University Rover Challenge

Complete Technical Documentation

& Startup Guide

University of Pittsburgh Robotics Club

ROS 2 Humble • NVIDIA Jetson Nano • Docker

Version 2.0 — January 2026

Platform:	ROS 2 Humble Hawksbill
Hardware:	NVIDIA Jetson Nano, Arduino
Deployment:	Docker Containers
GUI:	PyQt5

Contents

List of Figures

List of Tables

Chapter 1

Project Overview

1.1 Introduction

This document provides comprehensive technical documentation for the University of Pittsburgh Robotics Club's University Rover Challenge (URC) rover system. The system integrates advanced robotics technologies including:

- **6-wheel differential drive system** with GoBilda motors
- **Multi-sensor fusion** including GPS, IMU, ultrasonic sensors, and cameras
- **ROS 2 Humble** for distributed robot communication
- **Docker containerization** for reproducible deployments
- **PyQt5 GUI system** for teleoperation and monitoring

1.2 Competition Tasks

The University Rover Challenge consists of multiple mission types:

Table 1.1: URC Competition Tasks

Task	Description
Autonomous Navigation	GPS-guided traversal of challenging Martian-analog terrain
Equipment Servicing	Precise manipulation tasks using robotic arm
Extreme Delivery	Package delivery in harsh environmental conditions
Science Operations	Sample collection and in-field analysis
Teleoperation	Manual control for complex scenarios

1.3 Key Specifications

Table 1.2: Rover System Specifications

Metric	Value
Navigation Accuracy	±2m GPS positioning
Video Latency	<200ms streaming
Control Response	<50ms command execution
Battery Life	4+ hours autonomous operation
Operating Range	1km+ radio communication
Motor Control	6 GoBilda motors (0–255 PWM)
Camera Frame Rate	30 Hz
IMU Update Rate	10 Hz
Serial Baud Rate	115200 bps

Chapter 2

System Architecture

2.1 Technology Stack

The rover system leverages a modern technology stack for robotics development:

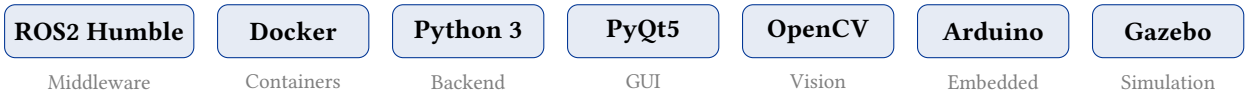

Figure 2.1: Technology Stack

2.2 High-Level Architecture

The rover system uses a **dual-container architecture** for optimal performance and separation of concerns.

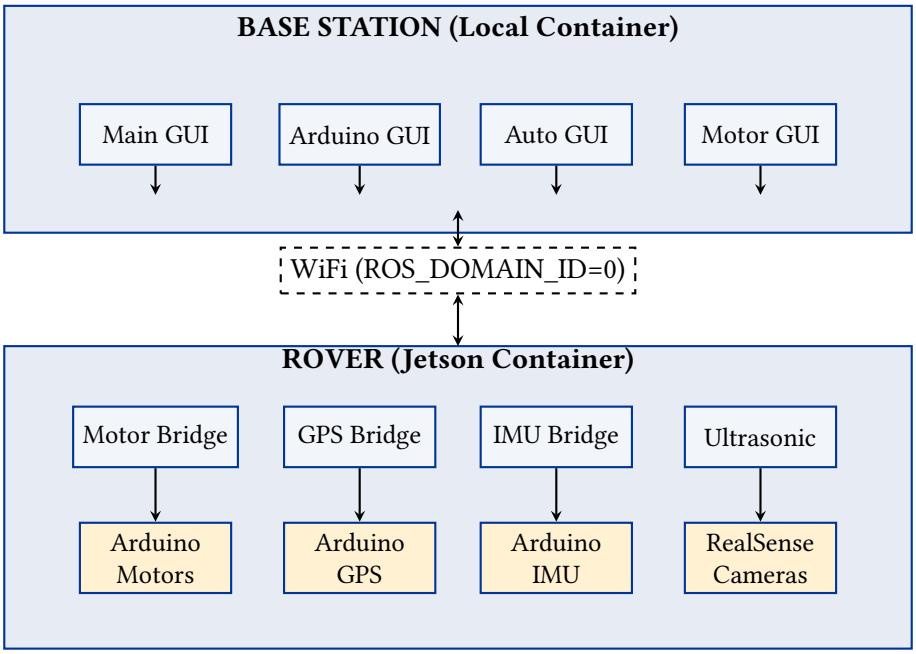

Figure 2.2: System Architecture Overview

2.3 Detailed Hardware Architecture

The following diagram shows the complete hardware interconnections of the rover system, including power distribution, communication pathways, and all sensor subsystems.

