中山大學

# 机器人导论

课程作业: assignment 3


组员：16305204 郑佳豪

17343144 余小煜

16326086 王润锋


提交日期：2019-10-10

Deadline：2019-10-10

# 1. 任务概要：

- 在 Homework2 的基础上，优化原有设计，并设计具有挑战性的路径。设计路径可以多样，可以是起始点在同一位置（环线），也可以不是，可含有以下各种难点：
    - 道路断续
    - 道路交叉
    - 急转弯
- 巡线小车控制算法包括但不限于 PID，但必须采用视觉巡线（不能贴地模拟光感巡线）

| 姓名 | 学号 | 比例 | 具体任务 |
| --- | --- | --- | --- |
| **郑佳豪** | **16305204** | **40%** | **视觉巡线算法的设计与实现、实验报告的编写、演示视频的录制** |
| **余小煜** | **17343144** | **40%** | **比赛路径的设计、实验报告的编写** |
| **王润锋** | **16326086** | **20%** | **视觉巡线算法的实现、实验报告的编写** |

# 2. 完成情况：

总体完成情况如下：

- 已实现机器人视觉巡线功能，废弃原有的贴地传感方案。
- 已完成比赛路径的设计。

**1. 视觉巡线**

由于我们在 HW2 中使用的是视觉传感器贴地模拟光感的实现方案，因此在本次作业中，我们需要使用“真正”的视觉巡线方案。在这个过程中，我们走了不少弯路，也因此耽误了比赛的参与。

- **简单版本：基于单行图片的黑色线中心判断**

一开始，我们使用的策略是这样的：针对图片中的某一行，提取**黑色线的中心位置**，并计算它与**图片竖向中心线**的距离，在此距离（偏差量）应用 PID 控制理论，得到调节量——Akermann 运动学模型的前轮转向角。

由于便携控制机器人的运动学状态，我们编写了 motor 函数，用于控制机器人的前进动力。

```lua
-- 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
```

*1 motor 工具函数*

除此之外，我们还编写了 steer 工具函数，其使用了 Ackermann 转向运动学模型的相关推导，用于实现两个转向轮能按照给定角度进行不同角度的转向。

```lua
-- 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
```

*2 steer 工具函数*

检测黑色线中心的核心算法是针对特定行，例如在下图中，我们使用第 0 行，从左至右扫描出黑色线的起始点和终止点，从而得到它的中心点位置。随后，由于我们的图像是 64*64 像素的，因此图像的竖向中心线的位置是 32。通过简单的反三角函数的使用，我们即刻求取黑色线中心点与第 0 行第 32 个元素（即我们选取的参考点的）的夹角。

```lua
calc_vision_angle = function()
    local y = 0

    image = sim.getVisionSensorImage(Vision_Sensor, 0, y, 64, 1, 0)
    sim.setVisionSensorImage(Display_Vision_Sensor, image)

    local first = true
    local endpoint_left = 0
    local endpoint_right = 192
    for i = 1, 192, 1 do
        if image[i] == 0 then
            if first then
                endpoint_left = i
                first = false
            end
            endpoint_right = i
        end
    end

    local h = 64 - y
    local angle = math.atan2(32 - (endpoint_left + endpoint_right) / 6, h) * 180 / math.pi
    return angle
end
```

*3　计算黑色线中心的偏离角度*

在获取到黑色线中心的偏离角度后，我们即可对此偏差量应用 PID 控制原理。经过多次调整，我们已得到了 PID 的三个较优的参数。

```lua
-- P Control
kp = 2
error = calc_vision_angle()
-- I Control
ki = 0
integral = integral + error
-- D Control
kd = 0
derivative = error - prev_error
prev_error = error
-- PID Control
steeringAngle = kp * error + ki * integral + kd * derivative

if (math.abs(steeringAngle)) > 20 then
    motor(10)
else
    motor(20)
end

-- Steer for calculated angle.
steer(steeringAngle)
```

*4　简单视觉巡线 PID 控制脚本*

- 最终版本

尽管在演示中，简单版本的机器车车速度很快，且巡线效果很好，但在复杂的场景中，表现却很糟糕（例如在急转弯路径中），因此我们想提高机器小车面对复杂场景的鲁棒性。

简单版本的局限性在于，它只观测了某一方的黑色像素分布情况，没有考虑全局，因此我们打算通过分析全局的黑色像素分布，尽可能提高机器人对复杂场景的鲁棒性。

我们通过调用 getVisionSensorImage 方法，并为其传入灰度处理的 Handle Flag，获取视觉传感器的结果，这是一个 64*64 的 Table。随后，我们对其左右两部分的像素点进行统计，从而获取左右两部分的像素差，作为偏差值。最终，我们对此偏差值进行 PID 控制。获

取偏差值的具体代码如下所示。

```
data = sim.getVisionSensorImage(Vision_Sensor + sim.handleflag_greyscale, 0, 0, 64, 64)
greyL = 0
greyR = 0

for i = 1, 32, 1 do
    for j = 30, 50, 1 do
        greyL = greyL + data[(i - 1) * 64 + j]
    end
end

for i = 33, 64, 1 do
    for j = 30, 50, 1 do
        greyR = greyR + data[(i - 1) * 64 + j]
    end
end
```

