Energy-Aware Temporal Logic Motion Planning for Mobile Robots

Tanmoy Kundu[1] and Indranil Saha[2]

***Abstract*— This paper presents a methodology for synthesizing a motion plan for a mobile robot to ensure that the robot never gets depleted with battery charge while carrying out its mission successfully. The specification of the robot is provided in the form of an LTL (Linear Temporal Logic) formula. A trajectory satisfying an LTL formula may contain a loop whose repetitive execution causes the depletion of battery charge in the robot. The motion plan generated by our methodology ensures that the robot visits the charging station periodically in such a way that it never gets depleted with battery charge while carrying out its mission optimally. Given a set of potential charging station locations and an LTL specification, our algorithm also finds the best location for the charging station along with the optimal trajectory for the robot. We encode the motion planning problem as an SMT (Satisfiability Modulo Theory) solving problem and use the off-the-shelf SMT solver Z3 to solve the constraints to find the location of the charging station and generate an optimal trajectory for the robot. We apply our methodology to synthesize energy-aware trajectories for robots with different dynamics in various workspaces and for various LTL specifications.**

I. Introduction

Traditionally, the objective of motion planning has been to move a robot from point A to point B while avoiding obstacles. Recently, there has been an increased interest in solving planning problems where complex specifications are captured using temporal logic [1], [2], [3], [4], [5], [6], [7]. Using temporal logic, one can specify requirements that involve temporal relationships between different operations performed by the robots. Such requirements arise in many robotic applications, including persistent surveillance [6], assembly planning [8], evacuation [9], search and rescue [10], localization [11], object transportation [12], and formation control [13]. In these applications, it is common that a robot has to perform a task repeatedly by following a preassigned trajectory. A major outstanding problem in performing a series of tasks perpetually is how to schedule the charging of the battery of the robot in an optimal way so that it never gets depleted with energy and becomes non-functional.

To ensure persistent power supply to the mobile robots, several approaches have been proposed in the past. Docking based autonomous recharging has been studied widely in the recent past (e.g. [14], [15], [16], [17], [18], [19], [20]). In another approach, a flexible tether connects the robot to a continuous power supply, thus the tethered robot's mission duration extends indefinitely without being reliant on the limited battery charge (e.g. [21], [22], [23], [24]). The problem of motion planning for the robots that can exploit natural power resources has been addressed by several authors (e.g. [25], [26], [15]). Lahijanian et al. [27] proposed a framework for exploring the resource-performance trade-off, which provides a mechanism to conserve energy at the cost of reduced but acceptable level of performance. In a recent work, we proposed a methodology to place a set of charging stations optimally in a workspace to support uninterrupted operation of a mobile robot [28]. However, none of the previous work has addressed the problem of charging station placement and motion planning for satisfying a temporal logic property perpetually while ensuring the continuous availability of energy.

In this paper, our objective is to generate a trajectory for a robot automatically from a given LTL specification that along with the functional requirement also takes into account the energy requirement of the robot. Our approach is based on the composition of motion primitives, that respects the constraints imposed by the temporal logic specification. The motion primitives are utilized to build a system of constraints where the decision variables encode the choice of motion primitives used at any discrete-time point on the trajectory. The system of constraints involve a Boolean combination of linear constraints, which can be solved using an off-the-shelf SMT (Satisfiability Modulo Theories) solver [29]. Our choice of the SMT-based approach has been driven by the recent success of SMT solvers to solve several task and motion planning problems (e.g. [30], [31], [32], [7], [33], [34], [35]).

The trajectories satisfying an LTL formula might be of infinite length, capturing the repetition of tasks. In the SMT constraints, we include constraints to ensure that the robots never get depleted with battery power. However, a naive encoding of the energy constraints leads to conservative trajectories that force the robot to visit the charging station more frequently than required. We provide an algorithm to find a trajectory that ensures the optimal number of visits to the charging station as well as optimal length of the trajectory under the assumption that the robot gets recharged fully while it visits the charging station.

We apply our technique to synthesize trajectories for a pickup-drop application. In such an application, a robot is required to (repeatedly) pick up objects from different gathering locations and drop the gathered objects at a drop location. We carry out experiments with robots with three different dynamics: Turtlebot, Dubin's Vehicle and Quadcopter. Our experimental results show that our technique can efficiently find the trajectory for the robot with minimal number of visits to the charging station while performing its designated task.

II. Problem

A. Preliminaries

Workspace: In this work, we assume that the robot operates in a 2-D workspace that we represent as a 2-D occupancy grid map. The grid decomposes the workspace

*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

into square shaped blocks that 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 obstacle-free locations in the workspace is denoted by F, where $F = W \setminus O$. We consider a subset of the obstacle-free locations in the workspace, $S \subseteq F$, as the potential charging station locations. We find the exact location of the charging station in S.

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

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 by executing a sequence of motion primitives.

With each motion primitive $\gamma \in \Gamma$, we associate a *pre-condition* $pre(\gamma)$, which is a formula over the states specifying under which conditions a motion primitive can be executed. We write $post(\sigma, \gamma)$ for the state the robot attains after executing the motion primitive γ. 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 an energy cost as denoted by $cost(\gamma)$, which represents the amount of energy spent by the robot while executing the motion primitive.

