Client使用说明

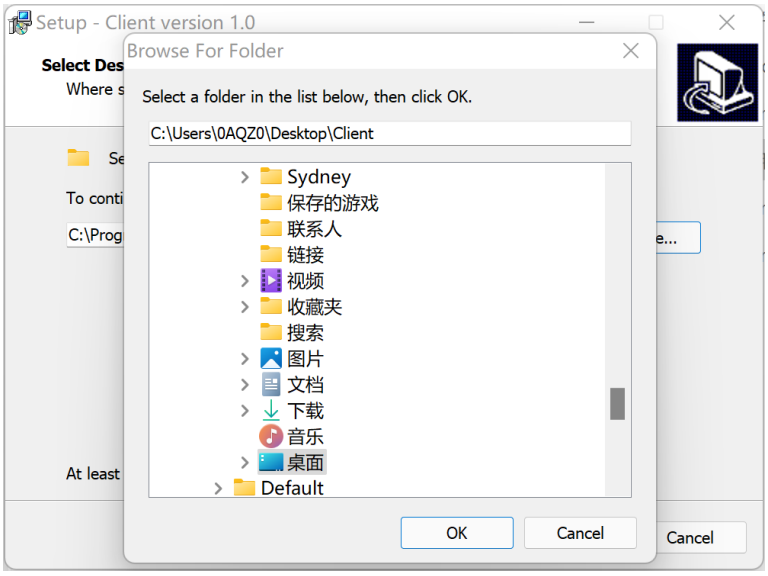
TOC

软件安装

Windows平台

在Windows平台打开setup.exe，选择安装目录，如`C:\Users\0AQZ0\Desktop\Client`，然后一直点下一步即可

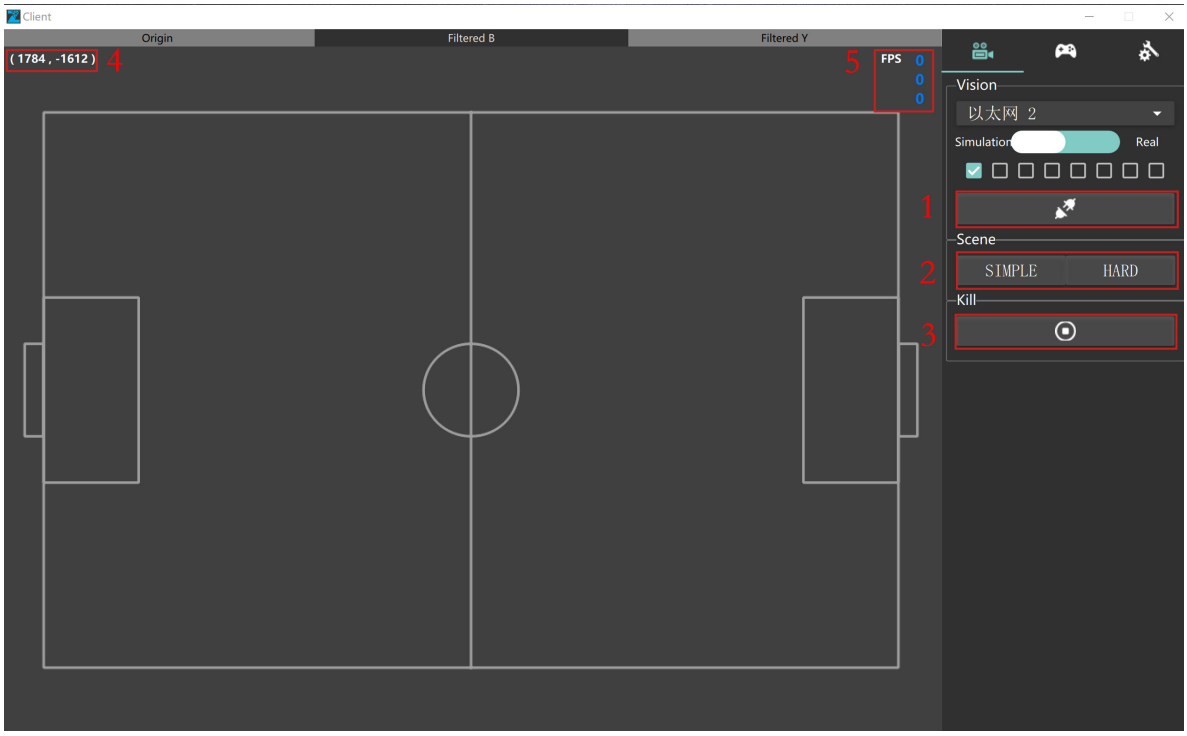
注意不要选择在需要管理员权限的目录，如C:\Program Files (x86)\Client，否则需要管理员权限打开



