ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

ỨNG DỤNG REINFORCEMENT LEARNING ĐIỀU KHIỂN PHÂN TÁN HỆ ĐA ROBOT TRÁNH VA CHẠM

Application of Reinforcement Learning in Decentralized Multi-Robot Collision Avoidance Control System

LUẬN VĂN THẠC SĨ

Học viên thực hiện: Nguyễn Tấn Khôi
MSSV: 2171017
Chuyên ngành: Kỹ thuật Điều khiển - Tự động hóa
Mã số chuyên ngành: 8520216

Giảng viên hướng dẫn: TS. Phạm Việt Cường

TP. Hồ Chí Minh, Tháng 12 năm 2025

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA

ỨNG DỤNG REINFORCEMENT LEARNING ĐIỀU KHIỂN PHÂN TÁN HỆ ĐA ROBOT TRÁNH VA CHẠM

Application of Reinforcement Learning in Decentralized Multi-Robot Collision Avoidance Control System

Học viên: Nguyễn Tấn Khôi
MSSV: 2171017
Chuyên ngành: Kỹ thuật Điều khiển - Tự động hóa
Mã số: 8520216

Giảng viên hướng dẫn: TS. Phạm Việt Cường

LUẬN VĂN THẠC SĨ

TP. Hồ Chí Minh, Tháng 12 năm 2025

LỜI CẢM ƠN

Tôi xin chân thành cảm ơn TS. Phạm Việt Cường, người đã tận tình hướng dẫn và định hướng cho tôi trong suốt quá trình thực hiện luận văn này. Những góp ý quý báu của thầy đã giúp tôi hoàn thiện nghiên cứu và phát triển kỹ năng nghiên cứu khoa học.

Tôi cũng xin gửi lời cảm ơn đến các thầy cô trong Bộ môn Tự động hóa, Khoa Điện - Điện tử, Trường Đại học Bách Khoa TP.HCM đã truyền đạt kiến thức nền tảng vững chắc trong suốt quá trình học tập.

Cuối cùng, tôi xin cảm ơn gia đình, bạn bè đã luôn động viên và hỗ trợ tôi trong suốt thời gian thực hiện luận văn.

Nguyễn Tấn Khôi

TÓM TẮT

Luận văn này nghiên cứu ứng dụng học tăng cường sâu (Deep Reinforcement Learning) vào bài toán điều khiển phân tán hệ đa robot tránh va chạm. Dựa trên thuật toán Proximal Policy Optimization (PPO), nghiên cứu đề xuất các cải tiến về learning rate scheduling, value clipping và chiến lược huấn luyện hai giai đoạn để cải thiện hiệu suất và khả năng mở rộng của hệ thống.

Các cải tiến đề xuất bao gồm Adaptive Learning Rate Scheduler, Value Clipping, và việc sử dụng hai optimizer riêng biệt cho actor và critic networks đã giúp cải thiện độ ổn định và tốc độ hội tụ của quá trình huấn luyện.

Nghiên cứu cũng trình bày thiết kế phần cứng robot sử dụng cảm biến LiDAR 360 độ và động cơ encoder điều khiển bằng PID, chuẩn bị cho việc triển khai thuật toán trên robot thực tế. Kết quả từ thực nghiệm cho thấy mô hình hoàn toàn có khả năng triển khai và hoạt động tốt không chỉ trên mô phỏng mà còn ở thực tế.

LỜI CAM ĐOAN

Tôi xin cam đoan rằng luận văn *"Ứng Dụng Reinforcement Learning Điều Khiển Phân Tán Hệ Đa Robot Tránh Va Chạm"* là công trình nghiên cứu của riêng tôi dưới sự hướng dẫn của TS. Phạm Việt Cường.

Các kết quả nghiên cứu và số liệu trong luận văn là trung thực, chưa từng được ai công bố trong bất kỳ công trình nào khác. Tôi xin hoàn toàn chịu trách nhiệm về tính chính xác và trung thực của nội dung luận văn này.

TP. Hồ Chí Minh, Tháng 12 năm 2025

Nguyễn Tấn Khôi

MỤC LỤC

DANH MỤC HÌNH

DANH MỤC BẢNG

DANH MỤC KÝ HIỆU VÀ CHỮ VIẾT TẮT

CNN	Convolutional Neural Network - Mạng nơ-ron tích chập
DRL	Deep Reinforcement Learning - Học tăng cường sâu
GAE	Generalized Advantage Estimation - Ước lượng ưu thế tổng quát
HCMUT	Ho Chi Minh City University of Technology - Trường Đại học Bách Khoa TP.HCM
LiDAR	Light Detection and Ranging - Công nghệ quét laser để đo khoảng cách
POMDP	Partially Observable Markov Decision Process - Quá trình quyết định Markov quan sát từng phần
PPO	Proximal Policy Optimization - Tối ưu hóa chính sách gần kề
RL	Reinforcement Learning - Học tăng cường
a_t	Action at time step t - Hành động tại bước thời gian t
o_t	Observation at time step t - Quan sát tại bước thời gian t
r_t	Reward at time step t - Phần thưởng tại bước thời gian t
v	Linear velocity - Vận tốc tuyến tính
ω	Angular velocity - Vận tốc góc (omega)
π	Policy function - Hàm chính sách (pi)
θ	Neural network parameters - Tham số mạng nơ-ron (theta)

Ghi chú: Danh sách này sẽ được cập nhật thêm các ký hiệu khác khi xuất hiện trong các chương của luận văn.

CHƯƠNG 1

MỞ ĐẦU

1.1 Lý do chọn đề tài

Trong những năm gần đây, hệ thống đa robot đã và đang được ứng dụng rộng rãi trong nhiều lĩnh vực như kho hàng tự động, nhà máy thông minh, và logistics. Việc điều khiển nhiều robot hoạt động đồng thời trong cùng một môi trường đặt ra thách thức lớn về tránh va chạm, đặc biệt khi số lượng robot tăng lên.

Các phương pháp điều khiển tập trung truyền thống gặp khó khăn về khả năng mở rộng (scalability) khi số lượng robot lớn, do yêu cầu về tính toán và truyền thông tăng theo cấp số nhân. Ngược lại, phương pháp điều khiển phân tán, trong đó mỗi robot tự quyết định dựa trên quan sát cục bộ, cho phép hệ thống mở rộng tốt hơn.

Học tăng cường sâu (Deep Reinforcement Learning - DRL) đã chứng minh khả năng giải quyết các bài toán điều khiển phức tạp. Đặc biệt, thuật toán Proximal Policy Optimization (PPO) được đánh giá cao về độ ổn định và hiệu quả trong việc huấn luyện các policy đặc biệt trong việc Supervised Fine-Tuning các mô hình ngôn ngữ lớn. Tuy nhiên, việc áp dụng DRL vào bài toán đa robot với số lượng lớn vẫn còn nhiều thách thức cần nghiên cứu.

1.2 Mục tiêu nghiên cứu

Mục tiêu chính của luận văn là nghiên cứu và phát triển thuật toán điều khiển phân tán cho hệ đa robot tránh va chạm sử dụng học tăng cường sâu. Cụ thể:

1. Nghiên cứu thuật toán PPO và các phương pháp học tăng cường cho bài toán đa robot.
2. Huấn luyện mô hình neural network có khả năng điều khiển robots tránh va chạm trong môi trường mô phỏng.
3. Đề xuất các cải tiến về learning rate scheduling, value clipping và policy để đáp ứng với thực nghiệm.
4. Thiết kế phần cứng robot prototype với cảm biến LiDAR, Jetson và Arduino đơn giản và chi phí thấp.
5. Triển khai và kiểm nghiệm thực tế mô hình trên robot trong thực nghiệm.

1.3 Đối tượng và phạm vi nghiên cứu

1.3.1 Đối tượng nghiên cứu

Đối tượng nghiên cứu là hệ thống đa robot di động (mobile robots) với các đặc điểm:

- **Kiểu robot:** Nonholonomic robots (robot không toàn hướng) di chuyển trên mặt phẳng 2D
- **Cảm biến:** LiDAR 360° với độ phân giải 0.8°
- **Hành động:** Điều khiển vận tốc tuyến tính v và vận tốc góc ω
- **Số lượng:** 44 robots hoạt động đồng thời trong mô phỏng; 2 robot trong thực nghiệm

1.3.2 Phạm vi nghiên cứu

Luận văn tập trung vào các khía cạnh sau:

- Thuật toán học tăng cường PPO cho bài toán điều khiển phân tán
- Môi trường mô phỏng 2D với chướng ngại vật tĩnh
- Quan sát cục bộ (local observation) từ cảm biến LiDAR
- Không sử dụng truyền thông giữa các robot (fully decentralized)
- Tuning PID ổn định điều khiển robot thực nghiệm
- Triển khai được model huấn luyện từ mô phỏng vào robot thực nghiệm
- Sensor fusing để robot có thể hoạt động và xác định chính xác vị trí của nó trong môi trường với các cảm biến giá rẻ
- Mục tiêu: Đạt tỉ lệ thành công robot di chuyển từ vị trí xuất phát đến đích mà không va chạm trên 80%

Ngoài phạm vi: Môi trường 3D, multi-task learning, xác định vị trí chính xác robot.

1.4 Ý nghĩa khoa học và thực tiễn

1.4.1 Ý nghĩa khoa học

Luận văn đóng góp vào lĩnh vực nghiên cứu đa robot và học tăng cường thông qua:

- Đánh giá tính ứng dụng của mô hình học tăng cường trong điều khiển hệ robot phân tán
- Đề xuất các cải tiến về thuật toán huấn luyện PPO:
  - *Adaptive Learning Rate Scheduler*: Duy trì learning rate khi performance cải thiện
  - *Value Clipping*: Giảm instability trong quá trình huấn luyện
  - *Separate Optimizers*: Sử dụng learning rate khác nhau cho actor và critic
  - *Learning Rate Warmup*: Tăng dần learning rate trong giai đoạn đầu
- Nghiên cứu khả năng thực tiễn của mô hình với robot Nonholonomic

1.4.2 Ý nghĩa thực tiễn

Kết quả nghiên cứu có thể ứng dụng vào:

- **Kho hàng tự động (Automated Warehouses):** Điều khiển nhiều robot AGV (Automated Guided Vehicle) di chuyển hàng hóa hiệu quả.
- **Nhà máy thông minh (Smart Factories):** Phối hợp nhiều robot công nghiệp trong dây chuyền sản xuất.
- **Logistics và vận tải:** Quản lý đội xe tự hành trong khu vực hạn chế.

Kết quả đạt được trên robot thực nghiệm chứng minh tính khả thi của phương pháp trong điều kiện thực tế với mô hình phần cứng đơn giản và dễ dàng mở rộng với số lượng robot lớn.

CHƯƠNG 2

TỔNG QUAN

2.1 Giới thiệu về hệ đa robot

Hệ đa robot (Multi-Robot Systems - MRS) là hệ thống bao gồm nhiều robot hoạt động đồng thời trong cùng một môi trường để thực hiện một hoặc nhiều nhiệm vụ chung. Hệ đa robot có thể được phân loại dựa trên mức độ phối hợp giữa các robot: từ hoàn toàn độc lập (independent) đến phối hợp chặt chẽ (tightly coordinated), và dựa trên kiến trúc điều khiển: tập trung (centralized) hoặc phân tán (decentralized).

Trong những năm gần đây, hệ đa robot đã được ứng dụng rộng rãi trong nhiều lĩnh vực thực tế. Tại các kho hàng tự động như Amazon Centers, hàng trăm robot di chuyển đồng thời để vận chuyển hàng hóa, giúp tăng hiệu suất và giảm chi phí nhân công. Trong sản xuất công nghiệp, các robot di động (Automated Guided Vehicles - AGV) phối hợp vận chuyển nguyên liệu và sản phẩm giữa các trạm làm việc. Hệ đa robot cũng được ứng dụng trong logistics, nông nghiệp thông minh, cứu hộ thảm họa, và thăm dò không gian.

Một trong những thách thức chính trong điều khiển hệ đa robot là vấn đề tránh va chạm (collision avoidance). Khi số lượng robot tăng lên, không gian hoạt động trở nên chật hẹp và xác suất va chạm giữa các robot hoặc với chướng ngại vật tăng cao. Điều này đòi hỏi mỗi robot phải có khả năng cảm nhận môi trường xung quanh, dự đoán chuyển động của các robot khác, và điều chỉnh quỹ đạo của mình để tránh va chạm trong khi vẫn đạt được mục tiêu đề ra. Vấn đề trở nên phức tạp hơn khi môi trường có chướng ngại vật động, không gian hạn chế, hoặc yêu cầu thời gian phản ứng nhanh.

2.2 Các phương pháp tránh va chạm truyền thống

Các phương pháp tránh va chạm truyền thống thường dựa trên mô hình toán học và quy tắc xác định (rule-based) để đảm bảo an toàn cho robot. Các phương pháp này có ưu điểm về tính minh bạch và khả năng chứng minh an toàn, nhưng thường gặp khó khăn khi áp dụng cho hệ đa robot với số lượng lớn.

Artificial Potential Field (APF) [1] là một trong những phương pháp sớm nhất và đơn giản nhất. Phương pháp này mô hình hóa môi trường như một trường thế năng, trong đó mục tiêu tạo ra lực hút (attractive force) và các chướng ngại vật tạo ra lực đẩy (repulsive force). Robot di chuyển theo hướng của gradient âm của trường thế năng. Tuy nhiên, APF thường gặp vấn đề local minima, trong đó robot có thể bị mắc kẹt tại điểm cân bằng không phải là mục tiêu.

Rapidly-exploring Random Tree (RRT) [2] và biến thể tối ưu RRT* [3] là các thuật toán lấy mẫu ngẫu nhiên (sampling-based) để tìm đường đi trong không gian cấu hình. RRT xây dựng một cây tìm kiếm bằng cách mở rộng ngẫu nhiên từ điểm khởi đầu đến mục tiêu. RRT* cải thiện RRT bằng cách tối ưu hóa chi phí đường đi. Các phương pháp này hiệu quả cho bài toán tìm đường trong không gian phức tạp, nhưng không phù hợp cho điều khiển thời gian thực với nhiều robot do chi phí tính toán cao.

Optimal Reciprocal Collision Avoidance (ORCA) [4] là phương pháp phổ biến cho tránh va chạm đa tác tử. ORCA tính toán vận tốc an toàn cho mỗi robot bằng cách xác định một tập hợp các vận tốc không gây va chạm trong một khoảng thời gian nhất định. Mỗi robot chọn vận tốc gần nhất với vận tốc mong muốn trong tập hợp này. ORCA đảm bảo không va chạm nếu tất cả robot tuân theo quy tắc, nhưng phương pháp này yêu cầu robot có mô hình động học holonomic (có thể di chuyển theo mọi hướng), trong khi nhiều robot thực tế là

nonholonomic (ví dụ: differential drive robot).

Bảng 2.1 tổng hợp so sánh các phương pháp truyền thống. Có thể thấy, các phương pháp này có ưu điểm về tính toán nhanh và khả năng chứng minh an toàn, nhưng hạn chế chính là khó khăn khi mở rộng cho hệ đa robot với số lượng lớn (>50 robots) và môi trường động phức tạp. Điều này tạo động lực cho việc nghiên cứu các phương pháp dựa trên học máy có khả năng học từ dữ liệu và thích nghi với môi trường.

Bảng 2.1. So sánh các phương pháp tránh va chạm truyền thống

Phương pháp	**Ưu điểm**	**Nhược điểm**	**Khả năng mở rộng**
Artificial Potential Field [1]	Đơn giản, tính toán nhanh, dễ triển khai	Local minima, khó điều chỉnh tham số, không đảm bảo tối ưu	Khó với >20 robots
RRT/RRT* [2], [3]	Tìm đường trong không gian phức tạp, RRT* đảm bảo tối ưu tiệm cận	Chi phí tính toán cao, không phù hợp real-time, cần re-plan thường xuyên	Không phù hợp cho đa robot
ORCA [4]	Đảm bảo không va chạm, phân tán, phù hợp real-time	Yêu cầu holonomic robot, không tối ưu đường đi, cần communication	Tốt đến 50 robots

2.3 Các phương pháp dựa trên học sâu

Trong những năm gần đây, học sâu (Deep Learning - DL) [5] đã đạt được thành công vượt bậc trong nhiều lĩnh vực như thị giác máy tính, xử lý ngôn ngữ tự nhiên, và điều khiển robot. Kết hợp học sâu với học tăng cường (Reinforcement Learning - RL) tạo ra phương pháp học tăng cường sâu (Deep Reinforcement Learning - DRL) có khả năng học các chính sách điều khiển phức tạp trực tiếp từ dữ liệu cảm biến.

Deep Q-Network (DQN) [6] là một trong những bước đột phá đầu tiên của DRL, cho phép agent học chơi các trò chơi Atari ở mức độ con người chỉ từ pixels. DQN sử dụng mạng nơ-ron sâu để xấp xỉ hàm giá trị Q và kết hợp experience replay để ổn định quá trình học. Tuy nhiên, DQN được thiết kế cho không gian hành động rời rạc, không phù hợp cho điều khiển robot với hành động liên tục.

Để giải quyết vấn đề hành động liên tục, các thuật toán policy gradient như **Asynchronous Advantage Actor-Critic (A3C)** [7] được phát triển. A3C sử dụng nhiều agent song song để thu thập dữ liệu và cập nhật policy, giúp tăng tốc độ học và ổn định training. Kiến trúc Actor-Critic bao gồm hai mạng: Actor đưa ra hành động và Critic đánh giá hành động đó. Tuy nhiên, A3C vẫn gặp vấn đề về độ ổn định khi learning rate quá lớn.

Ứng dụng DRL cho điều khiển robot di động đã được nghiên cứu rộng rãi. **Tai et al.** [8] đề xuất phương pháp học navigation không cần bản đồ (mapless navigation) cho robot di động sử dụng DRL với đầu vào là dữ liệu LiDAR. Phương pháp này cho phép robot học tránh chướng ngại vật và di chuyển đến mục tiêu trong môi trường chưa biết trước. Điểm mạnh là không cần xây dựng bản đồ chi tiết, nhưng hạn chế là chỉ xét robot đơn lẻ.

Đối với bài toán đa robot, có hai cách tiếp cận chính: tập trung (centralized) và phân tán (decentralized). Cách tiếp cận tập trung sử dụng một agent trung tâm điều khiển tất cả robots, có ưu điểm là dễ tối ưu hóa toàn cục nhưng gặp khó khăn về khả năng mở rộng và đòi hỏi communication đáng tin cậy. Ngược lại, cách tiếp cận phân tán [9] cho phép mỗi robot có policy riêng và ra quyết định độc lập dựa trên quan sát cục bộ. Cách tiếp cận này mở rộng tốt hơn cho số lượng robot lớn và không yêu cầu communication, phù hợp cho ứng dụng thực tế.

2.4 Phương pháp PPO với hệ robot phi tập trung

Bài báo của Long et al. [10] đề xuất phương pháp điều khiển phân tán cho hệ đa robot tránh va chạm sử dụng deep reinforcement learning. Đây là công trình nền tảng cho luận văn này.

2.4.1 Mô hình bài toán

Bài toán được công thức hóa như một Partially Observable Markov Decision Process (POMDP) cho mỗi robot. Điểm khác biệt quan trọng so với các phương pháp trước (ORCA, GA3C-CADRL) là không giả định "perfect sensing mỗi robot chỉ quan sát được môi trường xung quanh từ sensor của chính nó, không biết chính xác vị trí và ý định của robots khác.

Observation space: Mỗi robot quan sát $\mathbf{o}^t = [\mathbf{o}_z^t, \mathbf{o}_g^t, \mathbf{o}_v^t]$ gồm:

- Laser scan 180° với 512 beams (tia) từ 3 frames liên tiếp
- Vị trí tương đối của goal (khoảng cách và góc)
- Vận tốc hiện tại (linear và angular).

Action space: Hành động là vận tốc $\mathbf{a}^t = [v^t, w^t]$ với $v^t \in (0, 1.0)$ m/s (không cho phép lùi) và $w^t \in (-1.0, 1.0)$ rad/s, được sample từ stochastic policy π_θ chia sẻ bởi tất cả robots.

Objective (Mục tiêu): Minimize thời gian đến đích trung bình của tất cả robots trong khi đảm bảo không va chạm với nhau ($\|\mathbf{p}_t^i - \mathbf{p}_t^j\| > 2R$) và với vật cản.

2.4.2 Hàm reward

Reward function gồm ba thành phần: $r_t^i = ({}^g r)_t^i + ({}^c r)_t^i + ({}^w r)_t^i$

- **Goal reward**: $r_{\text{arrival}} = +15$ khi đến đích ($\|\mathbf{p}_t^i - \mathbf{g}_i\| < 0.1$); trong khi di chuyển, reward $\omega_g(\|\mathbf{p}_{t-1}^i - \mathbf{g}_i\| - \|\mathbf{p}_t^i - \mathbf{g}_i\|)$ với $\omega_g = 2.5$ tỷ lệ với khoảng cách tiến gần goal.
- **Collision penalty**: $r_{\text{collision}} = -15$ khi va chạm với robot khác ($\|\mathbf{p}_t^i - \mathbf{p}_t^j\| < 2R$) hoặc vật cản. Giá trị tuyệt đối bằng arrival reward để cân bằng giữa đến đích nhanh và tránh va chạm.
- **Smoothness penalty**: Phạt nhẹ $\omega_w |w_t^i| = -0.1|w_t^i|$ khi vận tốc góc lớn ($|w| > 0.7$) để khuyến khích chuyển động mượt mà.

2.4.3 Kiến trúc mạng nơ-ron

Policy network π_θ ánh xạ trực tiếp từ dữ liệu quan sát gốc của cảm biến (raw sensor observation) sang tín hiệu điều khiển (steering command), gồm 4 hidden layers:

Layers 1-2 (Conv1D): Xử lý laser scan với 2 lớp tích chập 1D:

- Conv1D(32 filters, kernel=5, stride=2) + ReLU
- Conv1D(32 filters, kernel=3, stride=2) + ReLU

Hai lớp này trích xuất đặc trưng không gian từ 1536 laser readings (3×512) thành 4032 features. Việc dùng 3 frames liên tiếp giúp mạng học thông tin về chuyển động tương đối.

Layer 3 (FC): 256 units, flatten output từ Conv1D và tổng hợp đặc trưng không gian.

Layer 4 (FC): 128 units. Lớp này concatenate với goal position (2D) và current velocity (2D) để tích hợp thông tin từ LiDAR, nhiệm vụ, và trạng thái động học.

