2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

Multi-Robot Multi-Goal Mission Planning in Terrains of Varying Energy Consumption

Jáchym Herynek Stefan Edelkamp

***Abstract*— This paper considers planning missions for a fleet of robots with limited energy. Each robot has size, heading, and velocity and its motion is described by non-linear differential equations. The dynamics of movements, existing obstacles, multiple robots, and waypoints are additional challenges, as the combined task and motion planning procedure prevents collisions. On their long-term missions, robots have to visit several waypoints in a cost-minimizing manner to satisfy the overall mission task. The robots consume energy and have to be recharged. The framework guides expanding a motion tree via a state projection to a discrete problem, whose solutions serve as search heuristics. Our experiments highlight that despite all these challenges, even sizable problem tasks can be solved even for complex environments.**

***Index Terms*— robotics, mission planning, motion planning, heuristic search**

I. Introduction

Mission planning is among the most essential topics in modern robotics research. Planning and following a feasible trajectory in a complex environment with a non-holonomic robot is both challenging and crucial for the real-world deployment of robotic systems.

In this paper, we propose a generic framework for multi-robot mission planning. Our framework considers and optimizes energy consumption in terms of the overall cost of the path and in terms of maintaining sufficient energy and recharging at predefined stations. Furthermore, our solver assumes state-dependent energy consumption, allowing us to model non-trivial terrain properties and varying energy consumption based on the state of the vehicle. An example of such a multi-robot multi-goal energy-aware routing problem is presented in Figure 1.

The contribution of the presented solver lies in the complexity of the problem it handles. Each of the individual sub-problems is challenging in its own right, and our solver tackles those problems simultaneously. In the continuous domain, the problem of finding a dynamically feasible path between two configurations for a non-holonomic vehicle is, in the general case, undecidable [1]. Regarding the discrete aspects of the problem, finding an optimal sequence of visits is the well-studied traveling salesman problem, known to be NP-complete [2]. Considering multiple agents also introduces robot-robot collision and complicates the planning further. The proposed solver can find feasible paths for a robot team under the energy depletion constraint in an obstacle-dense environment while avoiding collisions with other robots.

The presented work has been supported by the Czech Science Foundation (GAČR) under the research project number 22-30043S.

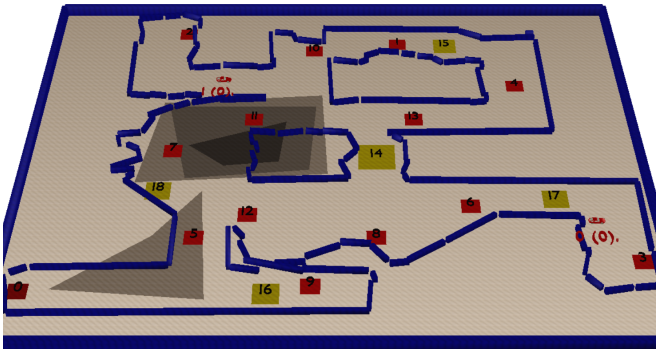


Figure 1: Example of a multi-robot multi-goal mission planning problem with energy (Scene 1). Goals shown in red, recharging stations in yellow, higher energy consumption areas are shown in darker colors.

The paper is structured as follows: Current approaches are described in Section II. Next, Section III formalizes the problem. The solver itself is described in detail in Section IV, and in Section V, the conducted experiments are analyzed.

II. Related Work

We are considering the physical multi-vehicle route planning problem with energy. In this chapter, the prior works on topics closely related to our problem are discussed, including motion planning, resource limitation, combinatorial search, and multi-agent planning.

The Vehicle Routing Problem (VRP), first addressed in [3], is the problem of finding an optimal route connecting a given set of points for a fleet of vehicles traveling from and returning to a common depot. Because it is a generalization of the traveling salesman problem, it is NP-complete. Since its formulation, the problem has seen much attention and many variants that aim to represent real-world scenarios more closely, such as pickup and delivery [4], VRP with time windows [5], and many more [6]. Among other approaches, Nested Monte-Carlo with Policy Adaptation [7] was recently shown to provide high-quality results for the VRP problem [8].

On the discrete side, in addition to the vehicle routing, our solver also requires searching a graph for the shortest paths between points simultaneously for each of the multiple vehicles. For a single vehicle, this problem can be solved in polynomial time using Dijkstra's algorithm [9] or its more sophisticated guided variants like A* [10]. These algorithms provide a solution for a single source shortest path query if we assume static obstacles. For scenarios with multiple

moving agents, Multi-Agent Path Finding (MAPF) [11] is a highly researched topic.

