2020 16th IEEE International Conference on Automation Science and Engineering (CASE)
August 20-21, 2020, Online Zoom Meeting

Energy-Aware Multi-Goal Motion Planning Guided by Monte Carlo Search

Yazz Warsame Stefan Edelkamp Erion Plaku

Abstract— **Autonomous robots need a reliable way to preserve their energy level while performing a persistent task such as inspection or surveillance. Toward this objective, this paper considers the multi-goal motion-planning problem with multiple recharging stations where a robot operating in a complex environment has to reach each goal while reducing the travel distance and the number of times it recharges. This paper develops an integrated approach that couples sampling-based motion planning with Monte-Carlo Tree Search (MCTS). The proposed MCTS searches over a discrete abstraction, which is obtained via a probabilistic roadmap, and uses a reward function to calculate when, where, and whether it is beneficial to recharge. This results in short tours that also reduce the number of recharges. Such tours are used to guide sampling-based motion planning as it expands a tree of collision-free and dynamically-feasible motions. Experiments with nonlinear dynamical robot models operating in obstacle-rich environments demonstrate the efficiency of the approach.**

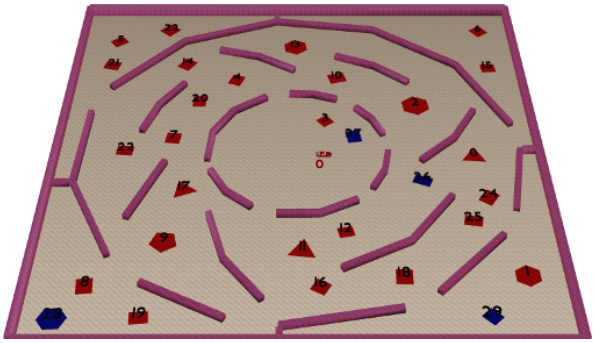


Fig. 1. An example of a multi-goal motion-planning problem with multiple recharging stations. Goal regions, recharging stations, and obstacles are shown in red, blue, and magenta, respectively. The robot is required to visit each goal while reducing the distance traveled and the number of recharges. Videos of solutions obtained by our approach for this and other cases can be seen at https://bit.ly/343zonK

I. Introduction

The demand for self-sufficient autonomous robots is becoming more prevalent as robots are increasingly used for cleaning homes, inspecting large structures, assisting in search-and-rescue missions, and working in warehouses [1], [2], [3]. These tasks usually have a long horizon and the energy required to fulfill the task often exceeds the initial capacity of a mobile robot. As such, it becomes necessary for the robot to also determine when and where to recharge in order to effectively complete its task. This gives rise to a challenging problem as the robot often also has to navigate its way through obstacle-rich environments in order to reach its desired goal locations. Moreover, the planned motions must take into account the underlying robot dynamics, which impose constraints on the velocity, acceleration, direction of motion, and turning radius among others.

To address these challenges, sampling-based motion planning has been often used to enable the robot to effectively reach multiple goal regions while avoiding collisions [4], [5], [6]. These motion planners, however, have assumed unlimited energy capacity, which is not practical in the real world as energy management is considerably important.

This work removes this assumption and considers the multi-goal motion-planning problem with multiple recharging stations. In these scenarios, the robot has to not only decide when it should recharge but also where it should recharge as the recharging stations may be scattered. The objective is to efficiently compute a collision-free and dynamically-feasible trajectory that enables the robot to visit each goal while also reducing the overall distance traveled and the number of times the robot recharges.

To account for the recharging stations, this work leverages the idea of using a discrete abstraction and discrete planning to effectively guide sampling-based motion planning. Motivated by related work in this area [7], the discrete abstraction is obtained via a probabilistic roadmap which provides a network of collision-free routes to facilitate navigation. To address the limitations of the related work (unlimited energy assumption), this paper develops an effective discrete planner based on Monte Carlo Tree Search (MCTS) using Nested Rollout Policy Adaptation (NRPA) in order to effectively compute roadmap tours that enable the robot to reach each remaining goal while reducing the overall distance traveled and the number of recharges. This is made possible by designing a reward function to calculate when, where, and whether it is beneficial to recharge. Such tours guide sampling-based motion planning to more effectively explore the continuous state space of collision-free and dynamically-feasible motions and to incrementally construct trajectories that reach the goals and recharge as indicated by the roadmap tours. Experiments with nonlinear dynamical robot models operating in obstacle-rich environments demonstrate the efficiency of the proposed approach in generating low-cost solutions even as the number of goals and energy considerations become more challenging.

This material is based upon work supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Y. Warsame is with the Dept. of Informatics, King's College London, London, UK. S. Edelkamp is with the Dept. of Informatics, University of Koblenz-Landau, Mainz, Germany. E. Plaku is with the Dept. of Electrical Engineering and Computer Science, Catholic University of America, Washington, DC, USA.

II. Related Work

Related work on planning with multiple recharging stations has often considered simplified settings that do not take into account the differential constraints imposed by the robot dynamics or the geometric constraints imposed by the obstacles in the environment. Even in these simplified settings the problem is challenging as the charging stations are often scattered and the objective is to reduce the distance and number of times the robot has to recharge.

Numerous conditions and strategies for when and where to recharge have been proposed over the years [8], [9], [10], [11], [12]. Fixed threshold approaches [13], [14], [15], where the robot decides to visit a recharging station after having traveled for a certain time or distance, have often been adopted due to the ease of implementation. The threshold is set at the beginning based on the largest distance or time needed to navigate between a recharging station and a goal region. These approaches, however, often lead to long tours and numerous recharges since they do not take into account the current robot location and the location of the nearest recharging station. Adaptive threshold [16], [17], [18] was introduced to handle the shortcomings of a fixed threshold by considering the current robot location and the nearest recharging station. If the robot has sufficient energy to travel to the nearest goal and then to the nearest recharging station, then the robot goes to the nearest goal; otherwise, it goes to the nearest recharging station. Even though the adaptive threshold improves over the fixed threshold it still can generate long tours especially when the charging stations are away from the current robot location. The work in [16] proposed an online heuristic algorithm that uses a rate-maximizing foraging model to increase the time a robot spends working instead of traveling to the charging station where the robot would not be doing any work. The work in [19] developed a genetic algorithm to solve an energy-aware multi-goal path-planning problem. Linear Temporal Logic has also been used to solve energy-aware pickup and delivery problems, but does not scale well [20].

While significant progress has been made, the simplified problem settings often limit the applicability of these approaches. In contrast, our approach considers a richer problem setting that takes into account multiple goals and charging stations as well as the geometric and differential constraints imposed by the obstacles and the robot dynamics.

III. Problem Formulation

This section defines the robot model and the multi-goal motion planning problem with recharging stations.

