**Reinforcement Learning‑Based Path Planning for a Mobile Robot**

**Abstract**

Autonomous mobile robots must plan safe, efficient routes in cluttered, partially known spaces. While classical planners like A\* and RRT are strong baselines, their reliance on handcrafted heuristics and static maps can limit adaptability. This thesis develops a reproducible benchmarking framework that pits a Proximal Policy Optimization (PPO) policy against an A\* planner inside a custom Gymnasium environment populated with procedurally generated rectangular obstacles. The agent observes normalized ego‑pose, goal direction, and omni‑directional proximity rays and selects among discrete actions (forward, turn left/right). Quantitative experiments show PPO achieves shorter average paths (≈ 758 px) but with a ≈ 6 % failure rate in tight corridors; A\* succeeds in all runs but expands more nodes and yields slightly longer paths (≈ 775 px). Visual analyses (training curves, comparative bar charts, trajectory plots) illustrate the trade‑off between adaptive learning and deterministic guarantees and provide a platform for future extensions (RRT/RRT\*, curriculum learning, sim‑to‑real).

**1. Introduction**

Mobile robots increasingly operate in semi‑structured environments such as warehouses and hospitals, where accurate, up‑to‑date maps are not always available and obstacles may change. Traditional pipelines combine a global planner (e.g., A\*) with a local controller, but require careful tuning and may degrade when assumptions are violated. Reinforcement Learning (RL) promises adaptive policies learned directly from interaction. The core research question in this work is: **Can a PPO‑trained policy deliver competitive or superior navigation performance compared to classical planners when both operate in identical, procedurally generated 2D maps with rectangular obstacles?**

**1.1 Contributions**

1. A custom Gymnasium environment with configurable rectangular obstacles, 360° proximity sensing, and discrete motion primitives.
2. A PPO‑based navigation agent trained with reward shaping that balances progress, safety, and oscillation avoidance.
3. A classical A\* baseline with grid inflation and unified metric logging for side‑by‑side comparison.
4. Visualization tools and scripts for training curves, baseline comparisons, and trajectory plots supporting qualitative analysis.

**2. Literature Review**

Graph‑based planners (A\*, D\* Lite) guarantee optimality under admissible heuristics but require discretization and frequent replanning as costs change. Sampling‑based planners (RRT/RRT\*, PRM) scale to high‑dimensional spaces yet often produce jagged, suboptimal paths unless post‑processed. In contrast, RL methods learn policies from experience: DQN pioneered deep value learning for control, while PPO stabilized policy gradients with clipped updates and has been applied successfully to navigation with LiDAR or visual inputs. Surveys highlight RL’s promise for local decision‑making, particularly when curricula and domain randomization are used. Together, these strands motivate a controlled, reproducible comparison between a learned policy and a deterministic planner on matched scenarios—a gap this thesis addresses.

**3. Environment Design**

We implement a 2D, top‑down Gymnasium environment rendered with Pygame. The arena measures 768 × 576 px; the robot is a circular agent of 12 px radius. Obstacles are axis‑aligned rectangles generated via rejection sampling subject to spacing and size constraints. Start and goal lie near left/right boundaries, respectively, with optional randomization. Observations (31‑D) include normalized position, heading unit vector, normalized goal distance and direction, and 24 ray‑cast proximity readings over 360°. The discrete action space comprises forward motion and in‑place left/right turns. Safety checks clamp forward motions when clearance is insufficient; episodes terminate on collision, goal, or after 400 steps.

**Design summary.** The environment yields reproducible, configurable scenarios and a compact observation/action interface suitable for both RL and classical baselines, enabling direct, fair comparison of methods.

**4. Agent and Reward Design**

The policy observes ego‑state, goal geometry, and omnidirectional obstacle proximity. Reward shaping balances efficiency and safety with: (i) progress reward toward the goal, (ii) small time penalties (lower for turns), (iii) alignment bonus to keep heading aligned with goal direction, (iv) safety penalty when rays read close obstacles, (v) directional heuristic to prefer turning toward free space when front is blocked, (vi) oscillation penalty to discourage dithering, and (vii) terminal ±1.0 rewards on success/collision. This shaping encourages decisive, smooth, and safe motion.

**5. RL Model Development**

We use Stable‑Baselines3 PPO with a two‑layer MLP (128‑128) for policy and value heads, rollout length 4096, batch size 1024, learning rate 3e‑4, γ = 0.99, λ = 0.95, clip range 0.2, and entropy coefficient 0.01. Training employs Monitor logging, TensorBoard, and an EvalCallback that periodically evaluates and checkpoints the best model. Recommended training duration is 200k–500k timesteps depending on map complexity.

**6. Experimental Setup**

**Scenario generation & seeding.** We evaluate PPO and A\* on identical procedurally generated maps by seeding the environment once per episode and passing the same seed base to both pipelines. Difficulty profiles control obstacle counts and sizes (e.g., *hard*: 8 obstacles).

**Baselines & logging.**

* *A*\*: Rasterizes obstacles to a grid inflated by agent radius and uses 8‑connected A\*; logs success, path length, and node expansions.
* *PPO*: Loads the trained model; performs deterministic rollouts; logs success, steps, path length, and cumulative reward.  
  Both append to a shared CSV (results/baselines.csv) with consistent schema to facilitate analysis.

