Graph-based Path Planning with Dynamic Obstacle Avoidance for Autonomous Parking

Farhad Nawaz[*1,2], Minjun Sung[1,3], Darshan Gadginmath[1,4], Jovin D'sa[1], Sangjae Bae[1],
David Isele[1], Nadia Figueroa[2], Nikolai Matni[2] and Faizan M. Tariq[*1]

Abstract— Safe and efficient path planning in parking scenarios presents a significant challenge due to the presence of cluttered environments filled with static and dynamic obstacles. To address this, we propose a novel and computationally efficient planning strategy that seamlessly integrates the predictions of dynamic obstacles into the planning process, ensuring the generation of collision-free paths. Our approach builds upon the conventional Hybrid A star algorithm by introducing a time-indexed variant that explicitly accounts for the predictions of dynamic obstacles during node exploration in the graph, thus enabling dynamic obstacle avoidance. We integrate the *time-indexed* Hybrid A star algorithm within an online planning framework to compute local paths at each planning step, guided by an adaptively chosen intermediate goal. The proposed method is validated in diverse parking scenarios, including perpendicular, angled, and parallel parking. Through simulations, we showcase our approach's potential in greatly improving the efficiency and safety when compared to the state of the art spline-based planning method for parking situations.

I. INTRODUCTION

(Semi)autonomous parking addresses the growing demand for efficient and safe vehicle maneuvering in constrained environments [1], [2]. Urbanization and increased vehicle traffic have resulted in congested parking lots, necessitating precise, collision-free navigation. This task is further complicated by the presence of dynamic obstacles [3], such as pedestrians and moving vehicles, in addition to static obstacles [4], including parked vehicles and structural elements. A functional planning module for autonomous parking must account for these complexities while ensuring safety, reliability, and computational efficiency.

Consider the scenario in Fig. 1, where the brown ego vehicle is trying to park in the spot between the two black vehicles. Meanwhile, a red car in the adjacent lane is overtaking the ego vehicle, and a pedestrian is walking across the empty parking spot to get on the sidewalk. The more transparent images denoted the predictions of the pedestrian and the red car. Path planning techniques in such scenarios must effectively model and avoid both static and dynamic obstacles, generate precise maneuvers for navigating tight spaces, and ensure computational efficiency to react

[1]Honda Research Institute (HRI), San Jose, CA 95134, USA.
[2]GRASP Lab, University of Pennsylvania, PA 19104, USA.
[3]University of Illinois at Urbana-Champaign, USA.
[4]University of California at Riverside, Riverside, CA, 92521, USA
[*]Corresponding authors: farhadn@seas.upenn.edu & faizan_tariq@honda-ri.com
All work was done when Farhad Nawaz, Minjun Sung and Darshan Gadginmath were employed by HRI.

Fig. 1: A parallel parking scenario with static cars (black), pedestrians, and a moving car (red). The brown car is the ego vehicle. The more transparent images of the red car and the pedestrian denote their respective predictions.

swiftly in dynamic environments. The complexity is primarily attributed to the non-convex geometry of the obstacle-free space, in addition to the non-linear and non-holonomic nature of vehicle dynamics [5], which impose strict motion constraints during the planning process. Furthermore, it has been formally established that identifying a collision-free path in such scenarios is, in general, an NP-hard problem [6], underscoring the computational intractability of this task. Thus, the current research gap is to generate path that simultaneously ensure (i) real-time efficiency, (ii) kinematic feasibility, and (iii) safety in the presence of both static and dynamic obstacles. To address these problems, *we develop a computationally efficient planning strategy that generates safe and reliable paths for autonomous parking maneuvers by explicitly accounting for the motion of dynamic obstacles in our graph-based search algorithm*. We summarize our contributions below.

Contributions: We propose a novel *time-indexed variant* of the conventional Hybrid A* algorithm that explicitly uses the *motion predictions of dynamic obstacles* to generate collision-free paths. Then, we present a strategy for path planning in larger parking lots that utilizes the *time-indexed* Hybrid A* algorithm as a sub-routine to compute local paths at each planning step by choosing an adaptive intermediate goal based on look-ahead point from the current state. We exploit the static map information and incorporate the vanilla A* cost as a heuristic to guide the ego vehicle towards the goal, resulting in improved computational performance and dynamically feasible paths.

We demonstrate through simulations in diverse parking scenarios that our method is computationally efficient compared to the state of the art spline-based approach while generating safer and smoother paths. The simulation videos are available at https://sites.google.com/view/t-ha-star/home.

II. RELATED WORK

Path planning approaches for autonomous vehicles typically fall into three categories: search-based, optimization-based, and learning-based methods. Search-based techniques, such as A⋆[7] and its variants [8], generate coarse obstacle-free paths, which serve as a guide for a low-level trajectory planner. Optimization-based methods incorporate detailed vehicle dynamics, optimizing trajectories for precision, comfort, and safety, but often result in computationally expensive and nonlinear NP-hard problems. Learning-based methods typically employ black-box models that map sensor inputs (e.g., camera or LiDAR data) to control actions or intermediate trajectories, often lacking interpretability.

1) Graph search-based methods: Algorithms such as conventional A⋆ [9] work only for holonomic robots since the motion primitives are linear (horizontal, vertical, diagonal). Rapidly-exploring Random Trees (RRT) [10] and its variants [11], [12] are a set of search techniques that randomly sample points within the grid-world and connect them to the tree used for planning. While such stochastic methods have greater potential to not get stuck at local optimum, they often produce paths with sharp curvature, making them difficult for the low-level trajectory follower to track. Hybrid A⋆ [13] is a search algorithm that improves upon normal A⋆ for non-holonomic robots. In Hybrid A⋆, motion between nodes follow the bicycle dynamics [5] with a given time discretization and a fixed set of motion primitives based on speed, gear, and steering angle. Therefore, the path generated by Hybrid A⋆ is dynamically feasible and makes it easier for the low-level trajectory planner to track. Our work builds upon conventional Hybrid A⋆ and extends it to avoid dynamic obstacles using their motion predictions.

