ATLAS: Autonomous Traffic Light And Sign System
A ROS2-based Implementation for Automated Navigation

Alberto Finardi, Federico Romano, Francesco De Vito

***Abstract*—This paper presents ATLAS (Autonomous Traffic Light And Sign System), a comprehensive ROS2-based solution for autonomous robot navigation with traffic sign and signal recognition capabilities. The system implements robust line-following using computer vision and PID control, along with traffic element recognition using ArUco markers. We detail the architecture, explain the communication between distributed nodes, and analyze parameter sensitivity. Our implementation demonstrates the integration of perception, decision-making, and control for autonomous navigation in structured environments. The experimental results validate the system's ability to follow paths reliably while recognizing and responding appropriately to traffic signs and signals, providing a foundation for more complex autonomous navigation applications.**

I. Introduction

Autonomous vehicles must accurately perceive and respond to environmental elements including lane markings, traffic signs, and signals. The ATLAS project addresses these challenges through a scaled-down simulation implemented on a mobile robot platform. Using a RoboMaster S1 equipped with cameras and sensors, the system demonstrates fundamental capabilities required for autonomous navigation: path following using computer vision and PID control, traffic light and sign recognition, and decision-making and motion control.

Unlike many existing implementations, ATLAS integrates perception, decision-making, and control into a cohesive system. The distributed ROS2 architecture allows for modularity, extensibility, and robust performance. A key design choice was to rely exclusively on computer vision and velocity commands, deliberately avoiding other sensors or odometry to make the robot "less aware" of its internal state and more responsive to the external environment.

The project provides default models for the Robomaster S1 robot with additional vision sensors and traffic light cubes (under `models` and `scenes` in the root folder). The traffic light cubes must be used carefully as their orientation is critical - if rotated incorrectly, the faces of the cube are misaligned and may not be recognized properly. Additionally, the project includes a complete test scene containing a circuit with all the traffic signs and turns designed to validate the system's functionality under various conditions.

It is also easily adaptable to the RoboMaster EP by using an appropriate model with the additional vision sensors for line-following functionality and updating the launch files to use the correct drivers. Moreover, since the system relies solely on computer vision and velocity commands, it can theoretically be deployed on any four-wheeled robot with compatible hardware and software interfaces.

II. Background

Autonomous navigation has been extensively studied in the robotics and computer vision communities. Path following using vision-based approaches has evolved from simple line detection techniques to more sophisticated deep learning methods. Traffic sign recognition has similarly progressed from template matching to convolutional neural networks.

In the educational context, smaller platforms like Duckietown provide environments for teaching autonomous navigation concepts. ATLAS builds upon these approaches while emphasizing the integration of multiple perception and control modules within the ROS2 framework.

III. System Architecture

ATLAS is built on ROS2, leveraging its distributed architecture and publish-subscribe communication model. The system consists of three primary nodes:

- **Line PID Node**: Processes camera images to detect lines and calculate appropriate steering commands
- **Traffic Detection Node**: Analyzes camera input to recognize traffic signs and signals
- **Controller Node**: Integrates information from the other nodes to control the robot's movement

Fig. 1 illustrates the system architecture and communication flow between nodes.

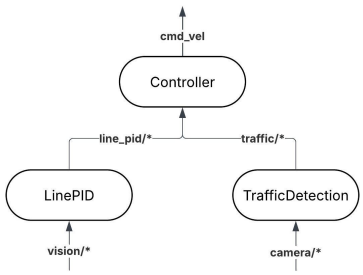

Fig. 1. ROS2 node graph showing the communication flow between the three primary nodes of the ATLAS system and their interactions with the robot's sensors and actuators.

A. System Deployment and Launch Configuration

The entire ATLAS system is deployed through a centralized ROS2 launch file, which configures all nodes and their parameters in a unified manner. This approach ensures consistent initialization and proper integration between components. Within the launch file, all topic names are specified with namespaces relative to the robot's name, allowing for potential multi-robot deployments with minimal configuration changes.

