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 High-Level Architecture

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

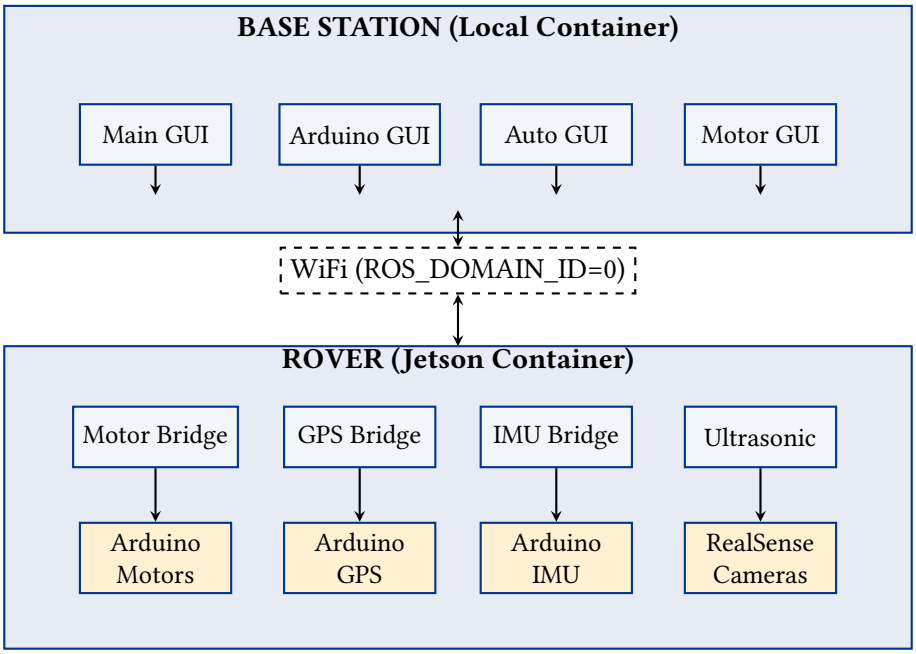

Figure 2.1: System Architecture Overview

2.2 Dual-Container Architecture

Table 2.1: Container Responsibilities

Container	Location	Purpose
Jetson Container	On-board rover	Hardware bridges, sensor acquisition, motor control, camera feeds
Local Container	Base station	GUI interfaces, visualization, high-level control, debugging

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.

2.3 Communication Flow

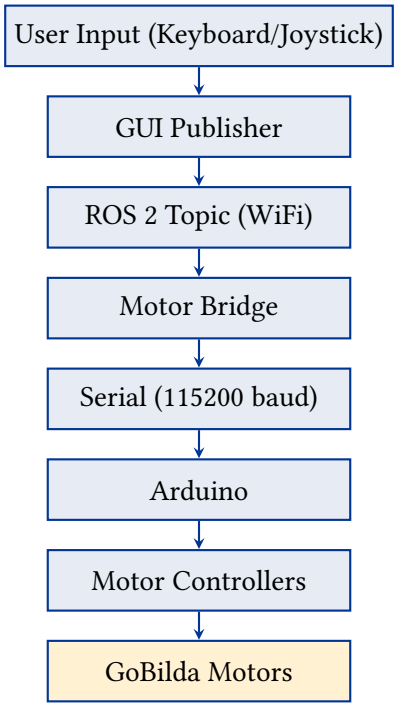

Figure 2.2: Motor Control Communication Flow

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/     # Legacy motor control
|   |-- 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
|
|-- simulation/               # Gazebo simulation
|   +-- colcon_ws/            # ROS 2 workspace
|
|-- 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 On-Board Computer

Table 4.1: 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)

4.2 Motor System

Table 4.2: Motor System Specifications

Component	Specification
Motors	6x GoBilda motors
Motor Controllers	3x dual H-bridge controllers
Arduino	Arduino Mega 2560 (recommended)
PWM Range	0–255
Wheel Base	0.7384 meters
Max Speed	1.0 m/s

