

车载软件开发基础

松灵小车任务1



**2022至2023学年第 1 学期**

|  |  |  |
| --- | --- | --- |
| 学号 | 姓名 | 团队作用 |
| E2021146 | 王定彬 | 程序实现和代码测试 |
| E2021201 | 宋智鑫 | 程序实现和环境搭建 |
| E2021198 | 陈干 | 传感器数据获取与测试 |
| E2021206 | 熊健霖 | 图像处理和模型验证 |
| 任课教师 | 刘骥 | |
| 成 绩 |  | |

|  |  |
| --- | --- |
| 任务书 | |
| 任务内容 | 实验分组进行，每组人数不超过4人。在松灵小车上编写ROS程序，实现如下功能：  （1）开启松灵小车的激光雷达、深度相机等传感器，控制小车移动（遥控器），编写程序，显示移动过程中，各类传感器数据：   * 命令行窗口显示小车的线速度和角速度 * OpenCV显示深度相机的RGB图像和深度图 * PCL显示激光雷达的点云数据   （2）编程实现用里程计（odometry）计算小车移动距离   * 小车靜止不动，读取里程计数据，记为a控制小车前进n米距离(n=1、2、3均可），读取里程计数据，记为b * 建立小车移动距离与里程计读数a、b之间的关系模型（ 前两步应进行多次以拟合更精确的模型） * 控制小车移动，利用上一步构建的模型，计算小车移动的距离，并将计算结果与实际值进行对比 |
| 程序规范 | （1）所有程序代码采用C++编写，使用git进行源代码管理；  （2）类名、变量名、函数名应符合C++的命名规范，并在代码中前后保持一致；  （3）涉及面向对象的程序，例如自定义的类，应符合面向对象的设计原则；  （4）正确使用头文件和源文件，自定义的头文件应符合头文件的编写原则，例如用条件宏定义确保头文件不被多次引用、不在头文件中进行类和函数的实现（模板除外）；  （5）项目必须是ROS项目，符合ROS的项目的规范，正确编写CmakeLists.txt等文件；  （6）程序能够在松灵小车上运行。 |
| 报告要求 | （1）报告至少应该包括人员分工、需求分析、程序设计、程序效果展示、总结分析4个部分；  （2）人员分工介绍组员各自的工作情况；  （3）需求分析侧重描述程序所需要实现的功能，功能预期的效果；  （4）程序设计描述组成程序的模块、类、函数以及他们之间的相互关系，若有算法，可以描述算法流程；  （5）程序效果展示除了程序运行效果截图之外，应该有必要的文字说明；  （6）总结分析可以分析实现的效果与理想情况的差异，分析导致这些差异的原因，切忌不要写成心得体会；  （7）报告应该格式规范、排版整洁、少语病和错误。 |
| 作业提交 | （1）含有git仓库（有.git目录）的完整源代码；  （2）程序功能演示的讲解视频；  （3）任务报告。 |
| 评分标准 | 按照五级制打分，分为优秀、良好、中等、及格、不及格，各评分项占总成绩的比例为：  （1）任务完成情况占评分的60%；  （2）程序规范占评分的20%；  （3）报告占评分的20%。  评分老师根据各部分的完成情况，直接给出总成绩。 |

报告的主要内容：

1. **人员分工 （介绍组员各自的工作情况）**

**传感器数据采集与显示**

* **负责人**：宋智鑫 ，陈干
* **分工描述**：
  1. **传感器初始化**：负责开启松灵小车上的激光雷达、深度相机等传感器，并确保所有传感器正常工作。
  2. **数据采集**：编写ROS程序来采集小车移动过程中的传感器数据。
     + 在命令行窗口显示小车的线速度和角速度。
     + 使用OpenCV显示来自深度相机的RGB图像和深度图。
     + 利用PCL显示激光雷达的点云数据。
  3. **小车控制**：编写程序，使用遥控器控制小车的移动，并确保所有采集的数据能够正确显示出来。

**里程计计算与模型建立**

* **负责人**：王定彬 ，熊建霖
* **分工描述**：
  1. **里程计数据读取**：编写程序，读取小车静止和移动时的里程计数据。
  2. **数据记录与模型建立**：
     + 记录小车在不同距离（例如1米、2米、3米）下静止和移动时的里程计读数a和b。
     + 多次重复实验，以获得足够的数据来拟合小车移动距离与里程计读数a、b之间的关系模型。
  3. **模型验证**：
     + 控制小车移动，利用建立的关系模型计算小车移动的距离。
     + 将计算出的小车移动距离与实际测量值进行比较，验证模型的准确性。

