**第二十五届中国机器人及人工智能大赛**

**智慧药房挑战赛项目**

**技 术 报 告**

|  |  |
| --- | --- |
| 学 校： | 重庆邮电大学 |
| 队伍名称： | 神里绫华后援团 |
| 参赛队员： | 吕炜楷 陈亦杨 刘凯旋 |
| 带队教师： | 石钧仁 付蔚 |

关于技术报告和研究论文使用授权的说明

本人完全了解第二十五届中国机器人及人工智能大赛有关保留、使用技术报告和研究论文的规定，即：参赛作品著作权归参赛者本人，比赛组委会和赞助公司可以在相关主页上收录并公开参赛作品的设计方案、技术报告以及参赛模型车的视频、图像资料，并将相关内容编纂收录在组委会出版论文集中。

参赛队员签名：7939c8ffb442c8532350eb6bca5bcbb2

带队教师签名：  a8edb017a69aa4b676f9aa748e711958

日 期： 2024.4.19

综述

本队以EPRobot小车为平台，通过深入研究其底盘、传感器技术、上层算法、图像处理、深度学习以及ROS等关键领域，成功编写出相应代码并进行了参数优化。这些努力使得我们实现了智慧药房中药品的分拣系统与快递小哥间的完全自主配送功能。

本文首先从问题定义出发，细致分析了比赛的核心任务。接着，针对这些任务的具体实现，我们详细探讨了所涉及的知识点，并对比了当前的主流解决方案。结合智慧药房的实际环境，我们进行了相应的技术改进，并详细阐述了所采用的技术方案及其实现过程。

在小车功能实现方面，我们取得了以下主要成果：

1.以ROS为核心，通过编写C++节点并利用其话题、服务等通信机制，我们成功实现了小车与move\_base导航框架的通信，从而使其在药房内能够自主完成多点的药品获取与配送任务，有效解决了比赛中的药品配送难题。

2.利用ROS中的rqt\_reconfig工具，我们对导航参数进行了细致的调整，有效解决了小车在行进过程中的碰撞问题及导航位置偏离问题。

3.基于pytorch深度学习平台，我们结合ROS编写了python节点，并利用卷积神经网络实现了药品需求的获取及手写数字的识别功能,以及药丸识别。

4.为了增强小车的实用性，我们还开发了一些扩展功能，如导航恢复机制以提高小车的容错性，以及语音播报功能使配送过程更加人性化。

最终，我们在实际环境中对系统进行了测试分析，并成功解决了遇到的各种临场问题，从而圆满完成了比赛的所有任务要求。

**关键词：**EPRobot ROS 图像处理 深度学习

目 录

[一、 问题分析 5](#_Toc14096)

[1.1 问题重述 5](#_Toc23229)

[1.2 问题分析 5](#_Toc22114)

[二、 技术方案 5](#_Toc31778)

[2.1 硬件设计 5](#_Toc6856)

[2.1.1 STM32驱动方法及实现 5](#_Toc16722)

[2.1.2 STM32驱动方法的具体应用 6](#_Toc28593)

[2.1.3 阿克曼底盘 7](#_Toc16973)

[2.1.4 PID底盘控制算法原理及实现 9](#_Toc9791)

[2.2 软件设计 10](#_Toc18502)

[2.2.1 自主导航 10](#_Toc5387)

[2.2.2 数字和字母图像识别 13](#_Toc219)

[2.2.3 药丸图像识别 14](#_Toc18174)

[三、 方案实现 15](#_Toc6158)

[3.1 建图环节 15](#_Toc14818)

[3.2 取药、配药环节 16](#_Toc22281)

[3.3 答题环节 18](#_Toc29344)

[3.4 图像识别环节 18](#_Toc5612)

[3.4.1 图片预处理 19](#_Toc7234)

[3.4.2 数字和字母图像识别 21](#_Toc320)

[3.4.3 药丸图像识别 22](#_Toc20532)

[3.5 扩展功能 23](#_Toc30656)

[3.5.1 语音模块 23](#_Toc22870)

[3.5.2 导航恢复模块 23](#_Toc11151)

[四、 测试分析 23](#_Toc4322)

[4.1 多点导航 23](#_Toc18977)

[4.2 数字和字母图像识别 26](#_Toc11054)

[4.3 药丸图片识别 27](#_Toc18521)

[4.4 语音功能测试 28](#_Toc15314)

[五、 作品总结 30](#_Toc7340)

# 问题分析

## 问题重述

在此次智慧药房竞赛中，参赛团队被要求设计一款药品配送小车，该小车将用于智慧药房内的药品分拣与快速配送任务。智慧药房的场地设置为一个长方形的空间，且在其中分布有障碍物。配送药品的区域设置有三个不同的窗口，分别是A、B、C，这些窗口负责配送不同种类的药品。而取药区则设置了四个窗口，供快递人员在此处取药。参赛队伍的主要任务是以尽可能准确和高效的方式完成药品配送任务。最终的比赛成绩将根据参赛队伍在15分钟内完成的订单数量及其准确性来综合评估。配送小车的任务包括以下三点：

1. 小车需要获取相应的信息（配送药品信息，手写数字板）；
2. 小车需要规避障碍物，还要保证准确停留在规定的配药和取药区域；
3. 小车需要尽量避免快递小哥过长的等待，以及药品数量的堆积问题。

## 问题分析

1、针对任务1的要求，小车需要通过相机获取当前轮次的配送信息和数字手写板的内容并发布给其他节点。

2、针对任务2，小车应当实现自主前往各个地点，同时要规避障碍物、保证停留的准确性。

3、针对任务3，为了避免药品数量的过多堆积与快递小哥的过长等待，需要设计每一轮药品配送的先后顺序，同时依赖识别手写数字板，合理调整各个药品的配送周期。

# 技术方案

## 硬件设计

### STM32驱动方法及实现

（1）GPIO驱动：使用GPIO口可以方便地驱动按钮、蜂鸣器等外设。可通过STM32提供的HAL库进行GPIO的配置与控制。

（2）定时器驱动：定时器通常用于实现PWM波形、计时、捕获等功能。可1使用TIM定时器驱动步进电机、舵机、直流电机等。

（3）ADC/DAC驱动：ADC可以对模拟量信号进行采样，DAC则可输出模拟量信号。这些功能可用于测量温度、湿度、光强等环境参数，并进行控制反馈。

（4）中断驱动：中断处理可提高STM32的响应速度和减少CPU占用率。可使用HAL库提供的NVCI模块来实现中断控制。

（5）通信驱动：STM32支持多种通信协议，如UART、I2C、SPI等。可使用STM32提供的硬件抽象层来方便进行通信控制。

### STM32驱动方法的具体应用

（1）STM32微控制器上驱动步进电机的一般步骤：

首先，需对微控制器的GPIO端口进行配置，设定其为输出模式，以实现对步进电机方向和速度的调控。

其次，初始化TIM定时器，利用它生成PWM波形，并调整定时器的频率和分辨率以满足步进电机的控制需求。

接着，编写控制算法，确保步进电机能够按照设定的方向、速度和步数进行旋转。

然后，启动已配置好的定时器，生成PWM波形，从而控制步进电机的旋转角度和速度。

最后，对代码进行调试和测试，确保步进电机能够按照预期进行旋转。

（2）使用STM32控制舵机时，具体操作步骤如下：

首先，选择一个合适的PWM输出引脚，并将其配置为PWM模式。

其次，根据舵机的控制要求，计算出所需的PWM占空比，该占空比通常由0度位置对应的PWM脉宽和目标角度与0度位置的偏移值共同决定。

然后，通过STM32的程序代码，设置所选PWM输出引脚的占空比，以实现对舵机转向角度的精确控制。

（3）STM32使用串口通信的基本步骤：

首先，进行串口初始化，这包括配置USART外设，并设置串口的波特率、校验位和停止位等关键参数。这通常可以借助STM32提供的HAL库或CMSIS库来完成。

接着，当需要发送数据时，将数据写入USART的数据寄存器，并等待发送完成。同样，发送和接收数据的功能可以通过STM32提供的HAL库或CMSIS库中的USART API来实现。

然后，在接收数据时，需要监听USART接收寄存器的状态，并在接收到数据时及时读取。为了提高系统的响应速度和效率，可以利用中断处理机制来实现USART数据的接收和发送。这通常涉及STM32提供的HAL库或CMSIS库中的NVIC模块的配置和使用。

最后，对于可能出现的各种通信错误，如帧错误、奇偶校验错误等，需要在程序中加入相应的错误处理逻辑，以确保通信的稳定性和可靠性。

### 阿克曼底盘

阿克曼结转向构底盘类似现实中真实的汽车底盘。底盘前两轮用于控制小车运动方向，后两轮用于控制其次速度。

阿克曼运动学模型：

阿克曼底盘的运动学模型如下图所示，它是一个全驱动模型，有三个输入（）和三个输出（X，Y，）

|  |
| --- |
| 图 |

后轮中心的线速度为：

因为很小可以近似认为与相等，就等于内外轮走过的距离差与两轮间距的比值：

所以可以求出角速度：

由图可知，和R之间存在一定关系：

其中R可以由和表示出来：

因此可以求得与和的关系：

由此可以得出阿克曼输入的运动模型如下：

|  |
| --- |
| 图 |

车辆后轮轴的和与和的关系如下：

车辆后轮轴的和与和的关系如下：

化简并得到角速度：

同时得到转弯半径和前轮偏向角：

整理合并得到车辆的运动学模型：

### PID底盘控制算法原理及实现

PID 控制器由以下几个单元构成，比例单元(P)、积分单元(I)和微分单元(D)。系统的输入e(t)和输出u(t)的关系为：

式子中，为比例系数；为积分时间常数；为微分时间常数：0和t分别为积分的上下限。对u(t)和e(t)进行拉普拉斯变换，算出PID调节器的传递函数：

根据PID控制器中的各个单元组成和公式可以推导出原理图。图中，r(t)是希望系统输出的目标值，y(t)是系统在经过调节后产生的输出；通过分析可以得到各个单元的作用。

比例单元的主要作用就是对系统误差快速做出响应并产生输出，从而减少误差。但是，比例单元不能消除全部误差，所以相应的系数不能太大或太小，否则可能不稳定或者出现迟滞。

积分控制单元的作用是用来消除静态误差，积分有着类似于“滞后”的作用，具有容错功能，如果作用过强会造成e(t)过大才进行调节，如果作用太弱就会造成e(t)值很小就进行调节，容易产生震荡，所以的配置也十分重要。

微分控制单元的作用是让系统能够快速响应，加快系统趋于稳定的速度，让系统具有更好的动态性能。适当地选择微分常数，可以使微分作用达到最优。

PID的实现包括：

（1）速度控制，当小车偏离期望速度时，P（比例）控制器将根据误差大小产生一个相对于的控制量；I（积分）控制器将根据误差变化速率产生另一个控制量；D（微分）控制器将根据误差变化率的变化率产生第三个控制量。将他们加权求和，作为小车控制信号的输出，控制小车电机的转速，从而达到期望速度值。

（2）由于舵机具有较大的滞后性，并且智能车在转弯时要求快速响应，因此采用的是PD调节器。利用摄像头采集到的图像计算路径中心与小车中心的偏差值，偏差值通过PD调节器产生的输出控制舵机进行打角，从而控制小车方向。另外，可以根据不同的偏差值给定不同的值和值，通过试验和测试确定对应的值，使智能车在转弯时快速响应并保持较快的速度。

综上所述不断调试选取最合适的，，值，可以让小车的底盘控制达到最佳状态，保持平稳且丝滑的运动状态。

## 软件设计

软件设计的内容主要包括自主导航和图像识别两大方面，本章将从这两大方面进行分析，描述涉及的解决方案，并进行相应分析。

### 自主导航

实现自主导航小车一般分为三个部分：建图、路径规划、控制。

1. 建图：这一过程主要是让机器人对其所在环境进行感知和解析。在此阶段，机器人会利用激光雷达、摄像头等传感器来收集环境信息，并通过特定的算法将这些信息转换成地图表示。建图有静态和动态之分，前者适用于环境变化不大的情况，而后者则更适合环境复杂或需要实时更新地图的场景。