Robot Model and Trajectories: The robot is defined as a tuple $\langle \mathcal{M}, \mathcal{Q}, \mathcal{S}, \mathcal{U}, f \rangle$ in terms of its geometric shape $\mathcal{M}$, configuration space $\mathcal{Q}$, state space $\mathcal{S}$, control space $\mathcal{U}$, and motion equations f. A configuration $q \in \mathcal{Q}$ defines the position and orientation. A state $s \in \mathcal{S}$ augments the configuration by including other variables, such as velocity and steering angle, that are necessary to define the robot motion. The notation $\text{CFG}(s)$ denotes the configuration component of a state $s \in \mathcal{S}$. A control $u \in \mathcal{U}$ defines external inputs, such as acceleration and steering angle, that are used to control the robot. The motion equations are often expressed as a set of differential equations of the form

$$\dot{s} \leftarrow f(s, u). \tag{1}$$

As an example, the motion equations of the vehicle model shown in Fig. 2 are defined as

$$\dot{x} = v\cos(\theta)\cos(\psi), \dot{y} = v\sin(\theta)\cos(\psi), \tag{2}$$

$$\dot{\theta} = v\sin(\psi), \dot{v} = a, \dot{\psi} = \omega, \tag{3}$$

where the state $s = (x, y, \theta, \psi, v)$ defines the position (x, y), orientation θ, steering angle ψ, and velocity v, while the control $u = (a, \omega)$ defines the acceleration a and steering rate ω. As another example, a snake model can be obtained by attaching N trailers to the car and augmenting f as

$$\dot{\theta}_i = (v/H)(\sin(\theta_{i-1}) - \sin(\theta_0))\textstyle\prod_{j=1}^{i-1}\cos(\theta_{j-1} - \theta_j), \tag{4}$$

where θ_i is the orientation of the i-th trailer, $\theta_0 = \theta$, and H is the hitch distance [4].

When applying a control $u \in \mathcal{U}$ to a state $s \in \mathcal{S}$ for a time step dt, the new state $s_{\text{new}} \in \mathcal{S}$ is computed by a function

$$s_{\text{new}} \leftarrow \text{SIMULATE}(s, u, f, dt), \tag{5}$$

which numerically integrates f for one time step dt. Applying a sequence of controls $\langle u_1, \ldots, u_{\ell-1} \rangle$ gives rise to a dynamically-feasible motion trajectory $\zeta : \{1, \ldots, \ell\} \to \mathcal{S}$, where $\zeta(1) \leftarrow s$ and $\forall i \in \{2, \ldots, \ell\}$:

$$\zeta(i) \leftarrow \text{SIMULATE}(\zeta(i-1), u_{i-1}, f, dt). \tag{6}$$

Multi-Goal Motion Planning with Recharging Stations: The world $\mathcal{W}$ contains goal regions $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$, recharging regions $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, and obstacles. When the robot visits a recharging region, its energy level e is set back to max capacity, i.e., $e = e_{\max}$. Starting from an initial state $s_{\text{init}} \in \mathcal{S}$ with an energy level e_{init}, the objective is to compute a collision-free and dynamically-feasible trajectory ζ such that the robot visits all the goals, recharging as necessary to avoid depleting the energy level. The energy consumption is assumed to be proportional to the distance d traveled by the robot. Nonlinear energy-consumption models can also be incorporated into the framework.

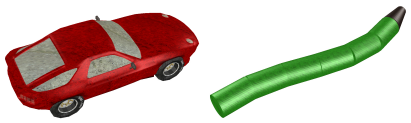


Fig. 2. Robot models (car, snake) used in the experiments.

IV. Discrete Problem Solving

Our overall approach relies on a discrete abstraction to guide sampling-based motion planning. The discrete abstraction represents a simplified problem setting that ignores the vehicle dynamics but takes into account the distance between goals, charging stations, current energy level, and the energy required to travel from one location to the other. We also develop an approach based on Monte-Carlo search to effectively solve the abstract problem.

A. Problem Abstraction as a Graph with Energy Constraints

The problem abstraction is represented as a tuple $\mathcal{A} = \langle V = V_{\text{goals}} \cup V_{\text{charge}} \cup \{v_{\text{curr}}\}, \mathcal{D}, \mathcal{E}, e_{\max}, e_{\text{curr}} \rangle$, where

- V_{goals} is the set of goal vertices;
- V_{charge} is the set of recharging vertices;
- v_{curr} is the current robot location;
- $\mathcal{D} = \langle d_{(v',v'')} : (v', v'') \in V \times V \rangle$ is a table, where $d_{(v',v'')}$ is the distance from v' to v'';
- $\mathcal{E} = \langle e_{(v',v'')} : (v', v'') \in V \times V \rangle$ is a table, where $e_{(v',v'')}$ is the energy required to travel from v' to v'';
- $e_{\max}$ is the maximum energy level at a charging vertex;
- e_{curr} is the current energy level, where $0 < e_{\text{curr}} \leq e_{\max}$.

Note that there is no restriction imposed on v_{curr}, so it could be a goal vertex, a charge vertex, or neither.

A solution for $\mathcal{A}$ is a tour $\tau = \langle \tau_1, \ldots, \tau_k \rangle$ that starts at v_{curr}, visits every vertex in V_{goals}, and ensures that there is sufficient energy to move from each τ_i to τ_{i+1}, i.e., remaining energy at τ_i is at least $e_{(\tau_i,\tau_{i+1})}$. When at τ_{i+1}, the robot's energy is set to $e_{\max}$ if $\tau_{i+1} \in V_{\text{charge}}$. Otherwise, $e_{(\tau_i,\tau_{i+1})}$ is subtracted from its energy level. A violation occurs when the robot's energy falls below 0. Preference is given to those tours that reduce the overall distance $\sum_{i=1}^{|\tau|-1} d_{(\tau_i,\tau_{i+1})}$.

Note that our overall approach will create numerous problem abstractions, each time modifying V_{goals}, $\mathcal{D}$, $\mathcal{E}$, v_{curr}, and e_{curr} depending on the progress that it makes. While initially the abstraction will include all the goal regions $\mathcal{G}_1, \ldots, \mathcal{G}_n$, the number of goals will be continually reduced as more and more goals are reached by the robot.

B. Monte-Carlo Search over the Problem Abstraction

To effectively compute low-cost tours for the abstract problem, we developed an approach based on Monte-Carlo Tree Search (MCTS) using Nested Rollout Policy Adaptation (NRPA) [21]. MCTS has been successfully used in games, planning, optimization, and many other areas [22], [23]. MCTS uses rollouts to guide the search, where a rollout is a randomized tour which uses a probability distribution based on policy weights to determine the next move. NRPA uses gradient ascent to emphasize moves that push rollouts into the neighborhood of the current best tour.

Pseudocode is shown in Alg. 1. Note that Alg. 1 follows the same design as the original NRPA algorithm [21]. Our contribution is the function ROLLOUT, which promotes the generation of energy-aware and short tours.

NRPA proceeds recursively, invoking ROLLOUT at level 0 to compute a randomized tour which uses the policy weights to determine the next move. ROLLOUT (Alg. 1b) starts from the current vertex v_{curr} and moves to another vertex until all the goals in V_{goals} have been visited. As shown in Alg. 1b3, only those vertices that can be reached with the current energy level are considered as its children. The next vertex, v_{next}, is selected with probability based on the policy weights (Alg. 1b5). To reduce the number of recharges, a penalty is applied when an unnecessary trip to a charging station is made, i.e., v_{next} is a charging station but there is an unreached goal that can be visited with the current energy level (Alg. 1b6-7). The vertex v_{next} is appended to the tour. The current energy level is set to $e_{\max}$ if v_{next} represents a charging station. Otherwise, the current energy is reduced by the amount of energy required to move to v_{next} (Alg. 1b8-9). If $v_{\text{next}} \in V_{\text{goals}}$, then v_{next} is removed from the remaining goal vertices. The process continues from v_{next} until all the goals have been reached. At the end, ROLLOUT returns the tour τ and its cost, computed as $\lambda * \text{penalty} + d$, where λ is a user-defined constant, penalty is the number of energy violations, and d is the overall distance traveled. This combination enables our approach to reduce the number of recharges and the overall distance.

