Charging Station Placement for Indoor Robotic Applications

Tanmoy Kundu[1] and Indranil Saha[2]

Abstract— **For an autonomous mobile robot, when the available power goes below a certain threshold, the robot needs to abort its current task and move towards a charging station to recharge its battery. The efficiency of an autonomous mobile robot depends significantly on the location of the charging stations. In this paper, we address the charging station placement problem for mobile robots in a controlled workspace. We propose two algorithms to place a number of charging stations so that a robot is always capable of reaching one of the charging stations from any obstacle-free location in the workspace without aborting its task too early. We reduce the charging-station placement problem to a series of SMT solving problems and use the off-the-shelf SMT solver Z3 to implement our algorithm. The algorithm produces as output the locations of the minimal number of charging stations in the workspace and the trajectories from all obstacle-free locations to one of the charging stations. Our experimental results show how our algorithm can efficiently find the location of the charging stations and robot trajectories to reach the charging stations. We demonstrate through simulation how the generated trajectories can be effectively used by a robot to reach a charging stations autonomously without getting depleted with power.**

I. Introduction

Autonomous indoor robots are an emerging class of robotic systems that have significant potential to transform our day-to-day life. Indoor robots are widely used in manufacturing or material handling in factories and warehouses [1], [2]. For example, in Amazon Warehouse, robots are used to pick an object from some place and drop it to another place. Robots also find place in our homes to help us in routine work [3]. For example, the robot Roomba from i-Robot Create can autonomously clean a room using vacuum cleaning technology. Recent trend suggests that robots will find applications in commercial spaces such as hotels, hospitals, offices, banks, malls, and museums [4].

Indoor robots are generally battery powered and require to charge their battery at a regular interval. Depending on the location of the charging station, the robots need to plan to recharge their battery at a regular interval so that they become devoid of power to move further. The power efficiency of the robots depends on the location of the power station significantly. In this paper, we address the following question: Given the workspace and the dynamics of the robots, what is the optimal locations for placing the charging stations?

*Tanmoy Kundu is supported by Visvesvaraya Ph.D. Fellowship by the Department of Electronics and Information Technology, Ministry of Communication and Information Technology, Government of India

[1] Tanmoy Kundu is with Department of Computer Science and Engineering, Indian Institute of Technology Kanpur tanmoy@cse.iitk.ac.in

[2] Indranil Saha is with Department of Computer Science and Engineering, Indian Institute of Technology Kanpur isaha@cse.iitk.ac.in

While planning for the charging station placement for the robots, the following questions need to be answered. First, what is the threshold on the amount of power based on which the robot decides when it aborts its current mission and starts moving towards a charging station? Second, How many charging stations should we install in a given workspace? Third, what will be the locations of the charging stations? These issues are not independent, for example, if the power threshold is high then the number of required charging station is less.

In this paper, we formulate two variants of charging station placement problem. In the first problem, we assume that the power threshold for the robot is given, and we aim to find the minimum number of charging stations and their locations. In the second problem, we assume that the number of charging station to be installed is given, and we attempt to find their locations and the value of the minimum power threshold for the robot.

We provide SMT-based algorithms to solve the above two problems. Both the problems can be formulated as a sequence of SMT solving problems. However, naive SMT-based formulation does not scale with the size of the workspace. We then provide an algorithm based on finding *unsatisfiable core* [5] in a set of constraints. In our context, an unsatisfiable core provides a small subset of the obstacle-free locations in the workspace that can not be served by the current power threshold or the current number of charging stations.

We have implemented our algorithm using the SMT solver Z3 [6]. Z3 provides a mechanism to indicate a subset of constraints that can be used to produce an unsatisfiable core. Through a series of experiments we demonstrate that our unsatisfiable core based algorithm outperforms the naive SMT-based algorithm significantly in terms computation time. We also demonstrate through ROS simulation how the trajectories generated by our tool can be used to steer a robot to the nearest charging station when the battery power of the robot reaches a pre-decided threshold.

Related work. The charging station placement problem is a variant of *facility location problem* [7], [8]. The inputs to the facility location problem are generally a set of users and a set of potential sites to establish some service facility. The goal of any algorithm solving the problem is to find a subset of the potential sites of size k such that some objective function, for example, minimizing the maximum distance between a user and its nearest facility, can be minimized. Most of the early work on this problems assumed the users to be static. However, recent work on facility location problem considers the users to be mobile and assume a flow model to characterize the mobility of the users [9], [10], [11].

Such models have been used to solve the refueling station location problem [12], [13], [14] and macro-cell planning for mobile users [15]. While at the higher level, the charging station location problems for mobile robots resembles facility location problem for mobile users, the complex dynamics of the robots entails different solutions to this problem.

The approach presented in this paper can be viewed as joint location planning for the charging stations and motion planning for the robots. Planning is a classical problem in AI and robotics [16], [17], [18] and we borrow ideas from the planning literature. For example, following the AI literature, we approximate the dynamics of the robots using a set of motion primitives [19], [18]. Our algorithmic solution to the problem is based on a reduction to a sequence of SMT solving problems. SMT solvers are recently being popular in solving the task and motion planning problems for robots (e.g. [20], [21], [22], [23], [24], [25], [26]). Composition of motion primitives by means of an SMT solver to generate trajectories satisfying some high level description was introduced in [23]. The capability of an SMT solver to generate an unsatisfiable core has been exploited in [24] to solve the multi-robot motion planning in a scalable manner.

Paper organization. The rest of the paper is organized as follows. In Section II, we introduce the notations used in the paper, and provide formal problem definition with illustrative examples. We describe our algorithms in Section III. In Section IV we describe our experimental setup and provide experimental results to demonstrate the efficacy of the proposed algorithms. Finally, we conclude in Section **??** with some insights on possible future directions.

