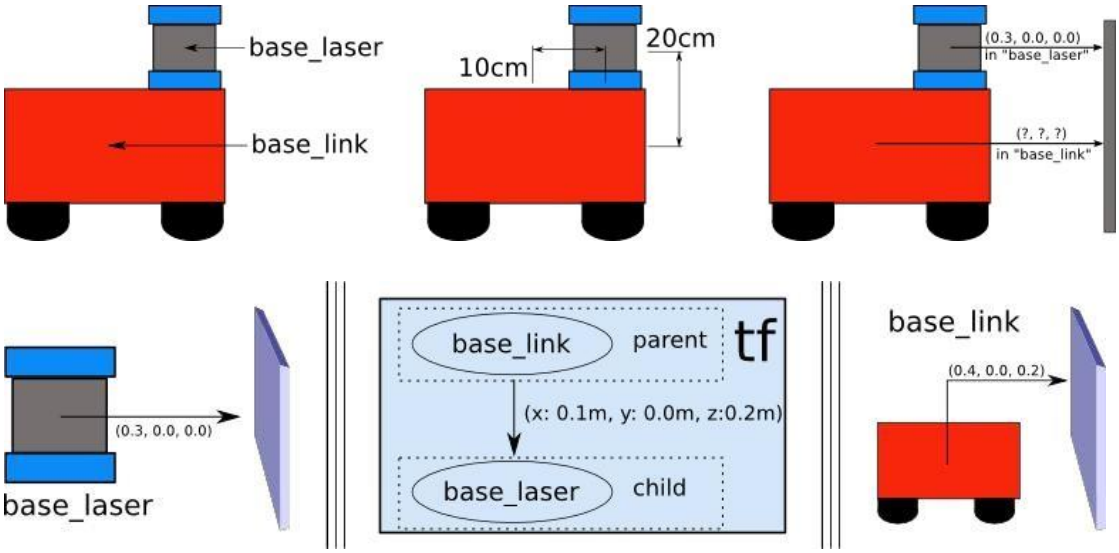
一、坐标变换简介

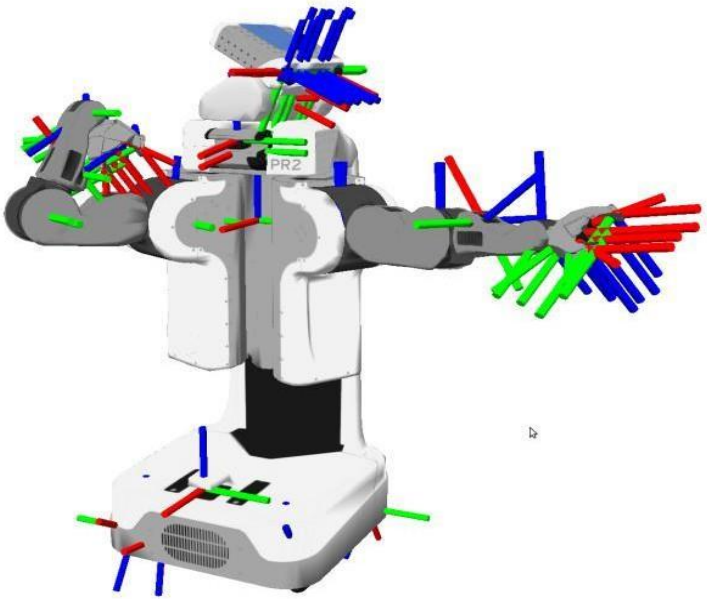
场景

机器人系统上， 有多个传感器， 如激光雷达、 摄像头等， 有的传感器是可以感知机器人周边的物体方位(以坐标的方式表示物体与传感器的横向距离、 纵向距离、 垂直高度等信息)的， 以协助机器人定位障碍物， 那么可以直接将物体相对该传感器的方位信息， 等价于物体相对于机器人系统或机器人其它组件的方位信息吗？ 显然是不行的， 这中间需要一个转换过程。具体案例如下:

场景 1： 现有一移动式机器人底盘， 在底盘上安装了一雷达， 雷达相对于底盘的偏移量已知， 现雷达检测到一障碍物信息， 获取到坐标分别为(x, y, z)， 该坐标是以雷达为参考系的， 如何将这个坐标转换成以小车为参考系的坐标呢？