Motion planning algorithms in continuous space are typically based on sampling algorithms like the Rapidly-exploring Random Tree algorithm (RRT) [12] and its asymptotically optimal variants [13] or the Probabilistic Roadmap algorithm (PRM) [14]. These algorithms sample random configurations in the space and connect them to the previously sampled ones to build a graph in which the search itself is done. RRT even avoids the boundary value problem, which is not, in general, solvable analytically.

There are many variants of these algorithms. In [15], the authors propose a variant of the RRT algorithm called GUST. It is guided by an approximation constructed over a simplification of the configuration space, which allows the planner to reach the goal much faster. This approach has been used to solve problems where continuous motion planning is combined with some other difficulty, such as in [16], where the motion planning problem is considered with multiple goals and energy optimization.

There are two straightforward ways of extending the sampling-based algorithms to multi-robot scenarios: centralized, which plans in the composite configuration space of all the robots, and decoupled, which plans for each of the robots individually and sees the other robots as obstacles. The centralized approach is typically computationally intractable, as the problem gets exponentially harder in the number of robots. On the other hand, the decoupled approach has poor results when the task requires high degree of coordination between the robots. In [17], a centralized approach is proposed, which attempts to mitigate the high dimensionality by using an implicit representation of the configuration space rather than an explicit one. In [18], a solver that provides solutions for a multi-agent system of robots under dynamic constraints is proposed. It is based on [15] and utilizes MAPF solutions to guide the sampling search in the composite space. The authors of [19] considered multi-robot physical motion planning problems with capacity and time window constraints but without inter-vehicle collisions.

Significant progress in the robotic and motion planning areas has also been achieved thanks to fleet robot mission competitions such as the DARPA subterranean challenge [20]. However, such competitions usually consider the terrain to be unknown prior to the search, so map generation and limited communication are more crucial research questions than routing and motion planning.

III. Problem Formulation

Our work consists of finding an optimal trajectory for multiple robots visiting a number of predefined goals under obstacle and energy constraints. We assume that the environment is known to the planner and that the planning is centralized in a combined robot configuration space.

A. Robot Definition

We consider a set of n agents $\mathcal{R} = \{r_j \mid j \in 1 \dots n\}$. Each individual agent in our scenario represents a robot system, defined by $r_j = \langle \mathcal{P}_j, \mathcal{C}_j, \mathcal{U}_j, f_j, \rho_j \rangle$, where $\mathcal{P}$ denotes the geometrical shape of the robot, $\mathcal{C}$ its configuration space, $\mathcal{U}$ its control space, f its motion equations and $\rho : \mathcal{C} \to \mathbb{R}$ the cost function.

In most experiments, a car-like robot has been used, but the approach is not robot-specific. A configuration of such a robot is defined by (x, y, θ, v, ψ), where x, y represents a position in 2D, θ represents the heading, v represents the velocity, and ψ represents the steering angle of the vehicle. The configuration space $\mathcal{C}$ of such a robot is then $\mathcal{C} = \mathbb{R}^3 \times \mathbb{S}^2$.

The motion of the car-like vehicle is governed by the following set of differential equations:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ \psi \\ u_1 \\ u_2 \end{bmatrix}, \tag{1}$$

where $[u_1, u_2] \in \mathcal{U}$ is the control input. For the sake of simplicity, we will further assume that all robots are the same.

B. Task Definition

Obstacles $\mathcal{O} \subset \mathcal{C}$ are a subset of the configuration space $\mathcal{C}$, which the robot cannot traverse. We denote the rest of the configuration space by $\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{O}$. Note that the obstacles might be robot-dependant (a terrain traversable for one type of robot might be impassable for another).

Each **goal** g_i is considered to be a convex and continuous subset of $\mathbb{R}^2$ (or $\mathbb{R}^3$ if flying robots are considered). In a given scenario, there are m goals, all of which have to be visited. The set of all goals considered in a given scenario is denoted by $\mathcal{G} = \{g_i \subset \mathbb{R}^2 \mid i \in 1 \dots m\}$.

