目录

1 绪论

ROS（Robot Operating System，机器人操作系统）是一个用于机器人编程的开源框架，它为机器人开发者提供了一系列的工具和库，使得构建复杂机器人系统变得更加容易。在 ROS 中，工作空间是组织代码的基础，它通常包含四个主要文件夹：src、build、devel 和可选的 install。src 文件夹用于存放源代码，build 文件夹在编译过程中生成中间文件，devel 文件夹则包含了编译后的可执行文件和库文件，而 install 文件夹用于存放安装后的功能包。

功能包是 ROS 中代码组织的基本单元，每个功能包都包含节点、库、配置文件等。package.xml 文件是功能包的重要组成部分，它描述了功能包的依赖关系和版本信息。节点是 ROS 中的执行单元，它们通过话题、服务、参数服务器等方式进行通信。话题是一种发布/订阅模式的通信机制，允许节点之间传递消息。消息是节点间传递的数据类型，可以是标准数据类型，也可以是自定义类型。

服务提供了一种请求/响应的通信方式，客户端向服务发送请求，服务端处理请求并返回响应。参数服务器则是一个分布式数据库，用于存储和查询参数，使得节点可以共享和访问这些参数。启动文件是用于同时启动多个节点、设置参数和配置环境的脚本，它们通常以 XML 或 Python 格式编写，极大地简化了 ROS 系统的启动过程。

在 ROS 系统的文件结构中，/bin 文件夹存放可执行文件，/etc 文件夹存放配置文件，/include 文件夹存放头文件，/lib 文件夹存放编译后的库文件，而/share 文件夹则存放功能包的共享资源，如 Launch 文件和配置文件。src 文件夹是存放功能包源代码的地方，而 devel 文件夹则包含了开发过程中生成的可执行文件、库文件、头文件和共享资源。

常见的文件夹如 src 和 devel 在 ROS 开发中扮演着重要角色。src 文件夹是开发者编写和修改代码的地方，而 devel 文件夹则包含了编译后的结果，使得开发者可以运行和测试他们的代码。了解这些文件夹的主要功能对于有效地管理和维护 ROS 项目至关重要。通过掌握 ROS 的工作空间、功能包、节点、话题、消息、服务、参数服务器和启动文件等基本概念，以及系统中的文件结构和常见文件夹的功能，开发者能够更加高效地开发、调试和部署机器人应用。

2 系统整体设计方案

本系统旨在通过手势识别技术，利用摄像头捕捉手势并将其转换为 ROS 控制命令和位置信息，进而实现对 turtlesim 仿真环境中海龟机器人的精确控制。系统主要分为两个模块：手势识别模块(aivirtualmouse.py)和控制执行模块(turtlesim_controller.py)。

首先，在手势识别模块中，系统通过摄像头实时捕捉视频流，并使用 MediaPipe 手势识别模型来检测手部关键点。根据手指的状态（例如：拇指向上表示悬停，食指向上表示前进，中指向上表示左转，无名指向上表示右转，小指向上表示后退），手势识别模块会生成相应的控制命令和位置信息。这些命令和位置信息会被发布到 ROS 的各个话题，包括/aivirtualmouse_command、/aivirtualmouse_position、/aivirtualmouse_marker 和/path。同时，手势识别模块还会在 OpenCV 窗口中显示识别结果和帧率（FPS）信息，并在 RViz 中显示手势轨迹和路径。

在控制执行模块中，系统订阅手势识别模块发布的手势命令和位置信息。当接收到手势命令时，控制执行模块会生成相应的 Twist 消息，控制海龟机器人的线速度和角速度。例如，悬停命令会停止海龟机器人的移动，前进命令会增加线速度，左转命令会增加角速度，右转命令会减少角速度，后退命令会减少线速度。当接收到手势位置信息时，控制执行模块会将这些信息转换为海龟机器人的位姿，并发布到 turtlesim 的位姿话题。此外，控制执行模块还订阅路径信息，存储路径中的位置点，并实现路径跟随逻辑，控制海龟机器人按照路径移动。

启动系统时，需要同时运行手势识别模块和控制执行模块。手势识别模块通过摄像头捕捉视频流，识别手势并发布相关信息；控制执行模块订阅这些信息，根据手势命令和位置信息控制海龟机器人的移动，并处理路径信息以实现路径跟随功能。系统以 30 Hz 的频率运行，确保了实时性和响应速度。通过模块化设计，系统具有较高的灵活性和可扩展性，支持增加更多的手势命令、提高手势识别精度、引入路径规划算法和多摄像头支持等功能。

通过上述设计，本系统不仅能够实现基于手势识别的机器人控制，还能在 RViz 中直观地展示手势轨迹和路径，为机器人仿真实验提供了一种创新的交互方式。

3 系统软件设计

3.1系统软件总体设计

本系统利用手势识别技术，通过摄像头捕捉手势并转换为 ROS 控制命令和位置信息，实现对 turtlesim 仿真中海龟机器人的精确控制。系统分为两个模块：手势识别模块(aivirtualmouse.py)和控制执行模块(turtlesim_controller.py)。在手势识别模块中，系统实时捕捉视频流，使用 MediaPipe 模型检测手部关键点，根据手指状态生成控制命令和位置信息，发布到 ROS 话题。模块在 OpenCV 窗口显示识别结果和 FPS，并在 RViz 中展示手势轨迹和路径。控制执行模块订阅手势命令和位置信息，生成 Twist 消息控制海龟机器人速度和方向。接收位置信息后，转换为位姿并发布到 turtlesim。模块还处理路径信息，实现路径跟随。启动系统时，需同时运行两个模块。手势识别模块捕捉手势并发布信息，控制执行模块根据这些信息控制海龟机器人移动。系统以 30 Hz 频率运行，保证实时性。模块化设计提高灵活性和可扩展性，支持功能增强。通过此设计，系统实现手势识别机器人控制，并在 RViz 中展示手势轨迹和路径，为机器人仿真提供新交互方式。(具体实现流程图如图 1 系统软件整体设计方案)

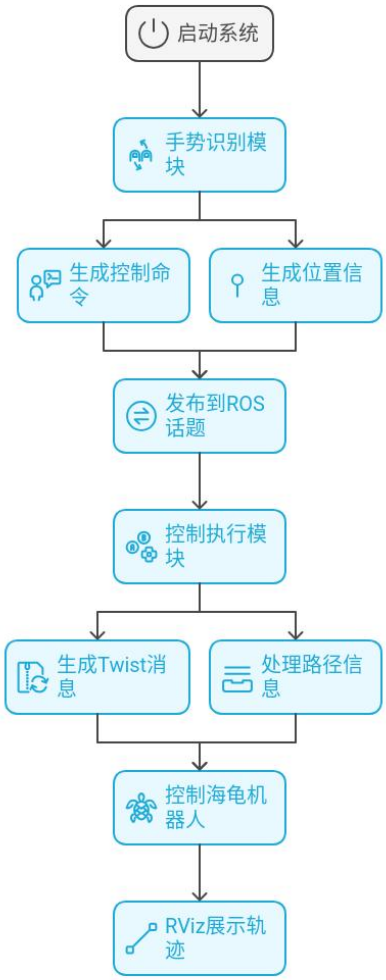

图 1 系统软件整体设计方案

3.2手势识别模块

3.2.1 功能描述

手势识别模块的主要功能是通过摄像头实时捕捉视频流并识别手势，将识别到的手势转换为 ROS 机器人控制命令和位置信息。该模块还会将手势信息和路径发布到 ROS 话题，并在 RViz 中显示手势轨迹。通过这些功能，手势识别模块为控制执行模块提供了实时的手势识别和位置数据，使海龟机器人能够根据用户的手势进行精确的移动控制。（具体实现框图如图 2 手势识别模块框图）

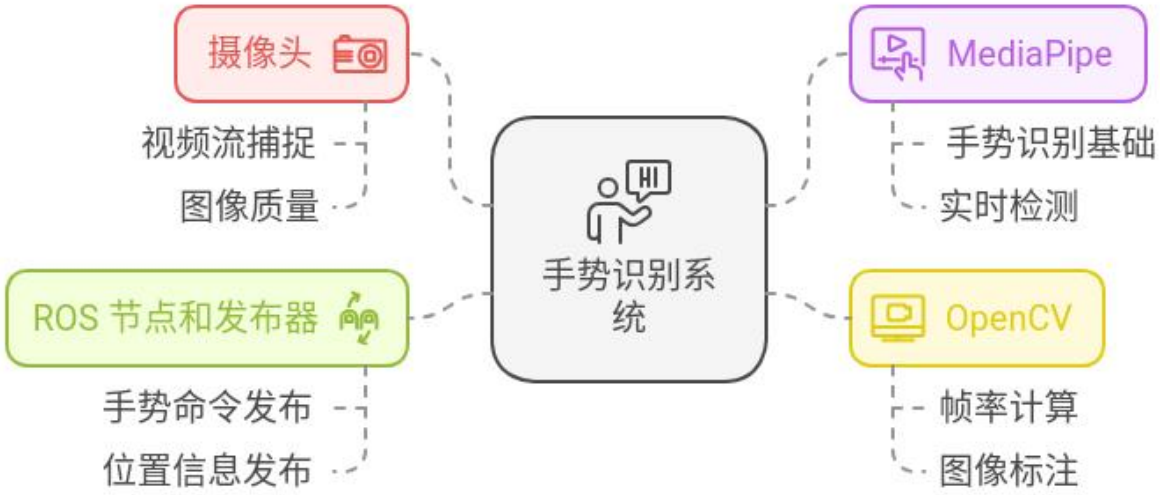

图 2 手势识别模块框图

主要组件

摄像头：用于捕捉视频流。为了确保图像质量和处理速度，摄像头的分辨率设置为 640x480。

MediaPipe：用于手部关键点的检测，提供手势识别的基础。MediaPipe 是一个高效的机器学习框架，能够实时检测和跟踪手部的各个关键点。

OpenCV：用于图像处理和显示。包括帧率计算、图像标注等功能。OpenCV 是一个强大的计算机视觉库，能够处理从摄像头获取的图像数据并进行实时标注。

ROS 节点和发布器：用于发布手势命令、位置信息、标记和路径。ROS 节点负责与 ROS 系统通信，发布器则将识别到的手势和位置信息发布到相应的 ROS 话题。

3.2.2 详细设计

初始化

设置摄像头分辨率和帧率：模块启动时，首先设置摄像头的分辨率为 640x480，并配置帧率为 30Hz，以确保图像质量和处理速度。

初始化 MediaPipe 手势识别模型：加载 MediaPipe 手势识别模型，该模型能够实时检测手部的关键点，为手势识别提供基础。初始化 ROS 节点和发布器：命令发布器：创建一个发布器，用于将手势命令发布到/aivirtualmouse_command 话题。位置发布器：创建一个发布器，用于将手势位置信息发布到

/aivirtualmouse_position 话题。标记发布器：创建一个发布器，用于在 RViz 中显示手势轨迹，通过发布 Marker 消息实现。路径发布器：创建一个发布器，用于发布手势位置的路径信息，通过 Path 消息记录并发布到/path 话题。

具体实现部分代码如图 3 手势识别初始化部分代码：

```
# Camera settings
wCam, hCam = 640, 480
frameR = 100
smoothening = 5

# Initialize Mediapipe
cap = cv2.VideoCapture(0)
cap.set(3, wCam)
cap.set(4, hCam)

mpHands = mp.solutions.hands
hands = mpHands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5)
mpDraw = mp.solutions.drawing_utils

# Initialize ROS node and publishers
rospy.init_node('aivirtualmouse_node', anonymous=True)
command_pub = rospy.Publisher('/aivirtualmouse_command', String, queue_size=10)
position_pub = rospy.Publisher('/aivirtualmouse_position', Twist, queue_size=10)
marker_pub = rospy.Publisher('/aivirtualmouse_marker', Marker, queue_size=10)
path_pub = rospy.Publisher('/path', Path, queue_size=10)

rate = rospy.Rate(30)  # Loop at 30 Hz
marker_id = 0  # Global variable for unique marker IDs
path = Path()
path.header.frame_id = "world"
```

