无人机赛课第2讲
--仿真环境

联系方式: nics-efc@tsinghua.edu.cn

时间：2021年12月23日 Thursday

目 录

目 录

1 仿真界面简介

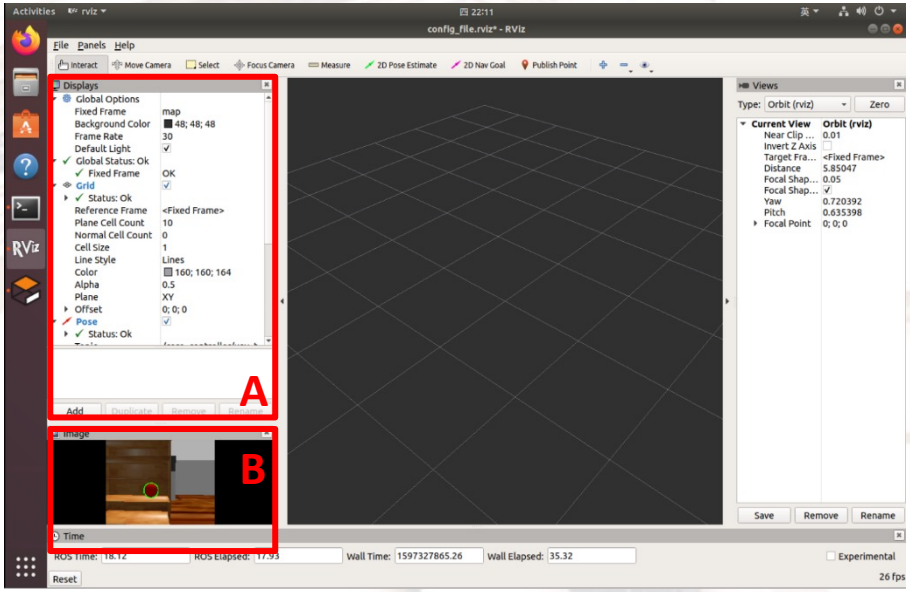
- **环境依赖**
  - 操作系统：Ubuntu 18.04
  - 相关软件：
    - ROS Melodic：机器人操作系统；
    - Gazebo 9.0.0：三维机器人仿真平台，能够模拟室内/室外等各种复杂环境中的机器人；
    - PX4 v1.9.2：自动驾驶仪固件，可用于驱动无人机；
    - MAVLink：消息传输协议, 用于地面控制终端（地面站）与无人机之间进行通信。

1 仿真界面简介

- **认识仿真界面**
  - 配置好仿真环境后，可以运行如下命令，先认识仿真界面：

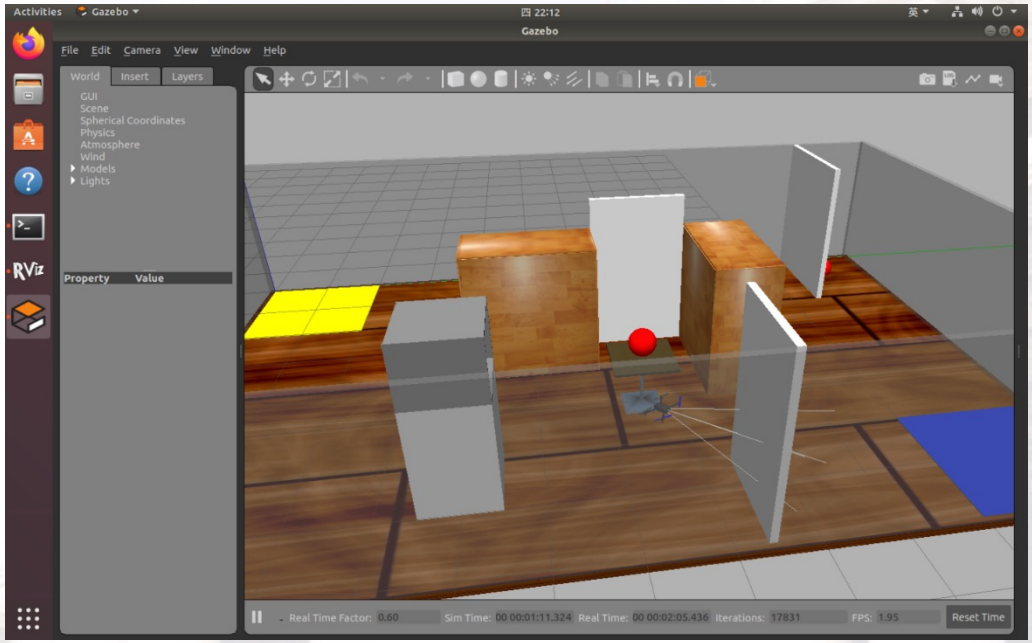
```
roslaunch uav_sim arena_test_py.launch
```

