**第十四届“博创杯”全国大学生嵌入式设计大赛**

**作品设计报告**

**智能隧管检测系统**

**Intelligent tunnel detection system**

**设**

**计**

**报**

**告**

队伍编号：

参赛学校：

作 者：

指导教师：

组 别：■本科组 □高职组

目录

**键入章标题(第 1 级)1**

键入章标题(第 2 级)2

键入章标题(第 3 级)3

**键入章标题(第 1 级)4**

键入章标题(第 2 级)5

键入章标题(第 3 级)6

**未找到目录项。**

**摘 要**

目前的交通道路有朝着地下化、隧道化方向发展的趋势，地铁隧道也将成为未来城市交通的主要发展方向。同时各类城市管道的长度也在逐年上升。对于隧道、管道等类似的封闭环境，现有的缺陷检测方法成本高昂，形式单一，有着工序繁琐、耗时极长等弊端，不利于城市交通与管线系统的可持续发展。

　　因此，本项目设计了一套以图像识别与超声波测距为主要传感器，基于STM32与树莓派的小型智能化隧管缺陷检测系统，并通过ROS实现了设备与上位机的通信。我们在把对管道、隧道的缺陷检测统一化的同时，设计了一套离线作业系统，并力求在满足精确、高效的同时，将成本控制在合理的范围之内，进而使得其能够被大规模投放使用，因而具有很强的实用性与可推广性。

同时，本项目也提出了一种基于超声波传感器与摄像头相配合的小直径隧管彩色三维重构方法，并在进行数学推导与可行性论证之后，使用程序模拟实现了三维重构的效果。

关键词：智能交通、图像识别、ROS、超声波测距、三维重构

**Abstract**

Nowadays, the traffics system are moving toward undergroundization and tunnelling. Subways are becoming the main stream of the development of future urban transportation. At the same time, the length of various types of urban pipelines has also increased year by year. For tunnels, pipelines and similar closed environments, the existing defect detection methods are expensive, not various, and have drawbacks such as cumbersome procedures, extremely long time-consuming, and are not conducive to the sustainable development of urban traffic and pipeline systems.

Therefore, this project designed a set of small sensor tunnel detection system based on STM32 and Raspberry Pi with image recognition and ultrasonic ranging as the main sensors, and achieved communication between the device and host computer through ROS. While we unify the detection of defects in pipelines and tunnels, we designed an online operation system, and strived to control the cost within a reasonable range while satisfying the accuracy and efficiency, and in turn making it possible to be put into large-scales, so it has a strong practicality and scalability.

At the same time, a colorful 3D reconstruction method for small-diameter tunnel tubes based on ultrasonic sensors and cameras is also proposed in this project. After mathematical derivation and feasibility demonstration, the effects of 3D reconstruction are achieved using program simulation.

Key words: Intelligent traffic, image recognition, ROS, ultrasonic ranging, 3D reconstruction

**第1章 绪 论**

## 1.1 视觉识别技术

视觉检测系统是用相机代替人眼睛去完成识别.测量.定位等功能。一般视觉检测系统由相机、镜头、光源组合合成，可以代替人工完成[条码字符](https://baike.baidu.com/item/%E6%9D%A1%E7%A0%81%E5%AD%97%E7%AC%A6)、裂痕、包装、表面图层是否完整、凹陷等检测，使用视觉检测系统能有效的提高生产流水线的检测速度和精度，大大提高产量和质量，降低人工成本，同时防止因为人眼疲劳而产生的误判。

## 1.2 三维重构技术

三维重建，指的是通过摄像机获取场景物体的数据图像，并对此图像进行分析处理，再结合计算机视觉知识推导出现实环境中物体的三维信息。

根据重建的稀疏程度不同，可以分为以下以下几类：

稀疏重建：

通常是重建一些图像特征点的深度，这个在基于特征的视觉SLAM里比较常见，得到的特征点的深度可以用来计算相机位姿。稀疏重建在实际应用，比如检测，避障，不能满足需求。

半稠密重建：

通常是重建图像纹理或梯度比较明显的区域，这些区域特征比较鲜明。半稠密重建在直接法视觉SLAM里比较常见。重建的三维点云相对稠密，可以满足部分应用需求。

稠密重建：

稠密重建是对整个图像或者图像中的绝大部分像素进行重建。与稀疏、半稠密相比，稠密重建对场景的三维信息理解更全面，更能符合应用需求。但是，由于要重建的点云数量太多，相对耗时。本项目所设计的三维重构方法属于半稠密重建。根据重建的实时性不同，可以分为离线重建和在线重建。

离线重建：

比如 SFM(运动恢复结构)技术，此技术根据在一段时间内获得的连续图像来重建一个三维环境。

在线重建：在线重建可以分为渐进式重建和直接式重建。在线重建或多或少都和VO或者(SLAM)有联系，因为重建的时候需要相机的位姿。我们所设计的是离线重建

## 1.3 ROS机器人系统

ROS---(Robot Operating System）是一个机器人软件平台，它能为异质计算机集群提供类似操作系统的功能。ROS提供一些标准操作系统服务，例如硬件抽象，底层设备控制，常用功能实现，进程间消息以及数据包管理。ROS是基于一种图状架构，从而不同节点的进程能接受，发布，聚合各种信息（例如传感，控制，状态，规划等等）。ROS可以分成两层，低层是上面描述的操作系统层，高层则是广大用户群贡献的实现不同功能的各种软件包，例如定位绘图，行动规划，感知，模拟等等。ROS的核心是节点。节点是一小段用Python或C++写成的程序，用来执行某个相对简单的任务或进程。多个节点之间互相传递信息，并可以独立控制启动或终止。某一节点可以面向其它节点针对特定标题发布信息或提供服务。

## 1.4 隧道检测现状

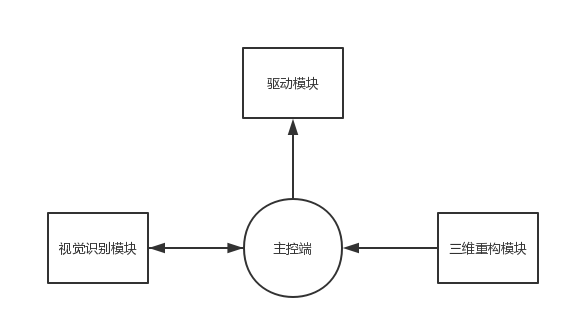
随着国家经济的不断发展，为了满足日益增长的通行和城市发展需求，各类管网在近十年不断增长。据统计，截止到2017年，仅国内地铁管网长度就已经达到了世界前列，上海与北京更分列第一第二。如此大规模的交通管网带来的不仅只有方便，也带来了大量的附加问题。其中，管道维护是目前很多部门关注的重点。以地铁管道为例，现有的管道检测技术无法准确检测出管道裂缝，更无法判断出管道损坏的位置的影响程度，目前大多使用人工肉眼检测，且管道条件复杂，人工检测危险重重，国内外多次出现管道工人意外身亡事故。

**第2章 系统方案**

## 2.1 系统总体设计

小车端与PC主控端通过蓝牙和WIFI进行连接，开发板与PC主控端通过USB有线连接，传感器与驱动部分用串口进行连接。最终通过ROS控制各模块并且进行数据传输，从而实现预期功能。

