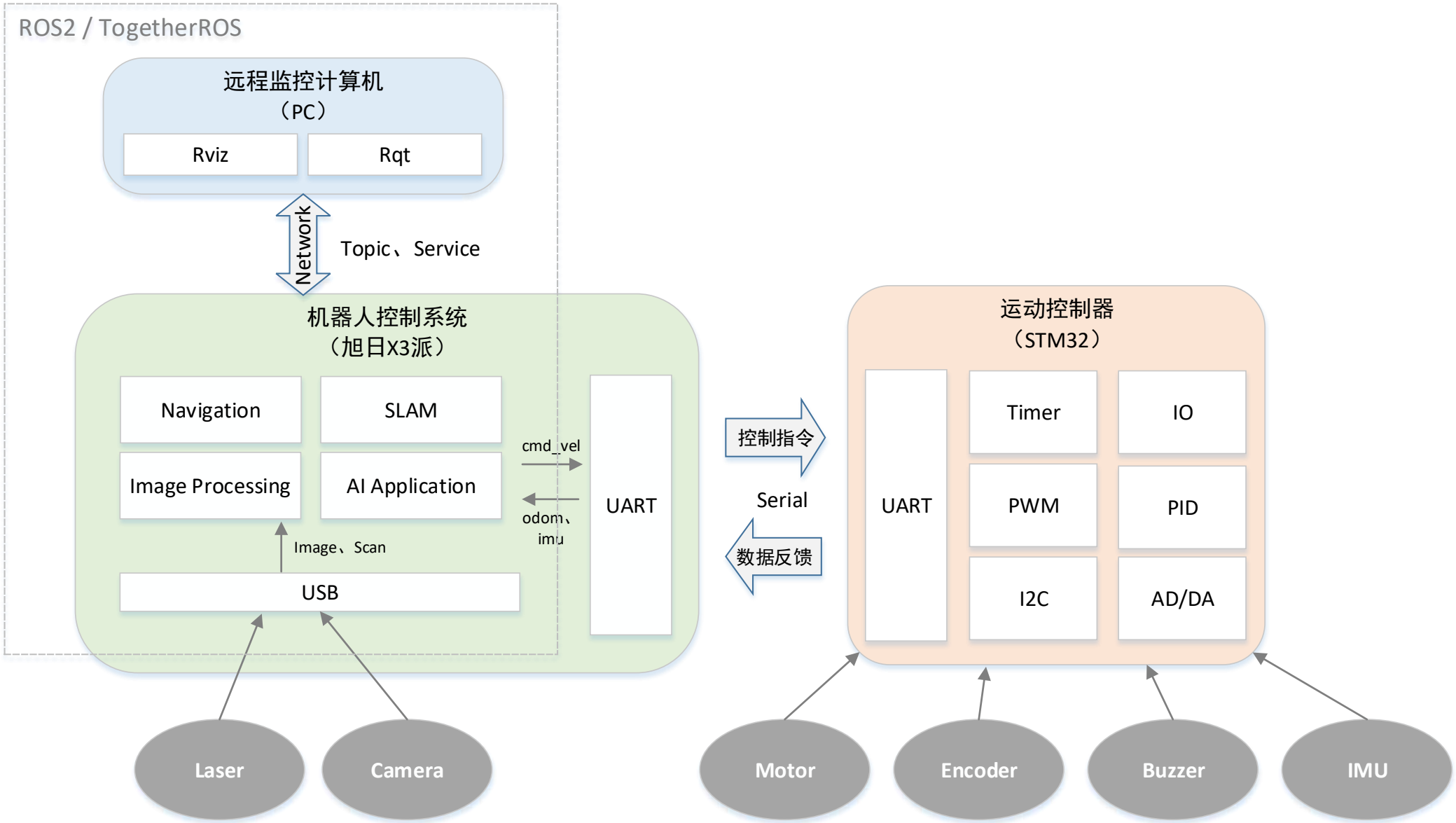
古月居
GYH.AI

ROS
暑期学校

如何开发智能小车的应用功能

主讲人：李乔龙

目录


ROS2 / TogetherROS
远程监控计算机
（PC）
Rviz
Rqt
Network
Topic、Service
机器人控制系统
（旭日X3派）
Navigation
SLAM
Image Processing
AI Application
Image、Scan
USB
cmd_vel
odom、
imu
UART
控制指令
Serial
数据反馈
运动控制器
（STM32）
UART
Timer
IO
PWM
PID
I2C
AD/DA
Laser
Camera
Motor
Encoder
Buzzer
IMU

目录

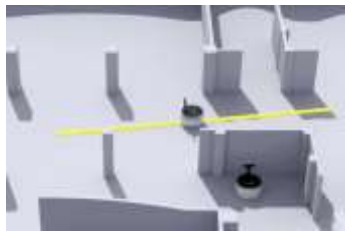
1. 运动控制

机器人速度控制

发布者节点

Topic: cmd_vel
Message: geometry_msgs/Twist
Data: linear, angular

订阅者节点



geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

速度控制示例

```
$ ros2 launch originbot_bringup originbot.launch.py
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

机器人速度控制

geometry_msgs/Twist Message

File: geometry_msgs/Twist.msg

Raw Message Definition

```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3    linear
Vector3    angular
```

Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

geometry_msgs/Vector3 Message

File: geometry_msgs/Vector3.msg

Raw Message Definition

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

linear.x ： x方向线速度，机器人以小车正前方为x方向，垂直地面向上为z方向，满足右手定则。单位m/s

angular.x ： 绕x轴逆时针旋转角度，机器人以小车正前方为x方向，垂直地面向上为z方向，满足右手定则。单位rad/s

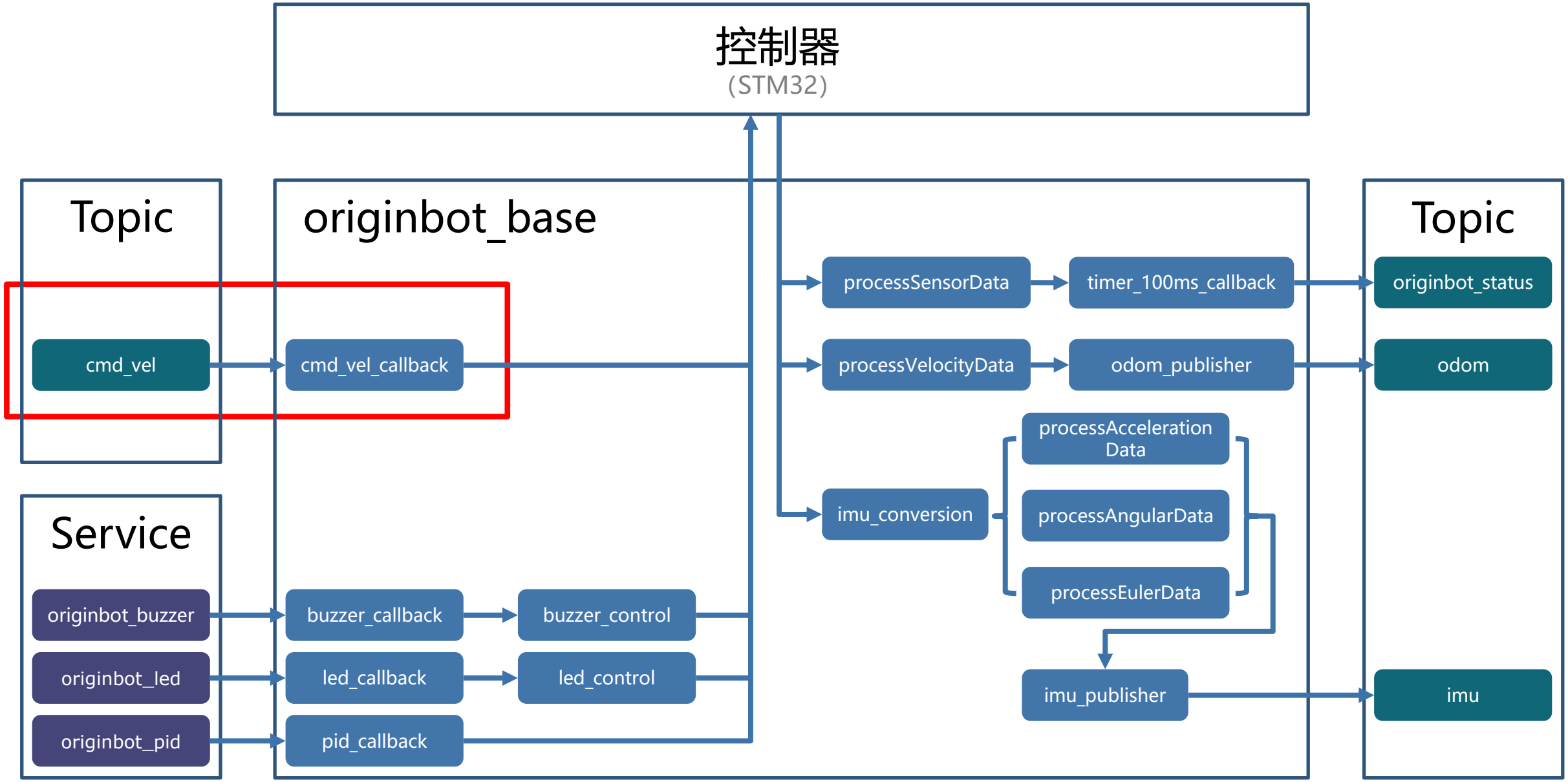
机器人速度控制

