Robotics: A New Mission for FRET Requirements

Robotics: A New Mission for FRET Requirements*

Gricel Vázquez[1][0000−0003−4886−5567], Anastasia Mavridou[2][0000−0002−3943−9753], Marie Farrell[3][0000−0001−7708−3877], Tom Pressburger[4], and Radu Calinescu[1][0000−0002−2678−9260]

[1] Department of Computer Science, University of York, York, UK
[2] KBR Inc., NASA Ames Research Center, Moffett Field, USA
[3] Department of Computer Science, The University of Manchester, Manchester, UK
[4] NASA Ames Research Center, Moffett Field, USA

Abstract. Mobile robots are used to support planetary exploration and safety-critical environments such as nuclear plants. Central to the development of mobile robots is the specification of complex required behaviors known as missions. In this paper, we use NASA's Formal Requirements Elicitation Tool (FRET) to specify functional robotic mission requirements. To examine the applicability of FRET in the mobile robotics domain, we studied robotic mission patterns specified in Linear Temporal Logic (LTL). These patterns were originally derived from a large repository that included patterns from the literature and consultation with industrial experts. We extend this repository with those found during our extensive literature review. Although FRET has been successfully used in the past in case studies within the aerospace domain, mobile robot requirements present new challenges in their specification. To this end, our work provides a methodological basis for using FRET in the specification of robotic mission requirements.

1 Introduction

Mobile robots can help to separate humans from hazards and inaccessible environments, such as nuclear plants [14] and planetary exploration [12]. For example, NASA's Curiosity and Perseverance rovers are mobile robots that explore Mars, enabling us to gather data and execute missions that are currently out of reach for human astronauts. Central to the development of such systems is the elicitation and specification of mission functional requirements that guide the

design and analysis process. Robotic mission requirements typically describe the high-level tasks that the robotic system must accomplish and how it should react in specific situations. These requirements are typically written in ambiguous natural language making their understanding and specification error-prone.

Formal languages, rooted in rigorous semantics, provide the means to precisely describe and reason about intended behavior; however, they are less intuitive than natural language. Even for experts, translating mission requirements into a formal language, like temporal logic, can be challenging [19,32]. To bridge the gap between intuitive natural language descriptions and unambiguous formal languages, specification patterns have been proposed by the research community. Recent work in [35] presents a collection of 22 temporal logic specification patterns for robotic missions, resulting from a systematic literature review and consultation with industrial partners and domain experts.

NASA's Formal Requirements Elicitation Tool (FRET) provides support for users to write their system's requirements using a restricted natural language. These requirements are automatically provided with temporal logic semantics that is more amenable to analysis and verification than the natural language requirements would be [9,19]. The ability to automatically generate temporal logic formalization from (structured) natural language requirements is useful, particularly in domains where those specifying the requirements may not be experts in logical formalization.

Our objective in this paper is to examine whether FRET, which has been used on aerospace use cases (e.g., [19,37,32]), can be adopted more generally in the robotics domain. Although FRET has been used effectively in specific robotic use cases [9,21], we examine the generalizability of FRET for mission specifications of mobile robots.

We begin by exploring the specification patterns identified in [35]. We examined the literature to find a more up-to-date set of patterns, which resulted in extending the original set with 6 new specification patterns and their associated temporal logic formalizations. We specify these patterns using FRET [23].

Specifically, this paper contributes:

1. A set of six newly identified robotic mission specification patterns that were derived from a systematic literature review (Section 2)
2. The specification using FRET of the patterns identified in [35], as well as our newly identified patterns (Section 3).
3. A study of the expressibility and applicability of FRET for robotic missions in Section 4.

2 Systematic Review: Identification of Robotic Patterns

We present our work on adding new robotic mission patterns to the set of patterns from [35], which, to the best of our knowledge, is the most extensive repository targeting robotic missions for mobile robots and their translation into Linear Temporal Logic (LTL) specifications. Each pattern is characterized by (i) a name; (ii) a description of the mission requirement; (iii) a template of the

Table 1. The venue sources used to obtain the primary studies for our literature review. We include the total number of publications per venue, which were identified using the queries defined in Section 2.

i Venues	**Acronyms**	**Type**	**Num publ.**
i.1 International Journal of Robotics Research	IJRR	Journal	29
i.2 Transactions on Software Eng.	TSE	Journal	1
i.3 Robotics and Automation Letters	RA-L	Journal	369
i.4 Transactions on Robotics	T-RO (Trob)	Journal	63
i.5 Transactions on Automation Science and Eng.	T-ASE	Journal	0
i.6 Software Eng. for Adaptive and Self-Managing Sys.	SEAMS	Conf. & Symposium	0
i.7 Symposium on Applied Computing	SAC	Conf. & Symposium	2
i.8 Foundations of Software Engineering	ESEC/FSE	Conf. & Symposium	0
i.9 Intelligent Robots and Systems	IROS	Conference	133
i.10 Int. Conference on Robotics and Automation	ICRA	Conference	137
i.11 Int. Conference on Automation Science and Eng.	CASE	Conference	32
i.12 International Conference on Advanced Robotics	ICAR	Conference	15
i.13 International Conference on Software Engineering	ICSE	Conference	9
i.14 Int. Conf. on Model Driven Eng. Languages and Sys.	MODELS	Conference	3
i.15 Simulation, Modeling and Progr. for Aut. Robots	SIMPAR	Conference	2
i.16 Software Engineering and Formal Methods	SEFM	Conference	2
ii Library/Search engine			
ii.1 IEEE Computer Society digital library			22
ii.2 Google Scholar search			16
		TOTAL	**835**

mission specification in temporal logic; (iv) variations of the pattern describing possible minor changes; (v) examples of how it is used and occurrences in the literature; and (vi) its relationship to other patterns. We refer to the original set from [35] as the *Initial Patterns*, and our enlarged set simply as *Patterns*.

LTL Background. We briefly review future time LTL operators [6,10] used throughout the paper. The **X** operator refers to the next time point, i.e., $\mathbf{X}\phi$ is true iff ϕ holds at the next time point. The **F** operator refers to at least one future time point, i.e., $\mathbf{F}\phi$ is true iff ϕ holds at some future time point including the present time. **G** is true iff ϕ is always true in the future. $\phi\mathbf{U}\psi$ is true iff ψ holds at some point t in the future and for all time points t' such that $t' < t$, ϕ is true. $\phi\mathbf{W}\psi$ has similar semantics but does not require that ψ holds at some point in the future, i.e., ψ remains false. The release operator $\phi\mathbf{V}\psi$ is defined as $!((!\phi)\mathbf{U}(!\psi))$.[5] This means that either ψ never holds and ϕ holds forever or that ψ occurs at time t and ϕ holds at all time points t' such that $t' \leq t$.