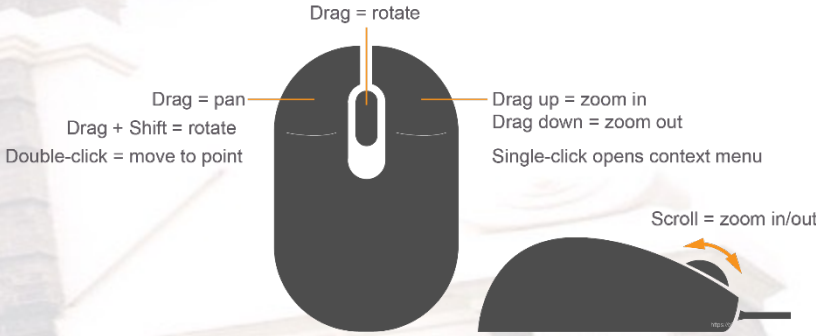
  - 该命令会启动Rviz和Gazebo两个窗口。
  - Rviz为ROS自带的图形化工具，下图中 A 区域为订阅的一些topic信息，B区域为image类型的消息，下图中是对无人机摄像头拍摄图片处理后的图像（比如检测红色的球）。



1 仿真界面简介

- **认识仿真界面**
  - 运行仿真例程后，会启动Rviz和Gazebo两个窗口。
  - Gazebo会显示模拟的仿真场景与无人机飞行状态，可以进行拖动以转换视角。
  - 下图中黄色区域为起飞点，蓝色区域为降落点。

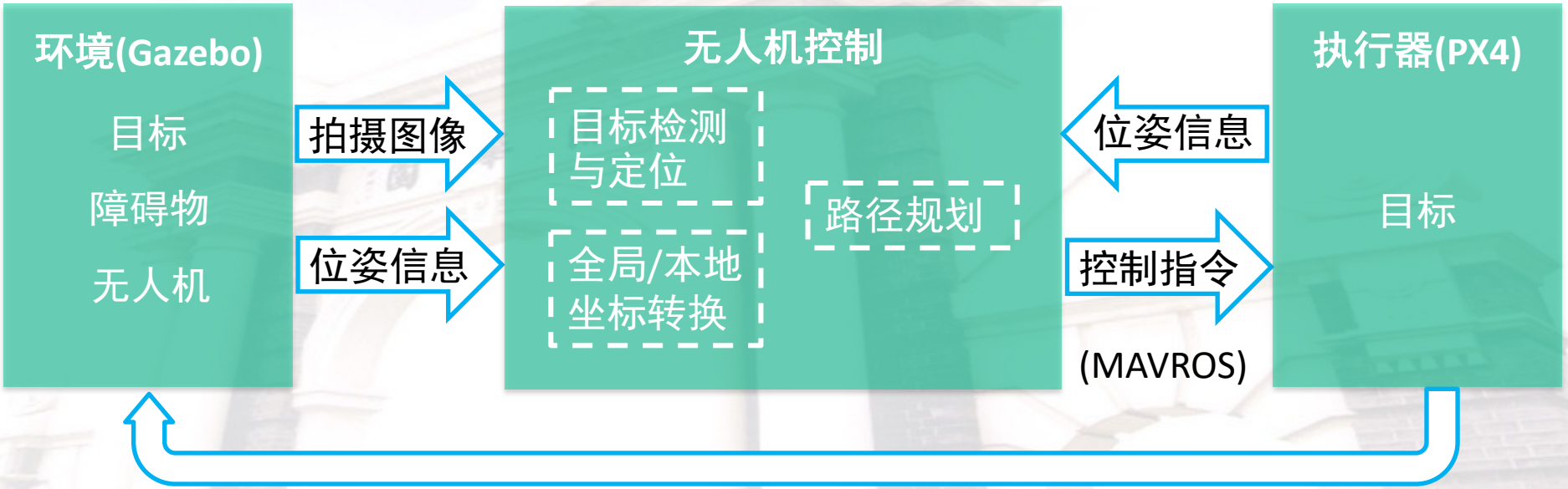




目 录

2 飞行控制流程

- **基本控制流程**
  - 无人机飞行流程如下图所示：
    - Gazebo完成物理场景的模拟，通过发布话题的方式输出场景中各种物体的状态信息，以及无人机的位姿信息和拍摄的图像等；
    - 无人机控制程序需要根据收集的数据进行信息融合并设计控制算法；
    - 控制程序借助于MAVROS向无人机发送飞行指令(但是在仿真环境中，这一部分已经封装为tello的控制接口)，由PX4完成指令的执行。



