机器人导论

课程作业: assignment 1

组员：16305204 郑佳豪

16326086 王润锋

提交日期：2019-09-15

Deadline：2019-09-15

1. 任务概要：

- 参考文档中 Tutorial – BubbleRob Tutorial 部分，学习：
  - 车身与轮子物理引擎的设计
  - 动力学模型等参数的设置
  - 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
- 搭建四轮小车
  - 搭建四轮小车，小车搭载一个单目彩色摄像头
  - 尺寸、底盘参数可参考 DJI RoboMaster S1 进行设计
  - 不采用麦克纳姆轮
  - 编写简单脚本使得小车能呈 S 型路线行走，当碰到障碍物后能够绕行

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

2. 完成情况：

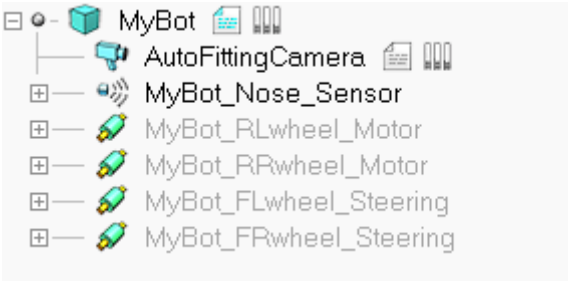
总体完成情况如下：

- 已初步熟悉 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、控制脚本的编写等功能。
- 已成功搭建装载一个单目彩色摄像头的四轮小车，使用 Ackermann 运动学模型，前轮可转向，且未使用麦克纳姆轮，能呈 S 型路线行走，碰到障碍物能够绕行。

1. 总体设计

机器人由长方体车身和四个轮子、传感器、彩色摄像头组成，两个前轮主要负责转向和动力输出，两个后轮作为从动轮，传感器负责检测前方障碍物并绕行。

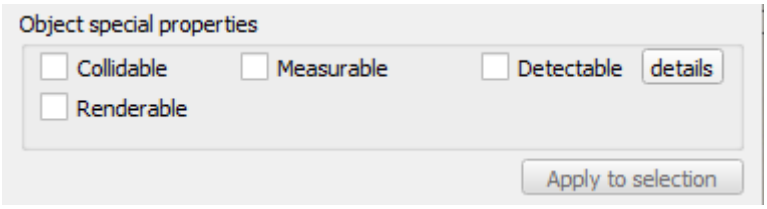
以下是 MyBot 机器人的总体设计层次图。其中 FLwheel_Steering 和 FRwheel_Steering 为前轮；RLwheel_Motor 和 RRwheel_Motor 为后轮；Nose_Sensor 为传感器，搭载单目彩色摄像头；AutoFittingCamera 为自适应摄像头，用于拍摄机器人运动时的姿态。