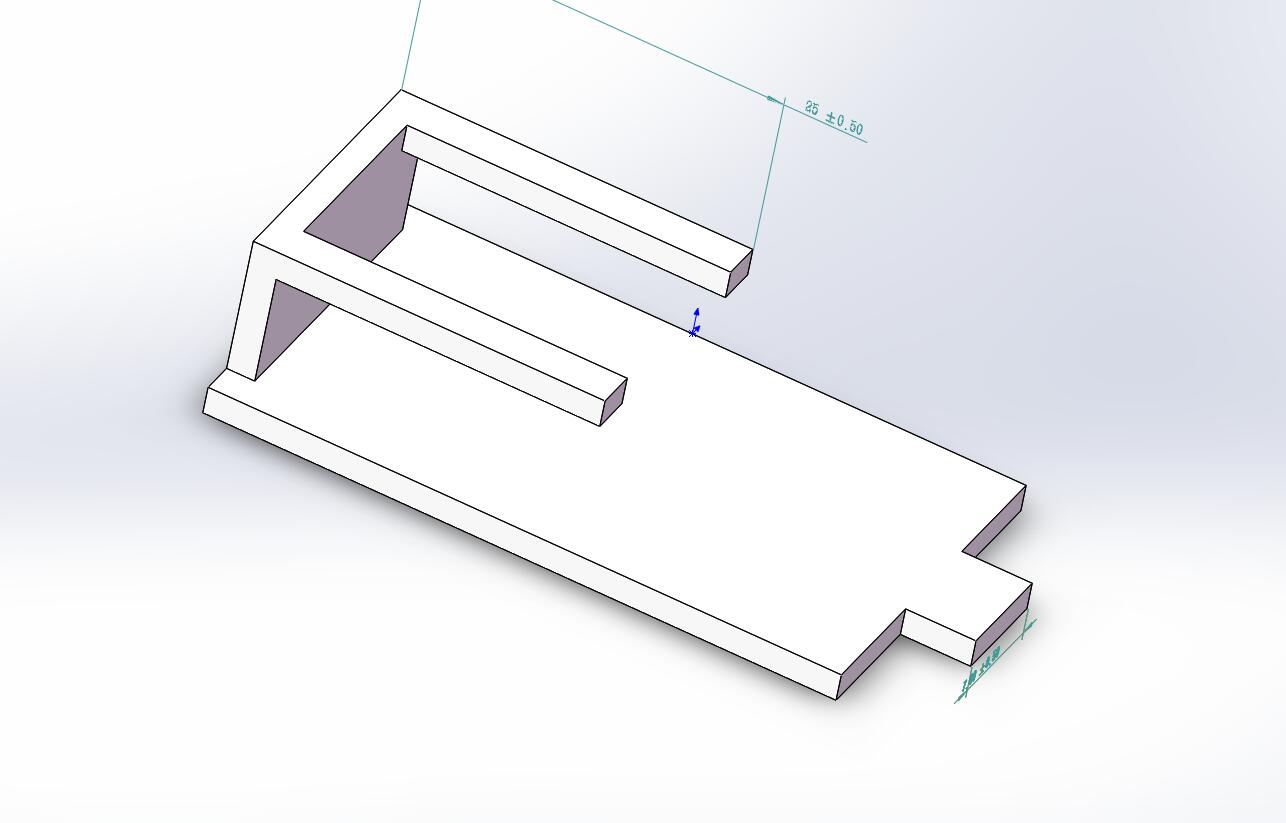
## 2.2 系统工作流程



**第3章 功能与指标**

## 3.1 硬件模块

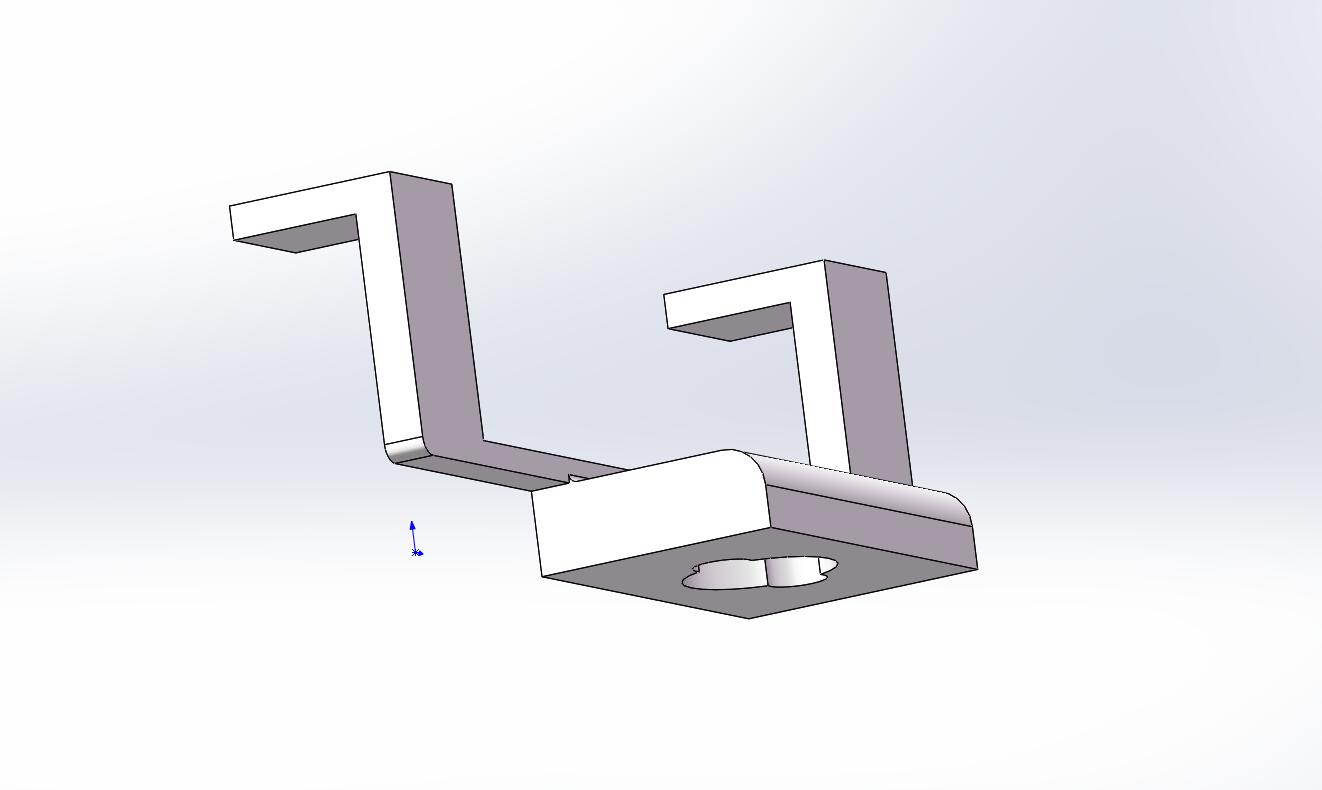
3.1.1 机械结构  
1. 相机底座  
 相机底座采用了夹持的设计思想，通过伸出的两个臂状长方体将相机紧紧地夹在底板上，使得镜头能够在整个平面内360度旋转；同时这两个“手臂”不会影响到边上的光敏电阻与红外补光灯，使得整个设备在阴暗的环境下也能正常、平稳地工作。