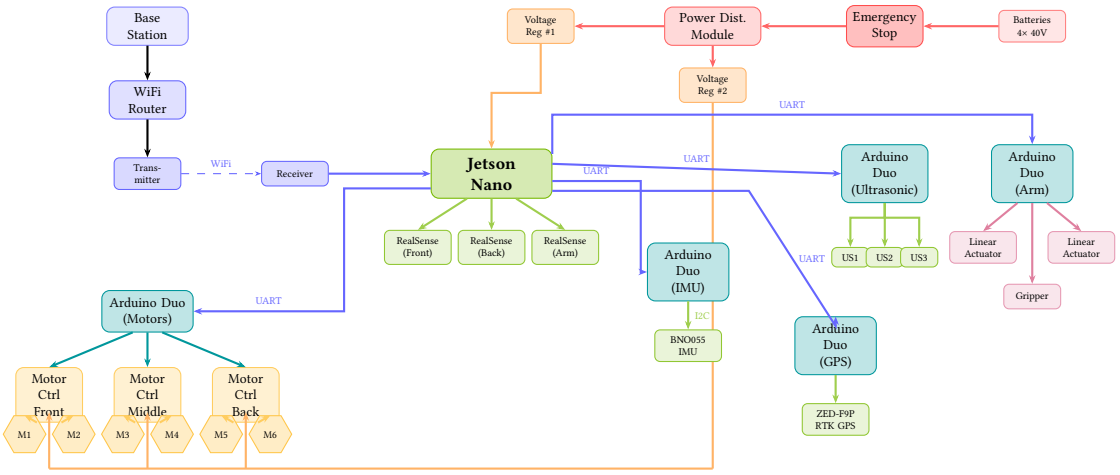

Figure 2.3: Detailed Hardware System Architecture

2.3.1 Hardware Subsystems Summary

Table 2.1: Hardware Subsystem Overview

Subsystem	Components	Description
Power	4× 40V batteries, 2× voltage regulators, E-stop	Provides isolated power rails for compute and motors
Communication	WiFi router, transmitter/receiver	Wireless link between base station and rover
Compute	Jetson Nano	Central processing unit for all onboard operations
Motors	6× GoBilda motors, 3× motor controllers	6-wheel differential drive system
Vision	3× Intel RealSense D400 cameras	Front, back, and arm-mounted RGB-D cameras
Sensors	BNO055 IMU, ZED-F9P RTK GPS, 3× HC-SR04	Navigation and obstacle detection sensors
Arm	2× linear actuators, gripper	4-DOF robotic manipulator with 5kg payload

Tip

The dual-container architecture saves Jetson processing power for real-time control while enabling easier debugging with local GUI applications. ROS 2 communication works seamlessly over WiFi.

2.4 Communication Flow

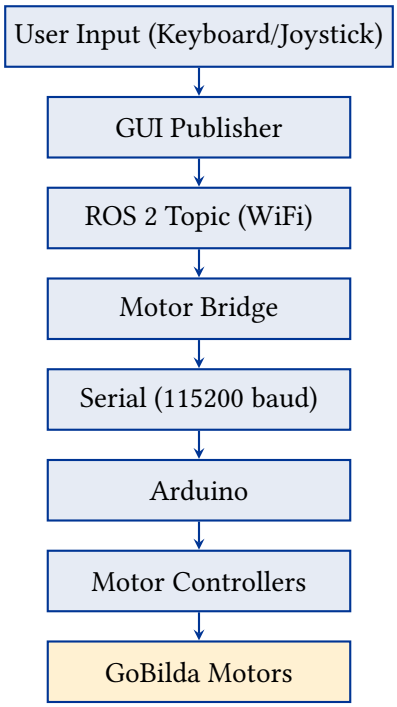

Figure 2.4: Motor Control Communication Flow

2.5 Bidirectional Data Flow

The system supports bidirectional communication: commands flow from the base station to the rover, while sensor data flows from the rover back to the base station.

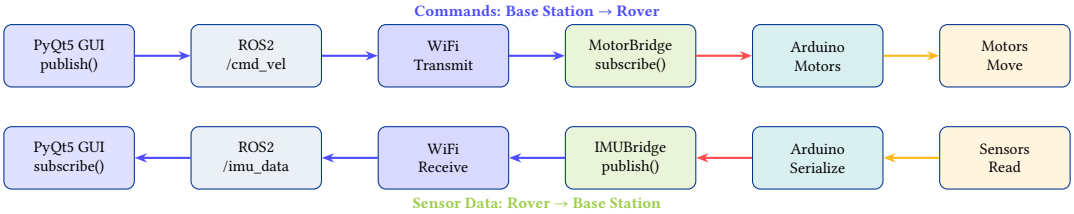

Figure 2.5: Bidirectional Data Flow

Chapter 3

Directory Structure

3.1 Project Layout

The project follows a modular organization:

```
URC/
|-- docker/                    # Docker configuration
|   |-- jetson/               # On-board Jetson container
|   |   |-- Dockerfile
|   |   |-- docker-compose.yml
|   |   |-- start.sh
|   |   |-- supervisord.conf
|   |   +-- setup.py
|   |-- jetson_local/         # Alternative Jetson setup
|   +-- local/                # Base station container
|       |-- Dockerfile
|       |-- docker-compose.yml
|       |-- start_mac.sh
|       |-- start_linux.sh
|       +-- sim-launch.sh
|
|-- ros_bridge/               # ROS 2 hardware bridges
|   |-- arduino_bridge_base/  # Base class for bridges
|   |-- motor_bridge/         # Motor control system
|   |-- motor_subscriber/     # Motor command handler
|   |-- gps_bridge/           # GPS sensor bridge
|   |-- imu_bridge/           # IMU sensor bridge
|   +-- ultrasonic_bridge/    # Ultrasonic sensor bridge
|
|-- guis/                     # PyQt5 GUI applications
|   |-- gen_gui.py            # Main control dashboard
|   |-- arduino_gui.py        # Hardware control interface
|   |-- auto_gui.py           # Autonomous navigation
|   |-- json_motorGUI.py      # Motor/Arm keyboard control
|   |-- publishers/           # ROS 2 publisher classes
|   |-- subscribers/          # ROS 2 subscriber classes
|   +-- camera/               # Camera utilities
|
|-- teleop/                   # Teleoperation system
|   +-- teleop_string_publisher.py
|
|-- odometry/                 # Navigation and localization
|   +-- gps/                  # GPS odometry utilities
|
|-- simulation/               # Gazebo simulation
|   +-- colcon_ws/            # ROS 2 workspace
|       +-- src/my_robot_description/
```

```
|
|-- README.md
|-- CONTRIBUTING.md
+-- Makefile
```

