2020 IEEE International Conference on Robotics and Automation (ICRA)
31 May - 31 August, 2020. Paris, France

Adaptive Directional Path Planner for Real-Time, Energy-Efficient, Robust Navigation of Mobile Robots

Mallikarjuna Rao Nimmagadda[1], Shreela Dattawadkar[1], Sriram Muthukumar[1], and Vinayak Honkote[1]

***Abstract*— Autonomous navigation through unknown and complex environments is a fundamental capability that is essential in almost all robotic applications. Optimal robot path planning is critical to enable efficient navigation. Path planning is a complex, compute and memory intensive task. Traditional methods employ either graph based search methods or sample based methods to implement path planning, which are sub-optimal and compute/memory-intensive. To this end, an Adaptive Directional Planner (ADP) algorithm is devised to achieve real-time, energy-efficient, memory-optimized, robust local path planning for enabling efficient autonomous navigation of mobile robots. The ADP algorithm ensures that the paths are optimal and kinematically-feasible. Further, the proposed algorithm is tested with different challenging scenarios verifying the functionality and robustness. The ADP algorithm implementation results demonstrate $40-60X$ less number of nodes and $40-50X$ less execution time compared to the standard TP-RRT schemes, without compromising on accuracy. Finally, the algorithm has also been implemented as an accelerator for non-holonomic, multi-shape, small form factor mobile robots to provide a silicon solution with high performance and low memory footprint ($28KB$).**

I. INTRODUCTION

Autonomous mobile robots, working intelligently and collectively to accomplish complex missions beyond the capability of a single robot, are required for a wide range of applications such as, search-and-rescue, precision agriculture, industrial automation, surveillance and mapping. Effective navigation is a critical capability required for enabling robust operation of such mobile robot systems [1]. In order to enable time critical functionalities on autonomous mobile robots, the navigation capabilities need to run in *real time*. Efficient realization of path planning is a key ingredient for effective autonomous navigation. In any mobile robotic application, path planning is a complex, compute and memory intensive operation. The operation of mobile robots in an unknown, cluttered environments makes the realization of a path planner further complex.

Traditionally, the path planning problem is solved by sampling based methods, such as, RRT (Rapidly exploring Random Tree). In these methods, the focus is on finding a feasible path quickly. The optimality and kinematic feasibility of the path are not guaranteed [2–10]. Many variants of sampling methods (RRT*, iRRT*, Bi-RRT, RRT-Connect) have been proposed to address the optimality limitations observed in standard sampling based methods. They usually work on continuous space to find an optimal path and require a lot of iterations to converge and are hence compute and memory intensive [11–15]. For better navigability, kinodynamic variants of sampling based methods (TP-RRT) were introduced [16]. However, these methods do not guarantee optimal paths and require large memory for storing path models in the form of lookup tables. The compute is also high as the states are sampled in continuous space [6, 16–26]. Separately, an alternative approach is proposed using graph search algorithms (A*/D*) which is applied over finite discretization (e.g., on a grid, or a cell decomposition of the state space) that is predetermined. The optimality of these algorithms is limited by the grid resolution. Higher the grid resolution, higher the probability of an optimal path and higher the memory storage. The memory requirements and execution time grow exponentially with dimensionality (finer grids) of the state space [27]. Thus, the existing approaches are complex, require high compute resources and memory requirements making them unsuitable for real-time autonomous navigation of mobile robots.

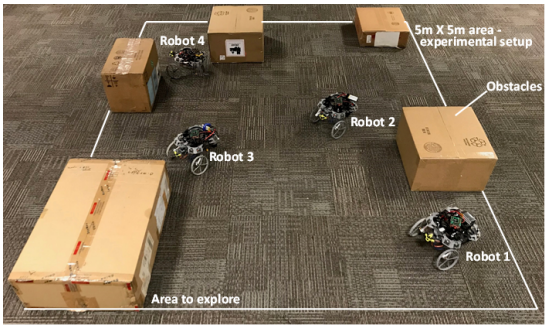


Fig. 1: Mobile robots in action for covering $5mx5m$ area

In swarm systems [28], low-cost, small form-factor and extended operating time are desired parameters. The computation tasks are required to be carried out with extreme energy efficiency. For real-time navigation, it is imperative for planner to run at a high frame rate and ensure the path is kinematically constrained. The state-of-the-art path planning techniques described above require capable cores/servers for efficient execution. They cannot be deployed on to these systems which are constrained by power, compute, memory and cost. Typically, for small form factor robots, path planning task is off-loaded on to a server/cloud due to limited availability of power, compute and memory resources. However, for smooth navigation and efficient mission accomplishment, real time planning with fast response time in the order of few milliseconds is desired. To meet these tight constraints, the planner needs to execute on the edge.

