

Smart Driving Sandbox Development Based on Qualcomm Platform and ROS2

Project Overview

This project demonstrates the development of a miniature intelligent driving system using the Qualcomm computing platform, Aidlux fusion system, ROS2, and ESP32 smart car. It covers the development and integration of three core autonomous driving modules: **environmental perception, motion control, and behavioral decision-making**, ultimately achieving a complete application that can operate autonomously in a customized sandbox.

Hardware Requirements



alt text



alt text

- **Smart Car:** Integrated with ESP32 microprocessor, camera, motor drive module, and Wi-Fi capability
- **Control System:** Rhino Pi-X1 with the following specifications:
 - AI Performance: ~48 TOPS INT8
 - CPU: 1× Kryo Prime @ 3.2GHz, 2× Kryo Gold @ 2.8GHz, 3× Kryo Silver @ 2.0GHz
 - GPU: Adreno 740, GPU performance (GFLOPS) 3000 GFLOPS, 3 tflops

- Memory: 16GB LPDDR5x (default), 24GB LPDDR5x (optional)
- Connectivity: Wi-Fi 6, Bluetooth 5.1, LAN
- Power: DC 12V/24V 3A

Software Requirements

- **Operating System:** AidLux (Android 14 + Ubuntu 22.04) or Ubuntu 22.04
- **Development Tools:**
 - ROS2 (Robot Operating System 2)
 - OpenCV for computer vision
 - C++ and Python programming languages
 - ADB tools for device communication
 - QtScrcpy for screen mirroring

System Architecture

The system adopts a **layered distributed architecture** with three main layers:

1. **Perception Layer** (car end): Collects environmental data in real-time via camera
2. **Decision Layer** (central control system): Processes sensor data and generates control commands

3. **Execution Layer** (car end): Receives commands and controls vehicle movement

Core Modules

- **Communication Module:** Enables data exchange between car and central system via Wi-Fi
- **Sensor Module:** Captures RGB images and transmits video stream to central system
- **Motion Control Module:** Parses speed commands and drives motors
- **Visual Perception Module:** Detects lane lines, zebra crossings, traffic lights, and stop lines
- **Decision Planning Module:** Implements finite state machine (FSM) for control behavior
- **Control Command Generation Module:** Converts control behavior to speed commands

Installation and Setup

1. Hardware Setup:

- Assemble the smart car with ESP32, camera, and motor drive module

- Set up the sandbox environment with road markings and traffic elements
- Connect the Rhino Pi-X1 to the network

2. **Software Setup:**

- Install RhinoPi X1 Android Image or RhinoPi X1 Ubuntu22.04 on the Rhino Pi-X1
- Set up ROS2 environment
- Install required dependencies (OpenCV, etc.)
- Configure network connection between car and central system

3. **ROS2 Package Creation:**

- Create interfaces for motion control (`tank_interfaces`)
- Implement motion control nodes in C++ and Python
- Create video stream publisher nodes
- Implement computer vision nodes for environmental perception

Usage

1. **Start the System:**

- Power on the Rhino Pi-X1 and smart car
- Establish Wi-Fi connection between car and central system

2. **Run the Nodes:**

- Start video stream publisher to receive camera feed
- Run computer vision node for environmental perception
- Start motion control node to receive and execute commands

3. Testing:

- Use ROS2 topic publishing to send speed commands
- Monitor the car's behavior in the sandbox environment
- Adjust parameters for optimal performance

Communication Protocol

- **Video Stream:** HTTP/MJPEG stream via

`http://192.168.4.1:81/stream`

- **Control Commands:** UDP protocol to `192.168.4.1:9000` with binary data format

Code Structure

```
src/
├── tank_interfaces/           # Custom ROS2 interfaces
│   └── msg/
│       └── TankCommand.msg   # Motion control message definition
├── tank_control/             # C++ motion control implementation
│   └── src/
│       └── tank_control.cpp   # UDP command sender
├── tank_control_py/          # Python motion control implementation
│   └── tank_control_py/
│       └── tank_control.py    # UDP command sender (Python)
├── video_stream_publisher/    # C++ video stream publisher
│   └── src/
│       └── video_stream_publisher.cpp
├── video_stream_publisher_py/ # Python video stream publisher
│   └── video_stream_publisher_py/
│       └── video_stream_publisher.py
└── computer_vision/          # C++ computer vision implementation
```

```
└─ src/
    └─ computer_vision.cpp
└─ computer_vision_py/          # Python computer vision
   implementation
    └─ computer_vision_py/
        └─ computer_vision.py
```

Key Features

- **Real-time Environmental Perception:** Detects lane lines, zebra crossings, traffic lights, and stop lines
- **Distributed Control Architecture:** Separates perception, decision, and execution layers
- **Dual-language Implementation:** Provides both C++ and Python code for flexibility
- **ROS2 Integration:** Leverages ROS2 for efficient communication and modular design
- **Custom Communication Protocol:** Optimized for reliable control of the ESP32-based smart car

Troubleshooting

- **Connection Issues:** Ensure both car and central system are on the same network
- **Video Stream Problems:** Check camera connections and network stability

- **Motion Control Issues:** Verify UDP port settings and command format
- **Detection Accuracy:** Adjust OpenCV parameters for different lighting conditions