Listing 3.1: Project Directory Structure

3.2 Key Directories

3.2.1 Docker Configuration

- `docker/jetson/` — On-board rover container with NVIDIA runtime support
- `docker/local/` — Base station container with X11 forwarding for GUI
- `docker/jetson_local/` — Alternative hybrid setup

3.2.2 ROS Bridge

- `arduino_bridge_base/` — Base class providing serial communication (248 lines)
- `motor_bridge/` — Unified motor control with multiple input sources
- `gps_bridge/` — NMEA sentence parsing and publishing
- `imu_bridge/` — BNO055 9-axis IMU integration
- `ultrasonic_bridge/` — Multi-sensor distance measurement

3.2.3 GUI System

- `gen_gui.py` — Main dashboard (623 lines) with camera feeds and sensor data
- `arduino_gui.py` — Joystick-style motor control (617 lines)
- `json_motorGUI.py` — Keyboard hotkey control (292 lines)
- `publishers/` — MotorPublisher, TwistPublisher classes
- `subscribers/` — IMUSubscriber, GPSSubscriber, data parsers

Chapter 4

Hardware Requirements

4.1 Power System

The rover uses a distributed power architecture with isolated rails for compute and motor systems.

Table 4.1: Power System Specifications

Component	Specification	Purpose
Batteries	4× 40V rechargeable	Main power source for entire rover
Voltage Regulator #1	40V → 5V	Dedicated power for Jetson Nano
Voltage Regulator #2	40V → motor voltage	Powers motor controllers
Emergency Stop	Relay cutoff switch	Immediately disconnects all power
Power Distribution	Central module	Routes power to subsystems

Important

The power system uses **isolated power rails** to prevent motor inrush current from causing brownouts on the Jetson Nano. Voltage Regulator #1 provides dedicated, stable power to the compute unit.

4.2 On-Board Computer

Table 4.2: Jetson Nano Specifications

Component	Specification
Computer	NVIDIA Jetson Nano (4GB)
Storage	64GB+ SD Card or NVMe SSD
Power	5V 4A barrel jack
Cooling	Active fan (recommended)

4.3 Motor System

The motor system uses 6× GoBilda motors with 3 dual H-bridge controllers. For complete specifications, see the Key Specifications table in Chapter 1 and the Hardware Subsystems table in Chapter 2.

4.3.1 Pin Configuration

Table 4.3: Arduino Motor Pin Configuration

Motor	PWM Pin	Direction Pin	Side
Front Left	2	53	Left
Middle Left	4	51	Left
Back Left	6	49	Left
Front Right	3	52	Right
Middle Right	5	50	Right
Back Right	7	48	Right

4.4 Sensors

Table 4.4: Sensor Configuration

Sensor	Model	Connection	Rate
IMU	Adafruit BNO055 (9-axis)	I2C via Arduino	10 Hz
GPS	u-blox ZED-F9P RTK (±2cm)	Serial 115200	1 Hz
Cameras	3× Intel RealSense D400	USB 3.0	30 Hz
Ultrasonic	3× HC-SR04	Arduino GPIO	10 Hz

4.4.1 Sensor Details

- **RTK GPS (ZED-F9P):** Uses carrier phase measurements instead of code-based positioning, achieving ±2cm accuracy vs ±5m standard GPS. Requires base station corrections.
- **IMU (BNO055):** 9-axis sensor (3-axis accelerometer, gyroscope, magnetometer) with onboard sensor fusion. Connected via I2C to dedicated Arduino.
- **Cameras (RealSense D400):** RGB-D cameras providing both color and depth data. Mounted at front, back, and on the robotic arm for manipulation tasks.
- **Ultrasonic (HC-SR04):** 3 sensors for obstacle detection. Range: 2cm–400cm. Pin configuration: Sensor 1 (Trigger=8, Echo=9), Sensor 2 (Trigger=50, Echo=51), Sensor 3 (Trigger=48, Echo=49).

4.5 Serial Port Configuration

Table 4.5: Serial Port Assignments

Device	Port	Baud Rate	Notes
Motors	/dev/ttyACM1	115200	Hardcoded in motor_subscriber
GPS	/dev/ttyACM0	9600	SoftwareSerial (RX=10, TX=11)
IMU	TCP:8888 or Serial	115200	TCP in Docker, serial standalone
Ultrasonic	/dev/ttyACM0	115200	Shared with GPS/IMU Arduino

Important

IMU Communication Options: The IMU bridge supports two modes:

- **TCP Mode (Docker):** Connects to `host.docker.internal:8888` for use inside containers
- **Serial Mode (Standalone):** Auto-detects from `/dev/ttyACM0`, `/dev/ttyUSB0`, or `/dev/cu.usbmodem*`

Chapter 5

Software Dependencies

5.1 System Requirements

- **Operating System:** Ubuntu 22.04 LTS
- **ROS Version:** ROS 2 Humble Hawksbill
- **Python:** 3.10+
- **Docker:** 20.10+