**综合测试与调整（4人）**

完成前两个阶段后，两组成员共同进行综合测试，确保所有功能正常工作，对照实验结果，调整程序和模型，直至满足实验要求。

1. **需求分析（侧重描述程序所需要实现的功能，功能预期的效果）**

为了确保松灵小车的各项功能得以实现，并满足项目要求，以下是针对四个主要任务的需求分析汇总：

**任务1：命令行窗口显示小车的线速度和角速度**

**需求描述**：

* 利用ROS系统获取小车的线速度和角速度数据。
* 在命令行窗口实时显示这些数据。

**功能预期**：

* 订阅ROS中的速度话题，获取小车的线速度和角速度信息。
* 开发一个ROS节点程序，将获取的数据实时打印在命令行窗口。
* 显示的数据应包含线速度的大小和方向，以及角速度的大小和旋转轴。
* 系统应能处理数据更新，确保命令行窗口中的数据实时刷新且准确无误。

**任务2：OpenCV显示深度相机的RGB图像和深度图**

**需求描述**：

* 接收并处理来自深度相机的RGB图像和深度图数据。
* 使用OpenCV库在图形用户界面中显示这些图像。

**功能预期**：

* 订阅ROS中深度相机的相关话题，以获取图像数据。
* 将ROS图像消息转换为OpenCV图像格式。
* 创建一个GUI窗口，分别显示RGB图像和深度图。
* 界面应能实时更新图像数据，以反映小车周围环境的变化。

**任务3：PCL显示激光雷达的点云数据**

**需求描述**：

* 接收激光雷达的点云数据，并在三维空间中进行可视化。

**功能预期**：

* 订阅激光雷达的话题，获取点云数据。
* 将ROS的点云数据格式转换为PCL库可以处理的格式。
* 使用PCL库的可视化工具，在新的窗口中展示点云数据。
* 点云数据应能准确反映小车周围环境的三维结构，并且实时更新。

**任务4：编程实现用里程计（odometry）计算小车移动距离**

**需求描述**：

* 利用里程计数据计算小车的实际移动距离。
* 在命令行窗口或图形用户界面中显示计算出的距离。

**功能预期**：

* 订阅/odom话题，获取小车的里程计数据，包括x和y坐标。
* 根据小车的初始位置和当前位置，计算小车的移动距离。可以使用欧几里得距离公式来计算两点之间的距离。
* 将计算出的距离实时显示在命令行窗口或GUI中。
* 系统应能处理数据的连续更新，并准确计算出小车的移动距离。

1. **程序设计（组成程序的模块、类、函数和关键算法）**

### 3.1 任务1

实时获取并显示松灵小车的线速度和角速度数据。

#include <ros/ros.h>

#include <nav\_msgs/Odometry.h>

/\*

   使用

   ```bash

   roslaunch scout\_bringup open\_rslidar.launch

   ```

   启动激光雷达，就可以获取/odom消息

   \*/

void speedCallback(const nav\_msgs::Odometry::ConstPtr &msg)

{

    //读取激光雷达相关的话题/odom，可以获取小车的线速度和角速度

    ROS\_INFO("Car linear velocity: vx=%0.6f,vy=%0.6f,vz=%0.6f", msg->twist.twist.linear.x, msg->twist.twist.linear.y, msg->twist.twist.linear.z);

    ROS\_INFO("Car angular velocity: wx=%0.6f,wy=%0.6f,wz=%0.6f", msg->twist.twist.angular.x, msg->twist.twist.angular.y, msg->twist.twist.angular.z);

    return;

}

int main(int argc, char \*\*argv)

{

    ros::init(argc, argv, "speed");

    ros::NodeHandle n;

    ros::Subscriber speed\_sub = n.subscribe("/odom", 10, speedCallback);

    ros::spin();

    return 0;

}

* 1. 任务2**用OpenCV显示相机的彩色和深度图**

//用OpenCV显示相机的彩色和深度图(/camera/color/image\_raw、/camera/depth/image\_rect\_raw)

