智能无人系统综合设计 无人机二维码追踪实验手册

在之前的课程中我们学习了 ROS 的基本概念，包括：节点、话题、消息、功能包、工作空间等。通过小乌龟仿真器和 TT 无人机我们学习了使用 ROS 命令发布和查看话题消息，以及使用 rviz 和 rqt 工具箱，还编写了两个功能包节点，实现了两只小乌龟的转圈，以及 TT 无人机人脸追踪。

本节课我们首先进一步将人脸追踪节点扩展至 QR code 二维码追踪，然后学习 ROS 中的坐标变换。通过坐标变换，我们会在 turtlesim 中实现两只小乌龟的相互跟随，并通过类似方式实现 TT 无人机对 apriltag 二维码的追踪。坐标变换所涉及的 tf/tf2 库，对后续课程 mini 无人车建图导航任务也至关重要。

回到上节课的思考，如何通过人脸追踪的方式，也就是根据目标框在图像中的位置和大小，实现 QR code 二维码追踪，并进一步实现当二维码原地转动时，无人机跟随转动至二维码的正前方？我们将给出一种实现，同学们想想还有没有更好的办法。

准备工作

下载 rmtt_ros_new.zip：

链接：https://pan.baidu.com/s/1Bkqu8ble-T3x7m3LacydNg

提取码：p1c2；

解压到~/tt_ws/src/（自行创建文件夹）；

在该工作空间（~/tt_ws/）下打开终端；

编译工作空间：catkin_make；

运行功能包中的节点前，请在该终端及新终端中输入：source ./devel/setup.bash；

或在 home 中打开终端，输入：gedit ~/.bashrc，将以上命令粘贴入文档并保存；

关闭所有终端，再次打开新终端可直接运行功能包中的节点。

QR code 二维码追踪

我们将在人脸追踪的基础上修改代码实现 QR code 二维码追踪，在 ~/tt_ws/src/rmtt_ros/rmtt_tracker/scripts 中有完整的人脸追踪代码，请同学们参考上节课的实验手册进行测试。

接下来看看 QR code 二维码追踪和人脸追踪的实现上有什么异同。

```
#!/usr/bin/python3
# coding=utf-8
# 环境准备：sudo apt-get install libzbar-dev
#           pip install pyzbar
# 输入话题：image_raw
# 输出话题：cmd_vel

import rospy
import rospkg
import sensor_msgs.msg
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Image
import cv2
from cv_bridge import CvBridge
import numpy as np
import pyzbar.pyzbar as pyzbar
```

安装和导入需要的函数库。

```
if __name__ == '__main__':

    bridge = CvBridge()
    rospy.init_node('qrcode_tracker')
    rospy.Subscriber("image_raw", Image, callback)
    pub = rospy.Publisher('cmd_vel', Twist, queue_size=1)
    rospy.spin()
```

从主函数看起，与人脸追踪类似，CvBridge()类是用来将 ROS 的图像格式转换为 OpenCV 图像处理库可以读取并处理的图像格式。这里给 CvBridge()类声明了一个对象 bridge，bridge 将可以使用 CvBridge()类中所定义的变量和函数。接下来定义了这个节点的名称，这个节点所订阅以及发布的话题。rospy.spin()的功能是一旦从订阅的话题接收到图片就调用 callback 回调函数，并且程序在此处循环，不再向下执行。

```
fw = 360
fh = 240
pid_w = [0.8, 0, 0] # pid parameters of yaw channel
pid_h = [0.8, 0, 0] # pid parameters of up channel
pid_f = [0.8, 0, 0] # pid parameters of forward channel
pid_l = [2.0, 0, 0] # pid parameters of left channel
```

我们定义了如上几个全局变量，fw 和 fh 是将摄像头看到的图像缩放到对应的宽和高。在控制参数部分，和人脸跟踪相比，新增了 pid_l 通道。这个通道用于向无人机输出水平平移的控制量。由于可以精确检测到二维码的左右边长，将使用检测到二维码的左右边长之比，判定二维码相对无人机转动的方向，并借助该比例生成水平平移控制量。

```

def callback(msg):
    global zero_twist_published
    img = bridge.imgmsg_to_cv2(msg)
    img = cv2.resize(img, (fw, fh))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    zero_twist_published = False
```

现在开始定义 callback 回调函数的内容，注意和人脸追踪的不同，比如变量名等。

```
def callback(msg):
    global zero_twist_published
    img = bridge.imgmsg_to_cv2(msg)
    img = cv2.resize(img, (fw, fh))
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    zero_twist_published = False

    # 从灰度图像中检测二维码
    barcodes = pyzbar.decode(gray)
    if barcodes :
        # 只提取识别到的第一个二维码
        barcode = barcodes[0]
        # 提取并绘制二维码的边界框的位置
        points =[]
        for point in barcode.polygon:
            points.append([point[0], point[1]])
        points = np.array(points,dtype=np.int32).reshape(-1,1, 2)
        cv2.polylines(img, [points], isClosed=True, color=(0,0,255),thickness=2)
        # 获取边界框左右边长比值
        x1 = points[0,0,0]
        y1 = points[0,0,1]
        x2 = points[1,0,0]
        y2 = points[1,0,1]
        x3 = points[2,0,0]
        y3 = points[2,0,1]
        x4 = points[3,0,0]
        y4 = points[3,0,1]
        if x2 < x4:
            hl = np.sqrt(np.square((x1 - x2)) + np.square((y1 - y2)))
            hr = np.sqrt(np.square((x4 - x3)) + np.square((y4 - y3)))
        else:
            hl = np.sqrt(np.square((x1 - x4)) + np.square((y1 - y4)))
            hr = np.sqrt(np.square((x2 - x3)) + np.square((y2 - y3)))
        ratio = hl / hr * 100

        # 画出图像中二维码的外接矩形边界框
        (x, y, w, h) = barcode.rect
        cv2.rectangle(img, (x, y), (x + w, y + h), (225, 225, 225), 2)
        # 提取二维码数据为字节对象，所以如果我们想在输出图像上
        # 画出来，就需要先将它转换成字符串
        barcodeData = barcode.data.decode("utf-8")
        barcodeType = barcode.type
        # 绘出图像上条形码的数据和条形码类型
        text = "{} ({})".format(barcodeData, barcodeType)
        cv2.putText(img, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
                    .5, (225, 225, 225), 2)
        # 向终端打印条形码数据和条形码类型
        print("[INFO] x: {} y:{} w:{} h:{} ratio:{} Found {} barcode: {}".format(x,y,w,h,ratio,
barcodeType, barcodeData))
        cx = x + w // 2
        cy = y + h // 2
        area = w * h
    else:
        cx = 0
        cy = 0
        area = 0
        ratio = 0.0001
    cv2.imshow("qrcode_detect_result", img)
    cv2.waitKey(1)
```