[1]Silicon and Systems Prototyping Lab, Intel Labs, India
mallikarjuna.rao.nimmagadda@intel.com,
shreela.dattawadkar@intel.com,
sriram.k.muthukumar@intel.com,
vinayak.honkote@intel.com

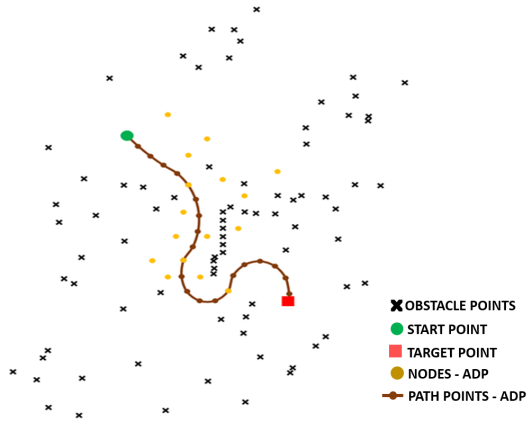


Fig. 2: Path from ADP on a random world. The work space is explored with only 16 nodes to find an optimal kinematically constrained path from start to goal position

To this end, an energy efficient Adaptive Directional Planner (ADP) is presented for enabling efficient autonomous navigation of small form factor robots. A sample setup is shown in Fig. 1. The key contributions of this paper are:

1) A novel ADP algorithm to achieve real-time, energy-efficient, memory-optimized, robust path planning.
2) An algorithm for non uniform discretization of w-space and exploration using informed search techniques from graph based methods.
3) An algorithm to down-size the obstacle map and explore the neighborhood efficiently through connected component labeling scheme.
4) A technique to consider the underlying non-uniform grids as continuous and employing search schemes from sampling based methods for optimal path computation.
5) Combining the sampling-based approach in continuous space and graph-based approach in discrete space to reduce the memory and compute complexity.

The remainder of the paper is organized as below. The preliminaries and problem formulation are presented in Section II. Overall methodology with the proposed algorithms is described in Section III. Experiments and results are discussed in Section IV. Conclusions are presented in Section V.

II. Preliminaries and Problem Formulation

The robot motion model is presented in Section II-A. In the second half of this section (Section II-B), the problem formulation is presented.

A. Robot Motion Model

In this paper, we aim at driving a kinematically constrained, any shape mobile robot in a planar scenario. The robot is required to move towards the target location while avoiding the obstacles and fulfilling the kinematic constraints. For holonomic robots, straight paths are employed as there are no kinematic constraints on them. However, for non-holonomic robots, various kinds of path models including circular trajectories, asymptotically heading trajectories,

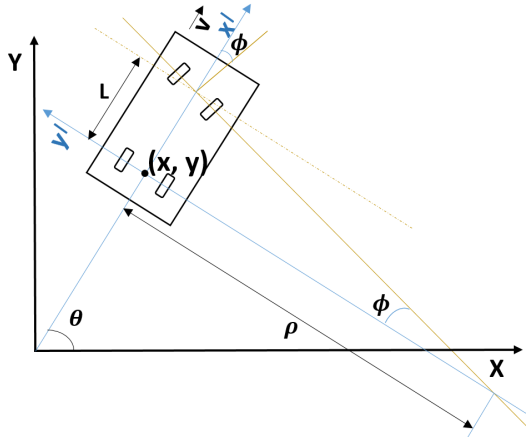


Fig. 3: Ackerman-steering model of robots

spiral segment trajectories are used considering the kinematic constraints of the robot. In Fig. 3, a typical representation for an Ackermann drive robot is presented for a simple car-like robot. This robot can be imagined as a rigid body that moves in a plane that has a state space denoted by $C = \mathbb{R}^2 X S^1$.

A state configuration for the robot is denoted by $p = (x, y, \theta)$ as shown in Fig. 3. The rigid body frame of the robot places the origin at the center of the rear axle, with its x-axis pointing along θ. Let v denote the forward speed of the robot, L denote the distance from the front to rear axles and ϕ denote the steering angle of the front wheels. The non-holonomic nature of the robot is related to a kinematic constraint that does not allow the robot to move laterally. This imposes a condition, $dy/dx = tan\theta$ which can be solved as:

$$\begin{aligned} &(dy/dt)cos\theta - (dx/dt)sin\theta = 0 \\ &\dot{x} = dx/dt = vcos\theta;\ \dot{y} = dy/dt = vsin\theta. \end{aligned} \tag{1}$$

Now, if the steering angle is fixed at ϕ, the robot moves in a circular trajectory that has a radius ρ. Let dw denote the distance travelled by the robot along the circular trajectory in dt. Then,

$$\begin{aligned} &dw = \rho\ d\theta;\ \rho = L/tan\phi; \\ &which\ gives,\ d\theta = \tfrac{tan\phi}{L} dw, \\ &as\ dw/dt = v;\ \dot{\theta} = d\theta/dt = (v/L)tan\phi. \end{aligned} \tag{2}$$

Usually, the steering angle is bounded by some $\phi_{max} < \pi/2$, so that $|v_\phi < \phi_{max}|$. The maximum steering angle implies a minimum turning radius.

