工程实践与科技创新 IV-E

Lecture 3: 移动机器人的运动学模型与仿真

授课老师：张晗、王景川 *2022 年春季*

1 移动机器人的 unicycle 运动学模型

移动机器人是最被广泛被使用的机器人类型之一。如今在自动驾驶领域被广泛地运用（如果把汽车也看成一种移动机器人的话）。对于自动驾驶的汽车而言，一个核心需求是：把自身相对于道路中的位置定位出来，以及如何依据自身的位置，控制油门、刹车和转向等控制量，使得车辆能够安全地超车、避开事故、向目的地行驶。

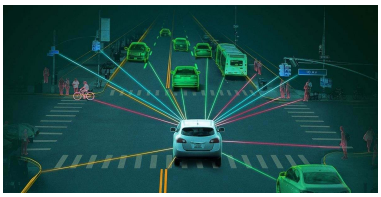


Figure 1: 自动驾驶汽车

无论是在定位还是控制，我们都需要对车辆进行建模。机器人（车辆）的模型通常分为两种：运动学模型和动力学模型。运动学模型表示的是机器人的位置与速度之间的关系，速度通常作为控制输入量，一般用于对动态性能不高的场景；而动力学模型表示的是机器人位置、速度、加速度与力（力矩）之间的关系，一般用于对动态性能要求很高的场景，如涉及到力反馈的场景等。

在机器人定位场景中，通常不涉及到力与力矩，对系统动态性能要求并不是很高（但对于高速运动的汽车的决策控制而言，并非如此），所以我们一般使用机器人系统的运动学来对其进行建模。

车辆底盘的模型如 Fig. 2所示。其中，后轮轴的中点位置坐标为 $[x, y]^T$，汽车的朝向为 θ，前轮的转角为 ϕ，前轮轴和后轮轴之间的距离为 L。其运动学需满足所谓的“非完整型约束”（nonholonomic constraint）：垂直于轮子方向上没有速度分量，即：对后轮轴的中点来说，有：

$$\dot{x}\sin\theta - \dot{y}\cos\theta = 0,$$

而对前轮轴的中点来说，有：

$$\frac{d}{dt}(x + L\cos\theta)\sin(\theta+\phi) - \frac{d}{dt}(y + L\sin\theta)\cos(\theta+\phi) = 0$$

我们可以验证，下列微分方程满足非完整性约束：

$$\begin{aligned}\dot{x} &= v\sin\theta,\\ \dot{y} &= v\cos\theta,\\ \dot{\theta} &= \frac{v\tan\phi}{L},\end{aligned}$$

其中，v 是后轮轴终点的线速度。在这个系统中，我们的控制量为 v（对应油门）和前轮的转角 ϕ。

该系统为非线性系统，非常复杂，尤其是车辆的朝向 θ 由 v 和 ϕ 共同决定。为了简化控制器设计，在移动机器人中，人们设计了如 Fig. 3所示结构的机器人，它同样满足非完整性约束。这种机器人被称为差速模型机器人，其运动学模型为

$$\begin{aligned}\dot{x} &= v\sin\theta,\\ \dot{y} &= v\cos\theta,\\ \dot{\theta} &= \omega,\end{aligned}$$

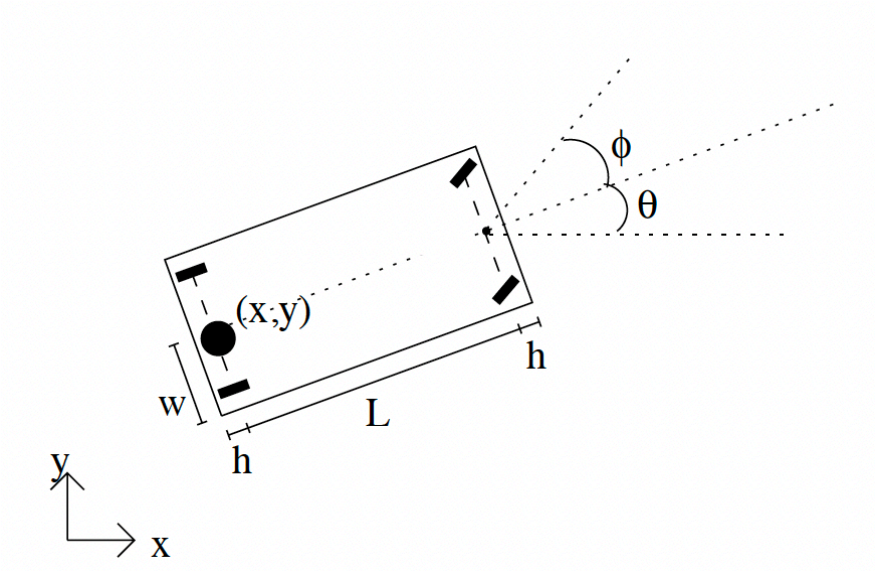


Figure 2: 车辆底盘模型

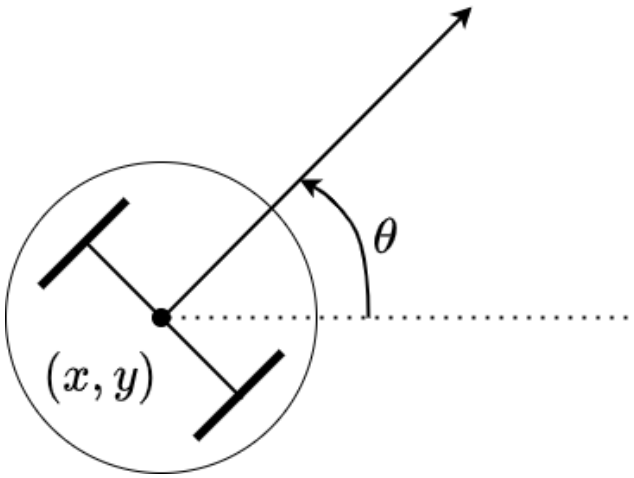


Figure 3: Unicycle 机器人模型

这种机器人可以独立地通过控制两个轮子的旋转速度，实现线速度 v 和角速度 w 的独立控制，这使得该种类型的机器人可以做到在线速度 v 为零的时候原地旋转；这是 Fig. 2中的机器人无法做到的。本课程中，主要考虑 Fig. 3的情况。

考虑到机器人在运动过程中，可能会出现打滑等不可预知的现象，所以我们在上述微分方程中加入高斯白噪声 w，即

$$\dot{X} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}}_{f(X,u)} + w,$$

其中，$\mathbb{E}[w] = 0$，$\mathrm{cov}(w(\tau_1), w_(\tau_2)) = \delta(\tau_1 - \tau_2)\tilde{Q}$，

$$\delta(\tau_1 - \tau_2) = \begin{cases} 1, & \text{如果}\tau_1 = \tau_2, \\ 0, & \text{其他} \end{cases}.$$

为了在计算机中实现上述机器人运动学模型，我们需要对其进行离散化。我们假设第 t_k 时刻与 t_{k+1} 时刻之间的机器人状态保持不变，即：$f(X(\tau), u(\tau)) \approx f(X(t_k), u(t_k))$，可得

$$\begin{aligned} X(t_{k+1}) &= X(t_k) + \int_{t_k}^{t_{k+1}} f(X(\tau), u(\tau))d\tau + \int_{t_k}^{t_{k+1}} w(\tau)d\tau \\ &\approx X(t_k) + \int_{t_k}^{t_{k+1}} f(X(t_k), u(t_k))d\tau + \underbrace{\int_{t_k}^{t_{k+1}} w(\tau)d\tau}_{w_k} \\ &= X(t_k) + (t_{k+1} - t_k) f(X(t_k), u(t_k)) + w_k, \end{aligned}$$

其中 $\mathbb{E}[w_k] = \int_{t_k}^{t_{k+1}} \mathbb{E}[w(\tau)]d\tau = 0$ 并且

$$\begin{aligned} &\mathrm{cov}(w_{k_1}, w_{k_2}) = \mathbb{E}\left[\int_{t_{k_1}}^{t_{k_1+1}} w(\tau_1)d\tau_1 \int_{t_{k_2}}^{t_{k_2+1}} w(\tau_2)d\tau_2\right] \\ &= \left[\int_{t_{k_2}}^{t_{k_2+1}} \int_{t_{k_1}}^{t_{k_1+1}} \mathbb{E}[w(\tau_1)w(\tau_2)]d\tau_1 d\tau_2\right] = \left[\int_{t_{k_2}}^{t_{k_2+1}} \int_{t_{k_1}}^{t_{k_1+1}} \delta(\tau_1 - \tau_2)\tilde{Q}d\tau_1 d\tau_2\right] \\ &= \begin{cases} (t_{k+1} - t_k)\tilde{Q} := Q_k, & \text{当}k_1 = k_2 = k \\ 0, & \text{其他} \end{cases}. \end{aligned}$$

对于线性系统而言，我们可以有更精确的离散化方程：

$$\begin{aligned} &\dot{x} = \tilde{A}x + \tilde{B}u + w \\ &x(t_{k+1}) = e^{\tilde{A}(t_{k+1} - t_k)}x(t_k) + \int_{t_k}^{t_{k+1}} e^{\tilde{A}(t_{k+1} - s)}\tilde{B}u(s)ds + \int_{t_k}^{t_{k+1}} e^{\tilde{A}(t_{k+1} - s)}w(s)ds \\ &x(t_{k+1}) \approx \underbrace{e^{\tilde{A}(t_{k+1} - t_k)}}_{A}x(t_k) + \underbrace{\int_0^{t_{k+1} - t_k} e^{\tilde{A}\tau}d\tau\tilde{B}}_{B}u(t_k) + \underbrace{\int_{t_k}^{t_{k+1}} e^{\tilde{A}(t_{k+1} - s)}w(s)ds}_{w_k}, \end{aligned}$$