图 3 手势识别初始化部分代码

手势识别：

读取摄像头帧：通过摄像头实时捕捉视频流，并读取每一帧图像。为了适应 MediaPipe 模型的输入要求，将读取到的 BGR 格式图像转换为 RGB 格式。

使用 MediaPipe 模型检测手部关键点：将处理后的图像输入到 MediaPipe 手势识别模型中，检测手部的关键点，生成手部 landmarks。

根据手指的状态判断手势命令：拇指向上：当检测到拇指向上时，发布"hover"命令，使海龟机器人停止移动。食指向上：当检测到食指向上时，发布"forward"命令，使海龟机器人向前移动。同时，获取食指的坐标作为位置信息。中指向上：当检测到中指向上时，发布 left 命令，使海龟机器人左转。无名指向上：当检测到无名指向上时，发布"right"命令，使海龟机器人右转。小指向上：当检测到小指向上时，发布"backward"命令，使海龟机器人后退。

具体实现部分代码如图图 4 手势识别部分代码：

```
global marker_id
pTime = 0  # For FPS calculation
while not rospy.is_shutdown():
    ret, frame = cap.read()
    if not ret:
        rospy.logerr("Failed to capture frame from camera")
        continue

    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    rgb_frame.flags.writeable = False

    results = hands.process(rgb_frame)

    rgb_frame.flags.writeable = True
    frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

    if results.multi_hand_landmarks:
        for hand_id, handLms in enumerate(results.multi_hand_landmarks):
            fingers = fingers_up(handLms)
            rospy.loginfo(f"Fingers up: {fingers}")

            if fingers == [True, False, False, False, False]:  # Thumb up
                publish_command("hover")
            elif fingers == [False, True, False, False, False]:  # Index finger
                publish_command("forward")
                x = handLms.landmark[8].x * wCam
                y = handLms.landmark[8].y * hCam
                publish_position(x, y)
                publish_marker(x, y)
                publish_path(x, y)  # 添加路径发布
            elif fingers == [False, False, True, False, False]:  # Middle finger
                publish_command("left")
            elif fingers == [False, False, False, True, False]:  # Ring finger
                publish_command("right")
            elif fingers == [False, False, False, False, True]:  # Pinky finger
                publish_command("backward")
            else:
                publish_command("hover")

            draw_hands(frame, handLms)
    else:
        rospy.loginfo("No hand detected")
        publish_command("hover")
```

图 4 手势识别部分代码

信息发布

发布手势命令：将识别到的手势命令发布到/aivirtualmouse_command 话题，控制执行模块会订阅该话题并根据命令控制海龟机器人的移动。

发布手势位置信息：当检测到食指向上时，将食指的坐标转换为米制单位，并发布到/aivirtualmouse_position 话题。控制执行模块会订阅该话题，并将位置信息转换为海龟机器人的位姿。

在 RViz 中显示手势轨迹：通过发布 Marker 消息到/aivirtualmouse_marker 话题，在 RViz 中显示手势轨迹。这样可以直观地展示手势的运动路径，便于调试和展示。

发布路径信息：通过发布 Path 消息到/path 话题，记录手势位置的路径信息。这些路径信息可以被控制执行模块订阅，实现路径跟随功能。

图像显示:在 OpenCV 窗口中显示实时视频流：使用 OpenCV 将处理后的图像显示在窗口中，以便用户实时查看手势识别的结果。标注手部关键点和连接线：在显示的图像中标注手部的关键点和连接线，帮助用户理解识别结果。显示帧率（FPS）信息：在图像中显示帧率信息，帮助调试和评估系统的性能。通过计算每秒处理的帧数，可以确保系统的实时性和响应速度。

信息发布部分代码展示如图 5 信息发布部分代码：

```python
def publish_marker(x, y, z=0.0):
    marker = Marker()
    marker.header.frame_id = "world"  # 确保和 RViz 的全局框架一致
    marker.header.stamp = rospy.Time.now()
    marker.ns = "aivirtualmouse"
    marker.id = 0  # Marker 的唯一 ID
    marker.type = Marker.SPHERE  # 设置为球形
    marker.action = Marker.ADD  # 确保是添加操作

    # 设置位置
    marker.pose.position.x = x / 100.0
    marker.pose.position.y = y / 100.0
    marker.pose.position.z = z
    marker.pose.orientation.x = 0.0
    marker.pose.orientation.y = 0.0
    marker.pose.orientation.z = 0.0
    marker.pose.orientation.w = 1.0

    # 设置 Marker 的尺寸
    marker.scale.x = 0.1  # 确保尺寸大于零
    marker.scale.y = 0.1
    marker.scale.z = 0.1

    # 设置 Marker 的颜色
    marker.color.a = 1.0  # 透明度 (1.0 为不透明)
    marker.color.r = 1.0  # 红色
    marker.color.g = 0.0  # 绿色
    marker.color.b = 0.0  # 蓝色

    # 发布 Marker
    marker_pub.publish(marker)
    print(f"Published Marker: x={x / 100.0}, y={y / 100.0}, z={z}")
```

图 5 信息发布部分代码

通过以上设计，手势识别模块能够实时捕捉和识别手势，并将识别结果通过 ROS 话题传递给控制执行模块，同时在 RViz 和 OpenCV 窗口中提供直观的可视化展示，确保系统具有高效性和灵活性。

3.3 控制执行模块

3.3.1 功能描述：

控制执行模块的主要功能是订阅手势识别模块发布的手势命令和位置信息，并根据这些信息控制 turtlesim 中的海龟机器人移动。此外，该模块还将手势位置信息转换为海龟机器人的位姿，并发布到 turtlesim 的位姿话题。同时，它还处理路径信息，实现路径跟随功能。通过这些功能，控制执行模块能够将手势识别模块的输出转化为具体的机器人控制行为，确保海龟机器人能够根据用户的手势进行精确的移动。（具体模块实现框图如图 6 控制执行模块框图）

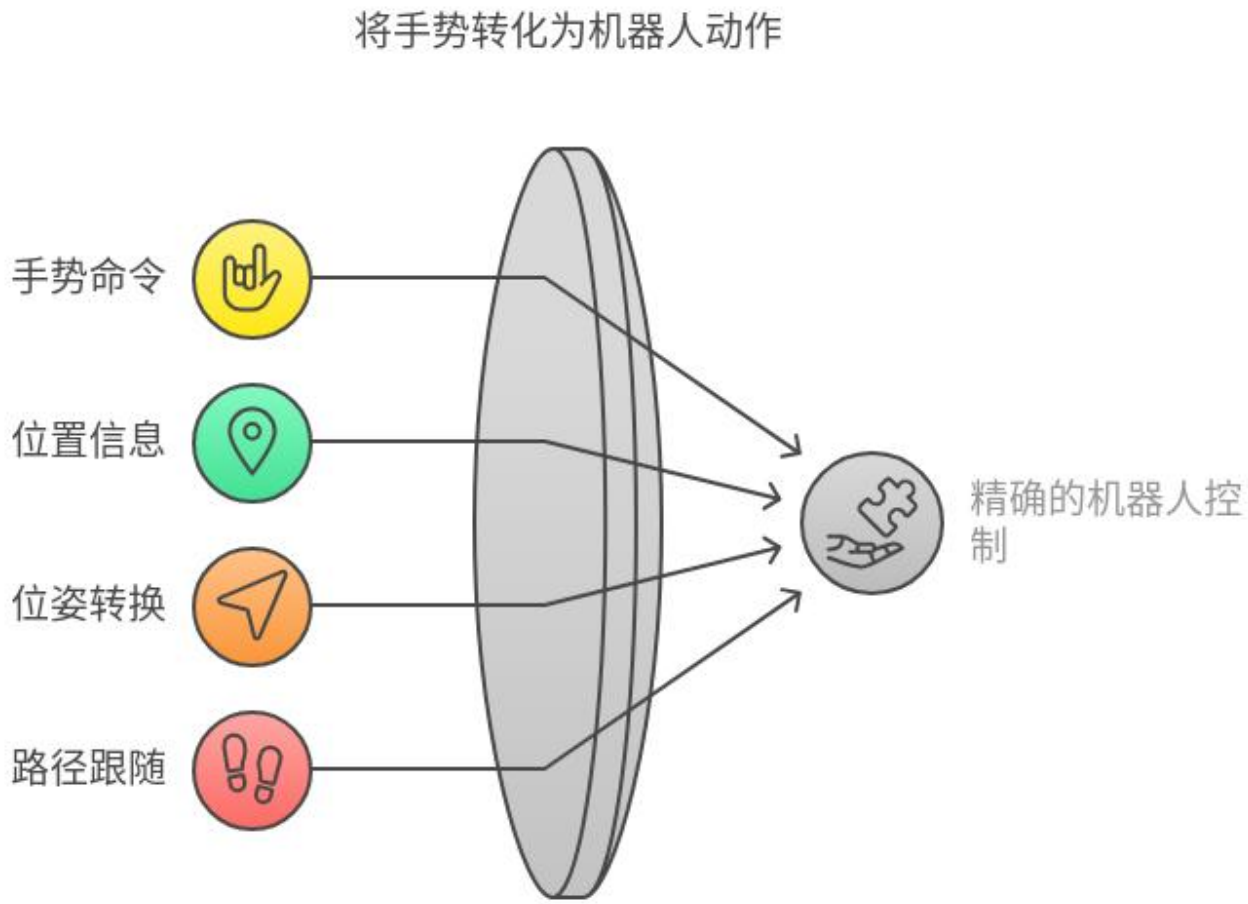

图 6 控制执行模块框图

3.3.2 详细设计

初始化：在初始化阶段，首先创建一个名为 turtlesim_controller 的 ROS 节点，并设置节点的初始化参数，如节点名称和频率。接着，初始化两个发布器：cmd_vel 发布器用于发布 Twist 消息，控制海龟机器人的速度和方向；pose 发布器用于发布 Pose 消息，控制海龟机器人的位姿。同时，初始化三个订阅器：手势命令订阅器订阅/aivirtualmouse_command 话题，接收手势识别模块发布的手势命令；

手势位置订阅器订阅/aivirtualmouse_position 话题，接收手势识别模块发布的手势位置信息；路径订阅器订阅/path 话题，接收手势识别模块发布的路径信息。（具体实现代码如图 7 控制初始化代码）

```
def __init__(self):
    rospy.init_node('turtlesim_controller', anonymous=True)
    self.cmd_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
    self.pose_pub = rospy.Publisher('/turtle1/pose', Pose, queue_size=10)  # 添加位姿发布器
    rospy.Subscriber('/aivirtualmouse_command', String, self.command_callback)
    rospy.Subscriber('/aivirtualmouse_position', Twist, self.position_callback)
    rospy.Subscriber('/path', Path, self.path_callback)  # 订阅路径话题
    self.rate = rospy.Rate(30)  # 30 Hz
    self.current_pose = Pose()  # 初始化当前位姿
    self.path = None  # 初始化路径
```