2) Optimization-based methods: Advances in computational power and numerical optimization have popularized optimization-based planning methods such as Model Predictive Control [14]. However, obstacle avoidance often results in non-convex problems [4], [15], [16], sometimes requiring integer variables [17], [18], making real-time implementation challenging. Prior work [19], [20] use dual variables [21] to smoothen constraints, enabling gradient and Hessian-based solvers. Other approaches [22], [23] leverage differential flatness [24] of the kinematic bicycle model [5] to generate spline-based trajectories, but still face computational complexity due to iterative optimization and curvature constraints. Hybrid methods [25], [26] combine search and optimization to handle dynamic obstacles, but are not validated in tightly constrained environments such as parking lots. A time-dependent Hybrid A star algorithm was also proposed in [27], which, unlike our approach, does not exploit closed-form Reeds-Shepp path solutions, relies on a precomputed free-space representation, and employs conservative Voronoi-based collision checking. Overall, the core challenge remains the non-convexity and computational burden of optimization-based planning in dynamic settings, making real-time planning difficult.

3) Learning-based methods: The rich class of deep neural network models are leveraged to learn driving policies using large data from simulations or expert demonstrations. Companies such as Tesla [28], [29] and Waymo [30], [31] have made significant progress in autonomous driving by using learning-based methods like deep reinforcement learning or imitation learning. The ability of deep models to handle complex, unstructured environments has been explored for parking maneuvers that utilize techniques such as supervised policy learning [32] or hybrid methods combining learning with optimization techniques [33]. Despite these advancements, learning-based approaches still face challenges in generalization, safety guarantees, and real-time deployment, particularly in highly constrained scenarios with dynamic obstacles [34].

III. TECHNICAL BACKGROUND

Let $\boldsymbol{x} = (X, Y, \theta)$ be the state of the vehicle, where (X, Y) is the center of the rear axis and θ is the heading angle. To model vehicle dynamics, we employ the kinematic bicycle model which is well suited for vehicle at low speeds [5], expressed as

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \Leftrightarrow \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v}{L}\tan(\delta) \end{bmatrix}. \quad (1)$$

The control input is $\boldsymbol{u} = \begin{bmatrix} v \\ \delta \end{bmatrix}$, where v and δ is the longitudinal velocity and steering angle of the front wheel, respectively. The wheelbase of the vehicle is L. A path $\mathcal{P} := \boldsymbol{x}(t)$, defined as the state trajectory $\boldsymbol{x} : \mathbb{R}_{\geq 0} \to \mathbb{R}^3$, is said to be *dynamically feasible* if there exists control inputs $\boldsymbol{u}(t) \in \mathbb{R}^2$ for all time $t \geq 0$ such that $\boldsymbol{x}(t) = \int_0^t f(\boldsymbol{x}(s), \boldsymbol{u}(s))ds$ for a given initial condition $\boldsymbol{x}(0)$.

The vehicle is modeled as a rectangle, and each obstacle is modeled as a 2D Cartesian point as given in Fig. 2. Let o_x be the distance from the edge of the vehicle to the obstacle along the vehicle's longitudinal direction, and let o_y be the distance along the lateral direction. The obstacle avoidance constraint is

$$\max(o_x, o_y) \geq d, \quad (2)$$

where $d > 0$ is a safety margin that intuitively enlarges the actual size of the vehicle. Constraint (2) precisely determines the proximity of each obstacle to the vehicle's edge, in contrast to methods that approximate obstacle distance using Euclidean distance [35], [36]. Prior work also assume the obstacle to be either a polytope [20], [19] or a circle [35], [36], but we do not assume any specific shape for the obstacle. Each point on the boundary of any arbitrary shaped obstacle can be represented as the red point in Fig. 2, which aligns with the raw point cloud data that we typically receive from sensors to detect obstacles [37], [38].

The boundaries of static obstacles are represented as a sequence of 2D Cartesian points $\mathcal{B} = \{\boldsymbol{b}_i\}_{i=1}^B$, where $\boldsymbol{b}_i \in \mathbb{R}^2$ and B is the number of points used to model the static obstacles. For example, consider the scenario in Fig. 3, where the ego vehicle should move from the blue state to the green state while avoiding the dynamic obstacle in red, and the static vehicles represented as black rectangles. The set $\mathcal{B}$
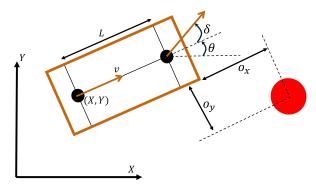
Fig. 2: Geometry of the vehicle and obstacle avoidance. The vehicle is the brown rectangle and obstacle is the red circle.

consists of linearly spaced points along the edges of each static vehicle, in addition to the boundaries of the drivable area. The trajectory of the ith dynamic obstacle is given by the output of a prediction model as $\boldsymbol{y}^i(t) \in \mathbb{R}^2$ for all $i \in \{1, 2, \ldots, O\}$ and time $t \geq 0$ where O is the number of dynamic obstacles. In Fig. 3, the dynamic obstacle moves horizontally from right to left at a constant velocity. Given the 2D Cartesian points for both the static and dynamic obstacles, the obstacle avoidance constraint is given by (2).

IV. PROBLEM STATEMENT

In this section, we formally define two problem statements to address the autonomous parking problem.

Problem 1: Given the initial state of the ego vehicle $\boldsymbol{x}_0$ and the goal state $\boldsymbol{e}$, a map of static obstacles $\mathcal{B}$, the predictions of dynamic obstacles $\{\boldsymbol{y}^i(s)\}_{i=1}^{O}$ for all $s \geq 0$, find a path $\mathcal{P} := \boldsymbol{x}(s)$ such that $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and $\boldsymbol{x}(s) = \boldsymbol{e}$ for all $s \geq S$ where $S \geq 0$ is some finite time, $\mathcal{P}$ is dynamically feasible, and avoids all the static and dynamic obstacles.

In Problem 1, the aim is to generate a *single path* from start to goal that avoids both the static and dynamic obstacles. Fig. 3 illustrates a target scenario for Problem 1.