目 录

2 数据采集

- **数据采集**
  - 无人机的状态和传感器数据发布在一些话题上：
    - **/tello/states**：无人机的位置和姿态信息；
    - **/iris/usb_cam/** 下的几个话题：包括相机参数(/iris/usb_cam/camera_info) 和无人机摄像头原始图像(/iris/usb_cam/image_raw)等。

```
/iris/usb_cam/camera_info
/iris/usb_cam/image_raw
/iris/usb_cam/image_raw/compressed
/iris/usb_cam/image_raw/compressed/parameter_descriptions
/iris/usb_cam/image_raw/compressed/parameter_updates
/iris/usb_cam/image_raw/compressedDepth
/iris/usb_cam/image_raw/compressedDepth/parameter_descriptions
/iris/usb_cam/image_raw/compressedDepth/parameter_updates
```

ROS命令	作用
rostopic list	列出所有话题
rostopic info <topic_name>	获取话题信息，包括消息类型、发布者和订阅者
rostopic type <topic_name>	获取消息类型
rosmsg show <message_type>	获取消息格式

  - 要了解消息格式中各项参数的详细含义可以参考官方文档 (http://docs.ros.org/melodic/api/**gazebo_msgs**/html/msg/**ModelState**.html)对于不同类型的消息只需要修改对应的关键字即可。

2 数据采集

- **数据采集——位姿信息**
  - Model_name：无人机名称("iris")；
  - Pose：包含了position(三维坐标)和orientation(四元数)；
  - Twist：包含了linear(线速度)和angular(角速度)；

```
lee@drone:~$ rostopic info /tello/states
Type: gazebo_msgs/ModelState

Publishers:
 * /env_util (http://drone:33063/)

Subscribers: None


lee@drone:~$ rosmsg info gazebo_msgs/ModelState
string model_name
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
geometry_msgs/Twist twist
  geometry_msgs/Vector3 linear
    float64 x
    float64 y
    float64 z
  geometry_msgs/Vector3 angular
    float64 x
    float64 y
    float64 z
string reference_frame

lee@drone:~$
```

话题/tello/states的消息格式

```
lee@drone:~$ rostopic echo /tello/states
model_name: "iris"
pose:
  position:
    x: 1.03875649011
    y: 1.01442229384
    z: -0.0214539083335
  orientation:
    x: 0.000662820381949
    y: 7.17837374829e-06
    z: 0.00406718334834
    w: 0.999991509283
twist:
  linear:
    x: 4.90448355623e-09
    y: -3.13336520069e-07
    z: 4.26333852767e-07
  angular:
    x: 5.69389616403e-06
    y: 8.90578975567e-08
    z: 9.10215068319e-10
reference_frame: "map"
---
model_name: "iris"
pose:
  position:
    x: 1.00561958737
    y: 1.06892639789
    z: -0.00754372527311
  orientation:
    x: 0.000662831770328
```

话题/tello/states的实时消息

2 数据采集

- **数据采集——位姿信息**
  - 以python为例，读取无人机位姿信息的方式为：
    - 首先订阅话题/tello/states；
    - 有消息到来时，取出其位姿信息；
    - 从位姿信息中读出位置和姿态角信息进行处理。

```python
#!/usr/bin/python
#-*- encoding: utf8 -*-

import rospy
import numpy as np
from gazebo_msgs.msg import ModelStates
from scipy.spatial.transform import Rotation as R

def gazeboposeCallback(msg):
    #取出无人机位姿信息
    pose = msg.pose
    position = np.array([pose.position.x, pose.position.y, pose.position.z])
    #将四元数转化为scipy中的对象，便于后续处理
    orientation = R.from_quat([pose.orientation.x, pose.orientation.y, pose.orientation.z, pose.orientation.w])
    print("position ", position)
    print("orientation ", orientation)

rospy.init_node('get_pose_node', anonymous=True)
gazeboposeSub_ = rospy.Subscriber('/tello/states', ModelStates, gazeboposeCallback)
rospy.spin()