2、路径规划：这是确定机器人从起点到终点最佳行走路线的过程。在此阶段，机器人会基于已建好的地图，同时考虑到环境中的障碍物和自身的动力学限制，通过算法选择一条既安全又高效的路径。常见的路径规划算法有A\*算法和Dijkstra算法等。

3、控制：此阶段是将路径规划得出的目标轨迹转化为机器人的具体控制指令，以实现其运动控制。机器人会利用里程计、惯性测量单元等传感器来获知自身的状态信息，并通过PID控制器、模型预测控制等算法计算出合适的控制指令，从而控制机器人的速度和转向，使其按照规划的路径行进。这部分功能在硬件设计中已得到实现。

这三个部分紧密相连，共同确保自主导航小车能够在未知环境中实现自主探索与路径跟踪。建图提供了环境信息，路径规划确定了最佳路径，而控制则将路径转化为具体的运动指令，使机器人能够避开障碍物并沿规划路径到达终点。

在EPRobot小车中，已集成了Google Cartographer这一激光SLAM算法用于地图构建。Cartographer是谷歌开发的开源SLAM框架，它基于激光和视觉数据，能在二维和三维环境中实现定位和建图。它采用优化的位姿图优化方法，并使用分布式图优化算法实现实时的定位和地图构建。鉴于智慧药房比赛场地为规则环形，通过Cartographer的回环检测功能，可以在建图过程中不断提高地图的精确度。

1. map\_server

假设已经通过Cartographer算法获得了一张二维的栅格地图,通常是xxx.pgm和xxx.yaml两个文件。接下来利用扫描得到的地图进行自主导航。我们利用map\_server这个包加载已有的导航地图并将其转换为导航可用的图，也就是costmap，这个costmap提供地图中每个点的代价。map\_server主要是加载我们扫描到的地图发布出costmap地图,map\_server这个节点对外提供了2张2维3层的地图，其中2张是指global\_costmap和local\_costmap。一张用于move\_base中的全局规划器（global\_planner）,另外一张用于move\_base中的局部规划器（local\_planner）。每张地图有三层（staticlayer(静态层)、obstaclelayer(动态层)、inflationlayer(膨胀层)）。map\_server发布"/map"和"/map\_metadata"两个topic, "/map"则被AMCL接收用于匹配激光传感器数据。/map\_metadata 被move\_base接收。

1. AMCL定位

在基于激光导航的小车中定位数据来源有两个地方轮式里程计和AMCL包。其中AMCL(称为自适应蒙特卡洛定位)，这个包主要是利用当前激光雷达获取到的数据在已有的地图中进行搜索匹配，定位出小车在地图中的位置。AMCL接收当前激光雷达的数据（/scan）、TF变换树和初始位姿估计（/Initialpose），在map\_server发布的地图中 /map 匹配定位机器人在地图中的位置，AMCL包估计地图与里程计（即"odom"和"map “）之间的TF变换，形成一个误差项，修正"odom"坐标系。AMCL会将定位数据发布在TF树和”/amcl\_pose"这个topic中。

1. move\_base

目前，开源的自主导航框架中，百度的Apollo、Autoware以及Move Base是备受瞩目的几个选项。在这些框架中，Move Base更为聚焦于机器人的导航任务，并未涉及全面的自动驾驶方案。它为机器人的目标位置定义提供了一种灵活的方法，并根据机器人的传感器数据实现路径规划和导航。Move Base的通用性很强，适用于移动机器人、无人机以及AGV（自动引导车）等多种机器人平台。

由于Move Base已经集成到ROS的导航功能包中，其使用非常便捷。在当前的比赛场景中，该框架已经能够满足任务需求，这也体现了ROS中避免重复开发、充分利用现有资源的理念。

当我们知道导航所需的地图（由map\_server提供）和机器人当前的位置（通过amcl模块确定）后，便可以开始路径规划。move\_base包主要由全局规划器和局部规划器构成。全局规划器基于map\_server提供的global\_costmap地图，利用Dijkstra算法规划出全局路径。而局部规划器则在local\_costmap地图上动态规划，实现避障、绕行等功能，其中涉及的主要算法是Teb算法。

move\_base包在运行过程中，需要接收机器人当前的速度信息（这通常是通过里程计测量的"odom"与"base\_link"之间的变换得到）以及机器人在地图中的精确位置（这是由ACML模块计算出的"odom"与"map"之间的TF变换确定的）。随后，它会根据这些信息计算出底盘的速度和转角控制参数，并将这些参数发布到"/cmd\_vel"这个topic上。最终，机器人底盘节点会从该topic上接收这些速度和角度控制指令，实现底盘的闭环控制。

7、move\_base参数整定

参数整定的合适程度直接决定导航效果的好坏。主要是配置move\_base模块的五个参数文件，包括：

1. local\_costmap\_params.yaml（局部costmap参数文件）
2. global\_costmap\_params.yaml（全局地图参数文件）
3. teb\_local\_planner\_params.yaml(局部规划器参数文件)
4. global\_planner\_params.yaml（全局规划器参数文件）
5. move\_base\_params.yaml（move\_base运行参数）

两个costmap文件主要是配置地图，比如地图的膨胀半径，它影响路径规划的路线。两个planner文件主要是配置规划器的参数，对导航效果有至关重要的作用。为了实现精准的停靠与避障，需要设置各个导航参数适合实际环境。

### 数字和字母图像识别

图像识别使用opencv技术，opencv是一个开源的计算机视觉库，它包含了大量的图像处理、计算机视觉和机器学习方面的通用算法。

1. 数据准备

首先，根据官方数据集和网上给的数据集，对图像中的每个数字或字母进行标注，这样机器学习模型就能知道每个图像对应的标签是什么。

2. 图像预处理

读取图像：使用OpenCV的imread函数读取图像。

灰度化：将彩色图像转换为灰度图像，减少计算量并突出边缘信息。

二值化：通过阈值操作将图像转换为黑白二值图像，使数字或字母与背景分离。

去噪：使用形态学操作（如腐蚀和膨胀）去除图像中的噪声。

轮廓检测：通过查找轮廓来定位图像中的数字或字母。

3. 训练分类器

训练一个卷积神经网络（CNN）来识别数字或字母。这通常需要一个包含多个卷积层、池化层和全连接层的网络结构。卷积神经网络的基本结构包括输入层、卷积层、池化层以及全连接层。卷积层自带激活函数，通常使用ReLU。卷积层和池化层的组合可以在隐藏层中出现多次，进行灵活组合。全连接层位于若干卷积层和池化层之后，其输出层使用Softmax激活函数来做图像识别的分类。

4. 模型评估与优化

交叉验证：使用交叉验证来评估模型的性能。

调整超参数：通过调整学习率、批次大小、网络层数等超参数来优化模型性能。

正则化：使用正则化技术（如L1、L2正则化或Dropout）来防止过拟合。

5. 模型应用与测试

加载模型：加载训练好的模型。

预处理新图像：对需要识别的新图像进行与训练时相同的预处理步骤。

预测：使用加载的模型对新图像进行预测，得到识别结果。

6. 结果后处理

后处理：根据识别结果进行一些后处理操作，如纠正识别错误、平滑输出等。

最终将小车获取相应的信息（配送药品信息，手写数字板）通过opencv技术完成识别和输出。

### 药丸图像识别

药丸识别主要使用TensorFlow和Keras构建深度学习模型，进行图像分类。TensorFlow的原理主要基于数据流图（dataflow graph）来进行数值计算。原理如下：

1.计算图（Computation Graph）：TensorFlow使用计算图来描述有向图的数值计算过程。在计算图中，节点（Nodes）表示数学运算，边（Edges）表示在这些节点之间传输的多维数据（即张量，Tensors）。这些节点可以被分配到计算设备上（如CPU或GPU），从而实现并行的执行操作。

2.数据流：TensorFlow中的“Flow”直观地表达了张量之间通过计算相互转化的过程，即数据流。数据（张量）从图的源节点流动到目标节点，经过一系列的数学运算和处理，最终得到计算结果。

3.组件和模式：TensorFlow有一个重要的组件client（客户端），它通过Session的接口与master（主节点）及多个worker（工作节点）相连。其中，每一个worker可以与多个硬件设备（device）相连，比如CPU或GPU，并负责管理这些硬件。而master则负责指导所有worker按流程执行计算图。TensorFlow有单机模式和分布式模式两种实现。在单机模式下，client、master、worker全部在一台机器上的同一个进程中；在分布式模式下，client、master、worker可以在不同机器的不同进程中，同时由集群调度系统统一管理各项任务。

优化器（Optimizer）：在训练神经网络时，TensorFlow使用优化器来调整网络中的权重和偏置，以最小化损失函数。优化器基于梯度下降算法的原理，通过计算损失函数对权重和偏置的梯度，并沿梯度的反方向更新权重和偏置，从而逐渐使损失函数达到最小值。

总的来说，TensorFlow通过构建计算图来描述数值计算过程，利用数据流图的思想实现并行计算和分布式计算，并通过优化器来训练神经网络。这些原理使得TensorFlow能够高效地处理大规模数据和复杂的机器学习算法。

# 方案实现

在本章，本队将详细阐述实现方案的流程以及过程中遇到的各类问题，和改进措施。为了从技术上更加合理的分配，将从以下五个环节展开描述。

## 建图环节

在建图过程中，我们团队提出了两种方案。首先，我们尝试利用小车自带的Cartographer算法进行建图。由于智慧药房的场地是环形结构，我们预期Cartographer算法的回环检测功能能充分发挥其优势，帮助我们精确构建地图。因此，我们初期采用了这种算法进行建图，实际效果见下图所示。

|  |
| --- |
| 图1：cartographer建图 |

然而，从图中可以看出，建图效果并不如预期般理想。我们分析后认为，可能是因为场地布局过于规整，缺乏足够的特征点，导致Cartographer算法在构建地图时遇到了困难。在深入了解二维栅格地图的原理后，我们了解到每个栅格可以表示为存在障碍物（通常以特定标记表示）或可通过的空白区域。栅格的大小可以根据具体环境和任务需求来设定，通常是正方形或矩形。考虑到这一特性，我们决定采用第二种建图方案，即使用Photoshop软件手工绘制地图。我们绘制了png格式的地图，并随后将其转换为pgm格式，以便在后续导航任务中使用。具体的实现流程如下：

计算每个像素点代表的实际距离：比赛场地为4.9m\*3.8m 的长方形空间，以一个像素代表0.05m，经过如下公式转换可获得各个数据：

|  |
| --- |
| 表1:栅格地图参数 |
| |  |  |  |  | | --- | --- | --- | --- | |  | 画布 | 外框 | 内框 | | 大小 |  |  |  | | 起始位置 |  |  |  | |

最终得到如下图所示的地图：

|  |
| --- |
| 图2：photoshop绘制栅格地图 |

## 取药、配药环节

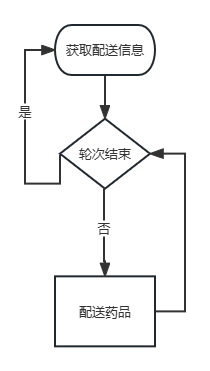
在药品的取送和配制环节中，共设有七个服务窗口。一旦配药小车识别到相关的配送指令，它会根据这些指令自动导航至指定的地点。为高效完成任务，我们可以将每一轮的工作划分为三个步骤。在首个步骤开始之前，小车会接收并确认配送信息。随后，在每个步骤中，小车会前往指定窗口取药，短暂停留后，对窗口上的手写数字进行识别，接着将药品送至相应位置，并最终返回起始点，准备开始下一步骤的工作。通过这种方式，配药小车能够有序、高效地完成整个取药、配药流程。