相机底座

2. 前部支架

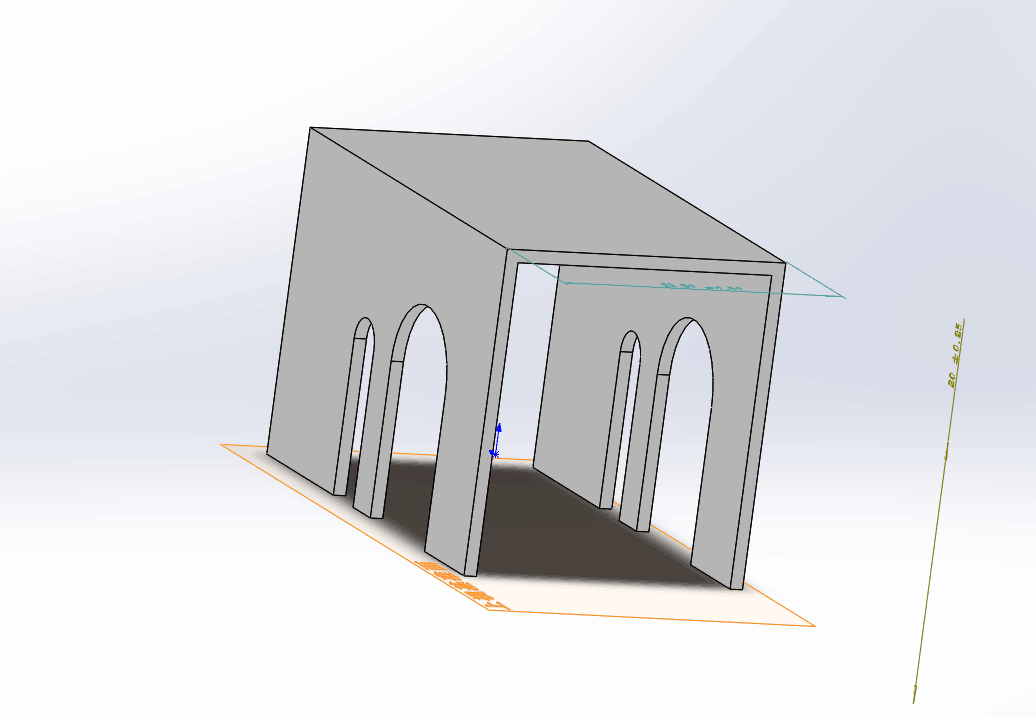
前部支架为支撑车辆前部的舵机与摄像头而设计，其符合力学原理的结构与定制的万向轮插槽使得整个车辆能自由、灵活地进行转向，并能够防止车体质量的不平衡所导致的倾斜与晃动，使得车辆能够在一般环境下平稳地前进。



前置支架

3. 驱动电机支架

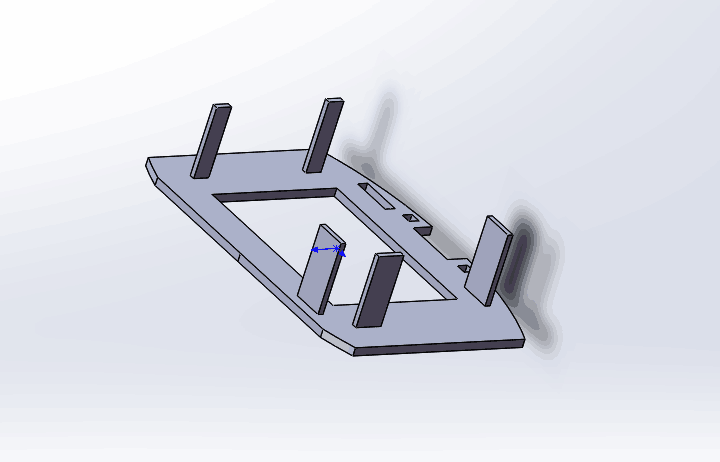
驱动电机支架专为直流减速电机而设计，其能够良好地对电机进行保护，并将电机的核心部件与外部环境隔离开来，有效防止电机过载等意外情况的发生。



电机支架

4. 主控板

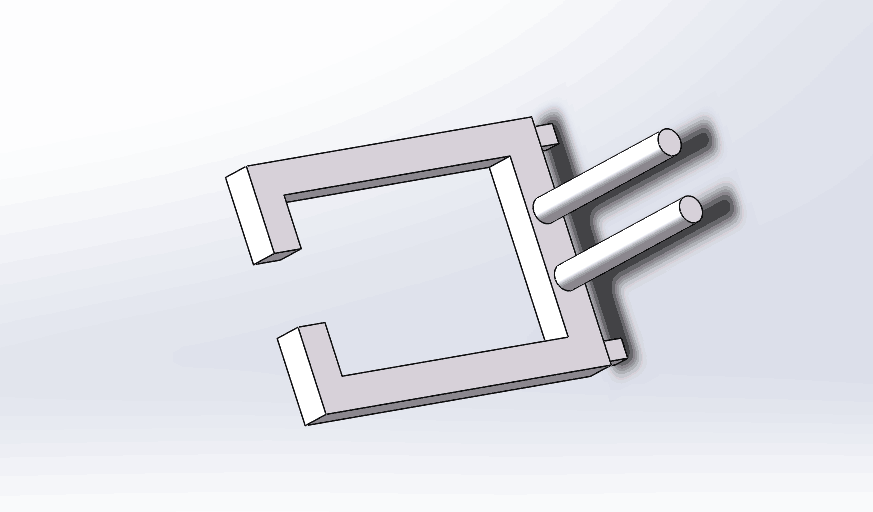
主控板是整个设备设计的核心，在它之下，连接了驱动电机支架与控制底座，在它中央的凹槽则是用于放置5V树莓派专用供电电池，而用保护壳进行保护的树莓派运算板则直接被放置在供电电池之上，被五个设计精巧的挡板紧紧地扣住；而在其前方，则是连接了舵机支架，进而与摄像头设备相连，使得整个系统在保证功能性的基础上将结构优化到了极致。



主控板

5. 舵机支架

舵机支架采用简单的支撑型设计，在其底部的两个圆柱向前伸出，平稳地拖住了质量较大的电机，而在支架的正下方的两个正方形凸口，则转为与主控板进行连接而设计，其能够牢牢地嵌入主控板之中，进而使得舵机能进行良好、可靠的运作。



舵机支架

3.1.2 视觉识别硬件结构

1. Rpi Camera

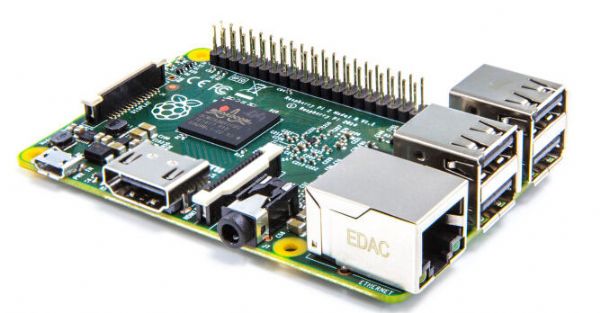
我们采用带自感应式红外补光的RPiCamera摄像头。当环境光较弱时，其两侧的红外补光灯会自动开启，使得机器人能在各种不同光照环境下工作。RpiCamera的数据流可直接通过树莓派发送到上位机，从而实现实时检测的功能。