2.1 Methodology: Systematic Review

We conducted a systematic review following the guidelines in [8] and [28]. We scanned 835 primary studies gathered from a list of well-known venues and digital resources. Our search spanned the last five years (2018–2022). Before 2018,

[5] We use $(\wedge, \vee, \neg)$ and (&,|,!) indistinguishably for the usual logical operators.

we relied on the extensive review from which the original mission patterns were conceived [35]. Our literature review focuses on the identification of robotic mission specifications and seeks to answer the following **exploratory questions**:

- (RQ1) What types of logics are used in the specification of these mission requirements?
- (RQ2) For studies that use LTL, what types of missions do they describe and how are these specified?
- (RQ3) Are there any newly-identified patterns that are not already captured by the *Initial Patterns* repository?

Search strategy. To identify the most relevant publications, we obtained an initial set of studies from (a) well-known high-impact venues in the areas of robotics and automation and (b) Google Scholar and IEEE Computer Society digital library. The first source provided high-impact papers ensuring the quality of the results, while search engines provided a wider search of the literature.

The list of venues is shown in Table 1. Papers were gathered from the DBLP database, filtering by year and venue, using the query: "*robot*|*MRS*\$ *task*|*schedule*|*allocation*|*mission*|*adapt*"[6], where spaces mean logic *and*'s, | logic *or*'s and \$ looks for exactly the word before this sign. For Google Scholar and IEEE Library, only the top 25 results sorted by relevance were considered. For both engines, we used the query "*(robots OR robot OR MRS) AND (task OR tasks OR schedule OR scheduling OR allocation OR mission OR missions OR adapt OR adaptation)*."

Inclusion criteria. We identified 835 *primary studies* after removing: 1) duplicates; 2) papers focusing on robotic hardware development rather than missions; and 3) two papers related to the *Initial Patterns*. We focus our literature review on formally specified missions, in comparison with the *Initial Patterns* repository [35], which considers missions described either in natural language or formally. We wanted to assess how often formalisms are used in robotic missions and gather information about the different logics used in the robotic missions domain. Such information is valuable for developing and identifying requirement specification patterns and extending formalization tools, such as FRET. As expected, the majority of papers do not use formalisms for mission specification.

Data items. For the studies that use LTL in the formalizations, we collected the following data:

- **Title**. Title and reference to the paper.
- **Summary**. English language Description of the LTL mission requirements.
- **Mission**. Set of LTL formulae that describe the mission requirements.
- **Pattern**. If the pattern already exists in *Initial Patterns*, then we provide its name. Otherwise, we introduce the name of the newly-identified pattern.

For each study, we manually extracted the mission specifications defined in temporal logic. We next present our findings concerning the three exploratory questions.

[6] MRS stands for Multi-Robot Systems.

2.2 Answering The Research Questions

At the beginning of this section, we outlined three research questions that we discuss, in light of our findings from the literature search, below.

(RQ1) What types of logics are used in the specification of these mission requirements?

We identified 16 papers that formally specify mission requirements; five use LTL and the rest use different logics. We encourage the reader to explore the background sections of these papers for a deeper understanding of these logics. Signal temporal logic (STL) is frequently used for the specification of robotic missions [27,42,41]. In [41], STL is used to declare tasks for the collaborative manipulation of robotic arms. A fragment of LTL, called capability temporal logic (CaTL), is used in [30] to generate specifications with absolute or relative timing of task completion, repetition frequencies, and different types of task interdependence, like sequencing or synchronization. In [25] a new specification language for mission specifications called Event-based Signal Temporal Logic, which is an extension of STL, is proposed. In [26], the authors implement a framework for the satisfiability of robotic missions described in an extended version of the Event-based STL logic by adding the ability to specify discrete uncontrolled event reactions. Metric temporal logic (MTL) was used in [38] for the specification of robotic manipulators. While FRET currently supports MTL, we are extending the pattern repository on *mobile* robot missions. Hence, the study of *motion planning* on robotic manipulators is deferred for future work.

For self-adapting mobile service robots, in [11], probabilistic computational tree logic (PCTL) encodes mission optimization metrics such as time or energy consumption. A repository of mission patterns defined in RPCTL, i.e., PCTL extended with rewards, is provided in [36] as a continuation of the *Initial Patterns* augmented with quantitative reasoning by adding the probabilistic and reward operators. RPCTL is also used in [44]. A variant of LTL, called sc-LTL, is used in [47] for the specification of missions to guide the multi-task planning problem using a Q-learning based approach. Finally, five studies formalize robotic mission specifications in LTL [5,18,33,40,46], which we detail next.

(RQ2) For studies that use LTL, what type of missions do they describe and how are these specified?

Table 2 presents the five studies with LTL formulae. The first column presents the LTL formulae; the second the corresponding pattern, if it exists; the third column provides an English description. The last column contains the reference to the corresponding research paper. In [18], the authors present a heuristic technique for updating robotic plans as new tasks are allocated. The robot tasks consist of moving boxes, pulling levers, travelling between locations and scanning rooms. In [5], the specification of multi-robot systems with collaborative tasks is studied. Continuing, [33] proposes a decentralized planner for part knowledge called MAPmAKER, evaluated on two case studies: one with robots in a residential facility and the other as a services provider. Next, [40] studies the allocation and planning of tasks with multiple robots in four variants of an office envi-

ronment; while [46] describes MRS missions in which robots have limited local information. Its case study consists of picking and dropping a box at different locations and performing activities such as scanning and taking pictures.

(RQ3) Are there identified patterns that are not already captured by the *Initial Patterns* repository?

From the gathered set of 42 formulae (first column, Table 2), 23 were already defined by a pattern in the *Initial Patterns* repository and one could not be identified as a pattern (identified in the second column as NA). For the remaining 17 formulae, we proposed a new set of six patterns: (1) *Visit with reaction*, (2) *Weak sequenced visit*, (3) *Continuous visit with reaction*, (4) *Weak patrolling*, (5) *Deliver with visit*, (6) *Maintain safe state* (depicted in column two within square brackets [ ]). The mission specification that is not defined as a pattern follows the form: $\mathbf{F}(l_1 \wedge a \wedge \mathbf{X}((a_1 \mathbf{U} a_2) \wedge \mathbf{F}\ a))$, where $\{a, a_1, a_2\}$ is a set of actions and l_1 is a reachable location. This is a very specific behaviour where action a is performed at location l_1, followed by two actions a_1,a_2 in a given order, performing action a again in the future; hence, this is not considered a pattern.

