Mind the gap: Robotic Mission Planning Meets Software Engineering

Mehrnoosh Askarpour
mehrnoosh.askarpour@polimi.it
Politecnico di Milano
Milan, Italy

Claudio Menghi
claudio.menghi@uni.lu
University of Luxembourg
Luxembourg, Luxembourg

Gabriele Belli
gabriele.belli@alten.it
Alten
Torino, Italy

Marcello M. Bersani
marcellomaria.bersani@polimi.it
Politecnico di Milano
Milan, Italy

Patrizio Pelliccione
patrizio.pelliccione@univaq.it
Chalmers | University of Gothenburg
and University of L'Aquila

ABSTRACT

In the context of robotic software, the selection of an appropriate planner is one of the most crucial software engineering decisions. Robot planners aim at computing plans (i.e., blueprint of actions) to accomplish a complex mission. While many planners have been proposed in the robotics literature, they are usually evaluated on showcase examples, making hard to understand whether they can be effectively (re)used for realising complex missions, with heterogeneous robots, and in real-world scenarios.

In this paper we propose ENFORCE, a framework which allows wrapping FM-based planners into comprehensive software engineering tools, and considers complex robotic missions. ENFORCE relies on (i) realistic maps (e.g, fire escape maps) that describe the environment in which the robots are deployed; (ii) temporal logic for mission specification; and (iii) Uppaal model checker to compute plans that satisfy mission specifications. We evaluated ENFORCE by analyzing how it supports computing plans in real case scenarios, and by evaluating the generated plans in simulated and real environments. The results show that while ENFORCE is adequate for handling single-robot applications, the state explosion still represents a major barrier for reusing existing planners in multi-robot applications.

KEYWORDS

Planning, Robotics, Formal Methods, Timed Automaton, Temporal Logic, Model Checking, Uppaal

ACM Reference Format:
Mehrnoosh Askarpour, Claudio Menghi, Gabriele Belli, Marcello M. Bersani, and Patrizio Pelliccione. 2020. Mind the gap: Robotic Mission Planning Meets Software Engineering. In *8th International Conference on Formal Methods in Software Engineering (FormaliSE '20), October 7–8, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3372020.3391561

FormaliSE '20, October 7–8, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7071-4/20/05…$15.00
https://doi.org/10.1145/3372020.3391561

1 INTRODUCTION

Robotic software engineering concerns the development of techniques that enable a systematic and rigorous development of robotic software [11]. As classical software, robotic software is not monolithic, as it is usually obtained by assembling already existing components, as well as developing brand new ones (when needed). The lack of systematic and rigorous techniques, which promote reuse of components and facilitate their integration, has been identified as one of the major challenges in the robotic domain [31].

Formal methods are mathematical approaches to software and system development, which support rigorous specification, design and verification [20]. These approaches have been largely used in the robotic domain [22], and one of their major applications is mission planning, which is the topic of this work. Given a high level defined goal, called *mission* [40], e.g., "robot A goes to position p, brings a box, goes to position q, waits 10 seconds and, finally, reaches position r and releases the box", robotic mission planning aims at computing a set of actions that, if performed, ensure the accomplishment of the goal.

Many FM-based planners have been used in the robotic domain (e.g., [21, 27, 48, 50]). However, planners are usually evaluated on showcase simple examples, have limited usage assumptions (often not explicitly documented), and their scalability properties are usually not thoroughly evaluated. This restricts the usage of these planners in industrial contexts, where there is an increasing need for Integrated Formal Methods (iFMs) [22, 38]. Integrated Formal Methods refer to the integration of multiple formal methods, or/and semi- or non-formal approaches, that complement each other. Following this line of thought, integrating mission planners within SE-based robotic frameworks requires to precisely understand when and how planners can be reused. Thus, there is a need for robotic mission planning to meet software engineering.

The goal of the paper is to perform a preliminary step to fill the gap between robotic mission planning and software engineering. To this end, we first identify a set of features that robotic mission planners should possess to address complex problems coming from the industrial domain. Specifically, these features refer to several aspects of the application, namely, the mission to be accomplished by the robots (**F1**), the robots capabilities (**F2**), the team of robot(s) (**F3**), and the environment in which the robots are deployed (**F4**).

We propose *formally vErified plaNning soFtware fOr Real-world sCEnarios* (ENFORCE), an approach for integrating FM-based planners and comprehensive software engineering tools, that allows designer to deal with mission planning for complex robotic scenarios. ENFORCE supports a systematic and rigorous design workflow, fosters the reuse of already implemented third-party components, and improves maintainability, as it promotes separation of concerns by modeling several aspects of the robotic application (i.e., robots, environment and mission) with distinct artifacts. Specifically, ENFORCE makes use of two distinct formal models to describe independently how robots perform actions and move within their environments, and the map that describes the environment in which the robots are deployed. The mission the (team of) robot(s) should achieve is specified by means of a temporal logic formula, written in terms of robot actions and positions of the environment. All these artifacts are automatically translated into the input language of a model checker, by means of a formally-defined translation. ENFORCE can leverage off-the-shelf model checkers (provided they enable mission planning) to determine (i.e., synthesize) the existence of a trace that satisfies the mission specification, and that can be used to derive the motion plans for the robots operating in the environment.

To implement our approach, we use the Uppaal [37] model checking tool, since it can be used to realize mission planning that features **F1**-**F4**. We rely on existing robotic techniques [8, 44] to represent the area in which the robots are operating as a tiling of the plane with squared cells. The area is modeled, with various precision levels, using Timed Automata (TA), i.e., the input modeling formalism of Uppaal. Locations of TA are used to capture both the cells of the map and the dynamics of the robots. The mission is specified in quantitative temporal logic (i.e., TCTL).

For the sake of simplicity, in this work, we assume that all robots are controllable, i.e., uncontrollable agents are not operating in the environment. Despite this simplification, we encounter different hurdles that hamper an effective integration of existing FM-based planning techniques in SE workflows.

To evaluate our approach, we assess the effectiveness and correctness of the overall synthesis procedure implemented in ENFORCE. The effectiveness is evaluated by checking whether ENFORCE synthesizes plans in a reasonable time, both in the case of single-robot (**RQ1**) and in the case of multi-robots (**RQ2**) applications, when maps of real environments are considered. Correctness is evaluated by checking whether the computed plans allow robots to achieve the specified mission. This is performed both by evaluating the behavior of the robots in simulated environments (**RQ3**) and in real environments (**RQ4**). Our results show that ENFORCE can effectively synthesize correct plans, in reasonable time, that ensure the mission satisfaction for simulated and real environments, with realistic size, and single-robot applications. Conversely, ENFORCE is not able to compute a plan in a matter of minutes for multi-robots applications deployed in realistic scenarios. We critically analyze and discuss our results.

Integrating existing planning techniques within SE workflows is not naive, yet requires an in-depth knowledge of the planner features and its scalability, especially when the number of robots increases. We believe that our thorough empirical evaluation paves the way for a more critical analysis of the features of existing FM-based planning techniques to promote their reuse within SE frameworks.

Table 1: Number of papers and venues considered in the feature collection

Venue	Acronym	#Paper
Formal Methods	FM	1
Conference on Formal Methods in Software Engineering	FormaliSE	3
Conference on Robotics and Automation	ICRA	2
Symposium on Intelligent Autonomous Vehicles		1
Conference on Intelligent RObots and Systems	IROS	3
Conference on Networking, Sensing and Control	ICNSC	1
Conference on decision and control	CDC	3
Journal of Systems and Control		1
Transactions on Robotics	T-RO	1
American Control Conference	ACC	3
European Control Conference	ECC	1
Journal of Automatica		2
Journal of China Information Sciences*		1

Structure. Sec. 2 describes the features of the planner we are considering in this work. Sec. 3 introduces the basic notions on Timed Automata (TA) and quantitative temporal logic (TCTL). Sec. 4 presents ENFORCE. Sec. 5 shows our experimental results and Sec. 6 discusses our findings. Finally, Sec. 7 concludes.