*5　获取图像左右部分的像素差异*

在获取到偏差值后，我们应用 PID 控制原理，输出转向角度，具体流程如以下代码所示。

```
-- P Control
kp = 2.3
error = (greyR - greyL) / 10
-- I Control
ki = 0
integral = integral + error
-- D Control
kd = 0
derivative = error - prev_error
prev_error = error
-- PID Control
steeringAngle = kp * error + ki * integral + kd * derivative

if (math.abs(steeringAngle)) > 20 then
    motor(1)
else
    motor(2)
end

-- Steer for calculated angle.
steer(steeringAngle)
```

*6　对图像左右部分的偏差值进行 PID 控制*

由于在速度较快情况下，我们并未找到此模型的较优 PID 参数，因此我们的机器人只能在较为低速的情况下，进行巡线，且对比于简单版本的机器人模型，巡线质量并没有足够的优势，估计还是因为 PID 参数没有调整到位。

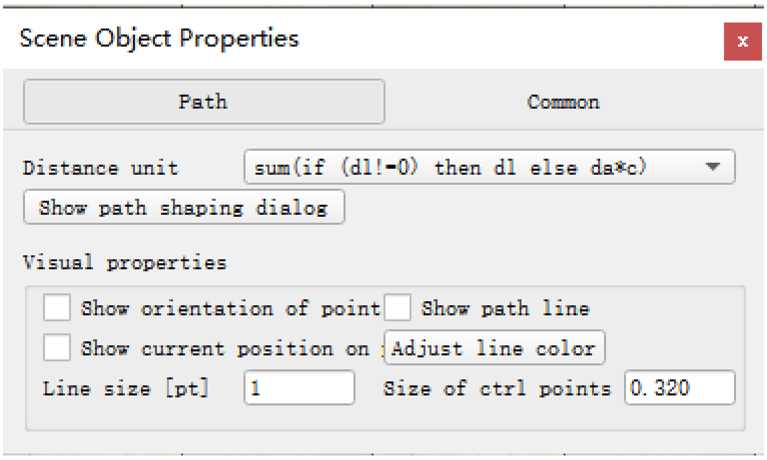
## 2. 路径设计

为了测试小车的巡线功能，让小车在敏捷性和稳定性上到达最佳平衡，我们针对性地设计不同类型的路径。

- **交叉的路径**

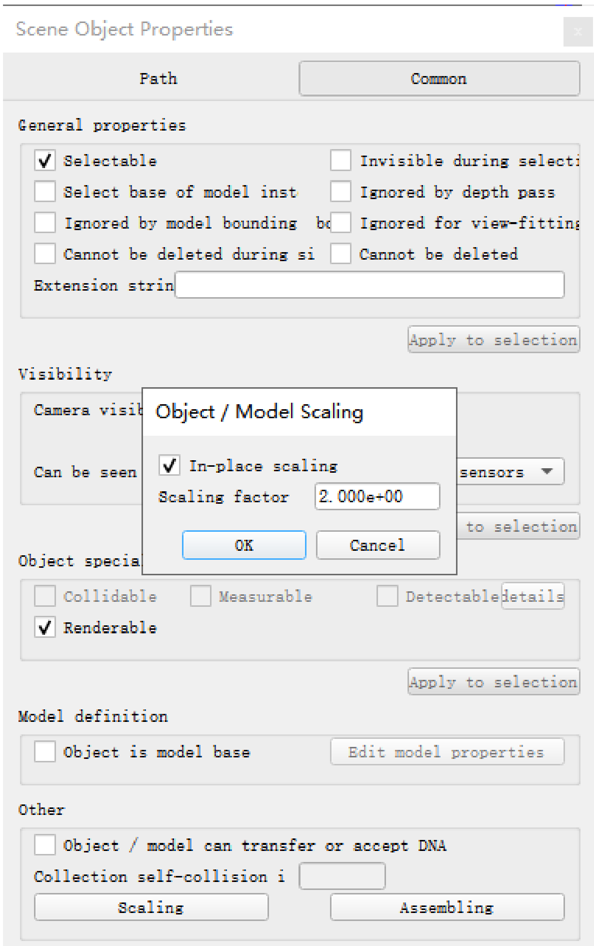
点击 [Menu bar --> Add --> Path --> Circle type] 添加路径，Ctrl 点击路径上的点来修改路径形状达到理想状态。

