

轮式机器人：运动学建模与仿真

周炜 3210103790

Part 1 Python 实现 TCP/UDP 通信

机器人在实际运动中，需要多个功能模块互相协作，而这些模块之间需要实现实时的相互通信。在这次实验中，我按照要求使用了UDP通信和python socket编程，通过本次实验我也发现，如果机器人有 n 个组件，则大概要写近似于 $2n(n-1)$ 个组件，这就为之后引入ROS埋下了伏笔

实验步骤

1. UDP通信的实现

使用 Python 编程，并且 `import socket import sys import select`
`socket.socket(family,type)`创建一个新对象
`sock.sendto(data,addr)`通过 socket 向指定地址发送数据
`socket.bind = ('localhost', 端口号)` 将 socket 绑定到本地的某个端口上，
`socket.recvfrom(bufsize[, flags])`从 socket 中接受数据
在本实验中，需要编写两个文件，`receive.py` 和 `send.py`

2. 信息发送端 `send.py`

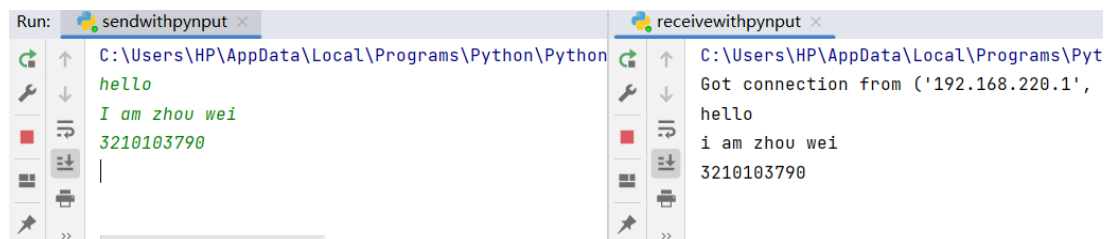
- `inputs = [sys.stdin, connection]` 监听键盘输入并发送到客户端
- 可以利用 `data = sys.stdin.readline()`直接读取一个字符串发送
- 可以使用 `connection.sendall(data.encode())`来进行编码发送

3. 信息接受端 `receive.py`

- 需要与发送端使用相同的端口 `sock.connect('localhost', 端口号)`
- 将读取到的文本内容进行编辑，并存储为文本文件，可以用 `with open('received_file.txt', 'w') as f:` 打开检测文件是否打开，然后用 `f.write(data)`实时接受收到的信息，并且写入文件

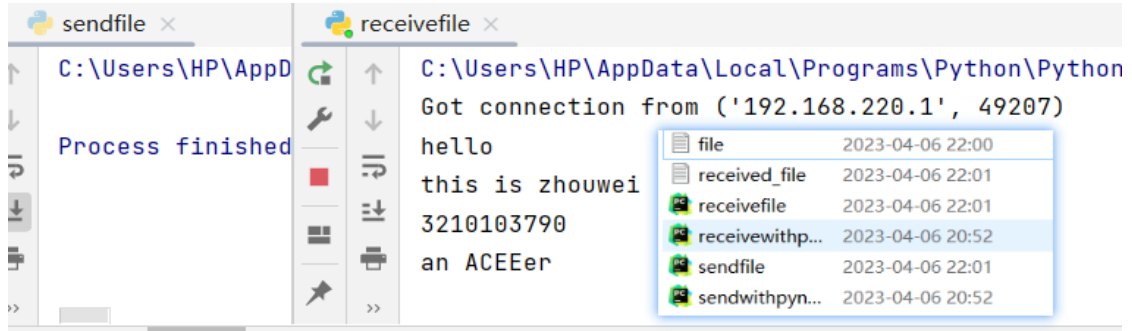
遇到的困难

- 在打开`send.py`之前，应该先开启`receive.py`
- 重新运行程序时会显示端口被占用，需要每次都更换端口，并且更换时需要保证发送端和接受端的端口相同
- `str.encode()`和`str.decode()`以指定编码格式需要一致，否则会出现乱码



发送端

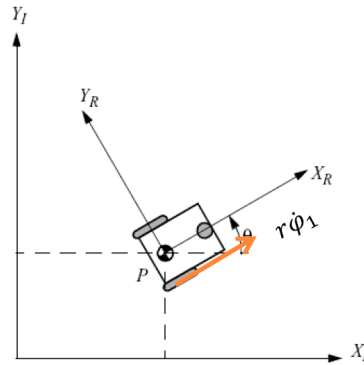
接受端



扩展实验：文件的收发

Part 2 基于作用的运动学建模

用空间中的一个点来表示机器人，定义全局坐标系和局部坐标系下机器人的速度



$$\text{全局: } \dot{X}_I = (\dot{x}_I, \dot{y}_I, \dot{\theta}_I)^T \quad \text{局部: } \dot{X}_R = (\dot{x}_R, \dot{y}_R, \dot{\theta}_R)^T$$

