Project Astra NZ - Complete Setup and Implementation Guide

This guide provides step-by-step instructions to implement the complete Project Astra agricultural rover system from hardware setup to autonomous operation.

Prerequisites

Hardware Requirements

- **Pixhawk 6C** with ArduPilot Rover firmware
- **Ubuntu companion computer** (Raspberry Pi 4+ or equivalent)
- **RPLidar S3** with USB connection
- **Intel RealSense D435i** camera
- **SimpleRTK2B Budget** GPS module
- **WiFi connectivity** for telemetry
- **Power systems** for all components

Software Requirements

- Ubuntu 20.04+ on companion computer
- Python 3.8+ with pip
- Git for repository management
- Mission Planner on ground station computer

Step 1: Initial System Setup

1.1 Clone Project Repository

```bash
# Navigate to home directory
cd ~

# Clone Project Astra repository
git clone https://github.com/ProjectAstraNZ/ProjectAstraNZ.git
cd ProjectAstraNZ

# Create rover-scripts directory if it doesn't exist
mkdir -p rover-scripts
cd rover-scripts
```

1.2 Hardware Connections

RPLidar S3:

- Connect to Ubuntu computer via USB
- Should appear as `/dev/ttyUSB0`
- Ensure 5V power supply is adequate

Intel RealSense D435i:

- Connect to Ubuntu computer via USB 3.0
- Blue USB port recommended for bandwidth
- Verify with `lsusb | grep Intel`

Pixhawk 6C:

- Connect to Ubuntu computer via USB
- Should appear as `/dev/ttyACM0` or similar
- Verify ArduPilot Rover firmware is installed

SimpleRTK2B Budget:

- Connect to Pixhawk UART port (GPS1 recommended)
- Configure for RTK corrections (NTRIP or base station)

1.3 User Permissions Setup

```bash
# Add user to dialout group for serial access
sudo usermod -aG dialout $USER

# Set device permissions (run after each reboot)
sudo chmod 666 /dev/ttyUSB0  # RPLidar
sudo chmod 666 /dev/ttyACM0  # Pixhawk

# Logout and login for group changes to take effect
```

Step 2: Python Environment Setup

2.1 Create Virtual Environment

```bash
```

```
# Create virtual environment for Project Astra
python3 -m venv ~/rover_venv

# Activate virtual environment
source ~/rover_venv/bin/activate

# Upgrade pip
pip install --upgrade pip
```

2.2 Install Required Libraries

bash

```
# Core sensor libraries
pip install rplidar
pip install pymavlink
pip install pyrealsense2

# Computer vision and analysis
pip install opencv-python
pip install numpy

# Web interface (optional)
pip install flask

# Additional utilities
pip install matplotlib
pip install pillow
```

2.3 Verify Installations

bash

```
# Test RPLidar library
python3 -c "import rplidar; print('RPLidar library OK')"

# Test RealSense library
python3 -c "import pyrealsense2; print('RealSense library OK')"

# Test MAVLink library
python3 -c "import pymavlink; print('MAVLink library OK')"

# Test OpenCV library
python3 -c "import cv2; print('OpenCV library OK')"
```

