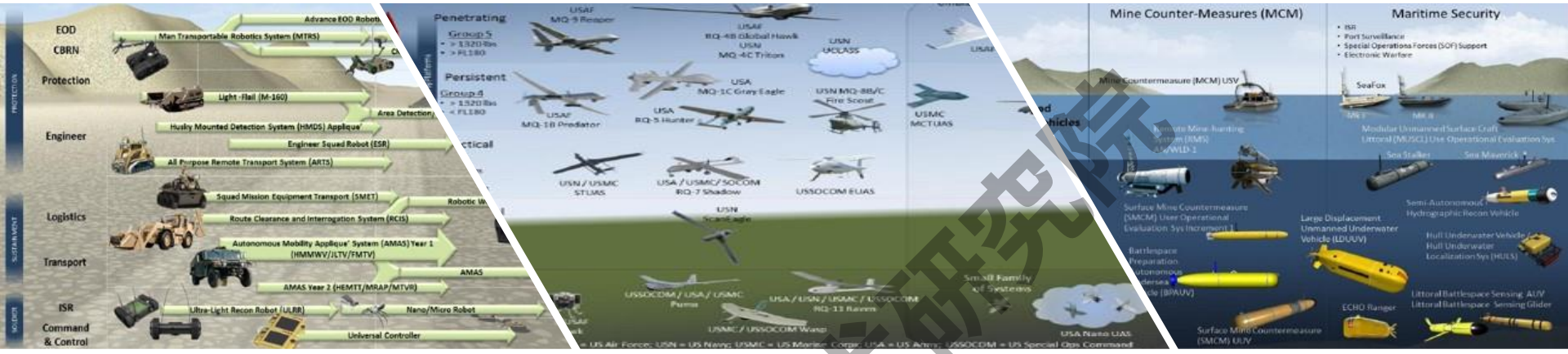


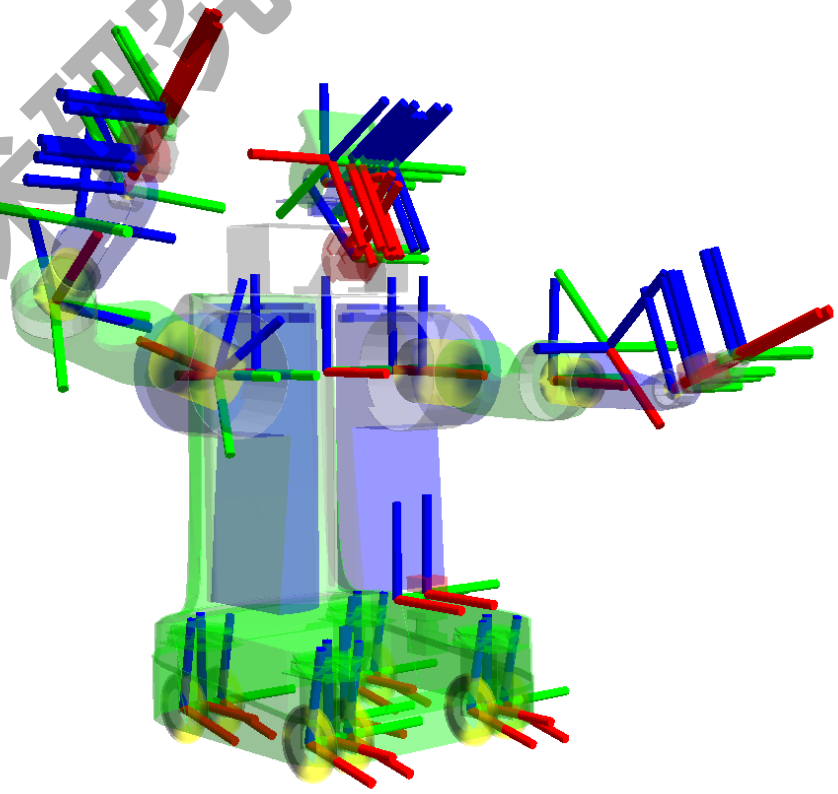
tf 变换和计算机视觉

- ROS 中的 tf 变换
- apriltag 二维码跟踪
- mediapipe 手势识别

2.4 tf 坐标变换

- 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。
- 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）。
- 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。**

右手系：红X，绿Y，蓝Z。



2.4 tf 坐标变换

- **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）。

```
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
```

2.4 小乌龟跟踪

- 安装例程：

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

- 解决代码兼容性问题：

  sudo apt-get install python-is-python3

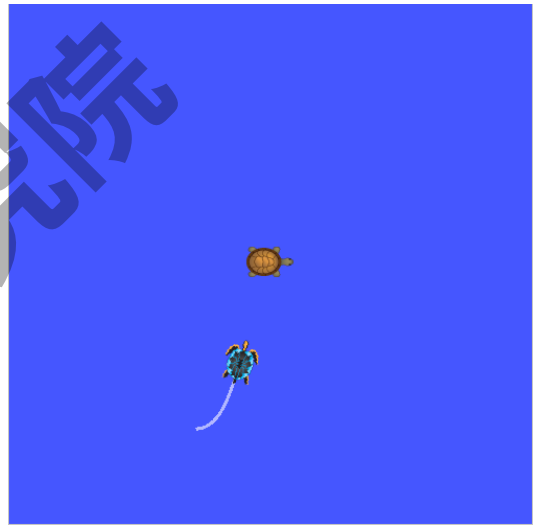
- 运行例程：

  roslaunch turtle_tf2 turtle_tf2_demo.launch

- 键盘控制节点：

  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。
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。



2.4 小乌龟跟踪

- 安装例程：

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

- 解决代码兼容性问题：

  sudo apt-get install python-is-python3

- 运行例程：

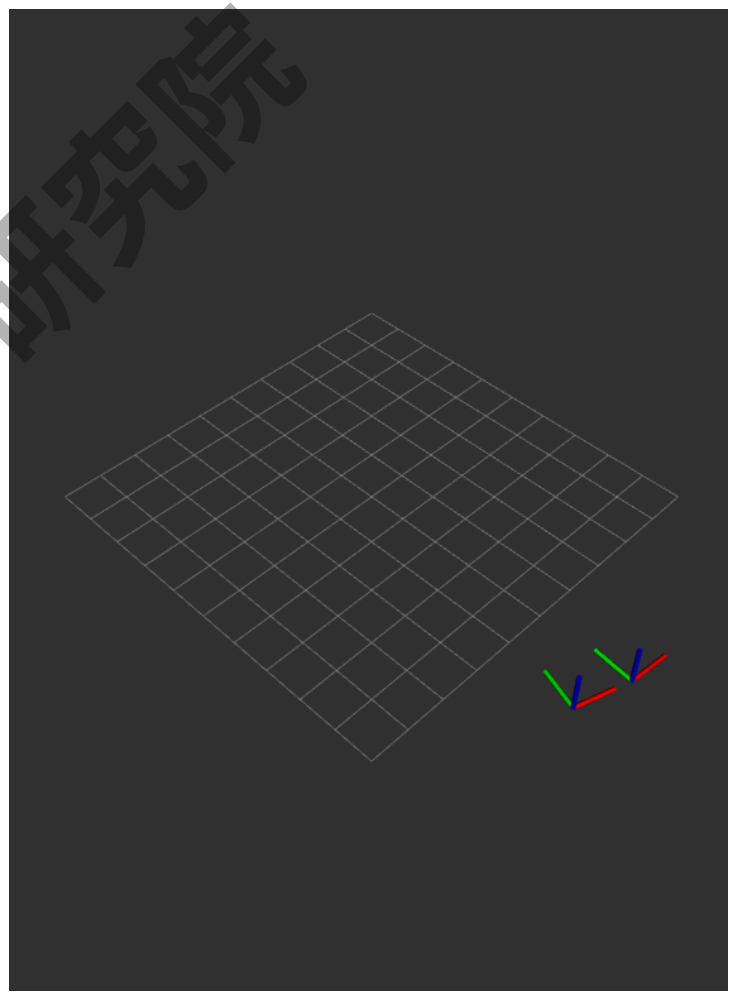
  roslaunch turtle_tf2 turtle_tf2_demo.launch

