

**ĐẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**Mô phỏng điều khiển TurtleBot 3 bám  
quỹ đạo tối ưu sử dụng giải thuật  
Model Predictive Control**

**Tiểu luận môn học Kỹ thuật điều khiển nâng cao**

**Học viên: Nguyễn Trường Sơn**

**Mã số học viên: 20025058**

**HÀ NỘI – 2021**

## **Mục lục**

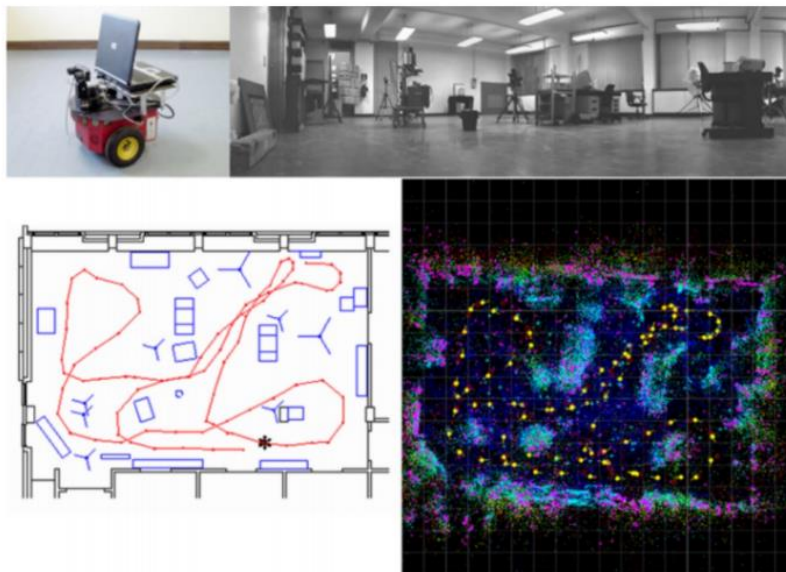
<b>I. SLAM (Simultaneous Localization and Mapping) và bài toán dẫn đường của Robot ..</b>	<b>3</b>
1. SLAM - Simultaneous Localization And Mapping.....	3
2. Bài toán dẫn đường cho robot .....	4
<b>II. Giải thuật MPC (Model Predictive Control) bám quỹ đạo với TurtleBot 3 .....</b>	<b>8</b>
1. Mô hình động học của TurtleBot 3.....	8
2. Giải thuật MPC trong bám quỹ đạo của robot .....	9
a. Bộ điều khiển Model Predictive Control (MPC) .....	9
b. Hệ thống điều khiển vòng kín với MPC trên robot điều khiển vi sai .....	11
3. Tích hợp giải thuật MPC vào Navigation Stack của ROS .....	13
<b>III. Mô phỏng giải thuật điều khiển MPC trên Gazebo.....</b>	<b>16</b>
1. Mô phỏng robot điều khiển vi sai bám quỹ đạo sử dụng MPC .....	16
2. Mô phỏng TurtleBot 3 bám quỹ đạo sử dụng Gazebo .....	19

## I. SLAM (Simultaneous Localization and Mapping) và bài toán dẫn đường của Robot

Cùng với sự phát triển của công nghệ Robot, các bài toán trong robot ngày càng được quan tâm nhiều hơn, một trong số đó là bài toán về xây dựng hệ thống bản đồ thời gian thực (SLAM - Simultaneous Localization and Mapping) và bài toán điều hướng hay dẫn đường (Navigation) của robot.

### 1. SLAM - Simultaneous Localization And Mapping

Bài toán định vị và xây dựng bản đồ hay SLAM là bài toán xây dựng bản đồ của môi trường xung quanh robot, đồng thời có thể dựa vào những thông tin từ bản đồ nhằm định vị, xác định vị trí hiện tại của robot trong môi trường. Bản thân SLAM là viết tắt của Simultaneous Localization and Mapping, có nghĩa là xử lý đồng thời hai bài toán là xác định vị trí hiện tại của robot và xây dựng bản đồ của môi trường xung quanh. Trong lĩnh vực robotics, để di chuyển chính xác trong môi trường, một robot phải có được thông tin bản đồ ngữ nghĩa của môi trường. Tuy nhiên, để xây dựng được chính xác bản đồ của môi trường xung quanh, robot cần phải biết chính xác vị trí hiện tại của nó. Hai vấn đề này có quan hệ hỗ trợ lẫn nhau.



*Hình 1 : Minh họa về quá trình định vị và xây dựng bản đồ của Robot*

Trong hình trên mô tả quá trình định vị thông qua việc xây dựng bản đồ của Robot. Tại mỗi thời điểm, robot sẽ xác định các tương quan về vị trí so với các điểm mốc có thể xác định được trước đó từ đó ước lượng ra vị trí chính xác thông qua các tương quan này. Trong các cánh tiếp cận của bài toán SLAM thì điều kiện cần để robot có thể xác định được vị trí hiện tại là cần có một điểm mốc trong môi trường để có thể tính toán ước lượng từ vị trí này tới vị trí hiện tại. Thông thường điểm mốc này là các điểm cố định theo thời gian, có nhiều đặc trưng dễ phát hiện. Các phương pháp được dùng để xác định mối quan hệ này có thể kể đến như sử dụng xử lý ảnh, sử dụng cảm biến hồng ngoại (IR sensor) hoặc là sử dụng cảm biến siêu âm (Ultrasonic sensor).

Một số thuật ngữ hay được sử dụng để mô tả về bài toán SLAM bao gồm:

- “Features”: đây là thuật ngữ dùng để chỉ những điểm đặc trưng của môi trường mà robot có thể nhận biết được. Trong thực tế, các feature thường là các điểm, các vật thể, góc cạnh, ... nằm xung quanh robot.
- “map”: thuật ngữ này dùng để vector các vị trí của features. Map được hình thành thông qua quá trình di chuyển của robot. Tại mỗi thời điểm robot sẽ xác định các features trong phạm vi định vị và cập nhật map dựa vào các thông số về quá trình di chuyển và sự tương quan giữa các features khác trong map.
- “pose” hoặc “robot pose”: thuật ngữ này để xác định trạng thái của robot, bao gồm các biến liên quan đến robot.
- “odometry”: đây là một thuật ngữ để mô tả quá trình sử dụng những dữ liệu thu được từ các loại cảm biến chuyển động để ước tính sự thay đổi vị trí theo thời gian của robot. Odometry thường được sử dụng để tăng tính chính xác trong quá trình định vị của robot.

Trong bài toán SLAM, việc sử dụng mối quan hệ giữa vị trí của features và pose của robot là vấn đề cốt lõi ảnh hưởng đến độ chính xác của bài toán. Khi robot di chuyển theo một quãng đường và một hướng cho trước, nếu các chuyển động là hoàn toàn chính xác thì việc xác định features và vị trí của robot có thể xác định dễ dàng bằng việc sử dụng thông tin từ việc dịch chuyển. Tuy nhiên do quá trình chuyển động không chính xác, robot phải tìm kiếm các features để xác định lại vị trí của nó. Khi xác định được các features, vị trí mới của robot sẽ được xác định thông qua sự thay đổi tương quan giữa robot và features tại vị trí cũ và mới. Quá trình xây dựng map được thực hiện sau khi đã xác định được vị trí của robot. Sau khi đã xác định được sự thay đổi vị trí của robot so với trước đó cũng như so với vị trí ban đầu, các features sẽ tham gia xây dựng map bằng cách xác định vị trí features trong map và bổ sung các features mới để tiếp tục mở rộng map. Hai quá trình kể trên tuy là trái ngược nhau nhưng lại bổ sung sự chính xác cho nhau, việc xây dựng map chính xác giúp xác định tốt hơn pose của robot, ngược lại việc xác định pose của robot và tương quan với các features nhằm xây dựng map chính xác.

Hiện nay có nhiều ứng dụng thực tế được triển khai sử dụng công nghệ SLAM. Trong lĩnh vực xe ô tô, các hãng xe lớn trên thế giới như GM, Mercedes-Benz, Audi, BMW và đặc biệt là Tesla đang đẩy mạnh đầu tư vào nghiên cứu xe tự lái. Các dòng xe của Tesla trong những năm qua đã cho thấy khả năng tự hành tốt trong một số điều kiện nhất định. Bên cạnh đó tại một số nước đã có đưa vào sử dụng xe bus không người lái. Trong lĩnh vực thiết bị bay không người lái, các dòng thiết bị ứng dụng cho các bài toán tự hành trong các môi trường khác nhau cũng được triển khai mạnh mẽ. Một ví dụ có thể kể đến như Yuneec H920 Drone được sử dụng để theo dõi quá trình xây dựng một sân vận động ở Sacramento, California; và phát hiện khu vực nào mà công trình bị chậm tiến độ. Yuneec H920 Drone cũng được NASA sử dụng trong máy bay không người lái Global Hawk để theo dõi thời tiết trong mùa bão năm 2019.