A **collision-free path** of the vehicle j is denoted by $\gamma_j : [0, 1] \to \mathcal{C}_{\text{free}}$. We denote a projection from the configuration space of robot r to the Euclidean space $\mathbb{R}^2$ by $\pi_{\mathbb{R}}(-)$. If for vehicle j and goal i at time t it holds that $\pi_{\mathbb{R}}(\gamma_j(t)) \in g_i$, that path is said to be visiting goal i.

The cost of a given path γ_i (in our case, this is the fuel consumption, but any other metric could be used) is then defined by

$$\text{Cost}(\gamma_i) = \int_0^1 \rho_i(\gamma_i(t))\,\text{dt}. \tag{2}$$

Notice also that the fuel consumption is a function of a robot configuration, $\rho_r : \mathcal{C} \to \mathbb{R}$. This allows us to formulate more complex fuel consumption patterns, such as increased fuel consumption when the agent is accelerating or when the agent is located in certain areas.

A path γ_i is considered to be **feasible** if it is collision-free and the energy level of the vehicle never drops below zero.

In addition to the goals, **recharging stations** where the energy may be refueled are also considered. Recharging stations are defined the same as goals: $\Phi = \{\phi_i \subset \mathbb{R}^2 \mid i = 1 \dots l\}$, and visiting a recharging station is defined in the same manner as visiting a goal.

The energy of a robot r at a time point t is denoted as $e_r(t)$. For every robot, it holds that $e_r(0) = e_{\text{start}}$, and if a recharging station is visited at time t, the energy level is reset to its maximum: $e_r(t) = e_{\max}$. If no recharging station is visited during a path segment between t and $t + \Delta t$, it holds that

$$e_r(t + \Delta t) = e_r(t) - \int_t^{t+\Delta t} \rho_r(\gamma_r(t))\,\mathrm{dt}. \tag{3}$$

A set of paths $\Gamma = \{\gamma_r \mid r \in \mathcal{R}\}$ is considered feasible, if each path individual $\gamma_j \in \Gamma$ is feasible and no two paths collide. The overall problem is then to find a feasible set of paths Γ such that the overall cost $\sum_{\gamma_r \in \Gamma} \text{Cost}(\gamma_r)$ is minimized and all goals are visited:

Problem 1 (Multi-Robot Multi-Goal Motion Planning):

$$\begin{aligned} &\Gamma^* = \arg\min_{\Gamma} \sum_{\gamma_r \in \Gamma} \text{Cost}(\gamma_r) \\ &\text{subject to:} \\ &\forall \gamma_r \in \Gamma : \gamma_r \text{ is feasible} \\ &\forall g \in \mathcal{G}, \exists \gamma_r \in \Gamma : \gamma_r \text{ visits } g \end{aligned} \tag{4}$$

IV. Proposed Method

The method proposed in this paper is based on the solver proposed in [18]. The proposed method applies the same high-level idea in terms of composite-state-space sampling-based planning and adapts it for the energy-based cost function in combination with the Monte-Carlo tree search to address a different aspect of the multi-robot motion planning problem. This section discusses the approach in further detail.

Broadly, the idea is constructing a discrete abstraction of the problem, which is then used to guide the sampling-based search in the continuous space to build a motion tree. The guidance of the discrete solver mitigates the inherent disadvantages of the sampling-based approach by focusing the search on a promising area. Thanks to this, it is feasible to compute the motion tree in the composite space of multiple robots.

A. Discrete Solver

The main tasks of the discrete solver are to assign a sequence of goals and recharging stations to each of the robots and to provide an approximate path to follow this sequence. This path does not, however, account for the robot dynamics.

The first step is to construct an approximate probabilistic roadmap of the geometric environment representation. Obstacles are taken into account, but vehicle dynamics are neglected. The edges in the resulting graph are weighted by an approximation of the cost of traversal. Once the roadmap is finished, a distance matrix between the goals is computed.

This matrix is then used whenever it is necessary to compute a new tour, i.e., the sequence of goals and recharging stations each robot should follow. For a tour to be feasible, the energy constraints must be met, and all goals must be visited. This part of the problem is essentially an energy-constrained variant of the Vehicle Routing Problem with energy replenishment, which is addressed by a Monte Carlo with Nested Rollout Policy Adaptation [7] algorithm.

