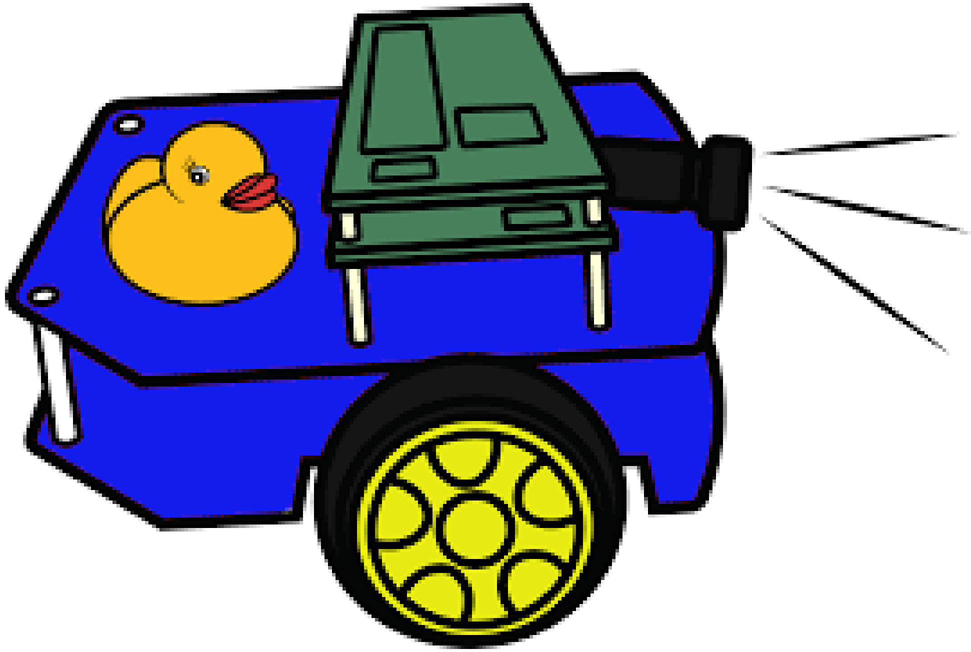
Duckiebot Autonomous Driving System



Group V

Supervisor: Prof. Dr. Amr Alanwar
Professorship of Cyber-Physical Systems
TUM School of Computation, Information and Technology

Heilbronn, 02.09.2025

Introduction

About This Project

- Self-driving vehicle system

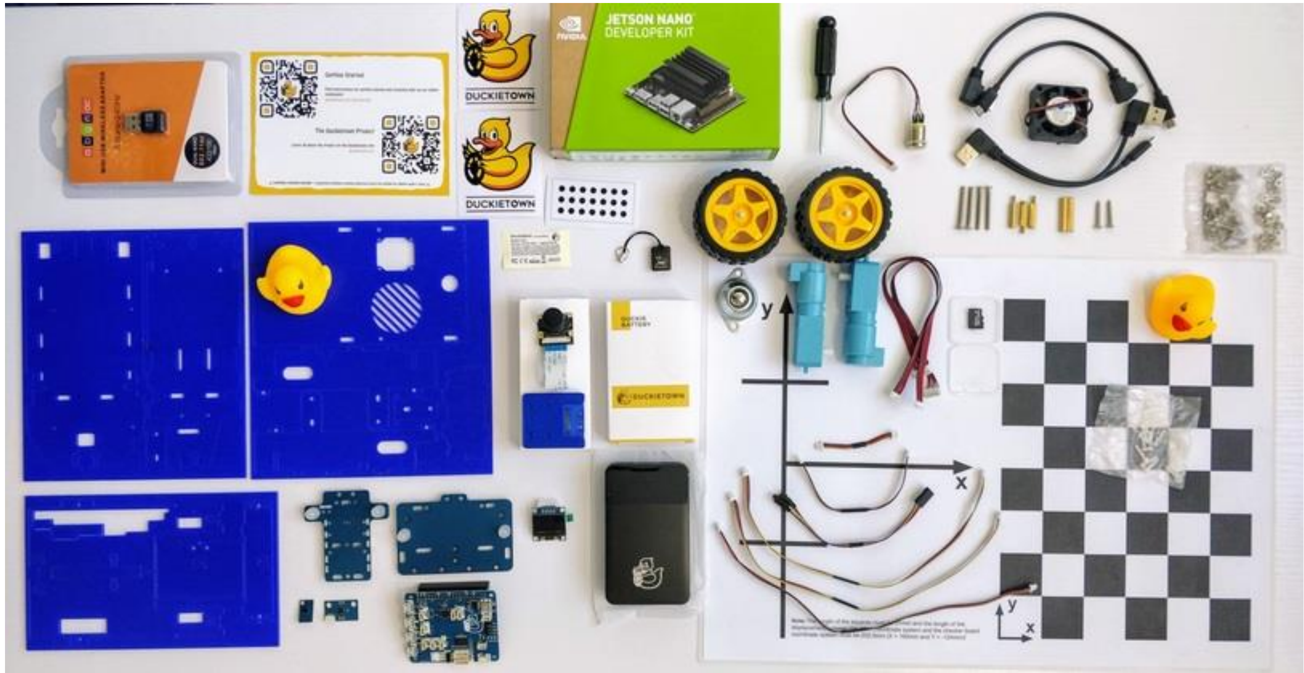
- Functionalities:
  - Color Recognize & ROS Control
  - Lane-keeping
  - Road environment detection
  - Reaction to traffic sign

- Challenges
  - Traffic light detection
  - Obstacle avoidance
  - …

About Our Group

- Group 1: Weiyuan Du, Jing Cheng
  - Contributions: map modification, ROS integration, reinforcement learning for Obstacle Avoidance
- Group 2: Ruilai Ma, Xinran Shen
  - Contributions: simulator setup, lane-keeping, intermodule integration
- Group 3: Yijie Kao
  - Contributions: road environment detection, intermodule integration
- Group 4: Yingtong Tian
  - Contributions: documentation management, demo recording
- All team members contribute to the documentation.

Hardware Build Components



Duckiebot DB21J

https://docs.duckietown.com/daffy/opmanual-duckiebot/intro.html

Visual Simulator

Gym-Duckietown Simulator

Our simulator :
Duckietown self-driving car simulator environments for OpenAI Gym.

https://github.com/duckietown/gym-duckietown

Project Architecture

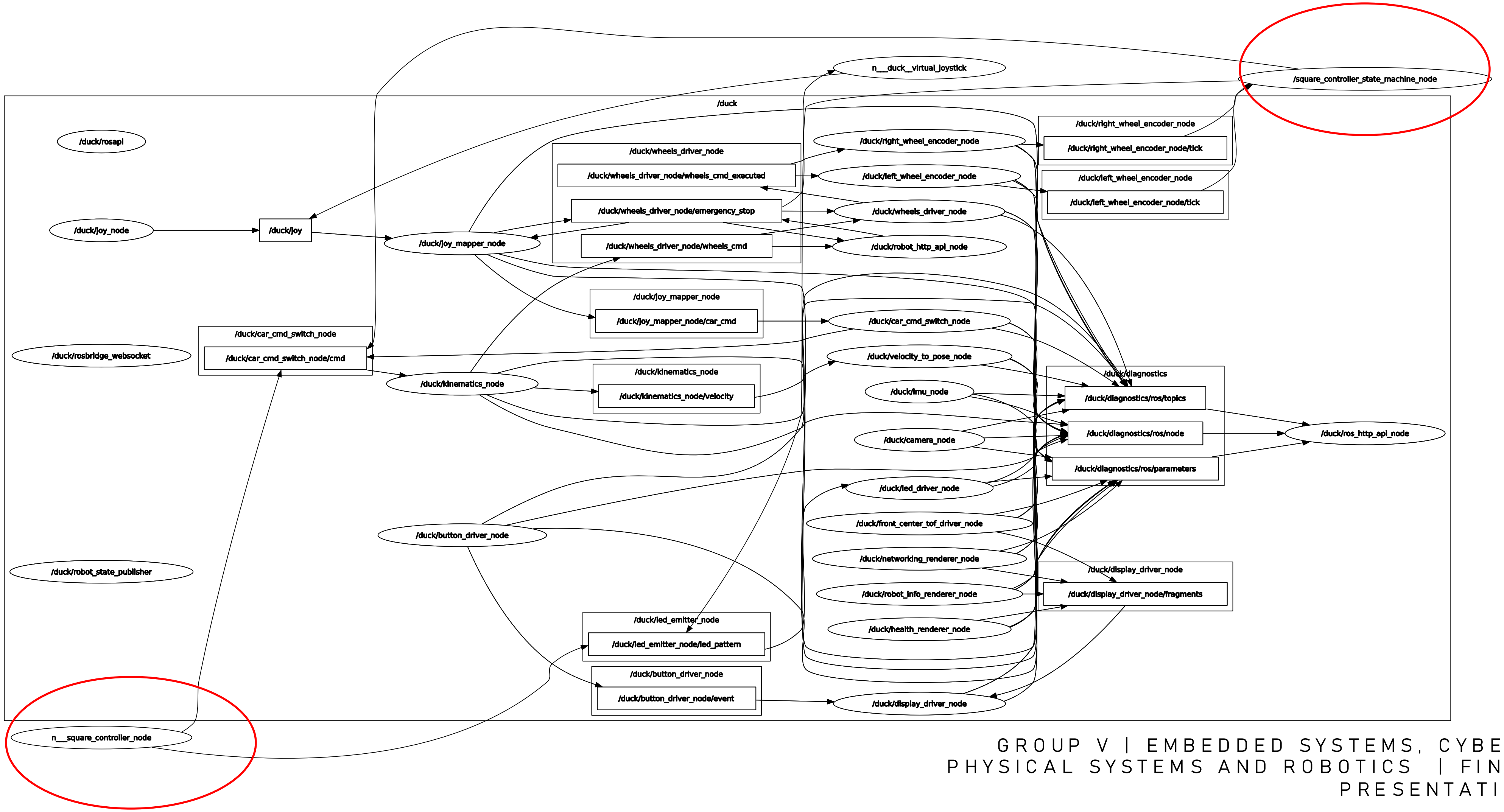
```
Duckiebot/
├── README.md
├── duckie-setup.md                  # Hardware setup guide for gym
|
├── autonomous_driving/              # Simulation & algorithms
|   ├── main.py
|   ├── perception.py               # Lane detection & computer vision
|   ├── controller.py               # PID control logic
|   └── utils.py                    # Helper functions
|
├── duckiebot-ros/
|   ├── Dockerfile
|   ├── launchers/                   # Startup scripts
|   └── packages/my_package/src/
|       ├── camera_reader_node.py    # Vision & LED control
|       ├── odometry_*.py            # Position estimation
|       ├── square_controller_*.py   # Movement controllers
|       └── wheel_*.py               # Motor control
|
├── false light detection/           # Traffic light detection
├── Reinforce learning/              # ML approaches
└── assets/
```