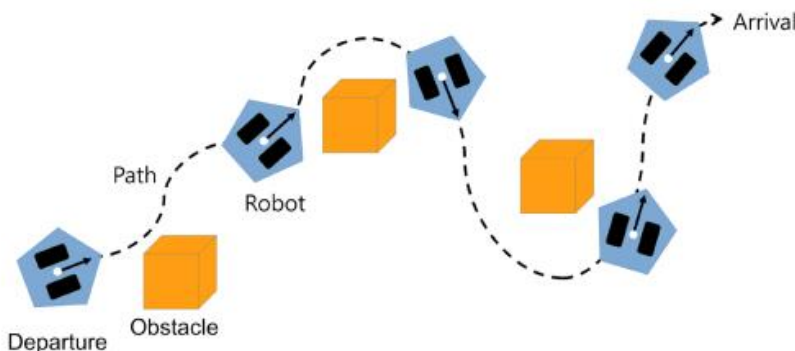
## **2. Bài toán dẫn đường cho robot**

Navigation hay dẫn đường là một trong những thuật ngữ khá phổ biến hiện nay, và nổi tiếng nhất chúng ta có thể nghĩ tới là GPS Navigation. Khi chúng ta muốn đi tới một địa điểm nào đó, chúng ta chỉ cần mở điện thoại lên, bật Google Map (hoặc một ứng dụng dẫn đường nào đó như HERE Maps, Waze, ...) chọn điểm cần đến. Sau đó, hệ thống

navigation sẽ kiểm tra khoảng cách từ địa điểm bạn đang đứng tới điểm cần đến và đưa ra được đường cần đi để tới đích. Bạn cũng có thể cài đặt những đặc điểm cụ thể cho đường đi như tránh đoạn tắc đường, thời gian ước lượng hay đường nào gần hơn, ....

Hệ thống Navigation có lịch sử khá gần với hiện tại. Vào năm 1981, Honda (Một công ty sản xuất xe ô tô của Nhật Bản) đã đề xuất ra một hệ thống analog dựa trên một 3-axis gyroscope và một bản đồ hình ảnh gọi là “ElectroGyrolocator”. Ngay sau đó hệ thống Etak Navigation được giới thiệu bởi Etak – một công ty về automotive của Mỹ. Đây là một hệ thống Navigation điện tử hoạt động bằng một la bàn số và một cảm biến được gắn vào bánh xe. Tuy nhiên, việc gắn thêm cảm biến này làm tăng giá thành của ô tô nhưng lại ảnh hưởng tới độ tin cậy của hệ thống Navigation. Sau đó, vào năm 2000 Mỹ đã dân sự hóa hệ thống vệ tinh định vị của mình, với 24 vệ tinh GPS (Global Positioning System) hệ thống thống Navigation dựa trên vệ tinh đã được phổ biến ra toàn cầu.

Trong lĩnh vực robot, Navigation là một trong những điểm nổi bật của robot di động, hệ thống này mang lại cho robot khả năng tự hoạt động trong môi trường. Hệ thống Navigation là một trong những thành phần thiết yếu, không thể thiếu được của robot di động. Dẫn đường cho robot đi tới đích là một trong những bài toán quan trọng của robot vì không như con người có khả năng thu thập thông tin tuyệt vời, robot cần phải biết được nó đang ở đâu và bản đồ của khu vực nó đang hoạt động. Đây cũng là những thông tin quan trọng dùng để tối ưu hóa quá trình di chuyển và đường đi của robot và tránh vật cản như tường hay đồ nội thất. Đây thực sự là một bài toán khó trong robot.



Hình 2 : Hệ thống Navigation trong Robot

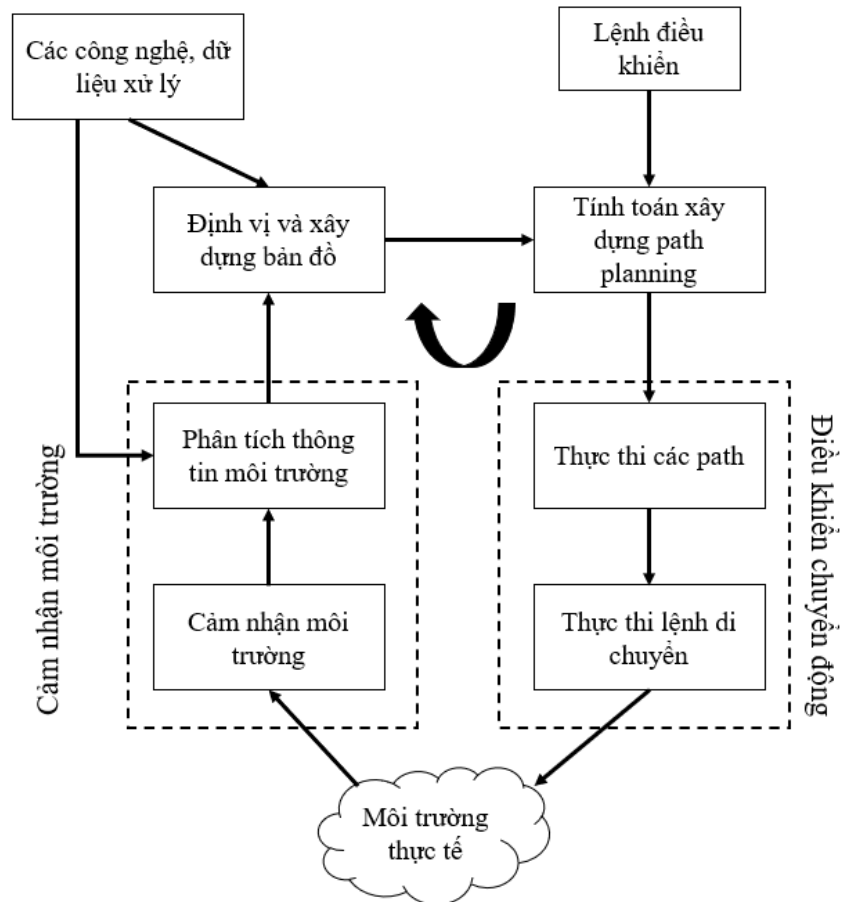
Vậy những thông tin cần thiết cho quá trình dẫn đường của robot? Điều này phụ thuộc khá nhiều vào thuật toán dẫn đường mà chúng ta sử dụng nhưng thông thường sẽ là các thông tin sau:

- Bản đồ: Yêu cầu đầu tiên của hệ thống Navigation là bản đồ (Map). Hệ thống Navigation thông thường sẽ thường được cung cấp bản đồ có sẵn (ví dụ như bản đồ của hệ thống GPS) nhưng với robot của chúng ta thì điều này là bất khả thi. Bởi vì chúng ta không thể tìm được tất cả các bản đồ tại mọi nơi của thế giới nhất là trong phòng hoặc khu vực đặt biệt. Từ đó dẫn tới việc chúng ta phải tự tạo bản đồ và đưa nó cho robot hoặc tự bản thân robot phải xây dựng bản đồ này. SLAM đã được phát triển để robot có khả năng tự tạo bản đồ mà không cần sự hỗ trợ của con người. Đây là một phương pháp để robot có thể tự tạo bản đồ trong quá trình khám phá một khu vực mới, nhận diện xung quanh và tính toán để đưa ra bản đồ.

- Pose của robot: Yếu tố thứ hai, tất nhiên là pose của robot. Trong bất kỳ quá trình điều khiển nào của robot, chúng ta cũng cần biết được pose của robot và trong Navigation cũng vậy. Chúng ta cần hiểu và sử dụng những mối liên quan về tọa độ cũng hướng của robot với môi trường để đưa ra được đường đi chính xác cho robot.

- Cảm biến: Yếu tố thứ ba là cảm biến. Trong quá trình di chuyển, robot phải phát hiện được những chướng ngại vật xung quanh mình hoặc ít nhất là chướng ngại vật trên hướng mà mình di chuyển và để làm được điều này robot cần được tích hợp thêm các cảm biến. Có nhiều loại cảm biến khác nhau có thể được sử dụng như các cảm biến về khoảng cách và cảm biến hình ảnh. Các cảm biến khoảng cách có thể là những cảm biến dựa trên laser để tính toán khoảng cách (ví dụ như LDS, LRF, Lidar), cảm biến siêu âm, cảm biến hồng ngoại, ... Các cảm biến hình ảnh thì có thể là stereo camera, moncamera, camera đa hướng, .... Gần đây có thể kể đến như các hệ thống Realsense, Kinect, Xtion là các hệ thống depth camera được sử dụng để xác định vật cản.

- Tính toán đường đi và điều khiển: Yếu tố quan trọng cuối cùng của hệ thống Navigation đó là tính toán và tối ưu hóa đường đi của robot tới đích. Đây được gọi là quá trình path search and planning. Có rất nhiều giải thuật có thể kể đến như Dijkstra, A, A-Star, RRT (Rapidly exploring Random Tree).