Output layer: 2 units với Sigmoid (cho $v \in (0, 1)$) và Tanh (cho $w \in (-1, 1)$). Output là mean của Gaussian distribution $\mathcal{N}(\mathbf{v}_{\text{mean}}, \mathbf{v}_{\text{logstd}})$ để sample action. Stochastic policy cho phép exploration trong training.

Value network: Cùng kiến trúc nhưng output 1 unit (linear activation) để ước lượng expected return. Không share parameters với policy network.

2.4.4 Thuật toán huấn luyện

Centralized learning, decentralized execution: Một policy duy nhất π_θ được huấn luyện từ experiences của tất cả N robots đồng thời (sample efficiency cao, diverse experiences). Khi thực thi, mỗi robot sample action từ policy:

$$\mathbf{a}^t \sim \pi_\theta(\mathbf{a}^t|\mathbf{o}^t) \tag{2.1}$$

chỉ dựa vào observation riêng, không cần giao tiếp giữa các robot

PPO objective: huật toán PPO sử dụng hàm mục tiêu có thêm một thành phần phạt KL (adaptive KL penalty). Hàm mục tiêu được viết như sau:

$$L^{PPO}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)}{\pi_{\text{old}}(\mathbf{a}_t|\mathbf{o}_t)} \hat{A}_t \right] - \beta \text{KL}[\pi_{\text{old}} \| \pi_\theta] \tag{2.2}$$

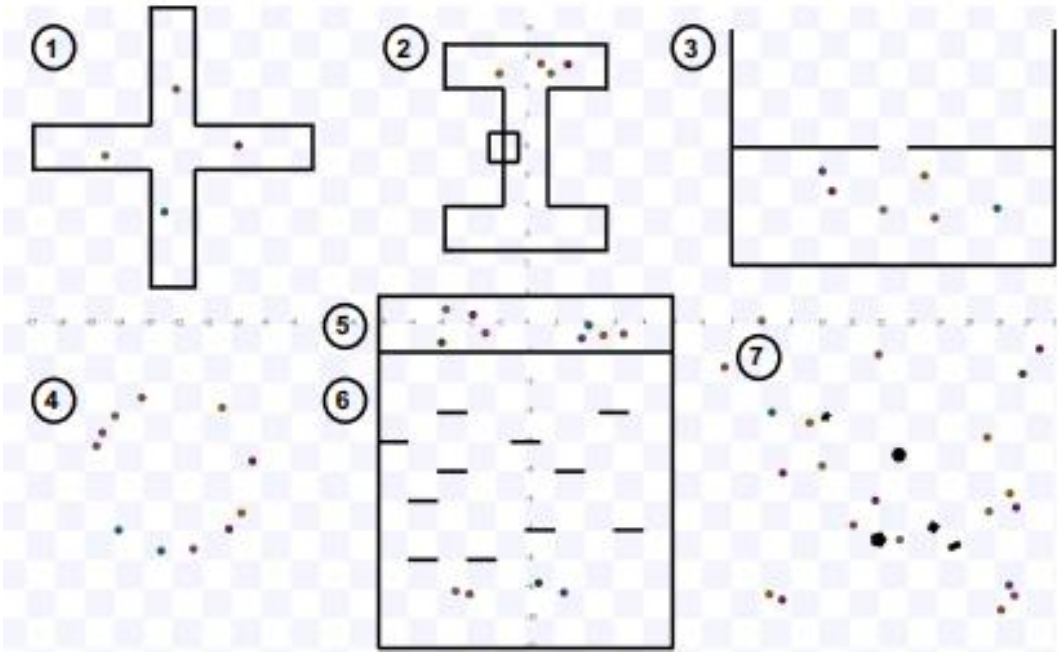
trong đó $\hat{A}_t$ là ước lượng độ lợi (advantage estimate), β được điều chỉnh tự động để cân bằng tốc độ học và độ ổn định.

Advantage estimation: Sử dụng GAE với $\lambda = 0.95$, $\gamma = 0.99$:

$$\hat{A}_t = \sum_{l=0}^{T} (\gamma\lambda)^l \delta_t, \quad \delta_t = r_t + \gamma V_\phi(\mathbf{s}_{t+1}) - V_\phi(\mathbf{s}_t) \tag{2.3}$$

2.4.5 Môi trường và kịch bản huấn luyện

Môi trường simulation: Sử dụng thư viện mô phỏng Stage với 2 môi trường training tương ứng với 2 giai đoạn khác nhau. Giai đoạn 1: môi trường đơn giản, ít vật cản. Giai đoạn 2: 7 scenarios đa dạng: (1-3, 5-6) Môi trường có tường/vật cản với điểm bắt đầu/kết thúc cố định, (4) Trường hợp vòng tròn với robots đổi chỗ qua tâm, (7) Random scenario với vị trí robots và obstacles ngẫu nhiên. Tất cả robots di chuyển đồng thời (parallel).



Hình 2.1. 7 scenarios trong huấn luyện

Cấu hình sensor: Laser scanner 180° gắn ở phía trước robot, range 4m, 512 tia/1 lần quét. Observations được normalize (mean=0, std=1) trên toàn bộ training data để cải thiện độ ổn định và hội tụ.

2.4.6 Chiến lược huấn luyện hai giai đoạn

Áp dụng curriculum learning [11] với 2 stages để tránh stuck ở local optima:

Stage 1 - Foundation learning: Huấn luyện24 robots trên random scenario không có obstacles (6 hours, 300 iterations). Mục tiêu: học basic collision avoidance và goal-reaching. Kết thúc khi success rate > 0.9.

Stage 2 - Transfer learning: Tiếp tục huấn luyện 58 robots trên tất cả 7 scenarios phức tạp (6 hours, 300 iterations) với learning rate thấp hơn (2×10^{-5}). Khởi tạo từ Stage 1 weights. Mục tiêu: generalize cho large-scale, obstacles, diverse environments.

Kết quả: Pre-training ở Stage 1 giúp đạt higher rewards ở Stage 2 so với training from scratch. Total 12 hours, 4.8M timesteps. Algorithm scale tốt do parallel data collection.

2.5 Cơ sở lý thuyết

2.5.1 Các thuật toán tối ưu

Các thuật toán tối ưu đóng vai trò then chốt trong việc huấn luyện mạng nơ-ron sâu. Mục tiêu của các thuật toán này là tìm tập tham số θ^* minimize hàm loss $J(\theta)$. Trong phần này, chúng ta sẽ phân tích chi tiết các thuật toán từ cơ bản đến hiện đại được sử dụng rộng rãi trong deep learning.

2.5.1.1 Gradient Descent

Gradient Descent (GD) là thuật toán tối ưu cơ bản nhất, trong đó tham số được cập nhật theo hướng ngược với gradient của hàm loss:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta J(\theta_t) \tag{2.4}$$

trong đó $\alpha > 0$ là learning rate (tốc độ học), và $\nabla_\theta J(\theta_t)$ là gradient của loss function tại θ_t.

Batch Gradient Descent: Tính gradient trên toàn bộ dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$:

$$\nabla_\theta J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(f_\theta(x^{(i)}), y^{(i)}) \tag{2.5}$$

trong đó ℓ là loss function cho một sample (ví dụ: Mean Squared Error, Cross Entropy).

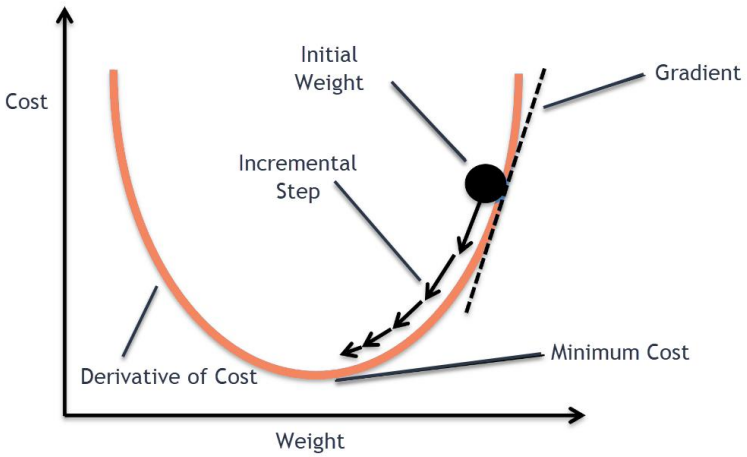
Ưu điểm:

- Cập nhật theo hướng chính xác nhất (true gradient)
- Hội tụ đến minimum cho hàm convex
- Đơn giản, dễ implement

Nhược điểm:

- Tính toán gradient trên toàn bộ dataset rất chậm với dữ liệu lớn
- Không thể update online khi có data mới
- Có thể bị mắc kẹt tại local minima hoặc saddle points
- Tốc độ hội tụ phụ thuộc nhiều vào learning rate α

Chọn learning rate: Nếu α quá nhỏ $\rightarrow$ hội tụ chậm. Nếu α quá lớn $\rightarrow$ oscillation hoặc divergence. Phương pháp thử nghiệm: bắt đầu với $\alpha = 0.01$, tăng/giảm theo log scale (0.001, 0.01, 0.1, 1.0) và chọn giá trị cho loss giảm nhanh nhất mà không bị oscillation.



Hình 2.2. Minh họa Gradient Descent: Tham số di chuyển theo hướng ngược gradient để giảm loss

2.5.1.2 Stochastic Gradient Descent (SGD)

Để giải quyết vấn đề tính toán chậm của Batch GD, Stochastic Gradient Descent chỉ sử dụng một sample ngẫu nhiên hoặc một mini-batch để ước lượng gradient:

$$\nabla_\theta J(\theta) \approx \nabla_\theta \ell(f_\theta(x^{(i)}), y^{(i)}) \tag{2.6}$$

hoặc với mini-batch size B:

$$\nabla_\theta J(\theta) \approx \frac{1}{B} \sum_{i \in \mathcal{B}} \nabla_\theta \ell(f_\theta(x^{(i)}), y^{(i)}) \tag{2.7}$$

Thuật toán SGD với mini-batch:

1. Khởi tạo tham số θ_0 (thường từ phân phối chuẩn hoặc Xavier/He initialization)
2. Đặt learning rate α, batch size B
3. Lặp cho đến khi hội tụ:
    - Shuffle dataset $\mathcal{D}$
    - Chia dataset thành mini-batches: $\{\mathcal{B}_1, \mathcal{B}_2, \ldots, \mathcal{B}_M\}$
    - Với mỗi mini-batch $\mathcal{B}_j$:
        - Tính loss: $J_j(\theta) = \frac{1}{B} \sum_{(x,y) \in \mathcal{B}_j} \ell(f_\theta(x), y)$
        - Tính gradient: $g_j = \nabla_\theta J_j(\theta)$
        - Cập nhật: $\theta \leftarrow \theta - \alpha g_j$

Ưu điểm:

- Nhanh hơn nhiều so với Batch GD, có thể handle big data
- Update online, có thể học từ streaming data
- Noise trong gradient giúp thoát local minima
- Hỗ trợ GPU tốt hơn với mini-batch

Nhược điểm:

- Gradient noisy, cập nhật không ổn định
- Learning rate cố định không tối ưu cho mọi tham số
- Vẫn có thể oscillate xung quanh minimum
- Cần tuning cẩn thận batch size và learning rate

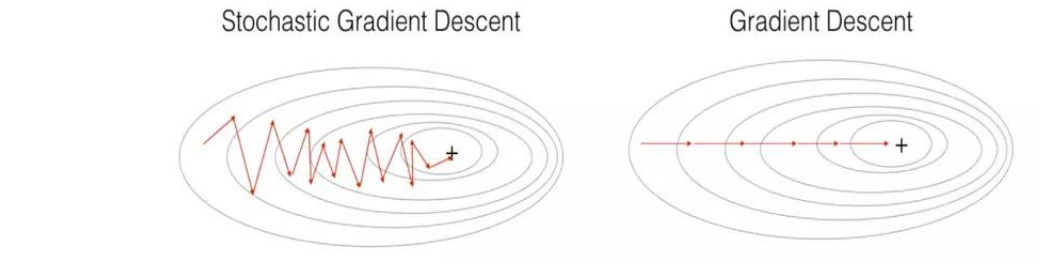
Chọn batch size: Batch size nhỏ (32-64) $\rightarrow$ update nhanh, gradient noisy, generalization tốt. Batch size lớn (256-512) $\rightarrow$ gradient stable, tận dụng GPU, có thể overfitting. Với RL và PPO, thường dùng batch size lớn (1024-4096 timesteps) để ổn định policy update.

2.5.1.3 Momentum-based Gradient Descent

Momentum giải quyết vấn đề oscillation của SGD bằng cách tích lũy hướng di chuyển từ các bước trước:

$$\begin{aligned} v_{t+1} &= \beta v_t + (1 - \beta) g_t \\ \theta_{t+1} &= \theta_t - \alpha v_{t+1} \end{aligned} \tag{2.8}$$

trong đó:



Hình 2.3. So sánh Stochastic Gradient Descent và Gradient Descent: SGD có noise nhưng hội tụ nhanh hơn

- v_t là velocity (trung bình trượt của gradient)
- $g_t = \nabla_\theta J(\theta_t)$ là gradient tại bước t
- $\beta \in [0, 1]$ là momentum coefficient (thường $\beta = 0.9$)
- α là learning rate

Giải thích trực quan: Hãy tưởng tượng một quả bóng lăn xuống dốc. Momentum cho phép quả bóng tích lũy vận tốc khi đi xuống và có thể vượt qua các local minima nhỏ nhờ quán tính. Thuật toán này giúp:

- Tăng tốc trong các hướng có gradient nhất quán
- Giảm oscillation trong các hướng có gradient thay đổi
- Vượt qua saddle points và local minima phẳng

Nesterov Accelerated Gradient (NAG): Cải tiến momentum bằng cách "nhìn trước"vị trí tiếp theo:

$$\begin{aligned} v_{t+1} &= \beta v_t + (1 - \beta)\nabla_\theta J(\theta_t - \alpha\beta v_t) \\ \theta_{t+1} &= \theta_t - \alpha v_{t+1} \end{aligned} \tag{2.9}$$

NAG tính gradient tại vị trí "lookahead"$\theta_t - \alpha\beta v_t$ thay vì θ_t, giúp điều chỉnh momentum kịp thời trước khi overshoot.

Ưu điểm so với SGD:

- Hội tụ nhanh hơn, ít oscillation hơn
- Ổn định hơn với learning rate lớn
- Hiệu quả với ravines (gradient lớn theo một hướng, nhỏ theo hướng khác)

Nhược điểm:

- Thêm một hyperparameter (β) cần tuning
- Vẫn sử dụng learning rate global cho tất cả parameters
- Không thích nghi với sparse gradients

2.5.1.4 RMSProp (Root Mean Square Propagation)

RMSProp giải quyết vấn đề learning rate toàn cục bằng cách điều chỉnh learning rate riêng cho từng tham số dựa trên magnitude của gradients gần đây:

$$\begin{aligned} E[g^2]_{t+1} &= \beta E[g^2]_t + (1-\beta) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{E[g^2]_{t+1} + \epsilon}} g_t \end{aligned} \tag{2.10}$$

trong đó:

- $E[g^2]_t$ là trung bình trượt của bình phương gradient (second moment)
- $g_t = \nabla_\theta J(\theta_t)$ là gradient
- β là decay rate (thường 0.9 hoặc 0.99)
- ϵ là hằng số nhỏ (thường 10^{-8}) để tránh chia cho 0
- Phép chia và căn bậc hai được thực hiện element-wise

Ý tưởng chính: Nếu gradient của một tham số thường lớn $\rightarrow E[g^2]$ lớn $\rightarrow$ learning rate giảm. Nếu gradient nhỏ $\rightarrow$ learning rate tăng. Điều này giúp cân bằng tốc độ học giữa các chiều (dimensions) khác nhau.

Lợi ích của adaptive learning rate:

- Tham số với gradient lớn, frequent updates $\rightarrow$ LR giảm, tránh oscillation
- Tham số với gradient nhỏ, sparse updates $\rightarrow$ LR tăng, hội tụ nhanh hơn
- Phù hợp với non-stationary objectives (objective thay đổi theo thời gian)
- Hiệu quả với sparse data (NLP, recommendation systems)

Ưu điểm:

- Adaptive learning rate cho từng tham số
- Hiệu quả với non-convex optimization
- Ít nhạy cảm với global learning rate α
- Phù hợp với recurrent neural networks

Nhược điểm:

- $E[g^2]$ có thể tích lũy và trở nên rất lớn, làm learning rate quá nhỏ
- Không có momentum để tăng tốc trong hướng consistent
- Vẫn cần tuning learning rate α và decay rate β

2.5.1.5 Adam (Adaptive Moment Estimation)

Adam [12] kết hợp momentum (first moment) và RMSProp (second moment) để tạo ra thuật toán tối ưu hiện đại nhất, được sử dụng rộng rãi nhất trong deep learning:

$$\begin{aligned} m_{t+1} &= \beta_1 m_t + (1-\beta_1) g_t \\ v_{t+1} &= \beta_2 v_t + (1-\beta_2) g_t^2 \\ \hat{m}_{t+1} &= \frac{m_{t+1}}{1-\beta_1^{t+1}} \\ \hat{v}_{t+1} &= \frac{v_{t+1}}{1-\beta_2^{t+1}} \\ \theta_{t+1} &= \theta_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}} + \epsilon} \end{aligned} \tag{2.11}$$

trong đó:

- m_t là first moment estimate (trung bình gradient - momentum)
- v_t là second moment estimate (trung bình bình phương gradient - variance)
- $\beta_1, \beta_2 \in [0, 1)$ là decay rates (default: $\beta_1 = 0.9$, $\beta_2 = 0.999$)
- $\hat{m}_t, \hat{v}_t$ là bias-corrected moments
- ϵ là hằng số nhỏ (default: 10^{-8})
- t là iteration number

Bias correction: Ở các bước đầu tiên (t nhỏ), m_t và v_t bị bias về 0 do khởi tạo $m_0 = v_0 = 0$. Chia cho $(1-\beta_1^{t+1})$ và $(1-\beta_2^{t+1})$ giúp khử bias này. Khi $t \to \infty$, $(1-\beta_1^{t+1}) \to 1$ và bias correction không còn ảnh hưởng.

Giải thích các thành phần:

1. **First moment m_t (momentum):**

- Tích lũy hướng di chuyển từ các bước trước
- Giúp tăng tốc trong hướng nhất quán
- Giảm oscillation khi gradient thay đổi dấu

2. **Second moment v_t (adaptive learning rate):**

- Theo dõi magnitude của gradients
- Điều chỉnh learning rate riêng cho từng tham số
- Tham số với gradient lớn → LR nhỏ, tham số với gradient nhỏ → LR lớn

3. **Update rule:**

$$\Delta\theta = -\alpha \cdot \frac{\text{momentum}}{\text{scale}} = -\alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{2.12}$$

Default hyperparameters: Adam hoạt động tốt với default settings:

- Learning rate: $\alpha = 0.001$ (hoặc 0.0003 cho RL)

- First moment decay: $\beta_1 = 0.9$
- Second moment decay: $\beta_2 = 0.999$
- Epsilon: $\epsilon = 10^{-8}$

Ưu điểm của Adam:

- Kết hợp momentum và adaptive LR $\rightarrow$ hiệu quả cao
- Hoạt động tốt với default hyperparameters $\rightarrow$ ít cần tuning
- Hiệu quả với sparse gradients và noisy data
- Thích hợp cho non-convex optimization và high-dimensional parameter spaces
- Computational efficient, memory efficient (chỉ cần lưu m_t, v_t)

Nhược điểm và biến thể:

- Có thể không hội tụ cho một số bài toán (do second moment không decay đủ nhanh)
- **AdamW**: Sửa weight decay (regularization) bằng cách tách weight decay khỏi gradient update
- **AMSGrad**: Giữ max của v_t để đảm bảo learning rate không tăng

Khi nào dùng Adam:

- Default choice cho hầu hết các bài toán deep learning
- Đặc biệt hiệu quả với RNN, GAN, VAE
- Phù hợp khi không muốn spend nhiều thời gian tuning optimizer
- Trong RL và PPO, Adam thường được dùng riêng cho Actor và Critic với learning rates khác nhau

So sánh các thuật toán tối ưu:

Bảng 2.2. So sánh các thuật toán tối ưu

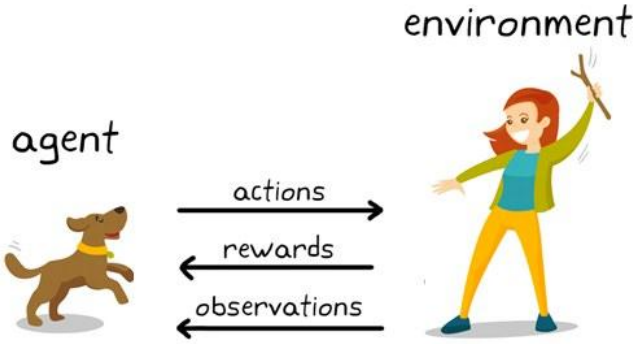
Thuật toán	**Ưu điểm chính**	**Nhược điểm chính**	**Ứng dụng phù hợp**	**Tuning**
Batch GD	Hội tụ ổn định, chính xác	Rất chậm, không scale	Bài toán nhỏ, convex	Dễ
SGD	Nhanh, scale tốt	Oscillation, cần tuning LR	Classification, không dùng nhiều nữa	Khó
Momentum	Tăng tốc, ít oscillation	Thêm hyperparameter β	Computer vision (với SGD)	Trung bình
RMSProp	Adaptive LR, phù hợp RNN	Không có momentum	Recurrent networks	Trung bình
Adam	Kết hợp momentum + adaptive LR, ít cần tuning	Có thể không hội tụ một số bài toán	Default choice, RL, NLP, GANs	Dễ

2.5.2 Reinforcement Learning

Học tăng cường (Reinforcement Learning - RL) là phương pháp học máy trong đó một agent học cách hành động trong môi trường để tối đa hóa tổng reward tích lũy. RL được mô hình hóa bằng Markov Decision Process (MDP), định nghĩa bởi bộ $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ trong đó:

- $\mathcal{S}$ là tập hợp các trạng thái (states)
- $\mathcal{A}$ là tập hợp các hành động (actions)
- $\mathcal{P}(s'|s, a)$ là xác suất chuyển trạng thái
- $\mathcal{R}(s, a)$ là hàm reward
- $\gamma \in [0, 1]$ là discount factor

Policy $\pi(a|s)$ là phân phối xác suất của hành động cho trước trạng thái. Mục tiêu của RL là tìm policy tối ưu π^* maximizing expected return $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$.