- 键盘控制节点：

  rosrun turtlesim turtle_teleop_key

- 可视化查看：

  rviz



2.4 小乌龟跟踪

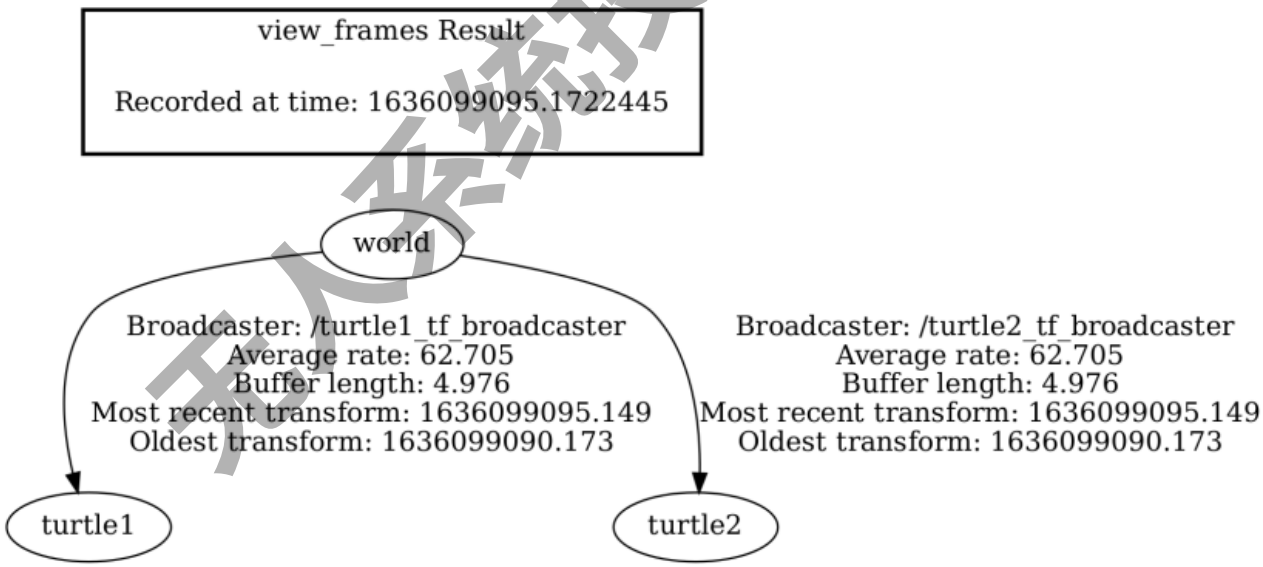
- 查看 tf 树：

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

  rosrun tf2_tools view_frames.py

- 监听坐标变换

  rosrun tf tf_echo turtle1 turtle2



2.4 实现小乌龟跟踪

- 创建一个名为 learning_tf2 的功能包，该功能包依赖 tf2、tf2_ros、roscpp、rospy 和 turtlesim 库：

  catkin_create_pkg learning_tf2 tf2 tf2_ros roscpp rospy turtlesim

- 启动 ROS master 和一个小乌龟仿真节点：

  roscore

  rosrun turtlesim turtlesim_node

- 调用服务生成一只名为 turtle2 的新的小乌龟：

  rosservice call spawn 2 2 0.1 "turtle2"

2.4 实现小乌龟跟踪

- 编写节点订阅 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. 转换 pose 信息格式；
  8. 发布。

- 完成后运行：
  rosrun learning_tf2 turtle1_tf2_broadcaster.py

- 编写节点广播turtle2相对world的坐标系信息。

- 完成后运行：
  rosrun learning_tf2 turtle2_tf2_broadcaster.py

```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()
```

2.4 实现小乌龟跟踪

- 编写节点
  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

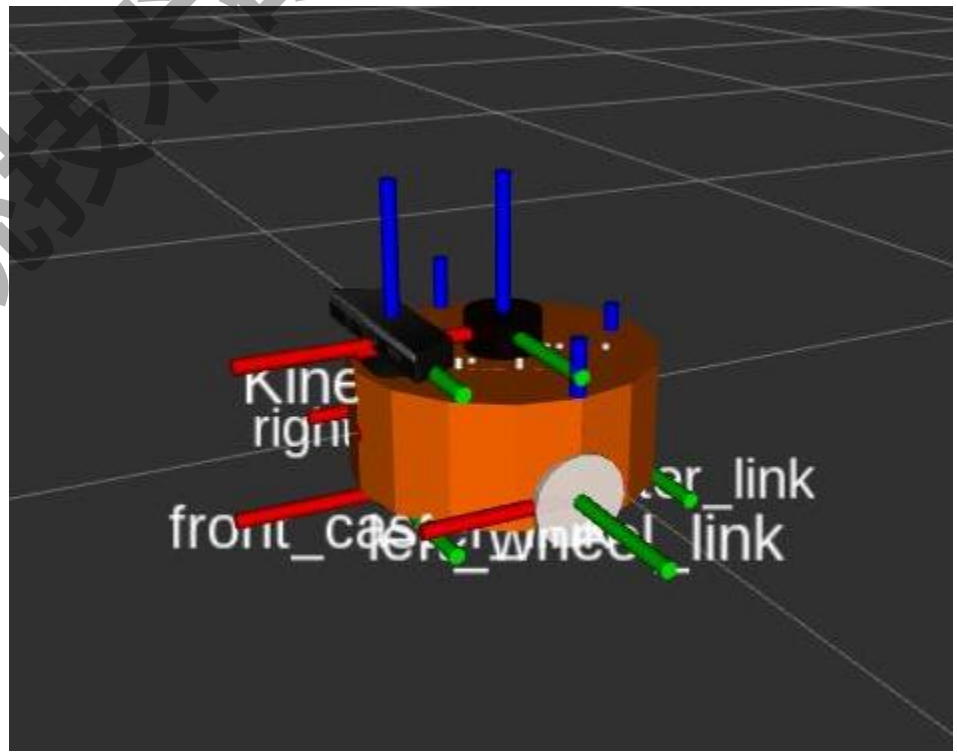
- 运行小乌龟键盘控制节点：
  rosrun turtlesim turtle_teleop_key

```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)
```

2.5 URDF

URDF 全称为 Unified Robot Description Format，中文可以翻译为“统一机器人描述格式”。根据该格式的设计者所言，设计这一格式的目的在于提供一种尽可能通用（as general as possible）的机器人描述规范。URDF创造的机器人模型包含的内容有：