Hình 3: Hệ thống điều khiển robot di động

Hình trên mô tả quá trình tương tác của robot với môi trường thực tế trong bài toán dẫn đường của robot, đây là một hệ thống lớn và phức tạp, tuy nhiên trong bài tiểu luận này, tôi tập trung vào bài toán lập kế hoạch và điều khiển chuyển động của robot. Thông thường, đối với bài toán lập kế hoạch chuyển động và điều khiển của robot, có một cách tiếp cận là giải bài toán tối ưu theo mỗi step chuyển động của robot, tối ưu quỹ đạo và đầu vào điều khiển tương ứng đồng thời (ví dụ như Dynamic Window Approach-DWA hay Timed Elastic Band-TEB). Tuy nhiên, nếu gặp môi trường phức tạp, hướng tiếp cận này trở nên quá phức tạp cho tính toán. Do đó, trong thực tế, cách tiếp cận phổ biến nhất là chia bài toán thành hai phần: tính toán đường đi tối ưu và điều khiển bám quỹ đạo tối ưu. Đối với phần tính toán đường đi tối ưu, nhiều thuật toán đã được phát triển như: dựa trên lý thuyết đồ thị (A-Star, D-Star...) và dựa trên phương pháp lấy mẫu (như Rapidly Exploring Random Trees-RRT, RRT\*...). Từ những giải thuật này, robot có thể lập ra một quỹ đạo tối ưu trong thời gian thực. Giai đoạn tiếp theo là điều khiển robot bám theo quỹ đạo này. Chúng ta có thể áp dụng các giải thuật như PID, Fuzzy, .... Trong tiểu luận này, chúng tôi ứng dụng giải thuật MPC cho bài toán bám quỹ đạo của robot. MPC đã được sử dụng rộng rãi để giải quyết bài toán bám quỹ đạo, vì nó có thể xử lý các mô hình động học phi tuyến và các ràng buộc một cách có hệ thống. Bằng cách giảm thiểu một hàm tối ưu, một chuỗi tối ưu của các đầu vào điều khiển trong tương lai được tạo ra và áp dụng sao cho các trạng thái của robot được điều chỉnh tối ưu theo quỹ đạo tham chiếu.

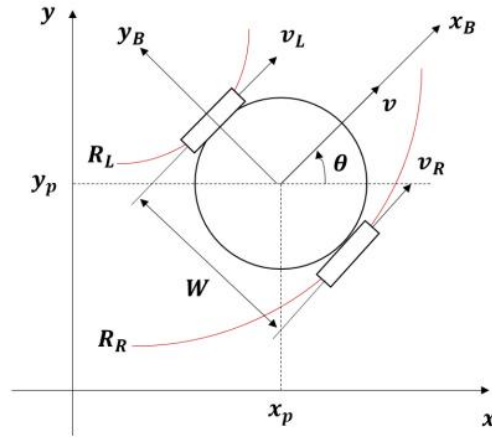
Bài tiểu luận được trình bày qua 4 phần, phần 1 là giới thiệu về bài toán SLAM và bài toán dẫn đường cho robot. Phần 2 sẽ trình bày về giải thuật MPC và ứng dụng giải thuật này trong bài toán bám quỹ đạo của Robot. Phần 3 trình bày về mô phỏng và kết quả mô phỏng giải thuật này trên TurtleBot 3 với Navigation Stack trong ROS. Cuối cùng là phần 4 kết luận.

## II. Giải thuật MPC (Model Predictive Control) bám quỹ đạo với TurtleBot 3

Nhằm áp dụng giải thuật MPC cho nhiệm vụ bám quỹ đạo của robot, trong bài tiểu luận này, chúng tôi sử dụng mô hình robot điều khiển vi sai, cụ thể là TurtleBot 3. TurtleBot 3 là một mẫu robot di động nhỏ gọn, được ứng dụng rộng rãi trong các chương trình giáo dục và nghiên cứu về robot. Và kết hợp với ROS (Robot Operating System), TurtleBot 3 cũng cấp rất nhiều công cụ cho người sử dụng nghiên cứu và phát triển các bài toán đi từ mô phỏng tới thực tế.

### 1. Mô hình động học của TurtleBot 3

Đầu tiên, chúng ta tìm hiểu về mô hình động học của TurtleBot 3. Như đã đề cập bên trên, TurtleBot 3 là robot dạng điều khiển vi sai (trừ một các mẫu thiết kế đặc biệt). Mô hình robot được thể hiện như hình dưới:



Hình 4: Hệ trục tọa độ cho robot điều khiển vi sai

Trong đó:

- góc  $\theta$  là góc tạo bởi trục từ tâm của hai bánh xe với trục  $x$ .
  - $v_L$  là vận tốc bánh trái
  - $v_R$  là vận tốc bánh phải
- ⇒ Vận tốc di chuyển của robot là

$$v = \frac{v_L + v_R}{2}$$

Từ đó ta có phương trình chuyển động của Robot là:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (1)$$

Với  $W$  là khoảng cách giữa 2 bánh xe, tốc độ xoay của robot được tính là:

$$\omega = \frac{v_R - v_L}{W}$$

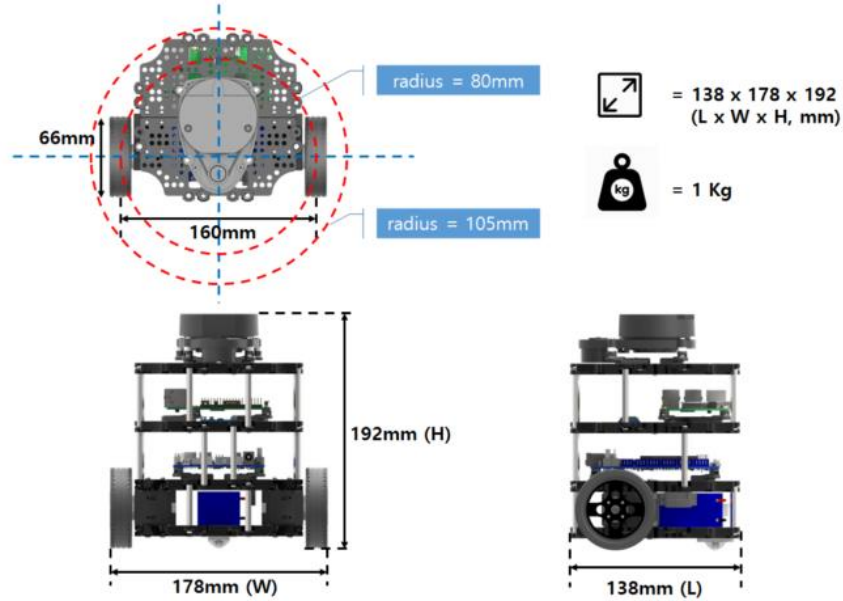
Từ đó ta tốc độ xoay của 2 bánh xe:  $\omega_R$ ,  $\omega_L$  được tính như sau:



$$\begin{bmatrix} \omega_L \\ \omega_R \end{bmatrix} = \frac{1}{R} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{W} & -\frac{1}{W} \end{bmatrix}^{-1} \begin{bmatrix} v \\ \omega \end{bmatrix} \quad (2)$$

Dưới đây là thông số kỹ thuật của mẫu robot tiến hành mô phỏng:

### TurtleBot3 Burger



Hình 5: Thông số của TurtleBot 3 Burger

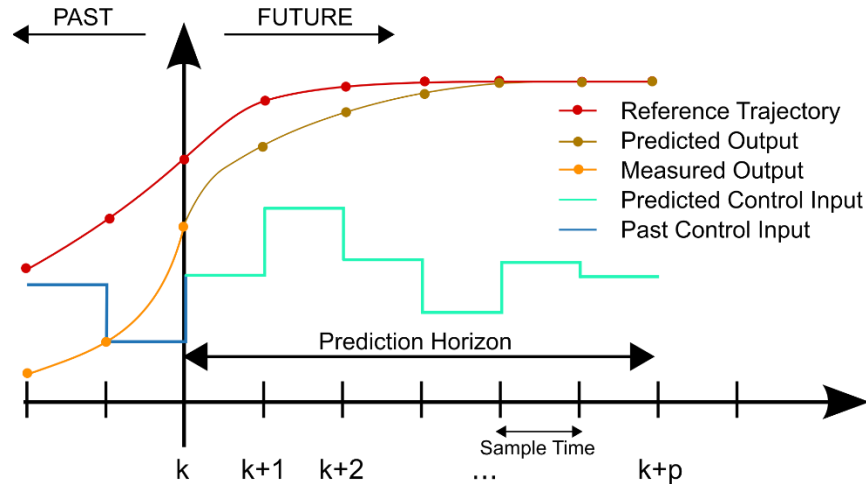
Dựa trên mô hình toán học như trên, chúng tôi tiến hành áp dụng giải thuật MPC vào nhiệm vụ bám quỹ đạo của Robot.

## 2. Giải thuật MPC trong bám quỹ đạo của robot

### a. Bộ điều khiển Model Predictive Control (MPC)

Trong quá trình làm việc với robot, chúng ta luôn quan tâm tới quá trình điều khiển của robot và vị trí của robot. Nhưng thực tế là, trong bài toán bám quỹ đạo này, chúng ta không thể điều khiển được robot bằng các tọa độ (x, y) và vận tốc cho trước được, phải thông qua bước biến đổi trung gian qua mô hình động học như ở phần 1. Có nhiều bộ điều khiển khác nhau để xử lý bài toán này từ đơn giản như PID, LQR... và MPC.