Problem 2: Given the initial state of the ego vehicle $\boldsymbol{x}_0$ and the global goal state $\boldsymbol{g}$, a map of static obstacles $\mathcal{B}$, the current state of ego vehicle $\boldsymbol{x}_t$ at time t, the predictions of dynamic obstacles $\boldsymbol{y}^i(s)$ for all $i \in \{1, 2, \ldots, O\}$ and $s \geq 0$, find a local path $\mathcal{P}_t := \boldsymbol{x}^t(s)$ at each time t such that $\boldsymbol{x}^t(0) = \boldsymbol{x}_t$, $\boldsymbol{x}^t(s) = \boldsymbol{g}$ for all $t \geq T, s \geq S$ where $S, T \geq 0$ is some finite time, $\mathcal{P}_t$ is dynamically feasible, and avoids all the static and dynamic obstacles.

In Problem 2, the aim is to generate a local path at each time step t from the current state of the vehicle $\boldsymbol{x}_t$ such that the vehicle eventually reaches the goal $\boldsymbol{g}$ without colliding with the static and dynamic obstacles. We refer to Problem 2 as an *online planning* problem where the vehicle should maneuver in a large parking lot as given in Fig. 4 by planning locally at each time step. Problem 1 is viewed as an *one-time planning* problem either for the final parking maneuver as given in Fig. 3 or to a local goal as shown in Fig. 4.

Remark 1: In principle, for the given $\boldsymbol{x}_0$ in Problem 2, any solution that solves Problem 1 with $\boldsymbol{e} = \boldsymbol{g}$ also solves Problem 2 with $T = 0$. However, finding a single path $\mathcal{P}_0$ from $\boldsymbol{x}(0)$ to $\boldsymbol{g}$ in a large parking lot such as Fig. 4

will be computationally expensive. Additionally, the vehicle typically has access only to the predicted trajectories of dynamic obstacles within its local sensing region. Predictions for obstacles located very far away or for very long time horizons are often highly uncertain. Hence, we decouple the *one-time planning* problem, and the *online planning* problem, referring to them as Problem 1 and Problem 2, respectively.

V. TIME-INDEXED HYBRID A STAR

In this section, we propose Algorithm 1 that aims to solve Problem 1. Algorithm 1 is a variant of the conventional Hybrid A* algorithm [13] where we index each node in the search procedure by time t in addition to the state $\boldsymbol{x}$ of the vehicle. The additional time dimension allows us to account for the behavior of dynamic obstacles and subsequently check for collision during the search procedure. The details of Algorithm 1 are given as follows.

Definition 1: A node $\boldsymbol{N}$ is defined as an object with the following attributes.

- $\boldsymbol{N}.\boldsymbol{x}$: state $\boldsymbol{x} = (X, Y, \theta)$ of the vehicle in node $\boldsymbol{N}$
- $\boldsymbol{N}.t$: time $t \geq 0$ at node $\boldsymbol{N}$
- $\boldsymbol{N}.\boldsymbol{P}$: parent node of $\boldsymbol{N}$, where $\boldsymbol{N}.\boldsymbol{P}$ is the immediate predecessor of $\boldsymbol{N}$ in the graph traversal.
- $\boldsymbol{N}.\tau$: state trajectory from the parent node $\boldsymbol{N}.\boldsymbol{P}$ to the current node $\boldsymbol{N}$ for a time horizon H where $\boldsymbol{N}.\tau(0) = \boldsymbol{N}.\boldsymbol{P}.\tau(H)$ and $\boldsymbol{N}.\tau(H) = \boldsymbol{x}$.
- $\boldsymbol{N}.\tau_o$: trajectory of the dynamic obstacles from the parent node $\boldsymbol{N}.\boldsymbol{P}$ to the current node $\boldsymbol{N}$ for a time horizon H where $\boldsymbol{N}.\tau_o(0) = \boldsymbol{N}.\boldsymbol{P}.\tau_o(H)$ and $\boldsymbol{N}.\tau_o(H) = \{\boldsymbol{y}^i(\boldsymbol{N}.t)\}_{i=1}^{O}$.
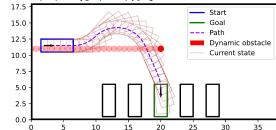

Fig. 3: Target scenario for Problem 1. The black rectangles are static vehicles, and the dynamic obstacle moves from right to left.
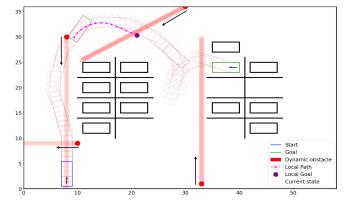

Fig. 4: Target scenario for Problem 2 with four dynamic obstacles where the local path of an intermediate time step is shown. The light brown rectangles denote the trajectory of the ego vehicle, and the black arrows denote the motion of dynamic obstacles.

- $N.c_g$: cost from the start node to node N
- $N.c_h$: heuristic cost from node N to the goal node.

The node N is the fundamental entity in Algorithm 1 that enables us to plan a collision-free path. It serves the same purpose as a typical node in graph-based planning algorithms [13], [39], but we have an additonal time dimension $N.t$ and the predictions of dynamic obstacles $N.\tau_o$. Definition 1 describes that the state of the vehicle at node N is $N.x$ at time $N.t$. The vehicle's trajectory to reach $N.x$ from its parent node $N.P$ is $N.\tau$ and the corresponding trajectory of dynamic obstacles is $N.\tau_o$.