```
222     try:
223         pub_thread.wait_for_subscribers()
224         pub_thread.update(x, y, z, th, speed, turn)
225 
226         print(msg)
227         print(vels(speed,turn))
228         while(1):
229             key = getKey(settings, key_timeout)
230             if key in moveBindings.keys():
231                 x = moveBindings[key][0]
232                 y = moveBindings[key][1]
233                 z = moveBindings[key][2]
234                 th = moveBindings[key][3]
235             elif key in speedBindings.keys():
236                 speed = min(speed_limit, speed * speedBindings[key][0])
237                 turn = min(turn_limit, turn * speedBindings[key][1])
238                 if speed == speed_limit:
239                     print("Linear speed limit reached!")
240                 if turn == turn_limit:
241                     print("Angular speed limit reached!")
242                 print(vels(speed,turn))
243                 if (status == 14):
244                     print(msg)
245                 status = (status + 1) % 15
246             else:
```

```
    def update(self, x, y, z, th, speed, turn):
        self.condition.acquire()
        self.x = x
        self.y = y
        self.z = z
        self.th = th
        self.speed = speed
        self.turn = turn
        # Notify publish thread that we have a new message.
        self.condition.notify()
        self.condition.release()
```

代码地址：https://github.com/ros-teleop/teleop_twist_keyboard/blob/8e1e14fdebd31b8e37ec1a453fe7c7fcf03e7648/teleop_twist_keyboard.py#L114

速度控制话题的订阅


控制器
(STM32)
Topic
originbot_base
Topic
cmd_vel
cmd_vel_callback
processSensorData
timer_100ms_callback
originbot_status
processVelocityData
odom_publisher
odom
processAcceleration Data
imu_conversion
processAngularData
processEulerData
Service
originbot_buzzer
buzzer_callback
buzzer_control
originbot_led
led_callback
led_control
imu_publisher
imu
originbot_pid
pid_callback

机器人速度控制

```
def run(self):
    twist_msg = TwistMsg()

    if stamped:
        twist = twist_msg.twist
        twist_msg.header.stamp = rospy.Time.now()
        twist_msg.header.frame_id = twist_frame
    else:
        twist = twist_msg
    while not self.done:
        if stamped:
            twist_msg.header.stamp = rospy.Time.now()
        self.condition.acquire()
        # Wait for a new message or timeout.
        self.condition.wait(self.timeout)

        # Copy state into twist message.
        twist.linear.x = self.x * self.speed
        twist.linear.y = self.y * self.speed
        twist.linear.z = self.z * self.speed
        twist.angular.x = 0
        twist.angular.y = 0
        twist.angular.z = self.th * self.turn

        self.condition.release()

        # Publish.
        self.publisher.publish(twist_msg)
```

速度控制话题的订阅

```
43    // 创建里程计、机器人状态的发布者
44    odom_publisher_   = this->create_publisher<nav_msgs::msg::Odometry>("odom", 10);
45    status_publisher_ = this->create_publisher<originbot_msgs::msg::OriginbotStatus>("originbot_status", 10);
46
47    // 创建速度指令的订阅者
48    cmd_vel_subscription_ = this->create_subscription<geometry_msgs::msg::Twist>("cmd_vel", 10, std::bind(&OriginbotBase::cmd_vel_callback, this, _1));
49
50    // 创建控制蜂鸣器和LED的服务
51    buzzer_service_ = this->create_service<originbot_msgs::srv::OriginbotBuzzer>("originbot_buzzer", std::bind(&OriginbotBase::buzzer_callback, this, _1, _2));
52    led_service_ = this->create_service<originbot_msgs::srv::OriginbotLed>("originbot_led", std::bind(&OriginbotBase::led_callback, this, _1, _2));
53    pid_service_ = this->create_service<originbot_msgs::srv::OriginbotPID>("originbot_pid", std::bind(&OriginbotBase::pid_callback, this, _1, _2));
54
```

创建速度控制话题的订阅者

```
468  void OriginbotBase::cmd_vel_callback(const geometry_msgs::msg::Twist::SharedPtr msg)
469  {
470      DataFrame cmdFrame;
471      float leftSpeed = 0.0, rightSpeed = 0.0;
472
473      float x_linear = msg->linear.x;
474      float z_angular = msg->angular.z;
475
476      //差分轮运动学模型求解
477      leftSpeed  = x_linear - z_angular * ORIGINBOT_WHEEL_TRACK / 2.0;
478      rightSpeed = x_linear + z_angular * ORIGINBOT_WHEEL_TRACK / 2.0;
479
480      // RCLCPP_INFO(this->get_logger(), "leftSpeed = '%f' rightSpeed = '%f'", leftSpeed * 100, rightSpeed * 100);
481
482      if (leftSpeed < 0)
483          cmdFrame.data[0] = 0x00;
484      else
485          cmdFrame.data[0] = 0xff;
486      cmdFrame.data[1] = int(abs(leftSpeed) * 1000) & 0xff;          //速度值从m/s变为mm/s
487      cmdFrame.data[2] = (int(abs(leftSpeed) * 1000) >> 8) & 0xff;
```

创建速度控制话题的回调函数

代码地址：/userdata/dev_ws/src/originbot/originbot_base/src

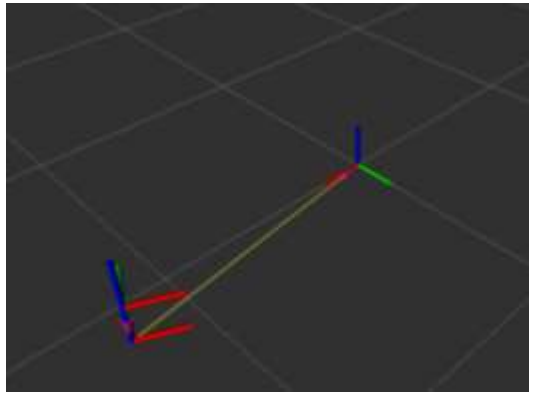
机器人里程计

发布者节点

Topic: odom
Message: nav_msgs/Odometry
Data: pose, twist

订阅者节点

TF: base_footprint → odom



里程计示例