MPC là một bộ điều khiển tối ưu và thường được sử dụng khi chúng ta có một mô hình có sẵn của hệ thống đang được điều khiển. Mục tiêu của bộ điều khiển này là giảm thiểu hàm chi phí được xác định trước trong khi đáp ứng các ràng buộc như động lực học của hệ thống, giới hạn cơ cấu chấp hành, ... Tại mỗi time step, MPC sẽ tính toán một tập hợp các hành động (tín hiệu điều khiển) để giảm thiểu hàm chi phí trong một khoảng thời gian cụ thể và cho hành động cho time step tiếp theo, quy trình này sẽ lặp lại ở time step tiếp theo. Điều này được minh họa ở hình bên dưới:



Hình 6: Minh họa bộ điều khiển MPC

MPC thể hiện ra các ưu điểm so với các phương pháp điều khiển khác:

- Các khái niệm đều trực quan, việc thực thi bộ điều khiển tương đối dễ dàng
  - Áp dụng được cho đa dạng các đối tượng công nghiệp có đặc tính động học đơn giản đến phức tạp
  - Thích hợp có các hệ thống nhiều input nhiều output (MIMO)
  - Có khả năng tự bù trễ
  - Có khả năng sử dụng luật điều khiển tuyến tính cho đối tượng có số lượng đầu vào, ra lớn
  - Đạt được hiệu quả cao nếu quỹ đạo biết trước (trong bài toán bám quỹ đạo)
- Tuy nhiên, vẫn có những nhược điểm như:
- Mô hình dự đoán phải thật chính xác để có thể dự báo trạng thái của quá trình trong miền dự báo.
  - Việc tính toán tín hiệu điều khiển phải thực hiện trực tuyến. Điều này yêu cầu thiết bị phải có năng lực tính toán tương đối tốt.

Như đã đề cập bên trên, bộ điều khiển MPC có mục tiêu là giảm thiểu hàm chi phí trong khi thỏa mãn các hạn chế về động lực học của cơ cấu chấp hành. Điều này được thể hiện qua mặt toán học như sau:

Given:  $\bar{x}_0$

For  $t=0, 1, 2, \dots, T$

- Solve  $\min_{x,u} \sum_{k=t}^T c_k(x_k, u_k)$   
s.t.  $x_{k+1} = f(x_k, u_k), \quad \forall k \in \{t, t+1, \dots, T-1\}$   
 $x_t = \bar{x}_t$
- Execute  $u_t$
- Observe resulting state,  $\bar{x}_{t+1}$

Hình 7: Bộ điều khiển MPC

Với  $x$  là trạng thái của hệ thống,  $u$  là tín hiệu điều khiển/thực thi,  $c$  là hàm chi phí (cost function) và  $f$  là mô hình động của hệ thống. Ngoài ra còn có thêm các

các ràng buộc khác như: trạng thái ban đầu (được sử dụng để tối ưu hóa) bằng trạng thái hiện tại quan sát được, các ràng buộc ban đầu về cơ cấp chấp hành của hệ thống.

Trong hình trên, ta có thể thấy được hàm chi phí và các ràng buộc của hệ thống được tính toán thông qua khoảng thời gian  $T$  (tức là tại mỗi mốc thời gian, quá trình lập kế hoạch lại toàn bộ quỹ đạo), điều này cơ bản không khả thi với hầu hết các trường hợp trong thực tế. Thay vào đó, có một cách tiếp cận thực tế hơn đó là lập kế hoạch cho 1 khoảng thời gian  $H$  cố định. Từ đó, cost function được tính từ  $k = t$  đến  $k = t + H$ . Vì lý do này, MPC còn được gọi là receding horizon control.

Về cơ bản, chúng ta có thể tóm gọn các khâu chính của bộ điều khiển MPC như sau:

- Tùy thuộc vào yêu cầu bài toán, lựa chọn kích thước cho horizon window phù hợp bằng cách chọn độ dài thời gian dự đoán ( $N$ ) và time-step ( $dt$ )
- Sử dụng quỹ đạo tham chiếu để xây dựng phương trình đường cong, từ đó sử dụng để tính toán các sai số (về vị trí và hướng).
- Lựa chọn mô hình động học phù hợp cho hệ thống
- Xác định các ràng buộc của hệ thống (ví dụ như về giới hạn vận tốc, góc xoay, ...)
- Xây dựng hàm chi phí cho bộ giải MPC tối ưu

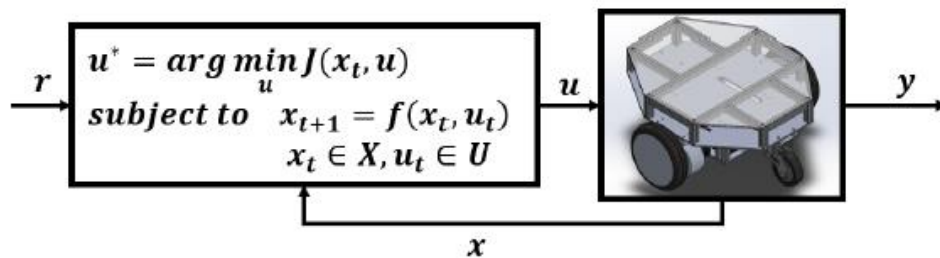
Sau khi xác định được mô hình động học và hàm chi phí tối ưu, chúng ta tiến hành ứng dụng bộ điều khiển MPC vào hệ thống:

- Bước 1: Đặt trạng thái ban đầu của bộ giải MPC là trạng thái hiện tại của robot
- Bước 2: Tiến hành tối ưu hàm chi phí, xây dựng horizon window
- Bước 3: Sử dụng tín hiệu điều khiển đầu tiên cho robot và loại bỏ tất cả các tín hiệu còn lại
- Bước 4: Khởi tạo lại hệ thống tại bước 1

Tiếp theo chúng ta sẽ xây dựng hệ thống điều khiển vòng kín cho robot của chúng ta.

#### **b. Hệ thống điều khiển vòng kín với MPC trên robot điều khiển vi sai**

Trong hệ tọa độ 2 chiều như hình 4, vị trí  $(x, y)$  robot, vận tốc  $v$  và góc nghiêng  $\theta$  được đặt là các biến trạng thái, gia tốc chuyển động  $a$  và vận tốc góc xoay  $\omega$  là đầu vào điều khiển  $u$ . Hệ thống điều khiển vòng kín được mô tả như hình 7.



Hình 8: Hệ thống điều khiển vòng kín với MPC

Để áp dụng MPC cho bài toán bám quỹ đạo, chúng ta xây dựng hai biến trạng thái về hai sai số trong quá trình điều khiển là cross track error  $d$  và orientation error

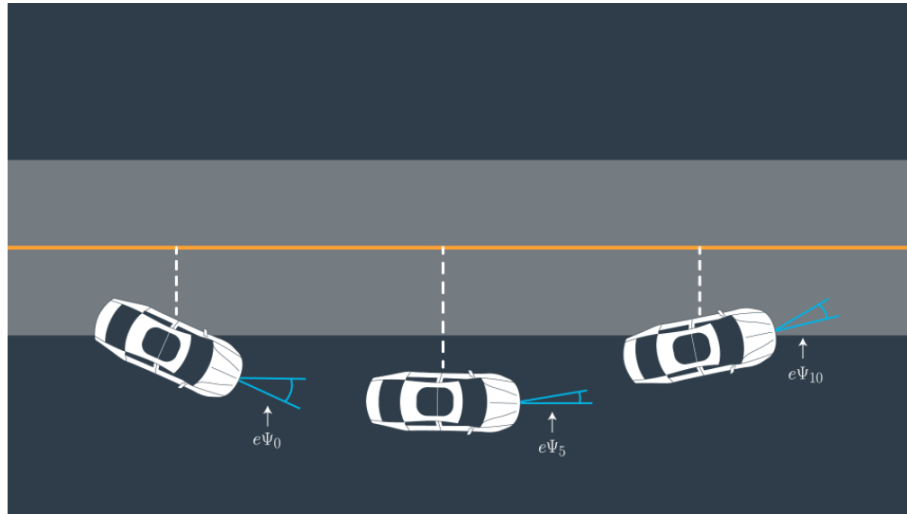
$\eta$ . Trong đó, cross track error biểu thị khoảng cách Euclid giữa vị trí robot (trong quá trình chạy thực tế) và quỹ đạo chỉ định (về mặt lý thuyết) trong hệ tọa độ 2 chiều, orientation error là sự khác biệt giữa hướng tiến của robot và đường tiếp tuyến của đường đi.

Từ đó, với biến trạng thái  $x = (x \ y \ \theta \ v \ d \ \eta)$ , phương trình chuyển động phi tuyến của robot được mô phỏng như một hệ thống rời rạc.

$$\begin{aligned} x_{t+1} &= x_t + v_t \cos(\theta) \cdot dt \\ y_{t+1} &= y_t + v_t \sin(\theta) \cdot dt \\ \theta_{t+1} &= \theta_t + \omega_t \cdot dt \\ v_{t+1} &= v_t + a_t \cdot dt \\ d_{t+1} &= g(x_t) - y_t + v_t \sin(\eta) \cdot dt \\ \eta_{t+1} &= \theta_t - \theta_t^* + \omega_t \cdot dt \end{aligned} \quad (3)$$

Trong đó,  $dt$  là time step,  $g(x_t)$  là hàm cuver fitting với đầu vào  $x_t$  đầu ra  $y$ .  $\theta_t^*$  là độ dốc của hàm  $g(x_t)$ .