紧接着，在 callback 回调函数中，利用 pyzbar 库，从转换后的灰度图像中检测二维码。由于可能检测到多个二维码，只利用检测到的第一个二维码进行追踪。代码中有详尽的中文注释，最终我们希望能够获取二维码的中心点位置、二维码像素面积、左右边长的比例。cv2.imshow 函数用来将图像以小窗口的形式进行显示，包括所有在图像上绘制的边框、显示的字符等。

```
83
84      # 获得各个通道的控制量并显示在终端
85      yaw_speed, up_speed, forward_speed, left_speed = pidtrack(cx, cy, area, ratio, fw, fh, pid_w,
    pid_h, pid_f, pid_l)
86      rc = "forward: " + str(forward_speed) + " up: " + str(up_speed) + "  yaw: " + str(yaw_speed) + "
    left: " + str(left_speed)
87
88      speed = Twist()
89      if yaw_speed != 0 or up_speed != 0 or forward_speed != 0 or left_speed != 0:
90          speed.linear.x = -forward_speed
91          speed.linear.y = -left_speed
92          speed.linear.z = -up_speed
93          speed.angular.x = 0.0
94          speed.angular.y = 0.0
95          speed.angular.z = -yaw_speed
96          rospy.loginfo(rc)
97          pub.publish(speed)
98          zero_twist_published = False
99      else:
100         if not zero_twist_published:
101             pub.publish(speed)
102             zero_twist_published = True
103             rospy.loginfo("no object detected")
```

将相应的变量输入到 pidtrack 函数，返回控制量，向控制指令相关话题发布四个通道的控制量，分别是：偏航、上下、前后和左右。rc 是一个长字符串用于利用 rospy.loginfo(rc)在终端显示。

```
105 def pidtrack(cx, cy, area, ratio, fw, fh, pid_w, pid_h, pid_f, pid_l):
106     ## PID
107     # yaw channel
108     error_w = cx - fw // 2
109     speed_w = pid_w[0] * error_w
110     speed_w = int(np.clip(speed_w, -100, 100)) / 100.0
111
112     # up channel
113     error_h = cy - fh // 2
114     speed_h = pid_h[0] * error_h
115     speed_h = int(np.clip(speed_h, -100, 100)) / 100.0
116
117     # forward channel
118     error_f = np.sqrt(area) - 70
119     speed_f = pid_f[0] * error_f
120     speed_f = int(np.clip(speed_f, -100, 100)) / 100.0
121
122     # left channel
123     if (ratio - 100) >= 0:
124         error_l = ratio - 100
125     else:
126         error_l = - 10000.0 / ratio + 100
127     speed_l = pid_l[0] * error_l
128     speed_l = int(np.clip(speed_l, -100, 100)) / 100.0
129
130     if cx != 0:
131         yaw_speed = speed_w
132     else:
133         yaw_speed = 0
134     if cy != 0:
135         up_speed = speed_h
136     else:
137         up_speed = 0
138     if area != 0:
139         forward_speed = speed_f
140     else:
141         forward_speed = 0
142     if ratio != 0.0001:
143         left_speed = speed_l
144     else:
145         left_speed = 0
146
147     return yaw_speed, up_speed, forward_speed, left_speed
```

pidtrack 函数与人脸追踪的 facetrack 类似，注意增加一个控制通道。

即将大功告成，如果代码正确，可以通过 rosrun 命令来运行 QR code 二维码追踪节点。我们还可以在当前 package 下的 launch 文件夹中创建如下的 xxx.launch 文件，这样我们便能够用 roslanuch 命令运行该节点。

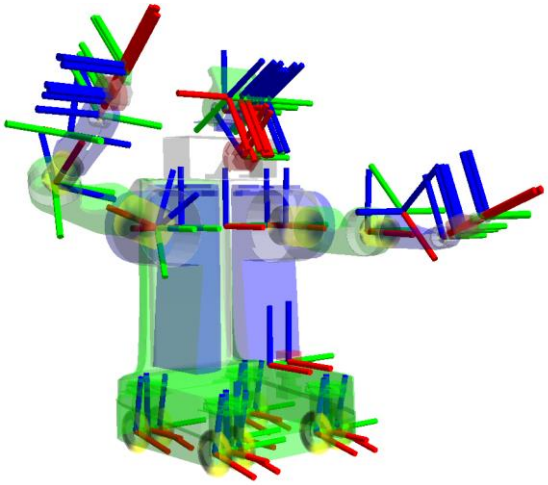
```
<?xml version="1.0"?>

<launch>
  <arg name = "ns" default="$(optenv RMTT_NAMESPACE /)" />
  <group ns="$(arg ns)">
    <node pkg="rmtt_tracker" type="rmtt_qrcode_tracker.py" name="qrcode_tracker" output="screen" respawn="true">
    </node>
  </group>
</launch>
```

通过编写 lanuch 文件，可以一次性运行多个节点。请同学们修改以上的 launch 文件，在 TT 无人机驱动正常启动后，一次性运行键盘控制和二维码追踪节点。

tf 变换基础知识

坐标变换是机器人学中一个非常基础同时也是非常重要的概念。机器人本体和机器人的工作环境中往往存在大量的组件元素，在机器人设计和机器人应用中都会涉及不同组件的位置和姿态，这就需要引入坐标系以及坐标变换的概念。



只要我们能够知道当前坐标系在参考坐标系的描述(平移和旋转)，我们就可以把当前坐标系里任何一个点的坐标变换成参考坐标系里的坐标。刚体的任一姿态（刚体坐标系相对于参考坐标系）可以经由三次基本旋转得到，用三个角来描述，这就是欧拉角。

ROS 中使用 tf 软件库进行坐标转换，现在已经有 tf2 库了，而且比 tf 库更加功能强大，建议优先选用 tf2 进行开发。在坐标变换中，两个坐标轴的关系（也就是转换信息）用一个 6 自由度的相对位姿表示：平移量（translation）+旋转量（rotation）。其中平移量

就是一个三维向量，旋转量可以用一个旋转量矩阵表示，tf 中没有表示旋转量矩阵的类型，而是通过四元数类型 tf::Quaternion 来表示。四元数是刚体姿态的另一种描述方式，理论基础是，刚体姿态可以经过某一特定轴经一次旋转一定角度得到，等价于欧拉角，等价于旋转矩阵。