Description of Algorithm 1: We initialize a CLOSEDSET that contains all the explored nodes and a COSTQUEUE that stores the nodes to be explored sorted by the cost function $N.c_g + N.c_h$. In line 8, we "pop" node N from the COSTQUEUE that has the least cost. If we are "close" to the goal as measured using the heuristic cost $N.c_h$ in line 14, we check if there is a direct obstacle free path to the goal using **Reeds-Shepp** paths [40]. If there is no such path, we explore the neighbors of the current node and add it to the COSTQUEUE as given in lines 22-24. We repeat the procedure until we reach the goal, or if the iteration j to explore nodes has reached I_m. Once we reach the goal, we **backtrack** the path by retracing the nodes from the goal to the start by following the parent nodes stored during the search. The

---

Algorithm 1: Time-indexed Hybrid A star

---

1 **Input**: $x_0, e, \mathcal{B}, y^i(s)$ for all $i \in \{1, 2, \ldots, O\}$ and $s \geq 0$, maximum iteration I_m;
2 **Output**: Path $\mathcal{P}$ from x_0 to e;
3 **Initialize:** Start node N, Goal node e_N, ;
4 iteration $j = 0, \mathcal{P} \leftarrow [x_0]$;
5 CLOSEDSET $= \{\}$, COSTQUEUE$[N] = N.c_g + N.c_h$;
6 **while** goal not reached (or) $j < I_m$ **do**
7 $\quad$ $j \leftarrow j + 1$;
8 $\quad$ $N \leftarrow$ Pop COSTQUEUE;
9 $\quad$ Add N to CLOSEDSET;
10 $\quad$ **if** $N.x = e$ **then**
11 $\quad\quad$ $\mathcal{P} \leftarrow$ **backtrack**(CLOSEDSET, e_N);
12 $\quad\quad$ **return** $\mathcal{P}$;
13 $\quad$ **end**
14 $\quad$ **if** $N.h < h_{\text{thresh}}$ **then**
15 $\quad\quad$ $\mathcal{R} \leftarrow$ **Reeds-Shepp**($N.x, e$);
16 $\quad\quad$ **if** $\mathcal{R}$ is obstacle free **then**
17 $\quad\quad\quad$ $\mathcal{P} \leftarrow$**backtrack**(CLOSEDSET, e_N);
18 $\quad\quad\quad$ **return** $\mathcal{P}$;
19 $\quad\quad$ **end**
20 $\quad$ **end**
21 $\quad$ **for** N' in **Neighbors**(N) **do**
22 $\quad\quad$ **if** $N' \notin$ CLOSEDSET (and) $\left(N' \notin \text{COSTQUEUE}\ \text{(or)}\ \text{COSTQUEUE}[N'] > N'.c_g + N'.c_h\right)$ **then**
23 $\quad\quad\quad$ COSTQUEUE$[N'] = N'.c_g + N'.c_h$
24 $\quad\quad$ **end**
25 $\quad$ **end**
26 **end**

---

maximum iteration I_m sets an upper limit on the computation time allocated for searching a path. If $j = I_m$, the path $\mathcal{P}$ defaults to keeping the vehicle stationary as initialized in line 3. This will be further clarified in Section VI.

A. Neighbors

In lines 21-23 of Algorithm 1, we explore neighboring nodes from the current node N to find a path to the goal. The bicycle model (1) is used to generate the neighbors with a discrete set of velocity inputs $v \in [-v_{\max}, v_{\max}]$ and steering inputs $\delta \in [-\delta_{\max}, \delta_{\max}]$. A neighbor N' to the current node N for an input $u = \begin{bmatrix} v \\ \delta \end{bmatrix}$ and time horizon H is obtained as follows.

$$N'.\tau(r) = \int_0^r f(x(s), u)dt \ \forall \ r \in [0, H], x(0) = N.x$$
$$N'.\tau_o(r) = \{y^i(N.t + r)\}_{i=1}^O \ \forall \ r \in [0, H],$$
$$N'.x = N'.\tau(H), \ N'.t = N.t + H, \ N'.P = N. \tag{3}$$

A neighboring node N' is valid if and only if the state trajectory $N'.\tau$ does not collide with the dynamic obstacles' trajectory $N'.\tau_o$ and the static obstacles $\mathcal{B}$. In practice, we discretize the trajectories and the roll out of dynamics in (3) at times $\{r_1, r_2, \ldots, r_K\}$ where $r_1 = 0$ and $r_K = H$. Collision with dynamic obstacles is checked for each $k \in \{1, 2, \ldots, K\}$ by evaluating the pair of points $(N'.\tau(r_k), N'.\tau_o(r_k))$ using (2).

B. Reeds-Shepp

If the current node N is "close" to the goal, we check if there exists a continuous obstacle free path to the goal. The current node is"close" to the goal if the heuristic cost function $N.c_h$ is less than a threshold h_{thresh}. We compute all possible Reeds-Sheep paths [40] from the current state $N.x$ to the goal state e. A Reeds-Shepp path is the optimal path between two states for a vehicle with bicycle dynamics (1) that moves only forward ($v = v_{\max}$) or backward ($v = -v_{\max}$) with extreme steering inputs $\delta \in \{-\delta_{\max}, \delta_{\max}\}$. The obstacle free Reeds-Shepp Paths $\mathcal{R}$ are sorted as per the user-defined cost c_g and the path with the least cost is chosen to move from the current state to the goal state.

C. Cost

The node N in line 8 has the least cost $N.c_g + N.c_h$ amongst all the nodes in COSTQUEUE. The cost function $N.c_g$ penalizes path length, steering angle, reverse motion and change in direction of motion and steering between subsequent nodes. The heuristic cost $N.c_h$ guides the search direction towards the goal using an under-estimate of the actual cost to the goal $e_N.g$. We use the solution of the $A^\star$ algorithm [7] for the heuristic cost which computes the shortest paths from each discrete point in a grid-world environment to the goal without using bicycle dynamics. We pre-compute the $A^\star$ costs using only the static obstacles, since dynamic obstacle avoidance is handled by our time-indexed Hybrid $A^\star$ algorithm.

As described in Remark 1, computing a single path from start to the global goal for a scenario such as Fig. 4 will not be feasible and practical. In the next section, we present an online planning strategy for parking in larger lots.

VI. ONLINE PLANNER

In this section, we propose Algorithm 2 that aims to solve Problem 2, where we find a local path $\mathcal{P}_t$ at each online planning step t that moves the vehicle towards the goal while avoiding dynamic obstacles.

We first initialize a global path $\mathcal{G}$ from $\boldsymbol{x}_0$ to $\boldsymbol{g}$ using conventional Hybrid A* [13] that avoids the static obstacles. Then, at each online planning step t, given the current state $\boldsymbol{x}_t$ and the predictions of dynamic obstacles $\boldsymbol{y}^i(s)$ for all $i \in \{1, 2, \ldots, O\}$ and $s \geq 0$, we compute a local path $\mathcal{P}_t$ that follows the global path $\mathcal{G}$ while avoiding dynamic obstacles. We utilize Algorithm 1 as a sub-routine in Algorithm 2 to plan the local path. The key idea in Algorithm 2 is to choose an appropriate intermediate goal $\boldsymbol{e}$ for Algorithm 1 to return a feasible local path $\mathcal{P}_t$.