Với hàm  $g(x_t)$  này, bước đầu tiên chúng ta cần chuyển đổi các tọa độ của quỹ đạo cần bám vào hệ tọa độ của robot. Sau đó sử dụng một phép ước lượng hàm đa thức bậc 3 nhằm xây dựng phương trình cho đường cong tại khoảng quỹ đạo đang xét (phần này chúng tôi đã tham khảo và thấy được đa thức bậc 3 có thể phù hợp với hầu hết các quỹ đạo, việc sử dụng đa thức bậc thấp hơn thì sai số sẽ lớn còn đa thức bậc cao hơn quá phức tạp để tính toán). Sau khi xây dựng được hàm cuver fitting, chúng tôi sử dụng hàm này để tính hai sai số đã được đề cập bên trên là cross track error và orientation error:



The dashed white line is the cross track error.

Hình 9: Cross track error và orientation error

Từ đó, ta xây dựng được hàm cost function với các ràng buộc như sau:

$$\begin{aligned} \min_{x,u} & \left\{ \sum_{t=0}^N \left[ w_v \|v_t - v_t^{ref}\|^2 + w_d \|d_t\|^2 + w_\eta \|\eta_t\|^2 \right] \right. \\ & + \sum_{t=1}^{N-1} \left[ w_\omega \|\omega_t\|^2 + w_a \|a_t\|^2 \right] \\ & \left. + \sum_{t=2}^N \left[ w_{\dot{\omega}} \|\omega_t - \omega_{t-1}\|^2 + w_a \|a_t - a_{t-1}\|^2 \right] \right\} \end{aligned} \quad (4)$$

Trong đó:

$$x_t = x(t) \quad (5a)$$

$$x_{k+1} = f(x_k, u_k), \forall k \in [t_1, \dots, t_{N-1}] \quad (5b)$$

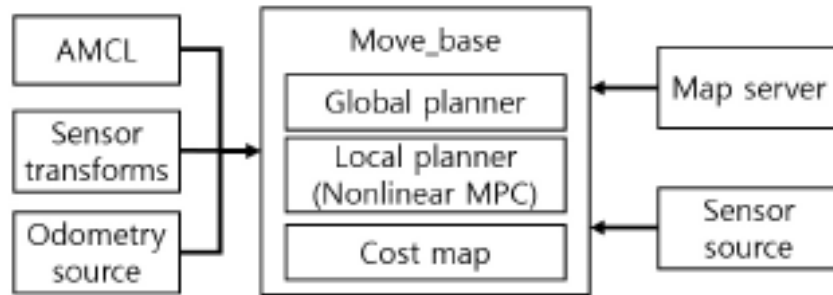
$$x_m \leq x_k \leq x_M, \forall k \in [t_2, \dots, t_N] \quad (5c)$$

$$u_m \leq u_k \leq u_M, \forall k \in [t_1, \dots, t_{N-1}] \quad (5d)$$

Ở đây,  $N$  là chiều dài của không gian dự đoán,  $w_v, w_d, w_\eta, w_\omega, w_{\dot{\omega}}, w_a$  là các trọng số của các biến trạng thái.  $x_m, u_m$  là giới hạn dưới cho các biến đầu vào,  $x_M, u_M$  là giới hạn trên. Phương trình (5a) biểu thị biến trạng thái  $x(t)$  là giá trị ban đầu của phần dự đoán chuyển động trong quá trình tối ưu hóa, (5b) là phương trình chuyển động, hai phương trình (5c) và (5d) là ràng buộc về phạm vi đầu vào của hệ thống. Bài toán này có thể được giải quyết thông qua Quadratic Programming. Trong bài tiểu luận này chúng tôi không trực tiếp xử lý lời giải bài toán này mà sử dụng một thư viện có sẵn trong C++, về phần cơ sở toán học có thể tham khảo link [Quadratic Programming Solver](#).

### 3. Tích hợp giải thuật MPC vào Navigation Stack của ROS

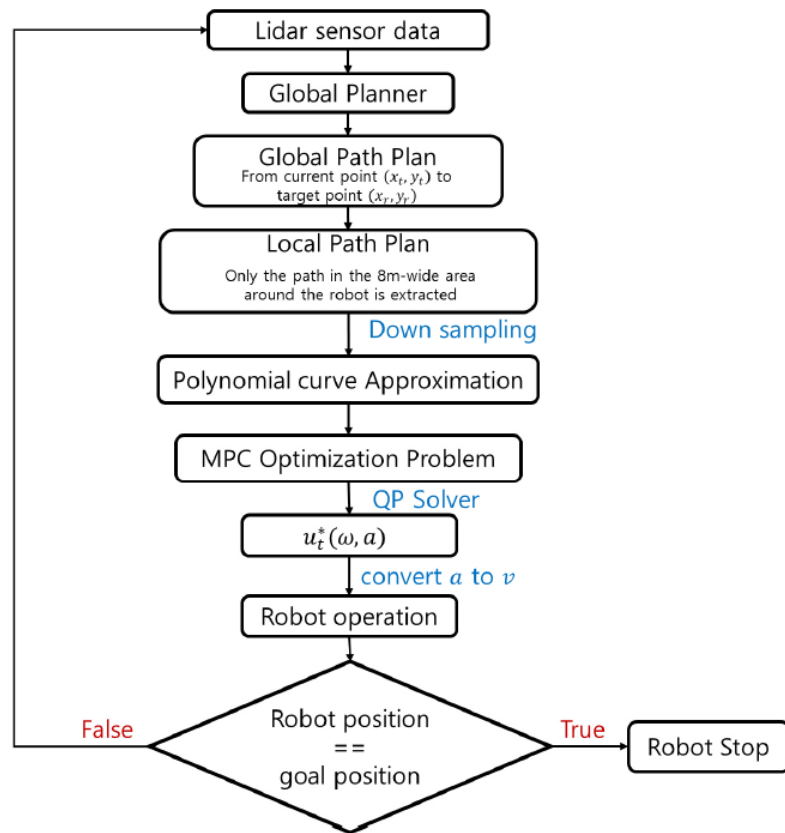
Trong bài tiểu luận này, để quá trình mô phỏng thực hiện một cách dễ dàng, chúng tôi sử dụng Navigation Stack trong ROS để tích hợp giải thuật MPC. Nhờ việc sử dụng Navigation Stack này, chúng ta có thể tận dụng được cơ sở bản đồ và các công cụ Global Planner cho hệ thống, từ đó việc xây dựng quỹ đạo cho trước tốt hơn.



Hình 9: Navigation Stack trong ROS

Trong nghiên cứu này, chúng tôi tập trung vào phần bám quỹ đạo cơ bản của robot, nên sẽ bỏ qua phần nhận biết và tránh vật cản của robot. Trong bài toán này, dựa trên bản đồ có trước, từ vị trí bắt đầu của robot, chúng ta sử dụng giải thuật tìm đường A\* (Global Planner) để tìm đường đi ngắn nhất tới đích, và coi đây là quỹ đạo mà robot cần đi theo. Tiếp đó, Nonlinear MPC (vai trò như Local Planner) sẽ dự đoán ra một quỹ đạo dài 5m từ vị trí bắt đầu của robot từ đó tính ra các giá trị điều khiển của robot (gia tốc và tốc độ xoay). Tốc độ  $v$  sẽ được tính toán thông qua gia tốc và thời gian. Cuối cùng, tốc độ mỗi động cơ sẽ được tính toán qua mô hình động học của robot (trong phần 1).

Giải thuật được mô tả như hình dưới:



Hình 10: Giải thuật MPC trong Navigation Stack

Các trọng số tối ưu hóa của MPC được thiết lập như trong Bảng 1. Ở đây chúng ta tập trung vào các trọng số về lỗi so với quỹ đạo (cross track error và orientation error). Ngoài ra để tránh các sai số do thay đổi hướng đột ngột (lực quán tính) các trọng số về gia tốc góc nghiêng và đột ngột cũng được xem xét. Ở đây chúng tôi đặt chu kỳ tính toán là 0.1s và thời gian dự đoán là 2s ( $N = 20$ ). Lời giải tối ưu cho MPC được giải bằng thư viện Ipopt (Internal Point optimizer).

