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.

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 $\langle M, Q, S, U, f \rangle$ in terms of its geometric shape $M$, configuration space $Q$, state space $S$, control space $U$, and motion equations $f$. A configuration $q \in Q$ defines the position and orientation. A state $s \in S$ augments the configuration by including other variables, such as velocity and steering angle, that are necessary to define the robot motion. The notation $CFG(s)$ denotes the configuration component of a state $s \in S$. A control $u \in U$ 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

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

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

$$\dot{x} = v \cos(\theta) \cos(\psi), \quad \dot{y} = v \sin(\theta) \cos(\psi),$$
$$\dot{\theta} = v \sin(\psi), \quad \dot{\psi} = a, \quad \dot{\psi} = \omega,$$

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

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

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

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

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

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

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

**Multi-Goal Motion Planning with Recharging Stations:** The world $W$ contains goal regions $G = \{G_1, \ldots, G_n\}$, recharging regions $C = \{C_1, \ldots, C_m\}$, and obstacles. When the robot visits a recharging region, its energy level $e$ is set back to max capacity, i.e., $e = e_{\text{max}}$. Starting from an initial state $s_{\text{init}} \in S$ with an energy level $e_{\text{init}}$, the objective is to compute a collision-free and dynamically-feasible trajectory $\zeta$ 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.

## 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 \( A = (V = V_{goals} \cup V_{charge} \cup \{v_{curr}\}, D, E, e_{\max}, e_{\curr}) \), where

- \( V_{goals} \) is the set of goal vertices;
- \( V_{charge} \) is the set of recharging vertices;
- \( v_{curr} \) is the current robot location;
- \( D = \{d(v',v'') : (v',v'') \in V \times V\} \) is a table, where \( d(v',v'') \) is the distance from \( v' \) to \( v'' \);
- \( E = \{e(v',v'') : (v',v'') \in V \times V\} \) 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 \( 0 < e_{\curr} \leq e_{\max} \).

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 \( A \) is a tour \( \tau = (\tau_1, \ldots, \tau_k) \) that starts at \( v_{curr} \), visits every vertex in \( V_{goals} \), and ensures that there is sufficient energy to move from each \( \tau_i \) to \( \tau_{i+1} \), i.e., remaining energy at \( \tau_i \) is at least \( e_{\max}(\tau_{i-1}) \). When at \( \tau_{i+1} \), the robot’s energy is set to \( e_{\max} \) if \( \tau_{i+1} \in V_{charge} \). Otherwise, \( e_{\max}(\tau_{i-1}) \) 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 \( \sum_{i=1}^{k} d(\tau_{i}, \tau_{i+1}) \).

Note that our overall approach will create numerous problem abstractions, each time modifying \( V_{goals}, D, E, v_{curr} \), and \( e_{\curr} \) depending on the progress that it makes. While initially the abstraction will include all the goal regions \( G_1, \ldots, G_n \), 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 NRPA (Alg. 1b8-9). If \( v_{curr} \in V_{goals} \), 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 \( \tau \) and its cost, computed as \( \lambda \cdot \text{penalty} + d \), where \( \lambda \) is a user-defined constant, \( \text{penalty} \) 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.

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:** $W$: world; $O$: obstacles; $G$: goal regions; $C$: recharging regions; $M$: 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 $R = (V_R, E_R)$

1. $V_R ← E_R ← ∅$, attempts ← 0
2. for $Z ∈ G * C$ do AddCFG($Z, O, M, V_R, Δ, V_{add}$)
3. repeat
4. for $k = 1$ to $n_{add}$ do AddCFG($W, O, M, V_R, Δ, V_{add}$)
5. for $q ∈ V_{add}$ do AddEdges($q, O, M, V_R, E_R, ∆$, attempts)
6. $V_{add} ← ∅$
7. until SAMECOMPONENT($Δ, \{q: Z ∈ G * C\}$)
8. return $R = (V_R, E_R)$

(a) AddCFG($Z, O, M, V_R, Δ, V_{add}$)

1. repeat $q ←$ (RandomPosition($Z$), RandomOrientation())
2. until ¬MeshCollision(TransformMesh($M, q, O$))
3. $V_R ← V_R ∪ \{q\}$; $V_{add} ← V_{add} U \{q\}$; $Δ[q] ←$ MakeSet()

(b) AddEdges($q, O, M, V_R, E_R, Δ$, attempts)