2 COLLECTION OF THE PLANNING FEATURES

We are considering mobile platforms and static/mobile manipulators, since they are broadly used in the industrial domain [4, 5], they are regulated by ISO standards [1, 2], and can work together with or under supervision of human operators [3]. In order to make our tool applicable on various available industrial robotic systems, we performed a thorough state-of-the-art analysis on robot motion/mission planners. We had collected 24 different works focusing on FM-based planners, that were presented in 14 different venues (see Table 1). These works were collected by considering the knowledge of the authors in the field, complemented by a snowballing literature review. We reviewed these papers, and we identified recurrent features of planners.

Based on the results of our review and a discussion with our industrial partner, we identified four main features (**F1**, **F2**, **F3**, and **F4**), discussed below.

F1 - Expressing explicit time concerns in robotic missions. Planners should analyze missions containing explicit time constraints. For example, the mission "visit region A, then B *within 10 seconds*" forces a robot to *first* visit A and *then* B within an explicit time constraint, i.e., within 10 seconds.

Logic-based languages are broadly used in the literature for specifying the mission that robots should achieve [22, 38]. The column *Log* and *Tp* of Table 2 report, respectively, the logic that has been used to specify missions for each paper, and whether explicit time concerns have been specified.

LTL has been extensively used to express a rich variety of behavior including robot missions [28, 32, 39, 40, 42]. It also benefits from a consolidated knowledge containing algorithms for verification and synthesis of controllers. However, it lacks a metric notion of time, and it is not able to express bounds on the delay between events. In LTL, one can express that an event B follows an event A,

Table 2: Analysis of related work on planning. *Log*: the used logic; *Tp*: explicit temporal concerns; *MA*: multi-agent teams; *Act*: actions rather than motion; *Sync*: team synchronization; *Mp*: map of realistic environment; *Exp*: real experiments; Tool: the used verification tool.

	F1		F2	F3		F4		
Ref.	*Log*	*Tp*	*Act*	*MA*	*Sync*	*Mp*	*Exp*	Tool
[30, 44]	TCTL	✓	✗	✓	✗	✗	✗	Uppaal
[8]	TCTL	✗	✗	✓	✗	✗	✓	Uppaal
[35]	CTL	✗	✗	✓	✗	✗	✓	C-SMV
[49]	LTL	✗	✗	✗	✗	✗	✓	✗
[27]	LTL	✗	✗	✗	✗	✗	✗	✗
[46]	LTL	✗	✓	✗	✗	✗	✓	✗
[32]	LTL	✗	✗	✓	✗	✗	✓	✗
[12]	LTL	✗	✓	✓	✓	✗	✓	✗
[33]	LTL	✗	✓	✓	✓	✗	✗	✗
[34]	LTL	✗	✓	✓	✓	✓	✓	✗
[47]	LTL	✗	✗	✓	✗	✗	✗	✗
[29]	LTL	✗	✓	✓	✓	✗	✗	✗
[21]	LTL	✗	✓	✓	✓	✓	✗	Matlab
[25, 28, 39, 48]	LTL	✗	✓	✓	✓	✗	✗	Matlab
[13, 26]	LTL	✗	✓	✗	✗	✗	✗	Matlab
[43]	MTL	✓	✓	✓	✓	✗	✓	✗
[52]	MTL	✓	✓	✗	✗	✗	✗	CPLEX
[42]	MTL	✓	✓	✓	✓	✗	✗	Matlab

but it is not possible to limit the time interval between the two events by imposing, for instance, that the delay is smaller than 5 time units. Therefore, a number of authors [8, 43, 44] have recently considered formalisms that allow for expressing explicit time constraints, such as Metric Temporal Logic (MTL) [36], or Timed Computation Tree Logic (TCTL) [6]. However, some of these works are mostly theoretical, some of them do not explicitly consider how robots synchronize and perform actions, and usually they are not validated on real maps but consider small environment abstractions (usually represented through small-size matrices of cells representing locations of the environment).

F2 - Replicating functionalities and actions of a robot. Industrial robots are used for many different activities, commonly realized by means of arms, or end effectors. Thus, planners have to consider not only how robots move in their environment, but also other types of functionalities, such as pick and place, grab, welding, assembly. Column *Act* in Table 2 shows if a paper covers planning by considering functionalities of robot other than movements.

F3 - Managing multi-robots and their action synchronization. Planners should be able to manage both single-robot and multi-robot systems. A robotic system could consist of multiple robots, which potentially collaborate, or compete, for achieving a given mission. Thus, their interaction, collaboration, and synchronization must be considered for planning their actions.

The analysis of the behavior of a team of robots, and the synthesis of the motion plans that regulate their movements over a bi-dimensional area have been studied in the last decade [32, 34, 43, 45]. In Table 2, column *MA* indicates if a paper supports multi-robots, and column *Sync* specifies if they collaborate to achieve a goal together, and need to synchronize their actions.

F4 - Considering realistic environments. Planners should be able to perform on realistic environments. Most of the planners in the literature consider abstractions of real environments that are represented by grid of cells, usually of very limited size, e.g., [44]. Thus, it is difficult to evaluate how the algorithms scale when the size of the environment (and of the corresponding grid of cells) grows, such as when planning must be performed on real buildings. In Table 2, the ✓symbol in column *Mp* indicates that the planning procedure has been applied on maps representing realistic buildings.

The majority of the planners, except for [21, 34], are only evaluated through simulation, and no experiments in real environments have been performed. In Table 2, column *Exp* indicates if an experimental evaluation in a real scenario has been performed, and algorithms are deployed on real robots.

Most of the planners are developed by means of ad-hoc solutions, rather than built on pre-existing solutions with proven effectiveness. The use of consolidated tools has many advantages. In many cases the implemented procedure is stable and efficient, as the tools include optimizations that work at the engine level. Moreover, consolidated tools might offer different options to the user for the analysis of the system, such as, for instance, the state space exploration policy (either breadth first or depth first). The user can use off-the-shelf tools without the need of implementing ad-hoc solutions on specific test cases. In the last column of Table 2, the ✗ symbol marks the works which used ad-hoc solutions. Otherwise, we reported the name of the tool used for planning.

Related Work. The robotic mission planning problem has been widely explored in the literature and in the FM community (see Table 1 and Table 2). Some of these works considered the problem of decision making and task planning as a two-player temporal logic game between the planner component and its environment [51], other focused on planning multi-robot systems and collision avoidance [9, 16, 30]. From a technological perspective, several solutions have been proposed, such as the use of Satisfiability Modulo Theories to verify the fulfilment of defined missions [15] or model checkers [18]. Planning robotic systems has also been analysed by considering other characteristics, spanning from hardware features [23, 23] to the provision of support to people with disabilities [14]. However, while these works provide FM-solutions for specific problems, less attention is usually given to making the solutions reusable, and to their evaluation from a SE perspective.

Some works have also been done to help users in engineering robotic applications. RoboChart [41] provides a language for modelling robotic applications. It allows user to verify the models of the robotic applications by reusing existing model checkers and theorem provers. RobotML [17] is a robotic modeling language provided as a Papyrus plugin, that enables the design of robotic applications, their simulation, and their deployment to multiple target execution platforms. FLYAQ [10] is a tool for defining missions of teams of multicopters. It enables the automatic generation of the detailed flight plan the multicopters have to follow.

3 BACKGROUND

This section recalls the definition of Timed Automata (TA) [7] and TCTL, which are the formalisms used in the rest of this work.

Table 3: The notation used in definition of a TA. c is a natural number, d is an integer, and $\sim \in \{<, =\}$.

Notation	Definition
X	finite set of clocks with real values
Y	finite set of integer variables
Act	finite set of actions
$\eta := x \sim c \mid \neg\eta \mid \eta \wedge \eta$	clock constraints ($x \in X$)
$\Gamma(X)$	set of clock constraints
$\zeta := y \sim d \mid y \sim y' \mid \neg\zeta \mid \zeta \wedge \zeta$	variable constraints ($y, y' \in Y$)
$\Gamma(Y)$	set of variable constraints
$assign(Y) := \{y := d \mid y \in Y\}$	set of assignments

Timed Automata. Given the notation introduced in Table 3, a *timed automaton* (TA) is defined by a tuple $\langle Q, q_0, v^0, I, T\rangle$, where

(1) Q is a finite set of locations,
(2) $q_0 \in Q$ is the initial location,
(3) $v^0 : Y \to \mathbb{N}$ is a function assigning each variable in Y with an integer value,
(4) $I : Q \to \Gamma(X)$ is an invariant assignment function, and
(5) $T \subseteq Q \times Q \times \Gamma(X) \times \Gamma(Y) \times Sync \times \wp(X) \times \wp(assign(Y))$ is a finite set of transitions such that $Sync = Act \times \{!, ?\}$.

The *configuration* of a TA is denoted by a pair (q, v), where $q \in Q$ is the current location of the automaton, and v is a function over $X \cup Y$ that assigns a non-negative real value to every clock of X and an integer to every variable of Y.

A configuration change $(q, v) \to (q', v')$ changes the configuration of the TA from (q, v) to (q', v') ,and occurs due to either a transition in T (*discrete transition*), or time elapsing (*time transition*).

When a discrete transition $(q, q', \sigma, \eta, \zeta, S, A) \in T$ is fired:

(1) the clock and the variable values in v satisfy, respectively, guards η and ζ, and v' satisfies the invariant $I(q')$;
(2) for each clock x, if x is in S, then it holds that $v'(x) = 0$, otherwise $v'(x) = v(x)$; and
(3) for each variable $y \in Y$, it holds that $v'(y) = d$ and $y := d$ is an assignment in A.

When a time transition is fired:

(1) location does not change $q = q'$;
(2) each variables $y \in Y$ retains its value, and $v'(x) = v(x) + \delta$, with $\delta \in \mathbb{R}_{\geq 0}$, for all $x \in X$; and
(3) invariant $I(q)$ is satisfied by all assignments of the clocks from v to v'.

A *run* or *execution* of a TA is a (possibly infinite) sequence of configurations $(q_0, v_0)(q_1, v_1)(q_2, v_2) \cdots$ such that, for any $i \geq 0$, $(q_i, v_i) \to (q_{i+1}, v_{i+1})$ is a discrete transition or a time transition. The set of all the executions of a TA A is indicated with $R(A)$.

When several TAs are considered, the configuration contains locations of all of them and the values of all their clocks and variables. The symbols in Σ are used to constrain the executions of the TA—i.e., the ways in which a network of TA synchronize while changing the configuration of the system. Each symbol $\sigma \in \Sigma$ that labels a transition has the form $\sigma?$ or $\sigma!$. Informally, two TAs synchronize when they simultaneously perform a discrete transition respectively labeled with $\sigma?$ and $\sigma!$.

For example, Figures 1 shows a network of TA representing a mobile robot with two components, i.e., a moving platform and a gripper, that is used in an industrial environment to move around and collect objects from a number of object storages which contain. The robot is able to load objects in a mounted bin (action ro_{load}) and move (action ro_{move}) within its environment.

The model of the moving platform described in Figure 1a shows that the robot alternates between "moving" and "still". In particular, the robot traverses the distance between its starting point and reaches the first storage (moving) and then stops for loading (still). The same thing is repeated from the first storage to the next one until all of the storages are met. This alternation is regulated by the values assigned to the clock x which constraints the duration of moving. The clock x constraint the robot movement to last at most ten time units. As the value of x reaches ten, the robot stops moving and x is reset.

We assume that a group of storages contain only five objects and the rest contain ten objects. The model of the gripper described in Figure 1b can alternate among three different states: "idle", when no object have to be loaded, "loading1", when five objects ($n = 5$) have to be loaded, and "loading2", when ten objects ($n = 10$) have to be loaded. Loading five objects requires 10 seconds, while loading ten objects requires 17 seconds. The robot is not supposed to move unless the loading action is completed (*idle*).

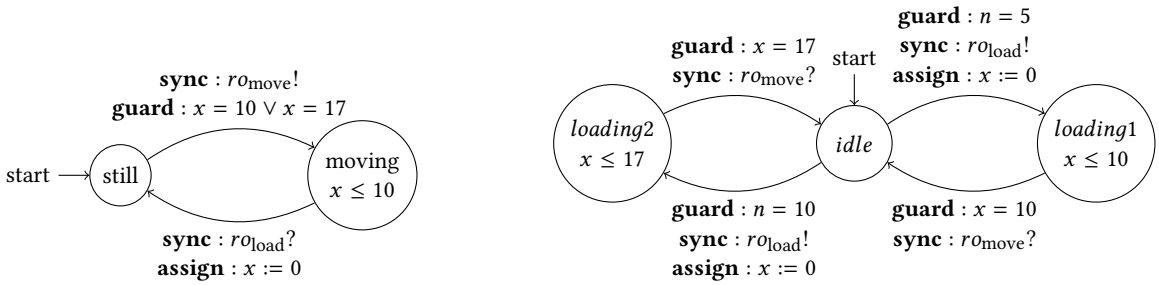
The synchronization of the two TAs in Figure 1 forces the mobile platform to become still before the gripper starts loading. Similarly, the gripper should reach the state "idle" before the mobile platform starts moving.

TCTL. Uppaal allows for specifying missions through an extension of the CTL logic, which also contains "time related" constraints (TCTL). Let ϕ be a boolean combination of formulae on variables and clocks such as, for instance, $x \leq 10 \wedge loading1$ indicating that clock x is less than or equal to 10 and that TA of Figure 1b is in location "$loading1$". TCTL allows the specification of properties in the form $\forall\mathcal{G}\,\phi$, $\forall\mathcal{F}\,\phi$, $\exists\mathcal{G}\,\phi$ and $\exists\mathcal{F}\,\phi$ whose semantics is defined as follows:

$\forall\mathcal{G}\,\phi \quad \Leftrightarrow$ for every execution, ϕ holds globally
$\forall\mathcal{F}\,\phi \quad \Leftrightarrow$ for every execution, ϕ holds eventually
$\exists\mathcal{G}\,\phi \quad \Leftrightarrow$ exist a path such that ϕ holds globally
$\exists\mathcal{F}\,\phi \quad \Leftrightarrow$ exist a path such that ϕ holds eventually.

4 FORMALLY VERIFIED PLANNING SOFTWARE FOR REAL-WORLD SCENARIOS

An overview of the ENFORCE framework is presented in Figure 2. ENFORCE promotes the reuse of FM-based planners by integrating low-level FM-based planners with higher-level SE artifacts. ENFORCE takes as inputs an image that describes the environment in which the robots will be deployed (1), e.g., the layout of a building, the models that describe the robots' behaviors (2), and the missions the robot should achieve, i.e., the properties of interest (3). Those artifacts are combined into a comprehensive model of the robotic application (5), and used to compute a set of plans ensuring the



(a) Timed Automaton modeling the moving platform of the robot. **(b) Timed Automaton modeling the gripper of the robot.**

Figure 1: A network of Timed Automata

mission achievement (6). Finally, the plan is performed by sending executable actions to the physical robots, or robot simulators (7).

In this section, we describe the inputs of ENFORCE (Section 4.1), and the procedures used by ENFORCE to compute the plans to be executed by the robots (Section 4.2).

4.1 Inputs

The inputs of ENFORCE are discussed in the following.

Environment Description (1). The environment processed by ENFORCE is described using a high-level description of the environment represented by the images of the buildings contained in classical building layouts, such as the one used to indicate emergency exits in public buildings. An example of environment that can be processed by ENFORCE, representing the Building 22 of Politecnico di Milano, is reported in Figure 3.

Models of the Robots (2). In our envisioned usage, the models of the robots are provided by third party companies, such as robots manufacturers, or designed by developers that want to use ENFORCE. The model of the robot describes how a robot (1) moves, (2) performs actions, and (3) synchronizes with other robots.

Robot Movements. The TA that models the robot includes (at least) five locations, one representing the idle state (s), and four corresponding to the movements in the four directions, i.e., up (u), down (d), left (l) and right (r). Note that, for simplicity we had only considered four directions for movement. However, the approach can be extended to consider more complex models of robot movements. Figure 4 presents an example of model of the robot discussed in the following. When the robot undertakes a motion action, one transition from location s to one of the locations representing the motions is performed. For the action α to be performed, a transition of the robot labeled with $\alpha!$ is fired. For any $\alpha \in \{u, d, l, r\}$, the TA changes the current location into location α to model that the robot has just finished action α. All the transitions leading from s to α are labeled with a guard that encodes the duration of the shift. The automaton measures the temporal delay ensued from the kinetics of the robot by means of a clock t. Let *speed* be the maximum speed of the robot and *tmove* be $\frac{span}{speed}$, i.e., *tmove* is the minimum time required to cover a distance of length *span*. The value *tmove* is used in the guards $t \geq tmove$ to set the duration of each transition labeled with action u, d, l and r, meaning that it takes to the robot at least *tmove* to move up, down, left, and right, respectively. Every time one of those transitions is performed, clock t is reset in order to begin the measure of the delay of the next action. Once a motion action is finished, the robot can either stop, or keep moving in the same direction. If the robot stops, a transition labeled with action "stay" (i.e., $s!$) is taken. Performing a "stay" action entails a change of the TA current location, which is then set to s, and a reset of the clock t. The time needed by the robot to stop is determined by the value *tstay*. Hence, the transitions from locations $\alpha \in \{u, d, l, r\}$ to s are guarded with $t \geq tstay$. If the robot keeps moving towards the same direction, then the same action α can be iterated multiple times, provided that the condition *tmove* is satisfied and the synchronization $\alpha!$ with the TA modeling the map can be realized.

Robot actions. Actions are encoded in the automaton through suitable TA locations. In Figure 4 locations a and b are associated with "operation a" and "operation b" to represent the execution of actions a and b. The locations are connected to state s by means of two transitions, one enabling the activation of the action, and the other representing its termination. The robot might start one action if some suitable conditions are satisfied. These constraints

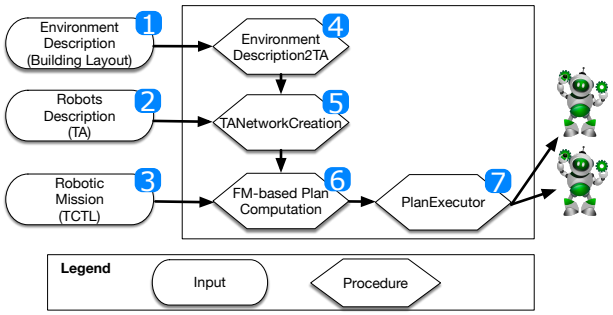


Figure 2: ENFORCE overview.

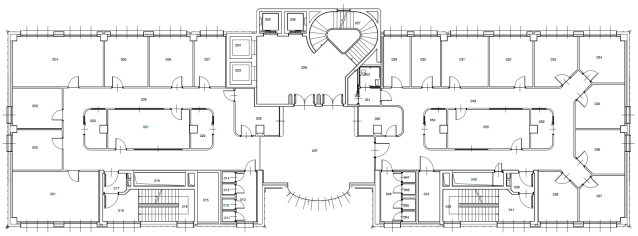
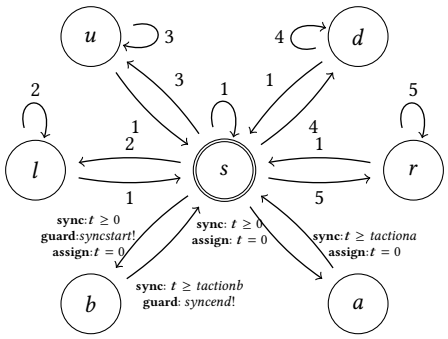


Figure 3: Building 22 of Politecnico di Milano.



The formulae corresponding the numeric labels of the edges:
(1) **guard**: $t \geq tstay$ **sync**: $s!$ **assign**: $t = 0$
(2) **guard**: $t \geq tmove$ **sync**: $l!$ **assign**: $t = 0$
(3) **guard**: $t \geq tmove$ **sync**: $u!$ **assign**: $t = 0$
(4) **guard**: $t \geq tmove$ **sync**: $d!$ **assign**: $t = 0$
(5) **guard**: $t \geq tmove$ **sync**: $r!$ **assign**: $t = 0$

Figure 4: An example of TA robot model able to move in four different directions and perform two different actions.

can be expressed in the guard of the transitions leading to their associated locations. Once a transition is fired, the robot remains in the target location for the entire duration $\mathtt{t}_{\mathtt{action0}}$ of the operation O, being O either a or b, and then, it returns in location s when $t \geq \mathtt{t}_{\mathtt{action0}}$.

Synchronization with other robots. Synchronizations among robots is modeled using appropriate actions, that are added in the set Act. In Figure 4, the robot synchronizes with another one on channel *syncstart* when it performs operation a. Once the operation is done, the robot notifies the termination on channel *syncend*. Given an action α, the coupling between two robots requires on one side sending a message on the channel (e.g., $\alpha!$) and on the other side receiving it (e.g., $\alpha?$).

Robotic Mission (3). We assume that the mission the robots have to perform is specified by users as a TCTL formula. In the future, we plan to generate the TCTL specification by supporting the usage of higher-level specification languages, such as robotic mission specification patterns [40] or robotic DLSs [24]. We currently support two different types of missions that can be expressed in TCTL, i.e., reachability and (ordered) execution of actions.

Reachability. Reachability properties require that within a given time limit a location should be finally reached. Let TIME be a global clock that is never reset, and let (x, y) be a target location that must be reached within a time bound t_{bound}. Let $l_{(x,y)}$ be a boolean variable that is set to one when the robot enters the location (x, y). The following TCTL formula states that the robot has reached (x, y) earlier than t_{bound} time units from the beginning of the execution.

$$\exists \mathcal{F}((l_{(x,y)} = \text{TRUE}) \wedge (\text{TIME} < t_{bound}))$$

Execution of Actions. Let $flag_a$ be a boolean variable (initially set to false) indicating that action a has been performed. The following TCTL formula specifies that action a is performed earlier than t_{bound} time units.

$$\exists \mathcal{F}((flag_a = \text{TRUE}) \wedge (\text{TIME} < t_{bound})) \quad (1)$$

Sometimes the mission may require a robot to perform a set of actions $\{a, b, c\}$ within a specific time bound. Hence, the property can be expressed by the formula below.

$$\exists \mathcal{F}((\bigwedge_{i \in \{a,b,c\}} flag_i = \text{TRUE}) \wedge (\text{TIME} < t_{bound})) \quad (2)$$

The specification of properties that require a robot to perform a set of actions in a specific order within a specific time-bound, can be done as follows. Consider the actions a, b, c to be done in this order. This mission can be expressed by using the TCTL formula

$$\exists \mathcal{F}((G = \text{TRUE}) \wedge (\text{TIME} < t_{bound})) \quad (3)$$

where variable G is an additional variable added to the model of the robots that holds only if a, b, and c follow the correct order.

4.2 Procedures

The main procedures of ENFORCE are discussed in the following. These procedures are inspired by the one proposed by Quottrup et al. [44] and Andersen et al. [8], which are well-known TA approaches for solving the planning problem in the robotic domain.

EnvironmentDescription2TA (4). The purpose of this procedure is to convert a high-level SE environment description into a lower-level FM-based modeling formalism. We propose two different instances, as alternative options, of the procedure to convert a high-level map of the environment into TA, namely *Encoding 1* and *Encoding 2*.

Encoding 1. Let W and H be the width and the height (expressed in meters) of the (rectangular) map where the robots move, and let *span* be the sampling span, i.e., width and the high of the squared cells that are used to tile this map. Let also be $(0, 0)$ be the coordinate of the bottom-left corner of the figure. We define two sets X, Y containing respectively the x and y Cartesian coordinates of the points (x, y), generated by the creation of a grid with square cells with width and high *span*. Formally,

$$X = \{x \mid 0 \leq x \leq W - span, (x \ \% \ span) = 0\}$$
$$Y = \{y \mid 0 \leq y \leq H - span, (y \ \% \ span) = 0\}$$

where % is the modulus mathematical operator.

For every $a_r \in Act$, let $Bk(a_r)$ be the set of positions (x, y), with $x \in X$, $y \in Y$, where the robot cannot take action a_r when it is in position (x, y), i.e., the action a_r is blocked. For instance, $Bk(u_r)$ are those positions (x, y) from which the robot cannot move to reach $(x, y + span)$. For each robot, the environment is represented as a TA, defined as $\langle Q, q_{x_0,y_0}, v^0, I, T \rangle$, whose locations represent the current position that is occupied by the robot, i.e., a location $q_{x,y} \in Q$ encodes position (x, y). Location q_{x_0,y_0} is the initial location of the robot. The TA constraints robot movements using actions $\{u_r, d_r, l_r, r_r, s_r\}$. Transitions in T are defined as follows, for any

$(x, y) \in X \times Y$:

$(q_{(x,y)}, q_{(x,y+span)}, \emptyset, \emptyset, (u_r, ?), \emptyset, \emptyset) \in T$ if $(x, y) \notin Bk(u_r)$;
$(q_{(x,y)}, q_{(x,y-span)}, \emptyset, \emptyset, (d_r, ?), \emptyset, \emptyset) \in T$ if $(x, y) \notin Bk(d_r)$;
$(q_{(x,y)}, q_{(x+span,y)}, \emptyset, \emptyset, (r_r, ?), \emptyset, \emptyset) \in T$ if $(x, y) \notin Bk(r_r)$;
$(q_{(x,y)}, q_{(x-span,y)}, \emptyset, \emptyset, (l_r, ?), \emptyset, \emptyset) \in T$ if $(x, y) \notin Bk(l_r)$;
$(q_{(x,y)}, q_{(x,y)}, \emptyset, \emptyset, (s_r, ?), \emptyset, \emptyset) \in T$

For example, the portion of the environment bounded by a red dashed rectangle in Figure 5a is converted into the TA in Figure 5c.

Encoding 2. Let $W, H, span, X, Y$ be defined as for Encoding 1, and r be a robot. The position of the robot r is represented using two real variables x_r and y_r, with $0 \leq x_r \leq W - span$ and $0 \leq y_r \leq H - span$. The set of locations Q of the TA contains only location q, which is also the initial state of the TA. Five transitions specify how the robot can change its position in the environment, depending on whether the robot is moving up, down, left, right or is remaining in its current location. Every transition is labeled with (i) a guard that specifies if the robot can move in a given direction; (ii) an event that will be used to synchronize the model of the environment with the actions taken by the robot r; and (iii) an assignment that updates the coordinates of the robot. For every action $a_r \in Act$, we define a guard γ_{ar} that is enabled only if the current position of the robot does not block the execution of the action a_r, i.e., the position of the robot is not in $Bk(a_r)$.

$$\gamma_{ar} = \bigwedge_{(x,y) \in Bk(a_r)} \neg (x = x_r \wedge y = y_r)$$

Then, the TA modeling the environment of the robot r contains the following five transitions:

$(q, q, \gamma_{ur}, \emptyset, (u_r, ?), \emptyset, y_r = y_r + span)$;
$(q, q, \gamma_{dr}, \emptyset, (d_r, ?), \emptyset, y_r = y_r - span)$;
$(q, q, \gamma_{rr}, \emptyset, (r_r, ?), \emptyset, x_r = x_r + span)$;
$(q, q, \gamma_{lr}, \emptyset, (l_r, ?), \emptyset, x_r = x_r - span)$;
$(q, q, \emptyset, \emptyset, (s_r, ?), \emptyset, \emptyset) \in T$;

For example, consider the portion of the environment bounded by a red dashed rectangle in Figure 5a, the portion of the guard of the transitions that allows the robot to go down is presented in Figure 5b.

Network Creation (5). To create a comprehensive model of the robotic application, the model of the robots and their environment are combined into a (single) network of TA. To this end, every TA modeling a robot r is added to the network. Furthermore, a copy of the TA describing the environment obtained in (4) is created for each robot r of the robotic application, and it is added to the network. This ensures that the robot r can perform a movement action $\alpha_r \in \{u_r, d_r, r_r, l_r\}$ only if a transition labeled with $\alpha_r!$ synchronizes with a transition labeled with $\alpha_r?$ of the copy of TA modeling its environment.

FM-based Plan Computation (6). Before executing the planning, the network of TA is modified depending on the type of the mission to be considered.

• *Reachability.* To handle reachability missions, the model of the environment of the robot r that should reach position (x, y) is modified as follows. If the environment is generated using *Encoding 1*, an assignment that sets variable $l_{(x,y)}$ to True is added to all the transitions that enter the state representing location (x, y). If the transition is generated using *Encoding 2*, an additional transition that sets the variable $l_{(x,y)}$ to True when the variables x_r and y_r are set to values (x, y) is added to the TA.

• *Execution of Actions.* To handle execution of actions, the TA of the robots are changed as follows. If the mission is specified as in formula 1, the assignment $flag_a$ = True is added to the transition that connect state a to state s. If the mission is specified as in formulae 2 and 3, $flag_a$ = True, $flag_b$ = True, $flag_c$ = True are respectively added to the transitions that connect states a, b and c to state s. Furthermore, for the mission specified in formula 3, (i) two additional guards requiring that that $flag_a$ and $flag_b$ are set to True are respectively added to the transitions that connect state s to states b and c; and (ii) the assignment G = True is added to the transition that connect state c to state s.

After performing these changes, the network of the TA and the TCTL mission are fed into the Uppaal model checker. If the network of TA contains a trace that satisfies the mission under analysis, it is returned by Uppaal. Otherwise, an error reporting that no plan is available is shown to the user.

Plan Executor (7). The plan executor uses the traces produced by the planner to generate the commands to be sent to the robots. The trace is iteratively parsed, and the actions contained in the trace are converted into commands that are sent to the actual physical robots, or to the robotic simulator.

5 EVALUATION

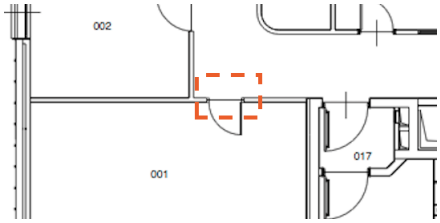
This section assesses the effectiveness of ENFORCE which is implemented as a standalone Python application. The source code and a complete replication package are available from [19]. In particular, we have conducted several experiments[1] in order to answer four main questions:

RQ1: Does ENFORCE effectively synthesize plans ensuring the satisfaction of the mission for single-robot applications? If yes, how long does the synthesis procedure take?
RQ2: Does ENFORCE effectively synthesize plans ensuring the satisfaction of the mission for multi-robot applications? If yes, how long does the synthesis procedure take?
RQ3: Are the plans computed by ENFORCE ensuring the mission satisfaction in simulated environments?
RQ4: Are the plans computed by ENFORCE ensuring the mission satisfaction in real environments?

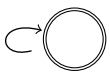
RQ1 - Single Robot Effectiveness. We have evaluated different scenarios that vary over different missions, environment maps, sampling steps values, and the encoding of the environment.

Experiment Design. We analysed three different maps described in Table 5, representing real buildings and their sizes and number of rooms, as described in the table. The maps are available in our online repository [19]. We used three different maps in order to make sure that our experiments consider test cases that are representatives of realistic maps. For each map, we assumed four different sampling steps (*ST*) 50cm, 75cm, 100cm, and 125cm. Furthermore, we tried both encoding 1 (C1) and encoding 2 (C2),

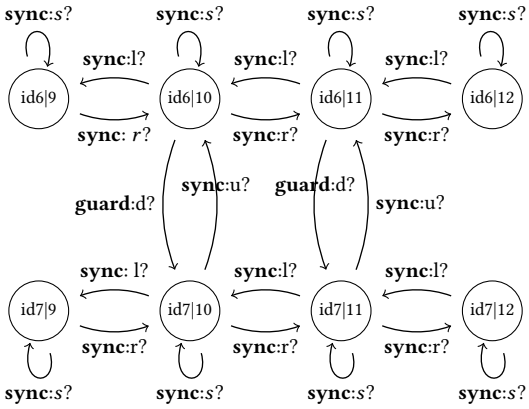
[1] All the experiments have been conducted on a machine with 8 GB 1600 MHz.



(a) A portion of the Building 22 of Politecnico di Milano.



(b) Encoding 2 applied to the portion of the environment contained in the red dashed box of Figure 5a



(c) Encoding 1 applied to the portion of the environment contained in the red dashed box of Figure 5a.

Figure 5: Two different encoding of the environment.

Table 4: Computation time (seconds) required by ENFORCE to generate plans for each scenario.

		*M*1						*M*2						*M*3					
		*T*1		*T*2		*T*3		*T*1		*T*2		*T*3		*T*1		*T*2		*T*3	
E	**ST**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**	**C1**	**C2**
*E*1	50	39.9	8.7	37.2	8.2	44.6	8.2	40.2	2.5	41.6	2.7	40.8	2.7	101.2	93.4	61.0	32.5	61.5	32.4
*E*1	75	4.8	2.3	7.4	2.3	6.1	2.0	4.3	0.9	4.5	1.0	5.4	0.8	38.7	22.7	13.7	8.0	12.8	7.7
*E*1	100	2.4	0.9	1.9	1.0	1.9	1.0	1.9	0.5	1.7	0.5	1.5	1.0	17 .9	8.7	5.6	3.6	5.2	3.3
*E*1	125	0.8	0.5	1.0	0.7	0.8	0.5	0.6	0.3	0.7	0.3	0.7	0.3	8.5	4.9	3.4	1.8	2.8	1.8
*E*2	50	5.2	1.8	5.5	1.7	5.2	1.8	3.1	0.5	2.8	0.5	2.9	0.5	24.0	9.7	6.6	2.6	6.8	2.6
*E*2	75	4.5	1.1	4.7	1.1	4.5	1.3	2.4	0.5	2.4	0.4	2.6	0.4	20.6	5.0	2.0	0.7	6.5	1.7
*E*2	100	1.4	0.6	1.3	0.5	1.4	0.5	0.6	0.2	0.7	0.2	0.6	0.2	7.0	2.8	3.5	6.1	2.1	0.7
*E*2	125	0.5	0.2	0.5	0.1	0.5	0.6	2.5	0.2	2.5	0.1	2.5	0.2	3.5	0.2	0.8	0.1	0.8	0.3
*E*3	50	9.8	10.0	8.5	10.4	9.2	9.6	8.0	3.1	6.9	3.3	7.2	3.0	6.5	124.8	15.6	29.6	14.9	28.8
*E*3	75	2.5	1.7	2.3	1.5	2.4	1.5	2.0	0.6	1.5	0.6	1.6	0.6	6.2	30.8	4.8	5.5	4.8	5.8
*E*3	100	1.0	1.0	0.8	1.3	0.9	1.2	0.8	0.5	0.5	0.6	0.5	0.7	6.4	14.8	1.9	4.3	2.0	3.7
*E*3	125	0.3	0.5	0.5	0.2	0.5	0.2	4.3	0.3	0.3	0.3	0.3	0.3	3.4	6.7	1.0	1.5	1.2	1.5

* *E*1: Jupiter Building, *E*2: Building 22 *E*3: Building 20;
*M*1: Mission1, *M*2: Mission2, *M*3: Mission3;
*C*1: Encoding1, *C*2: Encoding2;
*T*1: Time bound 1, *T*2: Time bound 2, *T*3: Time bound 3.

and three missions that the mobile robot has to achieve within a predefined time-bound: reaching a point (*M*1), reaching two points close to the initial position of the robot, and performing one action in each of these points (*M*2), and reaching two points far from the initial position of the robot, and performing one action in each of these points (*M*3). For every map, we considered three values *T*1, *T*2, and *T*3 as time-bound for the completion of the missions, i.e., *T*1, *T*2, and *T*3 are, respectively, equal to 150*s*, 500*s*, and 800*s* for map *E*1, 160*s*, 300*s*, and 600*s* for maps *E*2 and *E*3. ENFORCE has been applied on each map once for every sampling step (ST), every encoding, every mission and every time bound, which implies a total of 216 variations.

Table 5: ID, size, number of rooms (#R) and description of the environments considered in RQ1.

ID	Size	#R	Description
E1	$80m \times 90m$	80	Jupiter Building, Chalmers University
E2	$120m \times 50m$	57	Building22, Politecnico di Milano
E3	$70m \times 50m$	50	Building20, Politecnico di Milano

Results. The results are reported in Table 4, which shows that ENFORCE succeeded in computing a plan that satisfied the considered missions for all the cases. The magnitude of the highest time

measured for the encoding 1 and 2 is few minutes (respectively, 101.2 and 124 seconds).

The answer to **RQ1** is that on the considered scenarios, ENFORCE effectively synthesized plans ensuring the satisfaction of the mission for single-robot applications.

RQ2 - Multi-Robots Effectiveness. To answer this question, we studied a scenario with two robots, $E1$ map and a sampling step measuring 125cm, that is the most coarse-grained among those considered in **RQ1**.

Experiment Design. The robots are supposed to start from different initial locations, meet at the same pre-chosen point to execute a collaborative action, and finally move to a destination point within 1600 seconds. We performed the analysis with three values for destination point $P1$, $P2$, and $P3$ by increasing distance from the initial robots locations; meaning that $P3$ is the farthest from the initial point and $P1$ is the closest one to it.

Results. The experiment showed that in case of two robots, ENFORCE takes at least 45 minutes to compute a plan, or return a time out error. Therefore, the performance is not acceptable for practical usage of a planner in real cases scenarios.

The answer to **RQ2** is that for the three considered scenarios, ENFORCE was able to synthesize plans ensuring the satisfaction of the mission for multi-robots applications. However, computing the plans required at least 45 minutes.

RQ3 - Correctness in Simulated Environments. In order to better investigate the generated plans by ENFORCE, we realised them with Choreographe simulator[2], which allows for simulation of a Nao robot[3]. Our goal is to verify that the synthesized plans are correct, i.e, they allow satisfying the corresponding missions.

Experiment Design. We had used the environment $E1$ since it is the biggest (in terms of m^2) among the considered environment maps. Moreover, we had to consider missions that are feasible to be simulated with Choregroaphe. The simulated scenario was based on executing the following three missions by the robot in $E1$.

- the robot has to start from an initial location, reach a given position, get a set of items from a table, and return to its initial location;
- the robot has to start from its initial location, reach a given position, say a sentence or a warning, return to its initial location; and
- the robot has to start from its initial position, reach a given position, unload an object, and return to its initial position.

The movements, the say and the unload actions are simulated, respectively, by means of the *MoveAlong*, *Say*, and *Cartesian motion actions* of the Choreographe simulator.

Results. The simulator confirmed that the actions executed by the robots successfully satisfied the mission requirements.

The answer to **RQ3** is that on the three considered scenarios, the Choreographe simulator confirms the correctness of the plans computed by ENFORCE.

RQ4 - Correctness in Real Environments. As it was of utmost importance for our tool to be practical, we experimented the correctness of ENFORCE plans in real environments.

Experiment Design. We conducted the real experiments in $E1$, and the offices of our industrial partner PAL-robotics[4], with Turtlebot[5] and a TIAGO robot[6] and a sampling step of 50cm long. For this experiment, we had considered the following three missions, that were defined based on (i) the requirements of our industrial partner, and (ii) the type of the robots and facilities we had access to.

- the TurtleBot patrols the building during the night. It starts from the initial position P1, and reaches the location P2, where it checks the presence of an intruder. If an intruder is detected the robot calls the surveillance. Then, it goes back to P1.
- the TurtleBot delivers a box from one office to another. The robot starts from office P1, where a user loads it, and moves to the delivery point P2 within 2 minutes, where it unloads the box, and finally returns to P1.
- TIAGO starts from location P1, reaches first location P2 and then P3, and goes back to location P1 within 3 minutes. In every locations, an audio message is delivered (to users).

We used ENFORCE to compute the motion plans within the environments, that were enriched with the recording actions. To test the plans in the environment, we send ROS [7] navigation goals to the robots, to enforce the movement actions, and the Linux command Say to play audio messages.

Results. In all the considered scenarios, the robots effectively satisfy their missions. Videos of our experiments are available in our online repository [19].

The answer to **RQ4** is that on the three considered scenarios, our experiments confirm the correctness of the plans computed by ENFORCE.

6 DISCUSSION

The results of the evaluation lead us to the following findings. We present our discussion by relating it with the inputs of ENFORCE.