其中，$\mathbb{E}[w_k] = \int_{t_k}^{t_{k+1}} e^{\tilde{A}(t_{k+1}-s)}\mathbb{E}[w(s)]ds = 0$，且有

$$\begin{aligned}
\mathrm{cov}(w_{k_1}, w_{k_2}) &= \mathbb{E}\left[\int_{t_{k_1}}^{t_{k_1+1}} e^{\tilde{A}(t_{k+1}-\tau_1)}w(\tau_1)d\tau_1 \int_{t_{k_2}}^{t_{k_2+1}} e^{\tilde{A}(t_{k+1}-\tau_2)}w(\tau_2)d\tau_2\right] \\
&= \left[\int_{t_{k_2}}^{t_{k_2+1}} \int_{t_{k_1}}^{t_{k_1+1}} e^{\tilde{A}(t_{k+1}-\tau_1)}\mathbb{E}[w(\tau_1)w(\tau_2)]e^{\tilde{A}^T(t_{k+1}-\tau_2)}d\tau_1 d\tau_2\right] \\
&= \left[\int_{t_{k_2}}^{t_{k_2+1}} \int_{t_{k_1}}^{t_{k_1+1}} e^{\tilde{A}(t_{k+1}-\tau_1)}\delta(\tau_1-\tau_2)\tilde{Q}e^{\tilde{A}^T(t_{k+1}-\tau_2)}d\tau_1 d\tau_2\right] \\
&= \begin{cases} \int_0^{t_{k+1}-t_k} e^{\tilde{A}s}\tilde{Q}e^{\tilde{A}^T s}ds := Q_k, & \text{如果} k_1 = k_2 = k, \\ 0, & \text{其他.} \end{cases}
\end{aligned}$$

Q_k 一般具有结构。以 $\tilde{A} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ 为例，令 $t_{k+1} - t_k = \Delta t_k$，$\tilde{Q} = \begin{bmatrix} q_1 & q_2 \\ q_2 & q_3 \end{bmatrix}$，我们有

$$Q_k = \int_0^{\Delta t_k} e^{As}\tilde{Q}e^{A^T s}ds = \int_0^{\Delta t_k} \begin{bmatrix} q_1 + 2q_2 s + q_3 s^2 & q_2 + q_3 s \\ q_2 + q_3 s & q_3 \end{bmatrix} ds = \begin{bmatrix} q_1\Delta t_k + q_2\Delta t_k^2 + \frac{1}{3}q_3\Delta t_k^3 & q_2\Delta t_k + \frac{1}{2}q_3\Delta t_k^2 \\ q_2\Delta t_k + \frac{1}{2}q_3\Delta t_k^2 & q_3\Delta t_k \end{bmatrix}.$$

回想一下 Kalman 滤波器的迭代步骤，可分为计算 $p(x_{t+1}|y_{1:t})$ 的 prediction step:

$$\begin{aligned}
\hat{x}_{t+1|t} &= A\hat{x}_{t|t} + Bu_t, \\
P_{t+1|t} &= AP_{t|t}A^T + Q_k
\end{aligned}$$

以及把新的一帧传感器观测数据加入考虑的对 $p(x_{t+1}|y_{1:t+1})$ 的计算（Update）:

$$\begin{aligned}
\hat{x}_{t+1|t+1} &= \hat{x}_{t+1|t} + P_{t+1|t}C^T(CP_{t+1|t}C^T + \Sigma_v)^{-1}(y_{t+1} - C\hat{x}_{t+1|t}) \\
P_{t+1|t+1} &= P_{t+1|t} - P_{t+1|t}C^T(CP_{t+1|t}C^T + \Sigma_v)^{-1}CP_{t+1|t}.
\end{aligned}$$

在实际机器人实现中，我们不可能做到将控制周期与传感器观测完全同步，如下图所示：实际机

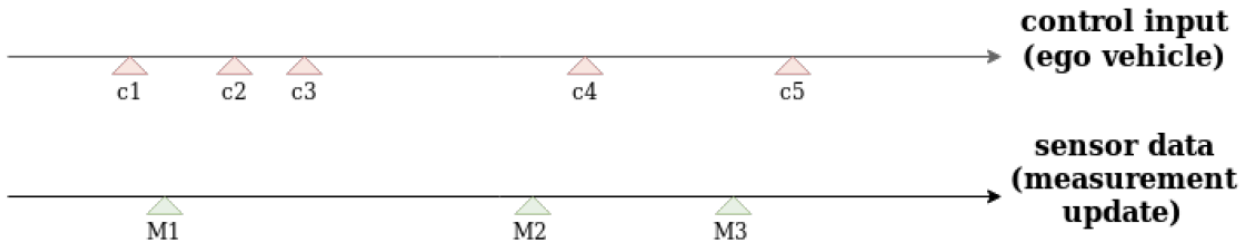


Figure 4: 实际机器人控制与传感器观测周期

器人的控制频率可能非常快，一般在 50Hz ~ 100Hz 不等。并且由于操作系统的指令周期、ROS 节点通信等问题（若不是实时操作系统），机器人的控制指令下发周期具有一定误差，并不完全均匀；而以相机为例，通常观测频率为 30Hz。这就会造成如 Fig. 4中所示的失步问题。在 Fig. 4中，若机器人的控制指令和传感器数据均由回调函数实现，则我们需要在实现时，对 Kalman 滤波器作如下处理：

1. 在 C_2 时刻，计算 $C_2 - M_1$ 的时长，对过程噪声做相应的调整后，对 M_1 时刻的 updated 状态进行 prediction step，此时对状态的估计是在 C_2 时刻的估计；

2. 在 C_3 时刻，计算 $C_3 - C_2$ 的时长，对过程噪声做相应的调整后，对 C_2 的 predicted 状态继续进行 prediction step（因为我们没有观测的更新），此时对状态的估计是在 C_3 时刻的估计；

3. 在 M_2 时刻，获得了一帧传感器数据，计算 $M_2 - C_3$ 的时长，对过程噪声做相应的调整后，先在 M_2 时刻先对 C_3 时刻对 C3 的 predicted 状态继续进行 prediction step，再依据 M_2 时刻的传感器数据进行 update；此时对状态的估计是在 M_2 时刻的估计（为了要进行 update，我们必须先进行 prediction）；

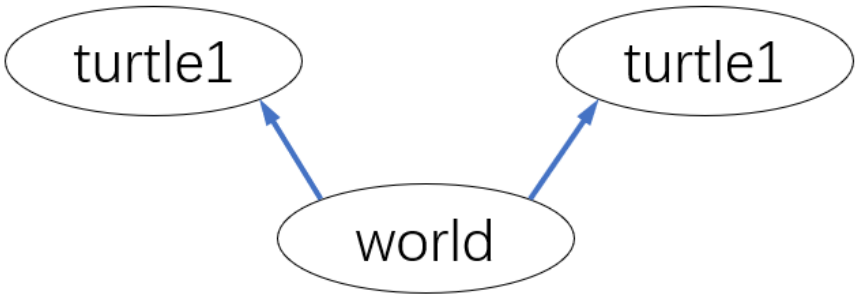
4. 在 C_4 时刻，计算 $C_4 - M_2$ 的时长，对过程噪声做相应的调整后，对 M_2 的 updated 状态进行 prediction step，此时对状态的估计是在 C_4 时刻的估计；

5. ...

2 ROS tf

2.1 概述

tf 是 ROS 提供的一个功能包，可以帮助用户处理多个坐标系之间的相对位置关系。如，在世界坐标系（world）中，有两只乌龟坐标系（turtle1，turtle2），我们可以使用 sendTransform() 函数给定二者相对于世界坐标系的位姿。tf 即建构出这样的坐标系关系：



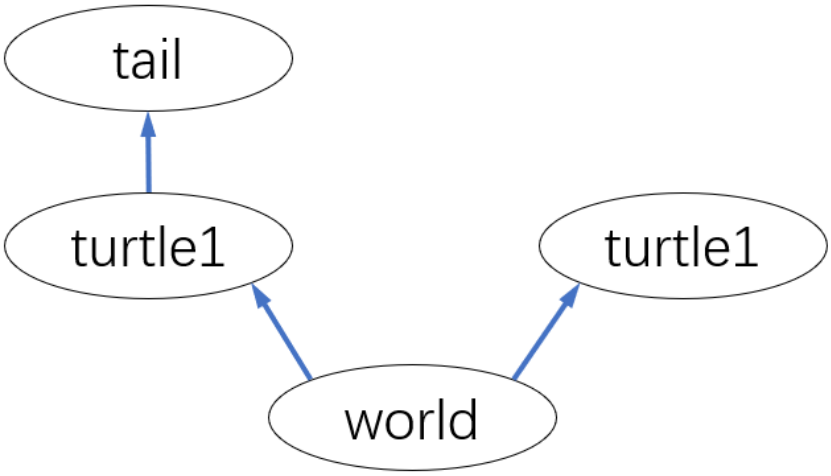
此时，如果我们想知道 turtle1 和 turtle2 的相对位姿关系，我们可以直接使用 lookupTransform() 查询，而不用自己写坐标变化的代码。

进一步，我们可以继续使用 sendTransform() 给出 turtle1 的尾巴相对于 turtle1 的位姿，此时 tf 会建立出这样的关系图：

这时我们可以用 lookupTransform() 查询图中任两坐标系的相对位姿。注意，tf 建立的坐标系关系必须是树形结构，不允许闭环，每个坐标系（除了 world 坐标系）必须仅有一个亲代（parent），但可以有多个子代（children）

2.2 四元数

我们知道，描述空间中两个刚体的相对位置关系，可以表示为二者连体基之间的坐标变换。这种变换可以分解为平移和旋转，即，对于向量 $\boldsymbol{\alpha}$，在参考基 $(\boldsymbol{e}_1^r, \boldsymbol{e}_2^r, \boldsymbol{e}_3^r)$ 下坐标为 $\boldsymbol{\alpha}^r$，在研究对象连体基



