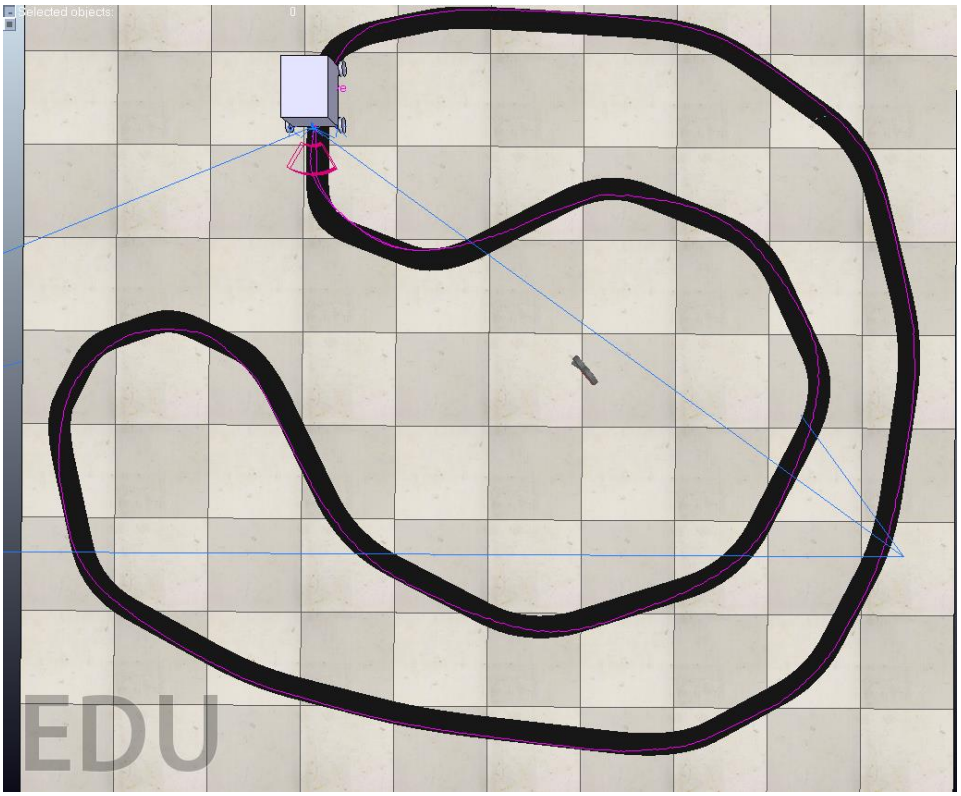
8　读取视觉传感器数据

在获取到数据后，我们即可用其来控制机器人转向的角度：当只有左传感器探测到线时，设置转向角度为左转 30 度；当只有右传感器探测到线时，设置转向角度为右转 30 度；其余情况，设置转向角度为 0 度，即进行直行运动。

```
steeringAngle = 0
-- If the line is on the left side, turn left.
if results[1] then
    steeringAngle = 30
end
-- If the line is on the right side, turn right.
if results[3] then
    steeringAngle = -30
end
-- Go straight.
if results[1] and results[3] then
    steeringAngle = 0
end
-- Steer for calculated angle.
steer(steeringAngle)
-- Motor Speed.
motor(5)
```

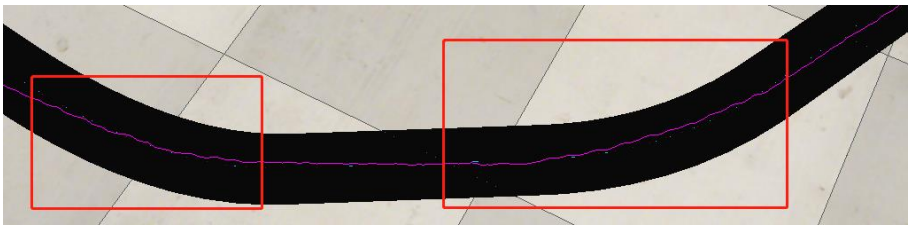
9　根据读取的数据控制转向角度

点击仿真按钮运行后，我们可以看到小车成功地完成了巡线任务，其运行轨迹鸟瞰图如下。



10　非 PID 控制的运行轨迹

尽管，我们成功实现了在不使用 PID 控制的条件下，机器人完成巡线任务的任务，但仍是存在问题。我们仔细查看运行轨迹，即可发现存在不少的抖动，这是需要改进的地方。下图是一个典型的轨迹抖动的示例。



