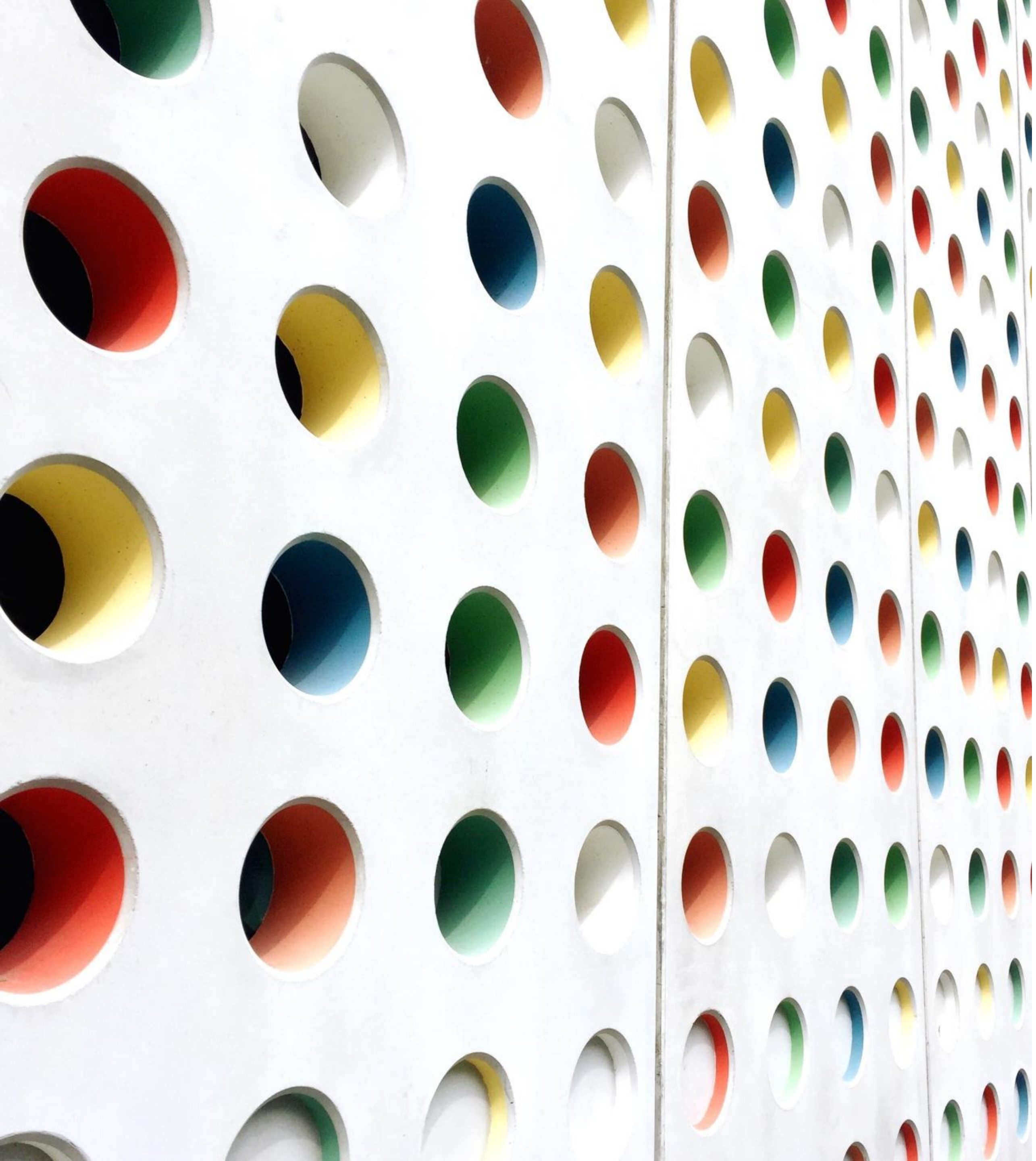


Path-planning algorithms

LIDIA FARGUETA PELUFO

Index

- Autonomous navigation steps
- Classification of path-planning algorithms
- Classical vs learning-based planners
- Difference between search-based and sampling-based planner.
- Some algorithms:
  - RTT*
  - RTT*-FND
  - GNN-based neural planner
  - GraphMP
  - Neural A* Search
- For testing in Duckiebot…
- Problems and questions

Autonomous navigation steps*

1. The sensory system captures the robot's surrounding environment (Perception)
2. Identification of robot's location in the environment (Localisation)
3. The robot decides how to manoeuvre to reach the goal without collision (Path-planning)
4. The robot's motions are controlled to follow the desired path (Motion control)

We focus on this one

* H. S. Hewawasam, M. Y. Ibrahim and G. K. Appuhamillage, "Past, Present and Future of Path-Planning Algorithms for Mobile Robot Navigation in Dynamic Environments," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 353-365, 2022, doi: 10.1109/OJIES.2022.3179617.

Classifications of path-planning algorithms

1. Classic & heuristic methods*
2. Classic & Learning-based methods**
3. Local & global methods
4. Static & dynamic methods
5. Search-based, sampling-based, data-driver based, reactive planners, Inverse Reinforcement Learning-Based, Hybrid...

We focus on this way of classifying

* H. S. Hewawasam, M. Y. Ibrahim and G. K. Appuhamillage, "Past, Present and Future of Path-Planning Algorithms for Mobile Robot Navigation in Dynamic Environments," in IEEE Open Journal of the Industrial Electronics Society, vol. 3, pp. 353-365, 2022, doi: 10.1109/OJIES.2022.3179617.

**Zang, X. *et al.* (2023) *GraphMP: Graph neural network-based motion planning with Efficient Graph Search*, *Advances in Neural Information Processing Systems*. Available at: https://papers.nips.cc/paper_files/paper/2023/hash/096961cae3c3423c44ea045aeb584e05-Abstract-Conference.html (Accessed: 21 June 2024).

Our classification (following GraphMP paper)

Classical Planners:

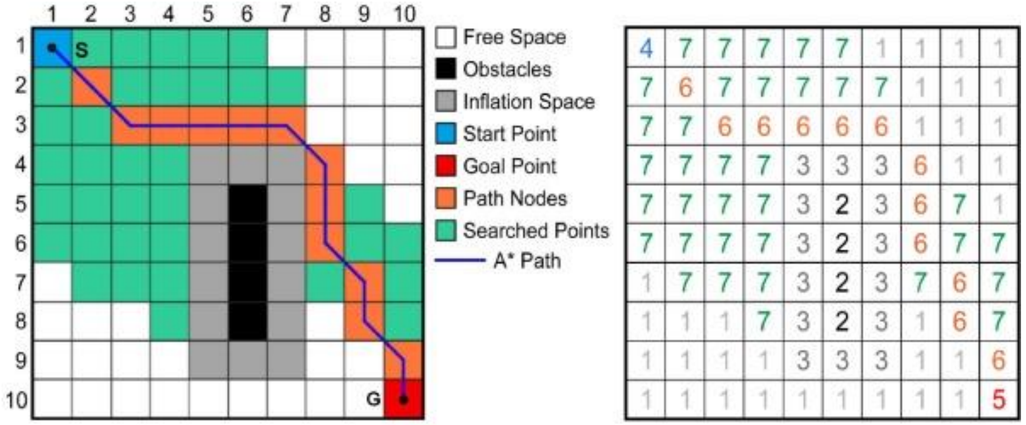
- Search-based planners: BFS, Dijkstra, A*, D*, WA*, Hybrid-A*.
- Sampling-based planners: RRT, RRT*, Informed-RRT*, PRM, BIT*, LazySP, RRT*FND

For dynamic environment

Learning-based Planners (can be search-based or sampling-based):

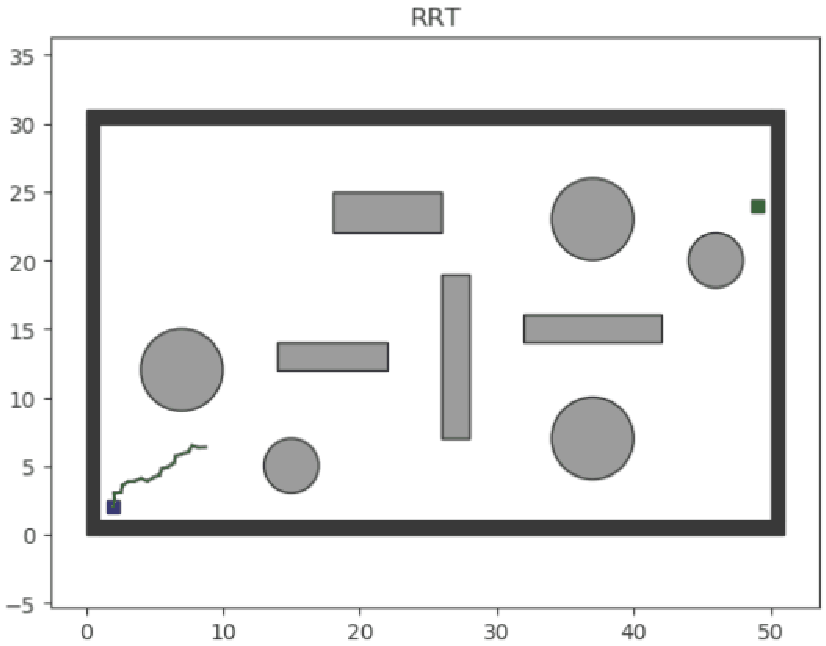
- Neural A*, GNN-based neural planner, GraphMP

Difference between search and sampling based



(a) Grid map.   (b) Matrix map.

Changgeng Li, Xia Huang, Jun Ding, Kun Song, Shiqing Lu,
Global path planning based on a bidirectional alternating search A* algorithm for mobile robots,
Computers & Industrial Engineering,
Volume 168, 2022, 108123,
ISSN 0360-8352, https://doi.org/10.1016/j.cie.2022.108123

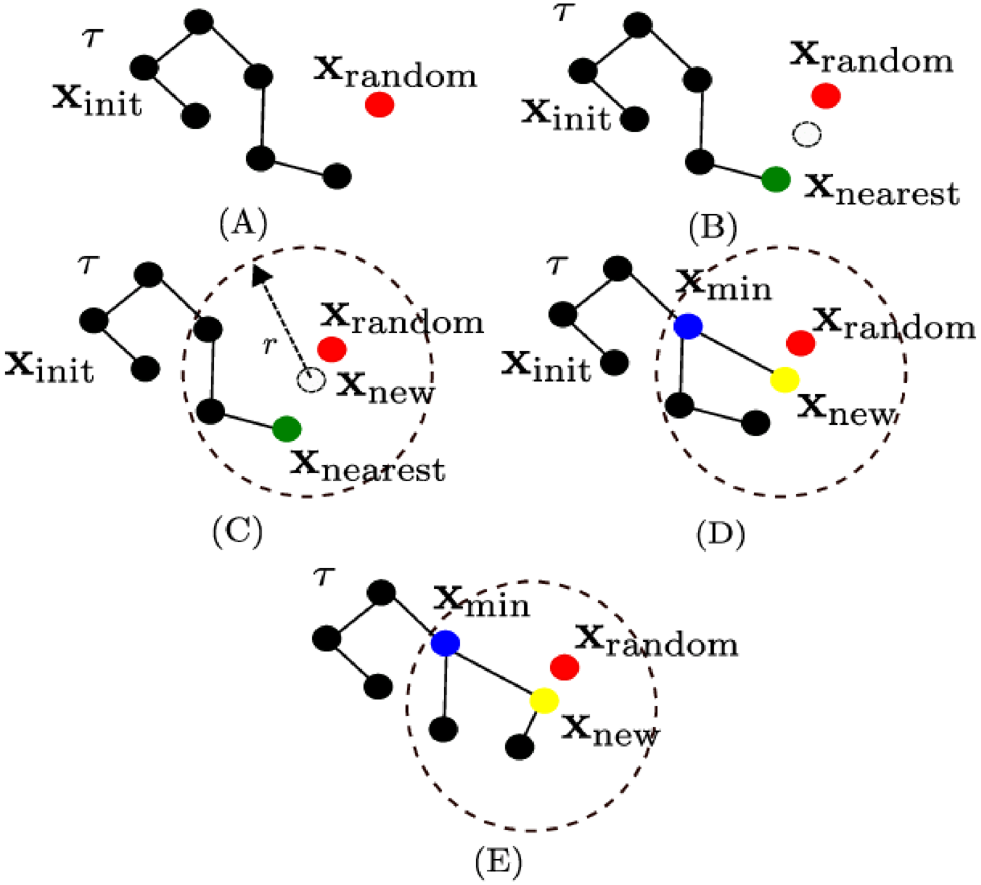


https://github.com/zhm-real/PathPlanning?tab=readme-ov-file

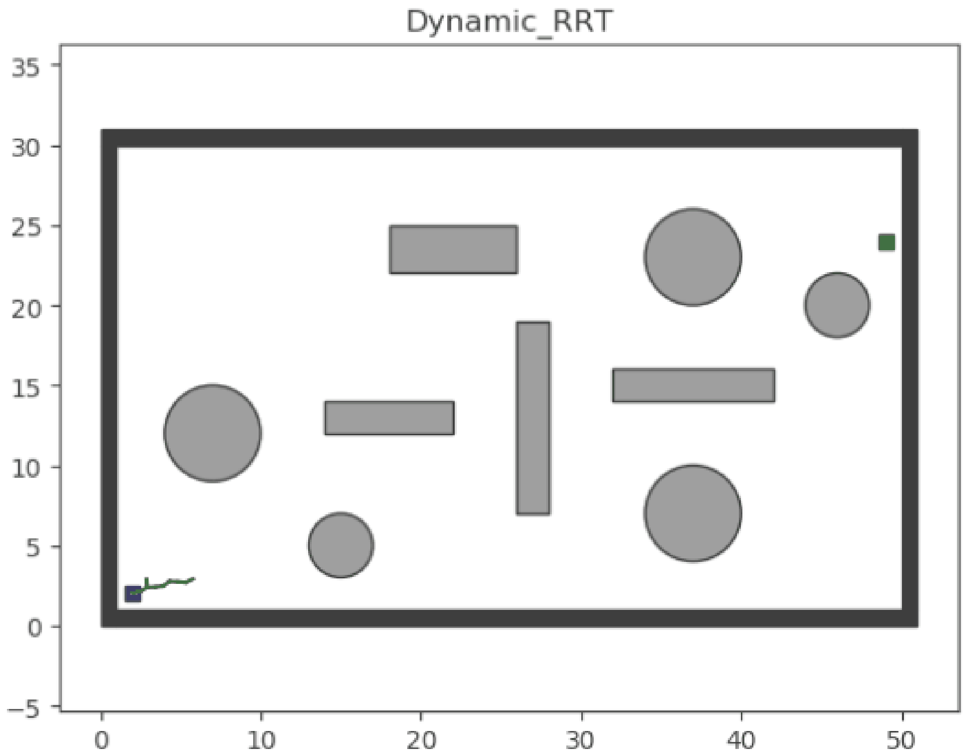
RRT* (https://arxiv.org/pdf/1105.1186)

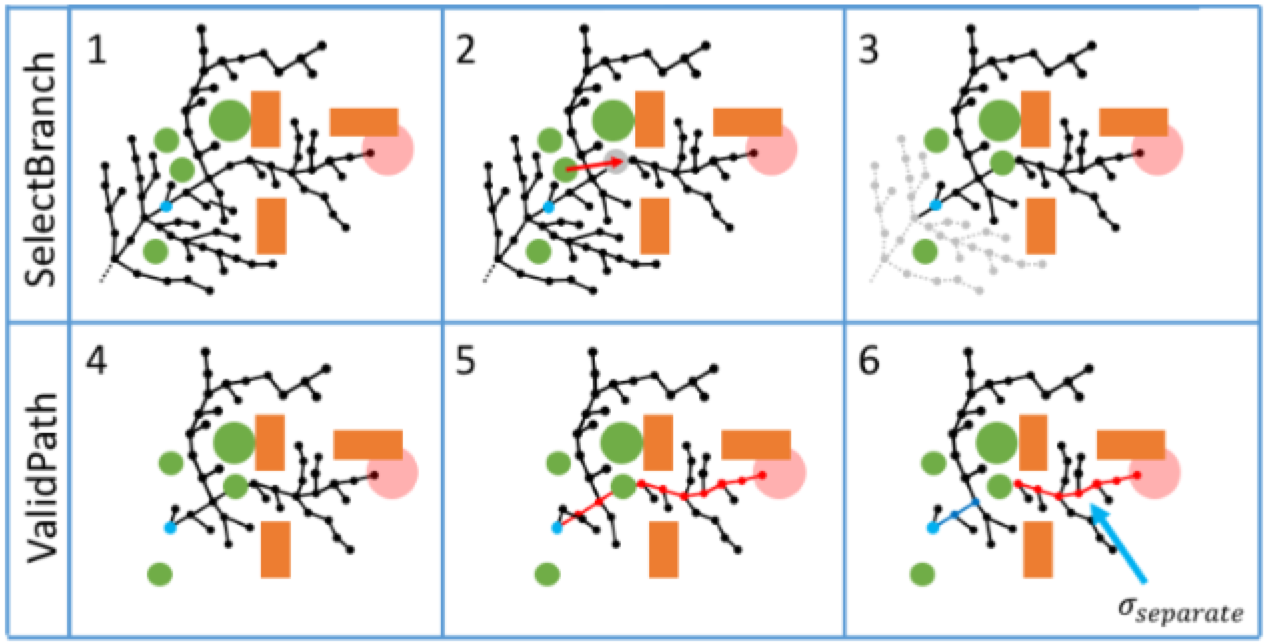
Algorithm 6: RRT*

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}\,(V))/\,\text{card}\,(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\min} \leftarrow x_{\text{nearest}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$; $c_{\min} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$
      then $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;



RRT*-FN-Dynamic



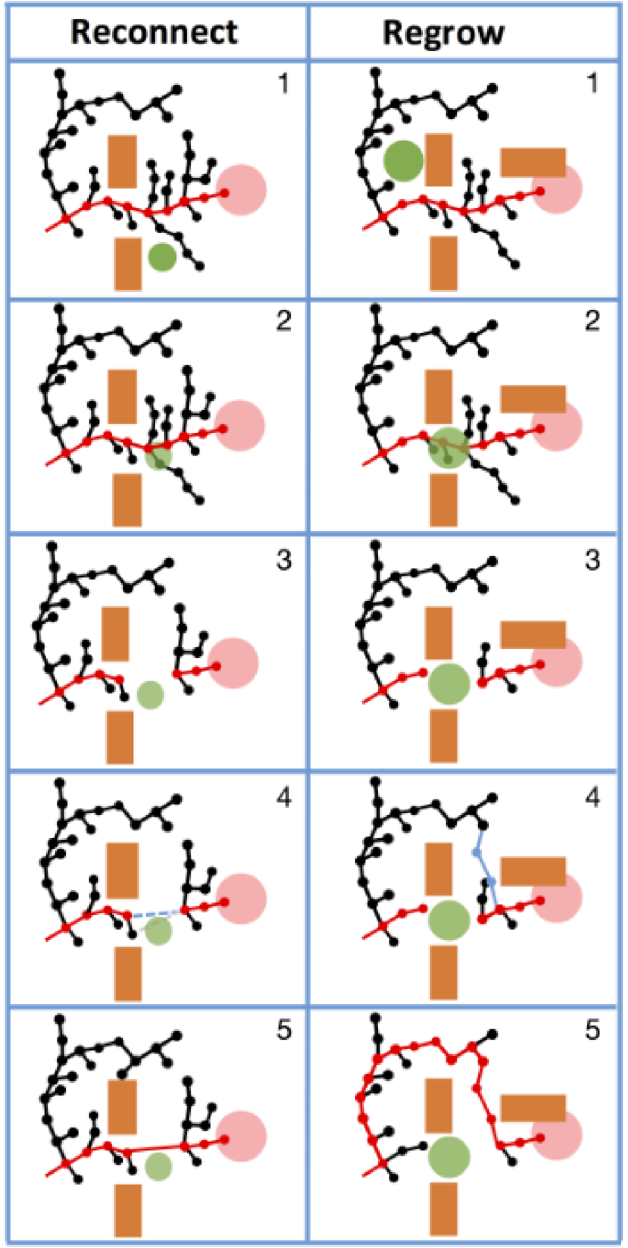


O. Adiyatov and H. A. Varol, "A novel RRT*-based algorithm for motion planning in Dynamic environments," 2017 IEEE International Conference on Mechatronics and Automation (ICMA), Takamatsu, Japan, 2017, pp. 1416-1421, doi: 10.1109/ICMA.2017.8016024. keywords:

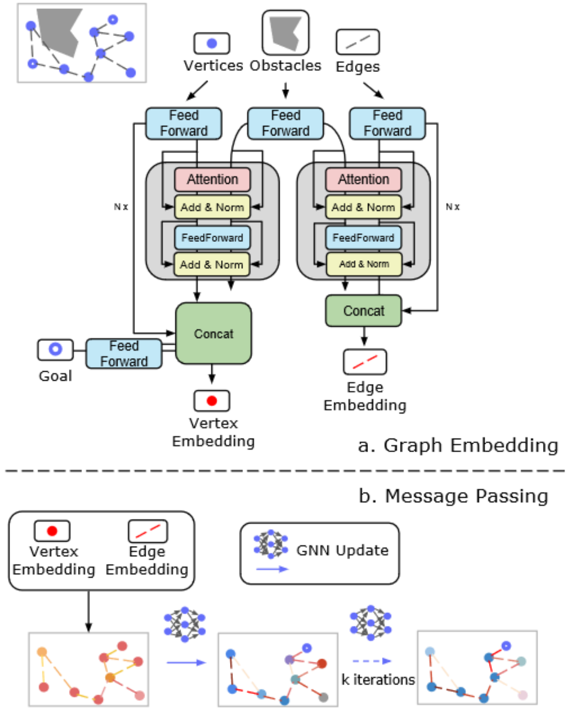
RRT*-FN-Dynamic

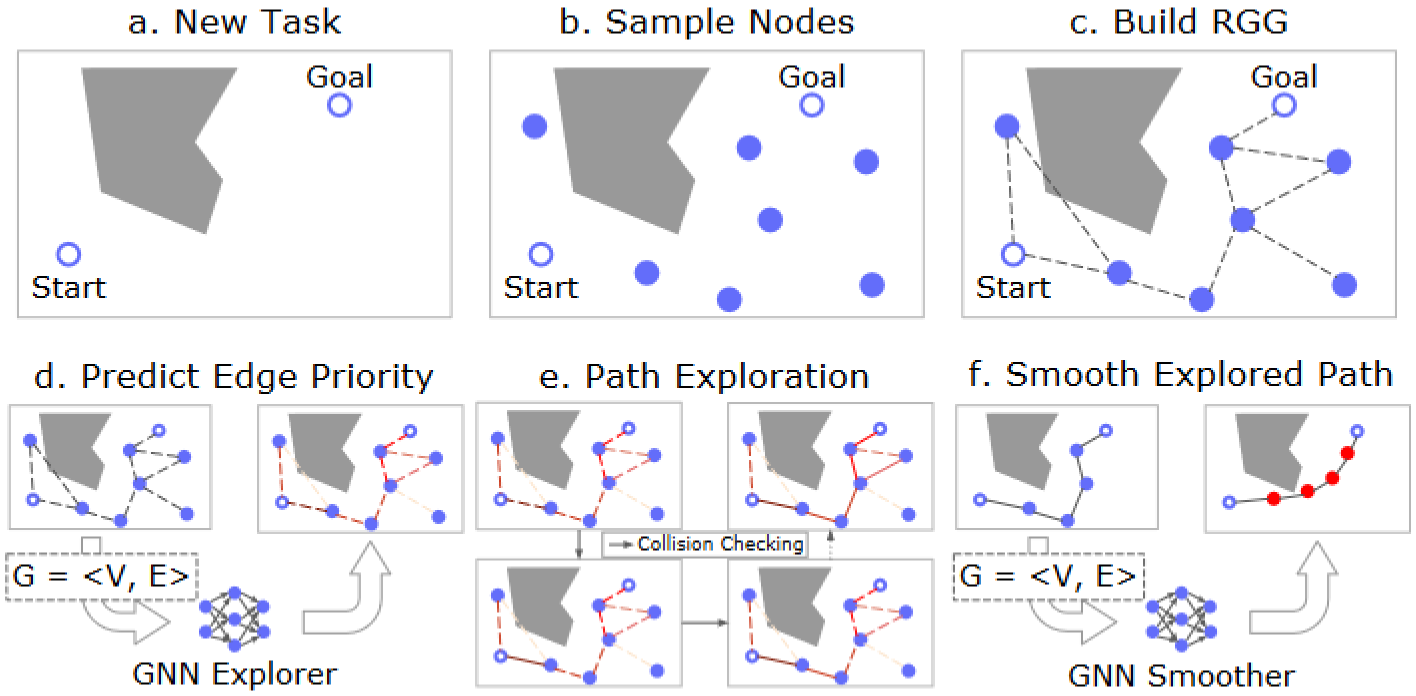
Algorithm 5 $\tau = (V, E) \leftarrow$ RRT*FND(p_{init})

```
τ ← RRT*FN(p_init) {Grow phase}
p_current ← p_init
σ ← SolutionPath(τ, p_current)
InitMovement()
while p_current ≠ p_goal do
    D ← UpdateObtacles()
    if DetectCollision(σ, p_current) then
        StopMovement()
        τ ← SelectBranch(p_current, τ)
        p_separate ← ValidPath(σ)
        ReconnectFailed ← true
        P_near ← Near(τ, p_separate)
        for p_near ∈ P_near do
            if ObstacleFree(p_near, p_separate) then
                τ ← Reconnect(p_near, p_separate, τ)
                ReconnectFailed ← false
                break
            end if
        end for
        if ReconnectFailed = true then
            τ ← Regrow(τ, p_separate, SetBias(σ_separate))
        end if
        σ ← SolutionPath(τ, p_current)
        ResumeMovement()
    end if
    p_current ← NextNode(σ)
end while
```



GNN-based neural planner*

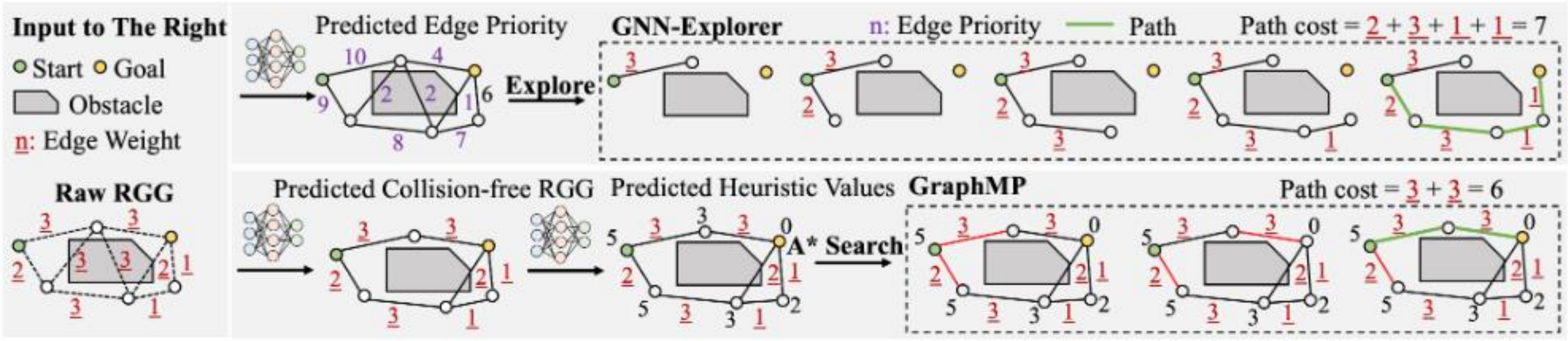




*Yu, C., & Gao, S. (2022, October 17). *Reducing collision checking for sampling-based motion planning using Graph Neural Networks*. arXiv.org. https://arxiv.org/abs/2210.08864

GraphMP*

Difference with GNN-based:



*Zang, X. *et al.* (2023) *GraphMP: Graph neural network-based motion planning with Efficient Graph Search, Advances in Neural Information Processing Systems*. Available at: https://papers.nips.cc/paper_files/paper/2023/hash/096961cae3c3423c44ea045aeb584e05-Abstract-Conference.html (Accessed: 21 June 2024).

GraphMP*

*Zang, X. *et al.* (2023) *GraphMP: Graph neural network-based motion planning with Efficient Graph Search, Advances in Neural Information Processing Systems*. Available at: https://papers.nips.cc/paper_files/paper/2023/hash/096961cae3c3423c44ea045aeb584e05-Abstract-Conference.html (Accessed: 21 June 2024).

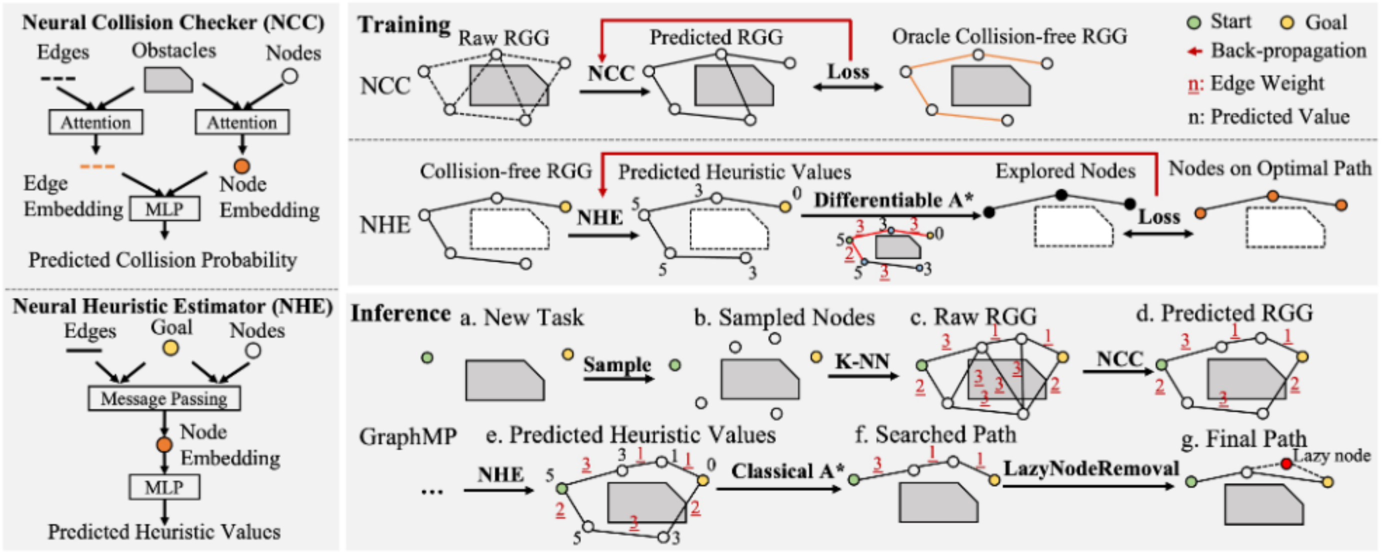
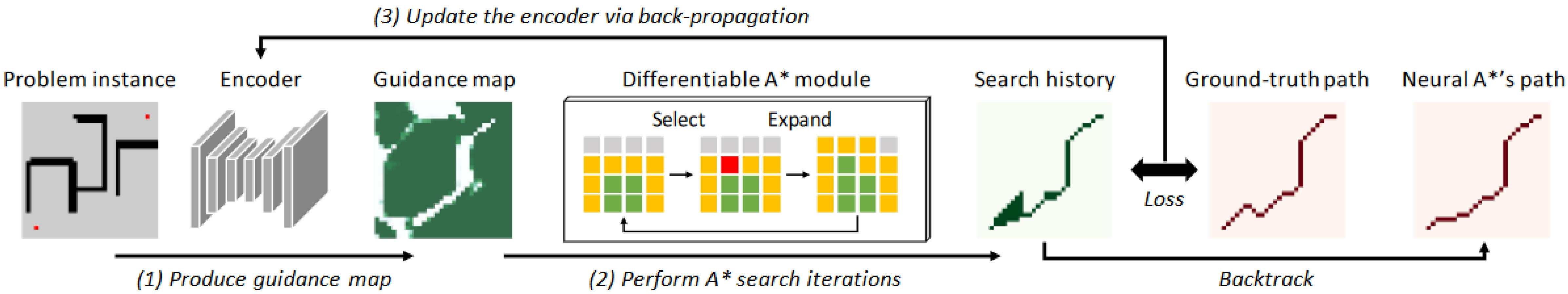


Figure 2: **(Left)**: The architectures of neural collision checker (**NCC**) and neural heuristic estimator (**NHE**). The neural collision checker takes a raw RGG and obstacles as input and predicts the collision status of all edges. The neural heuristic estimator takes the collision-free RGG and the goal as input and predicts the heuristic values of all nodes. **(Top right)**: The independent training phases of NCC and NHE. Notice that here we design and utilize a differentiable graph-based A* concatenated to NHE, enabling their joint training in an end-to-end manner. **(Bottom right)**: The main steps of the inference phase to solve the planning task. Once a valid solution is found, we perform the lazy node removal to further reduce the path cost.

Neural A* Search



Yonetani, R., Taniai, T., Barekatain, M., Nishimura, M., & Kanezaki, A. (2021, July 7). *Path planning using Neural A* search*. arXiv.org. https://arxiv.org/abs/2009.07476

For testing in the Duckiebot…

- We have RTT* implementation in simulation

- We have a repository with Neural A* Search simulation.

- We have the repository for GNN-based neural planner with simulation.

Ideas:

1. Learn how the sensors of the Duckiebot can be managed.

2. Learn how the Gym-Duckietown simulation works.

3. One step to begin once that is done, is trying to make RTT* (the easiest of the proposed algorithms) work in the Gym-Duckietown simulation. RTT* has already been implemented before in Duckiebot (https://www.youtube.com/watch?v=dS-TWh8cGXk) but repository not found.

4. If what proposed in step 4 works (also in the Duckiebot), we can proceed tryign to implement a more complex algorithm.

Problems and questions

- How much time does all that take? What is more reasonable and our limitations?
- Do we want a static or dynamic environment? (dynamic is more complex)
- etc