Recharge Primitive: The robot is equipped with a *recharge primitive*, denoted by ν, that the robot can use to recharge its battery to the maximum possible energy level e_{max}. Like a motion primitive, ν is associated with a precondition $pre(\nu)$ and a postcondition $post(\sigma, \nu)$. A state σ satisfies the precondition $pre(\nu)$ if $\sigma.x \in S \ \wedge \sigma.v = v_0$. We write $post(\sigma, \nu)$ to denote the state σ' such that $\sigma'.x = \sigma.x \ \wedge \ \sigma'.v = v_0 \ \wedge \ \sigma'.e = e_{max}$. Moreover, $intermediate(\sigma, \gamma) = \{\sigma.x\}$.

Execution Plan and Trajectory: The runtime behavior of the robot is described by a discrete-time transition system $\mathcal{T}$. Let σ_1 and σ_2 be two states of the robot. For a motion primitive or recharge primitive ρ, $\sigma_1 \xrightarrow{\rho} \sigma_2$ is a valid transition iff $\sigma_1 \models pre(\rho)$, $\sigma_2 = post(\sigma_1, \rho)$, and $intermediate(\sigma_1, \rho) \ \cap O = \emptyset$.

An *execution plan* for a robot is defined as a sequence of motion primitives and recharge primitives to be applied to the robot to move it in a way that it satisfies a user given specification and never gets depleted with battery energy. An execution plan is denoted by a (potentially infinite) sequence of primitives $\rho = (\rho_1 \rho_2 \ldots)$, where $\rho_i \in \Gamma \cup \{\nu\}$ for all $i \in \{1, 2, \ldots\}$.

Given the current location of the robot l_0 and an execution plan $\rho = (\rho_1 \rho_2 \rho_3 \ldots)$, the *trajectory* of the robot is given by $\eta = (\sigma_0 \sigma_1 \sigma_2 \ldots)$ such that for all $i \in \{1, 2, 3, \ldots\}$, $\sigma_{i-1} \xrightarrow{\rho_i} \sigma_i$. If any transition is invalid then the execution 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.

We represent by $\eta \circ \eta'$ the trajectory obtained by concatenating a finite trajectory η and another (finite or infinite) trajectory η'. For a finite trajectory η, the infinite trajectory obtained by repeating η is represented by η^ω, and the finite trajectory obtained by repeating η for $k \in \mathbb{N}$ times is represented by η^k .

B. Specification Language – Linear Temporal Logic (LTL)

We express the specification of a robot using Linear Temporal Logic [36], denoted by LTL. Let AP denote the set of atomic propositions. An atomic proposition captures the truth value of a condition defined based on the state of the robot. For example, we can define a proposition p_l which will be `true` if the robot is in a location $l \in F$, i.e., $\sigma.x = l$. From the atomic propositions in AP, any LTL formula can be formulated according to the following grammar:

$$\xi ::= \texttt{true} \mid p \mid \neg\xi \mid \xi_1 \wedge \xi_2 \mid \bigcirc \xi \mid \xi_1 \, \mathcal{U} \, \xi_2$$

The basic ingredients of an LTL formulae are the atomic propositions, the Boolean connectors like conjunction $\wedge$, and negation $\neg$ and two temporal operators $\bigcirc$ (next) and $\mathcal{U}$ (until). The semantics of an LTL formula is defined over an infinite trajectory σ. The trajectory σ satisfies a formula ξ, if the first state of σ satisfies ξ. For an LTL formula ξ, $\bigcirc\xi$ is `true` in a state if ξ holds at the next step. The Until operator $\mathcal{U}$ has two operands. The formula $\xi_1 \, \mathcal{U} \, \xi_2$ denotes that ξ_1 must remain `true` until ξ_2 becomes `true` at some point in future. The other LTL operators that can be derived from the above formulas are *Always* $\Box$ and *Eventually* $\Diamond$. The formula $\Box\xi ::= \neg\Diamond\neg\xi$, which denotes that the formula ξ has to hold for all the time points in the future. The formula $\Diamond\xi ::= \texttt{true} \, \mathcal{U} \, \xi$ denotes that the formula ξ will become true at some point in the future.

LTL Specification with Energy Constraint: The robot has to satisfy additional safety constraint that its charge $\sigma.e$ is always greater than 0. Let us denote the condition as the proposition $energy$. A trajectory satisfying this constraint is called an *energy-safe* trajectory. For any user given functional specification ξ, the robot should also satisfy the LTL property $\Box energy$. Thus, the combined LTL formula from which we aim to generate an execution plan for the robot is given by $(\xi \wedge \Box energy)$.

Representation of an infinite trajectory: An infinite trajectory that satisfies an LTL formula can be given a finite representation having a prefix and a suffix that can loop to generate a valid infinite trajectory [37]. Let $\kappa = (\sigma_0 \sigma_1 \sigma_2 \ldots)$ be a valid trajectory of the system for the formula ξ. The trajectory κ can be represented as: $\kappa = (\sigma_0 \sigma_1 \ldots \sigma_k)(\sigma_{k+1} \ldots \sigma_{L+1})^\omega$ where $0 < k \leq L$ and $\sigma_{L+1} = \sigma_k$. Such a representation of a trajectory is called an (L, k)-loop. The trajectory $(\sigma_{k+1} \ldots \sigma_{L+1})^\omega$ denotes an infinite trajectory that can be obtained by executing the finite sequence $(\sigma_{k+1} \ldots \sigma_{L+1})$ repetitively.