利用 tf 库管理坐标系主要要做的就是两件事：**监听 tf 变换和广播 tf 变换。**

监听 tf 变换：接收并缓存系统中发布的所有参考系变换，并从中查询所需要的参考系变换。

广播 tf 变换：向系统中广播参考系之间的坐标变换关系。

tf 变换描述某个坐标系(child_frame_id)相对于另一个参考坐标系(frame_id)在某个时刻的位姿关系（平移+旋转矩阵）。tf 维护的坐标系关系其实是将有关坐标系话题发布在 tf::tfMessage 类型消息中，该消息定义为 geometry_msgs/TransformStamped 类型的向量：

```
geometry_msgs/TransformStamped[] transforms
  std_msgs/Header header
    uint32 seq
    time stamp
    string frame_id
  string child_frame_id
  geometry_msgs/Transform transform
    geometry_msgs/Vector3 translation
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion rotation
      float64 x
      float64 y
      float64 z
      float64 w
```

通过 tf 变换实现小乌龟跟踪

tf 变换比较抽象，我们将通过小乌龟跟踪的例程来理解 tf 工具及其使用。

首先确保已经安装了该例程：

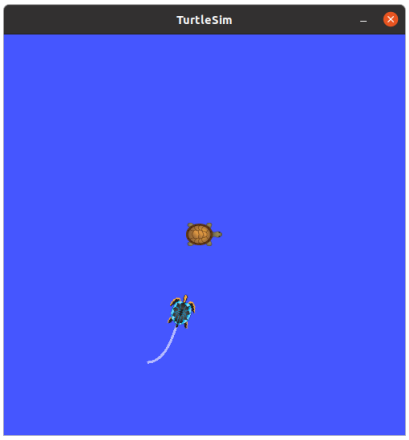
sudo apt-get install ros-noetic-turtle-tf2

由于 ROS noetic 已经全面支持 python3，需要解决代码兼容性问题，在终端输入：

sudo apt-get install python-is-python3

现在，可以运行例程了：

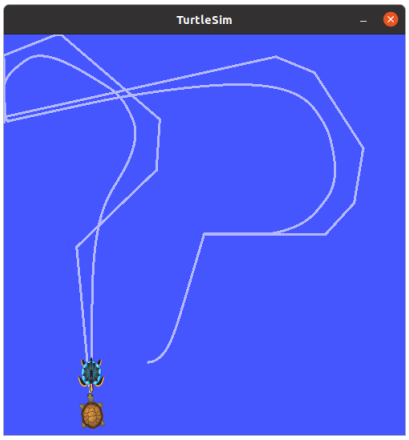
roslaunch turtle_tf2 turtle_tf2_demo.launch



应该能看到两只小乌龟。

打开键盘控制节点，通过方向键进行控制，看看会发生什么：

rosrun turtlesim turtle_teleop_key



该例程使用 tf2 库创建了三个坐标系：世界坐标系、turtle1 坐标系和 turtle2 坐标系。使用 tf2 广播器发布 turtle 标系，并使用 tf2 监听并计算 turtle 标系中的差异并移动一只乌龟跟随另一只乌龟。

由于 tf 工具箱和 ROS noetic 已经部分不兼容了，先安装 tf2 工具箱，然后查看一下当前有哪几个坐标系，它们之间存在怎样的坐标变换：

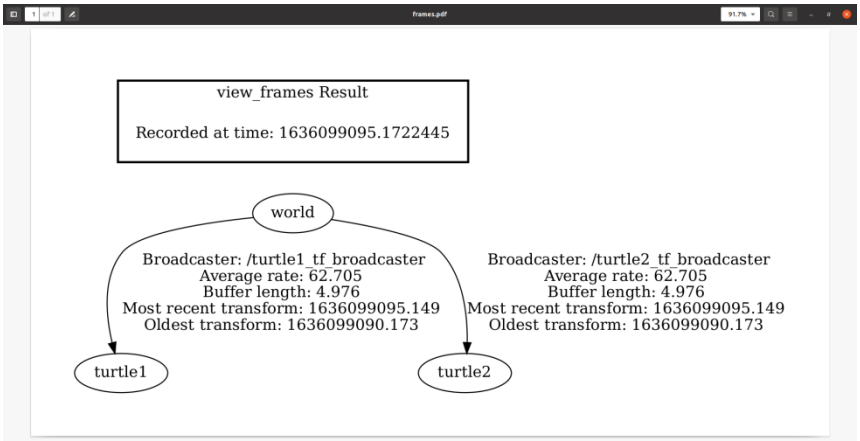
sudo apt-get install ros-noetic-tf2-tools

若无法定位，尝试下载以下包，解压至~tt_ws/src 下，catkin_make 进行编译：

链接：https://pan.baidu.com/s/1i2g0bicYHtwLxw2Nl802SQ 提取码：bn71

rosrun tf2 view_frames

以上命令会将 tf 树的 pdf 文件保存在当前目录。



在上图中，可以看到 tf2 广播的三个坐标系：世界坐标系，turtle1 坐标系和 turtle2 坐标系，并且世界坐标系是 turtle1 和 turtle2 坐标系的父级。 view_frames 还报告一些诊断信息，这些信息有关何时接收到最旧和最新的坐标系转换，以及发布的速度。

现在要让 turtle2 跟随 turtle1 运动，也就是让 turtle2 的本体坐标系需要向 turtle1 的本体坐标系移动，这就需要知道 turtle2 和 turtle1 之间的坐标变换关系。使用 tf_echo 工具在 tf 树中查找小乌龟坐标系间的转换关系，指令格式：

rosrun tf tf_echo [reference_frame] [target_frame]