Challenge

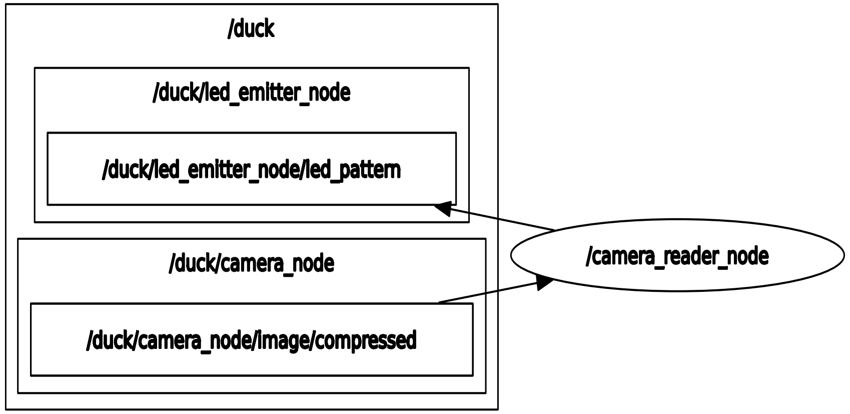
- Mac Users
- Architecture Compatibility
- Docker

Module 1:
Color Recognition & ROS Control

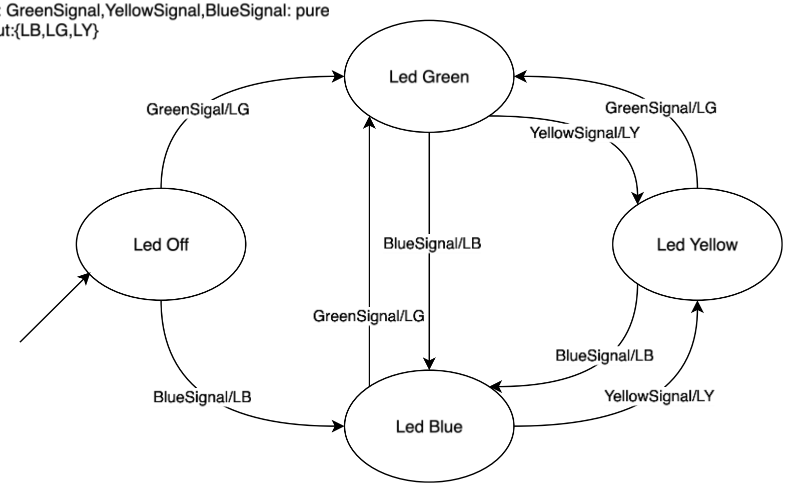
Overview of the Ros Node

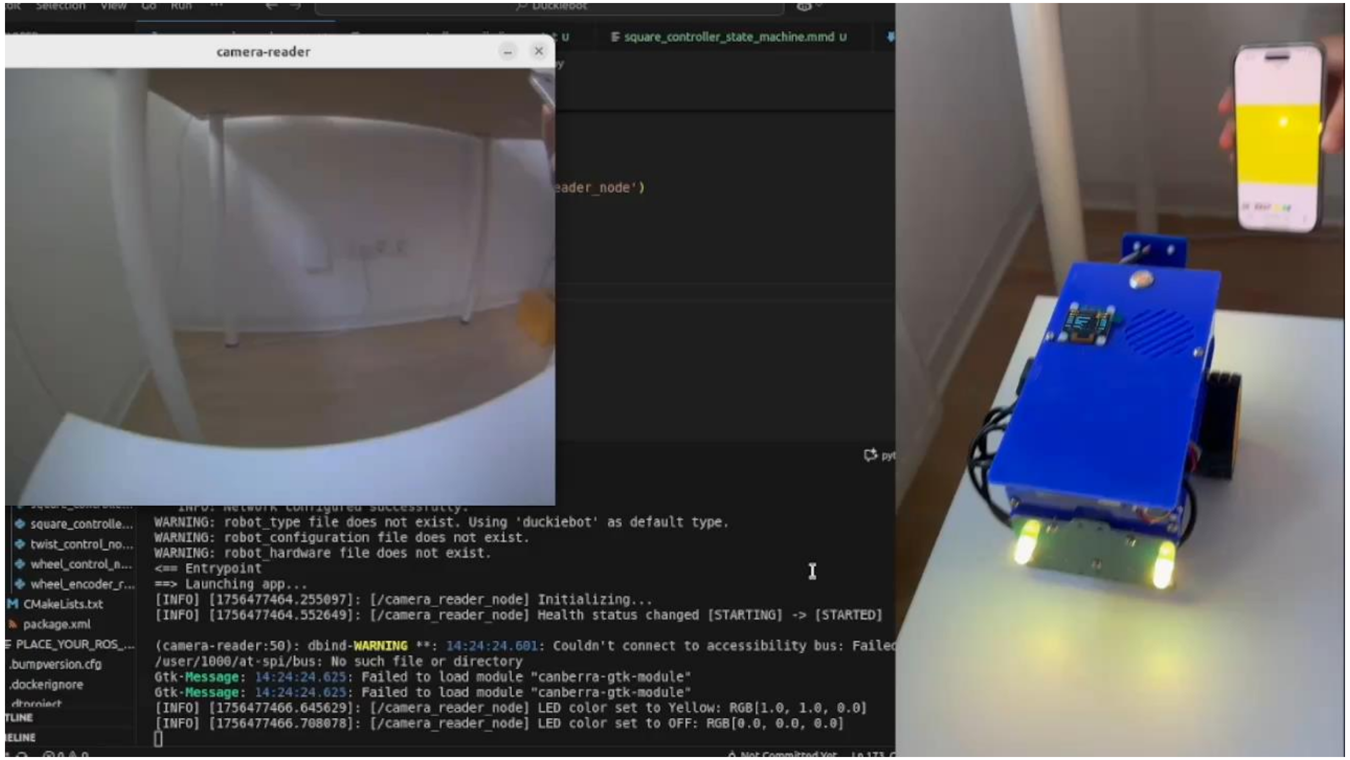


Camara Reader

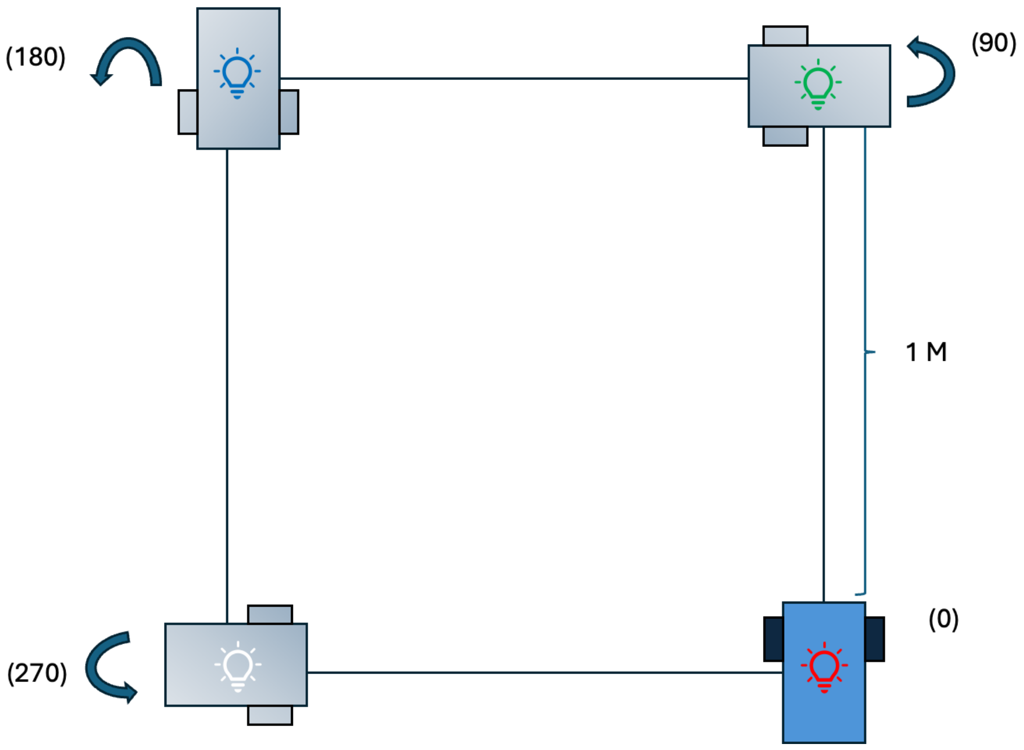


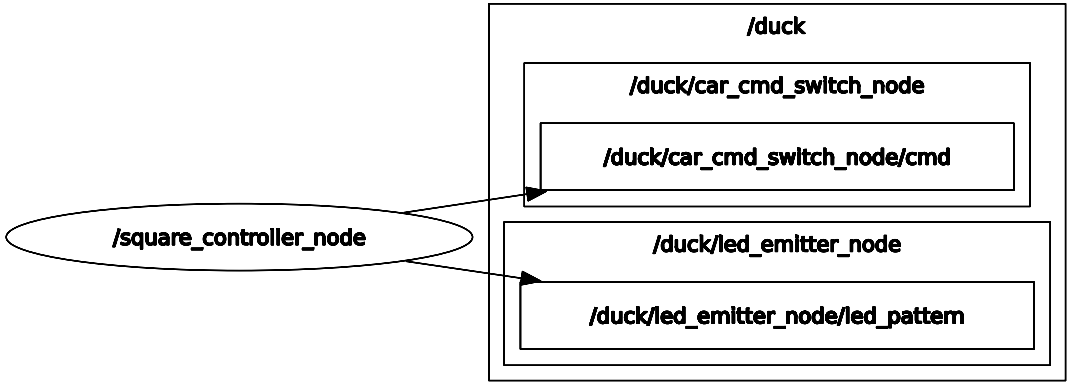
Input: GreenSignal,YellowSignal,BlueSignal: pure
Output:{LB,LG,LY}