Formalization of new patterns. Let $R = \{r_1, r_1, ..., r_n\}$ be a set of robots and $A = \{a_1, a_2, ..., a_m\}$ a set of actions that robots can perform, where a_i holds when any robot $r_j \in R$ executes action a_i. Let $L = \{l_1, l_2, ..., l_{n_1}\}$ be a set of locations and l_k holds when a robot reaches this location. We use the notation $l_\#$ to indicate any location in L, $a_\#$ any action in A, $(l_\#)^\omega$ an infinite trace of locations, and $(a_\#)^*$ a finite trace of actions. We define a logical disjunction of the set of locations to be visited as $d_x = \bigvee_{j=1}^{n_2} s_j$ where $s_j \in S$ belongs to a subset of locations $S \subseteq L$ and n_2 is the number of elements in S. For example, if locations l_1 or l_2 or l_4 are to be visited, then $d_1 = l_1|l_2|l_4$, which holds when any of these locations are visited. We denote $\{d_x\} = S$ the set of locations in d_x. For the previous example $\{d_1\} = \{l_1, l_2, l_4\}$. We define $g_{i,k} = l_i \wedge a_k$ as the visit of location l_i where a_k action is performed. Let $l_{\#-\{1,2,3\}}$ be any location except for locations $l_1,\ l_2$ or l_3; we define the new patterns as:

Visit with reaction: Visit a set of locations in an unspecified order. When at that location, an action must be carried out. In this case, we use the conjunction to assemble a formula with multiple locations and actions.

$$\bigwedge_{i=1}^{n} \mathbf{F}(l_i \wedge a_i), \ \texttt{where}\ a_i \in A$$

Example: Action a_1 must be done at location l_1 and action a_2 at location l_2, at least once. An example of a trace that satisfies the mission requirement is $l_\# \rightarrow \{l_2, a_2\} \rightarrow l_\# \rightarrow \{l_1, a_1\} \rightarrow (l_\#)^\omega$. The trace $l_\# \rightarrow a_2 \rightarrow l_2 \rightarrow \{l_1, a_1\} \rightarrow (l_\#)^\omega$ violates the requirement as action a_2 is never carried out when visiting l_2.

Weak sequenced visit: Visit n locations in a specific order, where the ith location $i \leq n$ exists in $\{d_i\}$. It does not prohibit the interleaving of locations.

Table 2. Collection of LTL mission specifications extracted from the systematic review. New patterns are depicted in bold square brackets **[ ]**.

Temporal Logic Formula	Pattern name (from *Patterns*)	Definition	Ref.
(!dropoff U (room2 ∧ pickup)) (!dropoff U (room3 ∧ dropoff)) F(room3 ∧ pulllever) GF(room1 ∧ scan ∧ usecamera) (!(room1scan) U (room4scan)) F(room1 ∧ scan)	Wait Wait **[Visit with reaction]** Patrolling Wait **[Visit with reaction]**	Pick up box from room2, drop it off in room3; pull the lever in room3. Repeatedly scan and take a picture in room1. Scan in room4, then scan in room 1.	[18]
GF(room2 ∧ scan) GF(room5 ∧ scan)	Patrolling Patrolling	Repeatedly travel between room2 and room5 and scan in those rooms.	
F(room1 ∧ usecamera) F(room4 ∧ pickup)	**[Visit with reaction]** **[Visit with reaction]**	Take a picture in room1 and pick up a box in room4, in any order.	
G(F(loadcarrier))	Patrolling	Periodically robot r1 loads debris on r2.	[33]
G(F(detectload ∧ F(unload)))	Sequenced patrolling	Robot r2 load debris and later unload.	
G(F(takesnapshot ∧ F(sendinfo)))	Sequenced patrolling	Robot r3 repeatedly takes and sends pictures.	
F(s1 ∧ (F(s2 | s3)))	**[Weak ordered visit]**	Reach a destination where service s1 is provided, then perform either s2 or s3.	
G(F(s4 | s5))	**[Weak patrolling]**	Visit s4 or s5 infinitely often.	
F(desk ∧ default ∧ X((carrybin U dispose) ∧ F(default)))	NA	Robot by the desk while not loaded. Carry the bin until garbage is disposed and put it away again to reach the default state.	[40]
F(desk ∧ emptybin ∧ X(desk ∧ default)) G(carrybin ⇒ !public)	**[Cont. visit with reaction]** Instant reaction	Place empty paper bin next to the desk. Avoid public areas while carrying a bin.	
F(printer) F(kitchen) G(battery_20)	Visit Visit **[Maintain safe state]**	Refill supplies at the printer room and the kitchen. Ensure sufficient battery.	
F(p ∧ carry U (d10 ∧ X!carry)) F(p ∧ carry U (d7 ∧ X!carry)) F(p ∧ carry U (d5 ∧ X!carry)) G(carry ⇒ !public) F(printer)	**[Deliver after visit]** **[Deliver after visit]** **[Deliver after visit]** Instant reaction Visit	Distribute copies (p) to desks d10, d7, d5, and avoid public areas while carrying the document. Printer has sufficient paper.	
F(m1 ∧ photo) F(m4 ∧ photo) F(m6 ∧ photo) G(!meeting ⇒ !camera) F(d5 ∧ carry U (d3 ∧ X!carry)) G(carry ⇒ !public) F(d11 ∧ guide U (m6 ∧ X!guide))	**[Visit with reaction]** **[Visit with reaction]** **[Visit with reaction]** Instant reaction **[Deliver after visit]** Instant reaction **[Deliver after visit]**	Take a photo in rooms m1, m4, and m6. Deliver document from d5 to d3. Guide a person at d11 to room m6. Turned off camera while not in meeting rooms. Document is not delivered through any public areas.	
(GF(t1)) (GF(t4)) G(w ∧ !o) (GF(t2)) (GF(t3))	Patrolling Patrolling **[Maintain safe state]** Patrolling Patrolling	Robots persistently survey locations t1, t2, t3 and t4. Always remain in the working space w and avoid obstacles in o.	[46]
F(r1t1) ∧ F(r2t2) ∧ F(r3t3) ∧ F(r4t4) (!r1t1 U r4t4) F(t4 ∧ F(t3))	Visit Wait Ordered visit	Each robot has a local task to perform. Robot r1 do task t1 only after r4 do t4. Do task t3 after t4.	[5]

$$\mathbf{F}(\underbrace{d_1 \wedge (\mathbf{F}(d_2 \wedge (\mathbf{F}(d_3 \wedge ...(\mathbf{F}(d_n)))))))}_{\text{F } d_i \text{ nested n times}}$$

Example: An instance of this property where $d_1 = l_1$ and $d_2 = l_2|l_3$ is written as $\mathbf{F}(l_1 \wedge \mathbf{F}(l_2|l_3))$. The traces $l_1 \rightarrow l_\# \rightarrow l_2 \rightarrow (l_\#)^\omega$ and $l_1 \rightarrow l_\# \rightarrow l_3 \rightarrow (l_\#)^\omega$ satisfy the instance of this property as l_2 or l_3 holds after l_1. The trace $l_2 \rightarrow (l_1)^\omega$ does not satisfy the requirement as l_2 never holds after l_1.