- 连杆 link
- 关节 joint
- 运动学参数 axis
- 动力学参数 dynamics
- 可视化模型 visual
- 碰撞检测模型 collision

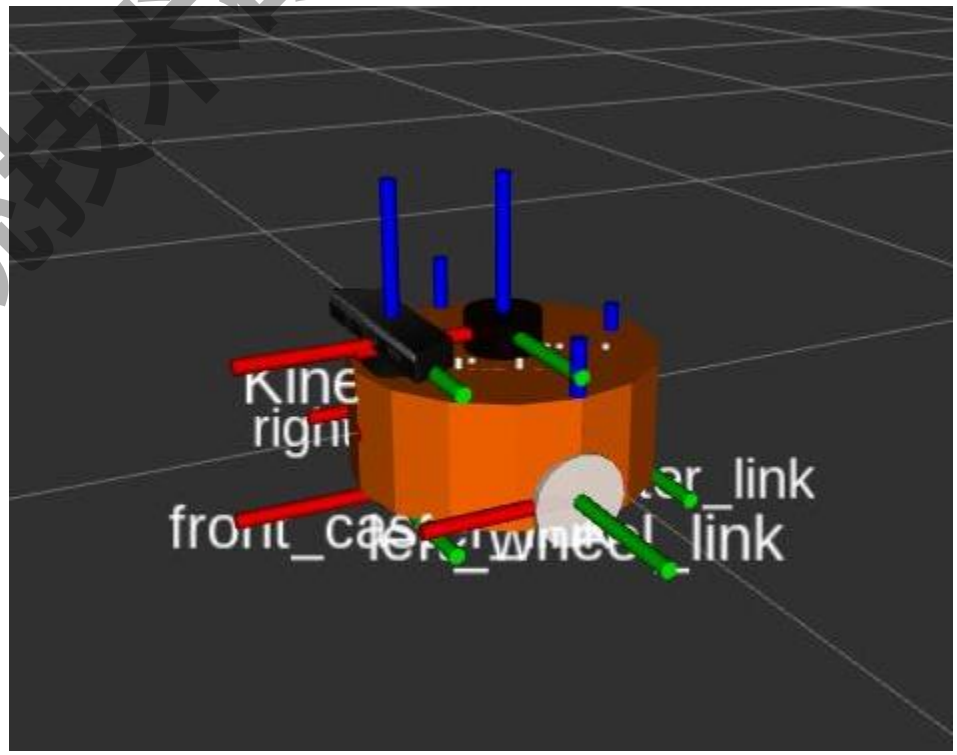


2.5 URDF

xacro 是 URDF 的升级版，是建立在 URDF 基础上的扩充，URDF 能做到的，xacro 都能实现。而URDF做不到宏定义、文件包含、条件判断等等的功能，xacro也能做到。

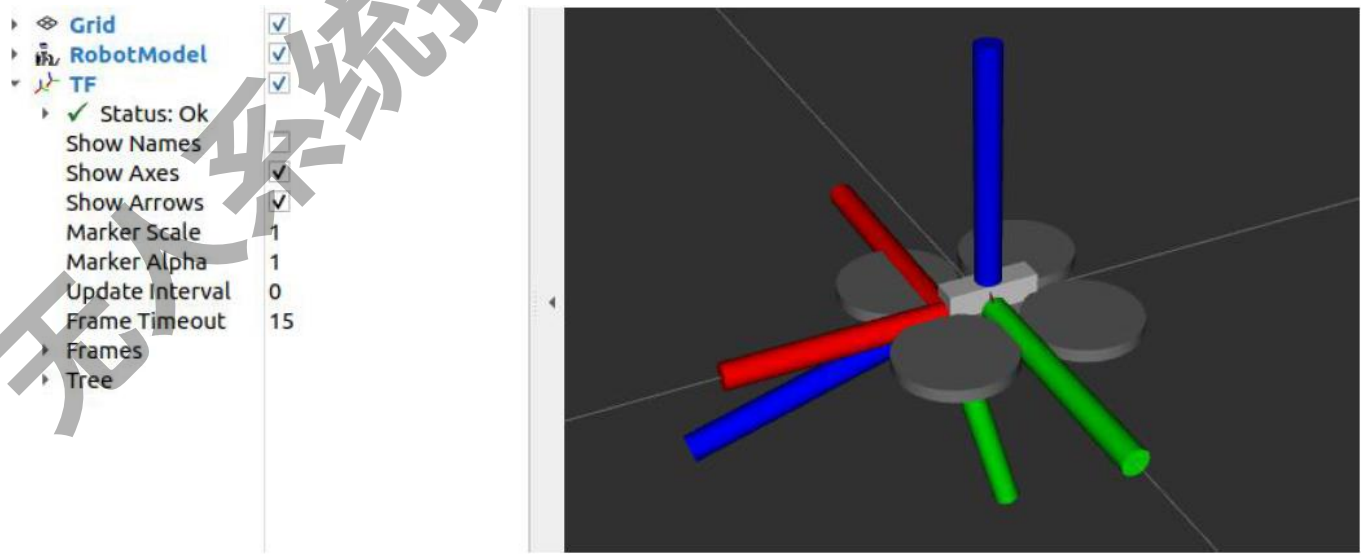
模型的描述对象是一个小车：

- 使用 URDF 文件来描述，需要列出4个轮子的参数。
- 使用 xacro，只需要定一个描述轮子的宏模板，再根据传入的不同参数分别进行4次调用。



- 启动无人机的描述文件（坐标变换）
  roslaunch rmtt_description rmtt_description.launch

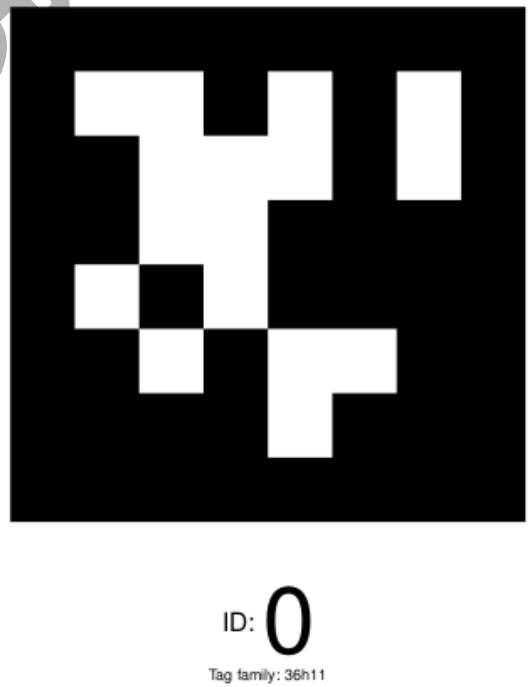
打开rviz添加RobotModel和TF，将Global Options->fixed frame更改为base_link，可以看到rviz的显示区域如下图所示：