```
$ ros2 launch originbot_bringup originbot.launch.py
$ ros2 run teleop_twist_keyboard
teleop_twist_keyboard
$ ros2 launch originbot_viz display_robot_tf.launch.py
```

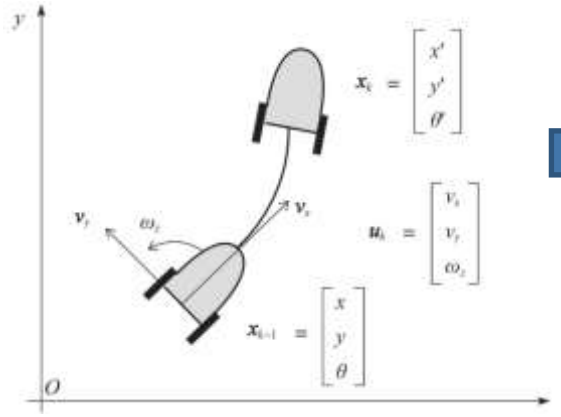
机器人里程计

编码器 ->角速度

轮子 -> 旋转半径

线速度

运动学方程



里程计信息

机器人速度控制

nav_msgs/Odometry Message

File: nav_msgs/Odometry.msg

Raw Message Definition

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

geometry_msgs/Pose Message

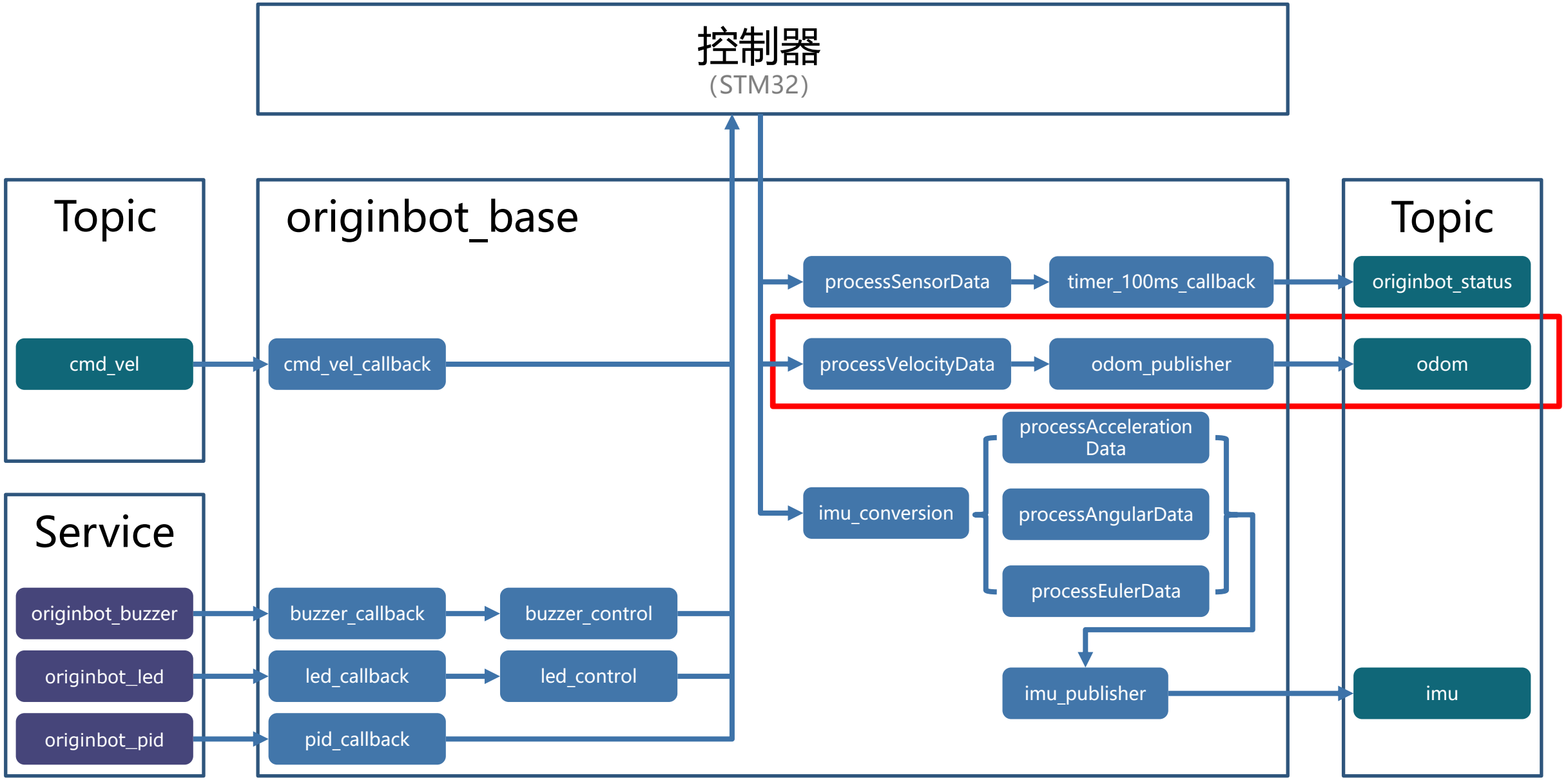
File: geometry_msgs/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

twist：速度信息
pose：姿态信息

机器人里程计


控制器
(STM32)
Topic
originbot_base
Topic
processSensorData
timer_100ms_callback
originbot_status
cmd_vel
cmd_vel_callback
processVelocityData
odom_publisher
odom
processAcceleration Data
imu_conversion
processAngularData
processEulerData
Service
originbot_buzzer
buzzer_callback
buzzer_control
originbot_led
led_callback
led_control
imu_publisher
imu
originbot_pid
pid_callback

里程计话题与TF的维护

```
42
43      // 创建里程计、机器人状态的发布者
44      odom_publisher_   = this->create_publisher<nav_msgs::msg::Odometry>("odom", 10);
45      status_publisher_ = this->create_publisher<originbot_msgs::msg::OriginbotStatus>("originbot_status", 10);
46
47      // 创建速度指令的订阅者
48      cmd_vel_subscription_ = this->create_subscription<geometry_msgs::msg::Twist>("cmd_vel", 10, std::bind(&OriginbotBase::cmd_vel_callback, this, _1));
49
50      // 创建控制蜂鸣器和LED的服务
51      buzzer_service_ = this->create_service<originbot_msgs::srv::OriginbotBuzzer>("originbot_buzzer", std::bind(&OriginbotBase::buzzer_callback, this, _1, _2));
52      led_service_ = this->create_service<originbot_msgs::srv::OriginbotLed>("originbot_led", std::bind(&OriginbotBase::led_callback, this, _1, _2));
53      pid_service_ = this->create_service<originbot_msgs::srv::OriginbotPID>("originbot_pid", std::bind(&OriginbotBase::pid_callback, this, _1, _2));
54
55      // 创建TF广播器
56      tf_broadcaster_ = std::make_unique<tf2_ros::TransformBroadcaster>(*this);
57
```

创建里程计的话题发布者和TF广播器

```
309 void OriginbotBase::odom_publisher(float vx, float vth)
310 {
311     auto odom_msg = nav_msgs::msg::Odometry();
312
313     //里程数据计算
314     odom_msg.header.frame_id = "odom";
315     odom_msg.header.stamp = this->get_clock()->now();
316     odom_msg.pose.pose.position.x = odom_x_;
317     odom_msg.pose.pose.position.y = odom_y_;
318     odom_msg.pose.pose.position.z = 0;
```

处理里程计信息的周期发布和TF更新

代码地址：/userdata/dev_ws/src/originbot/originbot_base/src

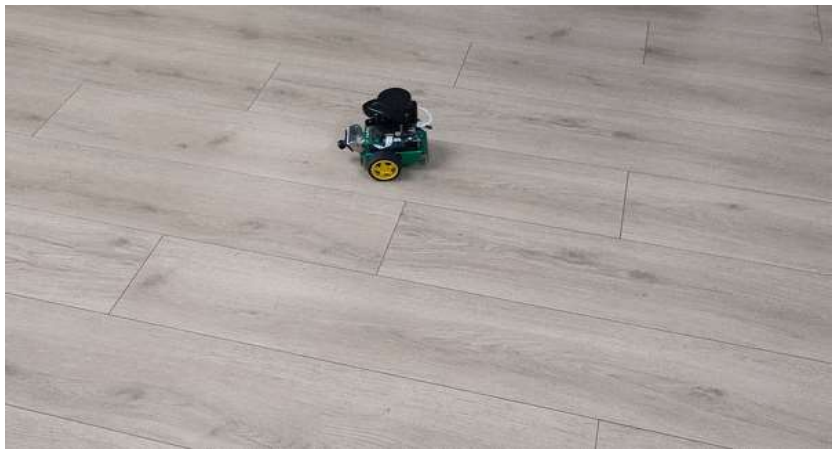
键盘控制

键盘控制

```
# 机器人终端1
$ ros2 launch originbot_bringup originbot.launch.py
# 机器人终端2/PC端终端
$ ros2 run teleop_twist_keyboard teleop_twist_keyboard
```

编程控制

```python
import rclpy                                    # ROS2 Python接口库
from rclpy.node import Node                     # ROS2 节点类
from geometry_msgs.msg import Twist             # 速度话题的消息