5.2 Python Dependencies

```
# Core ROS 2
rclpy
ros2cli

# Hardware Communication
pyserial>=3.5
pyrealsense2>=2.50.0

# GUI Framework
PyQt5>=5.15.0
PyQt5-sip

# Computer Vision
opencv-python>=4.5.0
numpy<2.0

# ROS 2 Message Bridges
cv_bridge
sensor_msgs
geometry_msgs
std_msgs
```

Listing 5.1: Python Package Requirements

5.3 System Packages (APT)

```
# Base development
python3-pip python3-pyqt5 python3-pyqt5.qtsvg

# Graphics libraries (for GUI)
```

```
libxcb-xinerama0 libxcb-cursor0 libxkbcommon-x11-0
libgl1-mesa-glx libgl1-mesa-dri mesa-utils x11-apps

# ROS 2 Humble
ros-humble-desktop ros-dev-tools
ros-humble-teleop-twist-keyboard ros-humble-joy

# Intel RealSense
librealsense2-utils librealsense2-dev
ros-humble-librealsense2-* ros-humble-realsense2-*
```

Listing 5.2: System Package Requirements

5.4 Arduino Libraries

- `Adafruit_Sensor` — Unified sensor abstraction layer
- `Adafruit_BNO055` — 9-axis IMU driver library

Chapter 6

Docker Configuration

6.1 Jetson Container

6.1.1 Overview

The Jetson container runs on-board the rover and manages all hardware interfaces.

```
cd docker/jetson
sudo docker build -t urc_jetson .
sudo docker-compose up -d
```

Listing 6.1: Build and Start Jetson Container

6.1.2 Key Features

- NVIDIA GPU support (`runtime: nvidia`)
- Privileged mode for device access
- Host network mode for ROS 2 communication
- Supervisor process management

6.1.3 Device Mappings

```
devices:
  - /dev:/dev                    # All devices
  - /dev/bus/usb:/dev/bus/usb    # USB devices
  - /dev/video0:/dev/video0      # Camera 1
  - /dev/video1:/dev/video1      # Camera 2
```

Listing 6.2: Docker Compose Device Configuration

6.1.4 Environment Variables

See Appendix A for the complete environment variable reference. Key variables for Jetson include `ROS_DOMAIN_ID`, `NVIDIA_DRIVER_CAPABILITIES`, and `NVIDIA_VISIBLE_DEVICES`.

6.1.5 Supervised Processes

Table 6.1: Supervisor Managed Processes

Process	Description	Auto-restart
setup	RealSense camera initialization	Yes
gps_bridge	GPS data publisher	Yes
motor_subscriber	Motor command handler	Yes
imu_bridge	IMU orientation publisher	Yes
ultrasonic_bridge	Ultrasonic sensor publisher	Yes

6.2 Local Container (Base Station)

6.2.1 macOS Setup

```
# Prerequisite: Install XQuartz and enable network clients
# System Preferences > Security > Allow connections

cd docker/local
chmod +x start_mac.sh
./start_mac.sh
```

Listing 6.3: macOS Container Startup

6.2.2 Linux Setup

```
cd docker/local
chmod +x start_linux.sh
./start_linux.sh
```

Listing 6.4: Linux Container Startup

6.2.3 Local Environment Variables

See Appendix A for the complete environment variable reference. Key variables for GUI display include `DISPLAY` and `QT_X11_NO_MITSHM`.

6.3 Simulation Container

The simulation uses Gazebo with a custom URDF robot model.

6.3.1 Simulation Workspace Structure

```
simulation/colcon_ws/
+-- src/
    +-- my_robot_description/
        |-- launch/
        |   |-- display.launch.py    # RViz visualization
        |   +-- launch_sim.launch.py # Full Gazebo simulation
        |-- urdf/
        |   +-- RAS_Rover.urdf       # Robot description
        +-- scripts/
            +-- robot_controller.py  # Simulation control
```

6.3.2 Building the Simulation

```
cd simulation/colcon_ws
colcon build --symlink-install
source install/setup.bash
```

Listing 6.5: Build Simulation Workspace

6.3.3 Launching Simulation

```
# Option 1: Via Docker
cd docker/local
./sim-launch.sh

# Option 2: Direct launch for RViz visualization
ros2 launch my_robot_description display.launch.py

# Option 3: Full Gazebo simulation
ros2 launch my_robot_description launch_sim.launch.py use_sim_time:=true
```

Listing 6.6: Simulation Startup

Chapter 7

Quick Start Guide

Tip

This chapter provides a condensed overview of the startup process. For comprehensive step-by-step procedures, troubleshooting, and verification commands, see Chapter 8: **Detailed Startup Procedures**.

7.1 Prerequisites

Warning

Linux Base Station Required: To communicate with the rover Jetson, you must use a Linux-based base station. This can be:

- A laptop or desktop running Linux (Ubuntu 22.04 recommended)
- Another Jetson device running the `jetson_local` configuration

macOS support is limited to simulation and local development only.

7.1.1 Install Docker

```
# Linux
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh

# macOS
brew install --cask docker
```

Listing 7.1: Docker Installation

7.1.2 Clone Repository

```
git clone https://github.com/pitt-robotics/URC.git
cd URC
```

Listing 7.2: Clone Repository

7.1.3 Connect Hardware

Warning

USB Connection Order is Critical! The motor Arduino is hardcoded to `/dev/ttyACM1`. Linux assigns device numbers in the order USB devices are plugged in.