Algorithm 1 MCTS_NRPA: Discrete Planner

Input: $\mathcal{A} = \langle V = V_{\text{goals}} \cup V_{\text{charge}} \cup \{v_{\text{curr}}\}, \mathcal{D}, \mathcal{E}, e_{\max}, e_{\text{curr}} \rangle$: discrete problem as defined in Section IV-A;
$\ell_{\max}$: maximum level for MCTS/NRPA search
Output: a tour that visits all the goals while seeking to reduce the overall distance and the number of recharges

1: pol $\leftarrow \mathcal{D}$ *// vector of policy weights*
2: **return** NRPA$(\ell_{\max}, \text{pol})$

(a) NRPA(ℓ, pol)
1: **if** $\ell = 0$ **then**
2: **return** ROLLOUT$(\text{pol}, v_{\text{curr}}, e_{\text{curr}})$
3: **else**
4: mincost $\leftarrow \infty$; $\tau \leftarrow$ null
5: **for** several times **do**
6: $\langle \tau_{\text{new}}, \text{cost} \rangle \leftarrow$ NRPA$(\ell - 1, \text{pol})$
7: **if** cost $\leq$ mincost **then** $\{\tau \leftarrow \tau_{\text{new}}; \text{mincost} \leftarrow \text{cost}\}$
8: pol $\leftarrow$ ADAPT(pol, τ)
9: **return** $\langle \tau, \text{mincost} \rangle$

(b) ROLLOUT$(\text{pol}, \text{v}, e)$
1: $V_{\text{rem}} \leftarrow V_{\text{goals}} \setminus \{v\}$; $\tau \leftarrow \langle v \rangle$; $d \leftarrow 0$; penalty $\leftarrow 0$
2: **while** $V_{\text{rem}} \neq \emptyset$ **do**
3: children $\leftarrow \{v' : v' \in V_{\text{rem}} \cup V_{\text{charge}} \wedge e \geq e_{(v,v')}\}$
4: **if** children $= \emptyset$ **then return** $\langle \tau, \infty \rangle$
//select child with probability $\frac{\text{pol}[(v,v_{\text{next}})]}{\sum_{v' \in \text{children}} \text{pol}[(v,v')]}$
5: $v_{\text{next}} \leftarrow$ SELECT(children, v, pol)
6: **if** $v_{\text{next}} \in V_{\text{charge}} \wedge \exists v' \in V_{\text{rem}}$ s.t. $e \geq e_{(v,v')}$ **then**
7: penalty $\leftarrow$ penalty $+ 1$
8: $\tau \leftarrow \langle \tau, v_{\text{next}} \rangle$; $d \leftarrow d + d_{(\text{v},\text{v}_{\text{next}})}$
9: **if** $v_{\text{next}} \in V_{\text{charge}}$ **then** $e \leftarrow e_{\max}$ **else** $e \leftarrow e - e_{(\text{v},\text{v}_{\text{next}})}$
10: **if** $v_{\text{next}} \in V_{\text{goals}}$ **then** $V_{\text{rem}} \leftarrow V_{\text{rem}} \setminus \{v_{\text{next}}\}$
11: $v \leftarrow v_{\text{next}}$
12: **return** $\langle \tau, \lambda * \text{penalty} + d \rangle$

V. Overall Approach

The overall approach uses the discrete abstraction and the discrete solver to guide sampling-based motion planning. The discrete abstraction is obtained via a probabilistic roadmap that provides a network of collision-free routes to facilitate navigation. A motion tree is incrementally expanded in the state space by adding collision-free and dynamically-feasible trajectories as branches. Tours obtained via the discrete planner over the abstraction are used to guide the motion-tree expansion. To facilitate the interaction between the discrete planner and the motion-tree expansion, the motion tree is partitioned into equivalence classes based on the discrete abstraction and energy considerations. Pseudocode is shown in Alg. 3. The main steps are described below.

A. Discrete Abstraction via Roadmaps

The discrete abstraction is based on a simplified problem setting, which ignores the robot dynamics. Leveraging the

Algorithm 2 Roadmap Construction.

Input: $\mathcal{W}$: world; $\mathcal{O}$: obstacles; $\mathcal{G}$: goal regions; $\mathcal{C}$: recharging regions; $\mathcal{M}$: robot geometry; n_{add}: number of configurations to add to roadmap at one time; $n_{\max}$: maximum number of roadmap configurations; $d_{\min}$: minimum distance to check for collisions during path interpolation
Output: roadmap $\mathcal{R} = (V_{\mathcal{R}}, E_{\mathcal{R}})$

1: $V_{\mathcal{R}} \leftarrow E_{\mathcal{R}} \leftarrow \Delta \leftarrow V_{\text{add}} \leftarrow \text{attempts} \leftarrow \emptyset$
2: **for** $\mathcal{Z} \in \mathcal{G} \cup \mathcal{C}$ **do** ADDCFG$(\mathcal{Z}, \mathcal{O}, \mathcal{M}, V_{\mathcal{R}}, \Delta, V_{\text{add}})$
3: **repeat**
4: **for** $k = 1 \ldots n_{\text{add}}$ **do** ADDCFG$(\mathcal{W}, \mathcal{O}, \mathcal{M}, V_{\mathcal{R}}, \Delta, V_{\text{add}})$
5: **for** $q \in V_{\text{add}}$ **do** ADDEDGES$(q, \mathcal{O}, \mathcal{M}, V_{\mathcal{R}}, E_{\mathcal{R}}, \Delta, \text{attempts})$
6: $V_{\text{add}} \leftarrow \emptyset$
7: **until** SAMECOMPONENT$(\Delta, \{q_{\mathcal{Z}} : \mathcal{Z} \in \mathcal{G} \cup \mathcal{C}\}) \vee$ (SAMECOMPONENT$(\Delta, \{q_{\mathcal{Z}} : \mathcal{Z} \in \mathcal{G}\}) \wedge |V_{\mathcal{R}}| \geq n_{\max}$)
8: **return** $\mathcal{R} = (V_{\mathcal{R}}, E_{\mathcal{R}})$

(a) **ADDCFG**$(\mathcal{Z}, \mathcal{O}, \mathcal{M}, V_{\mathcal{R}}, \Delta, V_{\text{add}})$
1: **repeat** $q \leftarrow \langle$RANDOMPOSITION$(\mathcal{Z})$, RANDOMORIENTATION$()\rangle$
2: **until** $\neg$MESHCOLLISION(TRANSFORMMESH$(\mathcal{M}, q), \mathcal{O})$
3: $V_{\mathcal{R}} \leftarrow V_{\mathcal{R}} \cup \{q\}$; $V_{\text{add}} \leftarrow V_{\text{add}} \cup \{q\}$; $\Delta[q] \leftarrow$ MAKESET$()$

(b) **ADDEDGES**$(q, \mathcal{O}, \mathcal{M}, V_{\mathcal{R}}, E_{\mathcal{R}}, \Delta, \text{attempts})$
1: neighs $\leftarrow$ NEARESTNEIGHBORS$(q, V_{\mathcal{R}}, \lceil \log_2 |V_{\mathcal{R}}| \rceil)$
2: **for** $q_{\text{neigh}} \in \text{neighs} \wedge (q, q_{\text{neigh}}) \notin \text{attempts}$ **do**
3: **if** $\neg$ISPATHINCOLLISION$(q, q_{\text{neigh}}, \mathcal{O}, \mathcal{M})$ **then**
4: $E_{\mathcal{R}} \leftarrow E_{\mathcal{R}} \cup \{(q, q_{\text{neigh}}), (q_{\text{neigh}}, q)\}$
5: JOINSETS$(\Delta[q], \Delta[q_{\text{neigh}}])$
6: attempts $\leftarrow$ attempts $\cup \{(q, q_{\text{neigh}}), (q_{\text{neigh}}, q)\}$