**Repetition & robustness.** We repeat across difficulty tiers and seeds, enabling statistical summaries (means/medians) and reproducibility.

**7. Results and Analysis**

**7.1 Quantitative metrics**

On *hard* difficulty, 100 episodes, seed = 123:

| **Algorithm** | **Success Rate** | **Mean Path (px)** | **Median Path (px)** | **Mean Steps** | **Mean Reward** | **Mean Nodes Expanded** |
| --- | --- | --- | --- | --- | --- | --- |
| PPO | 0.94 | 757.73 | — | 151.51 | 134.21 | — |
| A\* | 1.00 | 775.12 | 769.71 | — | — | 582.76 |

PPO yields shorter average paths but misses ≈ 6 % cases—typically narrow corridors; A\* succeeds on all runs but expands more nodes and produces slightly longer paths.

**7.2 Visual analyses**

We render comparison bar charts from the CSV, export training curves from TensorBoard, and plot representative PPO trajectories. These figures—comparison\_hard.png, training\_curve.png, and trajectory\_hard.png—substantiate quantitative claims and highlight qualitative behavior (smooth PPO turns, failure modes near bottlenecks).

**7.3 Interpretation**

**Performance trade‑off.** PPO’s policy exploits geometry to shorten paths but lacks deterministic guarantees; A\*’s determinism offers perfect success given a feasible path at the cost of higher search effort. **Learning behavior.** The smoothed reward curve stabilizes after initial fluctuations, indicating the shaping balances progress and safety effectively. **Sensitivity.** Results vary with A\*’s grid resolution and PPO’s sensor/reward parameters, suggesting avenues for ablation.

**8. Discussion**

This benchmark clarifies when a learned local policy can rival or exceed a deterministic planner in path efficiency, and where it still falls short (tight passages). The unified logging and shared seeds enable transparent, apples‑to‑apples comparisons, and the visual tooling makes it straightforward to diagnose behaviors (e.g., PPO oscillations, A\* expansion spikes). The framework is intentionally extensible to additional planners (RRT/RRT\*) and more realistic sensing or dynamics, supporting nuanced trade‑off studies between optimality guarantees and empirical efficiency.

**9. Conclusion and Future Work**

**Conclusion.** We delivered a complete navigation benchmark pitting PPO against A\* on identical procedurally generated maps. PPO demonstrates competitive path efficiency with small but notable failure rates in challenging geometries; A\* remains perfectly reliable but computationally heavier. The environment, scripts, and figures are all reproducible and suitable for further study.

**Future work.** Integrate RRT/RRT\* baselines under the same CSV schema; adopt curriculum learning to push PPO success closer to 100 %; incorporate dynamic obstacles and sensor noise; and explore sim‑to‑real transfer via domain randomization and evaluation on physical robots or higher‑fidelity simulators.

**10. Methodological Artifacts & Reproducibility**

**Evaluation pipeline.** Command‑line tools evaluate\_planners.py and evaluate\_rl.py populate results/baselines.csv with harmonized columns (algorithm, difficulty, episodes, success\_rate, mean/median path length, steps, mean reward, nodes expanded). Figures are generated with visualize\_results.py and plot\_training\_curve.py, and qualitative trajectories with plot\_trajectory.py.

**Training configuration.** PPO hyperparameters, logging, and checkpointing are specified in train.py; TensorBoard summaries are exported to PNG for inclusion in the manuscript.

**Environment details.** Geometry, obstacle generation, observation model, action space, and safety checks are defined in navigator/env.py, enabling deterministic reproduction of all reported experiments.

**References**

Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*. IEEE Transactions on Systems Science and Cybernetics.  
Koenig, S., & Likhachev, M. (2002). *D* Lite\*. Proceedings of AAAI.  
LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.  
Mnih, V., et al. (2015). *Human‑level control through deep reinforcement learning*. Nature.  
Schulman, J., et al. (2017). *Proximal Policy Optimization Algorithms*. arXiv:1707.06347.  
Zhu, Y., et al. (2017). *Target‑driven Visual Navigation in Indoor Scenes using Deep RL*. ICRA.  
Long, P., et al. (2018). *Towards Optimally Decentralized Multi‑Robot Collision Avoidance via Deep RL*. ICRA.  
Kober, J., Bagnell, J. A., & Peters, J. (2013). *Reinforcement Learning in Robotics: A Survey*. International Journal of Robotics Research.  
Chen, Y., et al. (2019). *Deep Reinforcement Learning for Motion Planning with Heterogeneous Agents*. International Journal of Robotics Research.

**Appendices**

**Appendix A — Key Scripts**

* train.py: PPO training loop with evaluation callback and TensorBoard logging.
* navigator/env.py: Environment dynamics, sensors, and reward computation.
* evaluate\_rl.py, evaluate\_planners.py: Benchmarking utilities that write CSV rows.
* plot\_training\_curve.py, visualize\_results.py, plot\_trajectory.py: Figure generation.

**Appendix B — Reproducibility Checklist**

1. Create a virtual environment and install dependencies from requirements.txt.
2. Train PPO (model.learn) to 200k–500k steps with logging enabled.
3. Evaluate PPO and A\* with shared seeds; append results to results/baselines.csv.
4. Generate figures (comparison bars, training curve) and trajectory plots.
5. Archive models and CSVs with the manuscript for end‑to‑end reproducibility.

Tahir Berk Topçu  
2018556065