B. Continuous Solver

Algorithm 1: MR-MG-MP

Input: Configuration q_{init}, Map $\mathcal{M}$
Output: Set of paths Γ

```
1  Π ← ConstructPRM(M)
2  T ← InitMotionTree(Π)
3  Insert(q_init, T)
4  while not solved do
5      g ← T.SelectGroup()
6      v ← g.SelectVertex()
7      τ ← g.GetTour()
8      q_target ← SampleTarget(v, τ)
9      ExtendTowards(v, q_target, T)
10 return ExtractPaths(T)
```

Algorithm 2: Insert

Input: Vertex v, Motion Tree $\mathcal{T}$

```
1  v.SetReachedGoals()
2  p ← v.GetParent()
3  τ ← p.GetTour()
4  Π ← T.GetPRM()
5  if not (IsFeasible(τ, Π) ∨ ReachedGoal(v, τ)) then
6      τ ← CreateNewTour(v, Π)
7  if IsFeasible(τ, Π) then
8      v.SetTour(τ)
9      T.AddVertex(v)
```

The continuous solver then queries the discrete solver for a tour and attempts to plan dynamically feasible paths for the robots to follow that tour using a sampling based approach. The algorithm builds a motion tree by iteratively selecting and expanding the nodes already in the tree, as described in Algorithm 1. Each node represents a composite state of all of the robots and holds additional information, such as its parent or the planned high-level tour.

In the proposed approach, the nodes in the tree are joined in groups with similar state of completion. The probabilistic roadmap naturally separates the configuration space into regions, and each robot can be seen as in one of the regions. If two vertices have the robots in the same region and have the same sets of visited goals, they are in the same group.

Initially, the tree contains only a single group and a single node at the starting configuration (Line 3). In every step, a group from the tree is selected, and a vertex from this group is expanded. The target configuration is selected based on the approximate solution provided by the discrete solver (Line 8). During this expansion (Algorithm 3), control input

Algorithm 3: ExtendTowards

Input: Vertex v, Configuration q, Motion Tree $\mathcal{T}$
Parameter: Iteration Cap k, simulation time Δt

```
1 visits ← GetNrExpansions (v.GetGroup ())
2 u ← {u_{r,stop} | r ∈ R}
3 for r ∈ R do
4     u_r ← RandomControl ()
5     if rand (0, 1) < Decay (visits) then
6         u_r ← Steer (v, q)
7 for i = 1 ... k do
8     v_new ← apply (v, u, Δt)
9     if not CollisionFree (v_new) then
10        break
11    Insert (v_new, T)
12    v ← v_new
```

is applied to the configuration for some Δt, steering the expansion towards the target configuration. This produces a new configuration, which is reachable by a known and dynamically feasible path. This new configuration is added to the tree as a member of either the original group or a new one based on its progress.

The whole planning process is done in the composite space of all the agents and thus relies heavily on the guidance from the discrete approximation. The discrete solver guides the expansion of the continuous solver and is invoked every time the current state of the solution changes (by reaching a goal) or whenever the discrepancy between the expected cost and the actual cost makes the existing plan infeasible (Algorithm 2, Line 6).

1) Group and Vertex Selection: The group to be expanded is selected based on a weight function, calculated from the unvisited goals, the distance from the root, and the number of previous expansions:

$$w\left(group\right) = \lambda^{\text{VISITS}} \left(\frac{\alpha}{|\tau_g|} + dist\left(group\right) \beta \right) \qquad (5)$$

where α, β, and λ are parameters, VISITS denotes the number of times the region was already visited, τ_g denotes the unvisited goals, and $dist\left(group\right)$ denotes how far the current group is from the root of the tree. The ensures that the computation does not infinitely expand the same group and λ should be selected so that $0 < \lambda < 1$ (close to 1).

The weight function has to assign greater weight to nodes closer to the solution. However, it can be quite tricky to estimate how close a node is to the solution, due to the unreliable nature of the approximation and the hard constraint on energy consumption. The cost of the current tour provided by the discrete solver would be a natural choice, but this does not work well. Since the discrete solver works on an approximation, it may prove too optimistic, and the energy constraint might cause the original plan to turn out infeasible. In such a case, it is necessary to replan, and the new plan is (most likely) perceived as strictly worse in terms of its cost. Therefore, the solver would continually select the groups before the replanning was triggered. Instead, the weight function is based on the number of remaining unvisited goals.

However, on its own, the α-term would be piecewise-constant. This is also undesirable since each node between two goals would be perceived as equal. The β-term addresses this issue by greedily biasing the selection towards regions further along the path. It should therefore be significantly (orders of magnitude) smaller than the α-term so that it never outweighs it.