To generate kinematically feasible paths for the robot, we use two path models among the set of various Parameterized Trajectory Generators (PTGs) namely, C-PTG and A-PTG [17]. PTGs are path models characterized by the trajectory parameter that determines the curvature of the trajectory. Constraints such as the robot speed limit or other kinematic constraints need to be considered while designing these PTGs. A car-like Ackermann-drive robot will impose a constraint on minimum turning radius, which becomes a maximum angular-to-linear velocity ratio that needs to be maintained during navigation.

B. Problem Formulation

Let $W = \mathbb{R}^n$ be the state space that contains obstacles for the planning problem. Let $W_o \subseteq W$ be the space with

obstacles. A configuration p describes the pose of the robot, and the configuration space C is the set of all possible configurations $p \in C$. For a planar robot, the configuration space is given by $C = \mathbb{R}^2 X S^1$. Let $B(p)$ denote the robot shape translated as a rigid body in configuration p. The set of configurations that avoids collision with obstacles is called the free space C_{free} in which the robot is allowed to move. Free space is given by:

$$C_{free} = p \in C | B(p) \cap W_o = \phi. \tag{3}$$

Let $p_{start} \in C_{free}$ be the initial state and $p_{goal} \in C_{free}$ be the desired final state. The optimal path planning problem is then defined as finding the least-cost continuous path $p(t) \in C_{free}$ for $t \in [0,T]$ from $p(0) = p_{start}$ to $p(T) = p_{goal}$ satisfying the kinematically constrained robot dynamics. Let $f(p(t))$ be the cost of an optimal path from p_{start} to p_{goal} constrained to pass through $p(t) \in C_{free}$. The function $g(p(t))$ captures the cost of the exact optimal path from p_{start} to $p(t)$ and $h(p(t))$ estimates the cost required to extend the path from $p(t)$ to p_{goal}. As $h(.)$ is a heuristic function that is generally unknown, $\hat{h}(.)$ may be used as an admissible estimate. $\hat{h}(.)$ estimates the least cost path from $p(t)$ to p_{goal} by computing the euclidean distance. Then, the function $\hat{f}(p(t))$ represents an admissible estimate of the cost of a path from p_{start} to p_{goal}:

$$\hat{f}(p(t)) := g(p(t)) + \hat{h}(p(t)). \tag{4}$$

An admissible heuristic $\hat{f}(.)$ never overestimates the actual cost to reach p_{goal}:

$$\forall p(t) \in C_{free},\ \hat{f}(p(t)) \leq f(p(t)),\ t \in [0,T]. \tag{5}$$

and thus guarantees an optimal path [27].

III. Methodology

Given the start pose, goal pose and the obstacle map, the functionality of Adaptive Directional Planner (ADP) is to find collision-free, kinematically-feasible paths for arbitrarily-shaped vehicles connecting the start pose to the goal pose. The ADP algorithm is presented in Algorithm 1. The algorithm starts with the tree, $\tau = (V,E)$, where the vertices V represent the robot state configurations and the edges E are the kinematically-feasible trajectories. Initially, the tree only contains the initial position of the robot, p_{start} as a vertex. The tree is grown from p_{start} towards p_{goal} by exploring the state space, W, using edges that are generated by PTGs.

ADP algorithm considers the continuous state space with different levels of space discretization. The first level of discretization is referred to as ADP grids (C_{ADP}). This has low resolution and it helps in maintaining the directionality towards the goal. The finer level of discretization with higher resolution, called collision grids (C_{CG}), are embedded in ADP grids to consider the feasible paths through the narrow spaces between the obstacles. The resolution of these collision grids is determined by the degree of optimality with which the paths have to be found in narrow spaces.