场景 2:现有一带机械臂的机器人(比如:PR2)需要夹取目标物， 当前机器人头部摄像头可以探测到目标物的坐标(x, y, z)， 不过该坐标是以摄像头为参考系的， 而实际操作目标物的是机械臂的夹具， 当前我们需要将该坐标转换成相对于机械臂夹具的坐标，这个过程如何实现？



当然， 根据我们高中学习的知识， 在明确了不同坐标系之间的的相对关系， 就可以实现任何坐标点在不同坐标系之间的转换， 但是该计算实现是较为常用的， 且算法也有点复杂，因此在 ROS 中直接封装了相关的模块: 坐标变换(tf)。

概念

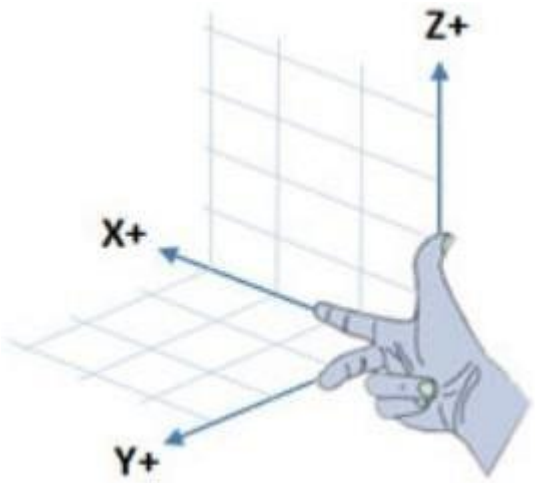
tf(TransForm Frame)是指坐标变换， 它允许用户随时间跟踪多个坐标系。 它在时间缓冲的树结构中维护坐标帧之间的关系， 并让用户在任何所需的时间点在任意两个坐标帧之间变换点、 向量等。 在 ROS 中已经提供了同名的库实现，并且随着 ROS 的迭代， 该库升级为了 tf2， 也即第二代坐标变换库。 本阶段课程主要内容也是以 tf2 为主。

完整的坐标变换实现由坐标变换广播方和坐标变换监听方两部分组成。 每个坐标变换广播方一般会发布一组坐标系相对关系， 而坐标变换监听方则会将多组坐标系相对关系融合为一棵坐标树（该坐标树有且仅有一个根坐标系） ， 并可以实现任意坐标系之间或坐标点与坐标系的变换。

另外需要说明的是， ROS 中的坐标变换是基于右手坐标系的。 右手坐标系的具体规则如下图所示： 将右手处于坐标系原点， 大拇指、 食指与中指互成直角， 食指指向的是 x 轴正方向， 中指指向的是 y 轴正方向， 大拇指指向的是 z 轴正方向。

作用

在 ROS 中用于实现不同坐标系之间的点或向量的转换。



案例安装以及运行

关于坐标变换的实现有一个经典的“乌龟跟随”案例， 在学习坐标变换的具体知识点之前， 建议先安装并运行此案例。

bash

```
sudo apt-get install ros-humble-turtle-tf2-py ros-humble-tf2-tools
ros-humble-tf-transformations
```

此外， 还需要安装一个名为 transforms3d 的 Python 包， 它为 tf_transformations 包提供四

元数和欧拉角变
换功能，
安装命令如下：

bash
```
pip3 install transforms3d
```

执行

启动两个终端， 终端 1 输入如下命令：

bash
```
ros2 launch turtle_tf2_py turtle_tf2_demo.launch.py
```

该命令会启动 turtlesim_node 节点， turtlesim_node 节点中自带一只小乌龟 turtle1， 除此之外还会新生成一只乌龟
turtle2， turtle2 会运行至 turtle1 的位置。

终端 2 输入如下命令：
bash
```
ros2 run turtlesim turtle_teleop_key
```

该终端下可以通过键盘控制 turtle1 运动， 并且 turtle2 会跟随 turtle1 运动

二、rviz2 查看坐标系关系

2.0 本教程文件结构

```
ros2_ws/
├── src/
│   ├── tf_broadcaster_py/
│   │   ├── setup.py
│   │   └── tf_broadcaster_py/
│   │       └── tf_broadcaster.py
│   ├── tf_listener_py/
│   │   ├── setup.py
│   │   └── tf_listener_py/
│   │       └── tf_listener.py
...
```