II. Problem

A. Preliminaries

1) Workspace: In this work, we assume that the robots operate in a 2-D workspace which we represent as a 2-D occupancy grid map. The grid decomposes the workspace into square shaped blocks which are assigned unique identifiers to represent their locations in the workspace. We denote the set of locations in the workspace by W and the set of locations covered by obstacles by O. The set of free locations in the workspace is denoted by F, where $F = W \setminus O$.

2) Robot State: The *state* of a robot σ consists of (1) its position in the space, $\sigma.x$ (which determines a unique block in the occupancy grid) and (2) its velocity configuration, $\sigma.v$, which represents current magnitude and direction of the velocity of the robot. We denote the set of all velocity configurations by V and assume it contains a value v_0 denoting that the robot is stationary.

3) Motion Primitives.: We capture the motion of a robot using a set of of *motion primitives* Γ. We assume that the robot moves in an occupancy grid in discrete steps of τ time units. A motion primitive is a short controllable action that the robot can perform in any time step. A robot can move from its current location to a destination location executing a sequence of motion primitives.

With each motion primitive $\gamma \in \Gamma$, we associate a a *pre-condition* $pre(\gamma)$, which is a formula over the states specifying under which conditions a motion can be executed. We write $post(\sigma, \gamma)$ for the state of a robot after the motion primitive γ is applied to a state σ satisfying $pre(\gamma)$. We use $intermediate(\sigma, \gamma)$ to denote the set of grid blocks through which the robot may traverse when γ is applied at state σ, including the beginning and end blocks. Each motion primitive γ is associated with a energy cost as denoted by $cost(\gamma)$, which represents the amount of energy spent by the robot while executing the motion primitive. To simplify the exposition, we assume that the execution of all the motion primitives for a robot lead to same amount of energy expenditure.

We assume that in Γ there exists a motion primitive that can be applied when the robot is at the velocity configuration v_0 and it keeps the robot in the same state. This special primitive is called the *rest primitive*.

4) Motion Plan and Trajectory: The runtime behavior of the robot r is described by a discrete-time transition system $\mathcal{T}$. Let σ_1 and σ_2 be two states of the robot and γ be the motion primitive applied to the robot in state σ_1. We define a transition $\sigma_1 \xrightarrow{\gamma} \sigma_2$ iff

- $\sigma_1 \models pre(\gamma)$ and $\sigma_2 = post(\sigma_1, \gamma)$.
- the trajectory of the robot between the states σ_1 and σ_2 does not pass through a block occupied by an obstacle, i.e., $intermediate(\sigma_1, \gamma) \ \cap O = \emptyset$.

A *motion plan* for a robot is defined as a sequence of motion primitives to be applied to the robot to move from from a location $l_i \in F$ to another location $l_f \in F$. A motion plan is denoted by $\rho = (\gamma_1, \ldots, \gamma_k)$, where $\gamma_i \in \Gamma$ for all $i \in \{1, \ldots, k\}$.

Given the current location of the robot l_0 and a motion plan $\rho = (\gamma_1, \ldots, \gamma_k)$, the *trajectory* of the robot is given by $\xi = (\sigma_0 \sigma_1 \ldots \sigma_k)$ such that for all $i \in \{1, \ldots, k\}$, $\sigma_{(i-1)} \xrightarrow{\gamma_i} \sigma_i$. If any transition is invalid then the motion plan does not lead to a valid trajectory. In the rest of the paper, we use the word "step" to denote a transition governed by a motion primitive.

B. Problem Definition

Given a workspace, our goal is to find a location for the charging stations for a robot so that the robot can reach the charging station from any location on its trajectory without getting depleted with power. The robot starts its operation from a given initial location and may follow any trajectory that enables it to satisfy the specification. Initially, the robot starts with full power. When the power available to the robot goes below a pre-decided threshold, the robot needs to abort its mission and reach the charging station to charge its battery.

We assume that the robot has access to a motion planner that knows the location of the charging stations. Given the current location of the robot and the available battery power to the robot, the motion planner can find a path for the robot to reach a charging station, if there exists a path for the robot from its current location to a charging station location, which can be realized with the available power to the robot in its current location. If the robot knows the threshold on the power with which it is possible to reach a charging station

from any location in the workspace, it is easy for the robot to determine when to abort its current mission and invoke the path planner. The robot has to only keep track of its power level, and when it goes below the threshold it can invoke the path planner.

In this work, we assume that each motion primitive of the robot requires equal amount of energy and thus, there is a co-relation between the amount of battery power left to the robot and the number of motion primitive the robot will be executed with the remaining power. To represent the battery power available to the robot, we use the number of steps the robot is capable of executing with the available power.

Formally, the charging station placement problems can be captured as follows:

Problem 2.1: Given a workspace and a threshold on the number of transitions d, minimize the number of charging stations N and and find their locations $CS = \{l_{c1}, \ldots, l_{cN}\}$ in the workspace so that the robot can always reach the charging station from any obstacle-free location $l_f \in F$ in the workspace without executing more than d motion primitives.

Mathematically, the problem can be written as:

$$\begin{array}{ll} \texttt{minimize} & N \\ \texttt{subject to} & \forall l_f \in F.\ \exists \xi = (\sigma_0 \ldots \sigma_d)\ \texttt{with} \\ & \sigma_0.x = l_f\ \wedge \\ & (\sigma_d.x = l_{c1} \vee \ldots \vee \sigma_d.x = l_{cN})\ \wedge \\ & \sigma_d.v = v_0 \end{array}$$

In this optimization problem, the decision variables are the motion plan $\rho = (\gamma_1, \ldots, \gamma_d)$ for each each $l_f \in F$ and the locations of the charging stations captured as CS.

