Robust Return-to-QR Navigation for Unitree Go2 Robot

PhD Progress Report

February 6, 2026

1 Overview

This report presents an in-depth exploration of a navigation system implemented for the Unitree Go2 quadruped robot. The primary goal is to enable the robot to return to a previously visited position identified via a QR code. This involves pose capture upon detection and closed-loop control based purely on odometry and onboard inertial data, without relying on a pre-built map or full SLAM pipeline. The resulting system is lightweight, low-latency, and suitable for constrained indoor environments.

2 System Architecture

2.1 Hardware Stack

- **Unitree Go2 Robot** with Sport Mode enabled
- **Onboard camera** with QR detection (using OpenCV externally)
- **Ubuntu 22.04 Host PC**, Ethernet interface (e.g., `enp58s0`)
- **No LiDAR or SLAM required** — purely odometry-based control

2.2 Software Dependencies

- **unitree_sdk2**: Official SDK for Unitree robots
- **CycloneDDS**: Data Distribution Service used for low-latency message passing
- **SportClient API**: Provides access to motion primitives (velocity control, stopping, obstacle avoidance)
- `<cmath>, <mutex>, <atomic>, <fstream>, <unistd.h>`: Used for quaternion math, concurrency and timing

3 QR Detection and Pose Capture

QR detection is handled by an external OpenCV-based vision module. When a QR code is recognized, a signal file `/tmp/qr_detected.txt` is created. The pose logging program continuously subscribes to `rt/sportmodestate` topic using DDS and, upon signal detection, logs the robot's pose in odometry frame.

```
void HighStateHandler(const void* message) {
    const auto* state = static_cast<const unitree_go::msg::dds_::
   SportModeState_*>(message);
    if (!std::filesystem::exists("/tmp/qr_detected.txt")) return;

    std::ofstream out("/tmp/robot_position_on_qr.txt");
    out << state->position()[0] << " " << state->position()[1] << " "
        << state->imu_state().quaternion()[0] << " "
        << state->imu_state().quaternion()[1] << " "
        << state->imu_state().quaternion()[2] << " "
        << state->imu_state().quaternion()[3] << std::endl;
    out.close();
}
```

Listing 1: Logging Pose Upon QR Detection

Note: The quaternion is later used to recover heading (ψ) using the standard ZYX Euler conversion:

$$\psi = \text{atan2}(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2))$$

4 Navigation Algorithm

4.1 Core Control Strategy

We treat return navigation as a relative control task in 2D (x, y, ψ). The control laws are as follows:

$$d = \sqrt{(x_t - x)^2 + (y_t - y)^2}$$
$$\theta_{err} = \text{wrapToPi}(\text{atan2}(y_t - y, x_t - x) - \psi)$$

where x_t, y_t are target QR coordinates and ψ is current yaw.

Velocity commands are computed by:

$$v_x = \text{clamp}(K_d \cdot d, 0, v_{x,max})$$
$$\omega_z = \text{clamp}(K_\theta \cdot \theta_{err}, -\omega_{max}, \omega_{max})$$

4.2 Control Heuristics

- Robot slows translationally when angular misalignment $|\theta_{err}| > \frac{\pi}{6}$
- Obstacle detection is inferred from low progress $\Delta d < \varepsilon$ over n iterations
- Backup logic: move backward for a brief time to recover from stuck state

4.3 Recovery Mode and Termination

If within tolerance $d < 0.1$ and $|\psi - \psi_t| < 10°$, the robot halts via 'StopMove()' command. All motion is issued through:

```
sport.Move(vx, 0, vyaw);
```

5 Full Return Navigation Code

The implementation combines quaternion parsing, heading control, velocity clamping, and recovery strategies. Below is the main loop excerpt:

```
double dist = sqrt(dx*dx + dy*dy);
double heading_err = wrapPi(desired_heading - yaw);

if (dist < pos_tol && fabs(yaw_err) < yaw_tol) {
    sport.Move(0,0,0);
    break;
}

if (no_progress_detected) {
    sport.Move(backup_vx, 0, 0);
    usleep(backup_ms * 1000);
}

sport.Move(vx, 0, vyaw);
```

Listing 2: Excerpt from Navigation Loop

6 Experimental Observations

- Navigation consistently converges to within 10 cm and 10° in lab testing
- Loop frequency: 50 Hz with `usleep(20000)` delay
- Performance depends on stable odometry and accurate quaternion orientation
- Tested successfully in presence of furniture and human motion

7 Next Steps

1. Parse target pose from `/tmp/robot_position_on_qr.txt` dynamically
2. Improve angular stability using integral control
3. Add keyboard interruption and safety features
4. Consider replacing manual logic with Nav2 if localization improves
5. Develop a 2D costmap for reactive avoidance

8 Conclusion

We present a mapless, feedback-based navigation system for the Unitree Go2 robot to return to a previously seen QR-coded location. Using quaternion math, closed-loop heading control, and minimal sensors, we demonstrate reliable indoor positioning. This work lays a strong foundation for lightweight autonomous routines in GPS-denied environments.