HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN I

—————o0o—————

BÀI TẬP LỚN MÔN TOÁN RỜI RẠC 2

Đề tài: **"THUẬT TOÁN BATCH-INFORMED RRT* (BIT*)"**

Giảng viên hướng dẫn:	TS. Nguyễn Kiều Linh
Nhóm thực hiện:	Nhóm 1
Lớp:	Toán rời rạc 2 - Nhóm 11
Niên khóa:	2023-2028
Hệ đào tạo:	Đại học chính quy

Hà Nội, 29/04/2025

NHẬN XÉT CỦA GIẢNG VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:)**

Hà Nội, ngày tháng năm 20...

Giảng viên hướng dẫn

NHẬN XÉT CỦA GIẢNG VIÊN PHẢN BIỆN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

Điểm: **(Bằng chữ:)**

Hà Nội, ngày tháng năm 20...

Giảng viên phản biện

Mục lục

1 Mở đầu 1

1.1 Giới thiệu 1

1.2 Thành viên nhóm và vai trò 2

2 Kiến thức chuẩn bị 3

2.1 Các thuật toán liên quan 3

2.1.1 Thuật toán RRT và RRT* 3

2.1.2 Thuật toán BIT* 5

2.1.3 Mối liên hệ giữa thuật toán BIT* và các thuật toán khác . 5

3 Mô phỏng và đánh giá thuật toán BIT* 10

3.1 Mã giả 10

3.2 Cài đặt và công cụ thực nghiệm 11

3.3 Mã nguồn triển khai 13

3.4 Phân tích và đánh giá kết quả 30

3.4.1 Kết quả 30

3.4.2 Phân tích 30

3.4.3 Nhận xét 32

4 Ứng dụng thuật toán 33

4.1 Ứng dụng thuật toán BIT* trong nhận dạng 33

4.2 Ứng dụng thuật toán BIT* trong tối ưu hóa 34

Kết luận 36

Tài liệu tham khảo 37

Danh sách hình vẽ

2.1 Minh họa RRT và RRT* 4
2.2 So sánh hiệu suất giữa RRT* và BIT* 6
2.3 Mô tả RRT* và BIT* 7

3.1 Mô phỏng tại khung hình thứ 3 30
3.2 Mô phỏng tại khung hình thứ 5 30
3.3 Mô phỏng tại khung hình thứ 7 30
3.4 Mô phỏng tại khung hình thứ 16 30

Danh sách bảng

2.1 Bảng so sánh RRT và RRT* 4
2.2 So sánh RRT với BIT* 6
2.3 So sánh các thuật toán RRT, RRT* và BIT* 8
2.4 So sánh BIT*, RRG và PRM 9

Chương 1

Mở đầu

1.1 Giới thiệu

Trong những năm gần đây, việc lập kế hoạch đường đi (path planning) cho robot và các hệ thống tự động trở thành một trong những chủ đề nghiên cứu quan trọng trong lĩnh vực trí tuệ nhân tạo và điều khiển. Đặc biệt, trong các môi trường phức tạp, không gian liên tục và có nhiều ràng buộc, bài toán lập kế hoạch hiệu quả, tối ưu và có khả năng mở rộng là một thách thức lớn. Các thuật toán lấy mẫu như RRT và RRT* đã được sử dụng rộng rãi nhờ tính mở rộng tốt và khả năng tìm kiếm nghiệm khả thi nhanh chóng. Tuy nhiên, chúng thường gặp khó khăn trong việc tìm ra nghiệm tốt một cách hiệu quả, đặc biệt trong không gian trạng thái có độ phức tạp cao hoặc chiều cao lớn. Trong khi đó, các phương pháp tìm kiếm trên đồ thị như A* lại tỏ ra hiệu quả trong không gian rời rạc, nhưng không phù hợp để áp dụng trực tiếp vào không gian liên tục. Thuật toán Batch-Informed RRT* (BIT*) ra đời như một giải pháp kết hợp ưu điểm của hai thế giới: tính có thứ tự của các thuật toán tìm kiếm dựa trên đồ thị và tính linh hoạt, hiệu quả mở rộng của các thuật toán dựa trên lấy mẫu. BIT* hoạt động bằng cách nhận ra rằng một tập hợp mẫu có thể tạo thành một đồ thị hình học ngẫu nhiên (Random Geometric Graph – RGG) ẩn, từ đó thực hiện tìm kiếm heuristic có thứ tự trên không gian liên tục một cách hiệu quả và có khả năng tối ưu bất tiệm cận. BIT* còn nổi bật với khả năng xử lý theo lô mẫu (batch), cho phép nó tái sử dụng thông tin từ các lần tìm kiếm trước, nâng cao tốc độ hội tụ đến nghiệm tối ưu, đồng thời duy trì hiệu suất theo thời gian thực. Nhờ các đặc điểm đó, BIT* trở thành một trong những thuật toán mạnh mẽ và hiện đại nhất trong lĩnh vực lập kế hoạch đường đi hiện nay.

1.2 Thành viên nhóm và vai trò

- **Thuật toán:**
  - **Nguyễn Thế Lưu - B23DCCN520**: Giới thiệu tổng quan về thuật toán BIT* và lý do cần thiết so với RRT, RRT*.
  - **Phạm Duy Cường - B23DCCN113**: Trình bày ý tưởng tổng quát và những cải tiến của BIT* so với RRT*.
  - **Phạm Đức Tuấn - B23DCCN892**: Phân tích chi tiết các bước hoạt động của BIT.
  - **Nguyễn Tuấn Anh - B23DCCN039**: Mã giả, giải thích cụ thể các bước và minh họa thuật toán.
  - **Nguyễn Đình Trà - B23DCCN836**: So sánh BIT* với các thuật toán khác như.
- **Báo cáo:**
  - **Nguyễn Đại Nghĩa - B23DCCN598**: Xây dựng cấu trúc báo cáo, tạo file Latex chính với định dạng chuẩn.
  - **Hoàng Trung Kiên - B23DCCN459**: Viết phần mở đầu, tổng quan lý thuyết, trích dẫn tài liệu.
  - **Nguyễn Hải Long - B23DCCN503**: viết phần mô tả chi tiết thuật toán, biểu đồ và ví dụ minh họa.
  - **Nguyễn Tuấn Anh - B23DCCN039**: Tổng hợp kiến thức toán học, kiểm tra lỗi trích dẫn tài liệu.

Chương 2

Kiến thức chuẩn bị

2.1 Các thuật toán liên quan

2.1.1 Thuật toán RRT và RRT*

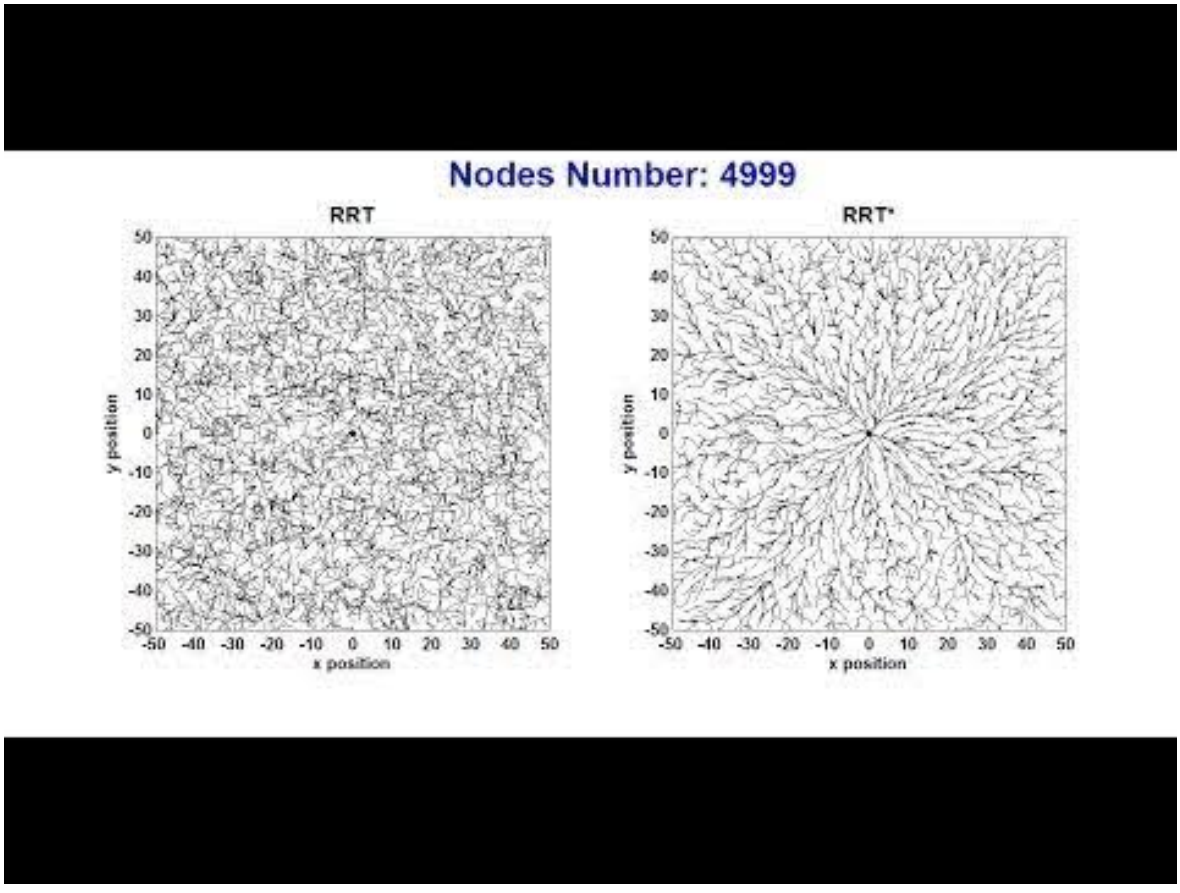
RRT và RRT* là các thuật toán cơ sở để xây dựng lên thuật toán BIT*. Do vậy giữa chúng có nhiều điểm tương đồng và cũng có những cải tiến riêng.