"""
创建一个发布者节点
"""
class PublisherNode(Node):

    def __init__(self, name):
        super().__init__(name)                                          # ROS2节点父类初始化
        self.pub   = self.create_publisher(Twist, 'cmd_vel', 10)        # 创建发布者对象（消息类型、话题名、队列长度）
        self.timer = self.create_timer(0.5, self.timer_callback)       # 创建一个定时器（单位为秒的周期，定时执行的回调函数）

    def timer_callback(self):                                           # 创建定时器周期执行的回调函数
        twist = Twist()                                                 # 创建一个Twist类型的消息对象
        twist.linear.x  = 0.2                                           # 填充消息对象中的线速度
        twist.angular.z = 0.8                                           # 填充消息对象中的角速度
        self.pub.publish(twist)                                         # 发布话题消息
        self.get_logger().info('Publishing: "linear: %0.2f, angular: %0.2f"' % (twist.linear.x, twist.angular.z))

def main(args=None):                            # ROS2节点主入口main函数
    rclpy.init(args=args)                       # ROS2 Python接口初始化
    node = PublisherNode("draw_circle")         # 创建ROS2节点对象并进行初始化
    rclpy.spin(node)                            # 循环等待ROS2退出
    node.destroy_node()                         # 销毁节点对象
    rclpy.shutdown()                            # 关闭ROS2 Python接口
```

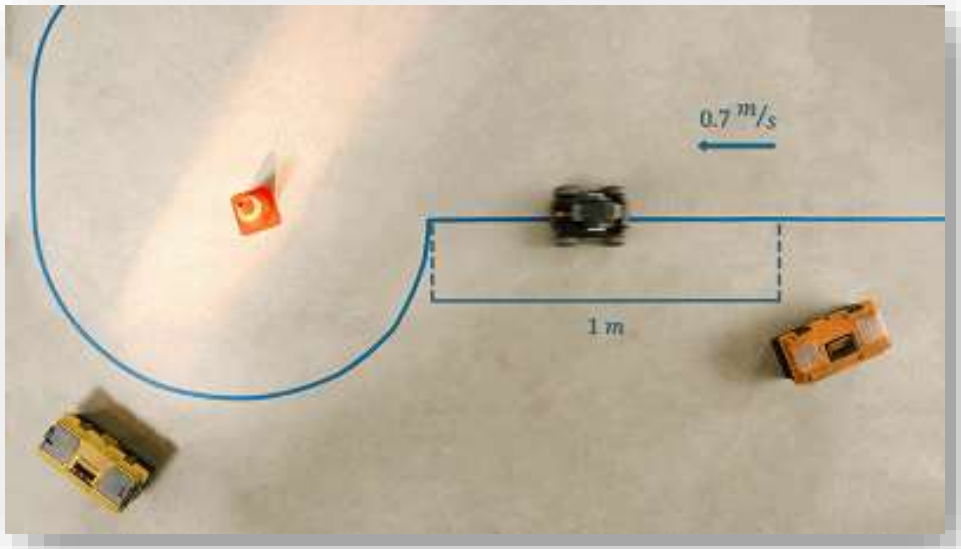


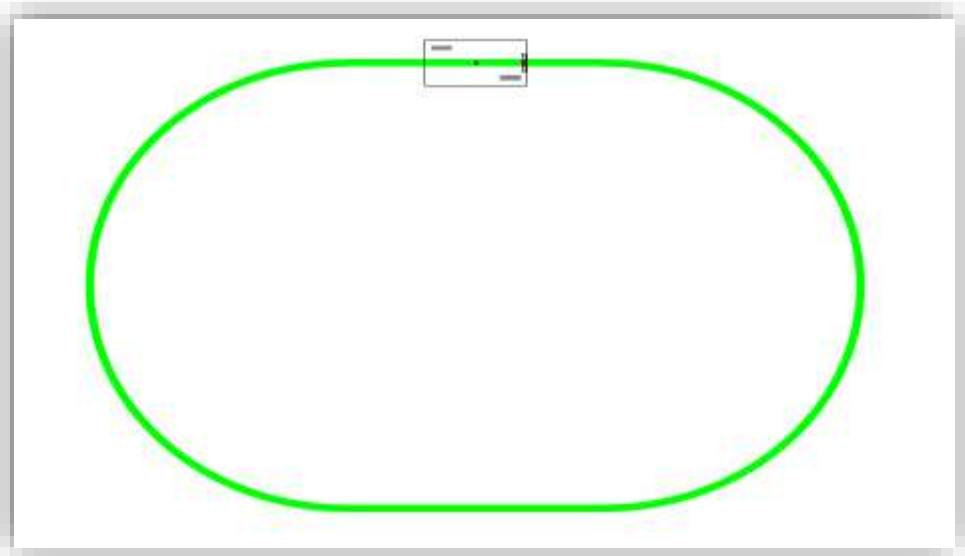
编程控制

```
# 机器人终端1
$ ros2 launch originbot_bringup originbot.launch.py
# 机器人终端2/PC端终端
$ ros2 run originbot_demo draw_circle
```

2. 视觉巡线

视觉巡线





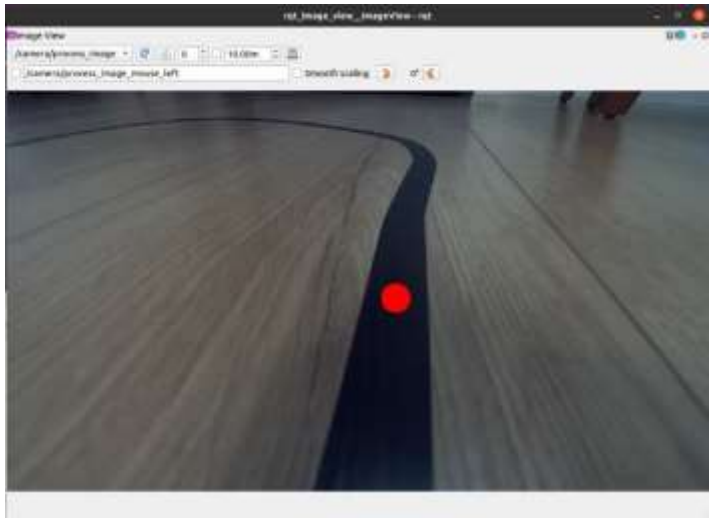
引导线识别


```
from rclpy.node import Node
from sensor_msgs.msg import Image
from geometry_msgs.msg import Twist

class Follower(Node):
    def __init__(self):
        super().__init__('line_follower')
        self.get_logger().info("Start line follower.")

        self.bridge = cv_bridge.CvBridge()

        self.image_sub = self.create_subscription(Image, '/image_raw', self.image_callback, 10)
        self.cmd_vel_pub = self.create_publisher(Twist, 'cmd_vel', 10)
        self.pub = self.create_publisher(Image, '/camera/process_image', 10)

        self.twist = Twist()
```

引导线识别

```
    def image_callback(self, msg):
        image = self.bridge.imgmsg_to_cv2(msg, 'bgr8')
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        lower_yellow = numpy.array([ 10,  70, 30])
        upper_yellow = numpy.array([255, 255, 250])
        mask = cv2.inRange(hsv, lower_yellow, upper_yellow)

        h, w, d = image.shape
        search_top = int(h/2)
        search_bot = int(h/2 + 20)
        mask[0:search_top, 0:w] = 0
        mask[search_bot:h, 0:w] = 0
        M = cv2.moments(mask)

        if M['m00'] > 0:
            cx = int(M['m10']/M['m00'])
            cy = int(M['m01']/M['m00'])
            cv2.circle(image, (cx, cy), 20, (0,0,255), -1)

            # 基于检测的目标中心点，计算机器人的控制参数
            err = cx - w/2
            self.twist.linear.x = 0.1
            self.twist.angular.z = -float(err) / 400
            self.cmd_vel_pub.publish(self.twist)

        self.pub.publish(self.bridge.cv2_to_imgmsg(image, 'bgr8'))

