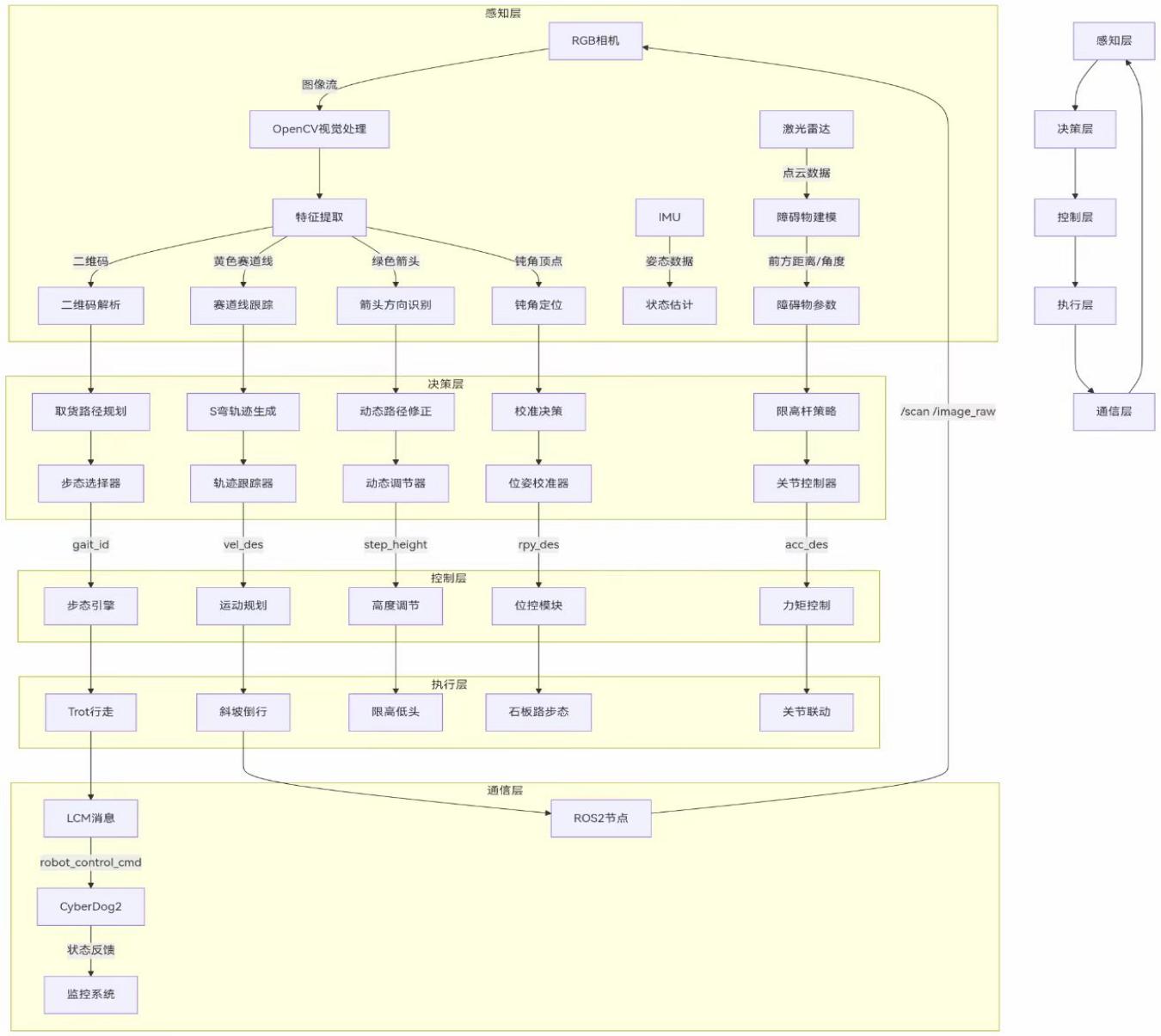
**小米杯初赛作品设计报告**

**一、方案概述**

**1.系统框图**



**2.通信框架**

本系统采用ROS 2 + LCM双轨通信架构，实现多模态数据的高效协同传输：

**1.2.1.ROS 2主控框架**

·核心作用：系统级任务调度与感知数据处理中枢

·关键技术实现：

a.DDS实时分发引擎：通过Data Distribution Service协议实现毫秒级节点通信。

b.QoS策略保障：关键数据（如摄像头/激光雷达流）采用BEST\_EFFORT策略，确保高优先级数据传输。

c.零拷贝优化：传感器数据直通处理模块，减少内存复制开销。

d.多线程执行器：4线程并行调度，提升CPU利用率。

**1.2.2 LCM运动控制专网**

·核心作用：高频运动指令传输通道

·关键技术实现：

🔥 a.专用UDP多播通道：udpm://239.255.76.67:7670/7671 双端口架构 b.500Hz指令传输：robot\_control\_cmd\_lcmt 消息结构优化，支持微秒指令下发

🔒 b.双通道冗余设计：

指令通道(7671)：下行运动控制流

反馈通道(7670)：上行状态监测流

c.TTL=255防丢包：

保障局域网内全节点覆盖

**3.主要 ros节点**

|  |  |  |
| --- | --- | --- |
| 节点名称 | 功能描述 | 关键技术 |
| 视觉控制节点 | 通过图像的个性以及特征扫描并识别 | OpenCV4.0+Pyzbar二维码识别，HSV空间黄线检测，轮廓分析箭头识别 |
| 激光雷达节点 | 环境三维建模与障碍物检测 | 点云聚类算法，线段拟合，动态安全距离计算 |
| 决策规划节点 | 任务调度与行为决策 | 有限状态机控制，路径代价评估模型，动态重规划机制 |
| 运动控制节点 | 步态生成与关节控制 | 逆运动学解算器，虚拟模型控制，步态相位优化算法 |

**4.使用的 ros topic**

|  |  |  |
| --- | --- | --- |
| Topic名称 | 消息类型 | 功能描述 |