Rpi Camera

2. Raspberry

我们采用RaspberryPi3B来对摄像头获得的图形进行预处理与储存。RaspberryPi3B是基于ARM内核的性能强大的开源硬件，我们在其上安装了最新的轻量级操作系统Raspbian，使得我们能够非常容易地与其交互。



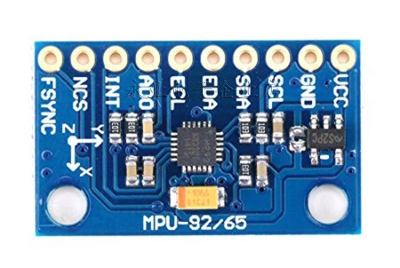
Raspberry Pi 3B

3.1.3. 三维重构硬件结构

我们设计的三维重构系统需要一个类似于激光雷达的传感器。我们的方案是将超声波与姿态传感器被捆绑在一起，挂载到可以三百六十度旋转的舵机上。同时，我们使用低功耗蓝牙模块来实现短距离的数据传输，使树莓派能够收到来自超声波传感器的数据。

1. 九轴姿态传感器

我们准备使用MPU9250模块来获取超声波雷达的位姿。MPU9250能够实时测量其三轴上的加速度、角加速度与地磁角，从而精确地估计出雷达的姿态角。



MPU9250

2. 超声波模块

为了提升距离的准确性，我们选用了US-100超声波传感器。US-100有着方向角较小、精度较大的特点，其探测范围可达2cm-450cm，而误差仅有0.3cm±1%，因而可以较好地检测出设备到管道内壁的距离。

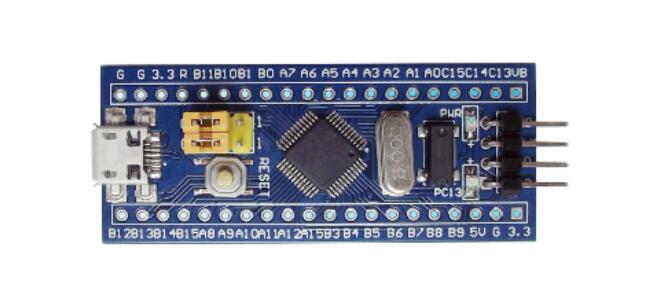


US-100

3.2.4. 小车运动模块硬件结构

1. STMF103C8T6

我们采用STM32F103C8T6最小系统板。STM32是基于ARM® Cortex® M 处理器内核的 32位闪存微控制器，集高性能、高实时性、数字信号处理、低功耗于一身，基于行业标准，具有大量工具和软件可供选择，是本项目控制部件的的理想选择。



STM32F103C8T6

2. 减速电机及橡胶轮

我们采用了通用减速直流电机与60mm直径的橡胶轮，其具有强磁、带抗干扰等特性。我们会为其提供5V直流供电，使得机器人能在较平缓的隧道内稳步前行。



直流减速电机与轮胎

## 3.2 软件模块

3.2.1. 视觉识别算法

我们采用了基于视觉识别技术的缺陷检测算法，其原理是不断去除图像上的噪声，仅仅留下我们需要的特征也即裂缝等缺陷的轮廓。

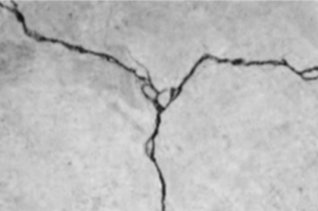
1. 噪点消除

　由于管道内复杂的环境与摄像头本身的物理特性，我们得到的初始图像数据往往带有较多噪点，这些噪点会影响成像的质量，进而干扰我们对缺陷特征的判断。因此，获取图像后我们首先要做的是使用滤波算法对图像去噪。

常用的滤波算法有均值滤波、高斯滤波、中值滤波等。经过我们多次尝试，在模拟的隧管环境下高斯滤波的去噪效果最好。除此之外它也有着较好的平滑化效果。

高斯滤波是一种线性平滑滤波，适用于消除高斯噪声，广泛应用于图像处理的减噪过程。通俗的说，高斯滤波就是对整幅图像进行[加权平均](https://baike.baidu.com/item/%E5%8A%A0%E6%9D%83%E5%B9%B3%E5%9D%87)的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到。高斯滤波的具体操作是：用一个模板（或称卷积、掩模）扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。

高斯滤波一般有两种实现方式，一是用离散化窗口滑窗卷积，另一种通过[傅里叶变换](https://baike.baidu.com/item/%E5%82%85%E9%87%8C%E5%8F%B6%E5%8F%98%E6%8D%A2)。而只有当离散化的窗口非常大，用滑窗计算量非常大（即使用可分离滤波器的实现）的情况下，可能会考虑基于傅里叶变化的实现方法。故在这里我们使用离散化窗口滑窗卷积。

高斯滤波效果演示一

高斯滤波效果演示二

2. 缺陷检测：边缘检测

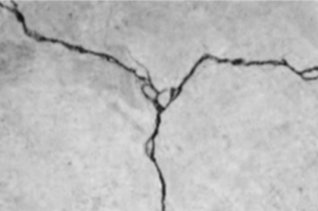
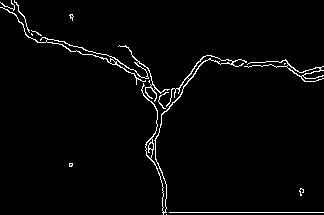
　　去除噪点之后，我们就可以对图像进行处理了。我们选用的主要缺陷检测手段是边缘检测算法。常用的边缘检测算法有Roberts算子，Sobel算子，Canny算子，形态学算子等等。

　　在对多种算子进行测试后，我们发现使用Canny算子进行腐蚀与膨胀运算后可以较好的检测出条状的裂缝，并且团状的潜在危险因素也能被呈现出来。

Canny算子在进行简单除噪后开始寻找图像中的亮度梯度，图像中的边缘可能会指向不同的方向，所以 Canny 算法使用 4 个 mask 检测水平、垂直以及对角线方向的边缘。原始图像与每个 mask 所作的[卷积](https://baike.baidu.com/item/%E5%8D%B7%E7%A7%AF)都存储起来。对于每个点我们都标识在这个点上的最大值以及生成的边缘的方向。这样我们就从原始图像生成了图像中每个点亮度梯度图以及亮度梯度的方向。由于较高的亮度梯度比较有可能是边缘，但是没有一个确切的值来限定多大的亮度梯度是边缘多大，所以 Canny 使用了[滞后](https://baike.baidu.com/item/%E6%BB%9E%E5%90%8E)阈值。

滞后阈值需要两个阈值——高阈值与低阈值。假设图像中的重要边缘都是连续的曲线，这样我们就可以跟踪给定曲线中模糊的部分，并且避免将没有组成曲线的噪声像素当成边缘。所以我们从一个较大的阈值开始，这将标识出我们比较确信的真实边缘，使用前面导出的方向信息，我们从这些真正的边缘开始在图像中跟踪整个的边缘。在跟踪的时候，我们使用一个较小的阈值，这样就可以跟踪曲线的模糊部分直到我们回到起点。