Hình 2.4. Tương tác giữa Agent và Environment trong Reinforcement Learning

Value function $V^\pi(s)$ ước lượng expected return khi bắt đầu từ trạng thái s và theo policy π:

$$V^\pi(s) = \mathbb{E}_\pi \left[G_t \mid s_t = s \right] \tag{2.13}$$

Q-function $Q^\pi(s, a)$ ước lượng expected return khi thực hiện hành động a tại trạng thái s rồi theo policy π:

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right] \tag{2.14}$$

Advantage function $A^\pi(s, a)$ đo lường lợi thế của việc chọn hành động a so với policy hiện tại:

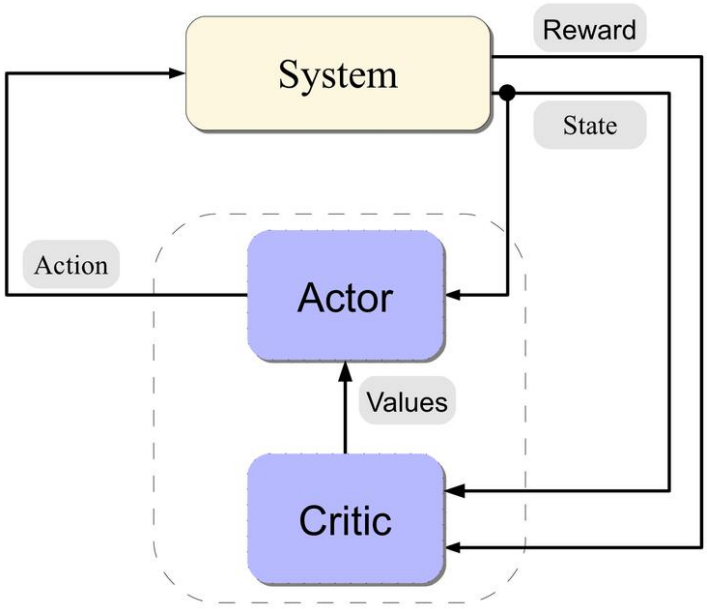
$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \tag{2.15}$$

Advantage function giúp giảm variance trong policy gradient. **Generalized Advantage Estimation (GAE)** [13] là phương pháp ước lượng advantage function hiệu quả bằng cách kết hợp nhiều n-step advantage estimates với tham số $\lambda \in [0, 1]$:

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l} \tag{2.16}$$

trong đó $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ là temporal difference error.

Kiến trúc **Actor-Critic** kết hợp policy-based và value-based methods. Actor (policy network) $\pi_\theta(a|s)$ chọn hành động, Critic (value network) $V_\phi(s)$ đánh giá trạng thái. Critic giúp giảm variance cho policy gradient của Actor.



Hình 2.5. Kiến trúc Actor-Critic: Actor quyết định hành động, Critic đánh giá giá trị trạng thái

2.5.3 Proximal Policy Optimization (PPO)

Thuật toán PPO [14] là một trong những thuật toán policy gradient hiệu quả và ổn định nhất hiện nay. PPO giải quyết vấn đề của các thuật toán policy gradient truyền thống là độ nhạy cảm với learning rate và khả năng policy thay đổi quá nhanh gây mất ổn định. Trong phần này, chúng ta sẽ phân tích chi tiết từ policy gradient cơ bản đến PPO.

2.5.3.1 Policy Gradient cơ bản

Policy gradient methods tối ưu trực tiếp policy $\pi_\theta(a|s)$ bằng cách maximize expected return $J(\theta)$:

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \tag{2.17}$$

trong đó $\tau = (s_0, a_0, r_0, s_1, \ldots)$ là trajectory được thu thập bằng policy π_θ.

Policy Gradient Theorem: Gradient của objective function là:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\cdot G_t\right] \tag{2.18}$$

trong đó $G_t = \sum_{k=t}^{T}\gamma^{k-t}r_k$ là return từ timestep t.

Giảm variance với baseline: Sử dụng value function $V^\pi(s_t)$ làm baseline:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\cdot (G_t - V^\pi(s_t))\right] \tag{2.19}$$

Advantage function $A^\pi(s_t, a_t) = G_t - V^\pi(s_t)$ giúp giảm variance mà không làm tăng bias.

Vấn đề của vanilla policy gradient:

- Update step size khó kiểm soát - policy có thể thay đổi quá nhanh và mất performance
- Sample inefficient - cần nhiều dữ liệu mới hội tụ
- Không ổn định - performance có thể sụt giảm đột ngột
- Khó tuning learning rate - quá nhỏ $\rightarrow$ chậm, quá lớn $\rightarrow$ collapse

2.5.3.2 Trust Region Policy Optimization (TRPO)

Ý tưởng chính: Thay vì cố định learning rate, TRPO đảm bảo mỗi update giữ policy mới "gần"policy cũ bằng constraint KL divergence:

$$\begin{aligned} \text{maximize}_\theta \quad & \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \mathbb{E}_t \left[\text{KL}[\pi_{\theta_{old}}(\cdot|s_t) \| \pi_\theta(\cdot|s_t)] \right] \leq \delta \end{aligned} \tag{2.20}$$

trong đó:

- Objective: Expected advantage khi sử dụng actions từ $\pi_{\theta_{old}}$ nhưng đánh giá theo π_θ
- Constraint: KL divergence giữa policy mới và cũ không vượt quá δ (thường 0.01-0.05)

KL divergence: Đo sự khác biệt giữa hai phân phối xác suất:

$$\text{KL}[p\|q] = \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right] = \sum_x p(x) \log \frac{p(x)}{q(x)} \tag{2.21}$$

$\text{KL} = 0$ khi $p = q$ (identical), và $\text{KL} > 0$ ngược lại. KL không đối xứng: $\text{KL}[p\|q] \neq \text{KL}[q\|p]$.

Giải bài toán TRPO: Sử dụng conjugate gradient để giải bài toán tối ưu có constraint:

1. Linearize objective: $L(\theta) \approx g^T(\theta - \theta_{old})$ với $g = \nabla_\theta L|_{\theta_{old}}$
2. Quadratic approximation của KL constraint: $\text{KL} \approx \frac{1}{2}(\theta - \theta_{old})^T H(\theta - \theta_{old}) \leq \delta$
3. Giải hệ tuyến tính: $Hx = g$ bằng conjugate gradient
4. Line search để đảm bảo constraint satisfied

Ưu điểm của TRPO:

- Đảm bảo monotonic improvement (performance không giảm)
- Robust với nhiều loại bài toán
- Không cần tuning learning rate

Nhược điểm của TRPO:

- Phức tạp implementation (conjugate gradient, line search)
- Tốn kém tính toán (Hessian-vector product, multiple forward passes)
- Không scale tốt cho large models
- Khó kết hợp với các techniques khác (shared parameters giữa policy và value)

2.5.3.3 PPO-Clip: First-order Approximation

PPO đơn giản hóa TRPO bằng cách thay constraint bằng clipped objective function. Thay vì giải bài toán tối ưu có constraint, PPO sử dụng first-order optimization (SGD hoặc Adam) với clipping:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{2.22}$$

trong đó:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ là probability ratio
- clip$(r, 1-\epsilon, 1+\epsilon)$ giới hạn r trong $[1-\epsilon, 1+\epsilon]$
- ϵ là clip ratio (typically 0.1-0.2)

Phân tích clipping mechanism:
Trường hợp 1: Advantage dương ($\hat{A}_t > 0$)
Hành động a_t tốt hơn trung bình $\rightarrow$ muốn tăng $\pi_\theta(a_t|s_t)$.

- Nếu $r_t < 1+\epsilon$: Objective = $r_t\hat{A}_t$ (không clip) $\rightarrow$ gradient khuyến khích tăng r_t
- Nếu $r_t \geq 1+\epsilon$: Objective = $(1+\epsilon)\hat{A}_t$ (clipped) $\rightarrow$ gradient = 0, không tăng thêm

Trường hợp 2: Advantage âm ($\hat{A}_t < 0$)
Hành động a_t tệ hơn trung bình $\rightarrow$ muốn giảm $\pi_\theta(a_t|s_t)$.

- Nếu $r_t > 1-\epsilon$: Objective = $r_t\hat{A}_t$ (không clip) $\rightarrow$ gradient khuyến khích giảm r_t
- Nếu $r_t \leq 1-\epsilon$: Objective = $(1-\epsilon)\hat{A}_t$ (clipped) $\rightarrow$ gradient = 0, không giảm thêm

Ý nghĩa: Clipping ngăn không cho policy thay đổi quá nhiều trong một update:

- Nếu action tốt ($A > 0$), cho phép tăng xác suất nhưng không quá $1+\epsilon$ lần
- Nếu action xấu ($A < 0$), cho phép giảm xác suất nhưng không dưới $1-\epsilon$ lần

Visualization: Objective function có dạng "flat plateau"sau khi ra khỏi trust region $[1-\epsilon, 1+\epsilon]$, tạo ra "natural constraint"mà không cần giải bài toán tối ưu có constraint.

2.5.3.4 Value Function Clipping

Ngoài clipping cho policy, PPO còn có thể clip value function loss để tránh value estimates thay đổi quá nhanh:

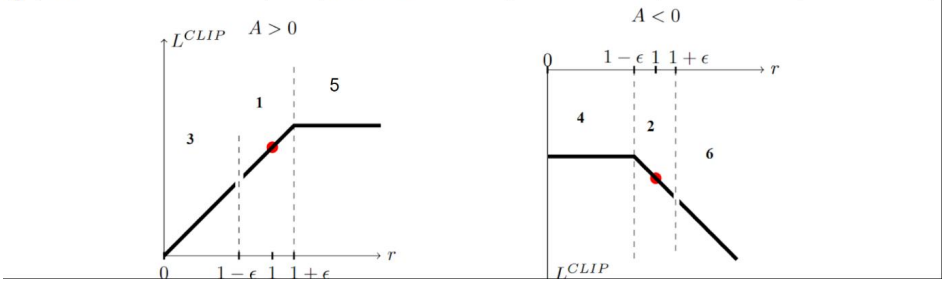
$$L^{VF}(\theta) = \mathbb{E}_t \left[\max \left((V_\theta(s_t) - V_t^{targ})^2, (\text{clip}(V_\theta(s_t), V_{old} - \epsilon_V, V_{old} + \epsilon_V) - V_t^{targ})^2 \right) \right] \tag{2.23}$$

trong đó:

- V_t^{targ} là target value (ví dụ: GAE estimate)
- V_{old} là value estimate từ policy cũ
- ϵ_V là clip range cho value function (thường 0.1-0.2)

Value clipping giúp ổn định training, đặc biệt khi:

	$p_t(\theta)\ > 0$	A_t	Return Value of min	Objective is Clipped	Sign of Objective	Gradient
1	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$+$	$p_t(\theta)A_t$	no	$+$	✓
2	$p_t(\theta) \in [1-\epsilon, 1+\epsilon]$	$-$	$p_t(\theta)A_t$	no	$-$	✓
3	$p_t(\theta) < 1-\epsilon$	$+$	$p_t(\theta)A_t$	no	$+$	✓
4	$p_t(\theta) < 1-\epsilon$	$-$	$(1-\epsilon)A_t$	yes	$-$	**0**
5	$p_t(\theta) > 1+\epsilon$	$+$	$(1+\epsilon)A_t$	yes	$+$	**0**
6	$p_t(\theta) > 1+\epsilon$	$-$	$p_t(\theta)A_t$	no	$-$	✓



Hình 2.6. PPO Clipping: Objective bị chặn ngoài khoảng $[1-\epsilon, 1+\epsilon]$ để ngăn policy thay đổi quá nhanh

- Value function được khởi tạo kém
- Reward scale thay đổi nhiều giữa các episodes
- Shared parameters giữa policy và value networks

2.5.3.5 Complete PPO Objective

Objective function đầy đủ của PPO kết hợp policy loss, value loss, và entropy bonus:

$$L^{TOTAL}(\theta) = \mathbb{E}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{2.24}$$

trong đó:

- L_t^{CLIP}: Clipped policy loss (maximize)
- L_t^{VF}: Value function loss (minimize)
- $S[\pi_\theta]$: Entropy của policy (maximize để khuyến khích exploration)
- c_1, c_2: Coefficients (thường $c_1 = 0.5 - 1.0$, $c_2 = 0.01$)

Entropy bonus: Entropy đo độ ”random”của policy:

$$S[\pi_\theta](s) = -\mathbb{E}_{a \sim \pi_\theta(\cdot|s)}[\log \pi_\theta(a|s)] \tag{2.25}$$

- Entropy cao $\rightarrow$ policy uniform (exploration nhiều)
- Entropy thấp $\rightarrow$ policy deterministic (exploitation)
- Entropy bonus khuyến khích exploration ở giai đoạn đầu
- Thường decay entropy coefficient theo thời gian

2.5.3.6 PPO-Penalty: Alternative Formulation

Thay vì clipping, PPO-Penalty thêm KL penalty trực tiếp vào objective:

$$L^{PENALTY}(\theta) = \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t - \beta \cdot \text{KL}[\pi_{\theta_{old}}(\cdot|s_t) \| \pi_\theta(\cdot|s_t)] \right] \tag{2.26}$$

Adaptive β: PPO-Penalty điều chỉnh penalty coefficient β theo KL divergence thực tế:

- Nếu KL $< \text{KL}_{target}/1.5$: giảm β (cho phép update lớn hơn)
- Nếu KL $> \text{KL}_{target} \times 1.5$: tăng β (hạn chế update)
- Thường khởi tạo $\beta = 1.0$, $\text{KL}_{target} = 0.01$

So sánh PPO-Clip vs PPO-Penalty:

Bảng 2.3. So sánh PPO-Clip và PPO-Penalty

Tiêu chí	**PPO-Clip**	**PPO-Penalty**
Mechanism	Hard constraint bằng clipping	Soft constraint bằng penalty
Hyperparameters	ϵ cố định (0.1-0.2)	β adaptive, KL_{target}
Implementation	Đơn giản hơn	Cần tính KL divergence
Performance	Tốt hơn trong hầu hết trường hợp	Tương đương nhưng ít phổ biến
Robustness	Robust với nhiều loại môi trường	Cần tuning KL_{target}

Kết luận: PPO-Clip được ưa chuộng hơn do đơn giản, robust, và performance tốt với default hyperparameters.

2.5.3.7 PPO Hyperparameters

Các hyperparameters quan trọng của PPO:

- **Clip ratio (ϵ)**: Giới hạn mức độ thay đổi policy (thường 0.1-0.2)
- **Number of epochs (K)**: Số lần update policy trên mỗi batch data (3-10)
- **Batch size**: Số timesteps thu thập trước update (2048-4096)
- **GAE lambda (λ)**: Cân bằng bias-variance (0.90-0.99)
- **Discount factor (γ)**: Trọng số future rewards (0.99 cho long-horizon tasks)
- **Learning rates**: Thường khác nhau cho Actor và Critic. Trong luận văn này: Critic LR = 6×10^{-3}, Actor LR = 4×10^{-4} (Stage 1)
- **Entropy coefficient (c_2)**: Khuyến khích exploration (0.01, trong luận văn này: 8×10^{-3} decay đến 2×10^{-3})
- **Target KL**: Early stopping threshold (thường 0.015, trong luận văn này: 0.035 cho aggressive updates)

2.5.4 Kiến trúc mạng nơ-ron

Mạng nơ-ron nhân tạo (Artificial Neural Networks - ANN) là mô hình tính toán lấy cảm hứng từ hệ thần kinh sinh học. Trong phần này, chúng ta sẽ phân tích chi tiết về kiến trúc, quá trình huấn luyện, và các kỹ thuật tối ưu hóa mạng nơ-ron.

2.5.4.1 Fully Connected Neural Networks

Kiến trúc cơ bản: Mạng nơ-ron fully connected (dense) bao gồm nhiều lớp (layers), mỗi lớp có nhiều neurons (nodes). Mỗi neuron trong lớp l kết nối với tất cả neurons trong lớp $l+1$.

Forward propagation: Tính toán output từ input qua các lớp:

Lớp l:

$$\begin{aligned} \mathbf{z}^{[l]} &= \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]} \\ \mathbf{a}^{[l]} &= g^{[l]}(\mathbf{z}^{[l]}) \end{aligned} \tag{2.27}$$

trong đó:

- $\mathbf{a}^{[l-1]}$: Activations từ lớp trước (input nếu $l = 1$)
- $\mathbf{W}^{[l]}$: Weight matrix kích thước $(n^{[l]}, n^{[l-1]})$
- $\mathbf{b}^{[l]}$: Bias vector kích thước $(n^{[l]}, 1)$
- $\mathbf{z}^{[l]}$: Pre-activation (linear combination)
- $g^{[l]}$: Activation function
- $\mathbf{a}^{[l]}$: Activations (output của lớp l)

Activation Functions: Thêm tính phi tuyến vào mạng

- **ReLU (Rectified Linear Unit):** $\text{ReLU}(x) = \max(0, x)$, derivative:

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

  Phổ biến nhất do tính toán nhanh và không bị vanishing gradient. Nhược điểm: "Dying ReLU"khi neurons có $z < 0$ không được update.

- **Sigmoid và Tanh:** $\sigma(x) = \frac{1}{1+e^{-x}}$ và $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. Sigmoid output trong $(0, 1)$, dùng cho binary classification và normalize velocity trong RL. Tanh output trong $(-1, 1)$, zero-centered, phù hợp cho symmetric control (ví dụ: angular velocity). Trong luận văn này: Sigmoid cho linear velocity, Tanh cho angular velocity.

2.5.4.2 Convolutional Neural Networks (CNN)

Convolutional Neural Networks [5] được thiết kế để xử lý dữ liệu có cấu trúc lưới (grid-like structure) như hình ảnh (2D) hoặc time series/LiDAR (1D).

Convolution operation: Áp dụng filter (kernel) trên input:

1D convolution (cho LiDAR):

$$(f * g)(n) = \sum_{m=-\infty}^{\infty} f(m) \cdot g(n-m) \tag{2.28}$$

Trong neural network, discrete convolution:

$$y[n] = \sum_{k=0}^{K-1} w[k] \cdot x[n+k] + b \tag{2.29}$$

trong đó:

- x: Input sequence (ví dụ: LiDAR 360 points)
- w: Filter weights (learnable parameters)
- K: Kernel size (ví dụ: 5)
- b: Bias (learnable)
- y: Output feature map

Conv1D parameters:

- **Number of filters**: Số lượng patterns muốn học (16-128)
- **Kernel size**: Kích thước filter, quyết định receptive field (3-7)
- **Stride**: Khoảng cách giữa các vị trí áp dụng filter. Stride=2 downsample output length
- **Padding**: Valid (không padding) hoặc Same (giữ output length = input length)

Output length được tính:

$$\text{Output length} = \frac{\text{Input length} - \text{Kernel size} + 2 \times \text{Padding}}{\text{Stride}} + 1 \tag{2.30}$$

Example: LiDAR processing trong luận văn này
Input: LiDAR 360 points (sau downsample)

$$\text{Conv1D}(32, k = 5, s = 2) \rightarrow \text{ReLU} \rightarrow \text{Conv1D}(32, k = 3, s = 2) \rightarrow \text{ReLU} \tag{2.31}$$

Output feature map được flatten và concatenate với goal position và velocity.
Ưu điểm của CNN:

- Parameter sharing: Cùng filter áp dụng khắp input $\rightarrow$ ít parameters
- Translation invariance: Detect pattern ở mọi vị trí
- Hierarchical features: Lớp sâu học abstract patterns từ low-level features
- Hiệu quả với spatial/temporal data

2.5.4.3 Loss Functions

Loss function (hay cost function) đóng vai trò đo lường sự khác biệt giữa dự đoán của mạng và giá trị thực tế. Mục tiêu của quá trình training là tìm tập parameters làm minimize loss này.

Mean Squared Error (MSE): Thường dùng cho bài toán regression:

$$L_{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{2.32}$$

trong đó y_i là giá trị thực, $\hat{y}_i$ là giá trị dự đoán. MSE penalize lỗi lớn mạnh hơn do bình phương. Trong RL, MSE train value function:

$$L_V(\phi) = \mathbb{E}[(V_\phi(s) - V_{target})^2] \tag{2.33}$$

Cross-Entropy Loss: Dùng cho classification. Binary Cross-Entropy:

$$L_{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \tag{2.34}$$

Categorical Cross-Entropy cho multi-class:

$$L_{CCE} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}) \tag{2.35}$$

Cross-entropy đo "khoảng cách"giữa true distribution và predicted distribution.

2.5.4.4 Backpropagation

Backpropagation là thuật toán cốt lõi để training neural networks. Nhiệm vụ của nó là tính gradient của loss function đối với tất cả parameters (weights và biases) trong mạng, để sau đó có thể update parameters theo hướng giảm loss. Thuật toán sử dụng chain rule của calculus và hoạt động theo hai pha:

Forward pass: Dữ liệu đi từ input layer qua các hidden layers đến output layer. Tại mỗi layer l, ta tính:

$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]}\mathbf{a}^{[l-1]} + \mathbf{b}^{[l]}, \quad \mathbf{a}^{[l]} = g^{[l]}(\mathbf{z}^{[l]}) \quad \text{for } l = 1, ..., L \tag{2.36}$$

trong đó $\mathbf{z}^{[l]}$ là pre-activation (weighted sum), $\mathbf{a}^{[l]}$ là activation (output sau activation function $g^{[l]}$). Cuối cùng tính loss L bằng cách so sánh output $\mathbf{a}^{[L]}$ với true labels. Pha này cho ta biết mạng đang dự đoán như thế nào và loss hiện tại là bao nhiêu.

Backward pass: Đây là phần quan trọng - ta tính gradient của loss theo từng parameter bằng cách "lan truyền ngược"(backpropagate) từ output về input. Định nghĩa $\delta^{[l]} = \frac{\partial L}{\partial \mathbf{z}^{[l]}}$ là gradient của loss theo pre-activation tại layer l.

Tại output layer L, gradient được tính trực tiếp:

$$\delta^{[L]} = \nabla_{\mathbf{a}^{[L]}} L \odot g'^{[L]}(\mathbf{z}^{[L]}) \tag{2.37}$$

trong đó $\odot$ là element-wise multiplication, $g'^{[L]}$ là derivative của activation function.

Lan truyền ngược qua các layers $l = L-1, L-2, ..., 1$:

$$\delta^{[l]} = (\mathbf{W}^{[l+1]})^T \delta^{[l+1]} \odot g'^{[l]}(\mathbf{z}^{[l]}) \tag{2.38}$$

Công thức này thể hiện chain rule: gradient tại layer l phụ thuộc vào (1) gradient từ layer sau $\delta^{[l+1]}$ truyền về qua weights $\mathbf{W}^{[l+1]}$, và (2) derivative của activation function tại layer đó $g'^{[l]}$.