- *Environment Description and Planning Precision* (1). From a software engineering standpoint, the reusability of planners in industrial applications depends on their performance and scalability. The experimental evaluation in Sec. 5, points out that mapping areas with a low sampling value ensures accurate movements, but increases the computation time, as it enlarges the size of the TA modeling the entire system. Information related to the sampling values, and the size of the maps that can be effectively processed by planners, are usually not extensively discussed in research papers. Conversely, in this work, we considered three

[2] http://doc.aldebaran.com/1-14/software/choregraphe
[3] http://www.ald.softbankrobotics.com/robots/nao
[4] http://pal-robotics.com/
[5] http://www.turtlebot.com/
[6] https://tiago.pal-robotics.com/
[7] http://wiki.ros.org/ROS

different maps of real environments, robotic missions, and single and multi-robot applications, to extensively discuss how the sampling step, and the size of the map, affect the performance of the ENFORCE (inspired by [8]). The sampling step and the size of the map do not pose a threat to single-robot applications, whereas dramatically affect the use of planners for multi-robots applications. However, choosing the right value for the sampling step is essential, as an unreasonably large value assigned to the sampling step does not allow ENFORCE to synthesize plans.
Possible Improvement. One solution to the previous problem would be to consider a dynamic sampling of the environment. Instead of decoding the whole environment once, dynamic sampling can consider only a local neighborhood of the current location at each time step, improving performance.

– *Robotic System Characteristics* (2). When software engineers want to design a robotic application, they have to find a planner that is suitable for the set of robots they are using, each one having specific characteristics. For this reason, the features and the assumptions of a planner should be clearly stated, to allow the designers to select the best option, that can satisfy the modeling requirements needed to represent the behavior of the robots. However, these data, as well as the performance of planners, are often not clearly stated and thoroughly evaluated, and it is often difficult to understand whether a planner meets certain expectations. For example, the size of the team of robots that can be managed by a planner (with certain performances) is often not precisely specified. In this work, we have considered one among many planners that have been proposed in the robotics literature (for the reasons specified in Section 4). While ENFORCE uses a well-known efficient and optimized model checker, the state explosion problem still represents the major obstacle to the adoption of FM-based solutions, in particular those based on TA-planners, in the robotic domain.
Possible Improvements. We believe that to enable and facilitate (re)-use of existing planners, additional effort is required to provide upfront documentation on the working assumptions, and more rigorous and thorough evaluations of planners. The research community has to develop guidelines to help developers and researchers in this direction. While the FM-community has been studying for several decades solutions to limit the state explosion problem, it is unknown whether this problem will be solved in the future. Therefore, to earn the benefits of FM-techniques in practical contexts, they must be wisely used. We believe that there is a need for understanding how to effectively (re)-use existing FM-based planners within robotic projects. For example, it is necessary to understand at which level of abstraction planning should be performed. Performing planning on a higher level of abstraction provides computational benefits, but opens new problems, i.e., how high-level models and plans can be bound to the physical world and the low-level problems that can occur as the mission is performed.

– *Characteristics of Missions* (3). Planners compute sequences of actions that ensure the satisfaction of a mission that the robotic team has to achieve. While robotic users often have a clear high-level idea of the goal the robotic application, they are usually not familiar with logic-based languages, that are the most recurrent means adopted for mission specification. Since understanding whether a logical language is suitable for expressing the mission under analysis might be difficult, the choice of a planner is, generally, not trivial. Furthermore, even when a language that is enough expressive to capture the missions is found, writing mission might still be hard.
Possible Improvements. We believe that it is important to formally state and discuss the type of missions that can be processed by the different planners, and to provide high-level languages that guide non-expert users during the mission specification. In our previous work, we performed an initial step in this direction. We proposed robotic mission specification patterns [40], that map recurrent mission specification problems to logic-based solutions, and integrated them into a robotic Domain Specific Language (DSL) [24].

7 CONCLUSION

In this paper, we proposed ENFORCE (formally vErified plaNning soFtware fOr Real-world sCEnarios), a robot planning framework that uses Timed Automata to represent the robot behavior and the environment in which the robots are deployed, and that adopts TCTL for the specification of the missions, including real-time constraints. The framework enables the automatic computation of plans, that ensure the satisfaction of a mission of interest. ENFORCE relies on Uppaal, a state-of-the-art model-checker for the computation of plans. To evaluate ENFORCE, we considered both simulated environments and real robots.

We believe that this work warns the research community to promote a systematic and extensive evaluation of FM-based planners in robotic applications, and to push for a closer integration of formal methods into software engineering. Our contribution increases the quality of software applications in robotics, as it fosters a deeper evaluation of FM-based planners and in particular those built upon TA, a deeper evaluation of their underlying assumptions, and use context. The analysis paves the way for more conscious and systematic reuse of existing planners across (different) SE platforms of robotic applications. Our discussion, in fact, clearly highlights the limitation of the FM-based planner used in this work with the purpose of strengthening the — still too weak — links between FM and SE.

ACKNOWLEDGMENTS

This work has received funding from the European Research Council under the European Union's Horizon 2020 research and innovation programme (grant No 694277 and No 731869). We also acknowledge financial support from Centre of EXcellence on Connected, Geo-Localized and Cybersecure Vehicle (EX-Emerge), funded by Italian Government under CIPE resolution n. 70/2017 (Aug. 7, 2017).

REFERENCES

[1] ISO 10218-1. 2011. *Robots and robotic devices – Safety requirements for industrial robots – Part 1: Robots.* Organization for Standardization, Geneva, Switzerland. 43 pages.
[2] ISO 10218-2. 2011. *Robots and robotic devices – Safety requirements for industrial robots – Part 2: Robot systems and integration.* Organization for Standardization, Geneva, Switzerland. 72 pages.
[3] ISO/TS 15066. 2016. *Robots and robotic devices – Collaborative robots.* Organization for Standardization, Geneva, Switzerland.

[4] ISO/TR 20218-1:2018. 2018. *Robotics – Safety design for industrial robot systems – Part 1: End-effectors.* Organization for Standardization, Geneva, Switzerland.

[5] ISO/TR 20218-2:2017. 2017. *Robotics – Safety design for industrial robot systems – Part 2: Manual load/unload stations.* Organization for Standardization, Geneva, Switzerland.

[6] Rajeev Alur, Costas Courcoubetis, and David L. Dill. 1993. Model-Checking in Dense Real-Time. *Information and Computation* 104, 1 (1993), 2 – 34.

[7] Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235.

[8] Michael S. Andersen, Rune S. Jensen, Thomas Bak, and Michael M. Quottrup. 2004. Motion planning in multi-robot systems using timed automata. *IFAC* 37, 8 (2004), 597 – 602. https://doi.org/10.1016/S1474-6670(17)32043-8 IFAC/EURON Symposium on Intelligent Autonomous Vehicles.

[9] Saddek Bensalem, Lavindra de Silva, Andreas Griesmayer, Felix Ingrand, Axel Legay, and Rongjie Yan. 2011. A Formal Approach for Incremental Construction with an Application to Autonomous Robotic Systems. In *Software Composition*. Springer, 116–132.

[10] Darko Bozhinoski, Davide Di Ruscio, Ivano Malavolta, Patrizio Pelliccione, and Massimo Tivoli. 2015. FLYAQ: Enabling Non-expert Users to Specify and Generate Missions of Autonomous Multicopters. In *Automated Software Engineering, ASE*.

[11] Davide Brugali. 2007. *Software engineering for experimental robotics*. Vol. 30. Springer.

[12] Yushan Chen, Xu Chu Ding, and Calin Belta. 2011. Synthesis of distributed control and communication schemes from global LTL specifications. In *Conference on Decision and Control and European Control Conference*. IEEE.

[13] Yushan Chen, Jana Tumova, and Calin Belta. 2012. LTL robot motion control based on automata learning of environmental dynamics. In *International Conference on Robotics and Automation*. IEEE.

[14] Alessio Colombo, Daniele Fontanelli, Axel Legay, Luigi Palopoli, and Sean Sedwards. 2015. Efficient customisable dynamic motion planning for assistive robots in complex human environments. *Journal of ambient intelligence and smart environments* 7, 5 (2015), 617–634.

[15] Rafael Rodrigues da Silva, Bo Wu, and Hai Lin. 2016. Formal design of robot Integrated Task and Motion Planning. In *Conference on Decision and Control (CDC)*. IEEE.

[16] Jonathan A. DeCastro, Javier Alonso-Mora, Vasumathi Raman, Daniela Rus, and Hadas Kress-Gazit. 2018. *Collision-Free Reactive Mission and Motion Planning for Multi-robot Systems*. Springer, 459–476. https://doi.org/10.1007/978-3-319-51532-8_28

[17] Saadia Dhouib, Selma Kchir, Serge Stinckwich, Tewfik Ziadi, and Mikal Ziane. 2012. Robotml, a domain-specific language to design, simulate and deploy robotic applications. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 149–160.

[18] Ha Thi Thu Doan, François Bonnet, and Kazuhiro Ogata. 2017. Model Checking of a Mobile Robots Perpetual Exploration Algorithm. In *Structured Object-Oriented Formal Language and Method*, Shaoying Liu, Zhenhua Duan, Cong Tian, and Fumiko Nagoya (Eds.). Springer International Publishing, Cham, 201–219.

[19] ENFORCE 2019. *ENFORCE: formaly vErified plaNning soFtware fOr Real-world sCEnarios*. https://github.com/Askarpour/ENFORCE

[20] Formal Methods Europe. 2019. *Formal Methods*. http://www.fmeurope.org/formalmethods/

[21] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. 2009. Temporal logic motion planning for dynamic robots. *Automatica* 45, 2 (2009), 343 – 352. https://doi.org/10.1016/j.automatica.2008.08.008

[22] Marie Farrell, Matt Luckcuck, and Michael Fisher. 2018. Robotics and integrated formal methods: necessity meets opportunity. In *International Conference on Integrated Formal Methods*. Springer, 161–171.

[23] Mohammed Foughali, Bernard Berthomieu, Silvano Dal-Zilio, Pierre-Emmanuel Hladik, Félix Ingrand, and Anthony Mallet. 2018. Formal Verification of Complex Robotic Systems on Resource-Constrained Platforms. In *Formal Methods in Software Engineering (FormaliSE)*.

[24] Sergio García, Patrizio Pelliccione, Claudio Menghi, Thorsten Berger, and Tomas Bures. 2019. High-Level Mission Specification for Multiple Robots. In *Conference on Software Language Engineering (SLE)*. ACM.

[25] Meng Guo and Dimos V. Dimarogonas. 2013. Reconfiguration in motion planning of single- and multi-agent systems under infeasible local LTL specifications. In *Conference on Decision and Control*. IEEE.

[26] Meng Guo, Karl Henrik Johansson, and Dimos V. Dimarogonas. 2013. Motion and action planning under LTL specifications using navigation functions and action description language. In *Intelligent Robots and Systems*. IEEE.

[27] Meng Guo, Karl Henrik Johansson, and Dimos V. Dimarogonas. 2013. Revising motion planning under Linear Temporal Logic specifications in partially known workspaces.. In *International Conference on Robotics and Automation*. IEEE.

[28] Meng Guo, Jana Tumova, and Dimos V. Dimarogonas. 2014. Cooperative decentralized multi-agent control under local LTL tasks and connectivity constraints. In *Conference on Decision and Control*. IEEE.

[29] Meng Guo, Jana Tumova, and Dimos V. Dimarogonas. 2015. Hybrid control of multi-agent systems under local temporal tasks and relative-distance constraints. In *Conference on Decision and Control (CDC)*. IEEE.

[30] Raju Halder, José Proença, Nuno Macedo, and André Santos. 2017. Formal Verification of ROS-Based Robotic Applications Using Timed-Automata. In *Formal Methods in Software Engineering, FormaliSE*.

[31] IFR. 2016. World Robotic Survey. https://ifr.org/ifr-press-releases/news/world-robotics-survey-service-robots-are-conquering-the-world-.

[32] Marius Kloetzer and Calin Belta. 2006. LTL Planning for Groups of Robots. In *International Conference on Networking, Sensing and Control*. IEEE.

[33] Marius Kloetzer and Calin Belta. 2007. Control of Multi-Robot Teams Based on LTL Specifications, In Conference on Management and Control of Production and Logistics. *IFACs*.

[34] Marius Kloetzer and Calin Belta. 2010. Automatic Deployment of Distributed Teams of Robots From Temporal Logic Motion Specifications. *IEEE Transactions on Robotics* 26, 1 (2010), 48–61. https://doi.org/10.1109/TRO.2009.2035776

[35] T. John Koo, Rongqing Li, Michael Melholt Quottrup, Charles A. Clifton, Roozbeh Izadi-Zamanabadi, and Thomas Bak. 2012. A framework for multi-robot motion planning from temporal logic specifications. *Science China Information Sciences* 55, 7 (2012), 1675–1692. https://doi.org/10.1007/s11432-012-4605-8

[36] Ron Koymans. 1990. Specifying real-time properties with metric temporal logic. *Real-Time Systems* 2, 4 (1990), 255–299. https://doi.org/10.1007/BF01995674

[37] Kim G Larsen, Paul Pettersson, and Wang Yi. 1997. UPPAAL in a nutshell. *International journal on software tools for technology transfer* 1, 1-2 (1997), 134–152.

[38] Matt Luckcuck, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher. 2019. Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR)* 52, 5 (2019), 100.

[39] Claudio Menghi, Sergio Garcia, Patrizio Pelliccione, and Jana Tumova. 2018. Multi-robot LTL Planning Under Uncertainty. In *Formal Methods*. Springer.

[40] Claudio Menghi, Christos Tsigkanos, Patrizio Pelliccione, Carlo Ghezzi, and Thorsten Berger. 2019. Specification Patterns for Robotic Missions. *IEEE Transactions on Software Engineering* (2019). https://doi.org/10.1109/TSE.2019.2945329

[41] Alvaro Miyazawa, Pedro Ribeiro, Wei Li, Ana Cavalcanti, Jon Timmis, and Jim Woodcock. 2019. RoboChart: modelling and verification of the functional behaviour of robotic applications. *Software & Systems Modeling* (2019), 1–53.

[42] Alexandros Nikou, Dimitris Boskos, Jana Tumova, and Dimos V. Dimarogonas. 2017. Cooperative planning for coupled multi-agent systems under timed temporal specifications. In *American Control Conference (ACC)*.

[43] Alexandros Nikou, Jana Tumova, and Dimos V. Dimarogonas. 2016. Cooperative task planning of multi-agent systems under timed temporal specifications. In *American Control Conference (ACC)*.

[44] Michael Melholt Quottrup, Thomas Bak, and Roozbeh Izadi-Zamanabadi. 2004. Multi-robot planning: a timed automata approach. In *International Conference on Robotics and Automation*. IEEE.

[45] E. Rabiah and B. Belkhouche. 2016. Formal specification, refinement, and implementation of path planning. In *International Conference on Innovations in Information Technology (IIT)*.

[46] Stephen L. Smith, Jana Tumova, Calin Belta, and Daniela Rus. 2010. Optimal path planning under temporal logic constraints. In *International Conference on Intelligent Robots and Systems*. IEEE/RSJ.

[47] Jana Tumova, Luis I. Reyes Castro, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. 2013. Minimum-violation LTL planning with conflicting specifications. In *American Control Conference*.

[48] Jana Tumova and Dimos V. Dimarogonas. 2016. Multi-agent planning under local LTL specifications and event-based synchronization. *Automatica* 70 (2016), 239 – 248. https://doi.org/10.1016/j.automatica.2016.04.006

[49] Jana Tumova, Alejandro Marzinotto, Dimos V. Dimarogonas, and Danica Kragic. 2014. Maximally satisfying LTL action planning. In *International Conference on Intelligent Robots and Systems*. IEEE/RSJ.

[50] Alphan Ulusoy, Stephen L. Smith, Xu Chu Ding, Calin Belta, and Daniela Rus. 2011. Optimal multi-robot path planning with temporal logic constraints. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 3087–3092. https://doi.org/10.1109/IROS.2011.6094884

[51] Ye Zhao, Ufuk Topcu, and Luis Sentis. 2016. High-level planner synthesis for whole-body locomotion in unstructured environments. In *Conference on Decision and Control (CDC)*. IEEE.

[52] Yuchen Zhou, Dipankar Maity, and John S. Baras. 2015. Optimal mission planner with timed temporal logic constraints. In *European Control Conference (ECC)*.