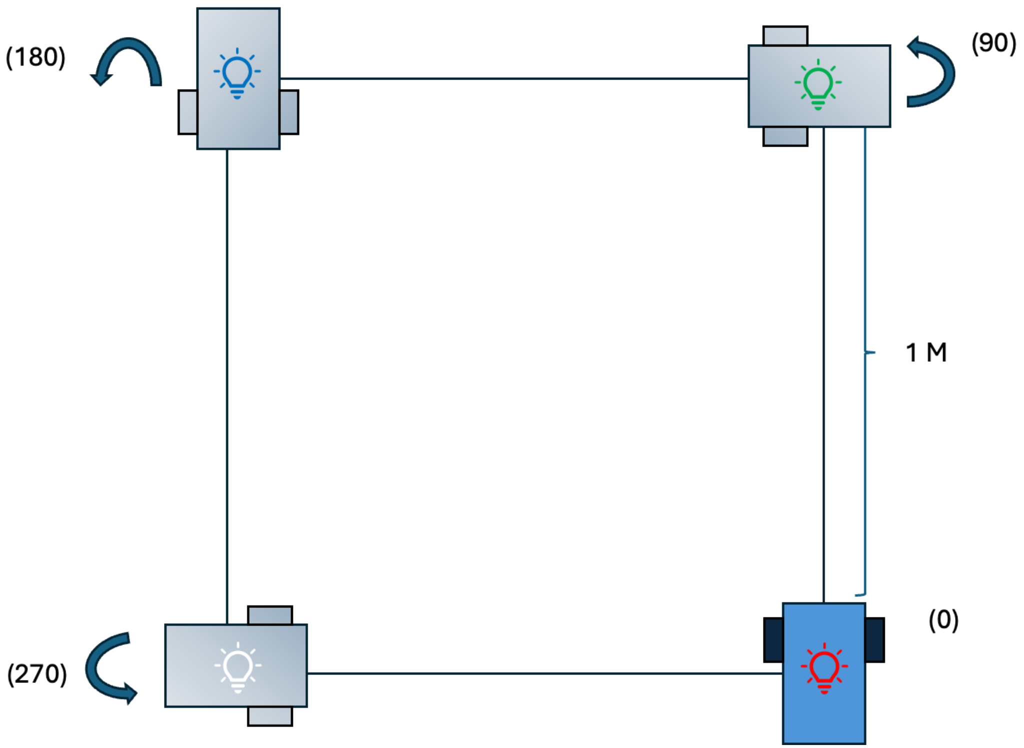
Square Controller



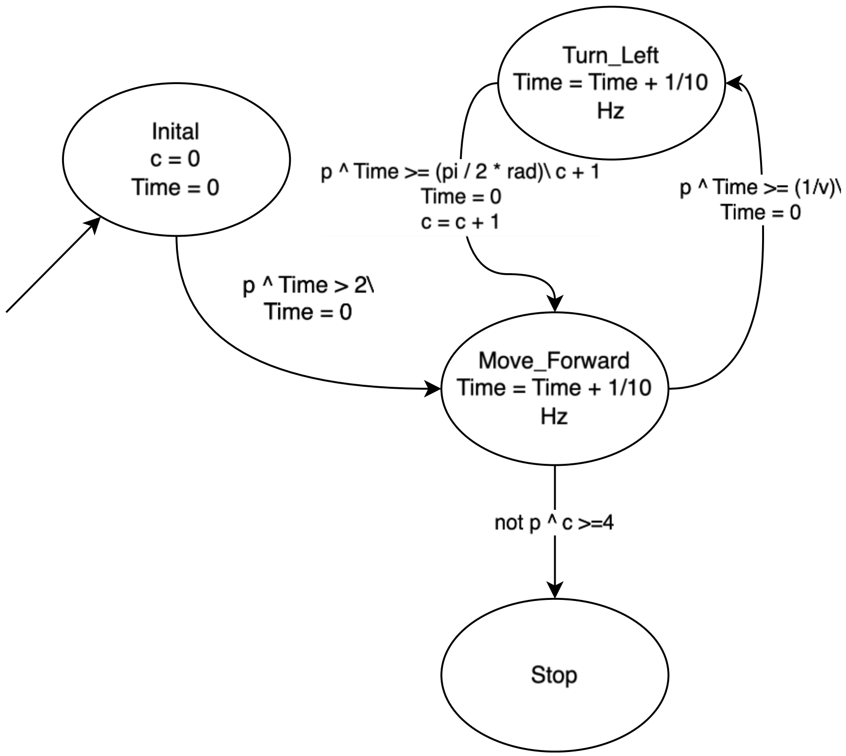


Drive the Duckiebot to walk in a square with open-loop control and change the led color to different color to indicating different phases.

Square Controller

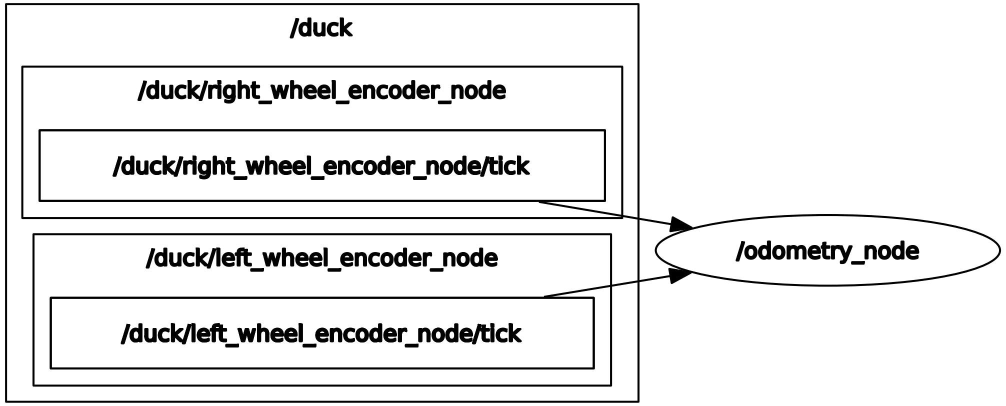


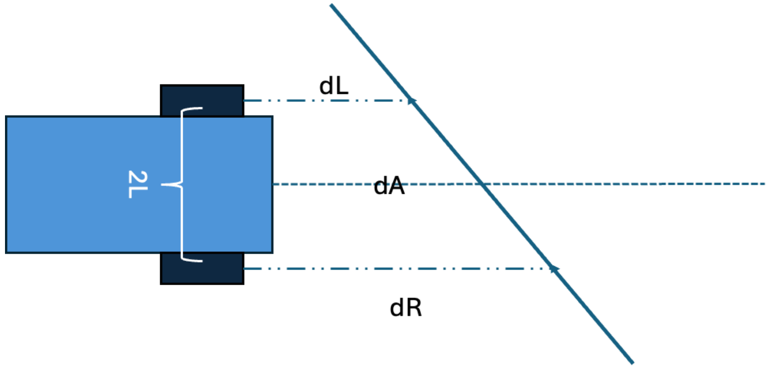
Input: p
Variable: Time,c
Ouput: c:{1,2,3,4}
10Hz