一旦这个过程完成，我们就得到了一个二值图像，每点表示是否是一个边缘点。进而获得我们所需要的边缘图像。

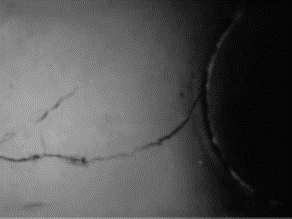
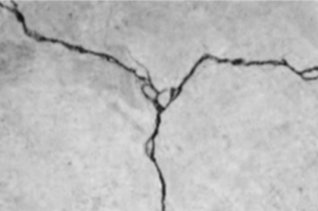
缺陷检测效果演示一

缺陷检测效果演示二

3. 缺陷检测：阈值分割

边缘检测算法之外，还有一类算法也能够较有效地检测出隧管壁上的缺陷，那就是阈值分割算法。阈值分割指以某一阈值为分界线将一幅图像按照灰度值转化成黑白二值图。我们将经过预处理的图像按灰度值分割成两个点集，当被分割成的两个点集之间方差最大时，该灰度值就是将图像二值化所需要的阈值。此处我们所采用的是一种自适应阈值分割算法：使用遗传算法求解的大津法(OTSU)。该方法耗时较长，可用于图像的离线处理过程之中。



使用遗传算法求解的大津法效果演示

其具体代码实现如下：

1. **import** cv2
2. **import** numpy as np
3. **from** matplotlib **import** pyplot as plt
4. **import** random
5. **import** matplotlib.patches as mpatches
6. **import** scipy.ndimage as ndi
7. **from** skimage **import** morphology,filters

10. imag = cv2.imread("D:test4.png")
11. cv2.imshow("orignal",imag)
13. gray = cv2.cvtColor(imag, cv2.COLOR\_BGR2GRAY)
14. cv2.imshow("gray",gray)

17. img = cv2.GaussianBlur(gray,(3,3),0)
18. cv2.imshow("Gau",img)

21. canny = cv2.Canny(img, 50, 150)
22. cv2.imshow("canny",canny)
24. kernel = cv2.getStructuringElement(cv2.MORPH\_RECT,(3, 3))
25. erosion = cv2.erode(img,kernel,iterations = 1)
26. dilation = cv2.dilate(erosion,kernel,iterations = 1)
27. cv2.imshow("deliation",dilation)
29. canny = cv2.Canny(dilation, 50, 150)
30. cv2.imshow("canny",canny)
31. **def** expand(k, m):
32. **for** i **in** list(range(len(k))):
33. k[i] = k[i][:2] + '0'\*(8\*m+2 - len(k[i])) + k[i][2:len(k[i])]
34. **return** k

37. **def** Hist(image):
38. a=[0]\*256
39. h=image.shape[0]
40. w=image.shape[1]
41. MN=h\*w
42. average=0.0
43. **for** i **in** list(range(w)):
44. **for** j **in** list(range(h)):
45. pixel=int(image[j][i])
46. a[pixel]=a[pixel]+1
47. **for** i **in** list(range(256)):
48. a[i]=a[i]/float(MN)
49. average=average+a[i]\*i
50. **return** a, average


54. **def** getTh(seed, m):
55. th = [0, 256]
56. seedInt = int(seed, 2)
57. **for** i **in** list(range(0, m)):
58. tmp = seedInt & 255
59. **if** tmp != 0:
60. th.append(tmp)
61. seedInt = seedInt >> 8
62. th.sort()
63. **return** th


67. **def** fitness(seed, p, average, m):
68. Var = [0.0] \* len(seed)
69. g\_muT = 0.0
71. **for** i **in** list(range(256)):
72. g\_muT = g\_muT + i \* p[i]
74. **for** i **in** list(range(len(seed))):
75. th = getTh(seed[i], m)
76. **for** j **in** list(range(len(th)-1)):
77. w = [0.0] \* (len(th)-1)
78. muT = [0.0] \* (len(th)-1)
79. mu = [0.0] \* (len(th)-1)
80. **for** k **in** list(range(th[j], th[j+1])):
81. w[j] = w[j] + p[k]
82. muT[j] = muT[j] +  + p[k] \* k
83. **if** w[j] > 0:
84. mu[j] = muT[j] / w[j]
85. Var[i] = Var[i] + w[j] \* pow(mu[j] - g\_muT, 2)
86. **return** Var


90. **def** wheel\_selection(seed, Var):
91. var = [0.0]\*len(Var)
92. s = 0.0
93. n = ['']\*len(seed)
94. sumV = sum(Var)
95. **for** i **in** list(range(len(Var))):
96. var[i] = Var[i]/sumV
97. **for** i **in** list(range(1, len(Var))):
98. var[i] = var[i] + var[i-1]
99. **for** i **in** list(range(len(seed))):
100. s = random.random()
101. **for** j **in** list(range(len(var))):
102. **if** s <= var[j]:
103. n[i] = seed[j]
104. **return** n


108. **def** Cross(Next, m):
109. **for** i **in** list(range(0, len(Next) - 1, 2)):
110. **if** random.random() < 0.7:
111. **if** m > 2:
112. tmp = Next[i][10:]
113. Next[i] = Next[i][:10] + Next[i+1][10:]
114. Next[i+1] = Next[i+1][:10] + tmp
115. **else**:
116. tmp = Next[i][6:]
117. Next[i] = Next[i][:6] + Next[i+1][6:]
118. Next[i+1] = Next[i+1][:6] + tmp
119. **return** Next


123. **def** Variation(Next):
124. **for** i **in** list(range(len(Next))):
125. **if** random.random()<0.06:
126. Next[i]=bin(int(Next[i],2)+2)
127. **return** Next


131. **def** genetic\_thres(image, k, m):
132. th = image
133. **for** i **in** list(range(image.shape[0])):
134. **for** j **in** list(range(image.shape[1])):
135. **for** t **in** list(range(1, len(k)-1)):
136. **if** k[t-1] <= image[i][j] < k[t]:
137. th[i][j] = int(k[t-1])
138. **return** th
139. image=canny
140. m = 3
141. items\_x = list(range(0, image.shape[0]))
142. items\_y = list(range(0, image.shape[1]))
143. random.shuffle(items\_x)
144. random.shuffle(items\_y)
145. x = items\_x[0:20\*m]
146. y = items\_y[0:20\*m]
147. seed = []
148. Var = 0.0
149. times = 0
150. k = 0
151. P, average = Hist(image)
152. **for** i **in** list(range(0, 20)):
153. code = int(0)
154. **for** j **in** list(range(0, m)):
155. code = code + gray[x[i\*j]][y[i\*j]] << j\*8
156. seed.append(bin(code))
158. **while** times < 2000:
159. Var = fitness(seed, P, average, m)
160. Next = wheel\_selection(seed, Var)
161. Next = Cross(Next, m)
162. Next = expand(Variation(Next), m)
163. seed = Next
164. times = times + 1
166. **for** j **in** list(range(len(Var))):
167. **if** Var[j] == max(Var):
168. k = getTh(Next[j], m)
169. **print** (k)
171. plt.subplot(131), plt.imshow(image, "gray")
172. plt.title("source image"), plt.xticks([]), plt.yticks([])