The presented approach selects the group at random with probability based on its weight and with bias towards the group with the highest weight. The strength of this bias can be adjusted to achieve faster computation (higher bias, more greedy) or higher quality solution (lower bias, more exploration). The vertex is then selected from this group probabilistically based on the distance from the target.

2) Tree Expansion: This step is described in Algorithm 3. Control input is applied to the selected vertex over a period of time Δt (Line 8), which creates a new vertex and a dynamically feasible connecting path, similarly to other randomized algorithms. However, the selection of control inputs in our approach is not entirely random, as the algorithm balances between a steering approach and a randomized approach.

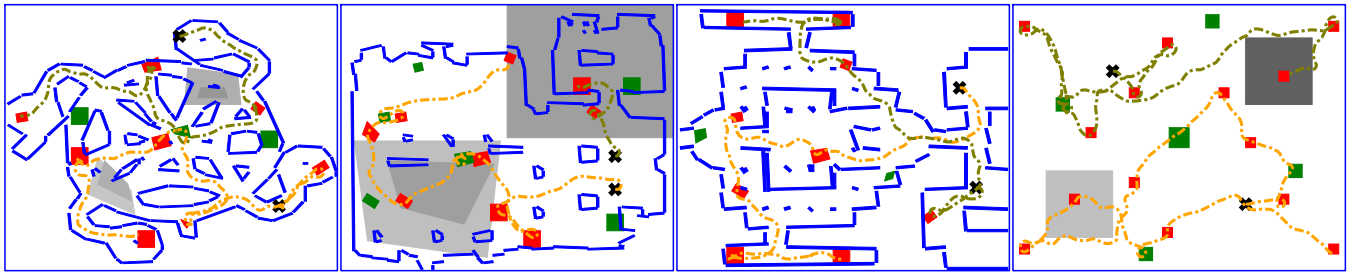
The steering approach attempts to find controls to guide the agent toward its currently selected goal. This usually works quite well, and for the presented types of robots, this approach can quickly reach the goal but can struggle in certain scenarios where a more complicated maneuver is necessary.

On the other hand, the randomized approach produces sub-optimal and somewhat erratic paths. It is, however, capable of finding feasible paths in an obstacle-dense environment or can turn around in a narrow corridor, thus complementing the steering in the areas where it struggles.

In the proposed method, the selection is done at random, with a decaying probability, which initially heavily favors the steering approach but decreases every time the group in question is expanded (Algorithm 3, Line 5). If the controller is able to select the correct solution, it will do so quickly. Otherwise, the repeated calls increase the probability of making random moves, which may improve the position for future steering calls or eventually move the vehicle toward the objective using just random moves.

V. Experiments

We ran our experiments on a single thread of a laptop with a Ryzen 7 7840U processor and 32 GB RAM. The experimental scenarios were selected to showcase the properties of the proposed solver. Obstacles were created based on public path-planning instances in [21], while the goals, terrain areas, and recharging stations were selected manually so that the solution existed and was non-trivial. Some of the selected testing problems are visualized in Figure 2, together with the best solution our solver provided across all the runs during the testing. We ran 50 tests for every scenario, and the results are summarized in this chapter.



(a) Scene 2 (b) Scene 3 (c) Scene 4 (d) Scene 5

Figure 2: Some of the experimental scenarios, based on benchmark maps from [21]. Obstacles are depicted in blue, goals in red, recharging stations in green, difficult terrain in shades of gray. Initial robot positions are marked by a black cross. Lines represent the best solution found by our solver. (Scene 1 is shown in Figure 1.)

Scene	1	2	3	4	5	6*
Goals	14	8	9	9	14	20
Recharging	5	3	5	2	5	4
Terrain	4	4	3	0	2	0
Max Energy	240	200	200	300	240	250
Median Init Time [s]	2.7	1.9	3.1	2.3	3.3	4.4
Median Total Time [s]	20.3	16.3	19.5	22.1	14.1	42.9
Discrete Solver [%]	75	35	59	30	67	77
Timeout Rate [%]	0	2	6	12	0	8

*Scene 6 was used for experiments with varying number of robots. Shown results are with four robots.