```

2 数据采集

- **数据采集**
  - 除了位姿信息，其余各种状态信息的获取都是类似的，只需要订阅相应话题，根据对应的消息格式读取数据即可。
  - 下面给出几个相关的、常用的ROS命令：

ROS命令	作用
rostopic list	列出所有话题
rostopic info <topic_name>	获取话题信息，包括消息类型、发布者和订阅者
rostopic type <topic_name>	获取消息类型
rosmsg show <message_type>	获取消息格式

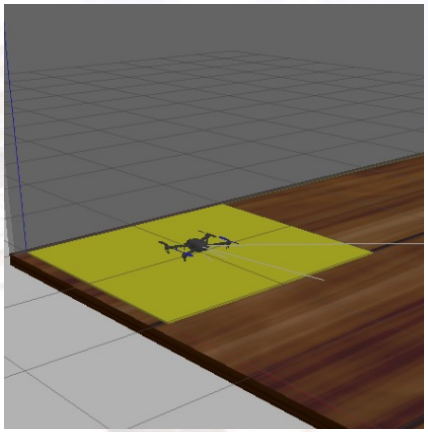
  - 要了解消息格式中各项参数的详细含义可以参考官方文档 (http://docs.ros.org/melodic/api/**gazebo_msgs**/html/msg/**ModelState**.html)对于不同类型的消息只需要修改对应的关键字即可。

2 数据采集

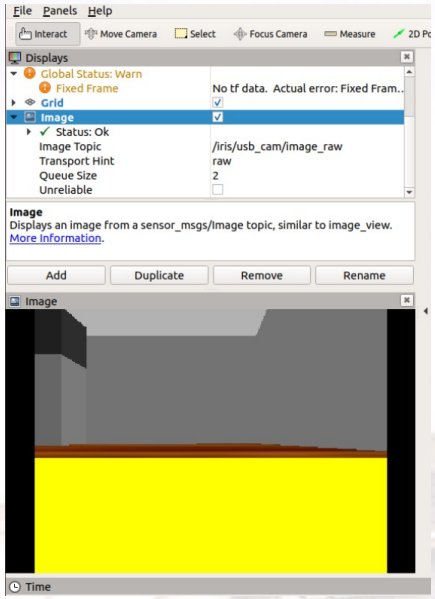
- **数据采集——图像信息**
  - ROS中的图片消息一般并不是单纯的图片，还包括了一些信息头、图像尺寸等信息，因此（以python为例）需要引入包 cv_bridge 进行格式转换；
  - 示例见下一页；

```
lee@drone:~$ rostopic info /iris/usb_cam/image_raw
Type: sensor_msgs/Image

Publishers:
 * /gazebo (http://drone:39389/)

Subscribers:
 * /rviz (http://drone:41567/)


lee@drone:~$ rosmsg show sensor_msgs/Image
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data

lee@drone:~$
```