1. **First:** Plug in IMU/GPS Arduino (gets `/dev/ttyACM0`)
2. **Second:** Plug in Motor Arduino (gets `/dev/ttyACM1`)
3. Connect RealSense cameras to USB 3.0 ports
4. Power on motor controllers
5. Verify serial ports: `ls /dev/ttyACM*` (should show ACM0 and ACM1)

7.1.4 Network Setup

1. Connect the rover (Jetson) to the team WiFi network
2. Connect the base station (Linux laptop/Jetson) to the **same WiFi network**
3. Verify connectivity: `ping <rover-ip-address>`

7.2 Option A: Full System Startup

Important

This option requires both the rover (Jetson) and base station (local machine) to be on the same WiFi network. The base station **must be running Linux** to communicate with the rover—use a Linux laptop or another Jetson with `jetson_local`.

7.2.1 Step 1: Start Jetson Container (on rover)

```
cd docker/jetson
sudo ./start.sh
```

Enter the container and start the ROS bridges:

```
# Enter the container
sudo docker exec -it pitt_urc_jetson bash

# Source ROS 2
source /opt/ros/humble/local_setup.bash
cd /app

# Start bridges (run each in separate terminal or use screen/tmux)
python3 -m ros_bridge.motor_subscriber.motor_subscriber
python3 -m ros_bridge.gps_bridge.gps_bridge
python3 -m ros_bridge.ultrasonic_bridge.ultrasonic_bridge
```

7.2.2 Step 2: Start Local Container (on base station)

```
# macOS
cd docker/local
./start_mac.sh

# Linux
cd docker/local
./start_linux.sh
```

7.2.3 Step 3: Launch GUI

```
# Inside local container
source /opt/ros/humble/local_setup.bash
cd /app
python3 -m guis.gen_gui
```

Chapter 8

Detailed Startup Procedures

8.1 Prerequisites

Warning

Linux Base Station Required: To communicate with the rover Jetson, you must use a Linux-based base station. This can be a laptop running Linux (Ubuntu 22.04 recommended) or another Jetson device running the `jetson_local` configuration.

8.1.1 USB Connection Order

Warning

USB Connection Order is Critical! The motor Arduino is hardcoded to `/dev/ttyACM1`. Linux assigns device numbers in the order USB devices are plugged in.

1. **First:** Plug in IMU/GPS Arduino (gets `/dev/ttyACM0`)
2. **Second:** Plug in Motor Arduino (gets `/dev/ttyACM1`)
3. Connect RealSense cameras to USB 3.0 ports
4. Power on motor controllers
5. Verify serial ports: `ls /dev/ttyACM*` (should show ACM0 and ACM1)

8.1.2 Network Setup

1. Connect the rover (Jetson) to the team WiFi network
2. Connect the base station (Linux laptop/Jetson) to the **same WiFi network**
3. Verify connectivity: `ping <rover-ip-address>`

8.2 Jetson On-Board Startup

```
# 1. SSH into Jetson
ssh nvidia@<jetson-ip>

# 2. Navigate to project
```

```
cd /path/to/URC

# 3. Build and start container (first time)
cd docker/jetson
sudo docker build -t urc_jetson .
sudo docker-compose up -d

# 4. Verify container is running
sudo docker ps

# 5. Enter the container
sudo docker exec -it pitt_urc_jetson bash

# 6. Source ROS 2 and navigate to app
source /opt/ros/humble/local_setup.bash
cd /app

# 7. Start ROS bridges (run each in separate terminal or use screen/tmux)
python3 -m ros_bridge.motor_subscriber.motor_subscriber
python3 -m ros_bridge.gps_bridge.gps_bridge
python3 -m ros_bridge.ultrasonic_bridge.ultrasonic_bridge
```

Listing 8.1: Complete Jetson Startup Procedure

8.3 Base Station Startup

8.3.1 macOS Procedure

```
# 1. Start XQuartz
open -a XQuartz

# 2. Enable network clients (if not done)
# XQuartz > Preferences > Security > Allow connections

# 3. Navigate to project
cd /path/to/URC/docker/local

# 4. Start container
./start_mac.sh

# 5. Source ROS 2
source /opt/ros/humble/local_setup.bash

# 6. Launch main GUI
cd /app
python3 -m guis.gen_gui
```

Listing 8.2: macOS Base Station Startup

8.3.2 Linux Procedure

```
# 1. Navigate to project
cd /path/to/URC/docker/local

# 2. Start container
./start_linux.sh

# 3. Source ROS 2
source /opt/ros/humble/local_setup.bash
```

```

# 4. Launch main GUI
cd /app
python3 -m guis.gen_gui
```

Listing 8.3: Linux Base Station Startup

8.4 Verification Steps

8.4.1 Check ROS 2 Connectivity

```
# List all active nodes
ros2 node list

# Expected:
# /motor_subscriber
# /gps_bridge
# /imu_bridge
# /ultrasonic_bridge

# List all topics
ros2 topic list

# Expected:
# /cmd_vel
# /motor_control_input
# /drive_data
# /gps_data
# /imu_data
# /ultrasonic_data
# /camera/gray/image_raw
# /camera/depth/image_raw
```

Listing 8.4: ROS 2 Verification Commands

8.4.2 Test Motor Commands

Warning

Be careful when testing motor commands! Ensure the rover is in a safe position before sending movement commands.

```
# Send stop command
ros2 topic pub --once /motor_control_input std_msgs/String \
    "data: '0,0,0,0,0,0'"

# Send forward command (be careful!)
ros2 topic pub --once /motor_control_input std_msgs/String \
    "data: '100,0,0,0,0,0'"
```