Optimal Trajectory: A trajectory κ for an LTL formula ξ is *optimal* if there does not exist another trajectory κ' satisfying ξ that can be synthesized using the motion primitives in

Γ such that $\texttt{length}(\kappa') < \texttt{length}(\kappa)$, where $\texttt{length}(\kappa)$ denotes the number of motion primitives used in realizing κ.

C. Problem Definition

In this section, we define our problem formally.

Definition 2.1 (Input Problem Instance): An Input problem instance can be given as a 6 tuple $\mathcal{P} = \langle W, O, S, \Gamma, I, \xi \rangle$, where I denotes the initial state of the robot where the robot starts its operation with full battery power and ξ denotes the LTL specification the robot has to satisfy.

Formally, the problem can be captured as follows:

Problem 2.2: Given an input problem instance $\mathcal{P} = \langle W, O, S, \Gamma, I, \xi \rangle$, determine the location of the charging station $l_c \in S$ and generate an optimal trajectory that satisfies the LTL formula ξ together with the constraint representing energy safety.

The generated trajectory should enable the robot to abort its mission and reach the charging station to recharge its battery whenever necessary, and resume its mission again after the recharge is done.

III. Algorithms

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

A. Energy-Safe Trajectory Generation

First, we present an algorithm that without ensuring optimality synthesizes a trajectory satisfying the given LTL specification and energy-safety. Our approach is based on reducing the problem to an SMT solving problem. Here we describe the system of constraints that models the planning problem introduced in Section II.

Given an input problem instance $\mathcal{P} = \langle W, O, S, \Gamma, I, \xi \rangle$ and a positive constant L, our objective is to generate an SMT formula that represents any valid trajectory of the robot in the form of an (L, k)-loop. The decision variables for the SMT formula are the motion primitives or recharge primitive to be used at each state and the variable representing the location where the loop starts (k). Below we present the encoding of the trajectory synthesis problem. The encoding is linear in L, and consists of three sets of constraints:

1) System Constraints: At each time instant $t \in \{0, \ldots, L+1\}$, the state of the robot is denoted by σ_t, and the primitive applied to the robot is denoted by ρ_{t+1}. The system constraints are captured by the following formula: $|[SYS]| = I(\sigma_0) \wedge \bigwedge_{t=0}^{L} T(\sigma_t, \rho_{(t+1)}, \sigma_{(t+1)})$. The transition relation $T(\sigma_t, \rho_{(t+1)}, \sigma_{(t+1)})$ is implemented as the following constraint: $\rho_{(t+1)} \in \Gamma \cup \{\nu\} \wedge \sigma_t \models pre(\rho_{t+1}) \wedge \sigma_{t+1} = post(\sigma_t, \rho_{t+1}) \wedge \textit{intermediate}(\sigma_t, \rho_{t+1}) \notin O$.

2) LTL Constraints: We generate the constraints |[LTL]| capturing the requirements imposed by the LTL formula using the *eventuality encoding* as described in [37]. In the LTL constraints, we introduce $L+1$ fresh variables $l_0, \ldots, l_L$ which are used as loop selector variables. At most one of these variables l_k can be true and indicates that the loop starts at the k-th step of the trajectory.

3) Energy Constraints.: The energy available to the robot decreases with the execution of each motion primitive. Obviously, the robot has to recharge its battery for its continued operation. The following constraint ensures this:

$$\forall k \in \{0, \ldots, L\} : \; l_k \Rightarrow (\sigma_{L+1}.e \geq \sigma_k.e) \qquad \text{(III.1)}$$

The above constraint, denoted by |[ENERGY]|, ensures that the battery energy available to the robot at the end of the execution of the loop (at state σ_{L+1}) is greater than or equal to the battery energy available to the robot at the beginning of the execution of the loop (at state σ_k). This is possible only if the robot visits the charging station while executing the loop part of the trajectory and recharges its battery using the recharge primitive.

4) Full Encoding: The full encoding of the problem is denoted by $|[\mathcal{P}, L]|$, and is given by the conjunction of the above-mentioned constraints.

$$|[\mathcal{P}, L]| \Leftrightarrow |[SYS]| \wedge |[LTL]| \wedge |[ENERGY]| \qquad \text{(III.2)}$$

To solve the problem posed in Section II, we start with $L = 2$, and solve the conjunction of the set of constraints captured in $|[\mathcal{P}, L]|$. If there exists a solution for the set of constraints, the trajectories for the robots can be extracted from the solution. If no solution exists, we attempt to solve the constraints for a larger value of L.

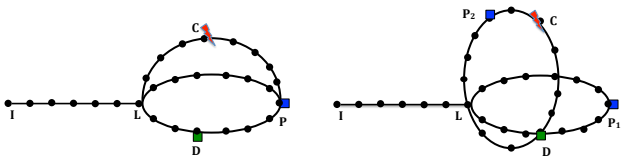
It is guaranteed that the loop part of a trajectory synthesized by the above methodology will traverse through a location $l_c \in S$ where the recharge primitive ν will be applied to the robot to charge its battery fully. Thus, the charging station can be placed at l_c.

The above algorithm works under the implicit assumption that the robot can traverse the loop part of the trajectory at least once with full battery charge.