话题/iris/usb_cam/image_raw的消息格式



Gazebo第三人称视角

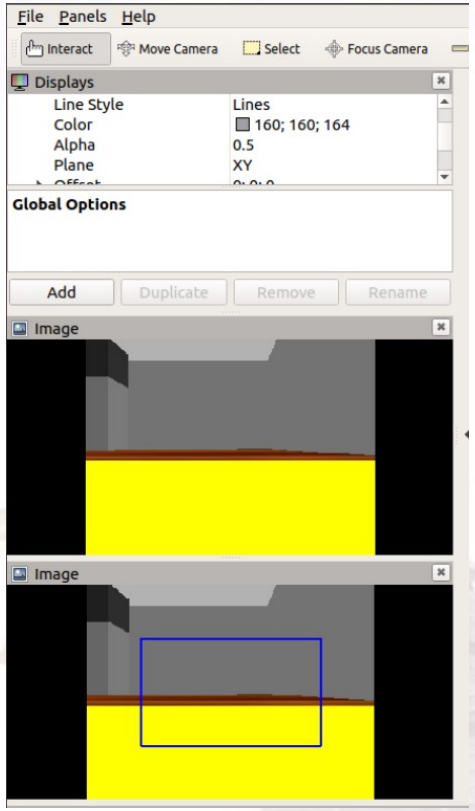


Rviz显示无人机拍摄图像

2 数据采集

- **数据采集——图像信息**
  - 以python为例，读取无人机拍摄图片的方式为：
    - 首先订阅话题/iris/usb_cam/image_raw ；
    - 有消息到来时，首先用 CvBridge 将消息转换为BGR格式的图片；
    - 对图片进行处理后再发布到指定话题上。
  - 下面的例子中在原始图像中间画了一个蓝色矩形框，rviz中效果如右图：

```python
#!/usr/bin/python
#-*- encoding: utf8 -*-

import rospy
import cv2
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

def imagesubCallback(data):
    try:
        bridge_ = CvBridge()
        #将sensor_msgs/Image类型的消息转化为BGR格式图像
        orgFrame_ = bridge_.imgmsg_to_cv2(data, 'bgr8')
        #打印图像尺寸
        orgFrame_copy = orgFrame_.copy()
        print("size of raw image: ",orgFrame_copy.shape)
        #在原始图像上画出矩形框
        cv2.rectangle(orgFrame_copy, (100, 100), (500, 300), (255,0,0), 2)
        #将BGR图像再转换为sensor_msgs/Image消息格式并发布
        image_result_pub_.publish(bridge_.cv2_to_imgmsg(orgFrame_copy))
    except CvBridgeError as err:
        print(err)


rospy.init_node('get_image_node', anonymous=True)
image_sub_ = rospy.Subscriber("/iris/usb_cam/image_raw", Image, imagesubCallback)
image_result_pub_ = rospy.Publisher("/get_image/image_processed", Image, queue_size=10)
rospy.spin()
```



目 录

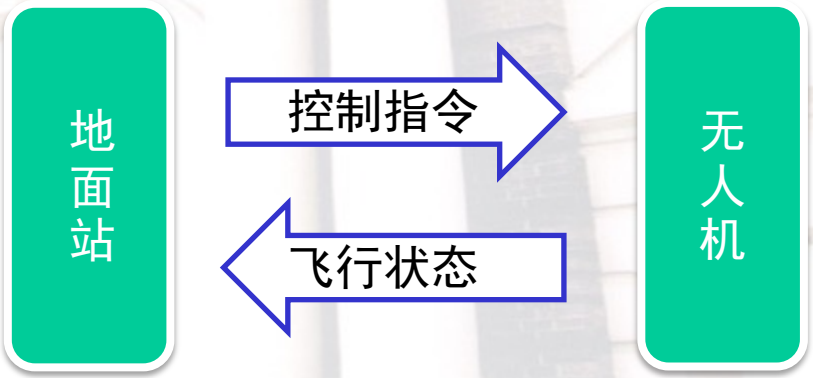
2 飞行控制

- **飞行控制**
  - 仿真环境中封装了常用的tello命令
  - 向话题 /tello/cmd_string 发布对应的字符串消息即可(控制频率不超过2Hz)

接口	含义
takeoff	起飞
land	降落
up x (20≤ x ≤500, 单位cm)	上升 x cm
down x	下降 x cm
forward x	向前 x cm
back x	向后 x cm
left x	向左 x cm
right x	向右 x cm
cw x (1≤ x ≤360, 单位deg)	顺时针旋转 x deg
ccw x	逆时针旋转 x deg
stop	在当前位置悬停

2 飞行控制

- **飞行控制——原理**
  - 无人机的飞行控制由PX4自动驾驶仪固件完成，PX4有多种不同的飞行模式(官方文档https://docs.px4.io/v1.9.0/en/flight_modes/index.html)
    - 手动模式(Manual)：通过手动的输入直接产生期望速度（比如摇杆）；
    - 外部控制(Offboard)：外部基于MAVLink通信协议，输入期望的位置、速度或者姿态信息（一般由计算机程序给出）；
    - 自动模式(Auto)：自动模式又包含多种不同子模式，比如起飞、降落等；
    - Others …
  - 我们在仿真实验中多采用外部控制模式，习惯上将控制指令发送方称为地面站，因此就需要地面站与无人机之间通过MAVLink通信协议进行通信。
  - MAVROS则是MAVLink协议在ROS中的封装，因此要想控制无人机，就需要借助MAVROS向无人机发送控制指令，然后由PX4执行。