Choosing intermediate goal: At the current planning step, we find the closest point on the global path $\mathcal{G}$ to the current state $\boldsymbol{x}_t$. Then, we initialize the intermediate goal using a look-ahead N_m from the closest point on $\mathcal{G}$. In line 10, we try to compute a path from $\boldsymbol{x}_t$ to the intermediate goal $\boldsymbol{e}$ that avoids all the obstacles within a finite time, which is directly proportional to the runtime of Algorithm 1. Instead of the actual runtime, we use the maximum iteration I_m of Algorithm 1 as the stopping criteria. If Algorithm 1 does not find a path within iteration I_m, then we adapt the intermediate goal to be one index closer to $\boldsymbol{x}_t$ along $\mathcal{G}$ and run Algorithm 1 again. We repeatedly adapt the intermediate goal as given in lines $9 - 11$ until we find a path $\mathcal{P}_t$. If the intermediate goal is the closest point on $\mathcal{G}$ to $\boldsymbol{x}_t$, we return the local path to be $[\boldsymbol{x}_t]$ as initialized in line 5 of Algorithm 2, which commands the vehicle to stay stationary. This intuitively explains that if the planner cannot find a feasible local path that follows the global path within a reasonable time, the vehicle stays stationary.

Algorithm 2: Online Planner

---
1 **Given**: $\boldsymbol{x}_0, \boldsymbol{g}, \mathcal{B}$, maximum look-ahead$=N_m$, maximum iteration $=I_m$;
2 **Initialize:** $\mathcal{G} \leftarrow$ global path from $\boldsymbol{x}_0$ to $\boldsymbol{g}$ using vanilla Hybrid A* that avoids $\mathcal{B}$;
3 **Input**: $\boldsymbol{x}_t, \boldsymbol{y}^i(s)$ for all $i \in \{1, 2, \ldots, O\}$;
4 **Output**: Local path $\mathcal{P}_t$;
5 **Initialize:** $\mathcal{P}_t \leftarrow [\boldsymbol{x}_t]$;
6 $i_c \leftarrow \operatorname{argmin}_{i \in \{1,2,\ldots,|\mathcal{G}|\}} \|\boldsymbol{x}_t - \mathcal{G}[i]\|$;
7 $i_g \leftarrow i_c + N_m$;
8 **while** $\mathcal{P}_t = \boldsymbol{x}_t$ (and) $i_g > i_c$ **do**
9      $\boldsymbol{e} \leftarrow \mathcal{G}[i_g]$;
10      $\mathcal{P}_t \leftarrow$ Algorithm1$\left(\boldsymbol{x}_t, \boldsymbol{e}, \mathcal{B}, \{\boldsymbol{y}^i(s)\}_{i=1}^{O}, I_m\right)$;
11      $i_g \leftarrow i_g - 1$;
12 **end**

VII. EXPERIMENTS

We evaluate our proposed Algorithms on a set of common parking scenarios, assuming that the perception system provides real-time information on static vehicles, curb boundaries, and dynamic pedestrians within the system's perception range. The task for the ego vehicle is to navigate from the initial position to a goal state while avoiding collisions. The vehicle dynamics are described by the kinematic bicycle model (1) discretized with a time step of 0.1 s for our simulations. The vehicle parameters are given in Table I. We use Honda Odyssey [41] as a reference to set the dimensions of the vehicle. We use 1 m/s as the velocity limit in our Algorithms to encourage caution, but will increase this limit in our future work. All simulations are performed in Python 3.8 on Ubuntu 20.04 with Intel Xeon E5-2643 v4 CPU.

Our time-indexed Hybrid A* implementation uses a state grid size of 2 m in both X and Y directions, and 20 deg in the heading angle. The steering input is discretized using 5 points in $[-\delta_{\max}, \delta_{\max}]$ and velocity is discretized using 3 points in $[-v_{\max}, v_{\max}]$. The length of each parking space is 6.5 m, and the width is 3.5 m as referred from [42]. The inclination angle with respect to the driving direction is 70 deg for angle parking. Each static vehicle is modeled as a rectangle with length and width given in Table I, and each dynamic obstacle is modeled as a circle of radius 0.5 m. Collision is checked using the geometry of the vehicle in Fig. 2 with a safety threshold of $d = 0.5$ m.

A. One-time Planning

We validate Algorithm 1 with $I_m = 500$ for the following parking scenarios as given in Fig. 5: perpendicular head-in, perpendicular reverse-in, angle head-in and parallel. In all the cases presented in this work, the ego vehicle parks in confined spaces with defined outer boundaries in the presence of other parked vehicles and dynamic obstacles. We compare Algorithm 1 — the time-indexed Hybrid A* (**t-HA***) — with the state of the art iterative spline-based collision avoidance method [22] (ItCA). The work in [22] iteratively refines 5th-order spline trajectories to track a reference path (e.g., Hybrid A* avoiding static obstacles). If collisions with (possibly dynamic) obstacles occur, the tracking cost is relaxed at collision points in each iteration, until the path is collision-free, or a maximum iteration count is reached.