A Time based FSM

Odometry and Activity





$$\Delta_{\text{wheel}} = \frac{2\pi R N_{\text{ticks}}}{N_{\text{total}}}$$

$$\Delta x = dA\cos(\theta^{(t)})$$

$$\Delta y = dA\sin(\theta^{(t)})$$

$$\Delta\theta = \frac{dr - dl}{2L}$$

$$dA = \frac{dr + dl}{2}$$

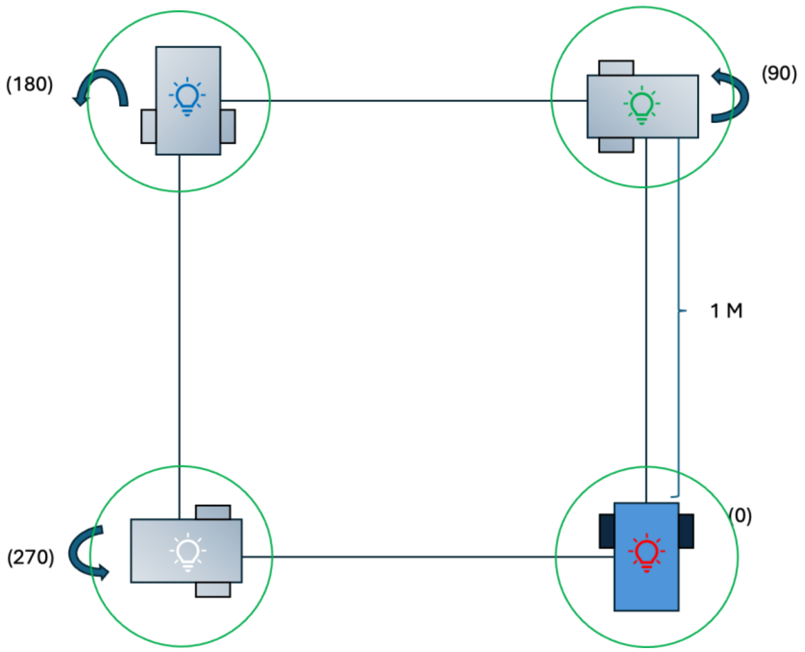
$$x_w^{(t+1)} = x_w^{(t)} + \Delta x$$

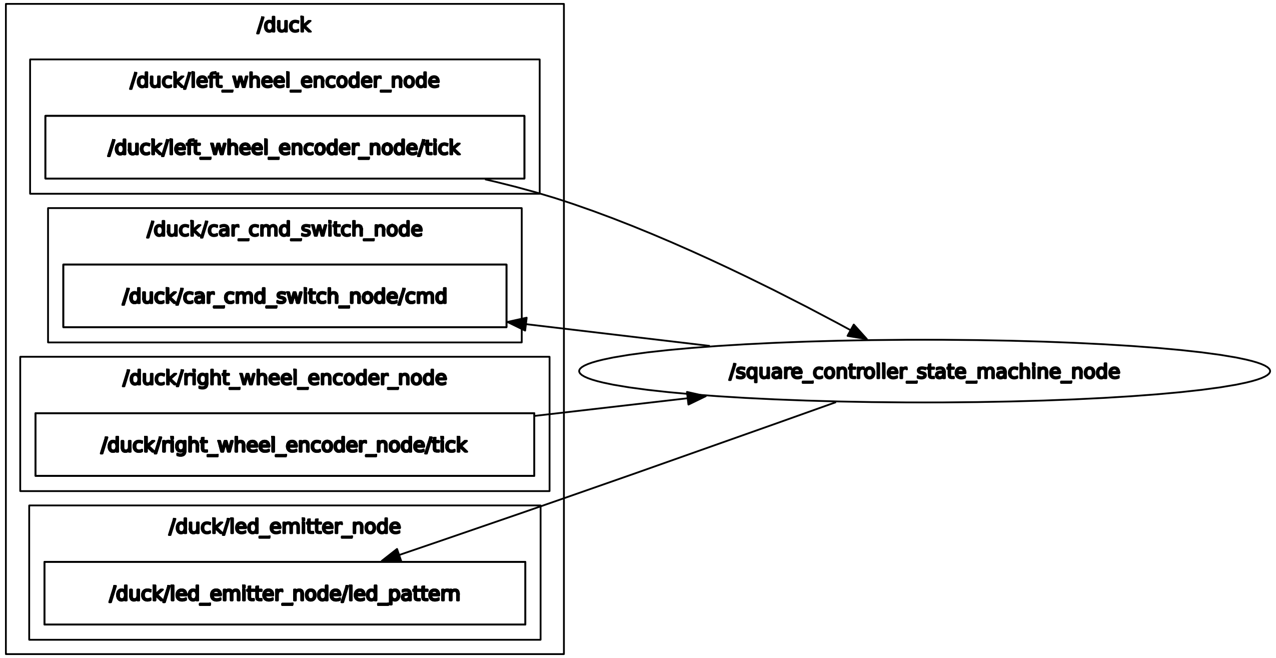
$$y_w^{(t+1)} = y_w^{(t)} + \Delta y$$

$$\theta_w^{(t+1)} = \theta_w^{(t)} + \Delta\theta$$

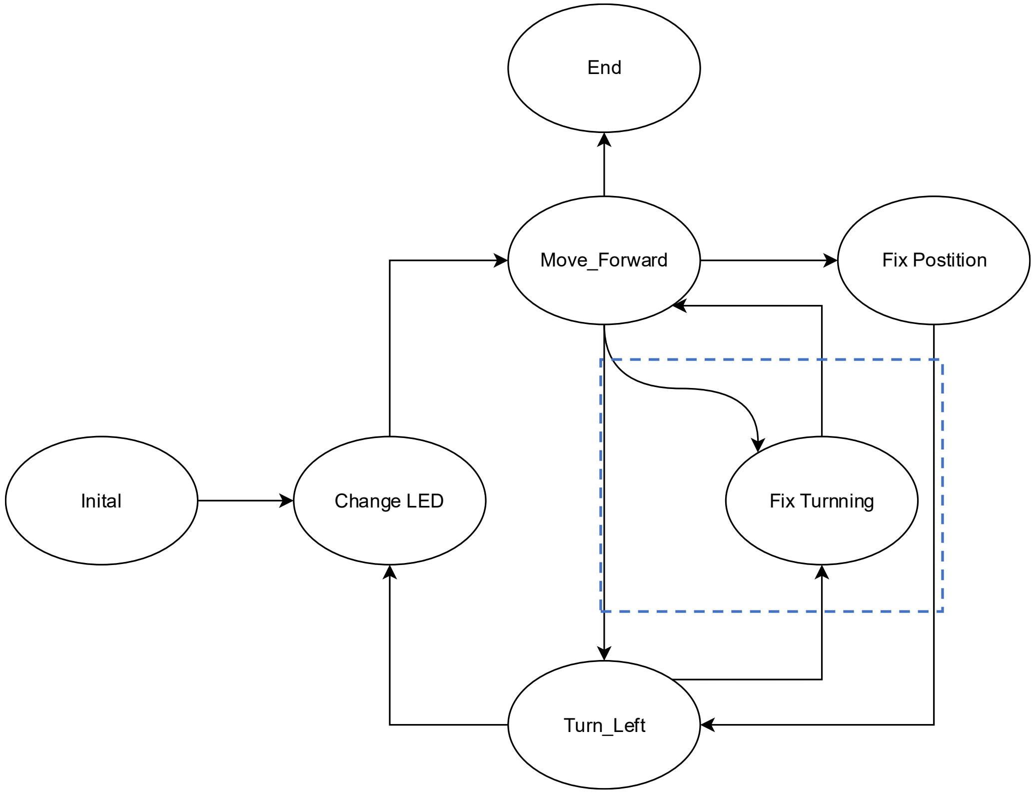
This node computes the pose estimate of the bot using wheel encoder data
and can be subscribed by other node for odometry information

Square Controller (Closed)

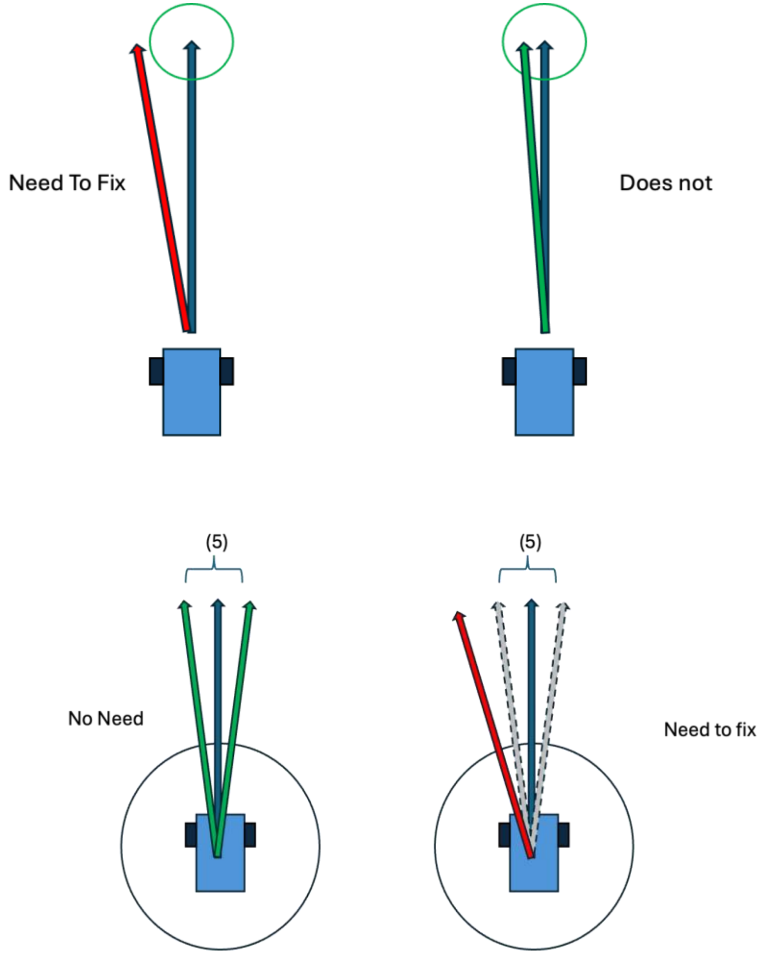




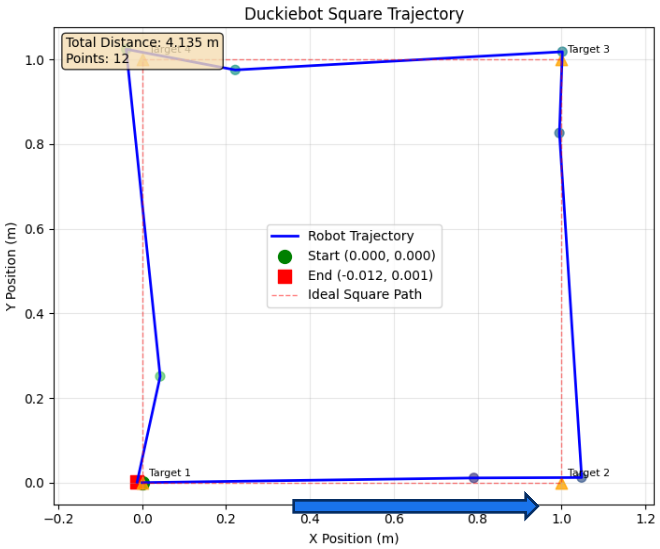
Square Controller (Closed)

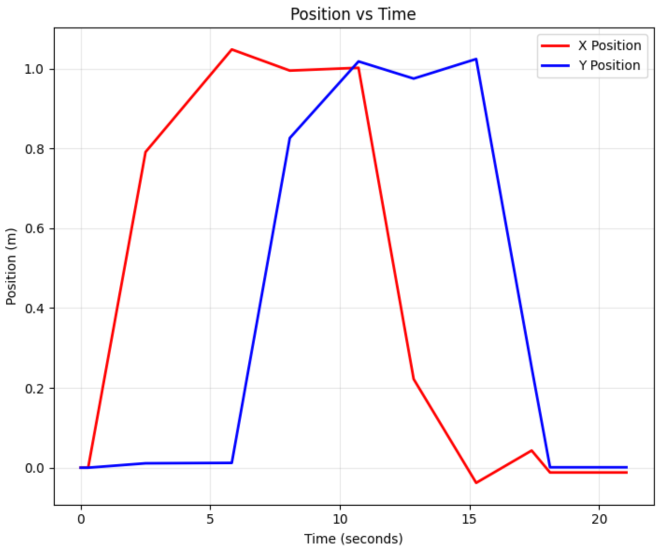


An Ideal State Machine for Square Task (Simplified)



Analyze the Performance





Challenge

- Correct ROS nodes
- Synchronization is important
- Gap between real and expected