2 飞行控制

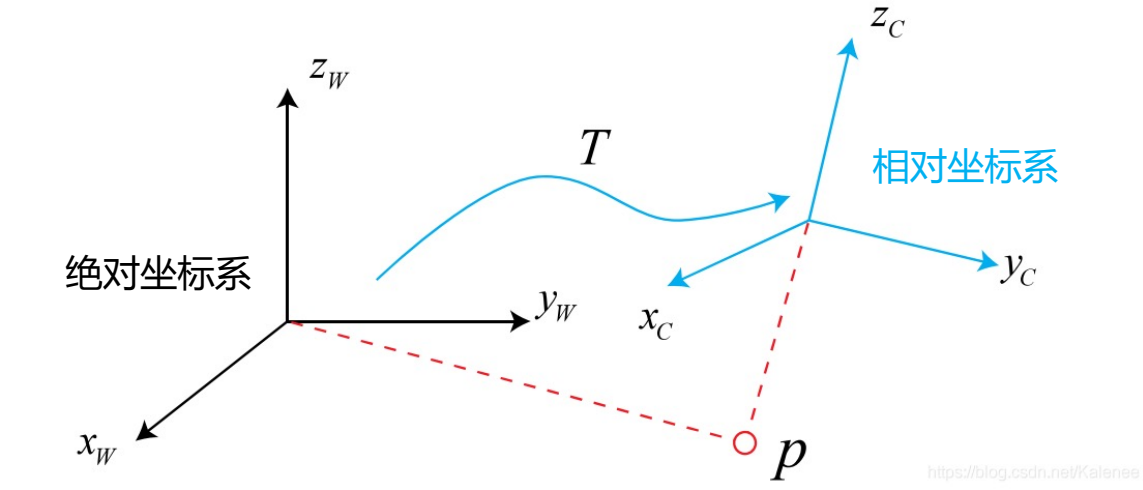
- **飞行控制——状态信息**
  - 在对无人机进行控制的时候，需要通过MAVROS获取无人机的状态信息，比较关键的ROS话题有：
    - **/mavros/state**：无人机状态信息；
    - **/mavros/global_position/global**：无人机GPS定位数据；
    - **/mavros/local_position/pose**：无人机相对于初始位置的相对位姿；
    - **/mavros/imu/data**：无人机惯导测量数据，包括朝向角、角速度、加速度以及各自对应的方差信息；
    - **/mavros/altitude**：无人机高度信息；
    - **/mavros/battery**：无人机电源信息。
  - 注1：前面提到的 /gazebo/model_states 中也可以获得无人机坐标，这个坐标是仿真场景中的绝对坐标，原点在设计时决定(我们的场景中原点是屋角)；
  - 注2：这里由MAVROS获得的世界坐标系GPS定位数据，包括经纬度等数据，使用得较少；
  - 注3：常用的是MAVROS本地坐标系，以初始化时的位置姿态作为原点，因此使用过程中需要与gazebo的绝对坐标进行坐标相互转换。

2 飞行控制

- **飞行控制——坐标转换**
  - 绝对坐标系与本地坐标系的相互转换可以应用矩阵运算完成：
    - 无人机初始化后，记录第一次收到的gazebo绝对坐标系下的位置position_init和姿态角orientation_init（可以转化为旋转矩阵 R）；
    - 后续绝对坐标p_global与相对坐标p_local的相互转换则可以由如下公式完成：

$$p_{local} = R * (p_{global} - p_{init})$$

$$p_{global} = R^{-1} * p_{local} + p_{init}$$



注：图片来自 https://blog.csdn.net/Kalenee/article/details/80792775

2 飞行控制

- **飞行控制——路径规划**
  - 可以将上面的所有操作集成在一个程序代码中(比如例程中的core_controller.py)，在launch文件中作为ROS节点运行。
  - 可以预先为无人机指定路径，飞行过程中只需要逐步到达各个目标节点即可，也可以在飞行过程中动态规划飞行路径，这将涉及到路径规划算法。
  - 若预先规划路径，只需要将关键的节点坐标保存在 /uav_sim/config/config.yaml 文件中，在launch中载入该文件，即可读取为ROS参数：

```
<node pkg="uav_sim" type="core_controller.py" name="core_controller" output="screen">
        <rosparam command="load" file="$(find uav_sim)/config/config.yaml" />
 </node>
```