2．1 创建 Python 脚本发布 TF

1. 在工作空间的 src 目录下创建一个新包：

bash

```
cd ~/《你们自己工作空间名字》/src
ros2 pkg create --build-type ament_python tf_broadcaster_py
```

2. 进入 tf_broadcaster_py 包目录，并编辑 setup.py 文件，确保 entry_points 部分包含我们将要创建的 Python 节点：

python

```
# setup.py
# 导入 setuptools 中的 setup 函数，用于配置和构建 Python 包
from setuptools import setup

# 定义包的名称
package_name = 'tf_broadcaster_py'

# 配置包的相关信息
setup(
    # 包名
    name=package_name,
    # 版本号
    version='0.0.0',
    # 需要包含的 Python 包列表
    packages=[package_name],

    # 定义要安装的数据文件
    data_files=[
        # 安装 ROS 2 包索引文件
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        # 安装 package.xml 文件
        ('share/' + package_name, ['package.xml']),
    ],

    # 声明包的依赖项
    install_requires=['setuptools'],
    # 指定包是否可以安全地以 zip 格式运行
    zip_safe=True,

    # 维护者信息
    maintainer='your_name',
    maintainer_email='your_email@example.com',
    # 包的描述
    description='Python TF broadcaster example',
    # 许可证信息
    license='Apache License 2.0',
```

```
    # 测试所需的依赖项
    tests_require=['pytest'],

    # 定义可执行脚本入口点
    entry_points={ 'console_s
        cripts': [
            # 创建一个名为 tf_broadcaster_node 的可执行文件，指向
tf_broadcaster_py.tf_broadcaster 模块的 main 函数
            'tf_broadcaster_node =
tf_broadcaster_py.tf_broadcaster:main',
        ],
    },
)
```

3. 在…workspace/src/tf_broadcaster_py/ 这个文件夹下创建一个 Python 文件 tf_broadcaster.py：

bash

```
 cd （你们工作空间的文件地址自己写）/src/tf_broadcaster_py/
touch tf_broadcaster.py
chmod +x tf_broadcaster.py
```

4. 编辑 tf_broadcaster.py 文件，编写发布 TF 的代码：（多写注释）

python

```
# tf_broadcaster_py/tf_broadcaster.py

# 导入必要的库
import rclpy  # ROS2 Python 客户端库
from rclpy.node import Node  # ROS2 节点基类
from geometry_msgs.msg import TransformStamped  # 坐标变换消息类型
import tf2_ros  # TF2 变换库
import math  # 数学函数库
from rclpy.time import Time  # ROS2 时间类型

# 定义 TF 广播器节点类
class TFBroadcaster(Node):

    def __init__(self):
        # 初始化节点，设置节点名称
        super().__init__('tf_broadcaster_node')
        # 创建 TF 广播器对象
        self.br = tf2_ros.TransformBroadcaster(self)
        # 创建定时器，每 0.1 秒调用一次 broadcast_callback
        self.timer = self.create_timer(0.1, self.broadcast_callback)
```

```
    def broadcast_callback(self):
        # 创建 TransformStamped 消息对象
        t = TransformStamped()

        # 设置消息头部信息
        t.header.stamp = self.get_clock().now().to_msg()  # 时间戳
        t.header.frame_id = 'world'  # 父坐标系名称
        t.child_frame_id = 'robot'   # 子坐标系名称

        # 设置平移变换
        t.transform.translation.x = 1.0 #  X 轴平移
        t.transform.translation.y = 2.0 #  Y 轴平移
        t.transform.translation.z = 0.0  # Z 轴平移

        # 设置旋转变换（使用四元数）
        # 根据当前时间计算旋转角度，实现连续旋转效果
        angle = self.get_clock().now().nanoseconds * 1e-9
        t.transform.rotation.x = 0.0
        t.transform.rotation.y = 0.0
        t.transform.rotation.z = math.sin(angle / 2.0)  # 绕 Z 轴旋转
的正弦分量
        t.transform.rotation.w = math.cos(angle / 2.0)  # 绕 Z 轴旋转
的余弦分量

        # 发布 TF 变换
        self.br.sendTransform(t)

# 主函数
def main(args=None):
    # 初始化 ROS2
    rclpy.init(args=args)
    # 创建节点实例
    node = TFBroadcaster()
    # 运行节点
    rclpy.spin(node)
    # 销毁节点
    node.destroy_node()
    # 关闭 ROS2
    rclpy.shutdown()

# 如果直接运行此脚本，则执行 main 函数
if __name__ == '__main__':
    main()
```