/\*

开启相机

roslaunch realsense2\_camera rs\_camera.launch

source devel/setup.bash

rosrun opencv opencv

\*/

#include <ros/ros.h>

#include <image\_transport/image\_transport.h>

#include <cv\_bridge/cv\_bridge.h>

#include <sensor\_msgs/image\_encodings.h>

#include <opencv2/imgproc/imgproc.hpp>

#include <opencv2/highgui/highgui.hpp>

static const std::string OPENCV\_WINDOW\_COLOR = "Image color window";

static const std::string OPENCV\_WINDOW\_DEPTH = "Image depth window";

class ImageConverter

{

  ros::NodeHandle nh\_;                         // ROS节点句柄

  image\_transport::ImageTransport it\_;         // 图像传输对象

  image\_transport::Subscriber image\_sub\_color; // 彩色图像订阅者

  image\_transport::Subscriber image\_sub\_depth; // 深度图像订阅者

  image\_transport::Publisher image\_pub\_color;  // 彩色图像发布者

  image\_transport::Publisher image\_pub\_depth;  // 深度图像发布者

public:

  ImageConverter()

      : it\_(nh\_) //初始化图像传输对象

  {

    //订阅输入视频流并发布输出视频流

    image\_sub\_color = it\_.subscribe("/camera/color/image\_raw", 1, &ImageConverter::imageCb, this);

    image\_sub\_depth = it\_.subscribe("/camera/depth/image\_rect\_raw", 1, &ImageConverter::depthCb, this);

    image\_pub\_color = it\_.advertise("/camera/color/camera\_info", 1);

    image\_pub\_depth = it\_.advertise("/camera/depth/camera\_info", 1);

    //创建opencv窗口

    cv::namedWindow(OPENCV\_WINDOW\_COLOR);

    cv::namedWindow(OPENCV\_WINDOW\_DEPTH);

  }

  ~ImageConverter()

  {

    // 销毁OpenCV窗口

    cv::destroyWindow(OPENCV\_WINDOW\_COLOR);

    cv::destroyWindow(OPENCV\_WINDOW\_DEPTH);

  }

  void imageCb(const sensor\_msgs::ImageConstPtr &msg)

  {

    cv\_bridge::CvImagePtr cv\_ptr; // 创建CvImage指针

    try

    {

      // 将ROS图像消息转换为OpenCV图像格式

      cv\_ptr = cv\_bridge::toCvCopy(msg, sensor\_msgs::image\_encodings::BGR8);

    }

    catch (cv\_bridge::Exception &e)

    {

      ROS\_ERROR("cv\_bridge exception: %s", e.what());

      return;

    }

    // 在视频流上绘制一个示例圆圈

    if (cv\_ptr->image.rows > 60 && cv\_ptr->image.cols > 60)

      cv::circle(cv\_ptr->image, cv::Point(50, 50), 10, CV\_RGB(255, 0, 0));

    // 更新GUI窗口

    cv::imshow(OPENCV\_WINDOW\_COLOR, cv\_ptr->image);

    cv::waitKey(3); // 等待3毫秒以处理窗口事件

    // 发布修改后的视频流

    image\_pub\_color.publish(cv\_ptr->toImageMsg());

  }

  void depthCb(const sensor\_msgs::ImageConstPtr &msg)

  {

    cv\_bridge::CvImagePtr cv\_ptr; // 创建CvImage指针

    try

    {

      // 将ROS深度图像消息转换为OpenCV图像格式

      cv\_ptr = cv\_bridge::toCvCopy(msg, sensor\_msgs::image\_encodings::TYPE\_32FC1);

    }

    catch (cv\_bridge::Exception &e)

    {

      ROS\_ERROR("cv\_bridge异常: %s", e.what()); // 报错信息

      return;

    }

    // 在深度图像上绘制一个示例圆圈

    if (cv\_ptr->image.rows > 60 && cv\_ptr->image.cols > 60)

      cv::circle(cv\_ptr->image, cv::Point(50, 50), 10, CV\_RGB(255, 0, 0));

    // 更新GUI窗口

    cv::imshow(OPENCV\_WINDOW\_DEPTH, cv\_ptr->image);

    cv::waitKey(3); // 等待3毫秒以处理窗口事件

    // 发布修改后的深度图像

    image\_pub\_depth.publish(cv\_ptr->toImageMsg());

  }

};

int main(int argc, char \*\*argv)

{

  // 初始化ROS节点

  ros::init(argc, argv, "opencv");