Module 2:
Lane Keeping – PID Control & Lane Servoing

Approach 1: PID Control

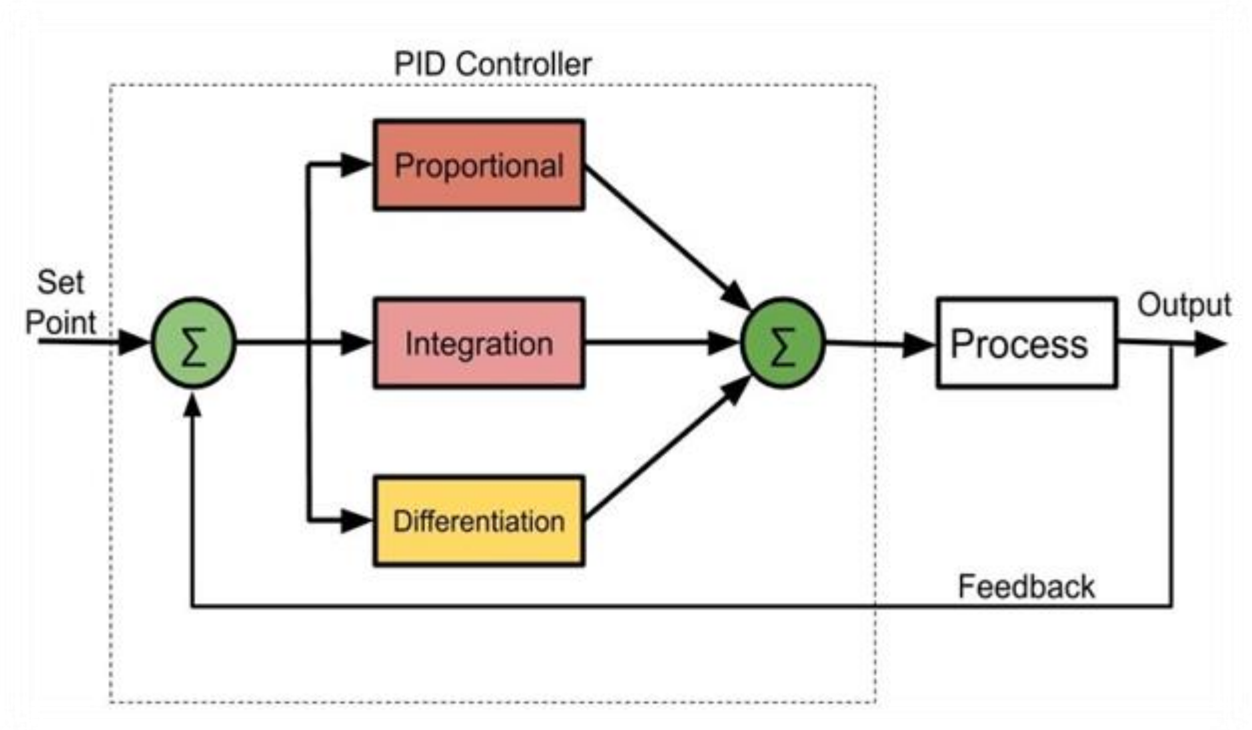


image from: https://www.flyeye.io/drone-acronym-pid/

PID Control

- Proportional adjustment of the steering force based on the error from the desired direction
- Integral adjustment of small biases, preventing them from building up over time
- Derivative adjustment reduces the tendency to over-correct from the proportional adjustment

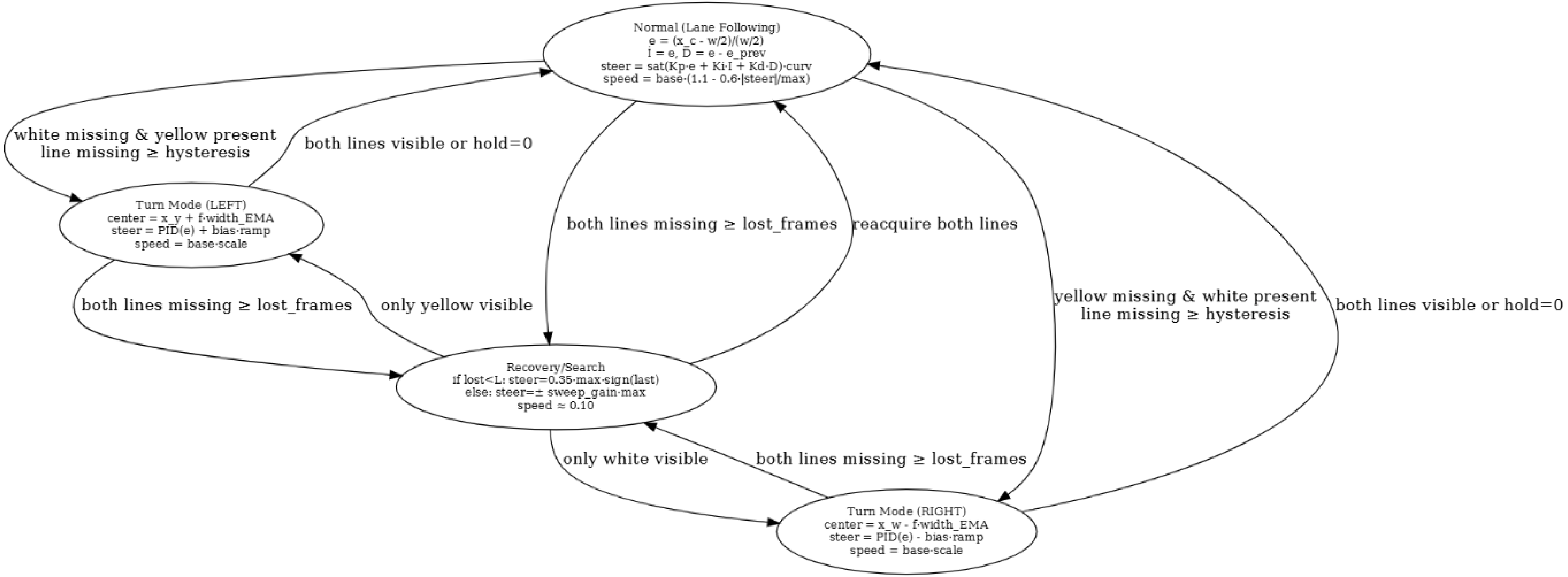
Preprocessing

- CLAHE enhances contrast of the observed image for better perception
- HSV (Hue Saturation Value) and LAB (Lightness – A – B) mask processing for left and right-side of the observed image
- Left and right lane marking extracted from color masks
- Lane center estimated from lane markings via horizontal slicing and Hough line transformation

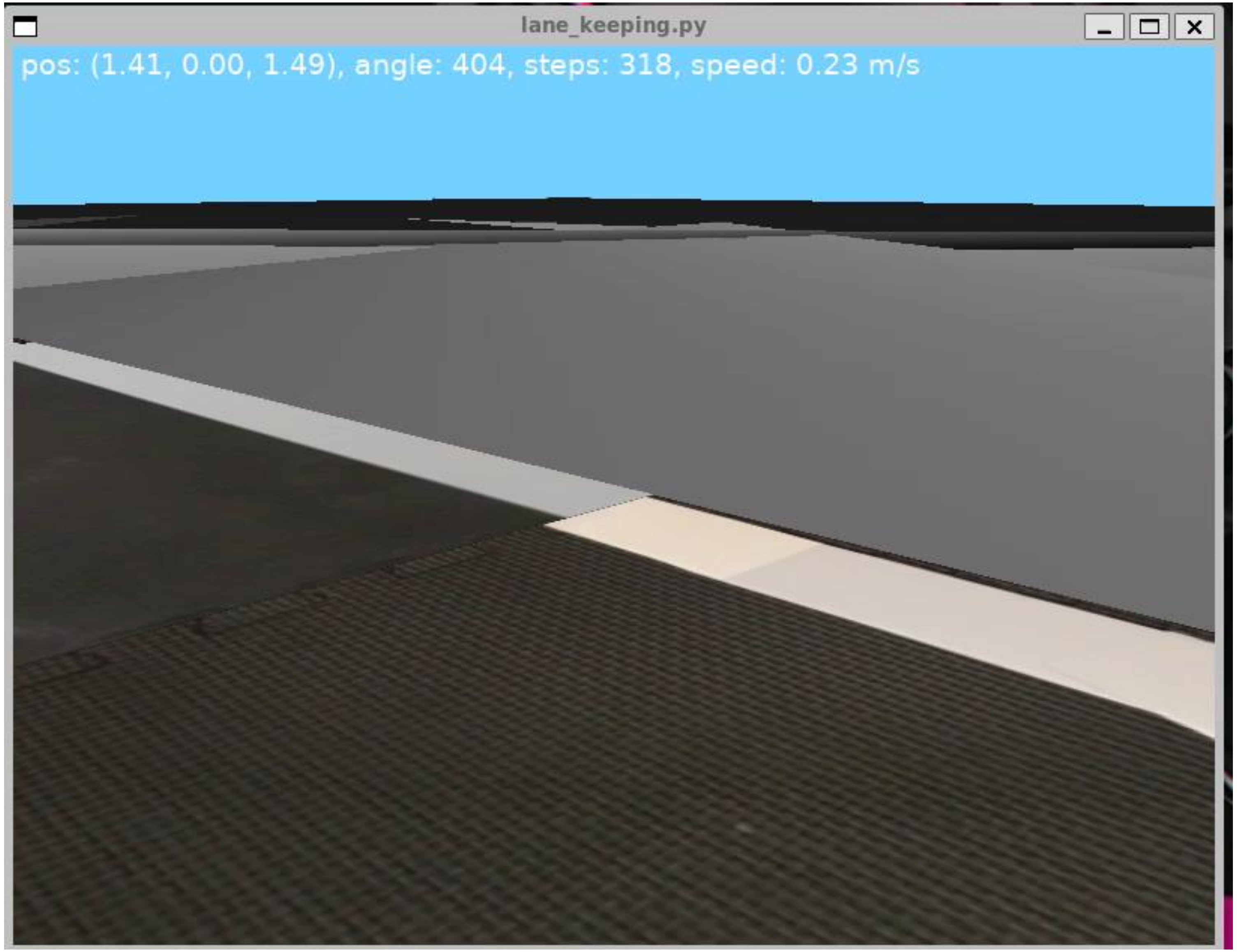
Fine Tuning: Turn Awareness

- PID Control is overly reliant on both line markings being present
- Solution: Adaptive turn awareness
- Heuristically determine steering direction when one reference marking is lost
- If both markings are lost, re-acquire lane keeping target