2.6 apriltag 二维码检测

apriltag 二维码检测主要包含三个步骤：

- 第一步是如何根据梯度检测出图像中的各种边缘。
- 第二步即如何在边缘图像中找出需要的四边形图案并进行筛选。
- 第三步是如何进行二维码编码和二维码解码。得到编码以后再与已知库内的编码进行匹配，确定解码出的二维码是否为正确。



apriltag 二维码检测

- 计算图像像素的梯度

100	90	80	80
80	70	90	100
90	60	100	80
90	100	110	80

→

10	20	10	0

$$\nabla I_x(x,y) = I(x-1,y) - I(x+1,y)$$

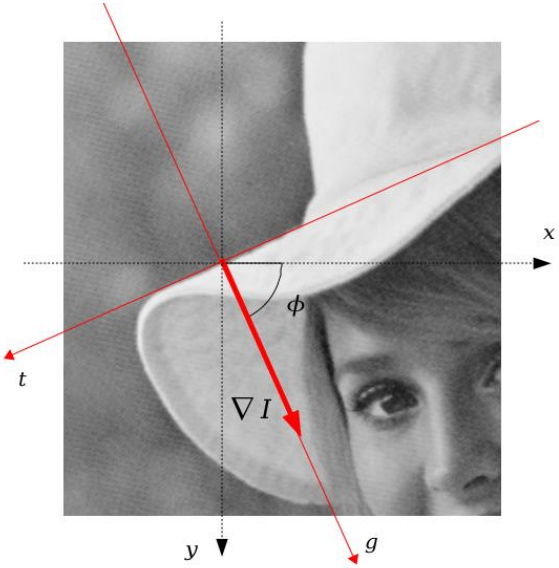
$$\nabla I_y(x,y) = I(x,y-1) - I(x,y+1)$$

$$\star \begin{pmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{pmatrix} =$$

Approximation of $I_x = \frac{\partial I}{\partial x}$.

apriltag 二维码检测

- 计算图像像素的梯度



$$\nabla I = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)^T$$

$$\star \begin{pmatrix} -1 & 0 & 1 \\ -1 & \mathbf{0} & 1 \\ -1 & 0 & 1 \end{pmatrix} =$$

Approximation of $I_x = \frac{\partial I}{\partial x}$.

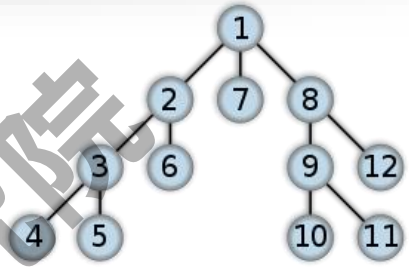
apriltag 二维码检测

- 计算梯度的强度和方向

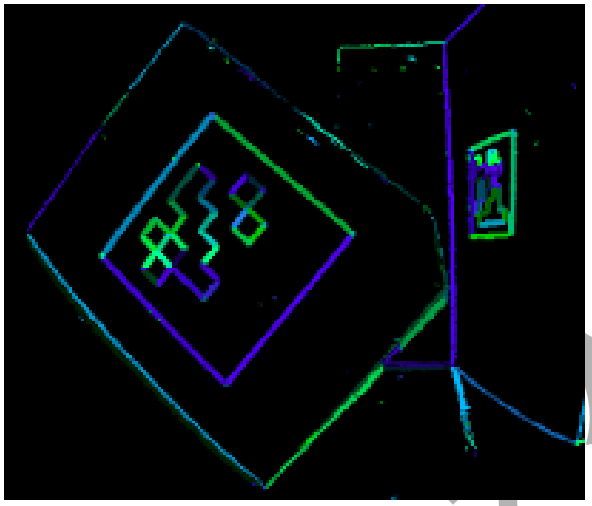
$$\nabla I(x,y) = \sqrt{I_x(x,y)^2 + I_y(x,y)^2} \qquad \Phi(x,y) = arctan\frac{I_y(x,y)}{I_x(x,y)}$$

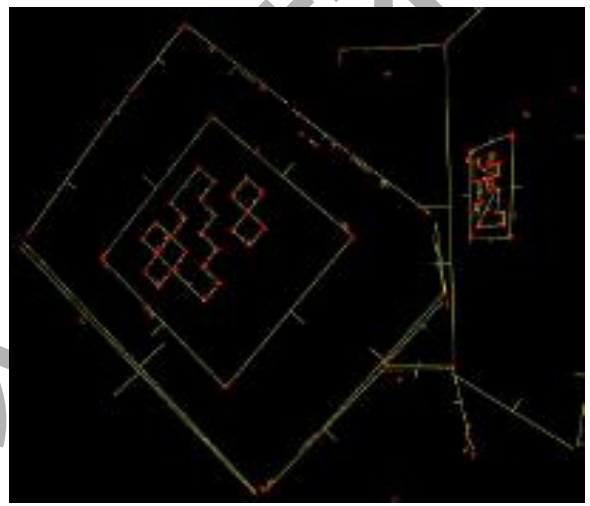
a) 原始图像　　b) 梯度强度图　　c) 梯度方向图

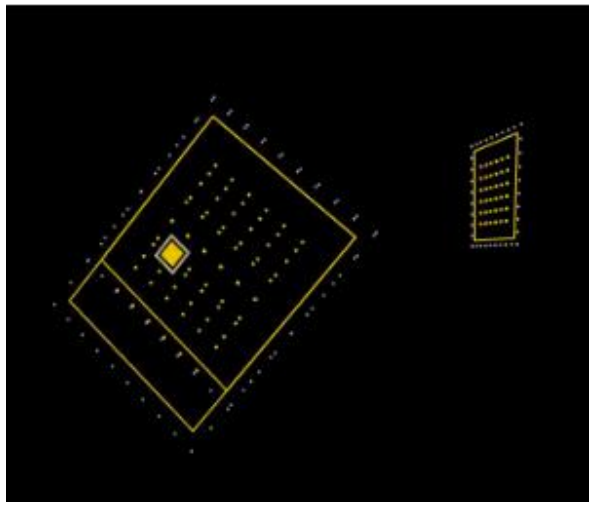
2.6 apriltag 二维码检测



- 利用像素梯度强度和方向信息进行聚类。
- 利用加权最小二乘进行线段拟合。
- 利用深度优先进行四边形检测，深度为 4 的时候，最后一条边与第一条边构成一个闭环。