B. Recharge-Optimal Trajectory Generation

The algorithm described in Section III-A synthesizes a trajectory that satisfies the functional requirement for the robot and ensures that the robot never gets depleted with battery charge. However, the generated trajectory is conservative —the robot may visit the charging station unnecessarily, i.e. even if it may be able to continue traversing multiple loops without visiting the charging station, it may still visit the charging station. To alleviate this problem, we devise Algorithm III.1 that generates a trajectory for the robot that makes minimal number of visits to the charging station. The algorithm involves two main steps for synthesizing the trajectory: in the first step, the trajectory η is synthesized without considering the energy constraints, and in the second step, the trajectory η_e is synthesized incorporating the energy constraints in the SMT problem and considering the loop starting point in η as the initial location. These two trajectories are then combined appropriately to generate the final trajectory that ensures the satisfaction of the given LTL formula, and the minimum number of visits to the charging station without ever getting depleted with battery power.

Let us illustrate Algorithm III.1 using an example. Suppose that the robot has to satisfy a functional specification $\Box(\Diamond(pick \wedge \Diamond drop))$, which requires the robot to pick an object from a location, drop the object in another location, and perform these two



(a) $\Box(\Diamond(p \wedge \Diamond d) \wedge en)$ (b) $\Box(\Diamond((p_1 \vee p_2) \wedge \Diamond d) \wedge en)$

Fig. 1. Trajectories for two specifications: p, p_1 and p_2 denote pickup locations, d denotes drop location and en denotes the energy proposition.

Algorithm III.1: Synthesis of optimal energy-safe trajectory for a robot

Input: $\mathcal{P}$: Input Problem Instance

Output: $traj$: Optimal trajectory satisfying the LTL specification and the energy constraint

function `synthesizeOptimalEnergySafeTraj` ($\mathcal{P}$)
begin
  $\eta := \langle \eta_{pre} \circ \eta_l \rangle \leftarrow$ `find_traj_without_energy_constraints`($\mathcal{P}$)
  $l_i \leftarrow$ `find_loop_initiation_location`(η)
  $\mathcal{P} \leftarrow \mathcal{P}(I = l_i)$
  $\eta_e \leftarrow$ `find_traj_with_energy_constraints`($\mathcal{P}$)
  if ($\eta \neq$ *NULL* && $\eta_e \neq$ *NULL*) **then**
    $e_{pre} \leftarrow$ `energy_consumption_in_prefix`(η_{pre})
    $e_l \leftarrow$ `energy_consumption_in_loop`(η_l)
    $e'_l \leftarrow$ `energy_consumption_in_loop`(η_e)
    $e_t \leftarrow$ `energy_needed_to_reach_charging_stn`(η_e)
    $e_{rem} \leftarrow e'_l - e_t$
    $k_1 \leftarrow \lfloor \frac{e_{max} - e_{pre} - e_t}{e_l} \rfloor$, $k_2 \leftarrow \lfloor \frac{e_{max} - e_{rem} - e_t}{e_l} \rfloor$
    $traj \leftarrow \eta_{pre} \circ \eta_l^{k_1} \circ (\eta_e \circ \eta_l^{k_2})^{\omega}$
    `return` $traj$
  else
    `return` NULL
  end
end

tasks repetitively. The algorithm first invokes the function `find_traj_without_energy_constraints` to generate a trajectory $\eta = \langle \eta_{pre} \circ \eta_l \rangle$ that satisfies the LTL specification and ignores the energy constraint. The synthesized trajectory is an (L, k)-loop where η_{pre} represents the prefix and η_l represents the loop. In Figure 1(a), initial location of the robot is denoted by I, the pickup location by P and the drop location by D. The location where the loop starts is denoted by L. Thus, the trajectory synthesized by `find_traj_without_energy_constraints` can be represented as: $\overline{IL} \circ (\overline{LP} \circ \overline{PD} \circ \overline{DL})^{\omega}$, where $\overline{IL}$ is the trajectory from the location I to the location L. The function `find_traj_without_energy_constraints` is implemented using the method presented in Section III-A, while excluding the energy constraint in (III.1) from the system of constraints in Equation III.2. Next, we find a trajectory using `find_traj_with_energy_constraints` function using the method presented in Section III-A and considering the loop point L as the initial location of the robot. In Figure 1(a), the location C denotes the location of the charging station that is synthesized by our algorithm. The trajectory η_e synthesized by `find_traj_with_energy_constraints` function can be represented as $(\overline{LC} \circ \overline{CP} \circ \overline{PD} \circ \overline{DL})^{\omega}$. Now, using the trajectories η and η_e, we generate the trajectory that visits the charging station for the minimal number of times in the following way. We compute the energy consumption in the prefix of the trajectory η (denoted by e_{pre}), the energy consumption in executing the loop for both η and η' (denoted by e_l and e'_l respectively), the energy consumption to reach the charging station from the loop point (denoted by e_t), and the energy consumption to complete the loop from the charging station (denoted by e_{rem}). In Figure 1(a), e_{pre}, e_l, e'_l, e_t and e_{rem} are the energy consumption to execute the trajectory fragment $\overline{IL}$, $\overline{LP} \circ \overline{PD} \circ \overline{DL}$, $\overline{LC} \circ \overline{CP} \circ \overline{PD} \circ \overline{DL}$, $\overline{LC}$ and $\overline{CP} \circ \overline{PD} \circ \overline{DL}$ respectively. In Algorithm III.1, k_1 denotes the number of times the loop $\overline{LP} \circ \overline{PD} \circ \overline{DL}$ will be traversed before visiting the charging station for the first time, and k_2 denotes the number of times the loop will be traversed between two consecutive visit of the charging station. Thus, the final trajectory is represented as $\eta_{pre} \circ \eta_l^{k_1} \circ (\eta_e \circ \eta_l^{k_2})^{\omega}$.