在该代码中，我们创建了一个 TF 广播器，它会每隔 0.1 秒发布一次坐标变换。world 是父坐标系，robot 是子坐标系，机器人围绕 Z 轴旋转。

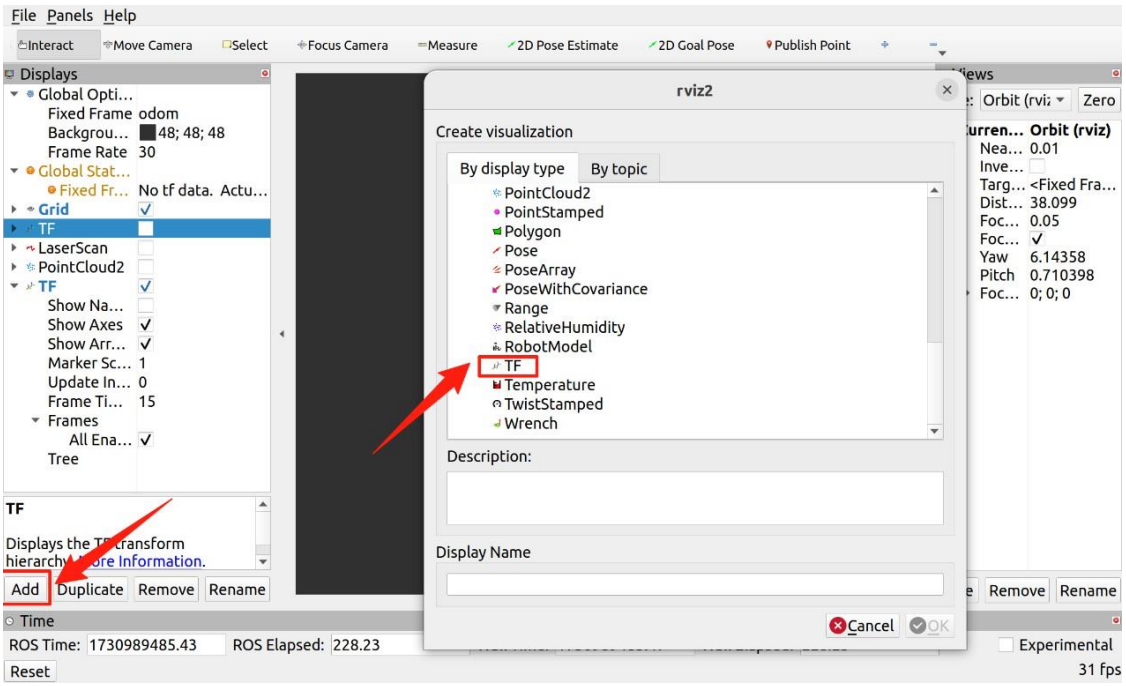
2.2. 配置 RViz 查看 TF

1. 启动 RViz：

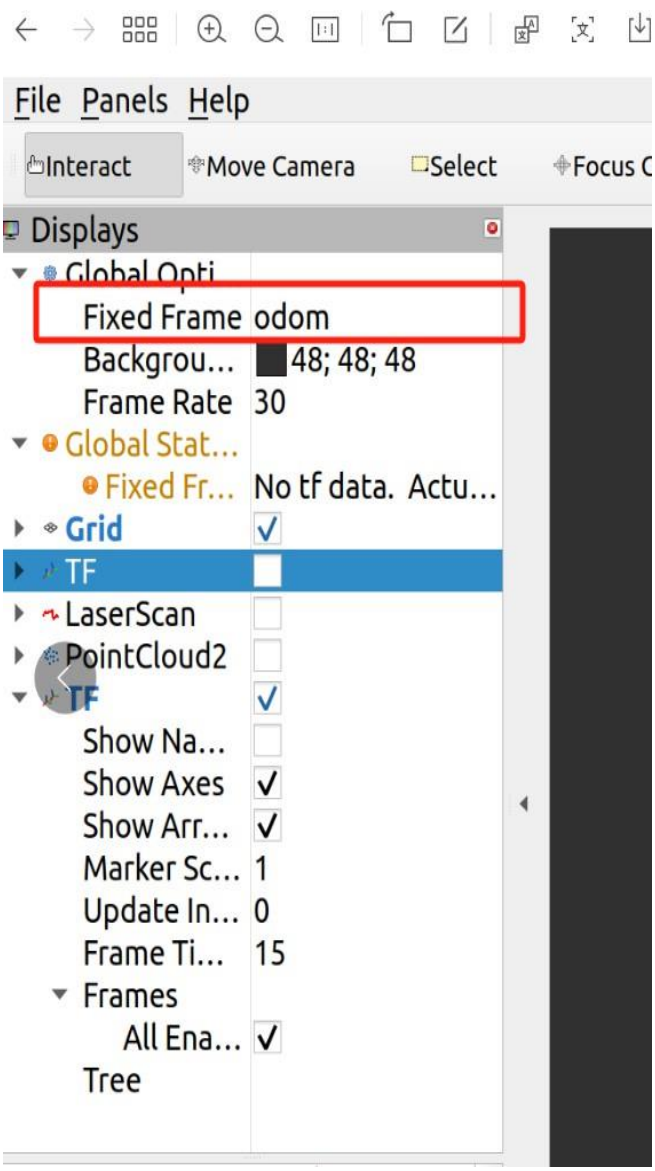
bash

```
rviz2
```

2. 在 RViz 中，点击 Add-添加以下显示项：
    - TF：用于显示 TF 坐标变换



确保你已经在 Global Options 中设置了 Fixed Frame 为 world。



2.3. 编译并执行

1. 确保你已经在工作空间下，然后运行以下命令进行编译：

bash

```
cd ~/《你们自己工作空间名字》
colcon build
source install/setup.bash
```

2. 运行发布 TF 的节点：

bash

```
ros2 run tf_broadcaster_py tf_broadcaster_node
```

2.4. 在 RViz 中查看 TF

bash
```
ros2 run tf2_ros tf2_echo world robot
```

此时，Rviz2 会显示从 world 到 robot 的坐标变换。你应该能够看到 robot 以 (1, 2, 0) 为中心并围绕 Z 轴旋转。

三、坐标变换

3.1. 创建 Python 节点进行坐标变换

1. 在你们自己的工作空间的 src 目录下创建一个新的包：

bash
```
cd ~/《你们自己的工作空间》/src
ros2 pkg create --build-type ament_python tf_listener_py
```

2. 进入 tf_listener_py 包目录，并编辑 setup.py 文件，添加执行坐标变换的节点：

python
```
# setup.py
# 导入 setuptools 中的 setup 函数，用于配置和构建 Python 包
from setuptools import setup

# 定义 ROS2 包的名称
package_name = 'tf_listener_py'

# 配置包的信息和依赖
setup(
    # 包名
    name=package_name,
    # 版本号（0.0.0 表示初始版本）
    version='0.0.0',
    # 指定要包含的 Python 包
    packages=[package_name],

    # 定义要安装的数据文件
```

```
    data_files=[
        # 安装 ROS2 包索引文件，用于包的发现
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        # 安装 package.xml 文件，包含包的元数据
        ('share/' + package_name, ['package.xml']),
    ],

    # 指定包的依赖项
    install_requires=['setuptools'],
    # 指定包是否可以安全地以 zip 格式运行
    zip_safe=True,

    # 包的维护者信息
    maintainer='your_name',
    maintainer_email='your_email@example.com',
    # 包的描述信息
    description='Python TF listener example',
    # 使用的许可证
    license='Apache License 2.0',
    # 运行测试所需的依赖项
    tests_require=['pytest'],

    # 定义可执行脚本的入口点
    entry_points={ 'console_s
        cripts': [
            # 创建一个名为 tf_listener_node 的可执行文件
            # 指向 tf_listener_py.tf_listener 模块的 main 函数
            'tf_listener_node = tf_listener_py.tf_listener:main',
        ],
    },
)
```