In Fig. 5, we qualitatively compare the paths generated by ItCA [22] with that of Algorithm 1 using the A* heuristic (**t-HA* + A***). The paths generated by t-HA* + A* avoids both static and dynamic obstacles in all scenarios. The ItCA method generates collision-free paths in the absence of dynamic obstacles for all the parking scenarios presented in this work. However, in the presence of multiple dynamic

TABLE I: Vehicle parameters

Parameter	Description	Value
L	Wheelbase length	3 m
V_L	Vehicle length	5 m
V_W	Vehicle width	2 m
$v_{\max}$	Velocity limit	1 m s^{-1}
$\delta_{\max}$	Steering limit	40 deg

obstacles, the ItCA path for reverse-in parking collides with a static obstacle, and the paths for angle and parallel parking collide with obstacles at multiple points. We rigorously test our simulation runs across different initial positions and velocities of the dynamic obstacles. The initial point of each obstacle are regularly spaced within the bounds given in Table II. The bounds in Table II are chosen so that the dynamic obstacles cover a sufficient area of the drivable region and are possibly on a collision course with the ego vehicle to verify if the algorithms can avoid the obstacles. We generate 100 initial points for the one obstacle in perpendicular head-in parking (Fig. 3). We choose 15 points for each obstacle in perpendicular reverse-in, angled head-in, and parallel parking (Fig. 5), resulting in a total of $15^2 = 225$ initial position pairs. The velocity of obstacles in the X and Y directions for each candidate initial point is sampled from the uniform distribution $[-0.7, 0.7]$ m/s. We run 50 tests for each set of initial points and velocities. We also evaluate two variations of t-HA*, one with a Euclidean heuristic cost function (t-HA* + Eucledian) and a grid-based A* cost (**t-HA* + A***).

The average performance and trajectory quality metrics for the experiments, along with one standard deviation, are presented in Table III. The average runtime of t-HA* + A* is 10-100 times faster than ItCA or when using the Eucledian heuristic. Since the A* heuristic exploits the information of static obstacles, the search procedure in Algorithm 1 explores nodes more optimally than when the Euclidean heuristic is used. The heading rate and curvature in ItCA are significantly larger than in t-HA*, rendering the spline paths infeasible for the vehicle to follow given its velocity and steering limits (Table I). The curvature is computed using the formula provided in Section III-E of [23].

In Fig. 6, the paths generated by t-HA* show that the ego vehicle yields to dynamic obstacles before proceeding to the parking spot. This mirrors typical human driving behavior by pausing to assess and adapt, ensuring both safety and smooth navigation. In contrast, ItCA often produces sharp

TABLE II: Parameters of the Parking Scenarios, where Initial and Goal state of the ego vehicle is $[X$ m, Y m, θ deg$]$ and $(X_i$ m, Y_i m$)$ is the Initial Position of Obstacle i.

Scenario	Initial, Goal State $[X, Y, \theta]$	Initial Position of Dynamic Obstacles
Perpendicular head-in (Fig. 3)	[2.0, 11.5, 0], [20.0, 5.0, −90]	$X_1, Y_1 \sim [15, 30], [7, 17]$
Perpendicular reverse-in (Fig. 5)	[2.0, 9.5, 0], [16.0, 1.0, 90]	$X_1, Y_1 \sim [12, 25], [8, 13]$ $X_2, Y_2 \sim [10, 18], [5, 8]$
Angle head-in (Fig. 5)	[2.0, 11.5, 0], [21.0, 5.0, −70]	$X_1, Y_1 \sim [11, 16], [5, 9]$, $X_2, Y_2 \sim [16, 25], [10, 15]$
Parallel (Fig. 5)	[4.0, 3.0, 0], [17.0, 0.0, 0]	$X_1, Y_1 \sim [10, 15], [1, 3]$, $X_2, Y_2 \sim [20, 28], [2, 4]$

turns, leading to infeasible curvature or failure to find a collision-free path within the iteration limit. Even with an increased limit of 100 — compared to 10 used in [22] — ItCA exhibits a high failure rate as given in Table. III, except when dynamic obstacles are positioned far from the initial reference path.

B. Online Planning

We validate Algorithm 2 for perpendicular reverse-in parking in a large surface lot depicting a dense traffic situation that includes multiple parked cars and dynamic obstacles as given in Fig. 7. We use a maximum look-ahead of $N_m = 5$ to choose an intermediate goal from the initial global path $\mathcal{G}$, and maximum iteration $I_m = 100$ as the stopping criteria in Algorithm 2. All other vehicle parameters and parking dimensions are the same as used in Section VII-A. The road width that separates adjacent parking rows is 10 m.

The trajectory of the ego vehicle navigating to a designated parking spot is illustrated in Fig. 7 with 15 dynamic obstacles. Each local path connects the current state of the ego vehicle to an intermediate goal, selected as described in Section VI. The maximum iteration limit I_m ensures that Algorithm 2 operates at a high enough frequency, enabling frequent refinement of the local path to closely track the global path while avoiding obstacles.

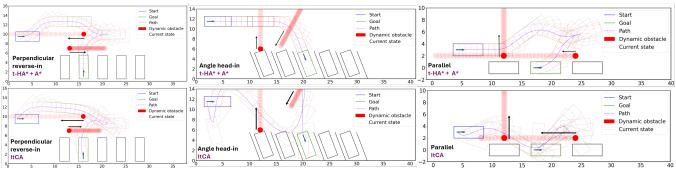We run experiments for three cases in the large surface

Fig. 5: Comparison of paths generated by our time-indexed Hybrid A* method (t-HA* + A*) and the iterative spline-based method (ItCA).
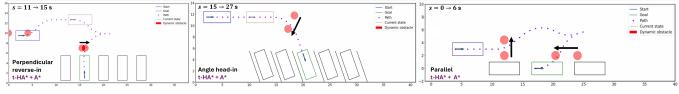

Fig. 6: Illustration of the ego vehicle's stationary behaviors at different time steps s in the paths generated by Algorithm 1 (t-HA* + A*).

TABLE III: Comparison of Average Performance and Trajectory Metrics, with One Standard Deviation, for all the Parking Scenarios.