State Machine for turn awarness



Demo


lane_keeping.py
pos: (1.41, 0.00, 1.49), angle: 404, steps: 318, speed: 0.23 m/s

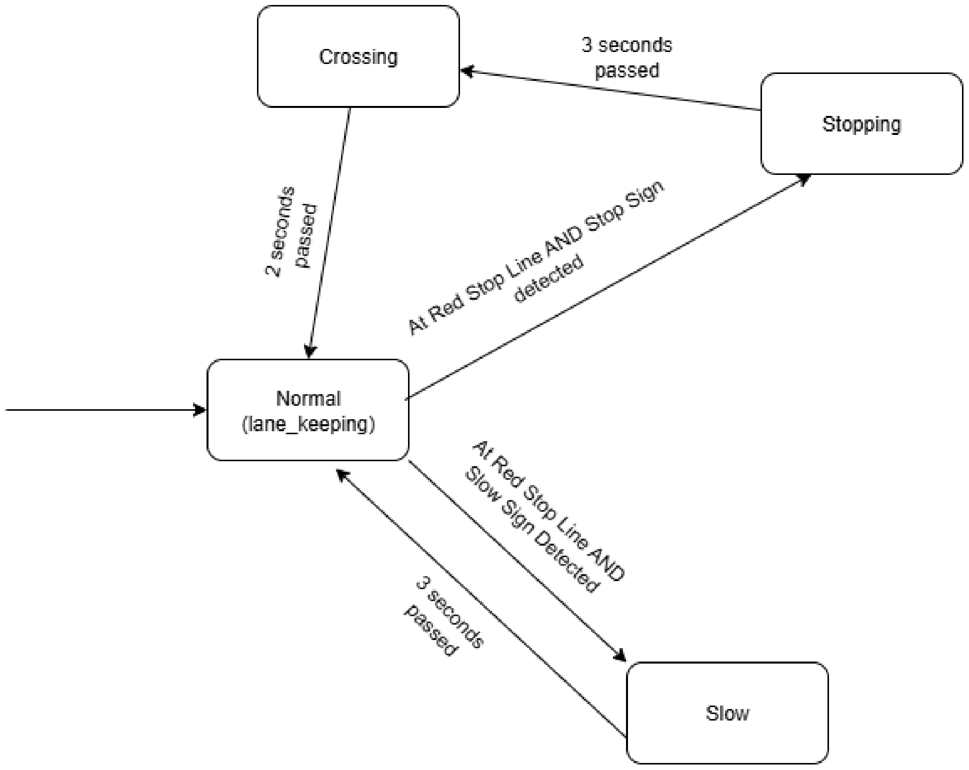
Approach 2: Lane Servoing

Preprocessing

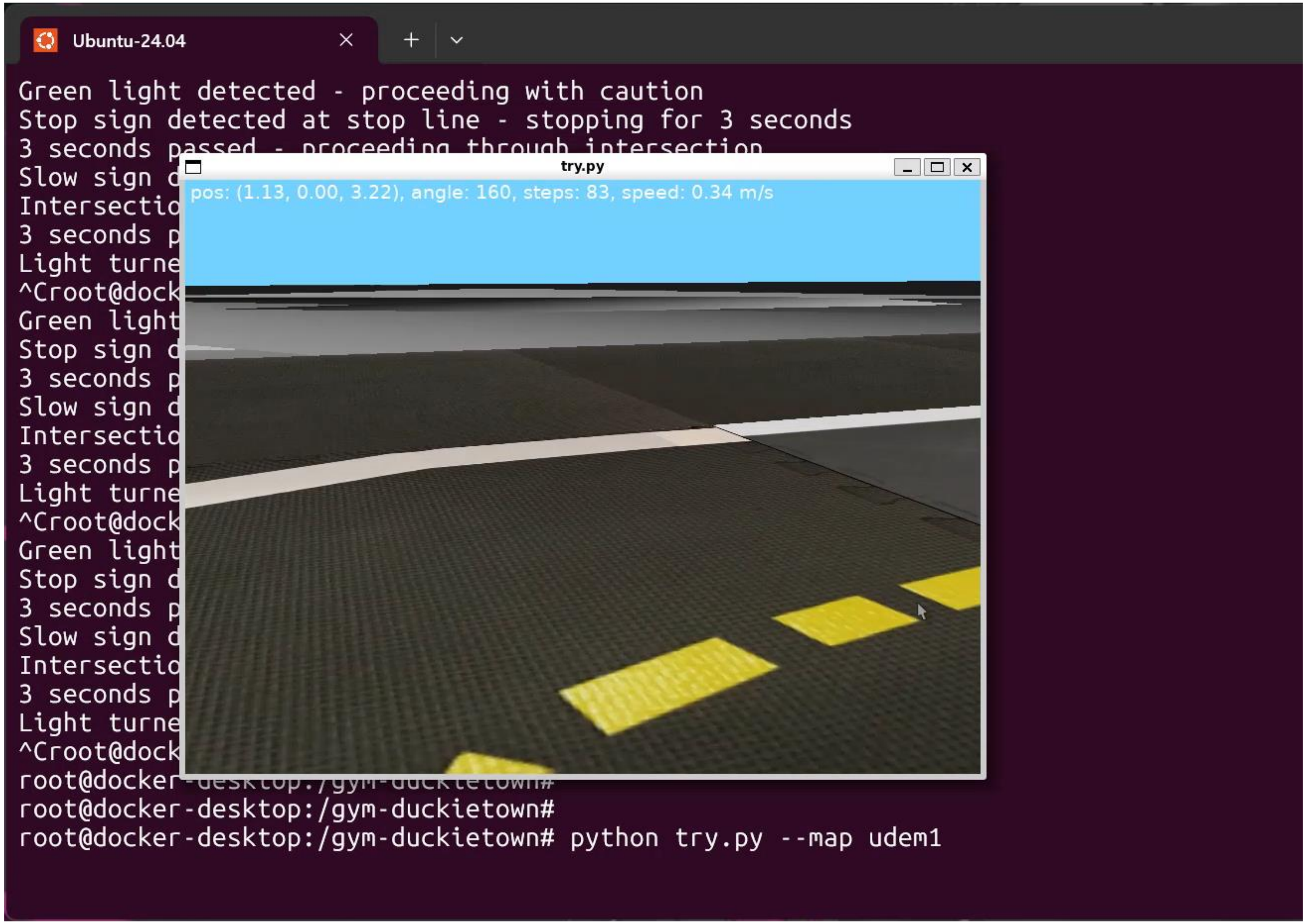
- Only apply HSV mask processing for left and right-side of the observed image
- Left and right lane marking extracted from color masks
- Lane center estimated from lane markings via weighted mean

Lane Servoing

- Simple steering towards desired direction
- Slows proportional to degree of turn
- Problem: traffic light detection

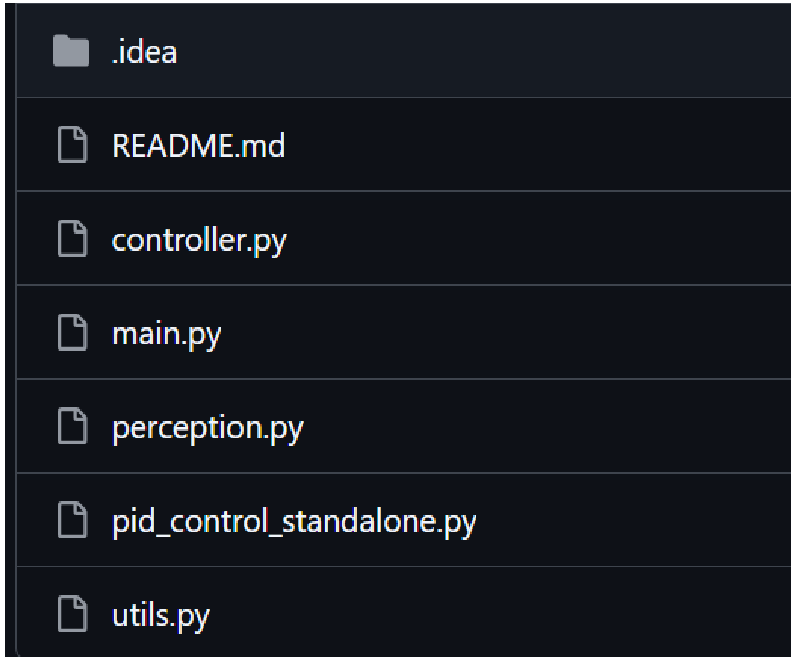


Demo


Ubuntu-24.04
Green light detected - proceeding with caution
Stop sign detected at stop line - stopping for 3 seconds
try.py
pos: (1.13, 0.00, 3.22), angle: 160, steps: 83, speed: 0.34 m/s
root@docker-desktop:/gym-duckietown#
root@docker-desktop:/gym-duckietown# python try.py --map udem1

Excursus: Project Organisation

- Originally, everything was developed in monolithic files
- Files were not easily portable and readable
- Thus the later lane servoing controller was split into four modules
- This has several benefits:
  - Portability
  - Maintainability
  - Readability



Module 3:
Traffic Light & Road Sign Detection

Objectives

- Recognize road signs (Stop, Intersection).
- Detect traffic lights (Red → Stop, Green → Go).
- Integrate recognition into driving decisions.

Traffic Light Recognition

Detection:

- Color filtering (red, green).
- Circle detection (Hough Circles).
- Region of interest (ROI)

Decision:

- Red → Stop car.
- Green → Resume motion.