Step 3: Create Core System Scripts

3.1 Hardware Detection Script

bash

```
cat > 0_hardware_check.py << 'EOF'
#!/usr/bin/env python3
"""
Hardware Detection and Verification Script
Checks all sensors and connections before system startup
"""

import os
import sys
import time

try:
    import pyrealsense2 as rs
    REALSENSE_AVAILABLE = True
except ImportError:
    REALSENSE_AVAILABLE = False

try:
    from rplidar import RPLidar
    RPLIDAR_AVAILABLE = True
except ImportError:
    RPLIDAR_AVAILABLE = False

try:
    from pymavlink import mavutil
    MAVLINK_AVAILABLE = True
except ImportError:
    MAVLINK_AVAILABLE = False

def check_rplidar():
    print("Checking RPLidar S3...")
    if not RPLIDAR_AVAILABLE:
        print("  ✗ RPLidar library not installed")
        return False

    if not os.path.exists('/dev/ttyUSB0'):
        print("  ✗ RPLidar not found at /dev/ttyUSB0")
        return False

    try:
        lidar = RPLidar('/dev/ttyUSB0', baudrate=1000000, timeout=2)
        info = lidar.get_info()
        health = lidar.get_health()
        lidar.disconnect()

        print(f"  ✓ RPLidar S3 connected - Model: {info['model']}, Health: {health[0]}")
```

```
        return True
    except Exception as e:
        print(f"  ✗ RPLidar connection failed: {e}")
        return False

def check_realsense():
    print("Checking Intel RealSense...")
    if not REALSENSE_AVAILABLE:
        print("  ✗ RealSense library not installed")
        return False

    try:
        pipeline = rs.pipeline()
        config = rs.config()
        config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
        config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)

        pipeline.start(config)

        # Test frame capture
        for _ in range(5):
            frames = pipeline.wait_for_frames(timeout_ms=1000)
            if frames.get_color_frame() and frames.get_depth_frame():
                break

        pipeline.stop()
        print("  ✓ RealSense D435i connected and streaming")
        return True
    except Exception as e:
        print(f"  ✗ RealSense connection failed: {e}")
        return False

def check_pixhawk():
    print("Checking Pixhawk connection...")
    if not MAVLINK_AVAILABLE:
        print("  ✗ MAVLink library not installed")
        return False

    # Try to find Pixhawk device
    pixhawk_device = None

    # Check by-id first
    by_id_path = '/dev/serial/by-id'
    if os.path.exists(by_id_path):
        for device in os.listdir(by_id_path):
            if 'Pixhawk' in device or 'Holybro' in device:
                pixhawk_device = os.path.join(by_id_path, device)
```

```
            break

    # Check ttyACM devices
    if not pixhawk_device:
        for i in range(10):
            device = f'/dev/ttyACM{i}'
            if os.path.exists(device):
                pixhawk_device = device
                break

    if not pixhawk_device:
        print("  ✗ Pixhawk device not found")
        return False

    try:
        mavlink = mavutil.mavlink_connection(pixhawk_device, baud=57600)
        mavlink.wait_heartbeat(timeout=10)
        mavlink.close()

        print(f"  ✓ Pixhawk connected at {pixhawk_device}")
        return True
    except Exception as e:
        print(f"  ✗ Pixhawk connection failed: {e}")
        return False

def main():
    print("Project Astra Hardware Check")
    print("=" * 40)

    rplidar_ok = check_rplidar()
    realsense_ok = check_realsense()
    pixhawk_ok = check_pixhawk()

    print("\nSummary:")
    print(f"RPLidar S3:      {'✓' if rplidar_ok else '✗'}")
    print(f"RealSense D435i: {'✓' if realsense_ok else '✗'}")
    print(f"Pixhawk 6C:      {'✓' if pixhawk_ok else '✗'}")

    if rplidar_ok and realsense_ok and pixhawk_ok:
        print("\n🎉 All hardware detected successfully!")
        print("System ready for Project Astra operation")
        return True
    else:
        print("\n⚠️  Hardware issues detected")
        print("Please resolve issues before proceeding")
        return False
```

```
if __name__ == "__main__":
    main()
EOF

chmod +x 0_hardware_check.py
```

3.2 Main Proximity Bridge Script

```
bash

# Copy your working proximity bridge
cp ../2_combo_proximity_bridge_fixed.py ./2_combo_proximity_bridge_fixed.py
chmod +x 2_combo_proximity_bridge_fixed.py
```

3.3 Row Following System Script

```
bash

# Copy the row following script from previous artifacts
cp ../8_row_following_system.py ./8_row_following_system.py
chmod +x 8_row_following_system.py
```

3.4 Crop Monitoring Script

```
bash

# Copy the crop monitoring script from previous artifacts
cp ../9_crop_monitoring_system.py ./9_crop_monitoring_system.py
chmod +x 9_crop_monitoring_system.py
```