Algorithm 1 ADP

1: **Input:** $p_{start}, p_{goal}, W, W_0$
2: **Output:** $p(t) \in C_{free}\ for\ t \in [0,T]$
3: $from\ p(0) = p_{start}\ to\ p(T) = p_{goal}$
4: $V \leftarrow \{p_{start}\}; E \leftarrow \phi; C \leftarrow \phi; \tau = (V,E)$
5: $PTG \leftarrow \{C-PTG, A-PTG\}$
6: $p_{cp} = p_{start}$
7: $C_{ADP} \leftarrow NON\_UNIFORM\_GRIDS(p_{start}, p_{goal}, W, W_0)$
8: **while** $p_{goal} \notin V$ **do**:
9: $P_N \leftarrow NEXT\_NODE(p_{cp})$
10: $W_o' \leftarrow SQUARE\_CLIP(p_{cp});\ W_o' \subseteq W_o$
11: **for each** $p_n \in P_N$ **do**:
12: $E_{PTG} \leftarrow \phi$
13: $\hat{h}(p_n) = \|p_{goal} - p_n\|$
14: $E_{PTG} \leftarrow PROC\_EDGE(p_{cp}, p_n, E_{PTG} = \{\phi\}, W_o')$
15: $e \leftarrow min\{e[d], \forall e \in E_{PTG}\}$
16: $e\left[\hat{g}(p_n)\right] = e\left[\hat{g}(p_{cp})\right] + e[d]$
17: $e\left[\hat{f}(p_n)\right] = e\left[\hat{g}(p_n)\right] + \hat{h}(p_n)$
18: $C \xleftarrow{+} \{e\}$
19: **end for**
20: **if** $\left(C \equiv \phi\right)$ **then**
21: break;
22: **else**
23: $e_{low} \leftarrow min\left\{e[\hat{f}(p_n)], \forall e \in C\right\}$
24: **end if**
25: $C \xleftarrow{-} \{e_{low}\}$
26: $p_{cp} \leftarrow e_{low}[p_n]$
27: $V \xleftarrow{+} \{p_{cp}\}$
28: $E \xleftarrow{+} \{e_{low}\}$
29: **end while**
30: **if** $\left(p_{goal} \in V\right)$ **then**
31: $BACK\_TRACK\left(\tau, p_{start}, p_{goal}\right)$;
32: **else**
33: $NoPath$
34: **end if**

NON_UNIFORM_GRIDS (Algorithm 1, Line 7): A continuous state space, $W = \mathbb{R}^n$, is composed of a state vector, $w = [w_1, w_2 ..... w_n]$, where n is the number of state variables that define W; w_i is a specific state variable. Then, for each state variable, lower and upper boundaries, w_i^l *and* w_i^u, are specified based on the location of the obstacles as follows:

$$w_i^l \leq w_i \leq w_i^u,\ i = 1,2,3.....n \tag{6}$$

The discretization consists of dividing W into *non-uniform grids*, known as ADP grids. Non-uniform grids have specific areas with different resolutions depending on the location and density of the obstacles in W. Since the underlying world is continuous, the robot is not forced to exactly be in the center of these grids and we identify all the positions of the bot as relevant ones. For $n = 2$, this becomes a 2-D continuous state space as shown in the Fig. 4a. Unlike the traditional grid based algorithms, the whole grid is not marked off as occupied because of presence of an obstacle in the ADP grid. From the current position of the bot (current node),

Algorithm 2 SQUARE_CLIP

Input: p_{cp}, W_o
Output: $W_o' \subseteq W_o, C_{ADP}, C_{CG}$
$C_{ADP} \leftarrow NeighborADPCells(p_{cp})$
for each $c_{ADP} \in C_{ADP}$ **do**:
  if $\left((W_o \cap c_{ADP}) \neq \phi\right)$ **then**
    $C_{CG} \leftarrow Divide(c_{ADP}, R_{CG})$
    for each $q_{obs} \in (W_o \cap c_{ADP})$ **do**:
      $C_{CG}[q_{obs}].add(q_{obs})$
    end for
  end if
end for

a neighboring area is identified for the node expansion. This is described in Algorithm 2.

SQUARE_CLIP (Algorithm 2): From the current position of the bot, the ADP grid is identified and eight neighboring ADP grids denoted by C_{ADP} are clipped. Each non-uniform ADP grid ($[w_i^l,\ w_i^u] \in C_{ADP}$), that has at least one obstacle present is further divided into $(M \times N)$ cells known as collision grids (C_{CG}), with a resolution of R_{CG} as shown in Fig. 4b. These finer collision grids are used to optimally store the transformed obstacle points with respect to p_{cp} of the bot. The transformed obstacle points are used to check for collision when trajectories are generated (Algorithm 3).

NEXT_NODE (Algorithm 1, Line 10): A method inspired by connected component labeling scheme in image processing is used to down-size the obstacle points [29]. The 3×3 ADP grids along with the collision grids and p_{cp} are provided as inputs to this algorithm. This 3×3 area is treated as a binary image with 0 for the presence of obstacle and 1 for free space. The centroid $p_n \in P_N$ of the free space in each neighboring ADP grid $\in C_{ADP}$ around p_{cp} is identified. Now two poses, current position of the bot p_{cp} and the local target

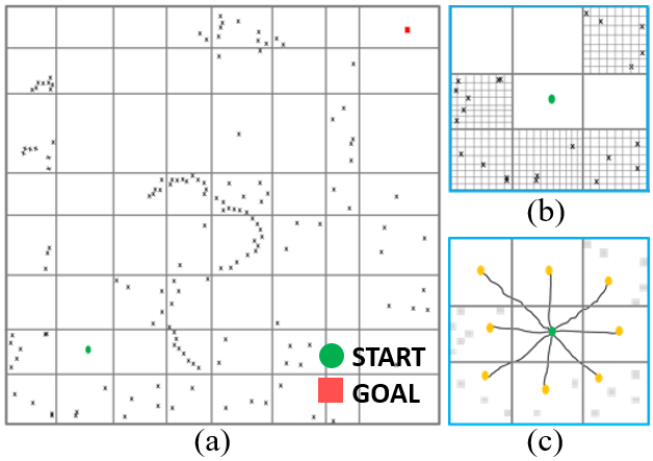


Fig. 4: (a) Non-uniform discretization of state space based on obstacle map known as ADP grids (b) Clipped area around the start position of the robot that includes 3x3 ADP grids. Based on the location of obstacles, ADP grids further goes through finer discretization (c) Next Node computes centroids of free spaces in neighboring 8 ADP grids and assigns them as local targets