11　非 PID 控制的轨迹抖动问题

5.　P、PI、PID

在实现基于 PID 的巡线功能之前，我们先分析 P、PI、PID 三者的差异。我们先分析一下 PID 三项的关系：

- 比例控制 P 是主要的控制方法，承担了 PID 控制中的大部分任务。
- 为了消除比例控制 P 留下的静态偏差，增加了积分控制 I。
- 微分控制 D 只为稳定而存在，其稳定效果应该大于积分控制 I 的失稳效果，在有大量噪音的系统中，不使用微分控制 D。

P 控制器，是最基本的控制形式，如果输出的结果偏移量或偏差为常量且可接受，则可使用此方法，缺点是存在静态偏差。

PI 控制器，积分控制 I 能消除比例控制 P 留下的静态偏差，能够满足不少控制要求，若不存在稳定性问题，则可使用此方法，其优点是良好的阻尼、静态偏移为零，缺点是反应缓慢。

PID 控制器，是一个完整的三项 PID 控制，在使用积分模式造成不稳定时使用，能够减

少上升空间、消除静态误差、减少超调。

6. 基于 PID 的巡线功能实现

我们将左右传感器的像素差异作为**偏差量**，将转向的角度作为**调节量**，来实现基于 PID 的巡线功能。

我们需要对前面的传感器数据的处理方式进行修改，具体变动如下。

```
-- Read sensors' data.
results = {-1, -1, -1}
for i = 1, 3, 1 do
    result, data = sim.readVisionSensor(Vision_Sensors[i])
    if (result >= 0) then
        results[i] = data[11]
    end
end
-- ......
```

12　PID 传感器数据的处理方式

由于我们直接使用左右传感器的像素差异作为偏差量，所以我们需要将 results 设置为传感器原始数据。

首先，我们实现 P 控制器，需要完成比例控制单元。比例单元，就是我们将偏差量乘以比例系数 kp，得到调节量。经过多次参数的调整，我们将 kp 设定为 50，在该设定下，机器人运动轨迹比非 PID 控制的轨迹平稳许多。

```
-- P Control
kp = 50
error = (results[3] - results[1])
-- P Control
steeringAngle = kp * error
```

13　P 控制器

随后，我们实现 PI 控制器，需要完成积分控制单元的设计。在积分单元中，我们对偏差量进行累加，得到积分量，将其乘以比例系数 ki，最后与 P 控制器一同作用于调节量。

```
-- P Control
kp = 50
error = (results[3] - results[1])
-- I Control
ki = 0.01
integral = integral + error
-- PI Control
steeringAngle = kp * error + ki * integral
```

14　PI 控制器

在实现完 PI 控制器后，我们需要完成微分单元的设计。我们现在有了比例控制单元，用于纠正当前的误差，有了积分控制单元，用于纠正过去的误差，还需要微分单元，用来及时预测未来，对还未发生的误差进行纠正。

