A Novel Narrow Region Detector for Sampling-Based Planners' Efficiency: Match Based Passage Identifier

Yafes Enes Şahiner[a], Esat Yusuf Gündoğdu[a] and Volkan Sezer[a]

[a]Smart and Autonomous Systems Lab., Istanbul Technical University, Istanbul, Türkiye

ARTICLE HISTORY
Compiled September 30, 2025

ABSTRACT
Autonomous technology, which has become widespread today, appears in many different configurations such as mobile robots, manipulators, and drones. One of the most important tasks of these vehicles during autonomous operations is path planning. In the literature, path planners are generally divided into two categories: probabilistic and deterministic methods. In the analysis of probabilistic methods, the common problem of almost all methods is observed in narrow passage environments. In this paper, a novel sampler is proposed that deterministically identifies narrow passage environments using occupancy grid maps and accordingly increases the amount of sampling in these regions. The codes of the algorithm is provided as open source. To evaluate the performance of the algorithm, benchmark studies are conducted in three distinct categories: specific and random simulation environments, and a real-world environment. As a result, it is observed that our algorithm provides higher performance in planning time and number of milestones compared to the baseline samplers.

KEYWORDS
autonomous robots; path planning; sampling-based algorithm; probabilistic roadmap method; narrow passage problem

1. Introduction

Path planning is a core component of autonomous systems. Path planning methods for robotic systems have been traditionally approached through two fundamental methodologies: deterministic and probabilistic. A* [1], Dijkstra [2], JPS [3], and Visibility Graph are some of the deterministic planners. These approaches are classified as deterministic due to their consistency in providing identical solutions to identical problems. In addition to deterministic methods, sampling-based methods like PRM (Probabilistic Roadmap) [4] and RRT (Rapidly-exploring Random Trees) [5], have been developed. These methods provide different solutions to the identical problem each time. However, these sampling-based algorithms have demonstrated efficiency, particularly in high-dimensional configuration spaces like robotic manipulators. A detailed description and comparative review of sampling-based algorithms can be found in [6].

PRM consists of two parts: the build phase and the query phase. In the build phase, the configuration space is sampled independently from the initial and target points,

CONTACT Yafes Enes Şahiner Email: sahiner18@itu.edu.tr

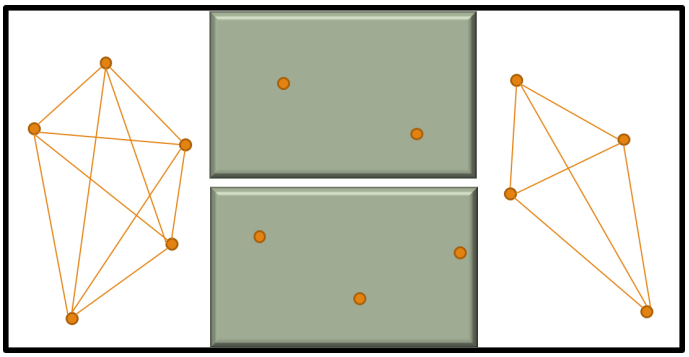


Figure 1.: Connectivity problem in narrow passage environment

then these nodes are connected with a suitable local planner to create a roadmap. The k-nearest neighbor approach can be used to determine the nodes to be connected. Each node is linked with a certain number of nodes nearest to it. This stage creates a roadmap that can be used for different initial and target points. However, the roadmap must be reconstructed or modified at certain time intervals in dynamic enviroments. In the query phase, the path is calculated on the roadmap with a graph solver such as A* or Dijkstra.

There are two characteristics that evaluate the effectiveness of PRM roadmaps: coverage and connectivity [7]. Coverage refers to whether the roadmap is spread over the entire map. If there is a roadmap with high coverage, a new sample can be directly connected to roadmap with a local planner. Connectivity is the ability to calculate a path between any two samples selected from roadmap. This means that the number of subgraphs is low in the roadmap with strong connectivity. Although PRM is generally efficient, its connectivity may not be ideal in environments with narrow passages as shown in Figure 1. Because when uniform sampling is performed the probability of sampling narrow passages is poor due to the small volume they occupy in the map. In such environments, both planning time and number of milestones of PRM are much higher.

To solve these drawbacks, narrow regions need to be prioritized. If narrow regions can be identified, PRM performance can be improved by sampling the relevant regions. In [8], it is stated that narrow passage detection is an more challenging task than motion planning. In this paper, a novel method that deterministically detects narrow regions using an occupancy grid map is presented. This method aims to improve connectivity of probabilistic roadmaps by weighted sampling of detected narrow passages according to their narrowness. However, it is not possible to produce roadmaps with good coverage by only sampling narrow passages. Therefore, in addition to narrow passage sampler, uniform sampler is used in a hybrid approach. As a result, roadmaps that have strong coverage and connectivity properties are obtained. We have tested our work in both simulation and real-world environments and compared it with other methods. As a result of the experiments, it has been observed that our algorithm is outperforming other methods in terms of planning time and number of milestones in roadmap because milestones generated by other samplers are not guaranteed to increase connectivity of roadmaps. Sampling around obstacles or dead-end regions causes more samples to calculate a valid path, which leads to worse results in terms of planning time and number of milestones. In addition, path length is usually increased as a result of the generation of a path over the samples in these areas. On the other

hand, Match Based Passage Identifier directly detects and samples narrow regions, so that it provides samples that increase connectivity of roadmaps.

To summarise, the major contributions of this paper are as follows:

- A novel algorithm that deterministically detects narrow passages is presented.
- A weighted sampler is proposed that generates milestones according to the narrowness of the regions on the map.
- The method is tested on custom and randomly generated maps and compared in the open source Open Motion Planning Library (OMPL) environment [9].
- The real-time applicability of the algorithm is shown through real-world experiments.
- The implementation codes of the algorithm is shared as open source [10].

The rest of the paper is organized as follows. Section 2 presents the related works on narrow passage problem. Section 3 describes the technical details of the method. Section 4 compares the performances of our algorithm and other methods with experiments. Section 5 concludes the paper and summarises the results. In addition, future work is provided for the further development of the study.

2. Literature Review

Various methods have been developed to prevent narrow passage problem encountered when using probabilistic roadmap method. The common point of almost all methods is to provide a connection on the narrow passage with a sampler that increases number of milestones in these areas.

Obstacle Based PRM (OBPRM) [11] is a sampler that aims to increase the number of samples at the boundaries of obstacles. For each sampling, a random point and a random direction are selected in the environment. Collision checking is performed from a random point until an obstacle is encountered in a random direction. The point before the collision occurs is considered as the milestone in the graph. Uniform Obstacle Based PRM (UOBPRM) [12] is a variant of OBPRM. In addition to sampling at the boundaries of the obstacles, is to distribute the samples homogeneously within the boundaries of the obstacles. To achieve this, line segments of a certain length are randomly placed on the map and a collision check is performed on this line segment at a certain interval of steps. With each validity change, it is added to the graph as a milestone. Performing additional collision checks during milestone determination for both of these increases the sampling time.

The method that can sample close to the obstacle boundaries without performing additional collision checks is Gaussian Sampler [13]. This method selects a random point for each sampling operation and a second point within a certain distance from the first point. If one of the two points is valid and the other is invalid, the valid point is added to the graph. Creating milestones at obstacle boundaries is similarly seen in retraction-based methods. Invalid samples are moved to the contact surface. In [14], the aim is to move the invalid sample to the nearest valid point. For this, the obstacle surface is sampled until the point that minimizes the distance locally is found. All sampled points are included in the graph. The methods described in [11], [12], [13], and [14] are intended for sampling at obstacle boundaries, as stated. Planning through these regions is not the best for the safety of the moving robot.

On the other hand, Medial Axis PRM (MAPRM) [15] is a planner that gives importance to safety and creates a route away from obstacles. First, it calculates the

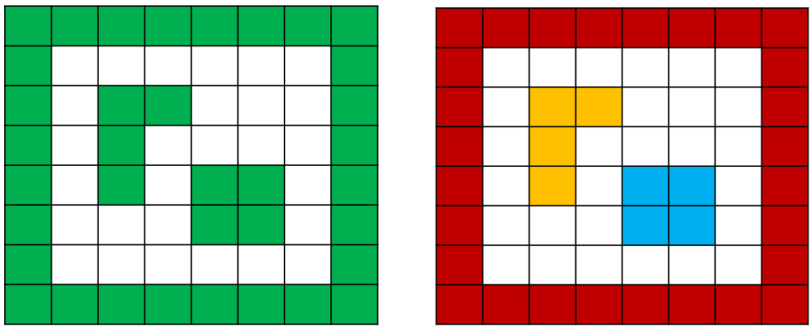
medial axis of the environment and then makes samples on the medial axis. However, calculating the medial axis consumes a certain amount of time. Although the methods described in [11], [12], [13] and [15] increase the number of samples in narrow passages compared to the uniform sampler, they are not intended for sampling directly on the narrow passages.

Randomized Bridge Builder (RBB) [16] is developed to provide intensive sampling within narrow passages. The method, known as the Bridge Test, uses three samples to create each milestone. Certain conditions must be met to create a milestone. The first sample should be on the obstacle, the second sample selected close to the first sample should be on the obstacle, and the third sample, calculated as the midpoint of the first two samples, should be on free space. RBB creates a milestone by doing with three samples what the Gaussian sampler does with two samples. Even though it samples more, it manages to concentrate more on narrow passages. The downside of the Bridge test is that it cannot distinguish between dead-end regions and narrow passages because it analyzes a local area of the map. In addition, Bridge Test is used in different planners. One of these is Learning Multi RRT (LM-RRT) [17]. This method samples repeatedly using RBB. Then, it clusters these samples and starts an RRT tree in each cluster. The selection of the tree to be grown is analogized to the multi-armed bandit problem. This problem is handled by ϵ_t-greedy, which is a reinforcement learning method.

Furthermore, there are different planners that address narrow passage problem with RRT methods. One of these is Spark PRM [18], which uses PRM and RRT hybridly. In this method, samples are made initially using PRM, and a graph is created. If a connectivity problem occurs, subgraphs that are connected with a low number of milestones are observed. In this case, Spark PRM starts RRT from a point belonging to the subgraph consisting of a small number of milestones. Thanks to the growing RRT tree, it is aimed to improve connectivity with other graphs. Additionally, a utility-guided approach to RRT [19] is proposed, where the expansion strategy is adapted to maximize expected utility, which depends on the selected nodes, their expansion direction, and expansion distance. Moreover, different methods are combined with PRM. An example of this is the hybrid use of PRM with the approximate cell decomposition [20]. It is aimed to ensure the connection of semi-filled cells with each other with PRM milestones. Thus, it is aimed to improve the connection feature in environments with narrow passages. Another planner that aims to increase sampling in narrow passage areas is the Toggle PRM [21]. In this method, two graphs are created, one of these grows on obstacles and the other in free space. While one of the graphs is growing, if an invalid sample is detected, it is added to the other graph.

Narrow passage problem is seen in multi-level motion planning. The method [22] is developed for high-dimensional motion planning problems by first projecting the configuration space into lower dimensions to obtain a base path. This base path then serves as an admissible heuristic for the complex robot model, allowing for efficient exploration of narrow passages. In addition, a method based on Minkowski sum [23] utilizes unions of ellipsoids to parameterize C-space obstacles in closed form, allowing the use of prior knowledge to avoid sampling invalid configurations for high-dimensional motion planning.

There are learning-based approaches to solve narrow passage problem. Sample Driven Connectivity Learning (SDCL) [24] is a new learning-based method for narrow passage problem. In this method, firstly, the regions that make PRM roadmaps difficult to connect are learned and then these regions are sampled. In the learning phase, the samples are divided into two the ones connected to goal point and the others,



(a) Occupancy grid map (b) Obstacle components

Figure 2.: Clustering of obstacle units

and then the binary classifier is trained with support vector machine. In the sampling stage, the region separating two classes is sampled and this process is repeated until a valid path is found, thus accelerating the connection process. Moreover, the method [25] focuses on learning sampling distributions. It uses a conditional variational autoencoder (CVAE) to sample from the latent space, conditioned on the specifics of planning problem, and prioritizes regions where optimal motion plans are likely to be found.

There are many studies on the solution of narrow passage problem. To make a fair comparison between our method and existing methods, OMPL is used. However, OMPL does not contain the implementation of all the algorithms described. The benchmark is performed with the Bridge Test, Gaussian, Obstacle-based, and Uniform samplers available in OMPL. Section 4 provides the details and results of the experiments.

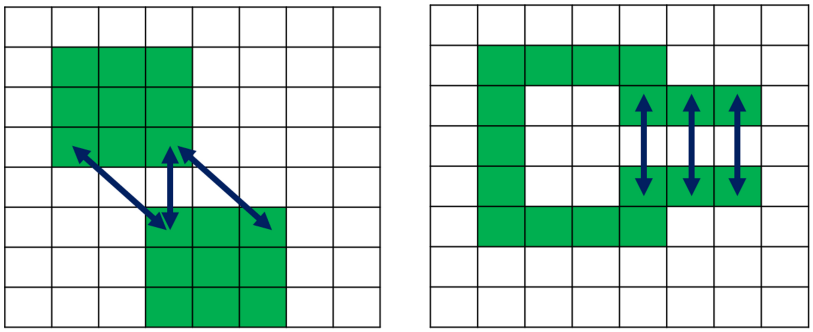
3. Technical Approach

In this section, the technical details of the Match Based Passage Identifier that detects the narrow passages in grid-based environments are given. Then, the details of the weighted sampling on the detected narrow passages are explained.

3.1. Passage Identification Algorithm

Match Based Passage Identifier comprises four steps: finding connected components, calculating foreign matches, calculating self-matches, and collision checking of matches. These stages are outlined in the pseudocode presented in Algorithm 1. To understand the algorithm, it is essential to clarify the terminology employed. The process operates on an occupancy grid map, where every cell shown as empty is termed a *free space unit*. These free space units, when adjacent to each other, form a structure called a *free space component*. Similarly, each cell marked as occupied on the map is referred to as a *obstacle unit*, and the structure resulting from the adjacency of such obstacle units is referred to as a *obstacle component*. For instance, the occupancy grid map shown in Figure 2a is clustered, with the resulting obstacle components shown in different colors in Figure 2b.

At the core of the algorithm is the concept of an *obstacle match*, which denotes a pair of obstacle units that potentially represent the start and end points of a narrow pas-



(a) Foreign matches (b) Self matches

Figure 3.: Obstacle matches

sage on the map. Obstacle matches are of two types: *foreign matches* and *self matches*. A *foreign match* is a match between obstacle units from different obstacle components, while a *self-match* is a match between obstacle units within the same obstacle component. Examples of these match types are shown in Figure 3, with specific instances of foreign matches shown in Figure 3a and self-matches shown in Figure 3b. Although there are more matches in the occupancy grid maps provided, only three examples are shown. The detailed methodology for calculating these matches is explained in the following sections.

Additionally, the algorithm employs a *passage value matrix*, a two-dimensional data structure equal to the dimensions of occupancy grid map. Initially, all elements of this matrix are set to the value of the shorter side of the map, resulting in a data structure that contains narrow passage lengths.

Algorithm 1 Narrow Passage Detection Algorithm

```
1: foreignMatches=[ ]; selfMatches=[ ]
2: components=findConnectedComps(map,obstacle)
3: foreignMatches=foreignMatcher(components)
4: for component in components do
5:     selfMatches.append(selfMatcher(component))
6: end for
7: passageValues=collisionCheck(foreignMatches ∪ selfMatches)
8: return passageValues
```

3.1.1. Finding Connected Components

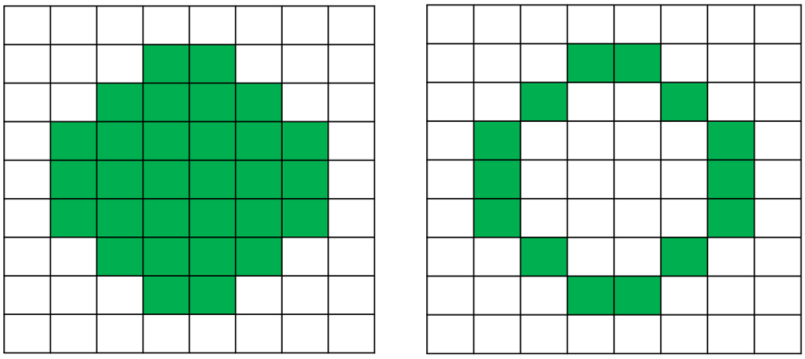
In this step, obstacle components consisting of obstacle units that are connected to each other are calculated via *findConnectedComps()* function. There are methods developed for this purpose [26].

3.1.2. Calculation of Foreign Matches

The method of determining foreign matches is that each obstacle unit is matched with the closest obstacle unit that is not in its self component. If there is only one obstacle component on the map, it means that no foreign matches can be made. The pseudocode showing the stages of the algorithm is given in Algorithm 2.

Algorithm 2 foreignMatcher

```
1: foreignMatches=[ ]
2: borderedComponents=border(components)
3: for component in borderedComponents do
4:     for unit in component do
5:         nearest=nearestInOtherComponents(unit)
6:         dist=getDistance(unit,nearest)
7:         if dist ≤ maxDistParam then
8:             foreignMatches.append({unit,nearest})
9:         end if
10:    end for
11: end for
12: return foreignMatches
```



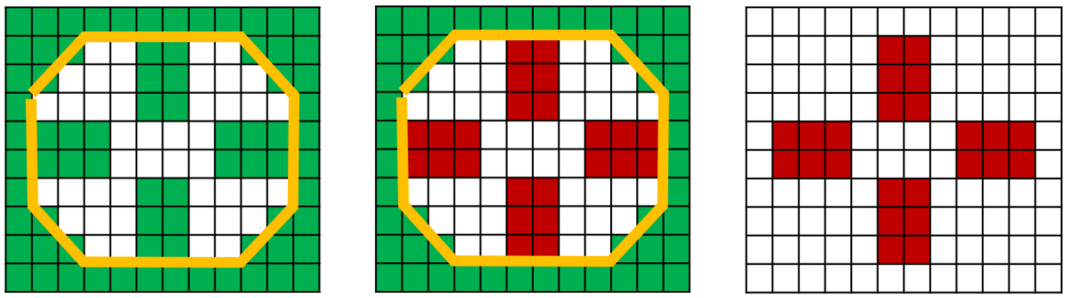
(a) Component (b) Bordered component

Figure 4.: Obtaining boundary of the component

The first step of the *foreignMatcher()* function is to determine the boundaries of components adjacent to free space units. This step is done by using the function called *border()*. In this way, the number of units of the components is reduced and the running time of the algorithm is considerably shortened. Reducing the number of units with the *border()* function does not affect the result because it is not possible for an obstacle unit that is not adjacent to the free space to form the starting or ending point of a narrow passage. An example of this operation can be seen in Figure 4.

The second step of the *foreignMatcher()* is to find the closest obstacle unit for each unit of the bordered component, but the closest obstacle unit must be a unit of other components. *nearestInOtherComponents()* function is used to select each obstacle unit closest to itself among the obstacle units belonging to other components. This function takes the obstacle unit as input whose nearest neighbor is calculated. This operation is applied for each obstacle unit in each obstacle component. These obstacle units are grouped with their matching units and added to the *foreignMatches* list.

The reason for the distance control is to reduce the number of matches given to the *collisionCheck()* function thus decreasing the running time of the algorithm. Additionally, with this filtering process, the matches focus on narrow passages. The filtering process is performed with *maxDistanceParam*.



(a) Calculated convex

(b) Separation of inside and outside units

(c) Extracted components

Figure 5.: Process of extracting components

3.1.3. Calculation of Self Matches

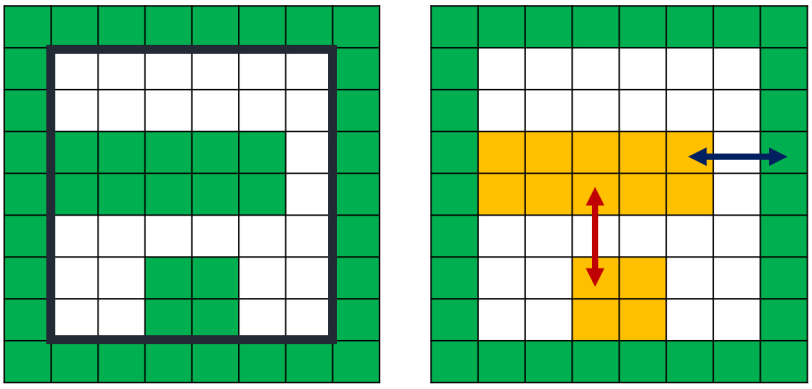
Complex-shaped obstacle components may have narrow passages within them. Therefore, only foreign matches are not enough to find all narrow passages in the environment. Narrow passages within the same component are calculated by using the *selfMatcher()* function. The general strategy of the function is extracting parts from the component and calculating foreign matches using those parts. The pseudocode of the *selfMatcher*() function is given in Algorithm 3.

Algorithm 3 selfMatcher

```
1:  selfMatches = [ ]
2:  cMap = createMap(component)
3:  freeComps = findConnectedComps(cMap, free)
4:  for freeComponent in freeComps do
5:      convex = convexHull(freeComponent)
6:      [insidePoints, outsidePoints] = separatePoints(component, convex)
7:      insideComps = findConnectedComps(insidePoints, obstacle)
8:      selfMatches.append(foreignMatcher(insideComps))
9:      selfMatches.append(foreignMatcher({insidePoints, outsidePoints}))
10:     for comp in insideComps do
11:         selfMatches.append(selfMatcher(comp))
12:     end for
13: end for
14: return selfMatches
```

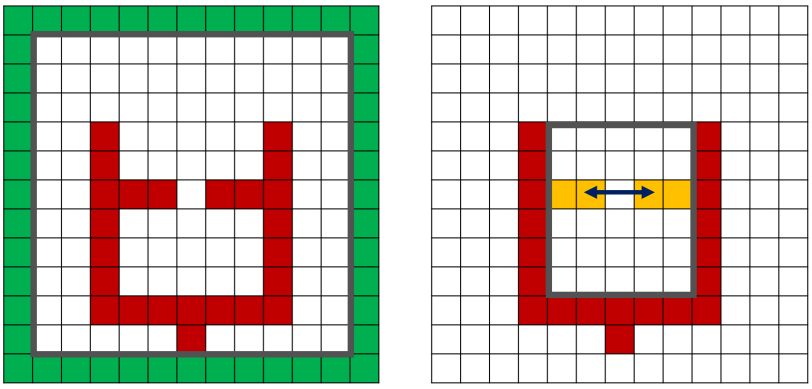
Initially, the smallest size map containing only the given component is created with *createMap()* function. Then, free space units are clustered to get free space components in the created map. After that, the extracting process is performed for each free space component. To determine the parts to be extracted, the procedure initiates with calculating a convex polygon containing the free space component via the *convexHull()* function. Subsequently, the obstacle units inside and outside the boundaries of the convex polygon are determined with *separatePoints()* function. The process of extracting parts from the component is given in Figure 5.

After the obstacle units within the polygon are determined, the *foreignMatcher()* function is used twice. The first usage of the *foreignMatcher()* calculates matches between the extracted parts from the component. The second usage calculates matches



(a) Calculated convex (b) Types of self matches

Figure 6.: Foreign matcher usages in self matcher



(a) Extracting component (b) Extracting children of the component

Figure 7.: Recursively extracting parts from the component

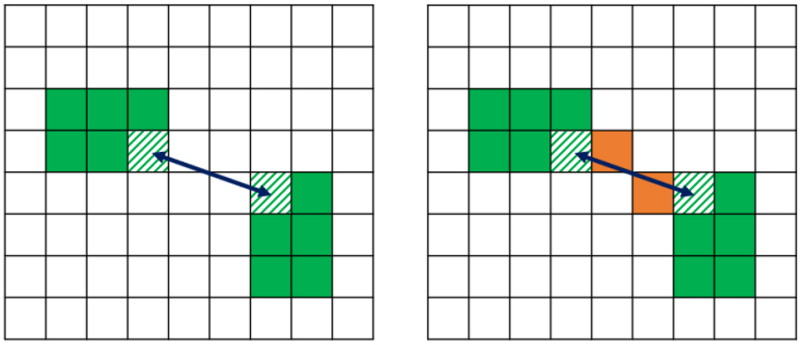
between inside and outside obstacle units. These two use cases are given in Figure 6. An example of a match made between the extracted parts is shown in red. Additionally, an example match between the inside point and the outside point is shown in blue.

After using *foreignMatcher()* function, *selfMatcher()* function is called recursively for each part extracted because the part may also contain a narrow passage within itself. An example of a narrow passage inside the extracted part is given in Figure 7. In summary, *selfMatcher()* function is a function that finds narrow passages recursively within the component itself. The end condition of recursion is that there are no parts left to be extracted.

3.1.4. Collision Checking of Matches

In this stage, *collisionCheck()* function is used to inspect the cells between matched units and in case of a valid match, the passage value matrix is updated. First, a line is drawn between two obstacle units using the Bresenham Algorithm [27]. Then, the points crossing the line on the occupancy grid map are checked. If all of them are free space units, it is considered to be a narrow passage. This process is repeated for each obstacle match.

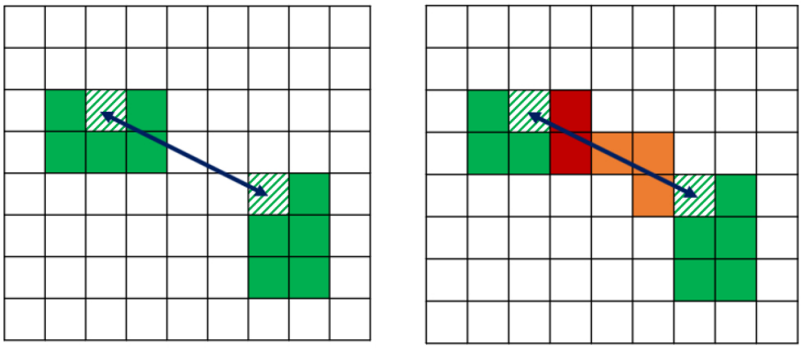
A foreign match is given in Figure 8a. In Figure 8b, the units between this match are



(a) Matching two units in different components

(b) Inspection of the cells in between

Figure 8.: Valid collision check



(a) Matching two units in different components

(b) Inspection of the cells in between

Figure 9.: Invalid collision check

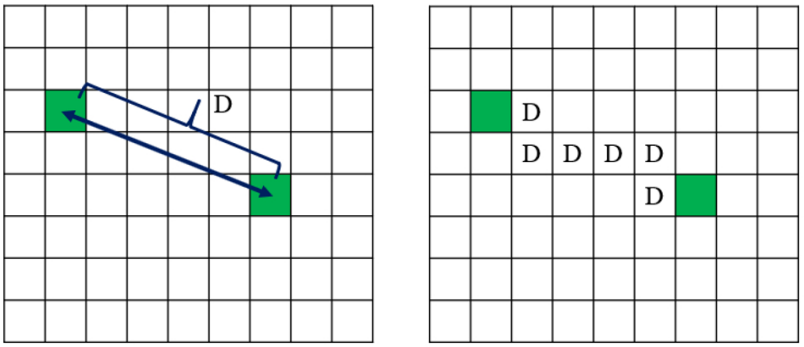
determined and it can be seen that they are all on the free space. The given example in Figure 8 is a valid match. Figure 9 shows an example of an invalid match. The test returns collision because there are obstacle units shown in red between the matches. Therefore, this match is not considered a narrow passage.

In cases where the *collisionCheck()* is successful, the passage value matrix needs to be updated. The passage value matrix is a matrix that is the same size as the occupancy grid map. In this matrix, given in Figure 10, the distance (D) value between the matches is placed on all coordinates on the line drawn between the matches. The passage value matrix obtained by calculating all matches on a sample map is shown in Figure 11.

There are values already placed in these units. Consequently, the previous distance value is compared with the current distance value. The smaller value is valid since the narrowest passage containing the relevant coordinate is sought to be determined.

3.2. Sampling Strategy

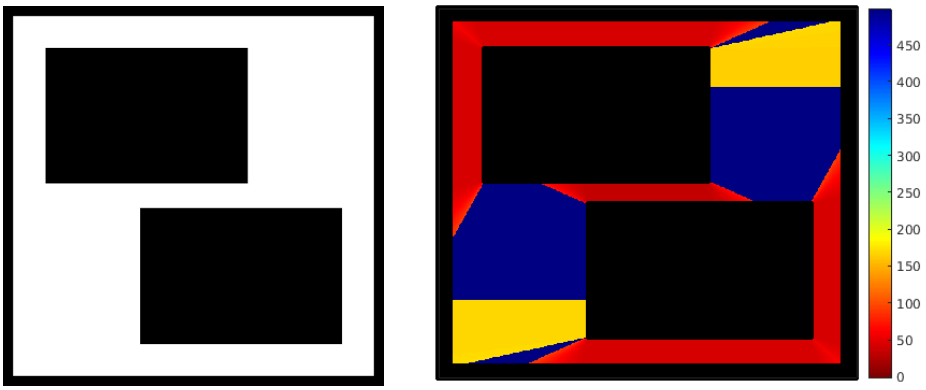
The method presented is based on the detection of narrow passages and guided sampling of these regions. The detection of the narrow passages is explained in Section 3.1. The sampling strategy is explained by using the narrow passage value matrix.



(a) Calculation of the distance between two units

(b) Replacing the values in the passage value matrix

Figure 10.: Updating passage value matrix



(a) The map to identify narrow passages

(b) Visualization of the passage value matrix

Figure 11.: Detection of narrow passages on the map and obtaining the passage value matrix

Initially, the passage value matrix A is normalized so that the sum of its elements is equal to one. The result is the two-dimensional discrete probability density function P. This process is given in Equation 1. Here, the $m \times n$ dimensional matrix A represents the passage value matrix where a_{ij} is the i^{th} row j^{th} column element of A. Also, the value S is the sum of the elements in matrix A. Each element a_{ij} in A is divided by S to find each element p_{ij} in the probability density function P.

$$\begin{aligned} S &= \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij} \\ p_{ij} &= \frac{a_{ij}}{S}, \quad \text{for } i = 1, \ldots, m \text{ and } j = 1, \ldots, n \end{aligned} \tag{1}$$

Subsequently, weighted sampling is performed using the density function P calculated in Equation 1. For this purpose, the cumulative distribution C is obtained first, given in Equation 2.

$$\begin{aligned} c_k &= \sum_{x=1}^{i-1} \sum_{y=1}^{n} p_{xy} + \sum_{y=1}^{j} p_{iy}, \\ &\text{for } k = 1, \ldots, m \times n \\ &\text{where } i = \left\lfloor \frac{k}{n} \right\rfloor \text{ and } j = k \bmod n \end{aligned} \tag{2}$$

In the $C = \{c_1, c_2, \ldots, c_n\}$ cumulative distribution array, c_k represents the k^{th} element value of the array and $c_n = 1$. To determine the index of the cell being sampled, a random variable U between 0 and 1 is selected. Then, the first value c_k greater than U is found as in Equation 3. Here, the value k represents the index to be sampled.

$$k_{selected} = \min\{k : U \leq c_k, k = 1, 2, \ldots, n\} \tag{3}$$

The process of converting the index selected from the cumulative distribution back to the 2D position on the map is given in Equation 4.

$$i = \left\lfloor \frac{k_{selected}}{n} \right\rfloor, \text{ and } j = k_{selected} \bmod n \tag{4}$$

Narrow passage sampling is performed as described in the given equations using the passage value matrix. However, narrow passage sampling alone is not sufficient to cover the map for PRM. Similarly, uniform sampling alone is not sufficient. Therefore, a hybrid sampling method is preferred to combine strengths of both.

To determine the ratio between uniform and narrow passage sampling, a series of simulations is conducted in which different proportions were investigated. In Table 3, results on varying hybrid sampling ratios are provided. It is observed that each map exhibited a distinct optimal proportion. The hybrid sampling technique is implemented at a rate of 1:1 to make it simple. In this way, with the uniform sampler, the roadmap

is spread more, and with the narrow passage sampler, critical regions are sampled and subgraphs are connected.

MBPI works on the idea that narrow passages can be identified through occupancy grid map during discretization. The ability to detect these passages depends on the grid's resolution. If the resolution is very high, narrow passages can be detected and prioritized during sampling. However, a very high resolution increases computational cost, prolonging the identification process. Conversely, if the resolution is too low to capture the details of narrow regions, the detection algorithm cannot identify them, causing the narrow passage sampler to behave like a uniform sampler. In this case, the hybrid sampling strategy simply becomes a uniform sampling strategy.

4. Experiments

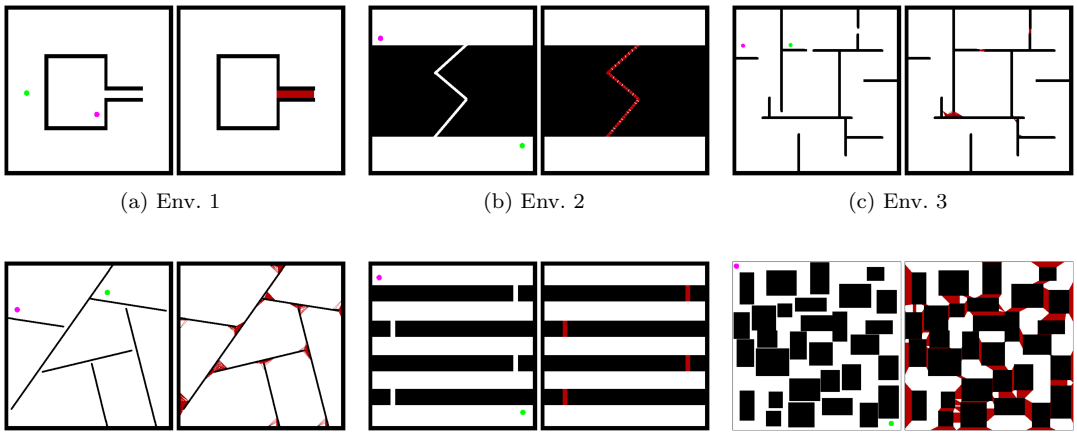
Three types of experiments are presented. Firstly, the benchmark environment to compare MBPI with the existing methods is introduced. Then, the results obtained in simulation and real-world test scenarios are given.

4.1. Benchmark Environment

The Open Motion Planning Library (OMPL) [9] is an open-source library that contains many sampling-based algorithms. This library provides a platform that allows comparing the performance of algorithms. MBPI is integrated by adding a new sampler to the OMPL library to compare with other methods [10]. Within the scope of the experiments, the performance of the algorithm is compared with Bridge Test, Gaussian, Obstacle-based, and Uniform samplers within OMPL. The parameters of these samplers are kept constant at their default values in OMPL. During this evaluation process, PRM is used with default parameters in OMPL. *maxDistanceParam*, which determines the maximum narrow passage width of the detection algorithm, is kept constant at 0.05 (i.e. 5% of the map size) in all experiments. Performance analysis is carried out through the metrics of planning time, number of milestones in the roadmap, and path length which is obtained by the solution length parameter of the OMPL. The system and software configurations used in the experiments are summarized in Table 1.

Table 1.: System and Software Configurations Used in Experiments

Component	**Detail**
RAM	64GB
Processor	Intel Core i7-10700F CPU @ 2.90GHz x 16
Graphics Card	NVIDIA GeForce GT 610
Operating System	Ubuntu 20.04.6 LTS (64-bit)
OMPL Version	1.6.0
ROS Version	ROS Noetic
Robot Model	Turtlebot3 Burger



(a) Env. 1 (b) Env. 2 (c) Env. 3

(d) Env. 4 (e) Env. 5 (f) Env. 6

Figure 12.: The environments used in the experiment (left) and the detected narrow passage zones by MBPI, marked in red (right). The initial and target points in each environment are indicated by magenta and green markers respectively.

4.2. Test Scenarios and Results

The performances of MBPI and the existing methods are compared in 3 different types of experiments.

4.2.1. Specific Map Experiments

MBPI and existing methods are compared for specific narrow passage scenarios. All of the prepared maps are 500×500 and are given in Figure 12. In addition, the regions detected as narrow passages are labeled. The experiments are repeated 500 times for the selected initial and target points in each environment and the statistics of these experiments are given in Table 2. Additionally, the milestones generated by the samplers in one experiment for each environment are given in Figure 13. The results obtained for each map can be interpreted as follows:

- Environment 1 is a bottle-like structure with a narrow gate. In this map, MBPI successfully detected the critical region that may affect the connectivity problem as shown in Figure 12a. Thus, narrow passage samples directly affect the performance of the algorithm positively. As seen in Table 2, MBPI gives the best result in planning time metric when compared with other methods. The samples generated by the methods are given in Figure 13a. Here, Bridge Test method generated samples in dead-end regions besides the critical narrow region. OBPRM and Gaussian, also can be seen in Figure 5, sample obstacle boundaries instead of narrow regions. This problem of Gaussian and OBPRM is common to all environments. For these reasons, our method outperforms the other methods.
- Environment 2 is a narrow zigzag corridor map. In this map, MBPI successfully detected the long thin corridor, which is the critical region that may affect the connectivity problem in the map, as shown in Figure 12b. As seen in Table 2, MBPI gives the best results in planning time and number of milestones metrics when compared with other methods. In this environment, Bridge Test's performance underperformed MBPI. The reason for this, the bridge test method

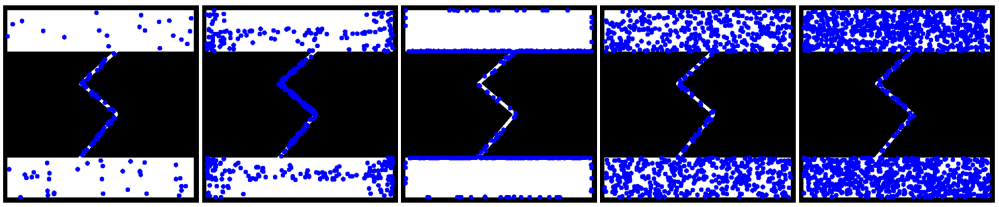
Table 2.: Performances of each sampler on each environment. Narrow passage detection time of MBPI is given in parenthesis for each environment in the sampler column.

Env. (Fig.)	**Sampler**	**Metrics** (Avg.(Std. dev.))		
		Planning Time (s)	Number of Milestones	Path Length (m)
1	MBPI (0.126s)	0.0060(0.0022)	224.15(95.01)	18.77(1.59)
	Bridge Test	0.0091(0.0145)	205.97(518.98)	22.40(3.24)
	OBPRM	0.0333(0.0262)	1604.50(1483.21)	21.80(5.31)
	Gaussian	0.0140(0.0116)	529.38(639.58)	18.54(2.43)
	Uniform	0.0127(0.0084)	439.77(434.16)	17.67(1.59)
2	MBPI (0.264s)	0.0046(0.0015)	160.08(68.86)	24.18(1.74)
	Bridge Test	0.0136(0.0290)	511.31(1518.74)	25.82(2.02)
	OBPRM	0.0216(0.0073)	1197.45(537.76)	22.53(0.74)
	Gaussian	0.0197(0.0076)	939.31(559.35)	22.96(1.02)
	Uniform	0.0299(0.0134)	1731.24(1108.67)	22.67(0.78)
3	MBPI (0.236s)	0.0054(0.0022)	239.58(125.72)	15.30(4.58)
	Bridge	0.0102(0.0060)	279.77(215.85)	33.31(9.96)
	OBPRM	0.0128(0.0102)	599.55(613.03)	37.21(15.38)
	Gaussian	0.0057(0.0032)	172.53(133.42)	36.00(6.50)
	Uniform	0.0059(0.0025)	192.18(100.58)	32.48(8.63)
4	MBPI (0.308s)	0.0072(0.0028)	306.95(159.18)	34.68(2.01)
	Bridge PRM	0.1059(0.0861)	4168.47(3542.68)	34.17(2.39)
	OBPRM	0.0203(0.0155)	1130.24(1042.75)	38.22(3.37)
	Gaussian PRM	0.0667(0.0485)	4123.15(3503.4)	33.13(1.64)
	Uniform PRM	0.0541(0.0384)	3195.84(2732.78)	33.41(1.75)
5	MBPI (0.254s)	0.0059(0.0024)	312.22(137.32)	53.75(1.17)
	Bridge Test	0.0111(0.0061)	466.72(354.17)	54.94(1.26)
	OBPRM	0.0331(0.0224)	1932.30(1642.11)	55.99(2.98)
	Gaussian	0.0128(0.0082)	565.74(492.67)	54.48(1.60)
	Uniform	0.0139(0.0078)	618.66(467.28)	54.18(1.52)
6	MBPI (0.387s)	0.0057(0.0021)	316.45(119.96)	24.29(2.95)
	Bridge Test	0.0094(0.0051)	499.52(314.16)	24.13(3.32)
	OBPRM	0.0261(0.0289)	1846.79(2359.44)	23.05(2.485)
	Gaussian	0.0076(0.0043)	393.83(257.52)	25.30(3.68)
	Uniform	0.0079(0.0041)	413.68(247.24)	25.35(3.81)

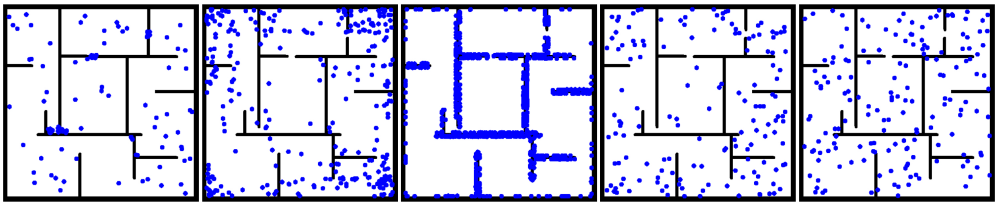
Table 3.: Number of Milestones Under Varying MBPI Hybrid Sampling Ratios (Uniform:Narrow)

Env.	**Number of Milestones** (Avg. (Std. dev.))				
	5:1	3:1	1:1	1:3	1:5
1	**73.12**(**41.55**)	80.27(57.82)	79.67(47.19)	144.56(91.96)	212.67(141.31)
2	254.93(129.56)	188.63(104.32)	114.14(54.52)	**93.62**(**48.61**)	129.30(97.03)
3	81.38(55.34)	**71.53**(**43.05**)	108.09(139.10)	280.63(413.6)	374.86(459.91)
4	322.64(166.63)	257.28(122.68)	**211.17**(**173.06**)	257.17(217.49)	510.52(646.80)
5	179.84(76.34)	**173.01**(**66.79**)	238.51(102.37)	474.46(264.27)	686.21(276.88)
6	320.51(151.13)	285.15(124.50)	**249.90**(**122.68**)	294.15(166.29)	316.49(180.52)

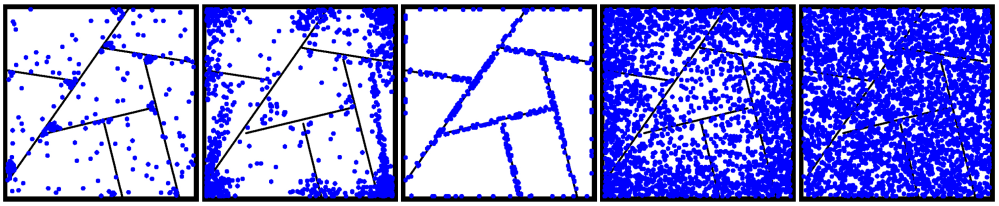
(a) Env. 1



(b) Env. 2

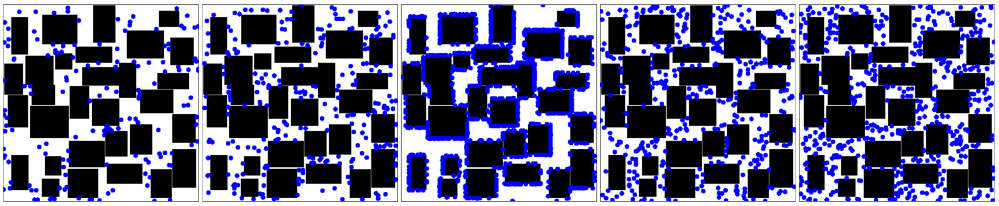


(c) Env. 3



(d) Env. 4

(e) Env. 5



(f) Env. 6

Figure 13.: Milestones taken by MBPI, Bridge Test, OBPRM, Gaussian and Uniform samplers respectively until finding a valid path between the starting and ending points in each environment.

creates a milestone when two samples coincide with the obstacle and the middle of these two samples coincides with the free space. However, there is a very thin zigzag corridor, so the probability of the middle of the two samples falling on the free space is quite low and the working performance of the bridge test is adversely affected.

- Environment 3 is a map containing rooms of different sizes. As shown in Figure 12c, MBPI successfully detects narrow passages. These narrow passages offer an alternative route from the initial to the target point. As seen in Table 2, MBPI is outperforming the other methods in path length metric. Because the alternative route that the other methods cannot detect is considerably shorter. The other methods reach the destination through the longer route because they cannot perform sampling focused on these narrow passages as can be seen in Figure 13c. Additionally, although MBPI is worse in the milestone metric, it is more efficient than the other methods in the planning time metric.
- Environment 4 is a map with long thin sticks placed at different angles. Figure 12d shows the regions that MBPI detects as narrow. Taking these regions into account, hybrid sampling has provided a advantage over other methods in all metrics. The noticeable result is that the bridge test method gives much worse results in planning time and number of milestones metrics compared to other methods. To sample in the narrow passage region, the bridge test requires two samples over obstacles. In this map, the bars creating the narrow passage are very thin, hence it is challenging for the bridge test to generate two obstacle samples. However, MBPI identifies critical regions and focuses directly on those regions as can be seen in Figure 13d, which makes it advantageous in all metrics.
- Environment 5 is a map representing a simple maze. In this map, MBPI succeeded in detecting the critical regions that may influence the connectivity problem, as shown in Figure 12a. Thus, narrow passage samples directly improve the performance of the algorithm. As can be seen in Table 2, MBPI gives the best result in all metrics compared to other methods.
- Environment 6 consists of a large number of rectangular obstacles. Figure 12f shows the regions detected as narrow by MBPI. Due to the large number of obstacle components, there are many narrow passages on the map. These narrow passages provide numerous alternative routes from the initial to the target points. As can be seen in Table 2, MBPI outperforms the other methods in planning time and number of milestone metrics.

4.2.2. Random Map Experiments

This section evaluates the performance of the algorithm on randomly generated maps through a Monte Carlo experiment. This experiment is crucial to understanding how the algorithm behaves in various and unpredictable environments.

In the process of generating random maps, a structure resembling the interior layout of office-like buildings is adopted. During this process, each obstacle is placed on the map perpendicular to another obstacle to simulate a multi-room environment. Examples of random maps generated with this strategy are shown in Figure 14.

A total of one thousand maps are generated using the defined strategy. The initial and target points of each map are randomly selected, ensuring a minimum distance between them. On these maps, PRM is executed with each sampler. For each path planning process, the planning time, number of milestones, and path length are measured. The results are combined to produce the graphs shown in Figure 15. In these

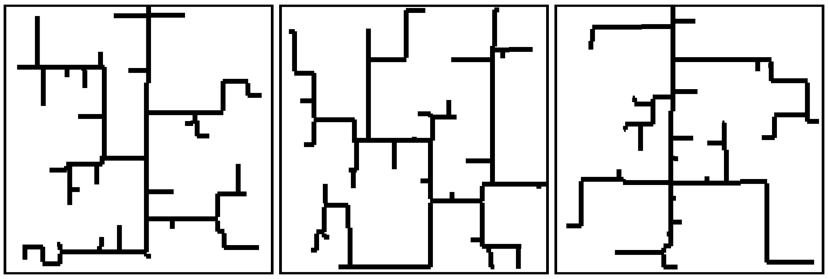
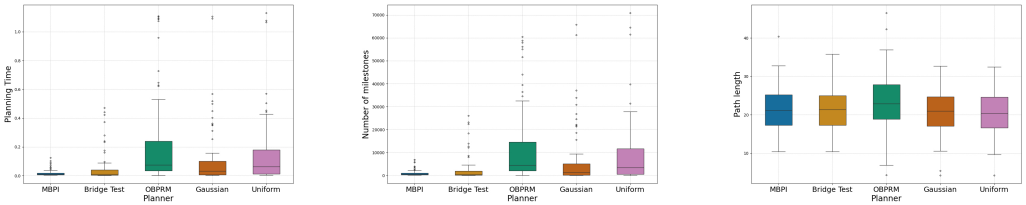


Figure 14.: Examples of generated random maps



(a) Planning time (b) Number of milestones (c) Path length

Figure 15.: Box plots of random map experiments

graphs, the results of each sampler for each metric are shown as box plots. In addition, Table 4 shows the averages and standard deviations of the results obtained from the experiment.

According to the results, the performance of MBPI is superior to other methods in terms of average planning time and number of milestones. In addition, the standard deviations of these metrics are lower than the other methods, indicating that MBPI gives more stable results. Furthermore, the performance in path length is near to the other methods.

4.2.3. Real-world Experiments

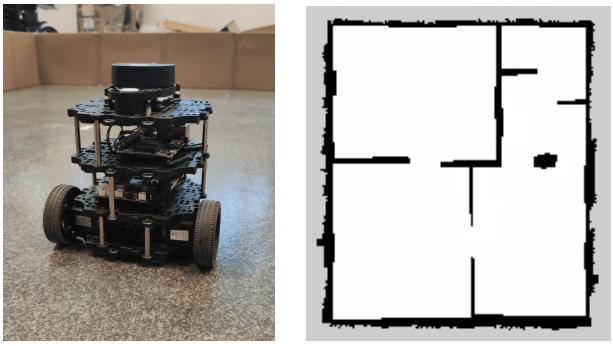
To evaluate the practical applicability and effectiveness of the presented work, the algorithm is tested on a robot in a real-world environment. As in previous experiments, existing methods are also tested in this environment to evaluate objectively the performance of MBPI.

The Turtlebot3 Burger [28] shown in Figure 16a is selected as the robot for real-world experiments. To test the sampler algorithms on the Turtlebot, they must be integrated into the Robot Operating System (ROS) environment. As a result, a new global planner plugin is implemented in the *move_base* package of ROS. The main role of the plugin is to connect OMPL planners and samplers to ROS. During the experiment, the existing navigation packages of the Turtlebot are utilized by replacing the global planner with this plugin. Furthermore, due to the presence of narrow passages in the environment, it is necessary to have a local planner algorithm that can work efficiently in such areas. After testing various planners, the Teb Local Planner has been found to work effectively in such environments. Therefore, the Teb Local Planner is preferred as the local planner.

The experiments are conducted in a static environment that had been previously

Table 4.: Random map experiment performances

Sampler	**Metrics** (Avg.(Std. dev.))		
	t_p[1]	n_m[2]	l_p[3]
MBPI	0.018(0.021)	999.1(1353)	21.52(5.60)
Bridge Test	0.046(0.093)	2265.9(4927)	21.64(5.29)
OBPRM	0.200(0.274)	11399.0(15003)	23.29(6.85)
Gaussian	0.099(0.187)	5643.6(11314)	20.77(5.63)
Uniform	0.143(0.215)	8770.1(13389)	20.75(5.67)



(a) Turtlebot3 Burger

(b) Map of the real-world environment generated by the Turtlebot

Figure 16.: Real-world environment

mapped. The map of the environment scanned by Turtlebot3 Burger is shown in Figure 16b. The experimental process of each sampler is video recorded along with synchronized visualizations from Rviz. The video can be accessed from supplementary files. In Figure 17 scenes from the experiment are shown.

Using each sampler, planning is performed 5 times between the same start and target points. Table 5 shows the results for the planning time, number of milestones in the roadmap, and path length results obtained from the experiment. According to the results shown in the table, the algorithms give similar results. MBPI provides the best results in planning time and path length metrics.

To further analyze the algorithm's performance, the combination of planning and identification times is evaluated in real-world experiments. The combined time is given in Table 4 and calculated using the Equation 5. In this context, t_p denotes the average planning time, n_p represents the number of planning instances, and t_i corresponds to the identification time.

$$\text{Combined Time} = \frac{t_p n_p + t_i}{n_p} \tag{5}$$

This formula highlights that as the number of replanning (n_p) attempts increases,

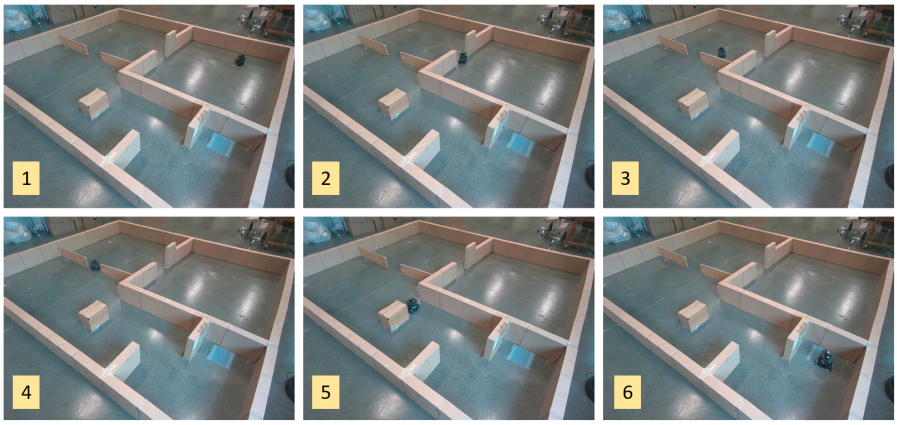


Figure 17.: Scenes from the real-world experiment

Table 5.: Real-world experiment performances. The combined time for MBPI is given in paranthesis.

Sampling Method	**Metrics** (Avg.)		
	t_p [1]	n_m [2]	l_p [3]
MBPI (0.0033)	0.0022	103.8	6.41
Bridge Test	0.0043	102.3	6.54
OBPRM	0.0052	303.9	6.53
Gaussian	0.0025	94.7	6.45
Uniform	0.0025	101.7	6.43

the impact of the identification time (t_i) becomes negligible. In static environments, the identification algorithm is executed only once, and the resulting probability density function is used repeatedly during replanning.

In dynamic environments, if narrow passages remain unaffected by moving obstacles, a single execution of the identification algorithm suffices, allowing the pre-computed narrow passages to guide the planning process. Additionally, when dynamic obstacles alter the position of narrow passages, the hybrid sampling strategy ensures that feasible paths can still be computed. However, this may reduce planning efficiency due to outdated narrow passages. To overcome this issue, the identification algorithm can be executed at a lower frequency, while replanning operates at a higher frequency to adapt to dynamic environment.

The results of the real-world test are parallel to the results obtained in simulation environments. These results indicate that the algorithm is applicable.

5. Conclusion and Future Work

This paper introduces an approach to identify narrow passages, showcasing its effectiveness within the PRM framework for generating roadmaps with high connec-

tivity and coverage. Uniform sampler, standard PRM sampler, provides good coverage by sampling entire map with equal probability. However, connectivity problem of roadmaps appears in environments with narrow passages. To solve this problem, MBPI detects and samples narrow passages. Moreover, a uniform sampler is utilized to ensure high coverage of roadmap. To achieve both, narrow passage sampler and uniform sampler are used hybridly.

MBPI is compared in simulation environment with Bridge Test, Gaussian, Obstacle-based, and Uniform samplers available in OMPL. In addition, algorithms are compared in a real-world environment using Turtlebot3 Burger. The experiments are conducted with metrics of planning time, number of milestones, and path length. The results of the experiments show that MBPI outperforms other methods in terms of planning time and number of milestones, and generally improves path length.

Further studies can be carried out to improve planning strategies in narrow passage environments. These include adapting our method to work with other sampling-based algorithms whose performance is degraded in narrow passage environments. Moreover, broadening the algorithm's applicability to high-dimensional spaces is expected to expand its utility, enabling the handling of more complex scenarios such as manipulator arm planning. Although main advantage of sampling-based methods is observed in high-dimensional environments, this work is anticipated to inspire future studies on adaptation to high-dimensional spaces and development of sampling-based algorithms. In addition, combining the narrow passage detection algorithm with quadtree [29] decomposition could offer advantages by providing a more efficient and better map representation. This approach increases likelihood of seeing narrow passages, which are very small.

Disclosure statement

The authors have no relevant financial or nonfinancial interests to disclose.

Funding

This work was supported by the Turkish Scientific and Technological Research Council (TUBITAK) under project no. 121E537.

Notes on contributors

The study was conceptualized by Y.E.Ş. Methodology design was carried out by Y.E.Ş., E.Y.G., and V.S. Formal analysis and investigation were conducted by Y.E.Ş., E.Y.G., and V.S. The original draft of the manuscript was prepared by Y.E.Ş., E.Y.G., and V.S. V.S. contributed to the review and editing of the manuscript. Additionally, V.S. provided supervision for the project.

References

[1] Hart PE, Nilsson NJ, Raphael B. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics. 1968;4(2):100–107.

[2] Dijkstra EW. A note on two problems in connexion with graphs. In: Edsger wybe dijkstra: His life, work, and legacy; 2022. p. 287–290.
[3] Harabor D, Grastien A. Online graph pruning for pathfinding on grid maps. In: Proceedings of the AAAI Conference on Artificial Intelligence; Vol. 25; 2011. p. 1114–1119.
[4] Kavraki LE, Svestka P, Latombe JC, et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation. 1996;12(4):566–580.
[5] LaValle S. Rapidly-exploring random trees: A new tool for path planning. Research Report 9811. 1998;.
[6] Orthey A, Chamzas C, Kavraki LE. Sampling-based motion planning: A comparative review. Annual Review of Control, Robotics, and Autonomous Systems. 2023;7.
[7] Rantanen MT, Juhola M. How to construct small probabilistic roadmaps with a good coverage? Advanced Robotics. 2014;28(22):1519–1531.
[8] Yershova A, Jaillet L, Siméon T, et al. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In: Proceedings of the 2005 IEEE international conference on robotics and automation; IEEE; 2005. p. 3856–3861.
[9] Sucan IA, Moll M, Kavraki LE. The open motion planning library. IEEE Robotics & Automation Magazine. 2012;19(4):72–82.
[10] Şahiner YE, Gündoğdu EY. Match-based-passage-identifier [https://github.com/yafesenes/Match-Based-Passage-Identifier-MBPI-Algorithm]; 2025.
[11] Amato NM, Bayazit OB, Dale LK, et al. Obprm: An obstacle-based prm for 3d workspaces. In: Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR); 1998. p. 155–168.
[12] Yeh HY, Thomas S, Eppstein D, et al. Uobprm: A uniformly distributed obstacle-based prm. In: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems; IEEE; 2012. p. 2655–2662.
[13] Boor V, Overmars MH, Van Der Stappen AF. The gaussian sampling strategy for probabilistic roadmap planners. In: Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C); Vol. 2; IEEE; 1999. p. 1018–1023.
[14] Saha M, Latombe JC, Chang YC, et al. Finding narrow passages with probabilistic roadmaps: The small-step retraction method. Autonomous robots. 2005;19:301–319.
[15] Wilmarth SA, Amato NM, Stiller PF. Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space. In: Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C); Vol. 2; IEEE; 1999. p. 1024–1031.
[16] Sun Z, Hsu D, Jiang T, et al. Narrow passage sampling for probabilistic roadmap planning. IEEE Transactions on Robotics. 2005;21(6):1105–1115.
[17] Wang W, Zuo L, Xu X. A learning-based multi-rrt approach for robot path planning in narrow passages. Journal of Intelligent & Robotic Systems. 2018;90:81–100.
[18] Shi K, Denny J, Amato NM. Spark prm: Using rrts within prms to efficiently explore narrow passages. In: 2014 IEEE International Conference on Robotics and Automation (ICRA); IEEE; 2014. p. 4659–4666.
[19] Burns B, Brock O. Single-query motion planning with utility-guided random trees. In: Proceedings 2007 IEEE International Conference on Robotics and Automation; IEEE; 2007. p. 3307–3312.
[20] Zhang L, Kim YJ, Manocha D. A hybrid approach for complete motion planning. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems; IEEE; 2007. p. 7–14.
[21] Denny J, Amato NM. Toggle prm: Simultaneous mapping of c-free and c-obstacle-a study in 2d. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems; IEEE; 2011. p. 2632–2639.
[22] Orthey A, Toussaint M. Section patterns: Efficiently solving narrow passage problems in multilevel motion planning. IEEE Transactions on Robotics. 2021;37(6):1891–1905.
[23] Ruan S, Poblete KL, Wu H, et al. Efficient path planning in narrow passages for robots

with ellipsoidal components. IEEE Transactions on Robotics. 2022;39(1):110–127.
[24] Li S, Dantam NT. Sample-driven connectivity learning for motion planning in narrow passages. In: 2023 IEEE International Conference on Robotics and Automation (ICRA); 2023. p. 5681–5687.
[25] Ichter B, Harrison J, Pavone M. Learning sampling distributions for robot motion planning. In: 2018 IEEE International Conference on Robotics and Automation (ICRA); IEEE; 2018. p. 7087–7094.
[26] Haralick RM, Shapiro LG. Computer and robot vision. Vol. 1. Addison-wesley Reading, MA; 1992.
[27] Bresenham JE. Algorithm for computer control of a digital plotter. In: Seminal graphics: pioneering efforts that shaped the field; 1998. p. 1–6.
[28] Amsters R, Slaets P. Turtlebot 3 as a robotics education platform. In: Robotics in Education: Current Research and Innovations 10; Springer; 2020. p. 170–181.
[29] Samet H. The quadtree and related hierarchical data structures. ACM Computing Surveys (CSUR). 1984;16(2):187–260.