$(\boldsymbol{e}_1^b, \boldsymbol{e}_2^b, \boldsymbol{e}_3^b)$ 下坐标为 $\boldsymbol{\alpha}^b$, 成立下式:

$$\boldsymbol{\alpha}^b = \boldsymbol{A}^{rb}(\boldsymbol{\alpha}^r - \boldsymbol{r}^b) = \boldsymbol{A}^{rb}\boldsymbol{\alpha}^r + \boldsymbol{t} \tag{1}$$

其中 $\boldsymbol{A}^{rb}$ 是方向余弦阵，表示旋转；$\boldsymbol{r}^b$ 是研究对象连体基的基点在参考系下的坐标，表示平移。

方向余弦阵 $A^{rb} \in \mathbb{R}^{3\times3}$，且是单位正交阵；用 9 个数字表达 3 个自由度的旋转非常浪费。ROS tf 使用四元数描述旋转。四元数是 Hamilton 找到的一种扩展的复数。一个四元数拥有 1 个实部和 3 个虚部，

$$\boldsymbol{q} = q_0 + q_1 i + q_2 j + q_3 k \tag{2}$$

其中 i, j, k 是三个虚数单位，满足:

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = k = -ji \\ jk = i = -kj \\ ki = j = -ik \end{cases} \tag{3}$$

我们能用一个纯虚四元数表示一个空间点，或用一个单位四元数表示一个旋转。上面的四元数 $\boldsymbol{q}$ 表示绕单位向量 $\boldsymbol{n}$ 旋转 θ 弧度，其中

$$\theta = 2\arccos q_0 \tag{4}$$

$$\boldsymbol{n} = \frac{(q_1, q_2, q_3)^T}{\sin\frac{\theta}{2}} \tag{5}$$

同理，绕 $\boldsymbol{n} = (n_1, n_2, n_3)^T$ 旋转 θ 弧度，可以表示成四元数:

$$\boldsymbol{q} = \cos\frac{\theta}{2} + in_1\sin\frac{\theta}{2} + jn_2\sin\frac{\theta}{2} + kn_3\sin\frac{\theta}{2} \tag{6}$$

在本课程中，我们只研究平面运动，平面上逆时针转动 θ 弧度，可以看作是绕 $(0,0,1)^T$ 转动 θ 弧度，或是表示成欧拉角 $(0,0,\theta)^T$，对应四元数

$$\boldsymbol{q} = \cos\frac{\theta}{2} + 0i + 0j + k\sin\frac{\theta}{2} \tag{7}$$

2.3 tf 的简单应用

这一小节，我们将应用 tf 功能包，完成小乌龟的跟踪任务。即，用键盘操作 turtle1，turtle2 则根据它与 turtle1 的相对位置自动跟踪。

继续使用上节课的 catkin workspace。首先创建一个新功能包

```
$ cd ~/catkin_ws
$ source ./devel/setup.bash
$ cd ./src
$ catkin_create_pkg learning_tf tf rospy turtlesim
$ cd ..
$ catkin_make
$ source ./devel/setup.bash
```

2.3.1 创建 tf 广播者（broadcaster）

在功能包中创建并编辑节点文件

```
$ roscd learning_tf
$ mkdir nodes
$ touch ./nodes/turtle_tf_broadcaster.py
$ chmod +x ./nodes/turtle_tf_broadcaster.py
```

编辑 nodes/turtle_tf_broadcaster.py 如下，例程中也可以找到：

```
#!/usr/bin/env python
import rospy
import tf
import turtlesim.msg

def handle_turtle_pose(msg, turtlename):
    br = tf.TransformBroadcaster()
    br.sendTransform((msg.x, msg.y, 0),
                     tf.transformations.quaternion_from_euler(0, 0, msg.theta),
                     rospy.Time.now(),
                     turtlename,
                     "world")

if __name__ == '__main__':
    rospy.init_node('turtle_tf_broadcaster')
    turtlename = rospy.get_param('~turtle')
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

代码解释：

```
16      turtlename = rospy.get_param('~turtle')
```

这行代码要求启动节点时给出参数 turtle 的取值。根据取值不同，该广播者将上传 turtle1-world 的相对位姿，或是 turtle2-world 的相对位姿。

```
17      rospy.Subscriber('/%s/pose' % turtlename,
18                       turtlesim.msg.Pose,
19                       handle_turtle_pose,
20                       turtlename)
```

这段代码是订阅 turtle 的位置 topic，获取 message 后调用回调函数 handle_turtle_pose。

```
7       br = tf.TransformBroadcaster()
8       br.sendTransform((msg.x, msg.y, 0),
9                        tf.transformations.quaternion_from_euler(0, 0, msg.theta),
10                       rospy.Time.now(),
11                       turtlename,
12                       "world")
```

这段代码是回调函数的内容。第 7 行创建了一个广播者对象，第 8 行将该 turtle 相对于 world 坐标系的位姿广播给 tf。其中第 8 行表示 turtle 坐标系基点在 world 坐标系的坐标，第 9 行是 turtle 坐标系相对于 world 坐标系的四元数。这里调用函数将欧拉角变为四元数。第 10 行是时间戳，第 11 行是子坐标系，第 12 行是父坐标系，即参考坐标系。

2.3.2 创建 tf 收听者（listener）

创建并编辑节点文件：

```
$ roscd learning_tf
$ mkdir nodes
$ touch ./nodes/turtle_tf_listener.py
$ chmod +x ./nodes/turtle_tf_listener.py
```

编辑./nodes/turtle_tf_listener.py 文件如下，例程中也可以找到：

```
1   #!/usr/bin/env python
2   import rospy
3   import math
4   import tf
5   import geometry_msgs.msg
6   import turtlesim.srv
7
8   if __name__ == '__main__':
9       rospy.init_node('turtle_tf_listener')
10
11      listener = tf.TransformListener()
12
13      rospy.wait_for_service('spawn')
```

```
14      spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
15      spawner(4, 2, 0, 'turtle2')
16
17      turtle_vel = rospy.Publisher('turtle2/cmd_vel', \
18          geometry_msgs.msg.Twist,queue_size=1)
19
20      rate = rospy.Rate(10.0)
21      listener.waitForTransform("/turtle2", "/turtle1", \
22          rospy.Time(), rospy.Duration(1.0))
23      while not rospy.is_shutdown():
24          try:
25              now = rospy.Time.now()
26              past = now - rospy.Duration(5.0)
27              listener.waitForTransformFull("/turtle2", now,
28                                            "/turtle1", now,
29                                            "/world", rospy.Duration(1.0))
30              (trans, rot) = listener.lookupTransformFull("/turtle2", now,
31                                            "/turtle1", past,
32                                            "/world")
33          except (tf.LookupException, tf.ConnectivityException, \
34                  tf.ExtrapolationException):
35              continue
36
37          angular = 4 * math.atan2(trans[1], trans[0])
38          linear = 0.5 * math.sqrt(trans[0] ** 2 + trans[1] ** 2)
39          cmd = geometry_msgs.msg.Twist()
40          cmd.linear.x = linear
41          cmd.angular.z = angular
42          turtle_vel.publish(cmd)
43
44          rate.sleep()
```

代码解释:

```
13      rospy.wait_for_service('spawn')
14      spawner = rospy.ServiceProxy('spawn', turtlesim.srv.Spawn)
15      spawner(4, 2, 0, 'turtle2')
```

这段代码是调用 turtlesim 提供的 spawn 服务，作用是在窗口中孵化一只新乌龟，位置在（4，2），角度为 0，名字是 turtle2。

```
17      turtle_vel = rospy.Publisher('turtle2/cmd_vel', \
18          geometry_msgs.msg.Twist,queue_size=1)
```

这段代码表示该节点作为 topic：turtle2/cmd_vel 的发布者，给 turtle2 控制指令。

```
13      listener.waitForTransform("/turtle2", "/turtle1", \
14          rospy.Time(), rospy.Duration(1.0))
```

这段代码是等待 tf 收到 turtle2 和 turtle1 位置的广播，确保 tf 的坐标系关系树上已经建立了这两个坐标系。

```
13      now = rospy.Time.now()
14      past = now - rospy.Duration(5.0)
15      listener.waitForTransformFull("/turtle2", now,
16                                    "/turtle1", now,
17                                    "/world", rospy.Duration(1.0))
```

这段代码是等待 tf 收到 now 时刻的坐标系位置关系的广播，确保 now 时刻 turtle1 和 turtle2 的相对位姿是可查询的。

```
13      (trans, rot) = listener.lookupTransformFull("/turtle2", now,
14      "/turtle1", past,
15      "/world")
```

这行代码向 tf 查询了 turtle2（此刻）相对于 turtle1（5 秒前）相对位姿。

```
13      angular = 4 * math.atan2(trans[1], trans[0])
14      linear = 0.5 * math.sqrt(trans[0] ** 2 + trans[1] ** 2)
15      cmd = geometry_msgs.msg.Twist()
16      cmd.linear.x = linear
17      cmd.angular.z = angular
18      turtle_vel.publish(cmd)
```

这段代码发送控制指令，要求 turtle2 跟踪 turtle1 的 5 秒前的位置。

2.3.3 编写 launch 文件并测试

创建并编辑 launch 文件：

```
$ roscd learning_tf
$ mkdir launch
$ touch ./launch/start_demo.launch
$ vim ./launch/start_demo.launch
```

编辑./launch/start_demo.launch 文件如下，例程中也可以找到：