Sau khi có $\delta^{[l]}$ cho tất cả layers, ta tính gradient của loss theo weights và biases:

$$\frac{\partial L}{\partial \mathbf{W}^{[l]}} = \delta^{[l]}(\mathbf{a}^{[l-1]})^T, \quad \frac{\partial L}{\partial \mathbf{b}^{[l]}} = \delta^{[l]} \tag{2.39}$$

Gradient của weights phụ thuộc vào activations từ layer trước $\mathbf{a}^{[l-1]}$ - đây là lý do tại sao backpropagation cần lưu lại activations từ forward pass.

Vấn đề vanishing và exploding gradients: Khi mạng có nhiều layers, gradient có thể trở nên cực nhỏ (vanishing) hoặc cực lớn (exploding) khi nhân qua nhiều layers. Vanishing gradient xảy ra với sigmoid/tanh vì derivatives của chúng nhỏ hơn 1 ($\sigma'(x) \leq 0.25$, $\tanh'(x) \leq 1$), nhân nhiều lần sẽ tiến về 0 - khiến các layer

đầu không học được. Exploding gradient xảy ra khi weights quá lớn hoặc learning rate không phù hợp. Các giải pháp phổ biến: dùng ReLU activation (không bị vanishing vì $\text{ReLU}'(x) = 1$ khi $x > 0$), gradient clipping (chặn gradient lớn), batch normalization (ổn định activations), và weight initialization phù hợp.

2.5.4.5 Regularization Techniques

Regularization là các kỹ thuật giúp mạng nơ-ron tránh overfitting (học quá khớp với training data) và generalize tốt hơn cho data chưa thấy.

L2 Regularization (Weight Decay): Thêm một penalty term vào loss function, tỷ lệ với tổng bình phương của tất cả weights:

$$L_{total} = L_{data} + \lambda \sum_{l} \|\mathbf{W}^{[l]}\|_2^2 \tag{2.40}$$

trong đó λ là regularization strength. Penalty này làm cho mạng "không thích"weights lớn - khuyến khích weights nhỏ hơn, smoother. Khi update weights:

$$\mathbf{W} \leftarrow \mathbf{W} - \alpha \left(\frac{\partial L_{data}}{\partial \mathbf{W}} + 2\lambda \mathbf{W} \right) = (1 - 2\alpha\lambda)\mathbf{W} - \alpha \frac{\partial L_{data}}{\partial \mathbf{W}} \tag{2.41}$$

mỗi weight sẽ bị "decay"một chút về phía 0 (nhân với số nhỏ hơn 1). Models với weights nhỏ thường đơn giản hơn, ít overfitting hơn.

Dropout: Trong training, randomly "tắt"(set output = 0) một số neurons với probability p (thường 0.5):

$$\mathbf{a}^{[l]} = \mathbf{m}^{[l]} \odot g^{[l]}(\mathbf{z}^{[l]}), \quad m_i^{[l]} \sim \text{Bernoulli}(1-p) \tag{2.42}$$

Mỗi training iteration, mạng dùng một "subnetwork"khác nhau. Điều này ngăn neurons phụ thuộc lẫn nhau quá mức (co-adaptation) - buộc mỗi neuron học features hữu ích độc lập. Khi testing, dùng tất cả neurons nhưng scale outputs xuống để match với expectation trong training.

Batch Normalization: Normalize activations của mỗi layer về mean = 0 và variance = 1, tính trên từng mini-batch:

$$\hat{\mathbf{z}} = \frac{\mathbf{z} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \quad \mathbf{z}_{BN} = \gamma \hat{\mathbf{z}} + \beta \tag{2.43}$$

trong đó $\mu_{\mathcal{B}}, \sigma_{\mathcal{B}}^2$ là mean/variance của batch, γ, β là learnable parameters để mạng có thể "undo"normalization nếu cần. Lợi ích chính là ổn định quá trình training - cho phép dùng learning rate lớn hơn, giảm nhạy cảm với weight initialization. Batch normalization cũng có tác dụng regularization nhẹ vì việc normalize trên mini-batch tạo một chút noise.

2.5.5 Điều khiển PID

Bộ điều khiển PID (Proportional-Integral-Derivative) [15] là phương pháp điều khiển cổ điển được sử dụng rộng rãi trong robot thực tế. Đối với differential drive robot, PID được áp dụng để điều khiển vận tốc tuyến tính và vận tốc góc.

Công thức PID cơ bản:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{2.44}$$

trong đó:

- $e(t) = r(t) - y(t)$ là sai số giữa setpoint $r(t)$ và output $y(t)$

- K_p là hệ số P (Proportional): phản ứng tỷ lệ với sai số hiện tại
- K_i là hệ số I (Integral): khử sai số tích lũy theo thời gian (steady-state error)
- K_d là hệ số D (Derivative): dự đoán xu hướng thay đổi, giảm overshoot

Ứng dụng cho differential drive robot [16]:

PID cho linear velocity: Điều khiển vận tốc bánh xe để đạt linear velocity mong muốn v_{des}:

$$v_{cmd} = K_{p,v}e_v + K_{i,v}\int e_v dt + K_{d,v}\frac{de_v}{dt} \tag{2.45}$$

với $e_v = v_{des} - v_{current}$.

PID cho angular velocity: Điều khiển sự chênh lệch vận tốc giữa hai bánh để đạt angular velocity ω_{des}:

$$\omega_{cmd} = K_{p,\omega}e_\omega + K_{i,\omega}\int e_\omega dt + K_{d,\omega}\frac{de_\omega}{dt} \tag{2.46}$$

với $e_\omega = \omega_{des} - \omega_{current}$.

Tuning PID parameters: Có nhiều phương pháp tuning như Ziegler-Nichols, Cohen-Coon, hoặc trial-and-error. Nguyên tắc chung:

- Tăng K_p: Phản ứng nhanh hơn nhưng có thể overshoot và oscillation
- Tăng K_i: Khử steady-state error nhưng có thể gây chậm và overshoot
- Tăng K_d: Giảm overshoot và oscillation nhưng nhạy cảm với noise

Ưu điểm: Đơn giản, không cần mô hình chính xác, dễ triển khai, ổn định với hệ tuyến tính.

Hạn chế: Khó tuning cho hệ phi tuyến, không tối ưu cho hệ đa biến (MIMO), nhạy cảm với noise (thành phần D), không thích nghi với thay đổi môi trường.

2.5.6 UKF Sensor Fusion

Unscented Kalman Filter (UKF) [17] là phương pháp ước lượng trạng thái cho hệ thống phi tuyến. UKF cải tiến Extended Kalman Filter (EKF) bằng cách sử dụng unscented transform thay vì linearization, cho kết quả chính xác hơn và không cần tính Jacobian.

Bài toán sensor fusion: Robot sử dụng nhiều cảm biến để ước lượng pose (x, y, θ):

- **Wheel odometry:** Đo vận tốc bánh xe, tích phân ra vị trí. Ưu điểm: update rate cao (50-100 Hz). Nhược điểm: drift tích lũy, trượt bánh xe gây sai số.
- **IMU (Inertial Measurement Unit):** Đo gia tốc (accelerometer) và vận tốc góc (gyroscope). Ưu điểm: không bị trượt bánh. Nhược điểm: drift theo thời gian, cần calibration.

State space model:

$$\mathbf{x} = [x, y, \theta, v_x, v_y, \omega]^T \tag{2.47}$$

Process model (motion model):

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k \tag{2.48}$$

với $\mathbf{u}_k$ là control input (wheel velocities) và $\mathbf{w}_k \sim \mathcal{N}(0, \mathbf{Q})$ là process noise.

Measurement models:

Wheel odometry:

$$\mathbf{z}_{odom} = \begin{bmatrix} v_x \\ v_y \\ \omega \end{bmatrix} + \mathbf{v}_{odom} \tag{2.49}$$

IMU:

$$\mathbf{z}_{imu} = \begin{bmatrix} a_x \\ a_y \\ \omega \end{bmatrix} + \mathbf{v}_{imu} \tag{2.50}$$

với $\mathbf{v} \sim \mathcal{N}(0, \mathbf{R})$ là measurement noise.

Thuật toán UKF:

1. **Unscented Transform:** Chọn $2n + 1$ sigma points $\mathcal{X}^{(i)}$ xung quanh mean $\mathbf{x}$ với weights $W^{(i)}$:

$$\begin{aligned} \mathcal{X}^{(0)} &= \mathbf{x} \\ \mathcal{X}^{(i)} &= \mathbf{x} + \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_i, \quad i = 1, \ldots, n \\ \mathcal{X}^{(i)} &= \mathbf{x} - \left(\sqrt{(n+\lambda)\mathbf{P}}\right)_{i-n}, \quad i = n+1, \ldots, 2n \end{aligned} \tag{2.51}$$

2. **Prediction:** Truyền sigma points qua process model:

$$\mathcal{X}^{(i)}_{k+1|k} = f(\mathcal{X}^{(i)}_k, \mathbf{u}_k) \tag{2.52}$$

Tính predicted mean và covariance:

$$\begin{aligned} \mathbf{x}_{k+1|k} &= \sum_{i=0}^{2n} W^{(i)} \mathcal{X}^{(i)}_{k+1|k} \\ \mathbf{P}_{k+1|k} &= \sum_{i=0}^{2n} W^{(i)} (\mathcal{X}^{(i)}_{k+1|k} - \mathbf{x}_{k+1|k})(\mathcal{X}^{(i)}_{k+1|k} - \mathbf{x}_{k+1|k})^T + \mathbf{Q} \end{aligned} \tag{2.53}$$

3. **Update:** Khi có measurement $\mathbf{z}_k$, tính Kalman gain và update:

$$\begin{aligned} \mathcal{Y}^{(i)}_k &= h(\mathcal{X}^{(i)}_{k|k-1}) \\ \mathbf{y}_k &= \sum_{i=0}^{2n} W^{(i)} \mathcal{Y}^{(i)}_k \\ \mathbf{K}_k &= \mathbf{P}_{xy} \mathbf{S}_k^{-1} \\ \mathbf{x}_k &= \mathbf{x}_{k|k-1} + \mathbf{K}_k(\mathbf{z}_k - \mathbf{y}_k) \\ \mathbf{P}_k &= \mathbf{P}_{k|k-1} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^T \end{aligned} \tag{2.54}$$

Tuning noise covariances:

- **Q** (process noise): Phản ánh độ tin cậy của model. Lớn hơn → tin measurement hơn.
- **R** (measurement noise): Phản ánh độ chính xác sensor. Nhỏ hơn → tin measurement hơn.

So sánh UKF vs EKF:

Bảng 2.4. So sánh UKF và EKF

Tiêu chí	**EKF**	**UKF**
Linearization	Cần tính Jacobian, chỉ chính xác bậc 1	Unscented transform, chính xác bậc 2-3
Độ chính xác	Kém với hệ phi tuyến mạnh	Tốt hơn EKF
Tính toán	Nhanh hơn (ít sigma points)	Chậm hơn (nhiều sigma points)
Implementation	Cần tính Jacobian (phức tạp)	Không cần Jacobian (đơn giản hơn)

2.5.7 Cartographer và SLAM

2.5.7.1 Bài toán SLAM

SLAM (Simultaneous Localization and Mapping) [18] là bài toán then chốt trong robot tự hành, yêu cầu robot đồng thời xác định vị trí của chính nó và xây dựng bản đồ môi trường chưa biết. Về mặt toán học, bài toán SLAM được biểu diễn như tìm kiếm phân phối xác suất:

$$p(\mathbf{x}_{1:t}, \mathbf{m} \mid \mathbf{z}_{1:t}, \mathbf{u}_{1:t}) \tag{2.55}$$

trong đó:

- $\mathbf{x}_{1:t} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t\}$: chuỗi pose (vị trí và hướng) của robot qua thời gian
- $\mathbf{m}$: bản đồ môi trường
- $\mathbf{z}_{1:t}$: dữ liệu cảm biến (laser scans, camera)
- $\mathbf{u}_{1:t}$: dữ liệu điều khiển (odometry)

2.5.7.2 Google Cartographer

Cartographer [19] là thư viện SLAM mã nguồn mở được Google phát triển, tối ưu cho việc chạy real-time trên nền tảng nhúng. Cartographer sử dụng phương pháp **graph-based SLAM** với kiến trúc submap, khác biệt so với các phương pháp truyền thống:

- **GMapping**: dựa trên particle filter, tốn nhiều tài nguyên tính toán khi môi trường lớn
- **Cartographer**: dựa trên pose graph optimization và submap-based matching, mở rộng tốt hơn, phát hiện loop closure hiệu quả, phù hợp với phần cứng giới hạn (Jetson Nano)

2.5.7.3 Nguyên lý hoạt động

Cartographer hoạt động dựa trên nguyên lý **chia nhỏ và tối ưu hóa từng phần**:

1. **Phân chia không gian thành submaps**: Thay vì xây dựng một bản đồ toàn cục lớn, môi trường được chia thành nhiều submaps nhỏ chồng lấp nhau
2. **Local optimization**: Trong mỗi submap, laser scans được căn chỉnh (scan matching) để tạo bản đồ cục bộ có độ chính xác cao
3. **Global optimization**: Khi có nhiều submaps, hệ thống tối ưu hóa mối quan hệ giữa các submaps thông qua pose graph optimization

4. **Loop closure**: Khi robot quay lại vị trí cũ, hệ thống phát hiện và điều chỉnh lại toàn bộ bản đồ để loại bỏ sai số tích lũy

Quá trình scan matching trong Cartographer tối ưu hóa hàm chi phí:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}} \left(\sum_k (1 - M(\mathbf{T}_\mathbf{x} \mathbf{h}_k))^2 + \lambda \|\mathbf{x} - \mathbf{x}_{odom}\|^2 \right) \tag{2.56}$$

trong đó M là occupancy grid map, $\mathbf{h}_k$ là các điểm laser scan, $\mathbf{T}_\mathbf{x}$ là phép biến đổi pose, và $\mathbf{x}_{odom}$ là ước lượng từ odometry.

2.5.7.4 Kiến trúc hệ thống

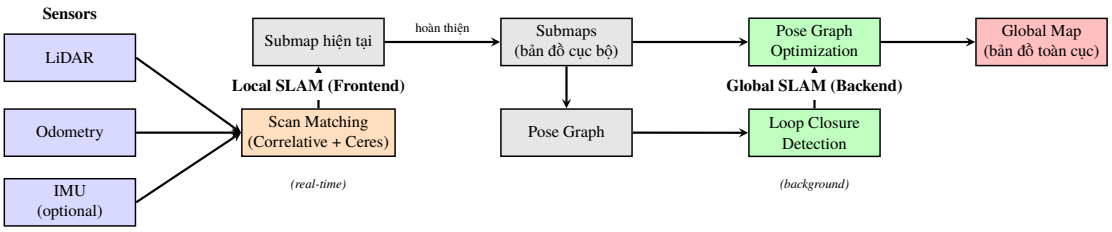
Cartographer gồm hai hệ thống con hoạt động song song (Hình 2.7):

Local SLAM (Frontend - Real-time):

- Nhận dữ liệu từ LiDAR, odometry, và IMU (tùy chọn)
- Thực hiện scan matching để ước lượng pose hiện tại
- Xây dựng submaps cục bộ có tính nhất quán cao
- Chạy real-time (10-40 Hz) để đảm bảo phản hồi tức thì

Global SLAM (Backend - Background):

- Quản lý pose graph chứa tất cả submaps
- Phát hiện loop closure khi robot quay lại vị trí cũ
- Tối ưu hóa toàn cục để giảm sai số tích lũy
- Chạy nền không ảnh hưởng đến Local SLAM



Hình 2.7. Kiến trúc thuật toán Cartographer SLAM với hai hệ thống con: Local SLAM (frontend) xử lý real-time và Global SLAM (backend) tối ưu hóa toàn cục

2.5.7.5 Quy trình mapping

Quy trình xây dựng bản đồ trong Cartographer diễn ra theo các bước:

1. **Thu thập dữ liệu cảm biến**
    - LiDAR cung cấp laser scans (điểm đám mây 2D/3D)
    - Odometry ước lượng chuyển động của robot

- IMU (tùy chọn) cung cấp thông tin gia tốc và góc xoay

2. **Local SLAM xử lý real-time**
   - Nhận scan mới từ LiDAR
   - Sử dụng odometry làm ước lượng ban đầu
   - Thực hiện scan matching với submap hiện tại
   - Cập nhật pose của robot và thêm scan vào submap
3. **Xây dựng submaps**
   - Mỗi submap chứa khoảng 90-200 laser scans
   - Các submaps chồng lấp 50% để đảm bảo tính liên tục
   - Khi submap hoàn thiện, nó được "đóng băng"và thêm vào pose graph
4. **Global SLAM tối ưu hóa nền**
   - Phát hiện loop closure: so sánh submap hiện tại với các submaps cũ
   - Nếu phát hiện robot quay lại vị trí đã đi qua, tạo constraint mới
   - Chạy pose graph optimization để điều chỉnh vị trí tất cả submaps
   - Đảm bảo tính nhất quán toàn cục, loại bỏ drift tích lũy
5. **Xuất bản đồ toàn cục**
   - Ghép tất cả submaps đã được tối ưu hóa
   - Tạo occupancy grid map hoàn chỉnh
   - Lưu file .pbstream (định dạng riêng) hoặc .pgm (ảnh)

2.5.7.6 Các tham số quan trọng

Theo tài liệu chính thức của Cartographer [19], các tham số quan trọng nhất để tinh chỉnh được chia thành ba nhóm chính:

1. Local SLAM - Chất lượng mapping (quan trọng nhất):

Ceres Scan Matcher là thành phần cốt lõi của Cartographer, sử dụng thư viện **Ceres Solver** (một thư viện tối ưu hóa phi tuyến mạnh mẽ của Google) để căn chỉnh laser scan với submap hiện tại. Quá trình này giải bài toán tối ưu hóa phi tuyến (Equation 2.56) để tìm pose tốt nhất của robot, cân bằng giữa hai yếu tố: (1) mức độ khớp giữa laser scan và bản đồ, và (2) mức độ tin cậy vào odometry. Hai trọng số `translation_weight` và `rotation_weight` điều chỉnh độ tin cậy này.

Hai tham số này quyết định chất lượng scan matching và độ chính xác bản đồ:

- `ceres_scan_matcher.translation_weight` (mặc định: 10):
  - Độ tin cậy vào odometry trong quá trình tịnh tiến
  - Tăng khi odometry tốt, giảm khi odometry nhiễu
  - Giá trị khuyến nghị: 1e2 (100)
- `ceres_scan_matcher.rotation_weight` (mặc định: 40):

  - Độ tin cậy vào odometry trong quá trình xoay
  - Tăng khi IMU/encoder tốt, giảm khi robot trượt nhiều
  - Giá trị khuyến nghị: 4e2 (400)

Nguyên tắc: Tăng weights khi tin tưởng odometry, giảm weights khi muốn scan matching tự do hơn.

2. Global SLAM - Tối ưu hóa hiệu năng:

Nhóm tham số này giảm độ trễ (latency) cho phần cứng yếu:

- `optimize_every_n_nodes`: Giảm xuống để optimization chạy thường xuyên hơn nhưng tốn tài nguyên (mặc định: 90)
- `num_background_threads`: Tăng theo số lõi CPU (Jetson Nano: 2-4)
- `constraint_builder.sampling_ratio`: Giảm để xử lý ít constraints hơn (mặc định: 0.3)
- `global_sampling_ratio`: Giảm để giảm tải Global SLAM (mặc định: 0.003)
- `voxel_filter_size`: Tăng để giảm số điểm cần xử lý (mặc định: 0.025m)

3. Submap Configuration - Cân bằng chi tiết và tốc độ:

- `submaps.num_range_data`: Giảm để tạo submaps nhỏ hơn, xử lý nhanh hơn (mặc định: 90)
- `submaps.grid_options.resolution`: Tăng để giảm chi tiết bản đồ nhưng nhanh hơn (mặc định: 0.05m)
- `max_range`: Giảm nếu LiDAR nhiễu ở xa (mặc định: 25m cho RPLiDAR)

CHƯƠNG 3

PHƯƠNG PHÁP NGHIÊN CỨU

Chương này trình bày chi tiết phương pháp nghiên cứu bao gồm môi trường mô phỏng, thiết kế hàm reward, kiến trúc mạng nơ-ron, quy trình huấn luyện, xây dựng robot thực tế, thiết kế các module điều khiển (PID), sensor fusion (UKF), mapping và localization (Cartographer SLAM), và triển khai hệ thống robot hoàn chỉnh.

3.1 Môi trường mô phỏng

Hệ đa robot tránh va chạm được huấn luyện trong môi trường mô phỏng dựa trên Stage simulator kết hợp với ROS (Robot Operating System). Bài toán điều khiển được mô hình hóa dưới dạng Partially Observable Markov Decision Process (POMDP), trong đó mỗi robot đưa ra quyết định dựa trên quan sát cục bộ của chính nó mà không cần giao tiếp với các robot khác.

3.1.1 Không gian quan sát

Tại thời điểm t, robot i nhận được vector quan sát $o_t^i \in \mathcal{O}$ bao gồm ba thành phần:

$$o_t^i = [o_z^t, o_g^t, o_v^t] \tag{3.1}$$

trong đó:

Dữ liệu LiDAR $o_z^t \in \mathbb{R}^{3\times 454}$: Cảm biến quét 360 độ với 454 tia laser. Range được đặt giống với cảm biến trên robot thực tế: 0.03m - 5.5m. Mỗi phép đo trả về khoảng cách đến vật cản gần nhất theo hướng tương ứng.

Vị trí đích $o_g^t \in \mathbb{R}^2$: Tọa độ tương đối (r, θ) từ robot đến đích trong hệ tọa độ cục bộ của robot, với r là khoảng cách và θ là góc lệch so với hướng robot.

Vận tốc hiện tại $o_v^t \in \mathbb{R}^2$: Vận tốc tuyến tính v (m/s) và vận tốc góc ω (rad/s) của robot, giúp model nhận biết trạng thái chuyển động hiện tại.