(c) **SAMECOMPONENT**$(\Delta, \{q_1, \ldots, q_m\})$
1: **for** $i = 1 \ldots m$ **do**
2: **for** $j = i + 1 \ldots m$ **do**
3: **if** FINDSET$(\Delta[q_i]) \neq$ FINDSET$(\Delta[q_j])$ **then**
4: **return** `false`
5: **return** `true`

(d) **ISPATHINCOLLISION**$(q_1, q_2, \mathcal{O}, \mathcal{M})$
1: $\mathcal{M}_{\text{path}} \leftarrow \emptyset$
2: nrSteps $\leftarrow \lceil$DISTANCE$(q_1, q_2)/d_{\min} \rceil$
3: **for** $k = 1 \ldots \text{nrSteps} - 1$ **do**
4: $q \leftarrow (1 - k/\text{nrSteps})q_1 + (k/\text{nrSteps})q_2$
5: $\mathcal{M}_{\text{path}} \leftarrow$ ADDMESH$(\mathcal{M}_{\text{path}}$, TRANSFORMMESH$(\mathcal{M}, q))$
6: **return** MESHCOLLISION$(\mathcal{M}_{\text{path}}, \mathcal{O})$

Probabilistic RoadMap (PRM) method [24], the discrete abstraction is obtained as a roadmap $\mathcal{R} = (V_{\mathcal{R}}, E_{\mathcal{R}})$ over the configuration space $\mathcal{Q}$. The objective is to create a dense roadmap that connects the goal regions and charging stations, providing several routes to go from one location to the other.

Pseudocode is shown in Alg. 2. The roadmap construction starts by adding a representative configuration, denoted by $q_{\mathcal{G}_i}$ and $q_{\mathcal{C}_j}$, for each goal $\mathcal{G}_i \in \mathcal{G}$ and recharging station $\mathcal{C}_j \in \mathcal{C}$ (Alg. 2:2). The roadmap is further populated by repeatedly sampling random configurations and keeping only those that are not in collision (Alg. 2:4). Attempts are then made to connect each configuration in $\mathcal{R}$ to several of its nearest neighbors (Alg. 2:5), where the distance metric $\rho : \mathcal{Q} \times \mathcal{Q} \rightarrow \mathbb{R}^{\geq 0}$ is $\rho(q', q'') = ||\text{POSITION}(q') - \text{POSITION}(q'')||_2$. The number of neighbors is set to $\lceil \log_2 |V_{\mathcal{R}}| \rceil$ to provide a dense roadmap, as recommended in the literature [25]. When the path from a configuration q to its neighbor q_{neigh} is not in collision, then the edge (q, q_{neigh}) is added to $E_{\mathcal{R}}$. Path collision checking is performed by adding the intermediate configurations along the path to a mesh to compute the volume swept by the robot as it moves from q to q_{neigh} and then using a collision-detection package such as PQP [26] to determine whether the mesh collides with the obstacles.

This process of adding and connecting configurations is repeated until all the goal and recharging configurations belong to the same connected component in $\mathcal{R}$ or a maximum number of iterations is reached (Alg. 2:7). A disjoint set data structure is used to efficiently keep track of the connected components in the roadmap.

To facilitate the tour computations, after the roadmap is constructed, we use Dijkstra's single-source shortest paths algorithm to compute the shortest paths from each configuration to each goal $q_{\mathcal{G}_i}$ and recharging $q_{\mathcal{C}_j}$ configuration. Dijkstra's algorithm is invoked $|\mathcal{G}| + |\mathcal{C}|$ times, each time using $q_{\mathcal{G}_i}$ (or $q_{\mathcal{C}_j}$) as the source. The notation $\text{SP}(\mathcal{R}, q', q'')$ is used to denote the shortest path in $\mathcal{R}$ from q' to q'', while $\text{SPD}(\mathcal{R}, q', q'')$ denotes the shortest path distance.

B. Using Tours and the Discrete Abstraction to Partition the Motion Tree into Equivalence Classes

To account for the dynamics, a motion tree $\mathcal{T}$ is rooted at the initial state s_{init} and is expanded in the state space $\mathcal{S}$. Each vertex $v \in \mathcal{T}$ has the fields $\{s, e, \text{goals}, q_{\text{nearest}}, \text{parent}, u\}$, which denote the following: (i) $v.s$: collision-free state in $\mathcal{S}$; (ii) $v.e$: remaining energy; (iii) $v.\text{goals}$: goals in $\mathcal{G}$ that have not been reached by $\zeta_{\mathcal{T}}(v)$, where $\zeta_{\mathcal{T}}(v)$ is the trajectory from the root of $\mathcal{T}$ to v; (iv) $v.q_{\text{nearest}}$: nearest configuration in the roadmap $\mathcal{R}$, i.e., $v.q_{\text{nearest}} = \text{argmin}_{q \in V_{\mathcal{R}}} \rho(\text{CFG}(v.s), q)$; (v) $v.\text{parent}$: pointer to the parent node in $\mathcal{T}$; and (vi) $v.u$: control in $\mathcal{U}$. By construction, $v.s \leftarrow \text{SIMULATE}(v.\text{parent}.s, u, f, dt)$ (except for the root, which has no parent).

A solution to the multi-goal motion-planning problem with energy constraints (Section III) is found when a vertex v is added to $\mathcal{T}$ such that $v.\text{goals} = \emptyset$, i.e., $\zeta_{\mathcal{T}}(v)$ has reached all the goals in $\mathcal{G}$.

We leverage the discrete abstraction to guide the motion-tree expansion from v. Essentially, each vertex $v \in \mathcal{T}$ gives rise to a discrete problem to find a short tour τ that starts at the nearest roadmap configuration, $v.q_{\text{nearest}}$, and visits each of the remaining goals while reducing the number of visits to charging stations. When computing the tour, we also have to account for the energy required to move from v to $v.q_{\text{nearest}}$, which, according to our model, is proportional to the distance traveled. Formally, the discrete problem associated with $v \in \mathcal{T}$ is defined as $\mathcal{A}(v) = \langle V = V_{\text{goals}} \cup V_{\text{charge}} \cup \{v_{\text{curr}}\}, \mathcal{D}, \mathcal{E}, e_{\max}, e_{\text{curr}} \rangle$, where

- $V_{\text{goals}} = \{q_{\mathcal{G}_i} : \mathcal{G}_i \in v.\text{goals}\}$,
- $V_{\text{charge}} = \{q_{\mathcal{C}_j} : \mathcal{C}_j \in \mathcal{C}\}$,
- $v_{\text{curr}} = v.q_{\text{nearest}}$,
- $\mathcal{D} = \langle d_{(q',q'')} : q', q' \in V \rangle$, $d_{(q',q'')} = \text{SPD}(\mathcal{R}, q', q'')$,
- $\mathcal{E} = \langle e_{(q',q'')} : q', q' \in V \rangle$, $e_{(q',q'')} \propto d_{(q',q'')}$, and
- $e_{\text{curr}} = v.e - \text{ENERGY}(\rho(\text{CFG}(v.s), v.q_{\text{nearest}}))$.