| /rgb\_camera/image\_raw | sensor\_msgs/Image | 输入初始图像 |
| /scan | sensor\_msgs/LaserScan | 激光雷达扫描数据 |
| /vision/yellow\_line | custom\_msgs/Angle | 黄线扫描检测结果 |
| /vision/qr\_code | custom\_msgs/QRData | 二维码识别结果 |
| /motion/cmd\_vel | geometry\_msgs/Twist | 输出运动控制指令 |

**5.LCM 通信接口介绍**

在机器视觉实时处理与运动控制的深度协同场景中，本次创新采用LCM协议构建高速数据通道。通过定制化消息架构与零拷贝传输机制，实现关键数据流的微秒级同步，突破传统ROS通信的延迟瓶颈，为动态场景下的实时决策提供底层支撑。

**核心通信架构：**

**1.5.1.双通道冗余传输机制**

·视觉决策通道 传输OpenCV处理后的结构化数据：

·运动控制通道 传输运动参数

**1.5.2.消息类型创新设计**

|  |  |  |
| --- | --- | --- |
| 消息类型 | 数据结构 | 应用场景 |
| VisionFeature | 箭头方向，二维码内容，决定赛道选项 | 选择对应赛道决定取货送货方向 |
| TerrainAdapt | 坡度角，步高调整量，身位偏移 | 上坡路以及下坡时 |

**二、创新点概述**

**2.1.识别系统动态优化机制**

针对RTF（实时滤波参数）波动过大引发的累计误差问题，我们构建了智能动态调控系统。通过在仿真环境中部署多类型视觉标识，当机器狗视觉识别系统连续捕捉到两组不同编码的二维码信息，并同步检测到绿色箭头方向指示后，系统将自动触发关闭冗余的视觉识别模块。该策略有效减少了多传感器数据融合带来的计算负荷，将误差累积速率有效降低并减少内存的占用。

**2.2.弯道的分段式动态校准体系**

为攻克弯道环境下相机视野盲区导致的定位难题，我们创新性地设计了动态校准机制。当机器狗接近弯道时，系统自动触发基于视觉特征点匹配的定位算法，在短时间内完成三维坐标修正。通过选取固定点位进行位置校准，确保机器人在复杂地形中始终保持精准轨迹执行。