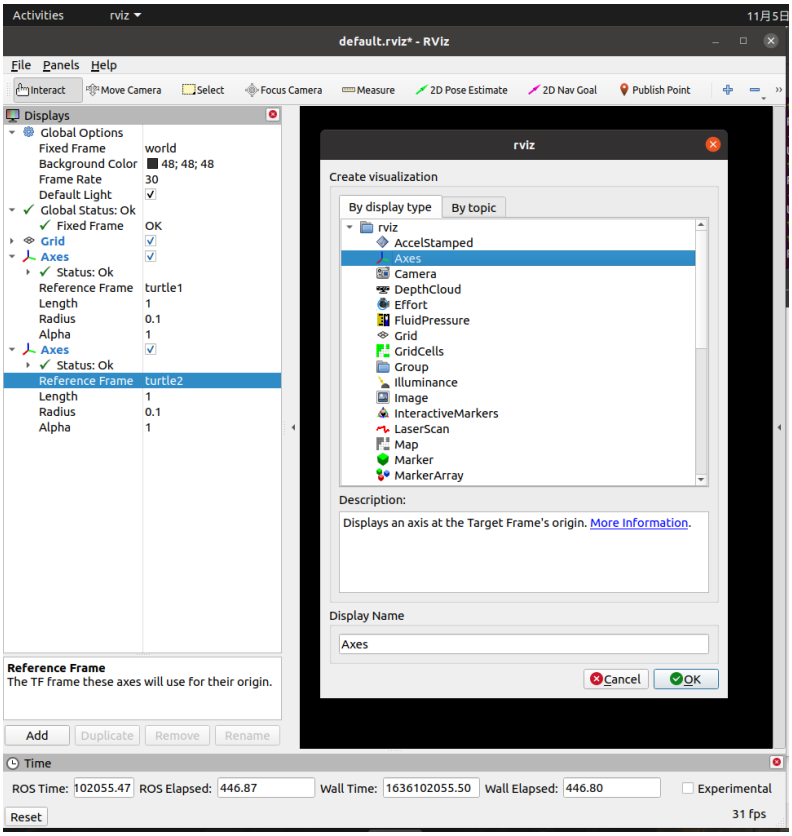
看一下 turtle2 坐标系相对于 turtle1 坐标系的变换：

rosrun tf tf_echo turtle1 turtle2

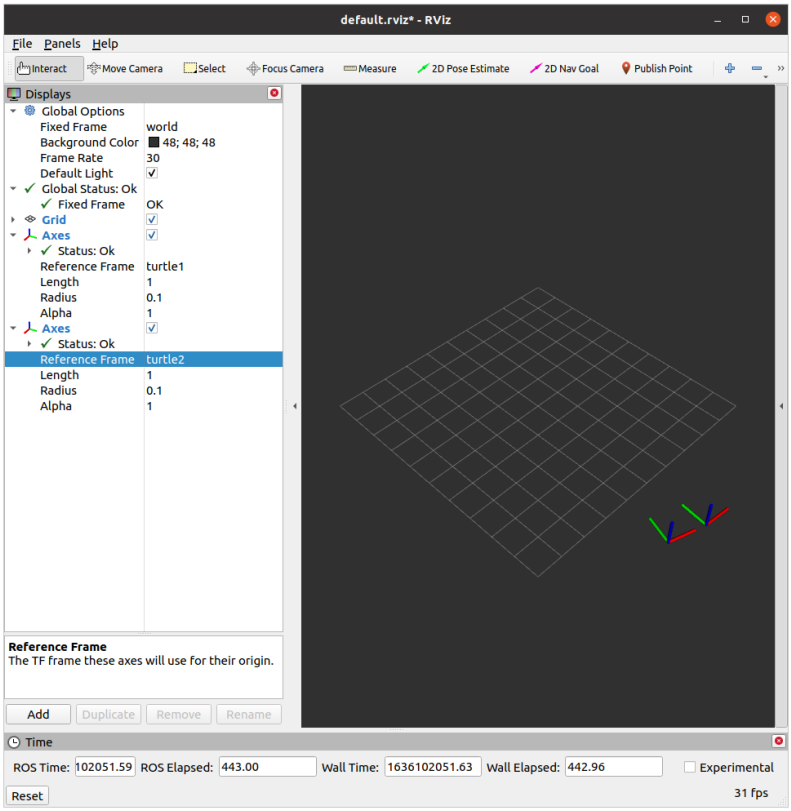
```
- Rotation: in Quaternion [0.000, 0.000, -0.261, 0.965]
            in RPY (radian) [0.000, 0.000, -0.528]
            in RPY (degree) [0.000, 0.000, -30.258]
At time 1636100724.570
- Translation: [-3.010, 0.541, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.110, 0.994]
            in RPY (radian) [0.000, 0.000, -0.220]
            in RPY (degree) [0.000, 0.000, -12.591]
At time 1636100725.562
- Translation: [-3.409, 0.298, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.052, 0.999]
            in RPY (radian) [0.000, 0.000, -0.105]
            in RPY (degree) [0.000, 0.000, -6.013]
At time 1636100726.557
- Translation: [-3.416, 0.171, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.028, 1.000]
            in RPY (radian) [0.000, 0.000, -0.056]
            in RPY (degree) [0.000, 0.000, -3.206]
At time 1636100727.564
- Translation: [-2.037, 0.100, 0.000]
- Rotation: in Quaternion [0.000, 0.000, -0.025, 1.000]
            in RPY (radian) [0.000, 0.000, -0.049]
            in RPY (degree) [0.000, 0.000, -2.828]
```

其中 turtle1 是 parent 坐标系，turtle2 是 child 坐标系。当移动乌龟时，会看到随着两只乌龟相对移动，坐标变换发生了变化。

在新的终端输入：rviz。在 rviz 中查看两只小乌龟本体坐标系的运动。



如图添加两个 Axes 然后选择正确的 Fixed Frame 和 Reference Frame。



通过键盘控制小乌龟运动，会看到两个坐标系也动了，并且一个跟随一个。

编写 tf 广播和监听节点

本节我们将学习以上过程是如何实现的，可参考以下代码：

链接：https://pan.baidu.com/s/1MjE0GXSIZc28e961zT7veQ 提取码：3ya0

创建一个名为 learning_tf2 的功能包，该功能包将用于学习 tf2。这功能包依赖 tf2、tf2_ros、roscpp、rospy 和 turtlesim 库。在~tt_ws/src 目录下打开终端，输入以下命令：

catkin_create_pkg learning_tf2 tf2 tf2_ros roscpp rospy turtlesim

启动 ROS master：

roscore

启动一个小乌龟仿真节点：

rosrun turtlesim turtlesim_node

还记不记得我们如何通过 rosservice call /spawn 服务命令调出一只新的小乌龟？我们也可以写一个节点实现此功能。在/learning_tf2/src 文件夹下创建名为 turtle_spawn.py 的 python 文件，并写入以下代码：

Open turtle_spawn.py ~/tt_ws/src/learning_tf2/src Save

turtle1_tf2_broadcaster.py | turtle_tf2_listener.py | turtle_spawn.py

```python
#! /usr/bin/env python3

import rospy
from turtlesim.srv import Spawn, SpawnRequest, SpawnResponse

if __name__ == "__main__":

    rospy.init_node("turtle_spawn")
    # 创建服务客户端
    client = rospy.ServiceProxy("/spawn",Spawn)
    # 等待服务启动
    client.wait_for_service()
    # 创建请求数据
    req = SpawnRequest()
    req.x = 1.0
    req.y = 1.0
    req.theta = 3.14
    req.name = "turtle2"
    # 发送请求并处理响应
    try:
        response = client.call(req)
        rospy.loginfo("乌龟创建成功，名字是:%s",response.name)
    except Exception as e:
        rospy.loginfo(e)
```