图3：多点导航实现框架

本队实现的具体做法如下：

首先，我们利用C++语言编写了一个ROS节点，该节点我们称之为多点导航节点。这个节点主要承担以下三个功能：

第一，运行导航功能包，将小车精准地放置在每个预定的目标点上。通过rviz中的init\_pose工具，我们不断调整小车的当前位姿，直到在rviz中显示的雷达点云数据与先前绘制的栅格地图相吻合，并记录这个精确的位姿信息。

第二，我们设计了一套服务通信机制。当需要识别配药信息或手写信息时，我们会向图像识别服务发送一个标志位。这个标志位采用二进制表示，其中1代表识别数字，而0则代表识别字母。服务接收到这个标志位后，会返回相应的识别结果。我们将这些识别结果存储在一个名为state[]的数组中，用于存储字母识别结果，并将数字识别结果直接打印出来。

第三，我们编写了一个话题订阅者，用于发布和订阅与move\_base导航节点相关的数据。具体来说，我们通过发布move\_base\_simple/goal话题数据，向move\_base导航节点发送目标点的位姿信息。同时，我们还订阅了move\_base\_simple/result话题数据，并在接收到数据后，通过回调函数解析其中的status数据字段。这个status字段中的数字3表示小车已成功到达目的地，而数字4则表示导航失败。通过这种方式，我们可以实时获取小车的导航结果。

## 答题环节

在答题环节主要的任务就是识别显示器中的配送信息以及手写数字板中的数字。本队通过编写服务器方，接受多点导航节点的请求数据（0，1），解析不同的数据后，调用相关的图像识别模型进行信息识别，并将结果返回给客户端。

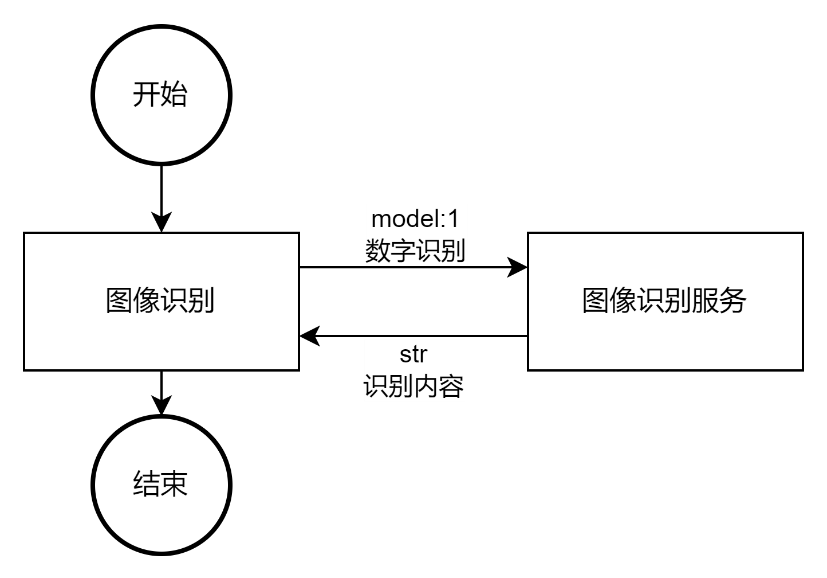
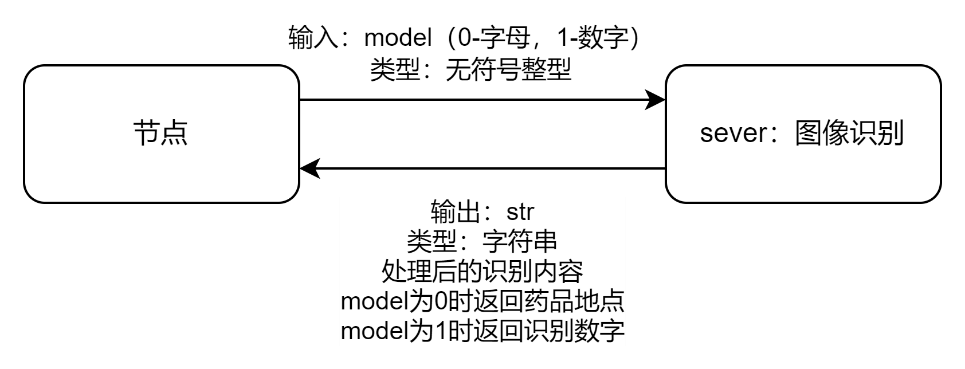


图4：答题环节实现框架

## 图像识别环节

图像识别环节需要实现的功能包括识别显示屏，获取药品及对应的目的地、识别手写板，获取数字并反馈。



通过编写ROS python节点，提供一个ROS服务，输入为整型，对应识别显示屏还是识别手写板，输出为字符串，为图像识别后获得的内容。

具体实现步骤如下：

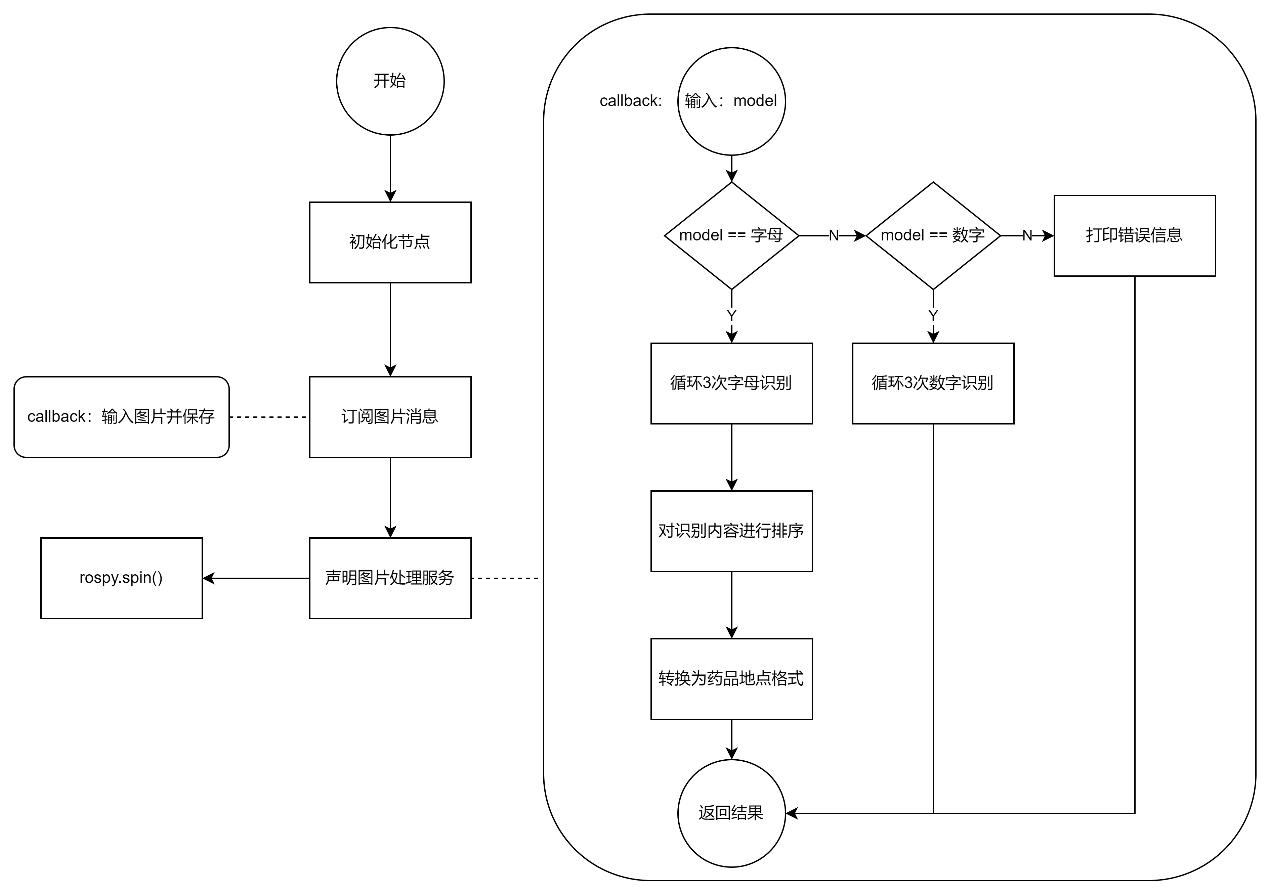


图5：图像识别具体实现框架

### 图片预处理

**方案一，轮廓筛选**

本方案主要依赖于识别板明显的粗框特征进行定位，并利用识别板内容的简单性和高对比度来获取其内容。

首先，将图片转换为灰度图，并采用自动阈值的最大类间方差法进行二值化处理，以选择能够最大化前景与背景类间方差的阈值。随后，提取图片中的所有轮廓，并根据轮廓的面积、多边形边数、矩形长宽比以及轮廓的包含关系进行筛选。经过筛选后，我们认定剩余轮廓中面积最小的轮廓为识别板区域。接着，在原始图片上对识别板区域进行矫正、裁剪和放缩，从而得到标准识别板图片。

接下来，为了获取标准识别板图片中的信息，我们再次将灰度图进行二值化处理（此时进行像素反转，以减少后续图像识别的计算量）。由于像素问题，进行一次膨胀腐蚀操作，使数字边缘更加圆润，笔画连接更加紧密。随后，获取图片的最外围轮廓，并进行面积筛选和包含关系筛选，以避免因数字笔画断裂而导致的重复框选。最后，以筛选后的轮廓为依据，对二值化后的图片进行裁剪，并放缩至统一尺寸，按实际空间顺序排列，完成图片的预处理。

**方案二：区域生长**

区域生长方法从某个像素点出发，依据一定准则逐步加入邻近像素，当满足特定条件时停止生长。鉴于识别板背景颜色统一，我们可以从板中一点开始向外生长，以找到整个识别板，并获取其内容。

假设识别板位于图片中心附近，我们以图片中心为起点进行区域生长。生长准则设定为像素差不超过预设阈值，以确保生长区域能够包围识别板信息而不超出其边界。生长完成后，对生长区域进行裁剪，并认定裁剪区域为识别板区域。之后，采用与方案一中相同的方法获取标准识别板图片中的内容。

**方案三：像素筛查**

本方案主要利用识别板内容颜色最深的特点进行筛选。

首先，将图片转换为灰度图并进行二值化处理（同样进行像素反转），采用设定阈值的方式，使图中仅保留识别板中的有效信息。然而，由于环境中可能存在干扰项，需要进行进一步处理。通过梯度运算放大有用信息，多次闭运算连接有用信息，再进行一次开运算以减少干扰项。最后，认为剩余区域中轮廓面积最大的为有用信息区域，稍微向外扩张后进行裁剪，认定其为识别板区域。后续步骤与方案一中获取识别板内容的方法相同。

鉴于本次比赛识别板带有固定边框且大小一致，经过综合比较，我们决定以轮廓筛选作为首选方案，其他两种方案作为备选。

表1：图像处理方案对比

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| 轮廓筛选 | 效果好，裁剪准确 | 筛查条件参数要求较高  需要计算和调试 |
| 区域生长 | 能适应各种识别板 | 对小车定点停车要求高  区域生长起点存在误差 |
| 像素筛查 | 实现简单  在特定条件下效果极佳 | 对环境要求较高  避免背景深色像素 |

### 数字和字母图像识别

鉴于图片预处理工作取得了显著成效，显著降低了图片识别的难度，因此，我们不需要构建过于复杂的图片识别模型。在识别工作开始之前，我们对图片进行了数据统一化处理，以减少其他干扰因素对分类结果的影响。我们主要考虑了两种模型进行图像内容的识别。