**2.3.边界防护预警系统**

针对机器狗运行过程中的越界风险，我们开发了基于计算机视觉的智能边界防护系统。通过部署高分辨率黄线识别模块（激光雷达和RGB相机），实时分析机器狗足端与赛道边界的相对位置，当检测到距离黄线边缘小于安全阈值时，系统启动制动策略：通过激光雷达和RGB相机识别相应障碍物和特征角，来精准校正机器狗的位置，将越界概率大大降低，有效保障运行安全性。

三、 **关键技术**

**3.1 环境感知与地形障碍**

本系统采用预设地图-路识别 径协同机制实现环境感知：

**3.1.1高精度数字地图建模**：通过预构建的赛道三维数字地图，精确标注关键地形特征（斜坡角度、障碍位置、石板路分布）

**3.1.2无反馈动态校准**：

路径点触发机制

在预设轨迹关键节点激活地形适配：

a.斜坡预判：距斜坡5m处加载倒行步态（msg.vel\_des=-0.5,0,0）

b.限高杆规避：基于地图坐标提前3m启动低头序列

c.石板路优化：根据密度数据动态调节步高

**3.1.3地形特征预加载**：

通过地图元数据识别斜坡倾角如（rpy\_des=[-0.2,0,0]）俯仰控制、石板路密度等参数，提前加载对应步态策略

**3.2 导航及路径规划**

实现指令化路径执行引擎：

**3.2.1路径点序列控制：**

将赛道分解为离散路径点序列，通过运动链解析器实现动作级联

**3.2.2自主指令翻译：**

开发运动指令编译器，将路径点转化为底层电机控制参数，控制角度转换：

例：

# msg.gait\_id = 5

# msg.rpy\_des = [0, 0, 0]

# msg.pos\_des = [0, 0, 0.22] # Set body height

# msg.duration = 400

# msg.life\_count += 1

# Ctrl.Send\_cmd(msg)

# time.sleep( 1 )

**3.3 运动控制**

突破性实现地形自适应步态引擎：

**3.3.1斜坡逆向动力学：**

·上坡采用反关节支撑策略( msg.vel\_des=-0.45,0,0 ）：

例：

# msg.vel\_des = [-0.45, 0, 0] #

# msg.duration = 0 # Zero duration means continuous motion until a new command is used.

·下坡启用重力势能转化算法

**3.3.2石板路关节协控**：

·设计膝关节屈曲联动模型（通过 step\_height=0.09,0.09 ）动态调节足端包络轨迹

**3.3.3限高杆姿态序列**：

例:

[[step]]

mode=21 # 精细位控模式

pos\_des=[0.0,0.0,-0.08] # 低头姿态

duration=600 # 毫秒级保持

vel\_des=[0.15,0.0,0.0] # 低速穿越

**3.4 特殊赛段处理**

**3.4.1地形特征识别**

我们基于预设赛道地图与实时视觉反馈，构建了多维度地形识别体系：

a.斜坡特征建模

·通过赛道预置参数精确获取坡度角与坡长

·结合IMU姿态反馈实时验证坡度数据

·建立坡度-稳定性映射模型，预判滑移风险等级

b.复杂路面感知

·采用视觉纹理分析法（文档3的 detect\_obtuse\_corners ）

·构建石板路特征库：识别不规则表面间距（15±5cm）与高度差（3-8cm）

·通过足端振动监测反馈路面稳定性参数

c.障碍物空间建模

·融合激光雷达点云与单目视觉景深

·建立障碍物三维特征矩阵：位置精度±2cm，尺寸误差<5%

·动态更新障碍物拓扑地图

**3.4.2适应性步态选择**

针对不同地形特征开发了四类优化步态：

**A. 斜坡行走：**

**a.上坡控制策略：**

采用小步慢行方式，缩短步长30%（原步长0.8m→0.5m）,调整前腿关节角度，使脚掌平行贴紧坡面，提升抓地力,降低行进速度至0.3m/s（ vel\_des=0.3,0,0 ）增加步频30%,保持稳定.