接着修改路径属性，首先取消勾选 Show orientation，Show path line 和 Show current position：



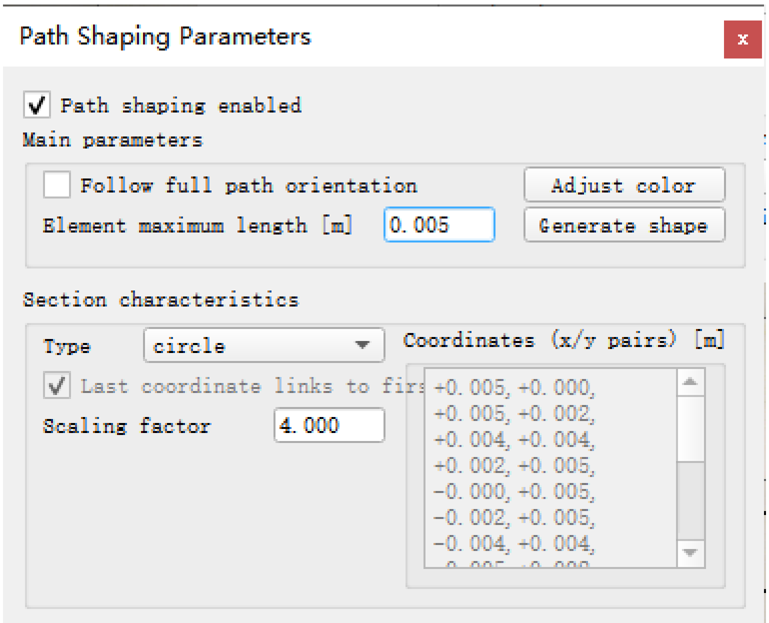

7 取消显示路径的相关参数

然后点击 [Common --> Scaling]，勾选 In-place scaling，再修改 Scaling factor 直到路径规模合适：



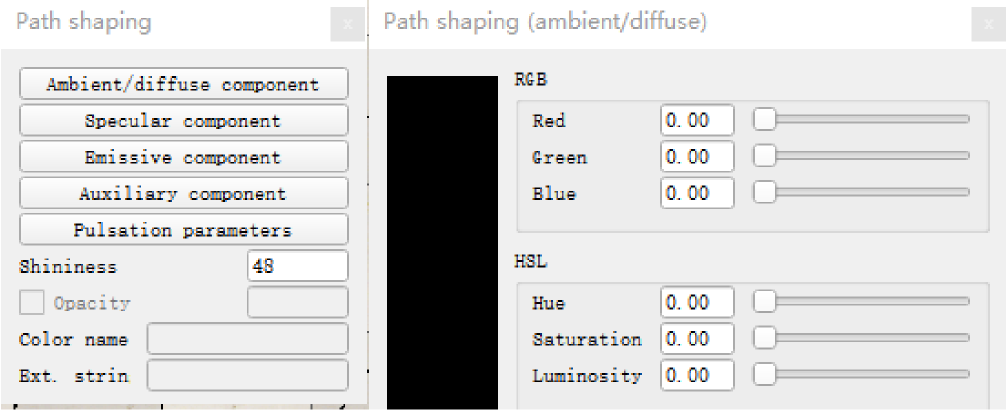

8 路径 In-place scaling 设置

打开 Path Shaping Parameters 修改属性，将 Scaling Factor 修改为 4.00：



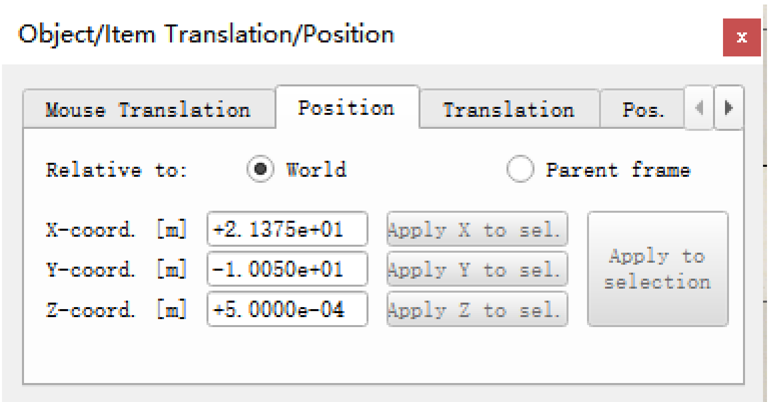

9　路径 Scaling factor 设置

点击 Adjust Color 修改路径颜色为黑色：



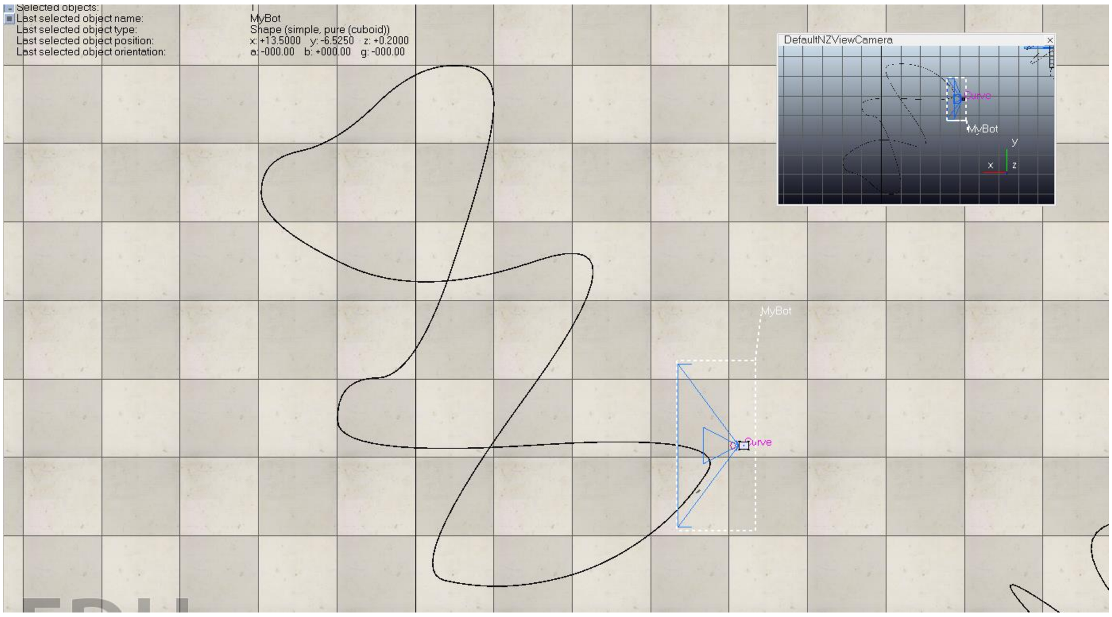

10　路径颜色设置

最后，我们将路径的 Z 坐标增加 0.5mm，如下：



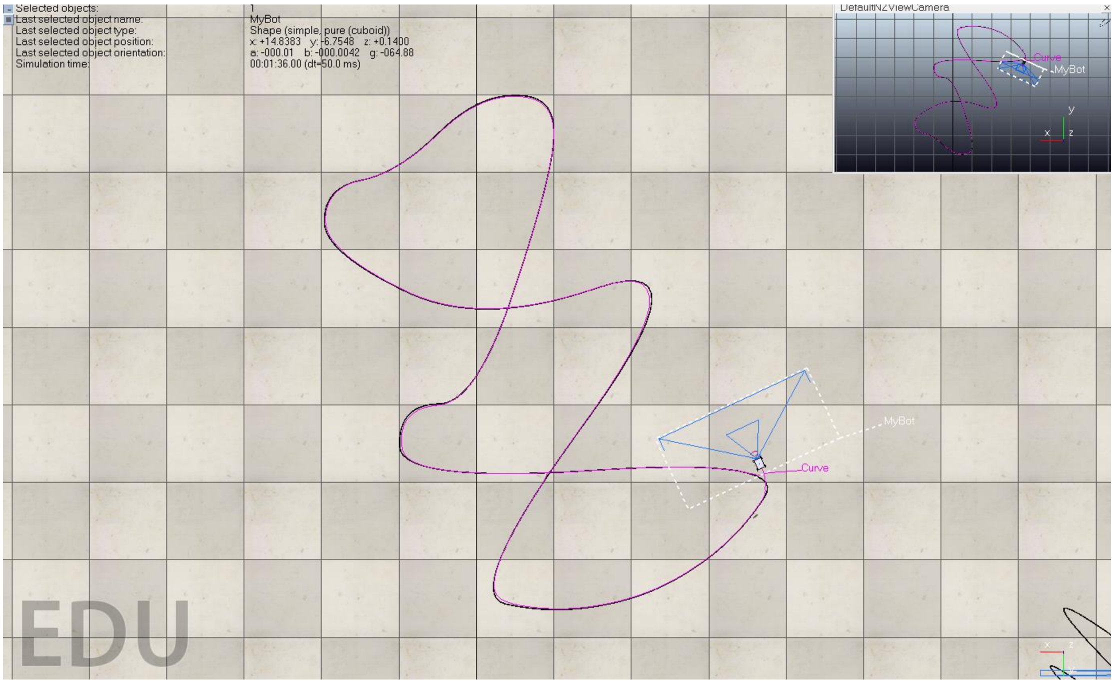

11　路径 z 坐标设置

至此，交叉路径设计完成，其最终效果如下图所示。




12　交叉路径鸟瞰图