方案一是采用KNN分类器。KNN是一种在监督学习中广泛应用的简单而有效的分类算法。它通过计算测试样本与训练集中每个样本之间的距离，选择距离最近的K个样本，并根据这些样本的类别频数来确定测试样本的预测类别。我们先通过图片预处理得到数据集，再通过手工分类得到参考集，然后加载KNN模型，设置合适的K值后即可进行图片识别。

方案二是利用CNN模型。CNN是卷积神经网络的简称，它擅长自动提取图像中的特征并进行分类。CNN模型通常由卷积层、池化层和全连接层组成。我们设计的CNN模型结构为：输入数据后，依次通过卷积层、非线性激活层、池化层，再重复一次卷积、激活、池化的过程，最后通过全连接层得到输出。对于字母识别，我们使用标准字体的预处理图片进行训练；而对于手写数字识别，由于字体种类较多，为防止过拟合，我们选择使用mnist数据集进行训练，以便模型能更好地适应各种字体的变化。

表2：识别算法模型对比

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| KNN | 操作简单，无需训练 | 参考集大，内存占用大，消耗算力 |
| CNN | 训练得当准确率高  泛化能力强  对硬件要求低 | 专业性强  需要进行合适的训练 |

经实践比较，本队选用CNN模型进行图片识别。在效果层面有着不错的效果。综上所述，图片检测中先以轮廓筛选获取有效信息，然后使用CNN模型进行图像识别。

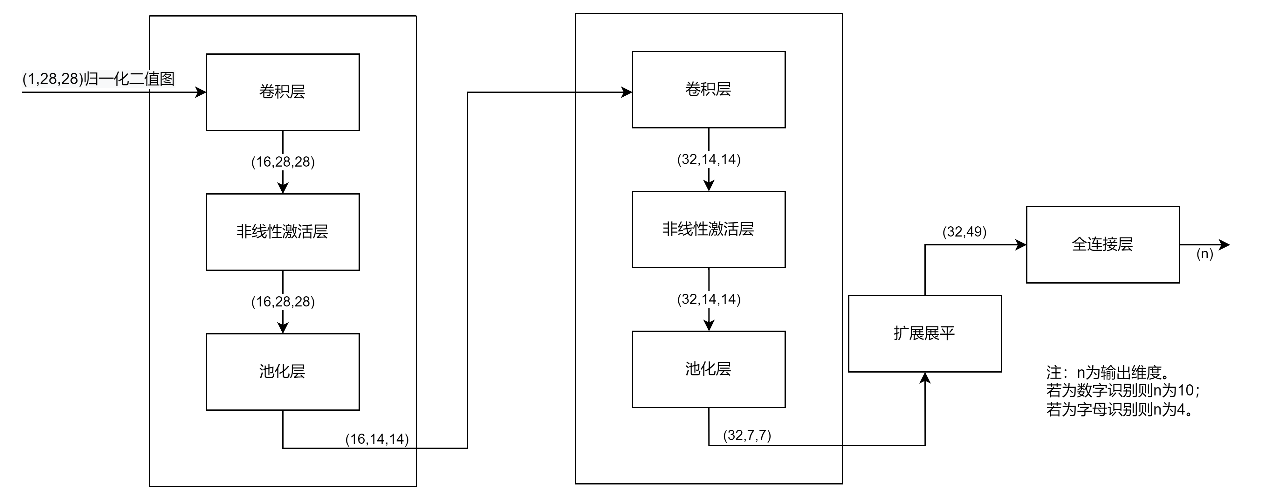


图6：模型搭建

### 药丸图像识别

TensorFlow实现图片识别的过程可以大致分为以下几个步骤：

1. 环境搭建：首先，安装TensorFlow库以及可能需要的其他依赖库，如NumPy、Matplotlib等。
2. 准备数据集：使用官方给的示例图片，将图片标记为“圆形药丸、双色胶囊、鱼肝油、单色胶囊、椭圆药丸”五类，将80%的图片作为训练集，剩下20%作为测试集。此外，可以在网上寻找合适的数据集扩充。
3. 数据预处理：在将数据输入模型之前，通常需要进行一些预处理操作，以提高模型的性能和准确率。这些预处理操作包括图像尺寸的调整、归一化、数据增强等。在train()函数中，定义了批处理大小、图像高度和宽度。使用tf.keras.preprocessing.image\_dataset\_from\_directory()函数从数据集中加载训练和验证数据集。打印出类别名称和一些数据集的信息。
4. 构建模型：使用TensorFlow构建一个图像识别模型。定义了一个Sequential模型，包括Rescaling层、卷积层、池化层和全连接层。编译模型时使用了Adam优化器、SparseCategoricalCrossentropy损失函数和准确率作为评估指标。
5. 训练模型：将预处理后的数据集输入模型进行训练。在训练过程中，设置一些超参数，如学习率、批次大小、训练轮次等。
6. 评估模型：使用测试集对训练好的模型进行评估。评估指标可能包括准确率、精确率、召回率等。绘制训练和验证准确率以及损失曲线，并调用plot\_confuse()函数显示混淆矩阵。在plot\_confusion\_matrix()函数中，计算混淆矩阵的准确率和错误率，并绘制混淆矩阵。如果需要，还可以将混淆矩阵归一化。

## 扩展功能

### 语音模块

在药品获取与配送流程中，为提升用户体验的人性化程度，我们起初尝试采用ROS（机器人操作系统）中的sound\_play功能包来实现语音提醒服务。这一功能包能够接收文字信息并将其转换为语音播放，同时支持播放本地存储的音频文件。然而，在实际应用中，我们发现使用sound\_play功能包需要持续运行一个语音服务节点，这导致了系统资源的持续占用，相较于其他播放方式效率较低。因此，为了优化这一环节，我们决定转而使用C++中的库函数system来调用Linux系统中的aplay工具。通过直接在代码中操作aplay，我们能够更加高效地播放音频文件，从而实现了药品获取与配送中的语音提醒功能。

### 导航恢复模块

在导航运行过程中因为定位的误差，存在小车无法导航至下一个点而出现“假死”的现象，有必要编写一个恢复模块，具体实现如下：

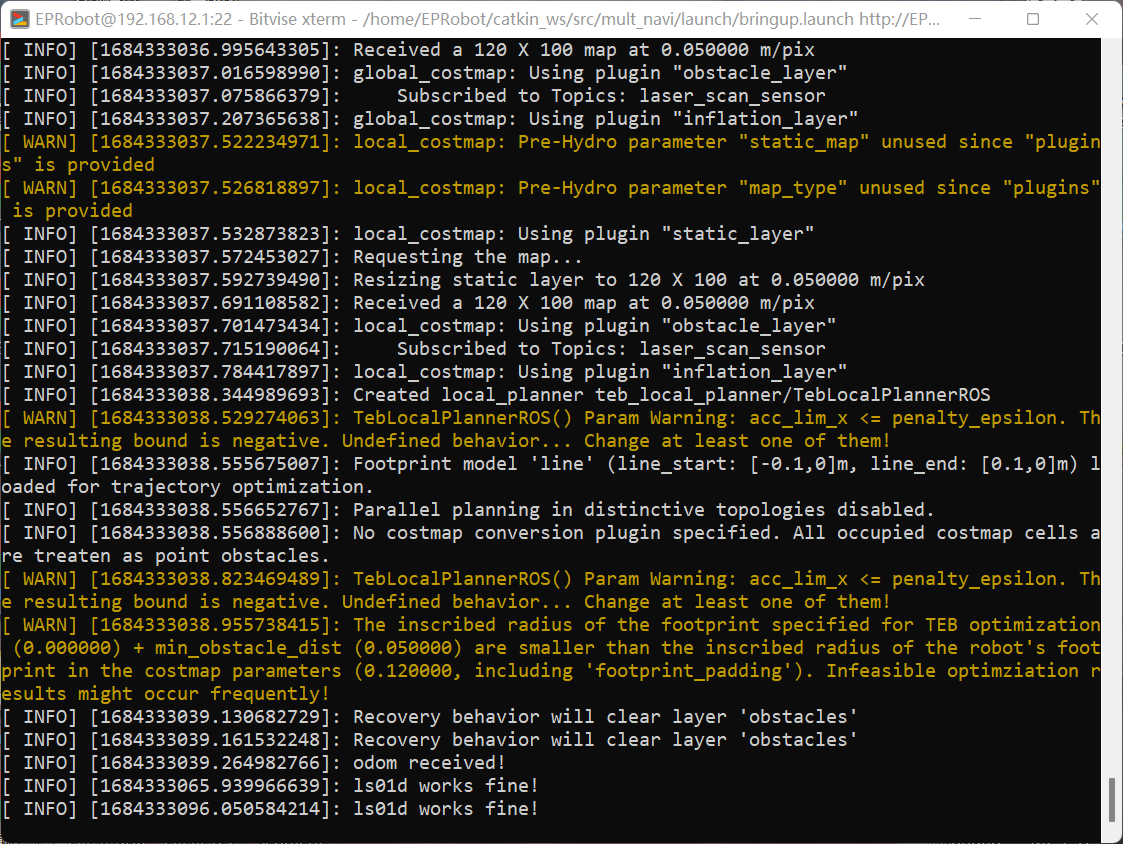
1. 编写ROS订阅方，当导航失败后，节点会接收到错误标志；
2. 初始化小车底盘速度，此时向/cmd\_vel话题发布后退速度，使其避开前方障碍物，同时在移动状态下，使其通过amcl定位算法不断修正当前位姿；
3. 重置地盘速度为0，并发布到/cmd\_vel话题，使小车停止。