Scenario	Number of dynamic obstacles	Method	Failure Rate ↓	Runtime ↓ (s)	Path length ↓ (m)	Distance to closest obstacle ↑ (m)	Heading rate ↓ (deg s^{-1})	Curvature ↓ (m^{-1})
Perpendicular head-in (Fig. 3)	1	ItCA	8%	0.823 ± 0.144	44.902 ± 8.277	2.246 ± 0.247	5.711 ± 4.202	0.118 ± 0.098
		t-HA* + Eucledian	0.00%	0.448 ± 0.061	43.223 ± 6.084	2.314 ± 0.261	4.708 ± 4.165	0.074 ± 0.071
		t-HA* + A*	**0.00%**	**0.018 ± 0.004**	**40.578 ± 8.312**	**2.983 ± 0.476**	**3.009 ± 1.461**	**0.051 ± 0.026**
Perpendicular reverse-in (Fig. 5)	2	ItCA	67.6%	1.485 ± 0.84	33.624 ± 9.216	1.538 ± 0.182	13.85 ± 20.355	1.21 ± 9.7
		t-HA* + Eucledian	0.00%	9.716 ± 0.217	40.902 ± 8.936	1.592 ± 0.435	4.867 ± 5.843	0.086 ± 0.101
		t-HA* + A*	**0.00%**	**0.048 ± 0.007**	**28.967 ± 4.569**	**1.75 ± 0.347**	**4.401 ± 2.388**	**0.072 ± 0.036**
Angle head-in (Fig. 5)	2	ItCA	85.7%	4.998 ± 0.428	56.134 ± 11.934	0.616 ± 0.241	25.67 ± 33.327	0.628 ± 2.294
		t-HA* + Eucledian	0.00%	4.749 ± 0.211	34.872 ± 10.944	1.374 ± 0.385	3.826 ± 0.917	0.047 ± 0.041
		t-HA* + A*	**0.00%**	**0.034 ± 0.005**	**27.08 ± 6.813**	**1.374 ± 0.065**	**2.471 ± 0.779**	**0.042 ± 0.013**
Parallel (Fig. 5)	2	ItCA	92%	0.781 ± 0.309	38.924 ± 9.869	0.953 ± 0.176	40.949 ± 41.067	0.852 ± 2.463
		t-HA* + Eucledian	0.00%	6.077 ± 0.432	**30.179 ± 7.557**	**2.045 ± 0.336**	**3.619 ± 5.514**	**0.058 ± 0.088**
		t-HA* + A*	**0.00%**	**0.026 ± 0.001**	32.713 ± 2.481	1.686 ± 0.169	6.421 ± 0.819	0.106 ± 0.014
Perpendicular reverse-in surface lot (Fig. 7)	4	**t-HA* + A***	15%	0.012 ± 0.005	92.368 ± 6.829	2.058 ± 0.191	3.694 ± 0.58	0.062 ± 0.028
	10	**t-HA* + A***	30%	0.012 ± 0.002	95.353 ± 19.461	1.912 ± 0.159	3.278 ± 4.32	0.051 ± 0.07
	15	**t-HA* + A***	35%	0.011 ± 0.016	76.141 ± 11.19	1.546 ± 0.213	2.804 ± 4.314	0.043 ± 0.07

lot environment, where each case corresponds to different number of dynamic obstacles: 4, 10 and 15. We conduct 20 experiments for each of the three cases, and 10 test runs for each experiment. The initial positions of the dynamic obstacles in each experiment are randomly sampled from the uniform distributions specified in Table IV. Different bounds are used for different obstacle sets so that the ego vehicle encounters dense traffic situations at varying time instances. The velocity of obstacles in the X and Y directions for each candidate initial point is sampled from the uniform distribution $[-0.7, 0.7]$ m/s. The initial state is $(8 \text{ m}, 1 \text{ m}, 90 \text{ deg})$ and the goal state is $(40 \text{ m}, 24 \text{ m}, 180 \text{ deg})$. The performance metrics for the surface lot scenarios are also summarized in

TABLE IV: Initial Position of Dynamics Obstacles for Surface Lot Environment (Fig. 7).

Total number of dynamic obstacles	Split of dynamic obstacles	Initial Position X m, Y m $\sim$
4	2	[0, 15], [7 , 40]
	2	[25, 40], [0, 40]
10	6	[0, 20], [7, 60]
	4	[20, 40], [0, 60]
15	5	[0, 20], [7, 20]
	4	[0, 20], [20, 60]
	6	[20, 40], [0, 60]

Table III. The relatively higher failure rate of t-HA* + A* in the surface lot environment is due to the absence of a feasible path caused by dense traffic conditions. In contrast, ItCA and t-HA* + Euclidean always generate unsafe paths, even when the iteration limit is increased to 1000, corresponding to an average maximum timeout of 10 seconds. The runtime for the surface lot scenario in Fig.7 corresponds to Algorithm 2, which computes a local path at each time step, unlike other scenarios described in Section VII-A that use Algorithm 1. The lower run-time for Fig. 7 is due to planning toward a nearer intermediate goal compared to the more distant goals in Figs.3 and 5 The trajectory metrics, including path length, distance to the closest obstacle, heading rate, and curvature, for the surface lot scenario, are calculated over the entire trajectory, from the start to the final goal state, while deviating slightly from the global path to avoid obstacles.

VIII. CONCLUSION AND FUTURE WORK

We proposed a *time-indexed* Hybrid A* algorithm that explicitly incorporates dynamic obstacle predictions to generate safe, reliable, and smooth paths across diverse parking scenarios. This was further extended to an online planning
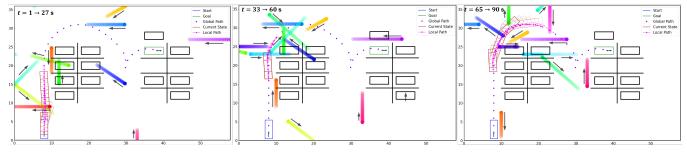

Fig. 7: Perpendicular reverse-in parking in a large surface lot with 15 dynamic obstacles shown as different coloured circles.

strategy for large surface lots via an adaptive goal-selection mechanism. Simulations across multiple parking settings demonstrate improved computational efficiency, safety, and feasibility over the state-of-the-art spline-based planner.

Future work will address the assumption of perfect trajectory predictions by the current method for dynamic obstacles, which overlooks uncertainties and latencies in the planning pipeline. We also aim to relax the assumption of a known goal state by extending our approach to actively explore the parking lot and identify suitable parking spots, all while accounting for uncertainty in the behavior of other agents.

REFERENCES

[1] D. C. Conner, H. Kress-Gazit, H. Choset, A. A. Rizzi, and G. J. Pappas, "Valet parking without a valet," in *2007 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2007, pp. 572–577.