Python 3 Tab Width: 8 Ln 1, Col 1 INS

完成后运行：

rosrun learning_tf2 turtle_spawn.py

注意运行先要开启 python 文件权限，之后将不再赘述。

在/learning_tf2/src 文件夹下创建名为 turtle1_tf2_broadcaster.py 的 python 文件，并写入以下代码：

Open turtle1_tf2_broadcaster.py ~/tt_ws/src/learning_tf2/src Save

```python
#!/usr/bin/env python3

import rospy
import tf2_ros
from tf.transformations import quaternion_from_euler, euler_from_quaternion
import turtlesim.msg
from geometry_msgs.msg import TransformStamped

def handle_turtle_pose(msg, turtlename):
    br = tf2_ros.TransformBroadcaster()

    t = TransformStamped()
    # Read message content and assign it to
    # corresponding tf variables
    t.header.stamp = rospy.Time.now()
    t.header.frame_id = 'world'
    t.child_frame_id = turtlename

    # Turtle only exists in 2D, thus we get x and y translation
    # coordinates from the message and set the z coordinate to 0
    t.transform.translation.x = msg.x
    t.transform.translation.y = msg.y
    t.transform.translation.z = 0.0

    # For the same reason, turtle can only rotate around one axis
    # and this why we set rotation in x and y to 0 and obtain
    # rotation in z axis from the message
    q = quaternion_from_euler(0, 0, msg.theta)
    t.transform.rotation.x = q[0]
    t.transform.rotation.y = q[1]
    t.transform.rotation.z = q[2]
    t.transform.rotation.w = q[3]

    br.sendTransform(t)

if __name__ == '__main__':
    rospy.init_node('turtle1_tf2_broadcaster')
    turtlename = 'turtle1'
    rospy.Subscriber('/%s/pose' % turtlename,
                     turtlesim.msg.Pose,
                     handle_turtle_pose,
                     turtlename)
    rospy.spin()
```

Python 3 Tab Width: 8 Ln 1, Col 23 INS

该文件订阅 turtle1 的 pose，然后广播相对 world 的坐标系信息。

实现流程：1. 导包；2. 初始化 ros 节点；3. 创建订阅对象；4. 回调函数处理订阅的 pose 信息；6. 回调函数中创建 tf 广播器 ；7. 将 pose 信息转换成 TransFormStamped；8. 发布。

完成后运行：

rosrun learning_tf2 turtle1_tf2_broadcaster.py

写一个节点发布 turtle2 相对 world 的坐标系信息，在/learning_tf2/src 文件夹下创建名为 turtle2_tf2_broadcaster.py 的 python 文件，修改以上代码的节点名和 turtlename。

完成后运行：

rosrun learning_tf2 turtle2_tf2_broadcaster.py

在/learning_tf2/src 文件夹下创建名为 turtle_tf2_lisener.py 的 python 文件，并写入以下代码：

Open turtle_tf2_listener.py ~/tt_ws/src/learning_tf2/src Save

```python
#! /usr/bin/env python3

import rospy
import tf2_ros
from geometry_msgs.msg import TransformStamped, Twist
import math

if __name__ == "__main__":

    rospy.init_node("sub_tfs_p")

    # 创建 TF 订阅对象
    buffer = tf2_ros.Buffer()
    listener = tf2_ros.TransformListener(buffer)
    # 处理订阅到的 TF
    rate = rospy.Rate(10)
    # 创建速度发布对象
    pub = rospy.Publisher("/turtle2/cmd_vel",Twist,queue_size=1000)
    while not rospy.is_shutdown():

        rate.sleep()
        try:
            #def lookup_transform(self, target_frame, source_frame, time, timeout=rospy.Duration(0.0)):
            trans = buffer.lookup_transform("turtle2","turtle1",rospy.Time(0))
            # rospy.loginfo("相对坐标:(%.2f,%.2f,%.2f)",
            #             trans.transform.translation.x,
            #             trans.transform.translation.y,
            #             trans.transform.translation.z
            #             )
            # 根据转变后的坐标计算出速度和角速度信息
            twist = Twist()
            # 间距 = x^2 + y^2  然后开方
            twist.linear.x = 0.5 * math.sqrt(math.pow(trans.transform.translation.x,2) + math.pow(trans.transform.translation.y,2))
            twist.angular.z = 4 * math.atan2(trans.transform.translation.y, trans.transform.translation.x)

            pub.publish(twist)

        except Exception as e:
            rospy.logwarn(e)
```

Python 3 Tab Width: 8 Ln 1, Col 24 INS

该文件订阅 turtle1 和 turtle2 的 tf 广播信息，查找并转换时间最近的 TF 信息，将 turtle1 转换成相对 turtle2 的坐标，计算线速度和角速度并发布。

实现流程：1. 导包；2. 初始化节点 ；3. 创建 tf 订阅对象；4. 处理订阅到的 tf；5. 查找坐标系的相对关系；6. 生成速度信息然后发布。

完成后运行：

rosrun learning_tf2 turtle_tf2_listener.py

运行小乌龟键盘控制节点：

rosrun turtlesim turtle_teleop_key

通过键盘遥控，看看有没有实现小乌龟的运动跟随。

Open | learning_tf2.launch ~/tt_ws/src/learning_tf2/launch | Save

learning_tf2.launch | turtle1_tf2_broadcaster.py | turtle_tf2_listener.py

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

<launch>
    <node pkg="turtlesim" type="turtlesim_node" name="turtle1" output="screen" />
    <node pkg="turtlesim" type="turtle_teleop_key" name="key_control"
output="screen"/>

    <node pkg="learning_tf2" type="turtle_spawn.py" name="turtle_spawn"
output="screen"/>

    <node pkg="learning_tf2" type="turtle1_tf2_broadcaster.py"
name="turtle1_tf2_broadcaster" output="screen"/>
    <node pkg="learning_tf2" type="turtle2_tf2_broadcaster.py"
name="turtle2_tf2_broadcaster" output="screen"/>
    <node pkg="learning_tf2" type="turtle_tf2_listener.py"
name="turtle_tf2_listenr" output="screen"/>