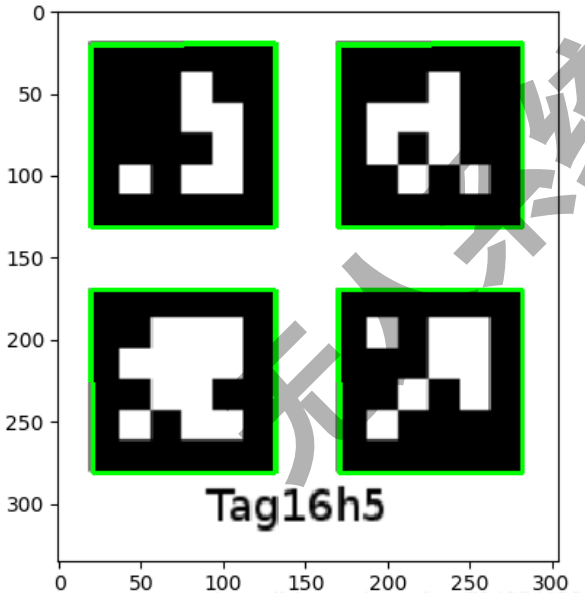


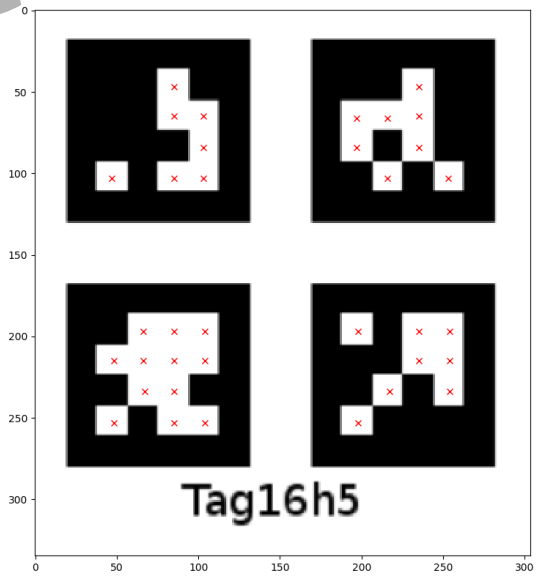




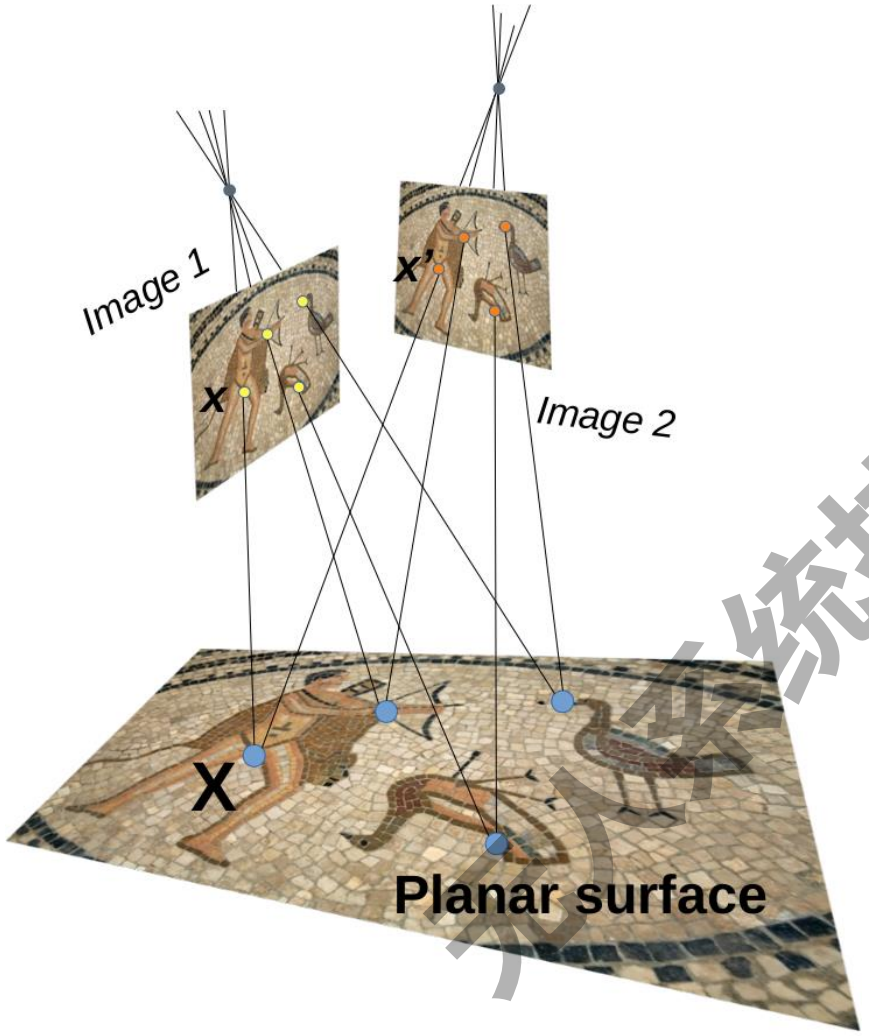
2.6 apriltag 二维码检测

- 确定黑色白色分界像素强度阈值。
- 对点阵进行二值编码，需要进行三次旋转，获得4组编码。
- 求解与编码库之间的汉明距离（Hamming Distance）。
- 设定阈值，确定编码。





2.6 apriltag 二维码检测



- 单应性（homography）矩阵是描述两个相同平面不同视角下的转换关系：$x' = Hx$

$$\lambda \begin{pmatrix} x_1' \\ x_2' \\ 1 \end{pmatrix} = \begin{pmatrix} H_1^1 & H_1^2 & H_1^3 \\ H_2^1 & H_2^2 & H_2^3 \\ H_3^1 & H_3^2 & H_3^3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix}$$

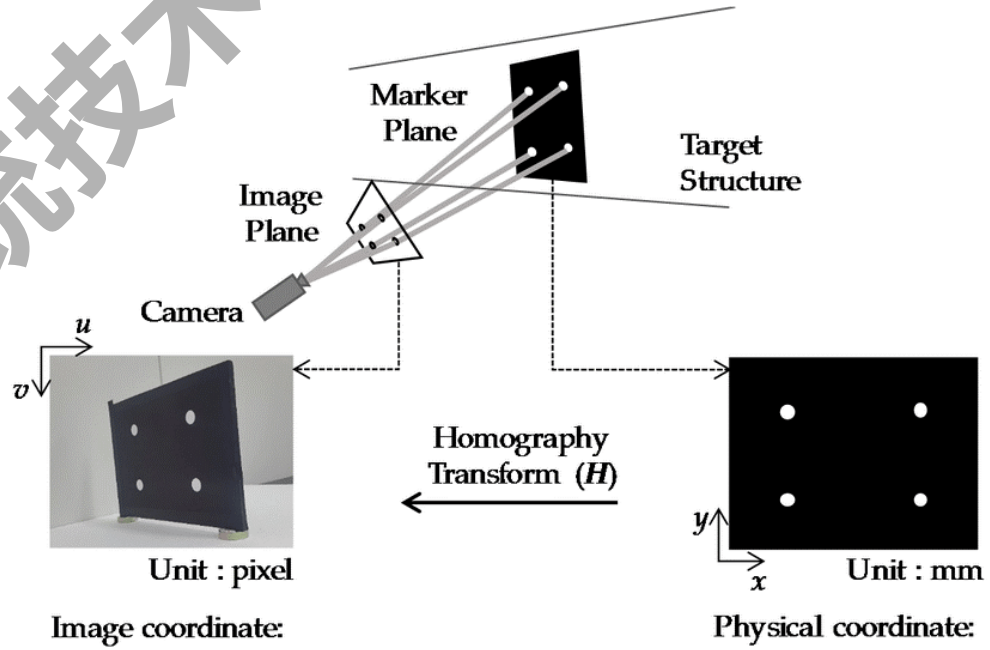
- 一般由一个3×3矩阵表示
- 至少需要四对匹配特征点进行求解
- 实际需要更多匹配特征点