def main(args=None):
    rclpy.init(args=args)
    follower = Follower()
    rclpy.spin(follower)
    follower.destroy_node()
    rclpy.shutdown()
```

视觉巡线

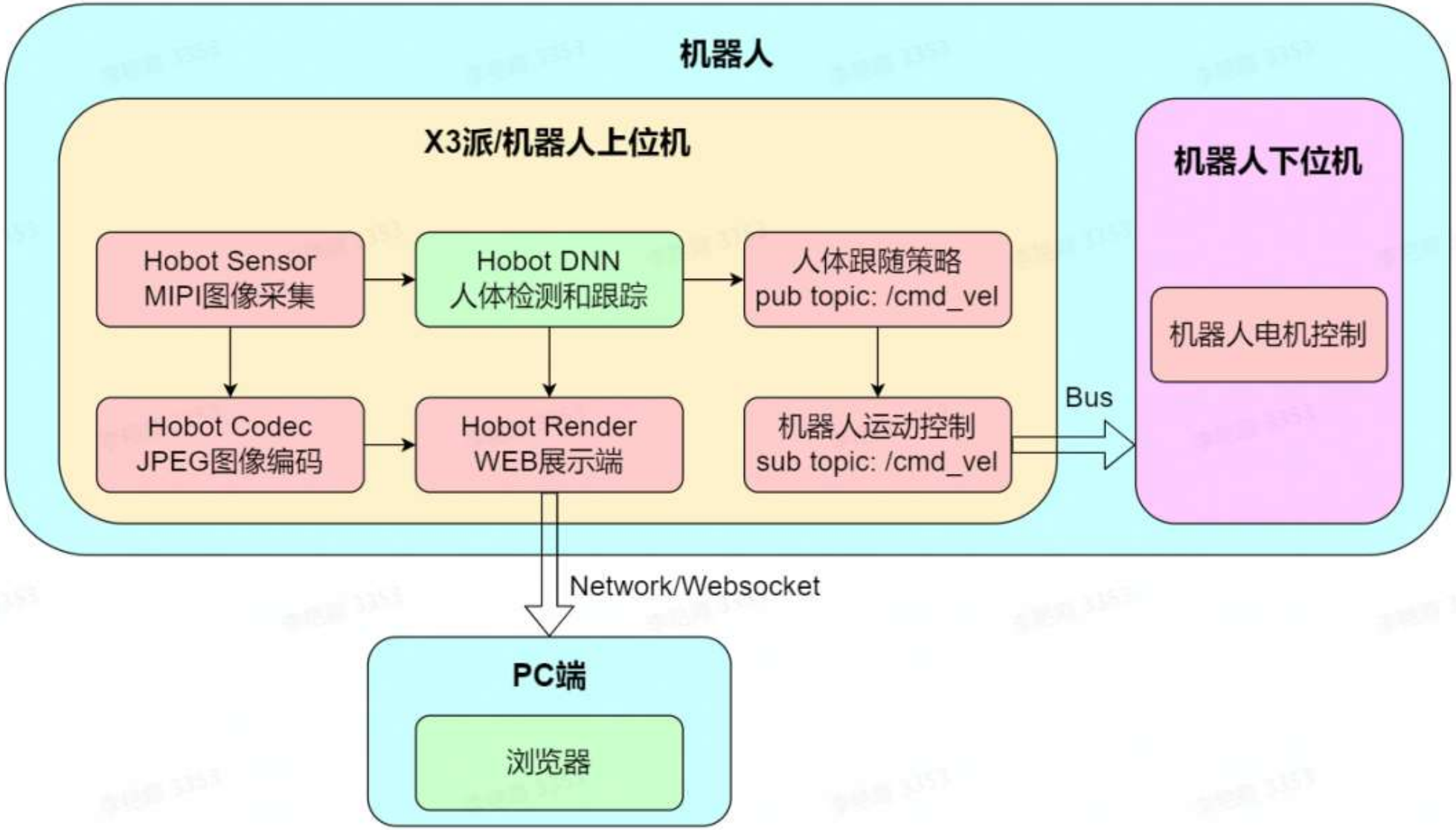


视觉巡线

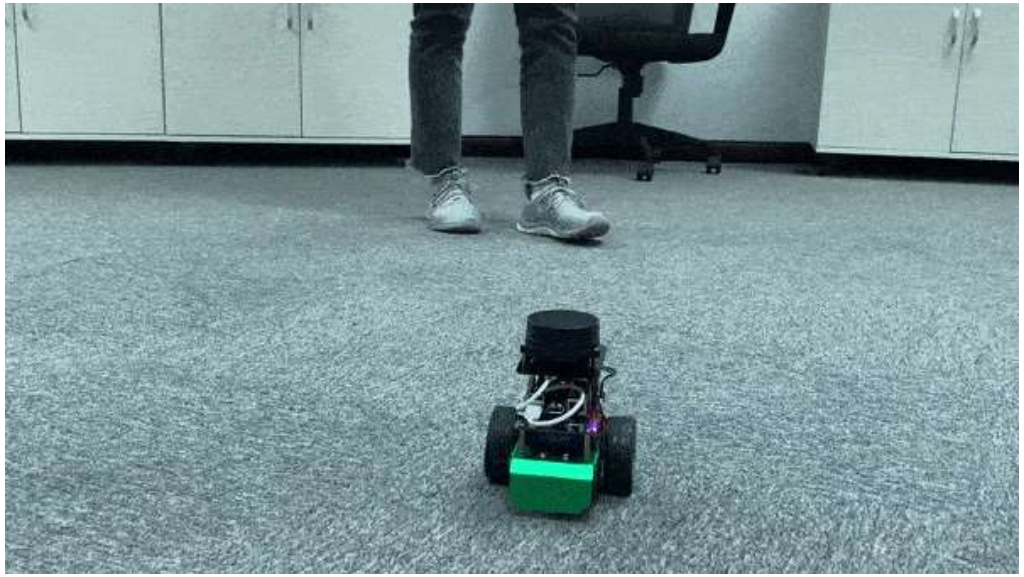
```
# 机器人终端1
$ export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
$ ros2 launch originbot_bringup originbot.launch.py use_camera:=true

# 机器人终端2
$ export RMW_IMPLEMENTATION=rmw_cyclonedds_cpp
$ ros2 run originbot_linefollower follower
# PC端
$ ros2 run rqt_image_view rqt_image_view
```

3. 人体跟随

人体跟随


机器人
X3派/机器人上位机
Hobot Sensor
MIPI图像采集
Hobot DNN
人体检测和跟踪
人体跟随策略
pub topic: /cmd_vel
Hobot Codec
JPEG图像编码
Hobot Render
WEB展示端
机器人运动控制
sub topic: /cmd_vel
Bus
机器人下位机
机器人电机控制
Network/Websocket
PC端
浏览器

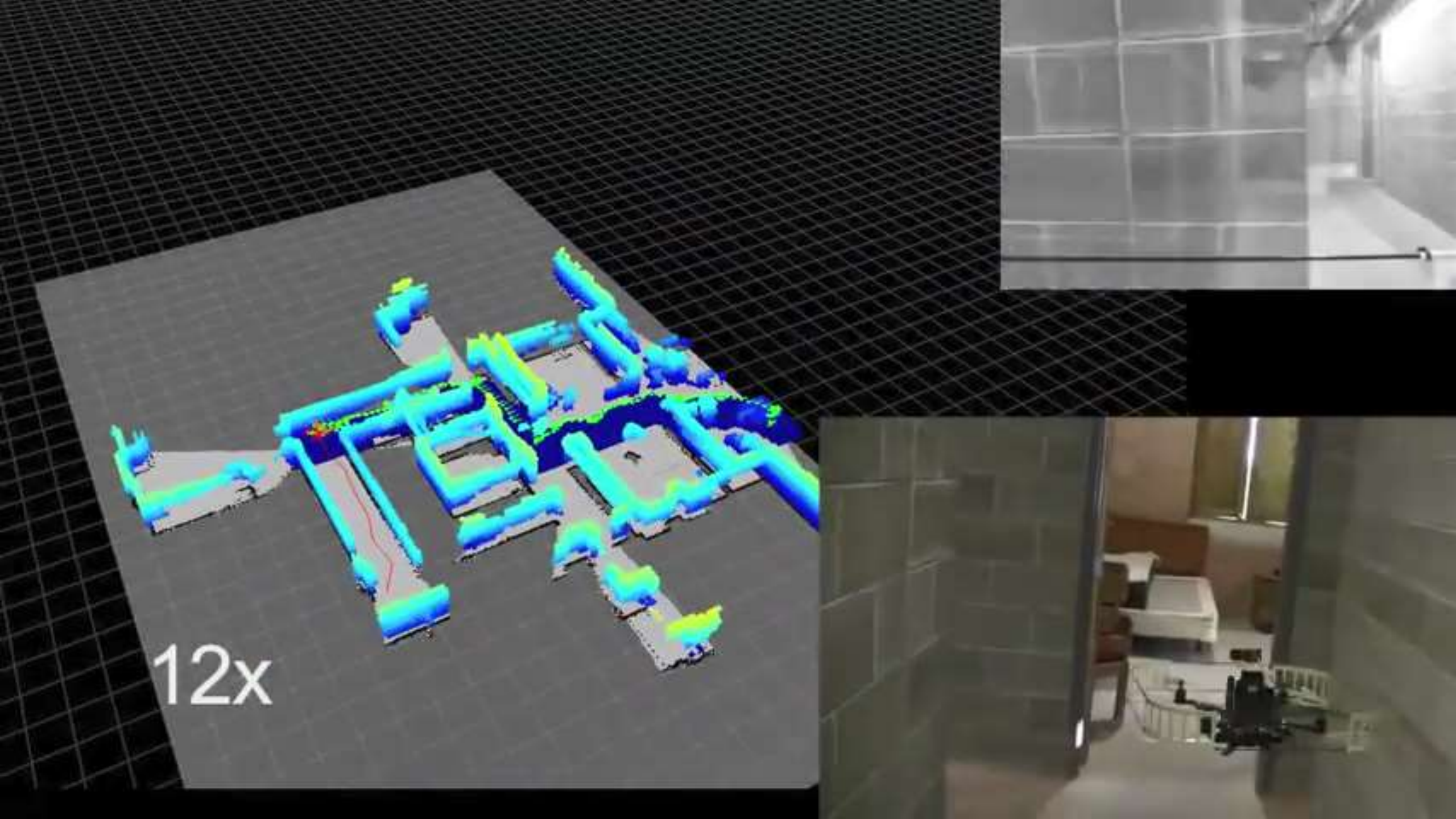
人体跟随



人体跟随