In the previous example, the loop in the trajectory synthesized without the energy constraint overlaps with the trajectory synthesized with the energy constraint. However, for several specifications, these two loops may not overlap. Nonetheless, Algorithm III.1 generates the correct trajectories for such specifications too. Let us consider the functional specification $\Box(\Diamond((pick_1 \vee pick_2) \wedge \Diamond drop)$. To satisfy this specification, it is enough if the robot visits one of the pickup locations repetitively. However, it is possible that in the optimal loop synthesized without the energy constraint, the robot visits the pickup location satisfying $pick_1$, whereas it visits the other pickup location in the loop synthesized with the energy constraints. The trajectories are shown in Figure 1(b).

The following theorems captures the optimality of the trajectory produced by Algorithm III.1.

Theorem 3.1: Optimal length of the trajectories with and without energy constraints: Given an input problem $\mathcal{P}$, Algorithm III.1 synthesizes optimal length trajectories with and without energy constraints.

Proof: In synthesizing η_l with `find_traj_without_energy_constraints` function, we start with a very small value (say 2) for the length of η for which it is guaranteed that a trajectory does not exist. In every iteration, we go on incrementing this value by one and checking its satisfiability of the constraints. Whenever for the first time the constraints are satisfiable for a value of the length of η, we synthesize η_l for that specific value. Thus, η_l is an optimal length trajectory. Similarly, we can prove that the trajectory with energy constraint, i.e η_e, is also an optimal length trajectory. ■

Theorem 3.2: Optimal number of visits to charging station: Given an input problem $\mathcal{P}$, Algorithm III.1 generates a trajectory that satisfies the functional LTL specification, ensures that the robot never gets depleted with charge and the robot visits the charging station for minimal number of times.

Proof: The proof of the above theorem follows from the observation that η_l and η_e are optimal-length trajectories satisfying the LTL formula without and with the energy constraint respectively; and the loop η_l is taken for maximum possible number of times (k_1 and k_2) before executing η_e.

This proves that the charging station is visited for a minimal number of times. ■

IV. Experiments

A. Experimental Setup

We have evaluated our planning algorithm through extensive simulation. We have implemented Algorithm III.1 in C++. The planner generates the execution plan for the robot using SMT solver Z3 [38]. All the experiments have been performed in a laptop with i7-6500U CPU @ 2.50GHz and 16 GB RAM.

1) Workspace: Three different types of workspaces have been used in our experiments —*Warehouse*, *Maze* and *Artificial floor* which are shown in figures 2(a), 2(b) and 2(c) respectively. We have used two different dimensions —17×17 and 27×27 —for the workspaces. The potential locations for the charging station include the grid cells adjacent to the outer walls of obstacles or on the inner boundary of the workspace, with the y-coordinate being in the range of [0,3]. For all the workspaces of size 17×17 and 27×27, the starting location of the robot is $(5, 0)$ and $(9, 0)$ respectively.

2) Robot Model.: We carry out experiments on three different kinds of robots. Their details are provided below.

Turtlebot. Turtlebot [39] is a widely used robot for academic research. The robot has four motion primitives - (i) *move forward*, (ii) *move backward*, (iii) *move left*, (iv) *move right*. The robot has a direction associated with it. If the robot wants to move towards a different direction, it has to first rotate appropriately and then move forward. We have not considered separate motion primitive for rotation. The primitives *move left* and *move right* include a rotation followed by a forward movement, thus have a larger energy cost than the *move forward* or *move backward* primitives.

Dubin's Vehicle. A Dubin's vehicle cannot make any sideways or reverse movement, it is capable of making a right turn (R), a left turn (L) or moving forward (F). We have considered four different headings (configurations) of the vehicle viz. *N*, *S*, *E* and *W*. For each heading, it can apply R, L or F primitive. Thus, it has a total of 12 motion primitives. Steering angle $\phi \in [-\pi/2, \pi/2]$ decides the amount of right or left turn. In practice, $|\phi_{max}| \ll \pi/2$. We decide the steering angle in a way that after executing the motion primitive, the car is at right angle to its previous configuration where the primitive was applied.

Quadcopter. For the quadcopter, we use the model of a NanoQuad quadrotors from KMel Robotics [40]. The motion primitives for the quadcopter are generated using the algorithm described in [41]. In 2-D, the velocity profile has 9 configurations consisting of one hover state and constant velocity in 8 uniform directions. Using the duration of $0.66sec$, the algorithm in [41] yields the set of 57 motion primitives.

3) LTL Specifications.: In our experiments, we consider a pick-and-drop application for a robot. In this application, a robot picks objects from a number of pickup locations and drops those objects in a drop location. Moreover, the robot performs its tasks repetitively. We consider three different variants of the problem.

Pickup in a user-given order. The robots should pick up objects from the pick up locations in a user given order. The LTL formula for the specification for two pickup locations is given below: $\square(\diamondsuit(pick1 \wedge \diamondsuit(pick2 \wedge \diamondsuit drop)))$.

Pickup in an arbitrary order. The robot is allowed to pick up objects from the pick up locations in any arbitrary order. The LTL formula for the specification for two pickup locations is given as: $\square(\diamondsuit pick1 \wedge \diamondsuit pick2 \wedge ((pick1 \vee pick2) \Rightarrow \diamondsuit drop))$.