A debug launch file is also provided, which includes additional visualization tools for monitoring the system's performance. This debug configuration can be used to visualize the cameras feed after image processing and/or with information overlay.

The vision system for the robot (`/vision/left`, `/vision/center`, `/vision/right`) is implemented directly in the Coppelia simulation environment using a Lua script. This script simulates the robot's cameras and publishes the visual data to the ROS2 network, where it's consumed by the Line PID Node.

It's important to note that while other topics use relative paths beginning with the robot name, the vision sensor topics created by the Lua scripts use absolute paths. This means that for multi-robot simulations, both the Lua scripts and the Line PID Node would need to be modified to ensure the correct topic mapping. The Lua scripts would need to generate unique topic names per robot, and the Line PID Node would need to subscribe to the correct robot-specific vision topics rather than the current absolute ones.

B. Code Structure and Organization

The ATLAS system is organized into three Python modules, each implementing one of the primary nodes. The structure follows ROS2 best practices with clear separation of concerns:

- `line_pid_node.py`: Implements the `LinePIDNode` class that handles line detection and PID-based steering

- `traffic_detection_node.py`: Implements the `TrafficDetectorNode` class for ArUco marker recognition
- `controller_node.py`: Implements the `ControllerNode` class that handles decision-making and motion control

Each module follows a consistent pattern with class definitions, ROS2 publisher/subscriber initialization, parameter declaration, callback methods for handling sensor inputs, and helper methods for processing data and controlling outputs.

C. Communication Paradigm

The nodes communicate through ROS2 topics using standard message types. The key topics include:

- `/vision/left`, `/vision/center`, `/vision/right`: Camera feeds for line detection
- `camera/image_color`: Camera feed for traffic sign detection
- `line_pid/steering`: Normalized steering values from the PID controller [-1.0 to +1.0]
- `line_pid/stop_line_detected`: Boolean indicating stop line detection
- `line_pid/lost_line_detected`: Boolean indicating lost line detection
- `traffic/id`: Detected traffic sign/light identifier
- `cmd_vel`: Velocity commands for the robot

Additional topics are available launching the debug configuration, including:

- `line_pid/debug/left_cam`: Debug camera feed from the left line-following camera (post-processed)
- `line_pid/debug/center_cam`: Debug camera feed from the center line-following camera (post-processed)
- `line_pid/debug/right_cam`: Debug camera feed from the right line-following camera (post-processed)
- `traffic/debug/overlay`: Debug camera feed from the Traffic Detection Node, overlaying the detected ArUco markers on the original image
- `traffic/debug/processed`: Debug camera feed from the Traffic Detection Node, overlaying the detected ArUco markers on the processed image

D. Parameter Management

Each node exposes configurable parameters through the ROS2 parameter system, allowing for adjustment of behavior. This design choice enables fine-tuning of the system without code modification and supports adaptation to different environmental conditions.

IV. Line PID Node Implementation

The Line PID Node is responsible for processing camera images to detect lines and calculate steering commands using a proportional-integral-derivative (PID) control approach.

A. Multi-Camera Line Detection

The system employs three cameras positioned to provide comprehensive path visibility. A deliberate design choice was made to use three low-resolution cameras (16×16 pixels) rather than a single high-resolution camera. This approach more accurately represents real-world robotic scenarios where processing constraints and sensor limitations are common, and it provides inherent redundancy and wider spatial awareness.

Each camera's image is processed independently through an image processing pipeline:

1) **Image Pre-processing:** The raw RGB image is converted to BGR if needed, resized to 16×16 resolution, and rotated 90 degrees counterclockwise.
2) **Grayscale Conversion and Blur:** The image is converted to grayscale and a Gaussian blur is applied to reduce noise.
3) **Binary Thresholding:** The blurred image is converted to a binary image using an inverted threshold.
4) **Contour Detection:** OpenCV's contour detection algorithm identifies continuous shapes in the binary image.
5) **Contour Analysis:** The largest contour is selected as the most likely candidate for the line if its area exceeds the minimum threshold.
6) **Position Normalization:** The horizontal position of the line is normalized to a range of 0.0 to 1.0 using the moments of the contour.