After computing τ, attempts are then made to expand $\mathcal{T}$ from v along τ. Since $\mathcal{T}$ typically has tens of thousands of vertices, it is infeasible to compute tours for each vertex $v \in \mathcal{T}$. For this reason, we group together vertices in $\mathcal{T}$ that provide the same discrete information. Specifically, a new vertex v_{new} belongs to the same equivalence class as v when they have the same set of remaining goals, map to the same roadmap

Algorithm 3 Overall search.

Input: $\mathcal{W}$: world; $\mathcal{O}$: obstacles; $\mathcal{G} = \{\mathcal{G}_1, \ldots, \mathcal{G}_n\}$: goals; $\mathcal{C} = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$: recharging stations; $\langle \mathcal{M}, \mathcal{Q}, \mathcal{S}, \mathcal{U}, f \rangle$: robot model; s_{init}: initial state; dt: time step; $t_{\max}$: runtime limit
Output: collision-free and dynamically-feasible trajectory that reaches each goal or `null` if no solution is found

1: $\mathcal{R} \leftarrow$ ConstructRoadmap$(\mathcal{W}, \mathcal{O}, \mathcal{G}, \mathcal{C}, \mathcal{M})$
2: $\Xi \leftarrow$ ShortestPaths$(\mathcal{R}, \mathcal{G}, \mathcal{C})$
3: $\langle \mathcal{T}, \Gamma \rangle \leftarrow$ initialize(s_{init})
4: **while** time$() < t_{\max}$ **do**
5: $\quad \Gamma_\tau \leftarrow$ SelectEquivalenceClass(Γ)
6: $\quad \sigma = \langle q_1, \ldots, q_m \rangle \leftarrow$ TourToPaths$(\mathcal{R}, \Xi, \tau)$
7: $\quad v \leftarrow$ SelectVertexFromEquivalenceClass(Γ_τ)
8: $\quad$ follow(v, σ)
9: $\quad$ **for** each new vertex v_{new} added to $\mathcal{T}$ **do**
10: $\quad\quad$ **if** $(\Gamma_{\tau_{\text{new}}} \leftarrow$ find$(\Gamma, v_{\text{new}})) =$ `null` **then**
11: $\quad\quad\quad \tau_{\text{new}} \leftarrow$ mcts_nrpa(DiscreteProblem$(\mathcal{R}, \Xi, \mathcal{G}, \mathcal{C}, v_{\text{new}})$)
12: $\quad\quad\quad$ **if** $\tau_{\text{new}} =$ `null` **then continue**
13: $\quad\quad\quad \Gamma \leftarrow \Gamma \cup \{\Gamma_{\tau_{\text{new}}}\}$
14: $\quad\quad \Gamma_{\tau_{\text{new}}} \leftarrow \Gamma_{\tau_{\text{new}}} \cup \{v_{\text{new}}\}$
15: $\quad\quad$ **if** v_{new}.goals $= \emptyset$ **then return** $\zeta_{\mathcal{T}}(v_{\text{new}})$
16: **return** `null`

(a) follow$(v_{\text{from}}, \sigma)$
1: $\mathcal{X} \leftarrow \{v_{\text{from}}\}$; $i \leftarrow 1$; count $\leftarrow 0$
2: **while** $i \leq |\sigma| \wedge$ rand$(0,1) \leq 1/$count **do**
3: $\quad$ count $\leftarrow$ count $+ 1$
4: $\quad p \leftarrow$ SampleTarget(σ_i)
5: $\quad v \leftarrow$ SelectVertex$(\mathcal{X}, p)$
6: $\quad$ **for** several steps **do**
7: $\quad\quad u \leftarrow$ controller$(v.s, p)$
8: $\quad\quad s_{\text{new}} \leftarrow$ simulate$(v.s, u, f, dt)$
9: $\quad\quad e \leftarrow v.e -$ energy$(\rho($cfg$(v.s),$ cfg$(s_{\text{new}})))$
10: $\quad\quad$ **if** $e < 0 \vee$ collision$(s_{\text{new}}) \vee \neg$near$(s_{\text{new}}, \tau)$ **then break**
11: $\quad\quad v_{\text{new}}.\langle s, \text{parent}, u \rangle \leftarrow$ NewVertex(s_{new}, v, u)
12: $\quad\quad$ **if** reached$(s_{\text{new}}, \mathcal{C})$ **then** $v_{\text{new}}.e \leftarrow e_{\max}$ **else** $v_{\text{new}}.e \leftarrow e$
13: $\quad\quad v_{\text{new}}.q_{\text{nearest}} \leftarrow \text{argmin}_{q \in V_{\mathcal{R}}} \rho(cfg(s_{\text{new}}), q)$
14: $\quad\quad v_{\text{new}}$.goals $\leftarrow v$.goals $\setminus \{$reached$(s_{\text{new}}, \mathcal{G})\}$
15: $\quad\quad$ **if** reached$(s_{\text{new}}, \sigma_i)$ **then**
16: $\quad\quad\quad \mathcal{X} \leftarrow \{v_{\text{new}}\}$; $i \leftarrow i + 1$; count $\leftarrow 0$
17: $\quad\quad$ **else**
18: $\quad\quad\quad \mathcal{X} \leftarrow \mathcal{X} \cup \{v_{\text{new}}\}$
19: $\quad\quad v \leftarrow v_{\text{new}}$

configuration, and the tour associated with v is compatible with the energy available at v_{new}. Formally, the equivalence class Γ_τ associated with τ is defined as

$$\Gamma_\tau = \{v : v \in \mathcal{T} \wedge v.q_{\text{nearest}} = \tau_1 \wedge v.\text{goals} = \text{GOALS}(\tau) \wedge v.e - \text{ENERGY}(\rho(\text{CFG}(v.s), v.q_{\text{nearest}})) \geq \text{MINSTARTENERGY}(\tau)\}, \quad (7)$$

where $\text{GOALS}(\tau)$ denotes the goals visited by τ, i.e.,

$$\text{GOALS}(\tau) = \{\mathcal{G}_i : \mathcal{G}_i \in \mathcal{G} \wedge \exists \tau_j \in \tau : \tau_j = q_{\mathcal{G}_i}\}, \quad (8)$$

and $\text{MINSTARTENERGY}(\tau)$ defines the minimum energy required to reach the first recharging station in τ (or the end of τ if τ has no recharging stations). Note that

$$\text{MINSTARTENERGY}(\tau) = \text{ENERGY}(\sum_{i=1}^{k-1} \text{SPD}(\mathcal{R}, \tau_i, \tau_{i+1})), \quad (9)$$

where k is the index of the first recharging station in τ (if τ has no recharging stations, then k is set to $|\tau| + 1$.)

Let Γ denote all the current equivalence classes. Initially, $\Gamma = \{\Gamma_{\tau_{\text{init}}}\}$, where τ_{init} is the tour obtained by invoking the discrete planner over the discrete problem $\mathcal{A}(v_{\text{init}})$ associated with the root vertex v_{init} of $\mathcal{T}$. When a new vertex v_{new} is added to $\mathcal{T}$, a search is performed to determine whether or not v_{new} belongs to an equivalence class in Γ. If not, a new equivalence class $\Gamma_{\tau_{\text{new}}}$ is created, where τ_{new} is the tour obtained by invoking the discrete planner over $\mathcal{A}(v_{\text{new}})$.