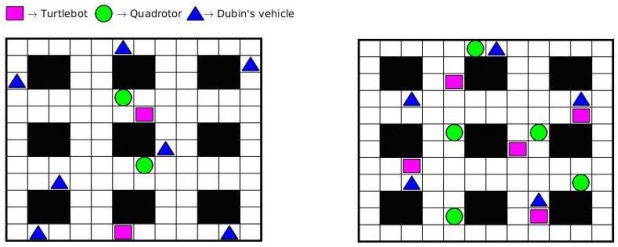
Problem 2.2: Given a workspace and a natural number N for the number of charging stations, find the location of the charging stations in such a way that maximum number of transitions required to reach one of the charging stations from any obstacle-free location in the workspace in minimized.

Mathematically, the problem can be written as:

$$\begin{array}{ll} \texttt{minimize} & d \\ \texttt{subject to} & \forall l_f \in F.\ \exists \xi = (\sigma_0 \sigma_1 \ldots \sigma_d)\ \texttt{with} \\ & \sigma_0.x = l_f\ \wedge \\ & (\sigma_d.x = l_{c1} \vee \ldots \vee \sigma_d.x = l_{cN})\ \wedge \\ & \sigma_d.v = v_0 \end{array}$$

In Problem 2.1, the number of transitions in a trajectory is given and the number of charging stations is a decision variable. On the other hand, in Problem 2.2, number of charging stations is given, whereas the number of transitions in the trajectory is a decision variable. In both the optimization problems, the motion plan $\rho = (\gamma_1, \ldots, \gamma_d)$ for each each $l_f \in F$ and the locations of the charging stations are also decision variables.

Though the required trajectories to reach a charging station from different obstacle-free locations may be of different length, we can attempt to synthesize the trajectories of the same length which is maximum among all the trajectories. Note that the presence of the rest primitive allows us to extend a trajectory to arbitrary finite length without changing the final state of the robot.



(a) d = 7 (b) ncs = 5

Fig. 1. The location of charging stations for different robots in a warehouse like workspace

C. Motivating Example

We illustrate the charging station placement problems through a simple example. Figure 1 shows the top view of a warehouse like workspace where a robot performs some tasks. The workspace is divided into cells using a rectangular grid. The black regions denote the locations occupied by obstacles. The robot may move through any obstacle-free location to perform its task.

In Figure 1(a), we show the location of the charging stations for three different robots for a fixed threshold for the number of transitions on a trajectory from any obstacle-free location in the workspace to a charging station. The threshold is taken as $d = 7$. The location of the charging stations for different robots have been shown in different symbols. As shown in the figure, the minimum number of charging stations required for the robot Turtlebot and the quadcopter is 2, whereas for Dubins vehicle, the number is 7. This is due to the fact that the dynamics of a Dubins vehicle is quite restricted. Also, unlike the Turtlebot and the quadcopter, the Dubins vehicle has a specific heading. From the same location, a dubins vehicle may be capable of reaching a charging station with a fixed threshold for d for some orientation, but not for the other orientations.

In Figure 1(a), we show the location of the charging stations when the number of charging stations is given. Here we assumed that the number of the charging stations is 5. For the Turtlebot and the quadcopter, the obtained value for d is 5, whereas for the Dubins vehicle, $d = 8$. Note that for the same workspace, increasing the value of the threshold d from 7 to 8 helps us reduce the number of charging stations for a Dubins vehicle by 2.

In the next section, we will describe our algorithms in detail.

III. Algorithms

In this section, we present our charging station placement algorithm.

A. Constraint Based Formulation

To solve the charging station placement problem for a workspace W, the set of motion primitives Γ for the robot and the set of locations for the obstacles O, we need to find the locations of a minimal number of charging stations CS given the value of d, or the minimum value of d and the

location of the charging stations, given the desired number of charging stations.

In both the cases, we need to find trajectories from all the obstacle-free locations in the workspace to one of the charging stations in CS. For the first problem, the number of charging stations and their locations are the decision variables and d is given. For the second problem, the charging station locations are given and d is a decision variable. In both the cases, we have the motion primitives that form the motion plan to move the robot from any obstacle-free location to a charging station location to be the decision variables.

The following set of constraints ensure that for all obstacle-free locations $l_i \in F$, for all velocity configuration $v_i \in V$, there exists a trajectory $\xi = (\sigma_0 \dots \sigma_d)$ and the corresponding motion plan $\rho = (\gamma_1 \dots \gamma_d)$ for reaching one of the charging station location $l_{cs} \in CS$ from l_i.

$\forall l_i \in F, \forall v_i \in V$:

$$\sigma_0.l = l_i \wedge \sigma_0.v = v_i \tag{III.1}$$

$$\forall t \in \{0, \dots, d-1\} : \sigma_t \models \mathit{pre}(\gamma_{(t+1)}) \tag{III.2}$$

$$\forall t \in \{1, \dots, d\} : \sigma_t = \mathit{post}(\sigma_{(t-1)}, \gamma_t) \tag{III.3}$$

$$\forall t \in \{0, \dots, d-1\} : \mathit{intermediate}(\sigma_t, \gamma_{(t+1)}) \notin O \tag{III.4}$$

$$\bigvee_{l_{cs} \in CS} \sigma_d.l = l_{cs} \wedge \sigma_d.v = 0 \tag{III.5}$$

Equation III.1 captures that initially the robot may be in any obstacle-free location in the workspace and may be in any velocity configuration. Equation III.2 ensures that at each time step t, the state σ_t of the robot should satisfy the precondition of the motion primitive applied to the robot at time instant t. Similarly, Equation III.3 ensures that the state σ_t should satisfy the postcondition of the motion primitive applied to the robot at time instant $t-1$. Equation III.4 captures that no location on a trajectory should be covered with an obstacle. Finally, Equation III.6 captures that at the final state of the trajectory, the robot will in one of the charging stations and stationary.