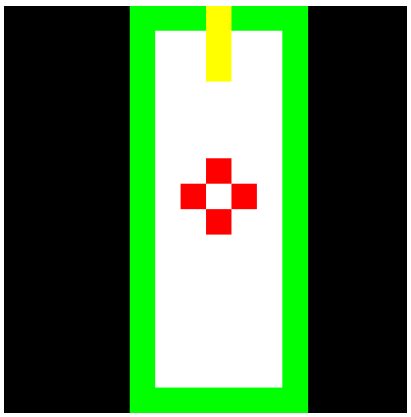

Fig. 2. Debug visualization from the line detection system. The white area represents the binary-thresholded image where the path line has been detected. The green borders show the edge of the track visible to the camera. The line position is shown with a yellow line. The red cross pattern highlights the centroid marker of the detected contour, showing the calculated center point of the line for PID control purposes.

The normalized positions have different target values based on camera position:

- Left camera: Target position 0.0 (left edge)
- Center camera: Target position 0.5 (center)
- Right camera: Target position 1.0 (right edge)

B. PID Control Implementation

Each camera has an independent PID controller with tunable parameters that can be adjusted through the ROS2 parameter system. The control system uses different proportional, integral, and derivative gains for each camera to account for their different perspectives.

The PID control implementation involves:

1) **Error Calculation:** The error is calculated as the difference between the detected line position and the target position for each camera.
2) **PID Term Calculation:** For each camera's error value, the system calculates proportional, integral, and derivative terms.
3) **Integral Anti-windup:** The accumulated integral value is clamped to configurable limits to prevent excessive overshoot.
4) **Derivative Filtering:** A moving average filter is applied to the derivative term to reduce sensitivity to noise.
5) **Sharp Turn Detection:** The system automatically detects sharp turns by analyzing discrepancies between camera readings and adjusts control parameters accordingly.

6) **Weighted Output Fusion:** The PID outputs from each camera are combined using a weighted average based on proportional gain parameters.
7) **Output Normalization:** The final steering command is normalized to a range of -1.0 to +1.0 using a hyperbolic tangent function.

C. Stop Line Detection

The system detects stop lines by analyzing positions from all three cameras simultaneously, looking for:

- **Multi-camera Confirmation:** Detection in all three cameras simultaneously
- **Position Consistency:** Line positions across all cameras within a configurable deviation threshold
- **Centered Detection:** Average position near the center of the field of view

D. Lost Line Detection

If no lines are detected across all cameras for a sustained period (configurable timeout parameter), the system signals a lost line condition, ensuring safe behavior when the robot encounters gaps or moves out of the designated route. This temporal threshold is crucial as it prevents false triggers from momentary detection failures, while still ensuring timely response to actual line loss situations.

E. Parameters Analysis

The Line PID Node's behavior can be significantly altered by adjusting its parameters (Table I).

TABLE I
LINE PID NODE PARAMETERS AND THEIR IMPACT

Parameter	Default	Impact of Changes
`threshold_value`	128	Higher values increase sensitivity to bright lines, lower values to darker lines
`min_contour_area`	3	Lower values increase sensitivity but may introduce noise
`debug`	False	Enable debug camera feeds
`stop_line_max_deviation`	0.15	Higher values allow more positional variation between cameras for stop line detection, but may cause false positives
`stop_line_center_min`	0.4	Lower values detect stop lines earlier (when they first appear at the bottom of the image)
`stop_line_center_max`	0.6	Higher values detect stop lines later (when they're closer to the center of the image)
`*_kp`	0.8-1.0	Higher proportional gain increases responsiveness but may cause oscillation
`*_ki`	0.02	Higher integral gain improves steady-state error but may cause overshoot
`*_kd`	0.35-0.4	Higher derivative gain reduces overshoot but may amplify noise
`integral_windup_limit`	0.2	Lower values reduce potential overshoot but limit error correction
`corner_detection_threshold`	0.25	Lower values make corner detection more sensitive

V. TRAFFIC DETECTION NODE IMPLEMENTATION

The Traffic Detection Node processes camera images to recognize traffic signs and signals using ArUco markers as fiducial references.