4.2.1 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

4.3 Sensors

Table 4.4: Sensor Configuration

Sensor	Model	Connection	Rate
IMU	Adafruit BNO055 (9-axis)	Serial 115200	10 Hz
GPS	RTK GPS module	Serial 115200	1 Hz
Cameras	2x Intel RealSense D435	USB 3.0	30 Hz
Ultrasonic	3x HC-SR04	Arduino GPIO	10 Hz

4.3.1 Ultrasonic Sensor Pins

Table 4.5: Ultrasonic Sensor Pin Configuration

Sensor	Trigger Pin	Echo Pin
Sensor 1	8	9
Sensor 2	50	51
Sensor 3	48	49

4.4 Serial Port Configuration

Table 4.6: Serial Port Assignments

Device	Port	Baud Rate	Notes
Motors	/dev/ttyACM1	115200	Hardcoded in MotorBridge
GPS	/dev/ttyACM0	115200	Configurable
IMU	Auto-detect	115200	Tries multiple ports
Ultrasonic	Arduino internal	115200	No separate port

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

```
environment:
  - ROS_DOMAIN_ID=0
  - ROS_LOCALHOST_ONLY=0
  - NVIDIA_DRIVER_CAPABILITIES=all
  - NVIDIA_VISIBLE_DEVICES=all
```

Listing 6.3: Jetson Environment Variables

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_bridge	Motor command handler	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 6.4: macOS Container Startup

6.2.2 Linux Setup

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

Listing 6.5: Linux Container Startup

6.2.3 Local Environment Variables

```
environment:
  - ROS_DOMAIN_ID=0
  - ROS_LOCALHOST_ONLY=0
  - QT_X11_NO_MITSHM=1
  - DISPLAY=${DISPLAY}
```

Listing 6.6: Local Container Environment

6.3 Simulation Container

```
cd docker/local
./sim-launch.sh

# Launch simulation
ros2 launch my_robot_description display.launch.py use_sim_time:=true
```

Listing 6.7: Simulation Startup

Chapter 7

Quick Start Guide

7.1 Prerequisites

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

1. Connect Arduino to USB port
2. Connect RealSense cameras to USB 3.0 ports
3. Power on motor controllers
4. Verify serial ports: `ls /dev/ttyACM*`

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.

7.2.1 Step 1: Start Jetson Container (on rover)

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

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
```

7.3 Option B: Simulation Only

```
cd docker/local
./sim-launch.sh

# In another terminal (inside container)
ros2 launch my_robot_description display.launch.py use_sim_time:=true
```

7.4 Option C: Development Mode

```
# Start local container
cd docker/local
./start_linux.sh

# Inside container
source /opt/ros/humble/local_setup.bash
cd /app

# Run individual components
python3 ros_bridge/motor_bridge/launch_motor_bridge.py
python3 -m guis.arduino_gui
```

Chapter 8

Detailed Startup Procedures

8.1 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. Check supervised processes
sudo docker exec -it pitt_urc_jetson supervisorctl status

# Expected output:
# gps_bridge          RUNNING   pid 123, uptime 0:01:00
# motor_bridge        RUNNING   pid 124, uptime 0:01:00
# setup               RUNNING   pid 125, uptime 0:01:00
# ultrasonic_bridge   RUNNING   pid 126, uptime 0:01:00

# 6. View logs
sudo docker exec -it pitt_urc_jetson tail -f /app/motor_bridge.log
```