```
<launch>
  <!-- Turtlesim Node-->
  <node pkg="turtlesim" type="turtlesim_node" name="sim"/>
  <node pkg="turtlesim" type="turtle_teleop_key" name="teleop" output="screen"/>
  <node name="turtle1_tf_broadcaster" pkg="learning_tf"
      type="turtle_tf_broadcaster.py" respawn="false" output="screen">
    <param name="turtle" type="string" value="turtle1" />
  </node>
  <node name="turtle2_tf_broadcaster" pkg="learning_tf"
      type="turtle_tf_broadcaster.py" respawn="false" output="screen">
    <param name="turtle" type="string" value="turtle2" />
  </node>
```

```
    <node pkg="learning_tf" type="turtle_tf_listener.py"
          name="listener" />
  </launch>
```

在终端中运行

```
$ roslaunch learning_tf start_demo.launch
```

即可用键盘操纵 turtle1，或是直接使用 rostopic 发布控制命令. 可以注意到 turtle2 会自动跟踪 turtle1 在 5 秒前的位置。

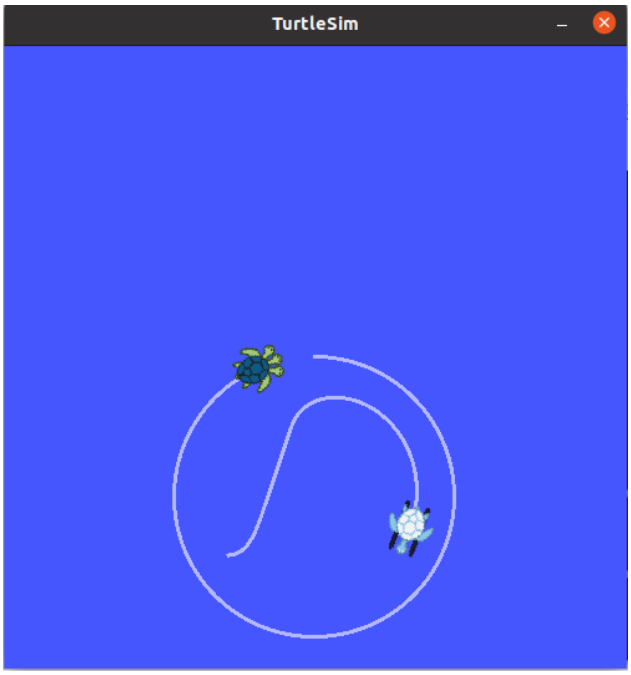


Figure 5: ROS tf 例程运行结果

3 ROS gazebo 搭建介绍

3.1 使用 gazebo 搭建模型

3.1.1 操作步骤

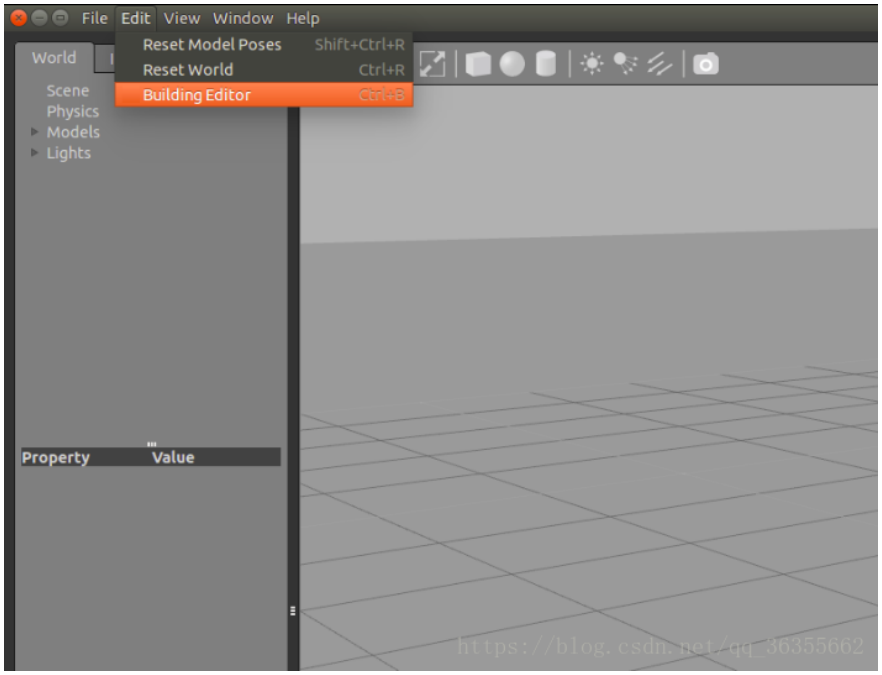
一、运行 roscore

```
$ roscore
```

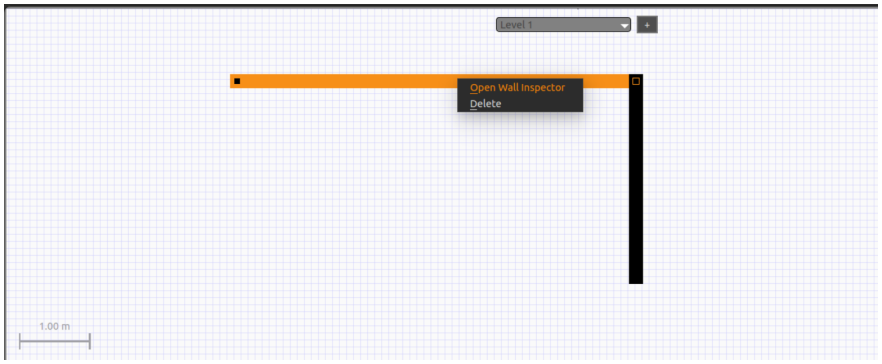
二、运行

```
$ rosrun gazebo_ros gazebo
```

进入 gazebo 界面
三、按下 ctrl + B 进入编辑模型的界面，也可以从界面内 Edit->Building Editor 进入

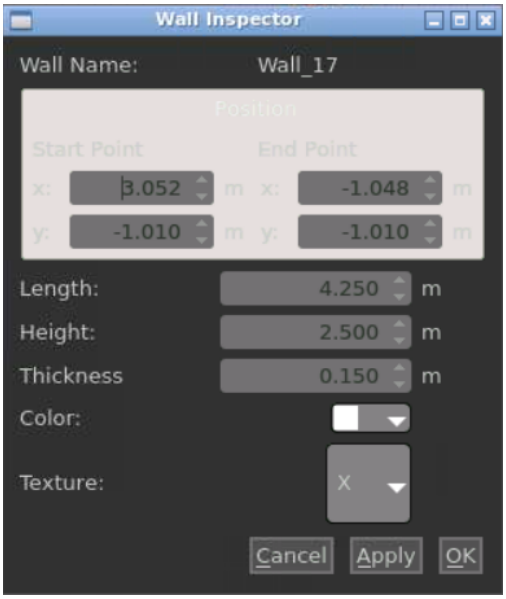


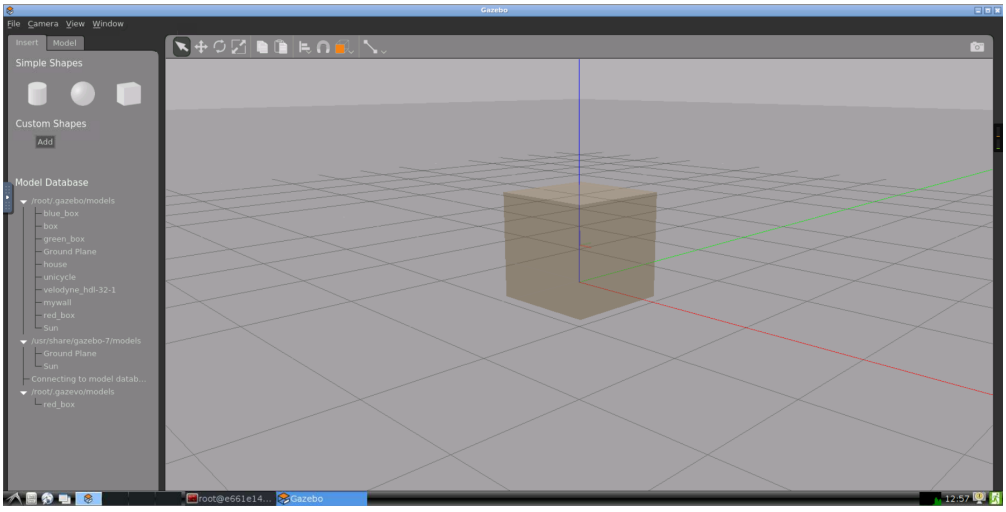
四、进入编辑模型的界面后，可以搭建墙壁、门等室内环境，也可以利用圆柱体、长方体等物体搭建机器人模型。



例如，点击 wall 搭建墙壁。右键单击搭好的墙壁，可以打开 wall inspector 编辑器，用来编辑 wall 长度等信息。
又例如，在模型编辑页面中建立长方体物块模型。

在左侧界面 model->links 中，双击 link_0 可以展开对 link 属性的编辑。如果我们想对其颜色更改，可以在选项 visual 中更改 Ambient、Diffuse、Specular、Emissive 四个选项。例如我想要一个红色的物块，我可以将 Ambient、Diffuse、Specular 中的红色设为 1，其余设为 0。
五、模型搭建好后，点击左上角 file->save as 保存模型。模型保存路径为：/root/.gazebo/models。(按 ctrl + H 显示隐藏文件)
六、退出 model editor，在 gazebo 界面中左侧可以查看并调整模型参数。





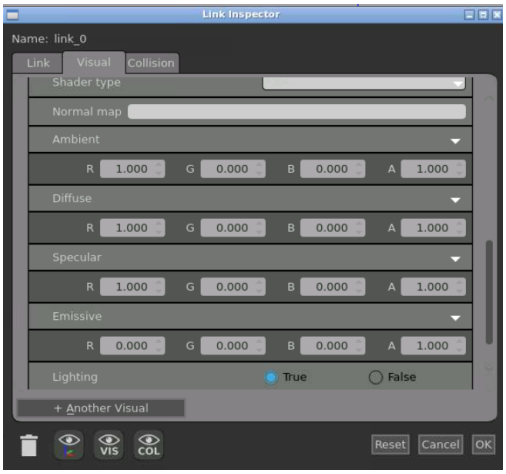
3.1.2 模型 SDF 文件注释

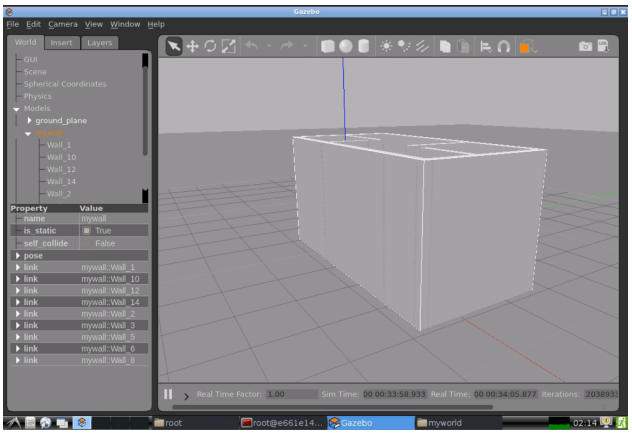
在/root/.gazebo/models/your_model_name 中可以看到 your_model_name.config 和 your_model_name.sdf 两个文件。其中.config 文件中为模型的元数据，而 model.sdf 中为模型的 SDF 描述。
在该路径下打开终端，输入

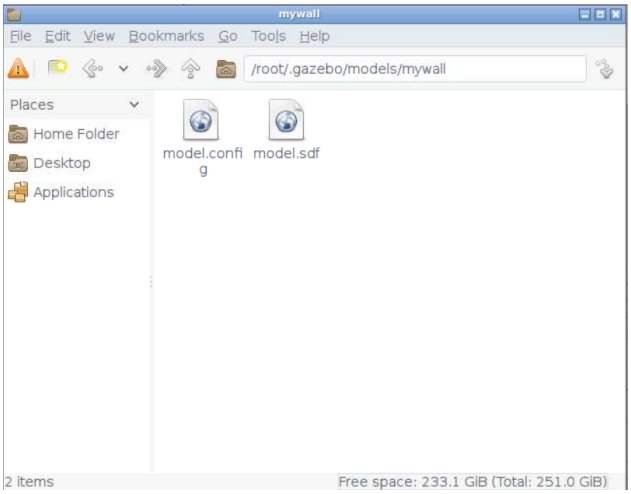
```
$ gedit your_model_name.sdf
```

可以查看或修改之前保存的模型
以我搭建的 mywall 文件为例，如下图：
文件的语句注释如下，可以在 gazebo 界面中修改 model 属性，也可以在文件中修改相应代码。

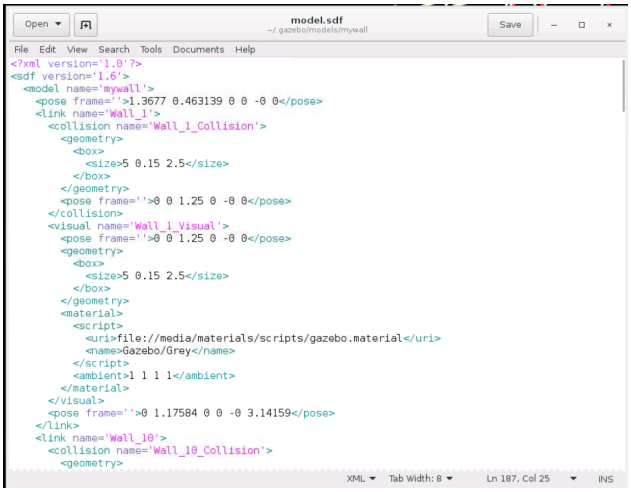
```
<xml version> #xml的版本
<sdf version> #sdf的版本
<model name> #模型名称
```