Algorithm 3 PROCESS_EDGE

Input: $p_{cp}, p_n, E_{PTG} = \{\phi\}, W_o'$
Output: E_{PTG}
for each $ptg \in PTG$ **do**:
  $p_r \leftarrow p_n - p_{cp}$
  $\alpha \leftarrow ptg^{-1}(p_r)$
  $d \leftarrow 0; t \leftarrow 0; p(t) = (0,0,0)$
  while $p_t \neq p_n$ **do**:
    if $\left(W_o' \cap \{q, \forall q \in POLY(p(t))\} \equiv \phi\right)$ **then**
      $t \leftarrow t + \delta t$
      $d \leftarrow d + \delta d$
      $p(t) \leftarrow ptg.Gen\left(p(t-\delta t), \delta t, \delta d, \alpha\right)$
    else
      $d \leftarrow \infty$
      break;
    end if
  end while
  $e \leftarrow \left(p_{cp}, p_n, \alpha, ptg, d, \hat{f}(p_n), \hat{g}(p_n)\right)$
  $E_{PTG} \xleftarrow{+} \{e\}$
end for

position p_n are treated as two pixels in a binary image. These two pixels are connected if there is an ordered sequence of pixels such that any two adjacent pixels in the sequence are neighbors. This is depicted in Fig. 4c. It is possible to find an edge between p_{cp} and p_n within the free space if they are connected. These edges are further processed and the trajectories are constructed as described in Algorithm 3.

PROC_EDGE_AND_TRAJ (Algorithm 1,Line 11-24)
For each $p_n \in P_N$, the cost $\hat{h}(p_n)$ is calculated and the trajectories are constructed using the set of Parameterized Trajectory Generators (PTGs), $PTG \leftarrow C-PTG,\ A-PTG$ as shown in Fig. 5b and 5c. For each $ptg \in PTG$, the relative position of the neighboring node p_n is computed

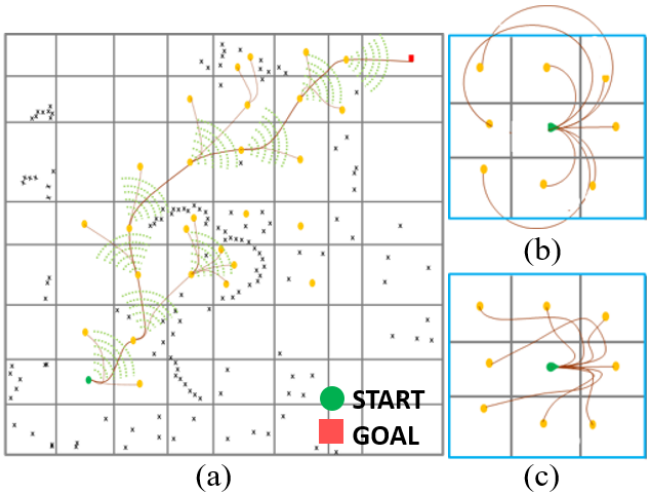
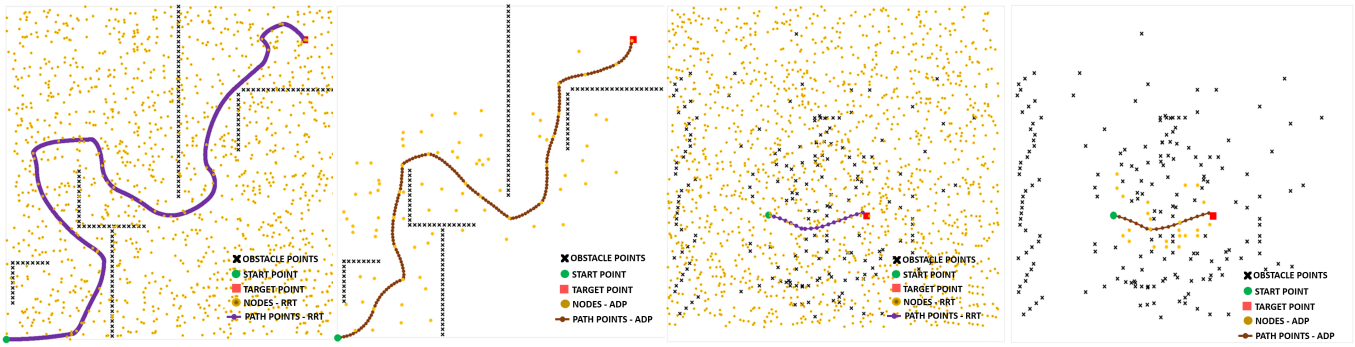