2.6 apriltag 二维码检测

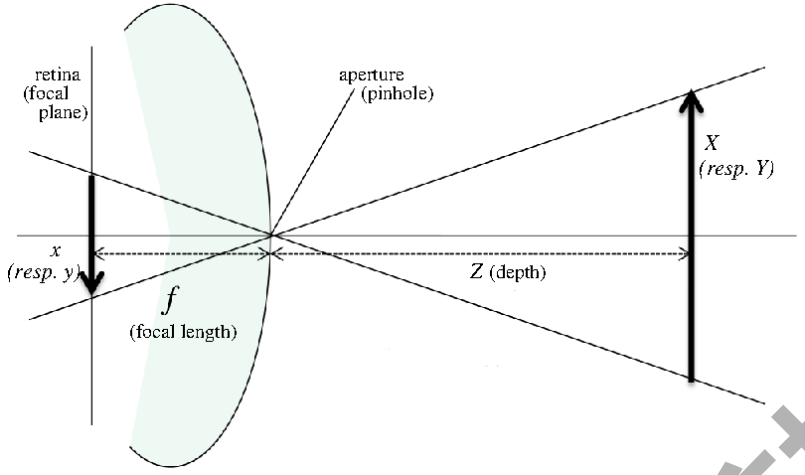
- 单应性矩阵（3*3）。
- 确定：a.相机的焦距；b.标签的物理尺寸。
- 计算旋转、平移矩阵。

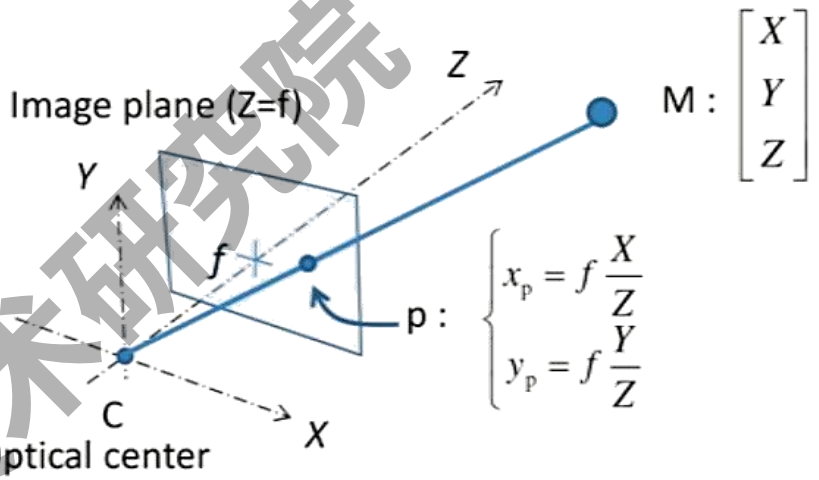
$$\begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} = sPE$$

$$= s \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R_{00} & R_{01} & T_x \\ R_{10} & R_{11} & T_y \\ R_{20} & R_{21} & T_z \\ 0 & 0 & 1 \end{bmatrix}$$



2.6 相机模型



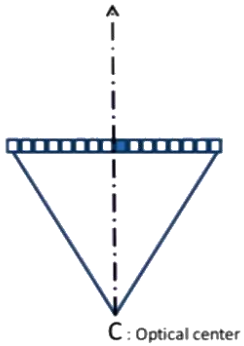


$$M:\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \qquad p:\begin{cases} x_p = f\dfrac{X}{Z} \\ y_p = f\dfrac{Y}{Z} \end{cases}$$

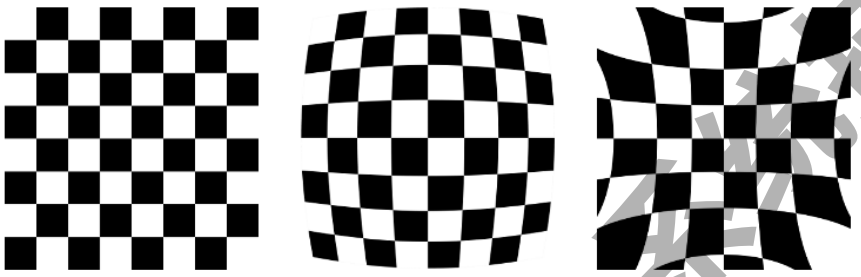
- 3D 空间点向摄像机2D平面的投影将损失空间信息

2.6 相机模型

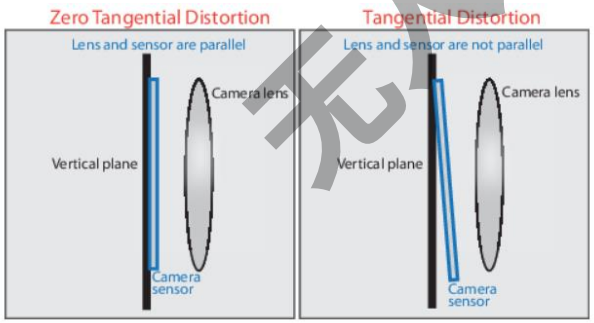
相机的内参



$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f/dx & 0 & u_0 \\ 0 & f/dy & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

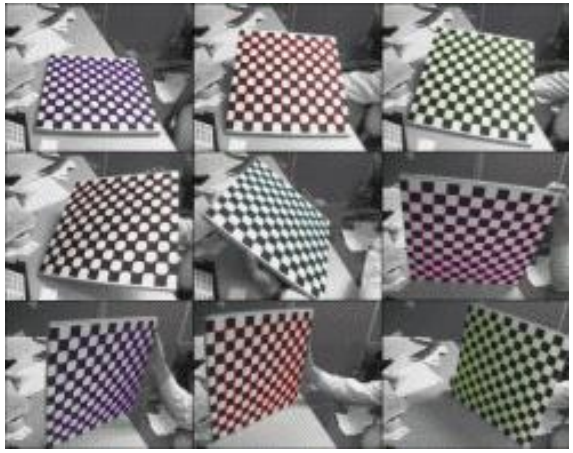


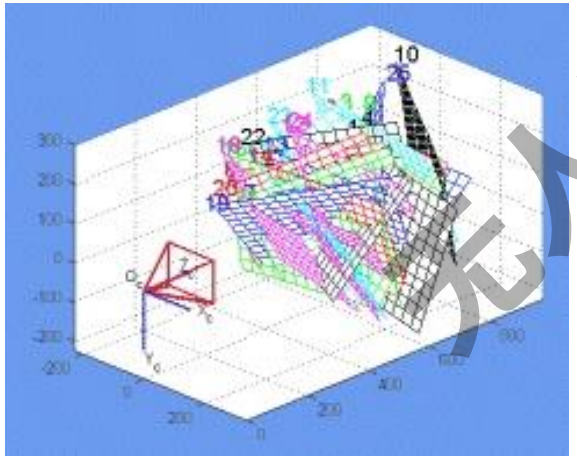
$$\begin{cases} x_d = x(1 + k_1 r^2 + k_2 r^4 + +k_3 r^6) \\ y_d = y(1 + k_1 r^2 + k_2 r^4 + +k_3 r^6) \end{cases}$$



$$\begin{cases} x_d = x + [2p_1 xy + p_2(r^2 + 2x^2)] \\ y_d = x + [2p_2 xy + p_1(r^2 + 2y^2)] \end{cases}$$