```
<pose frame> #模型在世界中的位置 [x y z pitch yaw roll]
<static> #模型是否固定
<link> #链接，包含模型的一个主体的物理属性
<collision> #用于碰撞检查
```




```
<box>(<sphere>,<cylinder>) #形状名字
<size>(<radius>) #x,y,z长度(半径)
<surface> #平面
<friction> #设置地面摩擦力
<visual>: #描述模型外观
<geometry> #描述几何形状
<inertial> #描述惯性元素
<mass> #描述模型质量
<sensor> #从world(环境) 收集数据用于plugin
<light> #描述光源
<joint> #描述关节, 关节连接两个link, 用于旋转轴和关节限制等
<plugin> #用于控制模型的插件
```

3.2 使用 gazebo 搭建仿真环境

3.2.1 操作步骤

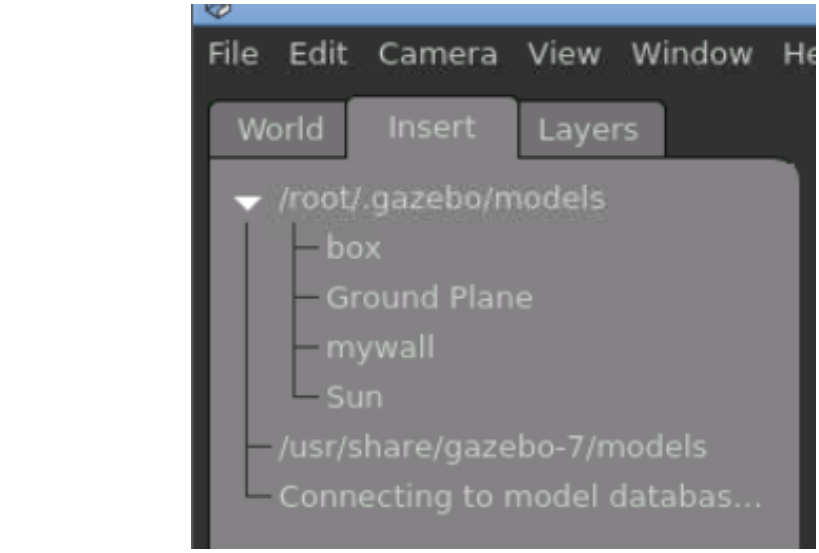
一、运行

```
$ roscore
```

二、运行

```
$ rosrun gazebo_ros gazebo
```

进入 gazebo 界面
三、点击左上角 insert，可以看到我们之前搭建好的模型。
点击模型，拖动到环境中即可。
　　上方工具栏中的按键分别为选择模式、移动模式、旋转模式、放缩模式、撤销、重做、长方体、球体、圆柱体、点光源、投影光源、线光源、复制、粘贴、多选、部分选择、切换视角。
四、搭建完成后，在界面左上角点击 File->save world as，选择路径保存，文件后缀名为.world



3.2.2 环境 world 文件注释

注：模型文件与环境文件均采用 sdf 语句，world 文件可以看作多个模型 sdf 文件的集合。

```
<xml version> #xml的版本
<sdf version> #sdf的版本
<model name> #模型名称
<pose frame> #模型在世界中的位置 [x y z pitch yaw roll]
<static> #模型是否固定
<link> #链接，包含模型的一个主体的物理属性
<collision> #用于碰撞检查
<box>(<sphere>,<cylinder>) #形状名字
<size>(<radius>) #x,y,z长度(半径)
<surface> #平面
<friction> #设置地面摩擦力
<visual>: #描述模型外观
<geometry> #描述几何形状
<inertial> #描述惯性元素
<mass> #描述模型质量
<sensor> #从world(环境) 收集数据用于plugin
<light> #描述光源
<joint> #描述关节，关节连接两个link，用于旋转轴和关节限制等
<plugin> #用于控制模型的插件
```

3.2.3 动态障碍物

由于动态障碍物依赖 gazebo8+ 的版本，课程中使用的 gazebo7 无法定义动态障碍物。

3.3 利用 urdf 文件搭建机器人模型

3.3.1 urdf 文件简介

urdf 全称为: Unified Robot Description Format 即: 统一机器人描述性格式。使用一个 urdf 文件可以描述一个机器人模型。

3.3.2 urdf 文件语句规范

```
#一个完整的urdf模型，由一系列<link>和<joint>组成
<link> #描述机器人某个刚体部分的外观和物理属性
  <inertial> #描述link的惯性参数
  <geometry> #描述link的形状和大小
  <collision> #描述link的碰撞属性
  <material> #指定颜色与材料
<joint> #描述机器人关节的动力学与运动学轨迹，
#连接两个link，并且有主次（父、子关节）
  <calibration> #描述关节的绝对位置
  <dynamics> #描述关节的物理属性，如阻尼值、物理静摩擦等
  <limit> #描述运动的极限值，包括运动限制，速度，力矩限制等
  <mimic> #描述该关节与已有关节的关系
  <safety_controller> #描述安全控制器参数
```

3.3.3 编写 urdf 文件搭建机器人

一、准备工作：初始化工作空间

```
$ mkdir -p catkin_ws/src
$ cd catkin_ws/src
$ catkin_init_workspace
$ cd ..
$ catkin_make
$ source ./devel/setup.bash
```

二、创建存放小车模型的功能包

```
$ cd src
$ catkin_create_pkg smartcar std_msgs rospy roscpp urdf gazebo_plugins gazebo_ros

xacro
$ cd smartcar
$ mkdir urdf
$ mkdir launch
$ cd urdf
$ touch myrobot.urdf
```

三、在 urdf 文件中输入如下内容：

```
<?xml version="1.0"?>
<robot name="smartcar">
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.25 .16 .05"/>
        </geometry>
        <origin rpy="0 0 1.57075" xyz="0 0 0"/>
    <material name="blue">
                <color rgba="0 0 .8 1"/>
        </material>
    </visual>
 </link>

 <link name="right_front_wheel">
    <visual>
      <geometry>
        <cylinder length=".02" radius="0.025"/>
      </geometry>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
  </link>