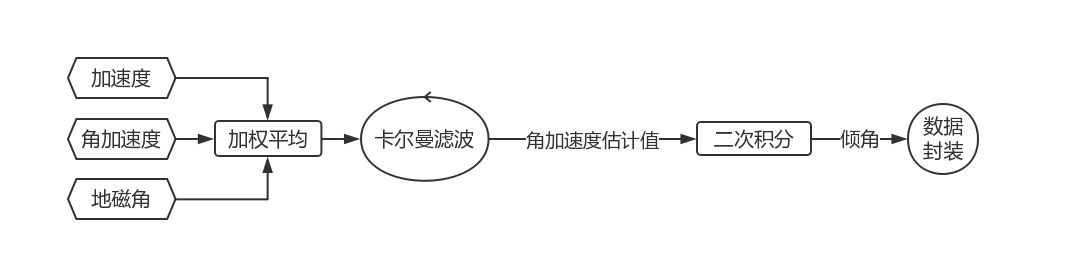
175. plt.subplot(132), plt.hist(image.ravel(), 256)
176. plt.title("Histogram"), plt.xticks([]), plt.yticks([])

179. th1 = genetic\_thres(image, k, m)
180. plt.subplot(133), plt.imshow(th1, "gray")
181. titleName = ''
182. **for** i **in** list(range(1, len(k)-1)):
183. titleName = titleName + str(k[i]) + ', '
184. titleName = titleName[:len(titleName)-2]
185. plt.title("threshold is " + titleName), plt.xticks([]), plt.yticks([])
186. plt.show()

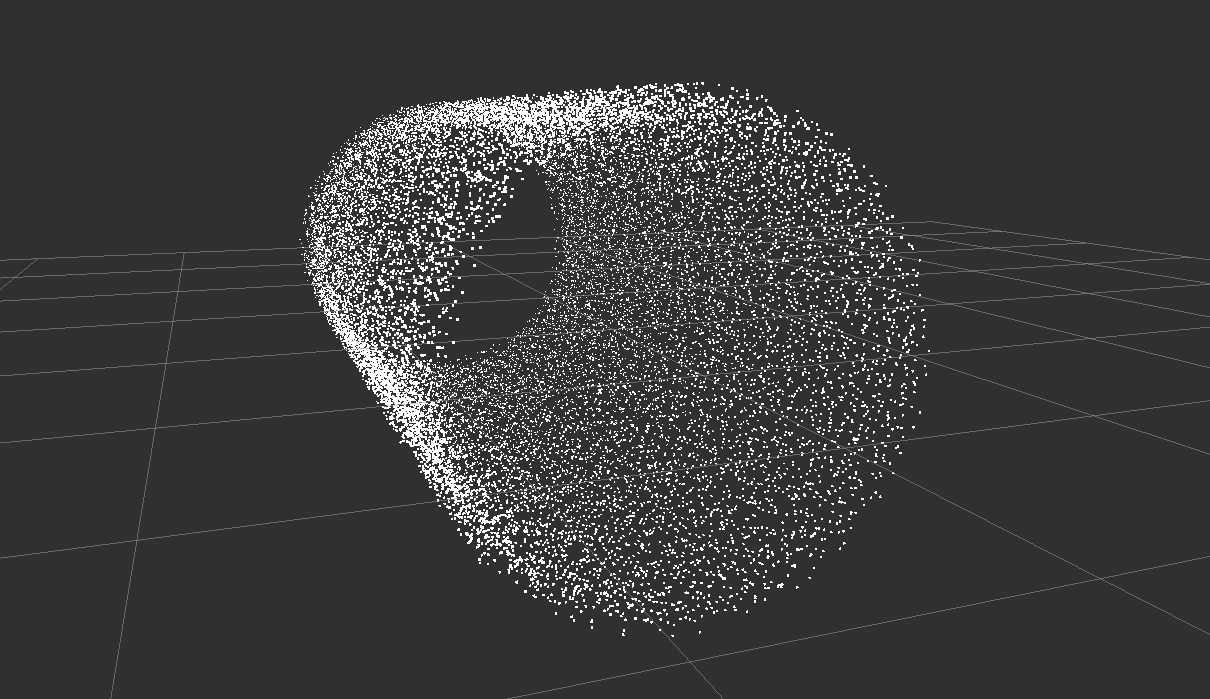
189. cv2.waitKey(0)