标定工具箱





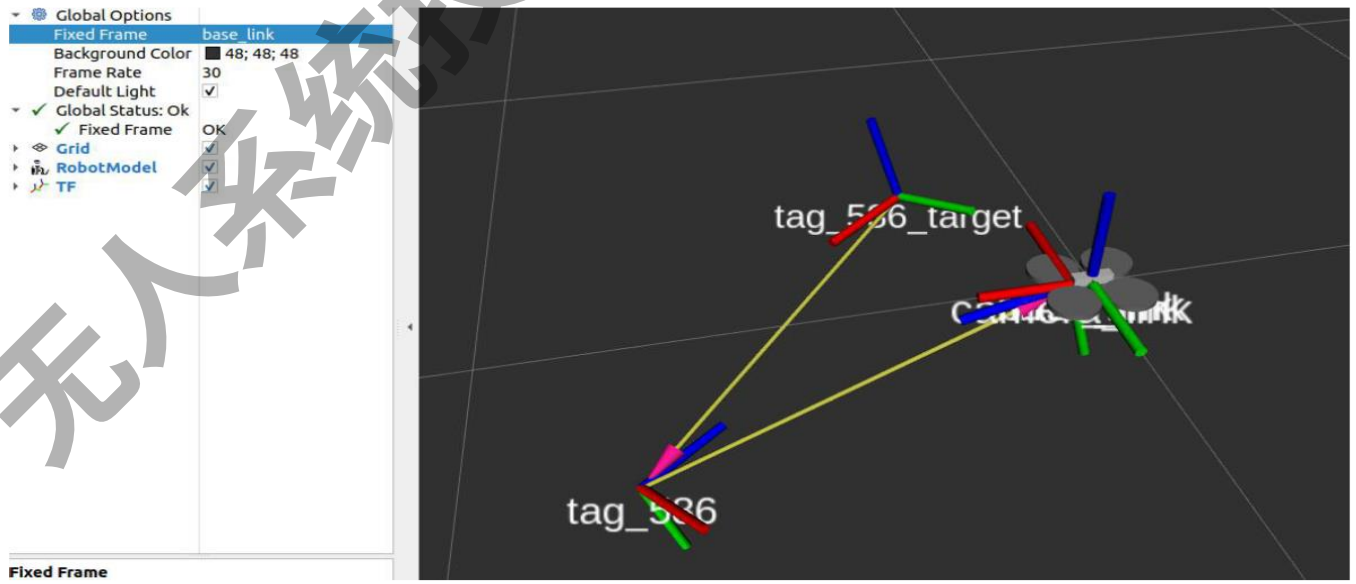
```
cameraParams =

  cameraParameters (具有属性):

    Camera Intrinsics 相机内参
                IntrinsicMatrix: [3x3 double]内参矩阵
                    FocalLength: [6.7831e+03 6.7899e+03]焦距
                  PrincipalPoint: [800.5859 859.4862]
                            Skew: 0

    Lens Distortion 镜头畸变
                RadialDistortion: [1.0269 -63.0788]径向畸变
            TangentialDistortion: [0 0] 切向畸变

    Camera Extrinsics相机外参
                RotationMatrices: [3x3x20 double]旋转矩阵
                TranslationVectors: [20x3 double]平移矩阵

    Accuracy of Estimation估计准确度
            MeanReprojectionError: 0.2182
                ReprojectionErrors: [35x2x20 double]
                ReprojectedPoints: [35x2x20 double]
```

2.6 apriltag 二维码跟踪

- 连接无人机到计算机，并启动无人机驱动：
  roslaunch rmtt_driver rmtt_bringup.launch
- 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。
- 启动二维码检测程序：
  roslaunch rmtt_apriltag detection.launch

2.6 apriltag 二维码跟踪

- 启动无人机跟踪节点：
  roslaunch rmtt_tracker rmtt_tag_tracker.launch
- 启动无人机遥控节点：
  roslaunch rmtt_teleop rmtt_teleop_key.launch
- 代码讲解



2.7 mediapipe 手势识别

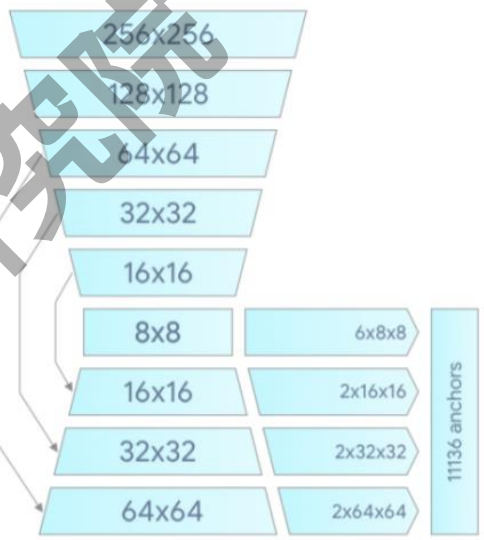
- MediaPipe是一款由Google开发并开源的数据流处理机器学习应用开发框架。
- MediaPipe是跨平台的，可以运行在嵌入式平台(树莓派等)，移动设备(iOS和Android)，工作站和服务器上，并支持移动端GPU加速。
- 使用MediaPipe，可以将机器学习任务构建为一个图形的模块表示的数据流管道，可以包括推理模型和流媒体处理功能。

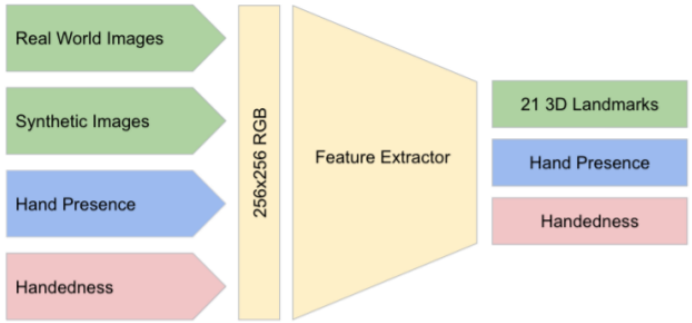
MediaPipe

	Android	iOS	C++	Python
Face Detection	✅	✅	✅	✅
Face Mesh	✅	✅	✅	✅
Iris	✅	✅	✅	
Hands	✅	✅	✅	✅
Pose	✅	✅	✅	✅
Holistic	✅	✅	✅	✅
Selfie Segmentation	✅	✅	✅	✅
Hair Segmentation	✅		✅	
Object Detection	✅	✅	✅	
Box Tracking	✅	✅	✅	
Instant Motion Tracking	✅			
Objectron	✅		✅	✅

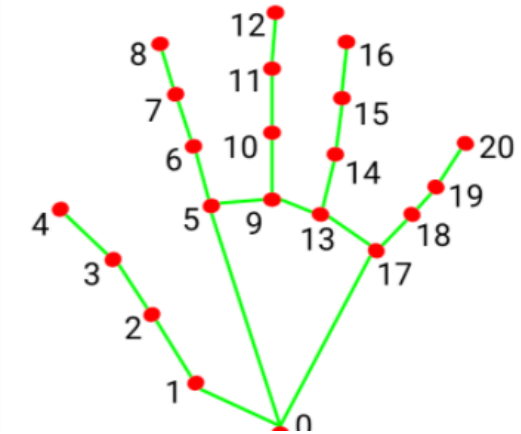
2.7 mediapipe 手势识别

- 使用轻量级神经网络预测手掌位置
- 只在手掌丢失时进行检测
- 可识别多种不同手掌大小
- 能识别被遮挡的手部
- 输出21个关节点





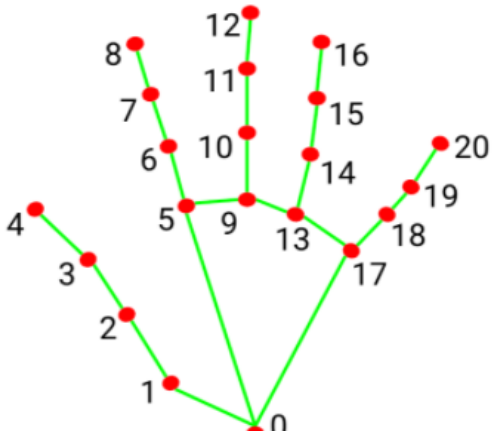
2.7 mediapipe 手势识别



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

- 通过回归对 21 个 3D 手关节坐标执行精确的关键点定位，即直接预测坐标。
- 手动标注了约 3万张包含 21 个3D坐标的真实图像。
- 谷歌还在各种背景下渲染出手部的优质合成模型，并将其映射为相应的 3D 坐标。
- 我们可以通过使用关键点坐标之间的位置信息来定义手势。