图 7 控制初始化代码

命令处理：控制执行模块订阅/aivirtualmouse_command 话题，根据接收到的手势命令生成相应的 Twist 消息。支持的命令包括：“hover”命令停止海龟机器人的移动；“forward”命令增加线速度，使海龟机器人向前移动；“left”命令增加角速度，使海龟机器人左转；“right”命令减少角速度，使海龟机器人右转；“backward”命令减少线速度，使海龟机器人后退。这些命令通过 cmd_vel 发布器发布到 turtlesim 的控制话题，实现对海龟机器人的实时控制。（具体实现代码如图 8 命令处理部分代码）

```
def command_callback(self, data):
    command = data.data
    twist = Twist()
    if command == "hover":
        twist.linear.x = 0.0
        twist.angular.z = 0.0
    elif command == "forward":
        twist.linear.x = 2.0  # 增加线速度
        twist.angular.z = 0.0
    elif command == "left":
        twist.linear.x = 0.0
        twist.angular.z = 4.0  # 增加角速度
    elif command == "right":
        twist.linear.x = 0.0
        twist.angular.z = -4.0  # 减少角速度
    elif command == "backward":
        twist.linear.x = -2.0  # 减少线速度
        twist.angular.z = 0.0
    else:
        twist.linear.x = 0.0
        twist.angular.z = 0.0
    self.cmd_vel_pub.publish(twist)
    print(f"Executed {command} command with linear.x={twist.linear.x} and angular.z={twist.angular.z}")
```

图 8 命令处理部分代码

位置处理：控制执行模块订阅/aivirtualmouse_position 话题，接收来自手势识别模块的手势位置信息。该模块将手势位置信息（食指的坐标）转换为海龟机器人的位姿，并发布到 turtlesim 的位姿话题。位置信息从米制单位转换为像素单位，便于在 turtlesim 中实际应用。具体来说，手势位置的 x 坐标和 y 坐标分别乘以 11/640 和 11/480，将手势位置转换为 turtlesim 中的位姿，然后通过 pose 发

布器发布这些位姿信息。（具体实现如图 9 位置处理部分代码）

```
def position_callback(self, data):
    x = data.linear.x * 100  # 将位置值转换为像素值
    y = data.linear.y * 100  # 将位置值转换为像素值
    print(f"Received position: x={x}, y={y}")
    self.current_pose.position.x = x
    self.current_pose.position.y = y
    self.pose_pub.publish(self.current_pose)  # 发布当前位姿
```

图 9 位置处理部分代码

路径处理：控制执行模块订阅/path 话题，接收来自手势识别模块的路径信息。路径中的每个位置点通过 PoseStamped 消息表示，并存储在 Path 消息中。该模块实现路径跟随逻辑，读取路径中的位置点，并根据这些位置点生成控制指令，使海龟机器人沿路径移动。路径跟随功能使用一个简单的 PID 控制器来计算线速度和角速度。具体来说，模块获取路径上的第一个点作为目标点，计算当前位姿与目标位姿之间的线距离和角距离，并根据这些距离生成相应的 Twist 消息。如果海龟机器人到达目标点，该点将从路径中移除，继续处理下一个目标点。通过这种方式，海龟机器人能够沿着预定的路径进行精确的移动。（具体实现如图 10 路径处理部分代码）

```
def path_callback(self, data):
    self.path = data
    print(f"Received path with {len(data.poses)} poses")
    for pose_stamped in data.poses:
        print(f"Pose at time {pose_stamped.header.stamp}:")
        print(f"  Position: x={pose_stamped.pose.position.x}, y={pose_stamped.pose.position.y}, z={pose_stamped.pose.position.z}")
        print(f"  Orientation: x={pose_stamped.pose.orientation.x}, y={pose_stamped.pose.orientation.y}, z={pose_stamped.pose.orientation.z}, w={pose_stamp

def run(self):
    while not rospy.is_shutdown():
        if self.path:
            # 处理路径逻辑，例如跟随路径
            pass
        self.rate.sleep()
```

图 10 路径处理部分代码

主循环：在主循环中，控制执行模块定期检查是否有路径信息。如果有路径信息，模块将调用路径处理逻辑，计算并发布相应的控制指令。主循环的频率设置为 30Hz，确保系统的实时性和响应速度。在每次循环中，如果路径中的位置点未被完全处理，模块将继续向海龟机器人发送控制指令，使它沿路径移动。一旦路径中的所有位置点都被处理完毕，模块将停止海龟机器人的移动。主循环通过 rospy.Rate(30).sleep()保持稳定的运行频率，直到 ROS 系统关闭。通过这些设计，控制执行模块能够高效地处理手势识别模块的输出，实现对 turtlesim 中海龟机器人的精确控制。（具体实现如图 11 主循环代码）

```
if __name__ == "__main__":
    controller = TurtlesimController()
    controller.run()
```

图 11 主循环代码

4 系统调试结果与分析

4.1环境搭建：

硬件环境：

摄像头（可以正常工作的摄像头，用于捕捉视频流。）

计算机（内存至少拥有 8G 以上内存）

软件环境：

Linux 系统（搭载 ubuntu20.04,可以兼容或者部署 ros 系统）

Ros 系统版本为 Noetic。

Anconda。虚拟 Anconda 环境下载 opencv，MediaPipe。

仿真调试

4.1.1 手部关键点检测：

当伸出食指时，摄像头的返回的关键点食指伸出（如图 12 食指时摄像头的画面），并且终端会有提示检测到了数据（具体如图图 13 终端的数据显示）。

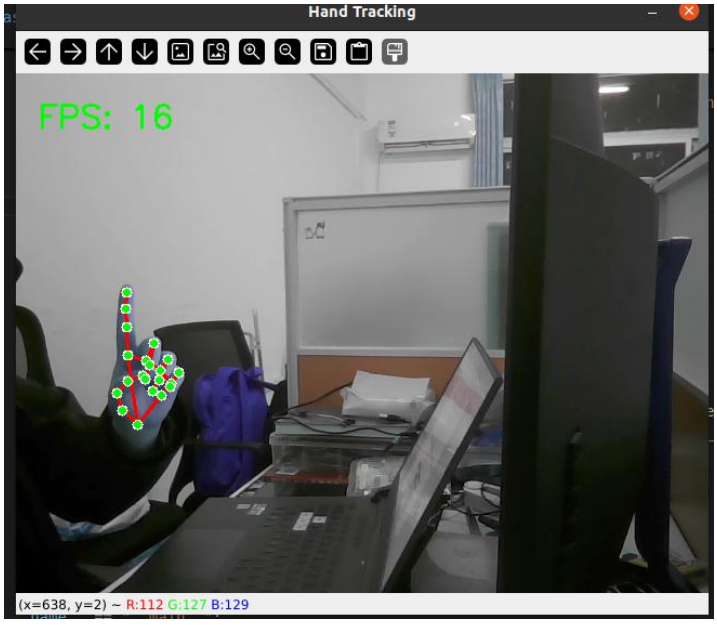

图 12 食指时摄像头的画面

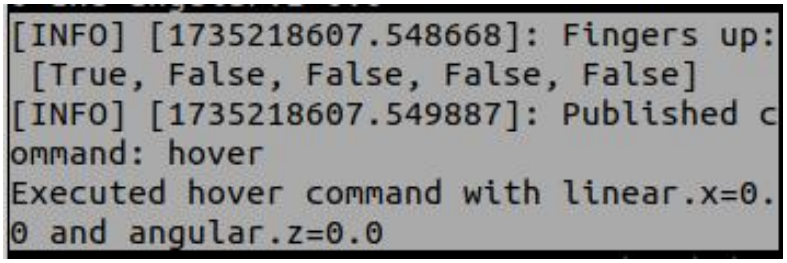

图 13 终端的数据显示

当伸出中指时，摄像头的返回的关键点中指伸出（如图 14 伸出中指时摄像头的画面），并且终端

会有提示检测到了数据（具体如图 15 终端的数据显示）。

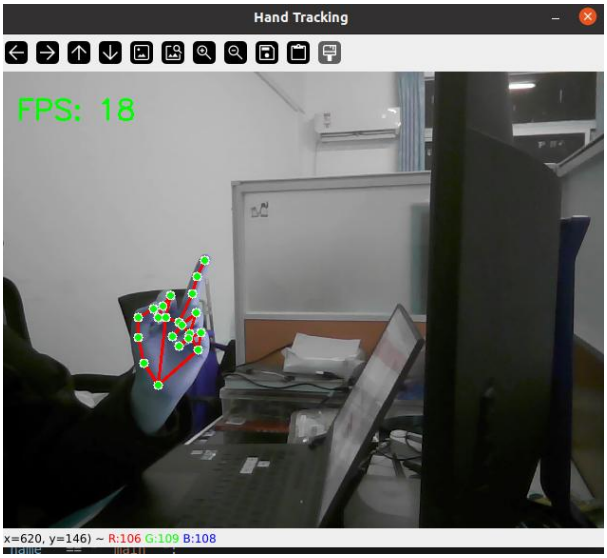

图 14 伸出中指时摄像头的画面

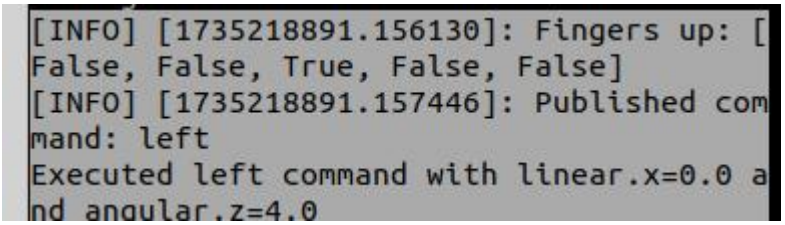

图 15 终端的数据显示

当伸出无名指时，摄像头的返回的关键点无名指伸出（如图 16 伸出无名指时摄像头的画面），并且终端会有提示检测到了数据（具体如图 17 终端的数据显示）。

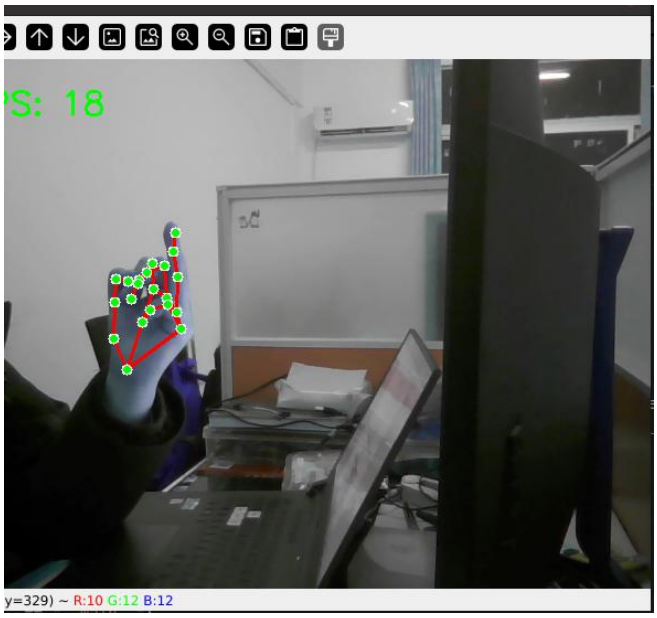

图 16 伸出无名指时摄像头的画面

```
[INFO] [1735219012.628959]: Fingers up:
[False, False, False, True, False]
[INFO] [1735219012.630205]: Published co
mmand: right
Executed right command with linear.x=0.0
 and angular.z=-4.0
```