Ubuntu平台

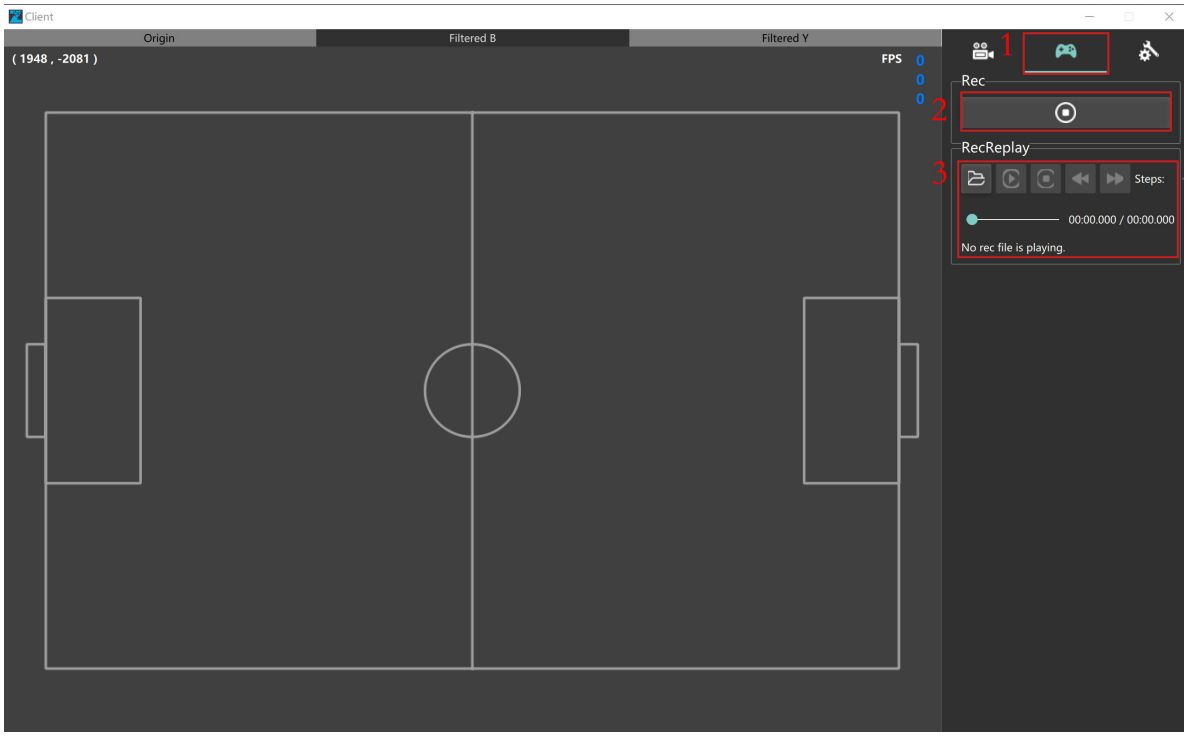
需要下载QT，并编译源码，不推荐使用

软件基本使用



- 红色方框1的按钮可以选择连接或者断开图像
- 红色方框2的按钮选择测试的场景，分为simple和hard
  - simple场景：场上16个机器人，除了需要控制的机器人其他为静态障碍物
  - hard场景：场上16个黄色机器人，其中5个为动态障碍物，其他为静态障碍物

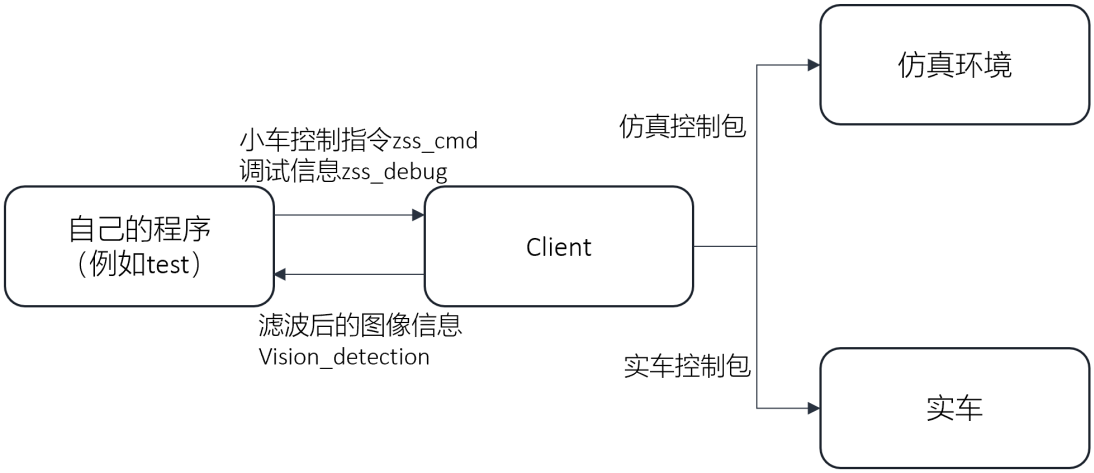
  需要运行dynamic_obstacles文件夹中的`main.py`，即`python main.py`
- 红色方框3的按钮可以快速关闭程序
- 红色方框4显示的是鼠标所指位置的坐标，单位mm
- 红色方框5分别显示图像帧数、蓝车控制指令帧数、黄车控制指令帧数
- 常用小技巧：连接图像后，使用`` ` ``、1-10、-、+、i、o、p、[可以控制蓝车数量从0-16变化，如果同时按住`ctrl`可以控制黄车数量；鼠标可以直接拖动机器人



