TRƯỜNG ĐẠI HỌC BÁCH KHOA TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

BỘ MÔN ĐIỀU KHIỂN TỰ ĐỘNG

ĐỀ CƯƠNG LUẬN VĂN THẠC SĨ

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

GVHD: TS. PHẠM VIỆT CƯỜNG

HVTH: Nguyễn Tấn Khôi

MSHV: 2171017

TP. HỒ CHÍ MINH, 2023

LỜI CẢM ƠN

Trong thời gian làm đề cương luận văn, em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến và chỉ bảo nhiệt tình của thầy cô, gia đình và bạn bè.

Em xin gửi lời cảm ơn chân thành đến thầy Phạm Việt Cường - giảng viên Bộ môn Tự động hoá trường đại học Bách Khoa TP. HCM - đã tận tình hướng dẫn em trong suốt quá trình làm đề cương luận văn. Cũng nhờ việc học môn Trí tuệ nhân tạo trong điều khiển của thầy mà em có được ý tưởng về đề tài này.

Em cũng xin chân thành cảm ơn các thầy cô giáo trong trường đại học Bách Khoa TP. HCM đã dạy dỗ cho em kiến thức về các môn đại cương cũng như các môn chuyên ngành, giúp em có được cơ sở lý thuyết vững vàng và tạo điều kiện giúp đỡ em trong suốt quá trình học tập.

Cuối cùng, em xin chân thành cảm ơn gia đình và bạn bè, đã luôn tạo điều kiện, quan tâm, giúp đỡ, động viên em trong suốt quá trình học tập và hoàn thành đề cương luận văn.

Với điều kiện thời gian cũng như kinh nghiệm của một học viên, đề cương luận văn này không thể tránh được những thiếu sót. Em rất mong nhận được sự chỉ bảo, đóng góp ý kiến của các thầy cô để em có điều kiện bổ sung, nâng cao kiến thức của mình tạo bước đệm cho việc hoàn thành Luận văn đạt được mục tiêu đề ra.

Tp. Hồ Chí Minh, ngày 3 tháng 12 năm 2023 .

Học viên

MỤC LỤC

1. GIỚI THIỆU

1.1 Tổng quan

Trong bối cảnh ngày nay, việc phát triển một hệ thống đa robot an toàn và hiệu quả đang phải đối mặt với các khó khăn nhất định. Với hệ thống đa robot như các robot trong nhà kho, thông thường đều sử dụng phương pháp tập trung nơi mà có một hệ thống trung tâm điều khiển tất cả robot, tuy nhiên để tăng được sự linh hoạt, khả năng mở rộng (scalable) và tối đa được hiệu quả tính toán thì hệ tập trung cần được thay thế bằng một hệ thống điều khiển phân tán với mỗi robot đều có năng lực xử lý độc lập. Có rất nhiều phương pháp điều khiển phân tán ra đời để đáp ứng nhu cầu trên, tuy nhiên bài toán này vẫn là một bài toán khó và các phương pháp nghiên cứu được ra đời trước đây đa số đều có hiệu quả kém hơn so với phương pháp tập trung. Những phương pháp được ra đời sau này có lợi thế hơn trước, nhờ vào sự phát triển mạnh mẽ trong các thuật toán học tăng cường tạo tiền đề để ứng dụng nó trong các phương pháp điều mà nhờ đó mà việc đều khiển phi tập trung cũng trở nên ổn định hiệu quả và có tính linh hoạt cao hơn

Dựa trên các nghiên cứu đã được thực hiện trước đây, đề cương lần này chú trọng đến việc phân tích và cách ứng dụng thực tế của một bài nghiên cứu sử dụng deep reinforcement learning để giải quyết bài toán điều khiển phi tập trung hệ đa robot tránh va chạm. Khác với các phương pháp được đề xuất trước đây, phương pháp được đề cập tới sẽ sử dụng các đặc trưng ở cấp độ cảm biến, dữ liệu từ các cảm biến như laser sẽ được ánh xạ sang thành các lệnh điều khiển cho mỗi đối tượng robot. Bên cạnh đó ở phương pháp này còn đề xuất phương án huấn luyện đa dạng môi trường và qua nhiều giai đoạn cho việc học chính sách tối ưu, điều này góp phần làm tăng hiệu năng của kết quả thu được và có thể so sánh được với các phương pháp điều khiển tập trung trước đây. Với việc nghiên cứu và điều chỉnh phù hợp ở đề cương lần này sẽ tạo điều kiện thuận lợi cho việc ứng dụng phương pháp này vào trong mô hình robot thực tế ở luận văn.

1.2 Mục tiêu đề tài

Đề cương luận văn này được thực hiện với những nhiệm vụ chính sau:

- Tìm hiểu các lý thuyết liên quan đến việc huấn luyện và thuật toán bao gồm: Reinforcement learning, neutral network, optimizer và thuật toán Proximal Policy Optimization (PPO)
- Đọc hiểu được phương pháp của bài báo được đề cập
- Tiến hành thực hiện và chuẩn bị cho phần huấn luyện theo các phương án đề xuất của bài báo. Kết quả phần training sẽ được thu thập và đánh giá ở luận văn
- Lên kế hoạch và thiết kế mô hình robot thực tế để áp dụng phương pháp đã nghiên cứu trong hệ thống thực tế

2. LÝ THUYẾT

2.1 Neutral network

2.1.1 Khái niệm

Neural Network là một trong những thuật toán Machine Learning mạnh mẽ nhất. Nó cũng là nền tảng của Deep Learning và là một khía cạnh quan trọng của Trí Tuệ Nhân Tạo (AI) và Máy Học (ML). Các tế bào thần kinh, hay còn được gọi là neuron, đóng vai trò quan trọng trong quá trình dẫn truyền xung động thần kinh. Mỗi neuron nhận xung động thần kinh từ neuron ở vị trí trước đó, truyền nó dọc theo axon và chuyển giao cho neuron ở vị trí sau đó thông qua các kết nối synapse. Sự truyền trực tiếp xung động thần kinh giữa các neuron trong mạng tạo nên cơ sở cho mọi chức năng thần kinh của bộ não. Dựa trên ý tưởng đó Neutral network sẽ mô phỏng cách não bộ con người hoạt động để xử lý thông tin và thực hiện các nhiệm vụ phức tạp.

2.1.2 Cấu Trúc Cơ Bản của Mạng Nơ-ron

Mạng nơ-ron là một mô hình tính toán có khả năng học từ dữ liệu và điều chỉnh chúng để thực hiện các nhiệm vụ cụ thể. Cấu trúc mạng nơ-ron gồm có các nơ-ron và liên kết giữa chúng. Trọng số và độ lệch được điều chỉnh trong quá trình huấn luyện để tối ưu hiệu suất của mô hình.

- Mô hình toán

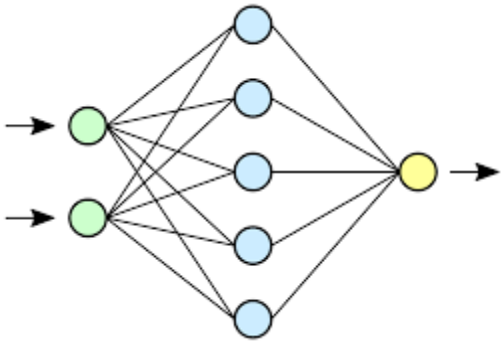


Figure 1: Mạng NN cơ bản, gồm 3 lớp và 8 nút

Lớp đầu tiên trong kiến trúc mạng nơ-ron là lớp input, chịu trách nhiệm nhận dữ liệu đầu vào của bài toán. Lớp ẩn, đặt ở giữa, tiếp nhận giá trị đầu ra từ lớp trước đó để thực hiện tính toán với mức độ phức tạp cao hơn trước khi chuyển gửi tới lớp tiếp theo. Lớp cuối cùng, là lớp output, đưa ra kết quả cuối cùng của mô hình.

Mỗi nút, trừ những nút thuộc lớp input, hoạt động tương tự như một thuật toán Logistic Regression. Nút này nhận vector đầu vào từ các nút ở lớp trước đó và xuất ra một phần tử của vector đó để truyền tới các nút thuộc lớp kế tiếp. Điều này tạo ra quá trình lan truyền thông tin qua các lớp và giúp mô hình học được biểu diễn phức tạp của dữ liệu.

- Hàm Kích Hoạt (Activation Function)

Mỗi nơ-ron trong mạng có một hàm kích hoạt riêng, thông thường là ReLU hoặc Sigmoid non-linear. Hàm này xác định cách nơ-ron phản ứng với đầu vào của nó và truyền giá trị ra cho các nơ-ron tiếp theo.

Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$

Tanh: $\tanh(x) = \frac{e^{x}e^{-x}}{e^{x}+e^{-x}}$

ReLU: $f(x) = \max(0, x)$

2.1.3 Huấn luyện và học

Quá trình huấn luyện mạng nơ-ron thường dựa trên thuật toán lan truyền ngược (Backpropagation). Trong quá trình này, mô hình học từ dữ liệu thông qua việc điều chỉnh các trọng số và độ lệch để giảm sai số giữa giá trị dự đoán và giá trị thực tế.

Thuật toán lan truyền ngược là quá trình tính toán và điều chỉnh độ chệch (error) của mô hình dự đoán so với giá trị thực tế. Quá trình này bắt đầu từ lớp output và lan truyền ngược về lớp input, cập nhật trọng số dựa trên độ chệch tính toán được.