  <joint name="right_front_wheel_joint" type="continuous">
    <axis xyz="0 0 1"/>
    <parent link="base_link"/>
    <child link="right_front_wheel"/>
    <origin rpy="0 1.57075 0" xyz="0.08 0.1 -0.03"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

  <link name="right_back_wheel">
    <visual>
      <geometry>
        <cylinder length=".02" radius="0.025"/>
      </geometry>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
  </link>
```

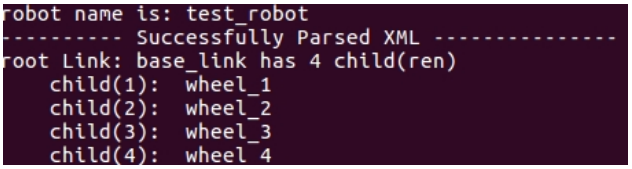
```
 <joint name="right_back_wheel_joint" type="continuous">
   <axis xyz="0 0 1"/>
   <parent link="base_link"/>
   <child link="right_back_wheel"/>
   <origin rpy="0 1.57075 0" xyz="0.08 -0.1 -0.03"/>
   <limit effort="100" velocity="100"/>
   <joint_properties damping="0.0" friction="0.0"/>
</joint>

<link name="left_front_wheel">
   <visual>
     <geometry>
       <cylinder length=".02" radius="0.025"/>
     </geometry>
     <material name="black">
       <color rgba="0 0 0 1"/>
     </material>
   </visual>
 </link>

 <joint name="left_front_wheel_joint" type="continuous">
   <axis xyz="0 0 1"/>
   <parent link="base_link"/>
   <child link="left_front_wheel"/>
   <origin rpy="0 1.57075 0" xyz="-0.08 0.1 -0.03"/>
   <limit effort="100" velocity="100"/>
   <joint_properties damping="0.0" friction="0.0"/>
 </joint>

 <link name="left_back_wheel">
   <visual>
     <geometry>
       <cylinder length=".02" radius="0.025"/>
     </geometry>
     <material name="black">
       <color rgba="0 0 0 1"/>
     </material>
   </visual>
 </link>

 <joint name="left_back_wheel_joint" type="continuous">
   <axis xyz="0 0 1"/>
   <parent link="base_link"/>
   <child link="left_back_wheel"/>
```

```
    <origin rpy="0 1.57075 0" xyz="-0.08 -0.1 -0.03"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

  <link name="head">
    <visual>
      <geometry>
        <box size=".02 .03 .03"/>
      </geometry>
          <material name="white">
                  <color rgba="1 1 1 1"/>
          </material>
    </visual>
  </link>

  <joint name="tobox" type="fixed">
    <parent link="base_link"/>
    <child link="head"/>
    <origin xyz="0 0.08 0.025"/>
  </joint>
</robot>
```

可以运行

```
$ check_urdf myrobot.urdf
```

查看文件书写是否正确。
结果如下所示，可见书写正确。



3.3.4 编写 launch 文件查看机器人

在 launch 文件夹下创建 mycar.launch 文件并输入以下代码：

```
$ cd ..
$ cd launch
$ touch mycar.launch
```

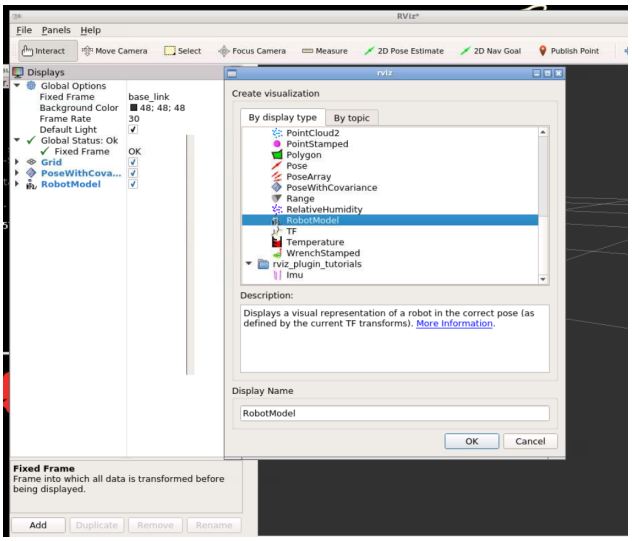
```
<launch>
        <arg name="model" />
        <arg name="gui" default="False" />
```

```
        <param name="robot_description" textfile=
        "$(find smartcar)/urdf/myrobot.urdf" />
        <param name="use_gui" value="$(arg gui)"/>
        <node name="joint_state_publisher" pkg="joint_state_publisher"
        type="joint_state_publisher" ></node>
        <node name="robot_state_publisher" pkg="robot_state_publisher"
        type="state_publisher" />
        <node name="rviz" pkg="rviz" type="rviz"/>
</launch>
```

在运行 launch 文件前，要先安装所需依赖，然后运行之前编写的 launch 文件。

```
$ sudo apt-get install ros-kinetc-joint-state-publisher
$ roslaunch smartcar mycar.launch
```

将 rviz 中 global options 内的 fixed frame 改为 base_link，点击左下角 add，选择 by display type —>robotmodel，可以看到我们利用 urdf 文件搭建好的小车模型。



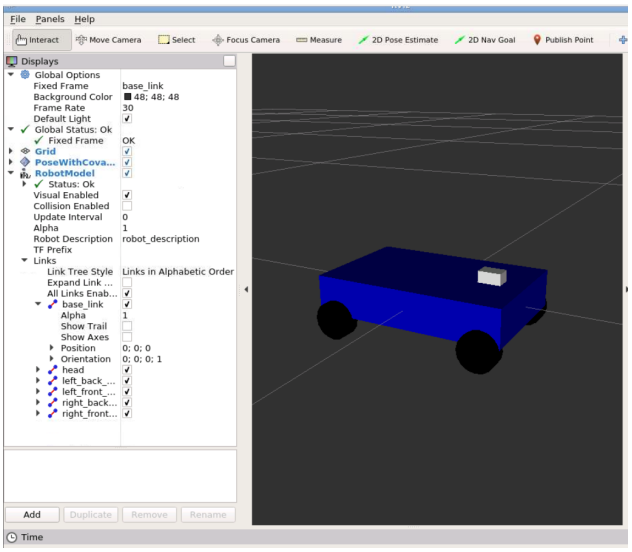
3.4 编写 launch 文件打开环境

与编写 launch 文件打开机器人模型相似，首先需要建立工作空间，我在之前建立好的 smart 包中添加 worlds 文件夹用于存放之前建立好的环境。

之前我们使用的都是 urdf 文件格式的模型，在很多情况下，ROS 对 urdf 文件的支持并不是很好，使用宏定义的.xacro 文件兼容性更好，扩展性也更好。所以我们把之前的 urdf 文件重新整理编写成.xacro 文件。

在 urdf 文件夹下新建 robot_body.urdf.xacro 文件，内容为

```
<?xml version="1.0"?>
<robot name="smartcar" xmlns:xacro="http://ros.org/wiki/xacro">
  <property name="M_PI" value="3.14159"/>
```




```
<!-- Macro for SmartCar body. Including Gazebo extensions,
but does not include Kinect -->
<include filename="$(find smartcar)/urdf/gazebo.urdf.xacro"/>

<property name="base_x" value="0.33" />
<property name="base_y" value="0.33" />

<xacro:macro name="smartcar_body">


      <link name="base_link">
      <inertial>
    <origin xyz="0 0 0.055"/>
    <mass value="1.0" />
    <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
    izz="1.0"/>
  </inertial>
  <visual>
    <geometry>
      <box size="0.25 .16 .05"/>
    </geometry>
        <origin rpy="0 0 0" xyz="0 0 0.055"/>
    <material name="blue">
        <color rgba="0 0 .8 1"/>
    </material>
 </visual>
 <collision>
    <origin rpy="0 0 0" xyz="0 0 0.055"/>
```

```
      <geometry>
        <box size="0.25 .16 .05" />
      </geometry>
    </collision>
  </link>


<link name="left_front_wheel">
       <inertial>
      <origin  xyz="0.08 0.08 0.025"/>
      <mass value="0.1" />
       <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
       izz="1.0"/>
    </inertial>
    <visual>
      <geometry>
        <cylinder length=".02" radius="0.025"/>
      </geometry>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 1.57075 1.57075" xyz="0.08 0.08 0.025"/>
      <geometry>
         <cylinder length=".02" radius="0.025"/>
      </geometry>
    </collision>
  </link>

  <joint name="left_front_wheel_joint" type="continuous">
    <axis xyz="0 0 1"/>
    <parent link="base_link"/>
    <child link="left_front_wheel"/>
    <origin rpy="0 1.57075 1.57075" xyz="0.08 0.08 0.025"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

  <link name="right_front_wheel">
       <inertial>
      <origin xyz="0.08 -0.08 0.025"/>
      <mass value="0.1" />
       <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
       izz="1.0"/>
```

```
    </inertial>
    <visual>
      <geometry>
        <cylinder length=".02" radius="0.025"/>
      </geometry>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
    </visual>
    <collision>
      <origin rpy="0 1.57075 1.57075" xyz="0.08 -0.08 0.025"/>
      <geometry>
         <cylinder length=".02" radius="0.025"/>
      </geometry>
    </collision>
  </link>

  <joint name="right_front_wheel_joint" type="continuous">
    <axis xyz="0 0 1"/>
    <parent link="base_link"/>
    <child link="right_front_wheel"/>
    <origin rpy="0 1.57075 1.57075" xyz="0.08 -0.08 0.025"/>
    <limit effort="100" velocity="100"/>
    <joint_properties damping="0.0" friction="0.0"/>
  </joint>