```
# 机器人终端1
$ ros2 launch originbot_bringup originbot.launch.py

# 机器人终端2
$ cd /userdata/dev_ws
$ cp -r /opt/tros/lib/mono2d_body_detection/config/ .
$ ros2 launch body_tracking hobot_body_tracking_without_gesture.launch.py
```

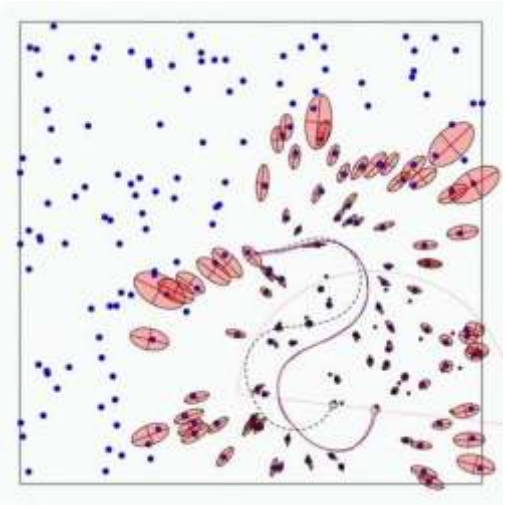
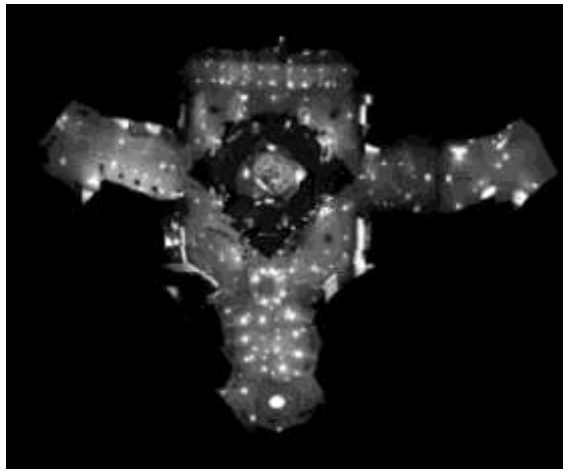
4. SLAM地图构建


12x

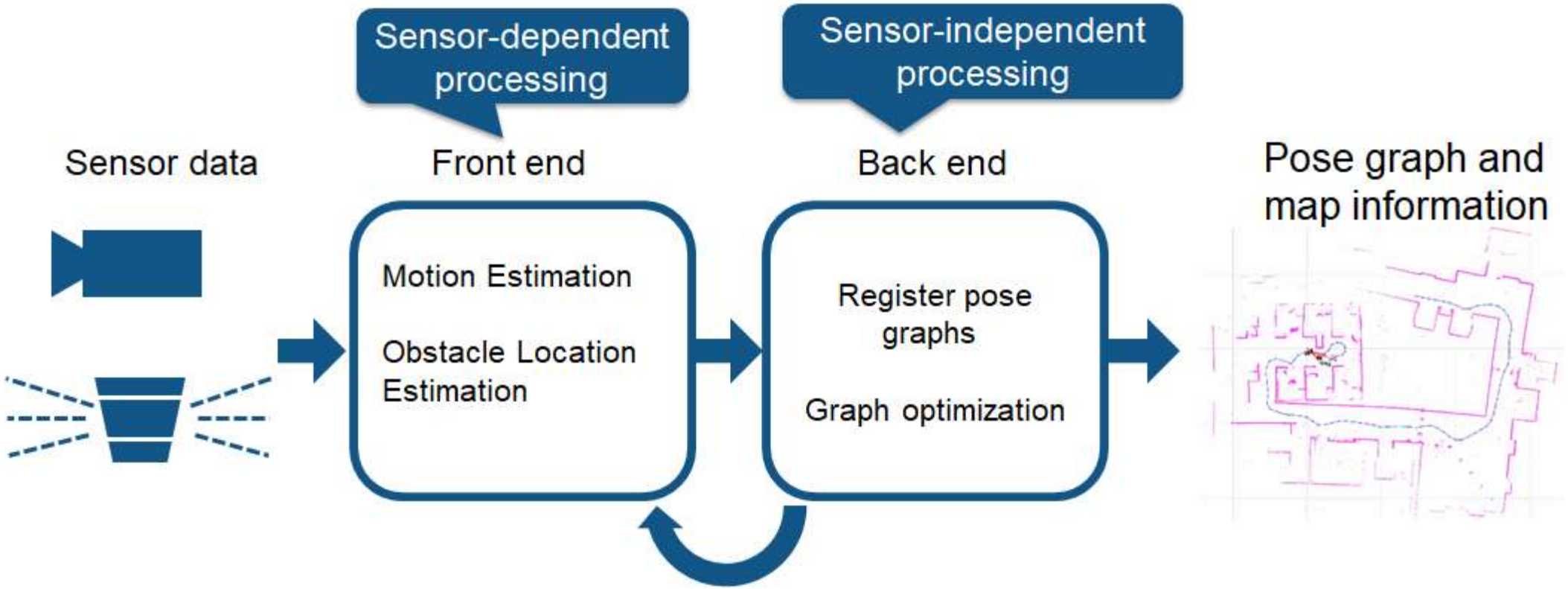
SLAM地图构建

- **SLAM**，全称simultaneous localization and mapping，即时定位与地图构建。即机器人自身位置不确定的情况下，在完全未知的环境中构建地图，同时利用地图进行**自主定位**。