Trong quá trình đó ta cần định nghĩa hàm mất mát (loss function). Hàm mất mát đánh giá sự chênh lệch giữa giá trị dự đoán của mô hình và giá trị thực tế. Một trong những hàm mất mát phổ biến là Mean Squared Error (MSE) cho các bài toán hồi quy và Cross-Entropy cho các bài toán phân loại.

MSE: $\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2$

Cross entropy: $H(p,q) = -\sum_i p_i \log q_i$

Sau khi tính toán độ chệch thông qua hàm mất mát, thuật toán gradient descent được sử dụng để điều chỉnh trọng số và độ lệch (thuật toán tối ưu). Gradient descent giúp mô hình học cách giảm thiểu độ chệch bằng cách di chuyển ngược dọc theo đạo hàm của hàm mất mát.

$\theta = \theta - \alpha\nabla J(\theta)$ [1]

Trong đó, θ là tập hợp các trọng số, α là hệ số học (learning rate), và $\nabla J(\theta)$ là đạo hàm riêng của hàm mất mát theo θ.

Error Backpropagation kết hợp lan truyền ngược, hàm mất mát, và gradient descent. Quá trình này được thực hiện qua nhiều vòng lặp (epoch) để mô hình học được biểu diễn phức tạp của dữ liệu. Khi mô hình đã học đủ, nó có khả năng đưa ra dự đoán chính xác trên dữ liệu mới.

2.1.4 So sánh với các thuật toán phân loại

3 thuật toán phân loại Logistic Regression, Support Vector Machine và Neural Network là các thuận toán cơ bản và phổ biến hiện nay và chúng có điểm mạnh và điểm yếu riêng.

- Regression: Logistic Regression thường được sử dụng cho các bài toán phân loại hai lớp. Điều này là một thuật toán tuyến tính, và nó phù hợp khi dữ liệu có mối quan hệ tuyến tính với biến phụ thuộc
- Support Vector Machine (SVM): SVM là một thuật toán phân loại mạnh mẽ và linh hoạt. Nó thường được sử dụng cho các bài toán phân loại và hồi quy
- Neutral network: Mạng nơ-ron thường được ưa chuộng trong các bài toán có độ phức tạp cao và dữ liệu lớn

2.1.5 Ứng dụng

- Nhận Diện Hình Ảnh và Video
- Dự Đoán Chuỗi Thời Gian
- Ngôn Ngữ Tự Nhiên và Dịch Máy
- Điều Khiển và Robot

2.2 Reinforcement Learning

Các thuật toán học máy thường được chia thành ba loại chính: học có giám sát, học không giám sát và học tăng cường. Trong học có giám sát, một mô hình học từ tập dữ liệu được gắn nhãn để suy ra mối quan hệ giữa đầu vào và đầu ra. Ngược lại, học không giám sát không sử dụng dữ liệu được dán nhãn mà chỉ sử dụng dữ liệu và cố gắng mô tả cấu trúc của nó.

Học tăng cường là loại thứ ba và tập trung vào cách một tác nhân thực hiện các hành động trong môi trường để đạt được phần thưởng tối đa. Không giống như học có giám sát, học tăng cường không sử dụng các cặp dữ liệu đầu vào và đầu ra được gắn nhãn trước và không đánh giá tính đúng đắn của các hành động.

Các khái niệm trong Reinforcement learning

- Agent: Thực thể thực hiện hành động trong môi trường.
- Environment: Nơi mà agent tương tác và học.
- Action: agent thực hiện hành động thay đổi môi trường dựa trên state
- Observation: quan sát sự thay đổi của môi trường tại state
- State: một trạng thái của môi trường
- Policy: chính sách quyết định cách mà một tác tử thực hiện các hành động trong môi trường tại một thời điểm nhất định. Nó xác định cách tác tử phản ứng và đưa ra quyết định dựa trên trạng thái hiện tại của môi trường. Có thể được hiểu đơn giản như một chiến lược trong quá trình tương tác với môi trường, giúp định hình hành vi của agent để đạt được mục tiêu hay phần thưởng tối đa.
- Reward: phần thưởng nhận được khi thực hiện một action. Ở mỗi hành động, môi trường trả về reward, mục tiêu là làm sao tối đa hóa tổng reward trong khoảng thời gian. Reward giúp xác định sự kiện có lợi và bất lợi đối với agent, cũng là cơ sở để điều chỉnh chính sách. Nếu một hành động mang lại phần thưởng thấp, chính sách có thể thay đổi, và agent sẽ lựa chọn các hành động khác trong tình huống tương tự ở tương lai.

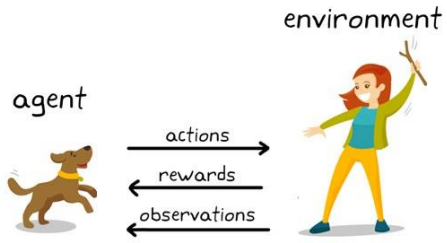


Figure 2. Minh hoạ cho tương tác agent và environment

Có thể kể đến một số thuật toán RL cơ bản như :

- Q-Learning:

  Q-Learning là một thuật toán off-policy, giúp tác tử học được hàm giá trị hành động tối ưu (Q-value).

  Nó sử dụng bảng giá trị để lưu trữ và cập nhật giá trị của mỗi cặp trạng thái-hành động.

- Policy Gradient Methods:

  Các phương pháp Policy Gradient tập trung vào việc trực tiếp tối ưu hóa chính sách thay vì học hàm giá trị.

  Thuật toán như REINFORCE sử dụng gradient của hàm mất mát để cập nhật trọng số của chính sách.

- Actor-Critic:

  Actor-Critic là sự kết hợp giữa phương pháp giảm biến (variance reduction) và phương pháp giảm giá trị (value reduction).

  Một mô hình "Actor" quyết định hành động, trong khi một mô hình "Critic" đánh giá giá trị của hành động đó.

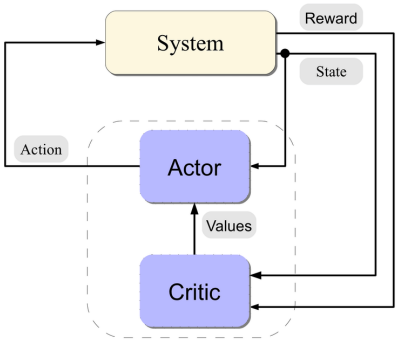


Figure 3. Sơ đồ mạng actor-critic

- Proximal Policy Optimization (PPO):

  PPO cũng là thuật toán dựa trên phương pháp actor-critic, nó là một thuật toán giảm biến dựa trên chính sách, tối ưu hóa chính sách dựa trên các mẫu dữ liệu hiện tại

  Nó giảm biến động giữa các cập nhật chính sách để đảm bảo tính ổn định.

2.3 Các thuật toán optimizer

2.3.1 Gradient Descent

Ý tưởng cơ bản của Gradient Descent là đi dọc theo đạo hàm của hàm mất mát để tìm ra hướng giảm nhanh nhất. Để thực hiện điều này, thuật toán tính toán đạo hàm (gradient) của hàm mất mát theo từng tham số, sau đó di chuyển theo hướng âm của gradient với một bước cập nhật được xác định bởi learning rate.

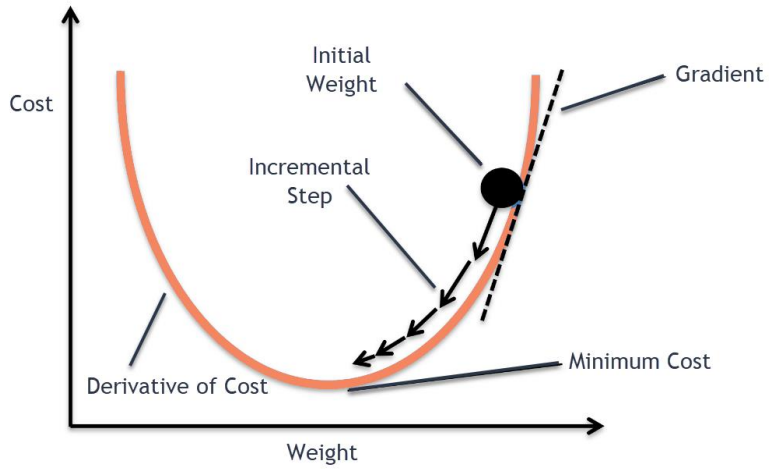


Figure 4. Mô phỏng thuật toán Gradient Descent

Công thức cập nhật của Gradient Descent có thể biểu diễn như sau:

$$x_{t+1} = x_t - \eta * \frac{\partial L}{\partial x}$$

x: tham số mô hình

η: learning rate

$\frac{\partial L}{\partial x}$: gradient của hàm mất mát

- Ưu Điểm:

+ Gradient Descent là phương pháp đơn giản và dễ triển khai, phổ biến trong nhiều loại mô hình máy học.

+ Đồng thời nó phù hợp cho nhiều loại hàm mất mát không phụ thuộc vào tính chất của hàm mất mát.

- Nhược Điểm:

+ Sự chọn lựa của learning rate đôi khi có thể khó khăn và ảnh hưởng đến tốc độ và ổn định của quá trình học.

+ Có thể dễ rơi vào điểm tối ưu cục bộ thay vì điểm tối ưu toàn cục, tùy thuộc vào điều kiện khởi tạo và đặc tính của hàm mất mát.

+ Trong trường hợp dữ liệu lớn, Gradient Descent có thể trở nên chậm do cần tính toán đồng thời trên toàn bộ tập dữ liệu.