<b>Weight</b>	<b>Value</b>
Weight for forward velocity error, $w_v$ (m/s <sup>2</sup> )	<b>100</b>
Weight for cross track error, $w_\eta$ (m <sup>-2</sup> )	<b>2000</b>
Weight for orientation error, $w_d$ (rad <sup>-2</sup> )	<b>100</b>
Weight for yaw rate, $w_\omega$ (rad/s <sup>-2</sup> )	<b>0.0</b>
Weight for acceleration, $w_a$ (m/s <sup>2</sup> ) <sup>-2</sup>	<b>0.0</b>
Weight for yaw acceleration, $w \cdot_\omega$ (rad/s <sup>2</sup> ) <sup>-2</sup>	<b>1000</b>
Weight for forward jerk, $w \cdot_a$ (m/s <sup>3</sup> ) <sup>-2</sup>	<b>50</b>

### III. Mô phỏng giải thuật điều khiển MPC trên Gazebo

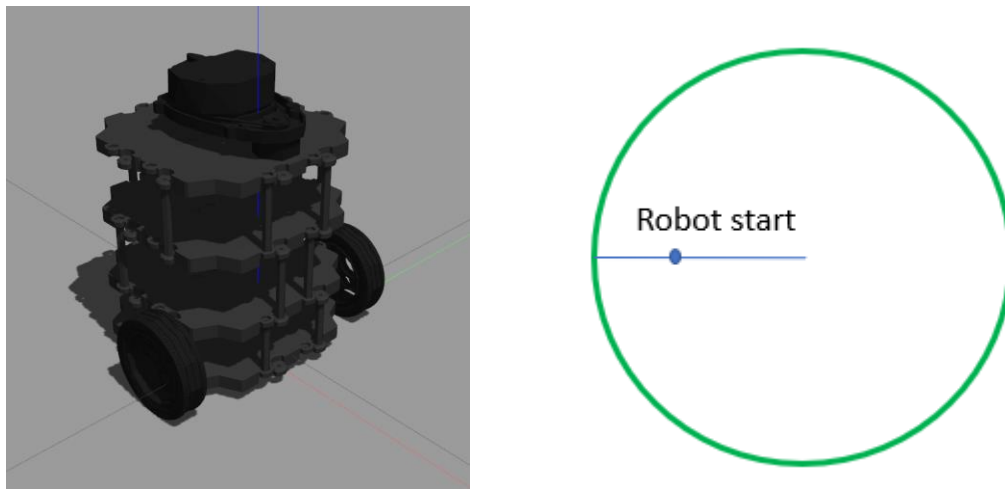
Trong phần mô phỏng này, chúng tôi tiến hành hai bài mô phỏng. Bài đầu tiên là chúng tôi xây dựng một mô hình robot điều khiển vi sai, và sử dụng MPC để điều khiển robot đi theo một quỹ đạo cho trước. Bài mô phỏng thứ hai, chúng tôi tích hợp MPC trở thành một Local Planner trong Navigation Stack của ROS và mô phỏng TurtleBot 3 bám theo quỹ đạo được tạo ra bởi Global Planner.

Thông số cơ bản của bài mô phỏng:

- Hệ điều hành Ubuntu 20.04
- ROS version: Noetic
- Gazebo version 11
- CPU: Intel Xeon E5-2620v4
- GPU: NVIDIA GTX 1080Ti
- RAM: 16GB

#### 1. Mô phỏng robot điều khiển vi sai bám quỹ đạo sử dụng MPC

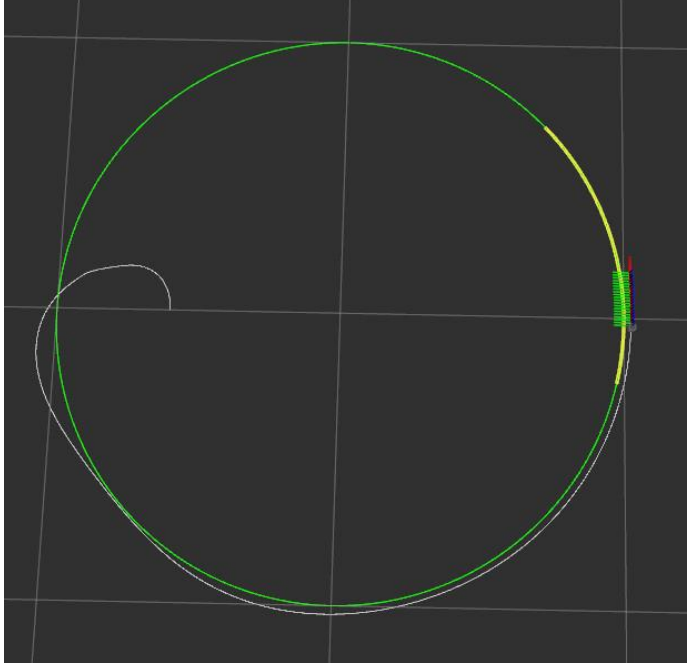
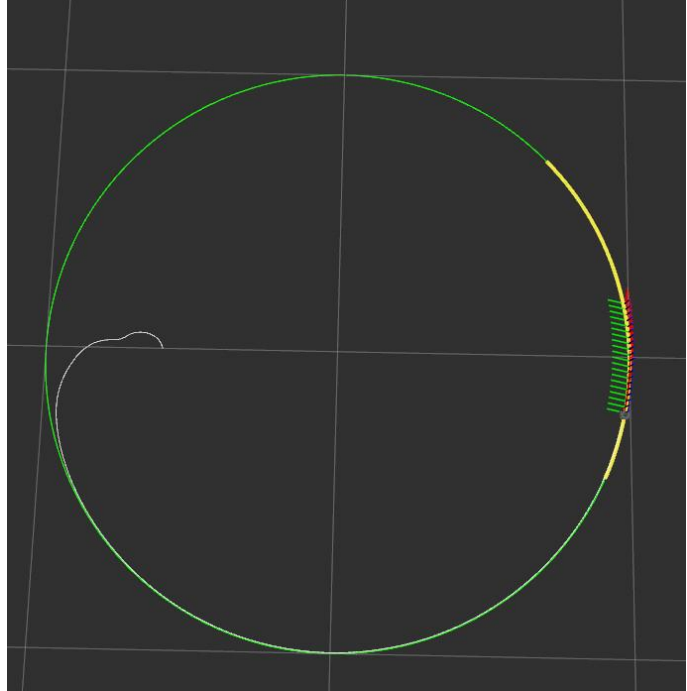
Trong bài mô phỏng này, chúng tôi sử dụng model TurtleBot 3 Burger:



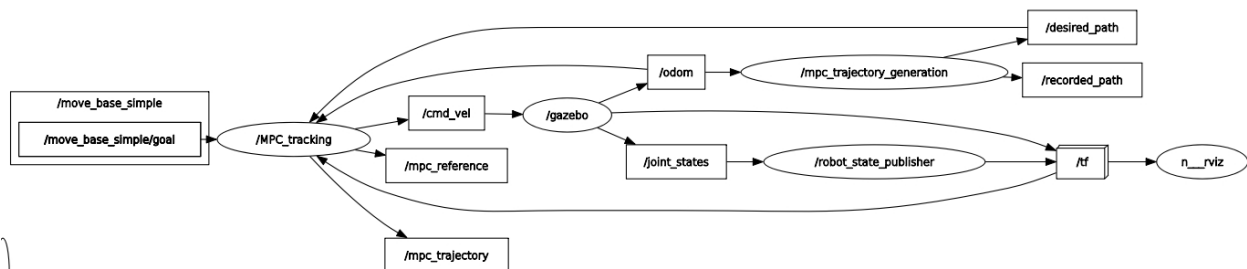
Hình 11: (a) TurtleBot 3 trên Gazebo, (b) Quỹ đạo mô phỏng của robot (hình tròn với  $r = 5$ , điểm xanh là điểm xuất phát của robot)

Dưới đây là kết quả mô phỏng với hai bộ tham số 1 và 2:

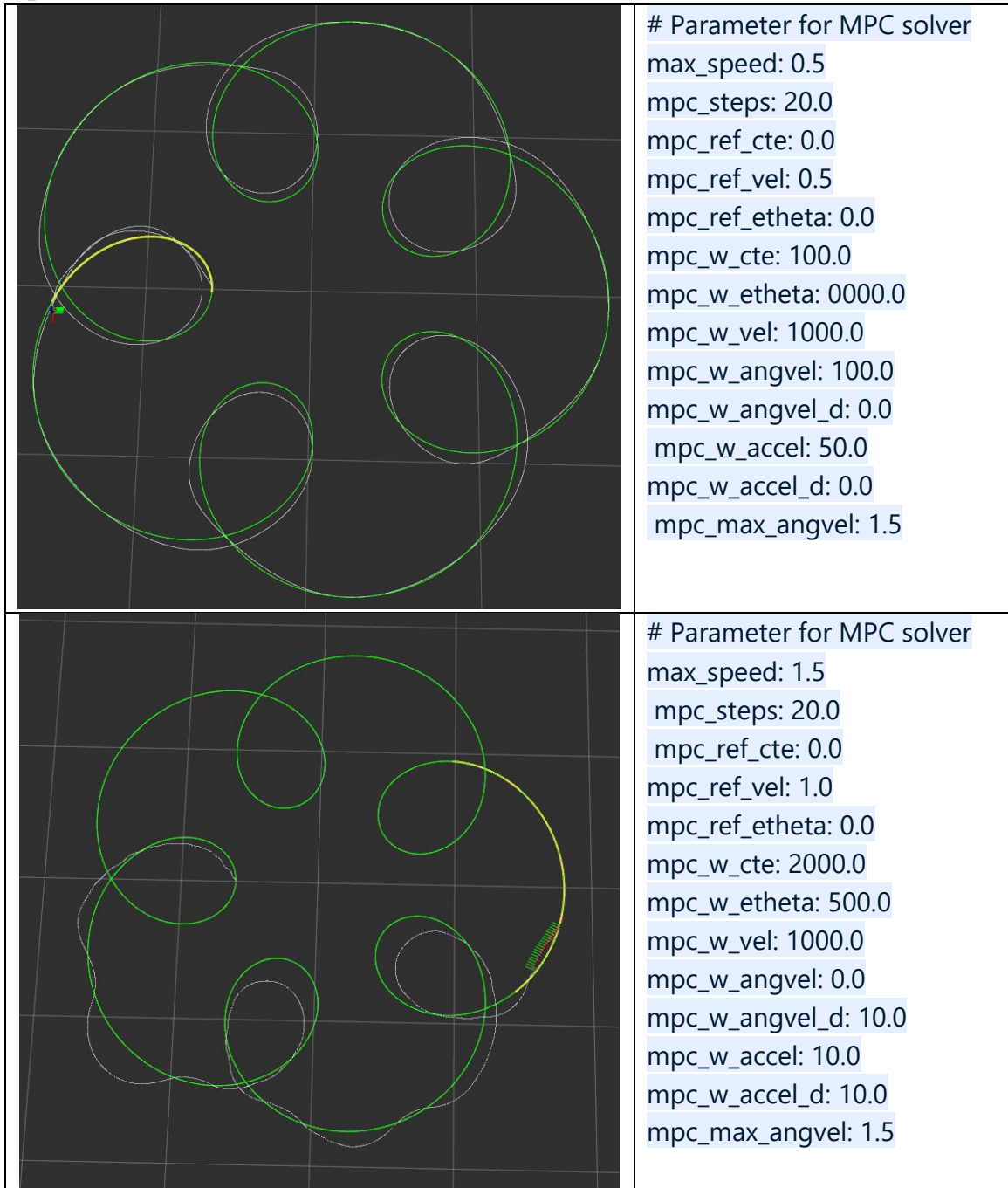


	<pre># Parameter for MPC solver max_speed: 0.5 mpc_steps: 20.0 mpc_ref_cte: 0.0 mpc_ref_vel: 0.5 mpc_ref_etheta: 0.0 mpc_w_cte: 100.0 mpc_w_etheta: 0000.0 mpc_w_vel: 1000.0 mpc_w_angvel: 100.0 mpc_w_angvel_d: 0.0 mpc_w_accel: 50.0 mpc_w_accel_d: 0.0 mpc_max_angvel: 1.5</pre>
	<pre># Parameter for MPC solver max_speed: 1.5 mpc_steps: 20.0 mpc_ref_cte: 0.0 mpc_ref_vel: 1.0 mpc_ref_etheta: 0.0 mpc_w_cte: 2000.0 mpc_w_etheta: 500.0 mpc_w_vel: 1000.0 mpc_w_angvel: 0.0 mpc_w_angvel_d: 10.0 mpc_w_accel: 10.0 mpc_w_accel_d: 10.0 mpc_max_angvel: 1.5</pre>

Hệ thống node trong ROS:



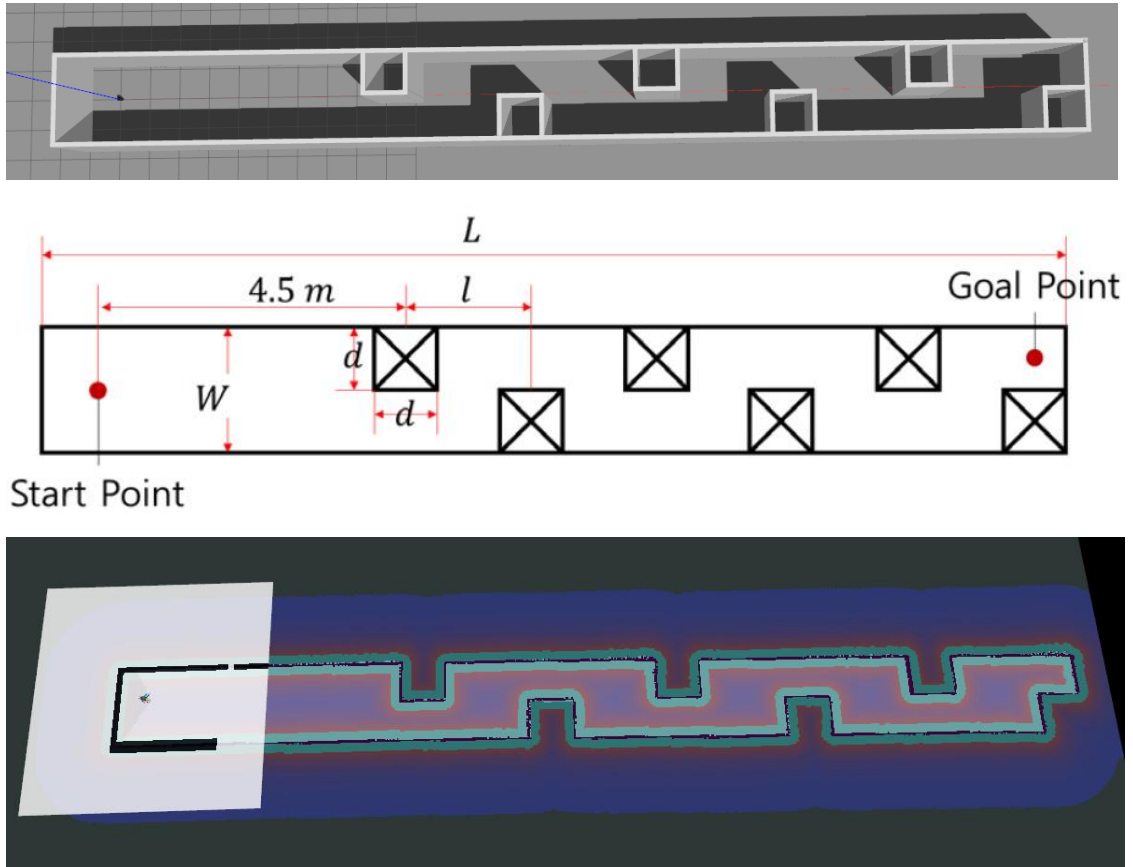
Cùng với bộ tham số như trên, chúng tôi thử nghiệm với quỹ đạo phức tạp hơn:



Chúng ta có thể thấy được rằng, dù cùng một bộ tham số, kết quả giữa hai loại quỹ đạo rất khác nhau. Điều này xảy ra có thể là do độ phức tạp của quỹ đạo các khúc cong là cho việc vượt lỗi quán tính của robot.

## 2. Mô phỏng TurtleBot 3 bám quỹ đạo sử dụng Gazebo

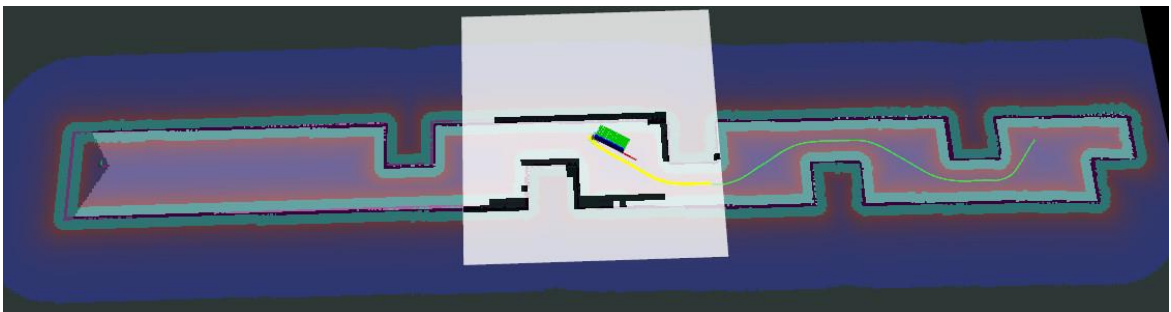
Trong bài mô phỏng thứ 2 này, chúng tôi sử dụng đã xây dựng MPC trở thành một Local Planner của Navigation Stack. Model robot thử nghiệm vẫn là TurtleBot 3 Burger với bản đồ như bên dưới:



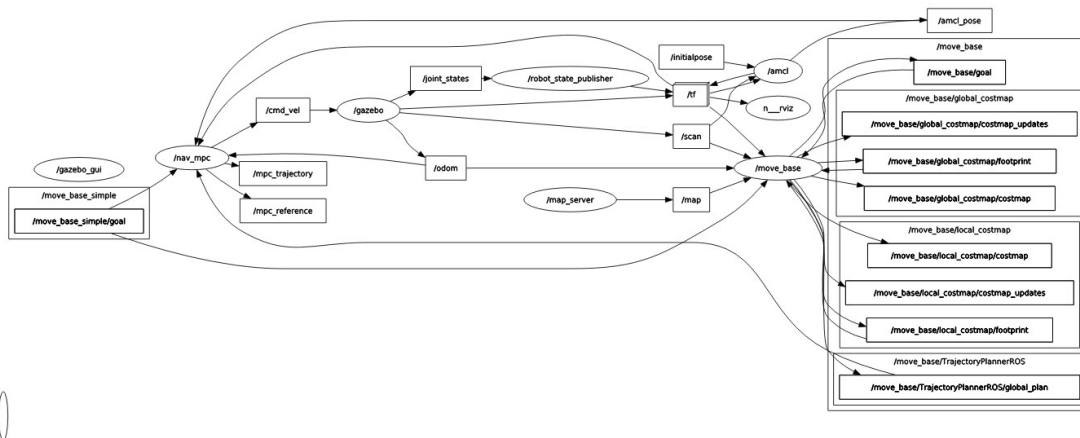
Hình 12: Mô trường thực nghiệm

Với  $L = 34(\text{m})$ ,  $l = 3(\text{m})$ ,  $W = 2.5(\text{m})$ ,  $d = 1.25(\text{m})$