  <link name="left_back_wheel">
    <inertial>
      <origin xyz="-0.08 0.08 0.025"/>
      <mass value="0.1" />
       <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
       izz="1.0"/>
    </inertial>
    <visual>
      <geometry>
        <cylinder length=".02" radius="0.025"/>
      </geometry>
      <material name="black">
        <color rgba="0 0 0 1"/>
      </material>
   </visual>
   <collision>
       <origin rpy="0 1.57075 1.57075" xyz="-0.08 0.08 0.025"/>
      <geometry>
         <cylinder length=".02" radius="0.025"/>
```

```
    </geometry>
  </collision>
</link>

<joint name="left_back_wheel_joint" type="continuous">
  <axis xyz="0 0 1"/>
  <parent link="base_link"/>
  <child link="left_back_wheel"/>
  <origin rpy="0 1.57075 1.57075" xyz="-0.08 0.08 0.025"/>
  <limit effort="100" velocity="100"/>
  <joint_properties damping="0.0" friction="0.0"/>
</joint>

<link name="right_back_wheel">
      <inertial>
     <origin xyz="-0.08 -0.08 0.025"/>
     <mass value="0.1" />
     <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
     izz="1.0"/>
  </inertial>
  <visual>
    <geometry>
      <cylinder length=".02" radius="0.025"/>
    </geometry>
    <material name="black">
      <color rgba="0 0 0 1"/>
    </material>
 </visual>
 <collision>
    <origin rpy="0 1.57075 1.57075" xyz="-0.08 -0.08 0.025"/>
    <geometry>
       <cylinder length=".02" radius="0.025"/>
    </geometry>
  </collision>
</link>


<joint name="right_back_wheel_joint" type="continuous">
  <axis xyz="0 0 1"/>
  <parent link="base_link"/>
  <child link="right_back_wheel"/>
  <origin rpy="0 1.57075 1.57075" xyz="-0.08 -0.08 0.025"/>
  <limit effort="100" velocity="100"/>
  <joint_properties damping="0.0" friction="0.0"/>
</joint>
```

```
  <link name="head">
        <inertial>
      <origin xyz="0.08 0 0.08"/>
      <mass value="0.1" />
      <inertia ixx="1.0" ixy="0.0" ixz="0.0" iyy="1.0" iyz="0.0"
      izz="1.0"/>
    </inertial>
    <visual>
      <geometry>
        <box size=".02 .03 .03"/>
      </geometry>
          <material name="white">
                <color rgba="1 1 1 1"/>
          </material>
     </visual>
     <collision>
      <origin xyz="0.08 0 0.08"/>
      <geometry>
         <cylinder length=".02" radius="0.025"/>
      </geometry>
    </collision>
  </link>

  <joint name="tobox" type="fixed">
    <parent link="base_link"/>
    <child link="head"/>
    <origin xyz="0.08 0 0.08"/>
  </joint>
  </xacro:macro>

</robot>
```

新建文件 gazebo.urdf.xacro, 写入如下内容:

```
<?xml version="1.0"?>

<robot xmlns:controller="http://playerstage.sourceforge.net/
    gazebo/xmlschema/#controller"
        xmlns:interface="http://playerstage.sourceforge.net/
        gazebo/xmlschema/#interface"
        xmlns:sensor="http://playerstage.sourceforge.net/
        gazebo/xmlschema/#sensor"
        xmlns:xacro="http://ros.org/wiki/xacro"
        name="smartcar_gazebo">
```

```
<!-- ASUS Xtion PRO camera for simulation -->
<!-- gazebo_ros_wge100 plugin is in kt2_gazebo_plugins package -->
<xacro:macro name="smartcar_sim">
    <gazebo reference="base_link">
        <material>Gazebo/Blue</material>
        <turnGravityOff>false</turnGravityOff>
    </gazebo>

    <gazebo reference="right_front_wheel">
        <material>Gazebo/FlatBlack</material>
        </gazebo>

        <gazebo reference="right_back_wheel">
        <material>Gazebo/FlatBlack</material>
    </gazebo>

    <gazebo reference="left_front_wheel">
        <material>Gazebo/FlatBlack</material>
    </gazebo>

    <gazebo reference="left_back_wheel">
        <material>Gazebo/FlatBlack</material>
    </gazebo>

    <gazebo reference="head">
        <material>Gazebo/White</material>
    </gazebo>

</xacro:macro>

</robot>
```

新建 myrobot.urdf.xacro 文件，写入如下内容：

```
<?xml version="1.0"?>

<robot name="smartcar"
    xmlns:xi="http://www.w3.org/2001/XInclude"
    xmlns:gazebo="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#gz"
    xmlns:model="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#model"
    xmlns:sensor="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#sensor"
    xmlns:body="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#body"
```

```
    xmlns:geom="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#geom"
    xmlns:joint="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#joint"
    xmlns:controller="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#controller"
    xmlns:interface="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#interface"
    xmlns:rendering="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#rendering"
    xmlns:renderable="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#renderable"
    xmlns:physics="http://playerstage.sourceforge.net/gazebo/
    xmlschema/#physics"
    xmlns:xacro="http://ros.org/wiki/xacro">

  <include filename="$(find smartcar)/urdf/robot_body.urdf.xacro" />

  <!-- Body of SmartCar, with plates, standoffs and Create -->
  <smartcar_body/>

  <smartcar_sim/>

</robot>
```

在 launch 文件夹下，新建 runworld.launch 文件。编写 launch 文件在启动 gazebo 时启动我们所搭建的环境。launch 文件如下所示：

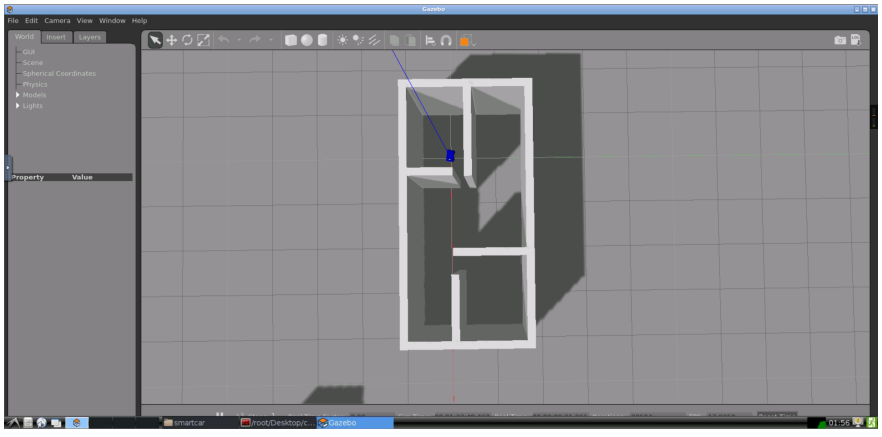
```
<launch>
    <!-- 设置launch文件的参数 -->
    <arg name="world_name" value="$(find smartcar)/worlds/
    testworld.world"/>
    <arg name="paused" default="false"/>
    <arg name="use_sim_time" default="true"/>
    <arg name="gui" default="true"/>
    <arg name="headless" default="false"/>
    <arg name="debug" default="false"/>
    <arg name="urdf_file" default="$(find xacro)/xacro.py
    '$(find smartcar)/urdf/myrobot.urdf.xacro'" />
    <!-- 运行gazebo仿真环境 -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find smartcar)/worlds/
        testworld.world" />
        <arg name="debug" value="$(arg debug)" />
        <arg name="gui" value="$(arg gui)" />
        <arg name="paused" value="$(arg paused)"/>
```

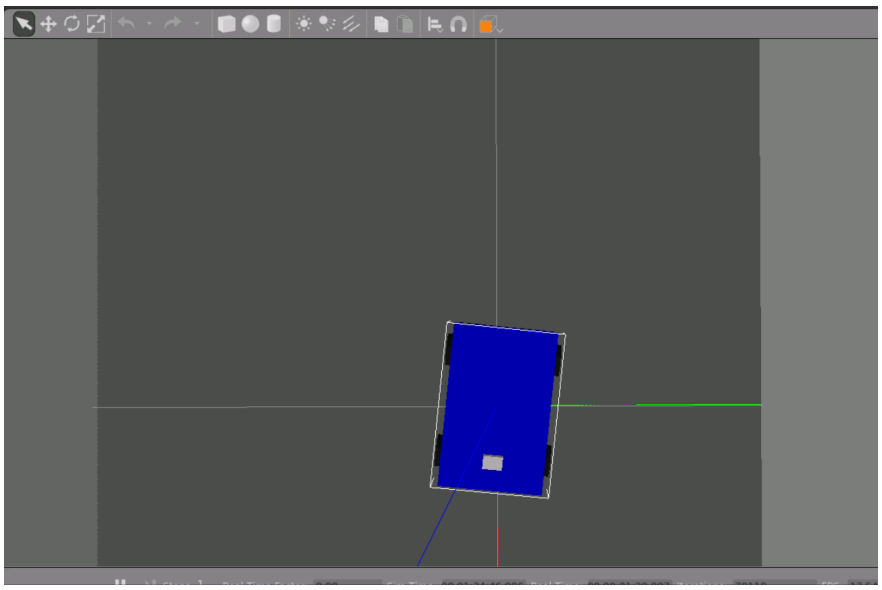
```
        <arg name="use_sim_time" value="$(arg use_sim_time)"/>
        <arg name="headless" value="$(arg headless)"/>
    </include>

    <!-- 加载机器人模型描述参数 -->
    <param name="robot_description" command="$(arg urdf_file)" />

    <!-- 运行joint_state_publisher节点，发布机器人的关节状态  -->
    <node name="joint_state_publisher" pkg="joint_state_publisher"
    type="joint_state_publisher" ></node>

    <!-- 运行robot_state_publisher节点，发布tf  -->
    <node name="robot_state_publisher" pkg="robot_state_publisher"
    type="robot_state_publisher"  output="screen" >
        <param name="publish_frequency" type="double" value="50.0" />
    </node>

    <!-- 在gazebo中加载机器人模型-->
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
    respawn="false" output="screen"
         args="-urdf -model mrobot -param robot_description"/>