Continuous visit with reaction: Visit a location for n consecutive steps. During the visit, perform an action at each time step.

$$\mathbf{F}(\underbrace{g_{1,1} \wedge (\mathbf{X}(g_{1,2} \wedge (\mathbf{X}(g_{1,3} \wedge ...(\mathbf{X}(g_{1,n})))))))}_{\text{X } g_{1,i} \text{ nested n times}}$$

Example: Location l_1 must be visited and perform action a_1 when arriving, and a_2 at the next time step. In this case, $g_{1,1} = l_1 \wedge a_1$ and $g_{1,2} = l_1 \wedge a_2$, hence the specification is written as $\mathbf{F}(l_1 \wedge a_1 \wedge (\mathbf{X}(l_1 \wedge a_2)))$. The trace $\{l_1, a_2\} \rightarrow \{l_2, a_2\} \rightarrow \{l_1, a_1\} \rightarrow \{l_1, a_2\} \rightarrow (l_\#)^\omega$ satisfies the requirement. The trace $\{l_1, a_1\} \rightarrow \{l_2, a_2\} \rightarrow (l_\#)^\omega$ violates it as the robot exits l_1 before doing a_2.

Weak patrolling: Visit infinitely often one or more locations within $\{d_i\}$.

$$\bigwedge_{i=1}^{n} (\mathbf{G}(\mathbf{F}(d_i)))$$

Example: At least one of locations l_1, l_2 or l_3 must be visited infinitely often. In this case, $d_1 = l_1|l_2|l_3$ and the specification is written as $\mathbf{G}(\mathbf{F}((l_1 \mid l_2 \mid l_3))$. The traces $l_1 \rightarrow (l_\# \rightarrow l_2)^\omega$ and $l_3 \rightarrow l_\# \rightarrow (l_1)^\omega$ satisfy the requirement, as at least one location is visited infinitely often. The trace $l_2 \rightarrow l_4 \rightarrow (l_{\#-\{1,2,3\}})^\omega$ does not satisfy the requirement.

Deliver after visit: Eventually start action one and, if not at the specified location, start action two. If not at the specified location, continue performing action two until the specified location is visited and stop performing action two afterwards.

$$\mathbf{F}(a_1 \wedge a_2 \mathbf{U}(l_i \wedge \mathbf{X}!a_2)), \texttt{ where } a_{1,2} \in A$$

Example: When a_1 happens, then carry an object (a_2) until location l_1 is reached. The trace $(\{a_1, a_2\} \rightarrow \{!a_1, a_2\})^* \rightarrow (a_2)^* \rightarrow l_1 \rightarrow !a_2 \rightarrow (l_\#)^\omega$ satisfies this property at some point as action a_2 holds until it arrives at location l_1 and, at the next time step, a_2 is false. Notice that when l_i is reached, the action a_2 can be true or false. The trace $\{a_1, a_2\} \rightarrow \{l_1, !a_2\} \rightarrow (a_2)^\omega$ does not satisfy the requirement because $!a_2$ does not hold after l_1.

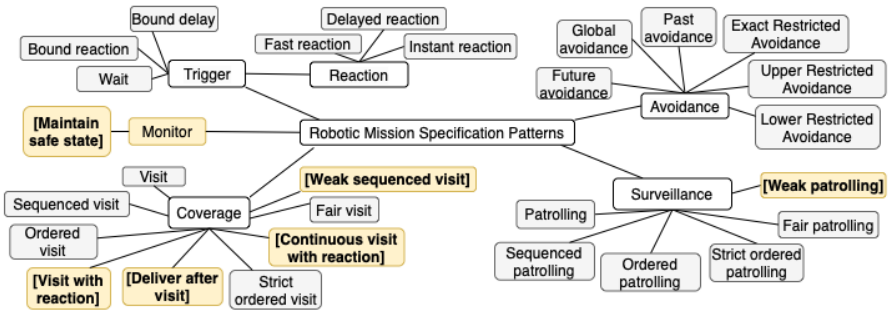

Fig. 1. *Patterns* for robotic missions for mobile robots organized in five categories: coverage, surveillance, avoidance, trigger and monitor. New patterns are depicted in yellow within square brackets [ ]. The rest are from the original catalogue [35].

Maintain safe state: Check that an action holds at every time point, for instance, for monitoring purposes; this formalization is also called an *invariant*.

$$\bigwedge_{i=i}^{n} \mathbf{G}(a_i)$$

Example: Measure the battery energy at all times to maintain it at least at 20%, where a_1 holds if the battery energy is $\geq$20%. A trace that satisfies the specification is $a_1 \rightarrow (l_{\#}, a_1)^{\omega}$, while the trace $l_1 \rightarrow (a_1)^{\omega}$ violates it because a_1 does not hold at the first time point.

We define *actions* as activities that the robots must execute within their operational domain, such as avoiding obstacles. Additionally, for the pattern *maintain safe state*, these actions include non-functional activities such as ensuring that the battery charge maintains a specific charge level.

Figure 1 shows the *Patterns* grouped into five categories: Coverage, Surveillance, Avoidance, Monitor, and Trigger. Note that Reaction is a subcategory of Trigger. As this robotic mission pattern catalogue is intended to grow as new patterns are identified, in this paper, we propose the *Monitor* category to capture the continuous monitoring of some mission-related parameter or behaviour, for instance, continual monitoring of battery energy level and maintaining the temperature within a specified range. The rest of the categories are taken from [35].

3 Expressing robotic missions in FRET

In this Section, we study the specification of *Patterns* in FRET.