图 17 终端的数据显示

当伸出小拇指时，摄像头的返回的关键点小拇指伸出（如图 18 伸出小拇指时摄像头的画面），并且终端会有提示检测到了数据（具体如图 19 终端的数据显示）。

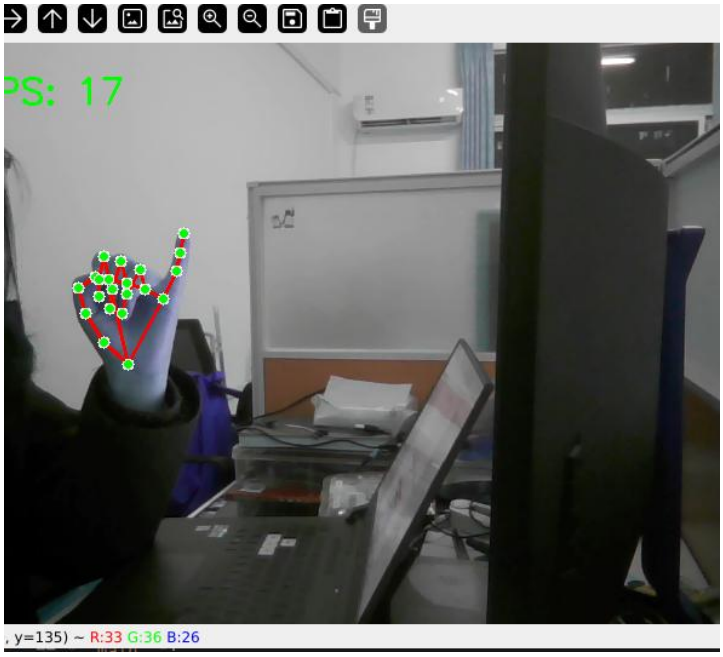

图 18 伸出小拇指时摄像头的画面

```
[INFO] [1735219147.287687]: Fingers up: [
False, False, False, False, True]
[INFO] [1735219147.288946]: Published com
mand: backward
Executed backward command with linear.x=-
2.0 and angular.z=0.0
```

图 19 终端的数据显示

根据以上代码，可以见得手指检测模块程序实现没有任何问题。

4.1.2 控制执行模块的检测

当启动节点时，乌龟最开始的位姿都是头偏向屏幕的右边（具体效果如图 20 乌龟初始位姿）。

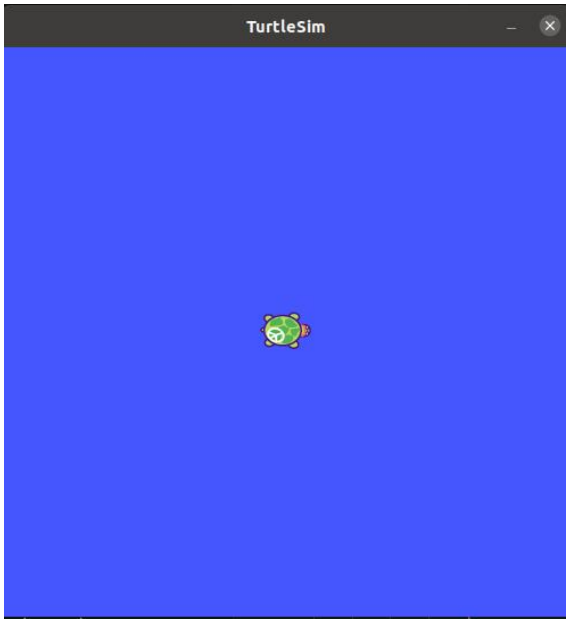

图 20 乌龟初始位姿

当伸出食指时，摄像头的返回的关键点食指伸出，并且终端会有提示检测到了数据（具体如图 21 终端信息），小乌龟前进（具体效果如图 22 小乌龟的位置变化）。

```
[INFO] [1735218607.548668]: Fingers up:
 [True, False, False, False, False]
[INFO] [1735218607.549887]: Published c
ommand: hover
Executed hover command with linear.x=0.
0 and angular.z=0.0
```

图 21 终端信息

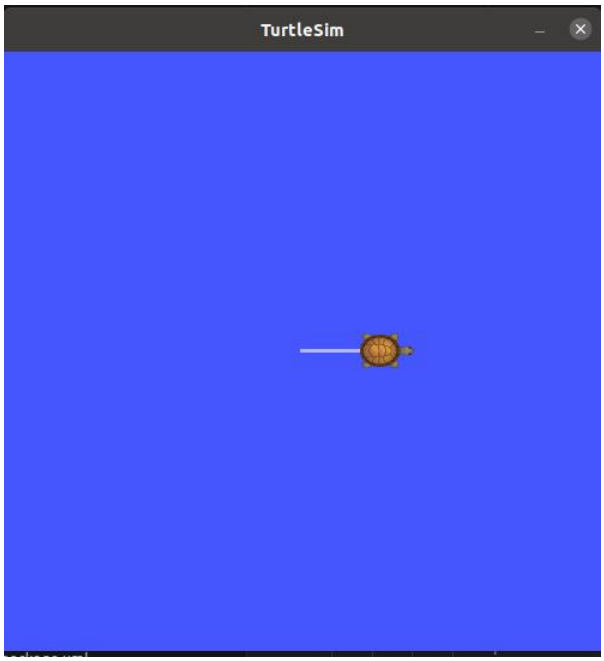

图 22 小乌龟的位置变化

当伸出中指时，摄像头的返回的关键点中指伸出，并且终端会有提示检测到了数据（具体如图 23 终端信息），小乌龟向左转（具体效果如图 24 小乌龟向左）。

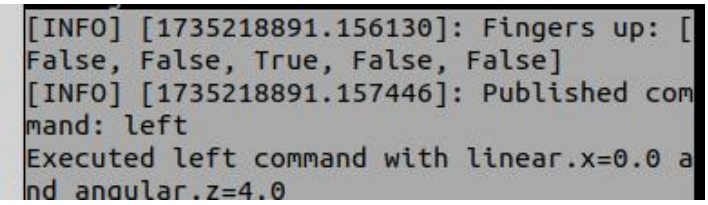

图 23 终端信息

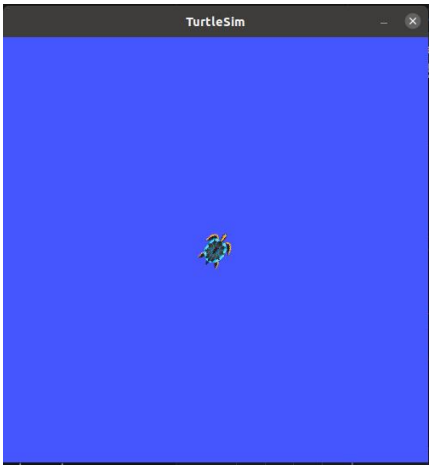

图 24 小乌龟向左

当伸出无名指时，摄像头的返回的关键点无名指伸出，并且终端会有提示检测到了数据（具体如图 25 终端信息），小乌龟向右转（具体效果如图 26 小乌龟向右转）。

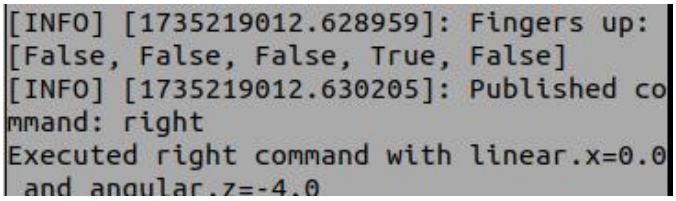

图 25 终端信息

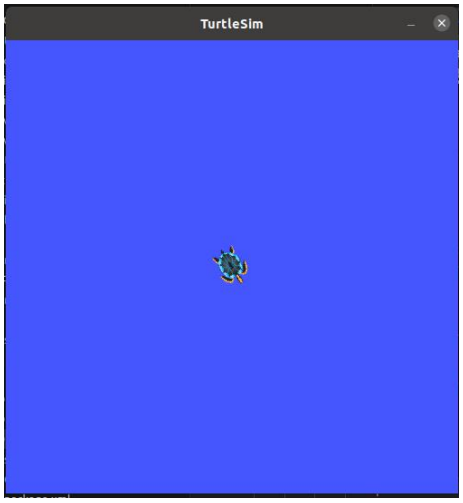

图 26 小乌龟向右转

当伸出小拇指时，摄像头的返回的关键点小拇指伸出，并且终端会有提示检测到了数据（具体如图 27 终端信息），小乌龟向后退（具体效果如图 28 小乌龟向后退）。

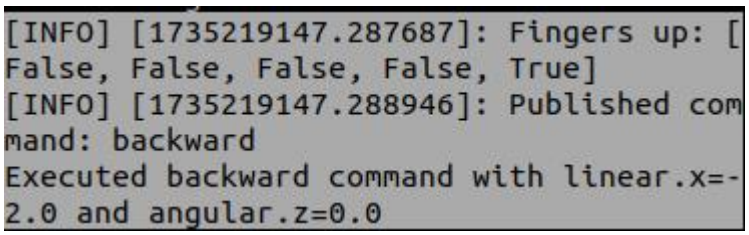

图 27 终端信息

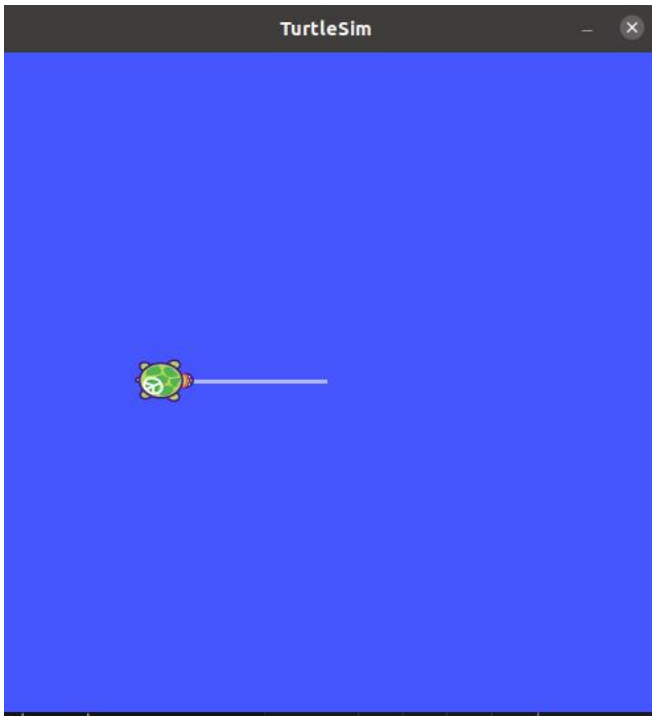

图 28 小乌龟向后退

经过这一轮测试，发现小乌龟的运动控制并没有任何问题，可以实现目标想要的前进后退转弯的效果。

经过共计两轮的检测，发现，小乌龟通过手势来实现前进后退转弯的功能并无大碍，可以继续进行使用。

4.2系统实现

创建 ROS 功能包，添加相关依赖，切换到 src 目录，创建功能包：catkin_create_pkg bigwork rospy cv2 mediapipe geometry_msgs std_msgs nav_msgs visualization_msgs，切换到功能包目录：cd bigwork，创建脚本目录：mkdir scripts。（实际代码实现如图 29 功能包的相关依赖、图 30 package 的相关依赖）

```
find_package(catkin REQUIRED COMPONENTS
  cv_bridge
  geometry_msgs
  rospy
  sensor_msgs
  std_msgs
  std_srvs
  turtlesim
  message_generation
)
```

图 29 功能包的相关依赖

```
buildtool_depend>catkin</buildtool_depend>
build_depend>cv_bridge</build_depend>
build_depend>geometry_msgs</build_depend>
build_depend>rospy</build_depend>
build_depend>sensor_msgs</build_depend>
build_depend>std_msgs</build_depend>
build_depend>std_srvs</build_depend>
build_depend>turtlesim</build_depend>
build_depend>message_generation</build_depend>
build_export_depend>cv_bridge</build_export_depend>
build_export_depend>geometry_msgs</build_export_depend>
build_export_depend>rospy</build_export_depend>
build_export_depend>sensor_msgs</build_export_depend>
build_export_depend>std_msgs</build_export_depend>
build_export_depend>std_srvs</build_export_depend>
build_export_depend>turtlesim</build_export_depend>
build_export_depend>message_generation</build_export_depend>
exec_depend>cv_bridge</exec_depend>
exec_depend>geometry_msgs</exec_depend>
exec_depend>rospy</exec_depend>
exec_depend>sensor_msgs</exec_depend>
exec_depend>std_msgs</exec_depend>
exec_depend>std_srvs</exec_depend>
exec_depend>turtlesim</exec_depend>
exec_depend>message_runtime</exec_depend>
exec_depend>python3-mediapipe</exec_depend>
exec_depend>python3-opencv</exec_depend>
exec_depend>python3-numpy</exec_depend>

!-- System dependencies -->
build_depend>Boost</build_depend>
exec_depend>Boost</exec_depend>
```