</launch>
```

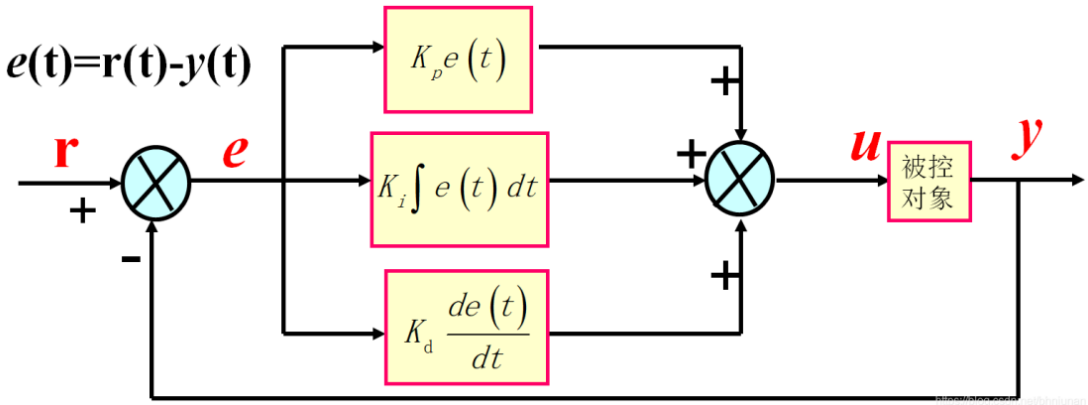
XML | Tab Width: 8 | Ln 2, Col 1 | INS

关闭所有终端，在/learning_tf2/launch 文件夹下，创建名为 learning_tf2.launch 的 launch 文件组织需要运行的节点。完成后运行：

roslaunch learning_tf2 learning_tf2.launch

PID 控制基础知识

PID 控制器是一种线性控制器，它主要根据给定值和实际输出值构成控制偏差，然后利用偏差给出合理的控制量。PID 里的 P 是 Proportion 的首字线，是比例的意思，I 是 Integral 的首字线，是积分的意思，D 是 Differential 的首字母，是微分的意思。



以无人机为例在没有控制系统的情况下，直接用信号驱动电机带动螺旋桨转动产生控制力，会出现动态响应太快，或者太慢，或者控制过冲或者不足的现象，无人机无法很好的完成跟踪、转弯等动作。为了解决这些问题，就需要在控制系统回路中加入 PID 控制器算法。在位置信息、速度信息和电机转速之间建立比例、积分和微分的关系，通过调节各

个环节的参数大小，使无人机系统控制达到动态响应迅速、既不过冲、也不欠缺，可以完美的跟踪与转弯。

以位置控制为例，比例控制信号为系数 P 乘以指定位置和受控对象位置的差值。P 值越大，被控对象反应越快，P 值越小，被控对象反应越慢。虽然 P 值大能够较快的到达指定位置，但是反应比较剧烈，总是因为过快冲过了头。相反 P 值小的反应比较平缓，但是它反应太慢，我们有时候接受不了。

为了不让被控对象冲过头，我们再给它加一个力，这个力就是 D，让这个力来起一个作用，就是让被控对象越接近指定位置的时候，接近的速度越慢，越远离指定位置的时候，接近的速度相对较快。这个 D 大家可以理解为受控对象靠近指定位置的一个阻力。

设置合适的 P 值和 D 值后，一些干扰如有风等就会让受控对象发生偏移，因为此时 P 和 D 产生的控制信号值均较小。所以我们再给它加一个力，这个力就是 I，积分控制。设置了 I 以后，I 会根据误差和误差经历的时间进行积分，然后决定施加给目标方向的力的大小，就能够让受控对象回到指定位置上。

apriltag 二维码追踪

rmtt_ros 中所包含的功能包如下图所示：

```
tianbot@ros2go:~/robomaster_ws/src/rmtt_ros$ tree -L 1
.
├── README.md
├── rmtt_apriltag
├── rmtt_description
├── rmtt_driver
├── rmtt_ros
├── rmtt_teleop
└── rmtt_tracker
```

rmtt_apriltag

无人机跟踪程序功能包。依赖于 PyPI 的 simple-pid，输入以下指令安装：

pip3 install simple-pid

二维码检测功能包，依赖于 apriltag_ros 功能包。输入以下指令安装：

sudo apt install ros-noetic-apriltag-ros

或从百度网盘下载：

链接：https://pan.baidu.com/s/1ueDxPrpluq0-l2ESvpcHFw

提取码：p5mh

解压将 apriltag、apriltag_ros 文件夹放到~/tt_ws/src 下并编译：catkin_make_isolated

source devel_isolated/setup.bash

rmtt_description

无人机的描述文件功能包，提供无人机的坐标变换。

rmtt_driver

无人机驱动。

rmtt_teleop

无人机遥控功能包。

rmtt_tracker

原理说明：

二维码追踪的功能与人脸追踪有很多不同，我们需要控制飞机到 apriltag 二维码的相对姿态。TT 的摄像头可以检测到二维码并且发布摄像头到二维码的坐标变换（camera_link -> tag_xxx）。我们可以将期望无人机到达的位置在二维码的坐标系中发布出来，这样只需要比较无人机坐标系（base_link）和期望坐标系的关系，用 pid 进行调节控制，就可以实现无人机的二维码跟踪。

实验步骤：

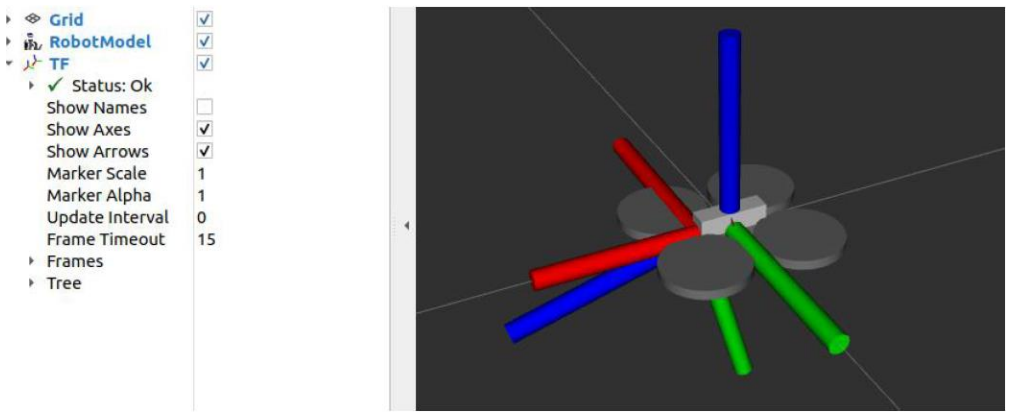
1、连接无人机到计算机，并启动无人机驱动：

roslaunch rmtt_driver rmtt_bringup.launch

2、启动无人机的描述文件（坐标变换）

roslaunch rmtt_description rmtt_description.launch

打开 rviz 添加 RobotModel 和 TF，将 Global Options->fixed frame 更改为 base_link，可以看到 rviz 的显示区域如下图所示：



3、启动二维码检测程序

roslaunch rmtt_apriltag detection.launch

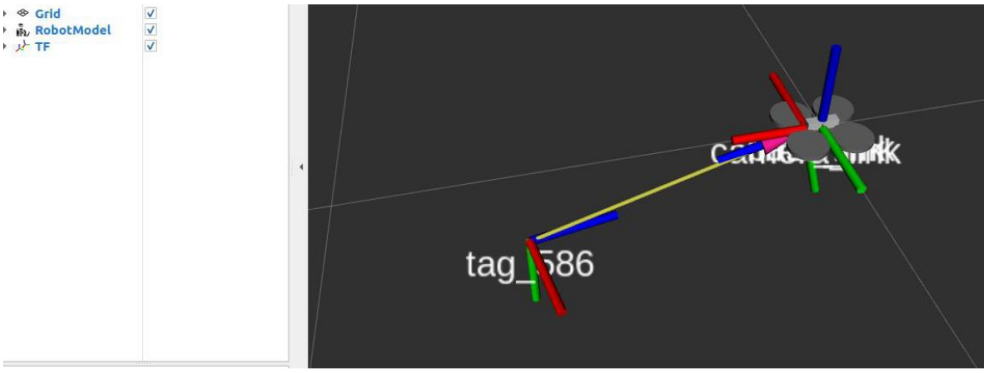
这时候飞机就开始检测二维码了。需要根据实际二维码的 id 修改 rmtt_apriltag/config/tags.yaml 文件中检测的二维码 id。