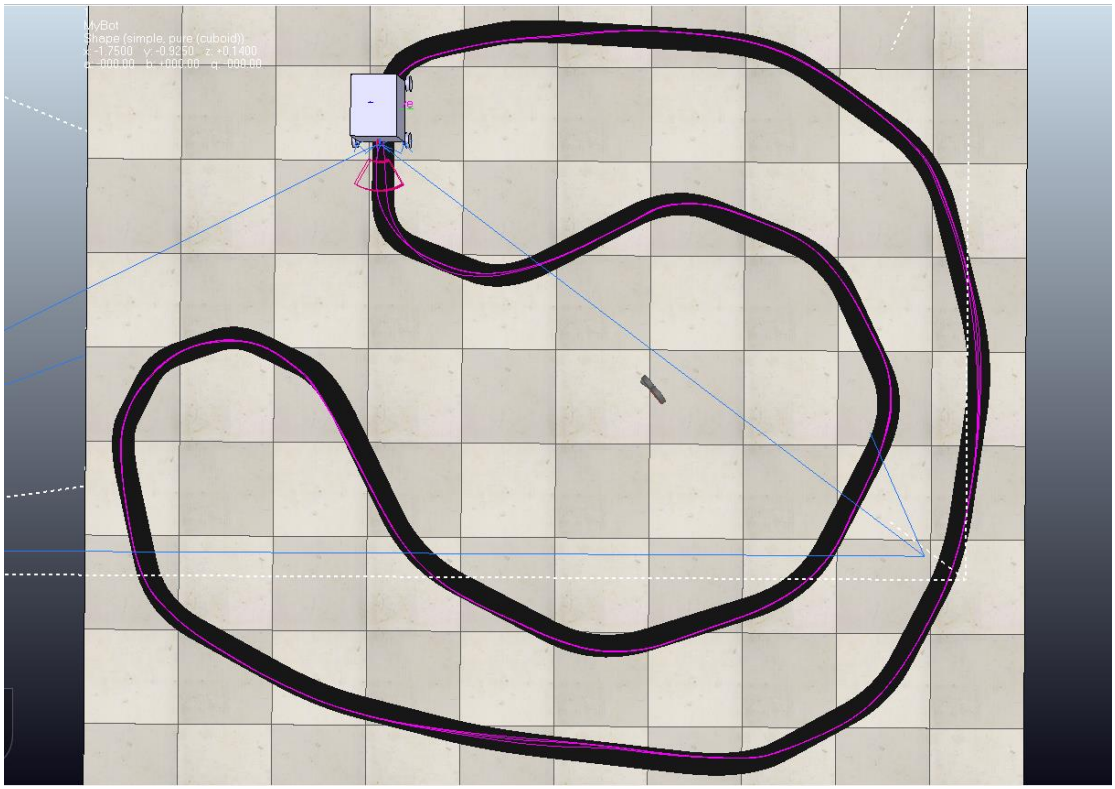
```
-- P Control
kp = 50
error = (results[3] - results[1])
-- I Control
ki = 0.01
integral = integral + error
-- D Control
kd = 1
derivative = error - prev_error
prev_error = error
-- PID Control
steeringAngle = kp * error + ki * integral + kd * derivative
```

15　PID 控制器

如上图所示，我们得到了 PID 控制器的表达式，虽然形式简单，但其各个部分的比例系数的确定是比较麻烦的，需要反复调整与测试，最终经过我们多次调整，发现 kp=50，ki=0.01，kd=1 是一组合适的参数。

3. 效果展示：

下面是机器人巡线的示意图，详细细节可参考视频《机器人导论 HW2 VREP 下实现视觉巡线小车》，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/av68748101



16　机器人巡线示意

我们可以看到，基于 PID 控制的巡线功能，相比于非 PID 的巡线，其轨迹更为平滑。

4. 存在问题：

本次实验，进行得较为顺利，途中虽遇到了很多问题，但最终还是成功解决了。在读取视觉传感器数据时，由于一开始我们并没有对其结果是否存在进行判定，导致运行时出现了 attempt to index global 'data' (a nil value) 的错误，在仔细研究 Tutorial 代码后，我们发现了此问题的根源，并将其解决。

PID 控制部分并未在 Tutorial 中提及，所以一开始时，我们并不知道如何实现，不知道如何下手。反复查看 PPT 和相关资料后，我们搞清楚了“谁是偏差量”和“谁是调节量”这两个核心的问题，后面的 PID 控制代码的编写是水到渠成的。在实验过程中，我们的参数调整，也算是有些坎坷，调整了很多次，但最终也找到了合适的参数组合。