B. Algorithms for finding minimum number of charging stations

1) A Brute-Force Algorithm: We start with providing a brute-force algorithm for solving the charging station placement problem. The inputs to our algorithm are the set of free blocks in the workspace (F), the set of blocks occupied by the obstacles (O), the set of motion primitives for the robot under consideration (Γ), and a limit on the number of steps that the robot can take before getting depleted with the battery power. In this algorithm, we use the variable ncs to denote the number of required charging stations, which is initially set to 1. The algorithm iterates until we find sufficient number of charging stations to cover all the free blocks in the workspace. In each iteration, we generate constraints C that capture (III.1)-(III.6). While generating the constraints, we assume that $|CS| = ncs$ and treat all $l_{cs} \in CS$ to be the decision variables. In any iteration, if the constraints in C are satisfiable then the function `solve_constraints` return "SAT" in the variable res, otherwise, the function returns "$UNSAT$" to indicate that the constraints in C are unsatisfiable. If $res = SAT$ then a solution of the constraints in returned in the variable $model$. The function `get_charging_stations` extracts the locations of the charging stations from $model$. The motion plan for any free location in F to reach one of the charging station is also extracted form the model by the `get_motion_plans` function.

Algorithm III.1: Algorithm to find new location

Input: F: Free blocks in the workspace, O: set of obstacles, Γ: the set of motion primitives, d: maximum number of steps to reach the charging station

Output: C_{final} : final positions of the charging station

```
1  function findChargingStation(F, O, Γ, d)
2  begin
3      ncs = 1;
4      while true do
5          C = generate_constraints(F, O, Γ, d, ncs);
6          [res, model] = solve_constraints(C);
7          if res = SAT then
8              CS = get_charging_stations(model);
9              P = get_motion_plans(model);
10             return ⟨CS, P⟩;
11         else
12             ncs = ncs + 1;
13         end
14     end
15 end
```

Theorem 3.1: Algorithm III.1 provides minimal number of charging stations to solve Problem 1.

Proof: From the soundness of the SMT solver it follows that if we get the result of the function `solve_constraints` to be unsatisfiable for $ncs = k$ in an iteration, there does not exist a solution to the problem with $ncs = k$. Thus, a satisfiable solution for $ncs = k$ and an unsatisfiable solution for $ncs = k-1$ ensures that at least k charging stations are required in the solution of the problem. In Algorithm III.1, as we start with $ncs = 1$ and increase ncs by 1 in each iteration until we get a solution, it is guaranteed that the solution provided by Algorithm III.1 contains minimum number of charging stations. ■

2) A Scalable Algorithm: Algorithm III.1 tries to find the minimum number of charging stations and their locations iteratively. As we will see in the experimental results, the algorithm scales poorly with the size of the workspace and complexity of the robot dynamics. The main limitation of the algorithm is that when it does not find a solution for a specific value of ncs, it does not attempt to learn any information from that exercise and initiate a fresh new search for $ncs+1$. However, while solving the constraint for an ncs, if there is no solution, we may extract the information for which part of the obstacle-free workspace cannot be covered using ncs charging station. This information may make the search process easier for $ncs+1$ charging stations. This insight leads to Algorithm III.2 that we present in this section.

The inputs to Algorithm III.2 are the same as those of Algorithm III.1 (for this moment, ignore the input δ). In this algorithm, η denotes the current set of charging stations. initially, η is empty and $ncs = 1$. We generate constraints

Algorithm III.2: Algorithm to find new location

Input: F: Free blocks in the workspace, O: set of obstacles, Γ: the set of motion primitives, d: maximum number of steps to reach the charging station,δ: relaxation parameter

Output: C_{final} : final positions of the charging station

```
1  function findChargingStation (F, O, Γ, d, δ)
2  begin
3      ncs = 1;
4      η = {  };
5      while true do
6          C = generate_constraints(F, O, Γ, d, η, ncs, δ);
7          [res, model, ucore] = solve_constraints(C);
8          if res = SAT then
9              CS = get_charging_stations(model);
10             P = get_motion_plans(model);
11             return ⟨CS, P⟩;
12         else
13             U = get_initial_locations(ucore);
14             flag1 = false;
15             ncs = ncs + 1;
16             while flag1 = false do
17                 C =
                    generate_constraints(U, O, Γ, d, η, ncs, δ);
18                 [res, model, ucore] =
                    solve_constraints(C);
19                 if res = SAT then
20                     η = get_charging_stations(model);
21                     flag1 = true;
22                 else
23                     ncs = ncs + 1;
24                 end
25             end
26         end
27     end
28 end
```

from the inputs F, O, Γ, and d, the set of charging stations η and ncs by using (III.1)-(III.6). In any iteration, $|\eta| \leq ncs$. Thus, we treat η elements in CS to be fixed and the rest $ncs - |\eta|$ elements in CS to be the decision variables. Moreover, we include the constraints of the form $\sigma_0.l = l_i$ for all $l_i \in F$ in the set of constraints that will be used for the generation of *unsatisfiable core*. Now, if we do not find a solution for a specific value of ncs, we can generate a set of locations U, $U \subseteq F$, that captures some of the obstacle-free locations in the workspace that cannot be served using ncs charging stations. Now, we increase the value of ncs by 1 and generate constraints only to cover the locations in U. We keep on increasing ncs until we get a solution that provides the locations of the charging stations η that cover the locations in U. With this set η, now we move to Line 6 to validate if the current locations of the charging stations η can serve all the locations in F. If not then we repeat the process. In any iteration, if the constraints can be solved successfully on Line 7, the model generated by the solver can be used for finding the location of the charging stations and the motion plans for all the locations in $l \in F$ to reach one of the charging stations.