# 测试分析

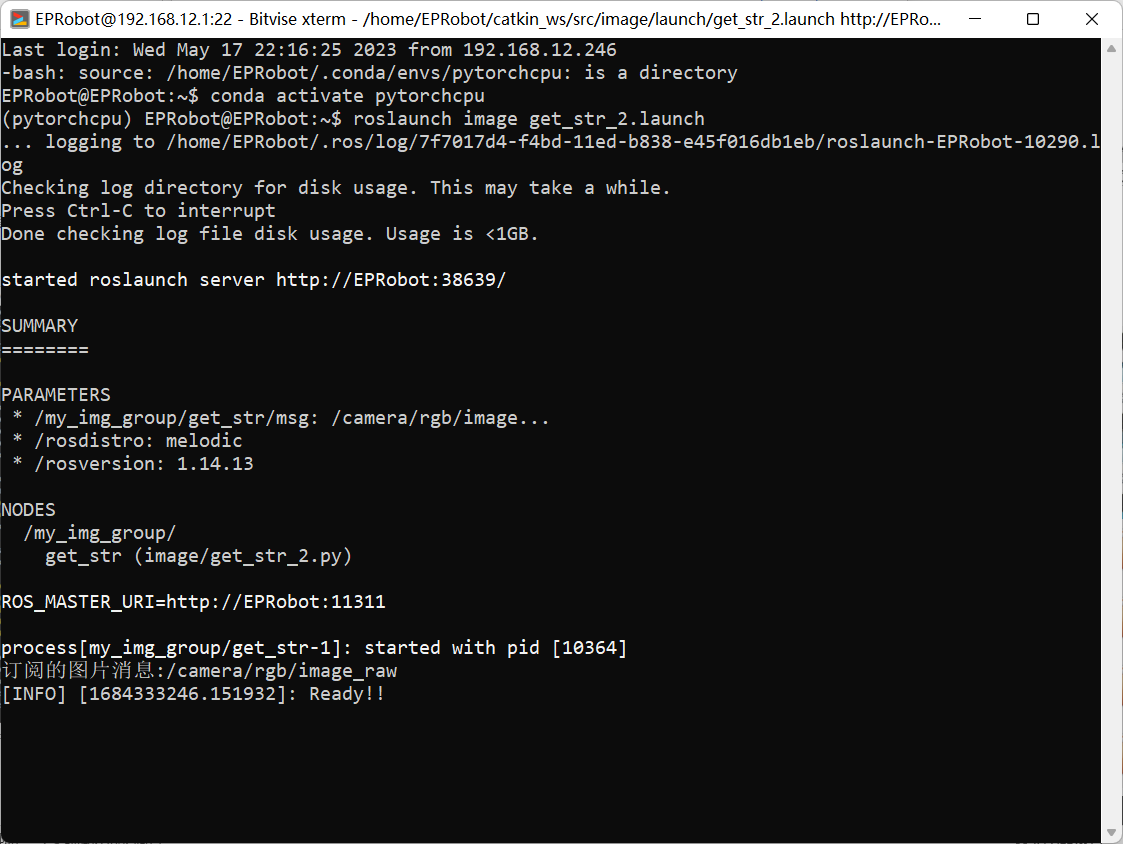
## 多点导航

为了便于测试导航的实际效果，编写通过相同服务的信息输入来判断多点导航的正确性。

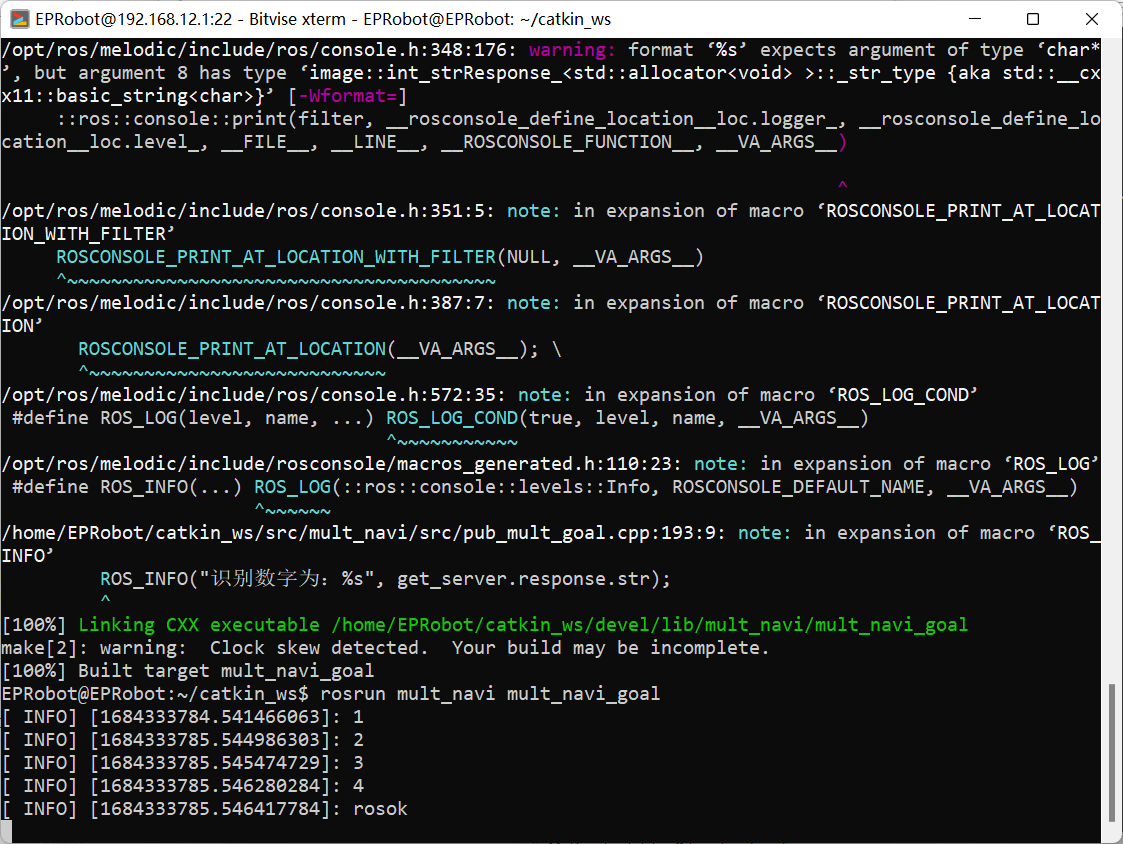
1. 启动navigation功能包



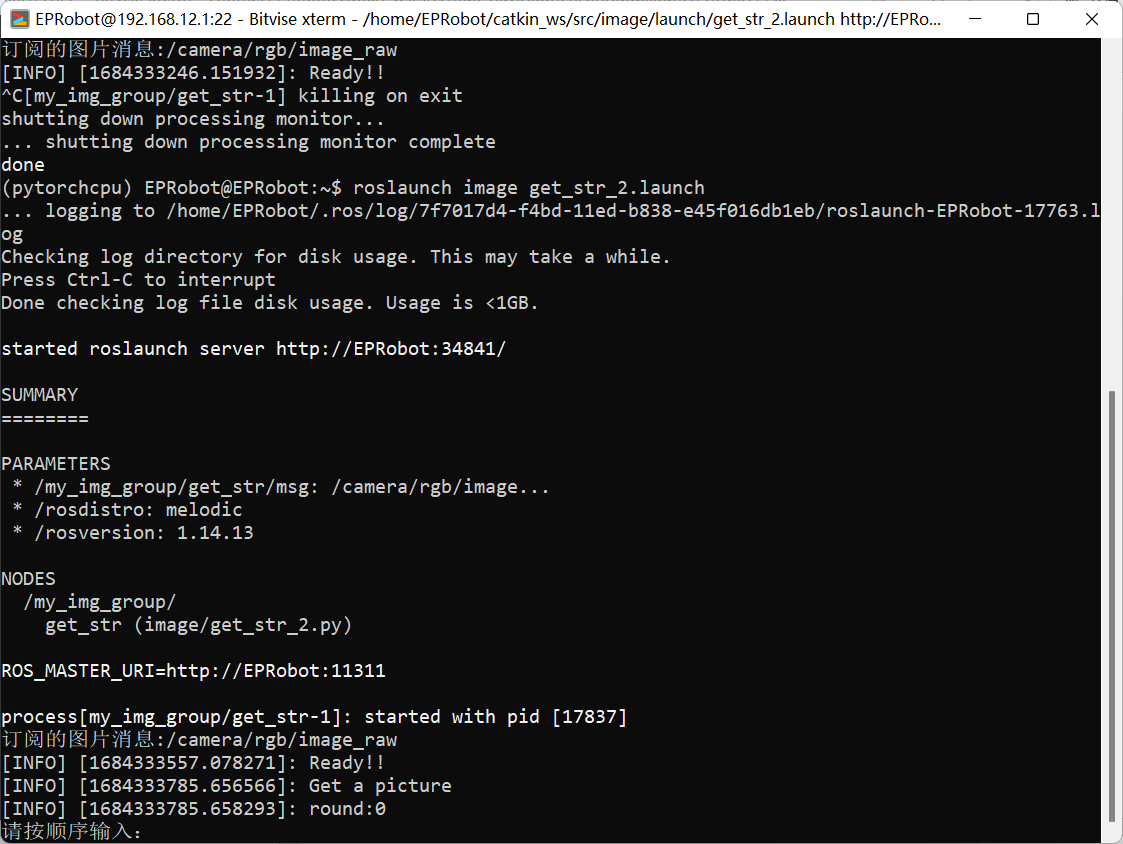
1. 启动修改后的图像处理节点（可手动输入信息，便于调试）



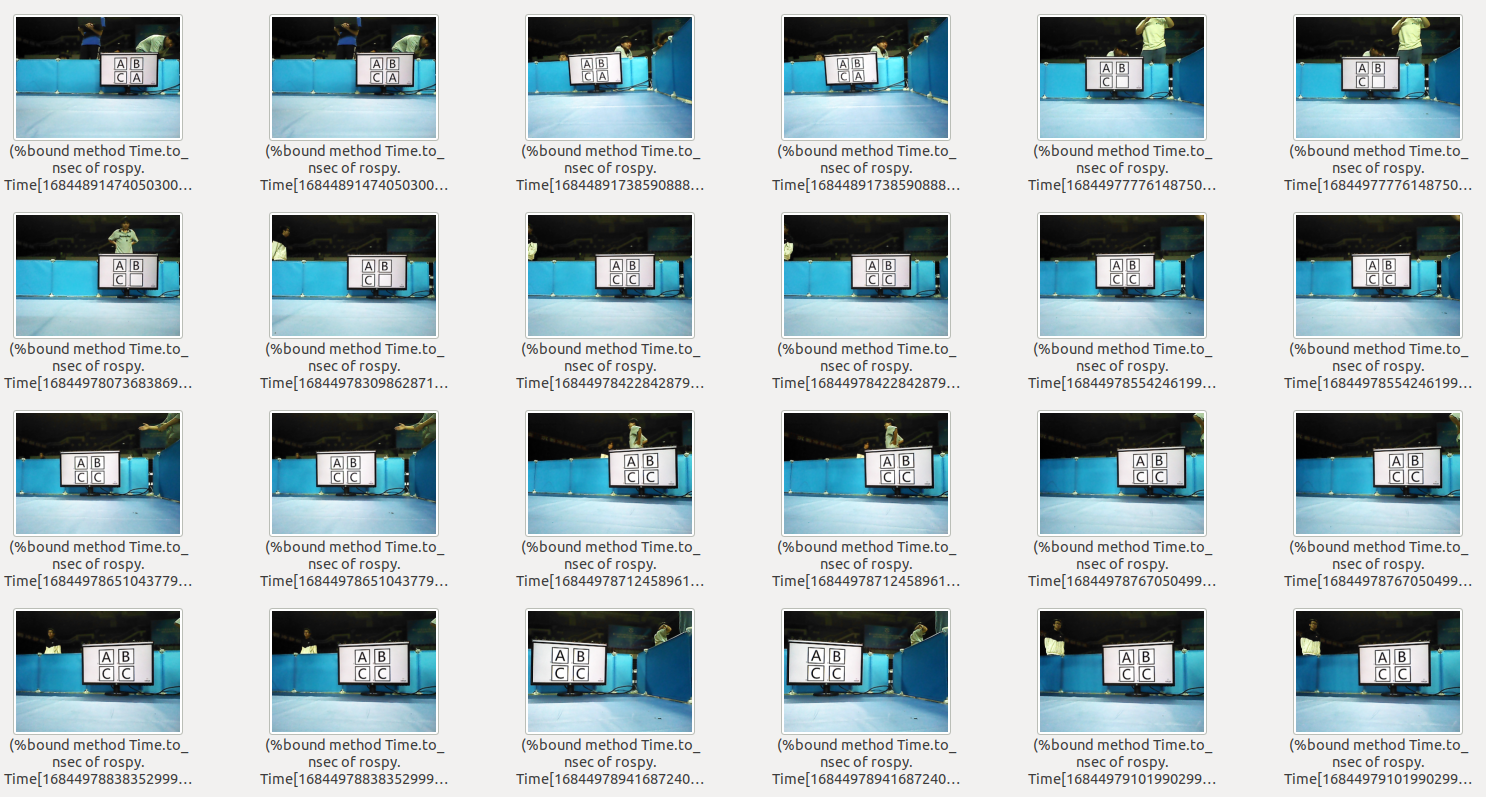
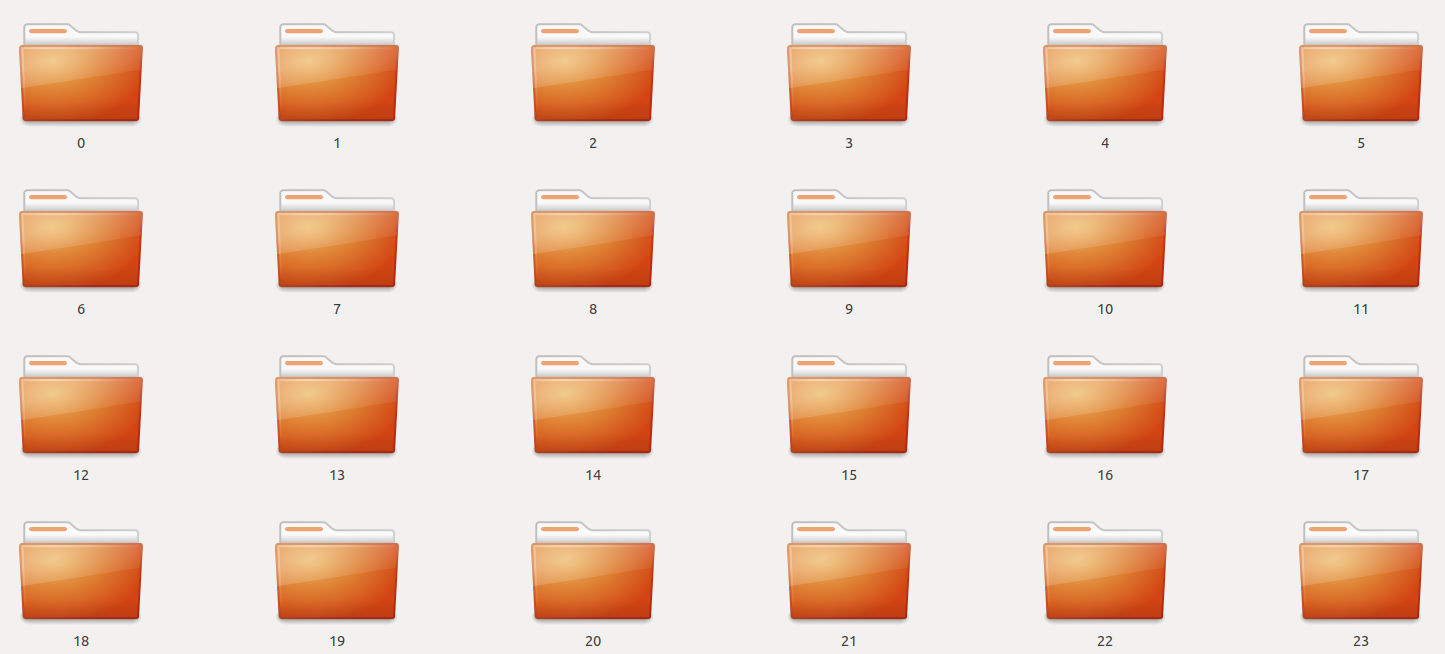
1. 启动多点导航节点