3.2.2 三维重构算法

1. 倾角估测



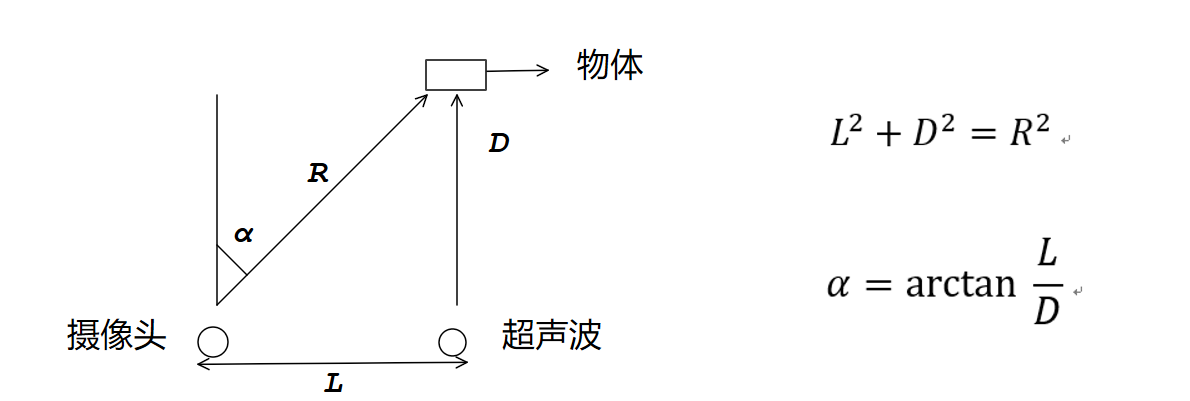
倾角估计算法原理

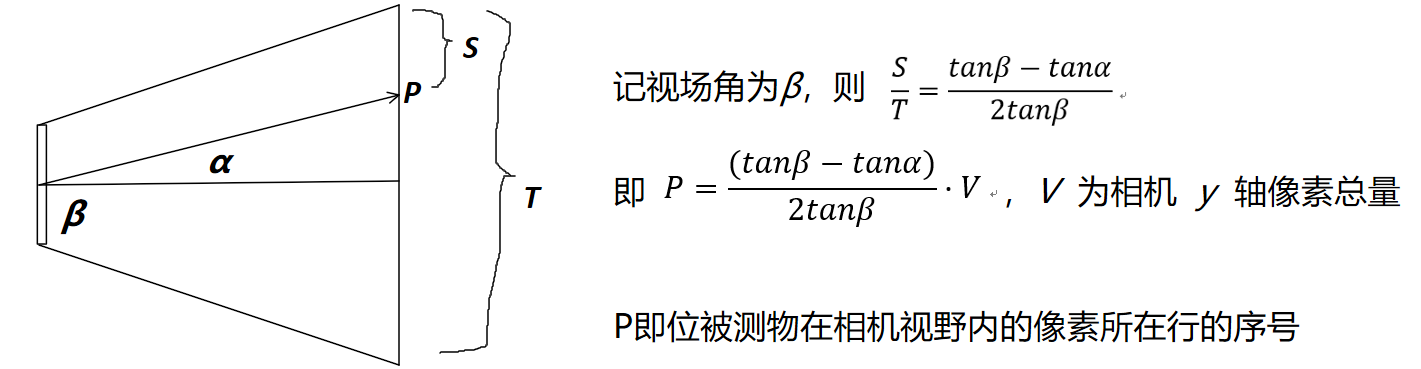


程序模拟的倾角估计生成的点云效果图

2. 点云染色

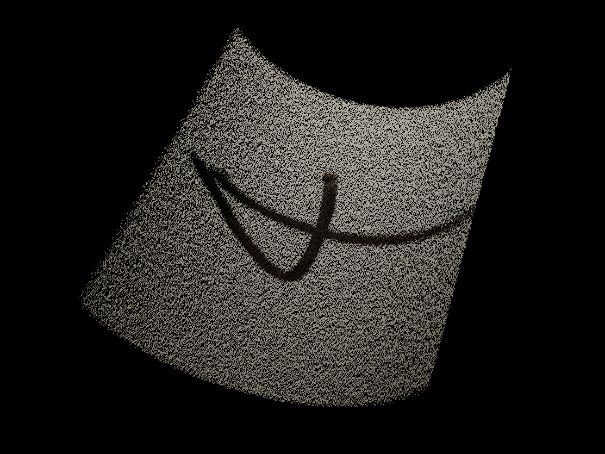
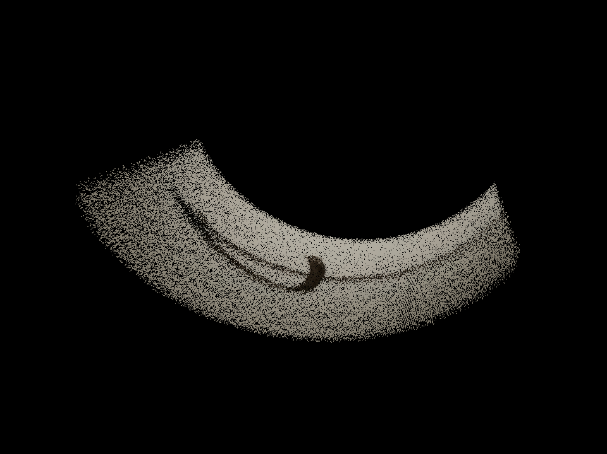
我们使用摄像头与超声波传感器之间的距离值L与点云到超声波传感器的距离D进行三角运算后可估算出物体在摄像机视场内与摄像机法线的夹角α，随后将其正切值与摄像机视角正切值的比例代入计算即可获得其在摄像机所获得的像素图上的具体位置，从而得到该点的像素值。



点云染色算法原理



管道内壁数字图案实拍照片

点云染色效果

其具体实现代码如下：

1. #include <pcl/visualization/cloud\_viewer.h>
2. #include <iostream>
3. #include <pcl/io/io.h>
4. #include <pcl/io/pcd\_io.h>
5. #include <opencv2/opencv.hpp>
7. **using** **namespace** std;
9. **int** user\_data;
10. **const** **double** u0 = 319.52883;
11. **const** **double** v0 = 271.61749;
12. **const** **double** fx = 528.57523;
13. **const** **double** fy = 527.57387;

16. **void** viewerOneOff(pcl::visualization::PCLVisualizer& viewer)
17. {
18. viewer.setBackgroundColor(0.0, 0.0, 0.0);
19. }
21. **int** main()
22. {
23. pcl::PointCloud<pcl::PointXYZRGB> cloud\_a;
24. pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud(**new** pcl::PointCloud<pcl::PointXYZRGB>);
26. cv::Mat color = cv::imread("rgb.jpg");
27. cv::Mat depth1 = cv::imread("depth.png");
29. cv::Size dsize = cv::Size(color.cols, color.rows);
30. cv::Mat depth = cv::Mat(dsize, CV\_8UC3);
31. resize(depth1,depth,dsize);
33. **int** rowNumber = color.rows;
34. **int** colNumber = color.cols;
36. cloud\_a.height = rowNumber;
37. cloud\_a.width = colNumber;
38. cloud\_a.points.resize(cloud\_a.width \* cloud\_a.height);
40. **for** (unsigned **int** u = 0; u < rowNumber; ++u)
41. {
42. **for** (unsigned **int** v = 0; v < colNumber; ++v)
43. {
44. unsigned **int** num = u\*colNumber + v;
45. **double** Xw = 0, Yw = 0, Zw = 0;
47. //Zw = ((double)depth.at<uchar>(u, v)) / 255.0 \* 10001.0;
48. //Xw = (u - u0) \* Zw / fx;
49. //Yw = (v - v0) \* Zw / fy;
50. Zw = ((**double**)depth.at<uchar>(u, v));
51. Xw = u;
52. Yw = v;
54. cloud\_a.points[num].b = color.at<cv::Vec3b>(u, v)[0];
55. cloud\_a.points[num].g = color.at<cv::Vec3b>(u, v)[1];
56. cloud\_a.points[num].r = color.at<cv::Vec3b>(u, v)[2];
58. cloud\_a.points[num].x = Xw;
59. cloud\_a.points[num].y = Yw;
60. cloud\_a.points[num].z = Zw;
61. }
62. }
64. \*cloud = cloud\_a;
66. pcl::visualization::CloudViewer viewer("Cloud Viewer");/\*
67. viewer.showCloud(cloud);
69. viewer.runOnVisualizationThreadOnce(viewerOneOff);
71. while (!viewer.wasStopped())
72. {
73. user\_data = 9;
74. }
75. \*/
76. **return** 0;
77. }