The algorithm described above may not provide a solution with minimal number of charging stations, as a solution obtained from the locations given by the unsatisfiable core may not be the optimal solution when we consider all the locations in the outer loop in Algorithm III.2. However, as we check the feasibility of the having minimal number of charging stations for several iterations only for the locations in U instead of the locations in F and the set U is significantly smaller than the set U, we achieve significant speed-up. Here, we comprise optimality to reduce the complexity of the problem and get a solution faster.

We will now describe how we use the parameter δ in our algorithm to get more optimal result. For $\delta = 0$, the algorithm works as described above. If $\delta > 0$ then in generating the constraints, we replace the constraints in (III.6) by the following constraints:

$$\bigvee_{l_{cs} \in CS} \sigma_d.l \in L^{\delta}_{cs} \wedge \sigma_d.v = 0 \tag{III.6}$$

where L^{δ}_{cs} denotes a set containing all the locations which are within δ *Manhattan distance* away from the location l_{cs}. A $\delta > 0$ let us consider the neighboring regions of the charging station locations obtained by considering only the locations provided by the unsatisfiable core, without expanding the search space too much.

C. Algorithms to Find the minimum number of transitions

In this section, we will present an algorithm to the solve Problem 2.2, that is, given a number of charging stations find the location of the charging stations and the minimum number of transitions with which a robot will be able to reach one of the charging stations from any location $l \in F$. The algorithm is rather straightforward. We start with $d = 1$ and we keep on increasing d until we can establish that for that value of d, there exists ncs charging stations locations CS so that a robot in any location in F will be able to reach one of the charging stations given in CS.

Algorithm III.3: Algorithm to find new location

Input: F: free blocks in the workspace, O: the set of obstacles, Γ: the set of motion primitives, ncs: the number of charging stations

Output: d_{min} : the minimum number of transitions

```
1  function findChargingStation (F, O, Γ,)
2  begin
3      d = 1;
4      while true do
5          C = generate_constraints(F, O, Γ, ncs);
6          [res, model] = solve_constraints(C);
7          if res = SAT then
8              CS = get_charging_stations(model);
9              P = get_motion_plans(model);
10             return ⟨d, CS, P⟩;
11         else
12             d = d + 1;
13         end
14     end
15 end
```

Theorem 3.2: Algorithm III.3 provides minimal number of steps for the robot to reach a charging station from any obstacle-free location in the workspace.

Proof: The proof is similar to the proof of Theorem 3.1. ■

Algorithm III.4: Algorithm to find charging station location to minimize d

Input: F: Free blocks in the workspace, O: set of obstacles, Γ: the set of motion primitives, ncs: the number of charging station,δ: relaxation parameter

Output: C_{final} : final positions of the charging station, d: the minimum number of steps to reach the charging station

```
1  function findChargingStation(F, O, Γ, ncs, δ)
2  begin
3      d = 1;
4      η = { };
5      while true do
6          C = generate_constraints(F, O, Γ, d, η, ncs, δ);
7          [res, model, ucore] = solve_constraints(C);
8          if res = SAT then
9              CS = get_charging_stations(model);
10             P = get_motion_plans(model);
11             return ⟨d, CS, P⟩;
12         else
13             U = get_initial_locations(ucore);
14             flag1 = false;
15             d = d + 1;
16             while flag1 = false do
17                 C =
                     generate_constraints(U, O, Γ, d, η, ncs, δ);
18                 [res, model, ucore] =
                     solve_constraints(C);
19                 if res = SAT then
20                     η = get_charging_stations(model);
21                     flag1 = true;
22                 else
23                     d = d + 1;
24                 end
25             end
26         end
27     end
28 end
```

As Algorithm III.3 suffers from the lack of scalability, we design an unsatisfiable core based algorithm (Algorithm III.4.) to solve Problem 2.2. The algorithm is similar to Algorithm III.2, the main difference is that here d is unknown, and ncs is given.

IV. Evaluation

A. Experimental Setup

We have carried out rigorous experiments to judge the efficacy of the proposed algorithms. Our experiments have been carried out in a laptop with Intel Core i7-6500U processor with 2.50GHz clock speed, 16GB RAM, and Ubuntu 14.04 on it. We use Z3 SMT solver [6] from Microsoft Research as backend constraint solver. The algorithms have been implemented in C++. For visualization of the workspace with the charging stations, we have developed a tool in MATLAB. Furthermore, we carry simulation in ROS [27] using Turtlebot robot.

1) Workspaces: Three types of workspaces, viz. *Warehouse*, *Artificial floor* and *Maze* have been used in two-dimensional setting. Number of free locations is largest in Artificial floor, and smallest in Warehouse. We have tested our algorithms for two different dimensions — 12×12 and 17×17 of these workspaces.

2) Robots and Motion Primitives: We have carried out our experiments with *three* different types of robots —*Turtlebot*, *Quadcopter* and *Dubin's vehicle*. These robots have very different set of motion primitives. The Turtlebot has nine motion primitives - one is the rest primitive and the others are to move the robot by one grid cell in eight uniform directions. For the quadcopter, we have 58 motion primitives that have been generated by the algorithm presented in [28], [29].

We have created 16 motion primitives for a Dubin's vehicle. Unlike the Turtlebot or quadcopter, we need to consider the orientation of the robot. While composing two motion primitives for a Turtlebot, we ensure that the orientation of the robot after executing the first motion primitive matches with the initial orientation required by the second primitive. Moreover, as the robot can be in any direction in any obstacle-free location initially, We consider all location-orientation combinations in formulating the constraints in our algorithms.