1. 输入指定配送顺序

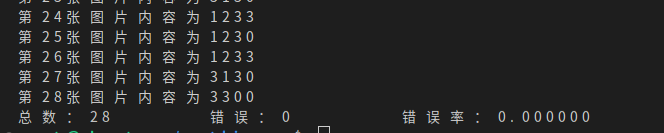


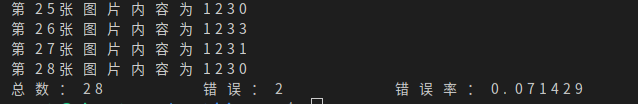
## 数字和字母图像识别

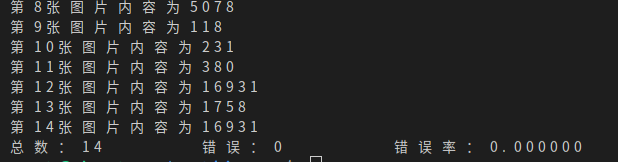


采用自动化测试，首先采集模拟比赛时拍摄的配送信息识别图像。

1. 然后进行统计分析。由图可知，在28张图片中，发生错误两张，错误率为7.1429%。

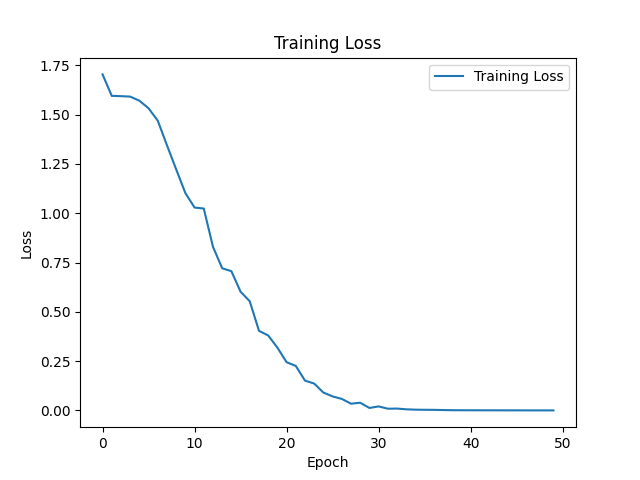


2）通过识别过程信息可得：在错误的两张图片中，都是在图片预处理发生了错误，经过修正在数据集中图片识别正确率达到100%

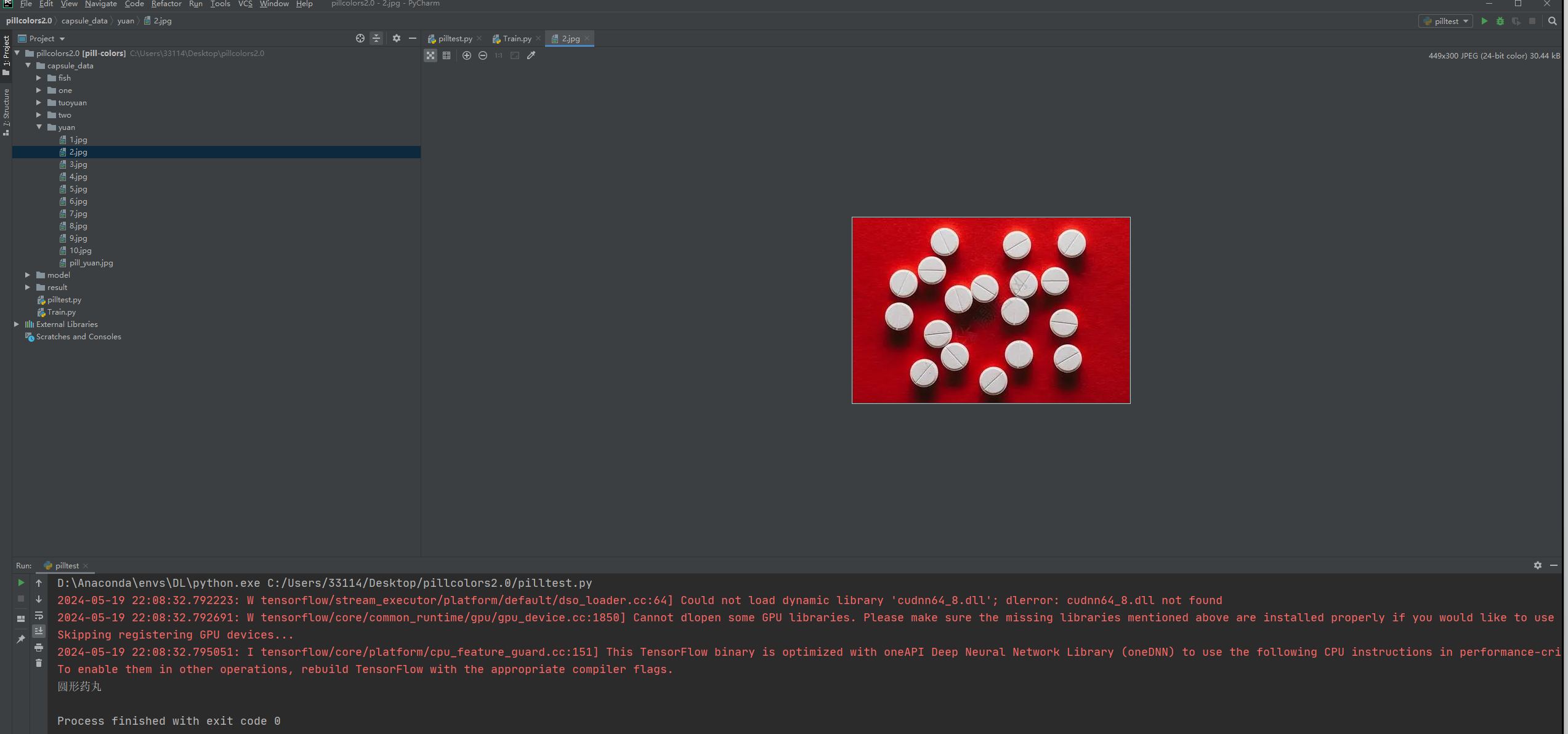


## 药丸图片识别

训练集损失函数：



测试集测试图片：



## 语音功能测试

|  |
| --- |
| 图Sound\_play功能包播放音频 |



图Aplay工具播放音频

# 作品总结

在本次智慧药房比赛中，本队成员充分发挥了各自的专长，分工合作，共同学习了小车底盘控制模型与算法、计算机图像识别与处理、深度学习以及ROS等相关知识。在此基础上，我们独立编写了相关程序，成功实现了小车自主完成药品分拣与配送的任务。

在硬件学习方面，我们深入研究了STM32单片机的驱动方法，对其有了更为深入的了解。同时，我们也对阿克曼底盘的运动学模型进行了充分学习，掌握了其运动原理。在PID控制算法的实现过程中，我们虽然遇到了不少困难，但通过团队协作和不懈努力，最终克服了这些难题。

在图片识别任务中，我们探索了多种常见的图片识别技术，深刻感受到了opencv和深度学习技术的强大之处。这些工具的使用极大地提高了我们的工作效率。当然，在具体实现过程中，我们也遇到了很多挑战，如矩阵计算、图片处理以及深度学习中的复杂概念等。但通过此次比赛，我们更加深刻地认识到，要善于利用现有工具，避免重复造轮子，同时拓宽信息获取渠道，以提高工作效率。

在任务执行过程中，由于前期基础知识的不足和解决问题能力的欠缺，我们花费了大量时间在学习基本技能和修复错误上。然而，正是这些经历让我们从零开始接触并掌握了ROS的基本通信方式和各类功能包的使用，使我们在机器人处理方面变得更加高效和便捷。

总的来说，本次比赛不仅锻炼了我们的团队协作能力和解决问题的能力，还让我们在实践中掌握了丰富的知识和技能。我们将继续努力，不断提升自己在各个领域的专业水平。

参考文献

1. 李崇智.基于模糊PID的悬吊式机械臂重力补偿控制系统设计[J].计算机测量与控制,2023,31(01):100-105+112.
2. 蒋红梅,黄鹏,唐军.基于机器人操作系统的自动驾驶小车系统研究[J].科技创新与应用,2023,13(17):46-49.
3. 高欣宇.基于激光雷达的智能小车SLAM及导航技术研究[D].沈阳工业大学,2023.
4. 许皓然.室内智能小车定位与智能巡线系统的研究[D].四川大学,2023.
5. 周琪.图书馆智能还书小车定位与导航研究[D].西华师范大学,2023.DOI:10.27859/d.cnki.gxhsf.2023.000821.
6. 李加定,缪文南.基于OpenMV的智能送药小车的设计[J].电子设计工程,2024,32(02):162-166.DOI:10.14022/j.issn1674-6236.2024.02.035.