- **定位**，在地图上估测机器人的位置坐标，或者用一个问题来表示：我在哪？
- **地图构建**，这一过程是根据传感器的结果来构建一张地图或者是修正当前地图，同时将结果提供给定位算法作为先验地图。

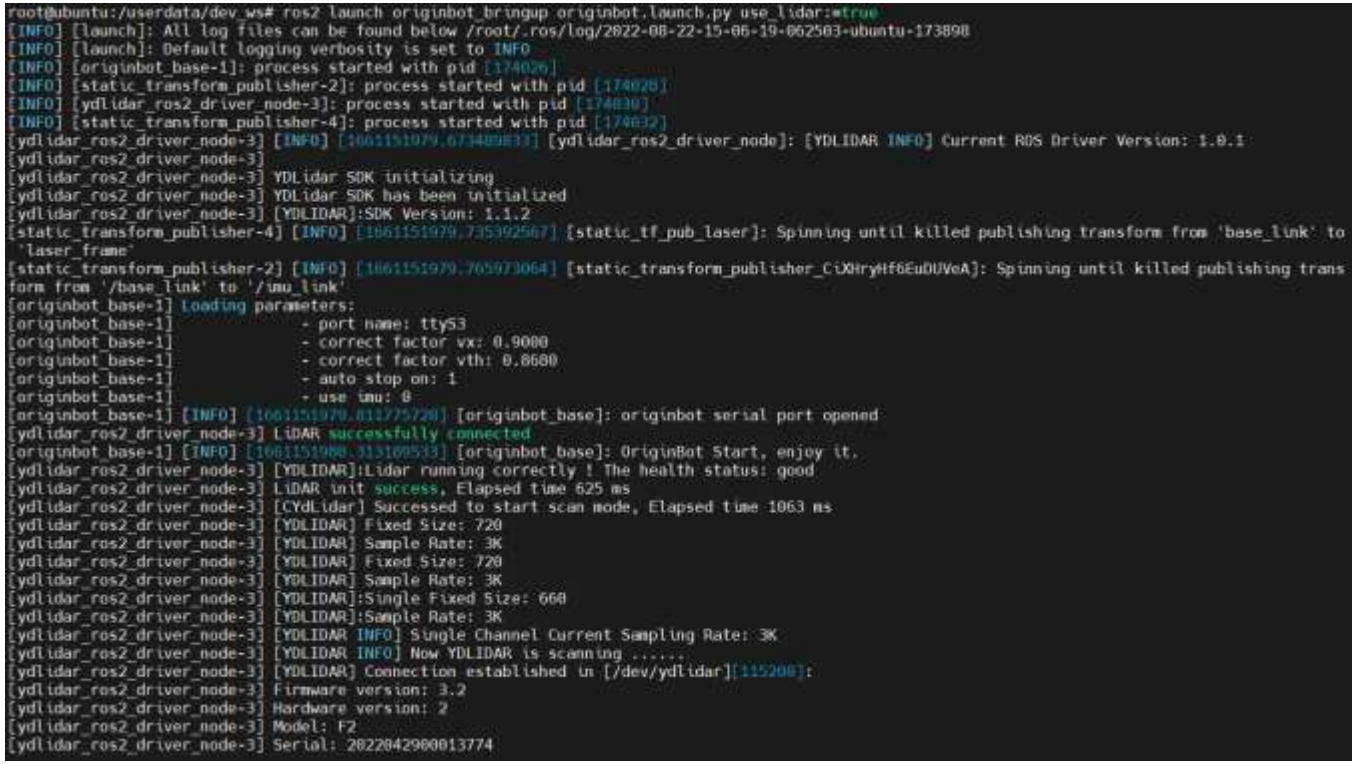


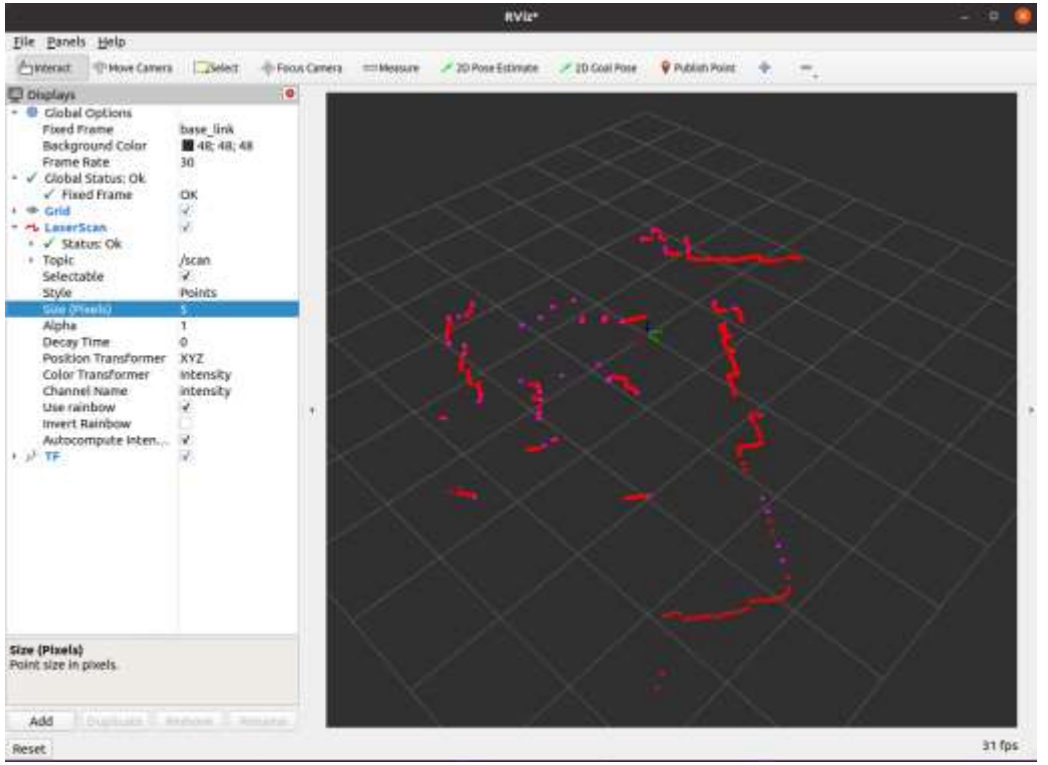



• SLAM介绍


Sensor-dependent processing
Sensor-independent processing
Sensor data
Front end
Back end
Pose graph and map information
Motion Estimation
Obstacle Location Estimation
Register pose graphs
Graph optimization

雷达数据可视化

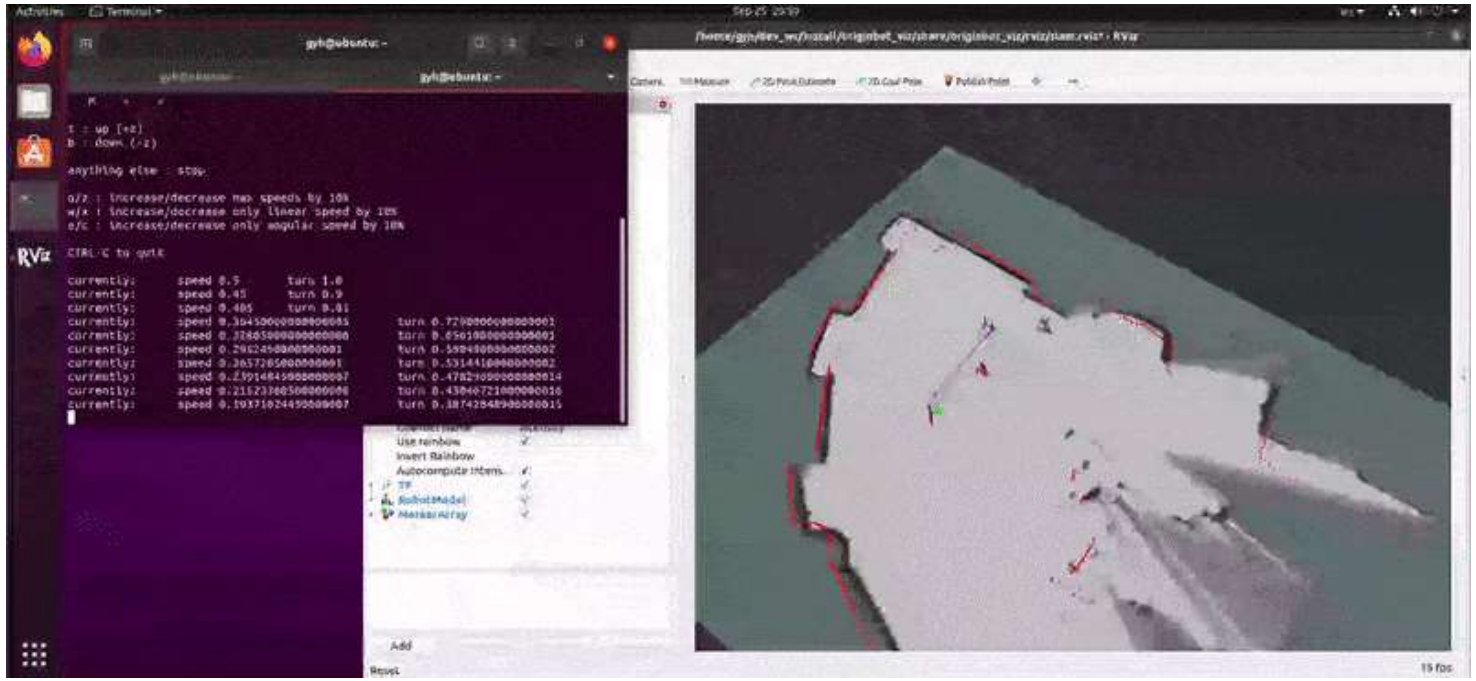




雷达数据可视化