```
NODES
  /rmtt/
    apriltag_ros (apriltag_ros/apriltag_ros_continuous_node)

ROS_MASTER_URI=http://localhost:11311

process[rmtt/apriltag_ros-1]: started with pid [137326]
[ INFO] [1620805973.352163585]: Initializing nodelet with 8 worker threads.
[ INFO] [1620805973.430806447]: Loaded tag config: 10, size: 0.12, frame_name: tag_10
[ INFO] [1620805973.430851508]: Loaded tag config: 20, size: 0.12, frame_name: tag_20
[ INFO] [1620805973.430864964]: Loaded tag config: 30, size: 0.12, frame_name: tag_30
[ INFO] [1620805973.430876467]: Loaded tag config: 586, size: 0.12, frame_name: tag_586
[ WARN] [1620805973.431182101]: No tag bundles specified
[ WARN] [1620805973.431470684]: remove_duplicates parameter not provided. Defaulting to true
```

如图所示，现在可以检测的是 10 号，20 号，30 号和 586 号标签。

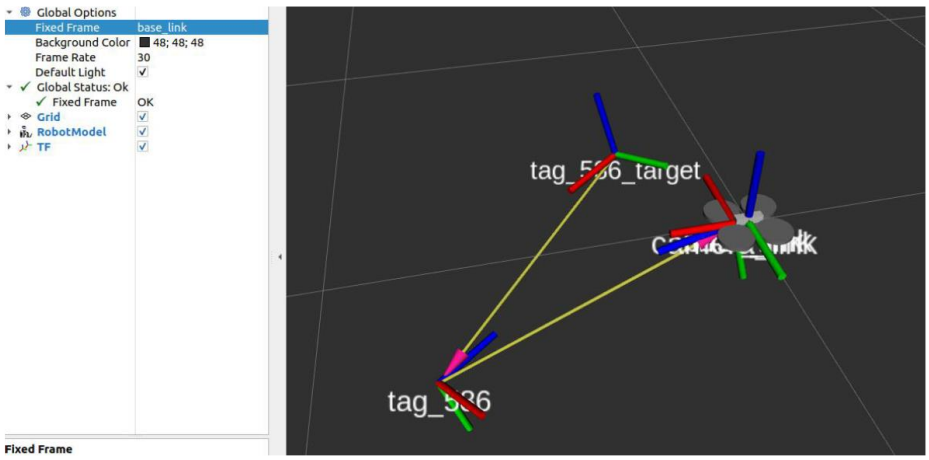


如上图所示，二维码标签的坐标可以在 base_link 中显示。

4、设置目标坐标系并且跟踪。注意下 apriltag 是可以一次检测多个二维码标签的，这里要指定我们需要跟踪哪个标签，默认是最后一个标签，id 为 586。当然需要在 rmtt_apriltag/config/tags.yaml 文件中进行设置。

roslaunch rmtt_tracker rmtt_tag_tracker.launch

可以看到我们设置了一个 tag_586_target 的坐标系，目标就是让飞机追踪这个坐标系。这时后可以用 rostopic echo /cmd_vel 命令查看飞机速度控制话题，应该有相应的输出。不过飞机还没有起飞，并不会动。



5、启动遥控程序

roslaunch rmtt_teleop rmtt_teleop_key.launch

用键盘遥控飞机，飞到能够观察到二维码的地方，飞机就开始追踪二维码了。可以思考一下，这个追踪程序与 QR code 二维码追踪有什么不同？

代码分解：

追踪程序的源代码在 rmtt_tracker/scripts/rmtt_tag_tracker.py

程序主要部分，一个是二维码检测的回调函数。这里主要是控制是否发送速度消息。在检测不到二维码的时候会将速度置 0，然后就不再发送。这样可以方便其他程序比如遥控来控制无人机。