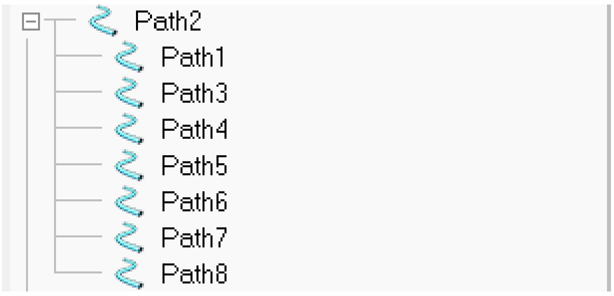
我们在此路径上，测试机器小车的巡线功能，小车性能表现良好。




13　机器人在交叉路径的巡线效果

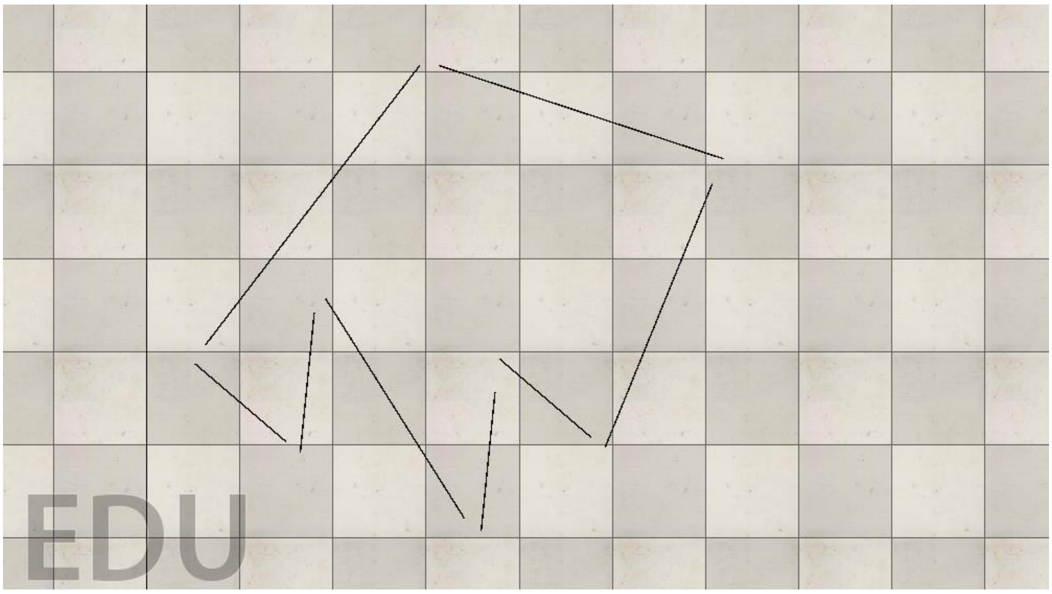
- **断续的路径**

断续路径的设计过程与之前类似，只是此处应用 Segment type 的 path，将几段路径以如下的层次结合在一起：




*14　断续路径的层次结构*

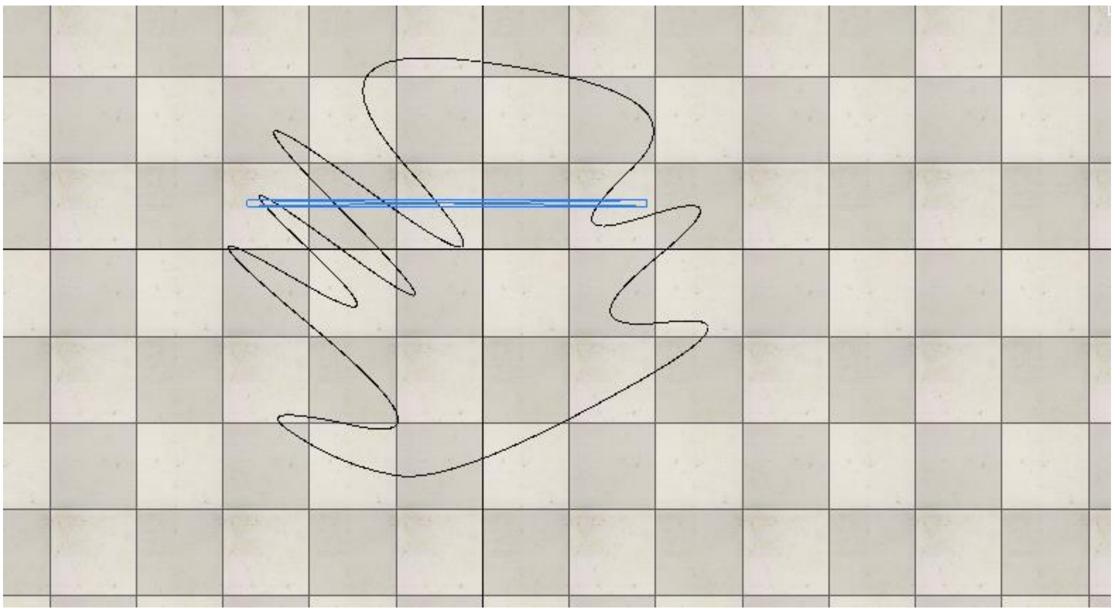
断续路径的最终效果如下图所示。




*15　断续路径鸟瞰图*

- **连续急转弯**

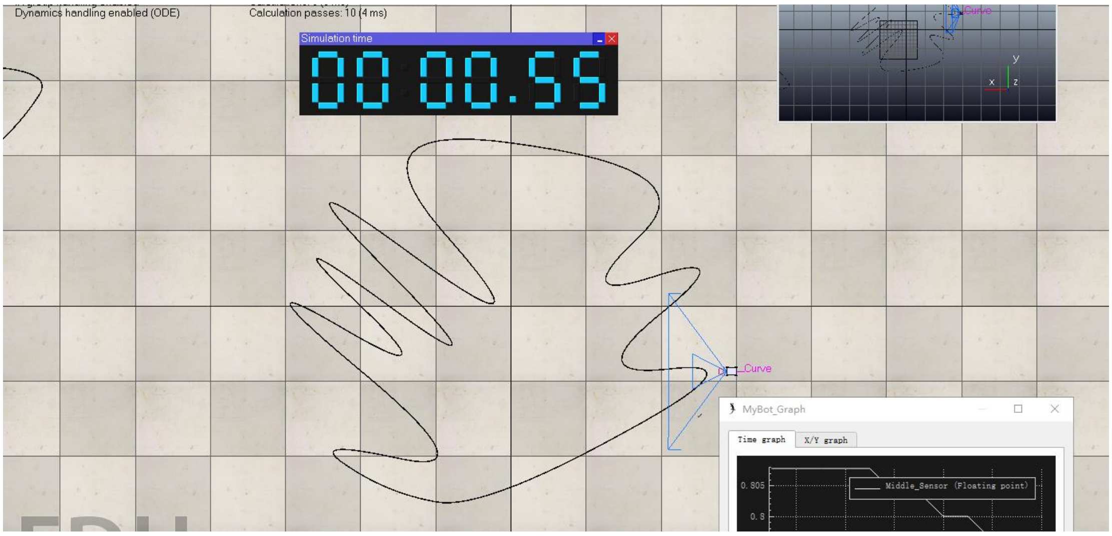
连续急转弯的路径设计过程，与类似前面两种路径类似，其最终效果如下图所示。



*16　连续急转弯路径鸟瞰图*

### 3. 计时器

我们需要在 Scene 添加 Simulation Time Display 来记录仿真时间。




17 仿真时间示意图

我们添加如下初始化语句来获取小车从起点出发的位置和仿真时间：

```
-- Get MyBot
MyBot = sim.getObjectHandle('MyBot')

-- Start
startPos = sim.getObjectPosition(MyBot, -1)
startTime = sim.getSimulationTime()
```

18 巡线小车初始化仿真计时器

其中 sim.getSimulationTime()原型如下：

**simGetSimulationTime / sim.getSimulationTime**

| | |
|---|---|
| Description | Retrieves the current simulation time |
| C synopsis | simFloat simGetSimulationTime() |
| C parameters | None |
| C return value | negative value (-1.0) if operation not successful, otherwise the simulation time |
| Lua synopsis | number simulationTime=getSimulationTime() |
| Lua parameters | Same as C-function |
| Lua return values | Same as C-function |
| Remote API equiv. | B0-based remote API: simxGetSimulationTime<br>Legacy remote API: - |