TABLE I: Instance data and computational times

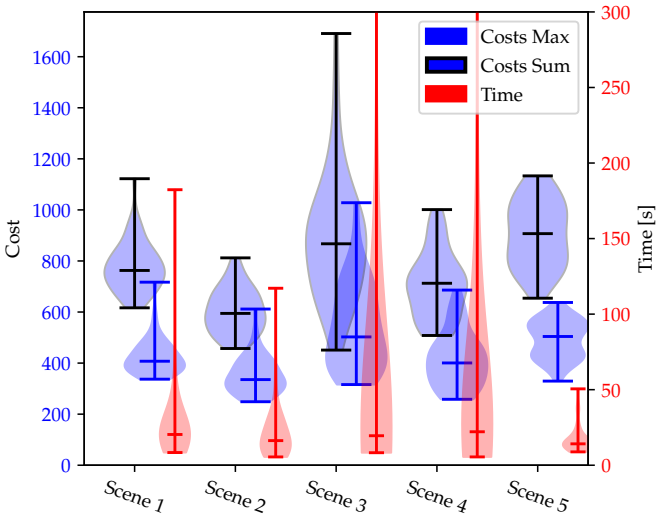


Figure 3: Cost of the path and computational time. Times are in red, total cost in blue-black, greatest single-vehicle cost in blue. Horizontal bar represents the median values. Outlier times (around 800s) are cut from the frame for clarity.

In our experiments, we used an energy function that was based on the distance traveled, increased by the terrain modifier and by the vehicle state (its turning rate). This selection of energy-function is to some extent arbitrary, not meant to be realistic. It serves to show that the approach works even with more complicated energy functions, and that even with the uncertainty in the discrete approximation of the energy, the solver is able to find feasible paths.

There was a hard cap on the number of iterations, which led to a few timeout finishes for some of the scenarios. The timeout rates are shown in the last line of Table I. The reason for the higher timeout rate in Scene 4 are the narrow passages. They are not wide enough for the vehicle to turn around in a single motion, and thus, the steering fails to navigate the area. Additionally, both of the vehicles encounter these areas roughly at the same time, which makes the situation even trickier for the solver to handle.

The computational times and solution lengths are shown in Figure 3 and in Figure 4. For each of the scenarios, both the maximum agent cost and the sum of those costs are depicted. As is seen, the computational times may differ widely depending on the selected problem, and the complexity of the given problem is not necessarily tied to the number of goals (consider Scene 4, where the obstacles pose a much bigger problem and much more time is spent in the continuous part of the computation than in the other scenarios as a result).

For the most part, the computational complexity stems from the discrete solver, taking up to 80% of the computation (see Table I), depending on the problem (harder obstacles require more time on the continuous side). However, re-planning can be triggered by any branch of the motion tree whenever a vehicle reaches a goal (or recharging station) or when its energy drops too low. For harder scenarios (such as scenarios where the energy constraint is more strict), this may happen quite frequently. Hence, we cannot express a direct relation between the number of goals and computational complexity.

It is also possible tweak the parameters of the discrete solver to better suit the complexity of the given problem. In our experiments, we tried multiple parameter settings. By increasing the depth and number of iterations in the discrete solver, we were able to achieve about 10% improvement in quality at the cost of significantly increasing computational times. The selected parameters work well for the presented problems, however, harder problems would likely require more computationally intensive settings. This can be seen in Figure 4, where the computational times are significantly higher for the case of 8 robots.

The solution quality in terms of the cost is clearly suboptimal even for the best results (as seen in Figure 2) since the paths tend to randomly twist and turn. However, the solver

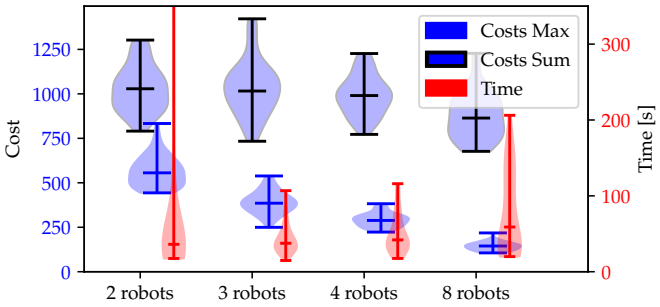


Figure 4: Computational time and solution length with respect to number of robots. Measured in Scene 6. Computational time in red, total cost (black-edged) and greatest single-vehicle cost (blue-edge) in blue. Outlier time (around $550s$) in the 2-robot case cut from the frame for clarity.

produces feasible paths that can navigate an obstacle-dense environment with multiple robots. As the solver is by nature greedy (due to the steering control selection and the group weight equation), this is expected. However, we can observe that the solver avoids higher energy consumption areas, such as in Scenes 3 and 5 (Figure 2); we can see that the solver favors longer routes in order to minimize the time the robots spend in higher energy areas.