  ImageConverter ic; // 创建ImageConverter对象

  ros::spin();       // 进入事件循环

  return 0;

}

* **3.3 任务3 PCL显示激光雷达的点云数据**

// PCL显示激光雷达的点云数据（/velodyne\_points）

/\*

先开启激光雷达数据  roslaunch scout\_bringup open\_rslidar.launch

配置环境变量 source devel/setup.bash

运行节点 rosrun pcl pcl

\*/

#include <iostream>

#include <ros/ros.h>

#include <sensor\_msgs/PointCloud2.h>

#include <string.h>

#include <pcl/io/pcd\_io.h>  // PCL库用于输入输出PCD文件

#include <pcl/io/io.h>      // PCL库基本输入输出功能

#include <pcl/point\_types.h> // 定义点云的数据类型

#include <pcl/point\_cloud.h> // 定义点云结构

#include <pcl/visualization/pcl\_visualizer.h> // PCL可视化工具

// #include <pcl/point\_types.h>

// #include <pcl/filters/statistical\_outlier\_removal.h>   //统计滤波

// #include <pcl/filters/random\_sample.h>    //随机采样

// #include <pcl/filters/voxel\_grid.h>      //体素滤波

// #include <pcl/filters/passthrough.h>     //直通滤波

#include <boost/thread/thread.hpp>// PCL和ROS消息转换

#include <pcl\_conversions/pcl\_conversions.h>

using namespace std;

// 创建PCL可视化对象

boost::shared\_ptr<pcl::visualization::PCLVisualizer> viewer1(new pcl::visualization::PCLVisualizer("实时点云可视化"));

ros::Publisher pub; // 声明ROS发布者

// 定义点云类型

typedef pcl::PointCloud<pcl::PointXYZ> PointCloud;

// 点云回调函数

void cloud\_cb (const sensor\_msgs::PointCloud2ConstPtr& input)

{

   // 声明存储原始数据与滤波后的数据的点云的格式

   pcl::PCLPointCloud2\* cloud = new pcl::PCLPointCloud2; // 原始的点云数据格式

   pcl::PCLPointCloud2ConstPtr cloudPtr(cloud);

   // 将ROS的点云消息转换为PCL格式

   pcl\_conversions::toPCL(\*input, \*cloud);

   pub.publish(\*cloud); // 发布点云数据

   // 创建点云对象并将ROS消息转换为PCL点云格式

   pcl::PointCloud<pcl::PointXYZ>::Ptr cloud1;

   cloud1.reset(new pcl::PointCloud<pcl::PointXYZ>);

   pcl::fromROSMsg(\*input, \*cloud1);

   // 移除当前所有点云

   viewer1->removeAllPointClouds();

   // 添加新的点云数据到可视化窗口

   viewer1->addPointCloud(cloud1, "实时点云");

   viewer1->updatePointCloud(cloud1, "实时点云"); // 更新可视化窗口

   viewer1->spinOnce(0.001); // 处理可视化窗口事件

}

int main (int argc, char\*\* argv)

{

  // 初始化ROS节点

  ros::init(argc, argv, "pcl");

  ros::NodeHandle nh;

  // 创建一个ROS订阅者，用于订阅点云数据

  ros::Subscriber sub = nh.subscribe("/rslidar\_points", 10, cloud\_cb);

  // 创建一个ROS发布者，用于发布点云数据

  pub = nh.advertise<pcl::PCLPointCloud2>("output", 10);

  // 进入ROS事件循环

  ros::spin();

}

* 1. **任务4里程计**

/\*

编程实现用里程计（odometry）计算小车移动距离

首先开启激光雷达

roslaunch scout\_bringup open\_rslidar.launch

然后新开一个窗口运行odometry.cpp文件

rosrun odometry odometry

\*/

#include <ros/ros.h>

#include <nav\_msgs/Odometry.h>

#include <iostream>

using namespace std;

double t\_recent;

double s;

void distanceCallback(const nav\_msgs::Odometry::ConstPtr &msg)

{

    double dt;

    double ds;

    double dv = 0; // 定义一个变量用于存储X轴速度

    // 读取激光雷达相关的话题/odom，可以获取小车速度和采集时间

    dt = msg->header.stamp.toSec() - t\_recent;

    t\_recent = msg->header.stamp.toSec();