Selective pickup. The robot should select one of the pickup locations and drop the picked up object to the drop location. The following LTL formula captures this specification: $\square((\diamondsuit pick1 \vee \diamondsuit pick2) \wedge ((pick1 \vee pick2) \Rightarrow \diamondsuit drop))$.

B. Results

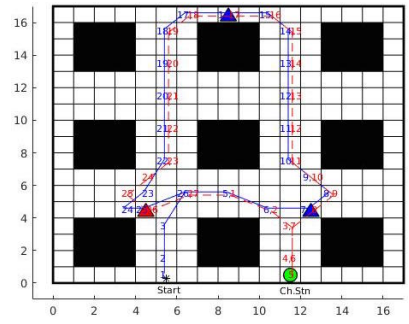
In Figure 2, we show some sample outputs pictorially in different workspaces of size (17×17), with two pickup locations and a drop location. The blue solid lines in the figures indicate a trajectory followed by the robot when it has enough amount of charge to complete the loop. On the other hand, if the robot does not have enough battery charge to complete the blue-coloured loop, it must visit a charging station. The location of the charging station, indicated by a green circle, is also synthesized by our algorithm. As loop starting point is same for both blue and red loops, and now the robot has enough charge after getting recharged, it can resume traversing the blue-coloured trajectory (which does not visit the charging station). Note that the red coloured trajectory also fulfills the designated task of visiting the pickup and drop locations as specified by the LTL property.

In fig. 2(a), a Warehouse workspace is shown where two pickup locations are at $P_1^w = (12, 4)$ and $P_2^w = (8, 16)$ and the drop location is at $D^w = (4, 4)$ grid positions. A Quadcopter is asked to perform a task of picking up in a *given order* which is "Pick P_1^w before P_2^w, and then visit D^w" in the workspace. The figure shows that the robot performs as expected.

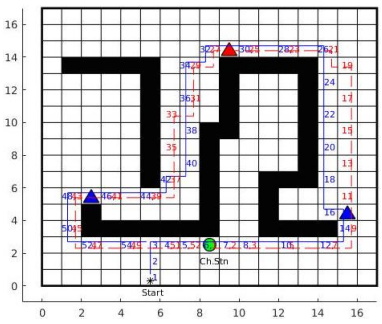
Figure 2(b) shows a Maze workspace with a Turtlebot robot. Two pickup locations are at $P_1^m = (2, 5)$ and $P_2^m = (15, 4)$, and drop location is at $D^m = (9, 14)$ grid locations. Here, it is specified that the robot can visit "P_1^m and P_2^m in any order, and then visit D^m". In this case, the charging station location is synthesized at $(8, 2)$, which is incident on the blue-coloured (and red) trajectory.

The above two examples in Figure 2(a) and Figure 2(b) are as per the illustration shown in Figure 1(a). Now, with an Artificial floor workspace and a Dubins vehicle, in Figure 2(c) we show an example of the scenario shown in Figure 1(b). Two pickup and one drop locations are at $P_1^a = (2, 6)$, $P_2^a = (15, 5)$ and $D^a = (6, 11)$ respectively. The specification is "visit any of P_1^a or P_2^a, and then visit D^a". Charging station is synthesized at $(15, 3)$. When the robot has sufficient charge to visit any one of P_1^a and P_2^a followed by D^a, it chooses to visit P_1^a (blue-coloured trajectory). On the other hand, when the robot lacks battery power and it has to visit the charging station, it chooses to visit P_2^a.

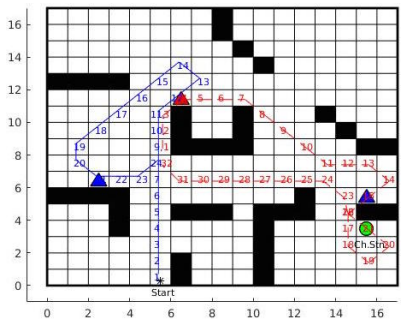
In Table I, Table II and Table III, we present the computation time of our algorithm for different workspaces, robots and specifications. The symbol τ denotes the time to synthesize η (the blue-coloured trajectory), and τ_e denotes the additional time required to synthesize η_e (the red-coloured trajectory). So, essentially it takes $\tau + \tau_e$ time generate the



(a) Trajectory : Robot - Quadcopter, Workspace - Warehouse, Specification - Ordered



(b) Trajectory : Robot - Turtlebot, Workspace - Maze, Specification - Unordered



(c) Trajectory : Robot - Dubin's vehicle, Workspace - Art. floor, Specification - Selective

Fig. 2. The trajectories —*blue*-colored trajectory: Path not visiting the charging station; *red*-colored trajectory: Path which visits the charging station. From the starting location upto the loop point both trajectories are same, so red-colored trajectory has been omitted.

