Welcome to Duckietown

Presented by Shivam Singh Chauhan, Pengwen Chen, Jasser

Date: September 04, 2025

What is Duckietown?

- Educational platform from MIT (2016) for AI/robotics, now global.
- Goals: Make autonomy accessible with low-cost hardware; bridge theory and practice.
- Components: Duckiebots (mobile robots), Duckietowns (mini-cities with roads/tape), ROS for software.
- Focus: Navigation, detection, tracking in real-world simulations.

Package 1 - Duckiebot Detection Node

01

Image Processing

Processes camera images to detect other Duckiebots using color filtering (HSV for red/orange).

02

Key Steps

Resize image, filter colors, apply erosion/dilation, find contours, create bounding boxes.

03

Outputs

Detection boolean, bounding boxes, processing time (and visuals if verbose).

04

Application

Enables obstacle avoidance in Duckietown environments.

Package 2 - Homography Estimation

Script: `ex_estimate_homography.py`

- Calls ROS service to compute homography matrix from camera image.
- Purpose: Maps 2D image pixels to 3D ground plane for accurate positioning.
- Input: Vehicle name, camera image.
- Output: Homography matrix (printed).
- Use: Essential for ground projection in navigation.

Package 3 - Ground Coordinate

The `ex_get_ground_coordinate.py` script converts normalized 2D image points to 3D ground coordinates via a ROS service.

1

Input

Vehicle name, normalized vector (e.g., 0.5, 0.7).

2

ROS Service

Translates image coordinates to real-world ground points.

3

Output

3D point (x, y, z).

This script supports mapping detections to physical space, crucial for robot interaction with its environment.

Package 4 - Obstacle Tracking Node

This ROS Node tracks obstacles (e.g., Duckiebots) over time using centroid association and smoothing.

Exponential Smoothing

Reduces noise in position data for stable tracking.

Nearest-Neighbor Matching

Associates current detections with existing tracks.

Pruning Old Tracks

Removes tracks that are no longer active or relevant.

Outputs smoothed obstacle positions as projected detections, providing stable tracking for dynamic environments.

Package 5 - Lane Filter Node

ROS Node Function

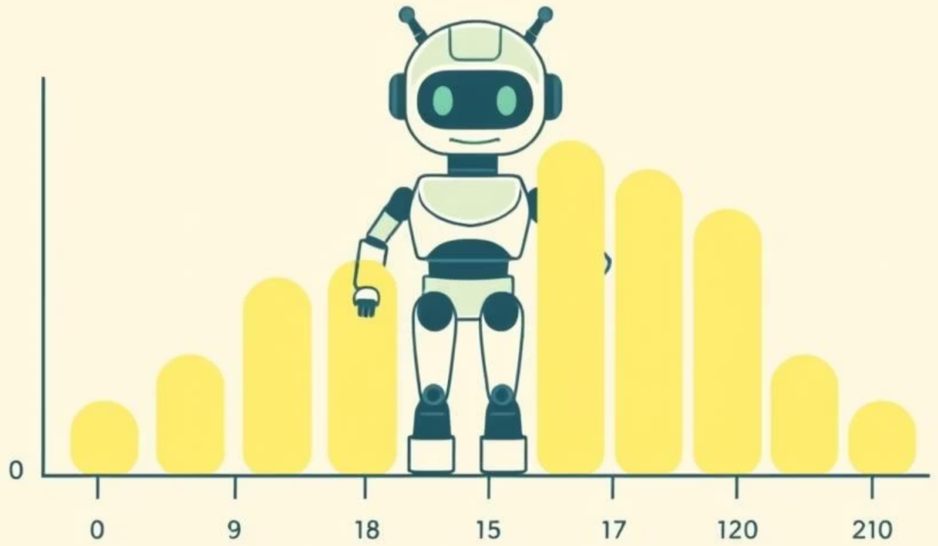
Estimates Duckiebot's lane pose (offset *d*, heading *phi*) from segment detections.

- Predict pose with velocity.
- Update with segments.
- Filter inliers.
- Estimate curvature.

Outputs

- Lane pose
- Filtered segments
- Belief image

This node is core for lane-following autonomy in Duckietown.



Integration, Conclusion, and Next Steps

How They Work Together:

Detection → Projection (homography/coords) → Tracking → Lane Filtering → Autonomous Navigation.

Benefits

Modular ROS design; real-world AI/robotics skills.

Conclusion

dt-core independent lane following and object detection.

Next Steps:

- Explore duckietown.org
- Build a Duckiebot
- Contribute to repos

Thank You!