    // 从小车的里程计数据中获取X轴速度

    dv = msg->twist.twist.linear.x;

    // 如果X轴速度的绝对值小于0.05，则将其设置为0，以减少误差

    if (fabs(dv) < 0.05)

    {

        dv = 0;

    }

    if (dt > 1 || dt < 0)

    { // 消息间隔太远则说明该时间差无效，这里直接舍去

        dt = 0;

    }

    ds = dv \* dt \* 1.0504; // 根据建立的线性模型进行校正

    s += ds;

    //  在ROS的日志中打印X轴速度

    ROS\_INFO("X\_速度：%.3f m/s ", dv);

    //  在ROS的日志中打印累计的移动距离

    ROS\_INFO("里程计的距离为：%.6f m", s);

    // ROS\_INFO("v = %.3f  m/s  ,   s = %.3f   m", dv, s);

    return;

}

int main(int argc, char \*\*argv)

{

    // 设置本地化环境,可以打印汉字

    setlocale(LC\_ALL,"");

    ros::init(argc, argv, "odometry");

    ros::NodeHandle n;

    s = 0;

    t\_recent = ros::Time::now().toSec();

    ROS\_INFO("init s=%f,t\_recent=%0.6fs", s, t\_recent);

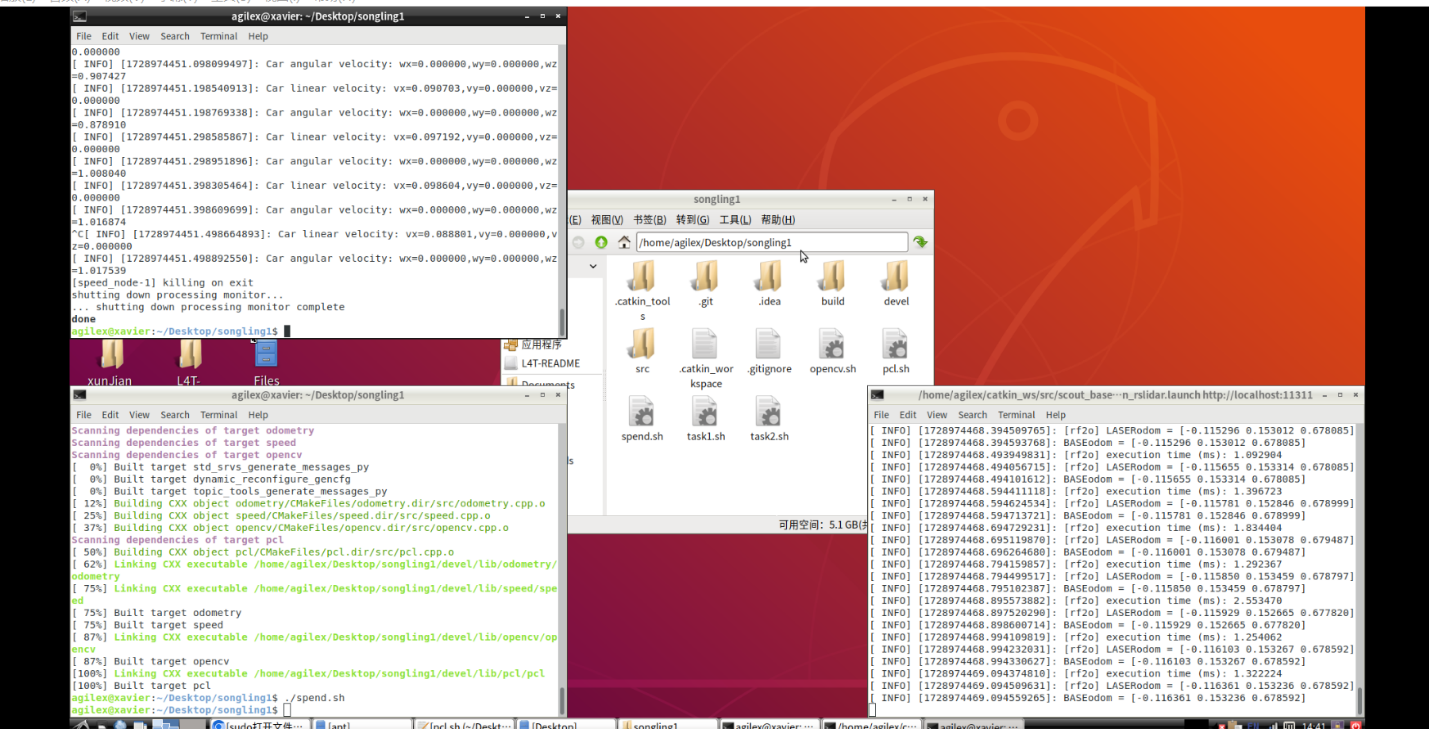
    ros::Subscriber speed\_sub = n.subscribe("/odom", 10, distanceCallback);

    ros::spin();

    return 0;