A. ArUco Marker Detection Logic

ArUco markers are square fiducial markers with a black background and an internal binary pattern that uniquely identifies each marker. The detection algorithm involves:

1) **Image Pre-processing:** The input image is converted to grayscale and processed with a Gaussian blur to reduce noise.
2) **Multi-threshold Approach:** Rather than using a single threshold value, which might fail under varying lighting conditions, the system applies multiple thresholds between configurable minimum and maximum values. This adaptive approach significantly improves detection reliability across different lighting scenarios.
3) **Contour Analysis:** For each detected marker, the system calculates the contour perimeter as a measure of marker size and the center position.
4) **Marker Filtering:** A filtering process selects the instance with the largest perimeter if multiple markers are detected.

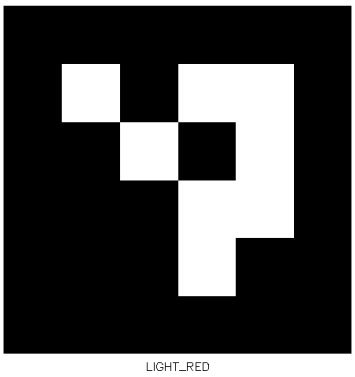

Fig. 3. Example ArUco markers used for traffic sign recognition, showing the distinctive patterns for different traffic elements.

B. Traffic Element Mapping

The system maps ArUco marker IDs to specific traffic elements:

- ID 0: Red traffic light
- ID 1: Yellow traffic light
- ID 2: Green traffic light
- ID 3: Speed limit 30 km/h
- ID 4: Speed limit 50 km/h

C. Detection Confidence Tracking

To improve reliability, the node implements a confidence tracking system:

- **Consistency Counting:** A confidence counter is incremented when the same marker ID is detected in consecutive frames.
- **Reset on Change:** The confidence counter is reset to 1 when a different marker ID is detected.
- **Confidence Threshold:** A detection is only considered valid when the confidence counter exceeds a configurable threshold.
- **Gradual Confidence Decay:** The confidence counter is decremented rather than immediately reset when no markers are detected.

D. Parameters Analysis

The Traffic Detection Node's behavior can be adjusted through several parameters (Table II).

TABLE II
TRAFFIC DETECTION NODE PARAMETERS AND THEIR IMPACT

Parameter	Default	Impact of Changes
`min_threshold`	40	Lower values detect darker markers but may increase false positives
`max_threshold`	150	Higher values detect lighter markers but may miss darker ones
`threshold_step`	20	Smaller steps improve detection at the cost of processing time
`min_detection_confidence`	10	Higher values reduce false positives but increase detection delay

VI. CONTROLLER NODE IMPLEMENTATION

The Controller Node integrates information from the Line PID Node and Traffic Detection Node to generate appropriate motion commands for the robot.

A. State Management Logic

The Controller Node maintains internal state information critical for decision-making:

- **Traffic Light State:** Boolean flags for red or yellow light detection
- **Line Detection State:** Tracking of stop lines or lost lines
- **Velocity State:** Current linear and angular velocities, including a "green light" reference velocity
- **Operation State:** Tracking of startup delay, running, or stopped modes

B. Motion Control Architecture

The Controller Node serves as the central integration point for the ATLAS system, subscribing to:

- **Steering Commands (`line_pid/steering`):** Normalized steering values (-1.0 to +1.0)
- **Stop Line Detection (`line_pid/stop_line_detected`):** Boolean signals for stop lines
- **Traffic Sign Information (`traffic/id`):** Identifiers for detected traffic signs and signals

The motion control loop runs at 30Hz, calculating appropriate linear and angular velocities and publishing velocity commands to the robot's `cmd_vel` topic.