1　机器人总体设计

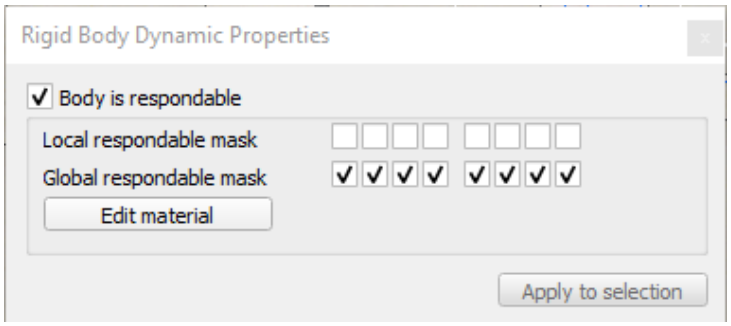
2. 车身构建

我们使用 Cuboid 作为车身主体，为避免主体与其他部件冲突，我们需要设置其 Object 属性，关闭 Collidable 等属性。



2　车身 Object 属性

同时，我们需要关闭 Local Respondable Mask 属性。

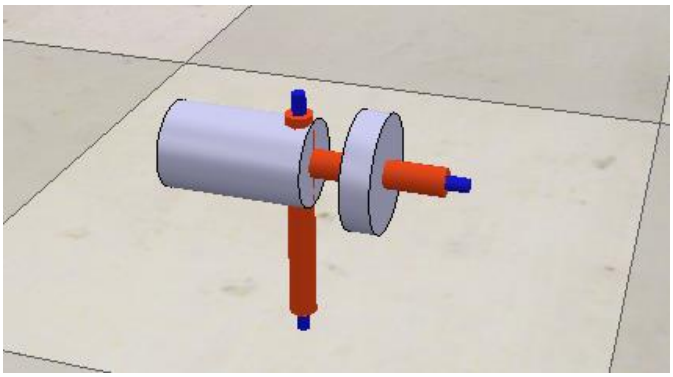


3　车身 Object 属性

3. 车轮构建

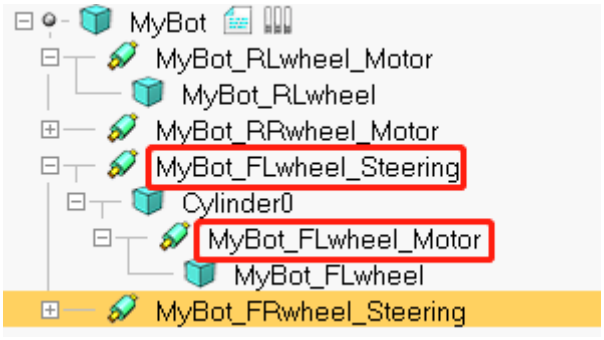
我们将机器人的车轮划分为两组：前轮组（Front）和后轮组（Rear）。前轮组负责转向和动力，后轮组则为从动轮。

由于我们希望前轮组既负责转向功能，又负责动力输出。为实现这两个功能，我们需要两个转动关节，其中一转动关节竖向放置，用于控制转向角度，另一转动关节横向放置，用于输出动力。



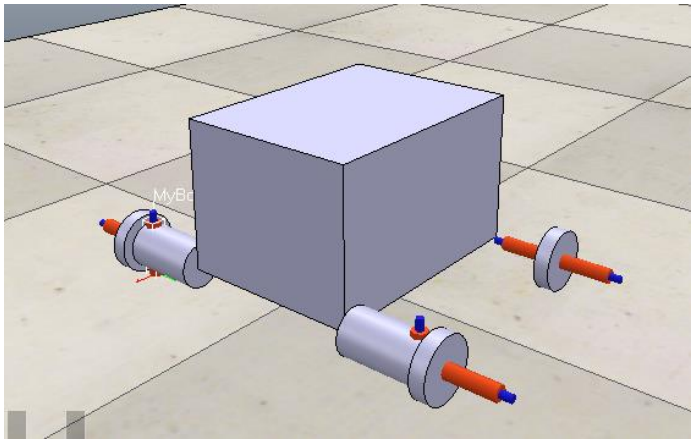
4　转向轮示意图

以下是 MyBot 的各车轮的层次图。注意到，以前左轮（FLwheel）为例，其由竖向关节（Steering）和横向关机（Motor）构成；与之不同的是，后左轮（RLwheel）则由横向关节（Motor）构成。



5　车轮层次图

以下是机器人的车轮拆解示意图（已设置各部件的可视状况）。拥有两个转动关节的即为前轮组，仅拥有一个转动关节的为后轮组。



6　车轮拆解示意图

4.　直线运动

为实现直线运动，我们需要编写脚本。而在这里，我们使用 Threaded Child Script 进行脚本的填写，其涉及到一个很重要的函数，名为 sysCall_threadmain，这是一个需要用户填充自定义代码的函数模板，模板如下。

```
function sysCall_threadmain()
    -- Put some initialization code here

    -- Put your main loop here, e.g.:
    --
    -- while sim.getSimulationState()~=sim.simulation_advancing_abouttostop do
    --     local p=sim.getObjectPosition(objHandle,-1)
    --     p[1]=p[1]+0.001
    --     sim.setObjectPosition(objHandle,-1,p)
    --     sim.switchThread() -- resume in next simulation step
    -- end
end
```

7　Threaded Script 模板

在 sysCall_threadmain 函数模板中，我们需要将变量的初始化放置在主循环之外，机器人的运动逻辑放置在主循环之中，这是需要注意的地方。

现在，我们进行直线运动的脚本编写。我们知道，在前面的车轮构建部分中，我们添加了两个前进电机。所以，我们可通过 setJointTargetVelocity 设置这两个前进电机同样的运行速度，实现机器人的直线运动。

为此，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
```

8　motor 工具函数

在上述脚本中，我们通过 getObjectHandle 获取两个前进电机的实例。在 motor 工具函数中，我们使用 setJointTargetVelocity 设置两个电机同样的运行速度。由于我们所添加的脚本是 Threaded 类型，我们需要在 while 主循环内，执行了 motor(0.8) 命令。按下模拟按钮后，我们即可观察到机器人进行直线运动。

下面是直线运动的全部脚本代码，后续由于篇幅限制，只贴出关键部分的代码。