Fig. 5: (a) ADP focuses the search to converge optimally from p_{start} to p_{goal} (b) Paths generated by circular ptg to connect the current position with local goals in the neighboring ADP grids. (c) Paths generated by asymptotic ptg to connect the current position with local goals in the neighboring ADP grids



(a) `Scenario 1: TP-RRT` (b) `Scenario 1: ADP` (c) `Scenario 2: TP-RRT` (d) `Scenario 2: ADP`

Fig. 6: Comparison of node points for RRT and ADP in `Scenario 1` and in `Scenario 2` (in Simulation on Intel Edison)

as, $p_r = p_n - p_{cp}$ as shown in Algorithm 3. The inverse ptg function (ptg^{-1}) returns the trajectory parameter α, which takes it from p_{cp} to that position. The trajectory generation begins with the initial values of the pose, distance and time being 0. The trajectory points are generated in the direction of α, while checking for the collision of the robot transformation $B(p)$ with W_o at every point. If there is a collision, the edge is penalized with a large value and the trajectory generation stops along that α. If the collision free trajectory leads to the desired p_n, then the edge details $\left(p_{cp}, p_n, \alpha, ptg, d, \hat{f}(p_n), \hat{g}(p_n)\right)$ are updated for the respective e_{ptg} and added to the set $\{E_{PTG}\}$. The set is then sorted according to the distance d and the edge with the lowest d is picked as the shortest edge e. Edge e is updated with the costs $\hat{f}(p_n)$ and $\hat{g}(p_n)$ and is added to the set of candidate nodes C. Thus the edges e are evaluated for all the neighboring nodes and added to C. These candidate nodes are used to build the tree.

TREE_EXPANSION (Algorithm 1, Line 25 28): The algorithm incrementally builds a tree, $\tau = (V, E)$, of collision free trajectories rooted at p_{start}. At each iteration, edge with the least cost, $f(\hat{p_n})$, in C is picked as the low cost edge e_{low}. The p_n corresponding to e_{low} is picked as the next current position p_{cp}. The pose p_{cp} and the edge e_{low} are added to the sets $\{V\}$ and $\{E\}$, respectively, as depicted in Fig 5(a). This procedure is repeated until the goal position p_{goal} is reached or $C = \{\phi\}$ (i.e, there are no more low cost edges and the search area is exhausted).

BACK_TRACK (Algorithm 1, Line 31): Once the p_{goal} is reached, the kinematically-feasible path is recovered by backtracking the tree $\tau = (V, E)$, from p_{goal} to p_{start}, as shown in Fig. 5. This path contains a series of paths points $p(t) \in C_{free}$ for $t \in [0, T]$, where $p(0) = p_{start}$ and $p(T) = p_{goal}$.

IV. Experimental Results

Simulations were configured considering the applicability of the ADP algorithm to small form factor Ackermann drive robot model prototyped with custom chassis [28]. Intel Edison platform was used to benchmark the performance of ADP owing to its relevance in on-board, edge-compute systems [30, 31]. ADP was compared with TP-RRT on a wide variety of simulated scenarios. Comparison with other variants of RRT and graph-based methods is omitted here due to the qualitatively different nature of the obtained paths. Further, comparison with two-step planning approaches (which first take the solution from sampling/graph based methods, and then incorporate the kinematical constraints through methods like quadratic programming) is excluded due to the inherent overhead observed in such approaches.

TABLE I: Comparison of TP-RRT and ADP on Intel Edison

	`Scenario 1`		`Scenario 2`	
	TP-RRT	ADP	TP-RRT	ADP
# Nodes explored	3572	87	1498	23
# Nodes on the path	37	22	8	5
Execution time (s)	53	2	22	0.5

We have evaluated the performance of our algorithm using four challenging scenarios as described below:
`Scenario 1`: An easy maze-like scenario with structured obstacles in a large w-space. Parameters $p(x, y, \theta)$ are: $p_{start} = (0, 0, 0)$; $p_{goal} = (9, 9, 0)$.
`Scenario 2`: A highly cluttered scenario with obstacles that requires driving the robot into a relatively narrow space. Parameters $p(x, y, \theta)$ are: $p_{start} = (1.64, -0.23, -0.17)$; $p_{goal} = (3.25, -0.25, 0)$.
`Scenario 3`: A highly cluttered scenario with obstacles that has relatively lesser headroom in the start position. The start and goal positions are both present in a narrow space between the obstacles. Parameters $p(x, y, \theta)$ are: $p_{start} = (-5.83, -6.18, 1.39)$; $p_{goal} = (-8.24, -6.63, 3.14)$.
`Scenario 4`: A maze-like scenario where there are narrow turns requiring smooth continuous maneuvers of the robot. Parameters $p(x, y, \theta)$ are: $p_{start} = (0.5, 0.5, 0)$; $p_{goal} = (8.5, 8.5, 0)$. Note that, (x, y) are in *meters* and θ is in radians.