附录

|  |
| --- |
| 附录1 |
| 多点导航代码 |
| 1. #include "tf/tf.h" 2. #include "cmath" 3. #include "ros/ros.h" 4. #include <iostream> 5. #include "std\_msgs/Bool.h" 6. #include "std\_msgs/String.h" 7. #include "mult\_navi/get\_state.h" 8. #include "tf/transform\_listener.h" 9. #include "geometry\_msgs/Twist.h" 10. #include "std\_msgs/Int8MultiArray.h" 11. #include <tf2/LinearMath/Quaternion.h> 12. #include <geometry\_msgs/PoseStamped.h> 13. #include <move\_base\_msgs/MoveBaseActionResult.h> 14. #include <geometry\_msgs/PoseWithCovarianceStamped.h> 15. #include "image/int\_str.h" 16. int state[4][4] = {{1, 4, 5, 0},{2, 4, 6, 0},{3, 4, 7, 0}}; 17. bool next\_pub = false, restore = false;     *//下一步导航、恢复标志位* 18. uint i=0, j=0;  *//状态下标* 19. int server\_mode; *//识别模式* 20. struct dispatch 21. { 22. int n = 0; 23. int key\_var[4][2]; 24. }; 25. *//生成调度* 26. struct dispatch create\_dispatch(image::int\_str letter) 27. { 28. struct dispatch dispatch; 29. int len = letter.response.str.length() / 2; 30. for(int i = 0; i < len; i++) 31. { 32. *// char a = letter.response.str[i \* 2] - '0';* 33. *// char b = letter.response.str[i \* 2 + 1] - '0';* 34. dispatch.key\_var[dispatch.n][0] = letter.response.str[i \* 2] - '0'; *//药* 35. dispatch.key\_var[dispatch.n][1] = letter.response.str[i \* 2 + 1] - '0'; *//地点* 36. dispatch.n++; 37. } 38. return dispatch; 39. } 40. void callback1(const move\_base\_msgs::MoveBaseActionResult::ConstPtr& callback1) *//订阅导航结果* 41. { 42. if(callback1->status.status == 3) 43. { 44. if(i == 0 && j == 0) 45. { 46. server\_mode = 0; 47. } 48. next\_pub = true; 49. } 50. if(callback1->status.status == 4) 51. { 52. restore = true; 53. } 54. } 55. void callback2(const std\_msgs::Int8MultiArray::ConstPtr& new\_state) 56. { 57. int x = 0, y = 0,z = 0; 58. for(x;x<3;x++) 59. { 60. for(y;y<3;y++) 61. { 62. state[x][y] = new\_state->data[z]; 63. z++; 64. } 65. } 66. } 67. bool next\_move(const mult\_navi::get\_state::Request& req, const mult\_navi::get\_state::Response& resp) 68. { 69. } 70. *//导航* 71. #define to\_goal(point) \ 72. position[point].header.stamp = ros::Time::now();\ 73. pub\_pose.publish(position[point]);\ 74. while(!next\_pub && ros::ok())\ 75. {ros::spinOnce(); if(restore) {restore = false;pub\_restore.publish(vel);ros::Duration(2).sleep();vel.linear.x = 0;pub\_restore.publish(vel);vel.linear.x = -0.2;next\_pub = true;}};\ 76. next\_pub = false 78. int main(int argc, char \*\*argv) 79. { 80. setlocale(LC\_ALL,""); 82. bool start = true; 83. ros::init(argc, argv, "pub\_pose"); 84. ros::NodeHandle nh; 85. ros::Publisher pub\_pose = nh.advertise<geometry\_msgs::PoseStamped>("/move\_base\_simple/goal", 1);   *//发布move\_base导航位姿* 86. ros::Publisher pub\_restore = nh.advertise<geometry\_msgs::Twist>("/cmd\_vel", 1);   *//恢复* 87. ros::Subscriber sub\_navi\_result = nh.subscribe<move\_base\_msgs::MoveBaseActionResult>("/move\_base/result",1,callback1);  *//订阅导航结果* 88. ros::ServiceClient get\_dispatch = nh.serviceClient<image::int\_str>("/my\_img\_group/get\_str");    *//识别信息 1:数字    0：字母* 89. *//初始化导航点位姿* 90. geometry\_msgs::PoseStamped position[9]; 91. tf2::Quaternion qtn[9]; 92. qtn[0].setRPY(0, 0, M\_PI / 2); qtn[4].setRPY(0, 0, M\_PI \* 1.5); 93. qtn[1].setRPY(0, 0, M\_PI); qtn[2].setRPY(0, 0, M\_PI \* (16.5/18)); qtn[3].setRPY(0, 0, M\_PI \* (16.5/18)); 94. qtn[5].setRPY(0, 0, -M\_PI \* (2.5/18)); qtn[6].setRPY(0, 0, 0); qtn[7].setRPY(0, 0, -M\_PI \* (2.5/18));qtn[8].setRPY(0, 0, 0); 96. for(int i = 0; i<9; i++) 97. { 98. position[i].header.frame\_id = "map"; 99. position[i].pose.orientation.x = qtn[i].getX(); 100. position[i].pose.orientation.y = qtn[i].getY(); 101. position[i].pose.orientation.z = qtn[i].getZ(); 102. position[i].pose.orientation.w = qtn[i].getW(); 103. position[i].pose.position.z = 0; 104. } 106. position[0].pose.position.x = 1.89; position[0].pose.position.y = 0.22; *//内测5cm* 107. position[1].pose.position.x = -0.55; position[1].pose.position.y = 0.8;*//* 108. position[2].pose.position.x = -1.0; position[2].pose.position.y = 1.68;*//* 109. position[3].pose.position.x = 0; position[3].pose.position.y = 1.68;*//* 110. position[4].pose.position.x = -1.85; position[4].pose.position.y = -0.26; 111. position[5].pose.position.x = -0.03; position[5].pose.position.y = -1.68;*//* 112. position[6].pose.position.x = 0.03;  position[6].pose.position.y = -0.8;*//* 113. position[7].pose.position.x =0.43;   position[7].pose.position.y = -1.68;*//* 114. position[8].pose.position.x = 0.8; position[8].pose.position.y = -0.8; 116. *//等待服务启动成功* 117. *//方式1* 118. ros::service::waitForService("/my\_img\_group/get\_str"); 119. *//初始化后退速度* 120. geometry\_msgs::Twist vel; 121. vel.angular.x = 0;vel.angular.y = 0;vel.angular.z = 0; 122. vel.linear.x = -0.2;vel.linear.y = 0;vel.linear.z = 0; 123. ROS\_INFO("1"); 124. *// ros::Rate r(1);* 125. ros::Duration(1).sleep(); 126. ROS\_INFO("2"); 127. image::int\_str get\_server; 128. ROS\_INFO("3"); 129. *// nh.setParam("save\_pic",0);* 130. struct dispatch state\_message; 131. ROS\_INFO("4"); 132. int temp\_goal; 133. while(ros::ok()) 134. { 135. ROS\_INFO("rosok"); 136. *//每一轮次开始进行字母识别* 137. get\_server.request.mode = 0; 138. if(get\_dispatch.call(get\_server)) 139. { 140. if(get\_server.response.str.length() != 0) *//结果为空* 141. { 142. state\_message =  create\_dispatch(get\_server); 143. } 144. else*//识别失败* 145. { 146. *//后退* 147. pub\_restore.publish(vel); 148. ros::Duration(2).sleep(); 149. vel.linear.x = 0; 150. pub\_restore.publish(vel); 151. vel.linear.x = -0.2; 152. *//回到识别点* 153. to\_goal(0); 154. } 155. } 156. *//拍照* 157. *// nh.setParam("save\_pic",1);* 158. *// r.sleep();* 159. *//执行state* 160. for(int i =0; i < state\_message.n; i++) 161. { 162. *//取药* 163. ROS\_INFO("开始取药"); 164. temp\_goal = state\_message.key\_var[i][0]; 165. to\_goal(temp\_goal); 166. ROS\_INFO("11111"); 167. system("echo ncut1234 | sudo -S chrt -r 99 aplay ~/put.wav"); 168. *// r.sleep();* 169. ros::Duration(1).sleep(); 170. ROS\_INFO("11111"); 171. *//数字点* 172. ROS\_INFO("去数字点"); 173. to\_goal(4); 174. *//每一轮次进行数字识别* 175. ROS\_INFO("识别数字"); 176. *//拍照* 177. *// nh.setParam("save\_pic",1);* 178. get\_server.request.mode = 1; 179. if(get\_dispatch.call(get\_server)) 180. { 181. ROS\_INFO("识别数字为：%s", get\_server.response.str); 182. } 183. *//送药* 184. ROS\_INFO("开始送药"); 185. temp\_goal = state\_message.key\_var[i][1] + 5; 186. to\_goal(temp\_goal); 187. system("echo ncut1234 | sudo -S chrt -r 99 aplay ~/remove.wav"); 188. *// r.sleep();* 189. ros::Duration(1).sleep(); 190. *//起点* 191. ROS\_INFO("回到起点"); 192. to\_goal(0); 193. } 194. *//清空调度消息* 195. state\_message.n = 0; 196. } 197. return 0; 198. } |