```
function sysCall_threadmain()
    -- Initialization
    -- 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

    -- The Main Loop
    while sim.getSimulationState() ~= sim.simulation_advancing_abouttostop
do      -- Set the speed of the motors.
        motor(0.8)

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

9　前进运动的完整代码

5. S 型运动

在实现机器人的直线运动后，我们需要在其基础上，使得机器人能够以 S 型路线前进。我们的思想很简单：在电机启动的情况下，控制转向角度，使其按照余弦函数变化，即可控制机器人的前进路线为 S 型路线。

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
```

10　steer 工具函数

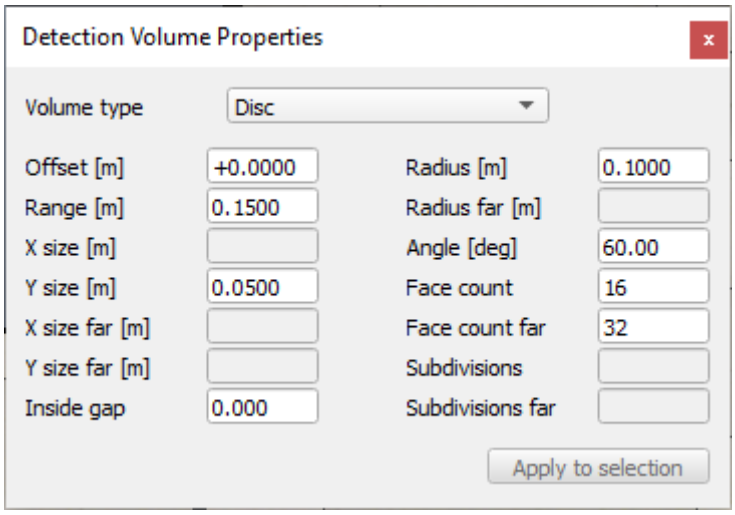
在 steer 工具函数中，我们为其传入 d 和 l 参数，它们分别与左右轮间距和前后轮间距有关，是转向运动学模型所必需的参数。我们通过 getObjectHandle 获取两个转向关节的实

例，通过 setJointTargetPosition 控制两个转向关节的角度，进而控制机器人的前进方向。

6. 传感器

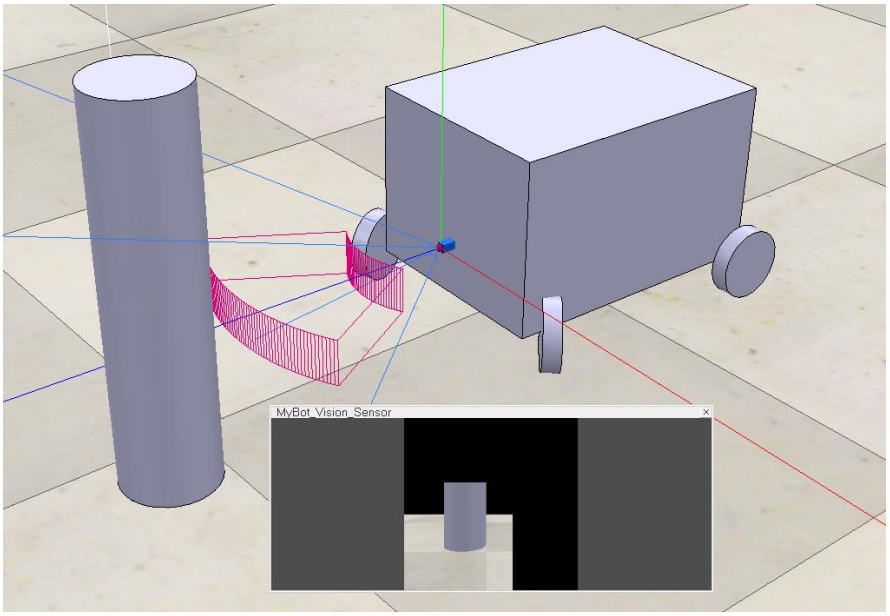
我们在 Body 添加 Proximity Sensor，用于搭载单目彩色摄像头，用于避障算法的障碍物检测。

下面是 Nose Sensor 的 Volume 的各项属性，我们将 Volume Type 设置为 Disc，并配置其的角度和半径等参数。



11 Nose Sensor 参数设置

在设置好 Nose Sensor 的属性后，我们为其添加单目彩色摄像头，并设置好摄像头属性，最后添加 Floating View 用于显示摄像头的视野。



12 摄像头工作示意图

7. 简单避障

我们使用这样的避障算法：当机器人在往左边运动的过程中，探测到了障碍物，则将转向角度设置为往右边运动的最大角度，该操作持续一段时间，直至无探测到障碍物或达到阈

值时间，随后设置转向角度为往左边运动的最大角度，继续原有的运动。

我们使用 Nose_Sensor 作为障碍物的探测感应器，通过获取其探测结果，来决定是否采取避障措施。