B. Results

In this section we present our experimental results. We ran Algorithm III.1 and Algorithm III.2 with fixed number of trajectory points $d = 7$ with various parameters like workspace type, workspace dimension, robot type, δ and number of trajectory points. The experimental results are shown in Table I and Table II. For Algorithm III.2, with the increase in δ, the number of charging stations decreases significantly. Though the execution time increases with the increase of δ, the result produced by Algorithm III.2 reaches closer to the optimal solution provided by Algorithm III.1 as we keep on increasing δ. We have carried out experiments up to $\delta = 2$. As we compare the results of naive algorithm, we can see that in many cases the optimal *ncs* is reached with $\delta = 2$. Most importantly, the execution time of Algorithm III.2 is 80–85% better compared to the Algorithm III.2 (*naive* algorithm).

To find out minimum number of steps d with fixed number of charging stations ncs, we executed AlgorithmIII.3 and AlgorithmIII.4. We did our experiment for $\delta = 0$ to $\delta = 3$. In Table III, we carried out our experiments for fixed $ncs = 5$ and *Turtlebot*, with varying parameters like workspace type, workspace dimension and δ. Table III shows that with the increase of δ our Algorithm III.4 steadily approaches to the optimal value of d, which is given by the naive Algorithm III.3 . TableIV contains results for two different robots —quadrotor and Dubin's vehicle with warehouse workspace. Execution time of Algorithm III.4 is $30 - 85\%$ better than the naive AlgorithmIII.3. For quadrotor, we have carried out experiments for $ncs = 7$ and for Dubins vehicle we have carried out experiments for $ncs = 9$.

Figure 2 shows how the computation time increases and the number of charging stations decreases steadily with the

Workspace	Algorithm	dimension 12×12			dimension 17×17		
		T	ncs	CS	T	ncs	CS
Warehouse	$\delta = 0$	0m 17s	3	(7 5),(4 1),(0 10)	2m 12s	9	(15 16),(1 10),(2 6),(16 9),(16 2),(14 4),(10 9),(4 13),(0 3)
	$\delta = 1$	0m 32s	3	(8 5),(4 1),(5 7)	3m 59s	7	(16 13),(3 10),(2 4),(14 4),(13 16),(7 0),(4 15)
	$\delta = 2$	0m 32s	2	(6 7),(5 0)	4m 0s	5	(13 16),(1 12),(3 4),(13 10),(12 2)
	naive	3m 22s	2	(3 6),(7 5)	47m 16s	4	(12 1),(12 13),(3 12),(4 1)
Artificial floor	$\delta = 0$	0m 13s	2	(8 5),(2 5)	1m 8s	4	(10 14),(3 2),(4 12),(14 5)
	$\delta = 1$	0m 26s	2	(8 5),(3 6)	3m 3s	5	(11 13),(4 4),(3 11),(14 8),(10 2)
	$\delta = 2$	0m 45s	2	(9 6),(2 5)	4m 55s	4	(12 10),(4 5),(1 11),(12 4)
	naive	5m 55s	2	(10 5),(3 6)	152m 16s	4	(15 3),(4 4),(13 11),(6 11)
Maze	$\delta = 0$	0m 15s	4	(10 4),(5 9),(5 2),(2 5)	1m 38s	7	(3 4),(14 10),(10 3),(8 10),(4 10),(5 14),(12 11)
	$\delta = 1$	0m 29s	3	(9 5),(4 8),(5 2)	3m 25s	7	(4 4),(14 10),(2 14),(6 10),(5 2),(14 4),(10 11)
	$\delta = 2$	1m 59s	3	(8 2),(9 8),(2 5)	7m 59s	6	(1 4),(14 10),(4 14),(10 6),(13 4),(4 7)
	naive	9m 36s	3	(1 7),(9 6),(4 2)	57m 36s	5	(10 14),(2 5),(13 5),(9 6),(3 12)

TABLE I

EXPERIMENTAL RESULTS FOR *Turtlebot* WITH FIXED VALUE OF $d = 7$

Workspace	Algorithm	Quadcopter			Dubin's vehicle		
		T	ncs	CS	T	ncs	CS
Warehouse	$\delta = 0$	1m 16s	2	(7 6),(4 6)	6m 33s	10	(9 4),(6 7),(11 5),(2 3),(5 11),(9 11),(1 8),(3 1),(11 2),(2 11)
	$\delta = 1$	5m 12s	2	(8 6),(3 5)	34m 12s	10	(3 5),(8 10),(2 3),(4 9),(9 3),(7 10),(7 1),(10 8),(1 8),(0 1)
	$\delta = 2$	7m 23s	2	(5 8),(6 4)	26m 46s	7	(10 0),(0 9),(5 11),(1 0),(2 3),(11 10),(7 5)
	naive	36m 31s	2	(3 10),(6 3)	> 200m		timeout
Artificial floor	$\delta = 0$	2m 59s	4	(7 4),(5 10),(4 5),(9 4)	14m 27s	13	(9 6),(6 9),(3 1),(1 9),(7 0),(10 7),(9 4),(8 2),(0 8),(7 10),(5 7),(5 2),(1 3)
	$\delta = 1$	10m 49s	4	(6 5),(8 9),(10 4),(1 5)	22m 36s	10	(10 7),(6 9),(4 4),(0 8),(8 2),(9 4),(1 3),(7 10),(6 1),(5 7)
	$\delta = 2$	4m 31s	2	(6 5),(6 8)	76m 57s	10	(8 2),(2 7),(10 7),(5 1),(1 3),(6 5),(6 9),(7 10),(10 4),(5 7)
	naive	43m 22s	2	(2 5),(9 6)	> 200m		timeout
Maze	$\delta = 0$	3m 7s	5	(6 9),(0 8),(9 6),(7 6),(2 7)	11m 31s	13	(2 6),(9 8),(8 2),(4 2),(2 9),(2 7),(9 6),(9 5),(8 4),(3 4),(7 9),(6 6),(7 7)
	$\delta = 1$	9m 5s	4	(5 9),(2 6),(11 3),(7 7)	14m 18s	8	(1 7),(8 9),(2 2),(8 2),(4 8),(9 5),(9 4),(7 7)
	$\delta = 2$	16m 58s	4	(7 7),(2 6),(10 2),(7 9)	45m 22s	8	(1 7),(8 2),(4 7),(2 2),(6 9),(9 8),(9 4),(7 7)
	naive	61m 31s	3	(1 7),(7 7),(9 8)	> 200m		timeout