3. 在 tf_listener_py 包中创建一个 Python 文件 tf_listener.py：

bash

```
cd tf_listener_py
touch tf_listener.py
chmod +x tf_listener.py
```

4. 编辑 tf_listener.py 文件，编写监听 TF 并进行坐标变换的代码：

python

```
# tf_listener_py/tf_listener.py
```

```
# 导入必要的库
import rclpy  # ROS2 Python 客户端库
from rclpy.node import Node  # ROS2 节点基类
from geometry_msgs.msg import PointStamped  # 带时间戳的点消息类型
import tf2_ros  # TF2 变换库
import tf2_geometry_msgs  # TF2 几何消息转换工具

# 定义 TF 监听器节点类
class TFListener(Node):

    def __init__(self):
        # 初始化节点，设置节点名称
        super().__init__('tf_listener_node')

        # 创建一个 TF 缓冲区和监听器
        # Buffer 用于存储和查询坐标变换
        self.tf_buffer = tf2_ros.Buffer()
        # 创建 TF 监听器，订阅 TF 话题并将数据存入缓冲区
        self.tf_listener = tf2_ros.TransformListener(self.tf_buffer,
self)

        # 创建一个定时器，每 1 秒执行一次 timer_callback 函数
        self.timer = self.create_timer(1.0, self.timer_callback)

    def timer_callback(self):
        try:
            # 查询从 world 坐标系到 robot 坐标系的变换
            # lookup_transform(目标坐标系, 源坐标系, 时间点)
            transform = self.tf_buffer.lookup_transform('robot',
'world', rclpy.time.Time())

            # 创建一个点，设置在 world 坐标系中的位置
            point_in_world = PointStamped()
            point_in_world.header.frame_id = 'world'  # 设置坐标系
            point_in_world.header.stamp =
self.get_clock().now().to_msg()  # 设置时间戳
            point_in_world.point.x = 3.0  # 设置 x 坐标
            point_in_world.point.y = 2.0  # 设置 y 坐标
            point_in_world.point.z = 0.0  # 设置 z 坐标

            # 将点从 world 坐标系转换到 robot 坐标系
            point_in_robot =
tf2_geometry_msgs.do_transform_point(point_in_world, transform)
```

```
            # 输出转换后的坐标
            self.get_logger().info(f'Point in "robot" frame:
({point_in_robot.point.x}, {point_in_robot.point.y},
{point_in_robot.point.z})')

        except tf2_ros.TransformException as ex:
            # 如果转换失败，输出警告信息
            self.get_logger().warn(f'Could not transform: {ex}')

# 主函数
def main(args=None):
    # 初始化 ROS2
    rclpy.init(args=args)
    # 创建节点实例
    node = TFListener()
    # 运行节点
    rclpy.spin(node)
    # 销毁节点
    node.destroy_node()
    # 关闭 ROS2
    rclpy.shutdown()

# 如果直接运行此脚本，则执行 main 函数
if __name__ == '__main__':
    main()
```

- 我们创建了一个 TFListener 节点，它使用 tf2_ros.Buffer 和 tf2_ros.TransformListener 来监听 TF 坐标变换。
- 每隔 1 秒，我们尝试从 world 坐标系转换到 robot 坐标系，并将一个坐标点从 world 变换到 robot 坐标系。
- 如果成功，我们会打印出在 robot 坐标系下的坐标位置。

3.2. 配置 RViz 查看 TF

1. 启动 RViz：

bash

```
rviz2
```

2. 在 RViz 中添加以下显示项：
    - **TF**：用于显示 TF 坐标变换。
    - **Point**：用于显示坐标点（如果你希望可视化转换后的坐标点，可以在后续扩展该功能）。

3.3. 编译并执行

1. 确保你在工作空间下，并运行以下命令进行编译：

bash
```
cd ~/《你们自己的工作空间》
colcon build
source install/setup.bash
```

2. 在第一个终端运行发布 TF 的节点（假设你已经按照第一部分中的步骤创建并运行了 tf_broadcaster_py 包）：

bash
```
ros2 run tf_broadcaster_py tf_broadcaster_node
```

3. 在另一个终端运行 TF 监听和坐标变换节点：

bash
```
source install/setup.bash
ros2 run tf_listener_py tf_listener_node
```

3.4. 在 RViz 中查看 TF 变换

- 在 RViz 中，确保 TF 显示项已经启用，可以看到 world 到 robot 坐标系的变换。
- 你可以通过终端看到每秒输出的坐标点，这些点是从 world 坐标系变换到 robot 坐标系后的结果。