FRETish Background: A FRETish requirement comprises up to six elements (of which the elements marked with a * are mandatory): 1) `scope` specifies the time intervals where the requirement is enforced; 2) `condition` is a

Boolean expression that whenever true specifies that the `response` shall happen; 3) `component*` is the system component that the requirement is levied upon; 4) `shall*` is used to express that the component's behavior must conform to the requirement; 5) `timing` specifies when the response shall happen, subject to the constraints defined in `scope` and `condition` and 6) `response*` is the Boolean expression that the component's behavior must satisfy. Since not everything can be expressed in pure FRETish, the language provides **escape-to-LTL** by allowing Boolean expressions to contain standard LTL operators such as `Globally` (meaning **G**), `Future` (meaning **F**), `Untl` (meaning **U**), `Releases` (meaning **V**) and `Nxt` (meaning **X**).

Table 3 shows how we specified the robotic mission *Patterns* in FRETish. The shaded rows indicate new patterns identified by this work that are not included in the *Initial Patterns* and are also not supported by the toolset that accompanies them, PsALM [34]. The second column contains the name of the pattern and the corresponding LTL formulation. Since both PsALM and FRET tools work on instantiated versions of the patterns, we present the formulations instantiated for a specific number of locations. For example, Table 3 lists the instantiated version of the `Visit` pattern for two locations `l0`, `l1`. The third column contains the pattern written as FRETish requirement(s). The plus (+) sign is used when multiple requirements are needed to express a single pattern. In certain cases, e.g., `Ordered Visit`, a pattern can be written as the composition of an existing pattern, e.g., `Sequenced Visit`, with additional FRETish requirements.

Table 3: Robotic Mission Patterns in FRETish.

#	Instantiated Pattern	Requirement(s) in FRETish
1	**Visit** F l0 ∧ F l1	`robot shall eventually satisfy l0` `+ robot shall eventually satisfy l1`
2	**Sequenced Visit** F (l0 ∧ (F l1))	`robot shall eventually satisfy l0 & Future(l1)`
3	**Ordered Visit** F (l0 ∧ (F l1)) ∧ (! l1) U (l0)	**Sequenced Visit pattern** `+ robot shall until l0 satisfy !l1`
4	**Strict Ordered Visit** F (l0 ∧ (F l1)) ∧ (! l1) U (l0) ∧ (! l0) U (l0 ∧ X (! l0 U (l1)))	**Ordered Visit pattern** `+ robot shall immediately satisfy Untl((!l0), l0 & Nxt(Untl(!l0,!l1)))`

5	**Fair Visit**
	F l0 ∧ F l1
	G (l0 ⇒ X ((! l0) W l1)) ∧
	G (l1 ⇒ X ((! l1) W l0))
		Visit pattern
		`+ whenever l0 robot shall at the next timepoint satisfy Releases(l1,!l0 | l1)`
		`+ whenever l1 robot shall at the next timepoint satisfy Releases(l0,!l1 | l0)`

6	**[Visit With Reaction]** F(l0 ∧ action)	`robot shall eventually satisfy l0 & action`
7	**[Weak Sequenced Visit]** F(l0 ∧ F(l1 ∨ l2))	`robot shall eventually satisfy l0 & (Future(l1 | l2))`
8	**[Continuous Visit With Reaction]** F((l0 ∧ a1) ∧ (X (l0 ∧ a2)))	`robot shall eventually satisfy (l0 & a1) & (Nxt(l0 & a2))`
9	**[Deliver After Visit]** F(a1 ∧ a2 U (l0 ∧ X !a2))	`robot shall eventually satisfy a1 & Untl(a2, l0 & Nxt(!a2))`
10	**Patrolling** (G F l0) ∧ (G F l1)	`whenever true robot shall eventually satisfy l0` `+ whenever true robot shall eventually satisfy l1`
11	**Sequenced Patrolling** G (F (l0 ∧ (F l1)))	`robot shall always satisfy Future(l0 & Future(l1))`

Table 3 – *Continued from previous page.*

#	Instantiated Pattern	Requirement(s) in FRETish
12	**Ordered Patrolling** G (F (l0 ∧ (F l1))) ∧ !l1 U l0 ∧ G(l1 ⇒ X((!l1) U l0))	**Sequenced Patrolling pattern** + robot shall until l0 satisfy !l1 + robot shall eventually satisfy l1 + whenever l1 robot shall at the next timepoint satisfy Untl(!l1,l0)
13	**Strict Ordered Patrolling** G (F (l0 ∧ (F l1))) ∧ !l1 U l0 ∧ G(l1 ⇒ X((!l1) U l0)) ∧ G(l0 ⇒ X (!l0 U l1))	**Ordered Patrolling pattern** + whenever l0 robot shall at the next timepoint satisfy (Untl(!l0,l1))
14	**[Weak Patrolling]** G (F (l0 ∨ l1))	whenever true robot shall eventually satisfy l0 | l1
15	**Fair Patrolling** (G F l0) ∧ (G F l1) ∧ G (l0 ⇒ X ((! l0) W l1)) ∧ G (l1 ⇒ X ((! l1) W l0))	**Patrolling pattern** + whenever l0 robot shall at the next timepoint satisfy Releases(l1,!l0 | l1) + whenever l1 robot shall at the next timepoint satisfy Releases(l0, !l1 | l0)
16	**Past Avoidance** (!l0) U p	robot shall until p satisfy !l0 + robot shall eventually satisfy p
17	**Global Avoidance** G(!(l0))	robot shall never satisfy l0
18	**Future Avoidance** G((c) ⇒ (G(!l0)))	whenever c robot shall never satisfy l0
19	**Upper Restricted Avoidance** !F(l0 ∧ X(F(l0 ∧X(F(l0))))) at most n times where n=2	robot shall immediately satisfy ! Future(l0 & Nxt(Future(l0 & Nxt(Future(l0)))))
20	**Lower Restricted Avoidance** F(l0 ∧ X(F(l0 ∧X(F(l0))))) at least n times where n=3	robot shall eventually satisfy (l0 & Nxt(Future(l0 & Nxt(Future(l0)))))
21	**Exact Restricted Avoidance** (!l0) U (l0 ∧ (X(!l0 U (l0 ∧ (X(!l0 U l0 ∧ X(G !l0))))))) exactly n=3 times	robot shall immediately Untl (!l0, l0 & (Nxt(Untl (!l0, l0 & (Nxt(Untl(!l0, l0 & Nxt(Globally !l0))))))))
22	**Instant Reaction** G(p1 ⇒ p2)	whenever p1 robot shall immediately satisfy p2
23	**Delayed Reaction** G(p1 ⇒ F(p2))	whenever p1 robot shall eventually satisfy p2
24	**Prompt Reaction** G(p1 ⇒ X(p2))	whenever p1 robot shall at the next timepoint satisfy p2
25	**Bound Reaction** G(p1 ⟺ p2)	robot shall always satisfy p1 <-> p2
26	**Bound Delay** G(p1 ⟺ X(p2))	robot shall always satisfy p1 <-> Nxt(p2)
27	**Wait** l0 U p	robot shall until p satisfy l0 + robot shall eventually satisfy p
28	**[Maintain Safe Space]** G action	robot shall always satisfy action

4 Discussion

Prior uses of FRET have typically been in the aerospace domain. However, FRET has also been studied in robotic applications, including inspection [9] and grasping [21]. Nevertheless, neither of these prior works systematically evaluated the expressibility of requirements for robotic systems as we do in this paper.

4.1 Expressibility of FRETish for Robotic Mission Requirements

The robotic mission requirements that we studied in this paper can be different from aerospace requirements written previously in FRET [32,15,19,37]. Their

difference lies mainly in the unique nature and intrinsic complexity of robotic missions that require complex LTL specifications, e.g., nested temporal operators and reachability properties based on multiple locations.

We were able to specify all 28 patterns using FRET, however, in 13 cases (Table 3 rows 2, 4, 7-9, 11-13, 15, 19-21, 26), we had to use FRET's escape-to-LTL feature for patterns with nested temporal operators in `Sequenced Visit` and `Untl` and `Nxt` In future work, we will study how to extend FRET to capture these patterns in pure FRETish. One way would be to compositionally build complex patterns that require nesting of operators from simpler ones. Prior work makes a first step toward this by refactoring FRETish requirements that share repeated segments [20].

Table 3 shows that in certain cases, multiple FRETish requirements are needed to specify a single pattern. This decomposition of a pattern is beneficial for analysis purposes - performance-wise and also analysis feedback can be more targeted to specific sub-requirements - but also simplifies and makes the FRETish requirements easier to understand. To this end, FRET also helps users think of semantic subtleties and make intentional decisions when writing requirements. Consider the `Ordered Visit` pattern that uses the *strong until* **U** operator. The FRETish language provides the until keyword as a timing field option, however its semantics is that of weak until **W**[7]. To be able to express strong until, e.g., p **U** q we need two requirements in FRETish, i.e., one that expresses p **W** q and a second that expresses **F** q. As a result, when writing a requirement with the until keyword, the user must decide whether this is the intended semantics of the requirement or whether the intended semantics should be strong until instead. In the latter case, the user needs to intentionally add an extra requirement (see row 16, Table 3).

Finally, there might be multiple semantically equivalent ways of specifying the same pattern in FRETish, however, due to space limitations, we only present a single option per pattern in Table 3.

4.2 Comparing *Patterns* with FRETish Robotic Requirements in the Wild

Prior work contains FRETish requirements that were specified for a rover inspection mission [9]. We observe that some of these (system-level) requirements correspond to the `Maintain safe space` pattern that we have identified. For example, `Rover shall always satisfy speed <= 10`. Further, [9] also has component-level requirements that fit this pattern. Our focus in this paper is on requirements at the system and mission levels.

Other related work uses FRET in the specification of requirements for a robotic grasping system [21]. Here, we found instances of the `Maintain safe space` pattern as well as instances of the `Global avoidance` pattern. For example, `SV shall always satisfy !collide(SV, TGT)`. We note that here the

[7] According to FRET developers, this was a design choice after studying that when the until keyword was used in a requirement, in most cases it meant *weak until*.

global avoidance pattern is phrased differently than we have shown in Table 3, which uses the `never` timing in FRET. However, these two structures, one using logical negation and the other using never timing, are semantically equivalent. Although this robotic system is somewhat different from the mobile robots that we focus our work on, it is interesting that these safety-related patterns appear.

Finally, in recent work on the NASA Ames Research Center project Troupe, which aims at developing a fleet of rovers capable of autonomously mapping their environment [43], we found instances of the `Maintain safe space` and `Delayed reaction` patterns. The latter, however, uses a bounded version of the `eventually` timing operator, i.e., `within 1 second`.

More work is needed to examine the applicability of the *Patterns* more widely, both for robotics and whether similar patterns are useful in other critical systems. That said, their appearance in the aforementioned works demonstrates their relevance. Providing a set of generic patterns thus gives developers and engineers a starting point for eliciting requirements for robotic missions using FRET.

4.3 On PsALM and FRET

PsALM [34] was specifically developed to support predefined robotic patterns, i.e., those in the *Initial Patterns* catalogue. FRET is a requirements elicitation tool that supports the use of predefined templates but also authoring and understanding of requirements written from scratch in structured natural language. For example, to enable compositional analysis, a user would need to write component-level requirements complementary to the mission-level ones as demonstrated in [9]. If a developer intends to only use predefined *Patterns* to synthesize temporal formulae, then they can use either tool. The selection of a tool also depends on the intended purpose beyond this formalization stage. PsALM can generate inputs for multiple planners, simulators and model checkers. In contrast, FRET supports the generation of input for model checkers and runtime monitoring tools. Moreover, when using pure FRETish, FRET can generate both pure future-time and pure past-time LTL formulae (the PsALM mission catalogue is currently restricted to future-time LTL). By leveraging the pure past-time LTL translation, we can further perform realizability checking or other types of analysis with tools that can only digest pure past-time LTL.

4.4 Threats to Validity

We limited our search terms to those specified in §2.1 but it is possible that slightly different terms might have yielded different results. These mission patterns are defined for mobile robots transitioning between locations to perform tasks. Hence, in comparison to [35], we decided to explicitly include the search terms for the *scheduling* and *allocation* of tasks. It is also true that limiting ourselves to the 25 entries deemed most relevant by the search engine might have resulted in some interesting papers being omitted. However, we note that a larger literature review of formal specification and verification of autonomous robotic systems also travelled five pages deep in their chosen search engine (also Google

scholar) [31]. Their review was older and more general than ours, spanning papers published from 2007–2018. Future work might analyse the papers found during their review (which revealed temporal logics as a popular specification formalism for robotics) in light of the patterns that we identified.

In [33], the authors refer to previous studies on specification patterns [29,16,24] but we couldn't find a reference to the specific guideline they follow. Hence, we followed the guidelines for Systematic Reviews in Software Engineering from [8,28]. Had we opted for a collection of search engines, rather than just Google Scholar, this may have impacted the patterns that we derived. Our results are thus limited to those that are identified using Google's algorithms. Notably, the venues that our papers were drawn from were robotics, rather than formal methods venues. Including more venues where temporal logic papers appear, such as Formal Methods Europe or SAFECOMP for example, might have provided a richer set of temporal logic formulas.

Limiting our search to papers published at mostly academic venues might have resulted in gaps in the patterns that we identified. Our future work will seek to validate these patterns alongside industrial partners to ensure that our patterns are applicable and that additional patterns, including those that were potentially overlooked by our literature review, are identified and added to our catalog. For instance, we will seek to extensively validate these patterns and extract new patterns through missions and robotic projects at NASA, such as Troupe [7]. Two other sources we will consider for the extraction of additional robotic patterns in future work are well-known robotics competitions and large-scale studies. Robotic competitions such as RoboCup [3] and the DARPA Robotics Challenge [2] provide insights into state-of-the-art robotic missions, and large-scale case studies such as those conducted by Amazon Robotics offer real-world applications to investigate. For instance, the deployment of multiple mobile robots in an Amazon warehouse setting [1] presents opportunities to formalize complex robot interactions and mission strategies through FRETish, and ultimately, logic languages. This further validation and expansion of our catalog of patterns will help to ensure the maximum impact of this work whilst encouraging a wider uptake of FRET in robotics development.

5 Related work

The specification of patterns in temporal logic is not a new concept. Developing sets of commonly occurring patterns is useful to guide developers in specifying their systems [16,24,39]. Existing tools should ideally express commonly occurring patterns to encourage their uptake and ensure that they are applicable in relevant domains. Robotic systems are frequently built of multiple (often pre-existing) components, each with its own requirements [13]. Identifying and expressing frequently used patterns thus enables uniformity and reuse.

Recently, several efforts have focused on creating repositories of robotic missions gathered from industry and literature that can be easily reused and implemented. For example, ROBOMAX [4] is a dynamic repository of robotic systems

with self-adaptation capabilities that researchers can expand with new missions. ROBOMAX contains missions described in natural language.

Other works use temporal logic formalizations for the description of robotic missions and in particular for the planning of mobile robots [17,22]. Many of these were considered in the creation of mission patterns. However, these patterns do not support reasoning about quantities such as the cost to complete a mission or the probability of mission success without failure. As the need for qualitative requirements grows [45,44], a second set of robotic patterns, presented in [36], expands the mission patterns of [35] with quantitative semantics adding probabilities and rewards to capture uncertainty in robotic mission specifications. We did not use this second repository as FRET does not currently support probabilistic requirements.

6 Conclusion

This paper contributes (1) newly identified robotic mission patterns that were derived from a systematic literature review. The paper studies (2) the specification in FRETish of 28 distinct patterns, both newly identified in this paper and previously described in [35]. Finally, (3) we discuss and examine the implication that these patterns have for the design and applicability of FRET, by examining its expressibility and comparing these patterns with pre-existing sets of FRETish requirements. The catalog presented in Table 3 provides a methodological basis for roboticists wishing to use FRET to specify functional mission requirements for robots that are engaged in common tasks such as patrolling. Previously, FRET has been predominantly used for aerospace systems case studies. This paper illustrates that FRET can be more widely applicable, focusing on the mobile robotics domain.

In future work, we will explore additional sources of robotic missions. This will entail examining large-scale robotic applications, gaining insights into robotic missions through interviews with industrial developers, and investigating existing missions from projects at NASA and robotic competitions. We also plan to study the applicabilility of these patterns to other autonomous systems, such as within the context of Unmanned Aircraft Systems (UAS).

Acknowledgements G. Vázquez performed part of this work during her internship with KBR Inc. at NASA Ames Research Center. M. Farrell's work is supported by a Royal Academy of Engineering Research Fellowship. A. Mavridou is supported by NASA Contract No. 80ARC020D001. T. Pressburger is supported by NASA's System-Wide Safety project in the Airspace Operations and Safety Program.

References

1. Amazon AWS and Amazon Robotics, case study. `https://aws.amazon.com/solutions/case-studies/amazon-robotics-case-study/`, Accessed: 2024-03-08

2. DARPA Robotics Challenge website. `https://www.darpa.mil/about-us/timeline/darpa-robotics-challenge`, Accessed: 2024-03-08
3. Robocup federation official website. `https://www.robocup.org`, Accessed: 2024-03-08
4. Askarpour, M., Tsigkanos, C., Menghi, C., Calinescu, R., Pelliccione, P., García, S., Caldas, R., von Oertzen, T.J., Wimmer, M., Berardinelli, L., et al.: Robomax: Robotic mission adaptation exemplars. In: Software Engineering for Adaptive and Self-Managing Systems. pp. 245–251. IEEE (2021)
5. Bai, R., Zheng, R., Liu, M., Zhang, S.: Multi-robot task planning under individual and collaborative temporal logic specifications. In: Intelligent Robots and Systems. pp. 6382–6389. IEEE (2021)
6. Baier, C., Katoen, J.: Principles of Model Checking. MIT press (2008)
7. Benz, N., Sljivo, I., Vlastos, P.G., Woodard, A., Carter, C., Hejase, M.: The troupe system: An autonomous multi-agent rover swarm. In: AIAA SCITECH 2024 Forum. p. 2894 (2024)
8. Biolchini, J., Mian, P., Candida, A.N., Travassos, G.H.: Systematic review in software engineering. Tech. Rep. 05, System engineering and computer science department COPPE/UFRJ (2005)
9. Bourbouh, H., Farrell, M., Mavridou, A., Sljivo, I., Brat, G., Dennis, L.A., Fisher, M.: Integrating formal verification and assurance: An inspection rover case study. In: NASA Formal Methods. pp. 53–71. Springer (2021)
10. Bozzano, M., Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A., Micheli, A., Mover, S., Roveri, M., Tonetta, S.: nuxmv 2.0.0 user manual. Fondazione Bruno Kessler, Tech. Rept., Trento, Italy (2019)
11. Cámara, J., Schmerl, B., Garlan, D.: Software architecture and task plan coadaptation for mobile service robots. In: Software Engineering for Adaptive and Self-Managing Systems. pp. 125–136 (2020)
12. Cardoso, R.C., Kourtis, G., Dennis, L.A., Dixon, C., Farrell, M., Fisher, M., Webster, M.: A review of verification and validation for space autonomous system. Current Robotics Reports **2**(3), 273–283 (2021)
13. Côté, C., Létourneau, D., Michaud, F., Brosseau, Y.: Software design patterns for robotics: Solving integration problems with marie. In: Workshop of Robotic Software Environment (2005)
14. Devlin-Hill, B., Calinescu, R., Cámara, J., Caliskanelli, I.: Towards scalable multi-robot systems by partitioning the task domain. In: Towards Autonomous Robotic Systems. pp. 282–292. Springer (2022)
15. Dutle, A., Muñoz, C., Conrad, E., Goodloe, A., Perez, I., Balachandran, S., Giannakopoulou, D., Mavridou, A., Pressburger, T., et al.: From requirements to autonomous flight: An overview of the monitoring icarous project. In: Formal Methods for Autonomous Systems (2020)
16. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: International Conference on Software engineering. pp. 411–420 (1999)
17. Fainekos, G.E., Kress-Gazit, H., Pappas, G.J.: Temporal logic motion planning for mobile robots. In: Robotics and Automation. pp. 2020–2025. IEEE (2005)
18. Fang, A., Kress-Gazit, H.: Automated task updates of temporal logic specifications for heterogeneous robots. In: Robotics and Automation. pp. 4363–4369. IEEE (2022)
19. Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R.: FRETting About Requirements: Formalised Requirements for an Aircraft Engine Controller. In: Requirements Engineering: Foundation for Software Quality. pp. 96–111. Springer (2022)