TABLE II

EXPERIMENTAL RESULTS FOR *Quadcopter* AND *Dubin's vehicle* WITH DIFFERENT WORKSPACES OF SIZE 12×12 AND FIXED VALUE OF $d = 7$

Workspace	Algorithm	Dimension 12×12			Dimension 17×17		
		T	d	CS	T	d	CS
Warehouse	$\delta = 0$	0m 10s	6	(11 9),(10 0),(9 4),(0 2),(4 10)	4m 56s	13	(6 3),(9 4),(16 14),(13 16),(16 13)
	$\delta = 1$	0m 17s	5	(11 10),(7 2),(8 5),(1 3),(2 8)	6m 15s	11	(1 10),(4 13),(12 3),(9 6),(7 16)
	$\delta = 2$	0m 24s	5	(10 7),(8 1),(7 5),(2 4),(4 9)	5m 56s	9	(1 12),(1 16),(9 4),(3 6),(9 16)
	$\delta = 3$	0m 37s	5	(10 7),(9 4),(1 3),(1 8),(1 7)	5m 21s	7	(12 3),(14 16),(3 4),(10 9),(4 13)
	naive	4m 14s	5	(9 11),(7 2),(1 3),(8 6),(2 7)	64m 26s	7	(13 12),(5 13),(11 3),(3 4),(4 2)
Artificial floor	$\delta = 0$	0m 14s	7	(0 8),(5 10),(7 0),(10 2),(1 5)	2m 56s	11	(10 1),(7 15),(9 8),(9 15),(11 7)
	$\delta = 1$	0m 30s	7	(6 5),(4 2),(0 8),(2 7),(8 2)	8m 8s	10	(4 12),(15 7),(6 7),(14 3),(7 3)
	$\delta = 2$	0m 40s	6	(4 4),(9 6),(6 1),(2 7),(3 8)	2m 44s	10	(10 1),(9 0),(15 7),(6 7),(6 3)
	$\delta = 3$	0m 18s	4	(3 8),(3 3),(9 3),(8 9),(5 2)	7m 33s	8	(9 9),(4 3),(5 5),(12 6),(6 11)
	naive	1m 45s	4	(3 6),(1 9),(9 3),(8 9),(3 3)	14m 34s	5	(1 4),(4 12),(13 3),(12 12),(8 4)
Maze	$\delta = 0$	0m 20s	8	(2 7),(6 9),(2 4),(1 9),(7 2)	52m 44s	15	(12 10),(14 8),(4 14),(9 5),(11 14)
	$\delta = 1$	0m 30s	7	(6 3),(9 8),(9 2),(2 5),(8 4)	67m 50s	12	(9 4),(8 12),(7 9),(10 9),(12 4)
	$\delta = 2$	1m 8s	6	(7 9),(9 4),(0 8),(6 3),(2 5)	15m 30s	8	(4 7),(10 5),(14 12),(6 12),(14 7)
	$\delta = 3$	0m 35s	5	(2 4),(6 3),(10 2),(7 9),(0 8)	15m 13s	9	(4 8),(14 12),(0 13),(6 9),(9 5)
	naive	3m 29s	5	(2 4),(6 3),(10 2),(7 10),(4 9)	94m 52s	7	(2 5),(15 4),(0 14),(10 15),(9 6)

TABLE III

EXPERIMENTAL RESULTS FOR *Turtlebot* WITH FIXED VALUE OF $ncs = 5$

Workspace	Algorithm	Quadcopter ($ncs = 7$)			Dubin's vehicle ($ncs = 9$)		
		T	d	CS	T	d	CS
Warehouse	$\delta = 0$	0m 55s	6	(3 1),(3 2),(4 10),(0 2),(9 7),(9 4),(6 0)	6m 51s	8	(9 4),(5 0),(9 11),(10 3),(0 9),(3 6),(11 1),(11 9),(1 4)
	$\delta = 1$	3m 2s	5	(2 0),(5 3),(4 9),(3 5),(7 10),(10 4),(6 0)	34m 51s	8	(9 3),(8 2),(9 0),(2 3),(7 5),(2 8),(5 8),(1 0),(8 9)
	$\delta = 2$	4m 47s	5	(6 3),(5 7),(3 10),(1 3),(9 4),(10 7),(3 1)	24m 45s	6	(0 5),(9 4),(9 11),(0 9),(5 4),(9 0),(2 0),(9 7),(4 9)
	$\delta = 3$	1m 16s	4	(5 3),(5 0),(1 7),(0 2),(6 11),(8 6),(11 5)	44m 27s	6	(9 11),(4 6),(9 8),(2 0),(0 9),(5 11),(1 3),(8 1),(10 3)
	naive	9m 55s	4	(7 10),(8 3),(8 8),(11 5),(4 10),(0 5),(3 3)	> 200m		timeout

TABLE IV

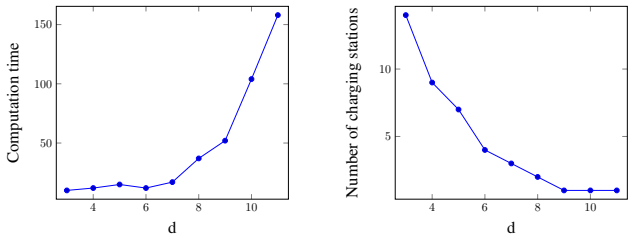
EXPERIMENTAL RESULTS FOR *Quadcopter* ($ncs = 7$) AND *Dubin's vehicle* ($ncs = 9$) WITH WORKSPACE SIZE 12×12

increase in d. Figure 3 shows how d decreases with the increase in ncs.