- RRT : thuật toán tìm đường đi bằng cách thêm các mẫu ngẫu nhiên trong không gian tìm kiếm vào cây.
- RRT* : Cải tiến từ thuật toán RRT, sử dụng hàm heuristic để ước lượng khoảng cách từ nút đến đích. Nếu 2 nhánh cùng đường đi, chỉ chọn nhánh có khoảng cách nhỏ hơn. Nhờ đó loại bỏ được các nhánh không tối ưu.

RRT so với RRT*

Thuật toán	**RRT (Rapidly-exploring Random Tree)**	**RRT* (RRT Star)**
Cách hoạt động	Xây dựng cây bằng cách mở rộng từ điểm bắt đầu, thêm vào cây các mẫu ngẫu nhiên trong không gian.	Cải thiện RRT bằng cách sử dụng heuristic để hội tụ dần đến đường đi tối ưu.
Tối ưu hóa	Không đảm bảo tối ưu.	Đảm bảo hội tụ về lời giải tối ưu khi số mẫu tiến tới vô hạn.
Ưu điểm	Khám phá nhanh không gian chỉ với ít mẫu.	Đường đi tốt hơn, hội tụ khi mẫu đủ nhiều.
Nhược điểm	Đường đi thường không tối ưu.	Tốn thời gian và tài nguyên hơn RRT.

Bảng 2.1: Bảng so sánh RRT và RRT*



Hình 2.1: Minh họa RRT và RRT*

2.1.2 Thuật toán BIT*

Giới thiệu chung về thuật toán BIT*

Thuật toán BIT* là một thuật toán lập kế hoạch đường đi dựa trên việc kết hợp các ưu điểm của cả phương pháp tìm kiếm trên đồ thị và phương pháp dựa trên lấy mẫu. BIT* hoạt động bằng cách nhận ra rằng một tập hợp các mẫu mô tả một đồ thị hình học ngẫu nhiên (RGG) ẩn. Điều này cho phép BIT* kết hợp hiệu quả tìm kiếm có thứ tự của các kỹ thuật dựa trên đồ thị, như A*, với khả năng mở rộng theo thời gian thực của các thuật toán dựa trên lấy mẫu, như RRT.

Ý tưởng của thuật toán BIT*

- Các thuật toán dựa trên lấy mẫu như RRT* có tính tối ưu bất tiệm cận nhưng có thể chậm trong việc tìm ra nghiệm tốt, đặc biệt là trong không gian trạng thái phức tạp hoặc chiều cao.
- Các phương pháp tìm kiếm trên đồ thị như A* hiệu quả khi không gian rời rạc hoặc được rời rạc hóa tốt, nhưng khó áp dụng trực tiếp cho không gian liên tục.
- BIT* tận dụng lợi thế của việc xử lý các mẫu theo lô (batch) để thực hiện tìm kiếm có thứ tự trên không gian lập kế hoạch liên tục, đồng thời duy trì hiệu suất theo thời gian thực.
- BIT* sử dụng hàm heuristic để hướng dẫn hiệu quả việc tìm kiếm trong một loạt các RGG ẩn ngày càng dày đặc, đồng thời tái sử dụng thông tin từ các lần tìm kiếm trước.

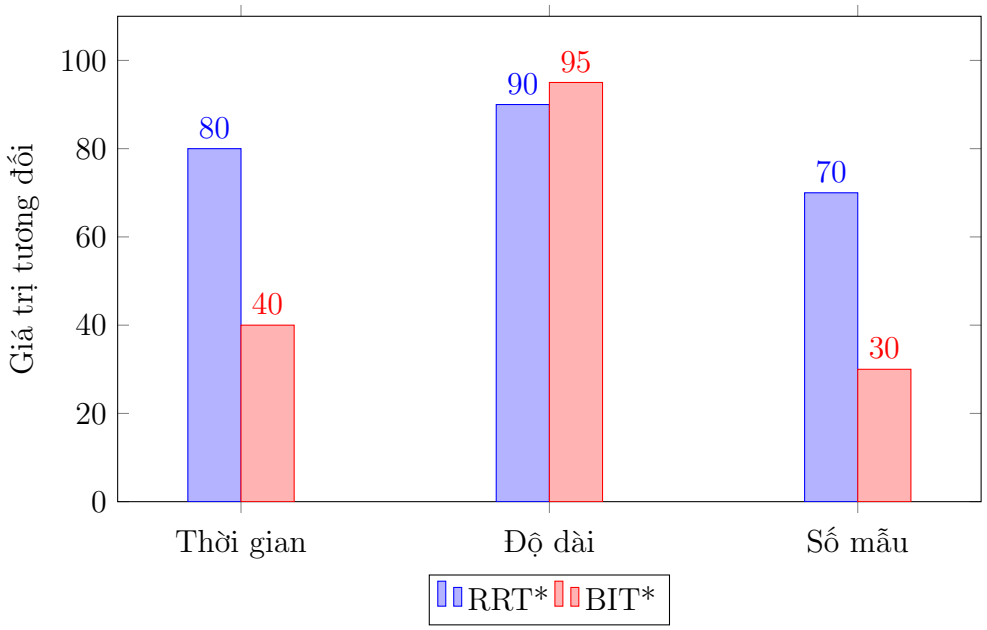
2.1.3 Mối liên hệ giữa thuật toán BIT* và các thuật toán khác

RRT* với BIT*

Như đã nói ở trên, thuật toán RRT* rất hữu ích trong việc tìm kiếm đường đi tối ưu đến đích. Tuy nhiên để đạt đường đi tối ưu này thì phải lấy rất nhiều mẫu, tốn nhiều thời gian và tài nguyên bộ nhớ.

Một giải pháp tối ưu hơn là thuật toán BIT*, phương pháp này vẫn kế thừa từ RRT* nhưng sử dụng thêm phương pháp lấy mẫu theo lô. BIT* thu thập một nhóm các mẫu (lô) và xử lý chúng cùng một lúc.

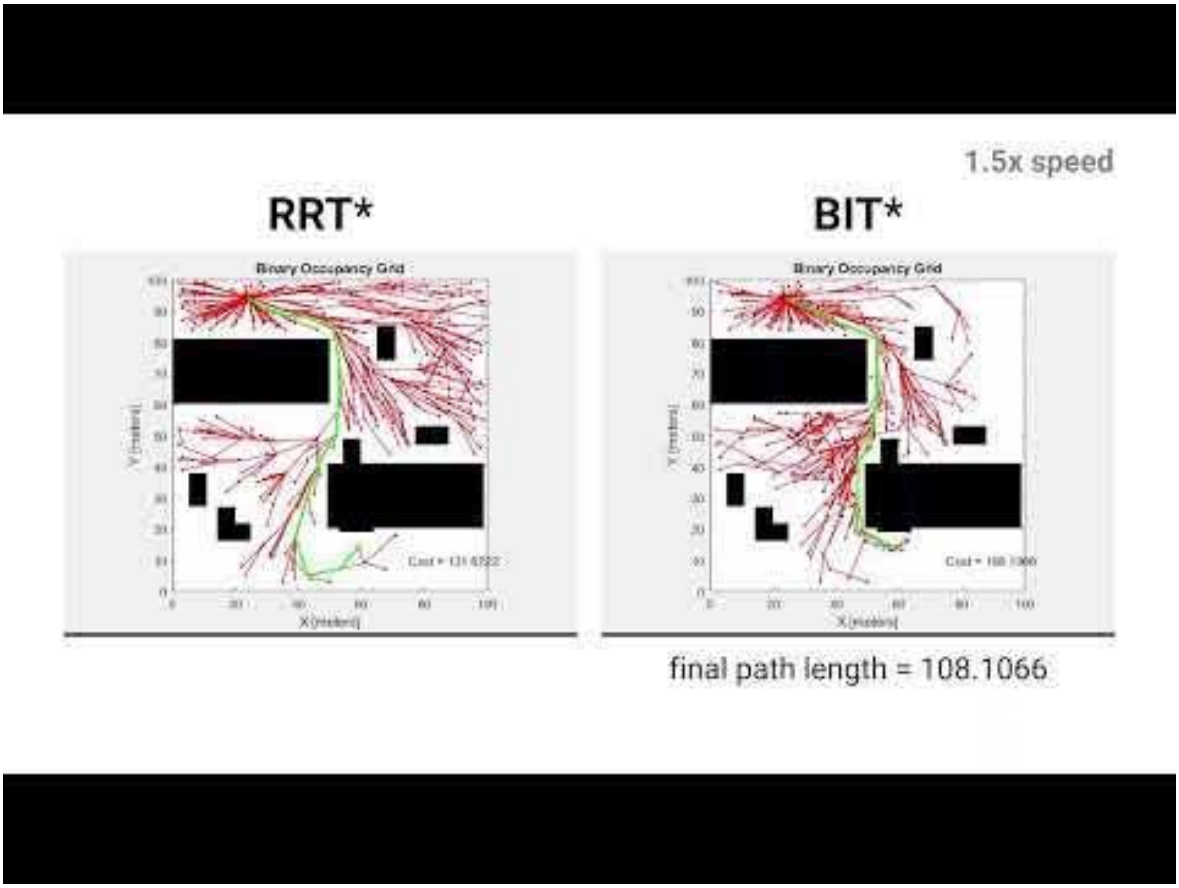
Phương pháp lấy mẫu theo lô giúp giới hạn không gian tìm kiếm. Điều này có nghĩa là thay vì tìm kiếm toàn bộ không gian, thuật toán tập trung vào các khu vực có khả năng chứa đường đi tối ưu.