Road Sign Detection:

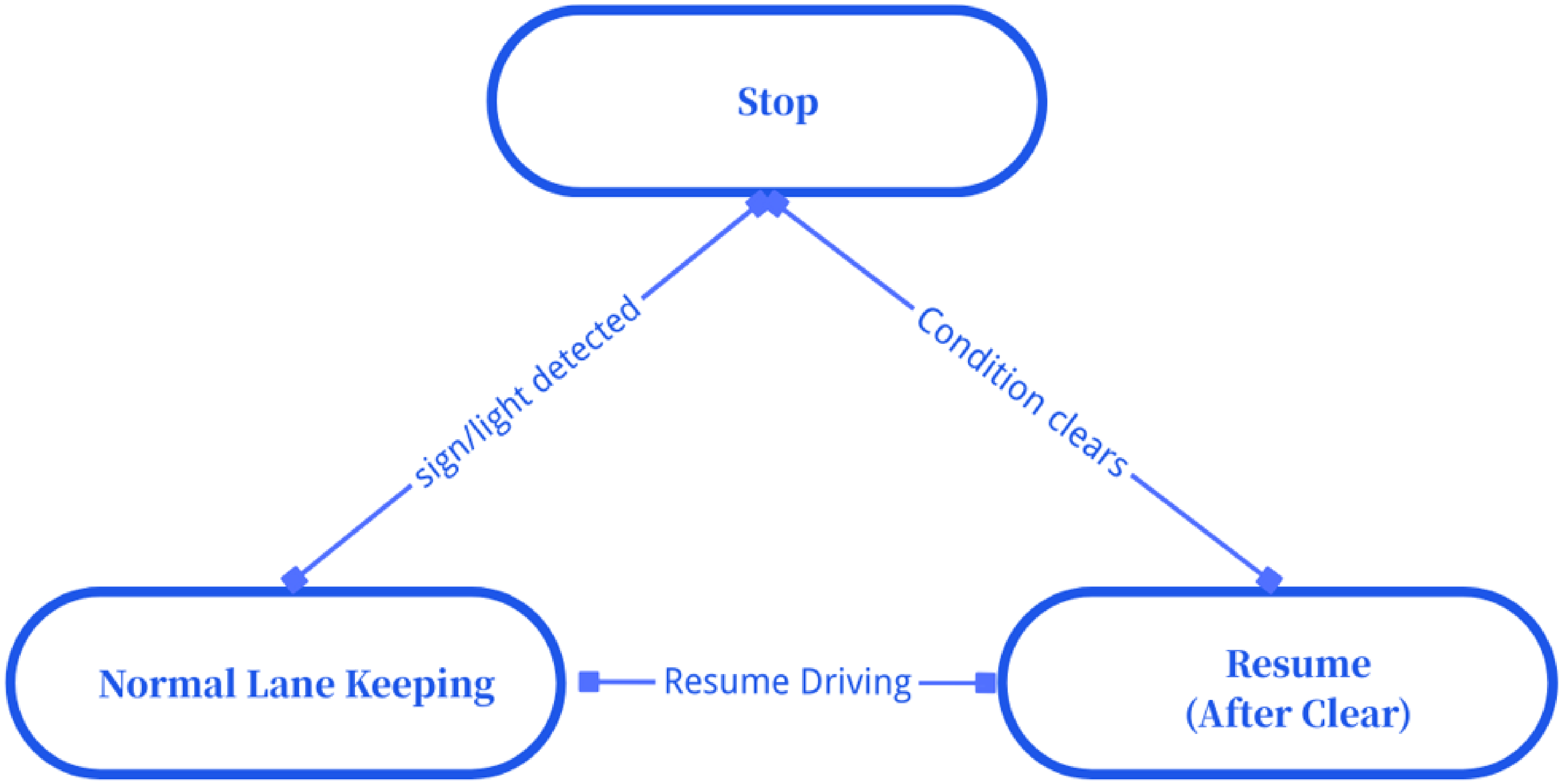
Image Processing:

- Color filtering (red, yellow)
- Color segmentation (blue background + red border).
- Shape detection (Hough transform / contour matching).

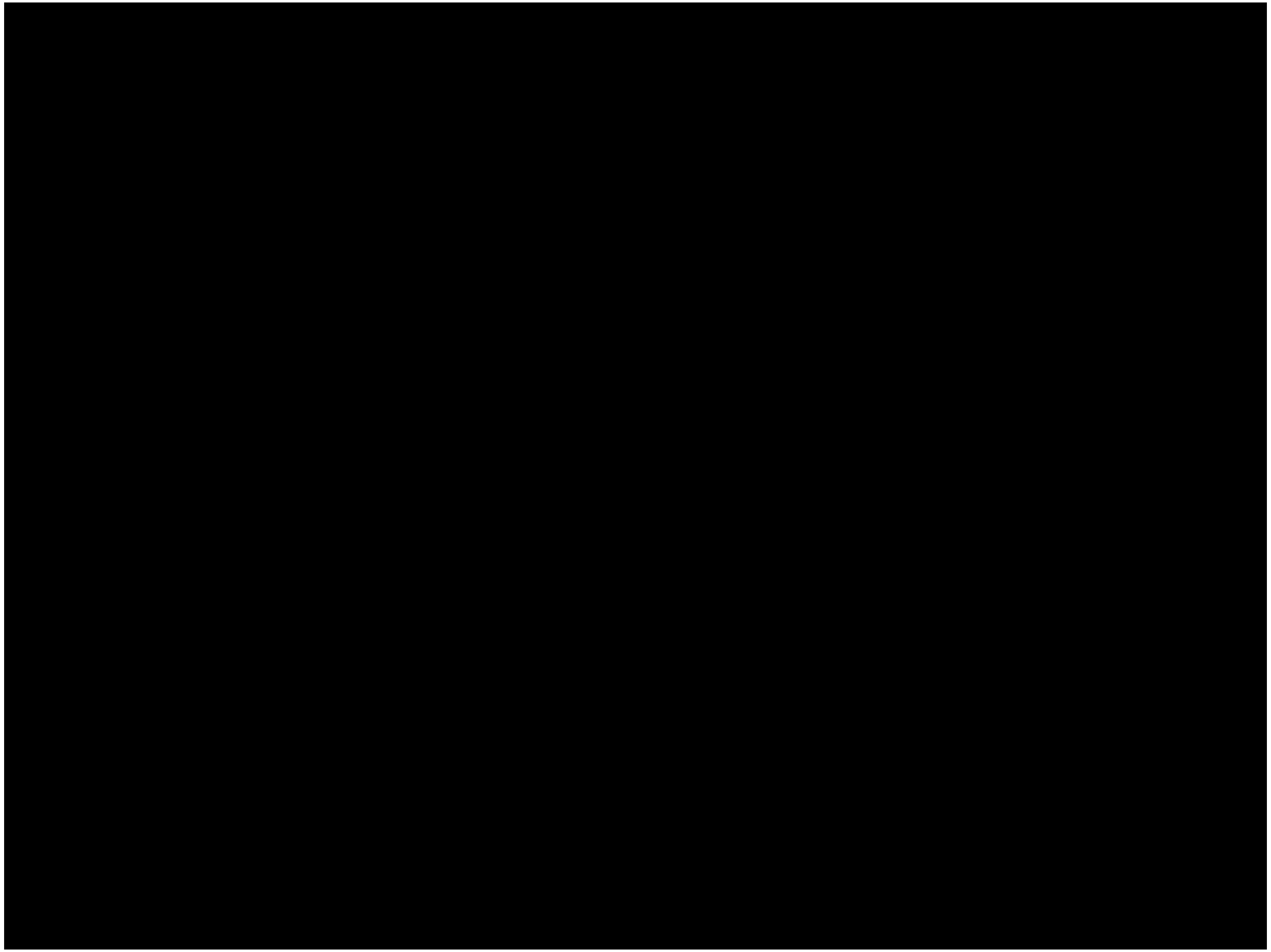
Classification:

- Template matching

Integration into Control



Demo



Module 4:
Reinforcement Learning for Obstacle Avoidance

System Design

Agent:

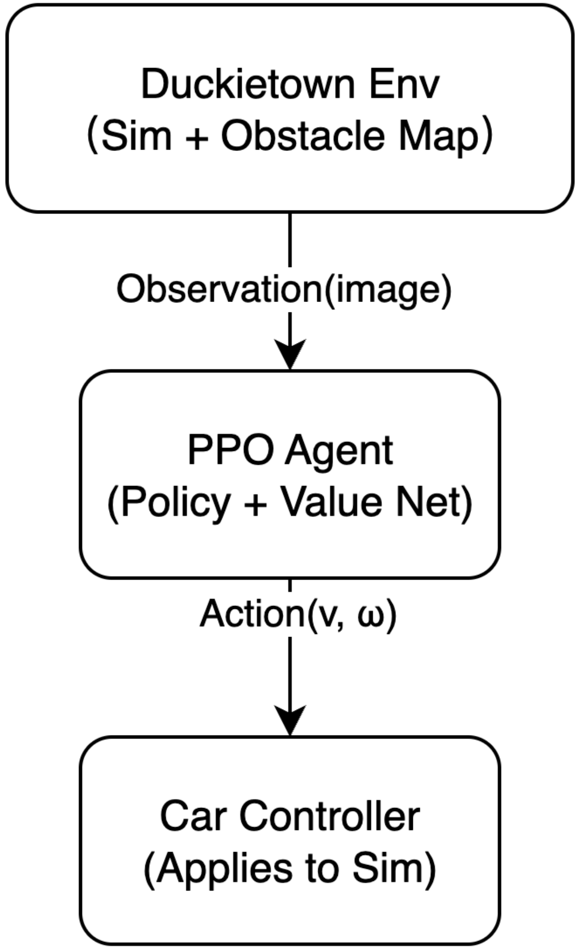
- PPO (Proximal Policy Optimization)
- Takes processed visual input and outputs control commands (v, ω)

Training Loop:

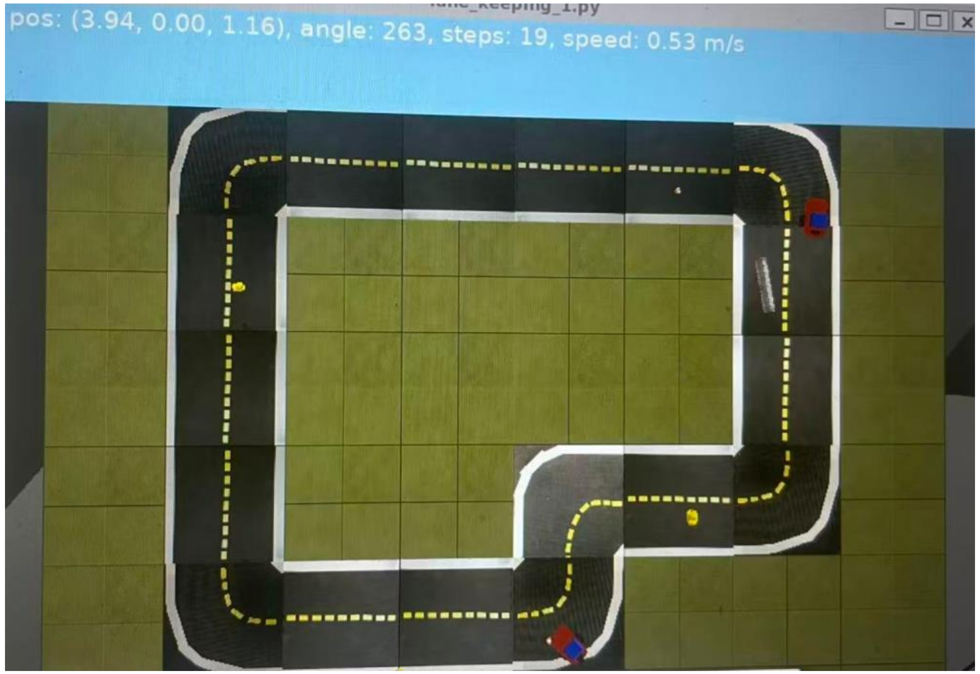
- Agent interacts with the environment
- Observes next state, reward, done signal
- Uses reward to update the policy network