```
20 def tag_callback(msg):
21     # Do not publish cmd_vel when there is no tag detected
22     global zero_twist_published
23     tag_detected = False
24 
25     for i in msg.detections:
26         # print (i)
27         if int(tag_id)  in i.id:
28             # print(i.pose.pose)
29             vel_pub.publish(vel)
30             zero_twist_published = False
31             tag_detected = True
32 
33     if not tag_detected:
34         if not zero_twist_published:
35             zero_twist = Twist()
36             vel_pub.publish(zero_twist)
37             zero_twist_published = True
```

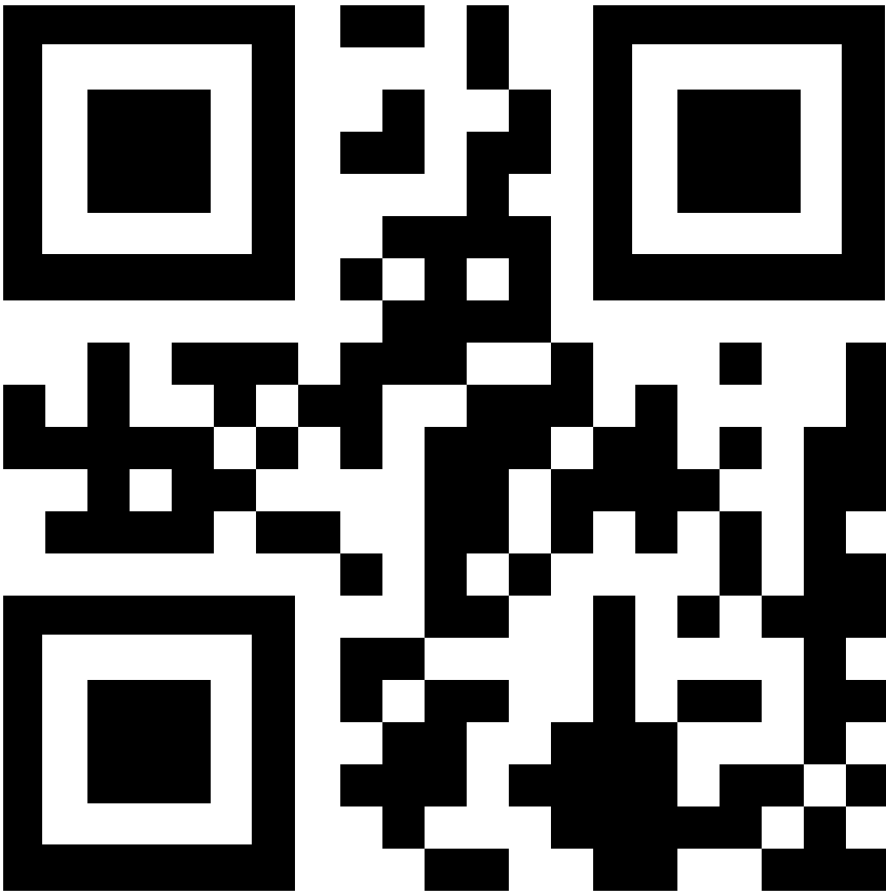
我们发布了一个静态变换。注意我们发布这个静态变换的时候，做了欧拉角的一个转换。主要是为了将坐标系转为与 base_link 相同的姿态。

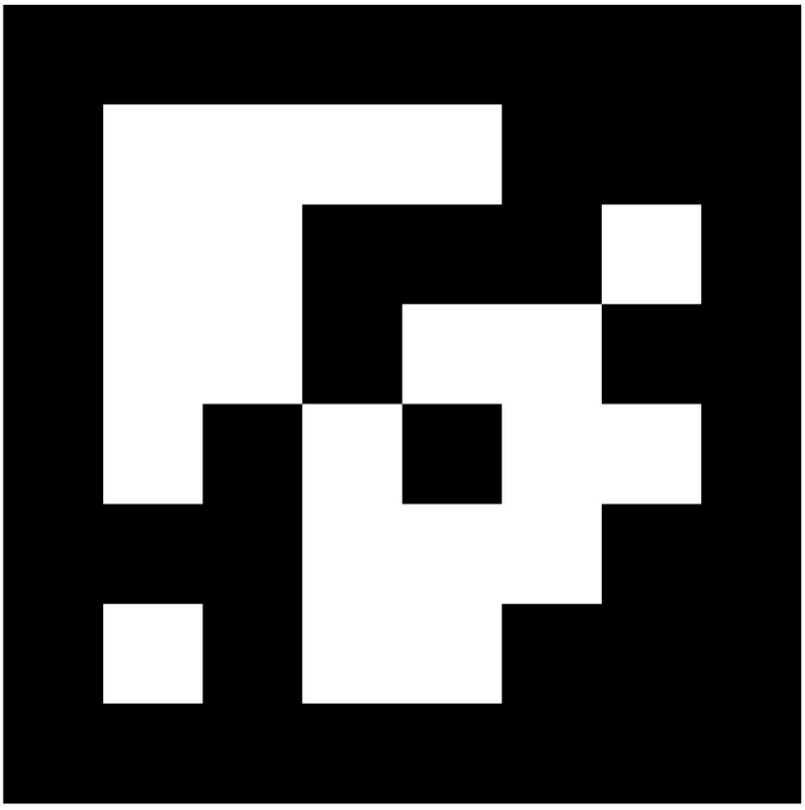
```
53
54      # a static broadcaster to pub the target frame
55      broadcaster = tf2_ros.StaticTransformBroadcaster()
56
57      static_transformStamped = TransformStamped()
58      static_transformStamped.header.stamp = rospy.Time.now()
59      static_transformStamped.header.frame_id = tag_name
60      target_frame = tag_name + "_target"
61      static_transformStamped.child_frame_id = target_frame
62
63      static_transformStamped.transform.translation.y += track_distance*np.tan(np.deg2rad(15))
64      static_transformStamped.transform.translation.z += track_distance
65
66      quat = quaternion_from_euler(np.deg2rad(-90), np.deg2rad(90), 0)
67      static_transformStamped.transform.rotation.x = quat[0]
68      static_transformStamped.transform.rotation.y = quat[1]
69      static_transformStamped.transform.rotation.z = quat[2]
70      static_transformStamped.transform.rotation.w = quat[3]
71      broadcaster.sendTransform(static_transformStamped)
```

最后，监听 tag_xxx_target 在 base_link 坐标系中的位姿，应用 simple_pid 进行速度调节。

```
73      # tf listener
74      tfBuffer = tf2_ros.Buffer()
75      listener = tf2_ros.TransformListener(tfBuffer)
76
77      # rate
78      rate = rospy.Rate(10.0)
79
80      # pid setup
81      pid_x = PID(0.6, 0, 0)
82      pid_y = PID(0.6, 0, 0)
83      pid_z = PID(0.6, 0, 0)
84      pid_a = PID(0.6, 0, 0)
85
86      while not rospy.is_shutdown():
87          try:
88              trans = tfBuffer.lookup_transform(target_frame, (rospy.get_namespace()-
    +'base_link').strip("/"), rospy.Time(), rospy.Duration(0.2))
89          except (tf2_ros.LookupException, tf2_ros.ConnectivityException,
    tf2_ros.ExtrapolationException) as e:
90              rospy.logwarn(e)
91              continue
92
93          vel = Twist()
94          vel.linear.x = pid_x(trans.transform.translation.x)
95          vel.linear.y = pid_y(trans.transform.translation.y)
96          vel.linear.z = pid_z(trans.transform.translation.z)
97          quaternion = [trans.transform.rotation.x, trans.transform.rotation.y,
    trans.transform.rotation.z, trans.transform.rotation.w]
98          vel.angular.z = pid_a(euler_from_quaternion(quaternion)[2])
99          rate.sleep()
```

PID 的调节需要同学们自己进行微调。这里注意 linear.y 和 angular.z 实际上应该有约束，可以适当调整 PID 上限，需要参考 simple-pid 文档自行实现。





ID = 5