[2] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[3] N. M. Nakrani and M. M. Joshi, "A human-like decision intelligence for obstacle avoidance in autonomous vehicle parking," *Applied Intelligence*, vol. 52, no. 4, pp. 3728–3747, 2022.

[4] B. Li, T. Acarman, Y. Zhang, Y. Ouyang, C. Yaman, Q. Kong, X. Zhong, and X. Peng, "Optimization-based trajectory planning for autonomous parking with irregularly placed obstacles: A lightweight iterative framework," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 970–11 981, 2021.

[5] R. Rajamani, *Vehicle dynamics and control*. Springer Science & Business Media, 2011.

[6] J. Canny, *The complexity of robot motion planning*. MIT press, 1988.

[7] 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.

[8] R. Fareh, M. Baziyad, M. H. Rahman, T. Rabie, and M. Bettayeb, "Investigating reduced path planning strategy for differential wheeled mobile robot," *Robotica*, vol. 38, no. 2, pp. 235–255, 2020.

[9] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[10] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[11] 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*. IEEE, 2014, pp. 2997–3004.

[12] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2537–2542.

[13] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," *Ann Arbor*, vol. 1001, no. 48105, pp. 18–80, 2008.

[14] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[15] C. Diehl, A. Makarow, C. Rösmann, and T. Bertram, "Time-optimal nonlinear model predictive control for radar-based automated parking," *IFAC-PapersOnLine*, vol. 55, no. 14, pp. 34–39, 2022.

[16] F. M. Tariq, D. Isele, J. S. Baras, and S. Bae, "Rcms: Risk-aware crash mitigation system for autonomous vehicles," in *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2023, pp. 3950–3957.

[17] F. M. Tariq, N. Suriyarachchi, C. Mavridis, and J. S. Baras, "Autonomous vehicle overtaking in a bidirectional mixed-traffic setting," in *2022 American Control Conference (ACC)*. IEEE, 2022, pp. 3132–3139.

[18] F. M. Tariq, D. Isele, J. S. Baras, and S. Bae, "Slas: Speed and lane advisory system for highway navigation," in *2022 IEEE 61st Conference on Decision and Control (CDC)*. IEEE, 2022, pp. 6979–6986.

[19] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2020.

[20] X. Chi, Z. Liu, J. Huang, F. Hong, and H. Su, "Optimization-based motion planning for autonomous parking considering dynamic obstacle: A hierarchical framework," in *2022 34th Chinese Control and Decision Conference (CCDC)*. IEEE, 2022, pp. 6229–6234.

[21] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[22] Z. Li, L. Xie, C. Hu, and H. Su, "A rapid iterative trajectory planning method for automated parking through differential flatness," *Robotics and Autonomous Systems*, vol. 182, p. 104816, 2024.

[23] W. Xu, Q. Wang, and J. M. Dolan, "Autonomous vehicle motion planning via recurrent spline optimization," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 7730–7736.

[24] P. Rouchon, M. Fliess, J. Lévine, and P. Martin, "Flatness, motion planning and trailer systems," in *Proceedings of 32nd IEEE Conference on Decision and Control*. IEEE, 1993, pp. 2700–2705.

[25] W. Lim, S. Lee, M. Sunwoo, and K. Jo, "Hierarchical trajectory planning of an autonomous car based on the integration of a sampling and an optimization method," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 2, pp. 613–626, 2018.

[26] L. Wang, Z. Wu, J. Li, and C. Stiller, "Real-time safe stop trajectory planning via multidimensional hybrid a*-algorithm," in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–7.

[27] A. Folkers, M. Rick, and C. Büskens, "Time-dependent hybrid-state a∗ and optimal control for autonomous vehicles in arbitrary and dynamic environments," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 15 077–15 083, 2020.

[28] G. Lan and Q. Hao, "End-to-end planning of autonomous driving in industry and academia: 2022-2023," *arXiv e-prints*, pp. arXiv–2401, 2023.

[29] "Tesla Model X Owner's Manual," 2021, [Online].

[30] E. Bronstein, M. Palatucci, D. Notz, B. White, A. Kuefler, Y. Lu, S. Paul, P. Nikdel, P. Mougin, H. Chen, *et al.*, "Hierarchical model-based imitation learning for planning in autonomous driving," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 8652–8659.

[31] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, R. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson, *et al.*, "Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 7553–7560.

[32] Y. Wu, L. Wang, X. Lu, Y. Wu, and H. Zhang, "Reinforcement learning-based autonomous parking with expert demonstrations," in *2023 7th CAA International Conference on Vehicular Control and Intelligence (CVCI)*. IEEE, 2023, pp. 1–6.

[33] M. Jiang, Y. Li, S. Zhang, C. Wang, and M. Yang, "Hope: A reinforcement learning-based hybrid policy path planner for diverse parking scenarios," *arXiv preprint arXiv:2405.20579*, 2024.

[34] "LA man nearly misses flight as self-driving Waymo taxi drives around parking lot in circles ," 2025, [Online].

[35] Y. Chen, H. Peng, and J. Grizzle, "Obstacle avoidance for low-speed autonomous vehicles with barrier function," *IEEE Transactions on Control Systems Technology*, vol. 26, no. 1, pp. 194–206, 2017.

[36] Y. Wang, E. Hansen, and H. Ahn, "Hierarchical planning for autonomous parking in dynamic environments," *IEEE Transactions on Control Systems Technology*, 2024.

[37] A. Popov, P. Gebhardt, K. Chen, and R. Oldja, "Nvradarnet: Real-time radar obstacle and free space detection for autonomous driving," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 6958–6964.

[38] M. Haris and J. Hou, "Obstacle detection and safely navigate the autonomous vehicle from unexpected obstacles on the driving lane," *Sensors*, vol. 20, no. 17, p. 4719, 2020.

[39] K. Karur, N. Sharma, C. Dharmatti, and J. E. Siegel, "A survey of path planning algorithms for mobile robots," *Vehicles*, vol. 3, no. 3, pp. 448–468, 2021.

[40] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[41] "2024 honda odyssey specifications and features," 2024, [Online].

[42] "Parking Code Regulations California Building Code (CBC) Title 24 Part 2," 2020, [Online].