Workspace	Specification	Robot	τ	τ_e	λ	λ_e	l_c
Warehouse	Ordered	Dubins	27s	2s	37	38	(10 2)
		Turtle	30s	13s	39	41	(6 3)
		Quad	49s	56s	21	26	(11 0)
	Unordered	Dubins	25s	3s	37	38	(12 3)
		Turtle	28s	16s	39	41	(12 3)
		Quad	1m 22s	1m 40s	21	25	(8 0)
	Selective	Dubins	9s	8s	27	31	(4 3)
		Turtle	10s	6s	31	33	(4 3)
		Quad	39s	49s	18	23	(10 0)
Maze	Ordered	Dubins	4m 8s	6s	43	44	(2 2)
		Turtle	7m 0s	14s	51	52	(5 2)
		Quad	14m 54s	28s	26	27	(15 3)
	Unordered	Dubins	5m 32s	18s	43	44	(3 2)
		Turtle	8m 8s	14s	51	52	(8 2)
		Quad	21m 1s	30s	26	27	(2 2)
	Selective	Dubins	3m 22s	46s	23	33	(16 3)
		Turtle	2m 27s	8s	25	29	(15 3)
		Quad	7m 27s	14s	15	18	(15 3)
Artificial Floor	Ordered	Dubins	1m 6s	4s	37	38	(8 3)
		Turtle	4m 15s	8m 45s	43	49	(8 3)
	Unordered	Dubins	1m 10s	6s	37	38	(6 3)
		Turtle	4m 23s	9m 2s	43	49	(4 3)
	Selective	Dubins	37s	18s	21	29	(16 2)
		Turtle	2m 47s	28s	25	31	(15 3)

TABLE I

EXPERIMENTAL RESULTS FOR DIFFERENT WORKSPACES OF SIZE 17×17, DIFFERENT LTL SPECIFICATIONS WITH TWO PICKUP LOCATIONS AND DIFFERENT ROBOTS

full trajectory satisfying both the LTL specification and the energy constraint. The symbol λ denotes the length of loop η_l of η, whereas λ_e is that of η_e. The length of η_e is greater than or equal to that of η_l. Therefore, for synthesizing the loop with energy constraint, we start with $L = \lambda$ instead of $L = 2$. It can be noticed that the value of τ_e is directly proportional to the length difference between λ_e and λ, i.e., $\lambda_e - \lambda$. Charging station location synthesized by our algorithm is denoted by l_c.

In Table I, experimental results for different workspaces of size 17×17, different LTL specifications and robots are shown. We have considered two pickup and one drop location in these experiments. Results in Table II show the scalability of our algorithm in terms of workspace size. This table contains results for Dubins vehicle in different workspaces of size 27×27 and for different LTL specifications with two pickup and one drop location. Scalability with respect to number of pickup locations is evident from Table III. Results are shown for varying number of pickup locations with different robots in a Warehouse workspace (17×17). It can be seen that trajectory lengths λ and λ_e increases monotonically with increasing number of pickup locations. As a result, computation time also increases with the increasing number of pickup locations.

We carry out simulation of the trajectories synthesized by our algorithm in ROS for a Turtlebot for three different workspaces and three different LTL specifications. A video of the simulation is submitted.

Workspace	Specification	τ	τ_e	λ	λ_e	l_c
Warehouse	Ordered	2m 18s	6s	51	52	(16 3)
	Unordered	2m 23s	3s	51	52	(10 3)
	Selective	1m 17s	3m 24s	39	51	(8 3)
Maze	Ordered	39m 25s	2m 5s	67	68	(8 0)
	Unordered	39m 8s	2m 12s	67	68	(15 0)
	Selective	27m 20s	0m 23s	51	52	(8 0)
Artificial Floor	Ordered	9m 35s	8m 22s	57	63	(10 3)
	Unordered	5m 7s	5m 38s	55	61	(16 3)
	Selective	2m 54s	2m 9s	31	45	(18 3)

TABLE II

EXPERIMENTAL RESULTS FOR DUBINS VEHICLE FOR DIFFERENT WORKSPACES OF SIZE 27×27 AND DIFFERENT LTL SPECIFICATIONS WITH TWO PICKUP LOCATIONS

Robot	# Pickup Locs	τ	τ_e	λ	λ_e	l_c
Dubins	1	9s	7s	27	31	(4 3)
	2	25s	1s	37	38	(10 1)
	3	36s	1s	39	40	(7 0)
	4	39s	2s	41	42	(12 3)
	5	1m 57s	23s	47	49	(12 3)
Turtle	1	9s	5s	31	33	(6 3)
	2	34s	17s	39	41	(12 3)
	3	51s	15s	43	45	(10 3)
	4	1m 49s	35s	49	51	(4 3)
	5	5m 12s	1m 17s	59	61	(4 3)
Quad	1	35s	42s	18	23	(5 0)
	2	1m 22s	1m 40s	21	25	(8 0)
	3	11m 37s	9m 27s	25	29	(6 0)
	4	48m 49s	17m 30s	29	33	(6 0)
	5	1h 42m	41m 46s	34	38	(8 0)

TABLE III

EXPERIMENTAL RESULTS FOR DIFFERENT ROBOTS WITH VARYING NUMBER OF PICKUP LOCATIONS IN A WAREHOUSE WORKSPACE WITH SIZE 17×17, AND LTL SPECIFICATION *"pickup in an arbitrary order"*

V. DISCUSSION

We present a methodology for the automatic synthesis of an execution plan for a robot from any LTL specification in such a way that the robot never gets depleted with energy while executing the plan. We have applied our technique to a set of case studies related to pick and drop use cases and have successfully synthesized trajectories for robots with various dynamics in different workspaces. Our future work will focus on extending our technique to synthesize energy-safe reactive plans to deal with dynamic obstacles and uncertainly in the environment and support multi-robot systems.

REFERENCES

[1] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Where's Waldo? Sensor-based temporal logic motion planning," in *ICRA*, 2007, pp. 3116–3121.

[2] S. Karaman and E. Frazzoli, "Sampling-based motion planning with deterministic μ-calculus specifications," in *CDC*, 2009, pp. 2222–2229.