19 getSimulationTime 说明

Sim.getObjectPosition()原型如下，我们传入-1 作为 relativeToObjectHandle 参数来获取小车的绝对位置：

**simGetObjectPosition / sim.getObjectPosition**

| | |
|---|---|
| Description | Retrieves the position of an object. See also sim.setObjectPosition, sim.getObjectOrientation, sim.getObjectMatrix and the other matrix/transformation functions. |
| C synopsis | simInt simGetObjectPosition(simInt objectHandle,simInt relativeToObjectHandle,simFloat* position) |
| C parameters | **objectHandle**: handle of the object<br>**relativeToObjectHandle**: indicates relative to which reference frame we want the position. Specify -1 to retrieve the absolute position, sim_handle_parent to retrieve the position relative to the object's parent, or an object handle relative to whose reference frame we want the position.<br>**position**: pointer to 3 values (x, y and z) |
| C return value | -1 if operation was not successful. In a future release, a more differentiated return value might be available |
| Lua synopsis | table_3 position=sim.getObjectPosition(number objectHandle,number relativeToObjectHandle) |
| Lua parameters | Same as C-function |
| Lua return values | **position**: table of 3 values (x, y and z) or nil in case of an error |
| Remote API equiv. | B0-based remote API: simxGetObjectPosition, simxGetObjectPose<br>Legacy remote API: simxGetObjectPosition |

*20 getObjectPosition 说明*

在小车的驱动调整模块添加以下语句。当小车再次到达起点时，会输出总用时。

```
-- End
position = sim.getObjectPosition(MyBot, -1)
if position == startPos then
    endTime = sim.getSimulationTime()
    print('time: ', endTime - startTime)
end
```