1. neighbors ← NearestNeighbors($q, V_R, \log _2 |V_R|)$
2. for $q_{neigh} ∈$ neighbors ∧ ($q, q_{neigh}$) $\notin$ attempts do
3. if ¬IsPathNCollision($q, q_{neigh}, O, M$) then
4. $E_R ← E_R ∪ \{(q, q_{neigh})\}$
5. JOINSETS($Δ[q], Δ[q_{neigh}]$)
6. attempts $←$ attempts $∪ \{(q, q_{neigh}), (q_{neigh}, q)\}$

(c) SAMECOMPONENT($Δ, \{q_1, q_2, ..., q_n\}$)

1. for $i = 1$ to $n$ do
2. for $j = i + 1$ to $n$ do
3. if FINDSET($Δ[q_i]$) $\neq$ FINDSET($Δ[q_j]$) then
4. return false
5. return true

(d) IsPathNCollision($q_1, q_2, O, M$)

1. $M_{path} ← ∅$
2. $nrSteps ←$ Distance($q_1, q_2/d_{min}$)
3. for $k = 1$ to $nrSteps − 1$ do
4. $q ← (1 − k/nrSteps)q_1 + (k/nrSteps)q_2$
5. $M_{path} ←$ AddMesh($M_{path}$, TransformMesh($M, q$))
6. return MeshCollision($M_{path}$, $O$)

Probabilistic RoadMap (PRM) method [24], the discrete abstraction is obtained as a roadmap $R = (V_R, E_R)$ over the configuration space $Q$. 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 $q_G$, and $q_C$, for each goal $G_i ∈ G$ and charging station $C_j ∈ C$ (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 $R$ to several of its nearest neighbors (Alg. 2:5), where the distance metric $ρ: Q × Q → R^0$ is $ρ(q', q'') = || $ POSITION($q'$) $−$ POSITION($q''$)||2. The number of neighbors is set to $\log _2 |V_R|$ 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 $E_R$. 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 $R$ 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 $q_G$ and recharging $q_C$ configuration. Dijkstra’s algorithm is invoked $|G| + |C|$ times, each time using $q_G$ (or $q_C$) as the source. The notation $SP(R, q', q'')$ is used to denote the shortest path in $R$ from $q'$ to $q''$, while $SPD(R, q', q'')$ 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 $T$ is rooted at the initial state $s_{init}$ and is expanded in the state space $S$. Each vertex $v ∈ T$ has the fields 

\{s, e, goals, nearest, parent, u\}, which denote the following:

(i) $v.s$: collision-free state in $S$; (ii) $v.e$: remaining energy; (iii) $v.goals$: goals in $G$ that have not been reached by $ζ_T(v)$, where $ζ_T(v)$ is the trajectory from the root of $T$ to $v$; (iv) $v.nearest$: nearest configuration in the roadmap $R$, i.e., $v.nearest = \arg \min _{q \in V_R} ρ(CFG(v.s), q)$; (v) $v.parent$: pointer to the parent node in $T$; and (vi) $v.u$: control in $U$. By construction, $v.s ←$ SIMULATE($v.parent.s, u, f, dt$) (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 $T$ such that $v.goals = ∅$, i.e., $ζ_T(v)$ has reached all the goals in $G$.

We leverage the discrete abstraction to guide the motion-tree expansion from $v$. Essentially, each vertex $v ∈ T$ gives rise to a discrete problem to find a short tour $τ$ that starts at the nearest roadmap configuration, $v.nearest$, 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 $v.nearest$, which, according to our model, is proportional to the distance traveled. Formally, the discrete problem associated with $v ∈ T$ is defined as $A(v) =$ $\langle V = V_{goals} \cup V_{charge} \cup \{v_{curr}\}, D, E, e_{max}, e_{curr} \rangle$, where

- $V_{goals} =$ \{ $q_G : G_i ∈ v.goals$\},
- $V_{charge} =$ \{ $q_C : C_j ∈ C$\},
- $v_{curr} =$ $v.nearest$,
- $D =$ \{ $d(q', q'') : q', q'' ∈ V$\}, $d(q', q'') =$ $SPD(R, q', q'')$,
- $E =$ \{ $e(q', q'') : q', q'' ∈ V$\}, $e(q', q'') ∝ d(q', q'')$ and
- $e_{curr} =$ $v.e = -$ ENERGY($ρ(CFG(v.s), v.nearest)$).