REFERENCES

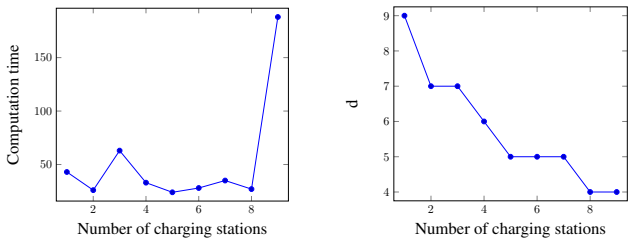
[1] R. Bogue, "Growth in e-commerce boosts innovation in the warehouse robot market," *Industrial Robot: An International Journal*, vol. 43, no. 6, pp. 583–587, 2016. [Online]. Available: http://dx.doi.org/10.1108/IR-07-2016-0194

[2] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *IEEE Spectrum*, vol. 45, no. 7, pp. 26–34, 2008.



(a) (b)

Fig. 2. The effect of increasing d on the computation time and the number of charging stations obtained



(a) (b)

Fig. 3. The effect of increasing the number of charging station on the computation time and d

[3] B. Tedeschi, "The year in robots: 10 home robots to lighten your domestic chores," *The New York Times*, 2014. [Online]. Available: https://www.nytimes.com/2014/12/25/garden/10-home-robots-to-lighten-your-domestic-chores.html?mcubz=3

[4] T. Deyle, "Why indoor robots for commercial spaces are the next big thing in robotics," *IEEE Spectrum*, 2017. [Online]. Available: https://spectrum.ieee.org/automaton/robotics/robotics-hardware/indoor-robots-for-commercial-spaces

[5] C. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, A. Biere, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, vol. 4, ch. 8.

[6] L. M. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *International Conference of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2008, pp. 337–340.

[7] Z. Drezner, *Facility location: a survey of applications and methods*. Springer, 1995.

[8] H. W. Hamacher and Z. Drezner, *Facility location: applications and theory*. Springer, 2002.

[9] M. Boccia, A. Sforza, and C. Sterle, "Flow intercepting facility location: Problems, models and heuristics," *J. Mathematical Modelling and Algorithms*, vol. 8, no. 1, pp. 35–79, 2009.

[10] W. Zeng, I. Castillo, and M. J. Hodgson, "A generalized model for locating facilities on a network with flow-based demand," *Networks and Spatial Economics*, vol. 10, no. 4, pp. 579–611, 2010.

[11] S. Mitra, P. Saraf, R. Sharma, A. Bhattacharya, S. Ranu, and H. Bhandari, "Netclus: A scalable framework for locating top-k sites for placement of trajectory-aware services," in *ICDE*, 2017, pp. 87–90.

[12] M. Kuby, L. Lines, R. Schultz, Z. Xie, J.-G. Kim, and S. Lim, "Optimization of hydrogen stations in florida using the flow-refueling location model," *International journal of hydrogen energy*, vol. 34, no. 15, pp. 6045–6064, 2009.

[13] S. Lim and M. Kuby, "Heuristic algorithms for siting alternative-fuel stations using the flow-refueling location model," *European Journal of Operational Research*, vol. 204, no. 1, pp. 51–61, 2010.

[14] S. MirHassani and R. Ebrazi, "A flexible reformulation of the refueling station location problem," *Transportation Science*, vol. 47, no. 4, pp. 617–628, 2012.

[15] S. Mitra, S. Ranu, V. Kolar, A. Telang, A. Bhattacharya, R. Kokku, and S. Raghavan, "Trajectory aware macro-cell planning for mobile users," in *INFOCOM*, 2015, pp. 792–800.

[16] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Pearson, 2009.

[17] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion*. A Bradford Book, 2005.

[18] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[19] R. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artif. Intell.*, vol. 2, no. 3/4, pp. 189–208, 1971.

[20] W. N. N. Hung, X. Song, J. Tan, X. Li, J. Zhang, R. Wang, and P. Gao, "Motion planning with Satisfiability Modulo Theroes," in *ICRA*, 2014, pp. 113–118.

[21] S. Nedunuri, S. Prabhu, M. Moll, S. Chaudhuri, and L. E. Kavraki, "SMT-based synthesis of integrated task and motion plans from plan outlines," in *ICRA*, 2014, pp. 655–662.

[22] Y. Wang, N. T. Dantam, S. Chaudhuri, and L. E. Kavraki, "Task and motion policy synthesis as liveness games," in *ICAPS*, 2016, p. 536.

[23] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Automated composition of motion primitives for multi-robot systems from safe LTL specifications," in *IROS*, 2014, pp. 1525–1532.

[24] ——, "Implan: Scalable incremental motion planning for multi-robot systems," in *ICCPS*, 2016, pp. 43:1–43:10.

[25] A. Desai, I. Saha, J. Yang, S. Qadeer, and S. A. Seshia, "DRONA: a framework for safe distributed mobile robotics," in *ICCPS*, 2017, pp. 239–248.

[26] I. Gavran, R. Majumdar, and I. Saha, "ANTLAB: a multi-robot task server," in *EMSOFT*, 2017.

[27] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[28] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *ICRA*, 2011, pp. 2520–2525.

[29] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-UAV motion replanning for exploring unknown environments," in *ICRA*, 2013, pp. 2452–2458.