并且全局坐标系和局部坐标系之间满足：

$$\dot{X}_R = R(\theta) \dot{X}_I$$

其中：

$$R(\theta) = \begin{pmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

本次建模的机器人为两轮差分驱动机器人，由两个独立驱动转速的固定标准轮和一个无驱动的随动脚轮组成，设轮子半径为 r ，两轮子到两轮中心 P 的距离为 l ，两轮旋转速度为 $\dot{\phi}_1, \dot{\phi}_2$

$$\dot{x}_R = \frac{r}{2}(\dot{\phi}_1 + \dot{\phi}_2)$$

$$\dot{y}_R = 0$$

若以左轮为中心逆时针转， $\omega_1 = \frac{r\dot{\phi}_1}{2l}$ ，若以右轮为中心旋转， $\omega_2 = \frac{r\dot{\phi}_2}{2l}$

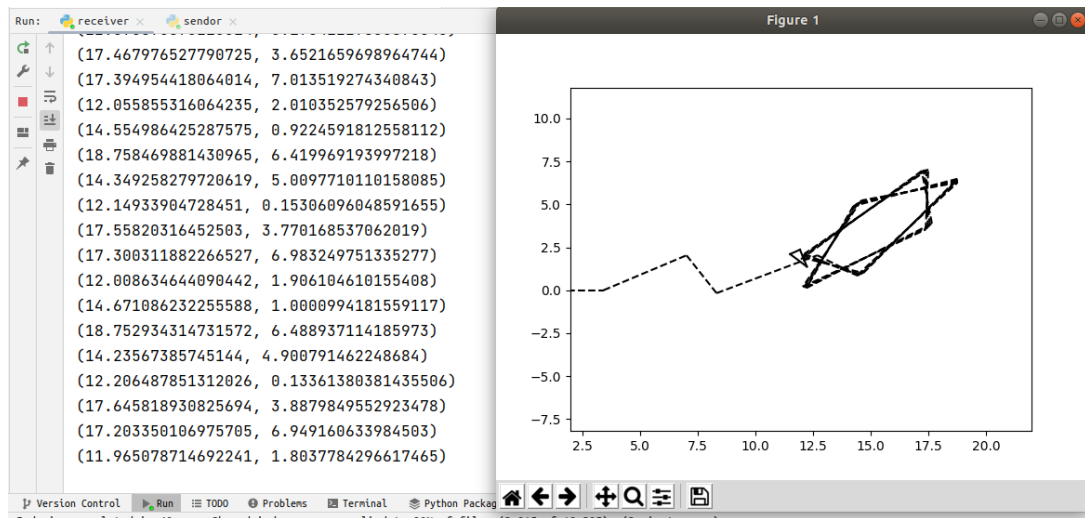
$$\dot{X}_I = R^{-1}(\theta)\dot{X}_R = R^{-1}(\theta) \begin{pmatrix} \frac{r}{2}(\dot{\phi}_1 + \dot{\phi}_2) \\ 0 \\ \frac{r}{2l}(\dot{\phi}_1 - \dot{\phi}_2) \end{pmatrix}$$

实验步骤

1. 使用 `pynput.keyboard` 里面的 `key` 和 `listener` 监听键盘，回调函数中用 `udp` 协议下发机器人位移速度和角速度
2. 同时运行两个进程，`sender.py`：使用 `pynput` 直接监听键盘，回调函数中直接进行速度的 `udp` 下发；`receiver.py`：开一个线程进行 `udp` 接收，将接收的 `vel` 存储为全局变量，主线程定时的调用 `matplotlib` 完成绘制运动轨迹

遇到的困难

1. 要设置好 `matplotlib` 的绘图模式，应该在一张图上作图而不是动一下就做一张新图
2. 设置合理的速度以精准控制小车运动，不会出现运动过快而找不到的情况
3. 如果 `matplotlib` 设置退出的快捷键，则 `pause` 的时间需要久一点，否则容易卡顿