Hình 2.2: So sánh hiệu suất giữa RRT* và BIT*

Thuật toán	**RRT (Rapidly-exploring Random Tree Star)***	**BIT (Batch Informed Trees)***
Cách hoạt động	Đã nói ở trên.	Kế thừa toàn bộ ưu điểm từ RRT*. Sử dụng thêm phương pháp phân lô và giới hạn không gian tìm kiếm giúp tìm kiếm đường đi tối ưu mà không phải lấy quá nhiều mẫu như RRT*
Ưu điểm	Tìm kiếm đường đi gần tối ưu, đảm bảo hội tụ đến đường đi tối ưu khi số lượng điểm mẫu tăng lên	Xử lý theo lô giúp tận dụng thông tin từ nhiều mẫu cùng lúc, tăng tốc độ tìm kiếm. Giới hạn không gian tìm kiếm bằng cách chỉ chọn các mẫu trong các khu vực có tiềm năng cao
Nhược điểm	Tính toán phức tạp hơn so với RRT, yêu cầu nhiều tài nguyên và thời gian hơn.	Khó xây dựng thuật toán heuristic và điều chỉnh tham số cẩn thận.

Bảng 2.2: So sánh RRT với BIT*



Hình 2.3: Mô tả RRT* và BIT*

Tổng kết

Thuật toán	**Đặc trưng**	**Ưu điểm**	**Nhược điểm**
RRT	Tìm đường đi bằng cách thêm các mẫu ngẫu nhiên trong không gian tìm kiếm vào cây.	Tìm kiếm được đường đi trong thời gian ngắn.	Đường đi tìm được có thể không phải là tối ưu.
RRT*	Cải tiến từ thuật toán RRT, sử dụng hàm heuristic để ước lượng khoảng cách từ nút đến đích.	Tìm được đường đi tối ưu.	Độ phức tạp thời gian, không gian cao.
BIT*	Cải tiến từ thuật toán RRT*, sử dụng phương pháp phân lô và giới hạn không gian để tăng hiệu quả.	Tìm được đường đi tối ưu với độ phức tạp thời gian, không gian thấp.	Thuật toán rất phức tạp.

Bảng 2.3: So sánh các thuật toán RRT, RRT* và BIT*

BIT* so với các thuật toán lấy mẫu khác

Thuật toán	Mô tả	Ưu điểm	Nhược điểm
BIT* (Batch Informed Trees)	Phát triển từ thuật toán RRT, sử dụng phương pháp phân lô và giới hạn phạm vi để tăng hiệu suất.	- Tối ưu hóa theo lô, giảm số lần kiểm tra. - Hiệu quả trong không gian lớn và phức tạp.	- Cần nhiều tài nguyên tính toán. - Phụ thuộc vào chất lượng của mẫu ban đầu.
RRG (Rapidly-exploring Random Graph)	Mở rộng từ RRT, tạo ra đồ thị thay vì cây để đảm bảo tính tối ưu tiệm cận.	- Đảm bảo tính tối ưu tiệm cận. - Khả năng tìm kiếm toàn diện hơn so với RRT.	- Phức tạp hơn RRT. - Cần nhiều bộ nhớ hơn do lưu trữ đồ thị.
PRM (Probabilistic Roadmap Method)	Kết hợp thuật toán lấy mẫu ngẫu nhiên để tạo đồ thị và sử dụng các thuật toán đồ thị để tìm đường đi.	- Hiệu quả trong không gian lớn. - Linh hoạt, dễ điều chỉnh số lượng điểm ngẫu nhiên.	- Phụ thuộc vào số lượng điểm ngẫu nhiên. - Đường đi có thể không tối ưu nếu số lượng điểm không đủ.

Bảng 2.4: So sánh BIT*, RRG và PRM

Chương 3

Mô phỏng và đánh giá thuật toán BIT*

3.1 Mã giả

* Đầu vào

- $x_{\text{start}}, x_{\text{goal}}$: điểm bắt đầu và đích.
- max_iter: số vòng lặp tối đa.
- m: số mẫu mỗi batch.

Khởi tạo

$$V \leftarrow \{x_{\text{start}}\} \quad \text{(Các đỉnh đã duyệt)}$$
$$E \leftarrow \emptyset \quad \text{(Cạnh của cây tìm kiếm)}$$
$$X_{\text{samples}} \leftarrow \{x_{\text{goal}}\} \quad \text{(Mẫu khởi tạo chứa đích)}$$
$$g(x_{\text{start}}) \leftarrow 0$$
$$c_{\text{best}} \leftarrow \infty \quad \text{(Chi phí tốt nhất chưa biết)}$$
$$Q_{\text{vertex}} \leftarrow \text{priority queue theo } g(v) + h(v)$$
$$Q_{\text{edge}} \leftarrow \text{priority queue theo } g(v) + c(v, x) + h(x)$$

Vòng lặp chính

Lặp đến max_iter:

- Nếu Q_{edge} và Q_{vertex} rỗng:
  - **PRUNE**: Loại bỏ các điểm không còn khả năng cải thiện lời giải.
  - Lấy m mẫu mới bằng `SampleInEllipsoid`.
  - Đưa lại các đỉnh V vào hàng đợi Q_{vertex}.
  - Tính bán kính kết nối r bằng `ComputeRadius`.

Mở rộng đỉnh

Lấy đỉnh tốt nhất v từ Q_{vertex}:

- Với mỗi mẫu x trong bán kính r quanh v:
  - Nếu có khả năng cải thiện ($g(v) + h(x) < c_{\text{best}}$), thì thêm cạnh (v, x) vào Q_{edge}.
- Với mỗi đỉnh w trong bán kính r quanh v (nếu v là mới):
  - Nếu (v, w) chưa tồn tại và có khả năng rewiring tốt hơn thì thêm vào Q_{edge}.

Xử lý cạnh

Lấy cạnh tốt nhất (v, x) từ Q_{edge}:

- Nếu không va chạm (`collisionFree`):

$$g_{\text{new}} \leftarrow g(v) + \text{cost}(v, x)$$

- Nếu x là điểm mới:
  - Thêm vào V, cập nhật $g(x)$, thêm (v, x) vào E.
  - Nếu $x = x_{\text{goal}}$, cập nhật c_{best}.
- Nếu x đã có trong cây nhưng đường đi mới tốt hơn:
  - Rewire: thay cạnh cũ bằng cạnh mới tốt hơn.

Các hàm phụ trợ

PRUNE$(V, E, X_{\textbf{samples}}, c_{\textbf{best}})$: Cắt tỉa các đỉnh và cạnh không giúp cải thiện lời giải:

$$\text{Loại bỏ } x \in X_{\text{samples}} \text{ nếu } g(x) + h(x) \geq c_{\text{best}}$$
$$\text{Loại bỏ } v \in V \text{ nếu } g(v) + h(v) > c_{\text{best}}$$

SampleInEllipsoid$(x_{\textbf{start}}, x_{\textbf{goal}}, c_{\textbf{best}}, m)$: Lấy m mẫu trong ellipsoid:

Lấy mẫu theo khoảng cách $g(x) + h(x) < c_{\text{best}}$ nếu đã có lời giải.

ComputeRadius(q): Tính bán kính kết nối theo lý thuyết RGG:

$$r = 2 \cdot \eta \cdot \left(1 + \frac{1}{n}\right)^{1/n} \cdot \left(\frac{\lambda(X_{\text{free}})}{\zeta_n}\right)^{1/n} \cdot \left(\frac{\log(q)}{q}\right)^{1/n}$$

3.2 Cài đặt và công cụ thực nghiệm

Ngôn ngữ lập trình

- Sử dụng ngôn ngữ lập trình Python phiên bản 3.12.5

Các thư viện hỗ trợ

- Thư viện **numpy**: Thư viện tính toán số học và xử lý mảng, dùng cho các phép toán Vector và ma trận
- Thư viện **heapq**: Thư viện cung cấp cấu trúc dữ liệu hàng đợi ưu tiên (Priority Queue) dựa trên Heap
- Thư viện **math**: Cung cấp các hàm toán học như Logarit, hằng số Pi và lũy thừa
- Thư viện **matplotlib.pyplot**: Thư viện vẽ đồ thị, dùng để tạo các khung hình trực quan hóa
- Lớp **Ellipse** từ thư viện **matplotlib.patches**: Cung cấp lớp Ellipse để vẽ hình Elip trong không gian 2D
- Lớp **KDTree** từ thư viện **scipy.spatial**: Cung cấp cấu trúc dữ liệu KDTree để tìm kiếm hàng xóm gần trong không gian đa chiều
- Thư viện **time**: Theo giời thời gian thực thi để kiểm sát giới hạn thời gian
- Thư viện **logging**: Ghi lại log thông tin, cảnh báo, và lỗi trong quá trình chạy thuật toán
- Thư viện **collections**: Cung cấp cấu trúc dữ liệu nâng cao như hàng đợi hai đầu
- Thư viện **os**: Quản lý hệ thống tệp, tạo thư mục và kiểm tra đường dẫn
- Thư viện **imageio**: Thư viện xử lý hình ảnh, tạo file GIF từ các khung hình PNG
- Thư viện **glob**: Tìm kiếm tệp theo mẫu tên, dùng để lấy danh sách file PNG

3.3 Mã nguồn triển khai

```
import numpy as np
import heapq
import math
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
from scipy.spatial import KDTree
import time
import logging
import collections
import os

# --- Cau hinh Logging ---
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
logger = logging.getLogger(__name__)

# --- Dinh nghia Moi truong ---
OBSTACLES = [((7.5, 2.5), (10.0, 7.5)),
    ((7.5, 8.0), (10.0, 15.0))]
SPACE_BOUNDS = (0, 20, 0, 20)

# --- Tao thu muc luu khung hinh ---
OUTPUT_DIR = "bit_star_frames"
if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)

# --- Ham Kiem tra Va cham ---
def point_in_rectangle(point, rect_coords):
    px, py = point
    rx1, ry1, rx2, ry2 = rect_coords
    return rx1 <= px <= rx2 and ry1 <= py <= ry2

def is_point_in_obstacle(point):
    for (p1, p2) in OBSTACLES:
        if point_in_rectangle(point, (p1[0], p1[1], p2[0], p2[1])):
            return True
    return False

def line_intersects_rectangle(p1, p2, rect_coords):
    x1, y1 = p1
    x2, y2 = p2
    rx1, ry1, rx2, ry2 = rect_coords
    dx = x2 - x1
    dy = y2 - y1
    t0, t1 = 0.0, 1.0
```

```
45      p = [-dx, dx, -dy, dy]
46      q = [x1 - rx1, rx2 - x1, y1 - ry1, ry2 - y1]
47      for i in range(4):
48          if abs(p[i]) < 1e-9:
49              if q[i] < 0: return False
50          else:
51              r = q[i] / p[i]
52              if p[i] < 0:
53                  if r > t1: return False
54                  if r > t0: t0 = r
55              else:
56                  if r < t0: return False
57                  if r < t1: t1 = r
58      return t0 <= t1
59
60  def is_collision_free(p_start, p_end):
61      """Kiem tra xem doan thang tu p_start den p_end co va cham
           voi chuong ngai vat khong."""
62      if is_point_in_obstacle(p_start) or is_point_in_obstacle(
           p_end):
63          logger.debug(f"Va cham: Diem bat dau {p_start} hoac
               diem ket thuc {p_end} nam trong chuong ngai vat.")
64          return False
65      for (obs_p1, obs_p2) in OBSTACLES:
66          rect_coords = (obs_p1[0], obs_p1[1], obs_p2[0], obs_p2
               [1])
67          if line_intersects_rectangle(p_start, p_end,
               rect_coords):
68              logger.debug(f"Phat hien va cham giua {p_start} va
                   {p_end} voi chuong ngai vat {obs_p1}-{obs_p2}")
69              return False
70      return True
71
72  # --- Ham Chi phi va Uoc luong ---
73  def euclidean_distance(p1, p2):
74      return np.linalg.norm(np.array(p1) - np.array(p2))
75
76  def g_actual(node, g_T):
77      return g_T.get(node, float('inf'))
78
79  def h_estimate(node, goal):
80      return euclidean_distance(node, goal)
81
82  def f_estimate(node, goal, g_T):
83      g_val = g_actual(node, g_T)
84      if g_val == float('inf'):
85          return float('inf')
86      return g_val + h_estimate(node, goal)
```

```
87
88  def cost(p_from, p_to):
89      dist = euclidean_distance(p_from, p_to)
90      if is_collision_free(p_from, p_to):
91          return dist
92      return float('inf')
93
94  def cost_estimate(p_from, p_to):
95      return euclidean_distance(p_from, p_to)
96
97  # --- Ham Lay mau trong Vung Elip ---
98  def sample_informed(m, c_best, x_start, x_goal, bounds=
        SPACE_BOUNDS):
99      """Lay mau trong vung elip xac dinh boi c_best de gioi han
            khong gian tim kiem."""
100     samples = []
101     min_x, max_x, min_y, max_y = bounds
102     max_tries_per_sample = 30
103     generated_count = 0
104
105     if c_best == float('inf'):
106         logger.info("Lay mau deu trong toan khong gian (c_best
                la vo cuc)")
107         while len(samples) < m:
108             tries = 0
109             while tries < max_tries_per_sample:
110                 generated_count += 1
111                 px = np.random.uniform(min_x, max_x)
112                 py = np.random.uniform(min_y, max_y)
113                 point = (px, py)
114                 if not is_point_in_obstacle(point):
115                     samples.append(point)
116                     break
117                 tries += 1
118             if generated_count > m * max_tries_per_sample * 3:
119                 logger.warning(f"Dung lay mau deu som sau {
                        generated_count} lan thu, thu duoc {len(
                        samples)} mau.")
120                 break
121     else:
122         c_min = euclidean_distance(x_start, x_goal)
123         center = np.array([(x_start[0] + x_goal[0]) / 2, (
                x_start[1] + x_goal[1]) / 2])
124         a = c_best / 2.0
125         b = np.sqrt(max(0.0, c_best**2 - c_min**2)) / 2.0
126         if b < 0.1:  # Dam bao b toi thieu
127             b = 0.1
```

```
128         v = (np.array(x_goal) - np.array(x_start)) / c_min if
               c_min > 1e-6 else np.array([1, 0])
129         C = np.array([[v[0], -v[1]], [v[1], v[0]]])
130 
131         logger.info(f"Lay mau trong elip: c_best={c_best:.2f},
               c_min={c_min:.2f}, a={a:.2f}, b={b:.2f}")
132 
133         while len(samples) < m:
134             tries = 0
135             while tries < max_tries_per_sample:
136                 generated_count += 1
137                 r = np.sqrt(np.random.uniform(0, 1))
138                 theta = np.random.uniform(0, 2 * np.pi)
139                 unit_ball_sample = np.array([r * np.cos(theta),
                        r * np.sin(theta)])
140                 ellipse_sample = C @ np.diag([a, b]) @
                       unit_ball_sample + center
141                 point = tuple(ellipse_sample)
142 
143                 if min_x <= point[0] <= max_x and min_y <=
                       point[1] <= max_y and not
                       is_point_in_obstacle(point):
144                     samples.append(point)
145                     break
146                 tries += 1
147             if generated_count > m * max_tries_per_sample * 3:
148                 logger.warning(f"Dung lay mau elip som sau {
                       generated_count} lan thu, thu duoc {len(
                       samples)} mau.")
149                 break
150 
151     logger.info(f"Tao duoc {len(samples)} mau khong va cham (
           tong so lan thu: {generated_count}). 5 mau dau: {samples
           [:5]}")
152     return samples
153 
154 # --- Ham Tinh Ban kinh ---
155 def radius(q, n=2, eta=2.0, lambda_Xf=100.0, zeta_n=math.pi):
156     if q <= 1: return float('inf')
157     try:
158         term_log = math.log(q) / q
159         if term_log < 0: term_log = 0
160         radius_val = 2.0 * eta * ((1.0 + 1.0/n)**(1.0/n)) * ((
               lambda_Xf / zeta_n)**(1.0/n)) * (term_log**(1.0/n))
161         return max(0.1, radius_val)
162     except ValueError:
163         return 0.5
164 
```

```
165 # --- Ham Cat tia ---
166 def prune(c_threshold, V, E, X_samples, g_T, x_start, x_goal,
        parent_map):
167     """Loai bo cac dinh va canh co chi phi uoc luong vuot
            nguong c_threshold."""
168     logger.info(f"Cat tia voi nguong c = {c_threshold:.2f}")
169     samples_to_remove = {x for x in X_samples if f_estimate(x,
            x_goal, g_T) >= c_threshold}
170     if samples_to_remove:
171         X_samples.difference_update(samples_to_remove)
172         logger.debug(f"Da cat {len(samples_to_remove)} mau.")
173 
174     vertices_to_prune = {v for v in V if f_estimate(v, x_goal,
            g_T) > c_threshold}
175     if vertices_to_prune:
176         logger.debug(f"So dinh can cat (f > c): {len(
                vertices_to_prune)}")
177         descendants = set()
178         queue = collections.deque(vertices_to_prune)
179         processed_for_descendants = set()
180 
181         while queue:
182             v_prune = queue.popleft()
183             if v_prune in processed_for_descendants: continue
184             processed_for_descendants.add(v_prune)
185             descendants.add(v_prune)
186             children = {w for u, w in E if u == v_prune}
187             for child in children:
188                 if child not in descendants:
189                     queue.append(child)
190 
191         logger.debug(f"Tong so hau due (bao gom dinh bi cat): {
                len(descendants)}")
192 
193         V.difference_update(descendants)
194         logger.debug(f"Da cat {len(descendants)} dinh (f > c va
                hau due).")
195 
196         edges_to_remove = {(u, v) for u, v in E if u in
                descendants or v in descendants}
197         if edges_to_remove:
198             E.difference_update(edges_to_remove)
199             logger.debug(f"Da cat {len(edges_to_remove)} canh
                    lien quan den dinh bi cat.")
200 
201         vertices_to_move = descendants - vertices_to_prune
202         X_samples.update(vertices_to_move)
```

```
203         logger.debug(f"Chuyen {len(vertices_to_move)} dinh hau
                due ve lai mau.")
204
205         for v_removed in descendants:
206             parent_map.pop(v_removed, None)
207             g_T.pop(v_removed, None)
208
209     unreachable_vertices = {v for v in V if g_actual(v, g_T) ==
            float('inf') and v != x_start}
210     if unreachable_vertices:
211         V.difference_update(unreachable_vertices)
212         X_samples.update(unreachable_vertices)
213         edges_to_remove_unreachable = {(u, v) for u, v in E if
                u in unreachable_vertices or v in
                unreachable_vertices}
214         E.difference_update(edges_to_remove_unreachable)
215         for v_unreachable in unreachable_vertices:
216             parent_map.pop(v_unreachable, None)
217             g_T.pop(v_unreachable, None)
218         logger.debug(f"Chuyen {len(unreachable_vertices)} dinh
                khong the tiep can ve lai mau va cat {len(
                edges_to_remove_unreachable)} canh lien quan.")
219
220 # --- Ham Mo rong Dinh ---
221 def expand_vertex(v_expand, Q_V_heap, Q_E_heap, V, V_old,
        X_samples, E, g_T, x_goal, r):
222     """Mo rong dinh v_expand bang cach tim cac hang xom gan va
            them cac canh tiem nang vao hang doi."""
223     current_points = list(V.union(X_samples))
224     if not current_points or len(current_points) < 2:
225         return
226     try:
227         point_coords = [tuple(map(float, p)) for p in
                current_points]
228         tree = KDTree(point_coords)
229     except (ValueError, TypeError) as e:
230         return
231
232     try:
233         v_expand_float = tuple(map(float, v_expand))
234         neighbor_indices = tree.query_ball_point(v_expand_float
                , r)
235     except (ValueError, TypeError) as e:
236         return
237
238     logger.debug(f"Mo rong dinh v={v_expand}, r={r:.2f}. Tim
            thay {len(neighbor_indices)} hang xom trong ban kinh.")
239     num_edges_added = 0
```

```
240     g_v_expand = g_actual(v_expand, g_T)
241     g_goal = g_actual(x_goal, g_T)
242 
243     for idx in neighbor_indices:
244         x_neighbor = current_points[idx]
245         if x_neighbor == v_expand: continue
246         if x_neighbor in X_samples:
247             cost_v_x_est = cost_estimate(v_expand, x_neighbor)
248             potential_total_cost = g_v_expand + cost_v_x_est +
                    h_estimate(x_neighbor, x_goal)
249             if potential_total_cost < g_goal:
250                 heapq.heappush(Q_E_heap, (potential_total_cost,
                        (v_expand, x_neighbor)))
251                 num_edges_added += 1
252 
253     if v_expand not in V_old:
254         for idx in neighbor_indices:
255             w_neighbor = current_points[idx]
256             if w_neighbor == v_expand: continue
257             if w_neighbor in V:
258                 if (v_expand, w_neighbor) in E or (w_neighbor,
                        v_expand) in E: continue
259                 cost_v_w_est = cost_estimate(v_expand,
                        w_neighbor)
260                 potential_total_cost = g_v_expand +
                        cost_v_w_est + h_estimate(w_neighbor, x_goal)
261                 if potential_total_cost < g_goal:
262                     if g_v_expand + cost_v_w_est < g_actual(
                            w_neighbor, g_T):
263                         heapq.heappush(Q_E_heap, (
                                potential_total_cost, (v_expand,
                                w_neighbor)))
264                         num_edges_added += 1
265 
266     if num_edges_added > 0:
267         logger.info(f"Mo rong {v_expand} them {num_edges_added}
                canh tiem nang vao Q_E.")
268 
269 # --- Ham Ve va Luu Khung hinh ---
270 def save_frame(V, E, path, x_start, x_goal, X_samples, c_best,
        frame_idx, iteration, bounds=SPACE_BOUNDS):
271     """Ve va luu mot khung hinh cua thuat toan BIT*, giong Fig.
            3/Fig. 4 trong tai lieu."""
272     fig, ax = plt.subplots(figsize=(10, 10))
273     min_x, max_x, min_y, max_y = bounds
274 
275     # Ve chuong ngai vat (mau xam) neu co
276     for (p1, p2) in OBSTACLES:
```

```
277             x1, y1 = p1
278             x2, y2 = p2
279             ax.fill([x1, x2, x2, x1], [y1, y1, y2, y2], color='grey
                    ', alpha=0.7)
280
281         # Tinh toan tham so elip
282         c_min = euclidean_distance(x_start, x_goal)
283         center = np.array([(x_start[0] + x_goal[0]) / 2, (x_start
                [1] + x_goal[1]) / 2])
284         v = (np.array(x_goal) - np.array(x_start)) / c_min if c_min
                > 1e-6 else np.array([1, 0])
285         angle_rad = np.arctan2(v[1], v[0])
286         angle_deg = np.degrees(angle_rad)
287
288         # Ve elip chinh (den net dut) neu co duong di
289         if c_best != float('inf'):
290             a = c_best / 2.0
291             b = np.sqrt(max(0.0, c_best**2 - c_min**2)) / 2.0
292             if b < 0.1:  # Dam bao b toi thieu de elip hien thi
293                 b = 0.1
294             ellipse = Ellipse(xy=tuple(center), width=2*a, height
                    =2*b, angle=angle_deg,
295                               edgecolor='black', facecolor='none',
                                      linestyle='--',
296                               linewidth=1.5, alpha=0.7)
297             ax.add_patch(ellipse)
298             logger.debug(f"Ve elip chinh khung {frame_idx}: tam={
                    center}, a={a:.2f}, b={b:.2f}, goc={angle_deg:.1f}")
299
300         # Ve cac dinh trong cay V (xanh duong)
301         if V:
302             plot_v = V - {x_start, x_goal}
303             if plot_v:
304                 final_v_x, final_v_y = zip(*plot_v)
305                 ax.plot(final_v_x, final_v_y, 'o', color='blue',
                        markersize=2, alpha=0.5)
306
307         # Ve cac canh trong cay E (xanh duong)
308         if E:
309             for (v, w) in E:
310                 ax.plot([v[0], w[0]], [v[1], w[1]], 'b-', alpha
                        =0.6, linewidth=0.5)
311
312         # Ve duong di hien tai (magenta)
313         if path:
314             path_x, path_y = zip(*path)
315             ax.plot(path_x, path_y, 'm-', linewidth=2, marker='o',
                    markersize=6, alpha=0.9)
```

```
316
317     # Ve diem bat dau (xanh la cay) va ket thuc (do)
318     ax.plot(x_start[0], x_start[1], 'o', markersize=12,
            markerfacecolor='lime', markeredgecolor='black', zorder
            =10)
319     ax.plot(x_goal[0], x_goal[1], 'o', markersize=12,
            markerfacecolor='red', markeredgecolor='black', zorder
            =10)
320
321     # Thiet lap gioi han va giao dien
322     ax.set_xlim(min_x, max_x)
323     ax.set_ylim(min_y, max_y)
324     ax.set_aspect('equal', adjustable='box')
325     ax.grid(False)  # Bo luoi
326     ax.set_title(f'BIT* - Lan lap {frame_idx}', fontsize=12)
327     plt.tight_layout()
328
329     # Luu khung hinh
330     frame_path = os.path.join(OUTPUT_DIR, f'frame_{frame_idx:04
            d}.png')
331     plt.savefig(frame_path, dpi=150)
332     plt.close(fig)
333     logger.info(f"Da luu khung hinh {frame_idx} tai {frame_path
            }")
334
335 # --- Thuat toan BIT* Chinh ---
336 def bit_star(x_start, x_goal, m=100, max_iterations=1500,
        timeout=60, patience=5):
337     """Thuat toan BIT* tim duong di toi uu tu x_start den
            x_goal."""
338     start_time = time.time()
339     logger.info(f"Bat dau BIT* tu {x_start} den {x_goal}")
340     V = {x_start}
341     E = set()
342     X_samples = {x_goal}
343     g_T = {x_start: 0.0}
344     parent_map = {x_start: None}
345     Q_V_heap = []
346     Q_E_heap = []
347     heapq.heappush(Q_V_heap, (f_estimate(x_start, x_goal, g_T),
                x_start))
348     r = float('inf')
349     V_old = set()
350     c_best = float('inf')
351     iteration = 0
352     solution_found = False
353     all_vertices_ever = set(V)
354     all_edges_ever = set()
```

```
355     frame_idx = 0
356     best_goal_cost = float('inf')
357     no_improvement_count = 0
358 
359     # Luu khung hinh ban dau
360     path = []
361     save_frame(V, E, path, x_start, x_goal, X_samples, c_best,
            frame_idx, iteration)
362     frame_idx += 1
363 
364     while iteration < max_iterations:
365         current_time = time.time()
366         if current_time - start_time > timeout:
367             logger.warning("Het thoi gian!")
368             break
369         logger.info(f"\n--- Lan lap {iteration + 1}/{
                max_iterations} (Thoi gian: {current_time -
                start_time:.2f}s) ---")
370         logger.info(f"|V|={len(V)}, |E|={len(E)}, |X_samples|={
                len(X_samples)}, |Q_V|={len(Q_V_heap)}, |Q_E|={len(
                Q_E_heap)}, g(goal)={g_T.get(x_goal, float('inf')):.2
                f}, c_best={c_best:.2f}, r={r:.2f}")
371 
372         # Kiem tra patience
373         current_goal_cost = g_T.get(x_goal, float('inf'))
374         if current_goal_cost < best_goal_cost:
375             logger.info(f"Cai thien chi phi den dich: {
                    best_goal_cost:.2f} -> {current_goal_cost:.2f}")
376             best_goal_cost = current_goal_cost
377             no_improvement_count = 0
378         else:
379             no_improvement_count += 1
380             if solution_found and no_improvement_count >=
                    patience:
381                 logger.info(f"Dung som sau {
                        no_improvement_count} lan lap khong cai thien
                         (patience={patience}).")
382                 break
383 
384         if not Q_V_heap and not Q_E_heap:
385             logger.warning("Ca hai hang doi rong.")
386             needs_resample_and_prune = False
387             if current_goal_cost < c_best:
388                 logger.info(f"Tim thay duong tot hon! c_best cu
                        : {c_best:.2f}, c_best moi: {
                        current_goal_cost:.2f}")
389                 c_best = current_goal_cost
390                 solution_found = True
```

```
391                 needs_resample_and_prune = True
392                 # Luu duong di moi
393                 path = []
394                 current = x_goal
395                 while current is not None:
396                     path.append(current)
397                     current = parent_map.get(current, None)
398                 path.reverse()
399                 # Luu khung hinh khi tim thay duong tot hon
400                 save_frame(V, E, path, x_start, x_goal,
                        X_samples, c_best, frame_idx, iteration)
401                 frame_idx += 1
402             elif not solution_found and iteration > 0:
403                 logger.info("Chua tim thay giai phap, lay mau
                        lai de kham pha them.")
404                 needs_resample_and_prune = True
405 
406             if needs_resample_and_prune:
407                 if c_best != float('inf'):
408                     prune(c_best, V, E, X_samples, g_T, x_start
                            , x_goal, parent_map)
409                     Q_V_heap = [(f_estimate(v, x_goal, g_T), v)
                             for v in V if g_actual(v, g_T) != float(
                            'inf')]
410                     heapq.heapify(Q_V_heap)
411                     Q_E_heap = []
412                     logger.info(f"Xay lai Q_V sau cat tia: kich
                             thuoc {len(Q_V_heap)}")
413                 else:
414                     Q_V_heap = [(f_estimate(v, x_goal, g_T), v)
                             for v in V if g_actual(v, g_T) != float(
                            'inf')]
415                     heapq.heapify(Q_V_heap)
416                     Q_E_heap = []
417 
418                 new_samples = sample_informed(m, c_best,
                        x_start, x_goal, bounds=SPACE_BOUNDS)
419                 if not new_samples and not V:
420                     logger.error("Khong tao duoc mau moi va
                            khong con dinh nao trong cay. Dung.")
421                     break
422                 if new_samples:
423                     X_samples.update(new_samples)
424                     all_vertices_ever.update(new_samples)
425 
426                 V_old = V.copy()
427                 r = radius(len(V) + len(X_samples))
```

```
428                 logger.info(f"Lay mau lai {len(new_samples)}
                        diem. Ban kinh moi r = {r:.2f}")
429
430                 if not Q_V_heap and V:
431                     for v_in_tree in V:
432                         if g_actual(v_in_tree, g_T) != float('
                                inf'):
433                             heapq.heappush(Q_V_heap, (
                                    f_estimate(v_in_tree, x_goal, g_T
                                    ), v_in_tree))
434                     logger.info(f"Dien lai Q_V voi {len(
                            Q_V_heap)} dinh tu cay V hien tai.")
435
436                 # Luu khung hinh sau khi lay mau va cat tia
437                 save_frame(V, E, path, x_start, x_goal,
                        X_samples, c_best, frame_idx, iteration)
438                 frame_idx += 1
439
440                 if not Q_V_heap and not Q_E_heap and not
                        X_samples:
441                     logger.error("Thuat toan ket: Ca hai hang
                            doi rong, khong con mau. Khong the tiep
                            tuc.")
442                     break
443             else:
444                 logger.info("Hang doi rong, nhung khong tim
                        thay giai phap tot hon va da co giai phap.
                        Tiep tuc ma khong lay mau/cat tia.")
445                 if not Q_V_heap and V:
446                     for v_in_tree in V:
447                         if g_actual(v_in_tree, g_T) != float('
                                inf'):
448                             heapq.heappush(Q_V_heap, (
                                    f_estimate(v_in_tree, x_goal, g_T
                                    ), v_in_tree))
449                     logger.info(f"(Khong lay mau lai) Dien lai
                            Q_V voi {len(Q_V_heap)} dinh.")
450                 if not Q_V_heap and not Q_E_heap:
451                     logger.warning("Ca hai hang doi van rong
                            sau khi co dien lai Q_V. Co the bi ket.")
452                     break
453
454         peek_qv = Q_V_heap[0][0] if Q_V_heap else float('inf')
455         peek_qe = Q_E_heap[0][0] if Q_E_heap else float('inf')
456
457         processed_in_batch = False
458         while Q_V_heap and peek_qv <= peek_qe:
459             processed_in_batch = True
```

```
460             f_est_v, v_to_expand = heapq.heappop(Q_V_heap)
461             if v_to_expand not in V:
462                 logger.debug(f"Bo qua mo rong dinh da bi cat {
                        v_to_expand}")
463                 peek_qv = Q_V_heap[0][0] if Q_V_heap else float
                        ('inf')
464                 continue
465             current_f_est_v = f_estimate(v_to_expand, x_goal,
                    g_T)
466             if abs(current_f_est_v - f_est_v) > 1e-6:
467                 heapq.heappush(Q_V_heap, (current_f_est_v,
                        v_to_expand))
468                 logger.debug(f"Dua lai dinh {v_to_expand} vao
                        hang doi voi f_est cap nhat {current_f_est_v
                        :.2f} (truoc la {f_est_v:.2f})")
469                 peek_qv = Q_V_heap[0][0] if Q_V_heap else float
                        ('inf')
470                 continue
471
472             logger.info(f"Mo rong dinh v = {v_to_expand} (f_est
                    ={f_est_v:.2f})")
473             expand_vertex(v_to_expand, Q_V_heap, Q_E_heap, V,
                    V_old, X_samples, E, g_T, x_goal, r)
474             peek_qv = Q_V_heap[0][0] if Q_V_heap else float('
                    inf')
475             peek_qe = Q_E_heap[0][0] if Q_E_heap else float('
                    inf')
476
477         if Q_E_heap and peek_qe < peek_qv:
478             processed_in_batch = True
479             est_total_cost, (v_m, x_m) = heapq.heappop(Q_E_heap
                    )
480
481             if v_m not in V or (x_m in V and x_m not in V) or (
                    x_m in X_samples and x_m not in X_samples):
482                 logger.debug(f"Bo qua canh ({v_m}, {x_m}) lien
                        quan den phan tu da bi cat.")
483                 iteration += 1
484                 continue
485
486             logger.info(f"Xu ly canh (v_m, x_m) = ({v_m}, {x_m
                    }) voi chi phi uoc tinh = {est_total_cost:.2f}")
487
488             current_g_vm = g_actual(v_m, g_T)
489             recalc_est_total_cost = current_g_vm +
                    cost_estimate(v_m, x_m) + h_estimate(x_m, x_goal)
490             current_g_goal = g_T.get(x_goal, float('inf'))
491
```

```
492             if recalc_est_total_cost >= current_g_goal:
493                 logger.debug(f"  Canh khong con hop le hoac
                        khong hua hen (chi phi uoc tinh={
                        recalc_est_total_cost:.2f} >= g_goal={
                        current_g_goal:.2f}). Bo qua.")
494                 iteration += 1
495                 continue
496
497             c_vm_xm = cost(v_m, x_m)
498             if c_vm_xm == float('inf'):
499                 logger.debug(f"  Canh ({v_m}, {x_m}) co chi phi
                        vo cuc (va cham). Bo qua.")
500                 iteration += 1
501                 continue
502
503             new_g_xm = current_g_vm + c_vm_xm
504             current_g_xm = g_T.get(x_m, float('inf'))
505
506             if new_g_xm < current_g_xm:
507                 logger.info(f"  Canh ({v_m}, {x_m}) cung cap
                        duong tot hon den x_m (g_moi={new_g_xm:.2f} <
                        g_hien_tai={current_g_xm:.2f})")
508                 g_T[x_m] = new_g_xm
509                 parent_map[x_m] = v_m
510                 all_edges_ever.add((v_m, x_m))
511
512                 if x_m in V:
513                     old_edges_to_xm = {(u, v) for u, v in E if
                            v == x_m}
514                     E.difference_update(old_edges_to_xm)
515                     E.add((v_m, x_m))
516                     logger.debug(f"Dinh tuyen lai: Cap nhat cha
                            cua {x_m} thanh {v_m}. Xoa {len(
                            old_edges_to_xm)} canh cu. Them canh moi.
                            ")
517                 else:
518                     if x_m in X_samples: X_samples.remove(x_m)
519                     V.add(x_m)
520                     E.add((v_m, x_m))
521                     heapq.heappush(Q_V_heap, (f_estimate(x_m,
                            x_goal, g_T), x_m))
522                     logger.info(f"    Them dinh {x_m} vao V va
                            canh ({v_m}, {x_m}) vao E. Them {x_m} vao
                            Q_V.")
523
524                 logger.debug(f"    Cap nhat g_T[{x_m}] = {g_T[
                        x_m]:.2f}")
525                 if x_m == x_goal:
```

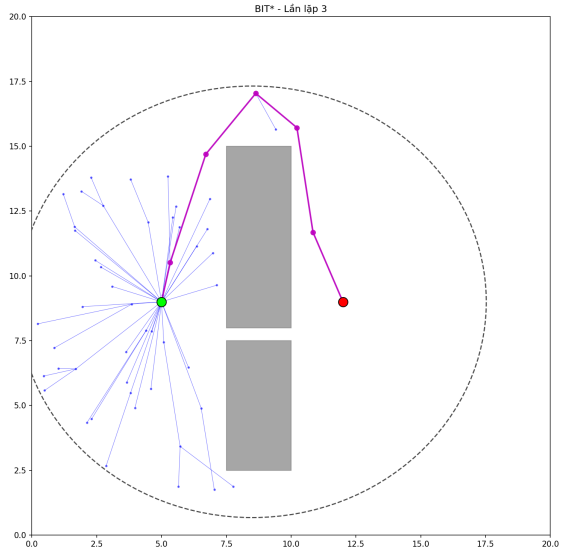
```
526                     logger.info(f"    Da den dich! Chi phi moi
                           g(goal) = {g_T[x_goal]:.2f}")
527                     if g_T[x_goal] < c_best:
528                         c_best = g_T[x_goal]
529                         logger.info(f"    Cap nhat c_best = {
                               c_best:.2f}")
530                         # Luu duong di moi
531                         path = []
532                         current = x_goal
533                         while current is not None:
534                             path.append(current)
535                             current = parent_map.get(current,
                                   None)
536                         path.reverse()
537                         # Luu khung hinh khi cap nhat dich
538                         save_frame(V, E, path, x_start, x_goal,
                               X_samples, c_best, frame_idx,
                              iteration)
539                         frame_idx += 1
540
541         if processed_in_batch or (not Q_V_heap and not Q_E_heap
                ):
542             iteration += 1
543         else:
544             logger.warning("Lan lap khong xu ly phan tu nao
                   trong hang doi. Co the bi ket.")
545             if not Q_V_heap and Q_E_heap:
546                 logger.info("Q_V rong, xu ly canh tu Q_E bat ke
                       .")
547                 pass
548             else:
549                 break
550
551     logger.info("\n=== Thuat toan Ket thuc ===")
552     logger.info(f"So lan lap: {iteration}")
553     logger.info(f"Thoi gian thuc thi: {time.time() - start_time
           :.2f} giay")
554     logger.info(f"Kich thuoc cuoi |V|={len(V)}, |E|={len(E)}, |
           X_samples|={len(X_samples)}")
555     final_goal_cost = g_T.get(x_goal, float('inf'))
556     logger.info(f"Chi phi cuoi den dich g(goal): {
           final_goal_cost:.2f}")
557
558     path = []
559     if final_goal_cost != float('inf'):
560         current = x_goal
561         while current is not None:
562             path.append(current)
```

```
563             current = parent_map.get(current, None)
564         path.reverse()
565         logger.info(f"Duong di tim duoc: {path}")
566         solution_found = True
567         # Luu khung hinh cuoi cung
568         save_frame(V, E, path, x_start, x_goal, X_samples,
                c_best, frame_idx, iteration)
569         frame_idx += 1
570     else:
571         logger.warning("Khong tim thay duong den dich.")
572         solution_found = False
573 
574     return V, E, g_T, path, solution_found, all_vertices_ever,
        all_edges_ever, []
575 
576 # --- Ham Tao GIF ---
577 def create_gif(output_path='bit_star_animation.gif', duration
    =0.05, fps=30):
578     """Tao GIF tu cac khung hinh trong thu muc bit_star_frames.
        """
579 
580     import imageio
581     import glob
582     images = sorted(glob.glob(os.path.join(OUTPUT_DIR, 'frame_
        *.png')))
583     if not images:
584         return
585     with imageio.get_writer(output_path, mode='I', duration=
        duration, loop=0) as writer:
586         for image_path in images:
587             image = imageio.imread(image_path)
588             writer.append_data(image)
589     logger.info(f"Da tao GIF tai {output_path}")
590 
591 # --- Chay thuat toan ---
592 if __name__ == "__main__":
593     start_node = (5.0, 9.0)
594     goal_node = (12.0, 9.0)
595 
596     V_final, E_final, g_T_final, final_path, found, all_verts,
        all_edges, history = bit_star(
597         start_node,
598         goal_node,
599         m=200,
600         max_iterations=500000,
601         timeout=60,
602         patience=5
603     )
```

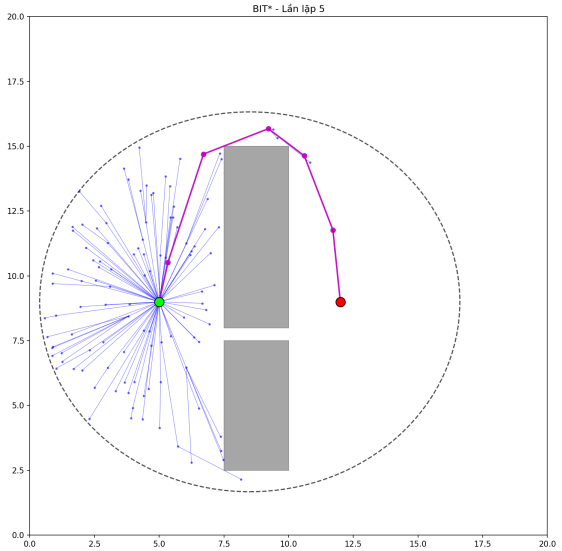
```
604
605     # Tao GIF tu cac khung hinh
606     create_gif()
607
608     if not found:
609         logger.error("BIT* khong tim duoc giai phap trong gioi
            han cho phep.")
```

3.4 Phân tích và đánh giá kết quả

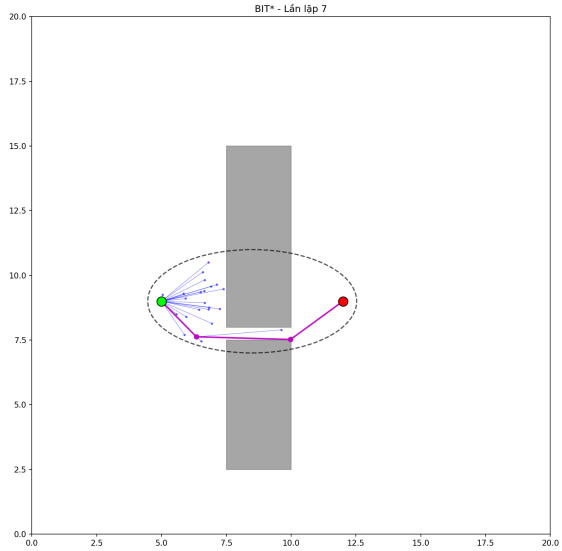
3.4.1 Kết quả



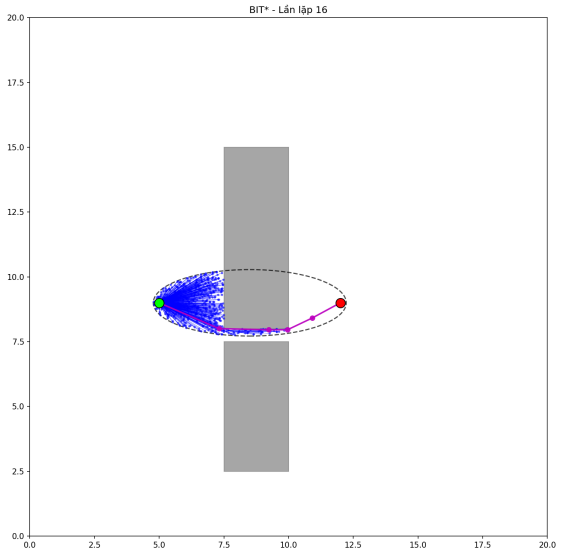
Hình 3.1: Mô phỏng tại khung hình thứ 3



Hình 3.2: Mô phỏng tại khung hình thứ 5



Hình 3.3: Mô phỏng tại khung hình thứ 7



Hình 3.4: Mô phỏng tại khung hình thứ 16

3.4.2 Phân tích

- **Hình 3.1:**
  - **Cây tìm kiếm:** Cây bắt đầu phát triển từ điểm khởi đầu, với nhiều đỉnh

và cạnh xanh dương lan tỏa xung quanh. Các đỉnh tập trung nhiều quanh chướng ngại vật, cho thấy thuật toán đang khám phá không gian khả thi.

- **Đường đi:** Đường đi đã xuất hiện, đi qua một số điểm trung gian nhưng chưa tối ưu. Hiện tại đường vòng qua phía trên của chướng ngại vật thứ hai với chi phí khá cao.
- **Elip lấy mẫu:** Elip khá lớn, bao phủ từ điểm đầu đến điểm kết thúc, cho thấy **c_best** (chi phí tốt nhất hiện tại) còn lớn. Thuật toán vẫn đang tìm kiếm rộng để xác định các đường đi khả thi.
- **Nhận xét:** Ở giai đoạn đầu, thuật toán đã tìm được một đường đi khả thi, nhưng chưa tối ưu.

- **Hình 3.2:**
  - **Cây tìm kiếm:** Cây tiếp tục mở rộng, với nhiều đỉnh và cạnh hơn. Một số nhánh đã tiến gần hơn đến điểm kết thúc, đặc biệt ở phía dưới chướng ngại vật thứ hai.
  - **Đường đi:** Đường đi được cải thiện, vẫn đi vòng phía trên nhưng có ít điểm trung gian hơn và mượt hơn so với lần lặp trước.
  - **Elip lấy mẫu:** Elip thu hẹp lại, cho thấy **c_best** đã giảm. Thuật toán đang tập trung lấy mẫu ở vùng hẹp hơn để tối ưu hóa đường đi.
  - **Nhận xét:** Đường đi đang được cải thiện. BIT* chuyển dần từ giai đoạn khám phá sang khai thác.

- **Hình 3.3:**
  - **Cây tìm kiếm:** Cây đã được cắt tỉa đáng kể, các nhánh không hứa hẹn bị loại bỏ. Các đỉnh tập trung tại vùng giữa hai chướng ngại vật.
  - **Đường đi:** Đường đi đã thay đổi, đi qua khe hẹp giữa hai chướng ngại vật — ngắn hơn và hợp lý hơn về mặt chi phí.
  - **Elip lấy mẫu:** Elip rất sát đường đi hiện tại, cho thấy **c_best** gần với chi phí tối ưu.
  - **Nhận xét:** BIT* hoạt động hiệu quả trong việc tối ưu đường đi và loại bỏ các nhánh dư thừa.

- **Hình 3.4:**
  - **Cây tìm kiếm:** Cây rất gọn, chỉ giữ lại những đỉnh và cạnh quan trọng.
  - **Đường đi:** Không thay đổi so với trước đó, đi giữa hai chướng ngại vật — rất sát đường tối ưu.
  - **Elip lấy mẫu:** Elip rất nhỏ, chỉ bao quanh đường đi tối ưu, cho thấy **c_best** đã hội tụ.
  - **Nhận xét:** Thuật toán đã hội tụ và đạt được kết quả gần tối ưu.

3.4.3 Nhận xét

- **Hiệu quả của BIT*:**
  - Thể hiện rõ ưu điểm trong việc kết hợp lấy mẫu thông minh (Informed Sampling) và cắt tỉa (Pruning). Elip lấy mẫu giới hạn không gian tìm kiếm, tập trung vào vùng có khả năng cải thiện đường đi.
  - Cơ chế cắt tỉa loại bỏ các nhánh không hứa hẹn, giảm đáng kẹn số lượng đỉnh và cạnh cần xử lý, giúp thuật toán nhanh chóng hội tụ.
- **Tiến trình tối ưu hóa:**
  - Từ lần lặp 3 đến lần lặp 7, đường đi được cải thiện đáng kể, từ một đường vòng dài sang đường ngắn nhất qua khe giữa hai chướng ngại vật.
  - Sau lần lặp 7, đường đi không thay đổi, và thuật toán tập trung tinh chỉnh cây tìm kiếm, loại bỏ các đỉnh và cạnh thừa để đạt hiệu quả tối đa.
- **Trực quan hóa**
  - **Cây tìm kiếm (xanh dương):** Ban đầu lan tỏa rộng, sau đó thu hẹp nhờ cắt tỉa, tập trung vào đường đi tối ưu
  - **Elip (đường nét đứt):** Thu hẹp dần qua các lần lặp, phản ánh việc **c_best** giảm và không gian tìm kiếm được giới hạn hiệu quả.
  - **Đường đi (magenta):** Thể hiện rõ quá trình tối ưu, từ đường vòng dài sang đường ngắn nhất qua khe hẹp.

Chương 4

Ứng dụng thuật toán

4.1 Ứng dụng thuật toán BIT* trong nhận dạng

Thuật toán **BIT* (Batch Informed Trees)** chủ yếu được sử dụng trong lập kế hoạch chuyển động cho robot. Tuy nhiên, trong lĩnh vực **nhận dạng**, thuật toán này cũng có thể được ứng dụng gián tiếp trong các hệ thống kết hợp giữa *nhận dạng và điều hướng*. Dưới đây là một số ứng dụng tiêu biểu:

1. Nhận dạng kết hợp điều hướng robot

Trong các hệ thống robot có khả năng nhận dạng (chẳng hạn như nhận diện khuôn mặt, mã QR, vật thể), BIT* được dùng để:

- Lập kế hoạch đường đi tối ưu đến vị trí có đối tượng cần nhận dạng.
- Tránh các chướng ngại vật trong quá trình di chuyển.
- Rút ngắn thời gian tiếp cận đối tượng sau khi nhận dạng.

Ví dụ: Một robot di chuyển trong nhà kho để tìm và nhận dạng sản phẩm qua mã vạch. Sau khi nhận dạng được vị trí sản phẩm, robot sử dụng BIT* để tìm đường đi nhanh và an toàn nhất để lấy hàng.

2. Nhận dạng trong không gian 3D

Trong các ứng dụng thực tế ảo (AR/VR) hoặc hệ thống bản đồ 3D:

- BIT* hỗ trợ lập kế hoạch chuyển động cho cảm biến (camera, LiDAR) để quét các khu vực cần nhận dạng một cách hiệu quả.
- Giúp robot xác định được các điểm quan sát tối ưu để thu thập thông tin nhận dạng.

3. Kết hợp với học sâu

Trong một số hệ thống tích hợp học sâu (deep learning):

- Mô hình nhận dạng (ví dụ: CNN, YOLO) phát hiện đối tượng cần quan tâm.
- BIT* được dùng để lập kế hoạch di chuyển nhanh tới vị trí đó, đảm bảo tránh vật cản và giảm thời gian.

4. Ứng dụng trong drone và robot tự hành

Các drone hoặc xe tự hành sử dụng BIT* trong môi trường có nhận dạng:

- Xác định đối tượng (con người, phương tiện, biển báo).
- Di chuyển tối ưu tới vị trí tương ứng thông qua BIT*, ngay cả trong môi trường phức tạp.

Ưu điểm của BIT* trong hệ thống nhận dạng

- Giảm số mẫu cần thiết trong lập kế hoạch đường đi.
- Hội tụ nhanh hơn đến lời giải tối ưu so với RRT*.
- Hoạt động hiệu quả trong môi trường lớn và phức tạp.

4.2 Ứng dụng thuật toán BIT* trong tối ưu hóa

Thuật toán **BIT* (Batch Informed Trees)** không chỉ được sử dụng trong các bài toán lập kế hoạch đường đi, mà còn có thể được xem như một phương pháp tối ưu hóa trong không gian liên tục. Dưới đây là các ứng dụng tiêu biểu trong lĩnh vực tối ưu:

1. Tối ưu hóa đường đi (Path Optimization)

- BIT* được thiết kế để tìm đường đi ngắn nhất giữa hai điểm trong không gian có chướng ngại vật.
- Quá trình tìm kiếm sử dụng chiến lược heuristic và lấy mẫu thông minh giúp giảm số lượng mẫu cần thiết.
- Việc xử lý theo lô (batch) giúp cải thiện tốc độ hội tụ đến đường đi tối ưu.

2. Tối ưu hóa đa mục tiêu (Multi-objective Optimization)

- BIT* có thể được mở rộng để giải quyết bài toán tối ưu hóa đa mục tiêu, chẳng hạn như:
  - Tối ưu hóa thời gian và mức tiêu thụ năng lượng cùng lúc.
  - Tối ưu hóa giữa quãng đường và độ an toàn trong môi trường phức tạp.
- Khi kết hợp với hàm đánh giá tổng hợp (cost function), BIT* giúp cân bằng giữa các mục tiêu này hiệu quả hơn.

3. Tối ưu hóa trong không gian ràng buộc

- BIT* sử dụng các kỹ thuật heuristic và vùng không gian giới hạn (ellipsoid) để tập trung vào các vùng có khả năng chứa nghiệm tối ưu.
- Điều này làm giảm đáng kể không gian tìm kiếm, đặc biệt hiệu quả trong các bài toán tối ưu có nhiều ràng buộc (constraint optimization).

4. Tối ưu hóa thời gian thực (Real-time Optimization)

- BIT* có khả năng tạo ra lời giải gần tối ưu trong thời gian ngắn, giúp ứng dụng tốt trong các hệ thống yêu cầu phản hồi nhanh như:
  - Robot điều hướng trong môi trường động.
  - Xe tự hành ra quyết định trong điều kiện thay đổi liên tục.
- BIT* hỗ trợ **tối ưu hóa liên tục** trong quá trình chạy (online replanning), giúp hệ thống thích nghi với dữ liệu mới.

5. Kết hợp với các phương pháp học máy

- Trong các hệ thống sử dụng học tăng cường (reinforcement learning) hoặc học sâu, BIT* có thể đóng vai trò như một phương pháp heuristic tối ưu hóa đường đi.
- Ví dụ: trong huấn luyện robot, BIT* có thể được dùng để cung cấp các đường đi gần tối ưu, hỗ trợ mạng học sâu học chính xác và nhanh hơn.

Kết luận

Từ những phân tích trên,thuật toán BIT* vượt trội hơn so với các phương pháp lập kế hoạch tối ưu dựa trên lấy mẫu hiện có và RRT, đặc biệt trong không gian cao chiều. Với cùng một khoảng thời gian tính toán, BIT* có khả năng tìm ra giải pháp cao hơn và thường tìm được các giải pháp có chất lượng tương đương nhanh hơn. Nó cũng hội tụ về giải pháp tối ưu nhanh hơn so với các phương pháp lập kế hoạch tối ưu tiệm cận khác, và gần đây đã được chứng minh là hoạt động tốt trên các bài toán có ràng buộc vi phân.

Tài liệu tham khảo

[1] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. *Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs.*

[2] C. Xie, J. van den Berg, S. Patil, and P. Abbeel. “Toward asymptotically optimal motion planning for kinodynamic systems using a two-point boundary value problem solver." In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015.