3. 三维重构

　　通过将获得的彩色点云数据打包封装成ROS消息发布到ROS系统里，让RVIZ对其进行监听与处理，即可在RVIZ中看到彩色的三维重构结果。

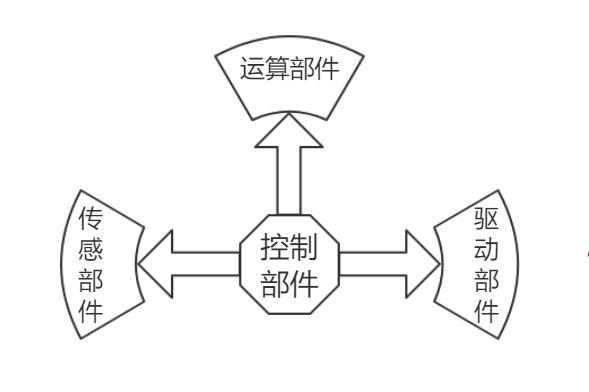
其具体实现代码如下：

1. #include <ros/ros.h>
2. #include <pcl/point\_cloud.h>
3. #include <pcl\_conversions/pcl\_conversions.h>
4. #include <sensor\_msgs/PointCloud2.h>
5. #include <bits/stdc++.h>
7. pcl::PointCloud<pcl::PointXYZ> gen\_pipe(**int** round, **int** layer, **int** length) {
8. pcl::PointCloud<pcl::PointXYZ> cloud;
9. cloud.width = round \* layer;
10. cloud.height = 1;
11. cloud.points.resize(cloud.width \* cloud.height);
13. **for** (**size\_t** i = 0; i < cloud.points.size (); ++i)
14. {
15. **double** theta = (**double**)(i % round) / round \* 2 \* acos(-1.0);
16. cloud.points[i].x = (**double**)(i / round + 1) / layer \* length;
17. cloud.points[i].y = cos(theta) \* (1 + rand () / (RAND\_MAX + 1.0f) / 7);
18. cloud.points[i].z = sin(theta) \* (1 + rand () / (RAND\_MAX + 1.0f) / 7);
19. }
20. **return** cloud;
21. }
23. main (**int** argc, **char** \*\*argv)
24. {
25. ros::init (argc, argv, "pcl\_create");
27. ros::NodeHandle nh;
28. ros::Publisher pcl\_pub = nh.advertise<sensor\_msgs::PointCloud2> ("pcl\_output", 1);
29. //pcl::PointCloud<pcl::PointXYZ> cloud = gen\_circle();
30. sensor\_msgs::PointCloud2 output;
31. //Convert the cloud to ROS message
32. ros::Rate loop\_rate(1);
33. **while** (ros::ok())
34. {
35. pcl::toROSMsg(gen\_pipe(200, 100, 5), output);
36. output.header.frame\_id = "world";
37. pcl\_pub.publish(output);
39. ros::spinOnce();
40. loop\_rate.sleep();
41. }
43. **return** 0;
44. }

**第4章 实现原理**

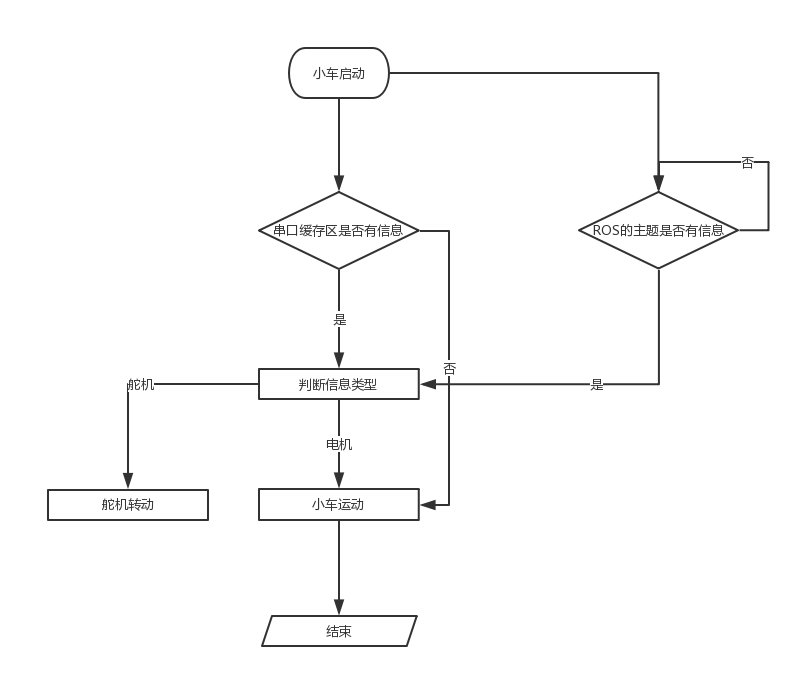
**第5章 系统框图**

## 5.1 硬件框图



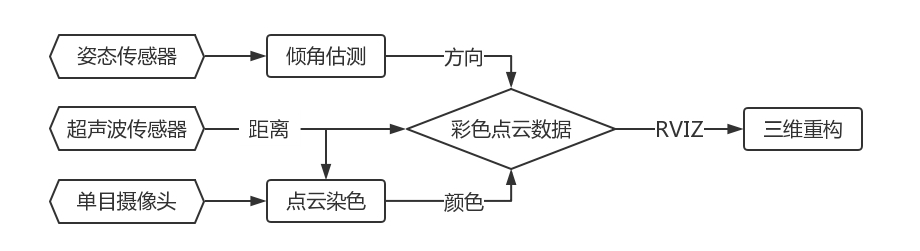
## 5.2 软件框图

5.2.1 控制系统



控制系统框图

5.2.2 视觉识别与三维重构框图



**第6章 测试方案**

## 6.1 硬件相关配置

PC端：接入主路由，连接魔法师平台开发板，通过USB对Arduino和STM32进行调试与程序烧录。

树莓派：连接电源，接入WIFI，接好与Rpi Camera的连线以及与Arduino的连线。

Arduino：接入电源，插好排线。

## 6.2 软件相关配置

PC端：打开实现配置好的含有ROS的Ubuntu虚拟机，启动ROS Core，记录ROS Core地址，打开Rviz，待命。

树莓派：修改配置为PC端ROS Core地址，启动ROS Core，启动主Node。

## 6.3 具体测试流程

1. PC端发送前进命令，测试在隧道内行进的流畅性和准确性。
2. PC端发送停止命令，之后发送检测命令，测试在静止环境下摄像头拍摄的清晰度以及检测的准确性和完整性。
3. PC端发送前进命令，测试在行进状态下检测的准确性和三维重构的准确性及速度。
4. 重复上述操作，直到达到预期状态为止。

## 6.4 结果分析

我们对小车前进的稳定性和检测的准确性进行了测试。

我们的小车在接受ROS发送的前进命令后，能够在较短时间内启动，并且在直行隧道内保持稳定速度直线前行，小车未出现大的车身振动，目视车身没有结构损坏。在弯道上，小车能够原地准确的转弯，进入下一隧道，准确性满足行进需求。

我们的摄像头在接受ROS发送的检测命令后，在静止环境下，摄像头转动频率稳定，Rviz传输的图像清晰，处理后的图像准确。在行进过程中，摄像头转动仍然能保持稳定，图像的清晰度未受大的影响，处理后图片仍满足设计要求。

综上，我们小车已初步具备预期的隧道检测的能力

**第7章 特色**

本项目的产品的硬件与相关算法均为自主设计，拥有很强的适应性与可扩展性，因而将能够面对复杂多变的各类隧管环境。且本项目的设备在其设计之初便预留较多的工业接口，可以安装多个特征传感器，从而实现“一次检测多种管道，一遍查出多种问题”，填补了此类产品的市场空白。

本项目已实现的管道缺陷检测方法主要依赖于对单目摄像头的图像进行处理，在成本和准确度上进行了取舍，有较强的可移植性和可扩展性。同时我们也采用了由现代生物进化论启发的遗传算法，从而能够使找到全局最优解的可能性大大增加。此外，通过边缘检测与阈值分割两种算法的配合，我们的检测方法能够较好的适应光照不均匀的情况，从而减少检测过程中产生的误差。

本项目提出了一种基于超声波的低成本三维重构思路，并实现了相关算法。该方案将能对管道内的环境进行较好的呈现。

本项目采用了分布式机器人操作系统ROS作为主体框架。ROS具有硬件抽象、底层设备控制、进程间与跨进程通信以及程序包管理等诸多功能，是时下新兴的智能机器人控制方案。我们实现了从STM32与树莓派端进行数据收集，随后发送至PC端进行处理，再经过PC端的数据处理与分析，将具体指令反馈给树莓派端的控制回路，从而实现了对机器设备进行精准合理的控制以及对数据的快速传输与储存。

本项目将定制的跨平台的设备管理程序，以实现快速、准确的管道问题检测，并且能精准定位问题管道区域所在位置。未来还将提供依托设备管理软件提供的集群操作功能，单个工作人员即可完成大面积城市隧管网络检测工作，从而极大地减少了人力成本。

**第8章 应用前景**