```
result = sim.readProximitySensor(Nose_Sensor)
-- If there is an obstacle.
if (result > 0) then
    avoidUntilTime = sim.getSimulationTime() + 5
end
```

13　检测障碍物

若探测到障碍物，设置避障操作的阈值时间，同时我们根据当前的运动方向，设置避障所需要的最大转动角度和方向。

```
if steeringAngle > 0 then
    -- When obstacle is in the left side, turn right.
    steeringAngle = -steeringAngleAbs
else
    -- When obstacle is in the right side, turn left.
    steeringAngle = steeringAngleAbs
end
-- Steer for obstacle avoidence.
steer(steeringAngle)
```

14　设置避障转向角度和方向

在超出阈值时间后，即机器人已远离障碍物后，我们重新设置机器人的转动方向为原有的运动方向，且转动角度设置为最大转动角度。

```
if steeringAngle > 0 then
    -- When previous direction is left.
    steeringAngle = steeringAngleAbs
else
    -- When previous direction is left.
    steeringAngle = -steeringAngleAbs
end
-- Restore the previous direction.
steer(steeringAngle)
```

15　恢复原有转向方向和角度

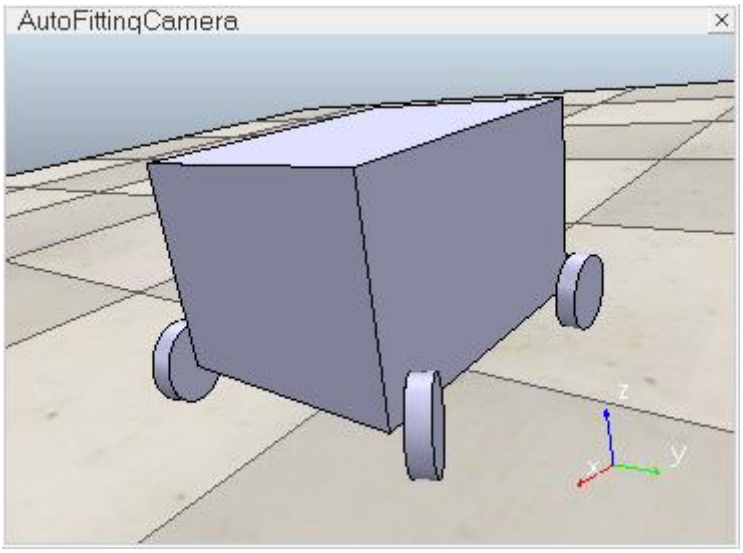
若机器人在前进过程中，未探测到障碍物，则其转动角度一直按余弦函数变化。