Listing 8.1: Complete Jetson Startup Procedure

8.2 Base Station Startup

8.2.1 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

8.2.2 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

8.3 Verification Steps

8.3.1 Check ROS 2 Connectivity

```
# List all active nodes
ros2 node list

# Expected:
# /motor_bridge
# /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

8.3.2 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

8.3.3 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

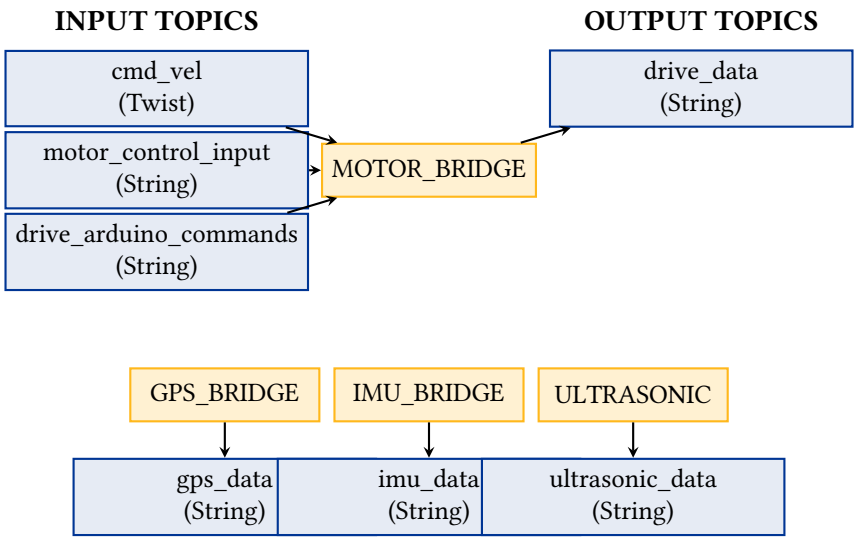

Figure 9.1: ROS 2 Topic Architecture

9.3 Message Formats

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

9.4 Bridge Class Hierarchy

```
ArduinoBridgeBase(Node)          # Base class (248 lines)
    |-- MotorBridge              # Motor control
    |-- GPSBridge                # GPS data
    |-- IMUBridge                # IMU data
    +-- UltrasonicBridge         # Ultrasonic data
```

Listing 9.5: Bridge Class Structure

Base Class Features:

- Serial port initialization (115200 baud)
- Generic ROS 2 topic subscription/publishing
- Timer-based Arduino reading (100ms default)
- Error handling and logging
- Automatic reconnection attempts

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_untrasonic_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`

```
import pyrealsense2 as rs

# Color stream
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)
```

Listing 10.5: RealSense Configuration

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`

```
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

```
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

```
# Check ROS environment
echo $ROS_DOMAIN_ID
echo $ROS_LOCALHOST_ONLY

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

# Verify discovery
ros2 daemon status
ros2 daemon start

# Check multicast
ros2 multicast receive
```

Listing 13.2: ROS 2 Debugging Commands

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_bridge
```

Listing 13.3: 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

14.2 MotorPublisher

```python
class MotorPublisher(GenericPublisher):
    """Publisher for motor control commands."""

    def publish_motor_command(self, motor_values: List[float]):
        """
        Publish motor command.

        Args:
            motor_values: List of 6 float values
                          [linear_x, linear_y, linear_z,
                           angular_x, angular_y, angular_z]
        """

    def set_motor_value(self, index: int, value: float):
        """Set a specific motor value (0-5)."""
```

```
16
17      def set_all_motors(self, value: float):
18          """Set all motors to the same value."""
19
20      def stop_all_motors(self):
21          """Emergency stop - set all motors to 0."""
```

Listing 14.2: MotorPublisher API

14.3 TwistPublisher

```
1  class TwistPublisher(GenericPublisher):
2      """Publisher for geometry_msgs/Twist messages."""
3
4      def publish_twist(self, linear_x=0, linear_y=0, linear_z=0,
5                        angular_x=0, angular_y=0, angular_z=0):
6          """Publish a full Twist message."""
7
8      def move_forward(self, speed: float):
9          """Move forward at specified speed."""
10
11     def move_backward(self, speed: float):
12         """Move backward at specified speed."""
13
14     def turn_left(self, angular_speed: float):
15         """Turn left at specified angular speed."""
16
17     def turn_right(self, angular_speed: float):
18         """Turn right at specified angular speed."""
19
20     def stop(self):
21         """Stop all motion."""
```

Listing 14.3: TwistPublisher API

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_bridge
supervisorctl tail -f motor_bridge

# 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.