左侧为地图中坐标，右侧为绘图的窗口

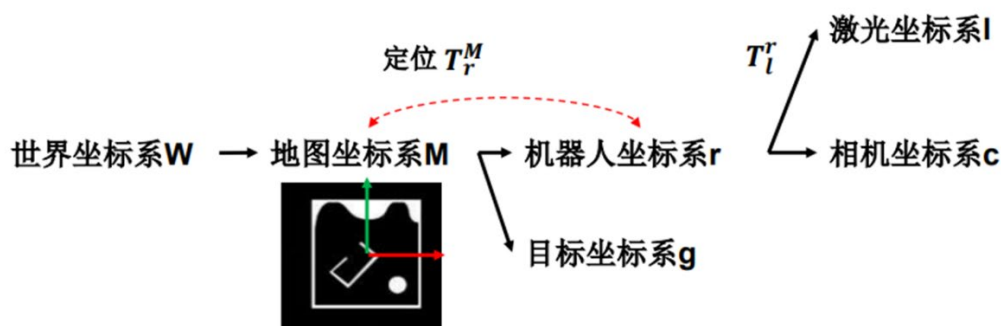
Part 3 ROS的通信机制、Gazebo 仿真和 Rviz 可视化

1. ROS里的通信分为消息和服务两种方式。
消息采用发布-订阅模式，互相不阻塞（小车自主前往指定位置时采用的通信方式）
服务类似于函数调用，阻塞到服务端回复
2. 对于复杂的机器人系统，要用大量坐标系及其之间的关系进行刻画，我们可以采用坐标系树和链式法则，使得坐标系可重用

$$\text{旋转矩阵: } R_r^M = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \quad \text{位姿矩阵: } T_r^M = \begin{bmatrix} R_r^M & t_r^M \\ 0 & 1 \end{bmatrix}$$

$$\text{链式法则: } T_S^M = T_r^M T_S^r$$

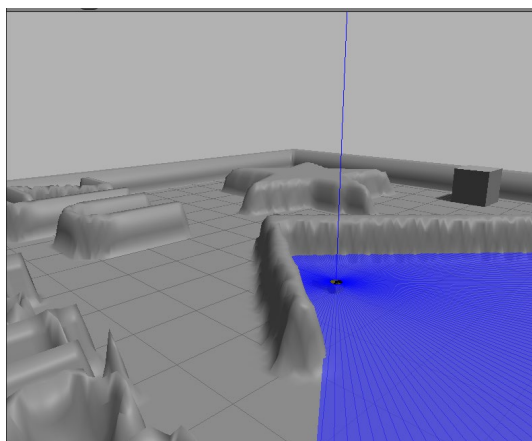
3. **Gazebo**是一个机器人仿真工具，里面的环境由用户产生，在现实环境中客观存在
Gazebo中的坐标信息添加到**TF**下
 地图信息在对应的坐标系下显示
 机器人信息在对应的坐标系下显示



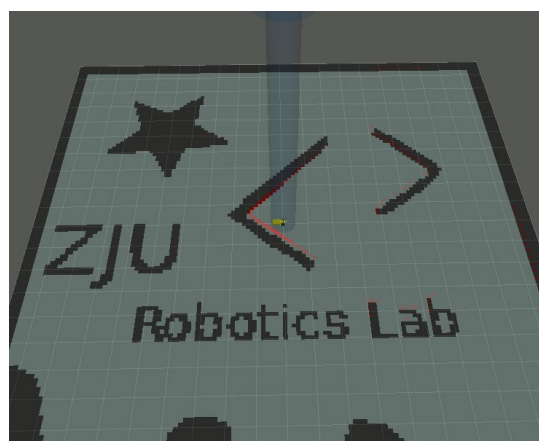
4. **Rviz**是一个三维可视化平台，在ROS提供了对应的工具，根据TF显示对应的元素，包括地图、激光等，可以利用Rviz实现机器人所处的物理空间可视化

实验步骤

1. ROS命令的学习和使用。使用catkin_make创建一个ROSPackage，用roslaunch指令启动gazebo仿真环境
2. 编写 kinematics.py
 订阅消息"/course_agv/velocity" (geometry_msgs.msg.Twist) "
 发送消息"/course_agv/left_wheel_velocity_controller/command"
 发送消息"/course_agv/right_wheel_velocity_controller/command"
3. 编写 robot_tf.py
 订阅消息"/gazebo/link_states" (gazebo_msgs.msg.LinkStates) "
 在消息中找到"robot_base"索引的link并获取姿态(位置及四元数角度)
 使用TF相关api进行发送