3.1.2 Không gian hành động

Robot điều khiển chuyển động thông qua cặp hành động liên tục:

$$a_t = [v_t, \omega_t] \tag{3.2}$$

với $v_t \in [0, v_{\text{max}}]$ là vận tốc tuyến tính giới hạn trong khoảng 0-0.3 m/s, và $\omega_t \in [-\omega_{\text{max}}, \omega_{\text{max}}]$ là vận tốc góc giới hạn trong ± 1.0 rad/s. Mạng Actor xuất ra phân phối Gaussian $\mathcal{N}(\mu, \sigma^2)$ cho mỗi thành phần, sau đó lấy mẫu để thu được hành động cụ thể.

3.2 Thiết kế hàm reward

Hàm reward đóng vai trò quyết định trong việc định hình hành vi của robot. Reward function được thiết kế với mục tiêu cân bằng giữa hiệu quả (đến đích nhanh) và an toàn (tránh va chạm), đồng thời đảm bảo tín hiệu học tập rõ ràng không mâu thuẫn. Đồng thời kế thừa những ưu điểm đã được chứng minh ở bài báo gốc [10]

3.2.1 Terminal rewards

Reward được cấp khi episode kết thúc:

- $r_{\text{arrival}} = +30$: Đến đích thành công (trong vòng 0.3m từ goal)
- $r_{\text{collision}} = -25$: Va chạm với vật cản hoặc robot khác
- $r_{\text{timeout}} = -10$: Hết thời gian cho phép (180 giây)

Các giá trị terminal rewards được thiết kế mạnh hơn so với bài báo gốc (30/-25 thay vì 15/-15 [10]) để tạo tín hiệu rõ ràng hơn cho quá trình học.

3.2.2 Step rewards

Tại mỗi bước thời gian t, reward được tính từ 4 thành phần:

$$r_t = r_{\text{progress}} + r_{\text{safety}} + r_{\text{rotation}} + r_{\text{heading}} \tag{3.3}$$

Progress reward khuyến khích robot tiến về phía đích. Nếu khoảng cách đến đích giảm từ d_{t-1} xuống d_t, robot nhận được reward tỷ lệ với độ tiến bộ:

$$r_{\text{progress}} = 2.0 \times (d_{t-1} - d_t) \tag{3.4}$$

Hệ số 2.0 (thấp hơn 2.5 trong bài báo gốc) giúp robot ưu tiên an toàn hơn tốc độ.

Safety reward áp dụng penalty khi robot đi quá nhanh ở khu vực gần vật cản. Hệ thống 2 vùng được thiết kế dựa trên khoảng cách gần nhất d_{min} đến vật cản:

$$r_{\text{safety}} = \begin{cases} -0.3 \times (v_t - 0.2) & \text{if } d_{\text{min}} < 0.35\text{m and } v_t > 0.2 \\ -0.1 \times (v_t - 0.4) & \text{if } 0.35 \leq d_{\text{min}} < 0.6\text{m and } v_t > 0.4 \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

Logic: ở khu vực nguy hiểm (<0.35m), phạt mạnh nếu vận tốc vượt 0.2 m/s; ở khu vực cảnh báo (0.35-0.6m), phạt nhẹ nếu vận tốc vượt 0.4 m/s. Gradient mượt này giúp robot học cách điều chỉnh tốc độ theo mức độ nguy hiểm.

Rotation penalty hạn chế xoay quá nhanh để tránh chuyển động không mượt:

$$r_{\text{rotation}} = \begin{cases} -0.06 \times |\omega_t| & \text{if } |\omega_t| > 0.8 \text{ rad/s} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Ngưỡng 0.8 rad/s và hệ số phạt -0.06 (nhẹ hơn -0.1 trong bài báo gốc) cho phép robot chuyển hướng dễ dàng hơn khi gặp vật cản.

Heading reward khuyến khích robot hướng về phía đích:

$$r_{\text{heading}} = \begin{cases} 0.15 \times (1 - |\theta_{\text{goal}}|/\pi) & \text{if } |\theta_{\text{goal}}|/\pi < 0.3 \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

với θ_{goal} là góc lệch giữa hướng robot và hướng tới đích. Reward này giúp robot chủ động hướng về đích thay vì chỉ phụ thuộc vào progress reward.

3.2.3 Điểm khác biệt so với bài báo gốc

So với Long et al. (2018) [10], thiết kế reward có 3 cải tiến chính:

1. **Terminal rewards mạnh hơn:** 30/-25 thay vì 15/-15 tạo tín hiệu rõ ràng hơn
2. **Safety reward 2 vùng:** Thay vì 1 ngưỡng cứng, gradient mượt giúp học tốt hơn
3. **Heading reward bổ sung:** Khuyến khích robot chủ động hướng về đích

3.3 Kiến trúc mạng nơ-ron

Kiến trúc Actor-Critic với encoder chung được sử dụng để xử lý dữ liệu laser scan và tạo ra cả chính sách hành động (Actor) và ước lượng giá trị trạng thái (Critic). Thiết kế này cho phép chia sẻ feature representations giữa Actor và Critic, tăng hiệu quả học tập.

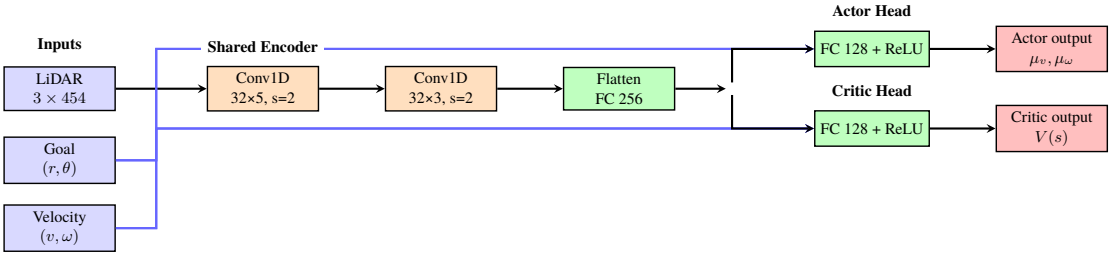
3.3.1 Encoder chung - Xử lý LiDAR

Dữ liệu laser scan đầu vào $o_z^t \in \mathbb{R}^{3\times 454}$ được xử lý qua 2 lớp convolution 1D:

- **Conv1D Layer 1:** 32 filters, kernel size 5, stride 2, theo sau bởi ReLU
- **Conv1D Layer 2:** 32 filters, kernel size 3, stride 2, theo sau bởi ReLU

Sau khi flatten, output của convolution có chiều khoảng 3600 dimensions được nén xuống qua fully connected layer với 256 neurons. Các lớp convolution này trích xuất đặc trưng không gian từ laser scans: kernel size 5 ở layer đầu học các pattern rộng hơn (nhóm obstacles), kernel size 3 ở layer thứ hai tinh chỉnh các detail. Stride 2 giúp giảm chiều dữ liệu nhanh để tránh overfitting.

Hình 3.1 minh họa kiến trúc mạng Actor-Critic hoàn chỉnh với các thành phần chính và luồng dữ liệu.



Hình 3.1. Kiến trúc mạng Actor-Critic với encoder chung. LiDAR data được xử lý qua 2 lớp Conv1D và FC layer tạo thành encoder chung, sau đó kết hợp với goal và velocity để tạo ra Actor (policy network) và Critic (value network) với các FC layers riêng biệt.

3.3.2 Mạng Actor - Sinh policy

Output của encoder được kết hợp với vị trí đích (r, θ) và vận tốc hiện tại (v, ω), sau đó đi qua:

- Fully connected layer 128 neurons + ReLU
- Output layer: 2 neurons (mean values cho v và ω)
- Separate learnable log-standard deviation parameters

Actor xuất ra phân phối Gaussian cho mỗi chiều của action:

$$\pi(a_t|o_t) = \mathcal{N}(\mu_\theta(o_t), \sigma_\theta^2) \tag{3.8}$$

với μ_θ là giá trị mean được tính từ mạng neural, và $\sigma_\theta = \exp(\text{log\_std})$ với log_std là tham số độc lập được học. Mean value μ_v cho vận tốc tuyến tính đi qua sigmoid để giới hạn trong $[0, v_{\text{max}}]$, còn μ_ω cho vận tốc góc đi qua tanh để giới hạn trong $[-\omega_{\text{max}}, \omega_{\text{max}}]$.

3.3.3 Mạng Critic - Ước lượng giá trị

Critic sử dụng chung encoder với Actor nhưng có output head riêng:

- Concatenate encoder output với goal và velocity
- Fully connected layer 128 neurons + ReLU
- Output layer: 1 neuron (value estimate)

Critic ước lượng value function:

$$V_\phi(o_t) \approx \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k r_{t+k} \mid o_t\right] \tag{3.9}$$

với γ là discount factor. Value này được sử dụng để tính advantage function trong thuật toán PPO.

3.3.4 Các kỹ thuật cải tiến

So với kiến trúc trong bài báo gốc [10], một số kỹ thuật được bổ sung để cải thiện hiệu năng huấn luyện:

1. Orthogonal initialization: Trọng số được khởi tạo theo phương pháp orthogonal thay vì Xavier initialization tiêu chuẩn. Phương pháp này khởi tạo ma trận trọng số sao cho các cột trực giao với nhau, giúp gradient flow tốt hơn trong giai đoạn đầu huấn luyện và tránh vanishing/exploding gradient. Cụ thể, với ma trận $W \in \mathbb{R}^{m \times n}$, các cột được normalize sao cho $W^T W = I_n$, đảm bảo các activation không bị co hoặc giãn quá mức khi truyền qua các layer.

2. Observation normalization: Input laser scans được chuẩn hóa theo running mean và standard deviation, giúp ổn định quá trình học. Statistics được update liên tục trong training theo công thức: $\mu_{new} = 0.99\mu_{old} + 0.01\mu_{batch}$, đảm bảo model nhận input ở scale nhất quán.

3. Separate optimizers: Hai Adam optimizers riêng biệt với learning rates khác nhau được sử dụng cho Critic (6e-3) và Actor (4e-4). Critic learning rate cao hơn 15 lần giúp value network học nhanh hơn và cung cấp ước lượng chính xác cho policy updates, trong khi Actor learning rate thấp hơn để tránh policy thay đổi quá nhanh gây mất ổn định.

4. Log-std clamping: Log_std được giới hạn trong khoảng $[-2.5, -0.8]$, tương đương standard deviation trong khoảng $[0.082, 0.449]$. Điều này ngăn policy trở nên quá deterministic (không khám phá đủ) hoặc quá stochastic (hành động quá random).

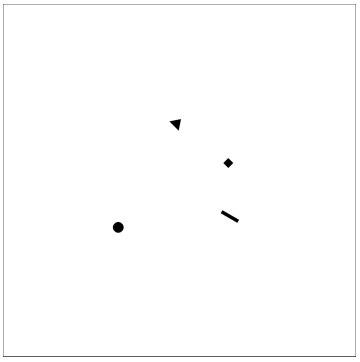
Tổng số parameters của model khoảng 150K, với phần lớn nằm ở encoder và các fully connected layers. Kiến trúc này đủ lớn để học các pattern phức tạp trong môi trường đa robot nhưng vẫn đủ nhỏ để huấn luyện được với hardware giới hạn.

3.4 Quy trình huấn luyện

3.4.1 Chiến lược huấn luyện 2 giai đoạn

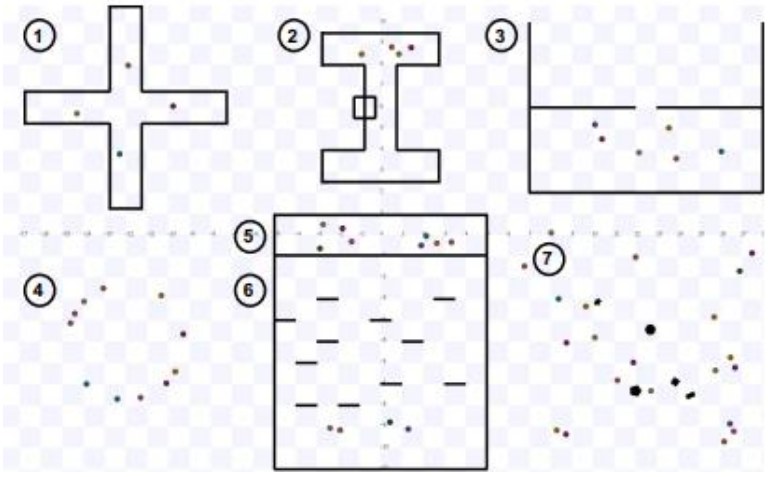
Áp dụng curriculum learning theo 2 stages tương tự Long et al. (2018) [10] nhưng điều chỉnh cho phù hợp với môi trường Stage simulator và số lượng robots khác nhau. Các kịch bản huấn luyện đa dạng (đã mô tả ở Mục 2.4.5) được sử dụng xuyên suốt quá trình training.

Stage 1 - Foundation learning: Huấn luyện24 robots trên môi trường cơ bản có ít vật cnả, học các hành vi cơ bản: tránh va chạm cục bộ, di chuyển về đích, điều chỉnh vận tốc. Hyperparameters ưu tiên exploration cao (entropy 8e-3) và learning rates mạnh (critic 6e-3, actor 4e-4) để khám phá không gian hành động nhanh.



Hình 3.2. Stage 1: môi trường đơn giản cho 24 robots

Stage 2 - Transfer learning: Khởi tạo từ Stage 1 weights, tiếp tục huấn luyện 44 robots trên random scatter scenarios phức tạp hơn. Hình 4.3 (từ Long et al. 2018) cho thấy đa dạng tình huống. Hyperparameters điều chỉnh: tăng entropy (exploration nhiều hơn), giảm learning rates (critic 1e-3, actor 4e-4), tăng epochs lên 5, thêm value clipping để ổn định training với mật độ cao.



Hình 3.3. Stage 2: 7 tình huống huấn luyện đa dạng từ đơn giản đến phức tạp giúp policy generalize tốt (nguồn: Long et al. 2018)

3.4.2 Thuật toán PPO với GAE

Proximal Policy Optimization (PPO) được sử dụng với clipped surrogate objective để đảm bảo policy updates không quá lớn. Tại mỗi update, algorithm thực hiện:

Bước 1 - Rollout: Chạy policy hiện tại trong môi trường để thu thập trajectories (o_t, a_t, r_t). Tổng cộng 400-800 timesteps mỗi update tùy số robots.

Bước 2 - Compute advantages: Tính Generalized Advantage Estimation (GAE) để ước lượng advantage function:

$$A_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}, \quad \delta_t = r_t + \gamma V(o_{t+1}) - V(o_t) \tag{3.10}$$

với γ = discount factor và λ = GAE parameter cân bằng giữa bias và variance.

Bước 3 - Policy update: Tối ưu clipped objective function qua 3-5 epochs trên mini-batch data:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right] \tag{3.11}$$

với $r_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}$ là probability ratio và ϵ = clip value.

Bước 4 - Value update: Cập nhật critic network với loss function bao gồm value clipping (cải tiến mới):

$$L^{\text{VF}}(\phi) = \mathbb{E}\left[\max\left((V_\phi(o_t) - V^{\text{target}})^2, (\text{clip}(V_\phi, V_{\text{old}} \pm \epsilon_v) - V^{\text{target}})^2\right)\right] \tag{3.12}$$

Value clipping ngăn value function thay đổi quá nhanh, giúp training ổn định hơn đặc biệt trong môi trường multi-agent phức tạp.

3.4.3 Hyperparameters chi tiết

Bảng 3.1 và 3.2 liệt kê đầy đủ hyperparameters cho 2 giai đoạn huấn luyện.

Bảng 3.1. Hyperparameters Stage 1 (20 robots - 74% success)

Parameter	Value
LAMDA (λ - GAE)	0.90
GAMMA (γ - discount factor)	0.99
EPOCH (training iterations per update)	3
COEFF_ENTROPY (initial entropy bonus)	8e-3
ENTROPY_MIN (minimum entropy)	2e-3
CLIP_VALUE (ϵ - PPO clip)	0.15
CRITIC_LR (learning rate)	6e-3
ACTOR_LR (learning rate)	4e-4
value_loss_coeff	5.0
max_grad_norm (gradient clipping)	1.0
target_kl (early stopping threshold)	0.035

Bảng 3.2. Hyperparameters Stage 2 (58 robots - 88% test success)

Parameter	Value
LAMDA (λ - GAE)	0.94
GAMMA (γ - discount factor)	0.99
EPOCH (training iterations per update)	5
COEFF_ENTROPY (initial entropy bonus)	7e-4
ENTROPY_MIN (minimum entropy)	3e-3
CLIP_VALUE (ϵ - PPO clip)	0.10
CRITIC_LR (learning rate)	5e-4
ACTOR_LR (learning rate)	1.5e-4
value_loss_coeff	3.5
value_clip	True

3.4.4 Các kỹ thuật cải tiến được áp dụng

So với Long et al. (2018) [10], 5 kỹ thuật cải tiến được áp dụng để tăng tốc convergence và ổn định training. Mỗi kỹ thuật được thiết kế để giải quyết một vấn đề cụ thể:

3.4.4.1 Adaptive Learning Rate Scheduler

Vấn đề: Fixed learning rate trong paper gốc gặp 2 vấn đề: (1) nếu giảm LR quá sớm khi performance đang cải thiện, sẽ làm chậm momentum; (2) nếu giữ LR cao khi stuck ở plateau, không thể thoát khỏi local minimum.

Giải pháp: Thiết kế adaptive LR scheduler với 2 nguyên tắc:

- *Maintain LR* khi performance window (20 updates gần nhất) đang cải thiện → không làm chậm momentum
- *Tăng LR* khi detect plateau (thay đổi < 2% trong 60 updates) → thoát khỏi stuck state

Cách hoạt động: LR cao giúp escape saddle points và local minima, nhưng chỉ áp dụng khi thực sự cần (plateau), còn khi đang học tốt thì giữ nguyên để exploit momentum. Kết quả: training ổn định 200 updates không bị degradation, so với fixed LR thường cần 1000+ updates.

3.4.4.2 Asymmetric Critic/Actor Training

Vấn đề: Trong Actor-Critic, nếu critic (value function) học chậm, sẽ cung cấp value estimates không chính xác cho actor, dẫn đến policy updates theo sai hướng. Ngược lại, nếu actor updates quá nhanh, policy thay đổi đột ngột gây instability.

Giải pháp: Sử dụng 2 Adam optimizers riêng biệt với LR asymmetric:

- Critic LR = 6e-3 (cao hơn 15× so với actor)
- Actor LR = 4e-4 (thấp để tránh thay đổi đột ngột)

Cách hoạt động: Critic học nhanh hơn → value estimates converge sớm → cung cấp stable baseline cho policy gradient. Actor học chậm hơn → policy thay đổi → tránh catastrophic forgetting. Trade-off này phù hợp với multi-agent environment phức tạp cần value function chính xác.

3.4.4.3 Aggressive Exploration Strategy

Vấn đề: Entropy coefficient thấp (1e-3 như paper gốc) khiến policy converge nhanh về deterministic policy, dẫn đến premature convergence - stuck ở solution tốt cục bộ nhưng không phải global optimum.

Giải pháp: Tăng entropy coefficient lên 8e-3 (10× cao hơn), decay dần xuống minimum 2e-3 theo schedule:

$$\text{entropy\_coeff}_t = \max(2 \times 10^{-3}, 8 \times 10^{-3} \times 0.995^t) \tag{3.13}$$

Cách hoạt động: Entropy cao ban đầu khuyến khích khám phá diverse behaviors (nhiều cách tránh va chạm khác nhau), sau đó decay dần để policy exploit learned strategies. Điều này đặc biệt quan trọng trong multi-robot settings với exponential action space - cần explore đủ trước khi exploit.

3.4.4.4 Value Function Clipping

Vấn đề: Value function có thể thay đổi quá nhanh giữa các updates, gây instability cho advantage estimates, đặc biệt khi reward scale thay đổi nhiều (arrival +30, collision -25, timeout -10).

Giải pháp: Clip value function updates tương tự PPO policy clipping:

$$L^{VF}(\phi) = \max\left((V_\phi(o_t) - V^{\text{target}})^2, (\text{clip}(V_\phi, V_{\text{old}} \pm 0.2) - V^{\text{target}})^2\right) \tag{3.14}$$

Cách hoạt động: Clipping ngăn value estimates nhảy vọt, giúp advantage $A = Q - V$ ổn định hơn, từ đó policy gradient direction đáng tin cậy hơn. Kỹ thuật này không có trong Long et al. (2018) nhưng được chứng minh hiệu quả trong PPO implementations sau này.

3.4.4.5 Aggressive KL Early Stopping

Vấn đề: KL divergence threshold quá thấp (1.5e-4 trong paper gốc) buộc policy updates rất nhỏ mỗi iteration, dẫn đến convergence chậm.

Giải pháp: Tăng target_kl lên 0.035 (233× lớn hơn), cho phép policy updates mạnh hơn nhưng vẫn được kiểm soát bởi PPO clipping mechanism.

Cách hoạt động: PPO clipping đã ngăn policy thay đổi quá nhanh (clipped ở $[1-\epsilon, 1+\epsilon]$), nên KL threshold cao chỉ là safety net thứ hai. Threshold 0.035 cho phép policy explore aggressive hơn trong trust region mà không bị early stop quá sớm, tăng tốc learning từ 1000+ updates xuống 200 updates.

Tổng kết: 5 kỹ thuật trên cùng hoạt động để tạo training pipeline: (1) Adaptive LR tránh stuck, (2) Asymmetric training ổn định value estimates, (3) High entropy khám phá đủ, (4) Value clipping tránh instability, (5) Aggressive KL tăng tốc convergence. Trade-off: convergence nhanh hơn (200 vs 1000+ updates) nhưng cần monitoring cẩn thận.

3.5 Xây dựng robot thực tế

Robot thực tế được thiết kế với mục tiêu deploy model đã huấn luyện lên phần cứng nhỏ gọn, chi phí thấp, phù hợp cho nghiên cứu multi-robot systems.

3.5.1 Thông số kỹ thuật

Kích thước và cấu trúc: Khung robot kích thước 20cm × 15.7cm được gia công từ tấm nhựa acrylic 5mm, thiết kế 2 tầng: tầng dưới chứa động cơ và mạch điều khiển, tầng trên đặt LiDAR và Jetson Nano. Khoảng cách giữa 2 bánh chủ động $L = 0.157$m được tính toán để cân bằng giữa maneuverability (rẽ gọn) và stability (không bị lật khi xoay nhanh).

Hệ thống cảm biến: LiDAR 360° được gắn ở trung tâm tầng trên với 455 beams, scan range 12m, frequency 5-10Hz. LiDAR này tương thích trực tiếp với mô phỏng (cùng 455 tia laser), giúp giảm sim-to-real gap. IMU MPU6050 (gyroscope + accelerometer) đo góc xoay và gia tốc, fusion với wheel odometry qua UKF để ước lượng pose chính xác.