After computing $τ$, attempts are then made to expand $T$ from $v$ along $τ$. Since $T$ typically has tens of thousands of vertices, it is infeasible to compute tours for each vertex $v ∈ T$. For this reason, we group together vertices in $T$ 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: $W$: world; $O$: obstacles; $G = \{G_1, \ldots, G_n\}$: goals; $C = \{C_1, \ldots, C_m\}$: recharging stations; $\langle M, Q, S, U, f \rangle$: robot model; $s_{\text{init}}$: initial state; $dt$: time step; $t_{\text{max}}$: runtime limit

Output: collision-free and dynamically-feasible trajectory that reaches each goal or null if no solution is found

1: $R \leftarrow \text{CONSTRUCTROADMAP}(W, O, G, C, M)$
2: $\Xi \leftarrow \text{SHORTESTPATHS}(R, G, C)$
3: $(T, \Gamma) \leftarrow \text{INITIALIZE}(s_{\text{init}})$
4: while $\text{TIME}(\Gamma) < t_{\text{max}}$ do
5: $\Gamma_{\text{e}} \leftarrow \text{SELECTEQUIVALENTCLASS}(\Gamma)$
6: $\sigma = (g_1, \ldots, g_m) \leftarrow \text{TOURTOPATHS}(\Xi, \Gamma)$
7: $v \leftarrow \text{SELECTVERTEXFROMEQUIVALENTCLASS}(\Gamma_{\text{e}})$
8: $\text{FOLLOW}(\gamma, \sigma)$
9: for each new vertex $v_{\text{new}}$ added to $T$ do
10: if $(v_{\text{new}} \not\in \text{FIND}(\Gamma, v_{\text{new}})) = \text{null}$ then
11: $\tau_{\text{new}} \leftarrow \text{MCTS_NRPA}(\text{DISCRETEPROBLEM}((R, \Xi, G, C, v_{\text{new}})))$
12: if $\tau_{\text{new}} = \text{null}$ then continue
13: $\Gamma_{\text{e}} \leftarrow \Gamma_{\text{e}} \cup \{\tau_{\text{new}}\}$
14: $\tau_{\text{new}} \leftarrow \Gamma_{\text{e}} \cup \{v_{\text{new}}\}$
15: if $v_{\text{new}}$.goals = 0 then return $\zeta_T(v_{\text{new}})$
16: return null
(a) FOLLOW($v_{\text{from}}$, $\sigma$)
1: $X \leftarrow \{v_{\text{from}}\}; i \leftarrow 1; \text{count} \leftarrow 0$
2: while $i \leq |\sigma| \land \text{RAND}(0,1) \leq 1/\text{count}$ do
3: $\text{count} \leftarrow \text{count} + 1$
4: $\rho \leftarrow \text{SAMPLETARGET}($ $\sigma_i$ $)$
5: $v \leftarrow \text{SELECTVERTEX}($ $X, \rho$ $)$
6: for several steps do
7: $u \leftarrow \text{CONTROLLER}(v, s, p, \tau)$
8: $s_{\text{new}} \leftarrow \text{SIMULATE}(v, u, f, dt)$
9: $e \leftarrow \text{ENERGY}(\rho(\text{CFG}(v, s, \text{CFG}(s_{\text{new}}))))$
10: if $e < 0 \lor \text{COLLISION}(s_{\text{new}}) \lor \neg \text{NEAR}(s_{\text{new}}, \tau)$ then break
11: $\tau_{\text{new}}(s, \text{parent, u}) \leftarrow \text{NEWVERTEX}(s_{\text{new}}, v, u)$
12: if $\text{REACHED}(s_{\text{new}}, \Gamma)$ then $v_{\text{new}}$.$e \leftarrow e_{\text{max}}$ else $v_{\text{new}}$.e $\leftarrow e$
13: $v_{\text{new}}$.$\text{qnearest} \leftarrow \text{argmin}_{v \in V_G} \rho(\text{CFG}(s_{\text{new}}), q)$
14: $v_{\text{new}}$.goals $\leftarrow v$.goals $\cup \{\text{REACHED}(s_{\text{new}}, \Gamma)\}$
15: if $\text{REACHED}(s_{\text{new}}, \sigma_i)$ then
16: $X \leftarrow \{v_{\text{from}}\}; i \leftarrow i + 1; \text{count} \leftarrow 0$
17: else
18: $X \leftarrow X \cup \{v_{\text{new}}\}$
19: $v \leftarrow v_{\text{new}}$

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

C. Overall Search

The 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 $s_{\text{init}}$, 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 $\Gamma_{\text{e}}$ from $\Gamma$ and then expand $\mathcal{T}$ to follow the tour $\tau$. The equivalence classes are updated after each new vertex is added to $T$. These methods are invoked repeatedly until a solution is found or a runtime limit is reached.