Listing 8.5: Motor Command Testing

8.4.3 Monitor Sensor Data

```
# IMU data
ros2 topic echo /imu_data

```

```
# GPS data
ros2 topic echo /gps_data

# Ultrasonic data
ros2 topic echo /ultrasonic_data
```

Listing 8.6: Sensor Data Monitoring

Chapter 9

ROS 2 Architecture

9.1 Node Overview

Table 9.1: ROS 2 Nodes

Node	Package	Purpose
motor_bridge	ros_bridge	Motor command handling and feedback
gps_bridge	ros_bridge	GPS NMEA data publishing
imu_bridge	ros_bridge	IMU orientation data publishing
ultrasonic_bridge	ros_bridge	Distance measurement publishing
gray_and_depth_publisher	guis.camera	Camera feed publishing

9.2 Topic Architecture

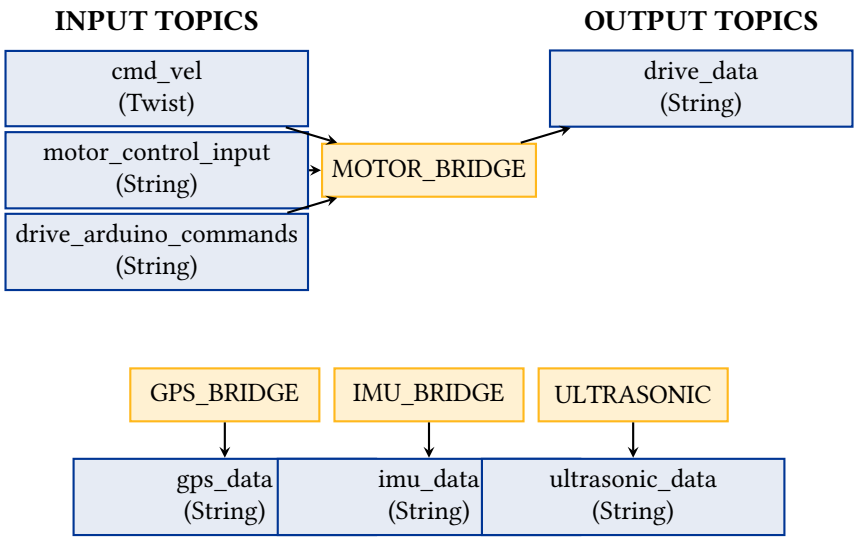

Figure 9.1: ROS 2 Topic Architecture

9.3 Message Formats

Important

Motor Control: Two Input Methods
The motor bridge accepts commands from two different topic types:

- `/cmd_vel` (geometry_msgs/Twist) — Standard ROS 2 velocity commands. Used by teleop_twist_keyboard, joystick drivers, and autonomous navigation nodes.
- `/motor_control_input` (std_msgs/String) — Direct 6-value control. Used by custom GUIs for fine-grained motor testing.

Both are converted to PWM signals by the MotorBridge. Use `/cmd_vel` for standard robot control; use `/motor_control_input` for direct hardware testing.

9.3.1 Motor Control Input (String)

```
Format: "linear_x,linear_y,linear_z,angular_x,angular_y,angular_z"
Example: "100,0,0,0,0,50"  # Forward with slight right turn
```

Listing 9.1: Motor Command Format

9.3.2 IMU Data (String)

```
Format: "X: <value>\tY: <value>\tZ: <value>"
Example: "X: 0.123\tY: 0.456\tZ: 0.789"
```

Listing 9.2: IMU Data Format

9.3.3 GPS Data (String)

```
NMEA sentences:
$GNGLL,4024.12345,N,07952.12345,W,123456.00,A,A*6B
$GAGSV,3,1,12,01,45,123,38,02,67,234,42,...
```

Listing 9.3: GPS NMEA Format

9.3.4 Ultrasonic Data (String)

```
Format: "distance1_cm, distance2_cm, distance3_cm, "
Example: "45, 120, 88, "
```

Listing 9.4: Ultrasonic Data Format

Chapter 10

Hardware Interfaces

10.1 Motor Control

10.1.1 Arduino Firmware

Location: `ros_bridge/motor_subscriber/motor_serial/motor_serial.ino`

```
// Simplified differential drive
left_speed = linear_x - angular_z;
right_speed = linear_x + angular_z;

// PWM mapping
pwm_left = map(left_speed, -100, 100, -255, 255);
pwm_right = map(right_speed, -100, 100, -255, 255);

// Direction Control
// HIGH = Forward
// LOW = Reverse
```

Listing 10.1: Motor Control Logic

10.2 IMU (BNO055)

Location: `ros_bridge/imu_bridge/imu_serial/imu_serial.ino`

```
Adafruit_BNO055 bno = Adafruit_BNO055(55);
bno.setExtCrystalUse(true);  // External crystal for accuracy

// Output Format
Serial.print("X: "); Serial.print(event.orientation.x);
Serial.print("\tY: "); Serial.print(event.orientation.y);
Serial.print("\tZ: "); Serial.println(event.orientation.z);
```

Listing 10.2: IMU Configuration

10.2.1 Data Processing

```
# IMUDataParser.py
import math

distance = math.sqrt(
    (x_cur - x_prev)**2 +
    (y_cur - y_prev)**2 +
    (z_cur - z_prev)**2
```

```
)
velocity = distance / time_delta
vertical_tilt = math.degrees(math.atan2(y_delta, z_delta))
horizontal_tilt = math.degrees(math.atan2(y_delta, x_delta))
```

Listing 10.3: IMU Data Parser

10.3 GPS

Location: `ros_bridge/gps_bridge/gps/gps.ino`

Table 10.1: NMEA Sentence Types

Type	Purpose
GNGLL	Geographic position (latitude/longitude)
GAGSV	GPS satellites in view + SNR
GBGSV	BeiDou satellites in view + SNR

10.4 Ultrasonic Sensors

Location: `ros_bridge/ultrasonic_bridge/multiple_ultrasonic_sensors/`

```
duration = pulseIn(echoPin, HIGH);
distance_cm = (duration / 2) / 29.1;
```

Listing 10.4: Ultrasonic Distance Calculation

10.5 RealSense Cameras

Location: `guis/camera/camera_cv_test.py`

The rover uses Intel RealSense D435 RGB-D cameras for vision and depth sensing.