图 30 package 的相关依赖

编写手势识别节点脚本,创建 aivirtualmouse_node.py 脚本：在 scripts 目录中创建 aivirtualmouse_node.py。编写手势识别逻辑，使用 Mediapipe 捕捉手部姿势，并发布命令、位置、标记和路径话题。编写控制器节点脚本,创建 turtlesim_controller.py 脚本：在 scripts 目录中创建 turtlesim_controller.py。编写订阅手势识别节点发布的话题的逻辑，并根据接收到的命令和位置信息控制 turtlesim 中的海龟移动。（实际代码目录如图 31 实际代码目录）

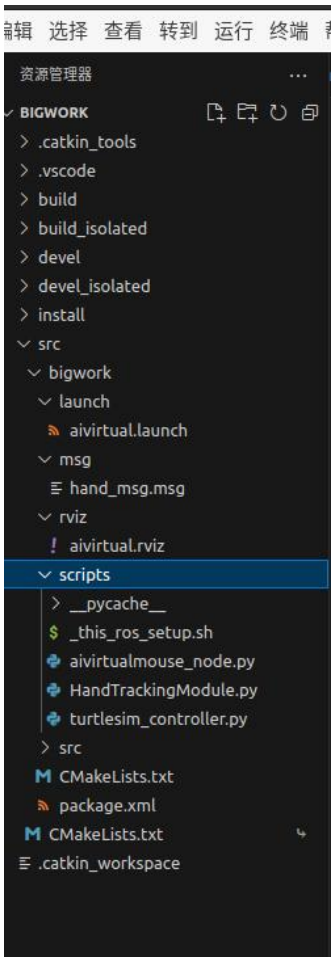

图 31 实际代码目录

编写 ROS 启动文件，创建 aivirtual.launch 文件：在 bigwork 功能包的 launch 目录中创建 aivirtual.launch。在文件中定义启动 turtlesim_node、aivirtualmouse_node、turtlesim_controller 和 rviz 的节点配置。（实际代码实现如）

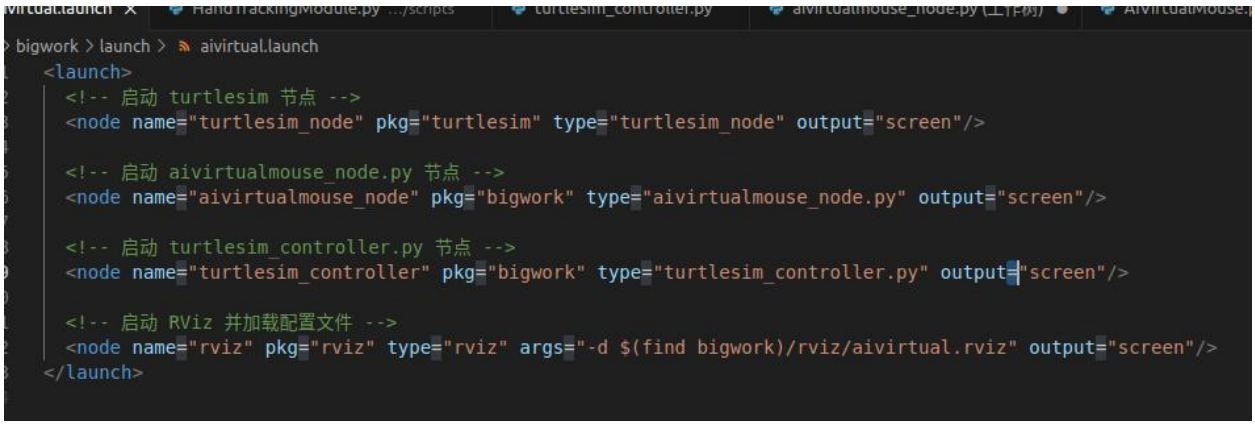

图 32 实际 launch 文件代码编写

编写 RViz 配置文件，创建 aivirtual.rviz 文件：在 bigwork 功能包的 rviz 目录中创建 aivirtual.rviz。配置 RViz 以显示手势识别的标记和路径（实际 rviz 代码实现如图 33 rviz 实际代码实现）。

```
<Configuration version="0.3">
  <GlobalOptions
    back_face_cull="false"
    background_color="48;48;48"
    background_color_alpha="1.0"
    fixed_frame="world"
    light_enabled="true"
    light_ground_plane_only="false"
    laundry_remote_md5="58f85c7a17c051b9c1f24461cc82b626"
    measured_frame_rate="29"
    minimum_frame_rate="29"
    multiplier="1.0"
    rog_alpha="0.0"
    rog_enabled="false"
    roi_mocap_in_use="false"
    screen_res="1920x1080"
    stereo_enabled="false"
    stereo_focal_distance="1.0"
    stereo_frame_id="unregistered"
    terrain_enabled="false"
    terrain_texture_id="unregistered"
    window_fullscreen="false"
  />

  <Scope name="My Visualizations">
    <Display name="/visualization.Markers" type="marker">
      <BoolProperty name="Approximate" value="false"/>
      <ColorProperty name="Color" value="255;0;0;255"/>
      <FloatProperty name="Duration" value="0"/>
      <StringProperty name="Frame ID" value="world"/>
      <StringProperty name="Marker ID" value="-1"/>
      <StringProperty name="Marker ID propagation Policy" value="拒绝"/>
      <StringProperty name="NS" value=""/>
      <BoolProperty name="Queue queue Action" value="false"/>
      <BoolProperty name="RGBA Scale enable" value="false"/>
      <FloatProperty name="RGBA Scale Value" value="1.0"/>
      <DoubleProperty name="Scale factor" value="1.0"/>
      <!-- 其他可能的属性配置 -->
    </Display>
  </Scope>
</Configuration>
</rviz>
```

图 33 rviz 实际代码实现

实际运行效果实现：

先启动 ros 的核心，使用 roscore 进行启动，具体实现如图 34 roscore 启动。

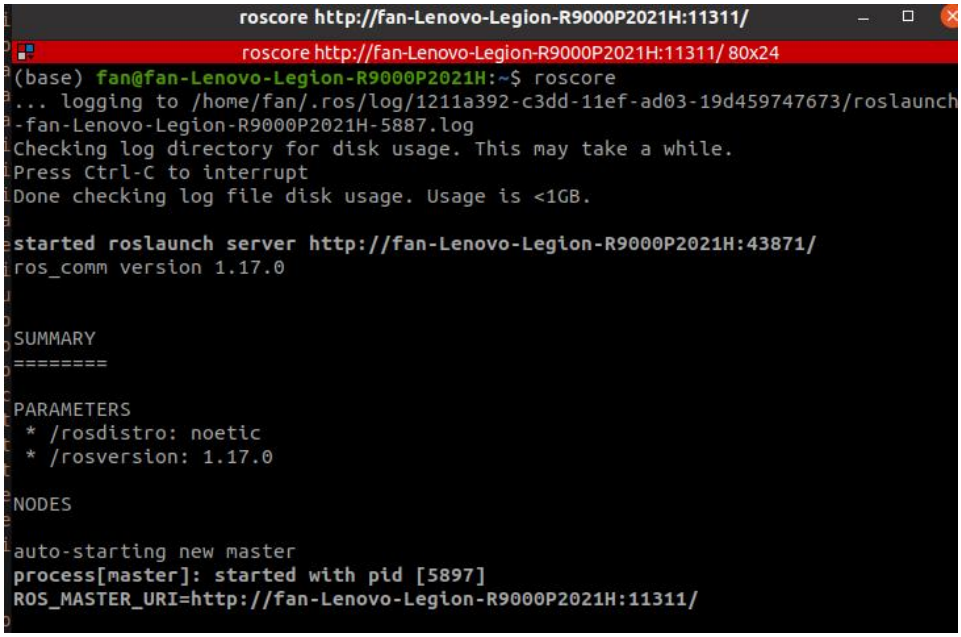

图 34 roscore 启动

对进入工作空间，将功能包进行编译，source 空间，然后运行 roslaunch 启动程序，具体实现如图 35 进入工作空间、编译、 36 source 并启动相关程序。