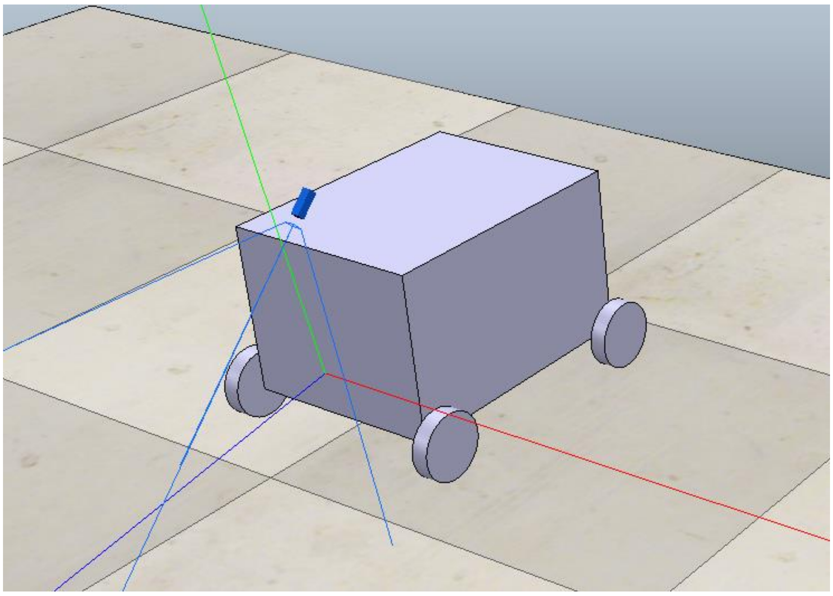
*21 巡线小车终止计时器*

至此，计时器设计完毕。

# 3. 效果展示：

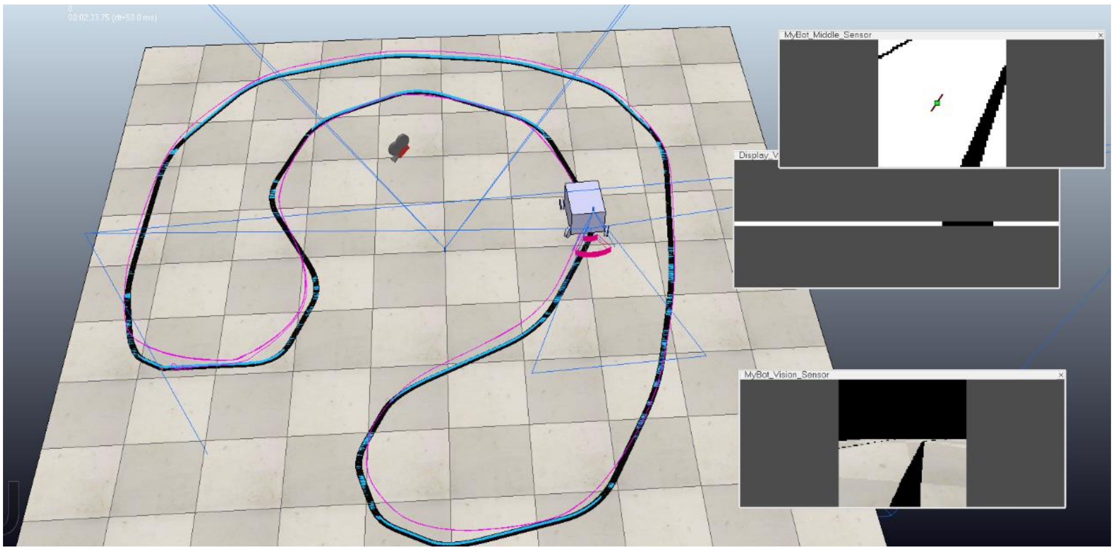
- **简单版本**

以下是简单版本的视觉巡线小车的静态演示图，注意到它的视觉传感器是斜向前下方的。



*22 简单版本的视觉巡线小车静态演示*

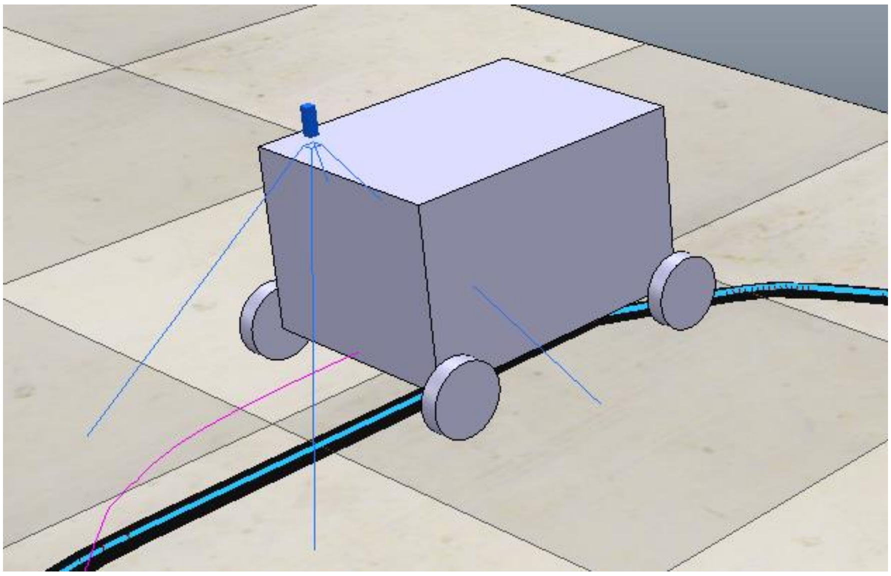
下面是机器人巡线的示意图，详细细节可参考视频《机器人导论 HW3 VREP 下实现基于 PID 控制的视觉巡线小车》，若无法跳转超链接，可复制并打开以下视频链接：https://www.bilibili.com/video/av70787064/




*23 简单版本的巡线效果*

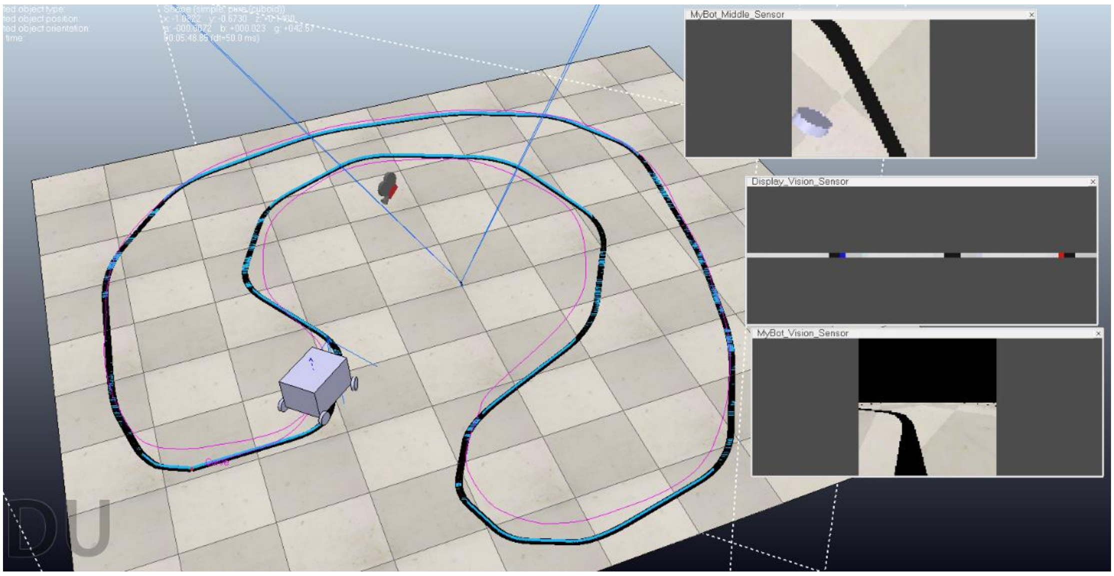
- **最终版本**

在最终版本中，为了提高对场景的鲁棒性，我们将视觉传感器垂直向地，其静态演示图如下。



*24 最终版本的视觉巡线小车静态演示*

在巡线测试中，由于最终版本的巡线小车未寻找到较优的 PID 参数，其运行轨迹与目标轨迹并不如简单版本那么贴合，循迹效果如下图所示，演示视频请参见 report 文件夹下的“演示视频.mp4”文件。




25　复杂版本的巡线效果

# 4. 存在问题：

本次实验，还是颇有挑战性的，例如在如何提升巡线小车对复杂场景的鲁棒性上，我们遇到了挺多问题，问题根源其实是我们把问题复杂化了，然后由于能力不足，导致时间和精力的浪费，最后耽误了今日下午的巡线比赛，我们队员也对自己表示很遗憾。