10.5.1 Camera Specifications

Table 10.2: RealSense D435 Configuration

Stream	Resolution	Format
Color	640×480 @ 30 FPS	BGR8
Depth	640×480 @ 30 FPS	Z16 (16-bit depth)
Grayscale	640×480 @ 30 FPS	MONO8 (converted)

10.5.2 Published Topics

Table 10.3: Camera ROS Topics

Topic	Message Type	Description
/camera/gray/image_raw	sensor_msgs/Image	Grayscale image (mono8)
/camera/depth/image_raw	sensor_msgs/Image	Depth image (16UC1)

10.5.3 Configuration Code

```python
import pyrealsense2 as rs

# Initialize pipeline
pipeline = rs.pipeline()
config = rs.config()

# Color stream (converted to grayscale for publishing)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)

# Depth stream
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)

# Start streaming
pipeline.start(config)
```

Listing 10.5: RealSense Configuration

Tip

The camera publisher converts color frames to grayscale before publishing to reduce bandwidth. Connect cameras to USB 3.0 ports for optimal performance.

Chapter 11

GUI System

11.1 Main Control GUI

File: `guis/gen_gui.py` (623 lines)

Table 11.1: Main GUI Components

Section	Function
Title Bar	Connection status (ONLINE/OFFLINE)
Navigation Tabs	Switch between competition modes
Camera Feeds	3 video displays (primary, secondary, auxiliary)
IMU Display	Speed, vertical tilt, horizontal tilt
System Controls	Toggle IMU, GPS, orientation
Emergency Stop	Red button for immediate halt

```
python3 -m guis.gen_gui
```

Listing 11.1: Launch Main GUI

11.2 Motor/Arm GUI Hotkeys

File: `guis/json_motorGUI.py` (292 lines)

Table 11.2: Motor Control Hotkeys

Key	Action
I	Forward
, (comma)	Backward
L	Turn Right
J	Turn Left
Q	Speed Up
Z	Slow Down
K	Stop

Table 11.3: Arm Control Hotkeys

Key	Action
0/9	Claw open/close
M/N	Base shift right/left
U/J	Bottom joint forward/backward
I/K	Middle joint forward/backward
O/L	Top joint forward/backward
Y/H	Wrist clockwise/counterclockwise
Escape	Emergency stop all

11.3 Publisher Classes

Location: `guis/publishers/publisher.py`

```python
class MotorPublisher(GenericPublisher):
    def __init__(self):
        super().__init__('motor_control_input', String)

    def publish_motor_command(self, motor_values):
        # motor_values: list of 6 floats
        msg = String()
        msg.data = ','.join(map(str, motor_values))
        self.publisher.publish(msg)

    def stop_all_motors(self):
        self.publish_motor_command([0, 0, 0, 0, 0, 0])
```

Listing 11.2: MotorPublisher Class

```python
class TwistPublisher(GenericPublisher):
    def __init__(self):
        super().__init__('cmd_vel', Twist)

    def move_forward(self, speed):
        msg = Twist()
        msg.linear.x = float(speed)
        self.publisher.publish(msg)

    def turn_left(self, angular_speed):
        msg = Twist()
        msg.angular.z = float(angular_speed)
        self.publisher.publish(msg)
```

Listing 11.3: TwistPublisher Class

Chapter 12

Development Workflow

12.1 Git Workflow

12.1.1 Branch Strategy

- `main` — Protected, requires PR review
- `feature/*` — New features
- `bugfix/*` — Bug fixes
- `hotfix/*` — Urgent production fixes

12.1.2 Standard Workflow

```
# 1. Fetch latest
git fetch origin

# 2. Create feature branch
git checkout -b feature/my-feature main

# 3. Make changes
# ... edit files ...

# 4. Stage and commit
git add .
git commit -m "feat: add my feature description"

# 5. Push to remote
git push origin feature/my-feature

# 6. Create Pull Request on GitHub
```

Listing 12.1: Git Development Workflow

12.1.3 Commit Message Format

```
type: description

Types:
- feat: New feature
- fix: Bug fix
- docs: Documentation
- refactor: Code refactoring
```

```
- test: Testing
- chore: Maintenance
```

Listing 12.2: Commit Message Types

12.2 Adding a New Sensor Bridge

12.2.1 Step 1: Create Arduino Firmware

```
// ros_bridge/new_sensor_bridge/new_sensor_serial/new_sensor_serial.ino
void setup() {
    Serial.begin(115200);
}

void loop() {
    // Read sensor
    float value = readSensor();

    // Send data
    Serial.println(value);
    delay(100);
}
```