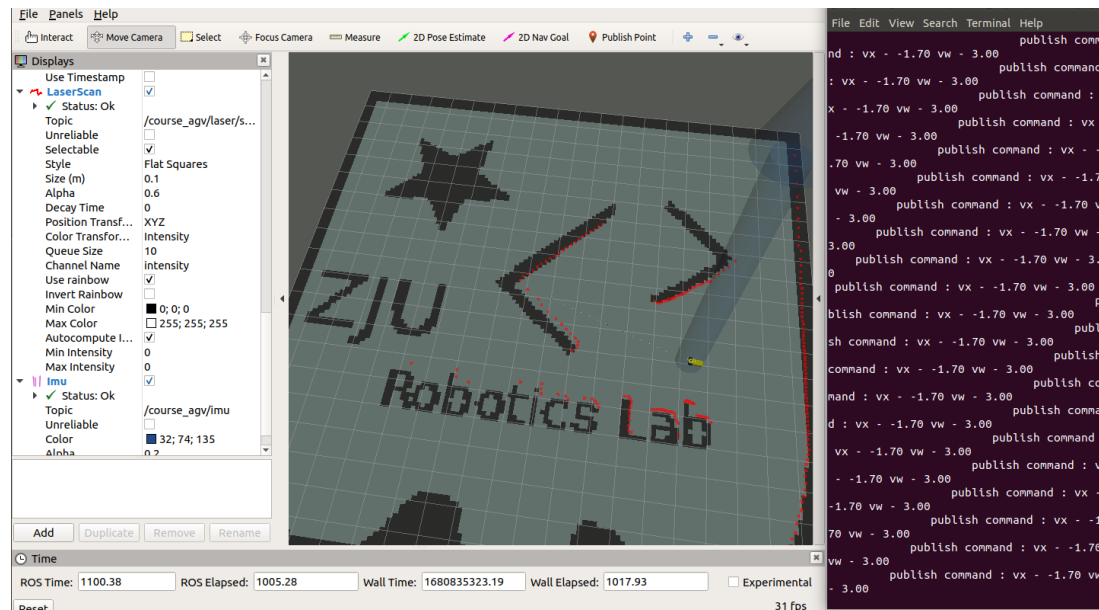


Gazebo 下的小车



Rviz 下的校车

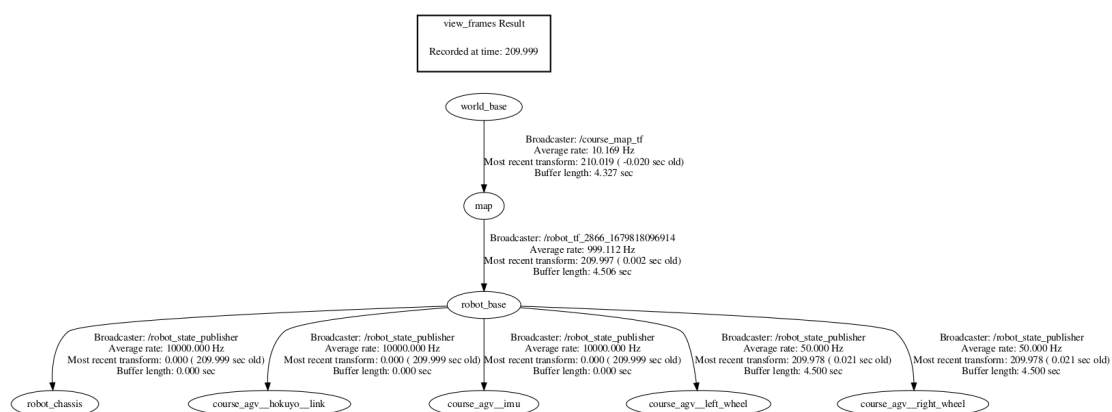
4. 使用`tf`的接口`tf_pub.sendTransform()`进行发送四元数
5. 使用`Rviz`查看可视化效果, 使用`roslaunch tf view_frames`获得坐标系树



使用键盘控制小车运动

遇到的困难

1. 安装ROS的过程中需要换源, 否则会下载很久之后仍然会失败, 我推荐之后的学习者使用 [鱼香ROS](#) 一键安装
2. 四元数不能是(0,0,0,0), 否则没有意义
3. 需要注意程序运行的先后顺序, 比如, 在运行`Rviz`后会显示坐标系错误, 需要运行`tf` (在`Rviz`之前运行`tf`会失败)
4. 每次运行`ros`的程序 (每个命令行窗口) 之前都要先在命令行中输入`source devel/setup.bash`, 并且记得要关掉`conda`的虚拟环境
5. 我们需要完成机器人坐标系下的位姿状态到地图坐标系下的坐标转换和地图坐标系到世界坐标系下的映射, 在函数中参数的顺序不能弄反
6. 出现`Done checking log file disk usage. Usage is <1GB.`的错误而无法启动`Rviz`, 是因为没有`source devel/setup.bash`, 而不是虚拟机内存大小的问题



上述程序所对应的坐标系树