  - 在python程序中可以通过以下代码获取该参数。

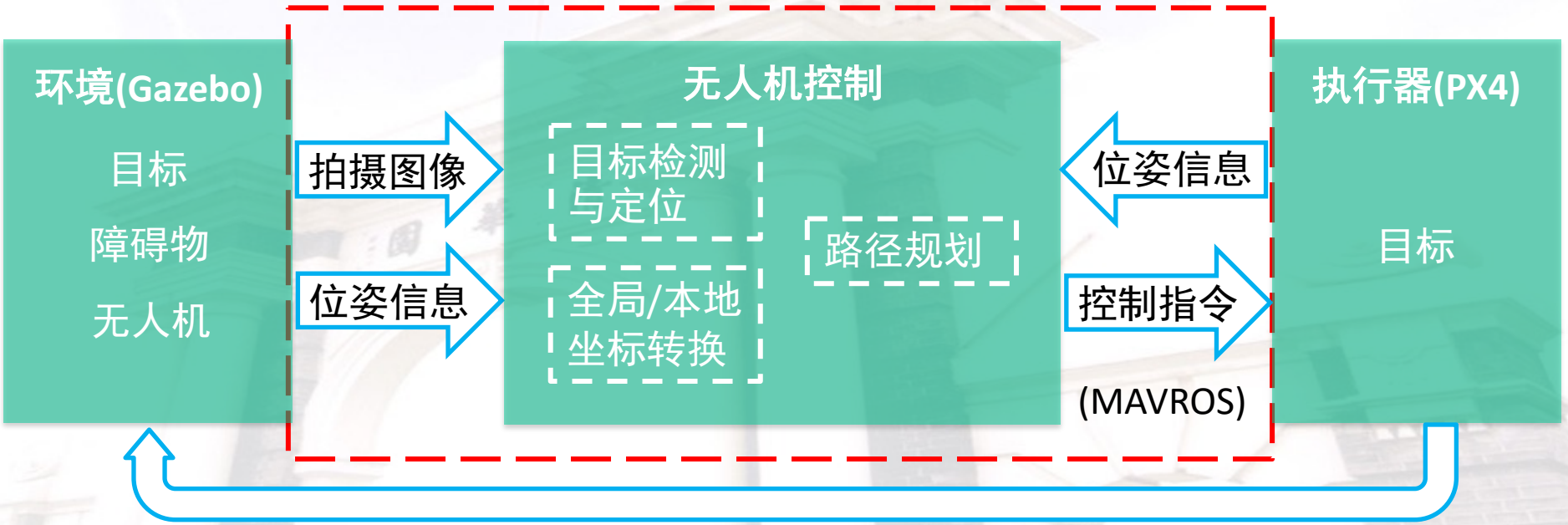
```
rospy.get_param('/core_controller/trajectory')
```

```
trajectory: [[1.,1.,2.,0.],
    [1., 4.5, 1., 0.],
    [1., 4.5, 1., 90.],
    [1., 4.5, 1., 180.],
    [1., 4.5, 1., 270.],
    [1., 4.5, 1., 0.],
    [1., 9.5, 1., 0.]]
```

目 录

3 小结

- **飞行控制流程**
  - 利用前面的基本操作，集成完整的无人机控制流程，下图虚线矩形框中就是我们要做的主要工作：
    - 收集自身与周围环境的目标信息
    - 根据收集信息决定控制指令
    - 发布控制指令并执行



谢谢！

附 录

- 1 认识仿真界面
- 2 基本功能介绍
  - 2.1 数据采集
  - 2.2 飞行控制
- 3 小结

2 飞行控制

- **飞行控制——起飞降落**
  - 要想对无人机进行控制，需要首先订阅相关话题和服务：
    - 订阅/mavros/state话题以获得无人机状态信息；
    - 订阅/mavros/set_mode以及/mavros/cmd/arming(land)服务以设置无人机飞行模式；
    - 向话题/mavros/setpoint_position/local发布无人机目标位置；

```
#订阅话题获取无人机状态
mavstateSub_ = rospy.Subscriber('/mavros/state', State, mavstateCallback)
#发布无人机目标位置
uav_target_pose_local_pub_ = rospy.Publisher('/mavros/setpoint_position/local', PoseStamped, queue_size=100)
#设置无人机状态
arming_client_ = rospy.ServiceProxy('/mavros/cmd/arming', CommandBool)
land_client_ = rospy.ServiceProxy('/mavros/cmd/land', CommandBool)
set_mode_client_ = rospy.ServiceProxy('/mavros/set_mode', SetMode)
```

  - ROS话题 /mavros/state 上发布了有关于无人机的状态信息，其中：
    - Connected：无人机是否连接；
    - Armed：无人机是否已经解锁；
    - Mode：当前无人机飞行模式；

```
lee@drone:~/catkin_sim/src/uav_sim/scripts$ rostopic info /mavros/state
Type: mavros_msgs/State

Publishers:
 * /mavros (http://drone:41729/)

Subscribers:
 * /fly_control_node (http://drone:41395/)


lee@drone:~/catkin_sim/src/uav_sim/scripts$ rosmsg show mavros_msgs/State
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
bool connected
bool armed
bool guided
bool manual_input
string mode
uint8 system_status

lee@drone:~/catkin_sim/src/uav_sim/scripts$
```