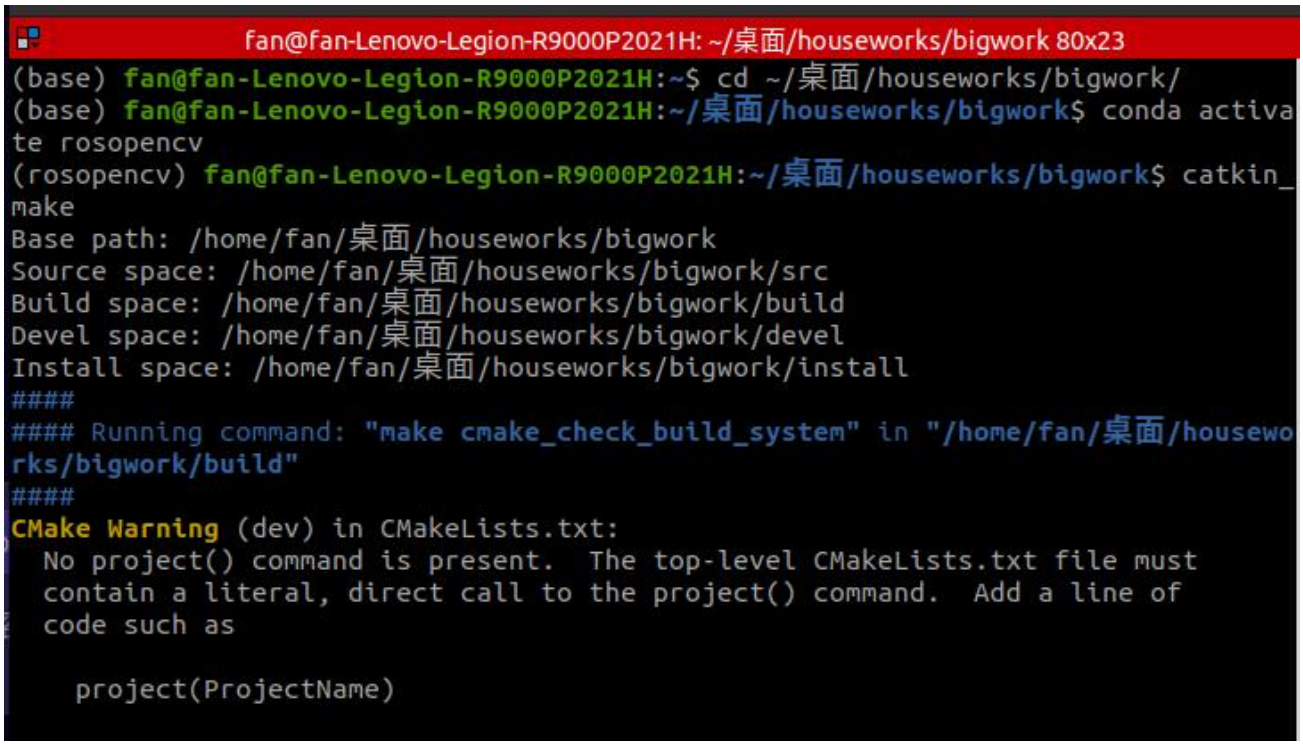

图 35 进入工作空间、编译

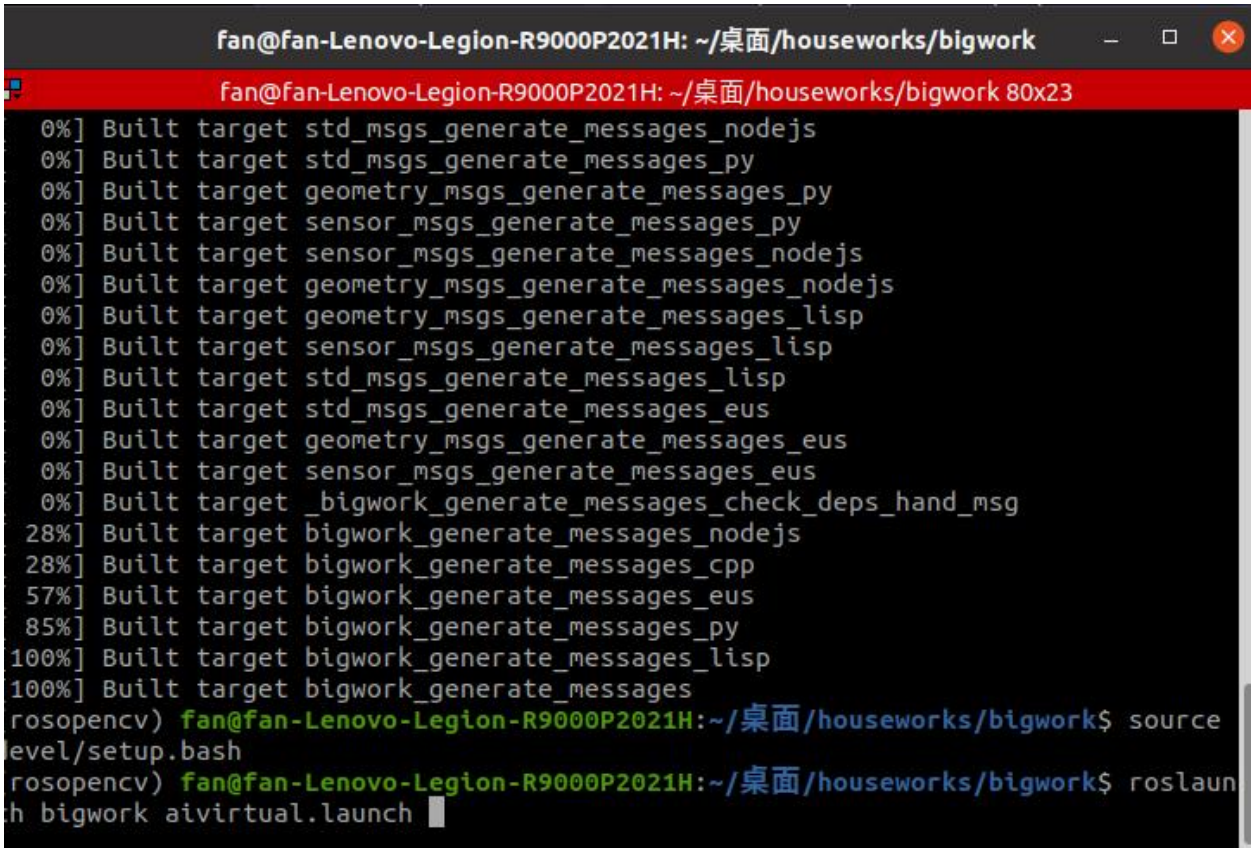

图 36 source 并启动相关程序

Rviz 的试图可视化处理，添加相关框图，然后添加相关话题进行通信，实现可视化。具体实现如图 37 添加显示窗格、图 38 添加显示物块和路径的组件。

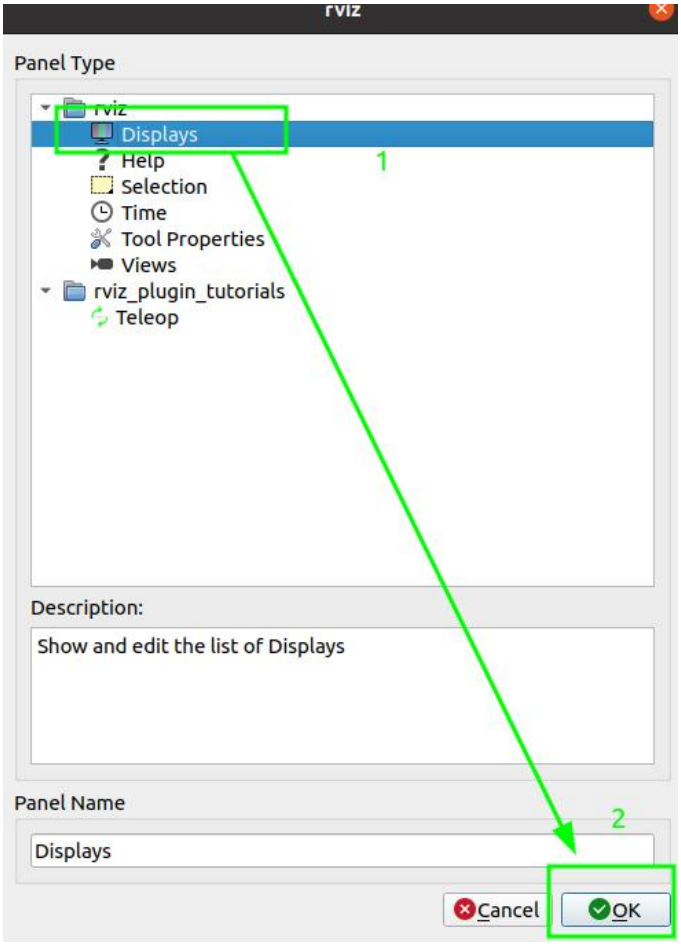

图 37 添加显示窗格

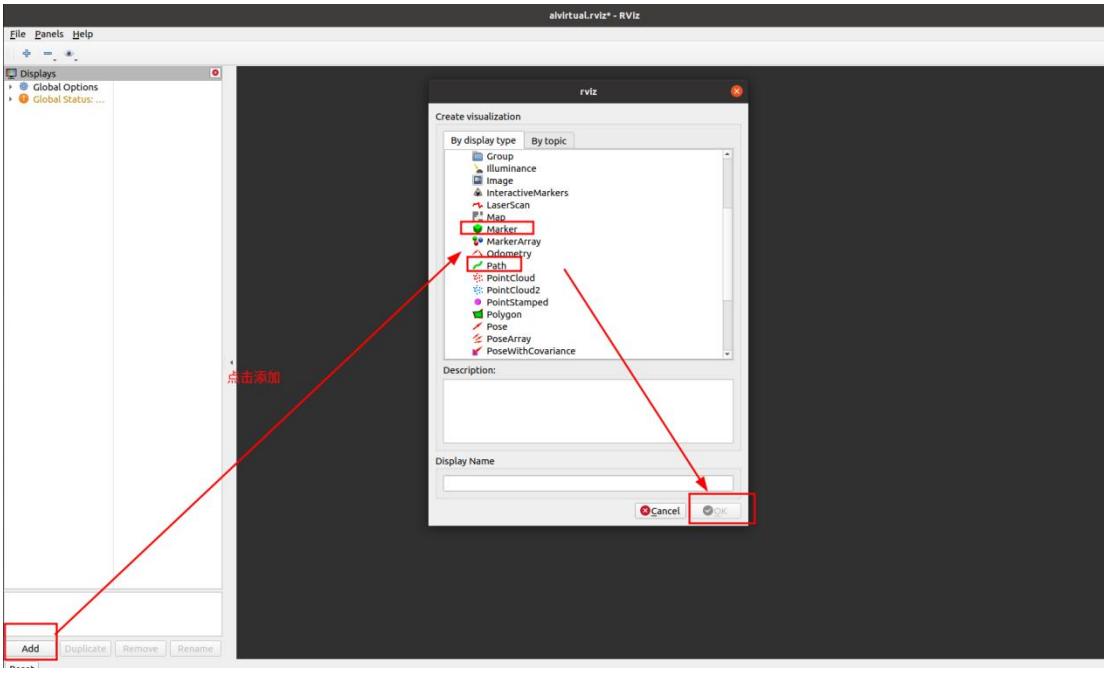

图 38 添加显示物块和路径的组件

最后可以实现的效果（具体实现如图 39 最终实现的效果图）：

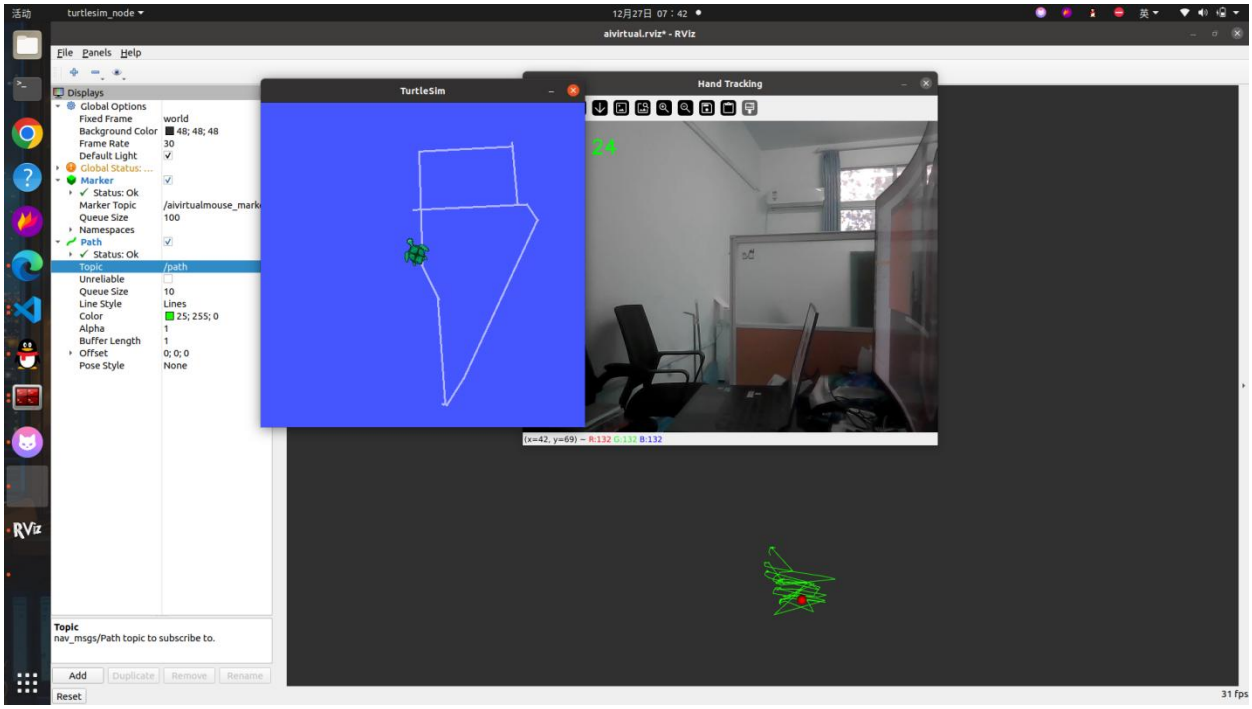

图 39 最终实现的效果图

5 总结

本系统主要用于通过手势识别来控制 turtlesim 节点中的海龟在屏幕上移动，并在 RViz 中进行可视化。具体实现了以下功能：首先，使用摄像头和 Mediapipe 库识别用户的手部姿势，检测手指是否抬起。其次，根据识别到的手势，发布相应的控制命令（悬停、前进、左转、右转、后退）到 ROS 话题 /aivirtualmouse_command 。如果食指抬起，还会发布食指的位置信息到 ROS 话题 /aivirtualmouse_position。然后，turtlesim_controller 节点订阅这些话题，根据接收到的命令和位置信息，控制 turtlesim 中的海龟移动。最后，将手势识别的路径信息发布到 ROS 话题/path，并在 RViz 中实时显示手势路径和标记。