Dưới đây là kết quả mô phỏng:

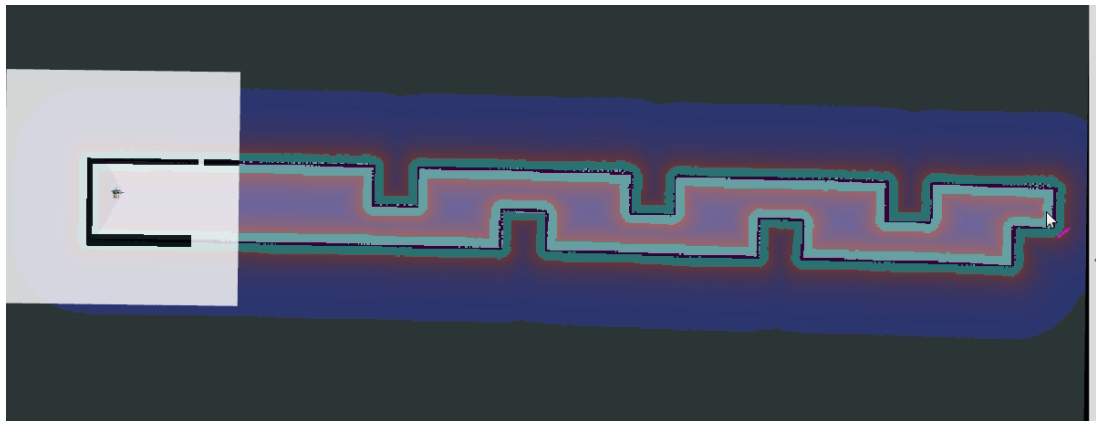


Hình 13: Kết quả mô phỏng

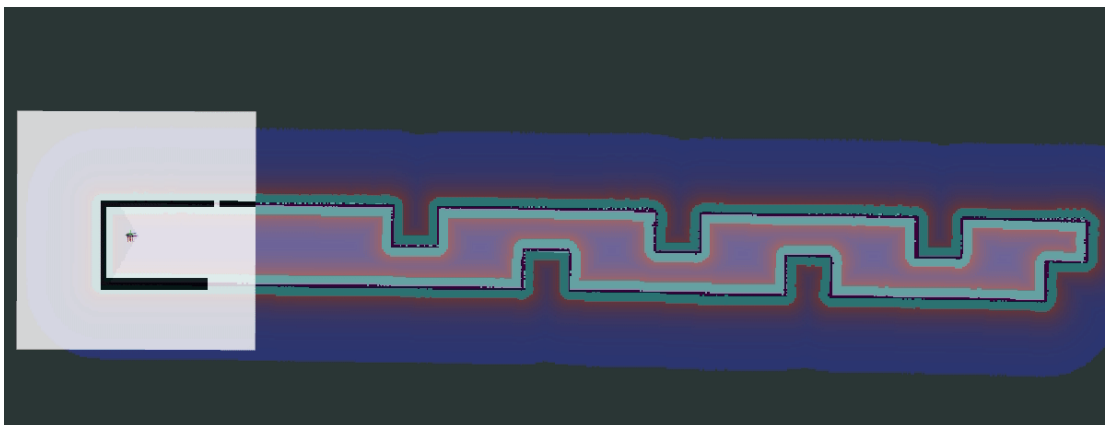


Hình 14: Hệ thống các node trong ROS

Chúng tôi thực hiện mô phỏng với hai bộ tham số tương tự trong bài mô phỏng 1 và thu được kết quả thời gian tới đích của bộ tham số 1 là 70s và bộ tham số 2 là 35s (vì sự thay đổi vận tốc max của robot).

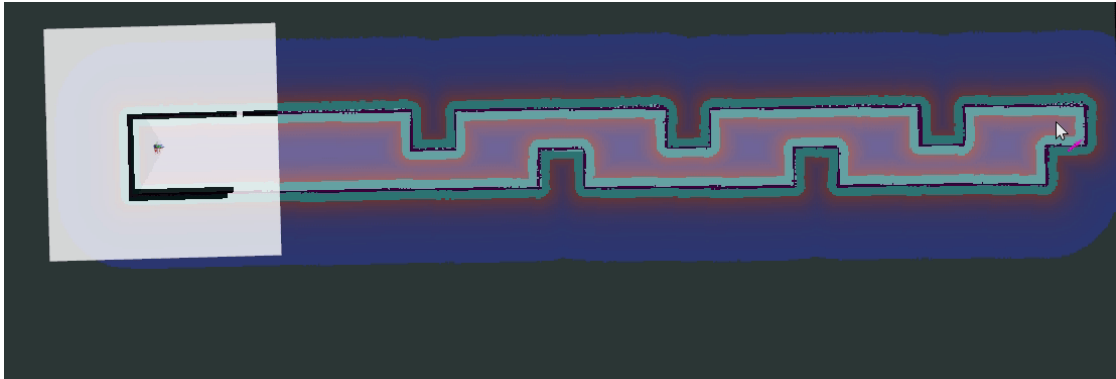


Hình 15: Kết quả bộ tham số 1

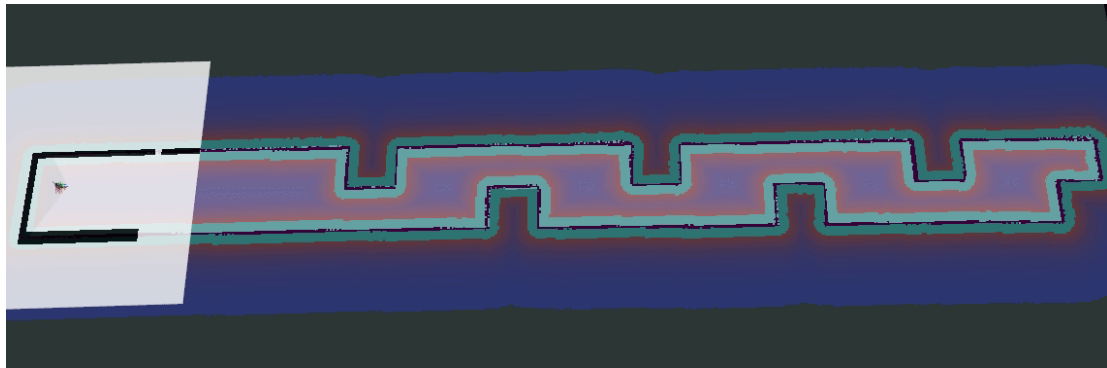


Hình 16: Kết quả bộ tham số 2

Chúng tôi cũng thử nghiệm so sánh với một giải thuật path following khác là Pure Pursuit.

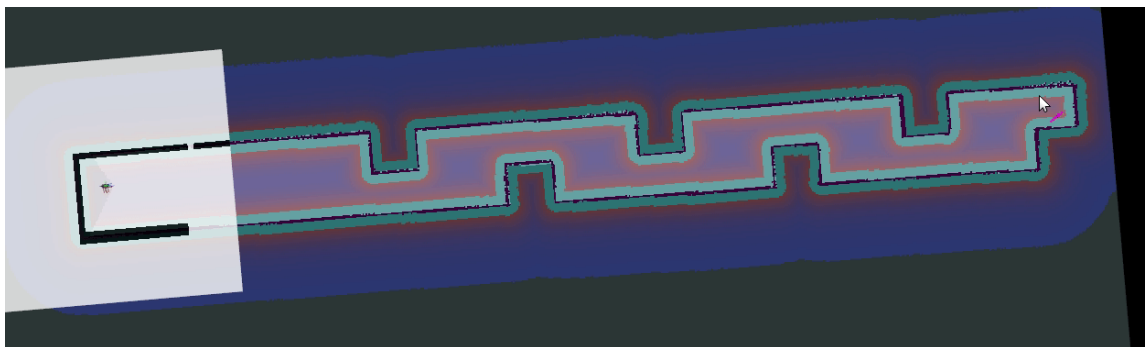


Hình 17: MPC và quỹ đạo cho trước



Hình 18: Pure Pursuit và quỹ đạo cho trước

Ngoài ra trên bản đồ này, chúng tôi cũng thử nghiệm giải thuật có sẵn của Navigation Stack là Dynamic Window Approach (DWA) nhưng không thành công, có thể là do bộ tham số chưa tối ưu.



Hình 20: Thử nghiệm giải thuật DWA

#### **IV. Kết luận**

Trong bài tiểu luận này, chúng tôi đã nghiên cứu và ứng dụng giải thuật điều khiển MPC vào việc bám quỹ đạo của robot TurtleBot 3. Nghiên cứu cho thấy khả năng tích hợp bộ điều khiển MPC vào hệ thống là khả thi. Tuy nhiên cần phải cải thiện thêm nhiều chức năng như nhận diện, tránh vật cản, việc điều chỉnh tham số phù hợp cho hệ thống. Trong các nghiên cứu tiếp theo, chúng tôi có thể tích hợp một số kỹ thuật tối ưu sử dụng học máy để quá trình điều chỉnh tham số được dễ dàng hơn đồng thời tiến hành các thử nghiệm thực tế cho hệ thống.