C. Traffic Sign Integration

The Controller Node implements a state machine approach to integrate traffic sign information into navigation behavior:

- **Traffic Lights:**
  - Red Light: Reduces speed and sets the "red light detected" flag
  - Yellow Light: Reduces speed and sets the "yellow light detected" flag
  - Green Light: Resumes movement at the previously set "green light" velocity
- **Speed Limits:**
  - Speed 30: Sets the linear velocity to a slower value
  - Speed 50: Sets the linear velocity to a faster value

D. Safety Features

The Controller Node implements several safety mechanisms:

- **Startup Delay:** A configurable delay before allowing movement
- **Lost Line Response:** Immediate stop when the line is lost
- **Stop Line Response:** Appropriate action based on traffic light state
- **Emergency Stop:** A dedicated method to halt movement when required (i.e. stop line and red light, or lost line)
- **Incorrect slowing down:** A configurable timeout is set to prevent unwanted slowdowns when traffic lights remain visible after the vehicle has passed through a stop line, thus ignoring the traffic light if still in field of view.

E. Parameters Analysis

The Controller Node's parameters significantly affect the robot's behavior (Table III).

TABLE III
CONTROLLER NODE PARAMETERS AND THEIR IMPACT

Parameter	Default	Impact of Changes
`linear_velocity_50`	1.0	Higher values increase maximum speed but may reduce control precision
`linear_velocity_30`	0.5	Higher values increase reduced speed mode but may compromise safety in constrained areas
`angular_velocity_max`	2.0	Higher values enable sharper turns but may cause instability
`startup_delay`	5.0	Longer delay ensures stable initialization but increases wait time
`stop_line_timeout`	2.0	Prevents unwanted slowdowns when traffic lights remain visible after the vehicle has passed through a stop line

VII. EXPERIMENTAL RESULTS

The ATLAS system was tested in a controlled environment with a predefined path and various traffic elements, demonstrating line following, navigation, stop line detection, traffic light recognition, and speed adjustment.

A. Line Following Performance

The line following system demonstrated robust performance with the following observations:

- **Speed-Accuracy Trade-off:** Lower speeds provided the most accurate line following with maximum turn angles.
- **Multi-Camera Advantage:** The multi-camera approach improved performance on curves.
- **Lighting Sensitivity:** Performance remained consistent under moderate lighting variations.
- **Recovery Capabilities:** The system successfully recovered from temporary line loss conditions.

B. Traffic Recognition Performance

The traffic sign recognition system showed the following characteristics:

- **Detection Range:** Optimal detection occurred at lower distances from the camera.
- **False Positive Elimination:** The confidence tracking system effectively eliminated false positives.
- **Lighting Adaptation:** The multi-threshold approach significantly improved detection under varying lighting conditions.
- **Processing Efficiency:** The system maintained sufficient responsiveness for real-time operation.

C. Integrated System Performance

Tests of the complete integrated system revealed:

- **Navigation Accuracy:** Successful navigation with low path deviation.
- **Traffic Rule Compliance:** Appropriate responses to all traffic elements.
- **Continuous Operation:** Consistent performance during extended testing.

D. Edge Cases and Robustness Analysis

The system successfully handled edge cases including simultaneous traffic signs, rapid traffic light changes, line interruptions, and sharp turns.

E. Edge Cases and Robustness Analysis

The system successfully handled edge cases including simultaneous traffic signs, rapid traffic light changes, line interruptions, and sharp turns.