C. Overall Search

Pseudocode for the overall approach is shown in Alg. 3. It starts by constructing the roadmap and computing the shortest paths from each goal and charge station to every roadmap vertex (Alg. 3:1-2), as described in Section V-A. The motion tree $\mathcal{T}$ is rooted at the initial state s_{init}, and the first equivalence class containing the root vertex is created (Alg. 3:3). The search is driven by methods to select an equivalence class Γ_τ from Γ and then expand $\mathcal{T}$ to follow the tour τ. The equivalence classes are updated after each new vertex is added to $\mathcal{T}$. These methods are invoked repeatedly until a solution is found or a runtime limit is reached.

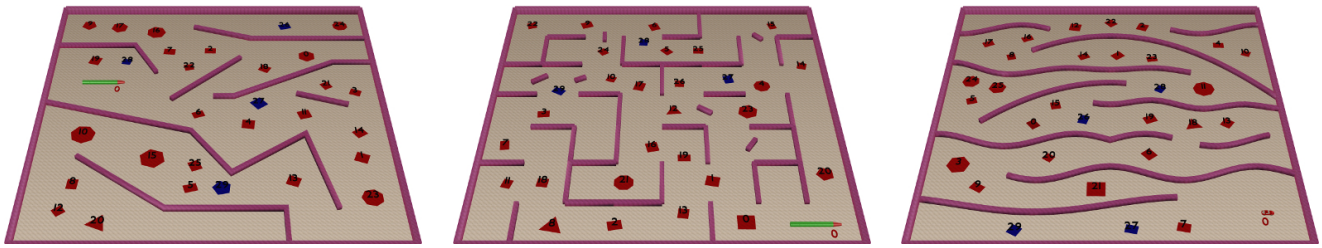
1) Selecting an Equivalence Class: A weight $w(\Gamma_\tau)$ is defined for each equivalence class Γ_τ as

$$w(\Gamma_\tau) = \alpha^{\text{NRSEL}(\Gamma_\tau)} / \text{COST}(\tau), \quad (10)$$

where $0 < \alpha < 1$, $\text{NRSEL}(\Gamma_\tau)$ denotes the number of times Γ_τ has been previously selected, and $\text{COST}(\Gamma_\tau)$ denotes the cost (as computed by MCTS_NRPA) of the tour τ. Among the equivalence classes in Γ, the one with the maximum weight is selected for expansion. This gives priority to equivalence classes associated with short tours and few recharges. The term $\alpha^{\text{NRSEL}(\Gamma_\tau)}$, $0 < \alpha < 1$, serves as a penalty to avoid selecting the same equivalence class again and again.

2) Expanding the Motion Tree to Follow the Tour: After selecting an equivalence class Γ_τ, the objective becomes to expand $\mathcal{T}$ to follow the tour τ. The tour τ is first converted into one roadmap path σ by concatenating the shortest paths that connect the vertices in τ in succession (Alg. 3:6). The vertex $v_{\text{from}} \in \Gamma_\tau$ from which to start the expansion of $\mathcal{T}$ is selected from Γ_τ with probability $v_{\text{from}}.e / \sum_{v \in \Gamma_\tau} v.e$ (Alg. 3:7). This selection seeks to make it easier to satisfy the energy constraints since expansions are promoted from vertices that have high levels of energy.

Starting from v_{from}, $\text{FOLLOW}(v_{\text{from}}, \sigma)$ seeks to reach $\sigma_1, \ldots, \sigma_m$ in succession, where $m = |\sigma|$. FOLLOW maintains a set $\mathcal{X}$ which contains the candidate vertices which can be expanded to reach σ_i. Initially, $\mathcal{X}$ contains v_{from} and the first objective is to reach σ_1. When a vertex v_{new} reaches σ_1, then $\mathcal{X}$ is reset to contain only v_{new} and the objective becomes to reach σ_2, and so on. More specifically, suppose the objective is to reach σ_i. A target point p is sampled within a distance d_{near} from σ_i (Alg. 3:FOLLOW:4) and the nearest vertex $v \in \mathcal{X}$ to p is selected (Alg. 3:FOLLOW:5). A PID controller is then used to generate a trajectory that steers the vehicle from v to the target p (Alg. 3:FOLLOW:7–8). The expansion from v stops if the new state s_{new}, obtained after each simulation step, is found to be in collision or its energy level drops below 0 (Alg. 3:FOLLOW:10). As the objective is to closely follow σ, the expansion from v also stops if s_{new}



(a) scene 2 (b) scene 3 (c) scene 4

Fig. 3. Scenes used in the experiments (scene 1 shown in Fig. 1.)

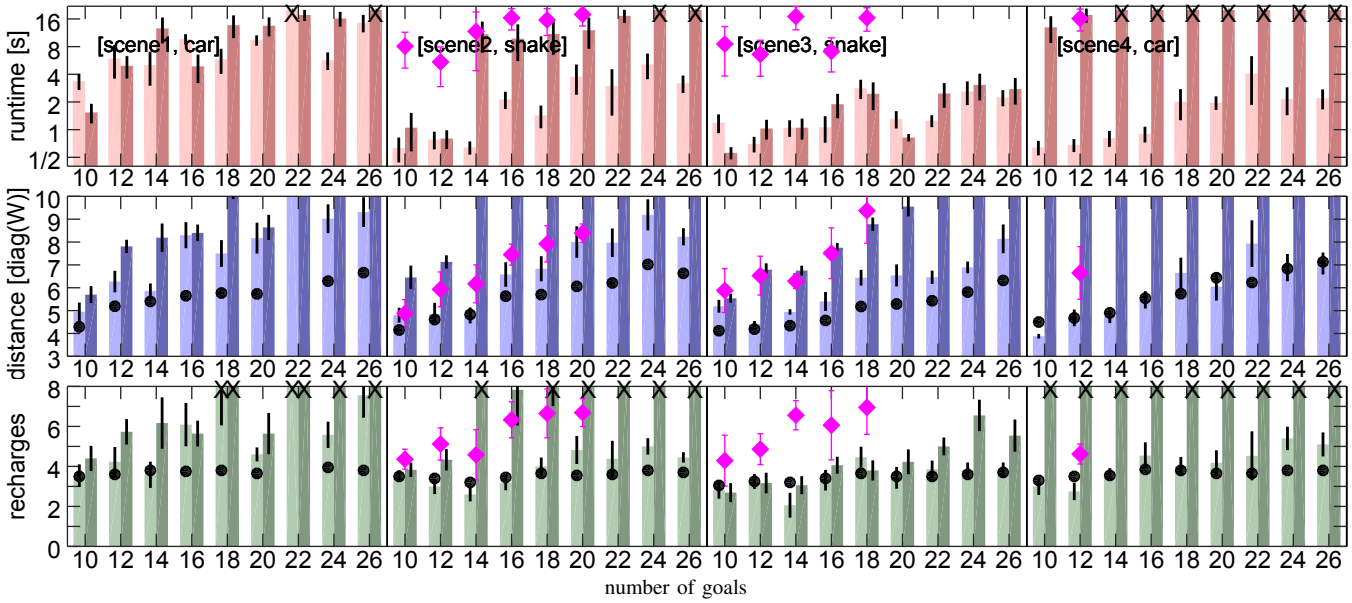


Fig. 4. Results when varying the number of goals. Results for $P_{\text{MCTS\_NRPA}}$ and P_{AT} are shown by the first and second bar, respectively. Results for SEQUENTIALRRT are shown by the diamond symbol. (top) Runtime results. An entry marked with X or a missing entry indicates that the planner was not able to solve the corresponding problem instances. (middle) Distance results. Distance is scaled by the length of the diagonal of the bounding box $\mathcal{W}$. The black circle indicates the overall distance of the tour computed by MCTS_NRPA when run from the initial state (the tour associated with the equivalence class of the root of the motion tree $\mathcal{T}$). (bottom) Recharges refers to the number of times the robot visited a charging station. The black circle indicates the number of recharges associated with the tour computed by MCTS_NRPA when run from the initial state. For these experiments, the number of charging stations is fixed at 4 and the available energy level is $e_{\max} = 250$, which is sufficient to travel 2.2 times along the diagonal of the bounding box of $\mathcal{W}$.