[3] A. Bhatia, L. E. Kavraki, and M. Y. Vardi, "Motion planning with hybrid dynamics and temporal goals," in *CDC*, 2010, pp. 1108–1115.

[4] T. Wongpiromsarn, U. Topcu, and R. M. Murray, "Receding horizon temporal logic planning," *IEEE Trans. Automat. Contr.*, vol. 57, no. 11, pp. 2817–2830, 2012.

[5] Y. Chen, J. Tumova, and C. Belta, "LTL robot motion control based on automata learning of environmental dynamics," in *ICRA*, 2012, pp. 5177–5182.

[6] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *I. J. Robotic Res.*, vol. 32, no. 8, pp. 889–911, 2013.

[7] 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.

[8] D. Halperin, J.-C. Latombe, and R. H. Wilson, "A general framework for assembly planning: The motion space approach," in *Annual Symposium on Computational Geometry*, 1998, pp. 9–18.

[9] S. Rodríguez and N. M. Amato, "Behavior-based evacuation planning," in *ICRA*, 2010, pp. 350–355.

[10] J. S. Jennings, G. Whelan, and W. F. Evans, "Cooperative search and rescue with a team of mobile robots," in *ICRA*, 1997, pp. 193–200.

[11] D. Fox, W. Burgard, H. Kruppa, and S. Thrun, "A probabilistic approach to collaborative multi-robot localization," *Autonomous Robots*, vol. 8, no. 3, pp. 325–344, 2000.

[12] D. Rus, B. Donald, and J. Jennings, "Moving furniture with teams of autonomous robots," in *IROS*, 1995, pp. 235–242.

[13] T. Balch and R. Arkin, "Behavior-based formation control for multi-robot teams," *IEEE Transaction on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, 1998.

[14] M. C. Silverman, B. Jung, D. Nies, and G. S. Sukhatme, "Staying alive longer: Autonomous robot recharging put to the test," Tech. Rep., 2003.

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

[16] B. Kannan, V. Marmol, J. Bourne, and M. B. Dias, "The autonomous recharging problem: Formulation and a market-based solution," in *ICRA*, 2013, pp. 3503–3510.

[17] G. P. Strimel and M. M. Veloso, "Coverage planning with finite resources," in *IROS*, 2014, pp. 2950–2956.

[18] J. Yu, J. Aslam, S. Karaman, and D. Rus, "Anytime planning of optimal schedules for a mobile sensing robot," in *IROS*, 2015, pp. 5279–5286.

[19] S. Mishra, S. Rodriguez, M. Morales, and N. M. Amato, "Battery-constrained coverage," in *CASE*, 2016, pp. 695–700.

[20] I. Shnaps and E. Rimon, "Online coverage of planar environments by a battery powered autonomous mobile robot," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 425–436, 2016.

[21] S. Kim, S. Bhattacharya, and V. Kumar, "Path planning for a tethered mobile robot," in *ICRA*, 2014, pp. 1132–1139.

[22] P. H. Borgstrom, A. Singh, B. L. Jordan, G. S. Sukhatme, M. A. Batalin, and W. J. Kaiser, "Energy based path planning for a novel cabled robotic system," in *IROS*, 2008, pp. 1745–1751.

[23] N. Xu, P. Braß, and I. Vigan, "An improved algorithm in shortest path planning for a tethered robot," Tech. Rep., 2012.

[24] S. McCammon and G. A. Hollinger, "Planning and executing optimal non-entangling paths for tethered underwater vehicles," in *ICRA*, 2017, pp. 3040–3046.

[25] I. Kelly, O. Holland, and C. Melhuish, "Slugbot: A robotic predator in the natural world," in *In Proc. 5th International Symposium on Artificial Life and Robotics*, 2000, pp. 470–475.

[26] D. Wettergreen, P. Tompkins, C. Urmson, M. Wagner, and W. Whittaker, "Sun-synchronous robotic exploration: Technical description and field experimentation," *IJRR*, vol. 24, no. 1, pp. 3–30, 2005.

[27] M. Lahijanian, M. Svorenova, A. A. Morye, B. Yeomans, D. Rao, I. Posner, P. Newman, H. Kress-Gazit, and M. Kwiatkowska, "Resource-performance tradeoff analysis for mobile robots," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1840–1847, 2018.

[28] T. Kundu and I. Saha, "Charging station placement for indoor robotic applications," in *ICRA*, 2018.

[29] 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.

[30] 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.

[31] 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.

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

[33] I. Saha, R. Ramaithitima, V. Kumar, G. J. Pappas, and S. A. Seshia, "Implan: Scalable incremental motion planning for multi-robot systems," in *ICCPS*, 2016, pp. 43:1–43:10.

[34] 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.

[35] I. Gavran, R. Majumdar, and I. Saha, "Antlab: A multi-robot task server," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5, pp. 190:1–190:19, 2017.

[36] A. Pnueli, "The temporal logic of programs," in *FOCS*, 1977, pp. 46–57.

[37] A. Biere, K. Heljanko, T. Junttila, T. latvala, and V. Schuppan, "Linear encoding of bounded LTL model checking," *Logical Methods in Computer Science*, vol. 2, no. 5:5, pp. 1–64, 2006.

[38] 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.

[39] "TurtleBot," http://www.turtlebot.com.

[40] "KMel robotics," http://kmelrobotics.com/.

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