在实验的一开始，我们就打算使用 Hough 直线检测技术，这也正是计算机视觉课程正在上课的内容。我们的设想是，针对视觉传感器传回的数据，我们对其进行 Hough 变换，找到其中的直线，随后判断特征直线的夹角，进而求取 PID 控制的偏差量。

由于 V-REP 并不支持 Hough 变换的调用，因此我们使用了 V-REP 的 Remote API 的功能，在环境配置上踩了不少的坑。我们通过调用 OpenCV 的相关接口，先后对图像进行二值化、腐蚀、边缘检测（Canny 算子）处理，最后引入 Hough 变换操作，从而成功在图像上绘制直线。在求取特征直线的夹角时，我们尝试了多种方法，都未得到较好的效果，并且由于算法设计并不完善，导致该版本下的巡线机器人并无鲁棒性。由于耽误了太长的时间，且并未获得出色的实验结果，我们放弃了此方案。

在查阅相关机器人技术论坛后，我们了解到了一种夹角检测方法，即在经过预处理的图像上（如二值化、腐蚀），我们通过调用 OpenCV 的 minAreaReact 方法，在图像上绘制包覆输入信息的最小斜矩形，并获取斜矩形的角度。有了这个想法，我们也快速完成了基本 demo 的验证，发现该算法对于 U 型弯和 V 型弯极其容易造成误判，导致巡线失败。再一次，经过多次尝试后，我们放弃了此方案。

期间，我们还实现了另一种“解决方案”，使用 OpenCV 探测图像的 5 个区域是否有黑色线，即用 OpenCV 视觉模拟 5 路光感巡线小车。但由于我们并未找到较好的 5 路区域的探测结果的表示方式，这导致了我们并未实现 PID 控制。后来，在完成最终版本代码后，我尝试在晚上搜索，找到了一篇文章，里面对 5 路巡线小车的 PID 控制有详细的阐述，打算日后好好研究一下，文章链接：https://www.arduino.cn/thread-49918-1-1.html 。

最终，我们还是回到了通过检测图片左右两部分的黑色像素的差异，实现巡线的实现方案上。奇怪的是，在前面摸索弯路时，我们并没有想到此方案，这是我们对本次实验的自我表现深感遗憾的主要原因。

不过，尽管走了很多弯路，我们还是了解到了很多机器人巡线的前沿信息，例如使用强

化学习来实现复杂场景下的巡线，也因此了解到机器人巡线的更多优秀的算法，只是现在的我们还不足以将其实现，希望我们能够在日后的学习中，能够继续学习各种各样的控制理论与算法思想，不断巩固老师和助教传授的知识。

## 5. 附录：

**简单版本的视觉巡线小车的控制脚本（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 Middle Sensor.
    Vision_Sensor = sim.getObjectHandle('MyBot_Middle_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

    calc_vision_angle = function()
        local y = 0

        image = sim.getVisionSensorImage(Vision_Sensor, 0, y, 64, 1, 0)
        sim.setVisionSensorImage(Display_Vision_Sensor, image)

        local first = true
        local endpoint_left = 0
        local endpoint_right = 192
        for i = 1, 192, 1 do
            if image[i] == 0 then
                if first then
                    endpoint_left = i
                    first = false
                end
                endpoint_right = i
            end
        end

        local h = 64 - y
        local angle = math.atan2(32 - (endpoint_left + endpoint_right)
/ 6, h) * 180 / math.pi
        return angle
    end

    Display_Vision_Sensor = sim.getObjectHandle('Display_Vision_Sensor'
)

    -- The Main Loop
    while sim.getSimulationState() ~= sim.simulation_advancing_abouttos
top do
        -- P Control
        kp = 2
        error = calc_vision_angle()
```

```lua
        -- I Control
        ki = 0
        integral = integral + error
        -- D Control
        kd = 0
        derivative = error - prev_error
        prev_error = error
        -- PID Control
        steeringAngle = kp * error + ki * integral + kd * derivative

        if (math.abs(steeringAngle)) > 20 then
            motor(10)
        else
            motor(20)
        end

        -- Steer for calculated angle.
        steer(steeringAngle)

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

**最终版本的视觉巡线小车控制脚本（Threaded）**

```lua
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 Middle Sensor.
    Vision_Sensor = sim.getObjectHandle('MyBot_Middle_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.
```

```lua
    -- +: 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

    Display_Vision_Sensor = sim.getObjectHandle('Display_Vision_Sensor'
)

    -- The Main Loop
    while sim.getSimulationState() ~= sim.simulation_advancing_abouttos
top do
        data = sim.getVisionSensorImage(Vision_Sensor + sim.handleflag_
greyscale, 0, 0, 64, 64)
        greyL = 0
        greyR = 0

        for i = 1, 32, 1 do
            for j = 30, 50, 1 do
                greyL = greyL + data[(i - 1) * 64 + j]
            end
        end
```

```lua
        for i = 33, 64, 1 do
            for j = 30, 50, 1 do
                greyR = greyR + data[(i - 1) * 64 + j]
            end
        end

        sim.setVisionSensorImage(Display_Vision_Sensor, data)

        -- P Control
        kp = 2.3
        error = (greyR - greyL) / 10
        -- I Control
        ki = 0
        integral = integral + error
        -- D Control
        kd = 0
        derivative = error - prev_error
        prev_error = error
        -- PID Control
        steeringAngle = kp * error + ki * integral + kd * derivative

        if (math.abs(steeringAngle)) > 20 then
            motor(1)
        else
            motor(2)
        end

        -- Steer for calculated angle.
        steer(steeringAngle)

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