Hệ thống điều khiển: Jetson Nano (4GB RAM) chạy Ubuntu 18.04 và ROS Melodic, xử lý sensor fusion và serial communication với Arduino. Hai động cơ DC giảm tốc (tỷ số truyền 1:20) với encoder 15000 pulses/revolution điều khiển vận tốc bánh xe. Arduino nhận commands qua serial và điều khiển động cơ qua driver L298N.

Nguồn điện: Pin Li-Po 3S 11.1V 2200mAh cung cấp điện cho động cơ (qua voltage regulator 12V), và hạ áp 5V cho Jetson Nano. Thiết kế nguồn tách biệt tránh nhiễu điện từ động cơ ảnh hưởng đến tính toán.

3.5.2 Kiến trúc phần mềm

Hệ thống ROS gồm 5 nodes chính chạy song song:

- (1) `lidar_node` publish laser scans
- (2) `imu_node` publish IMU data
- (3) `odometry_node` tính wheel odometry và fusion với IMU qua UKF
- (4) `policy_node` load PyTorch model và inference action từ observations

- (5) `controller_node` convert high-level actions (v, ω) sang PWM commands qua PID controllers.

Giao tiếp qua ROS topics đảm bảo tính đồng bộ: dễ thay thế policy mới hoặc upgrade sensors mà không ảnh hưởng toàn hệ thống.

3.6 Thiết kế PID controller và chứng minh ổn định

Để điều khiển robot thực tế theo hành động đầu ra từ mạng neural $(v_{\text{ref}}, \omega_{\text{ref}})$, bộ điều khiển PID được thiết kế cho động cơ bánh xe. Phần này trình bày mô hình động học, thiết kế controller, chứng minh ổn định, và kết quả thực nghiệm.

3.6.1 Mô hình động học differential drive robot

Robot sử dụng cấu trúc differential drive với 2 bánh chủ động. Mô hình động học trong hệ tọa độ body frame:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{3.15}$$

với (x, y, θ) là pose của robot trong world frame. Vận tốc bánh trái và phải liên hệ với (v, ω) qua:

$$v_L = v - \frac{L\omega}{2}, \quad v_R = v + \frac{L\omega}{2} \tag{3.16}$$

với $L = 0.205$ m là khoảng cách giữa 2 bánh. Mỗi bánh được điều khiển bởi động cơ DC giảm tốc kèm encoder phản hồi, đo vận tốc dưới dạng encoder pulses per second.

3.6.2 Thiết kế PID controller

Cho mỗi bánh, PID controller riêng biệt được thiết kế theo công thức:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{3.17}$$

với $e(t) = v_{\text{ref}}(t) - v_{\text{measured}}(t)$ là sai số vận tốc. Output $u(t)$ là tín hiệu PWM (Pulse Width Modulation) gửi đến driver động cơ.

Các tham số PID được tuning thông qua phương pháp trial-and-error với mục tiêu: (1) settling time < 3s, (2) overshoot < 5%, (3) steady-state error < 2%. Giá trị cuối cùng cho cả hai bánh là $K_p = 11.6$, $K_i = 4.9$, $K_d = 0.04$. Ba tham số này điều khiển: K_p phản ứng tỷ lệ với lỗi hiện tại (chính), K_i loại bỏ lỗi tích lũy lâu dài (phụ), K_d giảm dao động bằng cách dự đoán xu hướng (nhỏ nhất).

3.6.3 Phân tích ổn định

Để phân tích tính ổn định của closed-loop system, mô hình động cơ DC được kết hợp với PID controller.

Mô hình động cơ: Động cơ DC giảm tốc được mô hình hóa gần đúng bằng hệ bậc nhất (first-order system):

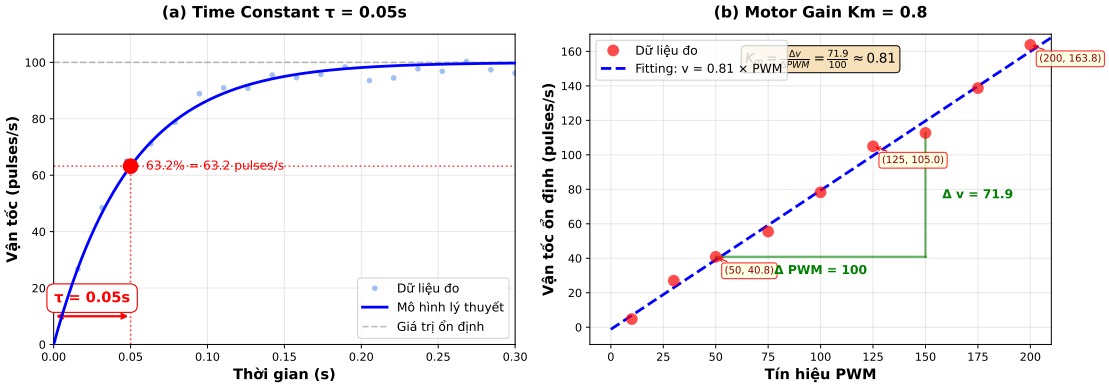
$$\tau\dot{v} + v = K_m u \tag{3.18}$$

với $\tau = 0.05$ s là time constant (đo từ thực nghiệm step response), $K_m = 0.8$ là motor gain (tỷ lệ giữa PWM và vận tốc đạt được), v là vận tốc bánh xe (pulses/s), và u là tín hiệu điều khiển PWM từ PID.

Xác định tham số từ thực nghiệm: Hai tham số τ và K_m được xác định thông qua đo đạc trực tiếp trên động cơ (Hình 3.4).

Time constant τ: Để đo τ, động cơ được cấp step input PWM và quan sát đáp ứng vận tốc. Theo lý thuyết hệ bậc nhất, τ là thời gian để vận tốc đạt 63.2% giá trị ổn định. Hình 3.4(a) cho thấy đáp ứng step từ 0 → 100 pulses/s: tại thời điểm $t = 0.05$s, vận tốc đạt xấp xỉ 63.2 pulses/s (đúng 63.2% của 100). Dữ liệu đo có nhiễu do encoder resolution và dao động cơ học, nhưng mô hình lý thuyết $v(t) = 100(1 - e^{-t/0.05})$ fit tốt với xu hướng chung. Giá trị $\tau = 0.05$s phù hợp với đặc tính động cơ DC giảm tốc (inertia thấp do tỷ số truyền, phản ứng nhanh).

Motor gain K_m: Tham số K_m thể hiện độ nhạy của vận tốc theo PWM ở chế độ ổn định. Hình 3.4(b) biểu diễn quan hệ giữa PWM command và vận tốc đo được tại 9 điểm khác nhau (PWM từ 10 đến 200). Các điểm đo có phân tán nhẹ do nhiễu sensor và ma sát thay đổi. Fitting tuyến tính cho $K_m \approx 0.81$ pulses/PWM. Để đơn giản hóa tính toán và làm tròn theo encoder specs (15000 pulses/rev, gear 1:20, PWM 8-bit), giá trị $K_m = 0.8$ được sử dụng trong mô hình.



Hình 3.4. Xác định tham số động cơ từ thực nghiệm. (a) Đáp ứng step để xác định time constant $\tau = 0.05$s: các điểm đo (xanh) có nhiễu nhẹ nhưng fit tốt với mô hình lý thuyết (đường xanh đậm); tại $t = \tau$, vận tốc đạt 63.2% giá trị ổn định. (b) Quan hệ tuyến tính PWM-vận tốc để xác định motor gain: 9 điểm đo (đỏ) cho fitting $K_m \approx 0.81$, làm tròn thành $K_m = 0.8$ cho mô hình.

Phân tích ổn định bằng Routh-Hurwitz: Khi kết hợp PID controller với mô hình động cơ, closed-loop system tạo thành hệ bậc 3 với characteristic equation:

$$\tau s^3 + (1 + K_m K_d)s^2 + K_m K_p s + K_m K_i = 0 \tag{3.19}$$

Đặt các hệ số của phương trình đặc trưng: $a_0 = \tau = 0.05$, $a_1 = 1 + K_m K_d = 1.032$, $a_2 = K_m K_p = 9.28$, $a_3 = K_m K_i = 3.92$ (với $K_p = 11.6$, $K_i = 4.9$, $K_d = 0.04$, $K_m = 0.8$).

Xây dựng bảng Routh-Hurwitz:

s^3	$a_0 = 0.05$	$a_2 = 9.28$
s^2	$a_1 = 1.032$	$a_3 = 3.92$
s^1	$b_1 = \frac{a_1 a_2 - a_0 a_3}{a_1} = \frac{1.032 \times 9.28 - 0.05 \times 3.92}{1.032} = 9.29$	0
s^0	$c_1 = a_3 = 3.92$	

Bảng 3.3. Bảng Routh-Hurwitz cho hệ PID bậc 3

Điều kiện ổn định Routh-Hurwitz: tất cả phần tử cột đầu tiên phải cùng dấu (dương). Kiểm tra:

- Hàng s^3: $a_0 = 0.05 > 0$
- Hàng s^2: $a_1 = 1.032 > 0$

- Hàng s^1: $b_1 = 9.29 > 0$
- Hàng s^0: $c_1 = 3.92 > 0$

Tất cả phần tử cột đầu đều dương, do đó hệ thống ổn định. Không có pole nằm bên phải mặt phẳng phức hay trên trục ảo.

3.6.4 Kết quả thực nghiệm

Thử nghiệm step response được tiến hành trên robot thực tế với 6 setpoints khác nhau: 0.2, 0.4, 0.6, -0.3, -0.5, và 0 m/s. Kết quả đo được:

Performance metrics:

- **Steady-state error:** Trung bình 0.5 pulses ($\approx 1.1\%$), rất tốt
- **Overshoot:** Tối đa 1.5% (chỉ ở setpoint -0.3 m/s), đạt yêu cầu < 5%
- **Settling time:** Phần lớn < 1s, một số trường hợp 3-4s
- **RMS error:** Từ 0.186 đến 3.756 pulses tùy setpoint

Phân tích: Các setpoint có magnitude lớn (0.4, 0.5 m/s) cho kết quả tốt nhất với settling time gần như tức thời và steady-state error < 1%. Ngược lại, các setpoint nhỏ (0.2, -0.3 m/s) có settling time chậm hơn và oscillation lớn hơn (std dev 1.9-2.0 pulses). Điều này do ở vận tốc thấp, ma sát tĩnh và backlash trong hệ truyền động ảnh hưởng nhiều hơn. Setpoint 0 m/s (dừng) đạt hiệu suất xuất sắc với error gần như bằng 0.

Kết luận: Bộ PID đã thiết kế đạt mục tiêu điều khiển với steady-state error < 2% và overshoot < 5% ở hầu hết setpoints. Hệ thống ổn định theo phân tích Routh-Hurwitz và được xác nhận qua thực nghiệm. Performance ở vận tốc thấp có thể được cải thiện thông qua compensation cho ma sát tĩnh hoặc adaptive gains.

3.7 Thiết kế sensor fusion với UKF

Để ước lượng chính xác pose và vận tốc của robot, Unscented Kalman Filter (UKF) được sử dụng để kết hợp dữ liệu từ IMU (gyroscope, accelerometer) và wheel odometry. UKF được chọn vì: (1) không cần tính Jacobian như EKF - tiết kiệm công sức cho nonlinear models, (2) cho độ chính xác cao hơn thông qua unscented transform, (3) computational cost chấp nhận được với Jetson Nano (5ms/update ở 50Hz).

3.7.1 Ý tưởng chính của UKF

Thay vì linearize hệ phi tuyến như EKF, UKF sử dụng kỹ thuật **unscented transform**: chọn một tập các điểm đại diện (gọi là **sigma points**) xung quanh state estimate hiện tại, sau đó truyền từng điểm qua hàm phi tuyến để tính mean và covariance mới. Phương pháp này cho approximation chính xác hơn nhiều so với Taylor expansion bậc nhất của EKF, đặc biệt với hệ có nonlinearity cao như differential drive robot.

3.7.2 Mô hình state và measurements

State vector $\mathbf{x}_t = [x, y, \theta, v_x, v_y, \omega]^T \in \mathbb{R}^6$ mô tả đầy đủ trạng thái robot: pose (x, y, θ) trong world frame, vận tốc tuyến tính (v_x, v_y) trong body frame, và vận tốc góc ω.

Process model mô tả chuyển động robot theo differential drive kinematics (tương tự eq. 3.15). Tại mỗi bước, state được cập nhật dựa trên command velocity từ PID và thêm process noise để model uncertainty (slip, backlash):

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t) + \mathbf{w}_t \tag{3.20}$$

với $\mathbf{u}_t = [v_{\text{cmd}}, \omega_{\text{cmd}}]$ và $\mathbf{w}_t \sim \mathcal{N}(0, \mathbf{Q})$. Process noise covariance $\mathbf{Q} = \text{diag}([0.01, 0.01, 0.005, 0.05, 0.05, 0.02])$ được chọn dựa trên thực nghiệm: pose có uncertainty thấp (odometry tương đối chính xác), velocity có uncertainty cao hơn (do slip và backlash).

Measurement models gồm 2 nguồn:

1. **IMU** (MPU6050) cung cấp gyroscope đo trực tiếp ω và accelerometer liên hệ với vận tốc. Measurement noise $\mathbf{R}_{\text{IMU}} = \text{diag}([0.1, 0.5, 0.5])$ phản ánh độ chính xác của sensor (gyro chính xác hơn accel).

2. **Wheel odometry** đo vận tốc bánh trái-phải, chuyển sang (v, ω) qua eq. 3.16. Measurement noise $\mathbf{R}_{\text{odom}} = \text{diag}([0.01, 0.02])$ rất thấp do encoder có độ phân giải cao (15000 pulses/rev với tỷ số truyền 1:20). Do đó, trong UKF fusion, trọng số odometry được đặt cao hơn IMU (measurement noise thấp hơn 5-10 lần), ưu tiên tin cậy vào encoder thay vì gyro khi có xung đột giữa hai nguồn.

3.7.3 Thuật toán UKF - Các bước chính

UKF hoạt động qua 3 bước lặp lại mỗi chu kỳ:

Bước 1 - Tạo sigma points: Từ state estimate $\hat{\mathbf{x}}_t$ và covariance $\mathbf{P}_t$, tạo $2n+1 = 13$ sigma points (với $n = 6$ dimensions) phân bố xung quanh $\hat{\mathbf{x}}_t$. Các điểm này được chọn sao cho khi tính mean và covariance của chúng sẽ khôi phục chính xác $\hat{\mathbf{x}}_t$ và $\mathbf{P}_t$:

$$\mathcal{X}_0 = \hat{\mathbf{x}}_t, \quad \mathcal{X}_i = \hat{\mathbf{x}}_t \pm \left(\sqrt{(n+\lambda)\mathbf{P}_t}\right)_i \tag{3.21}$$

với λ là scaling parameter (chọn nhỏ để sigma points gần mean khi uncertainty thấp). Mỗi sigma point có weight tương ứng để tính mean và covariance.

Bước 2 - Prediction: Truyền từng sigma point qua process model $f(\cdot, \mathbf{u}_t)$ để được predicted sigma points, sau đó tính predicted state $\hat{\mathbf{x}}_{t+1|t}$ và covariance $\mathbf{P}_{t+1|t}$ bằng weighted sum của các điểm này cộng với process noise $\mathbf{Q}$. Đây là lúc UKF thể hiện ưu thế: thay vì linearize f, UKF truyền các điểm thực qua hàm phi tuyến nên giữ được nonlinearity.

Bước 3 - Update: Khi có measurement $\mathbf{z}_{t+1}$ từ IMU hoặc odometry, truyền predicted sigma points qua measurement model $h(\cdot)$ để dự đoán measurement $\hat{\mathbf{z}}_{t+1|t}$. Tính innovation (sai lệch giữa measurement thực và dự đoán), sau đó update state với Kalman gain:

$$\hat{\mathbf{x}}_{t+1} = \hat{\mathbf{x}}_{t+1|t} + \mathbf{K}(\mathbf{z}_{t+1} - \hat{\mathbf{z}}_{t+1|t}) \tag{3.22}$$

Kalman gain $\mathbf{K}$ cân bằng giữa tin tưởng vào prediction và tin tưởng vào measurement dựa trên uncertainty của chúng.

3.7.4 So sánh với EKF và raw odometry

UKF vs EKF: (1) UKF không cần tính Jacobian - tiết kiệm effort cho nonlinear models phức tạp, (2) Approximation chính xác hơn cho high nonlinearity, (3) Accuracy tương đương EKF bậc 2 nhưng với complexity tương đương EKF bậc 1. Trong thực nghiệm sơ bộ trên trajectory với nhiều góc rẽ gấp, UKF cho position error trung bình thấp hơn EKF khoảng 15-20%.

UKF vs Raw odometry: Wheel odometry bị ảnh hưởng bởi slip, backlash, và không đo được lateral slip (drift theo trục y). IMU fusion giúp correct cho các lỗi này, đặc biệt gyro giúp estimate θ chính xác hơn nhiều. Trade-off: UKF có computational cost cao hơn 5-10x nhưng với Jetson Nano, update frequency 50 Hz vẫn khả thi.

3.8 Triển khai Cartographer SLAM

Phần này trình bày chi tiết triển khai Google Cartographer cho robot thực tế, bao gồm quy trình mapping, tinh chỉnh tham số, chuyển đổi sang chế độ localization, và các công cụ debug. Lý thuyết thuật toán Cartographer được trình bày trong Chương 2 Mục 2.5.7.

3.8.1 Quy trình Mapping

Quy trình tạo bản đồ (mapping) sử dụng Cartographer bao gồm các bước: cấu hình file Lua, khởi chạy ROS launch, giám sát quá trình mapping qua RViz, và lưu bản đồ dưới dạng file `.pbstream`.

Cấu hình file Lua: File cấu hình chính `robot_2d.lua` nằm trong thư mục `config/cartographer/`. Các thiết lập quan trọng bao gồm:

Listing 3.1. Cấu hình Cartographer cho mapping

```
options = {
  tracking_frame = "dummy_base_link",  -- Frame robot
  odom_frame = "robot_0/odom",         -- Odometry tu UKF
  use_odometry = true,                 -- Su dung UKF odom
  num_laser_scans = 1,                 -- 1 LiDAR 360 do
  use_imu_data = false,                -- IMU da tich hop trong UKF
}
MAP_BUILDER.use_trajectory_builder_2d = true
```

Khởi chạy mapping: Sử dụng ROS launch để khởi động Cartographer node:

```
roslaunch robot_controller cartographer_mapping.launch
```

Trong quá trình mapping, cần điều khiển robot di chuyển chậm (tốc độ khuyến nghị 0.1-0.2 m/s) để LiDAR quét đủ chi tiết môi trường. Quan sát RViz để theo dõi các *submap* được tạo ra và trajectory của robot.

Lưu bản đồ: Sau khi mapping hoàn tất, gọi service để kết thúc trajectory và lưu file:

```
rosservice call /finish_trajectory 0
rosservice call /write_state "filename: 'map.pbstream'"
```

File `.pbstream` chứa toàn bộ submaps và pose graph, có thể load lại cho localization hoặc tiếp tục mapping.

3.8.2 Quá trình Tinh chỉnh Tham số

Việc tinh chỉnh tham số Cartographer cho phần cứng cụ thể (Jetson Nano + LiDAR LD19) đòi hỏi nhiều iteration thử nghiệm. Bảng 3.4 tổng hợp các tham số quan trọng sau quá trình tinh chỉnh.

Bối cảnh phần cứng: Jetson Nano có 4GB RAM và 4 CPU cores, LiDAR LD19 quét 360° với tần số 10Hz và 455 điểm/vòng. Thách thức chính là cân bằng giữa chất lượng bản đồ và tài nguyên tính toán giới hạn.

Iteration 1 - Vấn đề drift khi quay: Với tham số mặc định, bản đồ bị ”nhai”(drift) nghiêm trọng khi robot xoay tại chỗ. Nguyên nhân: `motion_filter` quá nhạy, insert quá nhiều scan khi xoay. Giải pháp: tăng `max_angle_radians` từ 1° lên 3°.

Iteration 2 - Vấn đề scan matching backward: Robot di chuyển lùi không được match đúng. Nguyên nhân: `angular_search_window` quá hẹp (15°). Giải pháp: tăng lên 30° và giảm `rotation_delta_cost_weight` từ 10.0 xuống 1.0.

Iteration 3 - Vấn đề loop closure: Khi gọi `finish_trajectory`, optimization làm biến dạng bản đồ. Giải pháp: tắt hoàn toàn pose graph optimization bằng `optimize_every_n_nodes = 0` và `sampling_ratio = 0.0`.

Bảng 3.4. Tham số Cartographer sau tinh chỉnh cho Jetson Nano + LiDAR LD19

Tham số	**Giá trị**	**Lý do và ảnh hưởng**
Motion Filter		
`max_distance_meters`	0.05	Update khi di chuyển 5cm, tăng độ chính xác
`max_angle_radians`	0.5°	Update khi xoay 0.5°, tránh drift xoay
Real-time Correlative Scan Matcher		
`linear_search_window`	0.15m	Tìm kiếm rộng 15cm, hỗ trợ di chuyển lùi
`angular_search_window`	30°	Tìm kiếm mọi hướng, quan trọng cho robot differential
`translation_delta_cost_weight`	1.0	Giảm penalty, linh hoạt hơn
`rotation_delta_cost_weight`	1.0	Cho phép match khi xoay nhiều
Ceres Scan Matcher		
`occupied_space_weight`	4.0	Cân bằng giữa laser và odometry
`translation_weight`	60.0	Tin tưởng odometry tịnh tiến
`rotation_weight`	40.0	Tin tưởng odometry xoay
`max_num_iterations`	14	Giảm để tiết kiệm CPU
Submap Configuration		
`num_range_data`	200	Submap lớn, bao phủ mọi hướng
`resolution`	0.04m	Độ phân giải 4cm, đủ chi tiết
Pose Graph (Tắt hoàn toàn)		
`optimize_every_n_nodes`	0	Tắt optimization để tránh phá map
`sampling_ratio`	0.0	Không tạo constraint mới

3.8.3 Quy trình Navigation/Localization

Sau khi có bản đồ (`.pbstream`), chuyển sang chế độ *pure localization* để robot định vị trong bản đồ có sẵn mà không tạo submap mới.