2 飞行控制

- **飞行控制——起飞降落**
  - 无人机发布飞行指令之前，需要首先完成：
    - 确定无人机已经连接(connected=True)；
    - 将飞行模式设定为外部控制模式(mode="OFFBOARD")；
    - 将无人机解锁(armed=True)
  - 可以借助ROS的定时器Timer来定时检测无人机状态，借助回调函数来修改无人机状态：

```
#定时检测无人机状态并进行处理
statusloop_timer_ = rospy.Timer(rospy.Duration(1), statusloopCallback)

def statusloopCallback(self, event):
    if not current_state_.connected:                                    #等待无人机连接
        pass
    else:
        if fly_status==0:                                               #等待起飞
            if current_state_.mode != 'OFFBOARD' and (rospy.Time.now() - last_request_ > rospy.Duration(2.0)):
                response = set_mode_client_(0, 'OFFBOARD')              #切换为OFFBOARD模式
                if response.mode_sent:
                    rospy.loginfo('Offboard enabled')
                last_request_ = rospy.Time.now()
            else:
                if (not current_state_.armed) and rospy.Time.now() - last_request_ > rospy.Duration(2.0):
                    response = arming_client_(True)                     #请求解锁
                    if response.success:
                        rospy.loginfo('Vehicle armed')
                    last_request_ = rospy.Time.now()
                    fly_status = 1
        elif fly_status == 1:                                           #飞行状态
            pass
        else fly_status == 2:                                           #准备降落
            rospy.loginfo('Ready to land')
            if current_state_.mode != 'AUTO.LAND':
                response = set_mode_client_(0, 'AUTO.LAND')             #请求降落
                if response.mode_sent:
                    rospy.loginfo('Vehicle landing')
                    flight_state_ = FlightState.LANDING
                last_request_ = rospy.Time.now()
        else:                                                           #其他状态
            ...
```

2 飞行控制

- **飞行控制——起飞降落**
  - 需要注意的是在切换到OFFBOARD模式之前，必须先发送一些期望点信息给飞控。不然飞控会拒绝切换到OFFBOARD模式。
  - 因此我们需要在程序的一开始向话题/mavros/setpoint_position/local发送一些目标点信息

```
#在切换到offboard模式之前，必须先发送一些期望点信息到飞控中。不然飞控会拒绝切换到offboard模式。
rate = rospy.Rate(20);      #20Hz
for i in range(10):
    uav_target_pose_local_pub_.publish(uav_target_pose_local_);
    rate.sleep()
```

  - 这里/mavros/setpoint_position/local的消息格式为geometry_msgs/PoseStamped，包含了期望的 xyz 坐标和四元数形式的姿态角。

```
lee@drone:~/catkin_sim/src/uav_sim/scripts$ rosmsg show geometry_msgs/PoseStamped
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w

lee@drone:~/catkin_sim/src/uav_sim/scripts$
```

2 飞行控制

- **飞行控制——飞行命令**
  - MAVROS在持续订阅话题/mavros/setpoint_position/local，因此要想让飞机飞到指定目标点，只需要向该话题发送期望的目标位置和姿态信息，PX4就能控制无人机飞往指定位置。
  - 不过需要注意的是MAVROS采用的是相对坐标，也就是相对于无人机初始化时刻的位置姿态作为原点，因此在发布目标坐标的时候需要进行绝对坐标和相对坐标的相互转化。
  - PX4在OFFBOARD模式下，两条指令的时间间隔必须小于500ms，也就是目标坐标的发布频率必须要大于2Hz。如果超时，则飞控会立即切换回进入OFFBOARD 模式之前的飞行模式。因此可以借助于ROS的Timer定时发送目标位置信息：

```
#向mavros话题发布目标位置
uav_target_pose_local_pub_ = rospy.Publisher('/mavros/setpoint_position/local', PoseStamped, queue_size=100)
#定时器定时发送目标坐标
pub_interval_ = rospy.Duration(0.02)
publishloop_timer_ = rospy.Timer(pub_interval_, publishloopCallback)

def publishloopCallback(event):
    uav_target_pose_local_pub_.publish(uav_target_pose_local_);
```