- 点击红色方框1的按钮可以切换到log的工具栏

- 红色方框2的按钮可以开始或者停止录制log，默认开启
- 红色方框3内显示log播放信息，最左边文件夹按钮可以打开log文件进行播放（打开时需要断开图像连接），其他按钮分别是开始、停止、快退、快进

控制流程



基本流程

自己的程序接收来自Client的图像信息（场上机器人的位置、速度等），进行路径、速度规划之后发送小车控制指令给Client。

这个过程中，信息的传递是通过protobuf实现的。形象地说，UDP是交流的方式，Protobuf是语言。

Protobuf

protobuf(Protocal Buffers)是google开发的一种用于序列化结构化数据（JSON、XML)的一种方式。它灵活、高效，只需要关于数据结构的描述.proto，就可以使用compiler自动生成进行编码、解析的class，这个class提供了getter和setter这些method读写protobuf。参考链接

Demo程序讲解

主要提供了两个Demo程序，分别完成接收图像信息、发送控制指令：

- **ssl_demo_python**：Python例程，需要Python>=3、Protobuf包`pip install protobuf`
- **ssl_demo_c++**：C++例程，需要QT、VS Studio进行编译

以下重点讲解Python例程：

```
ssl_demo_python
 ├ proto                                定义数据结构的文件夹
 │ ├ vision_detection.proto             接收的图像的数据结构
 │ ├ zss_cmd.proto                      发送的控制指令的数据结构
 │ └ zss_debug.proto                    发送的调试信息的数据结构
 ├ action.py                            发送控制指令的实现代码
 ├ debug.py                             发送调试信息的实现代码
 ├ main.py                              主程序
 ├ vision.py                            接收图像数据的实现代码
 ├ vision_detection_pb2.py              自动生成的文件，无需阅读
 ├ zss_cmd_pb2.py                       自动生成的文件，无需阅读
 └ zss_debug_pb2.py                     自动生成的文件，无需阅读
```

接收图像信息

相关文件：

- vision.py
- vision_detection.proto
- vision_detection_pb2.py

在`vision.py`里实现了两个类，一个是`Vision`用于读取图像数据，一个是`Robot`用于存储每个机器人的数据。

Robot存储了机器人的ID、是否可见、位置、速度、朝向（分为图像滤波后和滤波前两种）

```
class Robot(object):
    def __init__(self, id, visible=False,
                 x=-999999, y=-999999, vel_x=0, vel_y=0, orientation=0,
                 raw_x=-999999, raw_y=-999999, raw_vel_x=0, raw_vel_y=0,
                 raw_orientation=0):
        self.id = id
        self.visible = visible
        # filtered vision
        self.x = x
        self.y = y
        self.vel_x = vel_x
        self.vel_y = vel_y
        self.orientation = orientation
        # raw vision
        self.raw_x = raw_x
        self.raw_y = raw_y
        self.raw_vel_x = raw_vel_x
        self.raw_vel_y = raw_vel_y
        self.raw_orientation = raw_orientation
```

Vision通过UDP读取图像数据包，并解析存储在`blue_robot`和`yellow_robot`两个数组中，用法如下：

```
from vision import Vision

# 实例化Vision这个类
vision_module = Vision()

# 读取我的机器人的位置，默认我的机器人为蓝车0号
my_robot = vision_module.my_robot
print('My robot:', my_robot.x, my_robot.y)
# 读取蓝车0号的位置，其他车号0-15类似
blue_robot_0 = vision_module.blue_robot[0]
print('Blue robot 0:', blue_robot_0.x, blue_robot_0.y)
# 读取黄车0号的位置，其他车号0-15类似
yellow_robot_0 = vision_module.yellow_robot[0]
print('Yellow robot 0:', yellow_robot_0.x, yellow_robot_0.y)
```