F. Video Demonstrations

Comprehensive video demonstrations of the ATLAS system capabilities are provided in the `media` folder, showcasing various operational scenarios and parameter sensitivity analysis:

- **traffic_light**: Demonstrates the robot's behavior when encountering traffic lights, showing the complete sequence from red light detection (causing the robot to stop at the stop line) through to green light detection (resuming normal operation).
- **pid_params**: Illustrates the impact of different PID parameter configurations on line-following performance, highlighting the trade-offs between responsiveness and stability discussed in the parameter analysis.
- **confidence**: Shows the effect of modifying the confidence threshold parameter in the Traffic Detection Node. Lower confidence values result in more reactive behavior but increased susceptibility to false positive detections.
- **angular_velocity**: Demonstrates the consequences of setting maximum angular velocity too low relative to linear velocity. The constrained turning capability results in turn radii that are too large to successfully navigate tight curves at the given speed. The robot successfully stops after losing line tracking for more than the configured timeout period.
- **linear_velocity**: Exhibits the effect of excessive linear velocity relative to maximum angular velocity. The high forward speed creates turn radius requirements that exceed the robot's steering capabilities, leading to path deviation and potential navigation failure. The robot successfully stops after losing line tracking for more than the configured timeout period.

These demonstrations validate the theoretical parameter sensitivity analysis presented in the implementation sections and provide practical insights into the system's operational envelope and tuning requirements.

VIII. Discussion

A. Integration of Multiple Perception Sources

The combination of line detection and marker detection creates a more robust navigation system than either approach alone. The multi-camera approach provided redundancy and broader spatial awareness, while the weighted fusion of camera inputs created a system that could smoothly transition between different perceptual regimes without explicit mode switching.

The deliberate choice to rely exclusively on computer vision and velocity commands, without using odometry or additional sensors, proved to be a significant advantage for environmental adaptability. By making the robot "less aware" of its internal state, the system was forced to continuously react to its environment rather than depending on potentially error-prone self-localization.

B. Parameter Tuning Trade-offs

Throughout development, we observed several trade-offs in parameter configuration:

- **Control Responsiveness vs. Stability:** Higher PID gains improved responsiveness but increased oscillation risk.
- **Speed vs. Accuracy:** Faster speeds improved efficiency but reduced detection reliability and control precision.
- **Detection Confidence vs. Latency:** Higher confidence thresholds reduced false positives but increased detection latency.
- **Image Processing Complexity vs. Computational requirements:** More sophisticated techniques improved detection but required additional computational resources.

C. Modular Design Benefits

The separation of perception, decision-making, and control into different nodes provided several advantages:

- **Isolated Testing:** Each component could be tested independently.
- **Reconfigurability:** Different configurations could be evaluated without modifying the controller node.
- **Failure Isolation:** Issues in one component did not necessarily compromise the entire system.

D. Simulation to Reality Transfer Challenges

For real-world deployment, several considerations were identified:

- **Lighting Variability:** Real-world lighting conditions are significantly more variable than simulation.
- **Processing Latency:** Real-world deployments might face increased processing latency due to hardware constraints.
- **Environmental Noise:** Physical environments introduce various noise sources not present in simulation.

IX. Conclusion

The ATLAS project successfully delivered a comprehensive ROS2-based solution for autonomous navigation with traffic sign and signal recognition. The system integrates line following capabilities with traffic element detection to create a unified navigation solution that follows paths and obeys traffic rules.

A. Possible Extensions

The modular architecture provides a foundation that could be extended in several ways:

- **Advanced Traffic Recognition:** Replacing ArUco markers with a CNN-based traffic sign recognition system.
- **Dynamic Obstacle Avoidance:** Integrating LIDAR or stereo vision systems for obstacle detection.
- **Path Planning:** Adding high-level path planning capabilities for complex environments.
- **Resource Optimization:** Optimizing algorithms for deployment on resource-constrained platforms.

The current implementation, complete with the Robomaster S1 model with custom vision sensors, ArUco-based traffic signs, and test circuit, provides a great foundation for extension of this autonomous

navigation systems. The included test scene allows for comprehensive validation of all system components under controlled conditions, while the modular architecture enables incremental improvements and extensions without requiring a complete redesign.

Code is available on GitHub at https://github.com/albertofinardi/atlas.