}

1. **程序效果展示**

****

**图1 线，角速度**

**图片包含 文本

描述已自动生成**

**图2深度相机**

**图形用户界面, 应用程序

描述已自动生成**

**图3激光雷达**

**图形用户界面, 应用程序

描述已自动生成**

**图4里程计**

1. **总结和分析**

本次基于松灵小车的ROS项目中，我们小组通过分工合作，成功实现了多种传感器数据的采集与显示，并且初步建立了通过里程计估算小车移动距离的模型。在这个过程中，我们遇到了一些挑战，也收获了许多宝贵的经验。

首先，在项目初期，环境搭建成了我们的第一个拦路虎。由于ROS对环境配置的要求较高，每位组员都需要在自己的机器上搭建ROS环境，这不仅考验了我们的耐心，也锻炼了我们在遇到问题时查找资料的能力。通过阅读官方文档和社区讨论，我们逐步解决了依赖项安装、编译配置等问题。

其次，在编写代码的过程中，我们深刻体会到了遵循编码规范的重要性。由于项目涉及到多个模块间的交互，一开始没有严格遵守C++命名规范导致了一些不必要的混淆。后来，我们统一了类名、变量名、函数名等的命名方式，并且加强了代码注释，这使得代码更加易于理解和维护。

此外，使用Git进行版本控制也是这次项目中的一个重要环节。通过提交和合并代码，我们学会了如何有效地协作开发，同时也能追溯到每个功能点的实现历史，这对于后期的调试和功能迭代非常有帮助。

在传感器数据处理方面，尤其是激光雷达和深度相机的数据流处理，我们学习到了很多图像处理的知识。OpenCV和PCL库的强大功能让我们能够快速地将原始数据转化为直观的图像和点云信息，这对后续的路径规划和避障算法开发打下了坚实的基础。

最后，在里程计数据的采集和建模过程中，我们意识到理论与实践之间存在的差距。尽管我们构建了初步的移动距离估算模型，但在实际测试中发现由于地面摩擦力等外部因素的影响，模型预测与实际情况存在一定的误差。这也提醒我们在未来的研究中需要更多地考虑实际应用中的各种不确定性和干扰因素。

总的来说，这次项目经历不仅让我们学到了许多技术知识，还提高了团队协作能力和解决问题的能力。尽管过程中遇到了不少困难，但我们相信这些经验将会对我们今后的学习和工作产生积极的影响。

1. **参考文献**

[1]pcl库的学习和使用

pclhttps://blog.csdn.net/weixin\_44411318/article/details/127587550?utm\_medium=distribute.pc\_relevant.none-task-blog-2~default~baidujs\_utm\_term~default-0-127587550-blog-136140300.235^v43^pc\_blog\_bottom\_relevance\_base7&spm=1001.2101.3001.4242.1&utm\_relevant\_index=3,2020-11-21.

[2]openCV使用USB摄像头并发布topic.https://blog.csdn.net/Leslie\_92672,2022-03-01.

[3]PCL库学习笔记.http://t.csdn.cn/gRBO2,2021-06-21.

[4]使用ROS\_BRIDGE获取深度数据.https://www.freesion.com/article/5782869347/,2019-05-18.

[5]PCL点云可视化.https://cdn.modb.pro/db/488053,2022-08-17.

[6]在ROS中使用OpenCV进行简单的图像处理---代码实现篇.http://t.csdn.cn/9wmfg,2022-01-20.

[7][ROS机器人应用查看里程计、IMU 话题信息](http://www.wtld.cn/a/75093.html).http://www.wtld.cn/a/75093.html,2020-07-11.

1. **完成效果视频：**

https://www.bilibili.com/video/BV1omSgYsELm/?share\_source=copy\_web&vd\_source=09354cc6a7a4648b50bc2816b3712cae

源码：