For the `Scenarios 1` and `2`, TP-RRT and ADP algorithms were executed on Intel Edison platform to find kinematically feasible paths. The maximum map-size considered was $(10m \times 10m)$. The results have been captured in Table I.

When compared to TP-RRT, the ADP algorithm sampled $50 - 60X$ lesser number of nodes during its search for the optimal path. As observed in Fig. 6a and 6c, the RRT expansion is random and the nodes are spread in all possible directions. However, the ADP expansion is guided by a cost function to move towards the goal. As observed

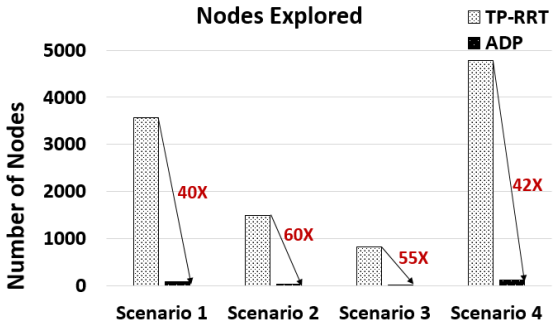
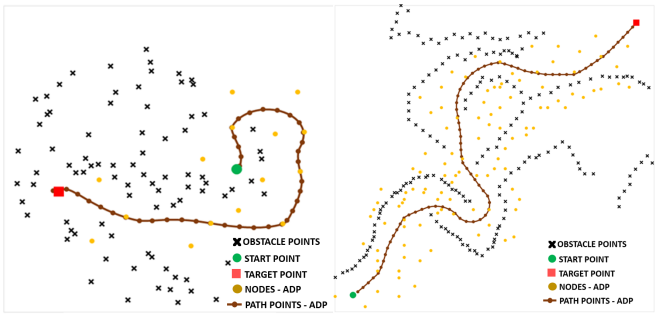


Fig. 7: Comparison of nodes explored by TP-RRT and ADP



(a) `Scenario 3: ADP` (b) `Scenario 4: ADP`

Fig. 8: `Scenario 3` and `Scenario 4` with ADP (silicon)

in Fig. 6b and 6d, the nodes in ADP are grown only in the directions which result in kinematically feasible optimal paths towards the goal position, significantly reducing the number of nodes. Further, the overall runtime with ADP is $40-50X$ less compared to the TP-RRT, demonstrating the optimal compute utilization.

Next, the results comparing the nodes explored by TP-RRT and ADP are compared for all the four scenarios in simulation on Intel Edison. In Fig. 7, the overall improvement in the number of nodes observed is plotted. It is seen that the ADP outperforms TP-RRT with improvement in the orders of $40X-60X$, demonstrating the efficiency of the proposed algorithm.

The ADP algorithm is implemented in silicon on 22nm Intel process node [28]. ADP algorithm is extensively tested in simulations and on silicon with thousands of test cases. For demonstrating the functionality, two challenging silicon test-cases `Scenarios 3 and 4` are presented here.

Results corresponding to `Scenario 3` are presented in Fig. 8a. In `Scenario 3`, the start position of the bot is surrounded by the obstacles in the direction towards the goal position. However, the ADP finds an optimal way out from the start position and constantly changes its orientation towards the target position. Even in the cluttered scenario like this, the efficient sampling of nodes and trajectory generation using ADP are evident.

Results corresponding to `Scenario 4` are presented in Fig. 8b. This is also a maze scenario that needs the robot to have a particular orientation at the narrow turns to smoothly navigate without losing the momentum. TP-RRT was not successful in finding a path in this scenario. Unlike in TP-RRT, the number of nodes that are sampled during the search in ADP are very less.

TABLE II: ADP Silicon Results

Scenarios	Path Length (m)	Time (ms)
`Scenario 3`	6.8	4.6
`Scenario 4`	16.4	13

It is clear from the plots in Fig. 8 that the ADP planner functions efficiently and provides an optimal path from the start position to the target position considering the kinematic constraints of the robot. In Table II, the measurement results on the ADP hardware are shown for these scenarios. At nominal operating conditions ($0.75V$, $100MHz$), the ADP power consumption is $14mW$. Note that, although the proposed planner is specifically focused for small form-factor robots in 2D, it can be scaled to a wide range of robots and extended to the 3D world.

V. Conclusion

In this paper, an Adaptive Directional Planner (ADP) algorithm is presented for addressing complex, compute and memory-intensive requirements of autonomous mobile robot navigation in unknown and challenging environments. The algorithm presented utilizes the combination of continuous space and space discretization approaches resulting in optimal path and quick path convergence. The ADP algorithm is tested with different challenging scenarios verifying the functionality and robustness. The experimental results clearly demonstrate the superiority of the ADP algorithm compared to standard TP-RRT with significant improvements in the number of nodes $(40-60X)$ and execution time $(40-50X)$. Further, the hardware implementation (accelerator) of the ADP results in low-power ($14mW$), low-memory footprint ($28KB$) and high-performance ($>30fps$) demonstrating extreme-energy efficiency.