is more than a distance d_{follow} away from σ. If the expansion is successful, a new vertex v_{new} is created with s_{new}, v, and u as its state, parent, and input control (Alg. 3:FOLLOW:11). If s_{new} reaches a recharging station, then its energy is set to $e_{\max}$; otherwise, the energy is reduced by the energy consumed to move from $v.s$ to s_{new} (Alg. 3:FOLLOW:12). If s_{new} reaches a goal $\mathcal{G}_i \in v.\text{goals}$, then $\mathcal{G}_i$ is not included in $v_{\text{new}}.\text{goals}$. If s_{new} reaches σ_{i+1}, then $\mathcal{X}$ is reset to contain only v_{new} as the new objective becomes to reach σ_{i+1}; otherwise, v_{new} is added to $\mathcal{X}$ (Alg. 3:FOLLOW:15–18).

FOLLOW continues until $\sigma_1, \ldots, \sigma_m$ have all been reached in succession. However, since constraints imposed by dynamics and obstacles could make it difficult or impossible to follow certain paths, FOLLOW also terminates when no new progress is made. Essentially, FOLLOW also terminates with an increasing probability when it seeks to reach σ_{i+1} but fails to do so repeatedly (Alg. 3:FOLLOW:2).

After invoking FOLLOW, the equivalence classes are updated (Alg. 3:9–15). For each new vertex v_{new} added by FOLLOW, a search is performed to determine whether v_{new} belongs to an existing equivalence class in Γ. If not, MCTS_NRPA is invoked to compute a tour σ_{new} over the discrete problem $\mathcal{A}(v_{\text{new}})$. If MCTS_NRPA is successful, a new equivalence class $\Gamma_{\sigma_{\text{new}}}$ is added to Γ. This process of selecting an equivalence class, expanding the motion tree to follow the tour associated with the selected equivalence class, and updating the equivalence classes continues until a solution is found or a runtime limit is reached.

VI. EXPERIMENTS AND RESULTS

Experiments are performed in complex environments, as shown in Figs. 1 and 3, where the robot has to wiggle its way through numerous obstacles in order to reach the goals. As the initial energy is often not sufficient to visit each goal, the robot also has to decide when and where to recharge. Experiments are conducted using robot models (car, snake) with nonlinear dynamics, as described in Section III.

1) Problem Instances: A problem instance is obtained by randomly placing the goals and recharging stations in the obstacle-free areas of the environment. A set of 30 problem

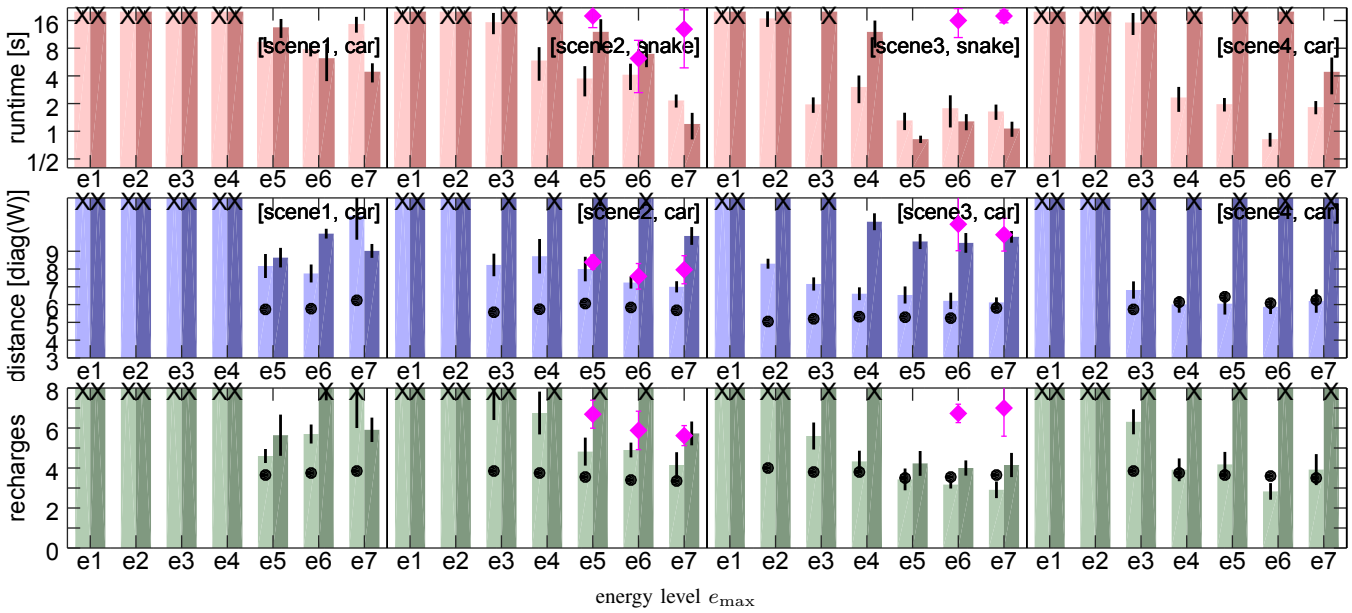


Fig. 5. Results when varying the energy level, where $e_1 = 150$, $e_2 = 200$, $e_3 = 250$, $e_4 = 300$, $e_5 = 400$, $e_6 = 500$, and $e_7 = 600$. Note that an energy level of $80\sqrt{2}$ is required to travel along the diagonal of the bounding box of $\mathcal{W}$.

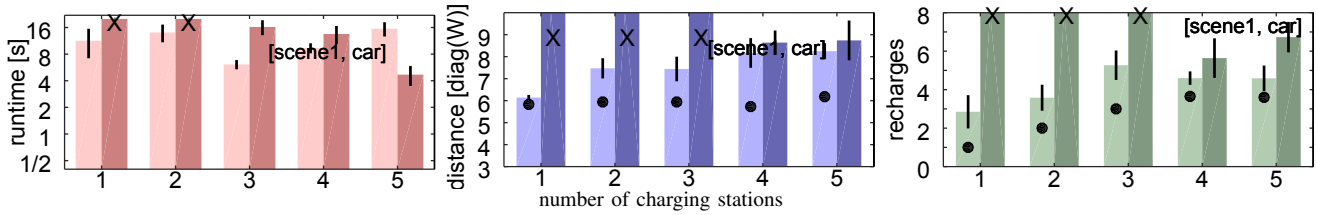


Fig. 6. Results when varying the number of charging stations. The energy is set to $e_{\max} = 300$ and the number of goals is set to 20. Note that SEQUENTIALRRT was not able to solve any of these problem sets.

instances, denoted by $\mathcal{I}_{\langle \text{scene}, n, m \rangle}$, is generated for each combination of a scene, number of goals n, and number of recharging stations m. The planner is then run on each problem instance. Results report on the runtime, travel distance, and number of recharges. The mean is computed after removing the results below the first or above the third quartile to avoid the influence of outliers. The runtime measures everything from reading the input until finding a solution or reaching the runtime limit (20s per run). Experiments were conducted on an Intel© Core i7, 2.6Ghz machine.