2.3.2 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) là một biến thể quan trọng của Gradient Descent được sử dụng rộng rãi trong quá trình huấn luyện mô hình máy học. Thuật toán này đặt trọng điểm vào việc sử dụng chỉ một điểm dữ liệu ngẫu nhiên tại mỗi bước, giảm sự phụ thuộc vào toàn bộ tập dữ liệu và tăng tốc quá trình học.

Mỗi lượt duyệt qua toàn bộ dữ liệu được gọi là một epoch trong quá trình huấn luyện mô hình. Trong Gradient Descent thông thường, mỗi epoch tương đương với một lần cập nhật θ, trong khi với Stochastic Gradient Descent (SGD), mỗi epoch đều ứng với N lần cập nhật θ, với N là số điểm dữ liệu. Mặc dù việc cập nhật từng điểm một có thể làm giảm tốc độ thực hiện một epoch, nhưng SGD lại có ưu điểm là chỉ yêu cầu một số lượng epoch nhỏ. Do đó, SGD thích hợp cho các bài toán với lượng cơ sở dữ liệu lớn, đặc biệt là trong Deep Learning và cho những bài toán yêu cầu mô hình thay đổi liên tục, tức là online learning.

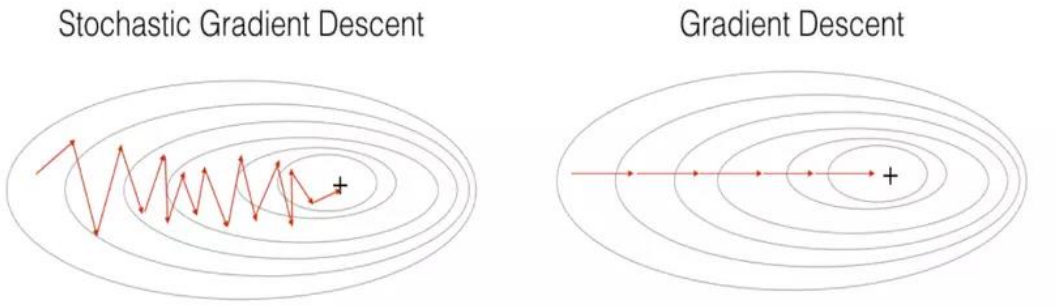


Figure 5. stochastic gradient descent vs gradient descent

2.3.3 GD với Momentum

Gradient Descent với Momentum là một biến thể của thuật toán Gradient Descent, trong đó được thêm vào một yếu tố động (momentum) để cải thiện hiệu suất của quá trình cập nhật trọng số. Nguyên lý hoạt động của thuật toán này bao gồm việc duy trì một vector velocity trước đó (v), biểu diễn hướng và độ lớn của bước cập nhật trước đó. Mỗi lần cập nhật, vector velocity

được điều chỉnh dựa trên gradient của hàm mất mát, và sau đó được sử dụng để cập nhật trọng số.

Vector velocity được cập nhật như sau:

$v_{t+1} = \beta + (1 - \beta)\nabla L(\theta_t)$

v: vector velocity

β hệ số momentum

$\nabla L(\theta_t)$: đạo hàm hàm mất mát tại θ

Công thức cập nhật trọng số: $\theta_{t+1} = \theta - \eta v_{t+1}$

Ưu điểm chính của Gradient Descent với Momentum bao gồm khả năng tăng tốc quá trình hội tụ và làm giảm dao động trong quá trình huấn luyện, đặc biệt là trên các bề mặt mất mát có hình dạng phức tạp. Nó cũng giúp ổn định hóa quá trình học, làm giảm ảnh hưởng của nhiễu và dao động. Và đặc biệt có thể giải quyết được vấn đề kẹt ở cực trị cục bộ tiến tới cực trị toàn cục

Sự hiệu quả của thuật toán phụ thuộc nhiều vào việc chọn lựa hệ số momentum (β), và việc điều chỉnh nó có thể không trực tiếp. Ngoài ra việc đạt cực trị cũng tốn nhiều thời gian hơn do có momentum ở quanh cực trị nên mất thời gian để hội tụ.

2.3.4 RMSProp

RMSprop (Root Mean Square Propagation) là một thuật toán tối ưu hóa sử dụng trong quá trình huấn luyện mô hình máy học. Nó giữ vai trò quan trọng trong việc ổn định quá trình học và điều chỉnh tỷ lệ học (learning rate) tự động.

Để điều chỉnh tỉ lệ học RMSProp sẽ tính toán giá trị của trung bình bình phương của các gradient trước đó, sau đó cập nhật trọng số bằng việc chia hệ số học (learning rate) cho đại lượng trung bình bình phương vừa tính

Công thức như sau:

$$v_{t+1} = \beta + (1 - \beta)\nabla L(\theta_t)^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{v_{t+1}} + \epsilon}\nabla L(\theta_t)$$

RMSprop tự động điều chỉnh tỷ lệ học cho phù hợp với độ lớn của gradient, giúp tối ưu hóa quá trình học. Phù hợp cho nhiều loại mô hình và bài toán học máy, đặc biệt là trong Deep Learning.

RMSprop có thể cho kết quả nghiệm chỉ là local minimum chứ không đạt được global minimum như Momentum. Vì vậy người ta sẽ kết hợp cả 2 thuật toán Momentum với RMSprop cho ra 1 thuật toán tối ưu Adam

2.3.5 Adam Optimizer

Adam (Adaptive Moment Estimation) là một thuật toán tối ưu hóa hiệu quả và phổ biến được sử dụng trong quá trình huấn luyện mô hình máy học. Nó kết hợp cả yếu tố động (momentum) và RMSprop để cung cấp một cách tự điều chỉnh tỷ lệ học cho từng tham số.

Giống như RMSprop, Adam sử dụng trung bình bình phương của bình phương đạo hàm trước đó (m_t) thêm cả một vector velocity (v_t).

Công thức như sau [2]

$$g_{t+1} = \nabla L(\theta_t)$$
$$m_{t+1} = \beta_1 m_t + (1-\beta_1) g_{t+1}$$
$$v_{t+1} = \beta_2 v_t + (1-\beta_2) g_{t+1}^2$$
$$\hat{m}_{t+1} = \frac{m_{t+1}}{(1-\beta_1^{t+1})};\ \hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$$

Cập nhật trọng số $\theta_{t+1} = \theta - \alpha * \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}}+\epsilon}$ [2]

- Ưu Điểm:

+ Adam có thể ứng dụng được trong nhiều loại mô hình và bài toán học máy.

+ Có thể tự động điều chỉnh tỷ lệ học và momentum cho từng tham số, tăng tính linh hoạt trong quá trình huấn luyện..

- Nhược Điểm:

+ Do là sự kết hợp của GD Momentum và RMSProp nên vẫn phải phụ thuộc nhiều vào các tham số học và sự điều chỉnh các tham số

+ Trong một số trường hợp, Adam optimizer có xu hướng bị overfitting nhất là trong các bài toán với tập dữ liệu nhỏ

2.4 Proximal Policy Optimization (PPO)

PPO là một thuật toán gradient-based, nghĩa là nó sử dụng gradient của hàm phần thưởng để cập nhật chính sách. Tuy nhiên, PPO khác với các thuật toán gradient-based khác ở chỗ nó sử dụng một phương pháp gọi là clipping để giới hạn tốc độ cập nhật của chính sách. Việc giới hạn thay đổi của chính sách tại mỗi epoch vì các nguyên nhân sau:

- Trên thực tế thì việc thay đổi chính sách càng nhỏ trong quá trình training thì khả năng đạt được giải pháp tối ưu càng cao. Điều này tránh được cực trị cục bộ
- Việc thay đổi lớn chính sách còn làm cho kết quả đi lệch hướng và tốn thời gian hoặc không thể tìm ra giải pháp

Figure 6. Minh hoạ PPO

Với PPO, chúng ta cập nhật chính sách một cách có kiểm soát. Để làm điều này, chúng ta cần đo lường mức độ thay đổi của chính sách hiện tại so với chính sách trước đó bằng cách tính tỷ lệ giữa chính sách hiện tại và chính sách trước đó. Sau đó, chúng ta giới hạn tỷ lệ này trong khoảng $[1-\epsilon, 1+\epsilon]$, giúp chính sách hiện tại không đi quá xa so với chính sách cũ.