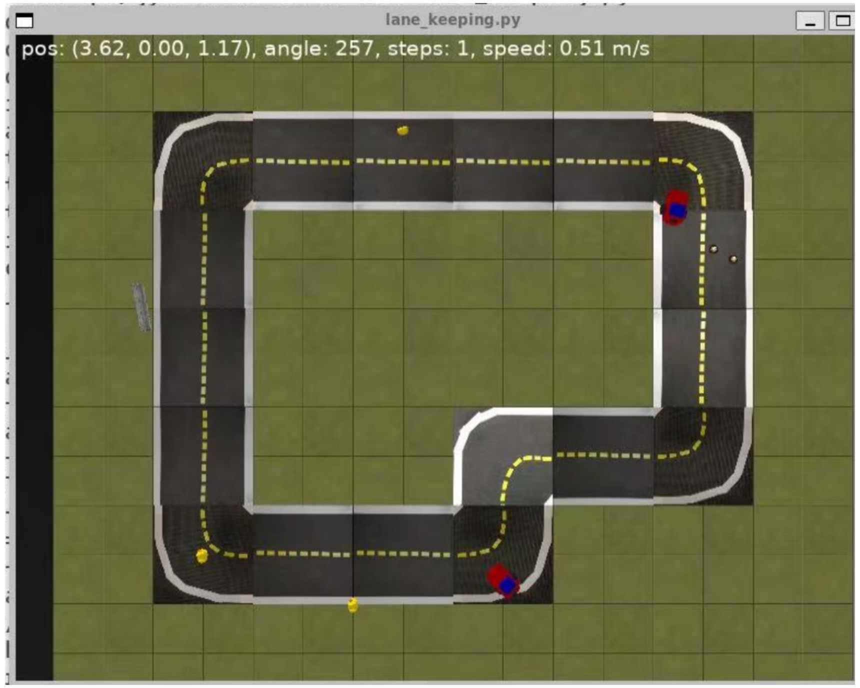
Reward Design:

- +1: Forward progress
- –1: Collision or lane exit
- –0.05: Sharp steering



Environment Randomization



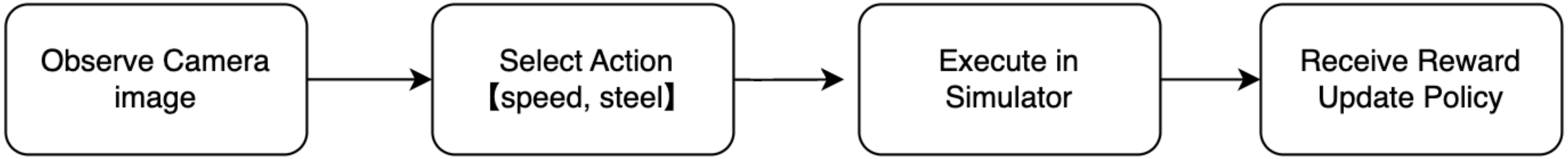


Training Process

PPO agent trained with Stable-Baselines3

CNN-based policy extracts visual features.

Training loop:



Preliminary Results – PPO Training

- **KL Approximation** & **Clip Fraction**:

→ Policy updates are large, indicating frequent clipping
- **Entropy Loss**:

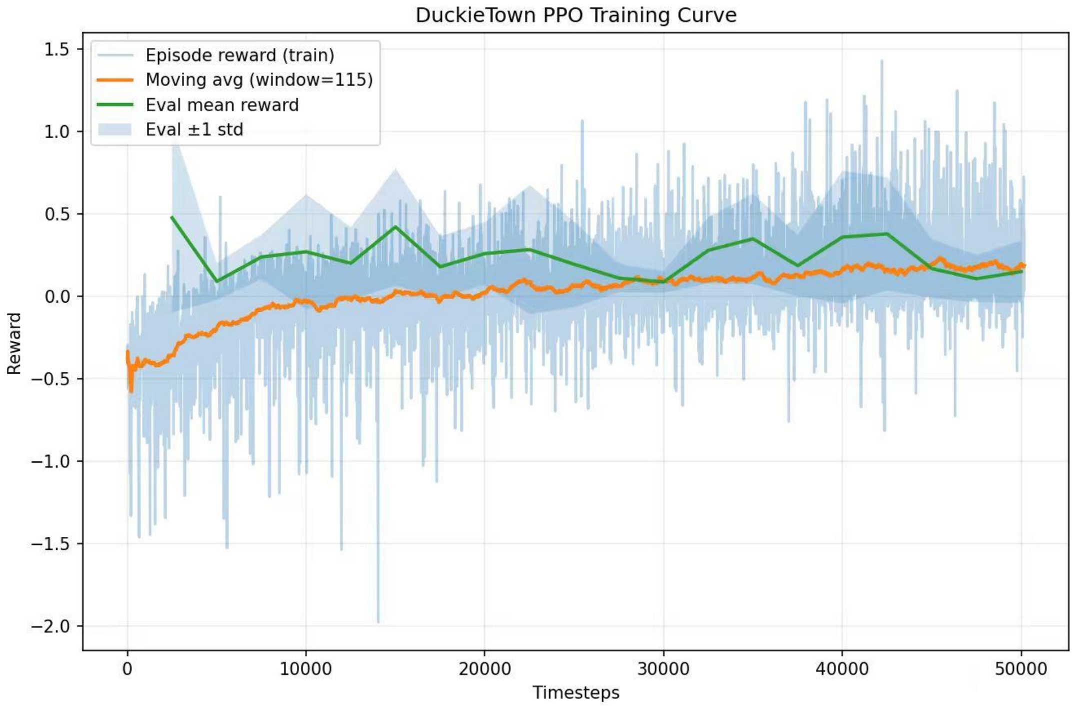
→ Policy is becoming less random, but not yet stable
- **Explained Variance**:

→ Negative values suggest value function is unreliable
- **Training Variance**:

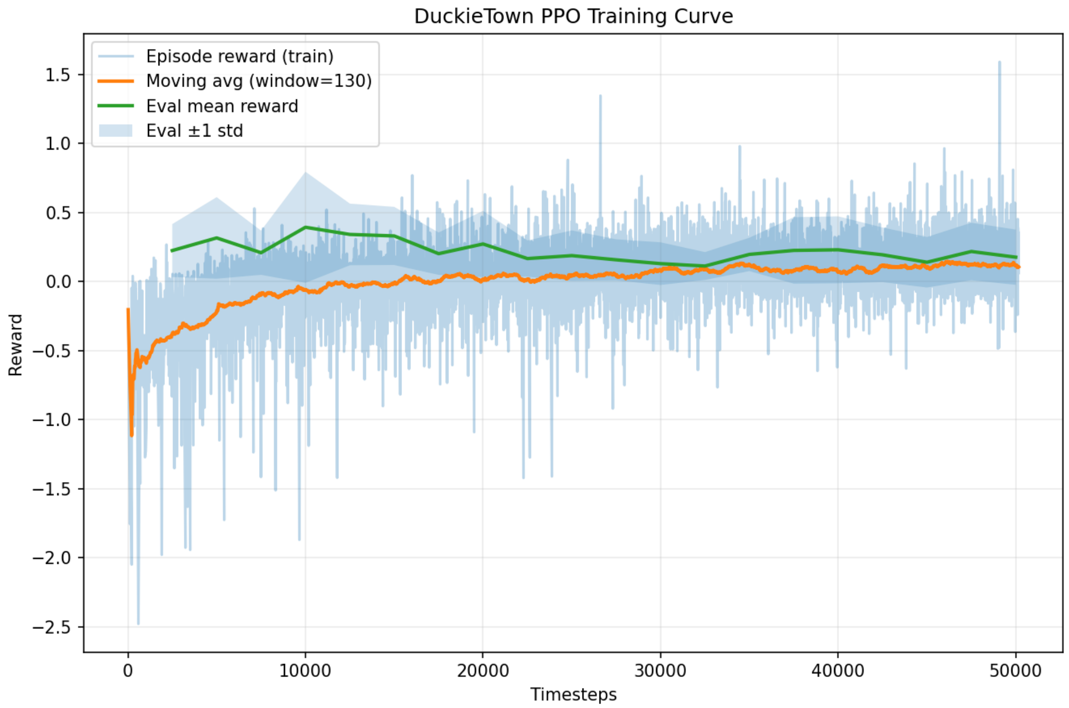
→ High variability shows partial success in lane-keeping

Metric	**Value**
KL Approximation	0.071
Clip Fraction	0.391
Entropy Loss	-2.6
Explained Variance	-0.362
Learning Rate	0.0003
Policy Gradient Loss	-0.0673
Value Loss	0.0109
Std (performance variance)	0.885

Preliminary Results – PPO Training



Learning_rate: 0.0003



Learning_rate: 0.0001

Future Directions for RL module

- **Domain Randomization:**

Add lighting and texture variability

- **Stability & Sample Efficiency:**

Smaller learning rates and better hyperparameter tuning

- **Real-world Transfer:**

Add ROS wrappers to deploy trained policy on real Duckiebot

Conclusion

- Our autonomous driving system is capable of basic lane-keeping and traffic sign recognition in the simulator

- The next step is to adapt our algorithms to physical agents, mainly the Duckiebot, in order to test them in a real traffic environment.

- Reinforcement learning provides the opportunity to realize obstical avoidance

Summary

Discussion:

Any Questions ?