```
-- Generate cos-like angle sequence.
tick = tick + 0.1
steeringAngle = steeringAngleAbs * math.cos(tick * 0.05)
-- Steer for specific angle.
steer(steeringAngle)
```

16　按余弦函数变化的转向角度

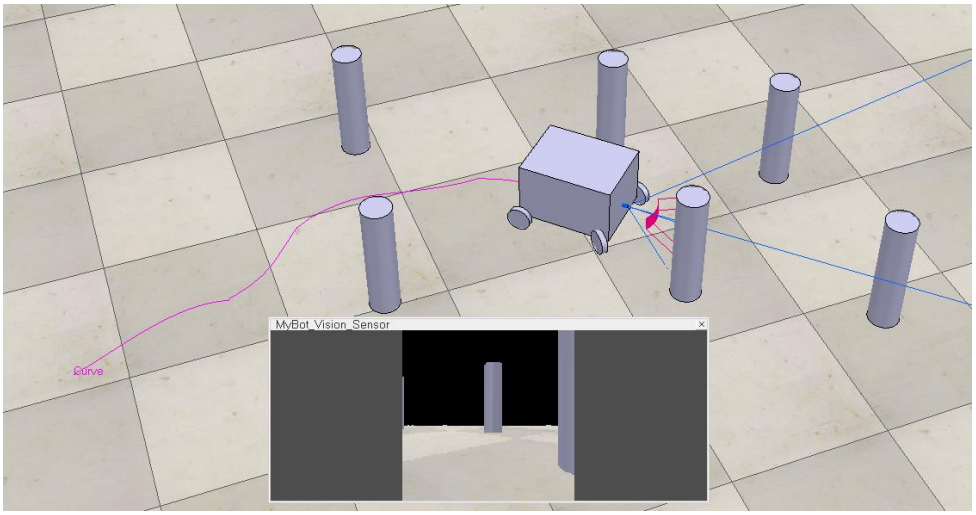
3. 效果展示：

以下是机器人转向的静态示意图，已隐藏转动关节等部件。



17　机器人转向静态示意图

下面是机器人避障的示意图，详细细节可参考视频《机器人导论 HW1 简单避障小车》，若无法跳转超链接，可复制并打开此视频链接：https://www.bilibili.com/video/av67860978/ 。



18　机器人避障路线

4. 存在问题：

本次搭建的四轮小车存在以下未解决的问题：避障算法过于简单，由于只装载了一个 Nose Sensor 且传感器类型为 Vision，我们无法获取障碍的方位信息，这使得我们的算法的避障方式不具备鲁棒性。

在搭建机器人的过程中，我们还遇到了其他的问题，但最终都解决了。在构建前轮转向关节时，我们一开始使用的是 Cuboid 连接竖向关节（用于转向）和横向关节（用于前进），后来在模拟时出现了剧烈的抖动现象。一开始，我们发现 Cuboid 物理引擎属性被设置了 Respondable 和 Collidable 等属性，怀疑是属性设置的原因，调整属性后，发现问题仍存在。最终，我们在比对 V-REP 提供的 Ackermann 模型时，发现问题可能出现在竖向关节和横向关节的连接上。于是，我们通过将 Cuboid 更换为 Cylinder，模拟过程出现的剧烈抖动问题就被成功解决。但当转弯角度过大时，前轮会与车体碰撞，使得前轮转弯角度无法到达预期角度，我们通过设置车体的 Local Respondable Mask 全为空，以及关闭 Collidable 等属性，修复了车体阻碍前轮转向的问题。

模拟期间，我们发现小车在未启用任何脚本的情况，出现了异常的移动情况。由于在 Tutorial 中出现了类似情况的解决办法，我们尝试用其提及的增大 Body 和 Wheel 的质量来解决此问题。经过了多次对不同部件的质量调整，我们将小车的异常移动稳定到一个可以接受的范围，但也算成功解决了此问题。

5. 附录：

MyBot 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 Sensing Nose.
    Nose_Sensor = sim.getObjectHandle('MyBot_Nose_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
    -- It represents the abs value of the max steering angle.
    steeringAngleAbs = 30
    -- It's used to generate cos-like angle sequence.
    tick = 0
```

```
    lock = 0
    -- Angle to steer.
    -- +: left
    -- -: right
    steeringAngle = -steeringAngleAbs
    -- It's used to control the total time of avoiding.
    avoidUntilTime = 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
        result = sim.readProximitySensor(Nose_Sensor)

        if (result > 0) then
            avoidUntilTime = sim.getSimulationTime() + 5
        end

        if (avoidUntilTime < sim.getSimulationTime()) then
            if (lock == 1) then
                lock = 0
                if steeringAngle > 0 then
                    steeringAngle = steeringAngleAbs
                else
                    steeringAngle = -steeringAngleAbs
                end
                -- Restore the corresponding tick.
                tick = math.acos(steeringAngle / steeringAngleAbs) / 0.
05
```

```
            end
            tick = tick + 0.1
            steeringAngle = steeringAngleAbs * math.cos(tick * 0.05)
            -- Normal speed.
            motor(0.5)
        else
            if (lock == 0) then
                lock = 1
                if steeringAngle > 0 then
                    -- When obstacle is in the left side, turn right.
                    steeringAngle = -steeringAngleAbs
                else
                    -- When obstacle is in the right side, turn left.
                    steeringAngle = steeringAngleAbs
                end
                -- Speed up when avoids obstacle.
                motor(1)
            end
        end

        -- Steer for specific angle.
        steer(steeringAngle)

        -- 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
```