|  |
| --- |
| 附录2 |
| 图像识别主要代码 |
| 1. *#图片识别服务* 2. *#!/usr/bin/env python3* 3. *# coding:utf-8* 4. from image.srv import int\_str 5. import rospy 6. from sensor\_msgs.msg import Image 7. import test\_letter 8. import test\_num 9. import my\_dispose 10. class GetStr(): 11. def \_\_init\_\_(self, name, msg, time, n): 12. """ 13. name：节点名 14. msg：图片消息 15. sever\_letter：字母服务名 16. sever\_num：数字服务名 17. time： 18. n 19. mode如下 20. 0:字母 21. 1:数字 22. """ 23. self.msg = msg  *# 图片消息* 24. self.img = None  *# 保存图片消息* 25. self.f = 1.0 / time  *# 重复频率* 26. self.n = n  *# 重复次数* 27. self.get\_picture = True  *# 图片消息标志位* 28. self.cnn\_letter = test\_letter.Test()  *# 字母模型* 29. self.cnn\_num = test\_num.Test()  *# 数字模型* 30. rospy.init\_node(name, anonymous=True)  *# 初始化节点* 31. rospy.Subscriber(msg, Image, self.\_\_callback\_msg)  *# 订阅图片消息* 32. rospy.Service(name, int\_str, self.\_\_callback)  *# 声明服务* 33. rospy.loginfo('Ready!!') 34. rospy.spin()  *# 进入循环* 35. def fun\_letter(self, img): 36. """ 37. 字母识别 38. 输入opencv支持格式图片 39. 输出图片识别结果，为一维列表 40. """ 41. cv\_img = my\_dispose.roscv\_msg(img) 42. try: 43. imgs = my\_dispose.letter(cv\_img) 44. s = ''.join(self.cnn\_letter.test(imgs)) 45. except Exception as e: 46. rospy.loginfo(e) 47. s = '' 48. return s 49. def fun\_num(self, img): 50. """ 51. 数字识别 52. 输入opencv支持格式图片 53. 输出图片识别结果，为一维列表 54. """ 55. cv\_img = my\_dispose.roscv\_msg(img) 56. try: 57. imgs = my\_dispose.letter(cv\_img) 58. s = ''.join(self.cnn\_num.test(imgs)) 59. except Exception as e: 60. rospy.loginfo(e) 61. s = '' 62. return s 63. def \_\_callback\_msg(self, img): 64. """ 65. 保存获取的图片 66. """ 67. if self.get\_picture == False: 68. self.img = img 69. self.get\_picture = True 70. rospy.loginfo('Get a picture') 71. def recycled(self): 72. """ 73. 迭代器，确保获得最新图片 74. """ 75. self.get\_picture = False 76. rate = rospy.Rate(self.f) 77. for i in range(self.n): 78. while self.get\_picture == False: 79. pass 80. rospy.loginfo('round:%d' % i) 81. yield i 82. self.get\_picture = False 83. rate.sleep() 84. def \_\_callback(self, req): 85. """ 86. mode如下 87. 0:字母 88. 1:数字 89. """ 90. mode = req.mode 91. if mode == 0: 92. for i in self.recycled(): 93. s = self.fun\_letter(self.img) 94. rospy.loginfo('output:"%s"' % s) 95. if len(s) == 4: 96. ans = []  *# 存储药品地点对* 97. for index, value in enumerate(s): 98. if value == '0':  *# 筛除空* 99. continue 100. ans.append(value + str(index)) 101. ans.sort(key=lambda x: x[0], reverse=True)  *# 按照cba送药顺序* 102. s = ''.join(ans) 103. break 104. elif mode == 1: 105. for i in self.recycled(): 106. s = self.fun\_num(self.img) 107. rospy.loginfo('output:"%s"' % s) 108. if len(s) != 0: 109. break 110. else: 111. s = 'No Model' 112. rospy.loginfo(s) 113. rospy.loginfo('OUTPUT:"%s"' % s) 114. return int\_str.\_response\_class(s) 115. if \_\_name\_\_ == '\_\_main\_\_': 116. name = 'get\_str' 117. param = '/my\_img\_group/get\_str/msg' 118. try: 119. msg = rospy.get\_param(param) 120. except Exception: 121. msg = '/usb\_cam/image\_raw' 122. print('订阅的图片消息:%s' % msg) 123. while rospy.has\_param(msg) == False: 124. print('等待%s' % msg) 125. time = 0.1 126. n = 3 127. GetStr(name, msg, time, n) 129. *#图片预处理* 130. def get\_waikuan( 131. cv\_img, 132. enable\_baohan=True, 133. min\_area=15000, 134. max\_area=10\*\*10, 135. *# 多边形近似参数，越小越精准* 136. approxPolyDP\_epslion=0.02, 137. *# 长宽比系数，越小越接近正方形* 138. min\_wh\_rate=0.2 - 0.1, 139. max\_wh\_rate=0.2 + 0.1, 140. *# 中心距离* 141. min\_center\_distance=20, 142. *# 向内收缩* 143. shousuo\_x=0.03, 144. shousuo\_y=0.005, 145. size=(600, 400), 146. ): 147. """ 148. 获取识别板 149. """ 150. contours = \_my\_dispose.edge(cv\_img)  *# 获取轮廓* 151. contours2 = \_my\_dispose.mianji\_contourArea( 152. contours, max\_area, min\_area)  *# 面积筛选* 153. sibianxing = \_my\_dispose.sibianxing( 154. contours2, approxPolyDP\_epslion)  *# 多边形筛选* 155. juxing = \_my\_dispose.changkuanbi( 156. sibianxing, min\_wh\_rate, max\_wh\_rate)  *# 长宽比筛选* 157. if enable\_baohan: 158. dingweikuang, jiegou = \_my\_dispose.baohan( 159. juxing, min\_center\_distance)  *# 包含条件筛选* 160. else: 161. jiegou = juxing 162. jiegou.sort(key=lambda x: cv.contourArea(x))  *# 获得最内部的轮廓* 163. img = \_my\_dispose.picture\_cutting\_hull(cv\_img, jiegou[0])  *# 图片变换裁剪* 164. img = \_my\_dispose.cut(img, img.shape[1] \* shousuo\_x, img.shape[0] \* shousuo\_y, 165. img.shape[1] \* (1 - shousuo\_x), img.shape[0] \* (1 - shousuo\_y)) 166. img = \_my\_dispose.image\_resize(img, size)  *# 图片放缩* 167. return img 168. def get\_neirong(cv\_img, max\_area, min\_area): 169. if len(cv\_img.shape) == 2: 170. img = cv\_img 171. else: 172. img = cv.cvtColor(cv\_img, cv.COLOR\_BGR2GRAY)  *# 转为灰度图* 173. img = cv.GaussianBlur(img, (5, 5), 1) 174. my\_file.img\_save(img, '/home/ncut/图片/w', 'p') 175. retval, img = cv.threshold( 176. img, 150, 255, cv.THRESH\_BINARY\_INV + cv.THRESH\_OTSU)  *# 二值* 177. kernel = cv.getStructuringElement(cv.MORPH\_RECT, (3, 3))  *# 定义矩形结构元素* 178. opened = cv.morphologyEx(img, cv.MORPH\_OPEN, kernel, iterations=1)  *# 开运算* 179. closed = cv.morphologyEx(opened, cv.MORPH\_CLOSE, 180. kernel, iterations=1)  *# 闭运算* 181. edges = cv.Canny(img, 50, 150, apertureSize=3)  *# 边缘检测* 182. contours = cv.findContours( 183. edges, cv.RETR\_EXTERNAL, cv.CHAIN\_APPROX\_SIMPLE)[-2] 184. contours2 = \_my\_dispose.mianji\_rect(contours, max\_area, min\_area)  *# 面积筛选* 185. contours3 = \_my\_dispose.zuiwaiceng(contours2)  *# 获取最外层* 186. return img, contours3 187. def letter(cv\_img): 188. min\_area = 1500 189. max\_area = 50000 190. size = (28, 28) 191. img = get\_waikuan(cv\_img, enable\_baohan=True)  *# 获取识别板* 192. img, contours = get\_neirong(img, max\_area, min\_area)  *# 获取识别板内容* 193. if len(contours) > 4: 194. contours = contours[:4] 195. if len(contours) == 4: 196. t = [] 197. for i in range(4): 198. t.append([contours[i], cv.minAreaRect(contours[i])]) 199. t.sort(key=lambda x: x[1][0][0] + x[1][0][1])  *# 按从左到右从上到下排列* 200. if t[1][1][0][0] < t[2][1][0][0]: 201. t[1], t[2] = t[2], t[1] 202. imgs = cut(img, [i[0] for i in t], size, 30, 30) 203. return imgs 204. return None 205. def number(cv\_img): 206. min\_area = 3000 207. max\_area = 50000 208. size = (28, 28) 209. img = get\_waikuan(cv\_img, shousuo\_y=0.1)  *# 获取识别板* 210. img, contours = get\_neirong(img, max\_area, min\_area)  *# 获取识别板内容* 211. if len(contours) >= 3: 212. t = [] 213. for i in range(len(contours)): 214. t.append([contours[i], cv.minAreaRect(contours[i])]) 215. t.sort(key=lambda x: x[1][0][0])  *# 按从左到右顺序* 216. imgs = cut(img, [i[0] for i in t], size, -50, -50) 217. return imgs 218. return None 220. *#模型识别* 221. *#!/usr/bin/env python3* 222. *# coding:utf-8* 223. *# import sys* 224. import torch 225. import torch.nn as nn 226. from torch.autograd import Variable 227. import torch.utils.data as Data 228. from torch.utils.data import Dataset, DataLoader 229. import my\_file 230. class CNN(nn.Module): 231. def \_\_init\_\_(self, out\_n): 232. super(CNN, self).\_\_init\_\_() 233. self.conv1 = nn.Sequential( 234. nn.Conv2d(  *# --> (1,28,28)* 235. in\_channels=1,  *# 传入的图片是几层的，灰色为1层，RGB为三层* 236. out\_channels=16,  *# 输出的图片是几层* 237. kernel\_size=5,  *# 代表扫描的区域点为5\*5* 238. stride=1,  *# 就是每隔多少步跳一下* 239. padding=2,  *# 边框补全，其计算公式=（kernel\_size-1）/2=(5-1)/2=2* 240. ),  *# 2d代表二维卷积           --> (16,28,28)* 241. nn.ReLU(),  *# 非线性激活层* 242. nn.MaxPool2d( 243. kernel\_size=2 244. ),  *# 设定这里的扫描区域为2\*2，且取出该2\*2中的最大值          --> (16,14,14)* 245. ) 246. self.conv2 = nn.Sequential( 247. nn.Conv2d(  *# --> (16,14,14)* 248. in\_channels=16,  *# 这里的输入是上层的输出为16层* 249. out\_channels=32,  *# 在这里我们需要将其输出为32层* 250. kernel\_size=5,  *# 代表扫描的区域点为5\*5* 251. stride=1,  *# 就是每隔多少步跳一下* 252. padding=2,  *# 边框补全，其计算公式=（kernel\_size-1）/2=(5-1)/2=* 253. ),  *# --> (32,14,14)* 254. nn.ReLU(), 255. nn.MaxPool2d( 256. kernel\_size=2 257. ),  *# 设定这里的扫描区域为2\*2，且取出该2\*2中的最大值     --> (32,7,7)，这里是三维数据* 258. ) 259. self.out = nn.Linear(32 \* 7 \* 7, out\_n)  *# 注意一下这里的数据是二维的数据* 260. def forward(self, x): 261. x = self.conv1(x) 262. x = self.conv2(x)  *# （batch,32,7,7）* 263. *# 然后接下来进行一下扩展展平的操作，将三维数据转为二维的数据* 264. x = x.view(x.size(0), -1)  *# (batch ,32 \* 7 \* 7)* 265. output = self.out(x) 266. return output 267. def train\_cnn( 268. cnn, 269. train\_loader, 270. EPOCH=10, 271. LR=0.001, 272. path=my\_file.path + 'modelsave/temp.pth', 273. ): 274. *# 添加优化方法* 275. *# optimizer=torch.optim.Adam(cnn.parameters(),lr=LR)* 276. optimizer = torch.optim.SGD(cnn.parameters(), lr=LR, momentum=0.5) 277. *# 指定损失函数使用交叉信息熵* 278. loss\_fn = nn.CrossEntropyLoss() 279. for epoch in range(EPOCH): 280. print(epoch) 281. *# 加载训练数据* 282. for step, data in enumerate(train\_loader): 283. *# cnn.train()* 284. x, y = data 285. *# 分别得到训练数据的x和y的取值* 286. b\_x = Variable(x) 287. b\_y = Variable(y) 288. output = cnn(b\_x)  *# 调用模型预测* 289. loss = loss\_fn(output, b\_y)  *# 计算损失值* 290. *# print(loss)* 291. optimizer.zero\_grad()  *# 每一次循环之前，将梯度清零* 292. loss.backward()  *# 反向传播* 293. optimizer.step()  *# 梯度下降* 294. torch.save(cnn, path) 295. print(epoch + 1) 296. def test\_cnn(cnn, test): 297. out = [] 298. with torch.no\_grad(): 299. for data in test: 300. *# cnn.eval()* 301. x, y = data 302. *# 分别得到训练数据的x和y的取值* 303. *# b\_x = Variable(x)* 304. *# b\_y = Variable(y)* 305. output = cnn(x)  *# 调用模型预测* 306. value, index = torch.max(output.data, dim=1) 307. out.append([output.data, index.item()]) 308. return out 309. class TheDataSet(Dataset):  *# 自定义数据集* 310. def \_\_init\_\_(self, datax, datay): 311. self.data = [] 312. for i in range(len(datax)): 313. self.data.append( 314. [ 315. torch.as\_tensor(datax[i].reshape([-1, 28, 28])), 316. torch.as\_tensor(datay[i]), 317. ] 318. ) 319. def \_\_len\_\_(self): 320. return len(self.data) 321. def \_\_getitem\_\_(self, index): 322. return self.data[index] 323. def get\_train(cv\_img\_list, label\_list, shuffle=True): 324. """ 325. img.shape=(28,28) 326. label.shape=(out\_n) 327. """ 328. datax, datay = [], [] 329. for i in range(len(cv\_img\_list)): 330. datax.append(torch.from\_numpy(cv\_img\_list[i]) / 255) 331. datay.append(torch.Tensor(label\_list[i])) 332. d = TheDataSet(datax, datay) 333. train\_loader = DataLoader(dataset=d, batch\_size=1, shuffle=shuffle) 334. return train\_loader 335. class test\_index(object): 336. def \_\_getitem\_\_(self, key): 337. return [] 338. def \_\_iter\_\_(self): 339. return self 340. def \_\_next\_\_(self): 341. return [] 342. def get\_test(cv\_img\_list): 343. """ 344. img.shape=(28,28) 345. """ 346. data = get\_train(cv\_img\_list, test\_index(), False)        return data |