2) Discrete Planners: Our approach is general and can be used with any discrete planner. As such, experiments are also conducted using a discrete planner based on the adaptive-threshold approach [16]. Our overall method (Alg. 3) is referred to as $P_{\text{MCTS\_NRPA}}$ when using MCTS_NRPA as its discrete planner and as P_{AT} when using the adaptive threshold.

3) Comparisons: As discussed in Section I, there are no other planners designed to solve multi-goal motion-planning problems with recharging stations that also take the robot dynamics into account. As such, the experimental evaluation focuses on showing the performance of our approach as we vary the number of goals, number of recharging stations, and the energy level. However, to provide a baseline comparison, we also conducted experiments with a modified version of RRT [27], [28], which we refer to as SEQUENTIALRRT. Essentially, MCTS_NRPA is first run to obtain a tour. RRT is then called consecutively to reach each region in succession.

4) Varying the Number of Goals: Fig. 4 summarizes the results when increasing the number of goals. In these experiments, the number of charging stations is kept at 4 and the available energy is set to a small value, which is not sufficient to visit all the goals. So, the robot has to decide when and where to recharge. The results show that SEQUENTIALRRT has difficulty finding solutions. This is expected since it lacks the interaction between the discrete planner and the motion-tree expansion, making it difficult to find alternative routes when expansions fail due to constraints imposed by the obstacles and the robot dynamics. In contrast, the interaction is key to our approach. The results show that our approach scales well. Even when considering over 20 goals, the approach is still able to find solutions quickly. Moreover, the solution trajectories tend to be short with only a small number of recharges. As expected, $P_{\text{MCTS\_NRPA}}$ tends to outperform P_{AT} since MCTS_NRPA generally provides better tours than the adaptive-threshold approach.

5) Varying the Energy Level: Fig. 5 summarizes the results when varying the energy level. The energy level is continually reduced until the problem becomes unsolvable.

The results show that our approach performs well, quickly finding solutions while reducing the distance traveled and the number of recharges. Note that $P_{\text{MCTS\_NRPA}}$ outperforms P_{AT}, while SEQUENTIALRRT has difficulty finding solutions.

6) Varying the Number of Recharging Stations: Fig. 6 summarizes the results when varying the number of recharging stations. Again SEQUENTIALRRT was not able to solve these problem sets within the runtime limit. In contrast, our approach performs well. Note that P_{AT} fails to solve the cases with 1 and 2 charging stations since the adaptive-threshold approach is a heuristic method that could fail to find tours in some cases. In contrast, MCTS_NRPA provides a more general approach that is able to find tours even for these challenging scenarios where the energy resources are scarce.

VII. DISCUSSION

This paper developed an effective approach for the multi-goal motion-planning problem with dynamics and recharging stations. The crux of the approach was coupling sampling-based motion planning with a discrete planner to determine when and where the robot should recharge. Experiments showed the effectiveness of the approach in generating collision-free and dynamically-feasible trajectories that enable the robot to reach all the goals while reducing the overall distance traveled and the number of visits to recharging stations. This work opens up several avenues for future research. One direction is to consider time windows of when all or some of the goals have to be reached. Such time windows are useful to represent constraints arising in logistics. Another direction is to consider moving obstacles, which would require replanning.

REFERENCES

[1] N. Correll, K. E. Bekris, D. Berenson, O. Brock, A. Causo, K. Hauser, K. Okada, A. Rodriguez, J. M. Romano, and P. R. Wurman, "Analysis and observations from the first amazon picking challenge," *IEEE Transactions on Automation Science and Engineering*, vol. 15, no. 1, pp. 172–188, 2018.

[2] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[3] S.-Y. Chien, H. Wang, and M. Lewis, "Human vs. algorithmic path planning for search and rescue by robot teams," in *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 54, no. 4. Sage Publications Sage CA: Los Angeles, CA, 2010, pp. 379–383.

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

[5] H. M. Choset, S. Hutchinson, K. M. Lynch, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[6] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[7] S. Edelkamp, E. Plaku, and Y. Warsame, "Monte-carlo search for prize-collecting robot motion planning with time windows, capacities, pickups, and deliveries," in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer, 2019, pp. 154–167.

[8] M. Silverman, B. Jung, D. Nies, and G. Sukhatme, "Staying alive longer: Autonomous robot recharging put to the test," *Center for Robotics and Embedded Systems (CRES) Technical Report CRES*, vol. 3, p. 015, 2003.

[9] B. Kannan, V. Marmol, J. Bourne, and M. B. Dias, "The autonomous recharging problem: Formulation and a market-based solution," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 3503–3510.

[10] D. Austin, L. Fletcher, and A. Zelinsky, "Mobile robotics in the long term-exploring the fourth dimension," in *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180)*, vol. 2. IEEE, 2001, pp. 613–618.

[11] M. Rappaport and C. Bettstetter, "Coordinated recharging of mobile robots during exploration," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6809–6816.

[12] G. Sharma, A. Dutta, and J.-H. Kim, "Optimal online coverage path planning with energy constraints," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2019, pp. 1189–1197.

[13] R. Huisman, "Scheduling the refuelling activities of multiple heterogeneous autonomous mobile robots," 2014.

[14] V. Marmol, B. Kannan, and M. B. Dias, "Market-based coordination of recharging robots," 2012.

[15] M. C. Silverman, D. Nies, B. Jung, and G. S. Sukhatme, "Staying alive: A docking station for autonomous robot recharging," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 1. IEEE, 2002, pp. 1050–1055.

[16] J. Wawerla and R. T. Vaughan, "Near-optimal mobile robot recharging with the rate-maximizing forager," in *European Conference on Artificial Life*. Springer, 2007, pp. 776–785.

[17] M. Rappaport, "Energy-aware mobile robot exploration with adaptive decision thresholds," in *Proceedings of ISR 2016: 47st International Symposium on Robotics*. VDE, 2016, pp. 1–8.

[18] S. Mishra, S. Rodriguez, M. Morales, and N. M. Amato, "Battery-constrained coverage," in *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE, 2016, pp. 695–700.

[19] F. Liu, S. Liang, and X. Xian, "Optimal robot path planning for multiple goals visiting based on tailored genetic algorithm," *International Journal of Computational Intelligence Systems*, vol. 7, no. 6, pp. 1109–1122, 2014.

[20] T. Kundu and I. Saha, "Energy-aware temporal logic motion planning for mobile robots," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8599–8605.

[21] C. D. Rosin, "Nested rollout policy adaptation for monte carlo tree search," in *Ijcai*, 2011, pp. 649–654.

[22] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*. Springer, 2006, pp. 282–293.

[23] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[24] L. Kavraki, P. Svestka, and M. H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*. Unknown Publisher, 1994, vol. 1994.

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

[26] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," Department of Computer Science, University of N. Carolina, Chapel Hill, TR99-18, 1999. [Online]. Available: http://gamma.cs.unc.edu/SSV/

[27] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[28] S. M. LaValle, "Motion planning: The essentials," *IEEE Robotics & Automation Magazine*, vol. 18, no. 1, pp. 79–89, 2011.