Step 4: ArduPilot Configuration

4.1 Required Parameters

Connect to Pixhawk with Mission Planner and set these parameters:

```
# Proximity Sensor Configuration
PRX1_TYPE = 2          # MAVLink proximity sensor
AVOID_ENABLE = 7       # All avoidance sources enabled
PRX1_ORIENT = 0        # Forward facing orientation

# GPS Configuration (for SimpleRTK2B)
GPS_TYPE = 1           # Auto detect
SERIAL3_PROTOCOL = 5   # GPS protocol (if using GPS1 port)
SERIAL3_BAUD = 230400  # High baud rate for RTK data

# Vehicle Configuration
FRAME_CLASS = 1        # Rover
SERVO1_FUNCTION = 26   # Ground steering (front motors)
SERVO3_FUNCTION = 70   # Throttle (rear motor)

# Navigation Parameters
NAVL1_PERIOD = 8       # Navigation L1 period
WP_RADIUS = 2          # Waypoint radius (meters)
SPEED_TURN_GAIN = 50   # Turn speed gain
```

4.2 Verify Configuration

In Mission Planner:

1. Go to Flight Data tab
2. Check GPS status shows RTK Float/Fixed when corrections available
3. Press Ctrl+F → Proximity to verify proximity sensor display
4. Check Parameters list to confirm all values set correctly

Step 5: Testing and Validation

5.1 Hardware Validation

```bash
# Activate virtual environment
source ~/rover_venv/bin/activate

# Run hardware check
python3 0_hardware_check.py
```

Expected output:

```
Project Astra Hardware Check
========================================
Checking RPLidar S3...
  ✓ RPLidar S3 connected - Model: 129, Health: Good
Checking Intel RealSense...
  ✓ RealSense D435i connected and streaming
Checking Pixhawk connection...
  ✓ Pixhawk connected at /dev/serial/by-id/usb-Holybro_Pixhawk6C_...

Summary:
RPLidar S3:      ✓
RealSense D435i: ✓
Pixhawk 6C:      ✓

🎉 All hardware detected successfully!
System ready for Project Astra operation
```

5.2 Proximity System Test

```bash
# Start proximity bridge
python3 2_combo_proximity_bridge_fixed.py
```

In Mission Planner:

1. Connect via UDP port 14550
2. Go to Flight Data → Press Ctrl+F → Proximity
3. Verify 8-sector radar display shows real-time obstacle data
4. Test by moving objects around sensors

5.3 Row Detection Test

Create simple test rows:

```bash
# Method 1: Tape lines on floor
# Use green painter's tape, 50cm apart

# Method 2: Garden/lawn edges
# Use existing landscaping features

# Start row detection
python3 8_row_following_system.py
```

Move camera over test rows and verify terminal output shows:

- "ROWS DETECTED" with lateral offset and heading error
- Steering guidance commands
- Detection rate statistics

5.4 Crop Monitoring Test

```
bash

# Start crop monitoring (saves images every 10 seconds)
python3 9_crop_monitoring_system.py
```

Check output directory for:

- Original images in `/images/` folder
- Analyzed images in `/analysis/` folder
- Text reports with crop health data

Step 6: Integrated System Operation

6.1 Full System Startup

Terminal 1 - Proximity Sensors:

```
bash

source ~/rover_venv/bin/activate
cd ~/ProjectAstraNZ/rover-scripts
python3 2_combo_proximity_bridge_fixed.py
```

Terminal 2 - Row Following (Optional):

```
bash

source ~/rover_venv/bin/activate
cd ~/ProjectAstraNZ/rover-scripts
python3 8_row_following_system.py
```

Terminal 3 - Crop Monitoring (Optional):

```
bash
```

```
source ~/rover_venv/bin/activate
cd ~/ProjectAstraNZ/rover-scripts
python3 9_crop_monitoring_system.py
```

6.2 Mission Planner Monitoring

1. **Connect**: UDP, port 14550
2. **Flight Data**: Monitor rover position and status
3. **Proximity Display**: Ctrl+F → Proximity for 8-sector view
4. **Parameters**: Verify RTK GPS status and proximity parameters

Step 7: Field Deployment Preparation

7.1 RTK GPS Setup

Base Station Configuration:

- Position base station with clear sky view
- Configure NTRIP corrections or local base
- Verify correction data transmission

Rover GPS Verification:

- Check GPS status in Mission Planner
- Confirm RTK Float/Fixed status
- Verify centimeter-level accuracy

7.2 Field Testing Protocol

1. **Static Testing**: Verify all sensors in field environment
2. **Manual Movement**: Push/carry rover through crop rows
3. **Proximity Validation**: Test obstacle detection with field objects
4. **Row Detection**: Verify camera can detect actual crop rows
5. **GPS Accuracy**: Confirm RTK positioning performance

7.3 Autonomous Operation Checklist

- [ ] All hardware tests pass
- [ ] Mission Planner shows proximity data
- [ ] RTK GPS achieving Fixed status
- [ ] Row detection working on actual crops
- [ ] Obstacle avoidance tested and verified
- [ ] Emergency stop procedures established

- [ ] Field boundaries defined in mission planner

Step 8: Maintenance and Troubleshooting

8.1 Common Issues and Solutions

RPLidar Communication Errors:

```bash
# If seeing "descriptor starting bytes" errors
# Run debug version to monitor success rate
python3 2_combo_proximity_bridge_debug.py
```

RealSense Connection Issues:

- Ensure USB 3.0 connection (blue port)
- Check power supply adequacy
- Verify no other applications using camera

Mission Planner No Proximity Data:

- Verify ArduPilot parameters: PRX1_TYPE=2, AVOID_ENABLE=7
- Check MAVLink connection and baud rates
- Restart proximity bridge script

Row Detection Poor Performance:

- Adjust HSV color ranges for crop type
- Verify adequate lighting conditions
- Check camera angle and height above ground

8.2 Performance Monitoring

Monitor these key metrics:

- **RPLidar Success Rate**: Should be >90%
- **RealSense Reliability**: Should be 100%
- **Row Detection Rate**: Should be >80% in good conditions
- **RTK GPS Status**: Should maintain Fixed when corrections available

8.3 Log Analysis

System Logs:

```bash
# Check system logs for errors
journalctl -u rover-service --since "1 hour ago"

# Monitor resource usage
htop
```

Mission Planner Logs:

- Review telemetry logs for navigation performance
- Check proximity sensor data for anomalies
- Analyze GPS accuracy and RTK status

Step 9: Advanced Configuration

9.1 Auto-Start Services

Create systemd services for automatic startup:

```bash
# Create proximity service
sudo tee /etc/systemd/system/proximity-bridge.service << EOF
[Unit]
Description=Project Astra Proximity Bridge
After=network.target

[Service]
Type=simple
User=$USER
WorkingDirectory=/home/$USER/ProjectAstraNZ/rover-scripts
Environment=PATH=/home/$USER/rover_venv/bin
ExecStart=/home/$USER/rover_venv/bin/python3 2_combo_proximity_bridge_fixed.py
Restart=always
RestartSec=5

[Install]
WantedBy=multi-user.target
EOF

# Enable and start service
sudo systemctl daemon-reload
sudo systemctl enable proximity-bridge.service
sudo systemctl start proximity-bridge.service
```

9.2 Web Monitoring Interface

```bash
# Create web dashboard for remote monitoring
python3 realsense_web_viewer.py
# Access via http://rover-ip:5000
```

9.3 Data Logging

Configure automatic data collection:

- GPS tracks for field coverage analysis
- Crop monitoring time-series data
- System performance metrics
- Proximity sensor logs for safety analysis

Conclusion

This guide provides complete implementation instructions for Project Astra NZ agricultural rover system. The modular design allows incremental deployment:

1. **Basic Operation**: Proximity sensors + Mission Planner monitoring
2. **Agricultural Mode**: Add row following for autonomous navigation
3. **Research Platform**: Include crop monitoring for data collection
4. **Production System**: Full integration with RTK GPS and automated operation

Each component can be tested independently before full system integration, ensuring reliable operation in agricultural environments.

For technical support and updates, refer to the Project Astra NZ repository and documentation.