发送控制指令

相关文件：

- action.py

- zss_cmd.proto
- zss_cmd_pb2.py

在`action.py`中实现了`Action`这个类，用于控制我的机器人（默认是蓝车0号）的vx和vw，用法如下：

注意本次实现的是差分驱动，不论是否发送vy都会被直接置零

```
from action import Action

# 实例化Action这个类
action_module = Action()
while True:
    # 给我的机器人发送vx为100，vw为10的速度
    action_module.sendCommand(vx=100, vw=10)
    time.sleep(0.02)
```

绘制Debug信息

相关文件：

- debug.py
- zss_debug.proto
- zss_debug_pb2.py

在`Debugger`中实现了`Debugger`这个类，用于绘制调试信息，例如画圆、画线、画点

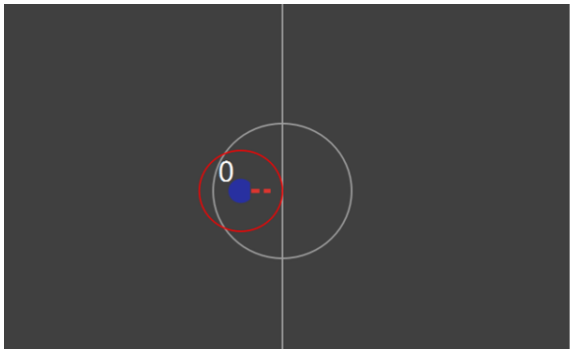
- draw_circle——画圆，参数：圆心坐标和半径
- draw_line——画线，参数：两点的坐标（x1,y1）和（x2,y2）
- draw_lines——画多条线，参数：两点的坐标列表，x1,y1,x2,y2都是列表
- draw_point——画点，参数：点的坐标（x,y）
- draw_points——画多个点，参数：点的坐标列表，x,y都是列表

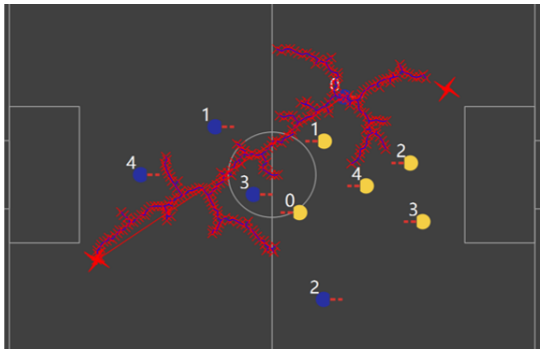
注意每次发送新的调试信息，会将上次发送的覆盖掉

```
from debug import Debugger
from zss_debug_pb2 import Debug_Msgs

# 实例化Debugger这个类
debugger = Debugger()
# 定义一个调试信息的包
package = Debug_Msgs()
# 画一个圆
debugger.draw_circle(package, x=0, y=500)
# 画一条线
debugger.draw_line(package, x1=0, y1=2500, x2=600, y2=2500)
# 画多条线
debugger.draw_lines(package, x1=[0,0], y1=[0,2000], x2=[2000,2000], y2=[0,2000])
# 画一个点
debugger.draw_point(package, x=500, y=500)
# 画多个点
debugger.draw_points(package, x=[1000, 2000], y=[3000, 3000])
# 发送调试信息
debugger.send()
```

示例：





主程序

相关文件：

- main.py

在`main.py`中实现了主函数，完成了图像数据的接收、控制指令发送、调试信息绘制。

- **图像数据接收**：通过实例化`Vision`这个类完成，会单独开一个线程读取图像数据，只需访问它的成员变量即可获得场上机器人的位置等信息。
- **路径规划&速度规划**：需要在注释处完成，可以实现A*或者RRT。
- **发送控制指令**：完成规划后得到规划的速度vx、vw，通过`Action`这个类完成发送。

```
from vision import Vision
from action import Action
from debug import Debugger
from zss_debug_pb2 import Debug_Msgs
import time

if __name__ == '__main__':
    vision = Vision()
    action = Action()
    debugger = Debugger()
    while True:
        # 1. path planning & velocity planning
        # Do something

        # 2. send command
        action.sendCommand(vx=100, vy=0, vw=0)

        # 3. draw debug msg
        package = Debug_Msgs()
        debugger.draw_circle(package, vision.my_robot.x, vision.my_robot.y)
        debugger.send(package)

        time.sleep(0.01)
```