The solver can also scale to a larger number of robots, as is shown in Figure 4. Notice that the time complexity does not increase much as the number of robots scales. This is partially caused by the fact that the same scenario with more robots scattered around the map has individually much shorter paths (and is thus much less limited by the fuel constraint). Even so, it clearly shows that our solver can be used for larger numbers of robots as well.

VI. Conclusion

Robot missions are complex endeavors. They may have different objectives or heterogeneous fleets, and plans often include exploration, communication and inspection. In our work, we addressed the physical vehicle problem with non-linear energy.

In robot mission planning, it is important to be able to plan a path in a fully observable environment with obstacles and energy limitations. The recharging aspect of the problem is especially important since current electric vehicles have limited capacities, and many longer missions would require swapping of batteries, maybe even multiple times during a single mission. Additionally, it is important to be able to represent non-linear energy consumption since the traversal of different terrains and slopes may require very different amounts of energy.

We extended the existing approaches to address the multi-goal energy-constrained task-motion planning problem with non-linear energy. Since we assume a black box, the proposed approach can be used with any robot model, as long as it is possible to simulate its behavior with a given control input and we have some notion of steering to guide the solver.

Our solver was able to consistently find feasible paths in obstacle-dense environments and avoid energy depletion and was able to take into account the non-linear energy consumption, both during the pre-computation phase as well as adjust the values when additional information about the energy consumption became available.

Acknowledgment

This research was funded by Czech Science Foundation grant number 22-30043S.

References

[1] M. S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1):67–100, 1995. Hybrid Systems.

[2] M. Jünger, G. Reinelt, and G. Rinaldi. Chapter 4 the traveling salesman problem. In *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 225–330. Elsevier, 1995.

[3] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Manage. Sci.*, 6(1):80–91, oct 1959.

[4] N. Bianchessi and G. Righini. Heuristic algorithms for the vehicle routing problem with simultaneous pick-up and delivery. *Computers & Operations Research*, 34(2):578–594, 2007. Reverse Logistics.

[5] K.C Tan, L.H Lee, Q.L Zhu, and K Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15(3):281–295, 2001.

[6] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016.

[7] C. D. Rosin. Nested rollout policy adaptation for Monte Carlo tree search. In *IJCAI*, volume 2011, pages 649–654, 2011.

[8] T. Cazenave, J.Y. Lucas, H. Kim, and T. Triboulet. Monte Carlo vehicle routing. In *ATT at ECAI 2020*, 2020.

[9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[10] 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*, 4(2):100–107, 1968.

[11] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[12] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 473–479 vol.1, 1999.

[13] Sertac K. and Emilio F. Sampling-based algorithms for optimal motion planning, 2011.

[14] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[15] E. Plaku. Region-guided and sampling-based tree search for motion planning with dynamics. *IEEE Transactions on Robotics*, 31(3):723–735, 2015.

[16] Y. Warsame and S. Edelkamp. Electric vehicle location-routing task-motion planning. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–7, 2023.

[17] R. Shome, K. Solovey, A. Dobson, D. Halperin, and K. E. Bekris. dRRT*: Scalable and informed asymptotically-optimal multi-robot motion planning. *Autonomous Robots*, 44:443 – 467, 2019.

[18] D. Le and E. Plaku. Multi-robot motion planning with dynamics via coordinated sampling-based expansion guided by multi-agent search. *IEEE Robotics and Automation Letters*, 4(2):1868–1875, 2019.

[19] Y. Warsame and S. Edelkamp. Capacitated multi-robot task allocation with time windows using location-routing task-motion planning. In *2023 21st International Conference on Advanced Robotics (ICAR)*, pages 42–48, 2023.

[20] T. H. Chung, V. Orekhov, and A. Maio. Into the robotic depths: Analysis and insights from the darpa subterranean challenge. *Annual Review of Control, Robotics, and Autonomous Systems*, 6(1):477–502, 2023.

[21] N. Sturtevant. Pathfinding benchmarks. https://movingai.com/benchmarks/. Accessed: 2024-02-28.