2.4.1 Phương pháp clipping

Đối với thuật toán reinforce (Monte Carlo Policy Gradient), mục tiêu của chúng ta là tối đa hóa giá trị kỳ vọng của tổng thưởng

$$L^{PG}(\theta) = E_t[log\pi_\theta(a_t|s_t) * A_t] \qquad [3]$$

- $log\pi_\theta(a_t|s_t)$: là log probability của action a tại state s
- A_t: là advantage để đánh giá hành động (A>0 là hành động tốt)

Với PPO công thức với phương pháp clipping được cập nhật như sau

$$L^{CLIP}(\theta) = E_t[\min(r_t(\theta) * A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) * A_t)] \quad [3]$$

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$: hàm tỉ lệ (ratio function)

  Hàm tỉ lệ chính là tỉ lệ xác suất giữa chính sách hiện tại π_θ và chính sách cũ $\pi_{\theta_{old}}$

  Qua đó có thể nhận thấy nếu tỉ lệ > 1 thì action và state sẽ giống chính sách hiên tại và ngược lại khi tỉ lệ < 1 thì giống với chính sách cũ
- Clipping part $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$

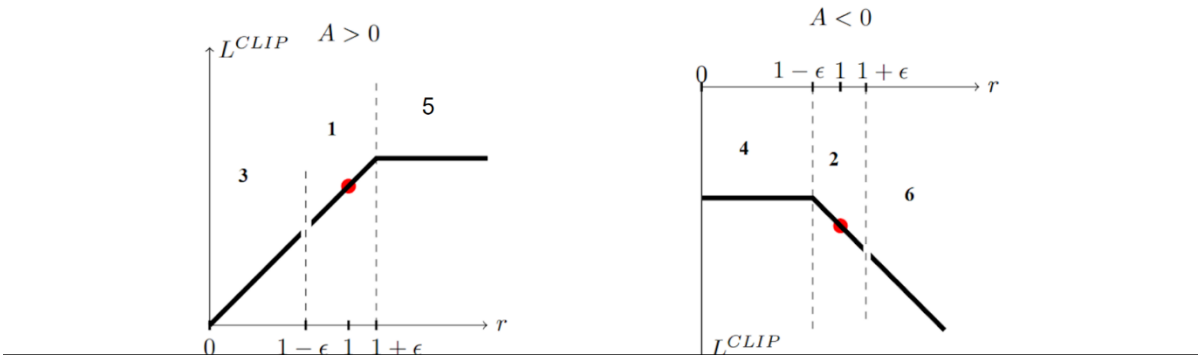
  hàm tỉ lệ sẽ bị chặn trong khoảng $[1-\epsilon, 1+\epsilon]$ (theo bài báo $\epsilon = 0.2$ [4]).

Clipping được sử dụng để đảm bảo rằng chính sách không bị cập nhật quá nhanh, chính sách mới không được quá khác biệt so với chính sách cũ. Điều này giúp giảm thiểu nguy cơ rơi vào cực trị cục bộ.

Dựa trên công thức có thể thấy chúng ta có 2 tỉ lệ (không bị chặn và bị chặn). Dưới đây là minh hoạ cho các phần bị chặn và không bị chặn

Table 1. Bảng biến thiên ratio [4]

	$p_t(\theta) > 0$	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	−	✓



Với A>0 là action được đánh giá tốt và ngược lại. Như đã nói ở trên khi ratio > 1 thì action và state gần về chính sách hiện tại và ngược lại. Trong trường hợp này việc giới hạn lại ratio giúp cho tránh việc thực hiện thay đổi chính sách quá lớn. Khi A>0 khả năng step được thực hiện

cao, ratio sẽ bị chặn trên ngưỡng $1+\in$ nên việc thực hiện gradient step sẽ không quá lớn so với cũ. Ngược lại A < 0 khả năng thực hiện step thấp nên ratio bị chặn tại $1-\epsilon$ khiến khả năng thực hiện step sai không quá lớn so với cũ

2.4.2 Ưu nhược điểm

- Ưu điểm
  - Hiệu quả: PPO đã được chứng minh là hiệu quả trong việc giải quyết nhiều vấn đề học tăng cường khác nhau.
  - Tốc độ học: PPO có tốc độ học nhanh hơn so với các thuật toán gradient-based khác.
  - Độ ổn định: PPO có độ ổn định cao hơn so với các thuật toán gradient-based khác.
- Nhược điểm
  - Yêu cầu nhiều dữ liệu: PPO yêu cầu nhiều dữ liệu hơn so với các thuật toán học tăng cường khác.
  - Khó điều chỉnh tham số: Tham số ϵ trong PPO có thể khó điều chỉnh để đạt được hiệu suất tối ưu.

2.5 Google Cartographer[1]

Cartographer [5] là một hệ thống SLAM (Simultaneous Localization and Mapping) mã nguồn mở được phát triển bởi Google. Nó là một trong những công cụ quan trọng để xây dựng bản đồ và định vị đồng thời cho robot di động. Cartographer sử dụng dữ liệu từ các cảm biến như LIDAR (Light Detection and Ranging) và IMU (Inertial Measurement Unit) để xây dựng bản đồ của môi trường xung quanh và đồng thời định vị robot trong bản đồ đó.

[1] Sửa đổi 01/2024

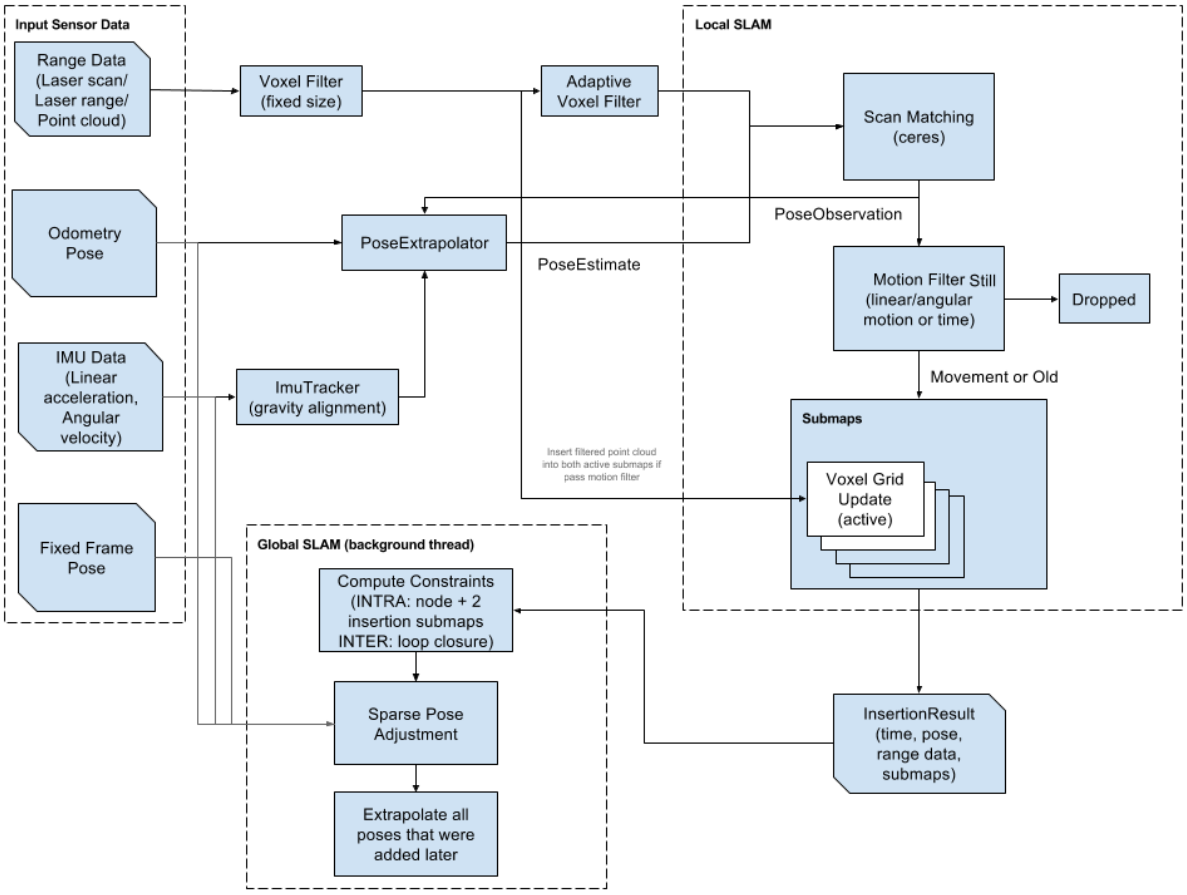


Figure 7 Cartographer diagram

Cartographer bao gồm 2 hệ thống: local SLAM và global SLAM.

Local SLAM (giống với frontend). Nhiệm vụ của nó là xây dựng một chuỗi các bản đồ con (submaps). Mỗi submap được thiết kế để đồng nhất cục bộ và sẽ có độ chệch theo thời gian.

Global SLAM (giống với backend). Chạy trong nền và nhiệm vụ chính của nó là tìm các loop closure constraints (thuật ngữ trong Cartographer). Điều này được thực hiện bằng cách so khớp các dữ liệu scan với các submap. Nó cũng sử dụng dữ liệu từ các cảm biến khác.

3. BÀI TOÁN ĐẶT RA

Bài toán cần xây dựng là việc tránh va chạm trong hệ đa robot trên ngữ cảnh các robot đều là nonholonomic (chuyển động có giới hạn) di chuyển trên một mặt phẳng Euclide với các chướng ngại vật. Trong quá trình huấn luyện có N robot đều được xem như là đồng nhất và dạng đĩa với bán kính R.

Tại mỗi thời điểm t, mỗi robot thứ i ($1 \leq i \leq N$) sẽ dùng một tập quan sát o_i^t để tính toán đưa ra một hành động a_i^t nhằm mục đích đưa nó về đích g_i từ điểm hiện tại p_i^t. Mỗi tập quan sát o_i^t được lấy từ phân phối xác xuất có liên quan đến lý thuyết trình bày về Reinforcement learning ở phần trên bao gồm trạng thái và thay đổi của mỗi robot thứ i ($s_i^t, o_i^t \sim \mathcal{O}(s_i^t)$), thông tin này chỉ chứa dữ liệu trạng thái của robot thứ i mà không có liên quan đến trạng thái và hoạt động của các robot khác. Công thức chỉ dựa trên một phần thông tin về môi trường giúp thuật toán dễ ứng dụng và bền vững.