20. Farrell, M., Luckcuck, M., Sheridan, O., Monahan, R.: Towards Refactoring FRETish requirements. In: NASA Formal Methods Symposium. pp. 272–279. Springer (2022)
21. Farrell, M., Mavrakis, N., Ferrando, A., Dixon, C., Gao, Y.: Formal modelling and runtime verification of autonomous grasping for active debris removal. Frontiers in Robotics and AI **8**, 639282 (2022)
22. Gavran, I., Majumdar, R., Saha, I.: Antlab: A multi-robot task server. ACM Trans. Embedded Computing Systems **16**(5s), 1–19 (2017)
23. Giannakopoulou, D., Mavridou, A., Rhein, J., Pressburger, T., Schumann, J., Shi, N.: Formal Requirements Elicitation with FRET. In: Requirements Engineering: Foundation for Software Quality (2020)
24. Grunske, L.: Specification patterns for probabilistic quality properties. In: Software Engineering. pp. 31–40 (2008)
25. Gundana, D., Kress-Gazit, H.: Event-based signal temporal logic synthesis for single and multi-robot tasks. IEEE Robotics and Automation Letters **6**(2), 3687–3694 (2021)
26. Gundana, D., Kress-Gazit, H.: Event-based signal temporal logic tasks: Execution and feedback in complex environments. IEEE Robotics and Automation Letters **7**(4), 10001–10008 (2022)
27. Innes, C., Ramamoorthy, S.: Automated testing with temporal logic specifications for robotic controllers using adaptive experiment design. In: Robotics and Automation. pp. 6814–6821. IEEE (2022)
28. Keele, S.: Guidelines for performing systematic literature reviews in software engineering. Tech. rep., EBSE (2007)
29. Konrad, S., Cheng, B.H.: Real-time specification patterns. In: Software Engineering. pp. 372–381 (2005)
30. Leahy, K., Serlin, Z., Vasile, C.I., Schoer, A., Jones, A.M., Tron, R., Belta, C.: Scalable and robust algorithms for task-based coordination from high-level specifications (scratches). IEEE Trans. Robotics **38**(4), 2516–2535 (2021)
31. Luckcuck, M., Farrell, M., Dennis, L.A., Dixon, C., Fisher, M.: Formal specification and verification of autonomous robotic systems: A survey. ACM Computing Surveys **52**(5), 1–41 (2019)
32. Mavridou, A., Bourbouh, H., Giannakopoulou, D., Pressburger, T., Hejase, M., Garoche, P.L., Schumann, J.: The ten lockheed martin cyber-physical challenges: Formalized, analyzed, and explained. In: Requirements Engineering. pp. 300–310. IEEE (2020)
33. Menghi, C., Garcia, S., Pelliccione, P., Tumova, J.: Multi-robot LTL planning under uncertainty. In: Formal Methods. pp. 399–417. Springer (2018)
34. Menghi, C., Tsigkanos, C., Berger, T., Pelliccione, P.: PsALM: Specification of Dependable Robotic Missions. In: Software Engineering. pp. 99–102. IEEE (2019)
35. Menghi, C., Tsigkanos, C., Pelliccione, P., Ghezzi, C., Berger, T.: Specification patterns for robotic missions. IEEE Transactions on Software Engineering **47**(10) (2019)
36. Menghi, C., Tsigkanos, C., Askarpour, M., Pelliccione, P., Vazquez, G., Calinescu, R., Garcia, S.: Mission Specification Patterns for Mobile Robots: Providing Support for Quantitative Properties. IEEE Trans. Software Engineering (2022)
37. Pressburger, T., Katis, A., Dutle, A., Mavridou, A.: Authoring, Analyzing, and Monitoring Requirements for a Lift-Plus-Cruise Aircraft. In: Requirements Engineering: Foundation for Software Quality. pp. 295–308. Springer (2023)

38. Saha, S., Julius, A.A.: Task and motion planning for manipulator arms with metric temporal logic specifications. IEEE Robotics and Automation Letters **3**(1), 379–386 (2017)
39. Salamah, S., Gates, A., Kreinovich, V.: Validated Templates for Specification of Complex LTL Formulas. Systems and Software **85**(8), 1915–1929 (2012)
40. Schillinger, P., Bürger, M., Dimarogonas, D.V.: Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems. Robotics Research **37**(7), 818–838 (2018)
41. Sewlia, M., Verginis, C.K., Dimarogonas, D.V.: Cooperative object manipulation under signal temporal logic tasks and uncertain dynamics. IEEE Robotics and Automation Letters **7**(4), 11561–11568 (2022)
42. Silano, G., Baca, T., Penicka, R., Liuzza, D., Saska, M.: Power line inspection tasks with multi-aerial robot systems via signal temporal logic specifications. IEEE Robotics and Automation Letters **6**(2), 4169–4176 (2021)
43. Sljivo, I., Perez, I., Mavridou, A., Schumann, J., Vlastos, P.G., Carter, C.: Dynamic Assurance of Autonomous Systems through Ground Control Software. AIAA/Scitech (2024)
44. Vázquez, G., Calinescu, R., Cámara, J.: Scheduling multi-robot missions with joint tasks and heterogeneous robot teams. In: Towards Autonomous Robotic Systems. pp. 354–359. Springer (2021)
45. Vázquez, G., Calinescu, R., Cámara, J.: Scheduling of missions with constrained tasks for heterogeneous robot systems. In: Formal Methods for Autonomous Systems (2022)
46. Yu, P., Dimarogonas, D.V.: Distributed motion coordination for multirobot systems under ltl specifications. IEEE Trans. Robotics **38**(2), 1047–1062 (2021)
47. Zhang, H., Kan, Z.: Temporal logic guided meta q-learning of multiple tasks. IEEE Robotics and Automation Letters **7**(3), 8194–8201 (2022)