</launch>
</launch>
```

运行

```
$ roslaunch smartcar runworld.launch
```

即可用 launch 文件运行自己搭建的 gazebo 环境。





4 下周实验室作业

1. 在 Gazebo 中搭建一个矩形房间；
2. 在 Gazebo 中实现圆柱体的移动，能用 rviz 中的 teleop_panel 面板实现圆柱体在 Gazeobo 中的运动；
3. 在 Gazebo 中用 KF 实现上述圆柱体的定位
4. 改变该圆柱体的运动学，实现机器人的 unicycle 运动学，并能用 rviz 中的 teleop_panel 面板实现圆柱体在 Gazeobo 中的运动。

5 附录

5.1 圆柱体 SDF 文件

为了后续融合激光雷达与相机共同完成任务，建立此 unicycl 模型 sdf 文件如下：

```
<?xml version="1.0" ?>
<sdf version="1.5">

    <!-- A global light source -->

  <model name="unicycle">
  <!-- Give the base link a unique name -->
  <link name="base">

    <!-- Offset the base by half the lenght of the cylinder -->
    <pose>0 0 0.02 0 0 0</pose>
<sensor type="ray" name="sensor">
```

```
  <!-- Position the ray sensor based on the specification. Also rotate
       it by 90 degrees around the X-axis so that the <horizontal> rays
       become vertical -->
  <pose>0 0 -0.004645 0 0 0</pose>

  <!-- Enable visualization to see the rays in the GUI -->
  <visualize>true</visualize>

  <!-- Set the update rate of the sensor -->
  <update_rate>30</update_rate>
<ray>
  <!-- The scan element contains the horizontal and vertical beams.
       We are leaving out the vertical beams for this tutorial. -->
  <scan>

    <!-- The horizontal beams -->
    <horizontal>
      <!-- The velodyne has 32 beams(samples) -->
      <samples>2</samples>

      <!-- Resolution is multiplied by samples to determine number of
           simulated beams vs interpolated beams. See:
           http://sdformat.org/spec?ver=1.6&elem=sensor#horizontal_resolution
           -->
      <resolution>1</resolution>

      <!-- Minimum angle in radians -->
      <min_angle>0</min_angle>

      <!-- Maximum angle in radians -->
      <max_angle>1.5707</max_angle>
    </horizontal>
  </scan>

  <!-- Range defines characteristics of an individual beam -->
  <range>

    <!-- Minimum distance of the beam -->
    <min>0.05</min>

    <!-- Maximum distance of the beam -->
    <max>70</max>

    <!-- Linear resolution of the beam -->
```

```
    <resolution>0.02</resolution>
  </range>
</ray>
  <plugin name="gazebo_ros_velodyne_hdl-32-1_base_controller" filename="libgazebo_r
        <topicName>/velodyne/laser/scan_without_noise</topicName>
        <frameName>scan_link_without_noise</frameName>
  </plugin>
</sensor>
     <inertial>
      <mass>1.2</mass>
      <inertia>
        <ixx>0.001087473</ixx>
        <iyy>0.001087473</iyy>
        <izz>0.001092437</izz>
        <ixy>0</ixy>
        <ixz>0</ixz>
        <iyz>0</iyz>
      </inertia>
    </inertial>
    <collision name="base_collision">
      <geometry>
        <cylinder>
          <!-- Radius and length provided by Velodyne -->
          <radius>.04</radius>
          <length>.04</length>
        </cylinder>
      </geometry>
    </collision>

    <!-- The visual is mostly a copy of the collision -->
    <visual name="base_visual">
      <geometry>
        <cylinder>
          <radius>.04</radius>
          <length>.04</length>
        </cylinder>
      </geometry>
    </visual>
  </link>

  <!-- Give the base link a unique name -->
  <link name="top">

    <!-- Vertically offset the top cylinder by the length of the bottom
        cylinder and half the length of this cylinder. -->
```

```
    <pose>0 0 0.06 0 0 0</pose>
<!-- Add a ray sensor, and give it a name -->
<sensor type="ray" name="sensor">

  <!-- Position the ray sensor based on the specification. Also rotate
       it by 90 degrees around the X-axis so that the <horizontal> rays
       become vertical -->
  <pose>0 0 -0.004645 0 0 0</pose>

  <!-- Enable visualization to see the rays in the GUI -->
  <visualize>true</visualize>

  <!-- Set the update rate of the sensor -->
  <update_rate>30</update_rate>
<ray>
    <noise>
      <!-- Use gaussian noise -->
      <type>gaussian</type>
      <mean>0.0</mean>
      <stddev>0.002</stddev>
    </noise>
  <!-- The scan element contains the horizontal and vertical beams.
       We are leaving out the vertical beams for this tutorial. -->
  <scan>

    <!-- The horizontal beams -->
    <horizontal>
      <!-- The velodyne has 32 beams(samples) -->
      <samples>2</samples>

      <!-- Resolution is multiplied by samples to determine number of
           simulated beams vs interpolated beams. See:
           http://sdformat.org/spec?ver=1.6&elem=sensor#horizontal_resolution
           -->
      <resolution>1</resolution>

      <!-- Minimum angle in radians -->
      <min_angle>0</min_angle>

      <!-- Maximum angle in radians -->
      <max_angle>1.5707</max_angle>
    </horizontal>
  </scan>

  <!-- Range defines characteristics of an individual beam -->
```

```
  <range>

    <!-- Minimum distance of the beam -->
    <min>0.05</min>

    <!-- Maximum distance of the beam -->
    <max>70</max>

    <!-- Linear resolution of the beam -->
    <resolution>0.02</resolution>
  </range>
</ray>
  <plugin name="gazebo_ros_velodyne_hdl-32-1controller" filename="libgazebo_ros_las
        <topicName>/velodyne/laser/scan</topicName>
        <frameName>scan_link</frameName>
  </plugin>
</sensor>

    <inertial>
     <mass>0.1</mass>
     <inertia>
       <ixx>0.000090623</ixx>
       <iyy>0.000090623</iyy>
       <izz>0.000091036</izz>
       <ixy>0</ixy>
       <ixz>0</ixz>
       <iyz>0</iyz>
      </inertia>
     </inertial>
    <collision name="top_collision">
      <geometry>
        <cylinder>
          <!-- Radius and length provided by Velodyne -->
          <radius>0.04</radius>
          <length>0.04</length>
        </cylinder>
      </geometry>
    </collision>

    <!-- The visual is mostly a copy of the collision -->
    <visual name="top_visual">
      <geometry>
        <cylinder>
          <radius>0.04</radius>
          <length>0.04</length>
```

```
        </cylinder>
      </geometry>
    </visual>
  </link>
  <link name="camera_link">

    <pose>0 0 0.085 0 0 0</pose>

<sensor type="camera" name="camera1">
    <pose>0 0 -0.00004645 0 0 0</pose>
    <!--每秒获取的图像次数，但如果物理仿真运行速度快于传感器生成速度，那么它可能会
    <update_rate>30.0</update_rate>
    <camera name="head">
      <!--匹配物理相机硬件上制造商提供的规格数据-->
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <!--gazebo_ros/gazebo_ros_camera.cpp-->
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>unicycle/camera1</cameraName>
      <!--图像topic-->
      <imageTopicName>image_raw</imageTopicName>
                  <!--相机信息topic-->
      <cameraInfoTopicName>camera_info</cameraInfoTopicName>


  <!--图像在tf树中发布的坐标系-->
      <frameName>camera_link</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
</sensor>
```

```
    <inertial>
     <mass>0.1</mass>
     <inertia>
       <ixx>0.000090623</ixx>
       <iyy>0.000090623</iyy>
       <izz>0.000091036</izz>
       <ixy>0</ixy>
       <ixz>0</ixz>
       <iyz>0</iyz>
      </inertia>
     </inertial>
    <collision name="camera_collision">
      <geometry>
        <cylinder>
          <!-- Radius and length provided by camera -->
          <radius>0.01</radius>
          <length>0.01</length>
        </cylinder>
      </geometry>
    </collision>

    <!-- The visual is mostly a copy of the collision -->
    <visual name="camera_visual">
      <geometry>
        <cylinder>
          <radius>0.01</radius>
          <length>0.01</length>
        </cylinder>
      </geometry>
    </visual>
  </link>
  <!-- Each joint must have a unique name -->
<joint name="camera_joint" type="fixed">
  <!-- Position the joint at the bottom of the top link -->
  <pose>0 0 -0.03 0 0 0</pose>

  <!-- Use the base link as the parent of the joint -->
  <parent>top</parent>

  <!-- Use the top link as the child of the joint -->
  <child>camera_link</child>

  <!-- The axis defines the joint's degree of freedom -->
  <axis>
```

```
    <!-- Revolve around the z-axis -->
    <xyz>0 0 1</xyz>

    <!-- Limit refers to the range of motion of the joint -->
    <limit>

      <!-- Use a very large number to indicate a continuous revolution -->
      <lower>-10000000000000000</lower>
      <upper>10000000000000000</upper>
    </limit>
  </axis>
</joint>
<joint type="revolute" name="joint">

  <!-- Position the joint at the bottom of the top link -->
  <pose>0 0 -0.036785 0 0 0</pose>

  <!-- Use the base link as the parent of the joint -->
  <parent>base</parent>

  <!-- Use the top link as the child of the joint -->
  <child>top</child>

  <!-- The axis defines the joint's degree of freedom -->
  <axis>

    <!-- Revolve around the z-axis -->
    <xyz>0 0 1</xyz>

    <!-- Limit refers to the range of motion of the joint -->
    <limit>

      <!-- Use a very large number to indicate a continuous revolution -->
      <lower>-10000000000000000</lower>
      <upper>10000000000000000</upper>
    </limit>
  </axis>
</joint>
</model>
</sdf>
```