Vector của tập quan sát của robot (hiểu là robot thứ i tuy nhiên i bị lược bỏ để công thức đơn giản) được định nghĩa

$o^t = [o_z^t, o_g^t, o_v^t]$ [6]

- o_z^t giá trị trả về của cảm biến scan 2D về thông tin môi trường
- o_g^t giá trị điểm goal dựa trên hệ toạ độ robot ở t
- o_v^t vận tốc robot tại t

Với o^t của mỗi robot là độc lập và dùng để tính hành động a^t theo chính sách π (giống nhau với tất cả robot):

$$a^t \sim \pi_\theta(a^t | o^t)$$

Trong đó θ là các trọng số của chính sách và a^t thực tế là tốc độ để đạt được đến đích vẫn đảm bảo tránh va chạm với robot khác hoặc vật cản $B_k (0 \leq k, = M)$ trong một khoảng thời gian Δt cho đến tập quan sát tiếp theo o^{t+1}. Do đó, vấn đề tránh va chạm của nhiều robot có thể được hiểu dưới dạng một vấn đề ra quyết định tuần tự dựa trên quan sát một phần. Chuỗi quyết định bao gồm cả tập quan sát và hành động $(o_i^t, v_i^t)_{t=0:t_i^g}$ cho robot thứ i được xem như là quy hoạch quỹ đạo l_i của robot từ vị trí bắt đầu tới goal ($p_i^{t=0} \rightarrow p_i^{t=t_i^g} \equiv g_i$) với t_i^g là thời gian đi đến goal.

Từ đó đưa ra được tập $\mathbb{L}$ là vector chứa quy hoạch quỹ đạo của N robot

$\mathbb{L} = \{l_i, i = 1, \dots, N \mid$

$v_i^t \sim \pi_\theta(a_i^t | o_i^t),$

$$p_i^t = p_i^{t-1} + \Delta t * v_i^t,$$

$$\forall j \in [1, N], j \neq i : \| p_i^t - p_j^t \| > 2R$$

$$\wedge \ \forall k \in [1, M] : \| p_i^t - B_k \| > R$$

$$\wedge \| v_i^t \| \leq v_i^{max}\}$$

Để tìm ra một chiến lược tối ưu, mục tiêu là tối thiểu được hàm kỳ vọng thời gian dịch chuyển trung bình của tất cả robot trong cùng một tình huống

$$\underset{\pi_\theta}{\operatorname{argmin}} \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} t_i^g | \pi_\theta\right] [6]$$

4. PHƯƠNG PHÁP THỰC HIỆN

4.1 Môi trường

Vấn đề chuỗi quyết định tuần tự được nêu ra ở phần 3 được giải quyết thống qua Partially Observable Markov Decision Process (POMDP). POMDP đơn giản là một mô hình RL đưa ra chuỗi các quyết định tuần tự tối ưu dựa trên thông tin quan sát từ môi trường. POMDP sẽ thường bao gồm các thành phần sau $\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \mathcal{O}$

- $\mathcal{S}$: không gian trạng thái
- $\mathcal{A}$: không gian hành động
- $\mathcal{P}$: mô hình chuyển đổi trạng thái
- $\mathcal{R}$: hàm phần thưởng
- Ω: không gian quan sát→ o ∈ Ω
- $\mathcal{O}$: tập quan sát của state hiện tại theo phân phối xác xuất (o ∈ $\mathcal{O}$)

Trong đó với việc mỗi robot đều là một cá thể độc lập (hoàn toàn phi tập trung) có các quy hoạch riêng mà không có ảnh hưởng bởi các thông tin robot khác, mô hình chuyển đổng trạng thái $\mathcal{P}$ dựa trên động học và động lực của các robot xác định gần như là không cần thiết. Do đó dưới đây là các mô tả về không gian quan sát, không gian hành động và hàm phần thưởng

4.1.1 Không gian quan sát (observation space)

Như đã đề cập ở phần 3, vector tập quan sát sẽ như sau $o^t = [o_z^t, o_g^t, o_v^t]$

Với o_z^t là thông tin từ cảm biến quét khoảng cách laser 2d, để chính xác hơn, o_z^t được được định nghĩa bao gồm 3 chuỗi thông tin khoảng cách gần nhất, trong góc quét $180^o(-90^o \rightarrow 90^o)$ với khoảng cách tối đa là 4 meters và 512 mẫu trả về trong mỗi lần quét. Các thông số trên được sử dụng trong quá trình trainning tuy nhiên sẽ thay đổi phù hợp với cảm biến khi ứng dụng trong thực tế. o_g^t là vector chứa thông tin vị trí của điểm goal trong hệ toạ độ cực lấy điểm gốc là vị trí hiện tại của robot. o_v^t là vector về vận tốc thẳng và vận tốc góc của robot. Các tập quan sát trên sẽ được chuẩn hoá bằng việc trừ đi cho giá trị trung bình và chia cho độ lệch chuẩn với số liệu của toàn bộ dữ liệu huấn luyện

4.1.2 Không gian hành động (action space)

Không gian hành động được định nghĩa là một tập các thông tin vận tốc thực hiện trong một không gian liên tục. Hành động của robot sẽ bao gồm chuyển động tịnh tiến và chuyển động xoay, $\rightarrow a^t = [v^t, \omega^t]$. Trong quá trình huấn luyện này, đặt khoảng vận tốc thẳng $v \in (0.0, 1.0)$ và khoảng vận tốc góc $\omega \in (-1.0, 1.0)$, lưu ý rằng vì cảm biến quét 2D được đặt ở góc 180 nên robot chỉ có thông tin phía trước robot nên việc chuyển động lùi là không được thực hiện

4.1.3 Hàm reward

Dựa trên yêu cầu về bài toán, hàm reward được thiết kế để đảm bảo tránh va chạm và tối thiểu thời gian di chuyển trung bình của tất cả robot bao gồm các yếu tố

$$r_i^t = ({}^g r)_i^t + ({}^c r)_i^t + ({}^\omega r)_i^t + ({}^b r)_i^t + ({}^s r)_i^t [2]$$

Tại mỗi thời điểm t reward được trả về cho robot thứ i là tổng của các thành phần ${}^g r, {}^c r, {}^\omega r, {}^b r$ và ${}^s r$. $({}^g r)_i^t$ là reward cho robot khi đến được goal:

$$({}^g r)_i^t = \begin{cases} r_{arrival} & \text{if } \| \mathrm{p}_i^{t-1} - g_i \| < 0.1 \\ \omega_g(\| \mathrm{p}_i^{t-1} - g_i \| - \| \mathrm{p}_i^t - g_i \|) & \text{otherwise} \end{cases} [6]$$

[2] Sửa đổi 01/2024

Robot sẽ được thưởng $r_{arrival}$ khi đến được goal (khoảng cách < 0.1), trong các trường hợp còn lại robot sẽ bị phạt với một penalty được tính dựa trên hiệu khoảng cách so với goal của thời điểm t -1 và thời điểm hiện tại nhân với một hệ số.

$$({}^{c}r)_i^t = \begin{cases} r_{collision} & \text{if } \| \mathrm{p}_i^t - \mathrm{p}_j^t \| < 2R \text{ or } \| \mathrm{p}_i^t - B_k \| < R \\ 0 & \text{otherwise} \end{cases} \quad [6]$$

Robot sẽ bị phạt với một reward collision (âm) khi bị va chạm, ở đây có 2 trường hợp bị va chạm. trường hợp đầu tiên là va chạm với robot khác (khoảng cách giữa 2 robot nhỏ hơn 2 lần bán kính 2R) và trường hợp robot va chạm với chướng ngại vật (khoảng cách robot với vật cản nhỏ hơn bán kính R). Tuy nhiên trên thực tế trong quá trình huấn luyện, ta chỉ xác định được 2 trạng thái là va chạm và không va chạm dựa trên cảm biến quét khoảng cách do đó việc xác định 2 trường hợp trên là không cần thiết.

$$({}^{\omega}r)_i^t = \omega_\omega |\omega_i^t| \qquad \text{if } |\omega_i^t| > 0.7$$

Để khuyến khích robot di chuyển mượt hơn ta đặt một penalty nếu mà góc xoay lớn hơn một hằng số xác định (góc dịch chuyển lớn) thì robot sẽ bị phạt

$$({}^{b}r)_i^t = \begin{cases} \frac{1}{\|\mathrm{p}_i^t - B_k\| + r_{break}} * \frac{1}{v + r_{break}} & \text{if } \| \mathrm{p}_i^t - B_k \| < 2R \text{ and } |v| > v_{\text{safe}} \\ 0 & \text{otherwise} \end{cases}$$

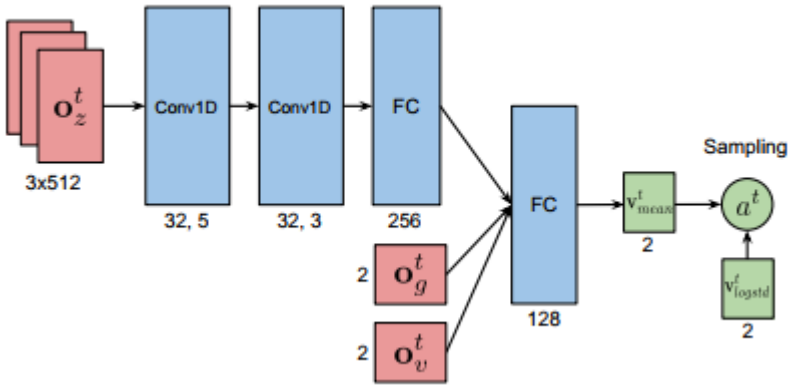
Khi robot tới càng gần vật cản thì robot cần phải đi chậm lại, đến tốc độ có thể kiểm soát được

$$({}^{s}r)_i^t = r_{stuck} \text{ if timeout}$$

Đặt các hằng số lần lượt là $r_{arrival} = 15, \omega_g = 2.5, r_{collision} = -15, \omega_\omega = -0.1, r_{break} = 1.2, v_{\text{safe}} = 0.2, r_{stuck} = -20$

4.2 Kiến trúc mạng

Với đầu vào là tập quan sát o^t và ngõ ra là hoành động (cụ thể là vận tốc diều khiển cho robot) ta định nghĩa chiến lược sử dụng mạng NN để ánh xạ o^t thành v^t. Kiến trúc của mạng NN sẽ bao gồm 4 lớp ẩn. Mô hình được mô tả bằng hình bên dưới



Trong đó, ngõ ra sau lớp fully connected là vận tốc điều khiển trung bình. Vận tốc này sau đó dược lấy mẫu với phân phối Gaussian với vector log độ lệch chuẩn

3 lớp ẩn đầu tiên để xử lý với tập quan sát của thông tin khoảng cách laser. Thiết kế như sau:

- 1st convolutional layer:
  - Input: 512x3 output: 255x32
  - Kernel size: 5
  - Stride: 2
  - Activation function: RELu
- 2nd convolutional layer:
  - Input: 255x32 output: 128x32
  - Kernel size: 3
  - Stride: 2
  - Activation function: RELu
- 3rd fully connected layer:
  - Input features: 128 x 32
  - Output features: 256

Sau khi tập quan sát môi quét môi trường được xử lý, vector kết quả sẽ được nối với vector thông tin của điểm goal trong hệ toạ độ robot với thông tin vận tốc hiện tại của robot. Vector tổng hợp này tiếp tục đi qua một lớp ẩn thứ 4:

- 4th fully connected layer:
  - Input features: 256+2+2=260
  - Output features; 128

Vector kết quả của lớp cuối cùng này sẽ được đưa qua 2 hàm activations khác nhau:

- Sigmoid để ràng buộc vận tốc tịnh tiến của robot trong khoảng [0.0, 1.0]
- Tanh để ràng buộc vận tốc góc trong khoảng [-1.0, 1.0]

Cuối cùng mạng nơ-ron ánh xạ vector quan sát đầu vào ot thành một vector vận tốc trung bình (v_{mean}^t). Hành động a^t sẽ được lấy mẫu từ phân phối Gaussian $\mathcal{N}(v_{mean}^t, v_{logstd}^t)$, trong đó v_{mean}^t đóng vai trò là giá trị trung bình và v_{logstd}^t tham chiếu đến một độ lệch chuẩn dưới dạng log, vector này chỉ được cập nhật độc lập trong quá trình huấn luyện. Điều này có nghĩa là độ lệch tiêu chuẩn của phân phối Gaussian không thay đổi trong quá trình triển khai mô hình, chỉ được điều chỉnh trong quá trình huấn luyện để cải thiện hiệu suất của mạng

4.3 Huấn luyện nhiều tình huống – nhiều giai đoạn

4.3.1 Thuật toán huấn luyện

Ở bài toán này sẽ tập trung ở việc học một chiến lược tránh va chạm mà có khả năng đảm bảo được tính bền vững và hiệu quả của hệ thống, kể cả khi số lượng robot lớn hoặc trong các môi trường, chướng ngại vật phức tạp. Như đã giới thiệu ở phần lý thuyết 2.4, phương pháp PPO là lựa chọn tối ưu trong bài toán này. Cụ thể hơn là mỗi robot sẽ thu thập các tập quan sát của nó tại mỗi thời điểm t o^t sau đó thực hiện action a^t được tính bởi chiến lược π_θ, các trọng số của chiến lược sẽ được cập nhật thông qua việc huấn luyện và thu thập thông tin trên tất cả robot cùng lúc.

Giai đoạn training sẽ được thực hiện như sau[3]

[3] Sửa đổi 01/2024

Algorithm 1 Training process

Initialize policy network π_θ and value function $V_\theta(s_t)$, and set hyperparameters
for *iteration* $= 1, 2, ...,$ **do**
  for *robot* $i = 1, 2, ...N$ **do**
    Run policy π_θ for T_i timesteps, collecting $\{o_i^t, r_i^t, a_i^t\}$, where $t \in [0, T_i]$
    Estimate advantages using GAE:
    $\hat{A}_i^t = \sum_{l=0}^{T_i} (\gamma\lambda)^l \delta_i^t$, where $\delta_i^t = r_i^t + \gamma V_\theta(s_i^{t+1}) - V_\theta(s_i^t)$
    break, if $\sum_{i=1}^{N} T_i > T_{max}$
  end for
  $\pi_{old} \leftarrow \pi_\theta$
  // Update policy
  for $j = 1, ..., Eb$ **do**
    $t = 1$
    while *sampler* **do**
      $\hat{A}_t \leftarrow \gamma \hat{A}_{t-1} + \delta_t$
      $\widetilde{P}_t \leftarrow \widetilde{P}_t \cdot \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$
      $t \leftarrow t + 1$
    end while
    $L_{CLIP}^{\pi}(\theta) = -\hat{\mathbb{E}}[\min\left(\widetilde{P}_t \hat{A}_t, \text{clip}\left(\widetilde{P}_t, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)]$
    $L^V(\theta) = -\hat{\mathbb{E}}[(\sum_{t'>t} \gamma^{t'-t} r_i^{t'} - V_\theta(s_i^t))^2]$

$$L_{PPO}^{\pi}(\theta) = \hat{\mathbb{E}}[L_{CLIP}^{\pi}(\theta) + c_1 \cdot L^V(\theta) + c_2 \cdot S[\pi_\theta](s_t)]$$

    Update with lr_π by Adam optimizer
  end for
end for

Đầu tiên khởi tạo chiến lược π_θ , hàm giá trị $V_\theta(s_t)$ và đặt các hằng số theo Table 2.

Tiếp theo là thu thập dữ liệu, mỗi robot thực hiện cùng một chiến lược π_θ để lấy batch data $\{o_i^t, r_i^t, a_i^t\}$ đến khi đạt T_{max}. Tham số advantages được tính theo Generalized Advantage Estimation (GAE) [7]. Trong đó hàm giá trị trạng thái $V_\theta(s_t)$ là cơ sở để ước lượng $\hat{A}_i^t$. Sau mỗi iteration thì policy cũ được gán lại để tính bước tiếp theo

Theo [3], phương pháp `KL Penalty Coefficient` có hiệu suất kém hơn khi so sánh với phương pháp clipping nên ở bước tiếp theo thuật toán sẽ áp dụng PPO với clipping .

Khi áp dụng PPO (phần 2.4), mạng giá trị V_θ chia sẻ trọng số với policy π_θ do đó ta có hàm mục tiêu như sau [3]

$$L_{PPO}^{\pi_\theta} = \hat{\mathbb{E}}\left[L_{CLIP}^{\pi_\theta} + c_1 \cdot L^{V_\theta} + c_2 \cdot S\left[\pi_{\{\theta\}}\right](s_t)\right]$$

Với $S\left[\pi_{\{\theta\}}\right](s_t)$ là giá trị entropy theo chính sách π_θ, do 2 mạng policy π_θ và mạng giá trị V_θ giống nhau và chia sẻ trọng số nên giá trị này sẽ tăng mức độ ngẫu nhiên và đa dạng của chính sách. Hàm mất mát giá trị L^{V_θ} được tính trên sự khác biệt giữa giá trị ước lượng tại trạng thái s_t giá trị thực tế. Và hàm mất mát clipping được tính bằng phương pháp clipping với hệ số $\epsilon = 0.2$, giá trị được chọn theo đó sẽ là nhỏ nhất giữa tích tỉ lệ chính sách nhân độ lợi và giá trị bị chặn trong khoảng $[1-\epsilon, 1+\epsilon]$.

Sau khi các giá trị hàm mất mát được cập nhật, thông số mạng được tối ưu ở bước cuối cùng bằng phương pháp Adam optimizer (phần 2.3.5)

Table 2. Bảng giá trị tham số

Tham số	**Giá trị**
$\boldsymbol{\lambda}$	0.95
$\boldsymbol{\gamma}$	0.99
$\boldsymbol{T_{max}}$	8000
$\boldsymbol{E_{\phi}}$	20
$\boldsymbol{\beta}$	1.0
$\boldsymbol{KL_{target}}$	$15e-4$
$\boldsymbol{\xi}$	50.0
l_{r_θ}	$5e-5(1st\ stage), 2e-5(2nd\ stage)$
E_V	10
$\boldsymbol{l_{r_\phi}}$	$1e-3$
$\boldsymbol{\beta_{high}}$	2.0
$\boldsymbol{\alpha}$	1.5
$\boldsymbol{\beta\_low}$	0.5

Thuật toán PPO này có thể dễ dàng mở rộng để áp dụng cho hệ thống nhiều robot với hàng trăm robot phi tập trung, vì mỗi robot độc lập thu thập dữ liệu. Việc thực hiện phi tập trung không chỉ giảm đáng kể thời gian thu thập mẫu mà còn làm cho thuật toán phù hợp để huấn luyện nhiều robot trong các tình huống khác nhau.

4.3.2 Các tình huống huấn luyện

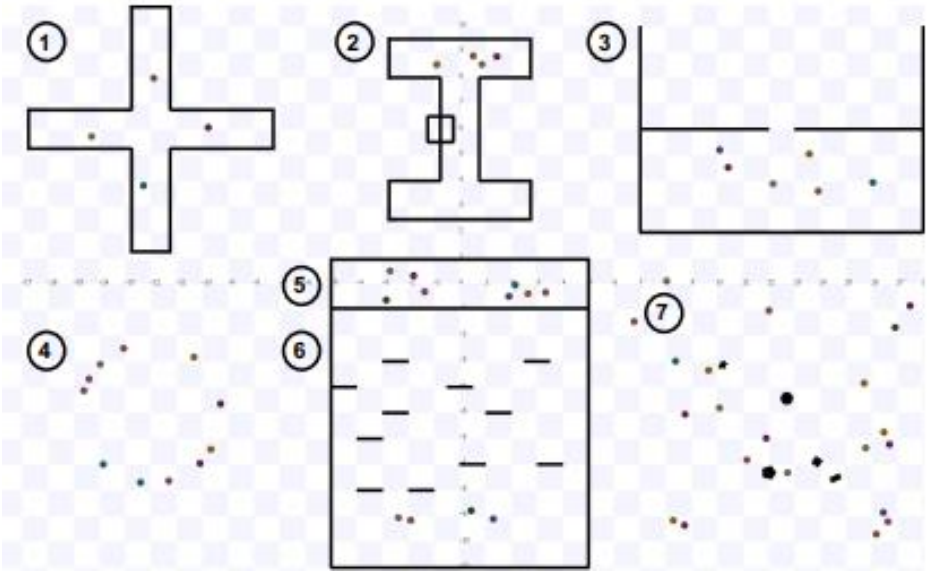


Figure 8. Các tình huống huấn luyện

Trên Figure 8 là tất cả các môi trường mô phỏng để huấn luyện cho robot. Để robot có thể hoạt động được trong các môi trường đa dạng và nhiều chướng ngại vật, robot được huấn luyện trong nhiều môi trường mô phỏng khác nhau với các độ khó khác nhau. Việc mô phỏng chuyển động robot và môi trường được thục hiện bằng Stage[4], các robot sẽ được di chuyển đồng thời, như đề cập ở trên các robot đều có dạng hình đĩa và đồng nhất với nhau.

Trong Figure 8, vùng huấn luyện của từng tình huống sẽ được giới hạn bởi các đường màu đen và các chướng ngại vật sẽ là các đường hoặc các hình dạng khối trong môi trường hoạt động. Ở các tình huống huấn luyện 1,2,3,5 và 6, mỗi robot được khởi tạo mở một vị trí ban đầu nhất định và sau đó sẽ ngẫu nhiên chọn các vị trí goal trong vùng huấn luyện tương ứng để robot thu thập dữ liệu. Với tình huống thứ 4, các robot được khởi tạo ở vị trí đầu ngẫu nhiên tạo thành một hình tròn, tình huống này hướng đến việc các robot phải đến goal ở vị trí bán cầu đối diện bằng việc băng ngang qua vùng tâm hình tròn nơi mà khả năng va chạm giữa các robot vô cùng lớn. Với tình huống huấn luyện thứ 7, vị trí khởi tạo của các robot cũng như là các chướng ngại

[4] Stage Simulator là một trình mô phỏng robot độc lập, được ứng dụng mô phỏng môi trường 2D. Stage có ưu điểm vượt trội hơn gazebo về tốc độ và tối ưu bộ nhớ do chỉ mô phỏng 2D và lược bỏ chi tiết nên rất thích hợp trong ứng dụng này
Docs: https://player-stage-manual.readthedocs.io/en/latest/

vật hoàn toàn ngẫu nhiên tại mỗi khi một epoch mới bắt đầu, việc này tạo tính đa dạng cho vị trí khởi tạo của robot và vật cản giúp tránh được overfitting.

Các tình huống huấn luyện đa dạng với độ phức tạp cao cộng với sự ngẫu nhiên trong việc chọn vị trí khởi tạo của robot sẽ giúp cho các robot có thông đa chiều trong tập quan sát; nhờ đó chất lượng và độ bền vững của chiến lược được huấn luyện sẽ cải thiện. Tuy nhiên việc có quá nhiều môi trường huấn luyện sẽ kéo dài thời gian huấn luyện và tăng khả năng xảy ra underfitting. Để khắc phục vấn đề trên, phương án huấn luyện nhiều giai đoạn được đề xuất ở phần tiếp theo.

4.3.3 Các giai đoạn

Như đã giải thích ở trên, để thúc đẩy nhanh thời gian đạt được điểm mà chiến lược thoả mãn yêu cầu bài toán, phương pháp huấn luyện 2 giai đoạn được đề xuất dựa trên ý tưởng từ "curriculum learning" [8]. Phương pháp này không chỉ cải thiện được thời gian đạt được chiến lược phù hợp mà còn đạt được reward cao hơn việc huấn luyện các tình huống hoàn toàn từ đầu với cùng số lượng epoch và tổng thời gian huấn luyện.

Cụ thể hơn, 2 giai đoạn gồm Stage 1 và Stage 2. Tại giai đoạn 1, số lượng robot được đưa vào huấn luyện là 20, 20 robot sẽ được huấn luyện ở tình huống ngẫu nhiên (môi trường 7 trên Figure 8) trong điều kiện không có các chướng ngại vật. Việc tạo môi trường đơn giản ở stage 1 sẽ giúp robot học nhanh hơn ở điều kiện khởi tạo ban đầu trong việc tránh nhau. Sau khi các robot đã đạt được đến điểm mà việc tránh nhau chấp nhận được, stage 1 được kết thúc và các thông số của chiến lược được lưu lại để sử dụng cho stage 2.

Thông số của policy được lưu tiếp tục được cập nhật ở stage 2. Tại stage 2, số lượng robot được tăng lên 58 và tất cả các tình huống huấn luyện ở Figure 8 sẽ được áp dụng. Với trọng số đã được cập nhật ở Stage 1, việc huấn luyện tiếp tục sẽ khiến cho quá trình học hiệu quả hơn. Kết quả cuối cùng của chiến lược trong stage 2 sẽ được ứng dụng vào mô hình robot thực tế.

4.4 Định vị robot trong môi trường [5]

Để robot di chuyển từ toạ độ bắt đầu đến toạ độ điểm kết thúc một các chính xác, việc xác định toạ độ chính xác trong môi trường thực tế là điều quan trọng. Nếu chỉ dựa trên hệ toạ độ robot

[5] Sửa đổi 01/2024

và vị trí tính toán theo cảm biến imu, kết quả đưa sẽ có sai số lớn và không đáng tin cậy do hiện tượng drifting.

Trong nghiên cứu này việc định vị robot sẽ sử dụng Google Cartographer. Thuật toán sẽ kết hợp giữ dữ liệu LIDAR và IMU để tạo ra bản đồ cho robot trong không gian.

Như đã đề cập ở phần 2.5, thuật toán sẽ bao gồm 2 phần local SLAM và global SLAM. Mỗi phần đều để ước lượng các trạng thái (scans) gồm $(\xi_x, \xi_y, \xi_\theta)$ lần lượt lại toạ độ x,y và góc xoay θ. Với gốc toạ độ đặt ở điểm 0,0.

Dữ liệu từ lidar được local SLAM sử dụng để tạo các submap dựa trên các scan. Mỗi scan xây dựng bằng các grid được xác định khoảng cách có vật cản (hits) và không gian (misses) như hình dưới

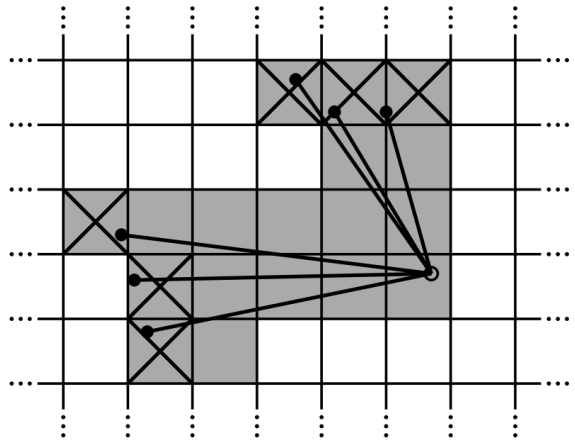


Figure 9 Bản đồ scan với hits and misses

Trong đó các phần có vật cản được đánh dấu x và phần không gian xác định bằng màu xám. Sau đó các scan $(\xi_x, \xi_y, \xi_\theta)$ được so khớp với nhau bằng thuật toán Ceres scan matching [6] được thêm vào các submap để tạo các submap hoàn chỉnh. Trong đó dữ liệu cảm biến IMU sẽ được dùng để ước lượng góc xoay θ hoặc ước lượng vị trí trong không gian mở với ít đặc trưng

Việc tạo ra các submaps thông qua matching các scans gần nhất sẽ dần tích luỹ sai số. Đối với các submap thì vài chục lần quét tích luỹ sai số không đáng kể. Tuy nhiên đối với toàn bộ map thì cần phải kết hợp nhiều submaps với nhau. Global SLAM sẽ được thực hiện việc đó để tạo

[6] Một phần của thư viện Ceres Solver [10]

bản đồ toàn cục dựa trên Sparse Pose Adjustment [9], các scan và submaps sẽ được lưu trữ trong bộ nhớ để thực hiện tối ưu hoá loop closing. Một bộ so khớp sẽ chạy trong nền nhiệm vụ của nó là sắp xếp lại các bản đồ con với nhau sao cho có thể tạo ra được một map hoàn chỉnh.

Thực hiện SLAM sẽ giúp robot xác định được bản đồ khu vực hoạt động và vị trí chính xác của nó thông qua toạ độ trên bản đồ.

4.5 Thiết kế mô hình robot thực tế

Để kiểm chứng tính hiệu quả của chiến lược, tiến hành xây dựng 3 mô hình robot trong thực tế. Do điều kiện về kinh phí và thời gian, số lượng robot được áp dụng là 3 và các cảm biến đặc biệt là cảm biến quét lidar được chọn loại có độ chính xác trung bình. Trong phạm vi đề cương mô hình sẽ được thiết kế trước, sau khi hoàn thiện phần thuật toán và chứng minh được tính khả thi ở phần tiếp theo của luận văn thì mô hình robot sẽ được lắp ráp thực tế

4.5.1 Thành phần cơ bản

Để đáp ứng được các yêu cầu robot cơ bản cần phải có đủ thành phần sau:

- Khung robot:
  Phần khung robot sẽ được thiết kế dưới dạng hình tròn đường kính 20cm, để có thể đặt các thiết bị và cảm biến, khung robot sẽ có 3 tầng. Với tầng đầu tiên là vị trí lắp đạt động cơ, mạch động cơ, bánh xe và pin. Tầng 2 sẽ bao gồm các bộ điều khiển, máy tính. Tầng trên cùng là nơi đặt các cảm biến. Kích thước của robot thực tế: 20cmx20cmx15.7cm
- Động cơ và cảm biến
  Để tiết kiệm được thời gian và chi phí, mô hình robot sẽ sử dụng 2 động cơ stepper.
  Các cảm biến của robot bao gồm:
  Cảm biến lidar 360
  Một số cảm biến va chạm để hỗ trợ robot nếu lidar không phát hiện được va chạm
  Cảm biến IMU
- Bộ điều khiển:
  Để thực hiện thuật toán robot sẽ được trang bị một máy tính nhúng raspberry pi. Hệ điều hành được sử dụng là ubuntu. Ngoài máy tính nhúng sẽ có thêm các arduino để đưa tín hiệu tới bộ điều khiển động cơ và xử lí các tín hiệu cảm biến đưa về cho bộ xử lí trung tâm

4.5.2 Mô hình thiết kế trên solidwork[7]

Mô hình robot được thiết kế mô phỏng trên solidwork. Sau khi hoàn thiện phần code và mô phỏng được thuật toán thì robot sẽ được gia công thực tế và ứng dụng

Các hình ảnh robot được thiết kế

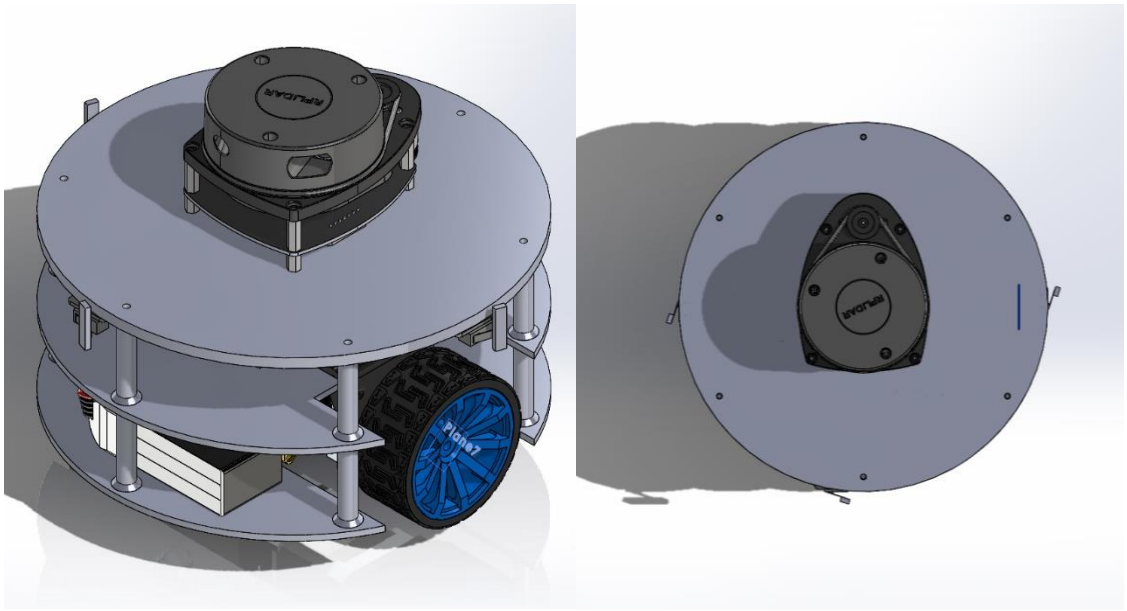


Figure 10. Tổng quan và mặt top

[7] SolidWorks là một phần mềm CAD (Computer-Aided Design) nổi tiếng được sử dụng để mô hình hóa 3D, mô phỏng, và thiết kế sản phẩm kỹ thuật.

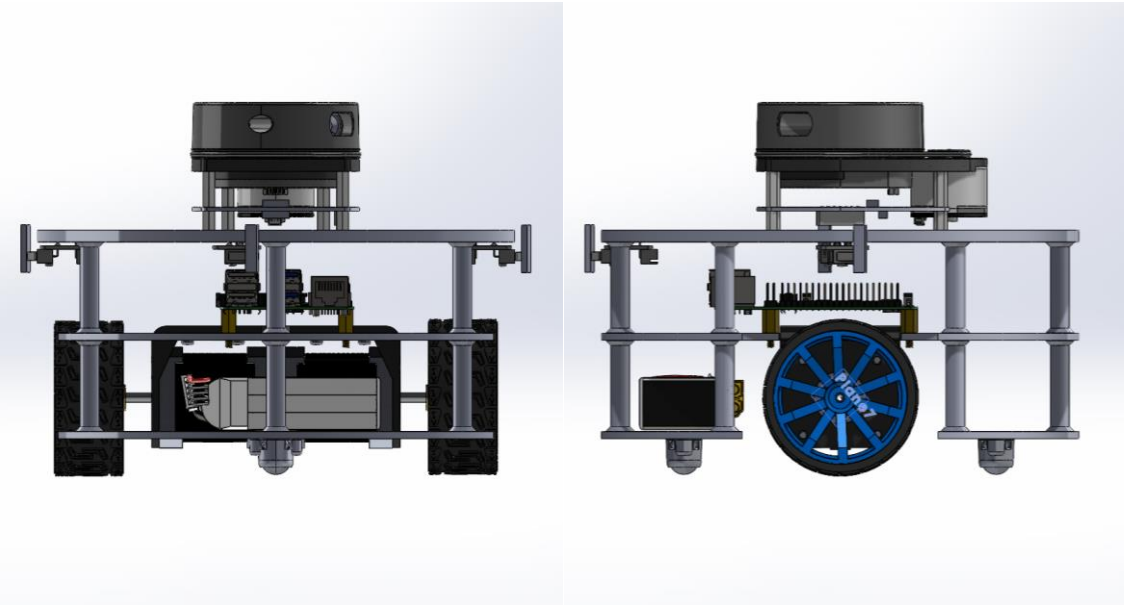


Figure 11. Mặt front và rear

5. KẾT QUẢ VÀ HƯỚNG PHÁT TRIỂN

5.1 Kết quả

Dựa trên mục tiêu của đề cương lần này, quá trình tìm hiểu và nghiên cứu bài báo tham khảo đã được hoàn thành cụ thể như sau:

- Hiểu được ý tưởng của thuật toán, thuật toán đã sử dụng phương pháp nào để huấn luyện, cập nhật trọng số, hàm tối ưu và cấu trúc mạng được sử dụng
- Nắm được những điểm mà có thể và chưa thể ứng dụng của thuật toán để tiến hành phần mô phỏng
- Đã code được các phần rời rạc cho thuật toán như cấu trúc mạng, các bước để tạo action và cập nhật trọng số, dựng được môi trường Stage để mô phỏng
- Đã vẽ được mô hình cho robot để có thể gia công
- Chưa thể hoàn thiện code để mô phỏng phần huấn luyện
- Chưa bắt đầu quá trình gia công phần cứng

5.2 Hướng phát triển

Trong luận văn sẽ tiếp tục hoàn thiện những phần còn dang dở. Việc mô phỏng và huấn luyện mạng sẽ được ưu tiên làm trước để đánh giá được chất lượng của thuật toán. Sau đó sẽ gia công và phát triển code cho robot để các robot có thể thực hiện được việc chuyển động đến các goal của chúng ở nhiều điều kiện khác nhau

6. MỤC LỤC HÌNH ẢNH

7. TÀI LIỆU THAM KHẢO

[1] S. Ruder, "An overview of gradient descent optimization algorithms," 9 2016.

[2] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal Policy Optimization Algorithms," 7 2017.

[4] Daniel Bick, "Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization," Netherlands, 2021.

[5] W. Hess, D. Kohler, H. Rapp and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," *Proceedings - IEEE International Conference on Robotics and Automation,* Vols. 2016-June, 2016.

[6] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," *Proceedings - IEEE International Conference on Robotics and Automation,* 2018.

[7] J. Schulman, P. Moritz, S. Levine, M. Jordan and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," 6 2015.

[8] Y. Bengio, J. Louradour, R. Collobert and J. Weston, "Curriculum learning," *Proceedings of the 26th Annual International Conference on Machine Learning,* pp. 41-48, 6 2009.

[9] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai and R. Vincent, "Efficient sparse pose adjustment for 2D mapping," *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings,* 2010.

[10] S. Agarwal, K. Mierle and Others, "Ceres Solver," *http://ceres-solver.org.*