1) Selecting an Equivalence Class: A weight $w(\Gamma_{\text{e}})$ is defined for each equivalence class $\Gamma_{\text{e}}$ as

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

where $0 < \alpha < 1$. $\text{NRSEL}(\Gamma_{\text{e}})$ denotes the number of times $\Gamma_{\text{e}}$ has been previously selected, and $\text{COST}(\Gamma_{\text{e}})$ denotes the cost (as computed by MCTS_NRPA) of the tour $\tau$. The equivalence classes in $\Gamma_{\text{e}}$, 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 $\alpha^{\text{NRSEL}(\Gamma_{\text{e}})}$, $0 < \alpha < 1$, 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 $\Gamma_{\text{e}}$, the objective becomes to expand $\mathcal{T}$ to follow the tour $\tau$. The tour $\tau$ is first converted into one roadmap path $\sigma$ by concatenating the shortest paths that connect the vertices in $\tau$ in succession (Alg. 3:6). The vertex $v_{\text{from}} \in \Gamma_{\text{e}}$ from which to start the expansion of $\mathcal{T}$ is selected from $\Gamma_{\text{e}}$ with probability $v_{\text{from}}.e / \sum v.e \in \Gamma_{\text{e}}.$ $v$.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 $v_{\text{from}}$, FOLLOW($v_{\text{from}}$, $\sigma$) seeks to reach $\sigma_1, \ldots, \sigma_m$ in succession, where $m = |\sigma|$. FOLLOW maintains a set $X$ which contains the candidate vertices which can be expanded to reach $\sigma_i$. Initially, $X$ contains $v_{\text{from}}$, and the first objective is to reach $\sigma_1$. When a vertex $v_{\text{new}}$ reaches $\sigma_1$, then $X$ is reset to contain only $v_{\text{new}}$ and the objective becomes to reach $\sigma_2$, and so on. More specifically, suppose the objective is to reach $\sigma_i$. A target point $p$ is sampled within a distance $d_{\text{near}}$ from $\sigma_i$ (Alg. 3:8) and the nearest vertex $v \in X$ to $p$ is selected (Alg. 3:9). A PID controller is then used to generate a trajectory that steers the vehicle from $v$ to the target point $p$ (Alg. 3:8). The expansion from $v$ stops if the new state $s_{\text{new}}$, obtained after each simulation step, is found to be in collision or its energy level drops below 0 (Alg. 3:10). As the objective is to closely follow $\sigma$, the expansion from $v$ also stops if $s_{\text{new}}$...
is more than a distance $d_{\text{follow}}$ away from $\sigma$. If the expansion is successful, a new vertex $v_{\text{new}}$ is created with $s_{\text{new}}$, $v$, and $u$ as its state, parent, and input control (Alg. 3:FOLLOW:11). If $s_{\text{new}}$ reaches a recharging station, then its energy is set to $e_{\text{max}}$; otherwise, the energy is reduced by the energy consumed to move from $v.s$ to $s_{\text{new}}$ (Alg. 3:FOLLOW:12). If $s_{\text{new}}$ reaches a goal $G_i \in v.goals$, then $G_i$ is not included in $v_{\text{new}}.goals$. If $s_{\text{new}}$ reaches $\sigma_{i+1}$, then $X$ is reset to contain only $v_{\text{new}}$ as the new objective becomes to reach $\sigma_{i+1}$; otherwise, $v_{\text{new}}$ is added to $X$ (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 $\sigma_{i+1}$ 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 $v_{\text{new}}$ added by FOLLOW, a search is performed to determine whether $v_{\text{new}}$ belongs to an existing equivalence class in $\Gamma$. If not, MCTS_NRPA is invoked to compute a tour $\sigma_{\text{new}}$ over the discrete problem $A(v_{\text{new}})$. If MCTS_NRPA is successful, a new equivalence class $\Gamma_{\text{new}}$ is added to $\Gamma$. 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
instances, denoted by $I_{(\text{scene}, n, m)}$, 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 $P_{\text{AT}}$ 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 $\text{SEQUENTIALRRT}$. 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 $\text{SEQUENTIALRRT}$ 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 $P_{\text{AT}}$ 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 $P_{\text{AT}}$, while $\text{SEQUENTIAL.RRT}$ has difficulty finding solutions.

6) Varying the Number of Recharging Stations: Fig. 6 summarizes the results when varying the number of recharging stations. Again $\text{SEQUENTIAL.RRT}$ was not able to solve these problem sets within the runtime limit. In contrast, our approach performs well. Note that $P_{\text{AT}}$ 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, $\text{MCTS.NRPA}$ 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