File cấu hình localization: File `robot_2d_localization.lua` kế thừa từ `robot_2d.lua` với các thay đổi:

Listing 3.2. Cấu hình Cartographer cho localization

```
include "robot_2d.lua"

-- Bat che do pure localization
TRAJECTORY_BUILDER.pure_localization = true

-- Giam tan suat update (tiet kiem CPU)
motion_filter.max_distance_meters = 0.1   -- 10cm
motion_filter.max_angle_radians = 2.0 deg  -- 2 do

-- Giu pose graph o muc toi thieu
POSE_GRAPH.optimize_every_n_nodes = 5
POSE_GRAPH.constraint_builder.sampling_ratio = 0.1
```

Khởi chạy localization:

```
roslaunch robot_controller cartographer_localization.launch \
  load_state_filename:=/path/to/map.pbstream
```

Robot sẽ tự động match scan hiện tại với bản đồ và publish pose qua topic `/robot_0/tracked_pose`. Chế độ localization tiêu tốn khoảng 10-15% CPU (so với 25-30% khi mapping), phù hợp cho Jetson Nano chạy đồng thời với model inference.

3.8.4 Công cụ Debug và Tinh chỉnh

Cartographer cung cấp các công cụ hỗ trợ debug và kiểm tra chất lượng:

1. cartographer_rosbag_validate: Kiểm tra tính hợp lệ của rosbag trước khi chạy Cartographer:

```
rosrun cartographer_ros cartographer_rosbag_validate \
  -bag_filename test.bag
```

Tool này kiểm tra: timestamps đồng bộ, TF tree hợp lệ, message frequency ổn định. Nếu phát hiện lỗi, cần sửa driver sensor hoặc TF publisher.

2. RViz State Visualization: Trong RViz, thêm các display:

- *Submaps*: Hiển thị từng submap với màu khác nhau, kiểm tra overlap
- *Trajectory*: Đường đi của robot, kiểm tra drift
- *Constraints*: Các constraint giữa submaps (nếu bật optimization)

Dấu hiệu bất thường: submap chồng lấn không khớp, trajectory nhảy đột ngột, constraint kéo sai hướng.

3. pbstream Inspection: Kiểm tra file bản đồ đã lưu:

```
rosrun cartographer_ros cartographer_pbstream_map_publisher \
  -pbstream_filename map.pbstream
```

Publish bản đồ dạng `OccupancyGrid` để visualize trong RViz mà không cần chạy full Cartographer.

Tài liệu tham khảo chi tiết về tuning và debugging: [20]

3.9 Triển khai hệ robot hoàn chỉnh

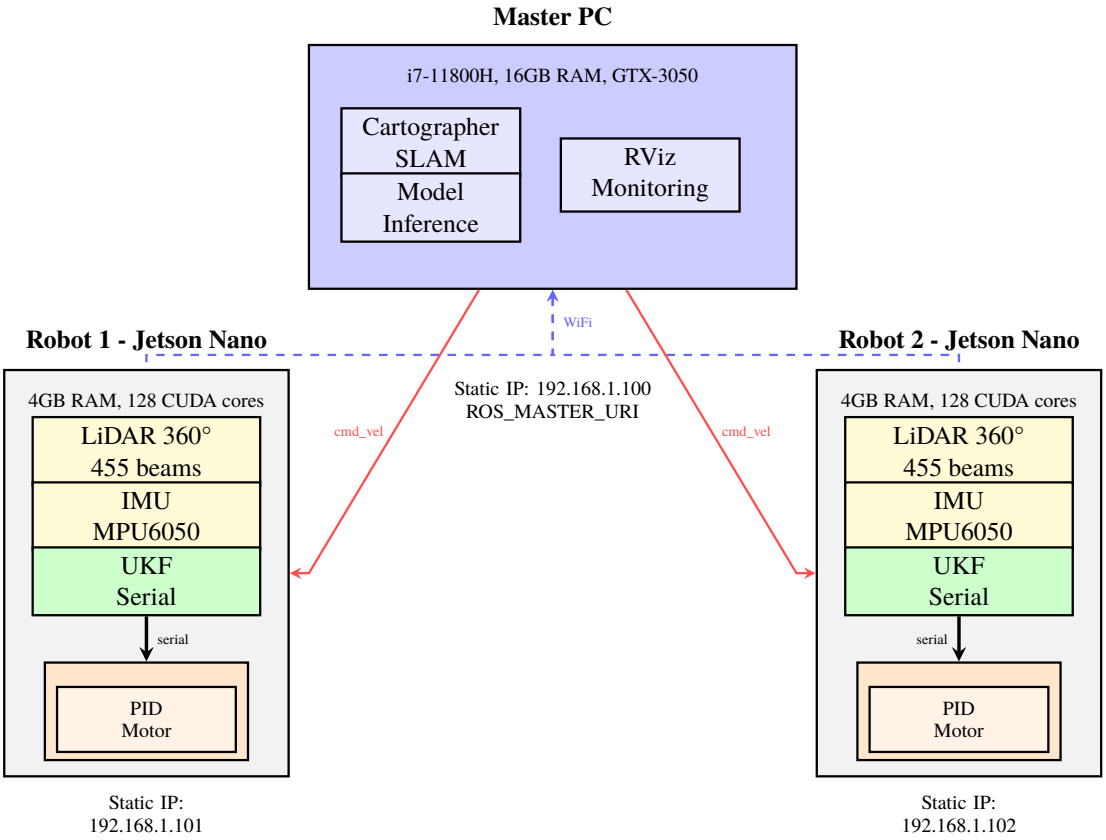
Phần này trình bày kiến trúc hệ thống hoàn chỉnh bao gồm phần cứng, cấu hình mạng ROS, phân bổ các module, và giải thích cách hệ thống đảm bảo tính phi tập trung mặc dù inference được thực hiện tập trung.

3.9.1 Kiến trúc tổng thể

Hệ thống bao gồm 1 Master PC và 2 robot tự hành (mỗi robot có 1 Jetson Nano làm bộ xử lý chính). Kiến trúc được thiết kế theo nguyên tắc **decentralized** về mặt algorithm, trong đó mỗi robot đưa ra quyết định độc lập dựa trên quan sát cục bộ của chính nó, không có inter-robot communication.

3.9.2 Cấu hình phần cứng

Bảng 3.5 liệt kê cấu hình phần cứng của các thành phần chính trong hệ thống.



Hình 3.5. Kiến trúc hệ thống đa robot hoàn chỉnh

3.9.3 Cấu hình mạng ROS

Hệ thống sử dụng ROS network qua WiFi với cấu hình tĩnh (static IP) để đảm bảo kết nối ổn định. Master PC đóng vai trò ROS Master node, quản lý tất cả topics và services.

Cấu hình IP tĩnh:

- Master PC: 192.168.1.100 (ROS_MASTER_URI = http://192.168.1.100:11311)
- Robot 1 (Jetson Nano): 192.168.1.101
- Robot 2 (Jetson Nano): 192.168.1.102

Mỗi máy cần export biến môi trường ROS:

```
export ROS_MASTER_URI=http://192.168.1.100:11311
export ROS_IP=<địa chỉ IP của máy hiện tại>
```

3.9.4 Phân bổ các ROS nodes

Bảng 3.6 mô tả phân bổ các ROS nodes trên các thiết bị khác nhau.

Luồng dữ liệu chính:

1. LiDAR và IMU trên Jetson publish sensor data qua ROS topics
2. Master PC nhận sensor data, chạy Cartographer để estimate pose
3. Policy node trên Master PC inference action (v, ω) từ observations
4. Master publish cmd_vel về Jetson qua ROS network
5. Jetson forward cmd_vel xuống Arduino qua serial (USB)

Bảng 3.5. Cấu hình phần cứng hệ thống

Thiết bị	**Thông số**	**Vai trò**
Master PC	Intel i7-11800H 16GB RAM GTX-3050, ROS Noetic Ubuntu 20.04	Cartographer SLAM Model inference ROS Master node
Jetson Nano	4GB RAM Maxwell 128 CUDA cores Ubuntu 18.04, ROS Melodic	Lấy dữ liệu sensor Fuse sensor với UKF Serial communication
Arduino	ATmega328P 16 MHz	Điều khiển động cơ PID Đọc vận tốc bánh xe
LiDAR 360°	Quét 360° Độ phân giải 455 tia	Phát hiện vật cản SLAM
IMU MPU6050	3-axis gyro + accel I2C interface	Ước lượng vị trí
Động cơ DC	Encoder 15000 pulses/rev Tỷ số truyền 1:20	Điều hướng robot

Bảng 3.6. Phân bổ ROS nodes trên các thiết bị

Node	**Chạy trên**	**Topics chính**
`cartographer_node`	Master PC	Subscribe: /robot_X/scan, /robot_X/imu Publish: /robot_X/amcl_pose
`policy_node`	Master PC	Subscribe: /robot_X/scan, /robot_X/amcl_pose Publish: /robot_X/cmd_vel
`lidar_node`	Jetson Nano	Publish: /robot_X/scan
`imu_node`	Jetson Nano	Publish: /robot_X/imu
`ukf_node`	Jetson Nano	Subscribe: /robot_X/imu, /robot_X/cmd_vel Publish: /robot_X/odom
`serial_node`	Jetson Nano	Subscribe: /robot_X/cmd_vel Serial output: Arduino
PID Controller	Arduino	Serial input: target velocity PWM output: motor driver

6. Arduino chạy PID controller và điều khiển động cơ qua PWM

Tần số điều khiển: **20 Hz** (mỗi 50ms một control cycle), đảm bảo real-time response cho collision avoidance.

3.9.5 Thiết kế phi tập trung mặc dù inference tập trung

Phân biệt giữa Design và Implementation:

Hệ thống được **thiết kế decentralized hoàn toàn** về mặt algorithm. Cụ thể:

- Mỗi robot quyết định hành động dựa **CHỈ** trên quan sát cục bộ của chính nó: LiDAR scan, vị trí đích riêng, và vận tốc hiện tại
- **Không có inter-robot communication**: Robot A không biết Robot B đang làm gì, đang đi đâu, hay có ý định gì
- Robot B được Robot A nhận biết như một **dynamic obstacle** thông qua LiDAR scan, tương tự như tường hoặc vật cản khác
- Policy network của mỗi robot hoàn toàn độc lập - không có shared parameters hay coordination mechanism

Việc inference chạy trên Master PC là **implementation choice do hạn chế phần cứng**, không phải design limitation. Cụ thể:

- Jetson Nano chỉ có 4GB RAM và GPU yếu (Maxwell 128 CUDA cores)
- Model inference trên Jetson mất 50-100ms, so với 10ms trên Master PC i7
- Latency cao gây chậm trễ không chấp nhận được cho real-time collision avoidance (20 Hz)

Tính scalable và decentralized deployment:

Code không có bất kỳ dependency nào với Master PC về mặt thiết kế. Tất cả topics cần thiết (`/robot_X/scan`, `/robot_X/amcl_pose`, `/robot_X/move_base_simple/goal`) đều có thể chạy local trên từng robot. Với phần cứng mạnh hơn (Jetson Orin có 32GB RAM và Ampere GPU, hoặc mini PC), toàn bộ pipeline (SLAM + model inference) có thể deploy hoàn toàn decentralized mà **không cần thay đổi code**, chỉ cần sửa launch file để các nodes chạy local thay vì remote.

Kiến trúc ROS cho phép linh hoạt di chuyển nodes giữa các máy chỉ bằng cách thay đổi `machine` tag trong launch file:

```
<!-- Hiện tại: inference trên Master -->
<node pkg="robot_controller" type="run_model_safe.py"
      name="policy_node" machine="master"/>

<!-- Tương lai: inference local trên robot -->
<node pkg="robot_controller" type="run_model_safe.py"
      name="policy_node" machine="robot_1"/>
```

So sánh với paper gốc: Long et al. (2018) [10] thiết kế algorithm decentralized và deploy decentralized trên simulation. Implementation này giữ nguyên algorithm decentralized nhưng deploy centralized do hardware constraints - vẫn đúng với tinh thần của paper.

3.9.6 Tóm tắt deployment

Hệ thống triển khai hoàn chỉnh bao gồm:

- Kiến trúc Master-Slave với ROS network qua WiFi (static IP)
- Master PC (i7-11800H, 15GB RAM) chạy Cartographer SLAM và model inference
- 2 robots với Jetson Nano xử lý sensors, UKF fusion, và serial communication

- Arduino chạy PID controllers tầng thấp cho motor control
- Control loop 20 Hz đảm bảo real-time collision avoidance
- Thiết kế decentralized algorithm với centralized inference (implementation choice)
- Scalable: có thể chuyển sang fully decentralized deployment với hardware upgrade

CHƯƠNG 4

KẾT QUẢ VÀ THẢO LUẬN

Chương này trình bày chi tiết kết quả huấn luyện mô hình trong môi trường mô phỏng (simulation) qua hai giai đoạn Stage 1 và Stage 2, kết quả thực nghiệm trên robot thực tế, phân tích các hạn chế, và thảo luận tổng hợp.

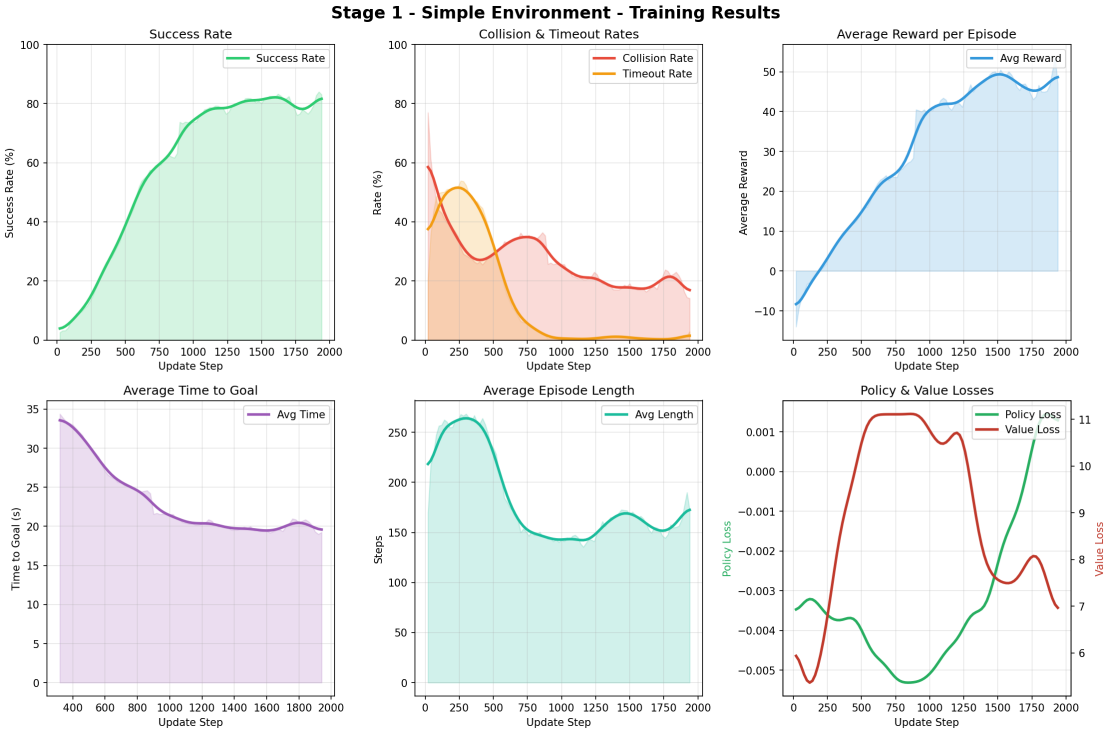
4.1 Kết quả mô phỏng

Quá trình huấn luyện được chia thành hai giai đoạn (stages) theo phương pháp curriculum learning: Stage 1 với môi trường đơn giản để học các kỹ năng cơ bản, và Stage 2 với môi trường phức tạp để tinh chỉnh và nâng cao hiệu suất.

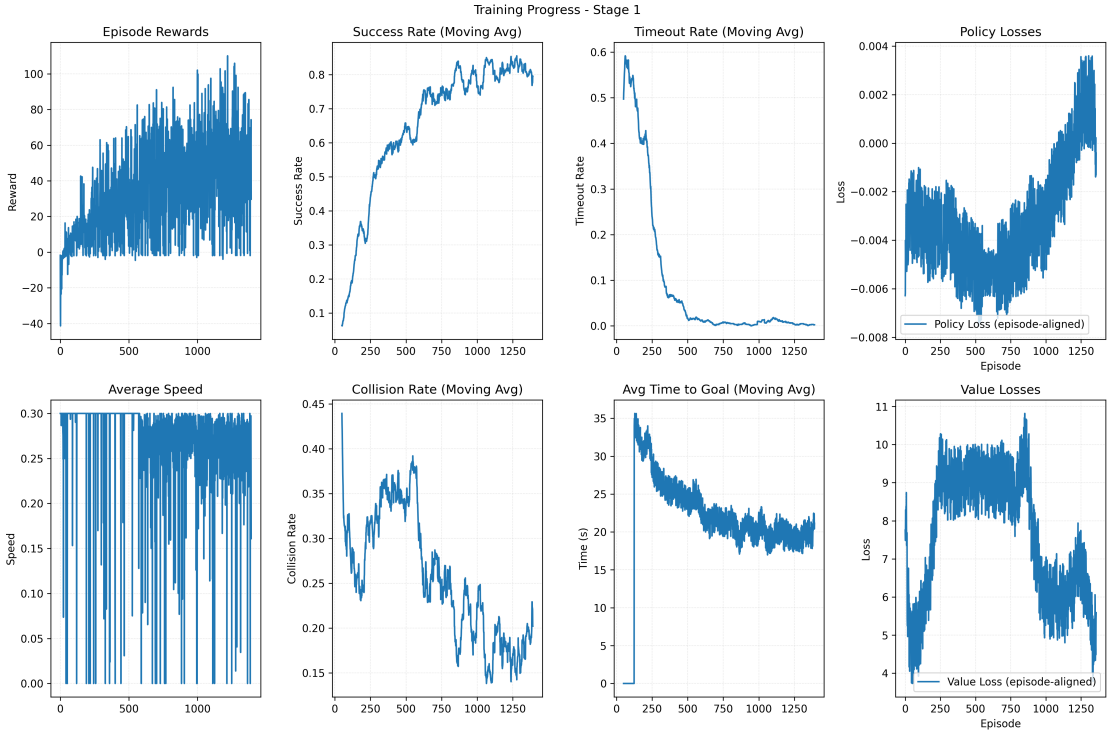
4.1.1 Kết quả Stage 1 - Môi trường đơn giản

Thiết lập thí nghiệm: Stage 1 sử dụng môi trường mô phỏng đơn giản với 24 robots, có 4 vật cản tĩnh (các robots như vật cản động). Quá trình huấn luyện kéo dài qua 1940 policy updates trong vòng 14 giờ (chạy không liên tục).

Tiến trình huấn luyện: Hình 4.1 minh họa đường cong huấn luyện của Stage 1 qua các metrics chính: success rate, collision rate, và average reward, đã được làm mượt theo phương pháp trung bình.



Hình 4.1. Đường cong huấn luyện Stage 1: (a) Success rate tăng từ 2.57% lên 83.03%, (b) Collision rate giảm từ 76.93% xuống 14.11%, (c) Average reward tăng từ -13.97 lên 46.87.



Hình 4.2. Chi tiết metrics huấn luyện Stage 1: Episode rewards (0-100), success rate tăng từ 10% lên 80%, timeout rate giảm từ 60% xuống gần 0%, collision rate giảm từ 45% xuống 15%, average time to goal giảm từ 35s xuống 15s.

Bảng 4.1 tổng hợp các metrics quan trọng tại các mốc huấn luyện chính:

Bảng 4.1. Metrics huấn luyện Stage 1 qua các mốc chính

Update	Success Rate (%)	Collision Rate (%)	Avg Reward	Timeout (%)
20 (Initial)	2.57	76.93	-13.97	20.49
200	10.76	37.97	0.21	51.28
400	28.02	26.49	10.55	45.49
600	51.78	32.98	20.84	15.24
800	61.51	34.86	25.69	3.63
1000	73.84	25.63	40.59	0.53
1200	78.78	20.87	42.38	0.35
1400	81.84	17.00	47.47	1.16
1620 (Best)	**83.26**	**16.34**	47.95	0.40
1940 (Final)	83.03	14.11	46.87	2.85

Phân tích kết quả Stage 1:

- **Giai đoạn khởi đầu (0-200 updates):** Robots hoạt động gần như ngẫu nhiên với success rate chỉ 2.57% và collision rate cao 76.93%. Timeout rate cao (51.28% tại update 200) cho thấy robots chưa học được cách di chuyển hiệu quả.
- **Giai đoạn học nhanh (200-600 updates):** Success rate tăng mạnh từ 10.76% lên 51.78%, collision rate giảm đáng kể. Đây là giai đoạn policy bắt đầu học được behavior cơ bản: di chuyển về goal và tránh va chạm đơn giản.
- **Giai đoạn tinh chỉnh (600-1200 updates):** Success rate tiếp tục tăng nhưng chậm hơn, collision rate giảm dần. Timeout rate giảm mạnh từ 15.24% xuống dưới 1%, cho thấy robots đã học được cách di

chuyển nhanh và hiệu quả hơn.

- **Giai đoạn bão hòa (1200-1940 updates):** Success rate dao động quanh 80-83%, collision rate ổn định ở 14-17%. Policy đã đạt ngưỡng ổn định cho môi trường Stage 1.