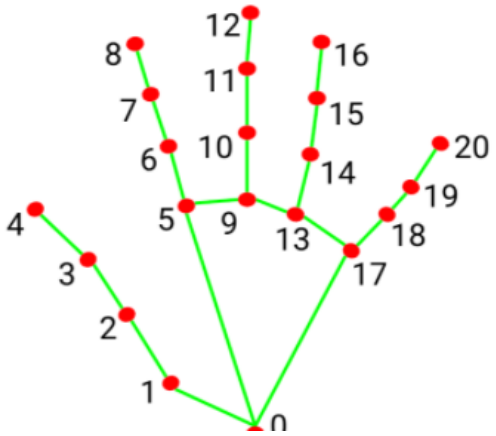
2.7 mediapipe 手势识别



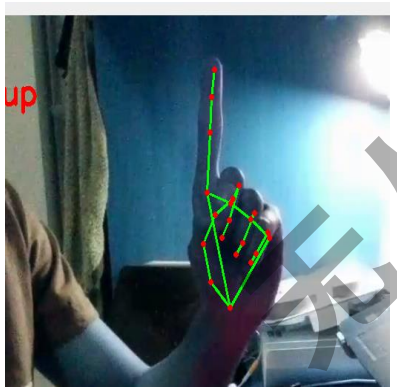
0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

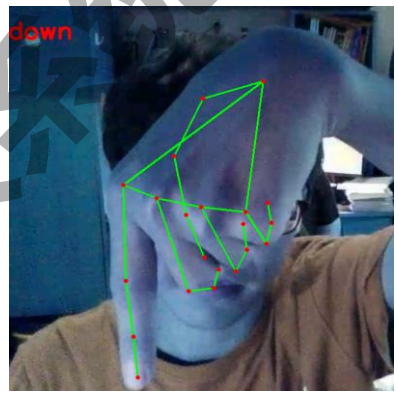
- 比如，我们将大拇指的关键点0到关键点2连成一个向量，关键点3到关键点4连成一个向量，通过计算两个向量之间的夹角，来定义大拇指的弯曲与伸展。
- 给定一个阈值，当向量夹角大于阈值时，我们认为大拇指是弯曲的，反之，则认为其实伸直的。
- 我们可以再次通过x，y坐标的大小关系，判断手指的指向。

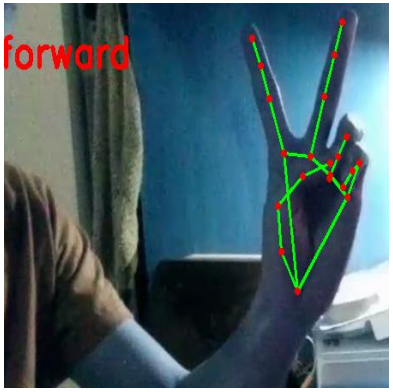
2.7 mediapipe 手势识别



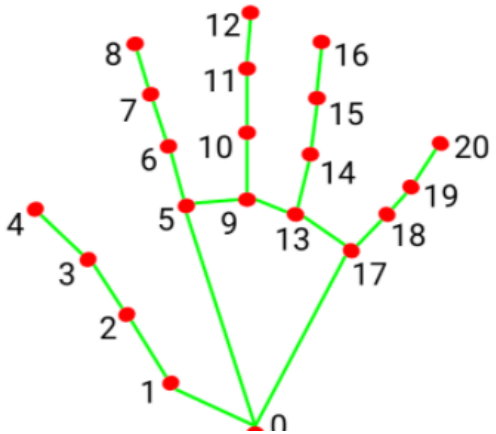
0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP



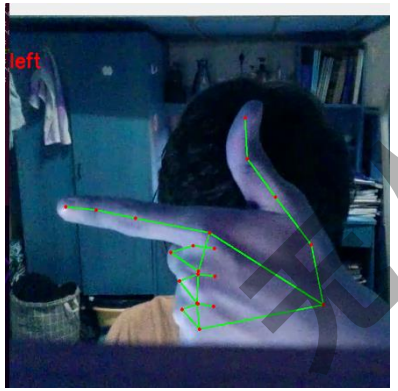


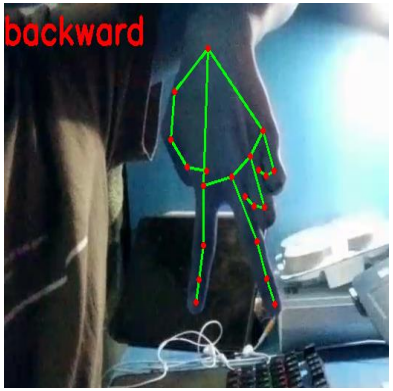


2.7 mediapipe 手势识别

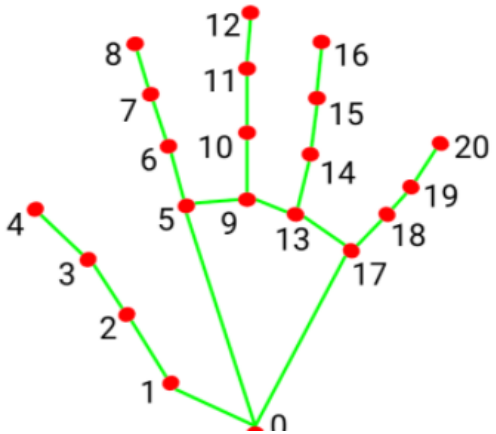


0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

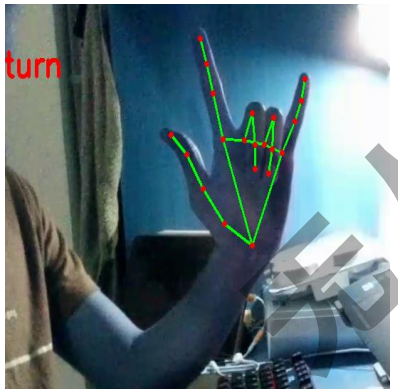


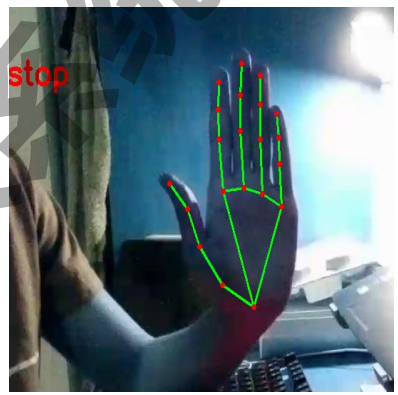


2.7 mediapipe 手势识别



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP





课 程 结 束 ， 欢 迎 提 问

THANK YOU FOR WATCHING