尽管该系统已经实现了一些基本功能，但仍有一些待改进的地方。首先，手势识别的准确性在复杂背景和光线条件下可能会有所下降，可以考虑引入更多的训练数据和优化算法，提高识别精度。其次，当前系统发布的命令较为简单，可以增加更多复杂的控制命令，如旋转角度调节、加速度控制等，使海龟的控制更加精细和多样化。此外，用户界面的友好性也有待提升，当前系统主要依赖命令行和 RViz 进行调试和可视化，可以开发一个图形用户界面（GUI），方便用户进行参数设置和状态监控。性能方面，手势识别和图像处理可能会占用较多的计算资源，可以优化代码，减少延迟，提高系统响应速度。最后，系统目前只支持单用户操作，可以考虑扩展为支持多个用户同时通过手势控制多个海龟，增强系统的互动性和可扩展性。

通过开发和实现这个基于手势识别的虚拟鼠标控制器系统，我深刻体会到了 ROS 框架的强大和灵活性。ROS 提供的节点通信机制使得不同功能模块可以独立开发和测试，最终通过发布和订阅话题的方式无缝集成。这一过程不仅提高了开发效率，还增强了系统的模块化和可维护性。同时，使用 Mediapipe 进行手势识别，让我认识到现代计算机视觉技术在实际应用中的高效性和可靠性。尽管系统在一些方面还存在改进的空间，但通过不断学习和优化，我相信这些不足可以逐步得到解决。这个项目不仅提升了我的编程和系统集成能力，还加深了我对人机交互领域的兴趣和理解。整体而言，这次开发经历让我更加自信地应对复杂的工程项目，并激发了我在机器人技术领域的进一步探索。

参 考 文 献

[1] 梁健. 基于 ROS 可更换式末端执行器采摘机器人研究与应用[D]. 扬州大学,2024.DOI:10.27441/d.cnki.gyzdu.2024.000880.

[2]涂虬,赵磊,王锋,等.嵌入式系统中机器人操作系统(ROS)多机通信图像传输效率改善的研究[J].上饶师范学院学报,2023,43(06):20-27.

[3]侯人鸾,佟琨,韩冰,等.智能机器人操作系统研究综述[J].杭州科技,2023,54(03):58-64.

[4] 高振华. 基于 ROS 的四舵轮全向引导车路径规划技术研究[D]. 哈尔滨商业大学,2023.DOI:10.27787/d.cnki.ghrbs.2023.000649.

[5] 袁晓蓉. 基于 ROS 的机器人实验平台的设计与实现[D]. 湖南大学,2023.DOI:10.27135/d.cnki.ghudu.2023.000535.

[6]王校峰,王建文,曹鹏勇,等.机器视觉主导的机械臂动态抓取策略研究[J].机床与液压,2022,50(17):38-42.

[7] 高凯. 基于 ROS 的机器人运动规划和视觉定位方法研究[D]. 沈阳航空航天大学,2022.DOI:10.27324/d.cnki.gshkc.2022.000530.

[8] 陈华. 基于 ROS 的室内移动机器人导航技术研究[D]. 南昌大学,2022.DOI:10.27232/d.cnki.gnchu.2022.002678.

附　录

aivirtualmouse_node.py

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

import rospy
import cv2
import mediapipe as mp
import time
import numpy as np
from std_msgs.msg import String
from geometry_msgs.msg import Twist, PoseStamped
from visualization_msgs.msg import Marker
from nav_msgs.msg import Path

# Camera settings
wCam, hCam = 640, 480    # Resolution
frameR = 100    # Frame reduction for smoother tracking
smoothening = 5    # Not used but can be implemented for smoothing

# Initialize Mediapipe
cap = cv2.VideoCapture(0)    # Use 0 for internal camera, 1 for external
cap.set(3, wCam)
cap.set(4, hCam)

mpHands = mp.solutions.hands
hands = mpHands.Hands(min_detection_confidence=0.5, min_tracking_confidence=0.5)
mpDraw = mp.solutions.drawing_utils

# Initialize ROS node and publishers
rospy.init_node('aivirtualmouse_node', anonymous=True)
command_pub = rospy.Publisher('/aivirtualmouse_command', String, queue_size=10)
position_pub = rospy.Publisher('/aivirtualmouse_position', Twist, queue_size=10)
marker_pub = rospy.Publisher('/aivirtualmouse_marker', Marker, queue_size=10)
path_pub = rospy.Publisher('/path', Path, queue_size=10)    # 新增路径发布器
```

```
rate = rospy.Rate(30)  # Loop at 30 Hz
marker_id = 0  # Global variable for unique marker IDs
path = Path()  # 初始化路径消息
path.header.frame_id = "world"  # 确保和 RViz 的全局框架一致

def publish_command(command):
    msg = String()
    msg.data = command
    command_pub.publish(msg)
    rospy.loginfo(f"Published command: {command}")

def publish_position(x, y):
    msg = Twist()
    msg.linear.x = x / 100.0  # Convert pixel to position
    msg.linear.y = y / 100.0  # Convert pixel to position
    position_pub.publish(msg)
    rospy.loginfo(f"Published position: x={msg.linear.x}, y={msg.linear.y}")

def publish_marker(x, y, z=0.0):
    marker = Marker()
    marker.header.frame_id = "world"  # 确保和 RViz 的全局框架一致
    marker.header.stamp = rospy.Time.now()
    marker.ns = "aivirtualmouse"
    marker.id = 0  # Marker 的唯一 ID
    marker.type = Marker.SPHERE  # 设置为球形
    marker.action = Marker.ADD  # 确保是添加操作

    # 设置位置
    marker.pose.position.x = x / 100.0
    marker.pose.position.y = y / 100.0
    marker.pose.position.z = z
    marker.pose.orientation.x = 0.0
    marker.pose.orientation.y = 0.0
    marker.pose.orientation.z = 0.0
    marker.pose.orientation.w = 1.0
```

```
    # 设置 Marker 的尺寸
    marker.scale.x = 0.1  # 确保尺寸大于零
    marker.scale.y = 0.1
    marker.scale.z = 0.1

    # 设置 Marker 的颜色
    marker.color.a = 1.0  # 透明度 (1.0 为不透明)
    marker.color.r = 1.0  # 红色
    marker.color.g = 0.0  # 绿色
    marker.color.b = 0.0  # 蓝色

    # 发布 Marker
    marker_pub.publish(marker)
    print(f"Published Marker: x={x / 100.0}, y={y / 100.0}, z={z}")

def publish_path(x, y, z=0.0):
    global path
    path.header.stamp = rospy.Time.now()
    pose_stamped = PoseStamped()
    pose_stamped.header.frame_id = "world"
    pose_stamped.header.stamp = rospy.Time.now()
    pose_stamped.pose.position.x = x / 100.0
    pose_stamped.pose.position.y = y / 100.0
    pose_stamped.pose.position.z = z
    pose_stamped.pose.orientation.w = 1.0
    path.poses.append(pose_stamped)
    path_pub.publish(path)
    print(f"Published Path: x={x / 100.0}, y={y / 100.0}, z={z}")

def draw_hands(img, handLms):
    """ Draw hands and connections on the frame """
    for id, lm in enumerate(handLms.landmark):
        h, w, c = img.shape
        cx, cy = int(lm.x * w), int(lm.y * h)
```

```
            cv2.circle(img, (cx, cy), 5, (255, 255, 255), cv2.FILLED)
        mpDraw.draw_landmarks(img, handLms, mpHands.HAND_CONNECTIONS,
                                    mpDraw.DrawingSpec(color=(0, 255, 0), thickness=2, circle_radius=2),
                                    mpDraw.DrawingSpec(color=(0, 0, 255), thickness=2, circle_radius=2))

def fingers_up(handLms):
    """ Check which fingers are up """
    fingers = [False] * 5
    ids = [4, 8, 12, 16, 20]
    for i, id in enumerate(ids):
        if id == 4:    # Thumb
            if handLms.landmark[id].x < handLms.landmark[id - 1].x:
                fingers[i] = True
        else:    # Other fingers
            if handLms.landmark[id].y < handLms.landmark[id - 2].y:
                fingers[i] = True
    return fingers

def main():
    global marker_id
    pTime = 0    # For FPS calculation
    while not rospy.is_shutdown():
        ret, frame = cap.read()
        if not ret:
            rospy.logerr("Failed to capture frame from camera")
            continue

        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        rgb_frame.flags.writeable = False

        results = hands.process(rgb_frame)

        rgb_frame.flags.writeable = True
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

```
if results.multi_hand_landmarks:
    for hand_id, handLms in enumerate(results.multi_hand_landmarks):
        fingers = fingers_up(handLms)
        rospy.loginfo(f"Fingers up: {fingers}")

        if fingers == [True, False, False, False, False]:    # Thumb up
            publish_command("hover")
        elif fingers == [False, True, False, False, False]:    # Index finger
            publish_command("forward")
            x = handLms.landmark[8].x * wCam
            y = handLms.landmark[8].y * hCam
            publish_position(x, y)
            publish_marker(x, y)
            publish_path(x, y)    # 添加路径发布
        elif fingers == [False, False, True, False, False]:    # Middle finger
            publish_command("left")
        elif fingers == [False, False, False, True, False]:    # Ring finger
            publish_command("right")
        elif fingers == [False, False, False, False, True]:    # Pinky finger
            publish_command("backward")
        else:
            publish_command("hover")

        draw_hands(frame, handLms)
else:
    rospy.loginfo("No hand detected")
    publish_command("hover")

cTime = time.time()
fps = 1 / (cTime - pTime)
pTime = cTime
cv2.putText(frame, f'FPS: {int(fps)}', (20, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
cv2.imshow('Hand Tracking', frame)

if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
                break

            rate.sleep()

        cap.release()
        cv2.destroyAllWindows()

if __name__ == "__main__":
        try:
            main()
        except rospy.ROSInterruptException:
            pass
```