References

[1] J. J. Craig, *Introduction to robotics: mechanics and control*, vol. 3. Pearson Prentice Hall Upper Saddle River, 2005.
[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.
[3] S. LaValle, "Planning Algorithms: Cambridge University Press, 2006," 2006.
[4] S. LaValle, "Motion planning: robotics automation magazine," *IEEE*, vol. 18, no. 1, pp. 79–89, 2011.
[5] 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.
[6] B. Donald, P. Xavier, J. Canny, J. Canny, J. Reif, and J. Reif, "Kinodynamic motion planning," *Journal of the ACM (JACM)*, vol. 40, no. 5, pp. 1048–1066, 1993.
[7] R. M. Murray, *A mathematical introduction to robotic manipulation*. CRC press, 2017.
[8] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.
[9] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, vol. 1994. Unknown Publisher, 1994.
[10] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[11] A. H. Qureshi, S. Mumtaz, K. F. Iqbal, Y. Ayaz, M. S. Muhammad, O. Hasan, W. Y. Kim, and M. Ra, "Triangular geometry based optimal motion planning using RRT*-motion planner," in *2014 IEEE 13th International Workshop on Advanced Motion Control (AMC)*, pp. 380–385, IEEE, 2014.
[12] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the RRT," in *2011 IEEE International Conference on Robotics and Automation*, pp. 1478–1483, IEEE, 2011.
[13] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2997–3004, IEEE, 2014.
[14] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 2, pp. 1178–1183, IEEE, 2003.
[15] K. Yingying, C. Xiong, and H. Jianda, "Bidirectional Variable Probability RRT Algorithm for Robotic Path Planning," in *Electrical, Information Engineering and Mechatronics 2011*, pp. 1745–1751, Springer, 2012.
[16] J. L. Blanco, M. Bellone, and A. Gimenez-Fernandez, "TP-Space RRT Kinematic Path Planning of Non-Holonomic Any-Shape Vehicles," *International Journal of Advanced Robotic Systems*, vol. 12, no. 5, p. 55, 2015.
[17] J.-L. Blanco, J. González, and J.-A. Fernández-Madrigal, "Extending obstacle avoidance methods through multiple parameter-space transformations," *Autonomous Robots*, vol. 24, no. 1, pp. 29–48, 2008.
[18] J.-L. Blanco, J. Gonzalez, and J.-A. Fernández-Madrigal, "The Trajectory Parameter Space (TP-Space): a new space representation for non-holonomic mobile robot reactive navigation," in *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1195–1200, IEEE, 2006.
[19] J. L. B. Claraco, "Contributions to localization, mapping and navigation in mobile robotics," *Universidad de M'alaga, M'alaga*, 2009.
[20] J. Blanco, J. González, and J. Fernández-Madrigal, "Foundations of parameterized trajectories-based space transformations for obstacle avoidance," *Motion Planning*, p. 23, 2008.
[21] S. Upadhyay and A. Ratnoo, "Continuous-curvature path planning with obstacle avoidance using four parameter logistic curves," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 609–616, 2016.
[22] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Path planning for autonomous vehicles in unknown semi-structured environments," *The International Journal of Robotics Research*, vol. 29, no. 5, pp. 485–501, 2010.
[23] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE conference on decision and control (CDC)*, pp. 7681–7687, IEEE, 2010.
[24] D. J. Webb and J. Van Den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *2013 IEEE International Conference on Robotics and Automation*, pp. 5054–5061, IEEE, 2013.
[25] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the driftless case," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2368–2375, IEEE, 2015.
[26] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *The International Journal of Robotics Research*, vol. 35, no. 5, pp. 528–564, 2016.
[27] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
[28] V. Honkote, D. Kurian, S. Muthukumar, D. Ghosh, S. Yada, K. Jain, B. Jackson, I. Klotchkov, M. R. Nimmagadda, S. Dattawadkar, *et al.*, "A Distributed Autonomous and Collaborative Multi-Robot System Featuring a Low-Power Robot SoC in 22nm CMOS for Integrated Battery-Powered Minibots," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 48–50, IEEE, 2019.
[29] R. C. Gonzalez and R. E. Woods, "Image processing," *Digital image processing*, vol. 2, p. 1, 2007.
[30] "Intel edison, one tiny platform, endless possibility." `http://www.intel.com/content/www/us/en/do-it-yourself/edison.html`. Accessed: 2019-09-14.
[31] B. Beavers, "The story behind the Intel Atom processor success," *IEEE Design & Test of Computers*, vol. 26, no. 2, pp. 8–13, 2009.