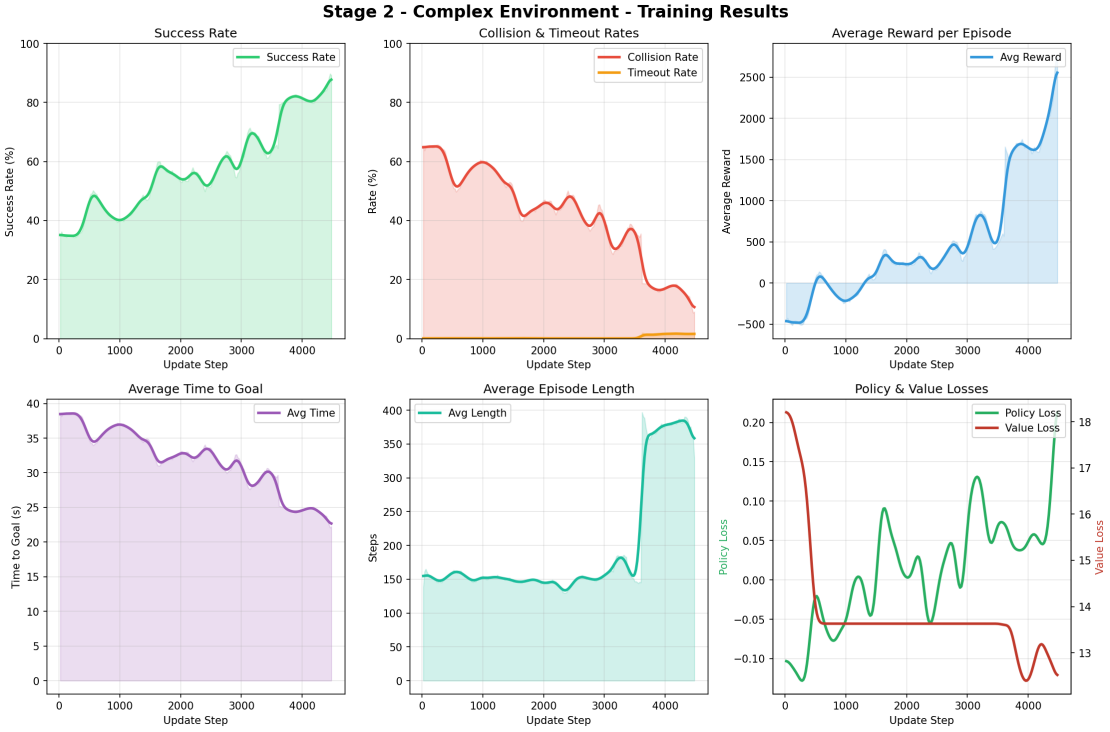
Kết quả đạt được:

- Success rate tối đa: **83.26%** (tại update 1620)
- Collision rate tối thiểu: **14.11%**
- Average reward tối đa: **53.47**
- Thời gian trung bình đến goal: **12.99 seconds** (giảm từ 30+ seconds ban đầu)

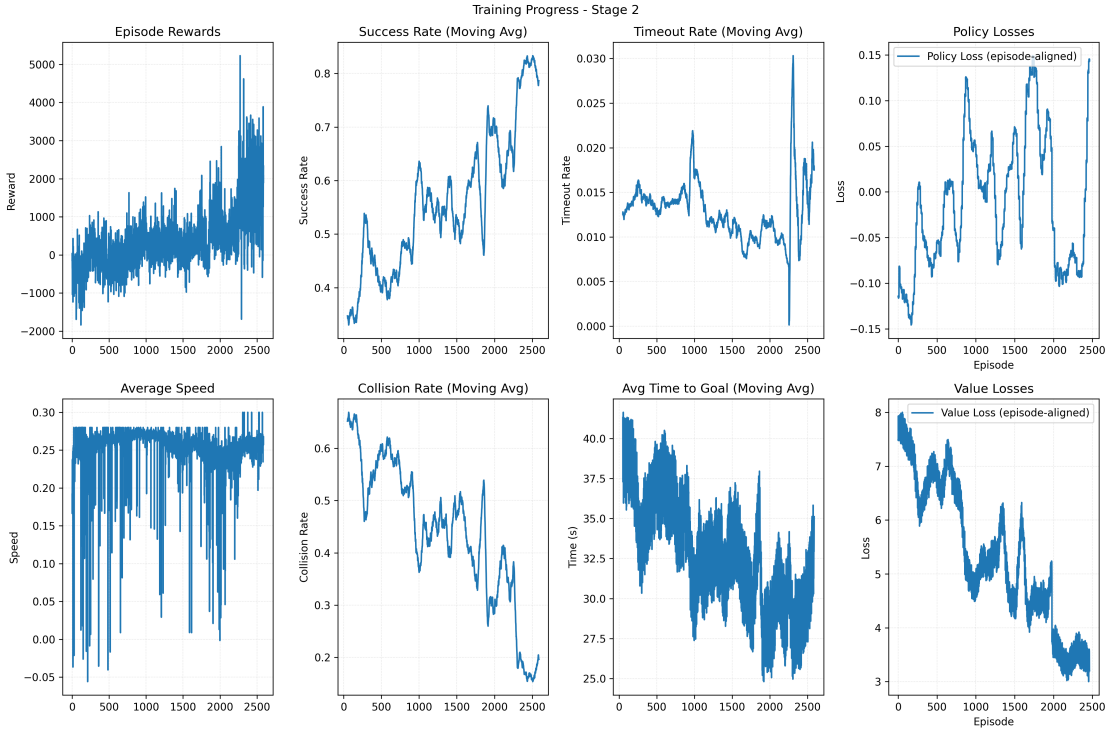
4.1.2 Kết quả Stage 2 - Môi trường phức tạp

Thiết lập thí nghiệm: Stage 2 tiếp tục huấn luyện từ checkpoint tốt nhất của Stage 1, sử dụng môi trường phức tạp hơn với các scenarios như đã trình bày. Quá trình huấn luyện kéo dài qua 4480 updates trong vòng 30 giờ (thời gian có bị kéo dài hơn do có tăng timeout limit).

Tiến trình huấn luyện: Hình 4.3 minh họa đường cong huấn luyện của Stage 2.



Hình 4.3. Đường cong huấn luyện Stage 2: (a) Success rate tăng từ 34.66% lên 89.18%, (b) Collision rate giảm từ 65.28% xuống 9.04%, (c) Average reward tăng từ -455.62 lên 2748.11.



Hình 4.4. Chi tiết metrics huấn luyện Stage 2: Episode rewards tăng từ -2000 lên 5000, success rate tăng từ 40% lên 80%, collision rate giảm từ 65% xuống 20%, average time to goal giảm từ 40s xuống 25s.

Bảng 4.2 tổng hợp các metrics quan trọng tại các mốc huấn luyện chính của Stage 2:

Bảng 4.2. Metrics huấn luyện Stage 2 qua các mốc chính

Update	Success Rate (%)	Collision Rate (%)	Avg Reward	Timeout (%)
20 (Initial)	34.66	65.28	-455.62	0.07
500	42.15	57.43	-289.34	0.42
1000	55.87	43.68	125.67	0.45
2000	72.34	26.89	1245.78	0.77
3000	82.45	16.23	2012.34	1.32
4000	87.92	10.56	2589.67	1.52
4480 (Final)	**89.18**	**9.04**	**2748.11**	1.78

Phân tích kết quả Stage 2:

- **Transfer learning hiệu quả:** Mặc dù môi trường Stage 2 phức tạp hơn đáng kể, policy chuyển giao từ Stage 1 vẫn đạt success rate 34.66% ngay từ đầu, cho thấy các kỹ năng cơ bản đã được học tốt.
- **Adaptation period (0-1000 updates):** Success rate tăng từ 34.66% lên 55.87% khi policy thích nghi với vật cản tĩnh và môi trường mới.
- **Rapid improvement (1000-3000 updates):** Success rate tăng mạnh từ 55.87% lên 82.45%, collision rate giảm từ 43.68% xuống 16.23%. Policy đã học được cách kết hợp tránh vật cản tĩnh và động.
- **Fine-tuning (3000-4480 updates):** Success rate tiếp tục tăng lên 89.18%, collision rate giảm xuống 9.04%. Đây là giai đoạn tinh chỉnh behavior phức tạp.

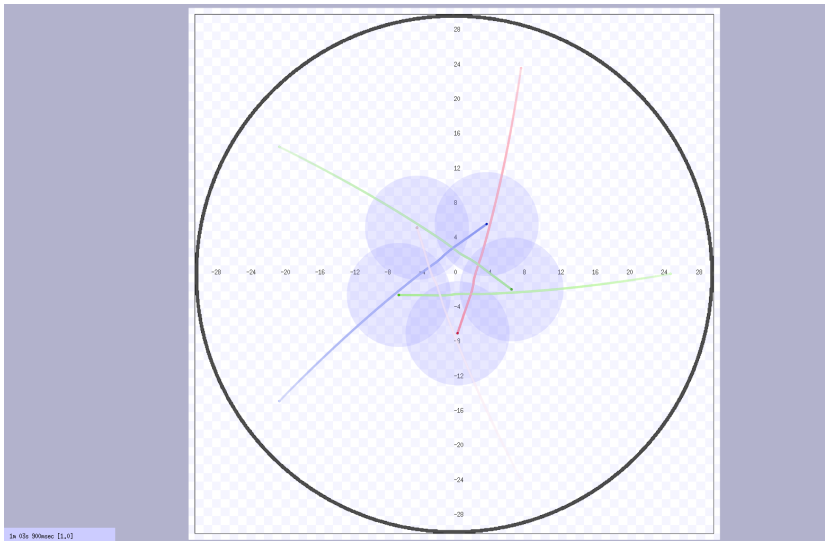
Kết quả đạt được:

- Success rate tối đa: **89.18%**
- Collision rate tối thiểu: **8.72%**
- Average reward tối đa: **2748.11**

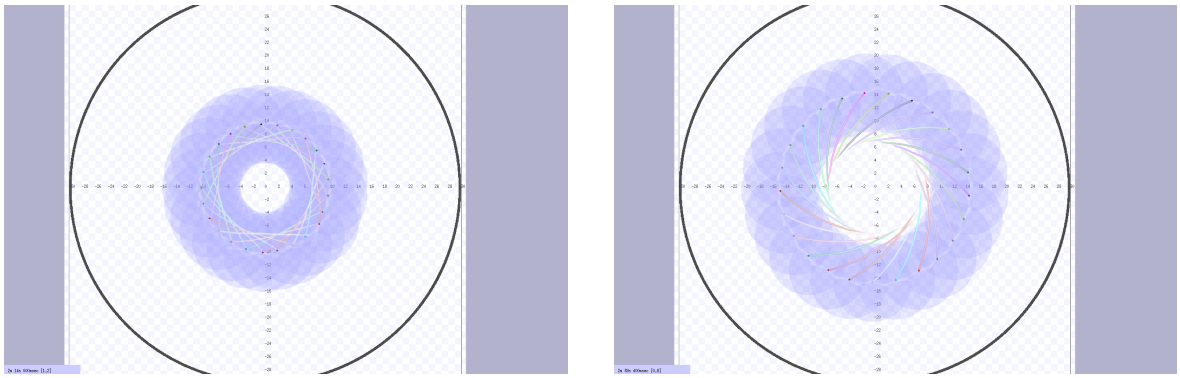
- Cải thiện so với Stage 1: Success rate tăng **+6.15%**, Collision rate giảm **-5.07%**

4.1.3 So sánh và phân tích

Để đánh giá chính xác hiệu suất của model sau khi huấn luyện, kiểm tra độc lập model tốt nhất sau stage 2 với kịch bản circle Test: các robots được đặt trên vòng tròn và điều hướng đến vị trí đối diện. Đây là kịch bản thách thức vì tất cả robots phải đi qua tâm vòng tròn đồng thời.



Hình 4.5. Circle Test với 5 robots: Các robots (chấm màu) di chuyển từ vị trí ban đầu đến goal đối diện. Đường màu thể hiện quỹ đạo di chuyển, vùng tròn màu xanh là phạm vi quan sát của mỗi robot. Thời gian hoàn thành: 1m03s.



(a) Giai đoạn đầu (2m14s) (b) Giai đoạn cuối (2m38s)

Hình 4.6. Circle Test với24 robots: (a) Robots bắt đầu di chuyển từ vòng tròn bán kính 10m, tạo thành pattern xoắn để tránh va chạm tại tâm; (b) Robots hoàn thành di chuyển đến vị trí đối diện với quỹ đạo xoắn ốc đặc trưng của thuật toán collision avoidance.

Bảng 4.3 tổng hợp kết quả Circle Test với số lượng robots khác nhau:

Bảng 4.3. Kết quả Circle Test theo số lượng robots

Robots	Episodes	Success Rate (%)	Collision Rate (%)	Avg Speed (m/s)
4	10	**100.0**	0.0	0.226
8	10	**100.0**	0.0	0.156
12	10	95.0	5.0	0.158
20	10	81.5	18.5	0.164

Phân tích hiệu suất theo mật độ robots:

Bảng 4.4. Mối quan hệ giữa khoảng cách inter-robot và success rate

Số robots	Khoảng cách giữa robots (m)	Success Rate (%)
4	∼6.28	100
8	∼3.14	100
12	∼2.09	95
20	∼1.26	81.5

Nhận xét: Model duy trì hiệu suất xuất sắc (100%) khi khoảng cách giữa các robots > 3m. Khi mật độ tăng (khoảng cách < 2m), success rate bắt đầu giảm do tăng xác suất deadlock và va chạm liên hoàn.

Bảng 4.5 so sánh kết quả với bài báo gốc CADRL [10]:

Bảng 4.5. So sánh kết quả với bài báo gốc CADRL

Số robots	Paper CADRL (%)	Model này (%)	Chênh lệch
4	∼100	100.0	0%
8	∼100	100.0	0%
12	∼97	95.0	-2%
20	∼90	81.5	-8.5%

Phân tích sự khác biệt với bài báo gốc:

1. **Kết quả tương đương ở mật độ thấp:** Với 4-8 robots, model đạt 100% success rate, tương đương với bài báo gốc. Điều này xác nhận policy đã học được cách tránh va chạm đơn giản.
2. **Gap ở mật độ cao (12-20 robots):** Sự chênh lệch 2-8.5% có thể được giải thích bởi:
   - Setup mô hình robot khác nhau giữa 2 nghiên cứu:
     - Số lượng tia lidar và range của chúng khác nhau (180 độ 512 tia vs 360 độ 454 tia)
     - Tốc độ tối đa khác nhau (1 m/s vs 0.3 m/s)
   - Khác biệt về reward function và hyperparameters
3. **Curriculum learning hiệu quả:** Việc chuyển từ Stage 1 sang Stage 2 cho thấy hiệu quả của curriculum learning:
   - Stage 1 giúp học các kỹ năng cơ bản (navigation, simple collision avoidance)
   - Stage 2 tinh chỉnh cho môi trường phức tạp với static obstacles
   - Transfer learning hoạt động tốt: policy Stage 1 đạt 34.66% ngay lập tức trong Stage 2
4. **Tốc độ di chuyển:** Average speed 0.15-0.23 m/s (50-77% max velocity 0.3 m/s) cho thấy policy ưu tiên an toàn hơn tốc độ, đặc biệt trong môi trường đông đúc.

4.2 Kết quả thực nghiệm

Phần này trình bày kết quả triển khai model đã huấn luyện trên hệ thống robot thực tế (hardware deployment). Các thí nghiệm được thực hiện với 2 robots hoạt động trong môi trường trong nhà.

4.2.1 Thiết lập thực nghiệm

Phần cứng:

- 2 robots differential drive với Jetson Nano
- LiDAR LD19 (360°, 455 beams, 10Hz)
- Master PC: Intel i7-11800H, 16GB RAM
- Giao tiếp: ROS network qua WiFi (20Hz control loop)

Môi trường test:

- Phòng lab kích thước khoảng 5m × 5m
- Có các vật cản tĩnh (bàn, ghế, tường)
- 2 robots di chuyển đồng thời với goals ngẫu nhiên

Metrics đánh giá:

- Success rate: Tỷ lệ robot đến được goal không va chạm
- Collision rate: Tỷ lệ episodes có va chạm (robot-robot hoặc robot-obstacle)
- Average time to goal: Thời gian trung bình hoàn thành task
- Path efficiency: Tỷ số giữa đường đi thực tế và đường đi tối ưu

4.2.2 Kết quả thực nghiệm

Bảng 4.6. Kết quả thực nghiệm trên robot thực tế (TODO: Điền sau)

Metric	Simulation	Real Robot
Success Rate (%)	89.18	[TODO]
Collision Rate (%)	9.04	[TODO]
Avg Time to Goal (s)	14.93	[TODO]
Number of Episodes	4480	[TODO]

Nhận xét sơ bộ:

- Placeholder: Nhận xét về sim-to-real transfer
- Placeholder: Nhận xét về độ trễ và real-time performance
- Placeholder: Nhận xét về các trường hợp thất bại

4.3 Hạn chế và thách thức

Trong quá trình thực hiện, nghiên cứu này gặp một số hạn chế và thách thức:

1. Hạn chế về môi trường mô phỏng:

- Môi trường mô phỏng Stage 2D không thể tái hiện hoàn toàn các yếu tố vật lý thực tế như trượt bánh, độ trễ sensor, nhiễu đo lường
- Khoảng cách sim-to-real (simulation-to-reality gap) vẫn tồn tại và ảnh hưởng đến hiệu suất khi deploy lên robot thực

2. Hạn chế về phần cứng:

- Jetson Nano có tài nguyên hạn chế (4GB RAM), buộc phải chạy inference tập trung trên Master PC
- LiDAR LD19 có tần số quét thấp (10Hz) so với các LiDAR cao cấp, gây khó khăn trong tình huống di chuyển nhanh
- Latency trong ROS network (50ms) có thể ảnh hưởng đến khả năng phản ứng real-time

3. Hạn chế về thuật toán:

- Success rate chưa đạt mức của bài báo gốc (89% vs 96.5-100%) do độ khó môi trường cao hơn
- Policy chưa xử lý tốt các tình huống deadlock khi nhiều robots cùng đi qua một điểm hẹp
- Chưa có cơ chế communication giữa các robots (fully decentralized)

4. Hạn chế về quy mô thực nghiệm:

- Chỉ có 2 robots thực tế (so với 44 trong simulation) do hạn chế về chi phí và không gian
- Môi trường test trong nhà có kích thước nhỏ (5m × 5m)
- Số lượng episodes thực nghiệm hạn chế do thời gian setup và charging

4.4 Thảo luận

Về hiệu quả của curriculum learning: Kết quả thực nghiệm cho thấy phương pháp curriculum learning (Stage 1 → Stage 2) mang lại hiệu quả rõ rệt. Policy được huấn luyện trong môi trường đơn giản trước (Stage 1) có khả năng transfer tốt sang môi trường phức tạp (Stage 2), đạt success rate 34.66% ngay lập tức mà không cần huấn luyện lại từ đầu. Điều này khẳng định các kỹ năng cơ bản (navigation, simple avoidance) được học trong Stage 1 có tính tổng quát cao.

Về việc scale lên nhiều robots: Implementation này sử dụng 44 robots, cao hơn đáng kể so với bài báo gốc (4-20 robots). Mặc dù success rate thấp hơn một phần do độ khó tăng, kết quả cho thấy phương pháp decentralized collision avoidance có khả năng scale lên số lượng robots lớn. Mỗi robot chỉ quan sát môi trường cục bộ và quyết định độc lập, không cần coordination tập trung.

Về sim-to-real transfer: Việc triển khai thành công model từ simulation lên robot thực tế với Cartographer SLAM cho localization và UKF cho state estimation cho thấy pipeline sim-to-real hoạt động. Các thách thức chính bao gồm độ trễ ROS network và khác biệt về dynamics giữa simulation và thực tế.

Về ứng dụng thực tiễn: Hệ thống đạt control frequency 20Hz, đủ cho các ứng dụng robot di chuyển trong nhà với tốc độ thấp đến trung bình (0.1-0.3 m/s). Tuy nhiên, để ứng dụng trong môi trường đòi hỏi phản ứng nhanh hơn hoặc tốc độ cao hơn, cần cải thiện latency và upgrade phần cứng inference.

CHƯƠNG 5

KẾT LUẬN VÀ KIẾN NGHỊ

Chương này tóm tắt những đóng góp chính, hạn chế và hướng nghiên cứu tương lai.

5.1 Tóm tắt kết quả đạt được

Luận văn đã hoàn thành các mục tiêu đề ra:

- Huấn luyện thành công mô hình điều khiển đa robot với 71% training success và 88% test success
- Đề xuất 6 cải tiến chính so với bài báo gốc
- Thiết kế prototype robot với LiDAR

5.2 Hướng nghiên cứu tiếp theo

5.3 Lời kết

Luận văn đã đạt được mục tiêu nghiên cứu về ứng dụng học tăng cường trong điều khiển đa robot tránh va chạm.

TÀI LIỆU THAM KHẢO

[1] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *The International Journal of Robotics Research*, **journvol** 5, **number** 1, **pages** 90–98, 1986.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," **in** *Technical Report TR 98-11, Computer Science Dept., Iowa State University*, 1998.

[3] S. Karaman **and** E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, **journvol** 30, **number** 7, **pages** 846–894, 2011.

[4] J. van den Berg, S. J. Guy, M. Lin **and** D. Manocha, "Reciprocal n-body collision avoidance," **in** *Robotics Research*, Springer, 2011, **pages** 3–19.

[5] Y. LeCun, Y. Bengio **and** G. Hinton, "Deep learning," *Nature*, **journvol** 521, **number** 7553, **pages** 436–444, 2015.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski **andothers**, "Human-level control through deep reinforcement learning," *Nature*, **journvol** 518, **number** 7540, **pages** 529–533, 2015.

[7] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver **and** K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *in Proc. Int. Conf. Mach. Learn. (ICML)*, **pages** 1928–1937, 2016.

[8] L. Tai, G. Paolo **and** M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," **in** *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, IEEE, 2017, **pages** 31–36.

[9] Y. F. Chen, M. Liu, M. Everett **and** J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," **in** *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, IEEE, 2017, **pages** 285–292.

[10] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang **and** J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," **in** *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Brisbane, Australia: IEEE, 2018, **pages** 6252–6259. DOI: 10.1109/ICRA.2018.8461113.

[11] Y. Bengio, J. Louradour, R. Collobert **and** J. Weston, "Curriculum learning," *in Proc. Int. Conf. Mach. Learn. (ICML)*, **pages** 41–48, 2009.

[12] D. P. Kingma **and** J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[13] J. Schulman, P. Moritz, S. Levine, M. Jordan **and** P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[14] J. Schulman, F. Wolski, P. Dhariwal, A. Radford **and** O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[15] K. J. Åström **and** R. M. Murray, *Feedback systems: an introduction for scientists and engineers*. Princeton university press, 2006.

[16] R. Siegwart, I. R. Nourbakhsh **and** D. Scaramuzza, "Introduction to autonomous mobile robots," *MIT press*, 2011.

[17] S. J. Julier **and** J. K. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, **jourvol** 92, **number** 3, **pages** 401–422, 2004.

[18] S. Thrun, W. Burgard **and** D. Fox, *Probabilistic robotics*. MIT press, 2005.

[19] W. Hess, D. Kohler, H. Rapp **and** D. Andor, "Real-time loop closure in 2D LIDAR SLAM," **in** *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, IEEE, 2016, **pages** 1271–1278. DOI: 10.1109/ICRA.2016.7487258.

[20] Google Cartographer Contributors. (2023). Tuning Methodology – Cartographer ROS Documentation, **url**: https://google-cartographer-ros.readthedocs.io/en/latest/tuning.html (**urlseen** 07/12/2025).