HandTrackingModule.py

```
import cv2
import mediapipe as mp
import math

class handDetector():
        def __init__(self, mode=False, maxHands=2, detectionCon=0.8, trackCon=0.8):
            self.mode = mode
            self.maxHands = maxHands
            self.detectionCon = detectionCon
            self.trackCon = trackCon

            self.mpHands = mp.solutions.hands
            self.hands = self.mpHands.Hands(
                static_image_mode=self.mode,
                max_num_hands=self.maxHands,
                min_detection_confidence=self.detectionCon,
                min_tracking_confidence=self.trackCon
            )
            self.mpDraw = mp.solutions.drawing_utils
            self.tipIds = [4, 8, 12, 16, 20]
```

```
    def findHands(self, img, draw=True):
        imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        self.results = self.hands.process(imgRGB)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms,
self.mpHands.HAND_CONNECTIONS)
        return img

    def findPosition(self, img, draw=True):
        self.lmList = []
        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                for id, lm in enumerate(handLms.landmark):
                    h, w, c = img.shape
                    cx, cy = int(lm.x * w), int(lm.y * h)
                    self.lmList.append([id, cx, cy])
                    if draw:
                        cv2.circle(img, (cx, cy), 12, (255, 0, 255), cv2.FILLED)
        return self.lmList

    def fingersUp(self):
        fingers = []
        if len(self.lmList) == 0:
            return fingers

        # 大拇指
        if self.lmList[self.tipIds[0]][1] > self.lmList[self.tipIds[0] - 1][1]:
            fingers.append(1)
        else:
            fingers.append(0)
```

```
            # 其余手指
            for id in range(1, 5):
                if self.lmList[self.tipIds[id]][2] < self.lmList[self.tipIds[id] - 2][2]:
                    fingers.append(1)
                else:
                    fingers.append(0)

            return fingers

        def findDistance(self, p1, p2, img, draw=True, r=15, t=3):
            if len(self.lmList) == 0:
                return None, img, None

            x1, y1 = self.lmList[p1][1:]
            x2, y2 = self.lmList[p2][1:]
            cx, cy = (x1 + x2) // 2, (y1 + y2) // 2

            length = math.hypot(x2 - x1, y2 - y1)

            if draw:
                cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), t)
                cv2.circle(img, (x1, y1), r, (255, 0, 255), cv2.FILLED)
                cv2.circle(img, (x2, y2), r, (255, 0, 255), cv2.FILLED)
                cv2.circle(img, (cx, cy), r, (0, 0, 255), cv2.FILLED)

            return length, img, [x1, y1, x2, y2, cx, cy]

        def isFingerInAir(self, fingers):
            if fingers[1] and fingers[2] and fingers[3]:
                return True
            return False
```

turtlesim_controller.py

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

```
import rospy
from geometry_msgs.msg import Twist, Pose
from std_msgs.msg import String
from nav_msgs.msg import Path

class TurtlesimController:
    def __init__(self):
        rospy.init_node('turtlesim_controller', anonymous=True)
        self.cmd_vel_pub = rospy.Publisher('/turtle1/cmd_vel', Twist, queue_size=10)
        self.pose_pub = rospy.Publisher('/turtle1/pose', Pose, queue_size=10)  # 添加位姿发布器
        rospy.Subscriber('/aivirtualmouse_command', String, self.command_callback)
        rospy.Subscriber('/aivirtualmouse_position', Twist, self.position_callback)
        rospy.Subscriber('/path', Path, self.path_callback)  # 订阅路径话题
        self.rate = rospy.Rate(30)  # 30 Hz
        self.current_pose = Pose()  # 初始化当前位姿
        self.path = None  # 初始化路径

    def command_callback(self, data):
        command = data.data
        twist = Twist()
        if command == "hover":
            twist.linear.x = 0.0
            twist.angular.z = 0.0
        elif command == "forward":
            twist.linear.x = 2.0  # 增加线速度
            twist.angular.z = 0.0
        elif command == "left":
            twist.linear.x = 0.0
            twist.angular.z = 4.0  # 增加角速度
        elif command == "right":
            twist.linear.x = 0.0
            twist.angular.z = -4.0  # 减少角速度
        elif command == "backward":
            twist.linear.x = -2.0  # 减少线速度
```

```
                twist.angular.z = 0.0
            else:
                twist.linear.x = 0.0
                twist.angular.z = 0.0
            self.cmd_vel_pub.publish(twist)
            print(f"Executed {command} command with linear.x={twist.linear.x} and
angular.z={twist.angular.z}")

        def position_callback(self, data):
            x = data.linear.x * 100  # 将位置值转换为像素值
            y = data.linear.y * 100  # 将位置值转换为像素值
            print(f"Received position: x={x}, y={y}")
            self.current_pose.position.x = x
            self.current_pose.position.y = y
            self.pose_pub.publish(self.current_pose)  # 发布当前位姿

        def path_callback(self, data):
            self.path = data
            print(f"Received path with {len(data.poses)} poses")
            for pose_stamped in data.poses:
                print(f"Pose at time {pose_stamped.header.stamp}:")
                print(f"    Position: x={pose_stamped.pose.position.x}, y={pose_stamped.pose.position.y},
z={pose_stamped.pose.position.z}")
                print(f"                Orientation:        x={pose_stamped.pose.orientation.x},
y={pose_stamped.pose.orientation.y},                    z={pose_stamped.pose.orientation.z},
w={pose_stamped.pose.orientation.w}")

        def run(self):
            while not rospy.is_shutdown():
                if self.path:
                    # 处理路径逻辑，例如跟随路径
                    pass
                self.rate.sleep()

    if __name__ == "__main__":
```

```
    controller = TurtlesimController()
    controller.run()
```

package.xml

```
<?xml version="1.0"?>
<package format="2">
  <name>bigwork</name>
  <version>0.0.0</version>
  <description>The bigwork package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="fan@todo.todo">fan</maintainer>

  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>MIT</license>

  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/bigwork</url> -->

  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->

  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
```

```
<!--  Note that this is equivalent to the following: -->
<!--  <build_depend>roscpp</build_depend> -->
<!--  <exec_depend>roscpp</exec_depend> -->
<!-- Use build_depend for packages you need at compile time: -->
<!--  <build_depend>message_generation</build_depend> -->
<!-- Use build_export_depend for packages you need in order to build against this package: -->
<!--  <build_export_depend>message_generation</build_export_depend> -->
<!-- Use buildtool_depend for build tool packages: -->
<!--  <buildtool_depend>catkin</buildtool_depend> -->
<!-- Use exec_depend for packages you need at runtime: -->
<!--  <exec_depend>message_runtime</exec_depend> -->
<!-- Use test_depend for packages you need only for testing: -->
<!--  <test_depend>gtest</test_depend> -->
<!-- Use doc_depend for packages you need only for building documentation: -->
<!--  <doc_depend>doxygen</doc_depend> -->

<buildtool_depend>catkin</buildtool_depend>
<build_depend>cv_bridge</build_depend>
<build_depend>geometry_msgs</build_depend>
<build_depend>rospy</build_depend>
<build_depend>sensor_msgs</build_depend>
<build_depend>std_msgs</build_depend>
<build_depend>std_srvs</build_depend>
<build_depend>turtlesim</build_depend>
<build_depend>message_generation</build_depend>
<build_export_depend>cv_bridge</build_export_depend>
<build_export_depend>geometry_msgs</build_export_depend>
<build_export_depend>rospy</build_export_depend>
<build_export_depend>sensor_msgs</build_export_depend>
<build_export_depend>std_msgs</build_export_depend>
<build_export_depend>std_srvs</build_export_depend>
<build_export_depend>turtlesim</build_export_depend>
<build_export_depend>message_generation</build_export_depend>
<exec_depend>cv_bridge</exec_depend>
<exec_depend>geometry_msgs</exec_depend>
```

```
    <exec_depend>rospy</exec_depend>
    <exec_depend>sensor_msgs</exec_depend>
    <exec_depend>std_msgs</exec_depend>
    <exec_depend>std_srvs</exec_depend>
    <exec_depend>turtlesim</exec_depend>
    <exec_depend>message_runtime</exec_depend>
    <exec_depend>python3-mediapipe</exec_depend>
    <exec_depend>python3-opencv</exec_depend>
    <exec_depend>python3-numpy</exec_depend>

    <!-- System dependencies -->
    <build_depend>Boost</build_depend>
    <exec_depend>Boost</exec_depend>

    <!-- The export tag contains other, unspecified, tags -->
    <export>
      <!-- Other tools can request additional information be placed here -->
    </export>
  </package>
```

aivirtual.launch

```
  <launch>
    <!-- 启动 turtlesim 节点 -->
    <node name="turtlesim_node" pkg="turtlesim" type="turtlesim_node" output="screen"/>

    <!-- 启动 aivirtualmouse_node.py 节点 -->
    <node name="aivirtualmouse_node" pkg="bigwork" type="aivirtualmouse_node.py" output="screen"/>

    <!-- 启动 turtlesim_controller.py 节点 -->
    <node name="turtlesim_controller" pkg="bigwork" type="turtlesim_controller.py" output="screen"/>

    <!-- 启动 RViz 并加载配置文件 -->
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(find bigwork)/rviz/aivirtual.rviz" output="screen"/>
```

```
    </launch>
```

CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0.2)
    project(bigwork)

    ## Compile as C++11, supported in ROS Kinetic and newer
    # add_compile_options(-std=c++11)

    ## Find catkin macros and libraries
    ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
    ## is used, also find other catkin packages
    find_package(catkin REQUIRED COMPONENTS
      cv_bridge
      geometry_msgs
      rospy
      sensor_msgs
      std_msgs
      std_srvs
      turtlesim
      message_generation
    )

    ## System dependencies are found with CMake's conventions
    # find_package(Boost REQUIRED COMPONENTS system)

    ## Generate messages in the 'msg' folder
    add_message_files(
      FILES
      hand_msg.msg
    )

    ## Generate services in the 'srv' folder
    # add_service_files(
    #   FILES
```

```
#   Service1.srv
#   Service2.srv
# )

## Generate actions in the 'action' folder
# add_action_files(
#   FILES
#   Action1.action
#   Action2.action
# )

## Generate added messages and services with any dependencies listed here
generate_messages(
  DEPENDENCIES
  geometry_msgs
  sensor_msgs
  std_msgs
)

######################################
## catkin specific configuration ##
######################################
## The catkin_package macro generates cmake config files for your package
## Declare things to be passed to dependent projects
## INCLUDE_DIRS: uncomment this if your package contains header files
## LIBRARIES: libraries you create in this project that dependent projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects also need
catkin_package(
  CATKIN_DEPENDS cv_bridge geometry_msgs rospy sensor_msgs std_msgs std_srvs turtlesim message_runtime
)

###########
## Build ##
```

```
############

## Specify additional locations of header files
## Your package locations should be listed before other locations
include_directories(
  ${catkin_INCLUDE_DIRS}
)

## Declare a C++ library
# add_library(${PROJECT_NAME}
#   src/${PROJECT_NAME}/bigwork.cpp
# )

## Add cmake target dependencies of the library
## as an example, code may need to be generated before libraries
## either from message generation or dynamic reconfigure
#   add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})

## Declare a C++ executable
## With catkin_make all packages are built within a single CMake context
## The recommended prefix ensures that target names across packages don't collide
# add_executable(${PROJECT_NAME}_node src/bigwork_node.cpp)

## Rename C++ executable without prefix
## The above recommended prefix causes long target names, the following renames the
## target back to the shorter version for ease of user use
## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg someones_pkg_node"
# set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node PREFIX "")

## Add cmake target dependencies of the executable
## same as for the library above
#   add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})
```

```
## Specify libraries to link a library or executable target against
# target_link_libraries(${PROJECT_NAME}_node
#   ${catkin_LIBRARIES}
# )

#############
## Install ##
#############

# all install targets should use catkin DESTINATION variables
# See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html

## Mark executable scripts (Python etc.) for installation
## in contrast to setup.py, you can choose the destination
catkin_install_python(PROGRAMS
  scripts/aivirtualmouse_node.py
  scripts/HandTrackingModule.py
  scripts/turtlesim_controller.py
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

## Mark _this_ros_setup.sh for installation
install(FILES
  scripts/_this_ros_setup.sh
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
)

## Mark executables for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_executables.html
# install(TARGETS ${PROJECT_NAME}_node
#   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
# )

## Mark libraries for installation
## See http://docs.ros.org/melodic/api/catkin/html/howto/format1/building_libraries.html
```

```
# install(TARGETS ${PROJECT_NAME}
#     ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#     LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
#     RUNTIME DESTINATION ${CATKIN_GLOBAL_BIN_DESTINATION}
# )

## Mark cpp header files for installation
# install(DIRECTORY include/${PROJECT_NAME}/
#     DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
#     FILES_MATCHING PATTERN "*.h"
#     PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#     # myfile1
#     # myfile2
#     DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

##############
## Testing ##
##############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_bigwork.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#     target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

《智能机器人操作系统》大作业成绩评定表

评　分　依　据	分	评分
1、认知与理解 ROS 的基本知识	20 分	
2、掌握 ROS 中的通信机制	30 分	
3、了解可视化工具的使用	20 分	
4、程序设计、调试及运行的步骤正确	20 分	
5、报告格式规范、流程图清晰、内容充实、语句通顺	10 分	
总分	100	

指导老师签字：_____

年　　月　　日