```
# 机器人端
$ ros2 launch originbot_bringup originbot.launch.py use_lidar:=true
# PC端
$ ros2 launch originbot_viz display_lidar.launch.py
```

SLAM地图构建



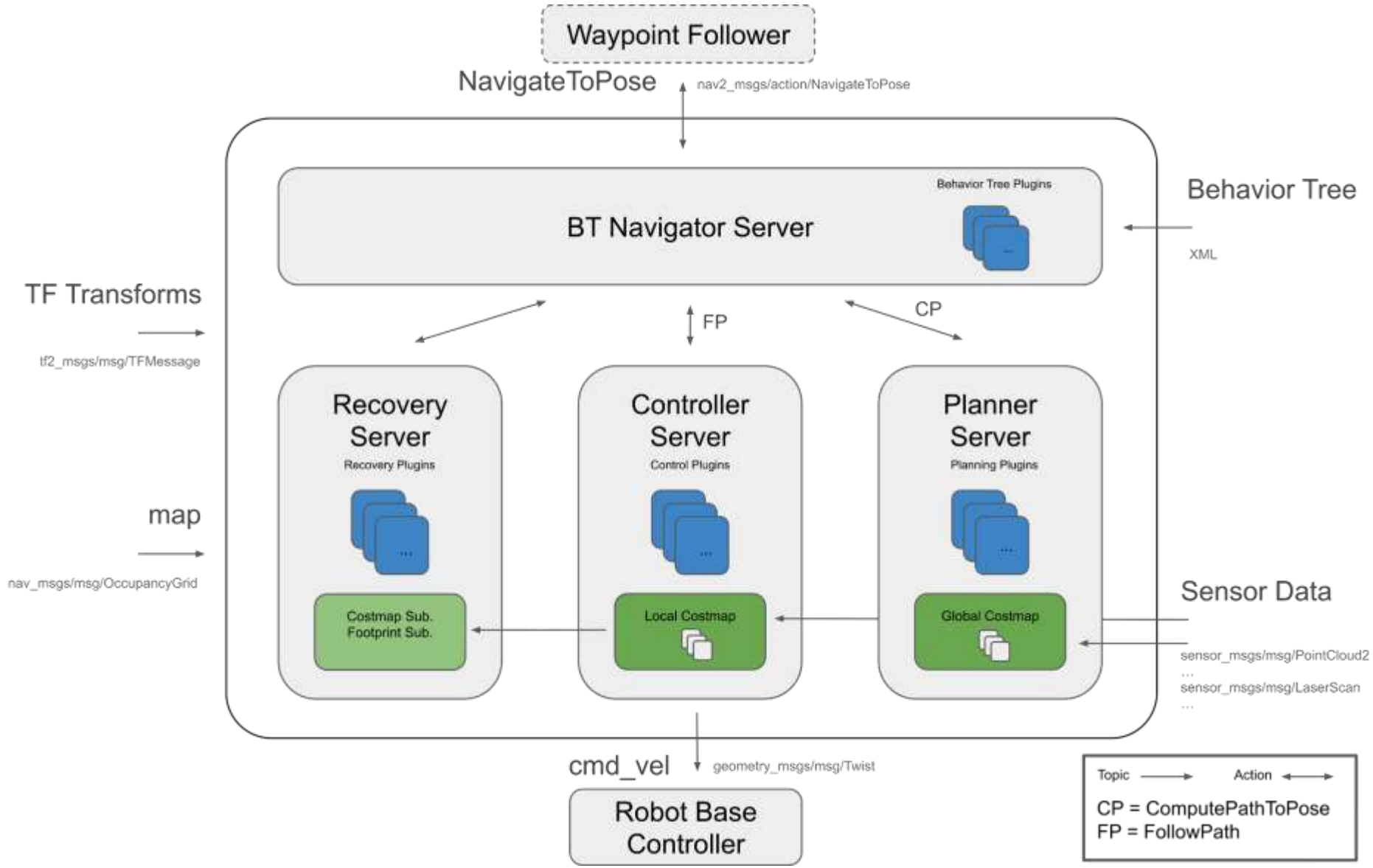
SLAM地图构建

```
# 机器人终端1
$ ros2 launch originbot_bringup originbot.launch.py
use_lidar:=true
# 机器人终端2
$ ros2 launch originbot_navigation cartographer.launch.py
# PC端
$ ros2 launch originbot_viz display_slam.launch.py
```

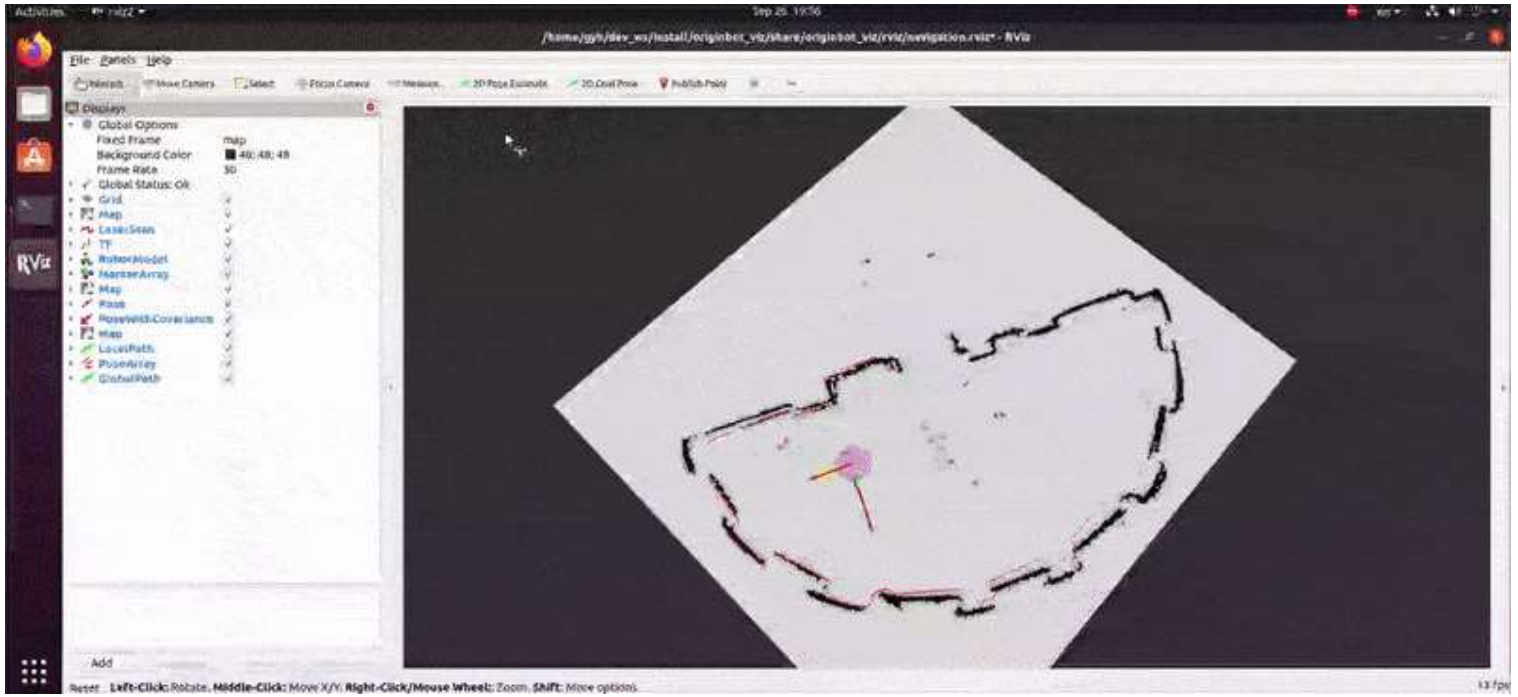
5. 自主导航


Panoview navigation

自主导航


Waypoint Follower
NavigateToPose
nav2_msgs/action/NavigateToPose
BT Navigator Server
Behavior Tree Plugins
Behavior Tree
XML
TF Transforms
tf2_msgs/msg/TFMessage
FP
CP
Recovery Server
Recovery Plugins
Controller Server
Control Plugins
Planner Server
Planning Plugins
map
nav_msgs/msg/OccupancyGrid
Costmap Sub.
Footprint Sub.
Local Costmap
Global Costmap
Sensor Data
sensor_msgs/msg/PointCloud2
...
sensor_msgs/msg/LaserScan
...
cmd_vel
geometry_msgs/msg/Twist
Robot Base Controller
Topic
Action
CP = ComputePathToPose
FP = FollowPath

自主导航



自主导航

```
# 机器人终端1
$ ros2 launch originbot_bringup originbot.launch.py use_lidar:=true
# 机器人终端2
$ ros2 launch originbot_navigation nav_bringup.launch.py
# PC端
$ ros2 launch originbot_viz display_navigation.launch.py
```

感谢观看

怕什么真理无穷，进一寸有一寸的欢喜

更多精彩，欢迎关注

古月居

古月学院

ROS
暑期学校

古月居 搜一下