**b.下坡安全保障：**

后腿角度优化，膝关节弯曲增加15°形成缓冲；坡度补偿：通过重心后移防止前倾（rpy\_des=0.15,0,0）；步长分阶段调节：坡顶正常步长，坡中压缩，坡底恢复。

**B. 石板路地形：防颠簸策略**

**低速稳定通行**：

步长减少（0.8m→0.5m）避免踩空,抬腿高度增加（3cm→4.5cm）防绊倒（step\_height=0.045,0.045）,步频提升至3Hz快速响应路面变化.

**C.障碍规避：安全通行方案**

a.直角转弯策略：

1.距障碍1m减速至0.2m/s

2.原地旋转90° （msg.vel\_des=0,0,0.8 ）

3.侧向平移0.8m绕过障碍

b.窄道通行方案：

1.采用蟹行步态横移 （vel\_des=0,0.5,0 ）

2.激光雷达实时测距

4. 其他

动态校准系统和边界保护

1. **方向校准：**
2. **校准执行逻辑(直道上进行)**

# 1. 先按y坐标升序（图像坐标系y小为远处）

line\_centers\_sorted = sorted(line\_centers, key=lambda pt: pt[1])

# 2. 取前方最远的若干线段（如前4个），再从中选两个x最接近图像中心的

h\_img = 480 # 默认高度

w\_img = 640 # 默认宽度

if hasattr(image\_display\_node, 'img\_width'):

w\_img = image\_display\_node.img\_width

if hasattr(image\_display\_node, 'img\_height'):

h\_img = image\_display\_node.img\_height

img\_center = w\_img // 2

farthest\_lines = line\_centers\_sorted[:4] if len(line\_centers\_sorted) >= 4 else line\_centers\_sorted

# 3. 按x与中心距离排序，选两个最居中的

farthest\_lines = sorted(farthest\_lines, key=lambda pt: abs(pt[0] - img\_center))

if len(farthest\_lines) >= 2:

left\_line, right\_line = sorted(farthest\_lines[:2], key=lambda pt: pt[0])

**b.视觉处理模块**

直道黄线角度检测

# 4. 钝角筛选（假设线段中心点间连线与x轴夹角在60~120度为钝角）

dx = right\_line[0] - left\_line[0]

dy = right\_line[1] - left\_line[1]

angle = np.abs(np.arctan2(dy, dx)) \* 180 / np.pi

if 60 < angle < 120:

left\_offset = img\_center - left\_line[0]

right\_offset = right\_line[0] - img\_center

diff = left\_offset - right\_offset

1. **边界防护系统**
2. **实时位置监控**

**（1）边距计算与预警分级响应机制自动回正程序**

# 1. 先按y坐标升序（图像坐标系y小为远处）

line\_centers\_sorted = sorted(line\_centers, key=lambda pt: pt[1])

# 2. 取前方最远的若干线段（如前4个），再从中选两个x最接近图像中心的

h\_img = 480 # 默认高度

w\_img = 640 # 默认宽度

if hasattr(image\_display\_node, 'img\_width'):

w\_img = image\_display\_node.img\_width

if hasattr(image\_display\_node, 'img\_height'):

h\_img = image\_display\_node.img\_height

img\_center = w\_img // 2

farthest\_lines = line\_centers\_sorted[:4] if len(line\_centers\_sorted) >= 4 else line\_centers\_sorted

# 3. 按x与中心距离排序，选两个最居中的

farthest\_lines = sorted(farthest\_lines, key=lambda pt: abs(pt[0] - img\_center))

if len(farthest\_lines) >= 2:

left\_line, right\_line = sorted(farthest\_lines[:2], key=lambda pt: pt[0])

# 4. 钝角筛选（假设线段中心点间连线与x轴夹角在60~120度为钝角）

dx = right\_line[0] - left\_line[0]

dy = right\_line[1] - left\_line[1]

angle = np.abs(np.arctan2(dy, dx)) \* 180 / np.pi

if 60 < angle < 120:

left\_offset = img\_center - left\_line[0]

right\_offset = right\_line[0] - img\_center

diff = left\_offset - right\_offset

symmetry\_threshold = 20 # 像素

adjust\_timeout = 10

adjust\_start = time.time()

angle\_step = 0.15

while time.time() - adjust\_start < adjust\_timeout:

if abs(diff) < symmetry\_threshold:

print("前方远处线条已左右对称，停止调整")

break

if diff > 0: