2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

LARVO-RRT*: Learning Guided Adaptive Reconfiguration with Vector Field Oriented RRT*

W. K. R. Sachinthana, S. M. Bhagya P. Samarakoon, M. A. Viraj J. Muthugala, and Mohan Rajesh Elara

Abstract— **Path planning is a challenging problem in robotics, for which numerous algorithms have been developed to address it. Sampling-based algorithms, such as the Rapidly Exploring Random Tree (RRT) and its variants, are renowned for their ability to explore the search space efficiently. However, these algorithms consume a considerable amount of computation time and memory to derive the optimal path. Furthermore, when it comes to different geometrical factors, such as narrow passages, fixed-shape robots often fail to navigate because of their structural constraints. This limitation raises the need to use reconfigurable robots, which are capable of changing their shape to access these confined areas. This paper proposes a novel path planning approach for a reconfigurable robot, based on a machine learning model, to address the aforementioned limitations. The proposed method employs a Convolutional Neural Network (CNN) model that predicts sample distribution, a flow field, and robot configurations, which is combined with RRT*, termed Learning Guided Adaptive Reconfiguration with Vector Field Oriented RRT* (LARVO-RRT*). The model has been trained using the optimal sample distributions and flow fields generated with the help of optimal paths from a customized A* algorithm. Experimental results demonstrate that the proposed method surpasses the existing learning and non-learning-based path planning algorithms in terms of time cost, iteration count, and path quality. Furthermore, the algorithm has been able to outperform the existing path planners even without considering the reconfigurations.**

I. Introduction

Path planning is one of the fundamental components of robotics, which involves computing a collision-free trajectory through which the robot can move from start to goal positions in an environment that may include static or dynamic obstacles [1]. Numerous path planning algorithms have been proposed in the literature. Grid-based path planners are a fundamental class of path planning algorithms. They operate by dividing the environment into a grid of uniform cells, where each cell represents a traversable or occupied region of space [1]. The grid-based algorithms such as A* [2] and D* [3] can guarantee an optimal path if one exists. However, these algorithms consume a certain amount of time and memory, which exponentially increases with the map size.

The sampling-based path planning algorithms have become increasingly central to robotics due to their ability to handle complex and high-dimensional environments [4]. The Probabilistic RoadMap (PRM) is a sampling-based planner that engages in a process of planning and query by laying a roadmap for creating a path in configuration space [5]. The Rapidly Exploring Random Tree (RRT) is an efficient sampling-based path planner that uses expansion of nodes in a tree structure-like form to build a graph to perform the search for an obstacle-free path [6]. The RRT algorithm demonstrates higher efficiency compared to PRM in the single-query scenarios, as it does not require creating a roadmap from sampled configurations [7]. Different types of RRT variants have been explored. RRT-connect utilizes two trees to perform a bidirectional search where one tree starts from the start and the other from the goal [8]. These algorithms rely solely on the samples drawn randomly from the space and do not consider the path optimality [9]. To address this, numerous RRT variants that consider path optimality have been proposed. RRT* guarantees to provide an asymptotically optimal path given the environment [10]. This algorithm integrates two new procedures, which are near-vertices and rewiring. The algorithm iteratively continues to run until an optimal path is found by rewiring the nodes. RRT*-smart is another variant, which removes the unnecessary nodes in each iteration [11]. Potential functions coupled with RRT* algorithms have been introduced, which guide the algorithm towards the optimal solution [12]. However, these algorithms consider the whole search space to draw the samples, which results in higher computational time and memory to find an optimal path. This is where the guided RRT* planners come in. The A*-RRT* algorithm [13] initially generates a coarser optimal path using the A* algorithm, which is used to bias the samples generated in RRT*. However, the A* algorithm consumes much more time when it comes to complex environments, which makes it lose the merit of using the guided sampling in A*-RRT* [4].

Recent research has adopted machine learning guided techniques with RRT* to boost the convergence. Semantic information has been utilized to learn the sampling distribution [14]. A conditional variational autoencoder has been proposed to learn an explicit sampling distribution [15]. In [16], a combined Generative Adversarial Network (GAN) and RRT* approach has been proposed, which aims for stronger generalization of the path derivation in complex environments. RL-RRT [17] utilizes a reinforcement learning model as the local planner to handle dynamic obstacles. Unlike these approaches, which utilize hand-designed heuristics, Neural-RRT* [4] uses a Convolutional Neural Network (CNN) model that learns the sample probability distribution

This research is supported by the National Robotics Programme under category National Robotics Programme 2.0, LEO 1.0: A New Class of Bed Making Robot, Award No. M25N4N2028, and also supported by A*STAR under its RIE2025 IAF-PP programme, Modular Reconfigurable Mobile Robots (MR)2, Grant No. M24N2a0039.

W. K. R. Sachinthana, S. M. Bhagya P. Samarakoon, M. A. Viraj J. Muthugala, and Mohan Rajesh Elara are with the Engineering Product Development Pillar, Singapore University of Technology and Design, 8 Somapah Rd, Singapore 487372.

of the optimal path for the environment, and use it for bias sampling. Most of these algorithms have been developed for robots with a fixed shape.

The mobility of a robot with a fixed shape configuration is constrained by geometric factors such as narrow passages, which makes the accessibility comparatively difficult [18]. To overcome this issue, reconfigurable robots have been introduced and applied in different applications such as exploration [19], cleaning [20], coverage [21], and space exploration [22]. However, the existing algorithmic approaches designed for fixed-shape robots cannot be directly applied to reconfigurable robots. Numerous path planning algorithms specifically designed for reconfigurable robots can be found in the literature. In [23], the authors have proposed a modified A* algorithm for a reconfigurable robot to navigate through narrow passages. An energy-efficient path planner for a reconfigurable robot in a complex environment using Energy-efficient Batch-Informed Trees* (EBITR*) has been introduced in [24]. An informed sampling-based RRT* approach applied to an inter-reconfigurable robot has been discussed in [25]. Several studies have used metaheuristic algorithms in the path planning of reconfigurable robots. For example, a genetic algorithm based path planning approach has been used for modular reconfigurable robots in [26]. However, most of the existing path planning approaches for reconfigurable robots are just a customization of the existing systems to incorporate the reconfigurability. Therefore, computation time and memory resources taken for the processing are even higher than that in fixed shape robots.

This paper proposes a novel energy-efficient path planning strategy for a reconfigurable robot utilizing an enhanced version of RRT* called Learning Guided Adaptive Reconfiguration with Vector Field Oriented RRT* (LARVO-RRT*). LARVO-RRT* is a CNN-based approach that extends the sampling-based motion planning by jointly predicting a path probability map, a configuration feasibility map, and a dense flow field. While prior learning-based methods focus primarily on path segmentation or sample probability distributions, our model provides both sample distributions and position-wise flow vector guidance, which ensures smoother convergence. Even without considering reconfigurations, the method is novel as it combines learning-based sample generation with flow-field guidance for steering, which boosts the convergence.

II. Path Planning

A. Problem Formulation

RRT* is an extension of the RRT algorithm that incorporates path optimization to achieve asymptotic optimality. Unlike the basic RRT, which quickly finds feasible paths without considering their quality, RRT* continuously refines the path by rewiring the tree as new nodes are added. This allows it to converge toward the optimal path over time. Although RRT* searches the entire configuration space, its iterative improvement process enables it to gradually reduce the cost of the solution. The proposed approach extends RRT* for a reconfigurable robot by incorporating a CNN model for biased sample generation and steering. Beyond finding a feasible path, the method minimizes a path cost composed of two terms: the travel distance and the reconfiguration cost.

B. Reconfigurable Robot Platform

This research has utilized a variant of Smorphi[1], a Tetris-inspired reconfigurable robot (refer to the Fig. 1). The platform has four modules that are connected using three hinges. The hinge joints allow the modules to rotate, through which the robot can change its shape configurations. All the connected modules work as a single unit, where all the motion commands are given by a single controller. Reconfigurations happen within the robot itself. The robot can reconfigure itself into different configurations. The chassis of the robot is equipped with 16 mecanum wheels, and each module owns four. This helps the robot perform both holonomic and non-holonomic motions depending on the task. The Raspberry Pi controller handles the control of the robot.

C. CNN Model

A CNN is an extended version of Artificial Neural Networks (ANN) used for extracting features from grid-like matrix datasets, such as images or videos. Here, a CNN model is composed to produce the promising samples for RRT planning, while also predicting the flow directions that guide the search, considering the path optimality. Fig. 2 depicts the proposed CNN model architecture.

Each sample of the input dataset for the model follows the format $I \in \mathbb{R}^{W \times H \times 3}$. Respective channels include the information about the obstacle occupancy map, start map, and goal map. For example, each pixel in the obstacle occupancy map $O^i \in \mathbb{R}^{W \times H}$ is derived from the grid $G \in \mathbb{Z}^{W \times H}$. The proposed model adopts a U-Net-based encoder-decoder architecture. The encoder extracts the hierarchical features, where the decoder reconstructs the output in order to predict three fields, which are path probability map ($S \in [0,1]^{W \times H}$), Flow field ($V \in \mathbb{R}^{W \times H \times 2}$), the robot configuration mode map ($M \in [0,1]^{W \times H}$). This is further enhanced using Atrous Spatial Pyramid Pooling (ASPP) to capture multi-scale contextual features. The final output of the model can be expressed as in (1).

$$S, V, M = Model.predict(I) \tag{1}$$

The encoder is designed in a way to reduce the spatial dimensions of the input while enhancing the depth of feature maps, which enables the network to learn both local and

[1]www.wefaarobotics.com

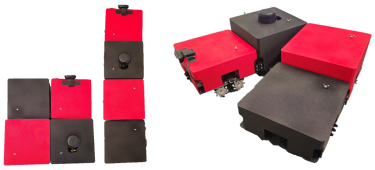


Fig. 1. Reconfigurable robot considered in this work

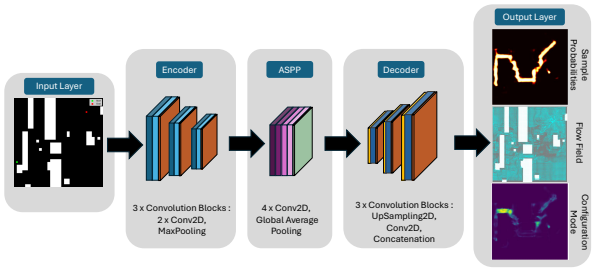


Fig. 2. CNN model architecture

abstract features. The proposed encoder consists of three convolution blocks, each containing two 3×3 convolution ReLU layers followed by a 2×2 max pooling. Convolution layers are formed as expressed in (2).

$$C_i = ReLU(Conv2D_{3\times3}(I_i)) \tag{2}$$

Moving deeper into the network, spatial dimensions are halved at each level following the order $320, 160, 80, 40$ while the number of filters is doubled as $32, 64, 128$. This approach allows for capturing low-level features such as edges and shapes, and high-level semantics such as obstacle and free space regions.

The ASPP module is applied in the bottleneck of the U-Net architecture. It performs dilated convolutions with varying dilation rates to extract the features at multiple scales. The dilated convolution process can be defined as in (3).

$$\mathcal{F}(x, y; d) = \sum_{m=-Q}^{Q} \sum_{n=-R}^{R} I(x - m\cdot d,\ y - n\cdot d)\cdot K(m, n) \tag{3}$$

(Q, R) represents the kernel size. Here, the idea is to absorb the multi-scale features from the encoder output, which helps the decoder later to rebuild a more robust feature map. This is particularly important in situations where the map consists of regions with different characteristics, such as narrow paths and large open areas. In the proposed approach, four dilation rates $(1, 2, 4, 8)$ are used for ASPP. Each branch is using 3×3 convolution-ReLU layers with 256 filters. A global average pooling branch is projected with 1×1 conv and bilinearly resized to the bottleneck size before concatenation. A final 1×1 conv layer fuses branches to 256 channels.

Then the decoder performs the bilinear upsampling of the semantic features extracted from the ASPP until it recovers the original resolution of the input. From the final decoder feature map, three output layers are defined, which are the path probability map, vector flow field, and configuration mode map, which are formed as mentioned in (4). σ refers to sigmoid activation function.

$$\begin{aligned} S = \sigma(Conv_{1\times1}(U_3)), M = \sigma(Conv_{1\times1}(U_3)), \\ V = Linear(Conv_{1\times1}(U_3)) \end{aligned} \tag{4}$$

D. Loss function definition

The proposed CNN model is trained under a multi-task training setting, which includes three output heads: the path probability map (S), the vector flow field (V), and the configuration mode map (M).

1) Path Probability Loss: The path probability head predicts the probability of each pixel belonging to the shortest path. As the pixel path probabilities are highly concentrated around a certain region and a majority of the amount is the background, there's a significant class imbalance, which may lead to improper model training. Therefore, class-weighted binary cross-entropy (BCE) with a soft dice loss is used to ensure accuracy while helping with the imbalanced data. BCE loss ($\mathcal{L}_{\text{BCE}}$) can be formulated as in (5).

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \tag{5}$$

BCE tends to be dominated by the majority class, so the dice loss is used to measure the overlap between the predicted region and the ground truth mask. This directly optimizes for the global shape of the path. Dice loss ($\mathcal{L}_{\text{Dice}}$) can be formulated as in (6).

$$\mathcal{L}_{\text{Dice}} = 1 - \frac{2\sum_{i=1}^{N} y_i\hat{y}_i + \epsilon}{\sum_{i=1}^{N} y_i + \sum_{i=1}^{N} \hat{y}_i + \epsilon} \tag{6}$$

N stands for the number of pixels. y_i is the true label. $\hat{y}_i$ is the predicted probability that a pixel i belongs to the optimal path or the optimal configuration. The probability head loss ($\mathcal{L}_{\text{Prob}}$) can be expressed as in (7).

$$\mathcal{L}_{\text{Prob}} = \mathcal{L}_{\text{BCE}(weighted)} + \mathcal{L}_{\text{Dice}} \tag{7}$$

2) Configuration Mode Loss: The configuration head creates a numerical map of feasible robot configurations. For this, the standard binary cross-entropy loss is applied.

3) Flow field Loss: The flow field head predicts the 2D vector field that represents per-pixel motion directions. In order to enhance the accuracy, a composite loss function is used which includes four terms: mean squared error, cosine similarity, divergence penalty, and total variation smoothing. Let V be the ground-truth flow field, $\hat{V}$ be the predicted flow field, λ_{cos}, λ_{div}, λ_{TV} be the weighting coefficients, then the flow field loss ($\mathcal{L}_{\text{flow}}$) can be represented as in (8).

$$\begin{aligned} \mathcal{L}_{\text{flow}} = \|\hat{V} - V\|_2^2 + \lambda_{\cos}\big(1 - \cos(\hat{V}, V)\big) + \lambda_{\text{div}}\,\|\nabla\cdot\hat{V}\|_2^2 \\ +\lambda_{\text{TV}}\,\text{TV}(\hat{V}) \end{aligned} \tag{8}$$

E. Model Training

In this section, the training process of the proposed CNN model is explained. The main aim is to improve the path quality, considering the energy consumption in terms of path cost. Standard RRT solely relies on selecting the nodes from uniform random sampling, whereas the proposed method gives more weight to the samples around the optimal path. To

achieve this, the model is trained using the paths generated by A^* algorithm, which is known as one of the promising algorithms that ensures the optimal path.

Initially, each map is represented as a grid. Then, given the start and goal states of the robot, the A^* algorithm is applied to find the feasible path. Initially, the cell size is coarser to make the computation faster. The utilized A^* algorithm is customized in order to consider the reconfigurability of the robot. Therefore, it outputs the optimal path incorporating the reconfigurations as well. The customized A^* algorithm is explained in Algorithm 1.

Compared to the traditional A^* algorithm, this algorithm includes more parameters and steps considering the reconfigurability. When creating the new nodes, it produces two new nodes at the same position with the two shapes (compact and extended). Also, when calculating the g-score, it considers the reconfiguration cost. Ultimately, it outputs an optimal path with reconfigurations.

The generated path is then going through a linear interpolation process to derive more waypoints for the training. Then, more sample probabilities are generated, biasing towards the $A*$ path using a Gaussian filter. The application of a Gaussian filter on a 2D array: $S(x, y)$ can be explained as in (9). The output values are then normalized to make sure they are within the range of [0,1]. The final output is used as the probability distribution map.

$$S_f(x,y) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} S(x-u,\ y-v) \cdot G(u,v) \tag{9}$$

These points are treated as a reference path. To enable guidance from any free-space point in the environment, geodesic distances to the nearest path point are computed using Dijkstra's algorithm over a graph of traversable cells. This results in geodesic maps storing the coordinates of the closest path point for each location in the environment. Let a considered point on the path ($\mathcal{P}$) be x', y'. Then the nearest point (x_p, y_p) is selected as explained in (10).

$$(x_p(x,y),\ y_p(x,y)) = \arg\min_{(x',y')\in\mathcal{P}} \ d_{\text{geo}}\big((x,y),(x',y')\big) \tag{10}$$

The geodesic maps are used to construct a vector field that guides the expansion of the search tree. For each traversable location (x, y) in the map, a directional vector (v_f) is computed by blending two components: the vector (v_a) pointing to the nearest point (x_p, y_p) on the reference path ($\mathcal{P}$), and the tangent vector (v_t), representing the local direction of travel along the path. This blend is weighted by the geodesic distance to the path, giving stronger emphasis to the tangent direction when near the path and prioritizing alignment when far. The resulting flow field provides a smooth and context-aware expansion direction, helping the planner follow high-confidence trajectories while still allowing detours when necessary. The flow field is used during the steering phase to bias the direction of node expansion. Let the blending factor be γ. Then the final flow vector is calculated as explained in (11). The resultant flow field is stored as a two-channel map.

$$\mathbf{v}_f(x,y) \ = \frac{\gamma(x,y)\,\mathbf{v}_t(x,y) \ + \ \big(1-\gamma(x,y)\big)\,\mathbf{v}_a(x,y)}{||\gamma(x,y)\,\mathbf{v}_t(x,y) \ + \ \big(1-\gamma(x,y)\big)\,\mathbf{v}_a(x,y)||} \tag{11}$$

The binary configuration map is generated for the modes of the robots by assigning values to cells. Each waypoint is assigned the shape of the robot (extended: 1, compact: 0) which is derived from the A^* path planner. These values are spread outward in a circular pattern using a predefined radius value. Using distance transforms, the closeness of the points to the nearest waypoint is calculated, and then relevant shape modes are inherited accordingly. Remaining unassigned cells are defaulted to compact mode to ensure the completeness of the map.

Algorithm 1 Reconfigurability-Integrated A*

Input: P_{init}, P_{goal}, $gridmap$ **Output:** $robot\_path$

$Node \leftarrow [P,\ shape,\ parent,\ g,\ h,\ f,\ r]$
$open \leftarrow \{Node(P_{\text{init}}, shape_{\text{init}}, \emptyset)\}$, $closed \leftarrow \emptyset$
$moves \leftarrow \{(-1,-1),(-1,0),(-1,1),(0,-1),...\}$
while $open \neq \emptyset$ **do**
  $n \leftarrow \arg\min_{x\in open} f$; $open \leftarrow open \setminus \{n\}$; $closed \leftarrow closed \cup \{n\}$
  if $P(n) = P_{\text{goal}}$ **then**
    return RECONSTRUCTPATH(n)
  end if
  for each $\Delta \in moves$ **do**
    $P' \leftarrow P(n) + \Delta$
    for each $s \in \{shape1, shape2\}$ **do**
      $c \leftarrow Node(P', s, n)$
      $c.r \leftarrow$ RECONFIGCOST($shape(n), s$)
      if COLLISION($c, gridmap$) **then**
        continue
      end if
      $c.g \leftarrow n.g +$ MOVECOST(Δ) $+ c.r$
      $c.h \leftarrow$ HEURISTIC(P', P_{goal})
      $c.f \leftarrow c.g + c.h$
      if $c \in closed$ **then**
        continue
      end if
      $open \leftarrow open \cup \{c\}$
    end for
  end for
end while

F. LARVO-RRT Algorithm*

The proposed method extends the classical RRT* algorithm by integrating learning-based sampling, flow field derivation, and robot shape adaptation. At its core, the planner constructs a tree of feasible robot configurations through iterative sampling, expansion, and rewiring. The

proposed algorithm is guided by a CNN that predicts a spatial probability distribution, per node vector flow field, and a configuration mode map over the input environment. The algorithm balances model-driven sampling and flow-field guidance with uniform exploration, enabling it to retain probabilistic completeness while accelerating convergence in structured environments.

The CNN model predicts the sample probability distribution according to the map that is fed. The generated probability distribution is highly biased around certain regions. While this deeply focused guidance accelerates the path planning in many scenarios, relying only on CNN output can lead to a considerable percentage of the space being ignored, as the probability values of the samples in those regions are very low. This may result in problematic situations, where the algorithm is unable to find feasible paths in complex or novel environments when model predictions are imperfect. In order to preserve the probabilistic completeness of the method, both neural guidance and uniform random sampling are used to produce the samples. Therefore, the CNN model produces αn samples where the random sampler produces $(1-\alpha)n$ samples. In order to have the proper balance between the sampling, $\alpha = 0.5$ is used.

The generated flow field acts as a guidance map that extends beyond the sample probabilities predicted by the CNN. Instead of relying completely on a thin, dilated trajectory path, each pixel is given a directional vector that represents how the robot should move locally towards the goal while respecting the shortest path derived. At any location in the map, the robot can acquire the direction information on how to move towards the nearest feasible path and ultimately to the goal. However, to balance the exploration, the algorithm uses probabilistic randomness in both sampling and directional influence. This allows the robot to explore alternative routes and recover from the uncertain predictions.

In addition to spatial coordinates and flow field, each sampled node includes a discrete shape configuration, representing different physical forms the robot can assume. Two shape types are considered: a compact square and an elongated rectangle, each suited to different navigation tasks. During sampling, the configuration is chosen based on the mode map predicted by the CNN or at random, depending on the sample source. The LARVO-RRT* algorithm is detailed in Algorithm 2.

First, the node tree is initialized. Then the map is fed to the trained model to acquire the probability, flow field, and configuration mode map. Then the samples are drawn from either the probability map and the mode map, or the uniform random sampler, depending on the random probability. The flow field is used to bias the sample steering towards the optimal path. Let the predicted flow vector at the node position be $\mathbf{v}_f$, then the flow direction for steering ($\mathbf{v}_d$) is calculated as in 12. For the node costs, both the distance cost and the reconfiguration cost have been considered.

$$\mathbf{v}_d = \beta \frac{\mathbf{v}_f}{\|\mathbf{v}_f\|} + (1-\beta)\frac{P_{\text{rand}} - P_{\text{nearest}}}{\|P_{\text{rand}} - P_{\text{nearest}}\|} \tag{12}$$

Algorithm 2 LARVO-RRT*

Input: P_{init}, P_{goal}, $Iter_{\max}$, $StepSize$, $Heatmap$, $FlowField$, $ModeMap$
Output: $Best\_Path$

$T \leftarrow \{P_{\text{init}}\}$
$c_{\text{best}} \leftarrow \infty$, $\quad Best\_Path \leftarrow$ None
for $i = 1$ to $Iter_{\max}$ **do**
  $P_{\text{rand}} \leftarrow$ Sample($Heatmap/Random$)
  $P_{\text{nearest}} \leftarrow$ GetNearestNode(T, P_{rand})
  $P_{\text{new}} \leftarrow$ FlowSteer($P_{\text{nearest}}, P_{\text{rand}}, FlowField$)
  P_{new}.shape $\leftarrow$ ShapeSelector(P_{new})
  if IsValid(P_{new}) **then**
    $X_{\text{near}} \leftarrow$ FindNearbyNodes(T, P_{new})
    $P_{\text{parent}} \leftarrow$ SelectBestParent($X_{\text{near}}, P_{\text{new}}$)
    P_{new}.parent $\leftarrow P_{\text{parent}}$
    P_{new}.cost $\leftarrow$ Cost($P_{\text{parent}}, P_{\text{new}}$)
    $T \leftarrow T \cup \{P_{\text{new}}\}$
    for all $P_{\text{near}} \in X_{\text{near}}$ **do**
      if Cost($P_{\text{new}}, P_{\text{near}}$) $< P_{\text{near}}$.cost **then**
        if IsValid(P_{near} via P_{new}) **then**
          P_{near}.parent $\leftarrow P_{\text{new}}$
          P_{near}.cost $\leftarrow$ UpdatedCost
          PropagateCostImprovement(P_{near})
        end if
      end if
    end for
    if IsNearGoal($P_{\text{new}}, P_{\text{goal}}$) **then**
      if P_{new}.cost $< c_{\text{best}}$ **then**
        $c_{\text{best}} \leftarrow P_{\text{new}}$.cost
        $Best\_Path \leftarrow$ BackTrack(P_{new})
      end if
    end if
  end if
end for

When a node reaches within a threshold distance of the goal, the path is extracted by backtracking through the parent links to the root. The full path includes both spatial positions and corresponding shape configurations, allowing for shape transitions along the trajectory. The planner keeps track of the lowest-cost path found during the search and reports performance metrics such as total cost, number of iterations, and computation time.

III. Results

In this section, the performance and generalization of the proposed algorithm are evaluated using several test cases by running multiple runs per map instance. Then the results from the proposed algorithm are compared against existing path planning algorithms, considering different metrics such as path optimality, iterations, and computation time.

A. Experimental Setup

The CNN model is trained on an NVIDIA RTX 4070 GPU with Tensorflow + CUDA support. The learning rate is set to 0.001. 12000 maps are generated with random obstacle

positioning. Each environment may or may not contain narrow passages. The size of each map is 320×320. These maps are fed to train the CNN model. For the simulations, a virtual reconfigurable robot model is created with two different configuration modes (extended and compact). Cell size of the environment is 1×1 pixels.

B. Path planning results

Multiple experiments were conducted to evaluate the performance of the proposed algorithm. The first set of experiments was conducted to evaluate the initial path achievement considering the path quality, time cost, and number of iterations. Table I depicts the results from the experiments. The proposed algorithm is compared against RRT, RRT* and Neural RRT*. Six different test cases were considered, where each does not require reconfiguration of the robot to move from the start to the goal. This experiment was conducted to validate the performance improvement that can be achieved by the proposed algorithm even without using the reconfigurations. Fig. 3 depicts the results of the first three simulations carried out for the reconfiguration-disabled cases.

From the six test cases, LARVO-RRT* shows strong improvements in both path quality and convergence efficiency compared to the other algorithms. It achieves paths that are on average about 19.47% shorter than RRT and 7.23% shorter than RRT*, while also reducing the number of iterations by roughly 45%. Although its runtime is longer than RRT and RRT*, this is mainly because those methods run extremely quickly in absolute terms, making even modest execution times appear comparatively higher. When compared to Neural RRT*, LARVO-RRT* delivers slightly lower costs (2%), requires approximately 23% fewer iterations, and is on average 20% faster, indicating a more efficient balance between speed and solution quality. Overall, these results highlight LARVO-RRT* as the most balanced and reliable approach across the six scenarios, even without considering the reconfigurations.

TABLE I

RESULTS COMPARISON (RECONFIGURATION NOT REQUIRED)

Cases		1	2	3	4	5	6
RRT	cost	455.9	783.6	644.3	409.1	452.6	449.8
	iter	465	1516	615	209	353	276
	time	**0.05**	**0.19**	**0.06**	**0.02**	**0.03**	**0.03**
RRT*	cost	404.4	652	546.6	367.5	395.6	407.6
	iter	461	1519	642	186	338	**258**
	time	0.06	0.19	0.07	0.02	0.04	0.03
Neural RRT*	cost	368.4	626.5	522.9	344.9	**382.9**	367.8
	iter	277	912	471	194	261	331
	time	0.1	0.33	0.17	0.07	0.1	0.12
LARVO RRT*	cost	**360.3**	**611.3**	**521.3**	**339.2**	385.6	**355.4**
	iter	**195**	**643**	**361**	**175**	**232**	266
	time	0.08	0.24	0.13	0.07	0.09	0.1

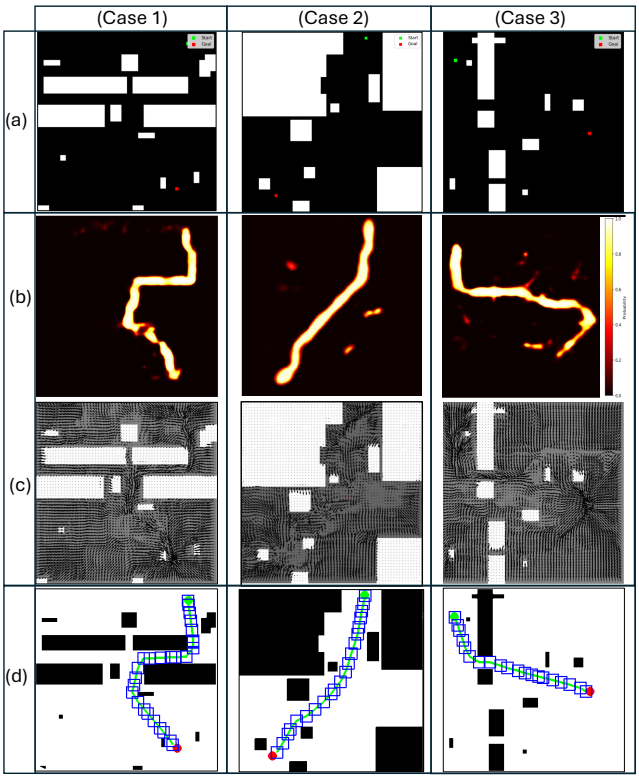


Fig. 3. LARVO-RRT* Path planning results (without reconfigurations): (a). Input map, (b). Predicted sample probabilities, (c). Predicted vector flow field, (d). Resulting path

For the second experiment, ten different test cases were considered, which require reconfigurations of the robot to reach the goal. These test environments consist of narrow passages that require the reconfiguration of the robot to move through. Therefore, all the other algorithms are customized to create nodes with both the extended and compact modes of the robot, thus enabling the reconfigurations. The proposed algorithm already utilizes the reconfiguration derived from both the CNN model and probabilistic randomness. The results from these cases are given in Table II. Fig. 4 depicts the results from the simulations carried out for the first three reconfiguration-enabled cases.

TABLE II

RESULTS COMPARISON (RECONFIGURATION REQUIRED)

	RRT			RRT*			Neural RRT*			LARVO-RRT*		
	cost	iter	time	cost	iter	time	cost	iter	time	cost	iter	time
1	1273.51	1980	0.17	786.43	1600	**0.14**	657.07	583	0.20	**609.52**	**470**	0.17
2	1305.74	2441	0.50	741.19	1123	**0.16**	**604.27**	644	0.25	606.86	**472**	0.17
3	856.33	1441	0.21	460.16	440	**0.05**	365.87	224	0.10	**357.75**	**193**	0.08
4	1077.61	415	**0.04**	582.47	441	0.05	433.54	277	0.11	**431.67**	**206**	0.08
5	1106.87	642	**0.06**	575.93	820	0.09	475.40	350	0.13	**457.83**	**267**	0.10
6	1426.90	708	**0.05**	760.61	586	0.06	591.45	508	0.19	**591.23**	**345**	0.13
7	1031.19	3066	0.57	588.81	3030	0.47	468.79	627	0.22	**452.90**	**490**	**0.18**
8	886.52	302	**0.03**	493.55	326	0.04	302.43	325	0.13	**302.02**	**289**	0.11
9	1011.29	637	**0.06**	616.80	662	0.08	456.98	400	0.15	**441.01**	**367**	0.14
10	1618.37	1823	**0.18**	833.61	1680	0.20	587.24	797	0.30	**567.33**	**603**	0.24

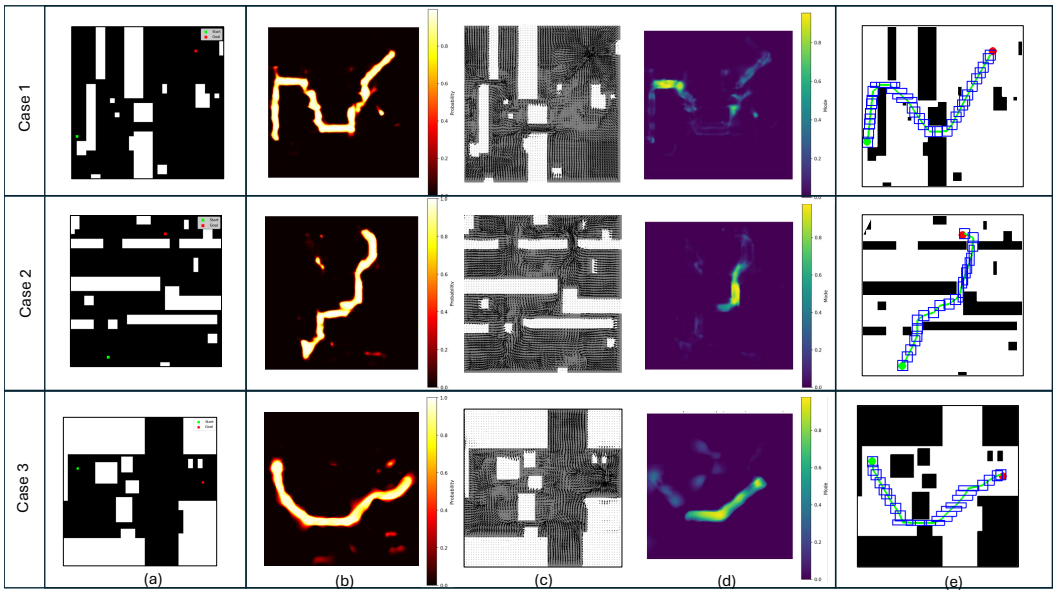


Fig. 4. LARVO-RRT* Path planning results (reconfigurations enabled): (a). Input map, (b). Predicted sample probabilities, (c). Predicted vector flow field, (d). Predicted robot configuration map, (e). Resulting path

Across these test cases, LARVO-RRT* consistently surpasses the other algorithms in terms of the initial path quality and efficiency. Overall it has been able to produce the paths with 58.44%, 25.18%, and 2.52% path cost reduction against RRT, RRT*, and Neural RRT*, respectively. Compared to RRT and RRT*, a significant improvement in path quality has been achieved with the proposed algorithm. Compared to Neural RRT*, the proposed approach has still been able to find lower-cost paths, which demonstrates the flow field influence. Considering the computation time and iterations, the proposed system has been able to achieve a 21.82% reduction in iterations and 21.35% time reduction compared to the Neural RRT* algorithm.

The third experiment is conducted to evaluate the final optimal path achievement using the algorithms. Test cases are considered for both reconfiguration and no reconfiguration scenarios. Fig. 5 depicts the results, which show the time, iterations taken to reach the final optimal solution with the proposed and other existing algorithms.

When it comes to the cases where the reconfiguration is needed (refer to Fig. 5 (c),(d)), LARVO-RRT* once again demonstrates clear advantages in both speed and convergence. Compared to RRT*, it achieves an impressive 84% reduction in runtime and requires about 70% fewer iterations, showing much higher efficiency in reaching solutions. When compared to Neural RRT*, LARVO-RRT* also provides noticeable improvements, running about 28% faster and converging with roughly 16% fewer iterations on average. These results underline LARVO-RRT*'s ability to not only outperform traditional RRT* by a large margin but also refine the performance of Neural RRT, making it consistently the most efficient option across these trials. Real-world experiments were conducted for both reconfiguration-required and non-reconfiguration-required cases. Fig. 6(a) depicts a scenario with reconfiguration disabled, whereas 6(c) depicts a scenario with reconfiguration enabled.

Considering the scenarios with no reconfigurations (refer to Fig. 5(a),(b)), LARVO-RRT* demonstrates a significant performance advantage, especially when compared to traditional RRT*. On average, it achieves a 78% reduction in execution time and requires about 70% fewer iterations, highlighting its efficiency in both speed and convergence. When compared to Neural RRT*, LARVO-RRT* still shows clear gains, running about 14% faster and converging with

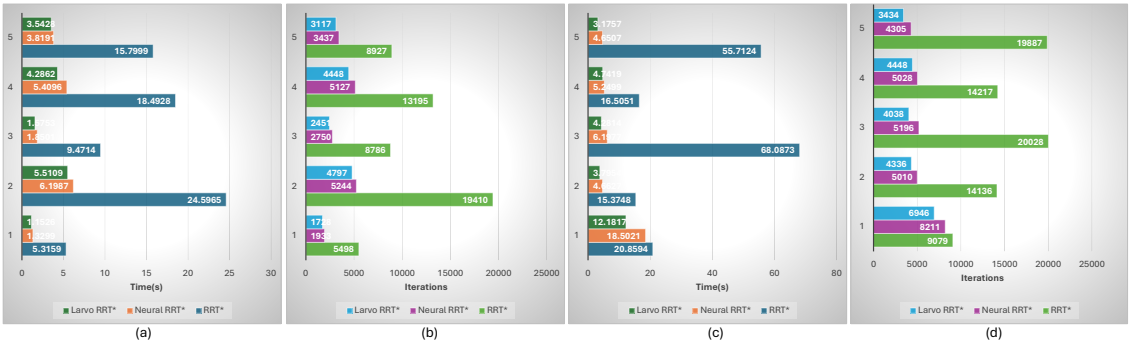
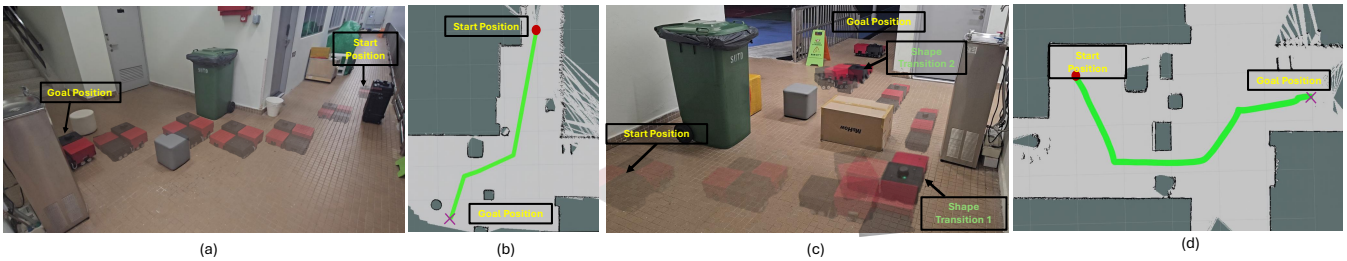


Fig. 5. Convergence comparison: (a),(c). Computation time (without, with reconfigurations), (b),(d). Number of iterations (without, with reconfigurations)



(a) (b) (c) (d)

Fig. 6. Real-world experiments: (a). Robot movement in the environment (without reconfigurations), (b). RViz visualization, (c). Robot movement in the environment (with reconfigurations), (d). RViz visualization

roughly 11% fewer iterations, while maintaining similar levels of path quality. These results emphasize that LARVO-RRT* not only dramatically improves scalability over RRT* but also refines the efficiency of Neural RRT*, making it the most practical and robust approach in this set of experiments.

IV. Conclusions

This paper proposed a novel path planning strategy for reconfigurable robots, improving the performance in cluttered environments. It engages RRT* coupled with CNN-based sampling and flow derivation to bias the sampling towards the more promising regions and steer the nodes towards the optimal path faster. Furthermore, the model proposes the configuration of the robot according to the space constraints identified. The algorithm has been compared against standard sampling-based path planners and neural-path planners, where the proposed system has been able to outperform them, considering different metrics such as path cost, number of iterations, and computation time.

References

[1] M. N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A comparative review on mobile robot path planning: Classical or meta-heuristic methods?" *Annual Reviews in Control*, vol. 50, pp. 233–252, 2020.

[2] C. Ju, Q. Luo, and X. Yan, "Path planning using an improved a-star algorithm," in *2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan)*, 2020, pp. 23–26.

[3] W. Zheng and H. Yu, "Research on the safety of agv path planning based on d* algorithm," in *2024 6th International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, 2024, pp. 216–220.

[4] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural rrt*: Learning-based optimal path planning," *IEEE Transactions on Automation Science and Engineering*, vol. 17, no. 4, pp. 1748–1758, 2020.

[5] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[6] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 1, 1999, pp. 473–479 vol.1.

[7] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[8] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[9] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 7681–7687.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[11] F. Islam, J. Nasir, U. Malik, Y. Ayaz, and O. Hasan, "Rrt*-smart: Rapid convergence implementation of rrt* towards optimal solution," in *2012 IEEE International Conference on Mechatronics and Automation*, 2012, pp. 1651–1656.

[12] A. H. Qureshi, K. F. Iqbal, S. M. Qamar, F. Islam, Y. Ayaz, and N. Muhammad, "Potential guided directional-rrt* for accelerated motion planning in cluttered environments," in *2013 IEEE International Conference on Mechatronics and Automation*, 2013, pp. 519–524.

[13] M. Brunner, B. Brüggemann, and D. Schulz, "Hierarchical rough terrain motion planning using an optimal sampling-based method," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 5539–5544.

[14] I. Baldwin and P. Newman, "Non-parametric learning for natural plan generation," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 4311–4317.

[15] A. H. Qureshi and M. C. Yip, "Deeply informed neural sampling for robot motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6582–6588.

[16] Y. Wang, J. Yu, Y. Kong, L. Sun, C. Liu, J. Wang, and W. Chi, "Socially adaptive path planning based on generative adversarial network," *IEEE Transactions on Intelligent Vehicles*, pp. 1–13, 2024.

[17] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4298–4305, 2019.

[18] W. K. R. Sachinthana, S. M. B. P. Samarakoon, M. A. V. J. Muthugala, and M. R. Elara, "Dynamic reconfiguration integrated nested a*: A path planner for reconfigurable robot to improve performance in confined spaces," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 5473–5480.

[19] C. Liu and M. Yim, "Reconfiguration motion planning for variable topology truss," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1941–1948.

[20] S. M. B. P. Samarakoon, M. A. V. J. Muthugala, A. Vu Le, and M. R. Elara, "htetro-infi: A reconfigurable floor cleaning robot with infinite morphologies," *IEEE Access*, vol. 8, pp. 69 816–69 828, 2020.

[21] W. K. R. Sachinthana, I. D. Wijegunawardana, S. M. B. P. Samarakoon, M. A. V. J. Muthugala, and M. Rajesh Elara, "Energy-efficient coverage path planning for a reconfigurable robot," *IEEE Access*, vol. 13, pp. 128 673–128 684, 2025.

[22] W. Reid, R. Fitch, A. H. Göktoğan, and S. Sukkarieh, "Sampling-based hierarchical motion planning for a reconfigurable wheel-on-leg planetary analogue exploration rover," *Journal of Field Robotics*, vol. 37, no. 5, pp. 786–811, 2020. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21894

[23] A. V. Le, V. Prabakaran, V. Sivanantham, and R. E. Mohan, "Modified a-star algorithm for efficient coverage path planning in tetris inspired self-reconfigurable robot with integrated laser sensor," *Sensors*, vol. 18, no. 8, 2018.

[24] P. T. Kyaw, A. V. Le, P. Veerajagadheswar, M. R. Elara, T. T. Thu, N. H. K. Nhan, P. Van Duc, and M. B. Vu, "Energy-efficient path planning of reconfigurable robots in complex environments," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2481–2494, 2022.

[25] A. V. Le, C. H. Dinh, V. Sivanantham, P. Veerajagadheswar, D. Q. Huy, B. V. Minh, G. Chen, and R. E. Mohan, "Inter-reconfigurable robot by modified informed sampling-based shortest path planning in cleaning and maintenance," *Robotica*, vol. 43, no. 6, p. 2121–2142, 2025.

[26] M. Mayer, Z. Li, and M. Althoff, "Efficient path planning for modular reconfigurable robots," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2024, pp. 3123–3129.