5. 附录：

MyBot PID 巡线控制脚本（Threaded）

```
function sysCall_threadmain()
    -- Initialization

    -- Get FL Steering Motor.
    FLwheel_Steering = sim.getObjectHandle('MyBot_FLwheel_Steering')
    -- Get FR Steering Motor.
    FRwheel_Steering = sim.getObjectHandle('MyBot_FRwheel_Steering')
    -- Get FL Motor.
    FLwheel_Motor = sim.getObjectHandle('MyBot_FLwheel_Motor')
    -- Get FR Motor.
    FRwheel_Motor = sim.getObjectHandle('MyBot_FRwheel_Motor')
    -- Get Vision Sensors
    Vision_Sensors = {-1, -1, -1}
    -- Get Left Vision Sensor.
    Vision_Sensors[1] = sim.getObjectHandle('MyBot_Left_Sensor')
    -- Get Middle Vision Sensor.
    Vision_Sensors[2] = sim.getObjectHandle('MyBot_Middle_Sensor')
    -- Get Right Vision Sensor.
    Vision_Sensors[3] = sim.getObjectHandle('MyBot_Right_Sensor')
    -- d = 0.2 * distance (cm) between left and right wheels.
    -- l = 0.2 * distance (cm) between front and rear wheels.
    d = 28 * 0.2
    l = 30 * 0.2
    -- Angle to steer.
    -- +: left
    -- -: right
    steeringAngle = 0
    -- It's used to control the total time of avoiding.
    avoidUntilTime = 0
```

```
    -- PID Control
    integral = 0
    derivative = 0
    prev_error = 0

    -- Steer Tool Function
    steer = function(angle)
        -- Conversion
        angle = angle * math.pi / 180
        leftAngle = math.atan(l / (-d + l / math.tan(angle)))
        rightAngle = math.atan(l / (d + l / math.tan(angle)))
        sim.setJointTargetPosition(FLwheel_Steering, leftAngle)
        sim.setJointTargetPosition(FRwheel_Steering, rightAngle)
    end

    -- Motor Tool Function
    motor = function(speed)
        sim.setJointTargetVelocity(FLwheel_Motor, speed)
        sim.setJointTargetVelocity(FRwheel_Motor, speed)
    end

    -- The Main Loop
    while sim.getSimulationState() ~= sim.simulation_advancing_abouttos
top do
        results = {-1, -1, -1}
        for i = 1, 3, 1 do
            result, data = sim.readVisionSensor(Vision_Sensors[i])
            if (result >= 0) then
                results[i] = data[11]
            end
        end

        -- P Control
        kp = 50
        error = (results[3] - results[1])
        -- I Control
        ki = 0.01
        integral = integral + error
        -- D Control
        kd = 1
        derivative = error - prev_error
        prev_error = error
        -- PID Control
        steeringAngle = kp * error + ki * integral + kd * derivative
```

```
        -- Steer for calculated angle.
        steer(steeringAngle)
        -- Motor Speed.
        motor(15)

        -- -- Since this script is threaded, don't waste time here.
        sim.switchThread()
    end
end
```

AutoFittingCamera 控制脚本

```
function sysCall_init()
    cam = sim.getObjectHandle('AutoFittingCamera')
    originalParent = sim.getObjectParent(cam)
    sim.setObjectParent(cam, -1, true)
    floatingView = sim.floatingViewAdd(0.2, 0.8, 0.4, 0.4, 0)
    sim.adjustView(floatingView, cam, 64)
    dampingFactor = sim.getScriptSimulationParameter(sim.handle_self, '
dampingFactor')
    if (not damping) then
        damping = 5
    end
    if (damping < 1) then
        damping = 1
    end
    if (damping > 20) then
        damping = 20
    end
    -- Get the object/model we want to fit into the view:
    options = 0
    local objectOrModelHandle = originalParent
    if (objectOrModelHandle ~= -1) then
        while (sim.getObjectParent(objectOrModelHandle) ~= -1) do
            objectOrModelHandle = sim.getObjectParent(objectOrModelHand
le)
        end
        if (sim.getModelProperty(objectOrModelHandle) == sim.modelprope
rty_not_model) then
            objectOrModelHandle = originalParent
            options = 1
        end
        objectsToFrame = {objectOrModelHandle}
    end
```

```
end

function sysCall_cleanup()
    if sim.isHandleValid(cam) == 1 then
        sim.setObjectParent(cam, originalParent, true)
    end
    sim.floatingViewRemove(floatingView)
end

function sysCall_sensing()
    sim.cameraFitToView(floatingView, objectsToFrame, options, 0.8)
    local p = sim.getObjectPosition(cam, -1)
    if (not previousPositions) then
        previousPositions = {}
        for i = 1, 5, 1 do
            table.insert(previousPositions, p)
        end
    end
    table.remove(previousPositions, #previousPositions)
    table.insert(previousPositions, 1, p)

    local cumul = {0, 0, 0}
    for i = 1, #previousPositions, 1 do
        cumul[1] = cumul[1] + previousPositions[i][1]
        cumul[2] = cumul[2] + previousPositions[i][2]
        cumul[3] = cumul[3] + previousPositions[i][3]
    end
    cumul[1] = cumul[1] / #previousPositions
    cumul[2] = cumul[2] / #previousPositions
    cumul[3] = cumul[3] / #previousPositions

    sim.setObjectPosition(cam, -1, cumul)
end
```