Listing 12.3: New Sensor Arduino Code

12.2.2 Step 2: Create Python Bridge

```
# ros_bridge/new_sensor_bridge/new_sensor_bridge.py
from ros_bridge.arduino_bridge_base.arduino_bridge_base import ArduinoBridgeBase
from std_msgs.msg import String

class NewSensorBridge(ArduinoBridgeBase):
    def __init__(self):
        super().__init__(
            node_name='new_sensor_bridge',
            topic_name='new_sensor_data',
            msg_type=String,
            serial_port='/dev/ttyACM0',
            baud_rate=115200
        )
```

Listing 12.4: New Sensor Python Bridge

12.2.3 Step 3: Add to Supervisor

```
# docker/jetson/supervisord.conf
[program:new_sensor_bridge]
command=python3 /app/ros_bridge/new_sensor_bridge/new_sensor_bridge.py
stdout_logfile=/app/new_sensor_bridge.log
stderr_logfile=/app/new_sensor_bridge_err.log
autorestart=true
```

Listing 12.5: Supervisor Configuration Entry

Chapter 13

Troubleshooting

13.1 Common Issues

Issue	Cause	Solution
Serial port not found	Device not connected	Check USB connections, verify `/dev/ttyACM*` exists
Motors not responding	Wrong port or baud rate	Verify `/dev/ttyACM1`, check 115200 baud
ROS topics not visible	Domain ID mismatch	Ensure `ROS_DOMAIN_ID=0` on both systems
GUI won't display	X11 forwarding issue	Check `DISPLAY` variable, restart XQuartz
Camera black screen	USB bandwidth	Use USB 3.0 ports, reduce resolution
High latency	Network congestion	Check WiFi signal, reduce message frequency

Table 13.1: Common Issues and Solutions

13.2 Serial Port Debugging

```
# List all serial ports
ls -la /dev/ttyACM* /dev/ttyUSB*

# Check port permissions
sudo chmod 666 /dev/ttyACM0

# Test serial communication
screen /dev/ttyACM0 115200

# Kill screen session: Ctrl+A, then K
```

Listing 13.1: Serial Port Debugging Commands

13.3 ROS 2 Debugging

For ROS 2 debugging commands (checking environment, sourcing, discovery verification), see Appendix B: Quick Command Reference.

13.4 Docker Debugging

```
# Check container status
docker ps -a

# View container logs
docker logs pitt_urc_jetson

# Enter running container
docker exec -it pitt_urc_jetson bash

# Check supervisor status (inside Jetson container)
supervisorctl status
supervisorctl restart motor_subscriber
```

Listing 13.2: Docker Debugging Commands

13.5 Log File Locations

Table 13.2: Log File Locations

Log	Location	Content
Motor Bridge	/app/motor_bridge.log	Motor commands, errors
GPS Bridge	/app/gps_bridge.log	NMEA sentences
Ultrasonic	/app/ultrasonic_bridge.log	Distance readings
Setup	/app/setup.log	Camera initialization

Chapter 14

API Reference

14.1 ArduinoBridgeBase

```python
class ArduinoBridgeBase(Node):
    """Base class for all Arduino serial bridges."""

    def __init__(self, node_name, topic_name, msg_type,
                 serial_port='/dev/ttyACM0', baud_rate=115200):
        """
        Initialize the bridge.

        Args:
            node_name: ROS 2 node name
            topic_name: Topic to publish/subscribe
            msg_type: ROS 2 message type (e.g., String)
            serial_port: Serial port path
            baud_rate: Baud rate (default: 115200)
        """

    def read_from_arduino(self):
        """Read data from Arduino and publish to ROS topic."""

    def write_to_arduino(self, data):
        """Write data to Arduino serial port."""
```

Listing 14.1: ArduinoBridgeBase API

Tip

For MotorPublisher and TwistPublisher API documentation, see the Publisher Classes section in Chapter 11 (GUI System).

Appendix A

Environment Variables

Table A.1: Environment Variable Reference

Variable	Default	Description
ROS_DOMAIN_ID	0	ROS 2 domain for multi-device networking
ROS_LOCALHOST_ONLY	0	Allow network communication (0=yes, 1=no)
DISPLAY	:0	X11 display for GUI
QT_X11_NO_MITSHM	1	Qt X11 compatibility
NVIDIA_VISIBLE_DEVICES	all	GPU access (Jetson)
NVIDIA_DRIVER_CAPABILITIES	all	GPU capabilities (Jetson)

Appendix B

Quick Command Reference

```
# Docker
docker build -t urc_jetson .
docker-compose up -d
docker exec -it pitt_urc_jetson bash
docker logs pitt_urc_jetson

# ROS 2
source /opt/ros/humble/local_setup.bash
ros2 node list
ros2 topic list
ros2 topic echo /topic_name
ros2 topic pub --once /topic std_msgs/String "data: 'test'"

# GUI
python3 -m guis.gen_gui
python3 -m guis.arduino_gui
python3 -m guis.json_motorGUI

# Supervisor (inside Jetson container)
supervisorctl status
supervisorctl restart motor_subscriber
supervisorctl tail -f motor_subscriber

# Serial
ls /dev/ttyACM*
screen /dev/ttyACM0 115200
```

Listing B.1: Quick Command Reference

Appendix C

Network Ports

Table C.1: Network Port Reference

Port	Protocol	Service
7400	UDP	ROS 2 DDS Discovery
8000	HTTP	Motor/Arm GUI server
11311	TCP	ROS Master (legacy)

Document Information

Field	Value
Document Version	2.0
Last Updated	January 2026
Maintained by	University of Pittsburgh Robotics Club
Repository	github.com/pitt-robotics/URC

For questions or contributions, please refer to `CONTRIBUTING.md` or open an issue on GitHub.